**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NÔNG LÂM TP HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**



LUẬN VĂN TỐT NGHIỆP

NGHIÊN CỨU MỘT SỐ KỸ THUẬT DEEP LEARNING TRONG THỊ GIÁC MÁY TÍNH VÀ ỨNG DỤNG TRONG CẢNH BÁO AN NINH

**Ngành : Công nghệ thông tin**

**Niên khóa : 2015 - 2019**

**Lớp : DH15DTB**

**Sinh viên thực hiện : Đinh Ngọc Thiện**

Thành phố Hồ Chí Minh, tháng 9 năm 2019

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NÔNG LÂM TP HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**



LUẬN VĂN TỐT NGHIỆP

NGHIÊN CỨU MỘT SỐ KỸ THUẬT DEEP LEARNING TRONG THỊ GIÁC MÁY TÍNH VÀ ỨNG DỤNG TRONG CẢNH BÁO AN NINH

**Giảng viên hướng dẫn: Sinh viên thực hiện:**

**ThS. Khương Hải Châu Đinh Ngọc Thiện**

Thành phố Hồ Chí Minh, tháng 9 năm 2019

NGHIÊN CỨU MỘT SỐ KỸ THUẬT DEEP LEARNING TRONG THỊ

GIÁC MÁY TÍNH VÀ ỨNG DỤNG TRONG CẢNH BÁO AN NINH

**CÔNG TRÌNH HOÀN TẤT TẠI**

NĂM

2019

**TRƯỜNG ĐẠI HỌC NÔNG LÂM TP HCM**

Cán bộ hướng dẫn: ThS. Khương Hải Châu

Cán bộ phản biện: TS. Nguyễn Văn Dũ

Luận văn cử nhân được bảo vệ tại HỘI ĐỒNG CHẤM LUẬN VĂN CỬ NHÂN TRƯỜNG ĐẠI HỌC NÔNG LÂM TP. HCM ngày 05 tháng 09 năm 2019

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐH NÔNG LÂM TP HCM**

**CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**

**Độc Lập-Tự Do-Hạnh Phúc**

**NHIỆM VỤ LUẬN VĂN CỬ NHÂN**

Họ tên sinh viên: **ĐINH NGỌC THIỆN**

Ngày tháng năm sinh: 18/08/1997

Chuyên ngành: Công nghệ Thông tin

Phái: **Nam**

Nơi sinh: Đồng Nai

Ngành: Công nghệ Thông tin

I. TÊN ĐỀ TÀI: **Nghiên cứu một số kỹ thuật deep learning trong thị giác máy tính và ứng dụng trong cảnh báo an ninh**

II. NHIỆM VỤ VÀ NỘI DUNG

Nghiên cứu và hiểu được một số kỹ thuật deep learning, kiến thức về thị giác máy tính. Xây dựng phần mềm nhận diện người lạ và thông báo đến người sử dụng.

III. NGÀY GIAO NHIỆM VỤ: **30/02/2019**

IV. NGÀY HOÀN THÀNH NHIỆM VỤ: **30/08/2019**

V. HỌ VÀ TÊN CÁN BỘ HƯỚNG DẪN: ThS. Khương Hải Châu

Ngày / /

CÁN BỘ HƯỚNG DẪN

(Ký và ghi rõ họ tên)

Ngày / /

CÁN BỘ PHẢN BIỆN

(Ký và ghi rõ họ tên)

Ngày / /

KHOA CNTT

(Ký và ghi rõ họ tên)

**LỜI CẢM ƠN**

Lời đầu tiên em xin được gửi lời cảm ơn tới Ban Giám hiệu Trường Đại học Nông Lâm thành phố Hồ Chí Minh đã tạo mọi điều kiện tốt nhất cho em nghiên cứu, thực hiện luận văn này.

Sau thời gian nghiên cứu và làm việc luận văn tốt nghiệp của em đến nay cơ bản đã hoàn thành. Có được kết quả đó, ngoài sự cố gắng nỗ lực của bản thân còn phải kể đến sự giúp đỡ rất lớn từ ThS. Khương Hải Châu, người đã trực tiếp hướng dẫn, cung cấp kiến thức cũng như kinh nghiệm quý giá cho em trong suốt thời gian làm luận văn. Qua đây em xin được bày tỏ lòng biết ơn sâu sắc tới cô, kính chúc cô luôn mạnh khoẻ và công tác tốt. Em xin chân thành cảm ơn các thầy cô của khoa Công nghệ Thông tin đã tận tâm nhiệt tình truyền thụ cho em không chỉ những kiến thức chuyên môn mà cả những kinh nghiệm quý báu.

Cuối cùng em xin gửi lời cảm ơn chân thành tới gia đình và bạn bè là nguồn động viên hỗ trợ vô cùng to lớn giúp em có thêm động lực và sự khích lệ để hoàn thành luận văn này.

Thành phố Hồ Chí Minh, tháng 8 năm 2019

Đinh Ngọc Thiện

**MỤC LỤC**

[1. Mở đầu 1](#_Toc19376726)

[1.1. Lý do chọn đề tài 1](#_Toc19376727)

[1.2. Mục tiêu 3](#_Toc19376728)

[1.3. Đối tượng và phạm vi nghiên cứu 4](#_Toc19376729)

[2. Tổng quan 5](#_Toc19376730)

[2.1. Tìm hiểu về ảnh 5](#_Toc19376731)

[2.2. Tìm hiểu về OpenCV 6](#_Toc19376732)

[2.3. Tìm hiểu về Dlib 7](#_Toc19376733)

[2.4. Tìm hiểu về Keras 8](#_Toc19376734)

[3. Phương pháp và nội dung nghiên cứu 11](#_Toc19376735)

[3.1. Tìm hiểu về deep learning 11](#_Toc19376736)

[3.2. Feed Forward Neural Network 13](#_Toc19376737)

[3.3. Hàm mất mát và tối ưu hóa 19](#_Toc19376738)

[3.3.1. Vector điểm số 20](#_Toc19376739)

[3.3.2. Hàm mất mát 21](#_Toc19376740)

[3.3.3. Tối ưu hóa 25](#_Toc19376741)

[3.4. Convolutional Neural Networks (CNNs) 28](#_Toc19376742)

[3.4.1. Cấu trúc của CNNs 28](#_Toc19376743)

[3.4.2. Transfer learning với CNN 41](#_Toc19376744)

[3.4.3. Siamese Neural Network 42](#_Toc19376745)

[4. Kết quả 45](#_Toc19376746)

[4.1. Kết quả thực nghiệm 45](#_Toc19376747)

[4.1.1. Mô hình liveness 45](#_Toc19376748)

[4.1.2. Mô hình identifier 47](#_Toc19376749)

[4.2. Sản phẩm đề tài 50](#_Toc19376750)

[5. Kết luận 57](#_Toc19376751)

[TÀI LIỆU THAM KHẢO 58](#_Toc19376752)

**DANH SÁCH CHỮ VIẾT TẮT**

AI **A**rtificial **I**ntelligence

BS **B**atch **S**ize

CNN **C**onvolution **N**eural **N**etwork

DL **D**eep **L**earning

DO **D**rop **O**ut

DNN **D**eep **N**eural **N**etwork

FC **F**ully **C**onnected

FF **F**eed **F**orward

lr **L**earning **R**ate

NN **N**eural **N**etwork

GD **G**radient **D**escent

SGD **S**tochastic **G**radient **D**escent

SVM **S**upport **V**ector **M**achine

**DANH MỤC CÁC HÌNH**

[Hình 1.1.1: Tình hình vi phạm pháp luật ở Việt Nam tháng 4 - 2019 [2] 1](#_Toc19176357)

[Hình 1.1.2: Tình hình vi phạm pháp luật ở Việt Nam tháng 5 - 2019 [3] 2](#_Toc19176358)

[Hình 1.1.3: Tình hình vi phạm pháp luật ở Việt Nam tháng 6 - 2019 [4] 2](#_Toc19176359)

[Hình 1.1.4: Tình hình vi phạm pháp luật ở Việt Nam tháng 7 - 2019 [5] 3](#_Toc19176360)

[Hình 2.1.1: Ma trận biểu diễn ảnh xám [26] 5](#_Toc19631445)

[Hình 2.1.2: Kết quả khi tách các ma trận trong tổ hợp biểu diễn ảnh màu 5](#_Toc19631446)

[Hình 2.2.1: Cấu trúc OpenCV [22] 6](#_Toc19631457)

[Hình 2.3.1: Cấu trúc Dlib [28] 7](#_Toc19631468)

[Hình 2.3.2: Các thuật toán machine learning trong Dlib [28] 8](#_Toc19631469)

[Hình 2.4.1: Cấu trúc Keras [6] 9](#_Toc19631480)

[Hình 2.4. 2: Các module xây dựng mô hình trong Keras [6] 9](#_Toc19631481)

[Hình 2.4.3: Qui trình hoạt động của Keras 10](#_Toc19631482)

[Hình 3.1.1: Deep Learning trong mối liên hệ với AI [10] 11](#_Toc19176416)

[Hình 3.1.2: Học thông qua đại diện [7] 12](#_Toc19176417)

[Hình 3.2.1: Perceptron (Neuron nhân tạo) [9] 13](#_Toc19441896)

[Hình 3.2.2: Phân loại tuyến tính 14](#_Toc19441897)

[Hình 3.2.3: Hàm tuyến tính 15](#_Toc19441898)

[Hình 3.2.4: Hàm sigmoid [8] 15](#_Toc19441899)

[Hình 3.2.5: Hàm ReLU (Rectified Linear Unit) [8] 16](#_Toc19441900)

[Hình 3.2.6: PLA biểu diễn các hàm logic cơ bản [1] 17](#_Toc19441901)

[Hình 3.2.7: Multi-layer perceptron biểu diễn hàm XOR [1] 18](#_Toc19441902)

[Hình 3.2.8: Overfitting 18](#_Toc19441903)

[Hình 3.2.9: Dung tích mạng thần kinh nhân tạo [8] 19](#_Toc19441904)

[Hình 3.3.1: “Học” trong ngữ nghĩa của “học sâu” - bước 1 [7] 20](#_Toc19176433)

[Hình 3.3.2: Vector điểm số 20](#_Toc19176434)

[Hình 3.3.3: “Học” trong ngữ nghĩa của “học sâu” - bước 2 [7] 21](#_Toc19176435)

[Hình 3.3.4: Drop-out [9] 23](#_Toc19176436)

[Hình 3.3.5: Cross Validation 24](#_Toc19176437)

[Hình 3.3.6: Data Augmentation 24](#_Toc19176438)

[Hình 3.3.7: “Học” trong ngữ nghĩa của “học sâu” - bước 3 [7] 25](#_Toc19176439)

[Hình 3.3.8: Nesterov Accelerated Gradient (NAG) [21] 26](#_Toc19176440)

[Hình 3.4.1: Khu vườn mạng thần kinh nhân tạo [24] 29](#_Toc19176443)

[Hình 3.4.2: Làm sắc nét ảnh với phương pháp tích chập [25] 31](#_Toc19176444)

[Hình 3.4.3: Convolution 32](#_Toc19176445)

[Hình 3.4.4: Vấn đề khi không có padding 32](#_Toc19176446)

[Hình 3.4.5: Convolution layer 33](#_Toc19176447)

[Hình 3.4.6: Max Pooling layer 34](#_Toc19176448)

[Hình 3.4.7: Các đội chiến thắng cuộc thi ILSVRC [18] 35](#_Toc19176449)

[Hình 3.4.8: Cấu trúc AlexNet [19] 35](#_Toc19176450)

[Hình 3.4.9: Cấu trúc VGG19 36](#_Toc19176451)

[Hình 3.4.10: Inception module (phiên bản đơn giản) [20] 37](#_Toc19176452)

[Hình 3.4.11: Inception module with dimension reductions [20] 37](#_Toc19176453)

[Hình 3.4.12: Tỉ lệ lỗi của mạng với 20 và 56 layer trên Cifar-10 [18] 38](#_Toc19176454)

[Hình 3.4.13: Khối dư (Residual block) [18] 38](#_Toc19176455)

[Hình 3.4.14: Khối dư thắt cổ chai (Bottleneck residual block) [18] 39](#_Toc19176456)

[Hình 3.4.15: Khối tích chập trong MobileNetV2 [12] 40](#_Toc19176457)

[Hình 3.4.16: CNN trong nhiệm vụ phân loại các đối tượng 41](#_Toc19176458)

[Hình 3.4.17: Siamese Neural Network [16] 42](#_Toc19176459)

[Hình 3.4.18: Chiến lược One-shot learning trong nhận diện ảnh [16] 43](#_Toc19176460)

[Hình 4.2.1: Hệ thống nhận diện khuôn mặt người lạ 50](#_Toc19606329)

[Hình 4.2.2: Màn hình sau khi khởi động phần mềm 50](#_Toc19606330)

[Hình 4.2.3: Màn hình cài đặt 51](#_Toc19606331)

[Hình 4.2.4: Cửa sổ thêm khuôn mặt 51](#_Toc19606332)

[Hình 4.2.5: Mành hình khi huấn luyện học khuôn mặt 52](#_Toc19606333)

[Hình 4.2.6: Màn hình đặt thông tin nhóm 52](#_Toc19606334)

[Hình 4.2.7: Cửa sổ giám sát khi phát hiện khuôn mặt giả mạo từ smartphone 53](#_Toc19606335)

[Hình 4.2.8: Cửa sổ giám sát khi phát hiện khuôn mặt giả mạo từ hình ảnh 53](#_Toc19606336)

[Hình 4.2.9: Cửa sổ giám sát khi phát hiện khuôn mặt trực tiếp 54](#_Toc19606337)

[Hình 4.2.10: Màn hình cập nhật khuôn mặt từ dữ liệu giám sát 54](#_Toc19606338)

[Hình 4.2.11: Màn hình đăng ký nhận thông tin từ nhóm 55](#_Toc19606339)

[Hình 4.2.12: Màn hình chính khi hệ thống phát hiện người lạ 56](#_Toc19606340)

[Hình 4.2.13: Màn hình chọn âm báo 56](#_Toc19606341)

**DANH MỤC CÁC BẢNG**

[Bảng 3.4.1: So sánh một số neural network cho các vấn đề của thị giác máy tính 30](#_Toc19176802)

[Bảng 3.4.2: Chiến lược sử dụng phương pháp Transfer Learning [8] 41](#_Toc19176803)

[Bảng 4.1.1: Kết quả dự đoán của mô hình huấn luyện với BS 32, lr 1e-4, tỉ lệ 50%:25%:25% 46](#_Toc19176808)

[Bảng 4.1.2: Kết quả dự đoán của mô hình huấn luyện với RMSprop đóng băng 20 layer, BS 32, lr 1e-4 46](#_Toc19176809)

[Bảng 4.1.3: Kết quả dự đoán của mô hình huấn luyện với RMSprop đóng băng 20 layer, BS 32, tỉ lệ 50%:25%:25% 47](#_Toc19176810)

[Bảng 4.1.4: Kết quả dự đoán của mô hình huấn luyện với RMSprop đóng băng 20 layer, lr 1e-4, tỉ lệ 50%:25%:25% 47](#_Toc19176811)

[Bảng 4.1.5: Kết quả dự đoán của các mô hình huấn luyện với BS 16, lr 1e-4, 3 nhãn, DO 0.5 48](#_Toc19176812)

[Bảng 4.1.6: Kết quả dự đoán của mô hình huấn luyện với lr 1e-4, 3 nhãn, DO 0.5 48](#_Toc19176813)

[Bảng 4.1.7: Kết quả dự đoán của mô hình huấn luyện với BS 16, 3 nhãn, DO 0.5 49](#_Toc19176814)

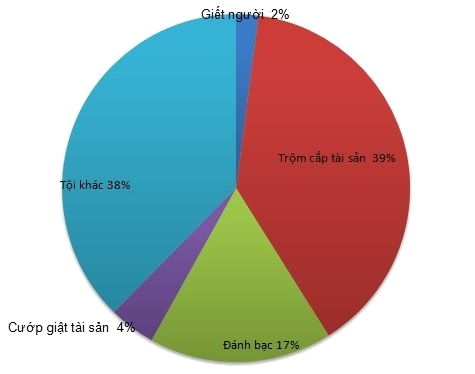
[Bảng 4.1.8: Kết quả dự đoán của mô hình huấn luyện với BS 16, 3 nhãn, lr 5e-4 49](#_Toc19176815)

[Bảng 4.1.9: Kết quả dự đoán của mô hình huấn luyện với BS 16, lr 5e-4, DO 0.2 49](#_Toc19176816)

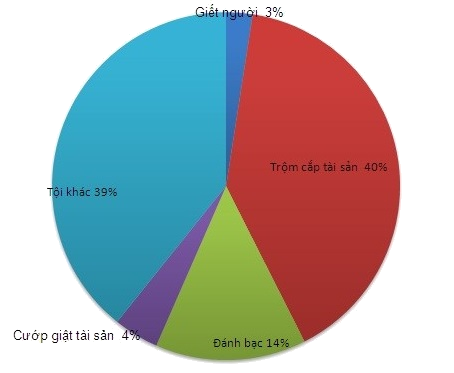
# Mở đầu

* 1. Lý do chọn đề tài

Năm 2006, Việt Nam trở thành thành viên thứ 150 của WTO sau 11 năm phấn đấu và trưởng thành. Điều này cho thấy trong sự chuyển động không ngừng của thời đại Việt Nam, chúng ta không còn đơn độc mà trở thành một đồng minh quan trọng của các quốc gia trên thế giới trong tất cả các lĩnh vực. Song song với sự phát triển của nền kinh tế trong những năm gần đây, Việt Nam được đánh giá là quốc gia có tình hình chính trị và kinh tế ổn định nhất. Tuy nhiên, bên cạnh sự phát triển kinh tế xã hội, Việt Nam cũng diễn ra khá nhiều diễn biến phức tạp, hàng loạt loại tội phạm nguy hiểm ảnh hưởng đến sự phát triển chung của xã hội diễn ra ngày càng phổ biến..



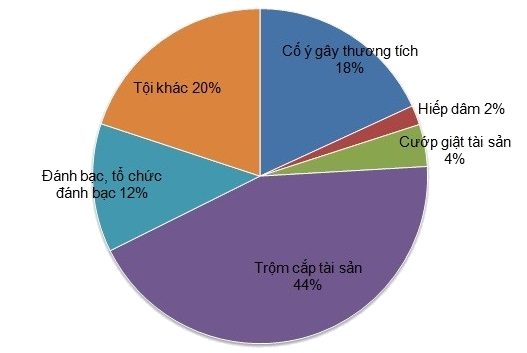
*Hình 1.1.1**: Tình hình vi phạm pháp luật ở Việt Nam tháng 4 - 2019 [2]*



*Hình 1.1.2**: Tình hình vi phạm pháp luật ở Việt Nam tháng 5 - 2019 [3]*



*Hình 1.1.3**: Tình hình vi phạm pháp luật ở Việt Nam tháng 6 - 2019 [4]*



*Hình 1.1.4**: Tình hình vi phạm pháp luật ở Việt Nam tháng 7 - 2019 [5]*

Từ thông tin của Bộ Công An Việt Nam [2] - [5], chỉ trong bốn tháng qua – từ tháng 4 đến tháng 7, số vụ vi phạm là 4.233 - 4.112 - 4.294 - 4.153, những thay đổi là không đáng kể nhưng nhìn chung con số trong số các vụ trộm có tỷ lệ tăng từ 39% - 40% - 41% - 44%. Đó là dấu hiệu của vấn đề trộm cắp đã và đang ngày càng có tỷ lệ vi phạm pháp luật cao hơn ở Việt Nam ngày nay.

Trước tình hình đó, nhiều hộ gia đình đã trang bị chuông báo khách bằng cảm biến hồng ngoại. Thực tế chúng không thể nhận ra đâu là chủ nhà, đâu là người lạ. Do đó, loại chuông này đã gây không ít phiền toái cho người sử dụng và cả hàng xóm của họ. Một số hộ lại chọn cách đặt video giám sát, nhưng khi xem lại băng ghi hình và phát hiện vụ việc thì trộm cũng đã tẩu thoát. Mong muốn giải quyết phần nào vấn nạn đó, đề tài hướng đến một sản phẩm là một phần mềm có khả năng nhận diện khuôn mặt và thông báo khi phát hiện người lạ.

* 1. Mục tiêu

- Nghiên cứu và hiểu được một số kỹ thuật deep learning, kiến thức về thị giác máy tính.

- Tìm hiểu và có khả năng vận dụng được các thư viện mã nguồn mở OpenCV, Dlib, Keras.

- Xây dựng phần mềm nhận diện người lạ và thông báo đến người sử dụng.

* 1. Đối tượng và phạm vi nghiên cứu

- Đối tượng nghiên cứu: hệ thống cảnh báo an ninh, các kỹ thuật deep learning và kiến thức về thị giác máy tính liên quan đến nhận diện khuôn mặt.

- Phạm vi đề tài: khuôn mặt không bị che bởi các vật thể khác, điều kiện ánh sáng bình thường, độ lệch hướng nhìn từ khuôn mặt đến camera không quá lớn (<30 độ). Đối tượng sử dụng là các hộ gia đình ở Việt Nam.

