ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KỲ

NHẬN DẠNG

PHÂN LOẠI HÌNH ẢNH QUẦN ÁO SỬ DỤNG MẠNG CNN

Giảng viên hướng dẫn:

Ths. ĐỖ VĂN TIẾN

Sinh viên thực hiện:

NGUYỄN THỊ PHƯƠNG HẢO 17520449

TP. Hồ Chí Minh, ngày 28/12/2019

Lời cảm ơn

Nhóm xin chân thành cảm ơn thầy Đỗ Văn Tiến đã dành nhiều thời gian giúp đỡ, tận tình hướng dẫn và giảng dạy cho chúng em suốt quá trình học tập. Cảm ơn thầy đã tạo điều kiện tốt cho chúng em và cung cấp đầy đủ kiến thức cần thiết cũng như giải quyết những khúc mắc để nhóm em có thể hoàn thành đồ án.

Qua quá trình học và thực hiện đồ án, nhóm em đã học hỏi được rất nhiều kiến thức bổ ích và thêm vào đó là những kinh nghiệm quý báu. Đó là những thứ sẽ giúp em rất nhiều trong công việc sau này.

Mặc dù đã cố gắng hoàn thành trong phạm vi và khả năng cho phép nhưng chắc chắn sẽ không tránh khỏi những sai sót. Chung em rất kính mong nhận được sự cảm thông và tận tình chỉ bảo của thầy.

Một lần nữa, xin chân thành cảm ơn thầy!

TP. HCM, ngày 28 tháng 12 năm 2019

Mục lục

I. GIỚI THIỆU VỀ CONVOLUTIONAL NEURON NETWORKS (CNNs) 5

1. Giới thiệu 5

2. Lớp tích chập (Convolutional Layers).....5

Quá trình tích chập (Convolutions) 6

Tham số.....8

3. Pooling.....9

Max Pooling.....9

II. OVERFITTING VÀ MỘT SỐ KỸ THUẬT GIẢI QUYẾT OVERFITTING ..11

1. Overfitting là gì?...... 11

2. Một số kỹ thuật xử lí overfitting.....11

a.Validation.....11

b.Image augmentation. 13

c.Dropout 13

III. CÁC BƯỚC TIẾN HÀNH 15

1. Thu thập dữ liệu.....15

a.Thư viện sử dụng: 15

b. Bộ dataset.....15

2. Xử lí dữ liệu 17

3. Xây dựng Model.....188

Thiết lập các layers 189

Biên dịch mô hình.....20

4. Huấn luyện và kiểm thử.....21

5. Đánh giá 23

IV. TRANSFER LEARNING 27

1. Định nghĩa 27

2. Mô hình VGG16.....27

3. Feature Exatractor.....29

4. Fine Tuning.....25

V. TÀI LIỆU THAM KHẢO…..…….33

I. GIỚI THIỆU VỀ CONVOLUTIONAL NEURON NETWORKS (CNNs)

1. Giới thiệu[1]

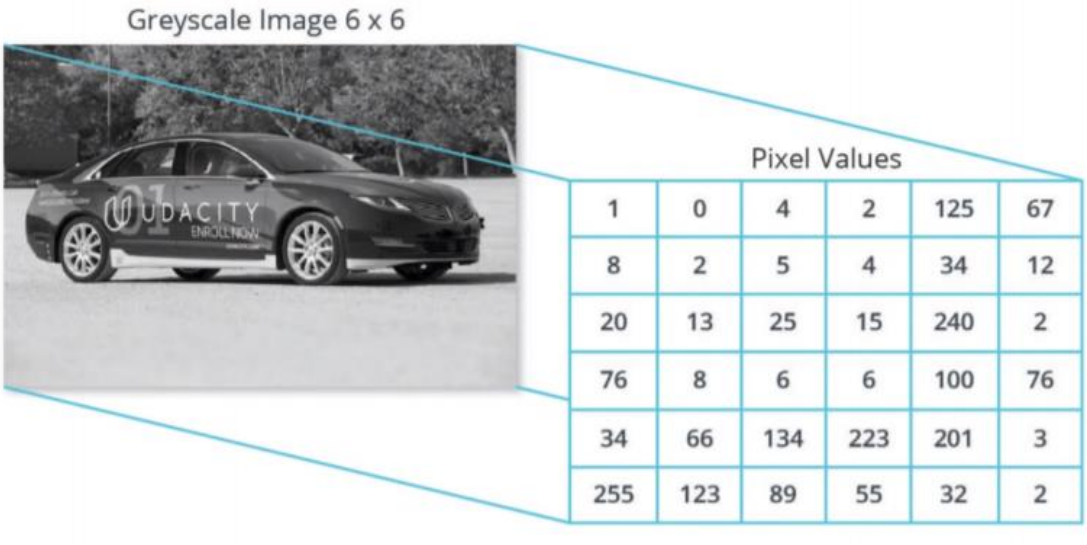
Mạng nơ-ron tích chập Convolutional Neural Networks - CNNs hay ConvNet) là mạng nơ-ron phổ biến nhất được dùng cho dữ liệu ảnh. Bên cạnh các lớp liên kết đầy đủ FC layers), CNN còn đi cùng với các lớp ẩn đặc biệt giúp phát hiện và trích xuất những đặc trưng - chi tiết (patterns) xuất hiện trong ảnh gọi là Lớp Tích chập Convolutional Layers). Chính những lớp tích chập này làm CNN trở nên khác biệt so với mạng nơ-ron truyền thống và hoạt động cực kỳ hiệu quả trong bài toán phân tích ảnh.

2. Lớp tích chập (Convolutional Layers)

Lớp tích chập được dùng để phát hiện và trích xuất đặc trưng - chi tiết của ảnh. Giống như các lớp ẩn khác, lớp tích chập lấy dữ liệu đầu vào, thực hiện các phép chuyển đổi để tạo ra dữ liệu đầu vào cho lớp kế tiếp (đầu ra của lớp này là đầu vào của lớp sau). Phép biến đổi được sử dụng là phép tính tích chập. Mỗi lớp tích chập chứa một hoặc nhiều bộ lọc - bộ phát hiện đặc trưng (filter - feature detector) cho phép phát hiện và trích xuất những đặc trưng khác nhau của ảnh. Bộ lọc ở lớp tích chập càng sâu thì phát hiện các đặc trừng càng phức tạp. Độ phức tạp của đặc trưng được phát hiện bởi bộ lọc tỉ lệ thuận với độ sâu của lớp tích chập mà nó thuộc về. Trong mạng CNN, những lớp tích chập đầu tiên sử dụng bộ lọc hình học (geometric filters) để phát hiện những đặc trưng đơn giản như cạnh ngang, dọc, chéo của bức ảnh. Những lớp tích chập sau đó được dùng để phát hiện đối tượng nhỏ, bán hoàn chỉnh như mắt, mũi, tóc, v.v. Những lớp tích chập sâu nhất dùng để phát hiện đối tượng hoàn hỉnh như: chó, mèo, chim, ô tô, đèn giao thông, v.v.

