

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

— \* —

ĐỒ ÁN  
**TỐT NGHIỆP ĐẠI HỌC**  
NGÀNH CÔNG NGHỆ THÔNG TIN

**ỨNG DỤNG KIẾN TRÚC MẠNG  
DENSENET  
TRONG BÀI TOÁN PHÂN GIỚI TÍNH**

Sinh viên thực hiện: **Nguyễn Tất Hòa**  
Lớp CNTT2.01 – k58  
Giáo viên hướng dẫn: TS. **Đinh Viết Sang**

HÀ NỘI 01-2018

# PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

## 1. Thông tin về sinh viên

Họ và tên sinh viên: Nguyễn Tất Hòa

Điện thoại liên lạc 0974564435

Email: 20131536@student.hust.edu.vn

Lớp: CNTT2.01

Hệ đào tạo: Đại học chính quy

Đồ án tốt nghiệp được thực hiện tại: Đại học Bách Khoa Hà Nội.

Thời gian làm ĐATN: Từ ngày 05/09/2017 đến 07/01/2017

## 2. Mục đích nội dung của ĐATN

Nghiên cứu và ứng dụng mô hình Deeplearning đối với bài toán phân loại ảnh cụ thể là nhận dạng giới tính và độ tuổi của đối tượng (người) trong ảnh.

## 3. Các nhiệm vụ cụ thể của ĐATN

- Tìm hiểu tổng quan về Deeplearning, framework Tensorflow: các khái niệm liên quan đến học máy, các kỹ thuật đánh giá hiệu năng thuật toán.
- Tìm hiểu cấu trúc trong mạng thần kinh nhân tạo (convolution neural network) các vấn đề gặp phải trong bài toán phân loại ảnh.
- Nghiên cứu và cài đặt cấu trúc mạng DenseNet.
- Ứng dụng vào bài toán nhận dạng giới tính và chuẩn đoán độ tuổi dựa trên các bộ dữ liệu.

## 4. Lời cam đoan của sinh viên:

Tôi – *Nguyễn Tất Hòa* - cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *TS. Đinh Viết Sang*.

Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

*Hà Nội, ngày tháng năm*  
Tác giả ĐATN

*Nguyễn Tất Hòa*

## 5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

*Hà Nội, ngày tháng năm*  
Giáo viên hướng dẫn

*TS. Đinh Viết Sang*

## LỜI CẢM ƠN

Trước hết, tôi xin bày tỏ lòng cảm ơn chân thành đến TS. Đinh Viết Sang, bộ môn Khoa Học Máy Tính, viện CNTT&TT, trường ĐHBK Hà Nội là người đã tận tình hướng dẫn tôi trong suốt quá trình làm đồ án tốt nghiệp.

Tôi xin cảm ơn các thầy, cô giáo ở Viện CNTT&TT và các thầy cô trong trường ĐHBK Hà Nội đã giảng dạy tôi trong suốt thời gian học tập tại trường, xây dựng cho tôi kiến thức nền tảng và những kiến thức chuyên môn để tôi có thể hoàn thành tốt đồ án này.

Tuy đã có cố gắng hoàn thành được đồ án tốt nghiệp theo dự định nhưng trong quá trình làm không thể không có những thiếu sót trong đồ án này. Kính mong nhận được dự góp ý của thầy cô và các bạn.

Sinh viên

Nguyễn Tất Hòa

## **TÓM TẮT NỘI DUNG ĐỒ ÁN TỐT NGHIỆP**

Trong những năm trở lại đây, học máy đã thúc đẩy tiến bộ trong đa dạng các lĩnh vực như nhận thức hình ảnh, dịch tự động, nhận diện giọng nói,... những vấn đề từng rất khó khăn với các nhà nghiên cứu trí tuệ nhân tạo. Lĩnh vực xử lý ảnh cũng là một điều không ngoại lệ, kể từ khi được open source vào tháng 11/2015, Tensorflow đã nhanh chóng trở thành một thư viện được ưa thích nhất của các lập trình viên trong Deep Learning.

Đồ án tốt nghiệp của em sẽ giới thiệu và giải thích về học máy cũng như Deep Learning sử dụng framework Tensorflow và hiệu quả của nó trong bài toán phân loại ảnh. Từ đó, giới thiệu về cấu trúc mạng DenseNet, mô hình có hiệu năng cao trong các bài toán nhận dạng và phân loại ảnh. Tiếp đó sẽ là quá trình áp dụng mô hình đối với bài toán nhận dạng giới tính dựa trên tập dữ liệu có sẵn.

## MỤC LỤC

<b>PHẦN 1. GIỚI THIỆU CHUNG</b>	7
1.1 Tổng quan về học máy	7
1.2 Phân loại bài toán học máy	7
1.3 Vấn đề trong học máy	9
1.4 Hàm mục tiêu và hàm tối ưu	10
1.3 Thư viện Tensorflow	11
<b>PHẦN 2. TỔNG QUAN KIẾN TRÚC MẠNG CNNs</b>	13
2.1 Mạng nơ-ron (Neural Networks)	13
2.1.1 Linear Classification	13
2.1.2 Perceptron	13
2.1.3 Kiến trúc một mạng nơ-ron	15
2.1.4 Lan truyền ngược (Backpropagation)	16
2.1.5 Sự triệt tiêu Gradient (Vanishing Gradient)	16
2.2 Mạng tích chập (Convolutional Neural Networks)	17
2.2.1 Tổng quan về kiến trúc	17
2.2.2 Lớp tích chập (Convolutional layer)	17
2.2.3 Lớp Relu	20
2.2.4 Lớp pooling	20
2.2.5 Lớp Fully-connected	21
2.2.6 Thiết kế của một CNN cơ bản	21
2.3 Các thuật toán tối ưu cho mạng CNNs	22
2.3.1 Gradient Descent	22
2.3.2 Momentum update	24
2.4 Các phương pháp tránh overfitting	25
2.4.1 Chuẩn hoá (regularization)	25
2.4.2 Loại bỏ kết nối (Drop out)	25
2.4.3 Sinh thêm dữ liệu (data augmentation)	27
2.5 Bài toán phân loại ảnh	28
<b>PHẦN 3. KIẾN TRÚC CỦA MẠNG DENSENET</b>	30
3.1 Tổng quan một số kiến trúc mạng hiện đại	30
3.1.1 Mạng VGG-Net	30
3.1.2 Mạng Network in Network	30
3.1.3 Mạng ResNet	31
3.2 Mạng DenseNet	32
<b>PHẦN 4. THỰC NGHIỆM VÀ ĐÁNH GIÁ</b>	37
4.1 Mô tả bộ dữ liệu	37
4.2 Cài đặt thực nghiệm	38
4.3 Kết quả thực nghiệm	40
4.4 Đánh giá	43
<b>KẾT LUẬN</b>	44
<b>TÀI LIỆU THAM KHẢO</b>	45

## DANH MỤC HÌNH ẢNH

Hình 1: Mô hình thuật toán học có giám sát .....	7
Hình 2: Ví dụ về overfitting trong học máy [3] .....	10
Hình 3: Kết quả dự đoán của hình ảnh [3] .....	11
Hình 4: Hàm softmax đưa ra kết quả phân loại [3] .....	11
Hình 5: Ví dụ về hàm phân loại tuyến tính Linear Mapping [1] .....	13
Hình 6: Cấu trúc mạng Perceptron .....	13
Hình 7: Biến đổi hàm Sigmoid trên trục số [3] .....	14
Hình 8: Biến đổi của hàm Tanh trên trục hàm số [3] .....	14
Hình 9: Biểu diễn trên trục số hàm ReLu (bên trái) và Leaky ReLu (bên phải) .....	15
Hình 10: Cấu trúc mạng nơ-ron [3] .....	15
Hình 11: Kiến trúc mạng nơ-ron thông thường và mạng CNNs [3] .....	17
Hình 12: Ví dụ về lớp convolution [12] .....	17
Hình 13: Minh họa về phép tính Convolution [3] .....	18
Hình 14: Ví dụ minh họa về đầu ra từ lớp Convolution [3] .....	19
Hình 15: Mô phỏng hoạt động của lớp max pooling với stride =2, filters = 2 [3] ...	21
Hình 16: Thuật toán SGD [3] .....	23
Hình 17: SGD và batch gradient descent [3] .....	23
Hình 18: Momentum update vs nesterov Momentum Update [3] .....	25
Hình 19: Mạng nơ-ron thông thường (trái) và mạng nơ-ron có sử dụng dropout (phải) [3] .....	26
Hình 20: Phương pháp sinh thêm dữ liệu trước khi huấn luyện [3] .....	27
Hình 21: Biến đổi không gian màu sắc [3] .....	27
Hình 22: Xoay, lật hình trái phải [3] .....	28
Hình 23: Crop, resize hình ảnh [3] .....	28
Hình 24: Bài toán nhận diện hình ảnh [10] .....	29
Hình 25: Khó khăn trong việc nhận diện hình ảnh [3] .....	29
Hình 26: Kiến trúc VGG-Net [8] .....	30
Hình 27: Kiến trúc một mạng NiN [11] .....	31
Hình 28: Một khối building trong mạng ResNet [6] .....	31
Hình 29: Kiến trúc mạng DenseNet với 5 dense block và growth rate $k = 48$ [1] ...	32
Hình 30: Mạng DenseNet với 3 dense block [1] .....	34
Hình 31: Ví dụ về transition layer [9] .....	34
Hình 32: Kiến trúc cụ thể trong mạng với growth rate $k = 32$ [1] .....	35
Hình 33: Kết quả so sánh giữa ResNet và Desenet-BC [1] .....	36
Hình 34: Model densenet cài đặt thử nghiệm .....	36
Hình 35: Sự phân phối của tập dữ liệu dựa trên độ tuổi [10] .....	37
Hình 36: Ví dụ về tập dữ liệu .....	37
Hình 37: Độ chính xác trên tập training .....	41
Hình 38: Hàm mục tiêu trên tập huấn luyện. ....	41
Hình 39: Độ chính xác trên tập thử nghiệm .....	42
Hình 40: Giá trị hàm mục tiêu trên tập thử nghiệm .....	42

## PHẦN 1. GIỚI THIỆU CHUNG

### 1.1 Tổng quan về học máy

Học máy là một phương pháp phân tích dữ liệu mà sẽ tự động hóa việc xây dựng mô hình phân tích. Sử dụng các thuật toán lặp để học từ dữ liệu, machine learning cho phép máy tính tìm thấy những thông tin giá trị ẩn sâu mà không được lập trình một cách rõ ràng nơi để tìm.

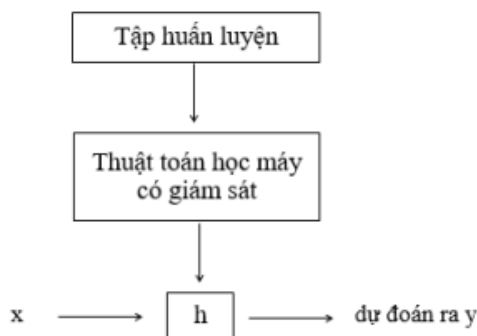
Nhờ vào các công nghệ điện toán mới, machine learning hiện nay không còn giống như machine learning trong quá khứ. Trong khi rất nhiều các thuật toán machine learning vẫn còn được sử dụng hiệu quả trong một thời gian dài, khả năng tự động áp dụng các tính toán toán học phức tạp trên dữ liệu lớn - ngày càng lớn hơn và nhanh hơn - là một sự phát triển gần đây dẫn đến sự bùng nổ của Deep learning. Dưới đây là một vài ví dụ được công bố công khai rộng rãi về các ứng dụng của machine learning:

- Những thông tin thời phùng về xe hơi tự lái của Google? Bản chất là machine learning.
- Những phân giới thiệu trực tuyến (recommendation) của Amazon và Netflix? Đó là các ứng dụng của machine learning trong cuộc sống hàng ngày.
- Biết những khách hàng nào đang nói về bạn trên mạng xã hội Twitter? Machine learning kết hợp với việc tạo ra quy tắc ngôn ngữ.
- Phát hiện gian lận? Một trong những ứng dụng rất quan trọng trong thế giới của chúng ta ngày nay.

### 1.2 Phân loại bài toán học máy

Các bài toán học máy rất đa dạng và phong phú, tuy nhiên chúng có thể được xếp vào hai nhóm chính dựa trên hình thức phân nhóm theo phương thức học:

🚦 Nhóm bài toán học có giám sát (**supervised learning**):



Hình 1: Mô hình thuật toán học có giám sát

Thuật toán dự đoán đầu ra của dữ liệu mới dựa trên các cặp đã biết từ trước. Cặp dữ liệu này còn được gọi là (data, label). Supervised learning là nhóm phổ biến nhất trong các thuật toán Machine Learning.

Chúng ta có tập hợp các cặp đầu vào  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , trong đó các cặp  $(x_i, y_i)$  gọi là mẫu huấn luyện tương ứng  $x_i$  là mẫu đặc trưng,  $y_i$  là nhãn của dữ liệu. Từ tập training này, chúng ta cần tìm ra một hàm số ánh xạ mỗi phần tử  $x_i$  sang một phần tử (xấp xỉ) tương ứng  $y_i$ .

$$y_i \approx f(x_i) \quad \forall i = 1, 2, \dots, N$$

Mục đích làm hàm xấp xỉ  $f$  thật tốt để khi có dữ liệu mới, chúng ta có thể tính được nhãn tương ứng của nó.

Ví dụ: trong nhận dạng chữ viết tay, ta có ảnh của hàng nghìn ví dụ của mỗi chữ số được viết bởi nhiều người khác nhau. Chúng ta đưa các bức ảnh này vào trong một thuật toán và chỉ cho nó biết mỗi bức ảnh tương ứng với chữ số nào. Sau khi thuật toán tạo ra (sau khi học) một mô hình, tức một hàm số mà đầu vào là một bức ảnh và đầu ra là một chữ số, khi nhận được một bức ảnh mới mà mô hình chưa nhìn thấy bao giờ, nó sẽ dự đoán bức ảnh đó chứa chữ số nào.

Thuật toán supervised learning còn được tiếp tục chia nhỏ thành hai loại chính:

- **Bài toán phân loại (classification):** Là bài toán trong đó nhãn đầu vào là các giá trị rời rạc. Bài toán phân loại thư spam và không spam là một bài toán phân loại. Trong bài toán phân loại thư, tập nhãn chỉ có một trong hai nhãn spam (tương ứng 0) và không spam (tương ứng 1). Bài toán phân loại ảnh, sẽ được trình bày trong luận văn này cũng là một ví dụ cho bài toán phân loại.
- **Bài toán hồi quy (regression):** là bài toán có giám sát trong đó tập nhãn là giá trị thực thay vì số nguyên như bài toán phân loại. Mục đích của bài toán là xây dựng mô hình cho phép đưa ra các dự đoán tương lai dựa trên các thông tin nhất định đã có. Ví dụ: bài toán ước lượng giá nhà dựa trên thông tin kích thước nhà.

#### Nhóm bài toán học không giám sát (**unsupervised learning**)

Nhóm bài toán mà chúng ta không biết được outcome hay nhãn mà chỉ có dữ liệu đầu vào. Thuật toán unsupervised learning sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó, ví dụ như phân nhóm (clustering) hoặc giảm số chiều của dữ liệu (dimension reduction) để thuận tiện trong việc lưu trữ và tính toán.

Các bài toán Unsupervised learning được tiếp tục chia nhỏ thành hai loại:



- **Bài toán phân cụm dữ liệu (clustering):** bài toán phân nhóm toàn bộ dữ liệu X thành các nhóm nhỏ dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Ví dụ: phân nhóm khách hàng dựa trên hành vi mua hàng. Điều này cũng giống như việc ta đưa cho một đứa trẻ rất nhiều mảnh ghép với các hình thù và màu sắc khác nhau, ví dụ tam giác, vuông, tròn với màu xanh và đỏ, sau đó yêu cầu trẻ phân chúng thành từng nhóm. Mặc dù không cho trẻ biết mảnh nào tương ứng với hình nào hoặc màu nào, nhiều khả năng chúng vẫn có thể phân loại các mảnh ghép theo màu hoặc hình dạng.
- **Bài toán Association:** bài toán khi chúng ta muốn khám phá ra một quy luật dựa trên nhiều dữ liệu cho trước. Ví dụ: những khách hàng nam mua quần áo thường có xu hướng mua thêm đồng hồ hoặc thắt lưng; những khán giả xem phim Spider Man thường có xu hướng xem thêm phim Bat Man, dựa vào đó tạo ra một hệ thống gợi ý khách hàng (Recommendation System), thúc đẩy nhu cầu mua sắm.