1. Tổng quan

## Tìm hiểu về ảnh

Ảnh là tập hợp các điểm ảnh (pixel). Các điểm ảnh này thường được thể hiện dưới hai dạng: Với loại ảnh xám (còn gọi là ảnh trắng đen), các điểm ảnh có giá trị từ 0 đến 255, trong đó 0 đại diện cho màu đen và 255 đại diện cho màu trắng, các giá trị gần 0 sẽ có màu tối hơn và các giá trị gần 255 sẽ có màu sáng hơn. Một ảnh xám có thể được biểu diễn bởi một trận gổm các giá trị từ 0 đến 255 như hình dưới.



*Hình 2.1.1: Ma trận biểu diễn ảnh xám [26]*

Với loại ảnh màu, các điểm ảnh lúc này bao gồm bao gổm 3 giá trị đỏ - lục - lam. Để biểu diễn ảnh màu, ta cần ba ma trận chồng lên nhau, mỗi ma trận trên tạo nên một kênh màu. Các kênh màu thường được sắp xếp theo thứ tự (red, green, blue), ta gọi đó là ảnh RGB. Ba giá trị tương ứng trên ba ma trận kết hợp với nhau tạo nên một điểm ảnh có màu sắc tương ứng, ví dụ (255, 0, 0) cho điểm ảnh màu đỏ, (0,255, 0) cho điểm ảnh màu lục. Khi tách tổ hợp ma trận trên ta được các ma trận có thể biểu diễn ảnh xám, tùy vào sự có mặt nhiều hay ít của kênh màu mà điểm ảnh trên ảnh xám có độ sáng tối khác nhau.



*Hình 2.1.2: Kết quả khi tách các ma trận trong tổ hợp biểu diễn ảnh màu*

## Tìm hiểu về OpenCV

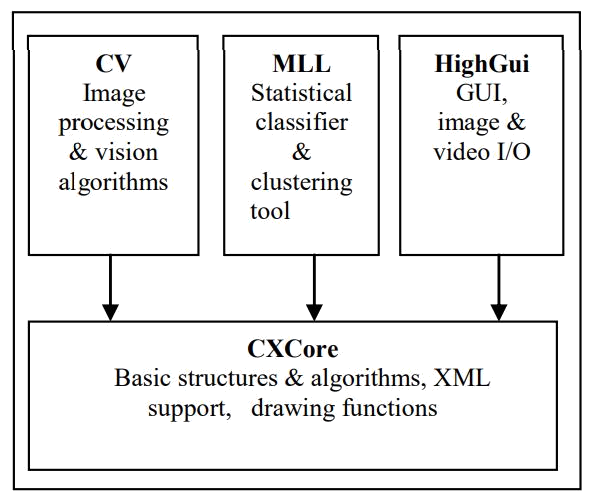
OpenCV (Open Computer Vision) là một thư viện mã nguồn mở hàng đầu hỗ trợ xử lý về thị giác máy tính và machine learning. Nó có các interface cho C ++, Python, Java và MATLAB và hỗ trợ Windows, Linux, Android và Mac OS. OpenCV chủ yếu hướng tới các ứng dụng thời gian thực [27]. OpenCV lưu mã màu theo thứ tự BGR tức là (blue, green, red) thay vì RGB thông dụng.

Các chức năng của openCV tập trung vào thu thập ảnh, xử lí ảnh và các thuật toán phân tích dữ liệu ảnh, bao gồm truy xuất và hiển thị ảnh, video; hỗ trợ cấu trúc dữ liệu ảnh như ma trận, vector; xử lí ảnh căn bản như phóng to thu nhỏ, đổi màu, vẽ lên ảnh; xử lí cấu trúc như tìm viền, thay đổi không gian 3D; phân tích dữ liệu ảnh như nhận dạng vật thể, theo dõi và phân tích chuyển động…

Các ưu điểm của OpenCV:

* Miễn phí.
* Được viết bằng ngôn ngữ C/C++ nên tốc độ xử lý nhanh chóng (khoảng 30 fps).
* Ít tốn RAM (60 – 70 mb).
* Có cộng đồng hỗ trợ lớn (do rất nhiều tập đoàn lớn sử dụng OpenCV trong ứng dụng của mình như Google, Intel, IBM, Microsoft…).

Cấu trúc cơ bản của OpenCV được thể hiện như trên hình 2.2.1.



*Hình 2.2.1: Cấu trúc OpenCV [22]*

CXCORE chứa cấu trúc dữ liệu, đại số ma trận, xử lý lỗi, vẽ và toán học cơ bản,… Nó cũng hoạt động để tải mã động thực hiện tất cả các cấu trúc, thuật toán và các bản vẽ cơ bản.

CV xử lý tất cả các thuật toán xử lý hình ảnh và thị giác máy tính như cấu trúc hình ảnh, phân tích chuyển động và theo dõi, nhận dạng mẫu, máy ảnh, hiệu chuẩn,…

HighGUI chịu trách nhiệm tạo cung cấp giao diện người dùng, xử lý các sự kiện chuột và bàn phím, thêm thanh theo dõi, đọc và ghi hình ảnh hoặc video.

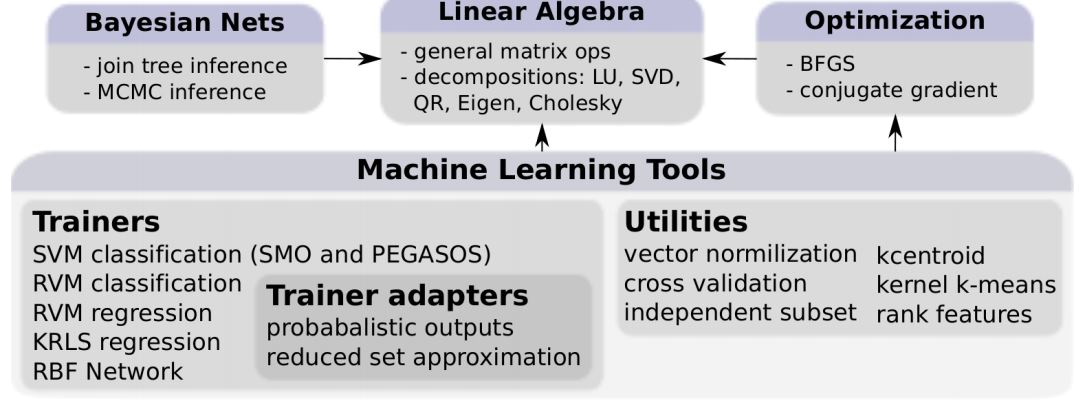
MLL bao gồm nhiều cụm, chức năng phân loại và phân tích dữ liệu.

## Tìm hiểu về Dlib

Dlib là bộ công cụ C ++ chứa các thuật toán và công cụ học máy để tạo phần mềm phức tạp trong C ++ để giải quyết các vấn đề thực tiễn. Nó được sử dụng trong cả ngành công nghiệp và học thuật trong nhiều lĩnh vực bao gồm robot, thiết bị nhúng, điện thoại di động và môi trường điện toán hiệu năng cao. Giấy phép nguồn mở của Dlib cho phép việc sử dụng nó trong bất kỳ ứng dụng nào và hoàn toàn miễn phí. [28]

Dlib được thiết kế cho các ứng dụng máy học và trí tuệ nhân tạo.

Dlib thường được sử dụng trong detect khuôn mặt, với phương pháp feature descriptor là HOG (Histogram Oriented Gradient) và bộ phân loại là SVM (Support Vector Machine). Đây được cho là phương pháp phát hiện khuôn mặt nhanh nhất trên CPU, tuy nhiên nó không làm việc tốt đối với khuôn mặt không chính diện.

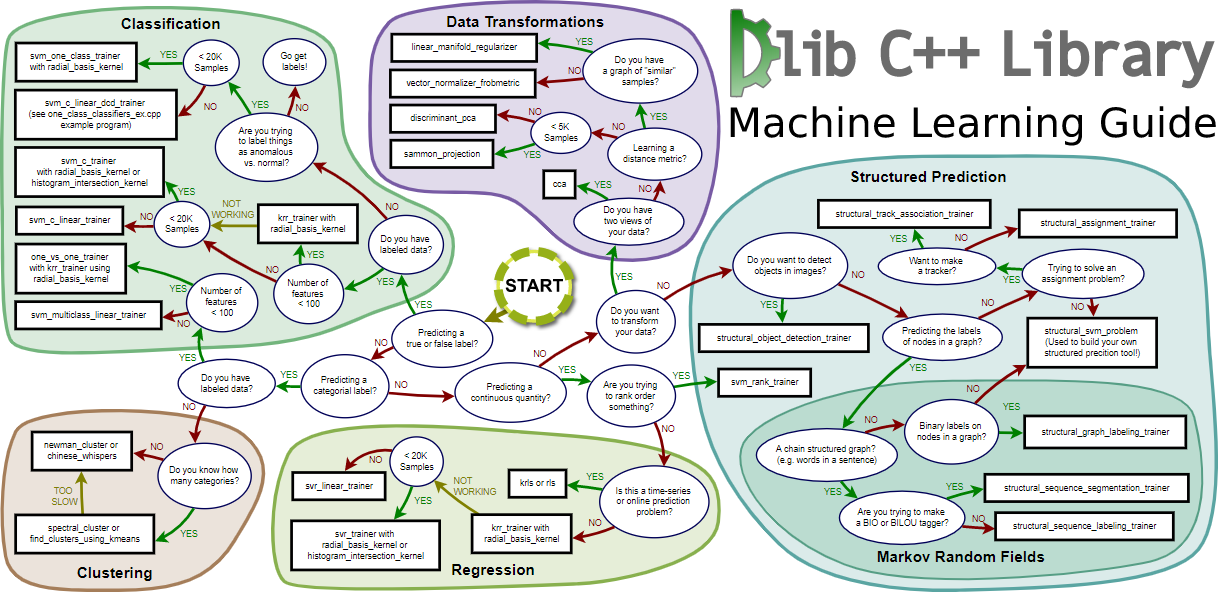


*Hình 2.3.1: Cấu trúc Dlib [28]*

Thư viện dlib gồm bốn thành phần, trong đó có hai thành phần chính là Linear Algebra và Machine Learning Tools.

Trong Linear Algebra, ba đối tượng đóng vai trò quan trọng là ma trận, vector và hình chữ nhật. Các function trong Linear Algebra đều thao tác trên các đối tượng này, ví dụ tìm góc giữa hai cạnh, đếm số điểm giữa hai cạnh, tính khoảng cách giữa một điểm đến cạnh của hình chữ nhật, trả về tâm một hình chữ nhật…

Dlib chứa nhiều thuật toán học máy hỗ trợ phân loại, gom cụm, hồi qui, biến đổi dữ liệu và dự đoán có cấu trúc. Tất cả được thiết kế để có tính module cao, thực hiện nhanh chóng và sử dụng đơn giản thông qua API C ++ gọn và hiện đại. Nó được sử dụng trong một loạt các ứng dụng bao gồm robot, thiết bị nhúng, điện thoại di động và môi trường máy tính hiệu năng cao.



*Hình 2.3.2: Các thuật toán machine learning trong Dlib [28]*

## Tìm hiểu về Keras

Keras là một thư viện neural network, được viết bằng Python và có khả năng chạy trên nền của TensorFlow, CNTK hoặc Theano. Nó được phát triển với chủ trương là cho phép trải nghiệm nhanh. Các ưu điểm của Keras:

- Cho phép tạo mô hình dễ dàng và nhanh chóng (thông qua tính thân thiện với người dùng, tính module hóa và khả năng mở rộng).

- Hỗ trợ cả mạng tích chập và mạng hồi qui, cũng như kết hợp cả hai.

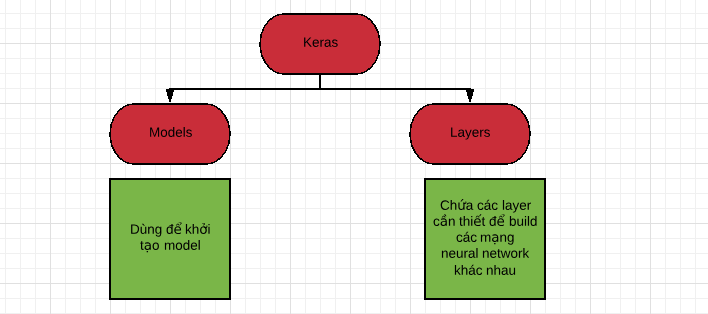
- Chạy trên cả CPU và GPU.

Cấu trúc của Keras gồm ba thành phần chính [6]:



*Hình 2.4.1: Cấu trúc Keras [6]*

Đầu tiên là các module dùng để xây dựng bộ xương cho model :



*Hình 2.4. 2: Các module xây dựng mô hình trong Keras [6]*

Model dùng để khởi tạo một mô hình trong keras.

Layers để tạo ra các layer trong mô hình như Dense để tạo layer FC, Conv để tạo layer tích chập,… và các thành phần định nghĩa cho layer như Activation để gọi hàm kích hoạt cho layer, Dropout để chuẩn hóa,… Layers cũng hỗ trợ tạo các layer theo ý muốn.

Tiếp theo là các function tiền xử lý dữ liệu gồm có Sequence Preprocessing, Text Preprocessing, Image Preprocessing. Ta chỉ quan tâm đến Image Preprocessing, nó hỗ trợ xáo trộn dữ liệu và các phương pháp tiền xử lý ảnh đơn giản như phóng to thu nhỏ, dịch chuyển...

Cuối cùng là các function trong bộ xương của Keras gồm các hàm mất mát, hàm tối ưu, metrics dùng để đánh giá accuracy của model, các hàm kích hoạt, Callbacks dùng để lưu lại mô hình nhằm tránh sự cố khi đang huấn luyện.

Qui trình hoạt động của Keras gồm năm bước:



*Hình 2.4.3: Qui trình hoạt động của Keras*

Đầu tiên ta cần định nghĩa một mô hình gồm một chuỗi các layer.

Tiếp theo, ta cấu hình cho mô hình. Trong bước này, ta định nghĩa thuật toán tối ưu, tốc độ học, batch size…

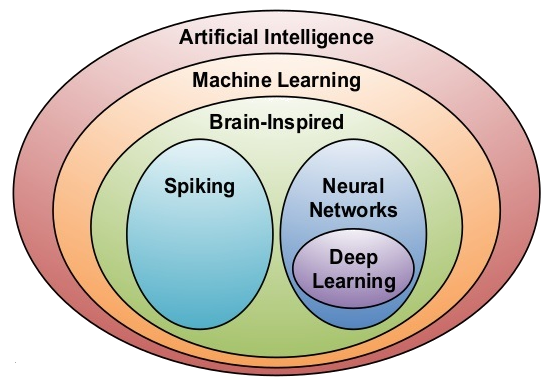
Để huấn luyện mạng yêu cầu việc chuẩn bị dữ liệu gồm load dữ liệu, xáo trộn dữ liệu, các bước tiền xử lý dữ liệu. Sau đó mạng được huấn luyện với các cấu hình ở bước trước.

Mạng thường được đánh giá trên dữ liệu chưa được huấn luyện. Nếu kết quả đánh giá chưa cho tỉ lệ lỗi thấp, ta quay lại bước đầu tiên để tạo lại mô hình phù hợp hơn hoặc bước thứ hai để cấu hình lại mô hình.

Khi việc đánh giá cho kết quả khả quan, ta có thể sử dụng mô hình để dự đoán trên dữ liệu mới.

# Phương pháp và nội dung nghiên cứu

## Tìm hiểu về deep learning

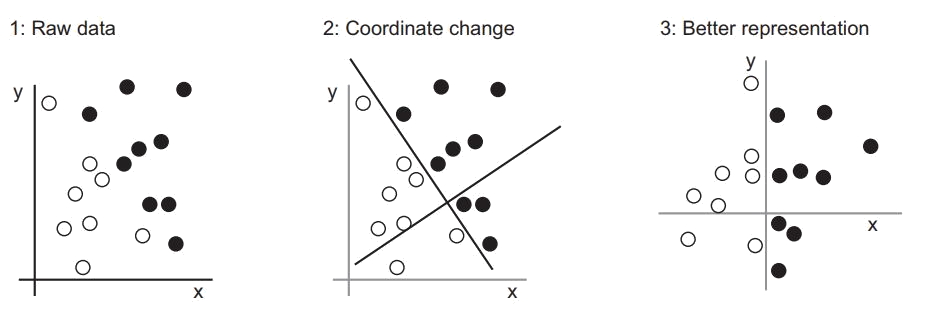


*Hình 3.1.1**: Deep* *Learning trong mối liên hệ với AI [10]*

Trước tiên, chúng ta cần định nghĩa về deep learning (DL) và cách DL liên quan đến toàn bộ lĩnh vực trí thông minh nhân tạo (Artificial Intelligence hay AI). AI là một kỹ thuật tạo ra những chương trình có khả năng bắt trước các phản ứng của con người, AI bao gồm nhiều cách tiếp cận không liên quan đến bất kỳ thuật toán học tập nào mà chúng ta gọi là symbolic AI. Ví dụ, các chương trình đánh cờ đời đầu, chỉ liên quan đến các quy tắc được code cứng được tạo bởi các lập trình viên.

Machine Learning (ML) mở ra một hướng tiếp cận mới. Nếu như trong cách tiếp cận cũ, ta đưa vào hệ thống dữ liệu và tập luật để nhận được kết quả trả về. Trong cách tiếp cận của ML, ta đưa vào hệ thống dữ liệu và kết quả kì vọng, thuật toán sẽ xử lý và trả về tập luật, ta có thể áp dụng các luật này vào dữ liệu mới để nhận được kết quả dự đoán.

Cách ngắn gọn để mô tả quá trình học tập là tìm ra một cách biểu diễn mới như ví dụ tìm luật mô tả các điểm màu đen và trắng [7]. Theo cách biểu diễn cuối cùng trên hình 3.1.2, vấn đề có thể được biểu diễn như một quy tắc đơn giản: các điểm đen là những vị trí có giá trị x lớn hơn 0 và các điểm trắng là các vị trí sao cho x nhỏ hơn 0.



*Hình 3.1.2**: Học thông qua đại diện [7]*

Trong lĩnh vực ML, có một lĩnh vực thường được gọi là brain-inspired. Brain-inspired hiện được cho là cách học máy tốt nhất mà chúng ta đã biết cho đến nay để học và giải quyết các vấn đề, vì vậy đây là một nơi đầu tiên để tìm kiếm một phương pháp học máy. Brain-inspired không đồng nghĩa với việc tạo ra một bộ não, mà nó nhằm mục đích mô phỏng một số khía cạnh về cách chúng ta hiểu về cách vận hành của bộ não con người.

Trong khu vực brain-inspired, một bộ phận được gọi là mạng thần kinh nhân tạo chủ yếu được xây dựng bằng cách sử dụng các đơn vị tính toán được lý tưởng hóa với các giá trị của các hàm kích hoạt và một bộ đầu vào có trọng số. Những đơn vị này thường được gọi là các neuron nhân tạo vì được lấy ý tưởng từ cách phản sinh học của các neuron trên não người.

Neural Network (NN) ở thế hệ đầu tiên sử dụng layer đặc trưng được lập trình thủ công và cố gắng nhận diện các đối tượng bằng cách học (cập nhật) các trọng số. Ta có thể dễ dàng kiểm soát việc học theo cách này nhưng về cơ bản những vẫn đề chúng có thể giải quyết còn khá giới hạn.

NN thế hệ thứ hai - Các multi-layer perceptron sử dụng các hàm kích hoạt phi tuyến tính để có thể xếp chồng nhiều hơn một layer và sự tồn tại của đạo hàm của chúng cho phép sử dụng các phương pháp tối ưu theo đạo hàm để huấn luyện.

Tiểu khu vực còn lại - Mạng thần kinh Spiking - được coi là thế hệ thứ ba của mạng thần kinh nhân tạo. Trong tiểu khu vực này, mặc dù hiệu quả của nó đã được mô tả trên lý thuyết, chúng chưa được sử dụng rộng rãi trong các ứng dụng trong thế giới thực do yêu cầu tính toán rất tốn kém. Vì lý do đó, chúng ta sẽ không đi sâu vào thế hệ mạng thần kinh này.

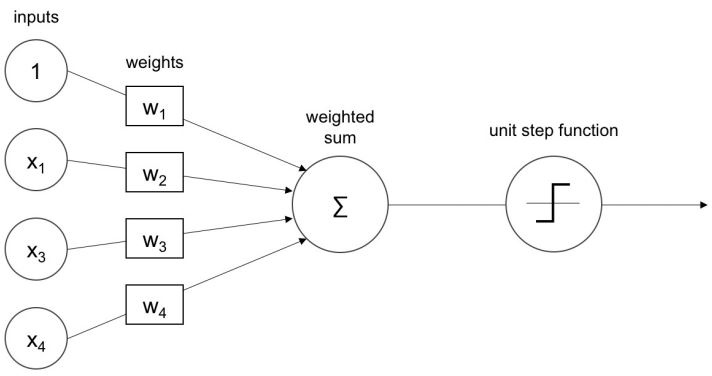
Trong miền của các mạng thần kinh, có một khu vực được gọi là học sâu. Ngày nay, số lượng layer của các mạng được sử dụng trong học sâu giao động từ năm đến hơn một nghìn. Độ sâu (depth) đề cập đến số lượng layer trong NN, NN có nhiều hơn một layer ẩn được gọi là Deep Neural Networks (DNNs), đó là lý do tại sao chúng ta gọi việc học trên NN nhiều layer là học sâu hay deep learning (DL).