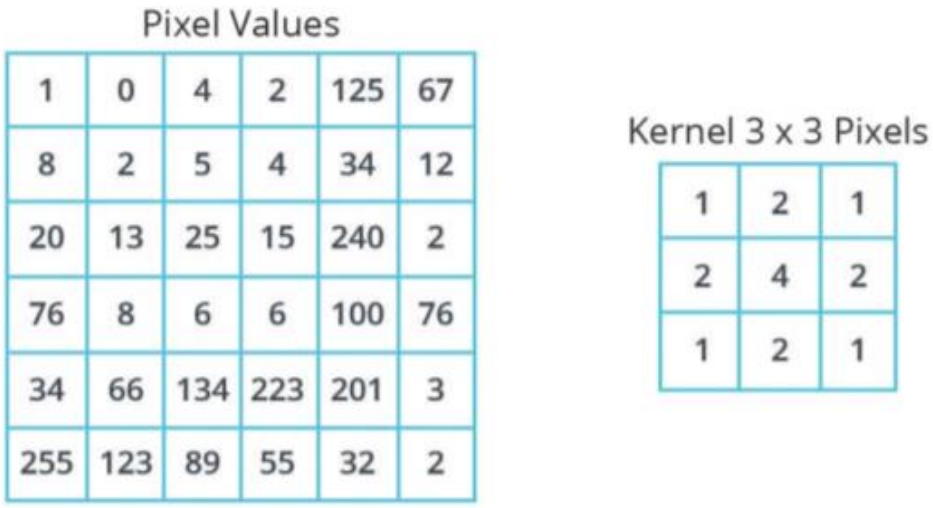
Quá trình tích chập (Convolutions)

Lấy một hình ảnh gray-scale như bên dưới làm ví dụ. Để đơn giản, chúng ta giả sử rằng hình ảnh chỉ có 6 pixels theo chiều cao và 6 pixels theo chiều dọc. Máy tính sẽ chuyển ảnh này sang một mảng 2 chiều. Bởi vì đây là ảnh gray-scale cho nên mỗi pixel sẽ có giá trị từ 0 đến 255. Hình 2.1 là một ví dụ.



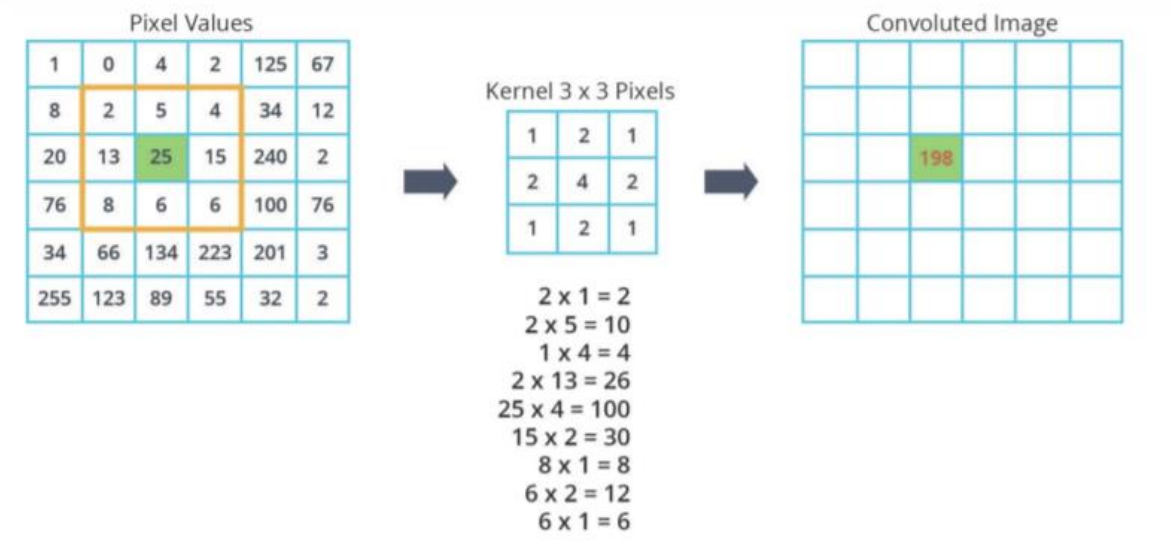
Hình 2.1.

Khi đưa vào neural network, các giá trị này sẽ được chuẩn hoá về các giá trị từ 0 đến 1. Tuy nhiên, ở đây chúng ta sẽ giữ nguyên giá trị gốc để dễ giải thích. Ý tưởng của convolutional layer là tạo một mảng khác được gọi là kernel hoặc filter, kernel này dùng để lọc ra các đặc trưng cần thiết của ảnh. Trong ví dụ này là một ma trận 3x3



Hình 2.2.

Sau đó, chúng ta có thể scan kernel này qua toàn bộ ảnh. Một convolutional layer sẽ áp kernel này lên các vùng của ảnh input Hình 2.3.).

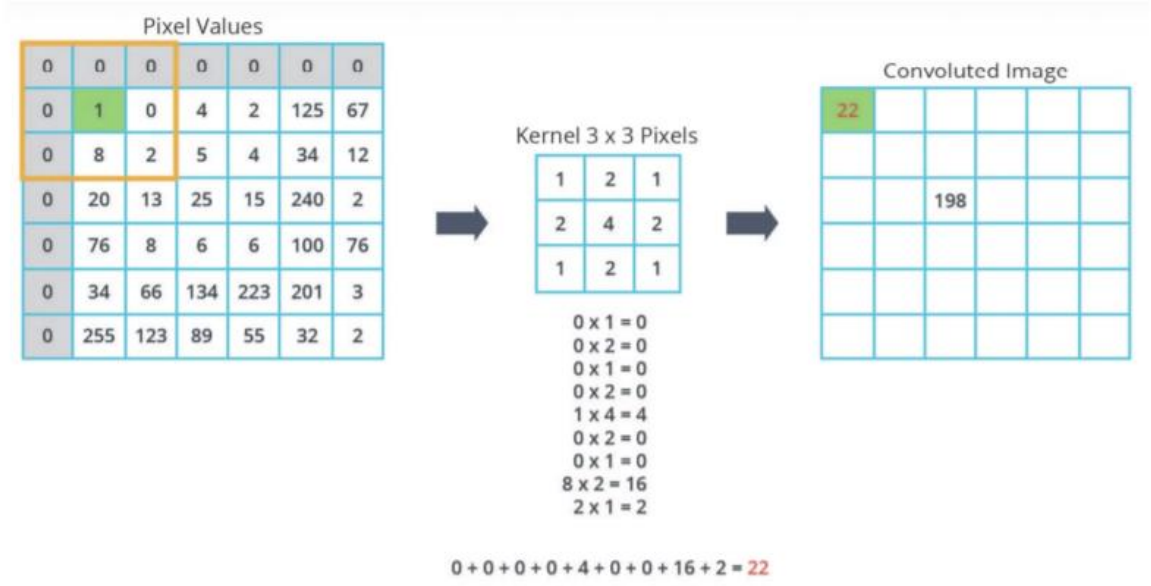


Hình 2.3.