### 1.3 Vấn đề trong học máy

#### 1.3.1 Giải thuật học máy

- Những giải thuật học máy nào có thể học một hàm mục tiêu.
- Với những điều kiện nào, một giải thuật học máy đã chọn sẽ hội tụ hàm mục tiêu cần học.
- Đối với một lĩnh vực bài toán cụ thể và đối với một cách biểu diễn các ví dụ (đối tượng) cụ thể, giải thuật nào sẽ thực hiện tốt nhất.

Như đã phân loại ở phần trước, mỗi bài toán sẽ lại có một giải thuật phù hợp, không phải giải thuật nào cũng dùng được, do đó việc lựa chọn được giải thuật sẽ ảnh hưởng rất nhiều đến kết quả.

#### 1.3.2 Tập huấn luyện

- Bao nhiêu ví dụ học là đủ.
- Kích thước của tập học ảnh hưởng như thế nào đến độ chính xác của hàm mục tiêu.
- Các ví dụ lỗi (nhiều) hoặc các ví dụ thiếu tính đại diện cũng sẽ ảnh hưởng tới độ chính xác.

Để dự đoán đạt được hiệu năng cao trên tập dữ liệu mới thì tập dữ liệu chuẩn bị cho quá trình training cần phải mang tính đại diện, tính chuẩn hóa cao,... Đối với mỗi bài toán thì tập học lại khác nhau, đặc biệt là bài toán phân loại ảnh thì việc dữ liệu lỗi, nhiễu là rất nhiều cần được xử lý.

#### 1.3.3 Quá trình học

- Chiến lược nào là tối ưu cho việc lựa chọn thứ tự sử dụng (khai thác) các ví dụ học.

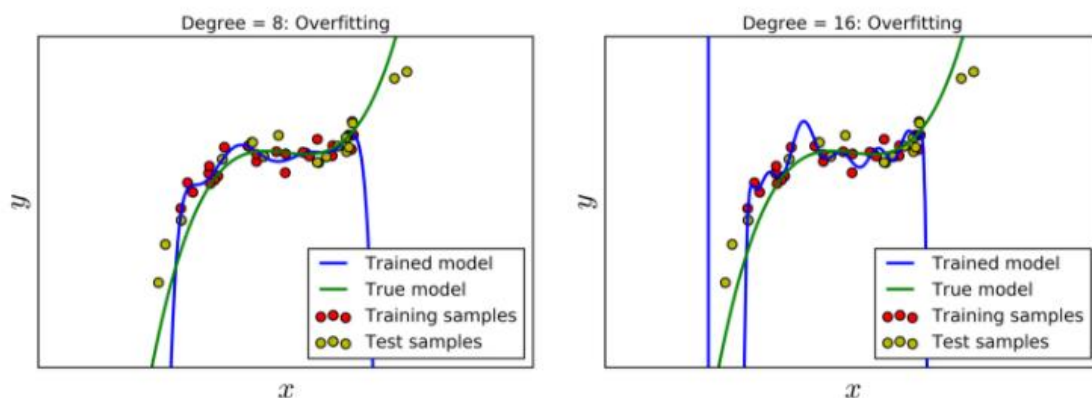
- Các chiến lược lựa chọn này làm thay đổi mức độ phức tạp của bài toán học máy như thế nào.
- Các tri thức cụ thể của bài toán có thể đóng góp thế nào đối với quá trình học.

### 1.3.4 Over-fitting

Overfit là hiện tượng mà mô hình học quá khớp với tập huấn luyện thay vì bản chất của dữ liệu (các class, ...), do đã nhắc đến ở bên trên việc tập dữ liệu xuất hiện lỗi (nhiều) có thể xảy ra nên việc quá khớp với dữ liệu học sẽ dẫn đến những dự đoán sai trên tập dữ liệu mới (chất lượng dự đoán thấp).

Nguyên nhân dẫn đến điều này:

- Lỗi (nhiều) trong tập huấn luyện (do quá trình thu thập, xây dựng tập dữ liệu).
- Số lượng các ví dụ học không đủ để đại diện cho toàn bộ tập (phân bố) của các ví dụ của bài toán.



Hình 2: Ví dụ về overfitting trong học máy [3]

Hình 2 cho thấy model đã được huấn luyện không dự đoán được chính xác đối với những dữ liệu test có khoảng cách xa với các dữ liệu huấn luyện mặc dù tăng độ chính xác đối với tập huấn luyện.

### 1.3.5 Under fitting

Underfit là tình trạng mà trong đó mô hình đạt kết quả đánh giá P thấp trên cả bộ dữ liệu huấn luyện và thử nghiệm. Lý do của vấn đề thường đến từ mô hình. Mô hình được xây dựng “quá đơn giản”, không thể biểu diễn tốt được dữ liệu.

## 1.4 Hàm mục tiêu và hàm tối ưu

### 1.4.1 Hàm mục tiêu

Đối với bài toán phân loại ảnh, đầu vào là  $N$  ảnh phân biệt, dữ liệu của một ảnh:

$$x_i \in R^D \text{ với } D \text{ là số chiều, } i \in \{1, 2, \dots, N\}.$$

Mỗi ảnh ứng với một nhãn đầu ra:

$$y_i \in \{1, 2, \dots, K\} \text{ với } i \in \{1, 2, \dots, N\}.$$

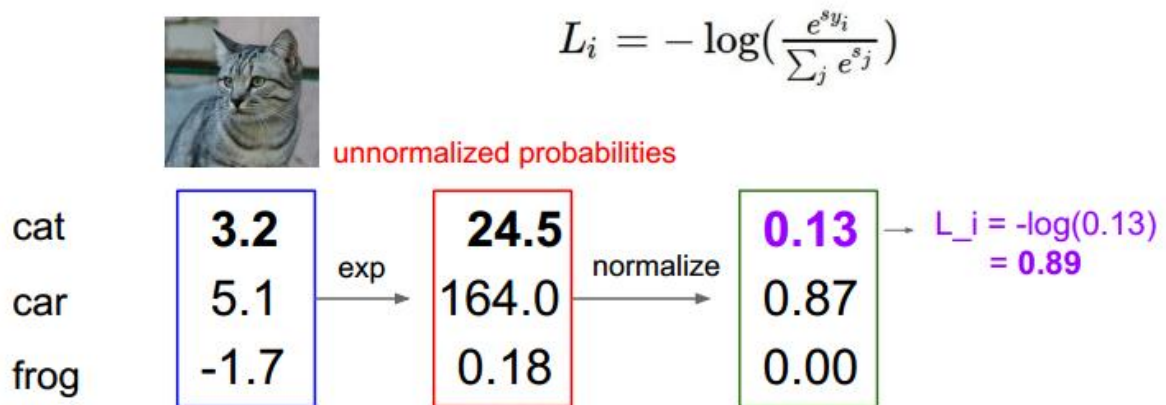
Sau khi được huấn luyện, hệ thống học máy sẽ từ dữ liệu của mỗi ảnh đưa ra số điểm ứng với từng loại, tức là tìm ra hàm:  $f: R^D \rightarrow R^K$  ánh xạ từ các điểm ảnh thành điểm số của ảnh đó đối với từng lớp (kết quả dự đoán đạt độ chính xác cao trên hàm  $f$ )



cat	<b>3.2</b>
car	5.1
frog	-1.7

Hình 3: Kết quả dự đoán của hình ảnh [3]

Giả sử, mô hình dự đoán đưa ra kết quả như hình số 3 với điểm số 3.2 cho **cat**, 5.1 cho **dog** và -1.7 cho **frog**. Ta cần đưa ra độ đo sai số giữa điểm số dự đoán và nhãn thực của đối tượng. Đó gọi là hàm mục tiêu (loss function).



Hình 4: Hàm softmax đưa ra kết quả phân loại [3]

Ví dụ: *loss function* là softmax được thể hiện trên hình 4.

#### 1.4.2 Hàm tối ưu

Trong quá trình huấn luyện, cần tìm các trọng số để tối thiểu giá trị hàm mục tiêu từ tập dữ liệu. Tức là tìm ra hàm số để đánh giá giá trị của hàm mục tiêu hay giải pháp tối ưu nhất trong tất cả các giải pháp tối ưu sao cho kết quả dự đoán được chính xác nhất.

### 1.3 Thư viện Tensorflow

Một thư viện open-source được phát triển bởi Google và công khai cho cộng đồng lập trình viên từ năm 2015.

Nền tảng được nhóm Google Brain phát triển nhằm phục vụ việc tìm kiếm hình ảnh, nhận biết giọng nói và các dấu vết dữ liệu.

Với TensorFlow, Machine Learning đang trở nên gần gũi và dễ dàng tiếp cận hơn cho lập trình viên. Một số đặc điểm đáng lưu ý của thư viện này:

- Chạy được trên cả Android IOS và Cloud.
- Hỗ trợ GPU.
- Hỗ trợ Distributed training.
- Rất nhiều ngôn ngữ frontend, nhiều nhất là Python hoặc C++.

#### **Ứng dụng:**

- Sử dụng trong rất nhiều ứng dụng của Google như phân loại email của Gmail, nhận biết phát âm và dịch tự động, nhận biết khuôn mặt trong Google Photo, tối ưu hoá kết quả tìm kiếm, quảng cáo trong Youtube,...
- Đặc trưng của TensorFlow là xử lý được tất cả các loại dữ liệu có thể biểu diễn dưới dạng data flow graph hay low level như xử lý chữ viết tay. TensorFlow được viết bằng C++, thao tác interface bằng Python vì thế performance rất tốt, dùng được cả CPU lẫn GPU nên TF có thể chạy trên cả PC thông thường lẫn một server cực lớn, thậm chí cả smartphone cũng có thể sử dụng được.
- Được sử dụng nhiều trong các bài toán xử lý, phân loại ảnh.

## PHẦN 2. TỔNG QUAN KIẾN TRÚC MẠNG CNNs

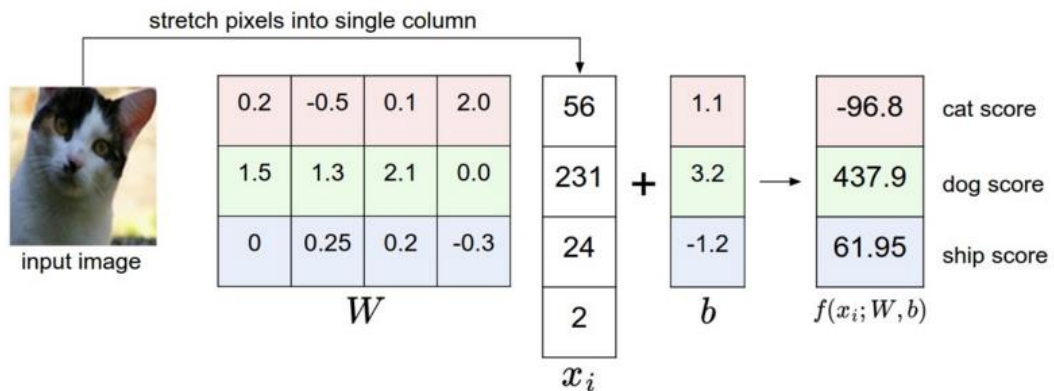
### 2.1 Mạng nơ-ron (Neural Networks)

#### 2.1.1 Linear Classification

Hàm phân loại tuyến tính đơn giản nhất có thể nói đến là linear mapping. Giả sử tập huấn luyện  $R^D$  có N phần tử được gán nhãn  $y_i$  với  $i = 1 \dots N$ ,  $y_i \in 1 \dots K$  (K là số class).

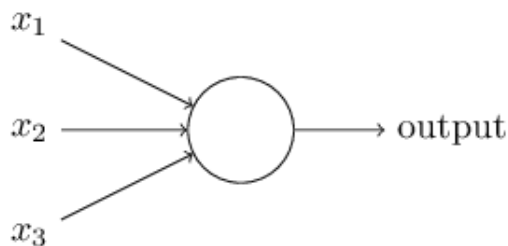
$$f(x_i, W, b) = Wx_i + b.$$

Trong hàm đánh giá nói trên trọng số  $W$  (weights) là một ma trận  $K \times D$  chiều,  $x_i$  là một ma trận  $D \times 1$  chứa tất cả giá trị pixels của ảnh đầu vào thứ  $i$  khi đã được trải dài thành một cột,  $b$  (bias vector) là một véc-tơ có kích thước  $K \times 1$ .  $W, b$  là 2 tham số cần được học từ tập huấn luyện.



Hình 5: Ví dụ về hàm phân loại tuyến tính Linear Mapping [1]

#### 2.1.2 Perceptron



Hình 6: Cấu trúc mạng Perceptron

Perceptron là một mô hình Neural Network đơn giản nhất được phát triển vào 1950-1960 bởi Frank Rosenblatt bởi nó chỉ có hai lớp và cũng chỉ hoạt động được trong một trường hợp rất cụ thể. Trong hình 6, chúng ta có ba giá trị input binary và một giá trị output binary. Trên thực tế, ta có thể cung cấp cho Perceptron nhiều hoặc ít giá trị vào. Nhiệm vụ của Perceptron là làm sao chuyển từ ba giá trị input này ra được đúng giá trị output như mong đợi:

$$output = \begin{cases} 0 & \text{nếu } Wx_i + b \leq 0 \\ 1 & \text{nếu } Wx_i + b > 0 \end{cases} \quad (2.1)$$

Trong thực tế, để có thể tính được đạo hàm riêng **loss function** trên các tham số, cần đầu ra một neuron có thể nhận một giá trị số thực nên để mô hình hoá một cách tổng quát ta có công thức

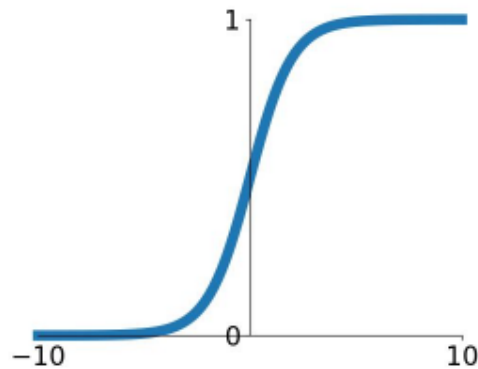
$$output = \sigma(Wx_i + b). \quad (2.2)$$

Hàm  $\sigma(z)$  được gọi là **activation function** thể hiện sự thay đổi giá trị đầu ra khi  $Wx_i + b$  vượt quá giá trị 0.