Tại sao chọn deep learning? Một số tính chất ưu việt của việc học sâu được liệt kê từ [7]: Đầu tiên là tính đơn giản hóa - Học sâu loại bỏ yêu cầu về các kỹ thuật trích xuất đặc trưng, nó thay thế các biện pháp thủ công phức tạp và nặng về kỹ thuật bằng các mô hình đơn giản, có thể đào tạo từ đầu đến cuối bằng thuật toán mà không cần thêm sự can thiệp từ con người.

Thứ hai, khả năng mở rộng - Học sâu có thể chạy song song, vì vậy chúng ta có thể tăng tốc quá trình đào tạo với nhiều GPU (Graphic Processing Unit) hoặc TPU (Tensor Processing Unit). Ngoài ra, các mô hình học sâu được đào tạo bằng cách lặp qua các tập dữ liệu con của dataset, do đó chúng có thể được đào tạo trên các tập dữ liệu có kích thước tùy ý.

Thứ ba, tính linh hoạt và khả năng tái sử dụng - Không giống như các phương pháp học máy trước đây, các mô hình học sâu có thể được đào tạo với dữ liệu bổ sung mà không cần bắt đầu lại từ đầu, khiến chúng có thể học trực tuyến liên tục. Các mô hình học sâu có thể tái sử dụng để huấn luyện lại theo nhiều mục đích khác nhau, hoặc giảm thời gian huấn luyện lại trên bộ dữ liệu mới.

## Feed Forward Neural Network



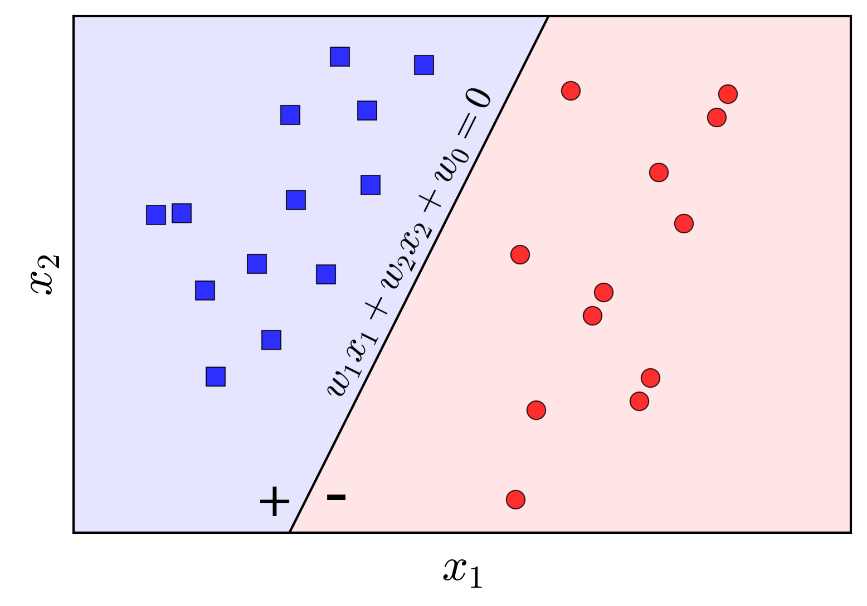
*Hình 3.2.1**: Perceptron (Neuron nhân tạo) [9]*

Feed Forward (FF) là dạng đầu tiên và đơn giản nhất của mạng thần kinh nhân tạo. Mạng FF có các kết nối giữa các neuron không tạo thành một vòng lặp.

Yếu tố đầu tiên và cơ bản nhất trong mạng thần kinh nhân tạo: perceptron. Perceptron là một đại diện nhân tạo của một tế bào thần kinh của con người. Tế bào thần kinh nhân tạo làm gì? Đầu tiên, tất cả các yếu tố đầu vào X được nhân với trọng số của chúng W. Thứ hai, cộng tất cả các giá trị nhân ta được tổng trọng số. Cuối cùng, đưa tổng trọng số đó vào hàm kích hoạt.

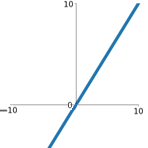
Hàm kích hoạt này quyết định xem tổng số có trọng số có được kích hoạt hay không. Hàm kích hoạt cũng giúp chuẩn hóa đầu ra của mỗi neuron thông qua quá trình mapping, ví dụ, từ 0 đến 1, từ -1 đến 1 hoặc từ 0 đến vô hạn.

Hàm kích hoạt cơ bản nhất là binary step còn được gọi là phân loại tuyến tính nhị phân. Binary step được sử dụng để phân loại dữ liệu thành hai phần. Hàm sgn là một dạng của binary step, nếu một điểm có tổng trọng số lớn hơn 0 thì điểm đó thuộc nhãn dương, nếu không thì thuộc nhãn âm. Chúng ta nhận thấy, một trong những trọng số là được kết nối với đầu vào chỉ là số 1, được gọi là bias. Không có bias này, chúng ta luôn có một đường thẳng đi qua gốc 0.



*Hình 3.2.2: Phân loại tuyến tính*

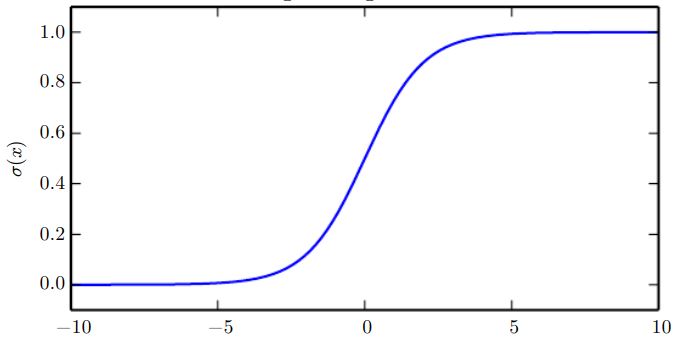
Nếu chúng ta muốn nhiều các perceptron như vậy được kết nối để phân loại nhiều hơn hai nhãn, chúng ta mong muốn mạng chỉ kích hoạt một neuron và các neuron khác phải là 0. Nhưng vấn đề xảy ra khi có một số neuron được kích hoạt, vì vậy chúng ta mong đợi hàm kích hoạt trả về một tỉ lệ gọi là độ tin cậy ví dụ 60% hoặc 80% được kích hoạt và không phải là nhị phân 100% hoặc 0% hay 1 hoặc 0 như trên.



*Hình 3.2.3: Hàm tuyến tính*

Với hàm tuyến tính, nó cung cấp một phạm vi kích hoạt, chúng ta có thể xếp một vài neuron lại với nhau và nếu có nhiều hơn một neuron được kích hoạt, chúng ta có thể lấy giá trị lớn nhất làm kết quả dự đoán.

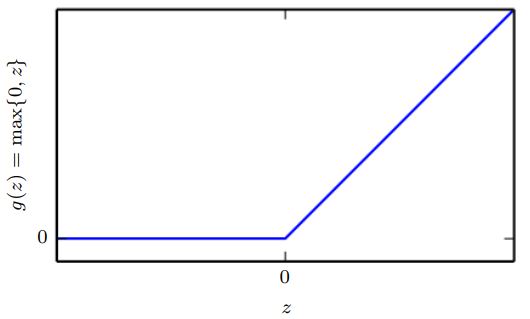
Hạn chế của hàm kích hoạt này cũng là hạn chế của thế hệ NN đầu tiên. Nếu chúng ta có hai hàm f (a) và f (b), chúng ta có thể tìm thấy hàm f (a + b) = f (a) + f (b). Đó là lý do tại sao cho dù chúng ta có bao nhiêu layer, sự kết hợp của các hàm tuyến tính theo cách tuyến tính vẫn là một hàm tuyến tính khác. Điều đó có nghĩa là các layer này có thể được thay thế bằng một layer duy nhất và chúng ta đánh mất khả năng xếp chồng các layer.



*Hình 3.2.4: Hàm sigmoid [8]*

Sigmoid có một sự khác biệt, nó là chức năng kích hoạt phi tuyến tính, chúng ta có thể xếp chồng các layer. Sigmoid tốt cho việc xem xét phân loại, các giá trị Y rất dốc giữa các giá trị X nằm gần giá trị 0, hàm này có xu hướng đưa các giá trị Y đến hai đầu của đường cong, điều đó có nghĩa là nó phân biệt rõ ràng trong dự đoán của đầu ra thay vì độ chắc chắn nửa chừng khiến cho việc quyết định gặp khó khăn.

Nhưng nhìn vào hai đầu của hàm sigmoid, các giá trị Y có xu hướng phản ứng rất ít với các thay đổi trong X. Điều đó có nghĩa là độ dốc tại khu vực đó sẽ quá nhỏ và mạng từ chối học thêm hoặc học chậm đáng kể, ta gọi đó là vanishing gradient. Hơn nữa, công thức với hàm exp làm tăng chi phí tính toán, và khi mạng được huấn luyện trên tập dữ liệu lớn và qua nhiều vòng lặp, lúc này chi phí tính toán là cực lớn.



*Hình 3.2.5: Hàm ReLU (Rectified Linear Unit) [8]*

ReLU cho đầu ra là x nếu x dương và 0 trong trường hợp còn lại. ReLU rất giống với tuyến tính, nhưng nó hoàn toàn là hàm phi tuyến, vì nó không có thuộc tính cộng của hàm tuyến tính, ví dụ, với Vậy đây là hàm phi tuyến với chi phí tính toán rất thấp, không bị vanishing gradient và chúng ta có thể xếp chồng các layer.

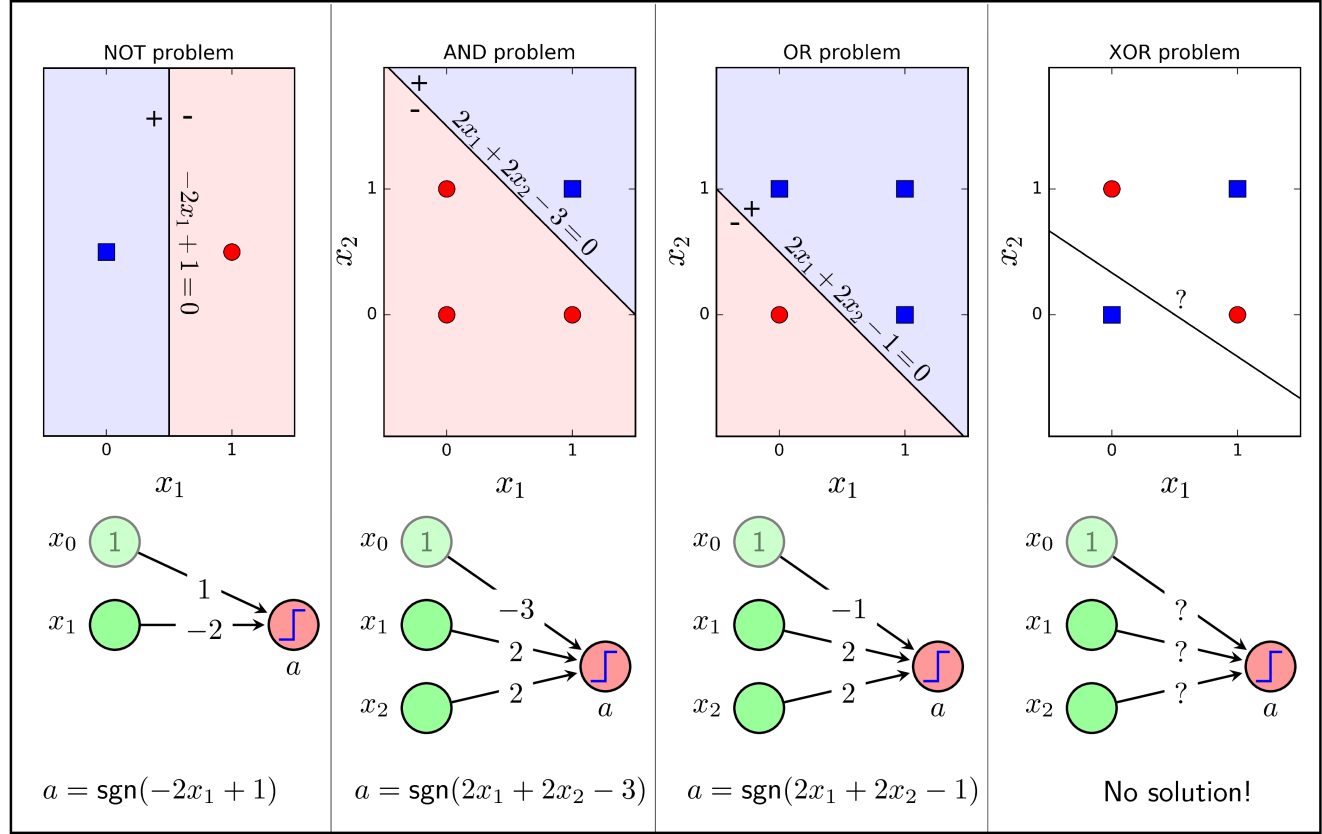
Một điểm khác, hãy tưởng tượng mỗi neuron có một hàm kích hoạt, do đó một mạng có 10 nghìn neuron sẽ có 10 nghìn hàm kích hoạt, tức chi phí tính toán rất cao. Vì vậy, nếu khoảng 50% mạng trả về 0, mạng sẽ gọn hơn, tính toán hiệu quả và nhanh hơn.

Nhưng cũng do đường nằm ngang, độ dốc trong vùng đó sẽ là 0. Điều đó được gọi là dying ReLU khiến các neuron chết và không phản hồi làm cho một phần đáng kể của mạng không hoạt động.

Chúng ta có các biến thể trong ReLU để giải quyết vấn đề đó bằng cách biến đường ngang thành thành phần không nằm ngang. Ví dụ: cho x nhỏ hơn 0 sẽ biến nó thành một đường nghiêng nhẹ để độ dốc sẽ khác 0 và có thể phục hồi trong quá trình huấn luyện.

Có một vấn đề với các hàm kích hoạt đó là nếu đầu vào của neuron luôn lớn hơn hoặc nhỏ hơn giá trị ngưỡng, ví dụ với ReLU là 0, chức năng kích hoạt lúc này là vô dụng. Chúng ta có phương pháp mean-substraction, được thực hiện bằng cách trừ đi giá trị trung bình từng chiều như công thức dưới đây [9]. Theo cách đó, dữ liệu của chúng ta sẽ trở về trạng thái zero-mean.

1. filter = input\_image\_data[0]
2. filter -= filter.mean()

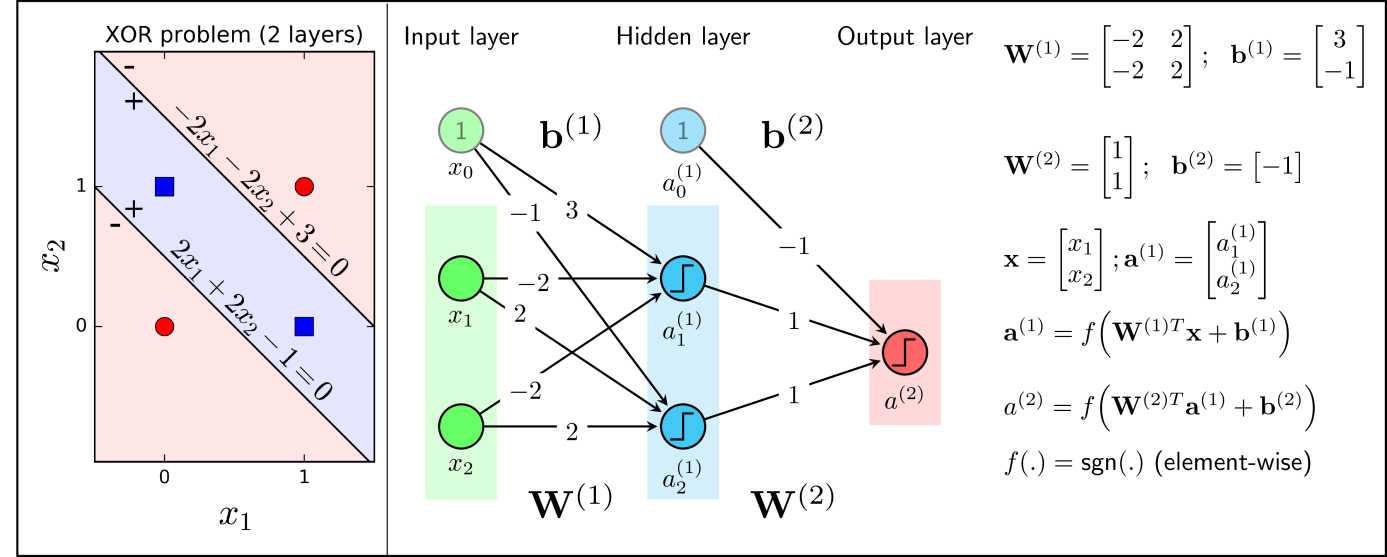


*Hình 3.2.6:* *PLA biểu diễn các hàm logic cơ bản [1]*

Phân tích về tầm quan trọng của việc xếp chồng các layer, chúng ta xem xét một ví dụ từ bài viết [1], một perceptron với hàm kích hoạt là hàm sgn giải các bài toán logic cơ bản: NO, AND, OR và XOR. Để có thể sử dụng PLA (Perceptron Learning Algorithm) có đầu ra là 0 hoặc 1, chúng ta thay thế đầu ra 0 của các hàm này thành -1, vì vậy trong hàng đầu tiên, hình vuông màu xanh có nhãn 1 và hình tròn màu đỏ có nhãn -1. Hàng thứ hai là mô hình perceptron với các hệ số tương ứng.

Ví dụ với hàm NOT, khi = 0, ta có . Khi = 1, . Cả hai trường hợp, dự đoán đều giống với ground truth. Tương tự, ta có thể kiểm chứng với hàm AND và OR.

Chúng ta thấy rằng, các hàm NOT, AND và OR có thể phân tách tuyến tính, một đường thẳng có thể chia đầu ra thành hai nhóm, nhưng với hàm XOR, vấn đề sẽ không còn đơn giản: nó không thể phân tách tuyến tính. Đó là trường hợp của underfitting và sự cần thiết của việc chồng các layer.



*Hình 3.2.7:* *Multi-layer perceptron biểu diễn hàm XOR [1]*

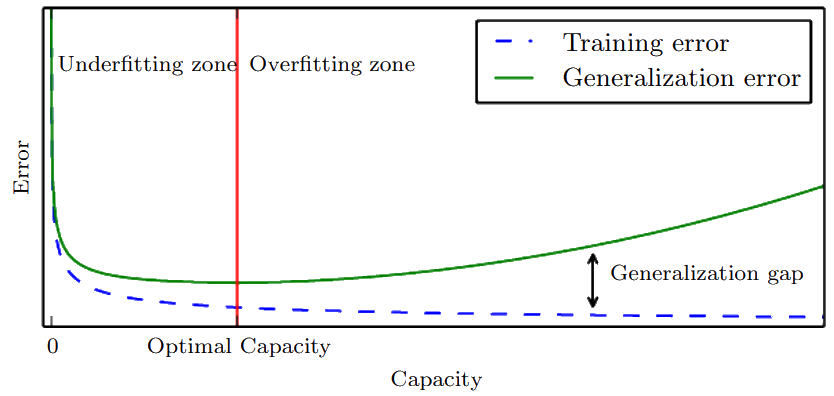
Chúng ta thấy rằng vấn đề đã được giải quyết bằng cách thêm một đường thẳng. Tại layer ẩn, output biểu diễn đường thẳng , output biểu diễn đường thẳng . Bởi vì hàm XOR chỉ cho một output, vì thế ta đưa and vào một layer nữa, output của layer này chính là kết quả cần tìm. Mô hình multi-layer perceptron với một layer ẩn như trên còn được gọi là mạng FF. Khi thêm nhiều layer ẩn cho mạng FF, ta được deep feed forward.

Như vậy, underfitting xảy ra khi mạng của chúng ta quá đơn giản và chúng ta gọi đó là high bias bởi vì dù chúng ta điều chỉnh mô hình như thế nào, nó cũng không thể học đủ các mẫu trong tập huấn luyện để có thể dự đoán với độ tin cậy đủ cao. Trong trường hợp trên, multi-layer perceptron đã giải quyết được vấn đề của underfitting, tuy nhiên không phải lúc nào việc tăng dung tích của mạng cũng phát huy tác dụng.



*Hình 3.2.8: Overfitting*

Trái ngược với underfitting là overfitting, xảy ra khi mô hình mạng quá linh hoạt và chúng ta gọi đó là high variance vì nó có thể thích ứng quá tốt với dữ liệu của tập huấn luyện, điều đó có nghĩa là mô hình của chúng ta học quá phù hợp với tập huấn luyện và không thể khái quát tốt để dự đoán cho tập kiểm thử. Chẳng hạn như ví dụ trên hình 3.2.8, ta chỉ cần một đường thẳng là đã có thể phân chia đường và lề đường, tuy nhiên tính phức tạp của mô hình đã cố gắng mô tả nhiễu hơn là dữ liệu cần học.