Tuy nhiên, chuyện gì sẽ xảy ra với các pixel ở cạnh?

Khi chúng ta đặt kernel chính giữa các pixel này thì kernel sẽ vượt ra ngoài ảnh. Trong trường hợp này, chúng ta có 2 sự lựa chọn. Một là không sử dụng các pixel

này trong convolution, vì vậy chỉ cần bỏ qua chúng và coi như chúng không tồn tại. Tuy nhiên, việc này sẽ làm giảm kích thước của hình ảnh ban đầu. Điều này đồng nghĩa với việc thông tin sẽ bị mất mát. Hai là zero-padding, trong trường hợp này, chúng ta sẽ thêm các pixel 0 xung quanh ảnh gốc như hình dưới sau đó tính toán bình thường như trên. Với cách này toàn bộ ảnh sẽ không bị mất thông tin.



Hình 2.4.

Như vậy ta thấy rằng một convolution chỉ là một quá trình apply một kernel hoặc filter lên các vùng khác nhau của ảnh input. Và tương tự như Dense layer, chúng ta thấy rằng convolution là một kiểu layer khác mà chúng ta sẽ sử dụng.

Tham số

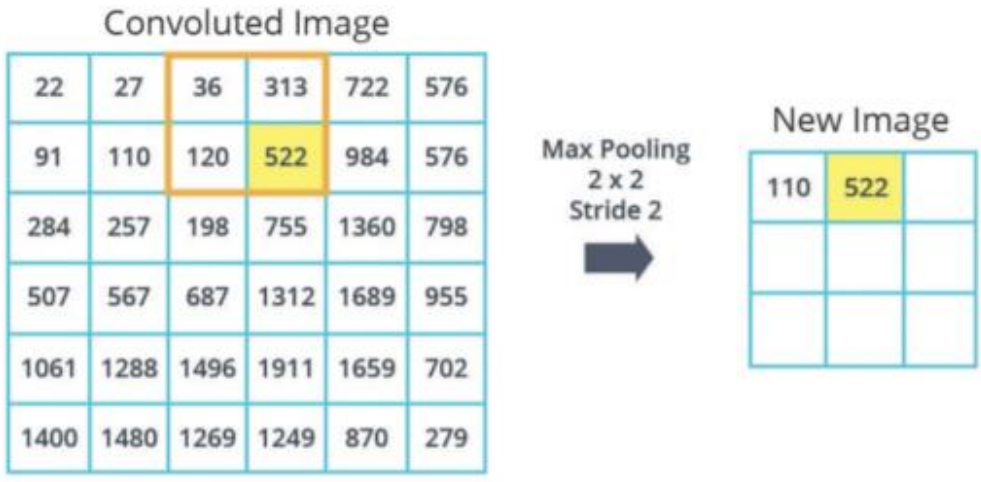
Ở ví dụ trên, chúng ta sử dụng một kernel 3 x 3 để tích chập với ảnh đầu vào. Các giá trị của kernel này sẽ được học tự động trong quá trình training. Vì vậy với mỗi convolution layer, chúng ta có một số lượng cố định các tham số. Trong trường hợp này, chúng ta có 3 * 3 = 9 tham số, cộng thêm với bias nữa chúng ta sẽ có 10 tham số. Tuy nhiên trong thực tế, người ta có thể áp nhiều kernel cùng lúc lên ảnh đầu vào. Kết quả sẽ tạo ra nhiều output. Mỗi kernel đảm nhận việc trích xuất các

đặc trưng khác nhau. Ví dụ, Nếu ta sử dụng 10 kernel lên ảnh input ở ví dụ trên, chúng ta sẽ có (3 * 3 + 1) * 10 = 100 tham số

Note: Số lượng tham số không phụ thuộc vào ảnh đầu vào. Đây là một tính chất quan trọng của lớp tích chập giúp CNN tránh được rủi ro overfitting (sẽ giới thiệu ở buổi sau) do có quá nhiều tham số nếu sử dụng lớp liên kết đầy đủ.

3. Pooling

Pooling là một quá trình xử lý làm giảm kích thước của một ảnh input bằng các vùng chọn. Hai loại Pooling phổ biến nhất bao gồm Max Pooling và Average Pooling. Hãy xem ví dụ bên dưới cho max pooling.



Hình 3.1. Ví dụ cho max pooling.

Max Pooling

Có pool size và stride. Như ví dụ trên sử dụng size là 22 pixel grid và stride là 2. Nhìn vào vùng màu cam ở trong hình, chúng ta sẽ chọn một pixel với giá trị cao nhất. Tham số thứ hai mà chúng ta sử dụng là stride. Tham số này xác định số lượng pixel để lướt vùng màu cam dọc theo toàn bộ hình ảnh. Trong trường hợp này, vùng màu cam sẽ được slide sang phải 2 pixels. Quá trình cứ tiếp tục từ trái sang phải, trên xuống dưới cho đến khi hoàn tất một ảnh mới. Ta thu được ảnh mới

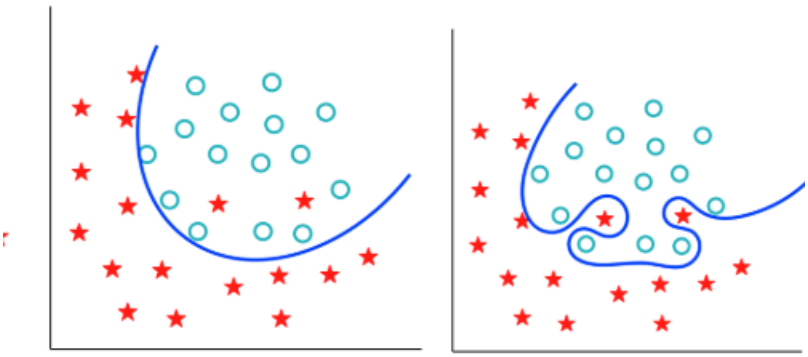
có kích thước nhỏ hơn ảnh gốc, chúng ta có thể gọi là downsampled ảnh gốc. Kích thước của ảnh mới sẽ phụ thuộc vào size và stride mà bạn chọn. Average Pooling cũng tương tự như Max Pooling, tuy nhiên thay vì chọn giá trị cao thất thì nó lại tính trung bình của tất cả và lưu giá trị đó vào mảng mới

II. GIỚI THIỆU OVERFITTING VÀ MỘT SỐ KỸ THUẬT GIẢI QUYẾT OVERFITTING

1. Overfitting là gì

Overfitting là hiện tượng mô hình dự đoán quá khớp với tập training dẫn đến không hiệu quả với tập testing.