Các hình bên dưới mô tả một số hàm activation:

- Sigmoid:

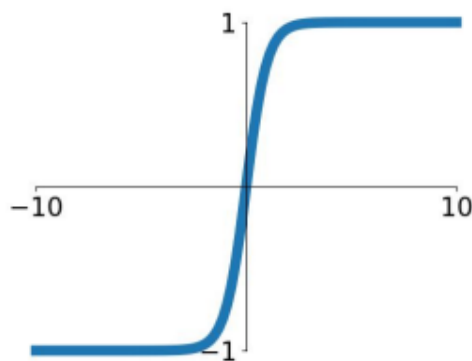
$$\sigma(z) = 1/(1 + e^{-z}).$$



Hình 7: Biến đổi hàm Sigmoid trên trục số [3]

- Tanh:

$$\tanh(z) = 2\sigma(2z).$$



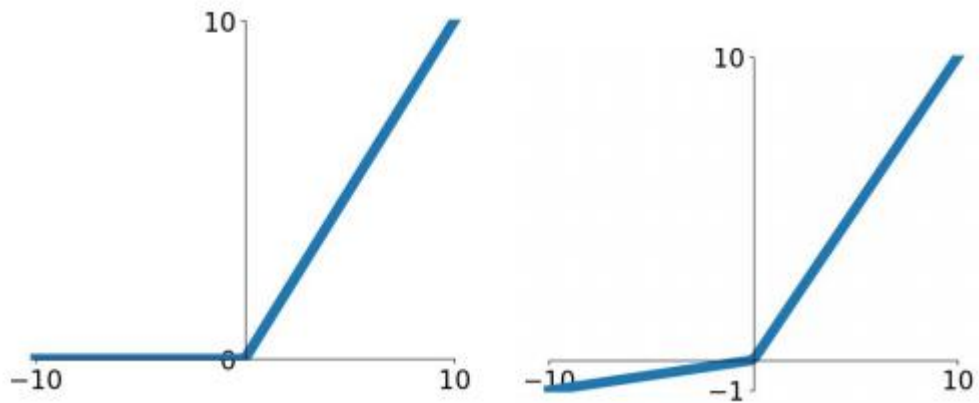
Hình 8: Biến đổi của hàm Tanh trên trục hàm số [3]

- Relu:

$$\sigma(z) = \max(0, z).$$

- Leaky Relu:

$\sigma(z) = \max(z, \alpha z)$  trong đó  $\alpha < 1$ .

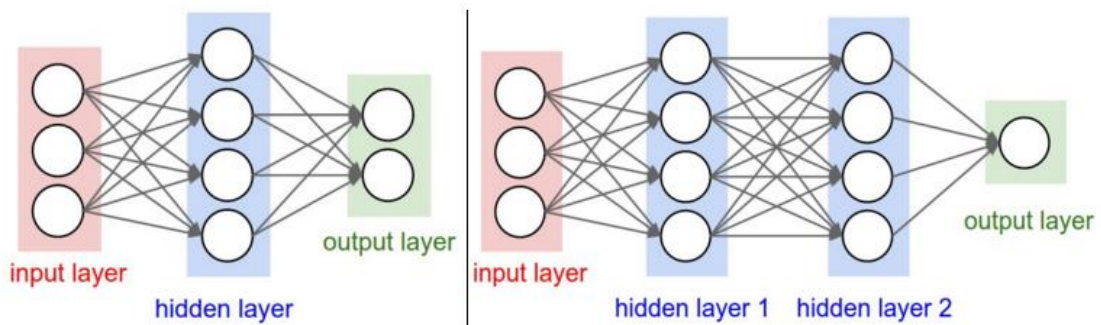


Hình 9: Biểu diễn trên trục số hàm ReLu (bên trái) và Leaky ReLu (bên phải)

### 2.1.3 Kiến trúc một mạng nơ-ron

Mạng nơ-ron (neural networks) là một mô hình các neuron được kết nối với nhau và xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ nơ-ron sinh học. Đối với các mạng neural thông thường, neuron sẽ được chia thành các tầng, đầu ra tầng trước là đầu vào của tầng sau. Các tầng được kết nối với nhau một cách đầy đủ (fully-connected layer).

Tầng đầu tiên và tầng cuối cùng được gọi là tầng đầu vào (input layers) và tầng đầu ra (output layers), các tầng ở giữa gọi là tầng ẩn (hidden layers). Mạng nơ-ron n tầng là mạng nơ-ron có n tầng không tính tầng đầu ra. Ví dụ như trên hình 8, mạng nơ-ron bên trái được gọi là mạng 2 tầng, mạng nơ-ron được gọi là mạng 3 tầng.



Hình 10: Cấu trúc mạng nơ-ron [3]

Quá trình từ đầu vào qua các tầng chuyển đến đầu ra được gọi là lan truyền xuôi (feed-forward). Quá trình học các tham số bằng cách tính ngược gradient qua các lớp được gọi là lan truyền ngược (back propagation).

Quá trình feed-forward trong một mạng neural network 3 tầng có thể mô tả bằng thuật toán (2.3):

$$\begin{aligned}h_1 &= \sigma(W_1 \cdot \text{dot}(X) + b_1) \\h_2 &= \sigma(W_2 \cdot \text{dot}(h_1) + b_2) \\out &= \sigma(W_3 \cdot \text{dot}(h_2) + b_3)\end{aligned}\tag{2.3}$$

#### 2.1.4 Lan truyền ngược (Backpropagation)

Thuật toán để tính đạo hàm riêng trên mạng nơ-ron một cách nhanh chóng được gọi là backpropagation. Phương pháp phổ biến để huấn luyện các mạng thần kinh nhân tạo được sử dụng kết hợp với các một số phương pháp tối ưu hóa như *gradient descent*. Phương pháp này tính toán gradient của hàm mục tiêu với tất cả các trọng số có liên quan trong mạng nơ-ron đó. Gradient này được đưa vào phương pháp tối ưu hóa, sử dụng nó để cập nhật các trọng số, để cực tiểu hóa hàm tổn thất.

Thuật toán học truyền ngược có thể được chia thành hai giai đoạn: lan truyền, cập nhật trọng số.

##### Giai đoạn 1: Lan truyền

- Lan truyền thuận của một đầu vào của mô hình huấn luyện thông qua mạng nơ-ron để tạo ra các kích hoạt đầu ra của lan truyền này.
- Truyền ngược của các kích hoạt đầu ra của lan truyền thông qua mạng lưới nơ-ron sử dụng mục tiêu huấn luyện mô hình để tạo ra các delta (sai lệch giữa giá trị mục tiêu và giá trị đầu ra thực tế) và tất cả đầu ra và các nơ-ron ẩn.

##### Giai đoạn 2: Cập nhật trọng số

- Nhân các delta đầu ra và kích hoạt đầu vào để có được gradient của trọng số của nó.
- Trừ một tỷ lệ (tỷ lệ phần trăm) từ gradient của trọng số.

#### 2.1.5 Sự triệt tiêu Gradient (Vanishing Gradient)

Vanishing Gradient là một khó khăn trong việc huấn luyện mạng thần kinh nhân tạo với các phương pháp dựa trên gradient. Vấn đề này làm cho việc học và điều chỉnh tham số của các lớp trước đó trong mạng thần kinh trở nên khó khăn. Điều này càng trở nên tồi hơn khi mà số lượng các lớp trong kiến trúc tăng dần.

Thông thường, việc thêm nhiều lớp ẩn có xu hướng làm cho mô hình học được hiệu quả hơn dẫn đến việc dự đoán trở nên chính xác hơn đó chính là điều tạo nên sự khác biệt của Deeplearning. Tuy nhiên, về cơ bản Vanishing Gradient không bắt nguồn từ mạng nơ-ron mà nó là vấn đề từ các phương pháp học dựa trên gradient xảy ra bởi một số hàm activation. Các phương pháp dựa trên gradient học các giá trị của tham số bằng cách hiểu rằng một sự thay đổi nhỏ trong giá trị của tham số sẽ ảnh hưởng tới đầu ra của mạng như thế nào. Khi thực hiện lan truyền ngược (backpropagation) để tính gradient của hàm mục tiêu, gradient có xu hướng nhỏ dần. Điều này có nghĩa

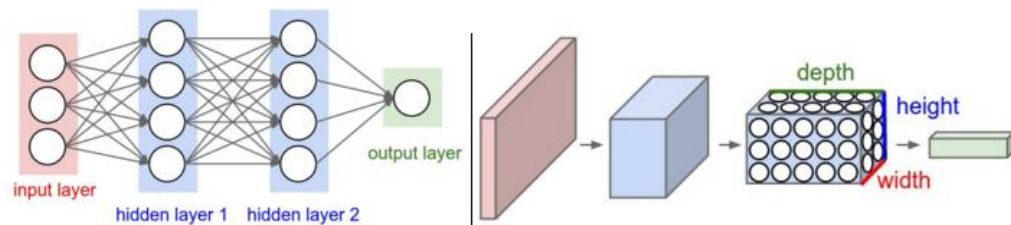


các lớp đứng trước sẽ học chậm hơn so với các lớp phía sau đó. Một mặt khác, các lớp đứng trước trong mạng lại có trách nhiệm học hỏi và phát hiện ra các tính năng đặc trưng của một class và để xây dựng nên các lớp tiếp theo. Vì thế, nếu các lớp trước mà không tốt thì các lớp sau được sinh ra từ lớp trước không thể đảm bảo sẽ đem lại được hiệu quả cho mô hình. Vấn đề này thường xuất hiện khi mô hình sử dụng các hàm Activation như là Sigmoid và Tanh nên đó cũng là lý do vì sao hai hàm này ít được sử dụng.

Tóm lại, Vanishing Gradient sẽ khiến cho quá trình huấn luyện trở nên lâu hơn nhưng hiệu năng của mô hình cũng bị giảm. Để tránh được điều này, mô hình cần sử dụng các hàm Activation có tính chất hạn chế sự nhỏ dần trong quá trình lan truyền ngược. Ví dụ như: ReLu, Leak ReLu,...

## 2.2 Mạng tích chập (Convolutional Neural Networks)

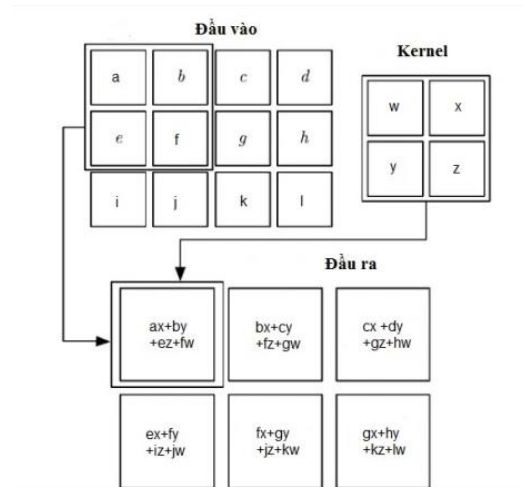
### 2.2.1 Tổng quan về kiến trúc



Hình 11: Kiến trúc mạng nơ-ron thông thường và mạng CNNs [3]

Convolutional Neural Networks (CNN) là một trong những thuật toán DeepLearning cho kết quả tốt nhất hiện nay trong hầu hết các bài toán về thị giác máy tính như phân lớp, nhận dạng... Về cơ bản CNN là một kiểu mạng ANN truyền thẳng, trong đó kiến trúc chính gồm nhiều thành phần được ghép nối với nhau theo cấu trúc nhiều tầng đó là: **Convolution**, **Pooling**, **ReLU** và **Fully connected**.

### 2.2.2 Lớp tích chập (Convolutional layer)



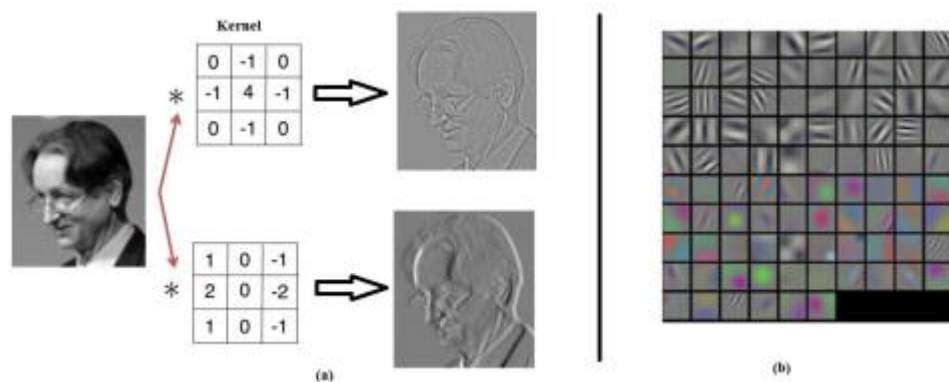
Hình 12: Ví dụ về lớp convolution [12]

Tầng Convolution (Conv) là tầng quan trọng nhất trong cấu trúc của CNN. Conv dựa trên lý thuyết xử lý tín hiệu số, việc lấy tích chập sẽ giúp trích xuất được những thông tin quan trọng từ dữ liệu. Hình 10 mô tả lý thuyết và cách thức Conv hoạt động trên một dữ liệu đầu vào được biểu diễn bằng một ma trận hai chiều. Ta có thể hình dung phép tính này được thực hiện bằng cách dịch chuyển một cửa sổ mà ta gọi là kernel trên ma trận đầu vào, trong đó kết quả mỗi lần dịch chuyển được tính bằng tổng tích chập (tích của các giá trị giữa 2 ma trận tại vị trí tương ứng), trong hình 10 là giá trị đầu ra khi dịch chuyển kernel kích thước  $2 \times 2$  trên toàn bộ ma trận kích thước  $3 \times 4$ .

Khi được áp dụng phép tính Conv vào xử lý ảnh người ta thấy rằng Conv sẽ giúp biến đổi các thông tin đầu vào thành các yếu tố đặc trưng( nó tương ứng như bộ phát hiện – detector các đặc trưng về cạnh, hướng, đốm màu,...).

Hình 13 là minh họa việc áp dụng phép tính Conv trên ảnh trong đó:

- (a) là kết quả biến đổi hình ảnh khi thực hiện phép Conv khác nhau cho ra kết quả khác nhau.
- (b) là trực quan hóa các kernel dùng để detector các đặc trưng về cạnh, hướng, đốm màu.



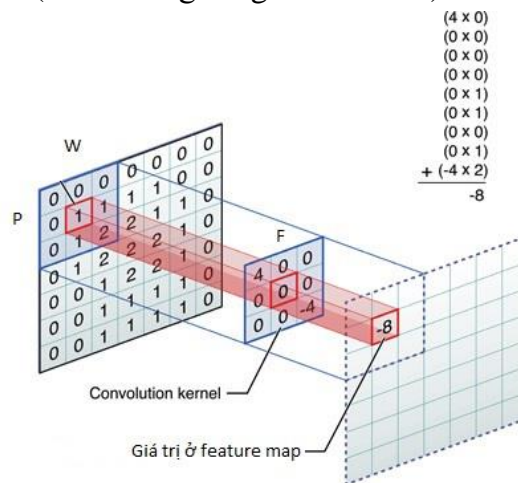
Hình 13: Minh họa về phép tính Convolution [3]

Để dễ hình dung, ta xét bài toán thực hiện tính giá trị đầu ra của một ảnh có kích thước  $W_1 * H_1 * D_1$  ( $D_1$  là chiều sâu của ảnh thực chất là giá trị tại 3 kênh màu tương ứng với ảnh RGB). Khi đó một Conv như một cửa sổ trượt (sliding window, có tên gọi là kernel, filter hay feature detector) - cửa sổ này thực chất cũng là một ma trận có kích thước  $F * F$  thực hiện trên mỗi chiều của ảnh đầu vào (ta sử dụng  $K$  filter). Trong quá trình xử lý sẽ dịch chuyển các filter trên toàn bộ bức ảnh theo  $S$  (stride) bước (tính bằng pixel). Người ta gọi mỗi vùng mà filter áp đặt lên ma trận đầu vào là receptive field. Trong một số trường hợp để cân bằng giữa số bước dịch chuyển và kích thước của ảnh người ta có thể chèn thêm  $P$  pixel (padding) với một giá trị màu cho trước (thường là 0) xung quanh viền của ảnh khi đó ta được ma trận đầu ra (feature map) là  $W_2 * H_2 * D_2$  trong đó:

$$\begin{aligned}
 W_2 &= (W_1 - F + 2 * P) / S + 1 \\
 H_2 &= (H_1 - F + 2 * P) / S + 1 \\
 D_2 &= K
 \end{aligned}
 \tag{2.4}$$

Giá trị tại các ô trong ma trận của filter có kích thước  $(F * F * D_1) + 1$  (cộng 1 ở đây là tham số ngưỡng của filter) sẽ tương ứng là trọng số, các giá trị này của mỗi filter không đổi trong quá trình dịch chuyển trên toàn bộ bức ảnh đầu vào. Đây cũng là tính chất quan trọng (dùng chung bộ trọng số – shared weights) làm giảm số tham số cần học trong quá trình huấn luyện mạng. Từ đó tổng số tham số cần học cho quá trình sử dụng Conv:

$$(F * F * D_1) * K + K \text{ (tham số ngưỡng của } k \text{ filter).}$$



Hình 14: Ví dụ minh họa về đầu ra từ lớp Convolution [3]

Trong hình 14 là ví dụ cụ thể, trong đó đầu vào là ảnh có kích thước  $(32 * 32 * 3)$  khi đó  $W_1 = H_1 = 32$  và  $D_1 = 3$ . Giả sử ta tiến hành sử dụng 6 filter ( $K = 6$ ) trong đó mỗi filter có kích thước  $(5 * 5 * 3)$  với bước dịch chuyển **stride**  $S = 1$  và **padding**  $P = 0$ . Tương ứng với mỗi filter sẽ cho một feature map khác nhau ở kết quả đầu ra trong đó: kích thước feature map là  $W_2 = H_2 = (W_1 - F) / 2 + 1 = 28$ . Mỗi nơ-ron trong một feature map sẽ có số tham số là  $5 * 5 * 3 + 1 = 76$ . Nếu không sử dụng tính chất shared weights thì số tham số cần học trong tất cả feature map trong cả 6 filter là:  $(28 * 28 * 6) * (5 * 5 * 3 + 1)$  mặc dù đã nhỏ hơn nhiều so với việc không sử dụng Conv nhưng con số vẫn lớn hơn so với  $(F * F * D_1) * K + K = 5 * 5 * 3 * 6 + 6$  tham số khi dùng chung bộ trọng số.