*Hình 3.2.9:* *Dung tích mạng thần kinh nhân tạo [8]*

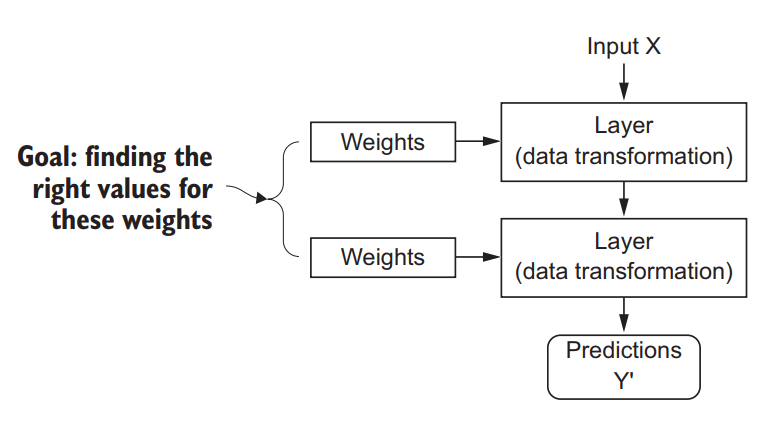
Chúng ta không thể có một mô hình vừa đơn giản mà lại vừa phức tạp. Vậy làm thế nào chúng ta có thể có được trạng thái cân bằng? Goodfellow trong [8] đưa ra biểu đồ (hình 3.2.9) cho thấy mối quan hệ giữa dung tích của mạng thần kinh nhân tạo và tỉ lệ lỗi. Mục tiêu của chúng ta khi huấn luyện mô hình NN là giảm lỗi huấn luyện hết mức có thể nhưng vẫn đảm bảo khoảng cách giữa lỗi huấn luyện và lỗi tổng quát nhỏ.

Tuy nhiên để đạt được trạng thái cân bằng yêu vẫn cầu người huấn luyện mạng phải có kinh nghiệm với mô hình và nắm rõ mức độ underfitting hay overfitting như thế nào, vì giảm dung tích mạng neuron có thể làm giảm overfitting nhưng nó cũng có thể đem lại vấn đề underfitting.

Thay vì cố gắng đạt được trạng thái hoàn hảo của mô hình, chúng ta có thể chấp nhận mô hình dung lượng lớn một chút, sau đó sử dụng các phương pháp regularization để giảm overfitting. Chúng ta sẽ phân tích chi tiết các phương pháp này trong phần tiếp theo.

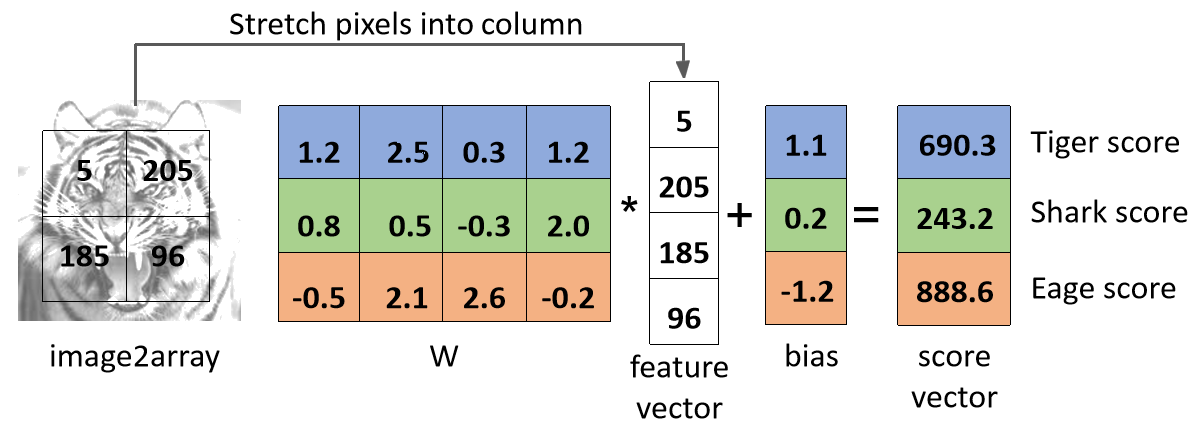
## Hàm mất mát và tối ưu hóa

### Vector điểm số



*Hình 3.3.1: “Học” trong ngữ nghĩa của “học sâu” - bước 1 [7]*

Trong quá trình học, DNN sau khi nhận một ảnh đầu vào, output của nó là một vectơ điểm số, mỗi điểm số tượng trưng cho một nhãn đối tượng; nhãn có số điểm cao nhất biểu thị nhãn có độ tin cậy cao nhất xuất hiện trong ảnh.

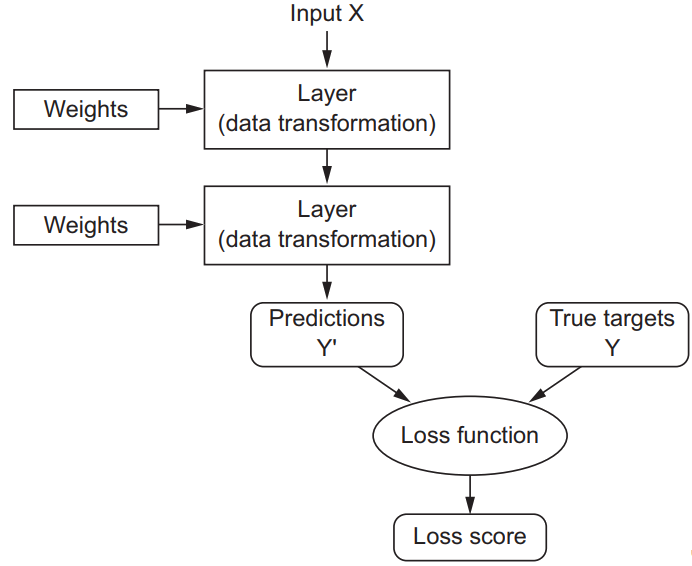


*Hình 3.3.2: Vector điểm số*

Hình 3.3.2 là một ví dụ cho giai đoạn học này: một ảnh đầu vào có 4 pixel và phân loại 3 nhãn. Mục tiêu của việc huấn luyện là xác định các trọng số để tối đa hóa điểm số của nhãn đúng và tối thiểu hóa điểm số của các nhãn sai (nhãn đúng là nhãn trùng với ground truth của ảnh, ngược lại là nhãn sai).

Mạng FF chỉ làm việc trên ảnh xám, vì vậy chúng ta phải chuyển đổi ảnh màu thành ảnh xám. Khi chuyển đổi ảnh sang ma trận, mỗi pixel có giá trị từ 0 đến 255, kéo dài nó thành cột chúng ta có vectơ đặc trưng. Sau đó đưa nó vào DNN và chúng ta có vectơ điểm số ở đầu ra. Nếu vẫn còn các layer sau khi nhận được vectơ điểm số, chúng ta sẽ áp dụng chức năng kích hoạt vào vectơ điểm và đưa nó tới layer kế tiếp, cho đến khi đến layer cuối cùng. Lúc này vectơ điểm là dự đoán của DNN về khả năng xuất hiện các đối tượng xuất hiện trong ảnh.

### Hàm mất mát



*Hình 3.3.3: “Học” trong ngữ nghĩa của “học sâu” - bước 2 [7]*

Sau khi đã có dự đoán, ta so sánh chúng với ground truth của ảnh đầu vào. Khoảng cách giữa điểm chính xác và điểm do DNN dự đoán dựa trên các trọng số được gọi là loss (L). Do đó, mục tiêu của việc đào tạo DNN là tìm ra một tập các trọng số để giảm thiểu loss trung bình trên toàn bộ tập dữ liệu huấn luyện.

(3.3.1)

Công thức hingle loss (SVM loss) 3.3.1 [23] có nghĩa là khi điểm số của nhãn được phân loại đúng là lớn nhất thì loss là 0, ngược lại thì lấy điểm số của nhãn sai trừ điểm số của nhãn đúng. Nhưng nhìn vào số Δ được thêm vào độ chênh lệch giữa điểm của nhãn sai và nhãn đúng, có ý nghĩa gì khi thêm một con số khác 0? Trong tình huống này nếu Δ = 0, thì chúng ta cần phải tối ưu hàm loss, nhưng chúng ta muốn có một khoảng cách giữa điểm số của các nhãn không đúng và nhãn đúng. Cuối cùng, chúng ta có giá trị loss trên toàn bộ dữ liệu là trung bình cộng của các hàm loss trên.

(3.3.2)

Không giống như SVM với việc tính toán điểm số không được hiệu chuẩn và không dễ dàng giải thích cho các nhãn được dự đoán, softmax (công thức 3.3.2 [23]) cho phép chúng ta tính toán xác suất cho tất cả các nhãn. Softmax thường được sử dụng làm bộ phân loại cho DNN.

Khi sử dụng softmax, chúng ta thường kết hợp nó với phương pháp one-hot encoding. Trong one-hot chỉ các vị trí đường chéo của ma trận có giá trị 1 và phần còn lại đều bằng 0, do đó không có sự giả định về sự tương đồng giữa các nhãn. Ví dụ: nếu chúng ta có vector điểm đầu ra là [0,7, 0,2, 0,1], điểm lớn nhất sẽ được đặt là 1 và điểm khác sẽ là 0 thì vector này có thể được chuyển đổi thành [1, 0, 0], sau đó so sánh với ground truth trong one-hot encoding, ví dụ [“sư tử”, “hổ”, “đại bàng”], ta chọn “sư tử” là dự đoán cuối cùng.

(3.3.3)

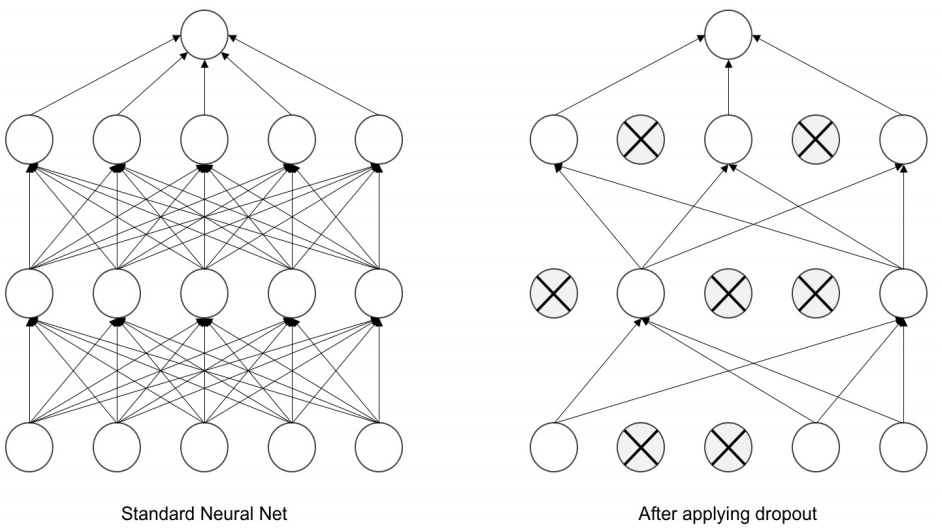
Softmax và SVM là hai trong số các bộ phân loại phổ biến nhất. Các hàm mất mà chúng ta vừa xem xét ở dạng rút gọn, trong thực tế, dạng chính thức được thêm vào một giá trị như công thức 3.3.3 [23]. Về cơ bản, chuẩn hóa R (W) là việc thay đổi mô hình một chút để tránh bị overfitting trong khi vẫn giữ được tính tổng quát của nó. Cụ thể hơn, chúng ta sẽ tìm cách chuyển giải pháp của vấn đề tối ưu sang một hàm gần nó. Hướng di chuyển sẽ là hướng để làm cho mô hình ít phức tạp hơn mặc dù giá trị của hàm mất mát tăng lên một chút.

(3.3.4)

Chuẩn hóa L2 (weight decay) là hình thức chuẩn hóa phổ biến nhất (công thức 3.3.4 [23]). Mục đích của nó là phạt một mức độ bình phương của tất cả các tham số trong ma trận.

Ý nghĩa của L2 chính là việc phạt các trọng số lớn có xu hướng gây nhiễu, bởi vì không thể có thành phần nào mang tính quyết định đến toàn bộ các yếu tố để ảnh hưởng đến kết quả là điểm số. Một ví dụ từ [23], giả sử ta có vector và hai vector trọng số , . Do đó tức cả hai vector trọng số sẽ cho cùng một kết quả điểm số, nhưng L2 phạt một mức là 1.0 trong khi chỉ bị phạt 0.25. Do đó được ưu tiên hơn vì giá trị cần chuẩn hóa của nó thấp hơn. Như vậy các vector trọng số nhỏ hơn và khuếch tán hơn được ưu tiên hơn thay vì một vài giá trị đầu vào rất lớn, điều đó có ý nghĩa quan trọng trong cải thiện hiệu suất tổng quát hóa của mô hình cũng như giảm overfitting.

Chúng ta có hai lưu ý trong việc chuẩn hóa: Thứ nhất, không giống như các trọng số, các bias không kiểm soát cường độ ảnh hưởng của kích thước đầu vào. Do đó, thông thường ta chỉ cần chuẩn hóa các trọng số W chứ không cần chuẩn hóa các bias b. Thứ hai, cường độ chuẩn hóa cần nhỏ đối với hàm mất mát để hàm chuẩn hóa không quan trọng bằng hàm mất chính.



*Hình 3.3.4: Drop-out [8]*

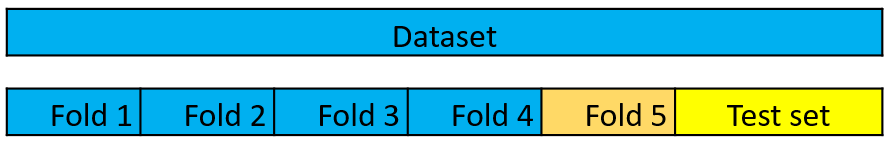
Chúng ta cũng có thể thực hiện chuẩn hóa bằng cách drop-out (DO): chỉ giữ lại một lượng neuron hoạt động với một xác suất p.

Early stopping cũng giúp chuẩn hóa, chúng ta có ba cách sau:

+ Cách thứ nhất: Xác định trước số epoch sẽ huấn luyện (số lượng epoch là số lần thuật toán học tập duyệt qua toàn bộ tập dữ liệu huấn luyện), cách này giúp đạt được mục đích early stopping nếu chúng ta đặt số epoch nhỏ, nhưng điều đó dễ dẫn đến việc huấn luyện chưa làm giảm loss đến vùng cực tiểu, mô hình có thể bị underfitted và không thể dự đoán hiệu quả trên tập dữ liệu kiểm tra.

+ Cách thứ hai: Dừng khi cập nhật của loss hoặc accuracy quá nhỏ, theo cách này chắc chắn tổn thất được giảm thiểu, giảm được các vòng lặp vô ích.

+ Cách thứ ba: Chiến lược tập dữ liệu xác thực: Chúng ta chia dữ liệu huấn luyện thành tập huấn luyện và tập xác thực với tỷ lệ 2: 1, 3: 1, v.v. Chúng ta chỉ huấn luyện với tập huấn luyện và đánh giá lỗi dựa trên tập xác thực một lần trong một khoảng thời gian, ví dụ mỗi 5 epochs. Sau đó dừng huấn luyện ngay khi lỗi trên tập xác nhận cao hơn lần kiểm tra ngay trước đó. Chúng ta sử dụng các trọng số mà mạng đã có trong bước trước đó làm kết quả của quá trình huấn luyện.

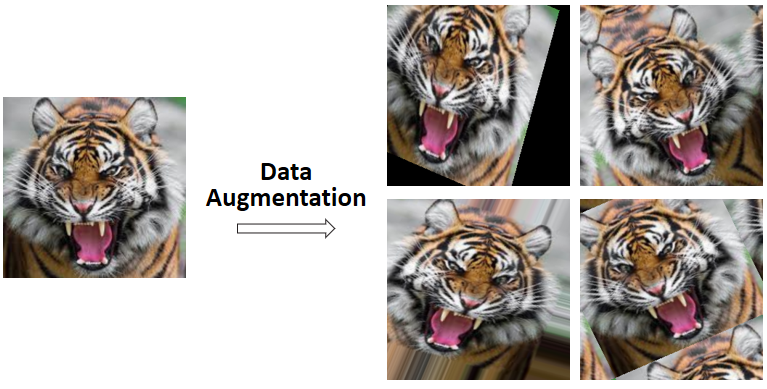


*Hình 3.3.5: Cross Validation*

Với cách thứ ba, ta còn có biến thể là áp dụng xác thực chéo, chúng ta chia ngẫu nhiên toàn bộ dữ liệu đào tạo thành K tập con, giá trị của K phụ thuộc vào kích thước tập dữ liệu. Giá trị cao hơn của K dẫn đến mô hình ít bị bias, nhưng với giá trị K lớn có thể dẫn đến tập xác thực bị overfitted.

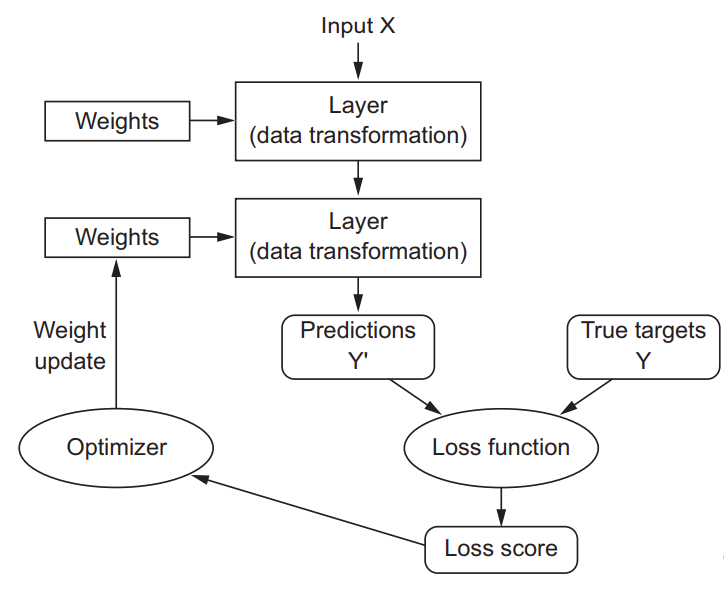
Sau đó, chúng ta huấn luyện mô hình bằng cách sử dụng các tập con K - 1 và xác thực mô hình bằng cách sử dụng các tập con còn lại. Ghi lại lỗi (hoặc accuracy). Chúng ta lặp lại quá trình này cho đến khi vượt qua tất cả K tập con. Sau đó lấy điểm trung bình của loss được ghi. Đó sẽ là chỉ số hiệu suất của mô hình.

Một phương pháp khác để chuẩn hóa là data augmentation, đó là một loại phương pháp chuẩn hóa có thể được thực hiện bằng cách biến đổi dữ liệu trên tập huấn luyện, chẳng hạn như phóng to, thu nhỏ, xoay, dịch chuyển lên xuống trái phải, lật hoặc chuyển màu. Cách này không những đem lại lợi ích giảm overfitting mà còn tăng kích thước bộ dữ liệu huấn luyện.



*Hình 3.3.6: Data Augmentation*

### Tối ưu hóa



*Hình 3.3.7: “Học” trong ngữ nghĩa của “học sâu” - bước 3 [7]*

Sau khi đã có loss, chúng ta quay lại để điều chỉnh trọng số. Việc điều chỉnh là công việc của thuật toán tối ưu hóa, thực hiện thuật toán back-propagation (lan truyền ngược), tính toán dựa trên quy tắc chain rule, hoạt động bằng cách chuyển các giá trị ngược qua mạng để tính toán mức độ ảnh hưởng của loss theo các trọng số. Quá trính feed forward và back-propagation cứ thế lặp đi lặp lại cho tới khi loss đạt tới khu vực tiệm cận của cực tiểu toàn phần.

W += - lr \* dW (3.3.5)

Trong đó:

W: Ma trận trọng số.

lr: Learning rate (tốc độ học).

dW: Đạo hàm của hàm mất mát theo trọng số.

Khi huấn luyện một mạng, các trọng số có thể được cập nhật theo thuật toán leo đồi. Trong công thức 3.3.5 [24], dấu trừ cho thấy rằng chúng ta phải đi ngược lại với đạo hàm, đó là lý do tại sao chúng ta gọi phương thức này là gradient descent (GD) với gradient nghĩa là đạo hàm, descent nghĩa là đi ngược, hạ xuống. Bội số của loss với từng trọng số, là đạo hàm riêng của loss đối với trọng số, được sử dụng để cập nhật trọng số. Đạo hàm này cho biết trọng số sẽ thay đổi như thế nào để giảm loss.

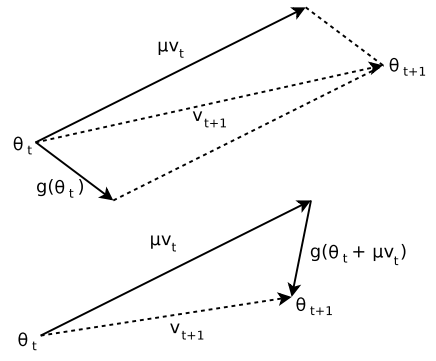
Chúng ta có một số đánh giá về phương pháp tối ưu hóa này. Đầu tiên, GD bị ảnh hưởng bởi điểm bắt đầu. Nếu có các khu vực có đạo hàm quá nhỏ, thuật toán sẽ bị chậm lại đáng kể.