Nguyên nhân có thể do ta chưa đủ dữ liệu để đánh giá hoặc do mô hình của ta quá phức tạp. Mô hình bị quá phức tạp khi mà mô hình của ta sử dụng cả những nhiễu lớn trong tập dữ liệu để học, dẫn tới mất tính tổng quát của mô hình.



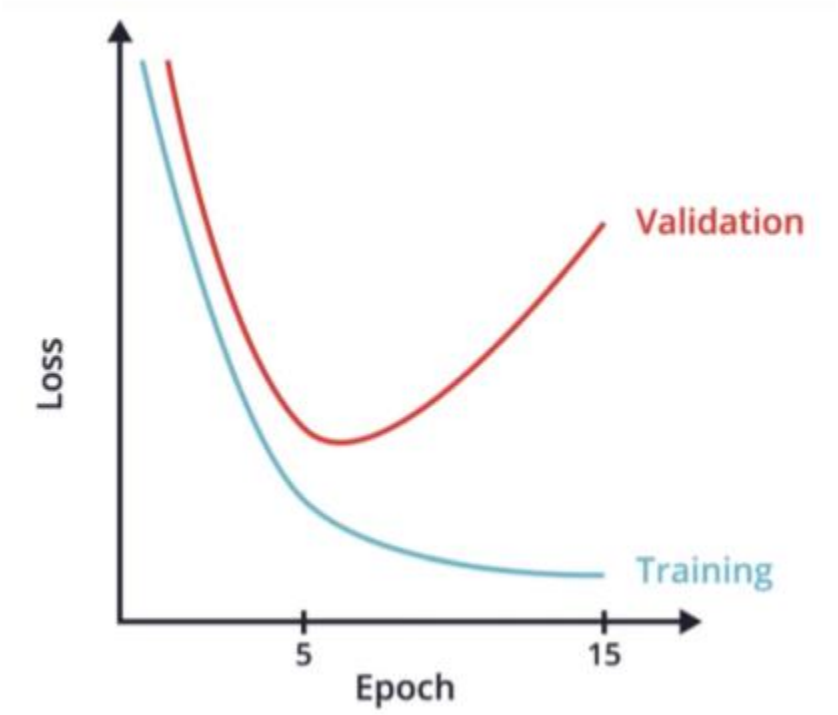
2. Một số kỹ thuật giải quyết overfitting

a. Validation [2]

Trong suốt quá trình training, neural network sẽ chỉ thấy training dataset để đưa ra quyết định thay đổi weights và biases như thế nào. Sau mỗi training epoch, chúng ta kiểm tra model đang làm việc như thế nào bằng cách nhìn vào traning loss

và validation loss (loss trên validation dataset). Một điều quan trọng cần chú ý là model không sử dụng bất kì phần nào của tập validation để điều chỉnh weights và biases. Bước validation chỉ nói cho chúng ta biết liệu model có làm việc tốt trên tập validation hay không

Bởi vì neural network không sử dụng tập validation để điều chỉnh weights và biases, cho nên chúng ta sẽ biết được liệu model có đang bị overfitting trên tập training hay không. Xem xét biểu đồ bên dưới



Chúng ta thấy rằng, sau vài epoch đầu thì validation loss bắt đầu tăng lên trog khi training loss vẫn đang trên đà giảm dần. Và khi kết thúc training, validation loss rất cao còn training loss rất nhỏ. Vấn đề này nói lên neural network đang bị overfitting tập training. Lý do là bởi vì model của chúng ta không khái quát đủ tốt để làm việc tốt trên validation dataset mà chỉ cố gắng fitting các weight và bias vào tập training. Do vậy tập validation ở đây có mục đích để chúng ta có thể xác định được số lượng epoch mà chúng ta nên train cho CNN. Validation cũng

có thể được sử dụng để chọn một model tốt nhất trong các model mà chúng ta đang thử nghiệm. Ví dụ, khi muốn thay đổi số lượng layer trong mạng neural, chúng ta có thể tạo ra nhiều biến thể của model với các kiến trúc khác nhau, sau đó so sánh chúng bằng cách sử dụng validation set. Kiến trúc nào có loss nhỏ nhất sẽ được chọn làm model tốt nhất.

b. Image augmentation [3]

Khi training một CNN để nhận diện các đối tượng cụ thể trong hình ảnh, bạn muốn CNN có thể xác định được những object đó với bất cứ kích thước và vị trí nào trong ảnh.

Nếu may mắn có lượng data lớn với nhiều mẫu khác nhau, thì model sẽ chạy tốt và ít bị overfit. Tuy nhiên, trong trường hợp chúng ta làm việc với tập training không có nhiều mẫu khác nhau, lúc đó CNN sẽ bị overfitting và sẽ không khái quát hoá data mà nó chưa biết trước. Vấn đề này có thể tránh được bằng phương pháp Image Augmention. Image Augmention là phương pháp để tạo ra nhiều ảnh training mới từ ảnh gốc bằng cách ứng dụng các phương pháp biến đổi hình ảnh như rotation, flipping, zoom,... Bằng cách này chúng ta sẽ tạo ra được nhiều data hơn. Từ đó CNN sẽ khái quát tốt hơn lượng data chưa biết trước và sẽ tránh được overfit.

3. Dropout [4]

Dropout là một cách khác để tránh overfitting khi training CNN. Dropout sẽ bỏ ngẫu nhiên vài neuron trong suốt quá trình training bao gồm quá trình feedforwark và backpropagation.

Bằng cách này chúng ta có thể tránh được overfit. Lý do là bởi vì nó không dựa vào tất cả neuron ở trên để giải quyết bài toán mà những neuron khác sẽ làm việc để hoàn thành công việc. Trong thực tế chúng ta có thể mô tả một xác suất để mỗi neuron có bị drop hay không trong mỗi training epoch. Vì chúng ta dùng xác suất nên sẽ có trường hợp chúng bị drop tất cả hoặc không có neuron nào bị drop cả.

Tuy nhiên, chúng ta cũng không cần quá lo lắng vì vấn đề này bởi vì chúng ta training nhiều epoch mà

III. CÁC BƯỚC TIẾN HÀNH

1. Thu thập dữ liệu

a. Thư viện sử dụng:

tensorflow [5]

sklearn [6]

numpy [7]

pandas [8]

matplotlib[9]

b. Bộ dataset

Fashion NMIST [10]

Tập dữ liệu này được thiết kế dưới tư tưởng của tập dữ liệu MNIST chứa khoảng 70,000 ảnh đen trắng phân thành 10 loại:bao gồm một tập huấn gồm 60.000 ví dụ và một bộ thử nghiệm gồm 10.000 ví dụ. Mỗi một ảnh là một loại quần áo hoặc giày dép với độ phân giải thấp (28 by 28 pixel), như hình minh hoạ bên dưới:

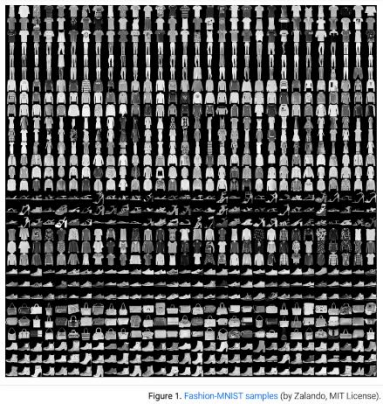


Figure 1. Fashion-MNIST samples (by Zalando, MIT License).