#### Sử dụng Conv có những ưu điểm sau:

- Giảm số lượng tham số: Ở ANNs truyền thống, các nơ-ron ở lớp trước sẽ kết nối tới tất cả các neural ở lớp sau (full connected) gây nên tình trạng quá nhiều tham số cần học. Đây là nguyên nhân chính gây nên tình trạng overfitting cũng như làm tăng thời gian huấn luyện. Với việc sử dụng Conv trong đó cho phép chia sẻ trọng

số liên kết (shared weights), cũng như thay vì sử dụng một kết nối đầy đủ sẽ sử dụng local receptive fields giúp giảm tham số.

- Các tham số trong quá trình sử dụng Conv hay giá trị của các filter – kernel sẽ được học trong quá trình huấn luyện. Như giới thiệu ở phần trên các thông tin này biểu thị thông tin giúp rút trích ra được các đặc trưng như góc, cạnh, đốm màu trong ảnh ... như vậy việc sử dụng Conv sẽ giúp xây dựng mô hình tự học ra đặc trưng.

### 2.2.3 Lớp Relu

Về cơ bản, covolution là một phép biến đổi tuyến tính. Nếu tất cả các neural được tổng hợp bởi các phép biến đổi tuyến tính thì một mạng neural đều có thể đưa về dưới dạng một hàm tuyến tính. Khi đó mạng ANN sẽ đưa các bài toán về logistic regression. Do đó tại mỗi neural cần có một hàm truyền dưới dạng phi tuyến. Có nhiều dạng hàm phi tuyến được sử dụng trong quá trình. Tuy nhiên, các nghiên cứu gần đây chứng minh được việc sử dụng hàm ReLu (Rectified Linear Unit) cho kết quả tốt hơn ở các khía cạnh:

- Tính toán đơn giản.
- Tạo ra tính thưa (sparsity) ở các neural ẩn. Ví dụ như sau bước khởi tạo ngẫu nhiên các trọng số, khoảng 50\% các neural ẩn được kích hoạt (có giá trị lớn hơn 0).
- Quá trình huấn luyện nhanh hơn ngay cả khi không phải trải qua bước tiền huấn luyện.

Như vậy tầng ReLu cơ bản chỉ đơn giản là áp dụng hàm truyền ReLu.

### 2.2.4 Lớp pooling

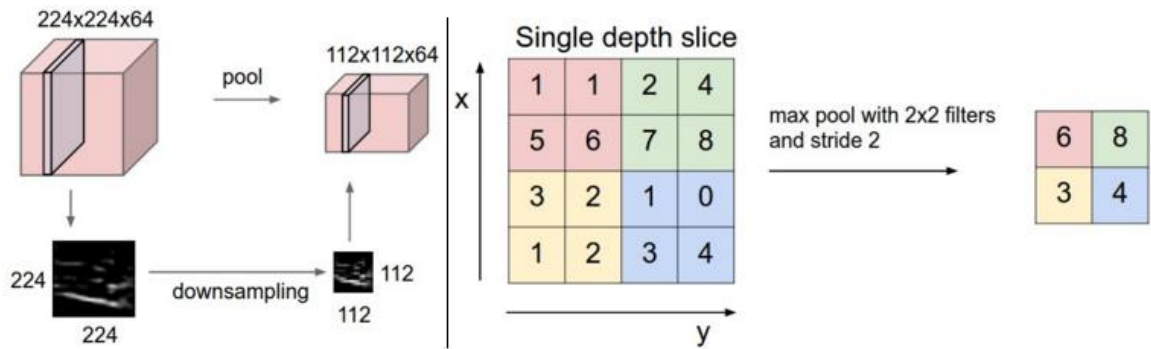
Tầng pooling (hay còn gọi subsampling hoặc downsample) là một trong những thành phần tính toán chính trong cấu trúc CNN. Xét về mặt toán học pooling thực chất là quá trình tính toán trên ma trận trong đó mục tiêu sau khi tính toán là giảm kích thước ma trận nhưng vẫn làm nổi bật lên được đặc trưng có trong ma trận đầu vào. Trong CNN toán tử pooling được thực hiện độc lập trên mỗi kênh màu của ma trận ảnh đầu vào. Có nhiều toán tử pooling như Sum-Pooling, Max-Pooling, L\_2-Pooling nhưng Max-Pooling thường được sử dụng.

#### Tổng quát hơn:

Ma trận đầu vào có kích thước  $W_1 \times H_1 \times D_1$ . Thực hiện toán tử pooling trên ma trận con của ma trận đầu vào có kích thước  $F * F$  với độ lớn của bước nhảy(stride) là  $S$  thì kích thước đầu ra  $W_2 \times H_2 \times D_2$  được tính:

$$\begin{aligned} W_2 &= (W_1 - F)/S + 1 \\ H_2 &= (H_1 - F)/S + 1 \end{aligned} \quad (2.5)$$

$$D_2 = D_1$$



Hình 15: Mô phỏng hoạt động của lớp max pooling với stride =2, filters = 2 [3]

### 2.2.5 Lớp Fully-connected

Fully-connected là cách kết nối các neural ở hai tầng với nhau trong đó tầng sau kết nối đầy đủ với các neural ở tầng trước nó. Đây cũng là dạng kết nối thường thấy ở ANN, trong CNN tầng này thường được sử dụng ở các tầng phía cuối của kiến trúc mạng.

### 2.2.6 Thiết kế của một CNN cơ bản

CNN có kiến trúc được hình thành từ các thành phần cơ bản bao gồm Convolution (CONV), Pooling (POOL), ReLU, Fully-connected (FC) về mặt xây dựng kiến trúc tổng quát CNN được mô tả như sau (dấu mũi tên thể hiện thứ tự sắp xếp các tầng từ trước đến sau).

$$Input \rightarrow [Conv \rightarrow ReLu] * N \rightarrow Pool \rightarrow [FC \rightarrow ReLu] * K \rightarrow FC$$

**Trong đó:**

- $[Conv \rightarrow ReLu] * N$  tức là trong kiến trúc này sau tầng Conv là tầng ReLu, trong CNN kiến trúc 2 tầng này có thể lặp N lần.
- Pool là tầng Pooling cho người thiết kế quyết định có thể có hoặc không.
- $[Conv \rightarrow ReLu] * N \rightarrow Pool$  trong kiến trúc CNN có thể lặp lại M lần kiểu sau tầng Conv là tầng ReLu và kế tới là tầng Pooling.
- $[FC \rightarrow ReLu] * K$  trong CNN có thể lặp K lần cấu trúc kiểu sau tầng FC là tầng ReLu nhưng trước nó phải có tầng  $[Conv \rightarrow ReLu]$ .

Ví dụ:

- $Input \rightarrow FC$  là một phân loại tuyến tính ( $N = K = M = 0$ ).
- $Input \rightarrow [Conv \rightarrow ReLu \rightarrow Pool] * 2 \rightarrow FC \rightarrow ReLu \rightarrow FC$  là một mạng CNNs với 2 lần  $Conv \rightarrow ReLu \rightarrow Pool$  trước khi cho ra 2 lớp FC với đầu ra.

## 2.3 Các thuật toán tối ưu cho mạng CNNs

### 2.3.1 Gradient Descent

Trong học máy, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc lớn nhất) của một hàm số nào đó. Ví như tìm giá trị nhỏ nhất của hàm mục tiêu, nhưng thực tế việc tìm được nó rất phức tạp thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minium và ở một mức nào đó, giá trị đó được coi như là nghiệm cần tìm của bài toán.

Các điểm local minium là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ các điểm cực tiểu, ta chỉ cần thay từng điểm đó vào hàm mục tiêu rồi tìm điểm là cho nó có giá trị nhỏ nhất. Tuy nhiên, trong hầu hết các trường hợp thì việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân đến từ sự đa dạng và phức tạp của hàm số, việc dữ liệu có số chiều lớn hoặc từ việc có quá nhiều dữ liệu.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với nghiệm của bài toán, sau đó dùng một phép toán lặp để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

GD là một trong những thuật toán phổ biến nhất để thực hiện việc tối ưu hóa và vì thế nó là cách phổ biến nhất để tối ưu hóa trong các mạng nơ-ron. Tại thời điểm này, tất cả các thư viện Deep-learning hiện đại nhất để triển khai các thuật toán khác nhau để tối ưu hóa của gradient descent.

**while True:**

    weights\_grad = evaluate\_gradient(loss\_fun, data, weights)

    weights += - step\_size \* weights\_grad

Vòng lặp đơn giản này lại là yếu tố thiết yếu của tất cả các kiến trúc mạng nơ-ron. Có nhiều cách khác nhau để thực hiện việc tối ưu hóa nhưng Gradient Descent hiện là phương pháp phổ biến nhất và nó được thiết lập để tối ưu hóa cho hàm mục tiêu trong các mạng nơ-ron.

### Batch Gradient descent

Vậy làm thế nào để tối ưu hoá các tham số  $W, b$  để hàm mục tiêu đạt giá trị nhỏ nhất, thuật toán được sử dụng là **Batch gradient descent** (BGD):

Trong mỗi vòng lặp chúng ta cập nhật các giá trị tham số theo công thức sau:

$$W_k \rightarrow W_k - \varepsilon \frac{\partial L}{\partial W_k} \quad (2.6)$$

$$b_k \rightarrow b_k - \varepsilon \frac{\partial L}{\partial b_k} \quad (2.7)$$



Trong đó  $\varepsilon$  được gọi là tốc độ hội tụ (*learning rate*).

Thuật toán này sẽ tính toán gradient cho toàn bộ tập dữ liệu và thực hiện cập nhật chỉ một lần. Việc này sẽ làm mất nhiều thời gian và tốn nhiều bộ nhớ trong quá trình huấn luyện

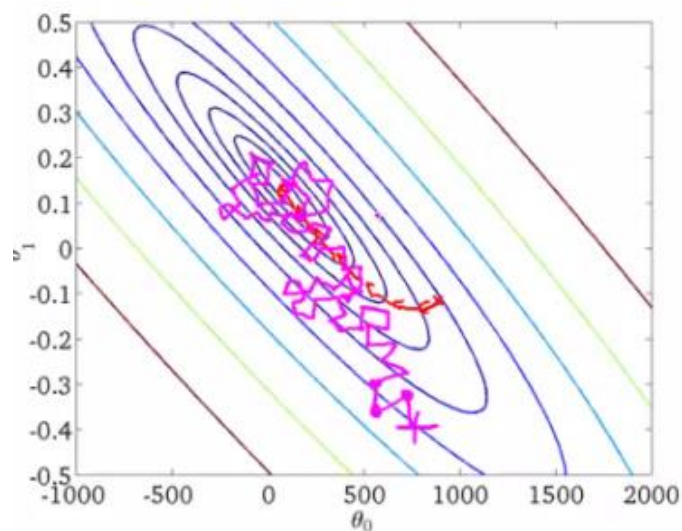
### Stochastic gradient descent:

