**TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

----- □ & □ -----



**BÁO CÁO BÀI TẬP LỚN**  
**THỊ GIÁC MÁY TÍNH**

**MÔ HÌNH GOOGLENET**

**THỰC NGHIỆM VỚI CIFAR-10**

**Giảng viên:** Th.s Thái Thị Nguyệt

**Sinh viên**: Nguyễn Huy Hiệu

**MSSV**: 1520065

**Lớp**: 65CS2

**---Hà Nội, tháng 6 năm 2023---**

Contents

[**I. Mô hình GoogLeNet** 3](#_Toc138771277)

[1. Tổng quát 3](#_Toc138771278)

[2. Kiến trúc 4](#_Toc138771279)

[3. Khối Inception 7](#_Toc138771280)

[4. Hàm Loss và các hyperparameter 8](#_Toc138771281)

[5. Cải tiến 8](#_Toc138771282)

[5.1 Inception v2 8](#_Toc138771283)

[5.2 Inception v3 10](#_Toc138771284)

[5.3 Inception v4 10](#_Toc138771285)

[5.4 Inception ResNet 11](#_Toc138771286)

[**II. Thực nghiệm với datasets CIFAR-10** 13](#_Toc138771287)

[2.1 Bộ dữ liệu Cifar-10 13](#_Toc138771288)

[2.2 Mô hình CNN 14](#_Toc138771289)

[2.2.1 Kiến trúc tổng quát: 14](#_Toc138771290)

[2.2.2 Thực nghiệm 15](#_Toc138771291)

[2.3 Mô hình mạng Fully Connected 16](#_Toc138771292)

[2.3.1 Kiến trúc tổng quát 16](#_Toc138771293)

[2.3.2 Thực nghiệm 18](#_Toc138771294)

[2.4 Mạng Resnet-18 19](#_Toc138771295)

[2.4.1 Kiến trúc của mô hình ResNet-18: 19](#_Toc138771296)

[2.4.2 Thực nghiệm 19](#_Toc138771297)

[2.4 Mạng Resnet-50 20](#_Toc138771298)

[2.4.1 Kiến trúc của ResNet-50 20](#_Toc138771299)

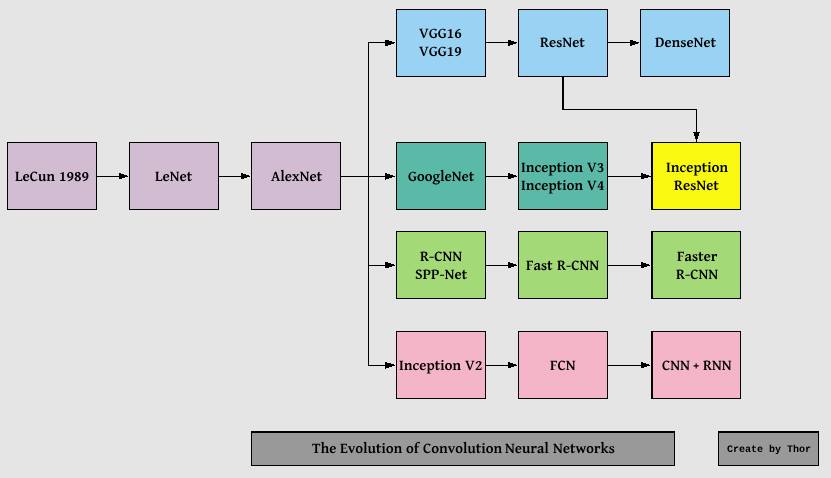
[2.4.2 Thực nghiệm 20](#_Toc138771300)

[2.5 Tổng quát 21](#_Toc138771301)

# **I. Mô hình GoogLeNet**

## 1. Tổng quát

Sau sự ra đời của mô hình CNN đầu tiên là LeCun 1989 và kết quả đột phá của các mô hình CNN như AlexNet và ZFNet đã dẫn đến sự phát triển và ra đời của rất nhiều các mô hình CNN một cách chóng mặt. đặc biệt trong các bài toán về thị giác máy tính, các mô hình CNN đã cho thấy hiệu suất vượt trội. Mô hình được em trình bày trong bài báo cáo này là mô hình CNN học sâu GoogLeNet của nhóm nghiên cứu thuộc Google DeepMind.