Fshion MNIST

Label của các hình ảnh sẽ thuộc 1 trong 10 nhãn sau:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Để sử dụng tập dữ liệu này thì ta tiến hành dowload nó từ kaggle theo link trên:

```
!kaggle datasets download --force -d 'zalando-research/fashionmnist'
```

Sau đó tiến hành giải nén :

```
# unzip dataset
!unzip 'fashionmnist.zip' -d 'my_data'
```

```
Archive:  fashionmnist.zip
  inflating: my_data/fashion-mnist_test.csv
  inflating: my_data/fashion-mnist_train.csv
  inflating: my_data/t10k-images-idx3-ubyte
  inflating: my_data/t10k-labels-idx1-ubyte
  inflating: my_data/train-images-idx3-ubyte
  inflating: my_data/train-labels-idx1-ubyte
```

Sau khi giải nén ta sẽ được:

2 file: test.csv và train.csv tương đương với 2 tập training set và test set

4 mảng numpy:

train_image:train-images-idx3-ubyte:hình ảnh tập huấn luyện (9912422 byte)

train-label:train-labels-idx1-ubyte:nhãn tập huấn luyện (28881 byte)

test_image: t10k-images-idx3-ubyte:hình ảnh thiết lập thử nghiệm (1648877 byte)

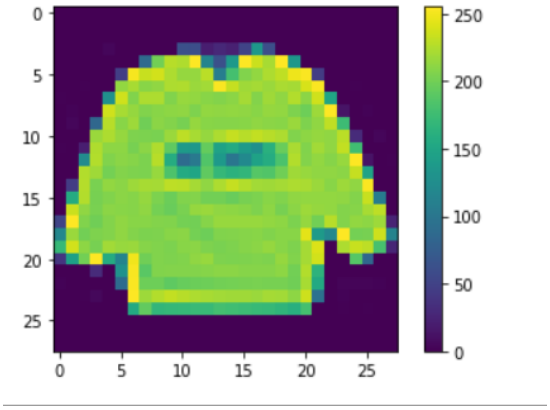
test-label:t10k-labels-idx1-ubyte:nhãn bộ kiểm tra (4542 byte)

Mỗi ảnh là một mảng NumPy 2 chiều, 28x28, với mỗi pixel có giá trị từ 0 đến 255. *Nhãn* là một mảng của các số nguyên từ 0 đến 9, tương ứng với mỗi lớp trong 10 lớp đã nói trên:

```
[14] class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                    'Sandal',      'Shirt',   'Sneaker',  'Bag',   'Ankle boot']
```

2. Xử lí dữ liệu

Phân tích ảnh đầu tiên trong tập dữ liệu, chúng ta sẽ thấy các pixel có giá trị từ 0 đến 255:



Giá trị của mỗi pixel trong dữ liệu ảnh là một số nguyên trong phạm vi [0,255]. Để mô hình hoạt động chính xác, các giá trị này cần được chuẩn hóa thành phạm vi [0,1].

x = (x - min) / (max - min) ; min=0 and max=255

Lưu ý: tiền xử lý này phải được áp dụng đồng thời cho cả *tập huấn luyện* và *tập kiểm thử*:

```
train_dataset_x = train_dataset[:,1:] / 255
train_dataset_x = train_dataset_x.reshape((num_train_examples, 28, 28, 1))
train_dataset_y = train_dataset[:,0]


test_dataset_x = test_dataset[:,1:] / 255
test_dataset_x = test_dataset_x.reshape((num_test_examples, 28, 28, 1))
test_dataset_y = test_dataset[:,0]
```

Vì khi đọc từ file csv vào thì mỗi hình ảnh sẽ có dạng vector 1 chiều mỗi hình chứa 784 pixel.

Trước khi đưa vào model CNN, phải định hình lại dữ liệu thành ma trận 3 chiều (28x28x1).

Nếu đây là hình ảnh RGB, sẽ có 3 kênh, nhưng vì MNIST có thang màu xám nên nó chỉ sử dụng một kênh.

Các nhãn được đưa ra dưới dạng số nguyên trong khoảng 0-9. Chúng ta cần one-hot encode

ví dụ: 8 được biểu diễn thành [0, 0, 0, 0, 0, 0, 0, 0, 1, 0].

```
train_Y_one_hot = to_categorical(train_dataset_y)
test_Y_one_hot = to_categorical(test_dataset_y)
```

Chia tập train và validation :

Tập train được sử dụng để Huấn luyện mô hình

Tập validation được sử dụng để Đánh giá Hiệu suất của Mô hình

Tỉ lệ: 80:20

3. Xây dựng Model

Thiết lập các layers

```
[81] model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,
                               input_shape=(28, 28, 1)),
        tf.keras.layers.MaxPooling2D((2, 2), strides=2),
        tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),
        tf.keras.layers.MaxPooling2D((2, 2), strides=2),
        tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation=tf.nn.relu),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),
        tf.keras.layers.Dense(10,  activation=tf.nn.softmax)

    ])
```

```
model.summary();
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 7, 7, 128)         73856
_________________________________________________________________
flatten_6 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_12 (Dense)             (None, 128)               802944
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1290
=================================================================
Total params: 896,906
Trainable params: 896,906
Non-trainable params: 0
_________________________________________________________________
```

layers.Conv2D lớp tích chập để phát hiện và trích xuất đặc trưng-chi tiết ảnh layers.

MaxPooling2D: giảm kích thước của ảnh input. Max Pooling lấy giá trị tối đa trong bộ lọc tích chập

Trong mạng neuron trên, lớp đầu tiên, tf.keras.layers.Flatten, chuyển đổi định dạng của hình ảnh từ mảng hai chiều (28x28) thành mảng một chiều (28x28 = 784). Tưởng tương công việc của layer này là cắt từng dòng của anh, và ghép nối lại thành một dòng duy nhất nhưng dài gấp 28 lần. Lớp này không có trọng số để học, nó chỉ định dạng lại dữ liệu.

layer tf.keras.layers.Dense(fully connected layer). Đây là các layer neuron được kết nối hoàn toàn (mỗi một neuron của layer này kết nối đến tất cả các neuron của lớp trước và sau nó). Layer Dense đầu tiên có 128 nút (hoặc neuron).

Layer cuối cùng là lớp *softmax* có 10 nút, với mỗi nút tương đương với điểm xác suất, và tổng các giá trị của 10 nút này là 1 (tương đương 100%). Mỗi nút chứa một giá trị cho biết xác suất hình ảnh hiện tại thuộc về một trong 10 lớp.