Trong trường hợp của các tập dữ liệu rất lớn, sử dụng BGD có thể khá tốn kém như đã nói ở trên, BGD còn tính toán lại gradient đối với những ví dụ tương tự trước đó và BGD không cho phép cập nhật model tức thời. Do đó, đối với các tập dữ liệu lớn hơn, cần tìm một thuật toán giải quyết được những vấn đề đó.

```
while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

Hình 16: Thuật toán SGD [3]

Trong Stochastic Gradient Descent (SGD), ngược lại so với BGD, nó thực hiện việc cập nhật tham số cho từng mẫu dữ liệu trong quá trình training. Do đó, SGD thường nhanh hơn và có thể được sử dụng để học online.



Hình 17: SGD và batch gradient descent [3]

Hình 17 minh họa cho việc sử dụng SGD. Đường màu hồng sử dụng SGD, đường màu đỏ sử dụng batch gradient descent.

SGD có thể di chuyển nhanh chóng đến gần điểm cực đại nhưng không chạm điểm cực đại, do mỗi lần chỉ cập nhật trên một mẫu dữ liệu. Biện pháp khắc phục là giảm

learning rate qua từng vòng lặp, như vậy tham số học ra sẽ hội tụ về điểm có hàm mục tiêu nhỏ nhất.

### 2.3.2 Momentum update

Momentum update là một phương pháp có tính hội tụ tốt hơn trên các mạng nơ-ron sâu. Ý tưởng của Momentum Update đến từ góc độ vật lý về vấn đề tối ưu hóa. Đặc biệt, hàm mục tiêu có thể được hiểu như là một chiều cao của một địa hình đồi núi (và như vậy sẽ có thể năng  $U = mgh$  và do đó  $U$  tỉ lệ với  $h$ ). Khi khởi tạo các tham số với số ngẫu nhiên tương đương với việc đặt một hạt với vận tốc ban đầu bằng 0 tại một vị trí nào đó. Quá trình tối ưu hóa có thể coi như tương đương với quá trình mô phỏng các vector tham số (hay một hạt) lăn đến vị trí thấp nhất của đồi núi.

Bởi vì lực tác động trên các hạt có liên quan đến gradient của thế năng (ví dụ:  $F = -\nabla U$ ), và lực tác động đó tương đương với gradient của hàm mục tiêu.

Mặt khác,  $F = ma$  (âm) vì vậy gradient là ở mọi vị trí sẽ tỷ lệ thuận với sự tăng tốc của các hạt. Vì thế điều này cho thấy sự khác biệt của phương pháp này so với phương pháp SGD đã nói ở trên, khi mà gradient trực tiếp được lấy tích phân của vị trí này. Thay vào đó, quan điểm vật lý cho thấy một bản cập nhật trong đó gradient chỉ trực tiếp ảnh hưởng đến tốc độ, do đó có tác dụng trên các vị trí:

$$\begin{aligned}v &= \mu \times v - \text{learning rate} \times dx \\x+ &= v\end{aligned}\tag{2.8}$$

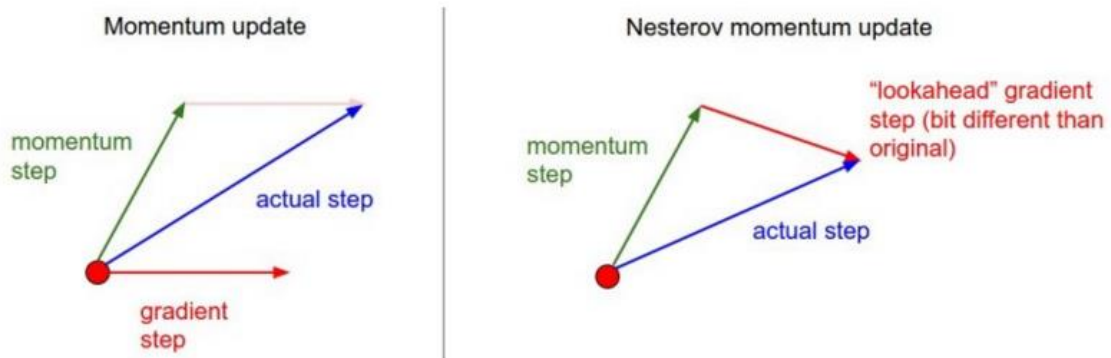
Trong công thức 2.4, giá trị tham số  $v$  được khởi tạo bằng 0, bên cạnh đó còn có thêm một siêu tham số  $\mu$  (**momentum**) đem lại hiệu quả trong việc tối ưu hóa (giá trị hay sử dụng bằng 0.9) hay trong ý nghĩa vật lý thì nó được coi như hệ số ma sát. Việc bổ sung thêm biến này sẽ khiến cho vận tốc hạt giảm dần và có thể dừng được tại đáy của đồi núi hay vị trí tối ưu của hàm mục tiêu. Với Momentum Update, vector tham số sẽ tăng vận tốc theo bất kỳ hướng nào có độ dốc phù hợp.

Một phiên bản phổ biến khác của Momentum Update là Nesterov Momentum:

$$\begin{aligned}x\_ahead &= x + \mu \times v \\v &= \mu \times v - \text{learning rate} \times dx\_ahead \\x+ &= v\end{aligned}\tag{2.9}$$

Công thức (2.9) thể hiện ý tưởng của Nesterov Momentum là thực hiện Momentum Update bằng giá trị của gradient tại điểm  $x\_ahead = x + \mu * v$ . Điều này giúp cho thuật toán trở nên nhanh hội tụ hơn.





Hình 18: Momentum update vs nesterov Momentum Update [3]

## 2.4 Các phương pháp tránh overfitting

### 2.4.1 Chuẩn hoá (regularization)

Trong huấn luyện, đôi khi không muốn “tối ưu quá”, vì như thế mô hình sẽ bị overfit vào dữ liệu huấn luyện và sẽ không đủ tổng quát để mô hình hóa cho dữ liệu mới. Regularization là cách để cân bằng hiện tượng này (hi sinh độ chính xác trên tập huấn luyện để làm giảm độ phức tạp của mô hình).

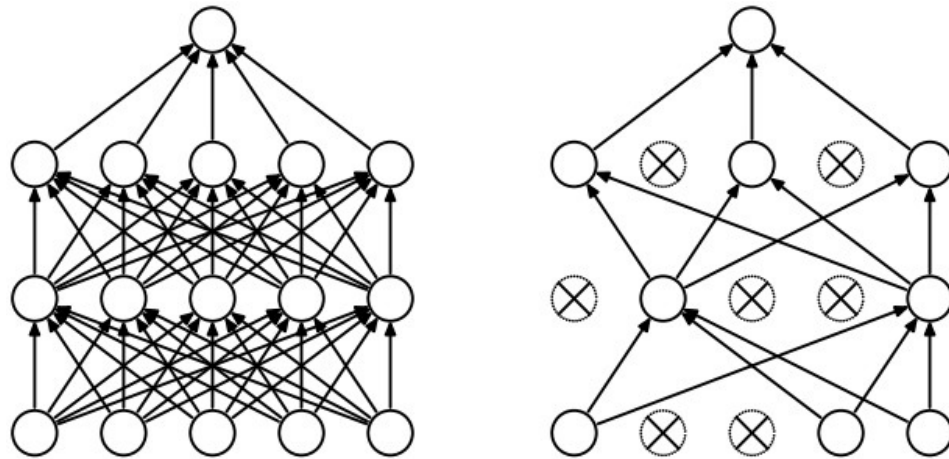
***L2 regularization*** là phương pháp phổ biến nhất cho tại thời điểm hiện tại.

Nó thêm vào hàm mục tiêu một lượng  $\frac{1}{2} \lambda w^2$  ( $\lambda$  được gọi là hằng số bình thường hóa). Bằng cách thêm vào một lượng nhỏ hàm mục tiêu, sau một số vòng lặp nhất định, mô hình sẽ có xu hướng sử dụng càng ít trọng số càng tốt.

***L1 regularization*** là một dạng phổ biến khác của regularization, thay vì như ***L2 regularization*** mỗi trọng số  $w$  chỉ được cộng thêm một lượng  $\lambda|w|$ . Nó cũng có thể kết hợp với ***L2*** như sau:  $\lambda_1|w| + \lambda_2 w^2$  (Elastic net regularization). ***L1*** tạo ra nên các vector thưa trong quá trình tối ưu hóa. Nói cách khác, nơ-ron mà sử dụng chuẩn ***L1*** sẽ tạo thành một tập hợp các giá trị đầu vào thưa thớt nhưng lại mang các đặc trưng quan trọng của nó. Trong nhiều trường hợp, ***L2 regularization*** đem lại hiệu năng cao hơn ***L1 regularization*** khi được sử dụng.

### 2.4.2 Loại bỏ kết nối (Drop out)

Drop out là phương pháp cực kì đơn giản và hiệu quả để giảm overfit cho mô hình học sâu. Khi huấn luyện, dropout chỉ giữ lại một vài nơ-ron active với xác suất  $p$  ( $p$  là một hyperparameter).



Hình 19: Mạng nơ-ron thông thường (trái) và mạng nơ-ron có sử dụng dropout (phải) [3]

Thuật toán sử dụng Dropout với một mạng nơ-ron 3 lớp:

```
""" Vanilla Dropout: Not recommended implementation (see notes
below) """
```

```
p = 0.5 # probability of keeping a unit active. higher = less
dropout
```

```
def train_step(X):
    """ X contains the data """
```

```
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3
```

```
    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

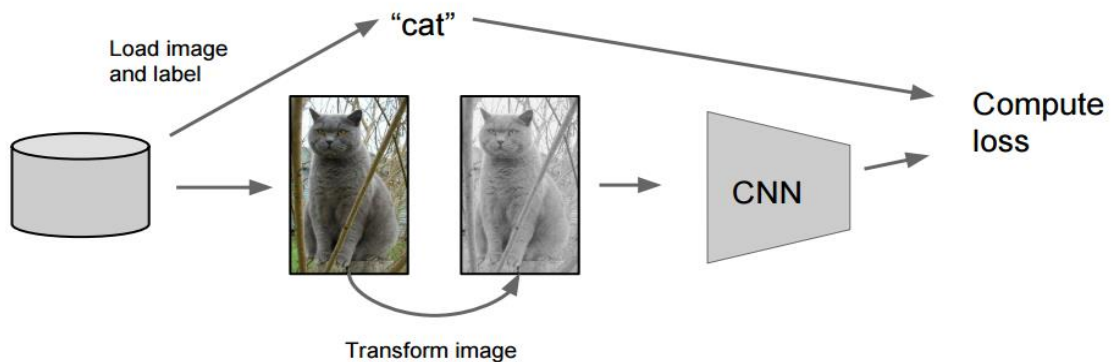
```
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the
activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the
activations
    out = np.dot(W3, H2) + b3
```

Trong đoạn code nói trên, hàm `train_step` thực hiện lọc dropout 2 lần: một ở lớp ẩn thứ 1 và một ở lớp ẩn thứ 2. Giá trị của đầu ra là không đổi và được lọc qua các ma trận nhị phân  $U_1$  và  $U_2$ . Chú ý rằng ở hàm `predict` chúng ta không cần sử dụng

dropout và như thế tại thời điểm kiểm tra độ chính xác của mô hình chúng ta có thể test trên tất cả các tham số. Giả sử một nơ-ron có đầu ra khi không có drop out là  $x$  với các trọng số và giá trị các điểm là gần như tương đương nhau, khi có dropout tại bước trên đầu ra mô hình là:  $(x * p + 0 * (1 - p)) / p = x$ .

### 2.4.3 Sinh thêm dữ liệu (data augmentation)

Đối với bài toán phân loại ảnh, ta có thể thay đổi dữ liệu của ảnh mà không cần thay đổi nhãn bằng các phép biến đổi ảnh.

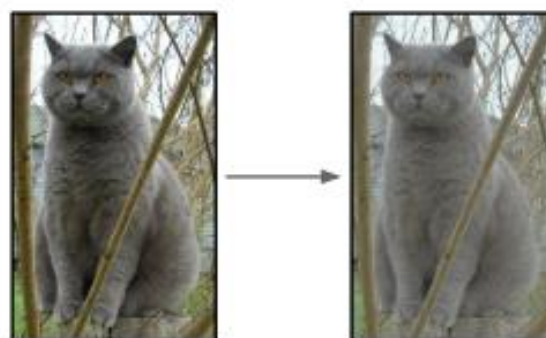


Hình 20: Phương pháp sinh thêm dữ liệu trước khi huấn luyện [3]

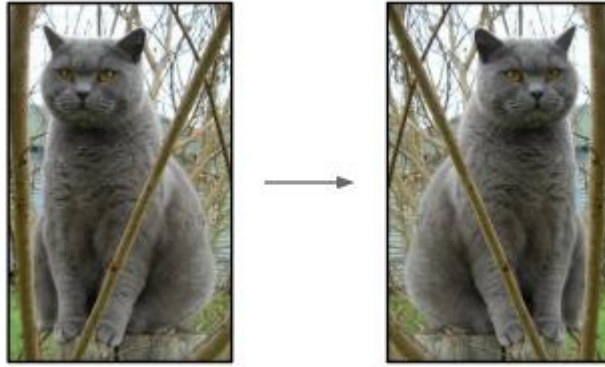
Cách làm này tạo thêm dữ liệu, giúp tránh overfit cũng như giúp việc nhận dạng tốt hơn khi điều kiện môi trường thay đổi (ảnh có nhiễu, vật thể nghiêng...).

Các phương pháp sinh thêm dữ liệu bao gồm:

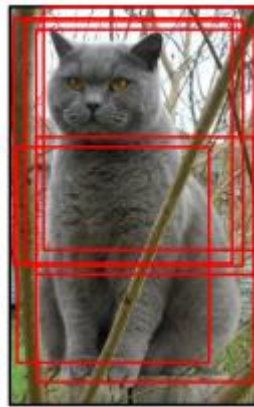
- Đối xứng ngang: Đối xứng ảnh qua trục dọc.
- Thêm nhiễu: Thêm vào ảnh 1 vài điểm ảnh nhiễu nhỏ mà người dùng vẫn có thể nhận biết nội dung của ảnh.
- Xoay ảnh: xoay ảnh 1 góc từ 0 đến 20°.
- Cắt và dẫn ảnh.
- Biến đổi affine.
- Biến đổi không gian màu sắc.



Hình 21: Biến đổi không gian màu sắc [3]



Hình 22: Xoay, lật hình trái phải [3]

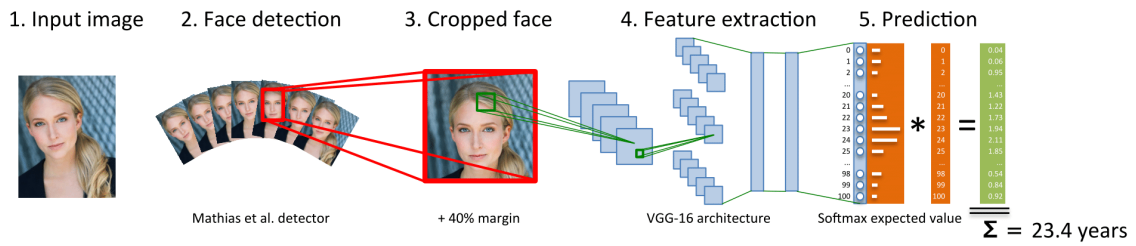


Hình 23: Crop, resize hình ảnh [3]

## 2.5 Bài toán phân loại ảnh

Bài toán phân loại ảnh là một lớp bài toán nhỏ hơn của bài toán phân loại khi đối tượng dữ liệu ở đây là ảnh và tập nhãn tương ứng là tập các loại đối tượng. Tuy vậy, bài toán phân loại thực trên thực tế cũng vô cùng đa dạng và phức tạp. Sự đa dạng, phức tạp của bài toán đến từ tập dữ liệu và tập nhãn. Nguồn dữ liệu ảnh có thể là dữ liệu ảnh trên các trang web, dữ liệu ảnh xã hội, ảnh công ty, sách vở... Kích thước ảnh có thể biến đổi từ rất nhỏ cho đến rất lớn, số lượng ảnh có thể thay đổi từ hàng trăm cho đến hàng triệu ảnh. Tập nhãn của ảnh cũng có thể thay đổi tùy vào bài toán. Trong một số bài toán cụ thể, tập nhãn có thể liên quan đến những đối tượng cụ thể (những mặt hàng, sản phẩm cụ thể) cho đến những bài toán khó khi tập nhãn mang tính trừu tượng hơn.

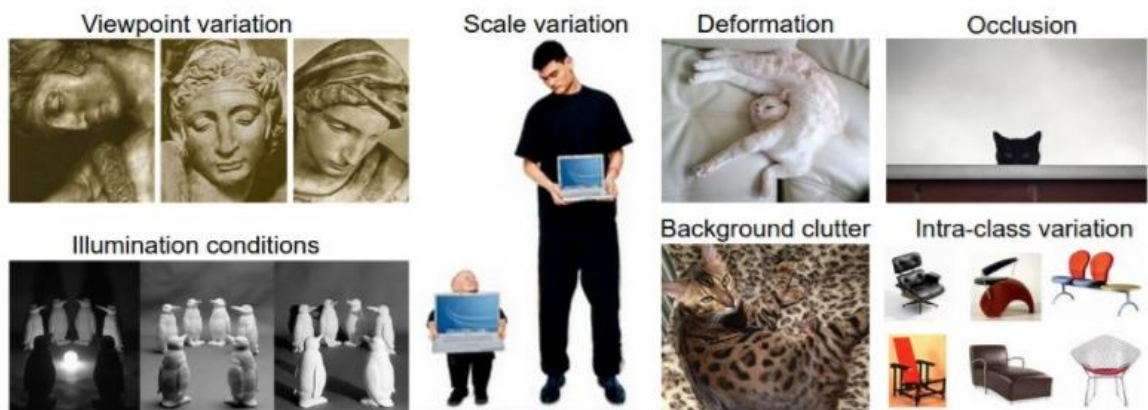
Để lưu trữ và xử lý, ảnh được số hóa trước khi đưa vào máy tính. Ví dụ, để biểu diễn ảnh trong hệ màu RGB, ảnh được số hóa thông qua 3 ma trận hệ số màu tương ứng với các màu đỏ (R), màu xanh lục (G) và màu xanh lam (B).



Hình 24: Bài toán nhận diện hình ảnh [10]

### Các khó khăn của bài toán nhận diện ảnh:

- Sự thay đổi góc nhìn: Đối tượng ảnh trong thực tế là các đối tượng ba chiều. Khi chụp ảnh, ta chỉ thể hiện hình chiếu đối tượng trong không gian 2 chiều. Tùy theo góc độ, hình ảnh biểu diễn của đối tượng có thể thay đổi khác nhau.
- Sự thay đổi tỉ lệ: Sự thay đổi kích thước đối tượng trong các ảnh khác nhau
- Biến dạng đối tượng: Trong nhiều trường hợp, đối tượng ảnh không ở hình dạng thường thấy, mà ở dạng đặc biệt, khiến cho việc phân loại đối tượng trở nên khó khăn hơn.
- Đối tượng bị che khuất: Đối tượng không xuất hiện đầy đủ mà chỉ hiển thị một phần.
- Điều kiện ánh sáng khác nhau: điều kiện khác nhau dẫn đến ảnh dữ liệu thu thập về đối tượng có sự biến đổi lớn.
- Nhầm lẫn với môi trường: Đối tượng trong ảnh dễ bị lẫn với môi trường xung quanh.
- Sự đa dạng trong một lớp: Một lớp đối tượng có thể có nhiều biến thể, hình dạng, mẫu mã khác nhau.