Thứ hai, GD bị ảnh hưởng bởi tốc độ học. Nếu chúng ta đặt tốc độ học quá thấp, quá trình hội tụ cần rất nhiều vòng lặp để đi được tới cực tiểu toàn phần. Ngược lại, thuật toán không thể đưa loss tới khu vực cực tiểu toàn phần.

Thứ ba, việc thêm động lượng có thể đưa thuật toán vượt qua cực tiểu cục bộ để đi tới cực tiểu toàn phần. Nhưng nhược điểm của phương pháp này là do động lực, chúng ta cần rất nhiều lần lặp vô ích.

(3.3.6)

Vì vậy, gia tốc Nesterov theo công thức 3.3.6, có thể được khái niệm hóa như một bản cập nhật chính xác cho động lượng cho phép chúng ta có được một dự đoán gần đúng các tham số của chúng ta sẽ ở sau bản cập nhật.



*Hình 3.3.8: Nesterov Accelerated Gradient (NAG) [20]*

Khi nói về GD, chúng ta thường đề cập đến batch GD. Batch có nghĩa là tất cả dữ liệu, tức là ta sử dụng tất cả các điểm dữ liệu trong một lần cập nhật trọng số, điều đó có nghĩa là nếu chúng ta có một bộ dữ liệu mười nghìn mẫu, thì chúng ta phải tính toán lại đạo hàm mười nghìn lần.

Trong Stochastic Gradient Descent (SGD), chúng ta sử dụng hàm mất mát với chỉ một mẫu ở mỗi lần lặp, thay vì lấy trung bình cộng của hàm mất mát trên toàn bộ dữ liệu.

Với SGD, ta cũng có một số điểm lưu ý: Sau mỗi epoch, chúng ta cần xáo trộn các mẫu huấn luyện để đảm bảo tính ngẫu nhiên. Bởi vì nó chỉ sử dụng một mẫu tại một thời điểm, đường dẫn đến cực tiểu toàn phần của nó sẽ bất ổn định hơn so với batch GD, nhưng chỉ sau một vài epoch, thuật toán đã đi vào vùng lân cận của cực tiểu toàn phần. Chúng ta có thể sử dụng mini-batch SGD tức SGD với n điểm dữ liệu (thay vì một mẫu nhưng ít hơn số mẫu của toàn bộ tập dữ liệu) ở mỗi lần lặp. Cuối cùng, chúng ta có thể áp dụng động lượng hoặc NAG vào SGD cũng như batch GD.

Khi so sánh phương pháp GD và biến thể của nó, ta có thể hình dung với việc lấy mẫu chiều cao, cân nặng các học sinh trong một trường tiểu học, rõ ràng việc lấy dữ liệu của toàn bộ học sinh là một việc tốn kém, nhưng một học sinh thì không thể đại diện cho toàn bộ học sinh, vì học sinh này có thể là một em có chiều cao khiêm tốn hoặc thừa cân, vì vậy cách tốt nhất là lấy mẫu là n học sinh, với n đủ để đại diện cho toàn bộ dữ liệu. Đó cũng là lí do mà mini-batch SGD thường được sử dụng trong huấn luyện học sâu.

SGD rất phổ biến, nhưng nếu tốc độ học tập khiến chúng ta gặp phân vân, Adagrad có thể xử lý vấn đề này. Tốc độ học 0,01 sẽ được đặt làm mặc định và sau đó thuật toán sẽ cập nhật nó như sau:

1. cache += (dW \*\* 2)
2. W += -lr \* dW / (np.sqrt(cache) + eps)

Hạn chế là ở mỗi mini-batch, bình phương đạo hàm được tích lũy trong mẫu số, khiến cho tốc độ học tập trở nên quá nhỏ và mạng sẽ không học được gì. Một trong những cách để ngăn chặn điều đó là RMSprop:

1. cache = decay\_rate \* cache + (1 - decay\_rate) \* (dW \*\* 2)
2. W += -lr \* dW / (np.sqrt(cache) + eps)

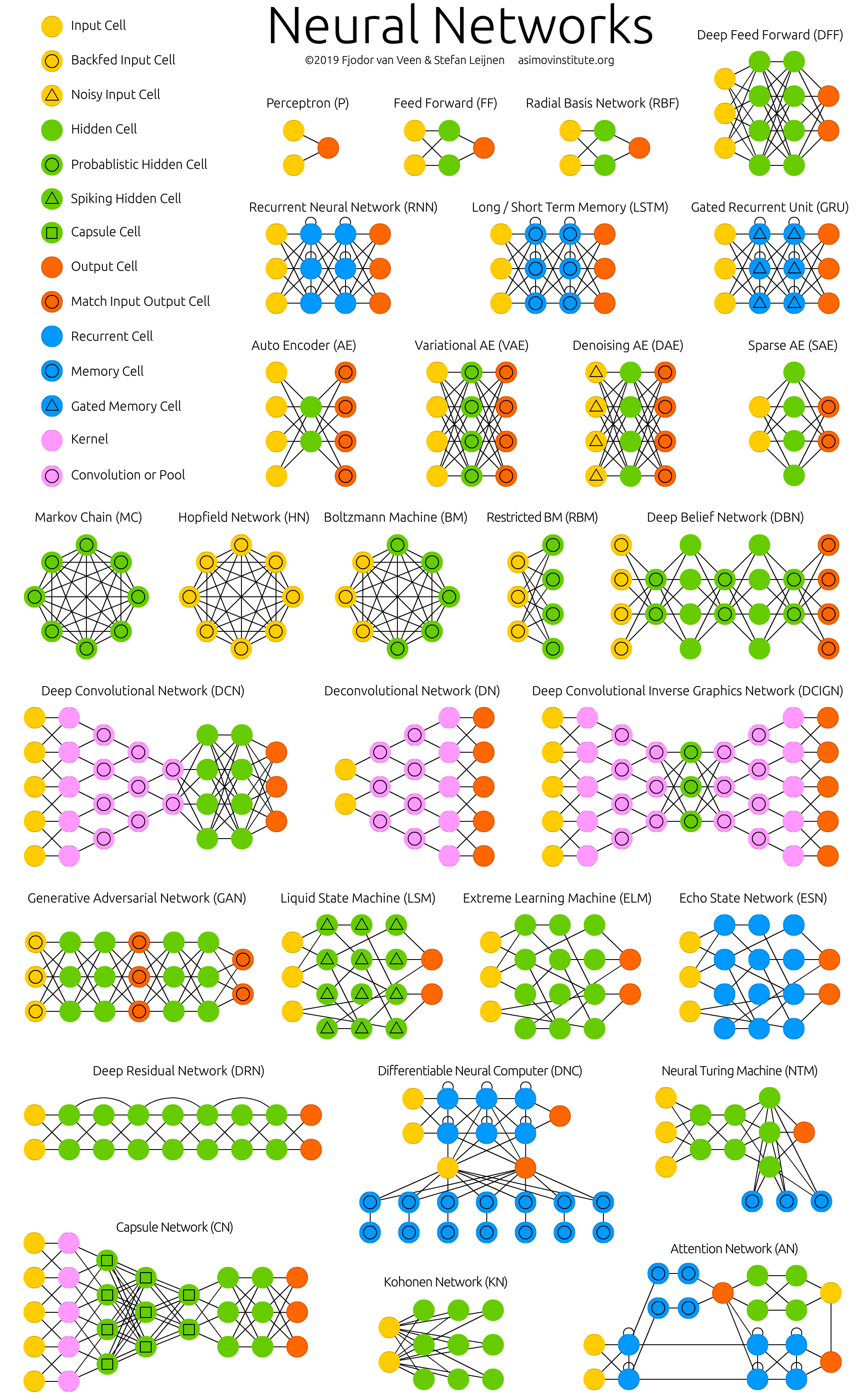
Bằng cách đó, chúng ta giới hạn sự tích lũy của bình phương đạo hàm đến một giới hạn nhất định. Khi thêm động lượng vào RMSprop, chúng ta có Adam:

1. m = beta1 \* m + (1 - beta1) \* dW
2. v = beta2 \* v + (1 - beta2) \* (dW \*\* 2)
3. W += -lr \* m / (np.sqrt(v) + eps)

Theo nhiều nghiên cứu, ví dụ [14], [15], Adam là phương pháp tối ưu tốt nhất cho tới thời điểm hiện tại. Tuy nhiên theo Goodfellow và các tác giả trong [8], việc chọn lựa thuật toán nào phụ thuộc phần lớn vào người dùng với thuật toán, vì lý do để dễ dàng trong việc điều chỉnh các hyperparameter (các tham số ban đầu như tốc độ học, batch size, số epochs,…).

## Convolutional Neural Networks (CNNs)

### Cấu trúc của CNNs



*Hình 3.4.1: Khu vườn mạng thần kinh nhân tạo [25]*

Ta biết rằng có rất nhiều loại NN, chúng ta đã tìm hiểu về các loại mạng cơ bản nhất là perceptron, feed forward và deep feed forward. Bởi vì DL là một lĩnh vực lớn và chúng ta không thể có được mọi thứ trong một nghiên cứu, do đó chúng ta sẽ so sánh đôi chút về chúng và chọn ra một trong số chúng và phân tích chi tiết hơn, để chúng ta có thể áp dụng vào giải quyết những vấn đề thực tiễn. Phần so sánh dưới đây được lấy cảm hứng từ [11].

|  |  |  |  |
| --- | --- | --- | --- |
| **Model properties** | **CNNs** | **DBNs/ DBMs** | **SDAs** |
| Generalization | ✔ | ✔ | ✔ |
| Unsupervised learning | ✖ | ✔ | ✔ |
| Training efficiency | ✖ | ✖ | ✔ |
| Feature learning | ✔ | ✖ | ✖ |
| Scale/rotation/translation invariance | ✔ | ✖ | ✖ |

*Bảng 3.4.1: So sánh một số neural network cho các vấn đề của thị giác máy tính*

Trên đây là so sánh về các mạng Convolutional Neural Networks (CNNs), Deep Boltzmann Machines (DBMs), Deep Belief Networks (DBNs) và Stack Denoising Auto-encoders (SDAs).

- Thuộc tính đầu tiên - tính khái quát hóa là khả năng đơn giản hóa mô hình để có thể dự đoán trên dữ liệu chưa được nhìn thấy (dữ liệu chưa được huấn luyện), tất cả các mạng được đem so sánh đều có khả năng này.

- Thuộc tính thứ hai - học tập không giám sát: CNNs dựa vào tính khả dụng của dữ liệu huấn luyện, nghĩa là dữ liệu đào tạo được gắn nhãn, trong khi DBNs hay DBMs và SDAs không chịu sự giới hạn này và có thể học không giám sát.

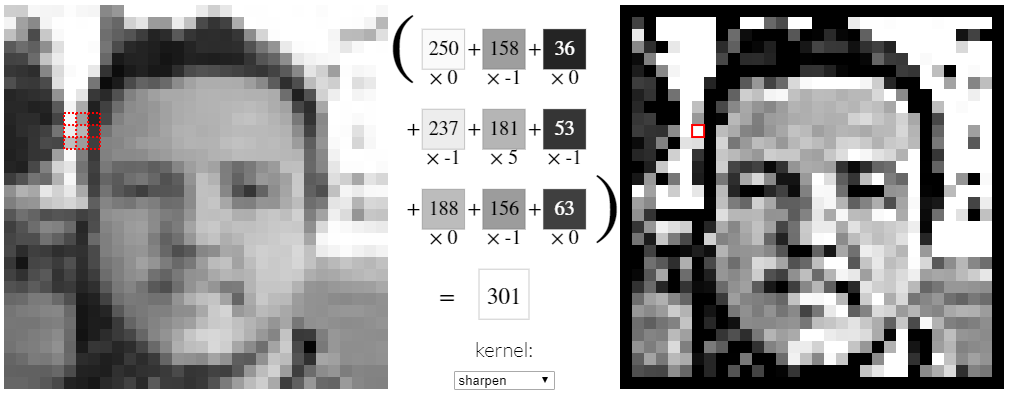
- Thuộc tính thứ ba - hiệu quả đào tạo: Denoising Auto-encoders có thể truy xuất đầu vào chính xác từ một phiên bản bị nhiễu, do đó dẫn đến mạng có thể hiểu được cấu trúc của phân phối đầu vào. Vì lý do này, SDAs có thể huấn luyện theo thời gian thực, trong khi các quy trình huấn luyện của CNNs và DBNs / DBMs tốn nhiều thời gian.

- Một khía cạnh tích cực chính của CNNs là tính năng học đặc trưng. Điều đó có nghĩa là trong khi các loại mạng khác yêu cầu trích xuất đặc trưng thủ công, trong CNNs, các đặc trưng được tự động học.

- Một ưu điểm khác của CNNs là chúng vẫn có thể nhận ra đối tượng đối với các phép biến đổi như dịch chuyển, phóng to thu nhỏ, xoay ... nó cho phép trừu tượng hóa danh tính hoặc danh mục của đối tượng từ các chi tiết cụ thể của đầu vào trực quan (ví dụ: vị trí / hướng tương đối của máy ảnh và vật).

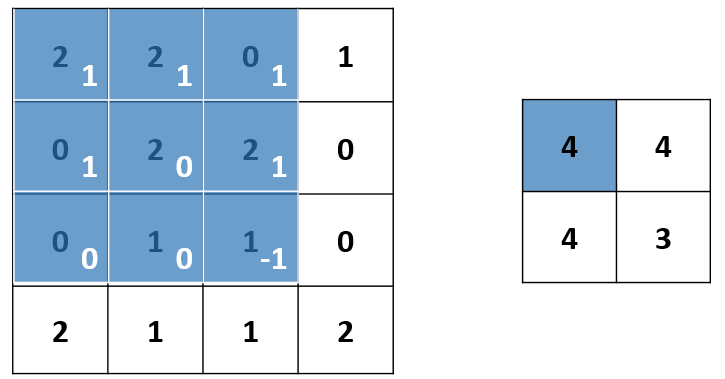
Tính năng học đặc trưng và tính năng làm việc trên các phép biến đổi đặc biệt quan trọng về các vấn đề thị giác máy tính như nhận dạng khuôn mặt. Đó cũng là lý do chính mà chúng ta nên chọn CNNs cho nghiên cứu này. Lưu ý rằng hầu hết các tập đoàn nổi tiếng đều đang sử dụng CNNs cho hệ thống nhận diện khuôn mặt của họ như Google, Facebook, Baidu, Apple,… vì thế, đó không phải là ý tưởng tồi khi chọn CNNs để khởi đầu cho việc nghiên cứu về deep learning.

Đầu tiên, CNN là gì? Nó giống như một mạng lưới thần kinh cơ bản như feed forward nhưng thêm các loại layer đặc biệt hoạt động hiệu quả trên ảnh. Vậy convolution hay tích chập có nghĩa là gì?



*Hình 3.4.2: Làm sắc nét ảnh với phương pháp tích chập [26]*

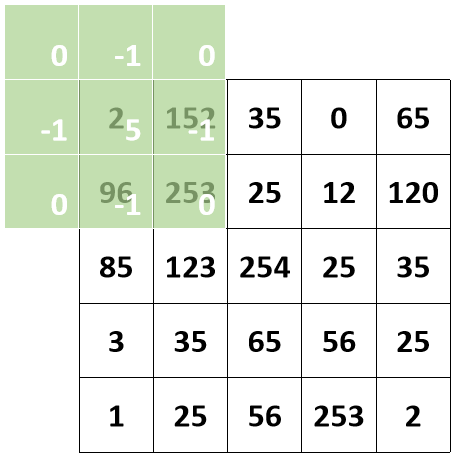
Khi chúng ta làm mờ, làm mịn hoặc phát hiện cạnh trong ảnh, ... chúng ta đang thực hiện tích chập trên ảnh. Tùy thuộc vào kernel trượt trên hình ảnh, chúng ta có đầu ra có tương ứng, ví dụ chúng ta có thể làm sắc nét ảnh bằng một kernel như trên hình 3.4.2, nhưng trong nhiệm vụ của DL, mục tiêu của chúng ta là tìm ra kernel này để mạng có thể phân biệt được các nhãn khác nhau.



*Hình 3.4.3: Convolution*

Để thực hiện tích chập, ta nhân hai ma trận từng phần tử với nhau (không phải là phép nhân hai ma trận), theo sau là phép tổng. Trong đó, một kernel có thể được hình dung như một ma trận nhỏ trượt qua, từ trái sang phải và từ trên xuống dưới, của ma trận đầu vào.

Trên hình 3.4.3, ta đang sử dụng kernel là ma trận và stride là 1 pixel mỗi lần trượt, chúng ta có thể sử dụng stride với giá trị khác hoặc kích thước kernel với chiều rộng và chiều cao là một số lẻ khác (vì chúng ta đang sử dụng trung tâm của kernel để trượt, một kernel có kích thước chẵn sẽ không có trung tâm) nhưng hầu hết các kernel được áp dụng cho học sâu với CNNs là ma trận vuông, điều đó cho phép chúng ta tận dụng các thư viện đại số tuyến tính đã được tối ưu hóa để hoạt động hiệu quả nhất trên ma trận vuông.

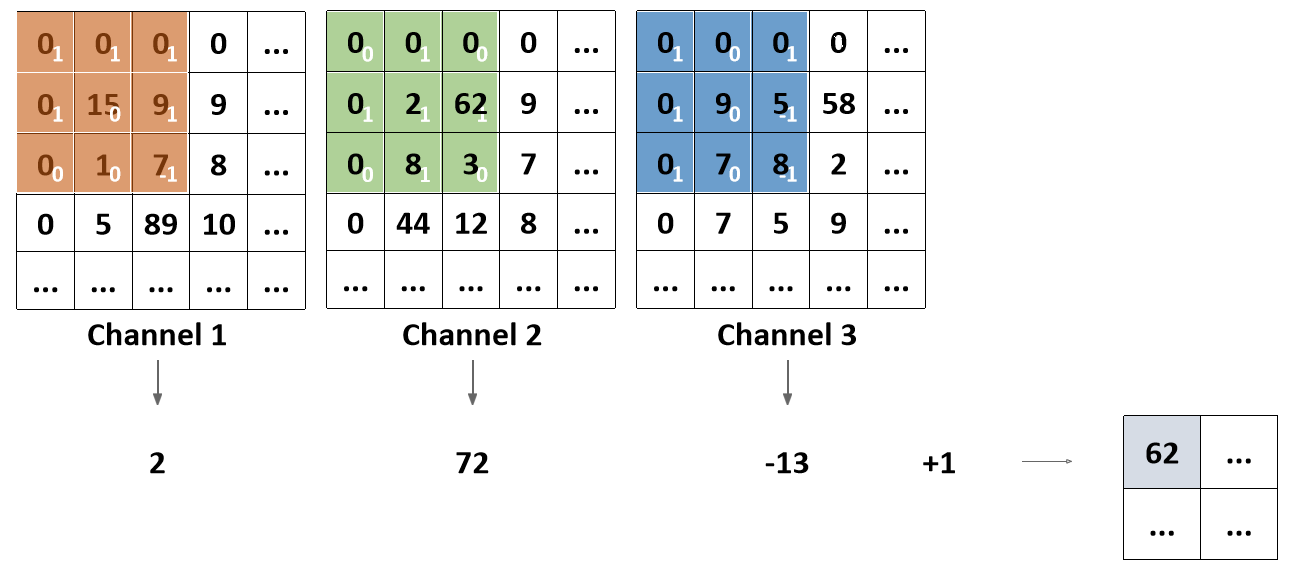


*Hình 3.4.4: Vấn đề khi không có padding*

Trong khi thực hiện tích chập, nếu hình ảnh không có padding như hình 3.4.4 thì ma trận đầu ra sẽ bị mất cạnh, điều đó có thể thấy rõ trên hình 3.4.2. Đó là bởi vì không có giá trị pixel hình ảnh đầu vào cho hàng đầu tiên và cột đầu tiên của kernel. Chúng ta có thể khắc phục sự cố này bằng cách bắt đầu tích chập ở vị trí hợp lệ đầu tiên hoặc áp dụng padding với giá trị bằng 0 xung quanh ảnh đầu vào gọi là zero-padding.

Sau khi thực hiện tích chập trên ảnh vuông có chiều rộng W, kernel có chiều rộng K, stride S, padding P, ta được đầu ra có kích thước là .

Với loại ảnh trắng đen, độ sâu của ảnh là 1, nhưng với ảnh màu RGB (ảnh có 3 kênh màu đỏ-lục-lam), độ sâu lúc này là 3. Thực hiện tích chập trên ảnh màu, ta thường sử dụng định nghĩa filter thay vì kernel. Một filter cũng có chiều sâu như ảnh, nếu xét trên ảnh xám thì filter và kernel là một, tuy nhiên khi xét trên ảnh màu RGB, filter là tập 3 kernel tương ứng với 3 kênh màu của ảnh. Sử dụng bao nhiêu filter để học, chúng ta có bấy nhiêu activation map.