Biên dịch mô hình [11]

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Loss: Hàm thiệt hại — dùng để đo lường mức độ chính xác của mô hình trong quá trình huấn luyện. Chúng ta cần giảm thiểu giá trị của hàm này để "điều khiển" mô hình đi đúng hướng (thiệt hại càng ít, chính xác càng cao).

Trình tối ưu hoá — Đây là cách mô hình được cập nhật dựa trên dữ liệu huấn luyện được cung cấp và hàm thiệt hại.

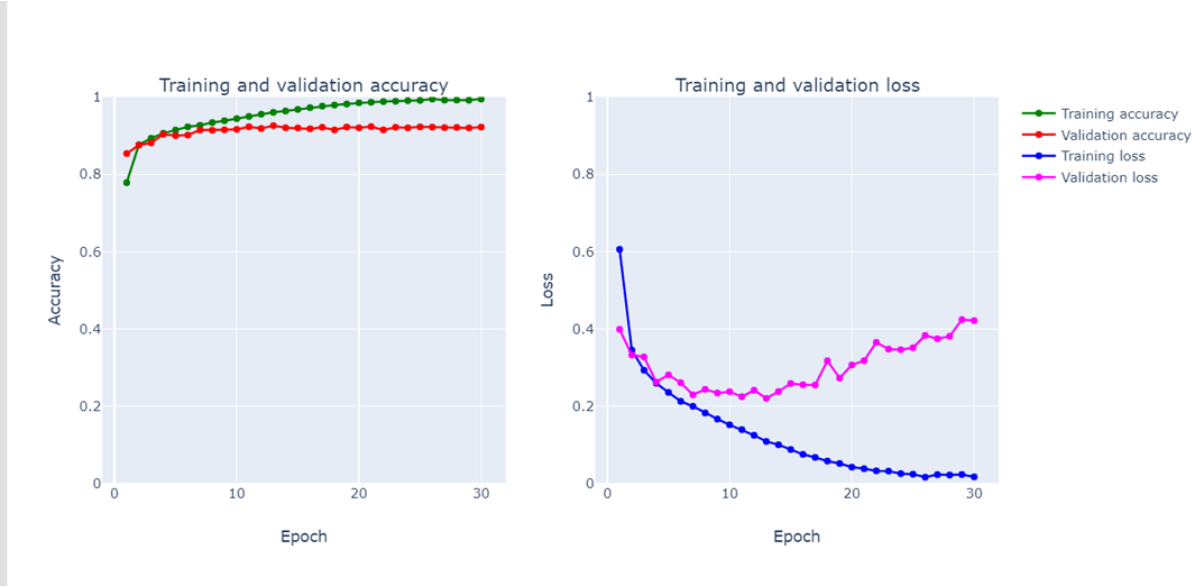
vì mục tiêu (các biến đầu ra) dưới dạng one-hot nên dùng 'categorical_crossentropy'

còn nếu dưới dạng vector nhị phân thì dùng categorical_crossentropy

Metric — dùng để theo dõi các bước huấn luyện và kiểm thử. Ví dụ sau dùng *accuracy*, tỉ lệ ảnh được phân loại chính xác.

4. Huấn luyện và kiểm thử

Sau khi train thử với 30 epoch:



Nhìn vào biểu đồ trên ta thấy rằng accuracy tăng theo thời gian và tiệm cận 100%, tuy nhiên validation accuracy chỉ dừng lại ở khoảng trên 85%

Đồng thời trong khi loss của tập train ngày càng giảm thì loss của validation tăng mạnh

Như vậy model có vẻ như bị overffiting. Để tránh hiện tượng overfitting ta sẽ thử tăng cường data xem sao.

Data Augment

Hàm sinh dữ liệu giúp cho chúng ta sinh ra trực tiếp dữ liệu để fit vào mô hình trong từng batch training

Việc tận dụng hàm sinh giúp ích rất nhiều trong quá trình training các dữ liệu lớn. Vì không phải lúc nào tập dữ liệu cũng cần phải load hết vào RAM gây lãng phí bộ nhớ, hơn nữa nếu như tập dữ liệu quá lớn thì có thể dẫn đên tràn bộ nhớ và thời gian tiền xử lý dữ liệu đầu vào sẽ lâu hơn.

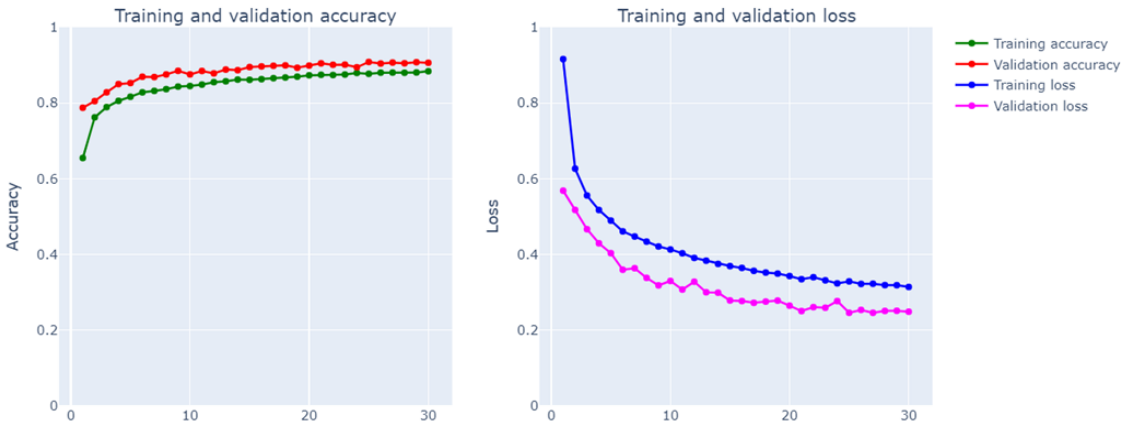
```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        rotation_range = 8,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        shear_range = 0.3,# shear angle in counter-clockwise direction in degrees
        width_shift_range=0.08,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.08,  # randomly shift images vertically (fraction of total height)
        vertical_flip=True)  # randomly flip images
batches = datagen.flow(X_train, y_train, batch_size=256)
```

Tuy nhiên vì dữ liệu được generate ra từ một ảnh gốc do đó ảnh sẽ không tự nhiên và vẫn sẽ có một tương quan nhất lớn nào đó, và dữ liệu đó không phải là dữ liệu mới. Do đó việc này cũng chưa thể tránh được hiện tượng overfitting.
Nên để tăng độ chính xác chúng ta cần thêm kỹ thuật Drop-out vào mô hình

```
model_2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,
                           input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10,  activation=tf.nn.softmax)

])
```

Sau đó , chúng ta train lại thì được kq sau:



=> giờ đây đã hết overfitting. Chúng ta tiến hành đánh giá mô hình.