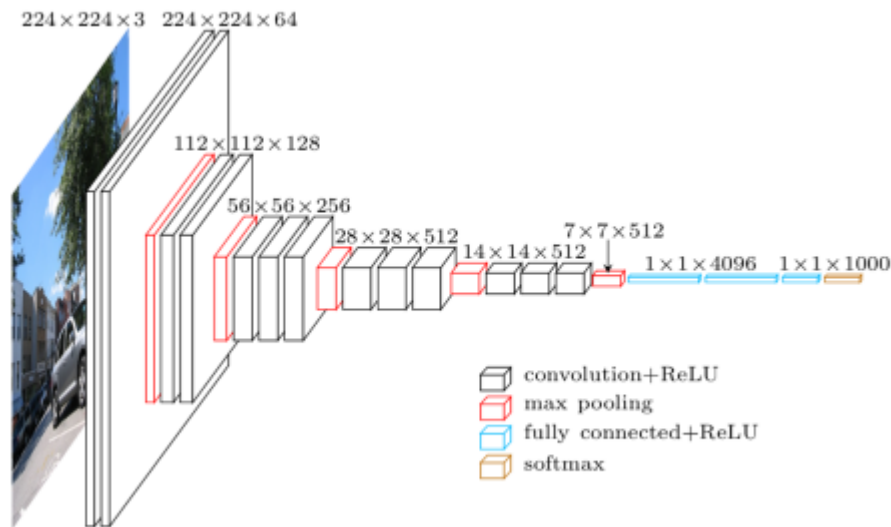
Hình 25: Khó khăn trong việc nhận diện hình ảnh [3]



## PHẦN 3. KIẾN TRÚC CỦA MẠNG DENSENET

### 3.1 Tổng quan một số kiến trúc mạng hiện đại

#### 3.1.1 Mạng VGG-Net



Hình 26: Kiến trúc VGG-Net [8]

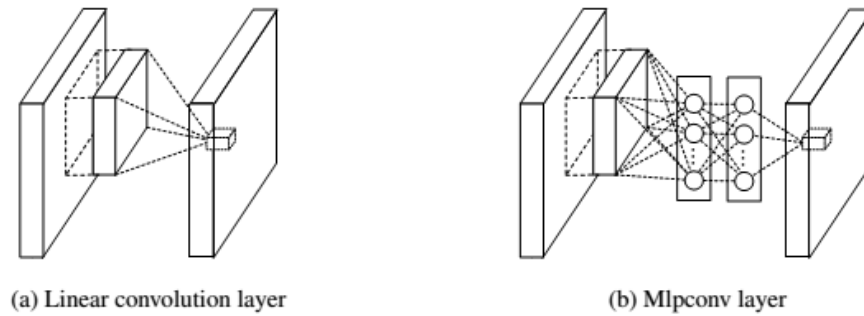
Kiến trúc mạng VGG được giới thiệu bởi Simonyan và Zisserman trong báo cáo của họ vào năm 2014. Kiến trúc này rất đặc trưng bởi tính đơn giản của nó, chỉ sử dụng lớp tích chập  $3 \times 3$  chồng lên nhau trong sự gia tăng chiều sâu. Việc giảm kích thước được xử lý bằng hàm max pooling. Hai lớp fully-connected, mỗi lớp có 4096 nodes và sau cùng làm hàm softmax.

VGG sử dụng nhiều **Conv** lớp filter nhỏ thay cho một lớp **Conv** với filter lớn có hiệu quả như nhau nhưng lại học sâu hơn, nhiều lớp phi tuyến tính hơn và dùng ít tham số hơn.

Ví dụ: sắp xếp ba lớp Conv  $3 \times 3$  liên tiếp (với lớp non-linearities ở giữa) sẽ đem lại hiệu quả tương đương như một lớp Conv  $7 \times 7$  nhưng số lượng tham số sẽ là  $3 \cdot (3^2 C^2)$  so với  $7^2 C^2$  cho mỗi C channel trên mỗi lớp.

#### 3.1.2 Mạng Network in Network

Các bộ lọc tích chập trong CNN là một mô hình tuyến tính tổng quát (generalized linear model-GLM) cho dữ liệu, được cho rằng có mức độ trừu tượng thấp. Thay thế GLM với một mô hình phi tuyến mạnh hơn có thể nâng cao khả năng trừu tượng của mô hình địa phương. Trong NIN, các GLM được thay thế bằng một cấu trúc “micro network”: một perceptron đa lớp sau lớp conv và cũng đồng thời một mạng lưới thần kinh để huấn luyện bằng cách lan truyền ngược.



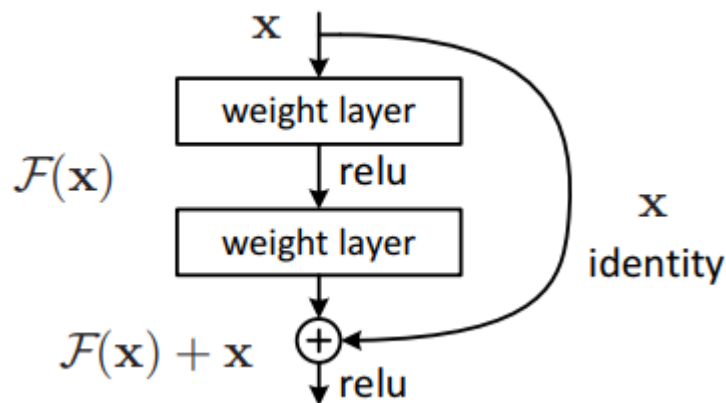
Hình 27: Kiến trúc một mạng NiN [11]

Kiến trúc này được gọi là lớp mlpconv, lớp này tính toán trên một địa phương đầu vào cho ra một vector với nhiều lớp perceptron sau lớp conv. Kiến trúc strack nhiều lớp mlpconv được gọi là Network-in-network (NiN). Có thể thấy việc kết nối giữa các đầu ra tương đương với việc kết nối giữa các đầu ra theo chiều sâu và tương đương với một lớp conv  $1 \times 1$ .

### 3.1.3 Mạng ResNet

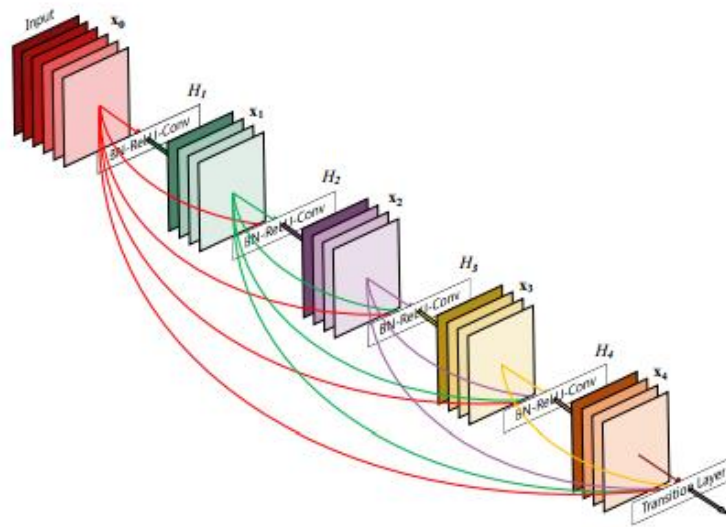
Deep residual learning networks (ResNet) là một mạng được phát triển bởi các nhà nghiên cứu từ Microsoft Research. Các kết quả khá ấn tượng ở chỗ nó đã nhận được vị trí đầu tiên trong ILSVRC 2015 phân loại ảnh. Kiến trúc ResNet được đề xuất kết nối dư (residual), từ lớp trước tới lớp hiện tại. Nói chung, đầu vào của lớp  $L_i$  được tổng kết từ đầu ra của lớp trước đó.

Hình 26 là một minh họa cho kết nối trong mạng ResNet.



Hình 28: Một khối building trong mạng ResNet [6]

## 3.2 Mạng DenseNet



Hình 29: Kiến trúc mạng DenseNet với 5 dense block và growth rate  $k = 48$  [1]

DenseNet là một kiến trúc mạng mới được xây dựng và phát triển bởi **Gao Huang** và đồng nghiệp. DenseNet là một kiến trúc đem lại hiệu năng cao so với các mạng hiện đại gần đây như VGG-Net, ResNet trên nhiều tập dữ liệu bao gồm cả tập dữ liệu có quy mô lớn như ImageNet. Hình 29 mô tả một kiến trúc mạng DenseNet. Khi CNN càng ngày càng sâu hơn, một vấn đề mới xuất hiện: “Thông tin của đầu vào và gradient đi qua nhiều lớp, nó có thể bị mất mát khi đi đến cuối mạng”. Vấn đề này đã được nhiều bài báo công bố và đưa ra các giải pháp về kiến trúc mạng khác nhau để giải quyết điều này. Ví dụ như kiến ResNet thì thêm các kết nối shortcut giữa lớp này với lớp trước đó. Không giống với ResNet cũng như các mô hình truyền thống, DenseNet liên kết các layer lại với nhau để tổng hợp các đặc trưng riêng. Các lớp phía trước được xem như đầu vào của các lớp phía sau. Do đó, trong DenseNet, lớp thứ  $l$  có  $l$  đầu vào bao gồm  $l$  đầu ra feature-map của các lớp trước đó và feature-map của nó cũng là đầu vào cho  $L - l$  lớp tiếp theo ( $L$  là độ sâu của mạng). Như vậy sẽ có  $\frac{L(L+1)}{2}$  kết nối trong mạng có  $L$  lớp thay vì  $L$  kết nối như các kiến trúc truyền thống trước đây. Bởi vì mật độ dày đặc của các connection trong mạng nên cái tên DenseNet Convolution Network được đặt cho kiến trúc mới này.

### Ưu điểm của mạng DenseNet:

- Số lượng tham số ít hơn so với các mạng truyền thống (tính toán nhanh hơn, thời gian training nhanh hơn), bởi vì không phải huấn luyện lại các feature-map. Kiến trúc lan truyền xuôi truyền thống có thể được xem như là một trạng thái cái mà được truyền từ lớp này đến lớp khác. Mỗi lớp sẽ đọc trạng thái từ lớp trước đó rồi xử lý và lại truyền cho lớp sau. Dù trạng thái được thay đổi nhưng thông tin truyền đi vẫn



được bảo tồn. Kiến trúc ResNet bảo tồn thông tin này thông qua một lớp shortcut. Một số biến thể của ResNet đã cho thấy rằng có nhiều lớp thì đóng góp rất ít hoặc là bị drop ngẫu nhiên trong suốt quá trình huấn luyện. Điều này khiến cho trạng thái của ResNet tương tự như các mạng Recurrent Neural Network, nhưng tham số lại lớn hơn đáng kể bởi vì mỗi lớp lại có trọng số riêng của nó. Mặt khác, mạng DenseNet lại phân biệt rõ ràng giữa thông tin được thêm vào mạng và thông tin cần được bảo tồn. Các lớp trong mạng DenseNet thì rất hẹp, có thể chỉ cần thêm vào một tập nhỏ các feature-maps tới “collective knowledge” của mạng và giữ cho các feature-maps không thay đổi, và lớp phân loại cuối cùng sẽ quyết định kết quả dựa trên các feature-maps có trong mạng.

- Cải thiện luồng thông tin cũng như gradient trong mạng, giúp cho quá trình huấn luyện trở lên dễ dàng hơn. Mỗi lớp có thể truy cập trực tiếp đến gradients từ hàm mục tiêu và đầu vào, dẫn đến một sự giám sát sâu tiềm ẩn. Điều này giúp cho kiến trúc mạng có thể sâu hơn, hơn thế nữa, nó còn giảm được vấn đề overfitting đối với các tập dữ liệu nhỏ.

### Cấu trúc mạng DenseNet.

Xem xét một hình ảnh  $x_0$  được truyền qua mạng tích chập. Mạng bao gồm  $L$  lớp, mỗi lớp trong số đó thực hiện một phép biến đổi phi tuyến tính  $H_l(.)$ , trong đó  $l$  là chỉ số của lớp thứ  $l$ .  $H_l(.)$  có thể là một hàm tổng hợp của các phép toán như Batch Normalization (BN), rectified linear units (ReLU), Pooling hoặc là Convolution(Conv). Chúng ta định nghĩa đầu ra của lớp thứ  $l$  là  $x_l$ .

**ResNet.** Trong các mạng tích chập truyền thống feed-forward kết nối đầu ra của lớp thứ  $l$  như là đầu vào của lớp thứ  $l + 1$ , làm tăng sự chuyển đổi của lớp sau:

$$x_l = H_l(x_{l-1})$$

ResNet đã thêm một skip-connection làm tăng các phép biến đổi phi tuyến tính với một hàm định danh:

$$x_l = H_l(x_{l-1}) + x_{l-1}$$

Một lợi thế của mạng ResNet là gradient có thể được cải thiện từ các lớp sau này đến các lớp trước đó. Tuy nhiên, hàm định danh(*identity*) và đầu ra của phép biến đổi phi tuyến tính lại được kết hợp bằng cách tổng kết lại lớp trước đó, điều này có thể làm cản trở dòng thông tin trong mạng.

### Dense connectivity.

Để cải thiện luồng thông tin giữa các lớp, một mô hình kết nối khác biệt: “Lớp thứ  $l$  nhận đầu vào là đặc trưng riêng(feature-map) tất cả đầu ra của các lớp trước nó” được đề xuất.

$$x_l = H_l([x_0, x_1, \dots, x_{l-1}])$$

Trong đó:  $[x_0, x_1, \dots, x_{l-1}]$  được hiểu là sự kết nối các đặc trưng riêng của các lớp từ 0 đến  $l - 1$ . Để thuận tiện cho việc triển khai mô hình, các đầu vào của hàm phi tuyến tính  $H_l(\cdot)$  được liên kết với nhau trong một single tensor.

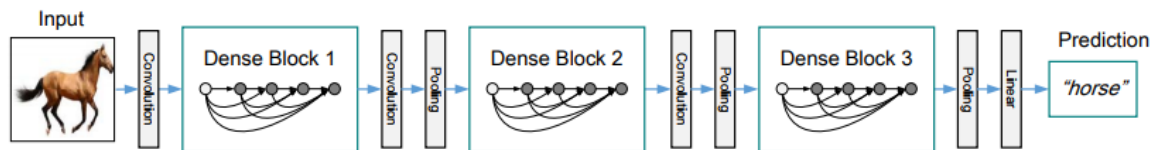
### Composite Function.

Trong quá trình tìm hiểu, phát triển và thử nghiệm nhiều lần thì phi tuyến tính  $H_l$  giống như là một hàm tổng hợp gồm các phép toán theo thứ tự:

$$BN \rightarrow ReLu \rightarrow Conv(3 \times 3) \rightarrow [Dropout]$$

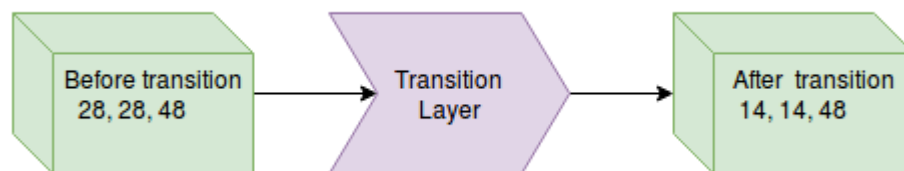
Đem lại hiệu năng cao trong cả quá trình huấn luyện cũng như dự đoán dữ liệu mới.

### Pooling layers.



Hình 30: Mạng DenseNet với 3 dense block [1]

Trong công thức trên, do các phép toán kết nối có thể không sử dụng được do sự thay đổi về kích thước của feature-map. Tuy nhiên, một phần thiết yếu của mạng nơ-ron là down-sampling các lớp tức là làm thay đổi kích thước của feature-map từ đầu ra của lớp trước sao cho phù hợp với đầu vào của lớp sau. Vì vậy, DenseNet được chia thành các dense block giống như hình 30, giữa hai block là một lớp chuyển tiếp (transition layers) bao gồm một lớp **BN**, một lớp **Conv**  $1 \times 1$  và cuối cùng là một lớp **average pooling**  $2 \times 2$ .