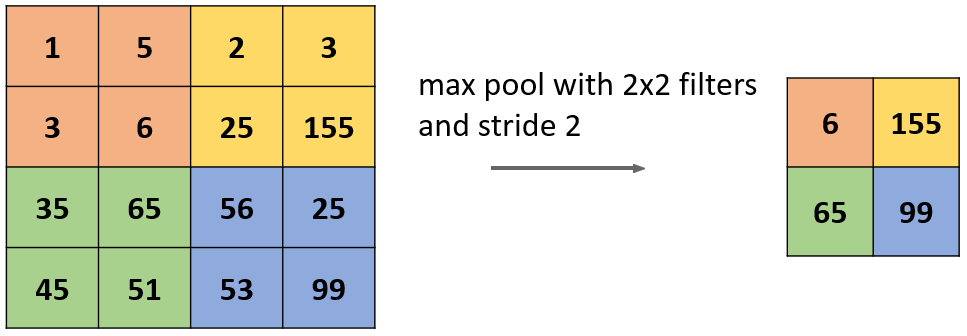
*Hình 3.4.5: Convolution layer*

Các kernel này bắt đầu với các giá trị ngẫu nhiên và mạng cập nhật (học) các giá trị kernel. Cách học này rất giống với cách chúng ta cập nhật ma trận trọng số trên mạng FF, nhưng thay vì mỗi trọng số chỉ được kết hợp với một đầu vào và một đầu ra, các kernel chia sẻ tham số trên toàn bộ layer mà nó thực hiện tích chập. Đó là lý do tại sao chúng ta sử dụng CNNs cho các nhiệm vụ của thị giác máy tính thay vì mạng FF cơ bản.

Chẳng hạn, ta có ảnh và chúng ta sử dụng 6 filter cho đầu ra . cho kết quả 12288 và bằng 11532. Trong layer này, mỗi kernel có kích thước nên mỗi filter có 9 tham số cộng với một tham số bias. Điều đó có nghĩa là chúng ta có 10 tham số cho mỗi filter và chúng ta có 6 filter nên tổng số tham số trong layer tích chập này là 6 lần 10, tương đương với 60 tham số.

Nếu chúng ta tạo ra một mạng FF thuần với layer đầu tiên có 12288 neuron đầu vào và 11532 neuron đầu ra. Và nếu các neuron này là FC (không có DO), thì các tham số số trong ma trận trọng số sẽ là được kết quả lớn hơn 141 triệu. Nếu chúng ta xác định một DNN với nhiều layer, lúc này lượng tham số trở nên quá lớn cho việc huấn luyện mạng.

Vì vậy, các layer tích chập đã chia sẻ tham số như thế nào? Hãy tưởng tượng trong FF thuần, chúng ta sẽ tìm các tham số để nhận diện toàn bộ một đối tượng. Cách mạng tích chập làm là nó học các đặc trưng cấp thấp như các cạnh và góc, sau đó kết hợp các đặc trưng này, chúng ta có thể nhận diện các đặc trưng ở mức độ cao hơn như mắt, tai, mũi. Các đặc trưng cấp cao lại dùng để nhận diện toàn đối tượng. Vì vậy, chúng ta có thể sử dụng cùng một tham số ở nhiều vị trí khác nhau trong hình ảnh đầu vào.

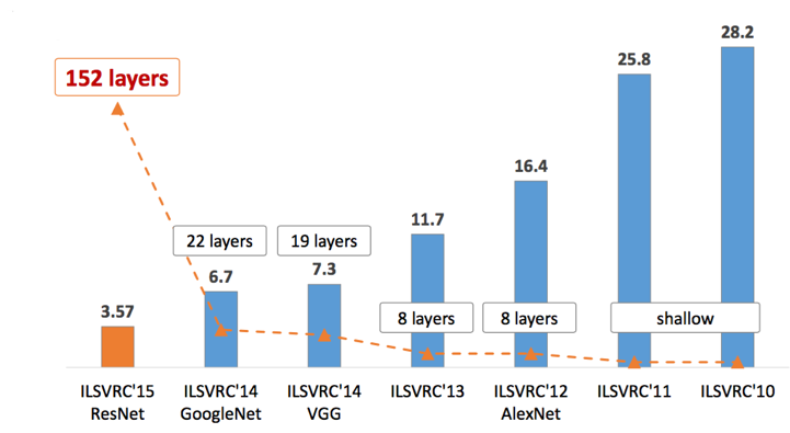


*Hình 3.4.6: Max Pooling layer*

Một loại layer khác được sử dụng trong CNNs là max pooling. Hình thức phổ biến nhất là max pooling với các kernel có kích thước được áp dụng với stride 2, giữ lại các giá trị lớn nhất và loại bỏ 75% các giá trị được kích hoạt. Chúng ta cũng có thể thực hiện average pooling bằng cách lấy giá trị trung bình thay vì giá trị lớn nhất. Cả hai loại pooling trên đều có mục đích thực hiện down-sampling nhằm giảm kích thước của đầu vào, nhưng với max pooling, việc mất thông tin cũng làm giảm overfitting. Mặc dù những ưu điểm này, các pooling layer có thể được thay thế bằng các layer tích chập với stride lớn hơn 1.

Để hiểu rõ hơn về CNNs, chúng ta sẽ nghiên cứu một số cấu trúc nổi tiếng của chúng. ImageNet là một cơ sở dữ liệu trực quan lớn với hơn 14 triệu ảnh, được thiết kế để sử dụng trong nghiên cứu phần mềm nhận dạng đối tượng trực quan. Dự án ImageNet tổ chức một cuộc thi phần mềm hàng năm, ImageNet Large Scale Visual Recognition Challenge (ILSVRC), nơi các chương trình phần mềm thi đấu để phân loại chính xác nhất 1000 đối tượng và cảnh vật.

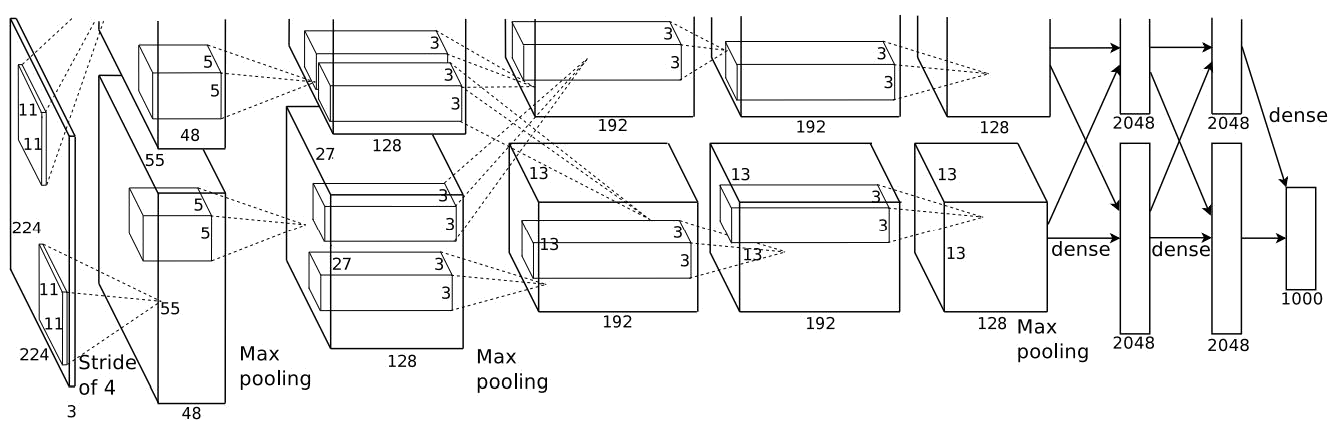
Do số lượng nhãn đối tượng khá lớn, thử thách ImageNet sử dụng top-5 error rate, chúng ta kiểm tra xem nhãn mục tiêu có phải là một trong 5 nhãn đầu tiên có độ tin cậy cao nhất hay không. Bằng cách đó, chúng ta có thể đánh giá hiệu quả đào tạo, nếu chúng ta có hai mô hình có cùng độ chính xác trên top-1 error rate, thì độ chính xác trên top-5 error rate sẽ xác định mô hình nào tốt hơn.



*Hình 3.4.7: Các đội chiến thắng cuộc thi ILSVRC [18]*

Các đội chiến thắng từ năm 2012 đều sử dụng mô hình mạng tích chập. Trong đó, chúng ta có thể thấy một cấu trúc thậm chí cho kết quả tốt hơn con người (5.1%).

* + - 1. **AlexNet**



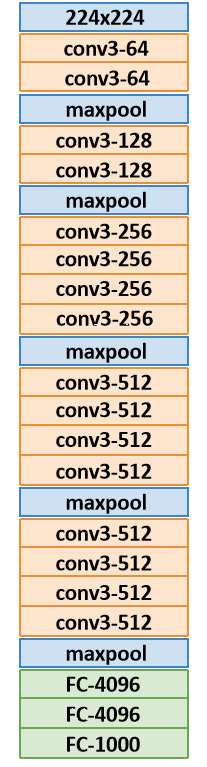
*Hình 3.4.8: Cấu trúc AlexNet [19]*

AlexNet là mô hình chung cơ bản cho các mô hình CNN, trong đó các khối layer tích chập và max pooling lặp đi lặp lại năm lần, theo sau là ba layer FC. Layer FC cuối cùng sử dụng hàm kích hoạt softmax và mỗi neuron để phân loại một đối tượng tiềm năng. Nhiệm vụ của ImageNet là phân loại 1000 đối tượng nên layer cuối cùng này có 1000 neuron.

Nếu để ý, chúng ta thấy rằng AlexNet yêu cầu ảnh đầu vào là , theo như [19], AlexNet sử dụng kích thước filter ở layer đầu tiên là 11, stride là 4 và không có padding, do , đây không phải là một số nguyên ứng với kích thước của một ma trận, vì thế kích thước có thể là do sự nhầm lẫn của tác giả. Ta có là một số nguyên, do đó có thể là một kích thước chính xác hơn cho cấu trúc AlexNet.

* + - 1. **VGGNet**

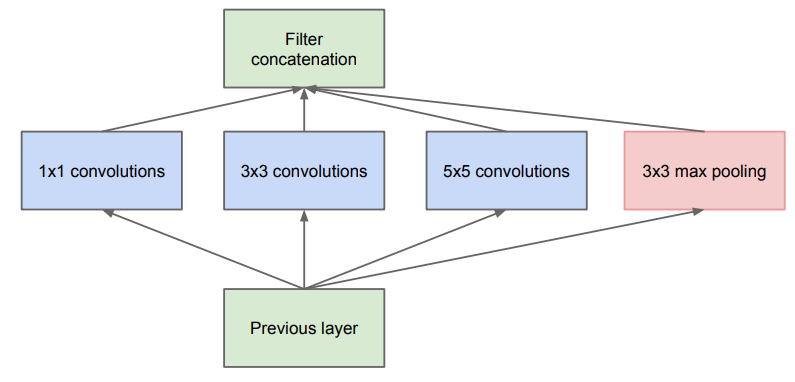
Một cấu trúc CNN khác được gọi là VGG19 (một trong biến thể của VGGNet [13]) rất giống với cấu trúc AlexNet, nhưng sâu hơn với 19 layer có khả năng học (các pooling layer chỉ có chức năng làm giảm kích thước đầu vào cho layer tiếp theo nên chúng thực chất không học gì cả hay nói cách khác chúng không có khả năng học). VGG19 chỉ sử dụng các layer tích chập và các max pooling layer .



*Hình 3.4.9: Cấu trúc VGG19*

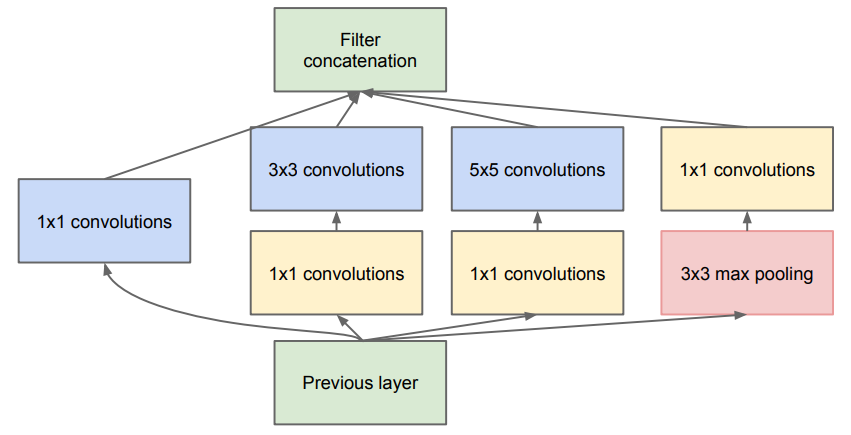
* + - 1. **GoogLeNet**

So sánh với các mạng truyền thống như AlexNet và VGGNet, chỉ sử dụng một chuỗi các layer xếp chồng lên nhau, GoogLeNet sử dụng các Inception module, có các nhánh. Nếu chúng ta gặp khó khăn về việc chọn lựa kích thước filter cho mạng tích chập, module này có thể giải quyết vấn đề đó bằng cách sử dụng nhiều kích thước filter.



*Hình 3.4.10: Inception module (phiên bản đơn giản) [20]*

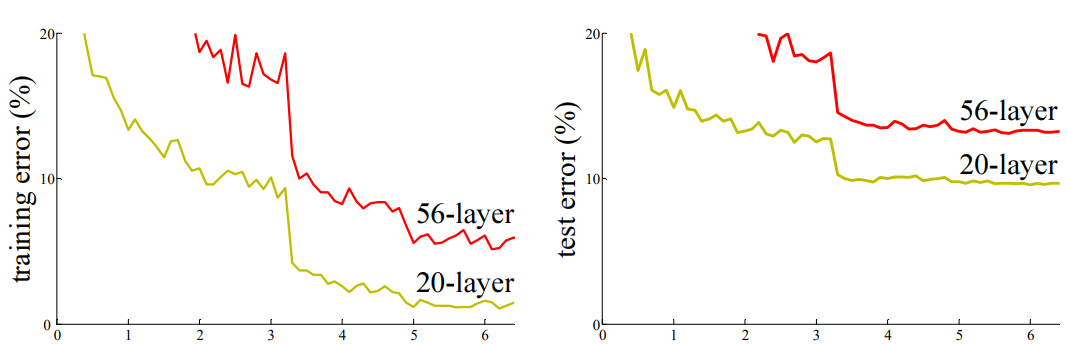
Do các khối này có nhiều nhánh nên chi phí tính toán sẽ tăng cao, các layer tích chập được thêm vào trước các layer tích chập lớn ( và ):



*Hình 3.4.11: Inception module with dimension reductions [20]*

Giả sử ta có một layer tích chập với đầu vào và đầu ra , nếu chúng ta sử dụng filter , thì chi phí tính toán sẽ là tương đương 2,4 triệu. Nếu chúng ta sử dụng một layer tích chập với 64 filter trước layer tích chập này, chi phí sẽ là tương đương 0,6 triệu. Chìa khóa để giảm kích thước là chúng ta thêm layer tích chập với số lượng filter nhỏ thay vì kết nối đầu vào trực tiếp với layer tích chập lớn.

* + - 1. **ResNet**

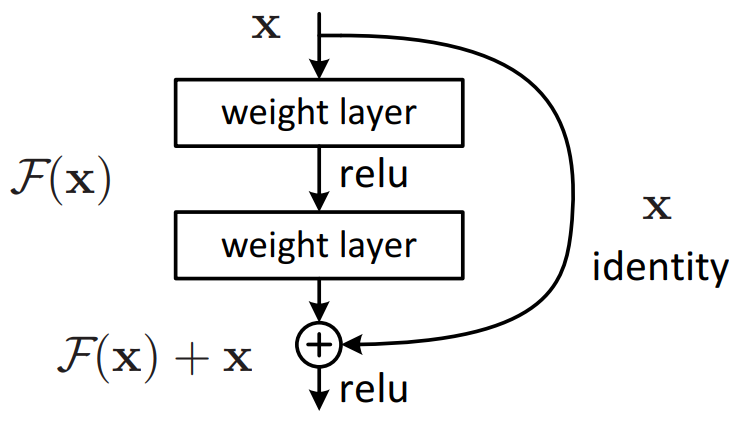


*Hình 3.4.12: Tỉ lệ lỗi của mạng với 20 và 56 layer trên Cifar-10 [18]*

Cho đến thời điểm này, chúng ta đã biết rằng các NN quá sâu có thể là nguyên nhân của việc overfitting, như chúng ta có thể thấy trên hình 3.4.12, nếu chúng ta xếp nhiều layer hơn, nó sẽ có tỷ lệ lỗi cao hơn trên cả dữ liệu huấn luyện và dữ liệu kiểm tra. Mạng sâu hơn không thể làm tốt hơn.

Nhưng nếu xem lại bảng xếp hạng về cuộc thi ILSVRC (hình 3.4.7), có vẻ như, chúng ta càng xếp nhiều layer, tỉ lệ lỗi càng giảm đến mức thấp hơn. Vì vậy, điều gì đã xảy ra khi chúng ta xếp quá nhiều layer? Giả thuyết mới trong DL nói rằng vấn đề nằm ở tối ưu hóa, các mô hình sâu hơn khó tối ưu hóa hơn.

Vấn đề đào tạo các mạng rất sâu đã được giảm bớt với sự ra đời của một loại layer mới trong NN - khối dư:



*Hình 3.4.13: Khối dư (Residual block) [18]*

Trong khối dư, ánh xạ định danh (identity mapping) không học bất kỳ tham số nào và chỉ được dùng để cộng đầu ra từ layer trước vào layer sau. Điều đó có nghĩa là nó sao chép các layer đã học từ mô hình nông hơn và đặt các layer bổ sung thành ánh xạ định danh. Bằng cách này, chúng ta sử dụng các layer mạng để huấn luyện với ánh xạ dư thay vì cố gắng huấn luyện trực tiếp để phù hợp với ánh xạ cơ bản mong muốn: huấn luyện thay vì một cách trực tiếp.

Toàn bộ cấu trúc ResNet:

+ Các khối dư.

+ Mỗi khối dư có hai layer tích chập .

+ Định kỳ mỗi sáu layer tích chập thì nhân đôi số lượng filter và thực hiện down-sampling bằng stride 2.

+ Ngoài các layer tích chập trong các khối dư, có thêm một layer tích chập đầu tiên ngay sau khi tiếp nhận ảnh đầu vào.

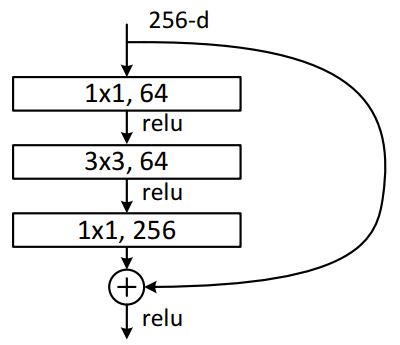
+ Chỉ có 2 pooling layer.

+ Loại bỏ các layer FC ngoại trừ layer duy nhất với 1000 neuron ở đầu ra cuối cùng cho mục đích phân loại.

+ Sử dụng 152 layer trong cuộc thi ILSVRC (số lượng layer là tùy ý, ta có thể thêm hoặc bớt các khối dư để thêm hay bớt số lượng layer).

Ngoài cấu trúc trên đây, sự thành công của ResNet không chỉ dựa trên các khối dư, nhưng còn phải kể đến sự trợ giúp của thuật toán khởi tạo trọng số (như Xavier / Glorot và MSRA) thay vì bắt đầu với các giá trị ngẫu nhiên của filter.

Trong phiên bản cập nhật của ResNet, khối dư thắt cổ chai có thể cải thiện hiệu quả để có thể học sâu với hơn 1000 layer. Trong phiên bản mới này, ResNet sử dụng ba layer tích chập. Các layer tích chập đầu tiên và cuối cùng sử dụng các filter có kích thước . Số lượng filter trong hai layer tích chập đầu tiên là 1/4 số lượng filter trong layer cuối cùng.