5. Đánh giá

```
[131] score = model_2.evaluate(test_dataset_x, test_Y_one_hot)
      print('Test loss:', score[0])
      print('Test accuracy:', score[1])
```

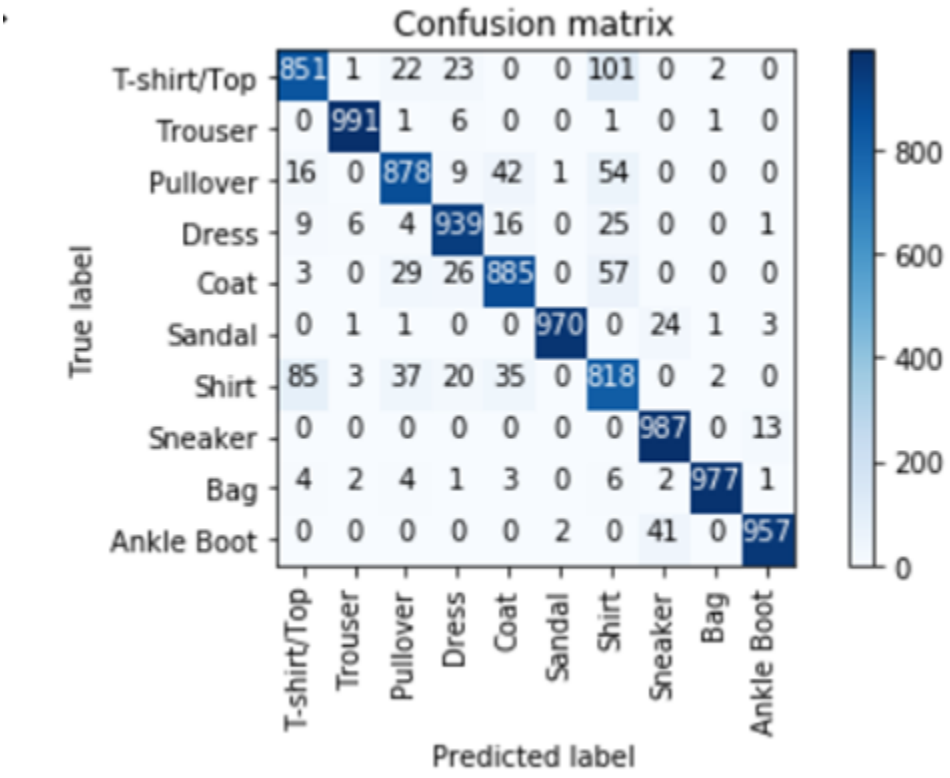
```
10000/10000 [==============================] - 6s 565us/sample - loss: 0.2373 - acc: 0.9151
Test loss: 0.2372579971909523
Test accuracy: 0.9151
```

Như vậy, mô hình đánh giá trên tập test với accuracy =0.9151(xấp xỉ 92%)

Với một bài toán phần lớp thì có nhiều cách đánh giá khác nhau. Riêng đối với accuracy (tức độ chính xác) thì công thức chỉ đơn giản là lấy số điểm dữ liệu dự đoán đúng chia cho tổng số dữ liệu. Điều này nghe qua thì có vẻ hợp lý nhưng trên thực tế đối với những bài toán dữ liệu bị mất cân bằng thì đại lượng này chưa đủ ý nghĩa. Giả sử chúng ta đang xây dựng mô hình dự đoán tấn công mạng (giả sử các request tấn công chiếm khoảng 1/100000 số lượng request).

Nếu mô hình dự đoán tất cả các request đều là bình thường thì độ chính xác cũng lên đến **99.9999%** và con số này thường không thể tin tưởng được trong mô hình phân lớp. Cách tính accuracy ở trên thường chỉ cho chúng ta biết được có bao nhiêu phần trăm dữ liệu được dự đoán đúng chứ chưa chỉ ra được mỗi class được phân loại cụ thể như thế nào. Thay vào đó chúng ta có

thể sử dụng **Confusion matrix**. Về cơ bản, confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class.



	precision	recall	f1-score	support
Class 0	0.90	0.84	0.87	1000
Class 1	0.98	0.99	0.99	1000
Class 2	0.92	0.83	0.87	1000
Class 3	0.88	0.96	0.92	1000
Class 4	0.89	0.83	0.86	1000
Class 5	0.99	0.97	0.98	1000
Class 6	0.71	0.79	0.75	1000
Class 7	0.93	0.98	0.96	1000
Class 8	0.98	0.99	0.99	1000
Class 9	0.98	0.96	0.97	1000
accuracy			0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

IV. TRANSFER LEARNING

1. Định nghĩa [13]

Tranfer learning là một phương pháp học máy mà trong đó, một mô hình đã được phát triển cho một task (pretrained network) được tái sử dụng ở một task khác. Phương pháp này mang đến một cách tiếp cận phổ biến và hiệu quả cao trong deep learning khi bạn có một tập dữ liệu vừa và nhỏ.
Pretrained network là một mạng đã lưu trước đó, được đào tạo trên một tập dữ liệu lớn (thường là trên một task phân loại ảnh quy mô lớn). Một số mạng pretrained thường được sử dụng như: VGG, ResNet, Inception, Inception-ResNet, Xception...

Có 2 cách để sử dụng một pretrained network:

- **Feature extractor**: Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pre-trained model, thì ta sẽ dùng linear classifier (linear SVM, softmax classifier,..) để phân loại ảnh. Hiểu đơn giản thì các đặc điểm ảnh (tai, mũi, tóc,…) giờ như input của bài toán linear regression hay logistic regression.
- **Fine tuning**: Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pre-trained model, thì ta sẽ coi đây là input của 1 CNN mới bằng cách thêm các ConvNet và Fully Connected layer.

2. Mô hình VGG16 [14]

VGG16 là mạng convolutional neural network được đề xuất bởi K. Simonyan and A. Zisserman, University of Oxford. Model sau khi train bởi mạng VGG16 đạt độ chính xác 92.7% top-5 test trong dữ liệu ImageNet gồm 14 triệu hình ảnh thuộc 1000 lớp khác nhau. Giờ áp dụng kiến thức ở trên để phân tích mạng VGG 16

Architect của VGG16 bao gồm 16 layer :13 layer Conv (2 layer conv-conv,3 layer conv-conv-conv) đều có kernel 3x3, sau mỗi layer conv là maxpooling downsize xuống 0.5, và 3 layer fully connection.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

3. Feature Extractor

Bước 1: Load model VGG16 của ImageNet dataset

```
conv_base=tf.keras.applications.VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH)
                 )
```

include_top=False bỏ phần fully connected layer ở cuối

Bước 2: Dùng pre-trained model để lấy ra feature của ảnh

```
train_features = conv_base.predict(np.array(train_X), batch_size=BATCH_SIZE, verbose=1)
test_features = conv_base.predict(np.array(test_X), batch_size=BATCH_SIZE, verbose=1)

print(train_features.shape, "\n",  test_features.shape, "\n")
```

```
60000/60000 [==============================] - 1433s 24ms/sample
10000/10000 [==============================] - 236s 24ms/sample
(60000, 1, 1, 512)
 (10000, 1, 1, 512)
```