Hình 31: Ví dụ về transition layer [9]

### Growth rate.

Nếu mỗi hàm  $H_l$  tạo ra  $k$  feature-maps thì lớp thứ  $l$  sẽ có  $k_0 + k * (l - 1)$  feature-maps đầu vào ( $k_0$  là số lượng channels của lớp đầu vào). Một khác biệt quan trọng giữa DenseNet và các kiến trúc mạng đã có hiện nay là DenseNet có thể các lớp rất hẹp.  $k$  (growth rate) được coi như là tốc độ tăng trưởng của mạng. Chỉ với một tỷ lệ  $k$  nhỏ cũng có thể cho hiệu quả huấn luyện cao trên tập dữ liệu. Lý do là vì mỗi lớp có khả năng truy cập tới tất cả các feature-maps trước đó trong block của nó, và vì thế lớp đó có quyền truy cập đến “collective knowledge” của mạng. Có thể xem

feature-maps như là trạng thái chung của mạng. Mỗi lớp sẽ thêm  $k$  feature-maps vào trạng thái này. Giá trị **growth rate** sẽ điều chỉnh số lượng thông tin mà mỗi lớp sẽ bổ sung vào trạng thái chung. Trạng thái này khi đã được lưu lại thì có thể được truy cập từ mọi nơi trong mạng và không giống như các kiến trúc mạng truyền thống, không cần phải sao chép nó từ lớp này sang lớp khác.

### Bottleneck layers.

Mặc dù mỗi lớp chỉ tạo  $k$  feature-maps, nó thông thường có nhiều hơn một đầu vào. Sử dụng lớp **Conv**  $1 \times 1$  như là *bottleneck layer* trước mỗi **Conv**  $3 \times 3$  để làm giảm số lượng feature-maps đầu vào cũng như số lượng tham số và vì thế để tăng thời gian, hiệu quả tính toán. Qua nhiều quá trình thử nghiệm và tính toán thì tổ hợp các hàm theo thứ tự :

$$BN - ReLU - Conv(1 \times 1) - BN - ReLU - Conv(3 \times 3)$$

được coi như là một lớp bottleneck (hay còn gọi là hàm phi tuyến tính  $H_1$  mà đã được giới thiệu trong công thức phía trên) đem lại hiệu quả tốt nhất.

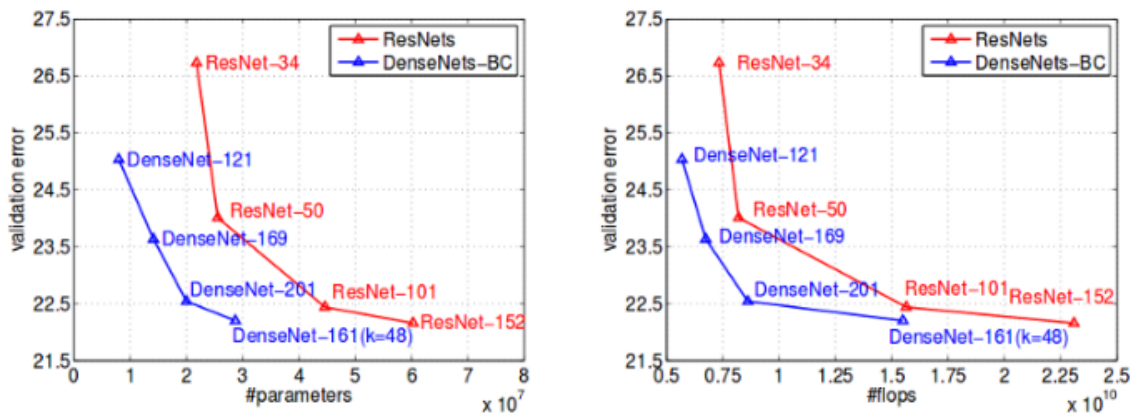
### Compression.

Để củng cố sự chặt chẽ cho mô hình, DenseNet có thể làm giảm đi số lượng feature-maps tại các lớp transition (lớp chuyển tiếp giữa các dense block). Khi một dense block chứa  $m$  feature-maps, thì lớp transition phía sau đó sẽ sinh  $[\theta m]$  feature-maps trong đó  $0 < \theta < 1$  được gọi là hệ số nén. Khi  $\theta = 1$  số lượng feature-maps đến và đi qua lớp transition là không thay đổi.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	$112 \times 112$	$7 \times 7$ conv, stride 2			
Pooling	$56 \times 56$	$3 \times 3$ max pool, stride 2			
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$	$1 \times 1$ conv			
	$28 \times 28$	$2 \times 2$ average pool, stride 2			
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$	$1 \times 1$ conv			
	$14 \times 14$	$2 \times 2$ average pool, stride 2			
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$	$1 \times 1$ conv			
	$7 \times 7$	$2 \times 2$ average pool, stride 2			
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$	$7 \times 7$ global average pool			
		1000D fully-connected, softmax			

Hình 32: Kiến trúc cụ thể trong mạng với growth rate  $k = 32$  [1]

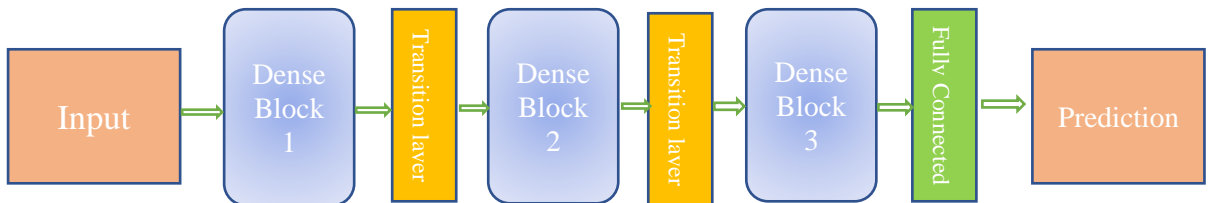
Tóm lại, mạng DenseNet trong quá trình thực nghiệm đem lại hiệu năng xấp xỉ với các mạng truyền thống hiện nay như Highway Network, Network in Network, ResNet, ... nhưng số lượng tham số nhỏ hơn rất nhiều và thời gian training cũng nhanh hơn.



Hình 33: Kết quả so sánh giữa ResNet và DenseNet-BC [1]

### Mô hình thực nghiệm

Mô hình được thiết kế và phát triển theo mô hình đề xuất của mạng DenseNet đem lại hiệu năng cao trên tập dữ liệu và tập test:



Hình 34: Model densenet cài đặt thử nghiệm

- ✚ **Dense block:** Các layer được liên kết với các layer trước đó để lấy được nhiều thông tin cho quá trình huấn luyện.  
 $BN \rightarrow ReLu \rightarrow Conv\ 1x1 \rightarrow dropout \rightarrow BN \rightarrow ReLu \rightarrow Conv\ 3x3 \rightarrow dropout$
- ✚ **Transition layer:** lớp chuyển tiếp làm down-sampling không gian (giảm không gian kích thước)  
 $BN \rightarrow ReLu \rightarrow Conv(1x1) \rightarrow Dropout \rightarrow Average\ Pooling(2x2)$
- ✚ **Fully-Connected:**  
 $BN \rightarrow ReLu \rightarrow Average\ Pooling \rightarrow Flatten \rightarrow dense$

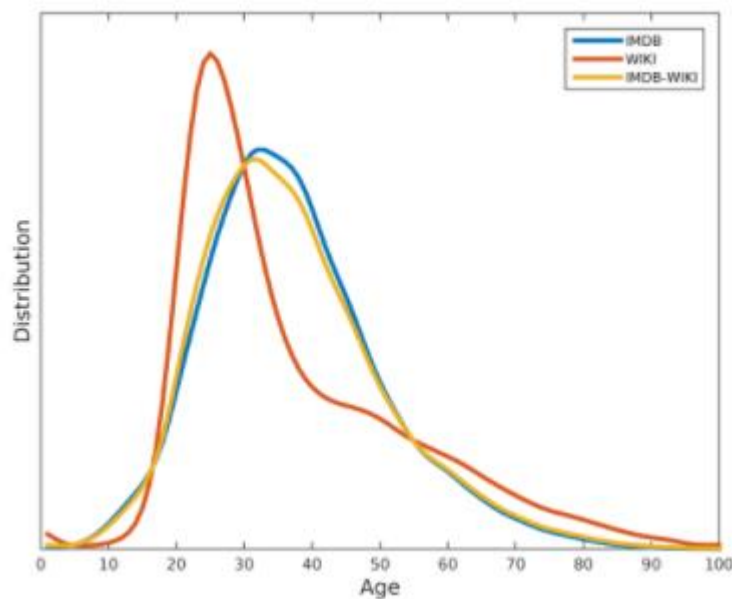
Hình 34 mô tả một mô hình bao gồm 3 lớp Dense Block ở giữa mỗi lớp là một lớp transition có chức năng làm giảm không gian, kích thước ảnh sao cho phù hợp với kích thước đầu vào của Dense Block kế tiếp. Giá trị input sau khi đi qua Dense Block cuối cùng sẽ được đưa vào lớp Fully Connected để có thể được phân loại.

## PHẦN 4. THỰC NGHIỆM VÀ ĐÁNH GIÁ

Để đánh giá được hiệu quả, thời gian training của mô hình sử dụng, báo cáo tiến hành các thử nghiệm về tham số các tham số có thể thay đổi như:

- Learning rate: Tốc độ hội tụ của mạng.
- Growth rate: Tốc độ tăng trưởng của mạng.
- Depth: Độ sâu của mạng.
- Image\_size: Kích thước ảnh.

### 4.1 Mô tả bộ dữ liệu



Hình 35: Sự phân phối của tập dữ liệu dựa trên độ tuổi [10]

Bộ dữ liệu wiki bao gồm 62328 ảnh độ phân giải 96x96 là khuôn mặt của nhiều người với các góc cạnh khác nhau chứa các thông tin về độ tuổi, giới tính.



Hình 36: Ví dụ về tập dữ liệu

Từ thông số được cung cấp, toàn bộ dữ liệu được đọc và xử lý đưa về cùng một chuẩn kích thước sau đó lưu lại thành file **tfrecord** để thuận tiện cho quá trình huấn luyện.

## 4.2 Cài đặt thực nghiệm


Chương trình thực hiện trên mô hình mạng DenseNet với các tham số đầu vào được thay đổi trong mỗi lần thực nghiệm để tìm được tham số tối ưu cho bài toán. Mô hình thiết lập tốc độ hội tụ (learning rate) được khởi tạo ban đầu bằng 0.1, sau mỗi một phần năm lần tổng số vòng lặp giá trị sẽ giảm đi 10 lần.

Bộ dữ liệu bao gồm 38000 ảnh được chia thành hai bộ: một bộ training và một bộ kiểm theo tỉ lệ lần lượt là 80% và 20%. Bộ dữ liệu training được chia thành hai tập dữ liệu train và validation. Hệ thống sẽ học train bộ dữ liệu training và kiểm tra hàm mục tiêu trên bộ dữ liệu validation trong  $n$  epochs (mỗi epoch là một lần huấn luyện trên toàn bộ tập dữ liệu huấn luyện). Giá trị  $n$  thực nghiệm được thiết lập bằng 300.

Trong quá trình training, bộ dữ liệu cũng được chia thành từng phần ngẫu nhiên. Hình ảnh cũng được tiền xử lý trước với một số phương pháp để làm đa dạng dữ liệu.

### Các tham số:

- Growth rate: [12, 48].
- Image size: kích thước của ảnh [32, 48, 64, 96].
- Learning rate khởi tạo bằng 0.1.
- Chu kì (epochs) thuộc [100, 300].
- Depth: Độ sâu của mạng [40, 100].

 Hàm mục tiêu: Mô hình thực nghiệm sử dụng weight decay bằng  $10^{-4}$ , hàm **Nesterov momentum** với giá trị momentum = 0.9 và không giảm trong quá trình huấn luyện (theo [1]).

```
logits = DenseNet(x=x, nb_blocks=nb_blocks, filters=growth_k, training=training_flag).
        model
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=label, logits=
        logits), name="cost")

costs = []
for var in tf.trainable_variables():
    costs.append(tf.nn.l2_loss(var))
    tf.summary.histogram(var.op.name, var)
l2_loss = tf.add_n(costs)

optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=
        nesterov_momentum, use_nesterov=True, name="optimizer")
train = optimizer.minimize(cost + l2_loss * weight_decay, name='train')

correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(label, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```



## ✚ Lóp transition

```
def transition_layer(self, x, scope):
    with tf.name_scope(scope):
        x = batch_normalization(x, training=self.training, scope=
            scope + '_batch1')
        x = tf_relu(x)
        x = conv_layer(x, filter=self.filters, kernel=[1, 1],
            layer_name=scope + '_conv1')
        x = tf_dropout(x, rate=dropout_rate, training=self.training)
        x = tf_average_pooling(x, pool_size=[2, 2], stride=2)
    return x
```

## ✚ Các Dense Block

```
def bottleneck_layer(self, x, scope):
    with tf.name_scope(scope):
        x = batch_normalization(x, training=self.training, scope=
            scope + '_batch1')
        x = tf_relu(x)
        x = conv_layer(x, filter=4 * self.filters, kernel=[1, 1],
            layer_name=scope + '_conv1')
        x = tf_dropout(x, rate=dropout_rate, training=self.training)

        x = batch_normalization(x, training=self.training, scope=
            scope + '_batch2')
        x = tf_relu(x)
        x = conv_layer(x, filter=self.filters, kernel=[3, 3],
            layer_name=scope + '_conv2')
        x = tf_dropout(x, rate=dropout_rate, training=self.training)
    return x

def dense_block(self, input_x, nb_layers, layer_name):
    with tf.name_scope(layer_name):
        layers_concat = list()
        layers_concat.append(input_x)

        # Run for bottle 0
        x = self.bottleneck_layer(input_x, scope=layer_name +
            '_bottleN_' + str(0))

        layers_concat.append(x)
        for i in range(nb_layers - 1):
            x = concatenation(layers_concat)
            x = self.bottleneck_layer(x, scope=layer_name +
                '_bottleN_' + str(i + 1))
            layers_concat.append(x)
    return x
```

#### 🚦 Lớp Fully-connected