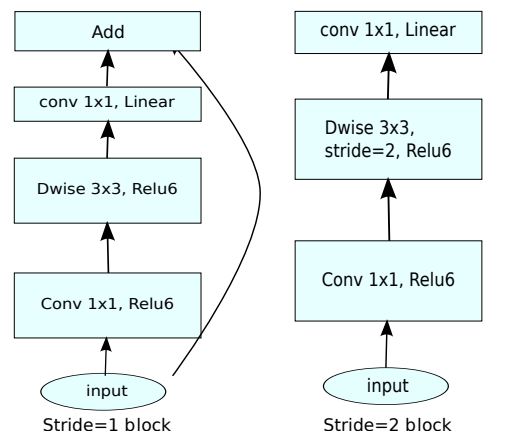
*Hình 3.4.14: Khối dư thắt cổ chai (Bottleneck residual block) [18]*

* + - 1. **MobileNetV2**

Sự thành công của các mô hình đã chiến thắng trong cuộc thi ILSVRC được đánh giá trên tỉ lệ lỗi, MobileNetV2 (MobileNet phiên bản 2) đã được tối ưu hóa cho thiết các bị di động. MobileNetV2 sử dụng khối dư thắt cổ chai như ResNet và ReLU6 để ngăn các giá trị được kích hoạt trở nên quá lớn:

(3.4.1)

Sự khác biệt trong các khối dư giữa ResNet và MobileNetV2 là ResNet sử dụng các layer tích chập truyền thống, MobileNetV2 sử dụng khối tích chập đặc biệt gọi là depthwise convolution:

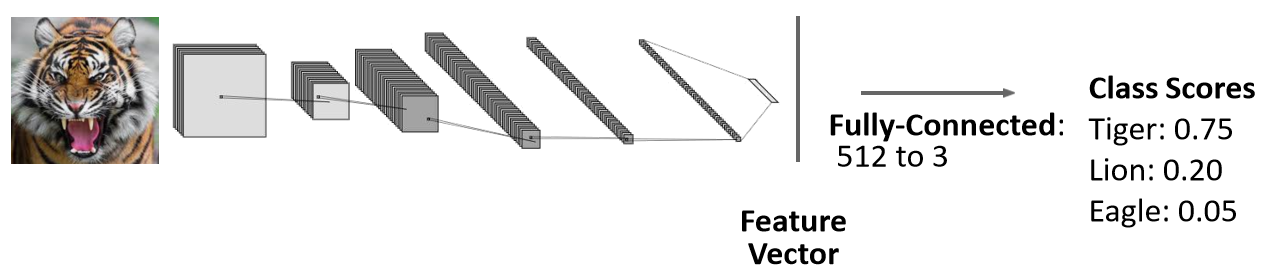


*Hình 3.4.15: Khối tích chập trong MobileNetV2 [12]*

Giả sử một layer tích chập truyền thống với kích thước đầu vào , số lượng channel C, kích thước filter là và D là số channel của đầu ra, chi phí tính toán của tích chập tiêu chuẩn sẽ là .

Trong depthwise convolution, tích chập được thực hiện độc lập cho từng channel đầu vào. Điều đó có nghĩa là số lượng channel đầu vào và đầu ra là như nhau và chi phí tính toán sẽ là .

Bây giờ chúng ta đã hiểu hơn về CNN với các layer và các cấu trúc giúp mạng xử lý hiệu quả trên ảnh, đó cũng là những kỳ tích tạo nên tiền đề sâu trong thị giác máy tính. Cách chúng ta áp dụng CNN để phân loại rất giống với FF thuần, mạng thông qua một hoặc vài layer FC cuối cùng đưa ra một vectơ điểm, quá trình tính toán hàm mất mát và tối ưu hóa cũng không khác với mạng FF thuần:



*Hình 3.4.16: CNN trong nhiệm vụ phân loại các đối tượng*

### Transfer learning với CNN

Như chúng ta đã đề cập, một trong những lý do tại sao chúng ta chọn DL là khả năng tái sử dụng. Bằng cách này, thay vì bắt đầu với giá trị khởi tạo ngẫu nhiên của các trọng số, vì hiếm khi ta có được bộ dữ liệu đủ kích thước, chúng ta có thể huấn luyện trước CNN trên một tập dữ liệu rất lớn, sau đó sử dụng CNN cho các nhiệm vụ khác.

Để áp dụng transfer learning, cách tiếp cận đầu tiên là sử dụng CNN như một phương pháp feature descriptor; chúng ta loại bỏ bộ phân loại cuối cùng và thay thế một bộ phân loại mới, sau đó huấn luyện lại trên bộ dữ liệu mới.

Cách tiếp cận thứ hai là tinh chỉnh các trọng số của mạng được đào tạo trước bằng cách tiếp tục thực hiện quá trình feed forward và back-propagation, để mạng có thể học thêm các mẫu đặc trưng từ dữ liệu mà nó chưa được huấn luyện trước đó.

|  |  |  |
| --- | --- | --- |
| Data size | Similar dataset | Different dataset |
| Smaller data | Fine-tune the output layers | Fine-tune the deeper layers |
| Bigger data | Fine-tune whole model | Train from scratch |

*Bảng 3.4.2: Chiến lược sử dụng phương pháp Transfer Learning [9]*

Có bao nhiêu layer cần để huấn luyện lại tùy thuộc vào kích thước của tập dữ liệu mới và sự tương đồng của nó so với tập dữ liệu gốc:

- Trường hợp đầu tiên, tập dữ liệu mới nhỏ và giống với tập dữ liệu gốc: Dữ liệu mới tương tự dữ liệu gốc, vì vậy chúng ta có thể có mô hình đã học gần như đủ các mẫu đặc trưng bộ dữ liệu mới này có, bây giờ ta chỉ cần một bộ phân loại mới phù hợp với bài toán cần giải quyết.

- Trường hợp thứ hai, tập dữ liệu mới là lớn và tương tự như tập dữ liệu gốc: Nếu chúng ta có một tập dữ liệu lớn, chúng ta không cần quan tâm đến overfitting, do đó, việc tinh chỉnh toàn bộ mạng có thể mang lại kết quả dự đoán tốt hơn.

- Trường hợp thứ ba, tập dữ liệu mới nhỏ nhưng rất khác so với tập dữ liệu gốc: Đây là trường hợp khó khăn nhất, vì chúng ta không có đủ dữ liệu để tinh chỉnh toàn bộ mạng và tập dữ liệu mới khác với tập dữ liệu gốc, nên mô hình có thể chưa học học đủ các mẫu đặc trưng để dự đoán trên tập dữ liệu mới. Do đó, chúng ta phải thử tinh chỉnh trên một số layer cuối cùng, sau đó tăng số lượng layer cần tinh chỉnh để tìm ra cách tốt nhất để huấn luyện lại mô hình. Khả năng thành công của trường hợp này không cao, ta thường cố gắng làm cho dữ liệu giống với ba trường hợp còn lại.

- Trường hợp cuối cùng, tập dữ liệu mới là lớn và rất khác so với tập dữ liệu gốc: Với một tập dữ liệu lớn, chúng ta hoàn toàn có thể huấn luyện từ đầu, nhưng rất đáng để thử với việc tinh chỉnh toàn bộ mạng thay vì sử dụng trọng số với giá trị khởi tạo ngẫu nhiên.

### Siamese Neural Network

Mạng Siamese là một loại NN xây dựng dựa trên CNN. Từ Siamese có nghĩa là được kết nối với nhau. Cấu trúc của mạng Siamese có thể được minh họa như hình dưới đây:



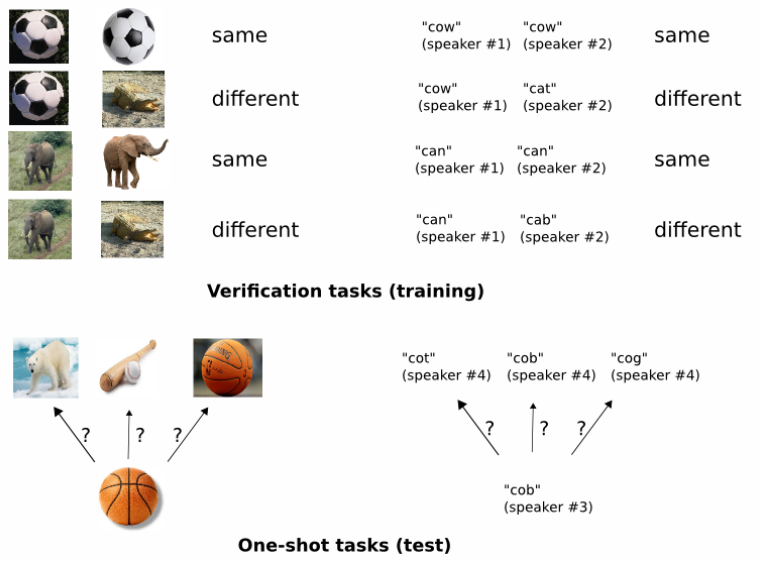
*Hình 3.4.17**: Siamese Neural Network [16]*

Thay vì học cách phân loại các đầu vào như CNN thuần, mạng Siamese học cách phân biệt giữa hai đầu vào bằng một cặp CNN có chia sẻ trọng số. Layer cuối cùng của hai mạng được đưa vào L1 (hoặc L2) để tính khoảng cách giữa hai vector đặc trưng, giá trị này sau đó được đưa vào layer đầu ra với hàm kích hoạt là sigmoid. Ta hy vọng điểm tương đồng ở đầu ra này sẽ là 1 nếu x1 và x2 là thuộc cùng một nhãn, ngược lại sẽ là 0. Hàm mất mát mà mạng Siamese sử dụng là contrastive loss:

(3.4.2)

Với mạng Siamese, chúng ta có thể thực hiện one-shot hay few-shot learning chỉ việc việc học nhanh từ một hoặc một vài mẫu mỗi nhãn. (Với cách học N-way K-shot, N là số nhãn và K là số mẫu thuộc mỗi nhãn).

Trong dự đoán, ta so sánh đối tượng x1 (cần tìm nhãn) với các mẫu trong cơ sở dữ liệu, nếu không có cặp nào có điểm tương đồng lớn hơn ngưỡng, ví dụ 0,5, thì không có nhãn phù hợp với x1, ngược lại, nhãn của mẫu có điểm tương đồng cao nhất với x1 sẽ là nhãn của x1.



*Hình 3.4.18**: Chiến lược One-shot learning trong nhận diện ảnh [16]*

Ưu điểm ở đây là chúng ta chỉ cần một hoặc một vài ảnh từ mỗi nhãn. Hơn nữa, mô hình phân biệt giữa hai đầu vào, vì vậy chúng ta chỉ cần huấn luyện một lần duy nhất mà không cần phải huấn luyện lại nó trên bộ dữ liệu mới. Tuy nhiên, khi lợi về thời gian huấn luyện, mạng Siamese lại bất lợi về mặt thời gian dự đoán, chúng ta phải ghép ảnh cần dự đoán với từng mẫu trong cơ sở dữ liệu, điều đó không hiệu quả cho các chương trình thời gian thực, đặc biệt bất lợi hơn khi ta sử dụng cơ sở dữ liệu lớn.

1. Kết quả

## Kết quả thực nghiệm

Hệ thống chạy các thuật toán deep learning được xây dựng bằng ngôn ngữ lập trình Python, hỗ trợ bởi thư viện Keras, vận hành trên laptop sử dụng hệ điều hành Windows 10, tốc độ CPU 2.0 Ghz, RAM 8Gb.

Đề tài sử dụng hai mạng thần kinh: Mạng đầu tiên được gọi là mô hình liveness được sử dụng để phát hiện khuôn mặt thật hoặc giả mạo (khuôn mặt thật là khuôn mặt được camera trực tiếp chụp từ người thật, khuôn mặt giả mạo là khuôn mặt được chụp từ hình ảnh hoặc màn hình điện thoại). Mô hình thứ hai được gọi là mô hình identifier được sử dụng để phân loại khuôn mặt.

### **Mô hình liveness**

Mô hình liveness sử dụng cấu trúc MobileNetV2 (35 layer convolution, 17 layer depthwise convolution). Ảnh trước khi được được đưa vào mạng sẽ được thay đổi kích thước về (kích thước mặc định là [12]).

Đề tài đã thu thập 3198 hình ảnh bao gồm ba nhãn: real (chụp trực tiếp lên khuôn mặt thật), image (chụp lên hình ảnh) và phone (chụp lên màn hình điện thoại), mỗi nhãn bao gồm 1066 ảnh. Bộ dữ liệu được chia thành tập huấn luyện, tập xác thực và tập kiểm tra với tỉ lệ 50%: 25%: 25%.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thuật toán tối ưu | Transfer learning | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| Adam | Các layer FC | 0.8858 | 0.6986 | 0.62 |
| Toàn mô hình | 0.9828 | 0.9962 | **0.99** |
| Đóng băng 20 layer đầu tiên | 0.9847 | 0.9849 | **0.99** |
| SGD | Các layer FC | 0.4775 | 0.3897 | 0.37 |
| Toàn mô hình | 0.7697 | 0.7894 | 0.78 |
| Đóng băng 20 layer đầu tiên | 0.7601 | 0.8197 | 0.81 |
| RMSprop | Các layer FC | 0.3233 | 0.4111 | 0.37 |
| Toàn mô hình | 0.9887 | 0.9836 | 0.98 |
| Đóng băng 20 layer đầu tiên | 0.9833 | **0.9984** | **0.99** |

*Bảng 4.1.1: Kết quả dự đoán của mô hình huấn luyện với BS 32, lr 1e-4, tỉ lệ 50%:25%:25%*

|  |  |  |  |
| --- | --- | --- | --- |
| Tỉ lệ dữ liệu | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| 70%:05%:25% | 0.9956 | 0.9889 | 0.99 |
| 65%:10%:25% | 0.9629 | 0.9777 | 0.97 |
| 60%:15%:25% | 0.9949 | 0.9926 | 0.99 |
| 55%:20%:25% | 0.9937 | 0.9972 | 0.99 |
| 45%:30%:25% | 0.9937 | 0.9852 | 0.99 |

*Bảng 4.1.2: Kết quả dự đoán của mô hình huấn luyện với RMSprop đóng băng 20 layer, BS 32, lr 1e-4*

|  |  |  |  |
| --- | --- | --- | --- |
| lr | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| 5e-5 | 0.9833 | 0.9855 | 0.98 |
| 5e-4 | 0.9828 | 0.9956 | 0.99 |
| 1e-3 | 0.9904 | 0.9833 | 0.97 |
| 5e-3 | 0.9771 | 0.6518 | 0.65 |
| 1e-2 | 0.9509 | 0.5061 | 0.48 |

*Bảng 4.1.3: Kết quả dự đoán của mô hình huấn luyện với RMSprop đóng băng 20 layer, BS 32, tỉ lệ 50%:25%:25%*

|  |  |  |  |
| --- | --- | --- | --- |
| BS | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| 4 | 0.6971 | 0.3259 | 0.36 |
| 8 | 0.8865 | 0.8398 | 0.82 |
| 16 | 0.9860 | **0.9989** | **0.99** |
| 64 | 0.9678 | 0.9911 | 0.98 |
| 128 | 0.9697 | 0.9655 | 0.96 |

*Bảng 4.1.4: Kết quả dự đoán của mô hình huấn luyện với RMSprop đóng băng 20 layer, lr 1e-4, tỉ lệ 50%:25%:25%*

### **Mô hình identifier**

Mô hình này do người sử dụng huấn luyện, việc chụp một tập dữ liệu lớn là điều không hợp lý, vì thế chúng ta cần sử dụng phương pháp transfer learning trên dữ liệu tương đồng.

Theo [17], VGGFace2 là bộ dữ liệu chứa 3,31 triệu ảnh của 9131 đối tượng, với trung bình 362,6 ảnh cho mỗi đối tượng. Hình ảnh được tải xuống từ Google Image Search và có các biến thể lớn về tư thế, tuổi tác, độ chiếu sáng, sắc tộc và nghề nghiệp (ví dụ: diễn viên, vận động viên, chính trị gia). Tác giả đã huấn luyện các mô hình VGG16 (13 layer tích chập), ResNet50 (53 layer tích chập) và SENet50 (85 layer tích chập).

Đề tài đã thực hiện transfer learning các layer FC trên VGG16, ResNet50 và SENet50 để huấn luyện ba nhãn, mỗi nhãn bao gồm 150 ảnh chia làm ba tập huấn luyện:xác thực:kiểm tra với tỉ lệ 75:25:50, tập kiểm tra chỉ được sử dụng để đánh giá mô hình và sẽ bị loại bỏ trong phiên bản sản phẩm (chỉ còn 100 ảnh mỗi nhãn). ResNet50 và SENet50 có thể là mô hình quá sâu so với tập dữ liệu quá nhỏ và bị overfitted, trong khi VGG16 cho kết quả khá tốt.

|  |  |  |  |
| --- | --- | --- | --- |
| Network | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| VGG16 | 1.0000 | 0.9867 | **0.98** |
| ResNet50 | 0.8523 | 0.3333 | 0.33 |
| SENet50 | 0.3858 | 0.3333 | 0.33 |

*Bảng 4.1.5: Kết quả dự đoán của các mô hình huấn luyện với BS 16, lr 1e-4, 3 nhãn, DO 0.5*

Dựa vào kết quả trên, ta chọn VGG16. Ảnh trước khi đưa vào VGG16 được thay đổi kích thước về để giảm chi phí tính toán so với kích thước mặc định , hai layer FC-4096 làm tăng dung tích mạng và giảm tốc độ xử lý được thay thế bằng một layer FC-512.

|  |  |  |  |
| --- | --- | --- | --- |
| BS | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| 8 | 0.9777 | 0.9867 | 0.97 |
| 16 | **0.9731** | **0.9867** | **0.98** |
| 32 | 0.8425 | **0.9867** | **0.98** |
| 64 | 0.9167 | **0.9867** | **0.98** |

*Bảng 4.1.6: Kết quả dự đoán của mô hình huấn luyện với lr 1e-4, 3 nhãn, DO 0.5*

|  |  |  |  |
| --- | --- | --- | --- |
| lr | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| 5e-3 | 0.9641 | **0.9867** | **0.99** |
| 1e-3 | 0.9776 | **0.9867** | **0.99** |
| 5e-4 | **0.9821** | **0.9867** | **0.99** |
| 1e-4 | 0.9507 | 0.9867 | 0.98 |

*Bảng 4.1.7: Kết quả dự đoán của mô hình huấn luyện với BS 16, 3 nhãn, DO 0.5*

|  |  |  |  |
| --- | --- | --- | --- |
| DO | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| 0 | 1.0000 | 0.9867 | 0.99 |
| 0.2 | 0.9910 | 0.9867 | **1.00** |
| 0.3 | 0.9865 | 0.9867 | 0.98 |
| 0.4 | 0.9865 | 0.9867 | 0.99 |
| 0.6 | 0.9910 | 0.9867 | 0.99 |

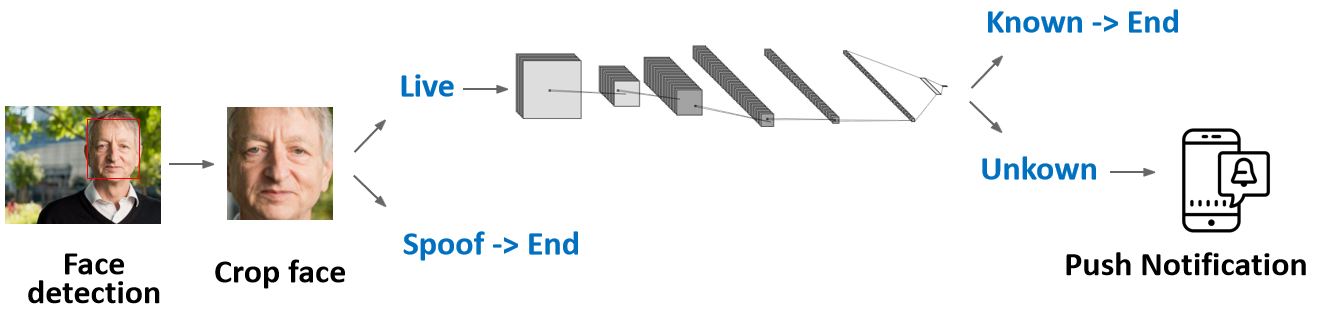
*Bảng 4.1.8: Kết quả dự đoán của mô hình huấn luyện với BS 16, 3 nhãn, lr 5e-4*

|  |  |  |  |
| --- | --- | --- | --- |
| Số nhãn | Tập dữ liệu | | |
| Huấn luyện | Xác thực | Kiểm tra |
| 2 | 0.9860 | 1.0000 | 1.00 |
| 4 | 0.9483 | 0.9867 | 0.96 |
| 5 | 0.9507 | 0.9867 | 0.97 |
| 6 | 0.9297 | 0.9733 | 0.97 |

*Bảng 4.1.9: Kết quả dự đoán của mô hình huấn luyện với BS 16, lr 5e-4, DO 0.2*

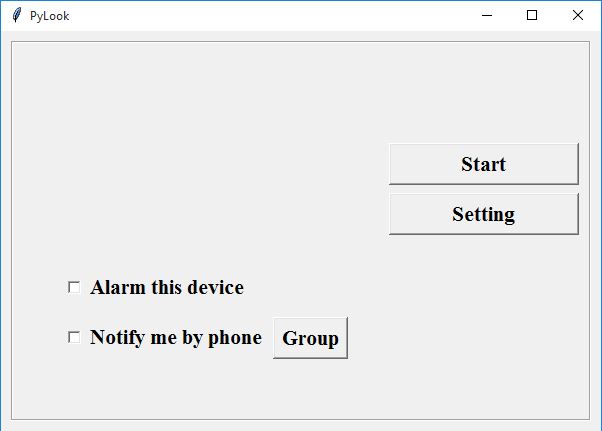
## Sản phẩm đề tài

Sản phẩm đề tài là một phần mềm gồm các chức năng cập nhật khuôn mặt, huấn luyện khuôn mặt, gửi thông báo đến nhóm khi phát hiện người lạ và một app có các chức năng đăng ký nhận thông tin từ nhóm, đặt âm báo.