Bước 3: Đưa vào bộ phân lớp

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC

clf = OneVsRestClassifier(LinearSVC())
model=clf.fit(svm_train, svm_y)
```

B4: Kết quả

	precision	recall	f1-score	support
0	0.80	0.82	0.81	1000
1	0.99	0.98	0.98	1000
2	0.81	0.81	0.81	1000
3	0.85	0.89	0.87	1000
4	0.78	0.83	0.80	1000
5	0.96	0.95	0.96	1000
6	0.69	0.59	0.64	1000
7	0.92	0.94	0.93	1000
8	0.97	0.97	0.97	1000
9	0.96	0.95	0.95	1000
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

4. Fine Tuning

Bước1: Load model VGG16 của ImageNet dataset

```
base_model=tf.keras.applications.VGG16(weights='imagenet',
                 include_top=False,
                 input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH)
                )

for layer in base_model.layers:
    layer.trainable = False
```

include_top=False bỏ phần fully connected layer ở cuối

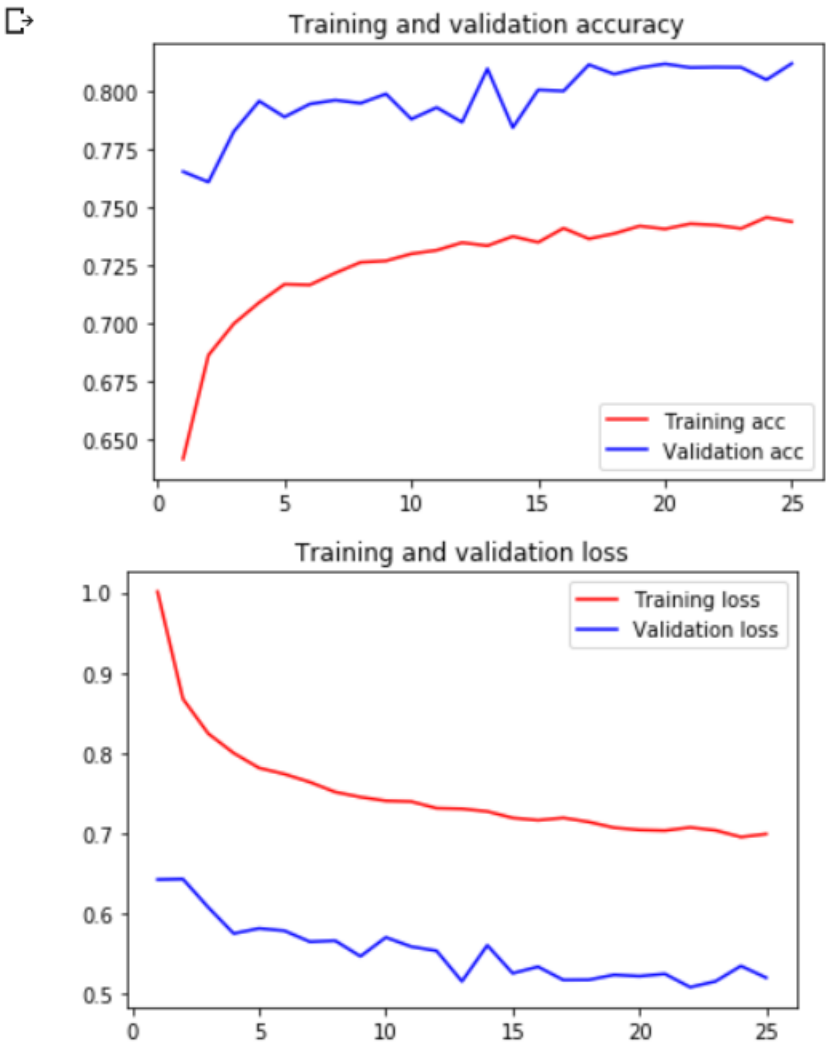
layer.trainable: Đóng băng các layer trong pre-trained model

Bước 2: Khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pre-trained model, thì ta sẽ coi đây là input của 1 CNN mới bằng cách thêm các ConvNet và Fully Connected layer

```
last_layer = base_model.get_layer('block5_pool')
x=last_layer.output
x=layers.Flatten()(x)
x= layers.Dropout(0.4)(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dense(10, activation='softmax')(x)

model_3 = models.Model(inputs=base_model.input, outputs=x)
model_3.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
history= model_3.fit_generator(aug_train.flow(Xtrain,ytrain, batch_size=32),
                               validation_data=(aug_val.flow(valid_X, valid_label,batch_size=32)),epochs=25,verbose=1)
```

Bước 3: Kết quả



	precision	recall	f1-score	support
0	0.43	0.80	0.56	1000
1	0.98	0.84	0.91	1000
2	0.51	0.76	0.61	1000
3	0.49	0.70	0.58	1000
4	0.54	0.52	0.53	1000
5	0.57	0.18	0.28	1000
6	0.15	0.17	0.16	1000
7	0.91	0.10	0.17	1000
8	0.63	0.65	0.64	1000
9	0.82	0.68	0.75	1000
accuracy			0.54	10000
macro avg	0.60	0.54	0.52	10000
weighted avg	0.60	0.54	0.52	10000

IV TÀI LIỆU THAM KHẢO

[1] Mạng nơ-ron tích chập - Convolutional Neural Network (CNN)

https://dlapplications.github.io/2018-07-17-cnn-introduction/

[2] Bài 15: Overfitting

https://machinelearningcoban.com/2017/03/04/overfitting/#-validation-1

[3] Data Augmentation for Deep Learning
https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9

[4] Bài 10: Các kỹ thuật cơ bản trong deep learning
https://nttuan8.com/bai-10-cac-ky-thuat-co-ban-trong-deep-learning/

[5] tensorfolow

https://www.tensorflow.org

[6] Sklearn
https://scikit-learn.org
[7] Numpy
https://numpy.org/doc/
[8] Pandas
https://pandas.pydata.org/pandas-docs/stable/
[9] Matplotlib
https://matplotlib.org/
[10] Fashion Mnist Dataset
https://www.kaggle.com/zalando-research/fashionmnist

[12] Model class API

https://keras.io/models/model/

[13] Transfer Learning cho bài toán phân loại ảnh

https://machinelearningcoban.com/2017/07/02/tl/

Bài 9: Transfer learning và data augmentation

https://nttuan8.com/bai-9-transfer-learning-va-data-augmentation/

[14] Giới thiệu về các pre-trained models trong lĩnh vực Computer Vision

https://viblo.asia/p/gioi-thieu-ve-cac-pre-trained-models-trong-linh-vuc-computer-vision-3Q75wB1GlWb