```
x = batch_normalization(x, training=self.training, scope='linear_batch')
x = tf_relu(x)
x = global_average_pooling(x)
x = flatten(x)
x = Linear(x)
```

### 4.3 Kết quả thực nghiệm

Chương trình được cài đặt bằng ngôn ngữ Python kết hợp với các thư viện: Tensorflow, OpenCV,... OpenCV một thư viện mã nguồn mở hàng đầu cho thị giác máy tính (computer vision), xử lý ảnh và máy học, và các tính năng tăng tốc GPU trong hoạt động thời gian thực. Tensorflow đã được giới thiệu ở phần trước đó.

Bảng cấu hình máy thử nghiệm:

STT	Phần cứng	Loại
1	CPU	CPU Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz - 56 cores
2	RAM	126GB
3	GPU	GK107GL [Quadro K600]

Độ đo sự chính xác:

$$\text{accuracy} = \frac{1}{n} \sum_{k=0}^n x_k$$

Trong đó:

$$x_k = \begin{cases} 1 & \text{nếu dự đoán chính xác} \\ 0 & \text{nếu dự đoán sai} \end{cases}$$

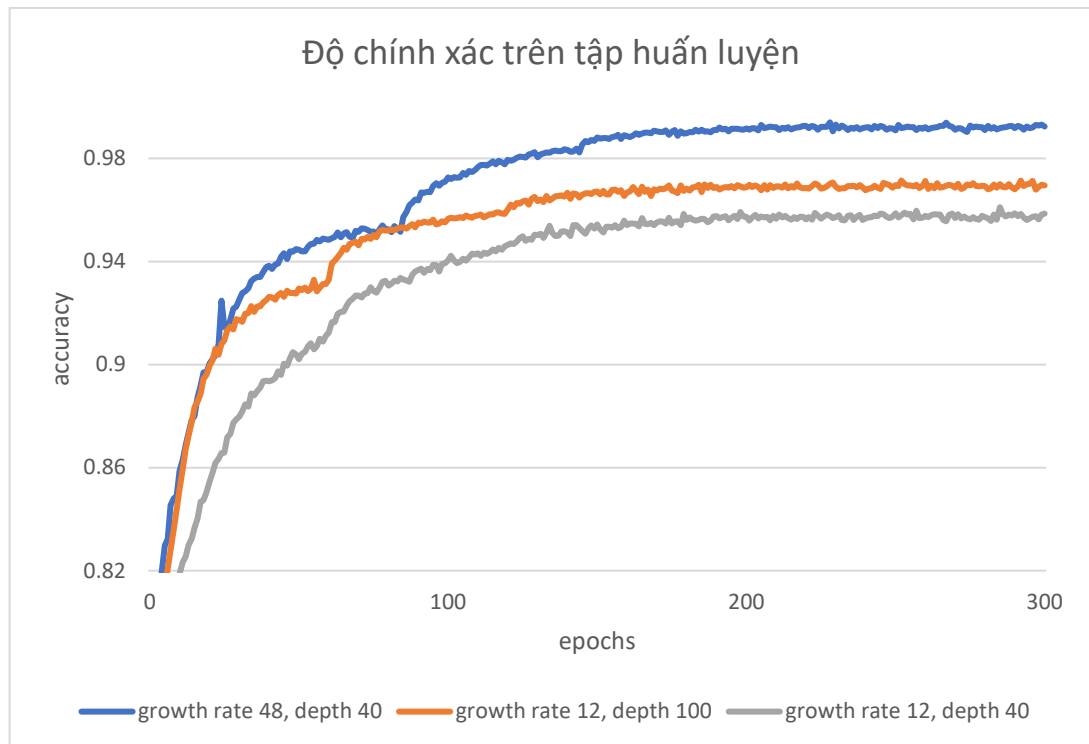
Các biểu đồ bên dưới thể hiện hiệu năng của mô hình mạng DenseNet đối với dữ liệu trên tập training và tập test với kích thước ảnh 96x96, kích thước của một batch là 64, các tham số khác đã được mô tả như bên dưới.

Tập dữ liệu được huấn luyện 300 lần (epochs). Các biểu đồ bên dưới thể hiện hiệu quả của mô hình trong đó trực tiếp tương ứng với độ chính xác (Hình 37, 39) hoặc giá trị của hàm mục tiêu (Hình 38, 40), còn trục hoành tương ứng với số chu kỳ huấn luyện:

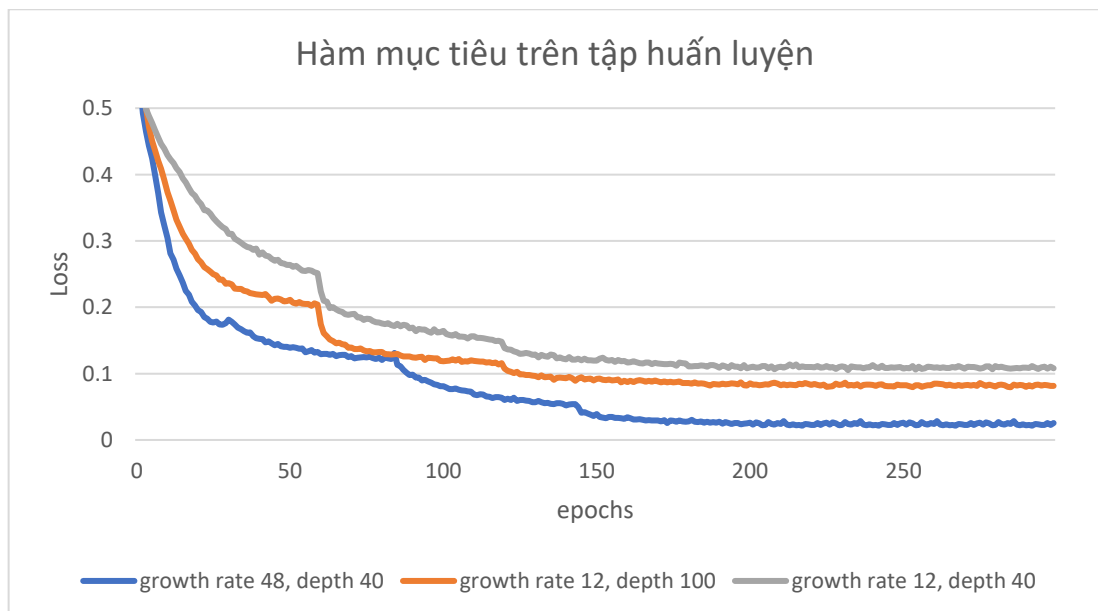
- Đường màu xanh lam đại diện cho mô hình có độ sâu 40 và tốc độ tăng trưởng bằng 48.
- Đường màu cam đại diện cho mô hình có độ sâu 100 và tốc độ tăng trưởng bằng 12.
- Đường màu xám đại diện cho mô hình có độ sâu 40 và tốc độ tăng trưởng bằng 12.



Hình 37 và hình 38 mô tả độ chính xác và giá trị hàm mục tiêu của các mô hình trong quá trình training.

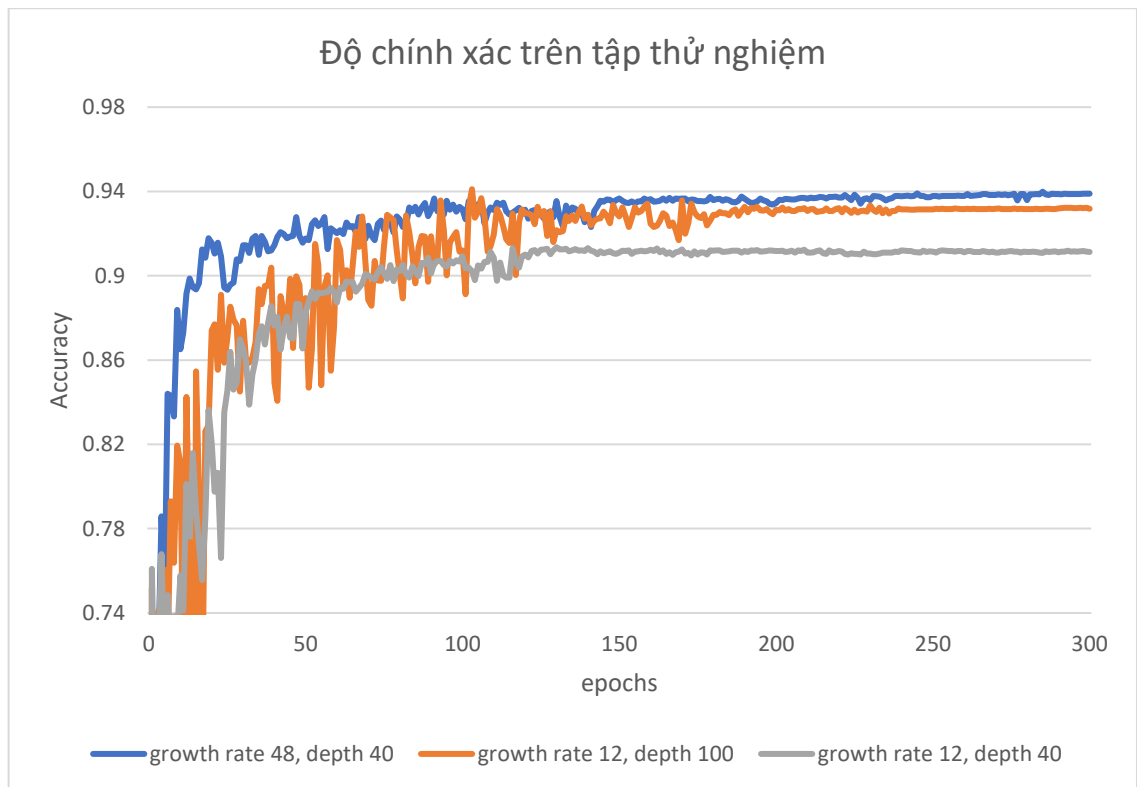


Hình 37: Độ chính xác trên tập training

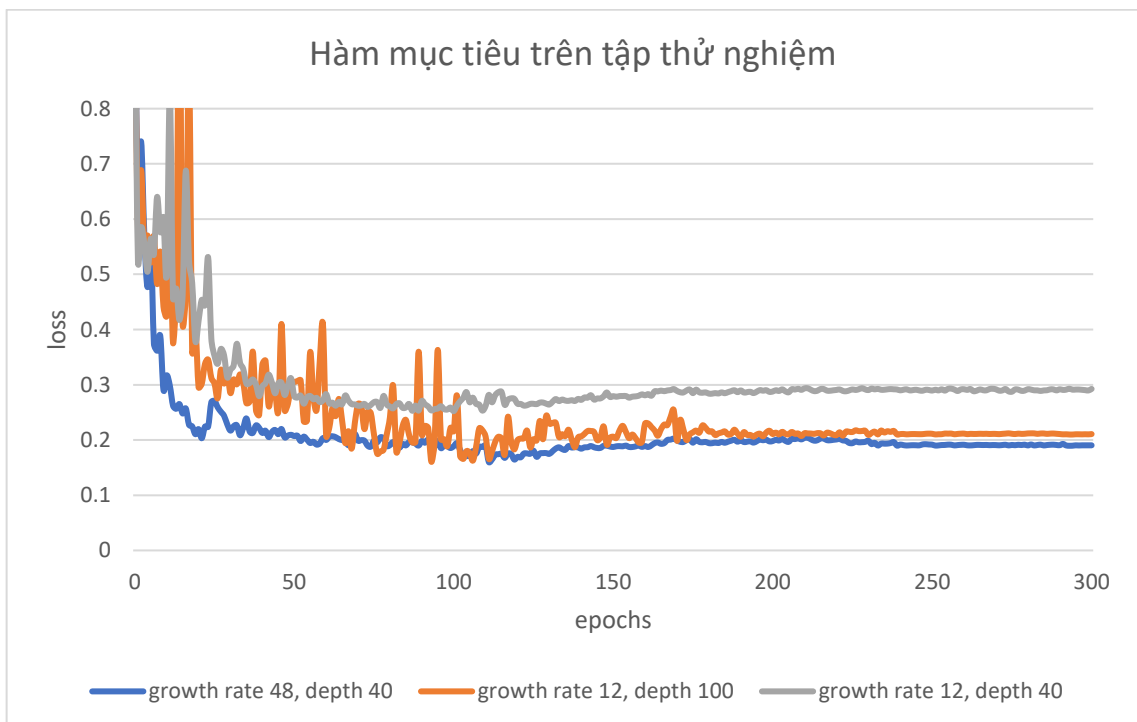


Hình 38: Hàm mục tiêu trên tập huấn luyện.

Hình 39 và hình 40 mô tả độ chính xác và hàm mục tiêu trong quá trình dự đoán trên bộ dữ liệu thử nghiệm đối với các mô hình.



*Hình 39: Độ chính xác trên tập thử nghiệm*



*Hình 40: Giá trị hàm mục tiêu trên tập thử nghiệm*

Bảng mô tả độ chính xác của từng mô hình:

Mô hình	Độ chính xác (trên tập test)	Thời gian thực hiện
Grow rate $k = 12$ Độ sâu của mạng 40	91.13%	29.5h
Grow rate $k = 12$ Độ sâu của mạng 100	93.18%	30h
Grow rate $k = 48$ Độ sâu của mạng 40	93.22%	35h

#### 4.4 Đánh giá

Kết quả trên biểu đồ và trong bảng cho thấy mô hình có hiệu năng cao trên bộ dữ liệu cả tập huấn luyện và tập thử nghiệm. Đối với hai mô hình cuối trong bảng đem lại hiệu năng gần xấp xỉ nhau. Qua đây có thể thấy rằng với tốc độ tăng trưởng (growth rate) nhỏ nhưng độ sâu của mạng lớn thì có thể tương đương với tốc độ tăng trưởng lớn nhưng giảm chiều sâu của mạng. Nếu càng tăng tốc độ tăng trưởng và độ sâu của mạng thì mô hình sẽ có hiệu năng tốt hơn nhưng thay vào đó thời gian huấn luyện sẽ cũng tăng theo.

## KẾT LUẬN

Trong quá trình học hỏi và tìm hiểu về Deeplearning nói chung và mạng DenseNet nói riêng em đã tích lũy, trau dồi thêm được nhiều kiến thức bổ ích. Qua đây, em cũng thấy được ý nghĩa to lớn và những ứng dụng quan trọng trong lĩnh vực thị giác máy tính...

- Về mặt lí thuyết, em đã tìm hiểu và trình bày được trong đồ án:
  - Các khái niệm cơ bản về học máy cũng như nền tảng mô hình mạng nơ-ron và các vấn đề khi huấn luyện một mạng nơ-ron sâu.
  - Kiến trúc mạng CNNs, cũng như mạng DenseNet các khái niệm liên quan và các nguyên lí hoạt động. Ngoài ra đồ án còn trình bày sơ lược về các kiến trúc hiện đại của mạng CNNs cũng như các kĩ thuật tăng độ chính xác của mô hình trong bài toán phân loại ảnh.
- Về mặt thực nghiệm, em đã cài đặt thành công mô hình mạng DenseNet trên bộ dữ liệu “*IMDB-WIKI – 500k+ face images with age and gender labels*”. Tuy nhiên, độ chính xác của mô hình chưa được như công bố trên các bài báo khoa học. Việc xác định các hyperparameter để tối ưu còn gặp nhiều khó khăn do thời gian học của một mô hình tương đối lâu (>20 tiếng) đối với bộ dữ liệu ban đầu.

Phương hướng phát triển: Trong bộ dữ liệu nói trên không chỉ có thông tin về giới tính mà còn có cả thông tin về độ tuổi. Vì vậy hướng tiếp theo sau khi hoàn thành được mô hình dự đoán giới tính, em sẽ tiếp tục nghiên cứu và tìm hiểu để ứng dụng mô hình đối với bài toán dự đoán lứa tuổi.

## TÀI LIỆU THAM KHẢO

- [1] Huang, Gao, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. CVPR 2017, pp. 4700-4708.
- [2] An overview of gradient descent optimization algorithms. URL: <http://ruder.io/optimizing-gradient-descent/>. Ngày truy cập: 15/12/2017.
- [3] CS231n: Convolutional Neural Networks for Visual Recognition. URL: <http://cs231n.stanford.edu/>. Ngày truy cập: 15/12/2017.
- [5] TensorFlow™ is an open source software library for numerical computation using data flow graphs. URL: <https://www.tensorflow.org/>. Ngày truy cập: 19/12/2017.
- [6] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
- [7] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- [8] Model and pre-trained parameters for VGG16 in TensorFlow. URL: <http://www.cs.toronto.edu/~frossard/post/vgg16/> Ngày truy cập: 14/12/2017
- [9] Notes on the Implementation of DenseNet in TensorFlow. URL: <https://medium.com/intuitionmachine/notes-on-the-implementation-densenet-in-tensorflow-beeda9dd1504>. Ngày truy cập: 16/12/2017
- [10] IMDB-WIKI – 500k+ face images with age and gender labels. URL: <https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/>. Ngày truy cập: 28/11/2017
- [11] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.
- [12] Convolutional Neural Network. URL: <https://tiendv.wordpress.com/2016/12/25/>. Ngày truy cập 20/11/2017.
- [13] Khóa học về Deep Learning trên Udacity. URL: <https://classroom.udacity.com/courses/ud730>. Ngày truy cập: 10/12/2017
- [14] Khóa học về Machine Learning. URL: <https://www.coursera.org/learn/machine-learning/>. Ngày truy cập: 15/12/2017
- [15] CS20Si: Tensorflow for Deep Learning Research. URL: <https://web.stanford.edu/class/cs20si/syllabus.html>. Ngày truy cập: 15/12/2017
- [16] Difference between Batch Gradient Descent and Stochastic Gradient Descent URL: <https://towardsdatascience.com/difference-between-batch-gradient-descent-and-stochastic-gradient-descent-1187f1291aa1>. Ngày truy cập: 09/12/2017