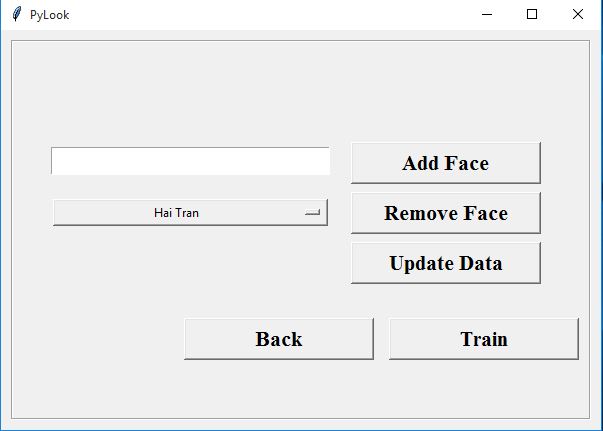
*Hình 4.2.1**: Hệ thống nhận diện khuôn mặt người lạ*

Cách hệ thống nhận diện khuôn mặt người lạ được minh họa như hình 4.2.1. Sau khi phát hiện khuôn mặt, hệ thống cắt vị trí có khuôn mặt và đưa vào lần lượt các mô hình liveness và identifier. Nếu là khuôn mặt trực tiếp và chưa được huấn luyện, hệ thống gửi ảnh đến smartphone, smartphone đẩy một thông báo cho người dùng.



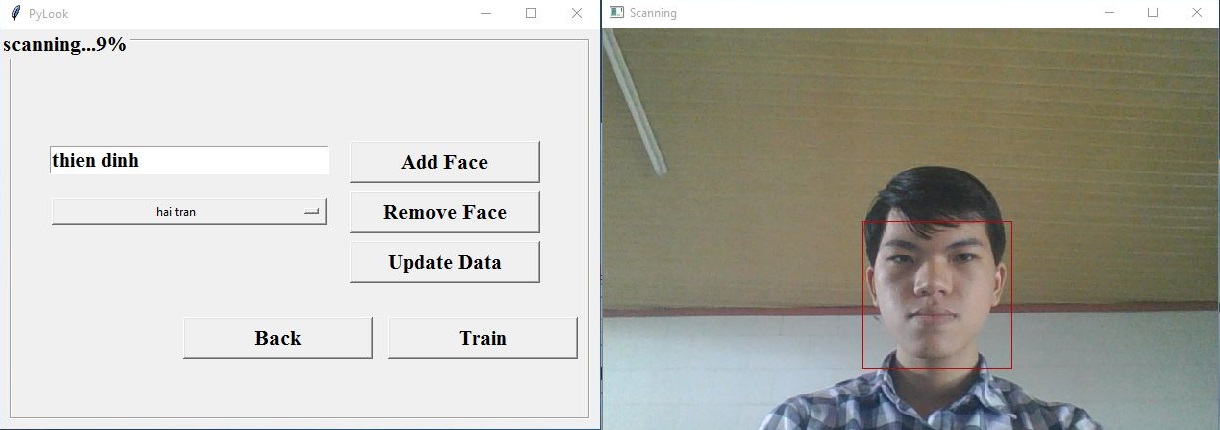
*Hình 4.2.2: Màn hình sau khi khởi động phần mềm*

Sau khi khởi động phần mềm, ta được màn hình như hình 4.2.2. Để thêm xóa khuôn mặt, ta vào Setting, màn hình cài đặt hiện ra như hình 4.2.3.



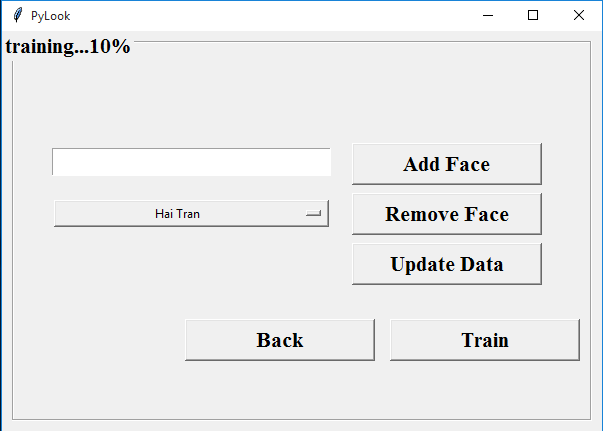
*Hình 4.2.3: Màn hình cài đặt*

Để thêm khuôn mặt, ta nhập tên người cần thêm vào textbox, sau đó chọn Add Face, cửa sổ thêm khuôn mặt hiện ra như hình 4.2.4. Lúc này người cần thêm khuôn mặt đưa khuôn mặt trước camera và thực hiện các động tác ngửa mặt lên xuống, trái phải cho đến khi lưu đủ 100 ảnh thì hệ thống tự động đóng cửa sổ thêm khuôn mặt. Để xóa khuôn mặt của người nào ra khỏi hệ thống, ta chỉ cần chọn dropdown, chọn tên người cần xóa, cuối cùng chọn Remove Face.



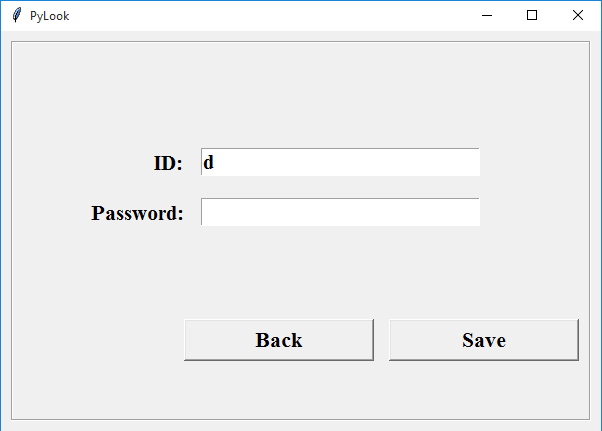
*Hình 4.2.4: Cửa sổ thêm khuôn mặt*

Sau khi thêm hay xóa khuôn mặt, hệ thống vẫn chỉ nhận diện các khuôn mặt đã được huấn luyện trước đó cho đến khi ta huấn luyện lại trên dữ liệu mới đã cập nhật bằng cách chọn Train. Trong khi huấn luyện, hệ thống thông báo mức độ hoàn thành như hình 4.2.5.



*Hình 4.2.5: Mành hình khi huấn luyện học khuôn mặt*

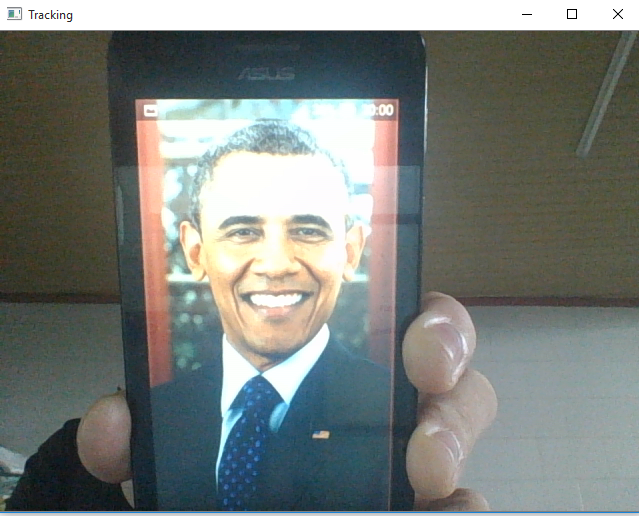
Để bật chức năng gửi thông báo đến nhóm, tại màn hình chính ta tích vào checkbox Notify me by phone và chọn Group để đặt thông tin cho nhóm.



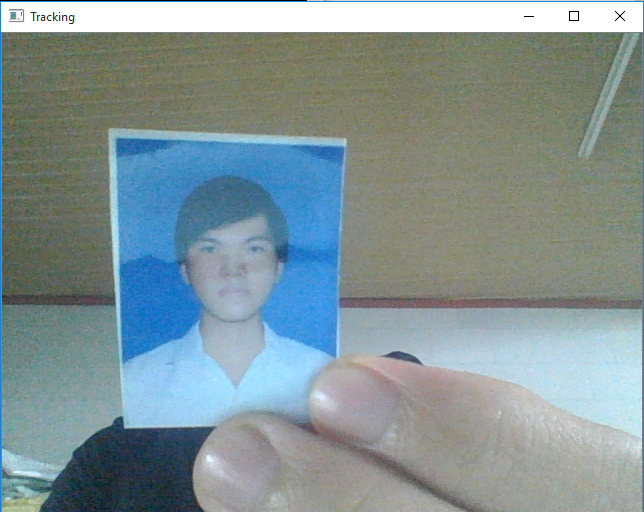
*Hình 4.2.6: Màn hình đặt thông tin nhóm*

Tại màn hình đặt thông tin nhóm như hình 4.2.6, ta đặt id và password cho nhóm theo textbox tương ứng. Để lưu thông tin lại, ta chọn Save. Hệ thống thông báo lưu thông tin thành công nếu id nhóm chưa được tạo hoặc id và password trùng với nhóm đã được tạo trong cơ sở dữ liệu trên Firebase. Nếu thông tin nhóm đã được đặt trước đó trong phần mềm, khi vào màn hình đặt thông tin nhóm hệ thống sẽ hiện sẵn id nhóm, lúc này ta có thể tạo id mới hoặc cập nhật password mới.

Để khởi động chức năng giám sát, tại màn hình chính chọn Start.



*Hình 4.2.7: Cửa sổ giám sát khi phát hiện khuôn mặt giả mạo từ smartphone*



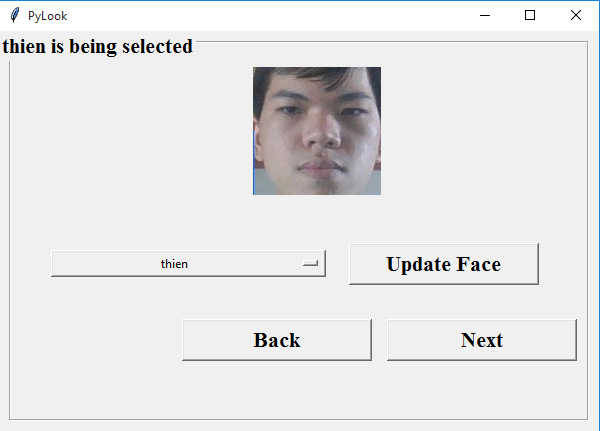
*Hình 4.2.8: Cửa sổ giám sát khi phát hiện khuôn mặt giả mạo từ hình ảnh*



*Hình 4.2.9: Cửa sổ giám sát khi phát hiện khuôn mặt trực tiếp*

Trên cửa sổ giám sát, hệ thống chỉ nhận diện các khuôn mặt trực tiếp như hình 4.2.9 và bỏ qua các khuôn mặt trên ảnh hoặc trên smartphone như hình 4.2.7 và 4.2.8, nếu đó là người đã được huấn luyện, hệ thống hiện tên người đó, nếu không hệ thống hiện “undefined” và gửi ảnh đến nhóm nếu chức năng thông báo qua nhóm được bật.

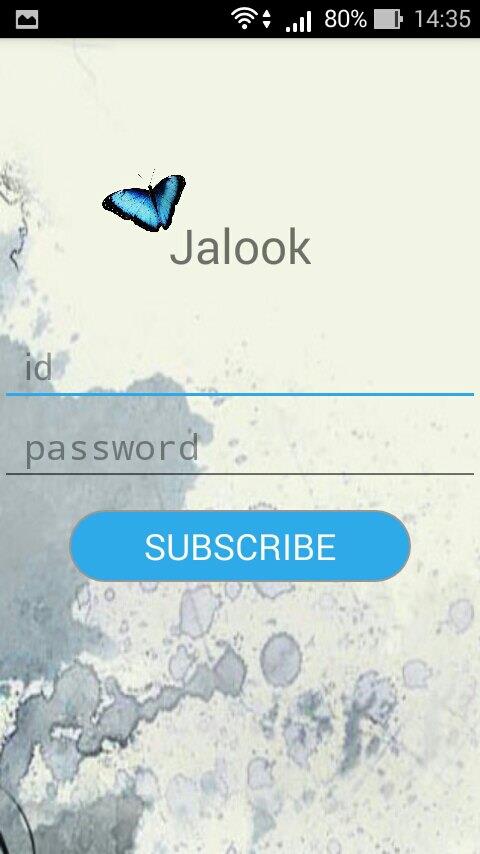
Trong khi giám sát, hệ thống lưu ngẫu nhiên một số khuôn mặt. Tại màn hình cài đặt, ta chọn Update Data để cập nhật các khuôn mặt này.



*Hình 4.2.10: Màn hình cập nhật khuôn mặt từ dữ liệu giám sát*

Trên màn hình cập nhật khuôn mặt từ dữ liệu giám sát như hình 4.2.10, hệ thống hiện nhãn dự đoán và khuôn mặt được dự đoán, nếu dự đoán này đúng, ta có thể bỏ qua bằng cách chọn Next. Nếu hệ thống đoán sai, ví dụ khuôn mặt đã được huấn luyện nhưng lại được dự đoán là “undefined”, ta có thể cập nhật lại nhãn trên dropdown và chọn Update Face. Sau khi cập nhật dữ liệu xong, ta trở về màn hình cài đặt và huấn luyện lại để hệ thống có thể dự đoán tốt hơn.

Trên app, để đăng ký nhận thông tin từ nhóm, ta nhập id và password vào màn hình đăng ký như hình 4.2.11.



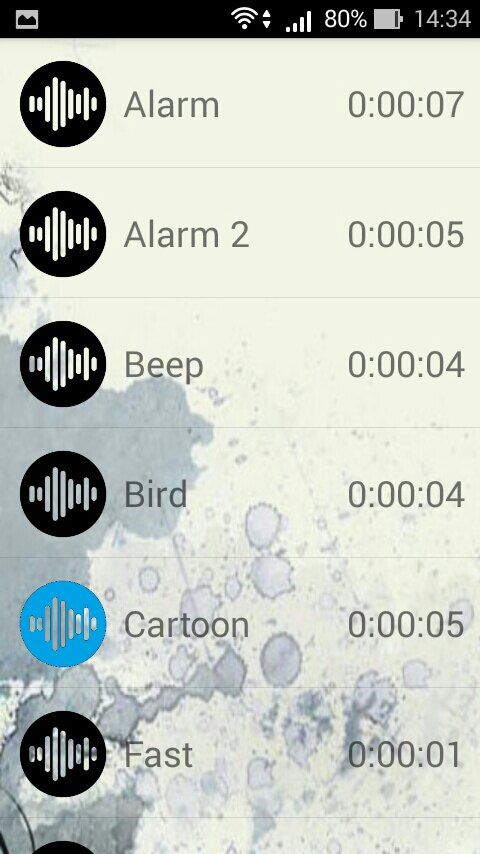
*Hình 4.2.11: Màn hình đăng ký nhận thông tin từ nhóm*

Sau khi đăng ký nhận thông tin, app sẽ thông báo khi có ảnh được gửi tới, mở app lên, ta có thể xem ảnh như hình 4.2.12.



*Hình 4.2.12: Màn hình chính khi hệ thống phát hiện người lạ*

Ta có thể cập nhật âm báo như hình bằng cách chọn nút ba chấm, chọn Set sound, màn hình chọn âm báo hiện lên như hình 4.2.13.



*Hình 4.2.13: Màn hình chọn âm báo*

Để nghe thử âm thanh, ta chọn tên âm thanh một lần. Để đặt âm thanh làm âm báo, ta chọn đúp vào tên âm thanh đó.

# Kết luận

Luận văn này về cơ bản đã hoàn thành, mang lại cho chúng ta kiến thức nền tảng về deep learning, khả năng huấn luyện neural network với framwork Keras. Mặc dù những kiến thức đã được nghiên cứu còn rất cơ bản, nhưng chúng đã cho những kết quả nhất định. Vẫn còn vô số những kỹ thuật và phương pháp để nghiên cứu và thu được kết quả tốt hơn, nhưng cái mới hơn thường được tạo ra dựa trên những cái cơ bản hơn, vì vậy cần phải hiểu những kiến thức cơ bản để hiểu những kiến thức có mức độ cao hơn một cách tốt hơn.

Phần mềm tới nay đã có thể học khuôn mặt, phát hiện khuôn mặt trực tiếp, phát hiện người lạ bằng khuôn mặt và gửi thông báo cho người dùng thông qua một app trên smartphone. Tuy nhiên, để sử dụng trong thế giới thực, dự án cần có khả năng kết nối với hệ thống nhúng (raspberry pi, nano pc, trộm) và camera giám sát cũng như hỗ trợ cùng lúc nhiều camera. Ngoài ra, việc phát hiện người ra thông qua khuôn mặt còn một số bất cập. Chẳng hạn, chúng ta không thể yêu cầu người ra vào tháo khẩu trang, kính hoặc mũ; khi khuôn mặt quá xa camera thì ảnh rất nhỏ thậm chí con người cũng không phân biệt được đó là khuôn mặt của ai. Hướng phát triển của đề tài là sử dụng deep learning để nhận diện dáng đi thay cho khuôn mặt của người lạ (xử lý một số frame ảnh liên tiếp thay vì cố gắng làm cho phần mềm chạy thời gian thực).

Hy vọng các giải pháp cùng phản hồi của thầy cô và bạn bè, đề tài sẽ sớm được áp dụng trong tương lai.

### TÀI LIỆU THAM KHẢO

**Tài liệu tiếng Việt:**

1. Vũ Hữu Tiệp. Multi-layer Perceptron và Backpropagation. Link: <https://machinelearningcoban.com/2017/02/24/mlp/>. [Truy cập: 2019-08-01].
2. Tình hình vi phạm pháp luật tháng 4 / 2019. Link: <http://bocongan.gov.vn/thong-tin-thong-ke/phong-chong-toi-pham-va-vppl/tinh-hinh-vi-pham-phap-luat-thang-42019-d101-t25362.html>. [Truy cập: 2019-08-08].
3. Tình hình vi phạm pháp luật tháng 5 / 2019. Link: <http://bocongan.gov.vn/thong-tin-thong-ke/phong-chong-toi-pham-va-vppl/tinh-hinh-vi-pham-phap-luat-thang-52019-d101-t25565.html>. [Truy cập: 2019-08-08].
4. Tình hình vi phạm pháp luật tháng 6 / 2019. Link: <http://bocongan.gov.vn/thong-tin-thong-ke/phong-chong-toi-pham-va-vppl/tinh-hinh-vi-pham-phap-luat-thang-62019-d101-t25774.html>. [Truy cập: 2019-08-08].
5. Tình hình vi phạm pháp luật tháng 7 / 2019. Link: <http://bocongan.gov.vn/thong-tin-thong-ke/phong-chong-toi-pham-va-vppl/tinh-hinh-vi-pham-phap-luat-thang-72019-d101-t25976.html>. [Truy cập: 2019-08-08].
6. Tìm hiểu về thư viện keras trong deep learning. Link: <https://thorpham.github.io/blog/2018/05/25/keras/> [Truy cập 2019-08-01].

**Tài liệu tiếng Anh:**

1. François Chollet. Deep Learning with Python. First edition, Manning Publications Co, 2018, 9781617294433.
2. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning. http://www. deeplearningbook.org. MIT Press, 2016, 9780262035613.
3. Rajalingappaa Shanmugamani. Deep Learning for Computer Vision. First edition, Packt Publishing, 2018, 978-1-78829-562-8.
4. Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. IEEE, 2017.
5. Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis. Deep Learning for Computer Vision: A Brief Review. 2018.
6. Mark Sandler, A. Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. CVPR, 2018.
7. Karen Simonyan, Andrew Zisserman. Very deep convolutional neural networks for large-scale image recognition. ICLR, 2015.
8. S. Ruder. An overview of gradient descent optimization algorithms. 2016.
9. Diederik P. Kingma, Jimmy Lei Ba. Adam: A method for Stochastic Optimization. ICLR, 2015.
10. G. Koch, R. Zemel, and R. Salakhutdinov. Siamese Neural Networks for One-shot Image Recognition. ICML Workshop on Deep Learning, 2015.
11. Qiong Cao, Li Shen, Weidi Xie, Omkar M. Parkhi and Andrew Zisserman. VGGFace2: A dataset for recognising faces across pose and age. IEEE International Conference on Automatic Face & Gesture Recognition, 2018.
12. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. CoRR, 2015, pp. 770-778.
13. A. Krizhevsky, I. Sutskever, G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012.
14. Christian Szegedy et al. Going deeper with convolutions. CVPR, 2015.
15. Sutskever, James Martens, George Dahl, Geoffrey Hinton. On the importance of initialization and momentum in deep learning. ICML 2013.
16. Kavita Khobragade. A Comparative study of Converting Coloured Image to Gray-scale Image using Different Technologies.
17. Andrej Karpathy. Linear classify. Link: <http://cs231n.github.io/linear-classify/>. [Truy cập: 2019-08-01].
18. Andrej Karpathy. Learning. Link: <http://cs231n.github.io/neural-networks-3/>. [Truy cập: 2019-08-01].
19. Fjodor Van Veen. The Neural Network zoo. Link: <http://www.asimovinstitute.org/neural-network-zoo/>. [Truy cập: 2019-08-01].
20. Victor Powell. Image Kernels. Link: <http://setosa.io/ev/image-kernels/>. [Truy cập: 2019-08-01].
21. Link: <https://opencv.org/>. [Truy cập: 2019-08-01].
22. Link: <http://dlib.net/>. [Truy cập: 2019-08-01].