GoogLeNet (kết hợp giữa “Google và mạng CNN đầu tiên là LeNet) là một mô hình được đề xuất trong bài báo Going deeper with convolutions bởi các kỹ sư Google bao gồm Christian Szegedy, Yangqing Jia, Vincent Vanhoucke,… và các cộng sự vào năm 2015. Bài báo này tập trung giải quyết câu hỏi: kích thước nào của bộ lọc tích chập là tốt nhất. Mô hình GoogLeNet đã giành chiến thắng trong cuộc thi ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). GoogLeNet còn có tên gọi khác là Inception v1 vì đây là một mô hình sử dụng một khối lặp lại nhiều lần là khối Inception module. Trong bài báo gốc, các tác giả đã cho thấy thấy sự giảm đáng kể của tỷ lệ phân loại sai của mô hình này so với các mô hình AlexNet (chiến thắng ILSVRC 2012) và ZFNet (chiến thắng ILSVRC 2013) và một số mô hình khác. Mô hình GoogLeNet được sử dụng nhiều trong các bài toán phát hiện vật thể trong ảnh và phân loại ảnh.

Mô hình GoogLeNet sử dụng tổng cộng 22 tầng học sâu, bao gồm 9 khối inception. Mô hình sử dụng một số kỹ thuật mới so với các mô hình CNN trước đó như việc sử dụng các khối tích tập 1x1, sử dụng một khối Global Average Pooling (GAP) để trích chọn đặc trưng cuối cùng thay vì các khối FullyConnected xếp chồng lên nhau và kỹ thuật quan trọng nhất là việc áp dụng các khối Inception chồng lên nhau. Mô hình có thể coi là đã giải quyết được vấn đề về việc chọn kernel trong tầng tích chập thông qua việc sử dụng nhiều nhánh tích chập với các kernel size khác nhau như 1x1, 3x3, 5x5.

Mô hình đã đạt được thông số như 43.9% mAP, 6*:*67% Top-5 error.

## 2. Kiến trúc

Mô hình có kích thước ảnh đầu vào là một ảnh RGB kích thước 224x224. Mô hình này có độ sâu 22 tầng có khả năng học và 27 tầng nếu tính các tầng pooling.

Mô hình Inception v1 có kiến trúc tổng quát như sau:

|  |  |
| --- | --- |
|  | Đầu tiên mô hình sẽ đi qua một khối chồng lên nhau còn được gọi là khối Stem tương tự với mô hình LeNet và AlexNet trước khi đi vào các khối Inception, bao gồm:   1. Tầng Convolutional với 64 bộ lọc có kích thước là 7x7 và bước nhảy là 2. 2. Tầng Max Pooling với kích thước bộ lọc 3x3 và bước nhảy là 2. 3. Tầng Convolutional tiếp theo với 64 bộ lọc có kích thước là 1x1 và kích hoạt bằng hàm ReLU. 4. Tầng Convolutional tiếp theo với 64 bộ lọc có kích thước là 3x3 và kích hoạt bằng hàm ReLU   Trong bài báo gốc, giữa tầng thứ 2 và 3 của khối Stem là một tầng LocalRespNorm (Local Response Normalization).  Như đã đề cập, mô hình GoogLeNet sử dụng 9 khối Inception chồng lên nhau chia làm 3 phần, mỗi phần ngăn cách nhau bằng một lớp 3x3 MaxPooling. 3 đoạn Inception được sử dụng được gọi là Inception 3, Inception 4 và Inception 5:   * Phần Inception 3 bao gồm 2 khối Inception 3a và Inception 3b. * Phần Inception 4 bao gồm 5 khối Inception 4a, 4b, 4c, 4d, 4e * Phần Inception 5 bao gồm 2 khối Inception 5a và Inception 5b. |

Cuối cùng mô hình sẽ đi qua một tầng được gọi là Global AvgPool (GAP) trước khi thực hiện phân loại thay vì sử dụng một tầng kết nối đầy đủ như AlexNet. Tầng này lấy một bản đồ đặc trưng có kích thước 7x7 và tính trung bình để thu được đầu ra của kích thước 1x1.  Điều này giúp giảm số lượng thông số có thể huấn luyện được xuống tới 0 và cải thiện độ chính xác top-1 lên đến 0,6%.

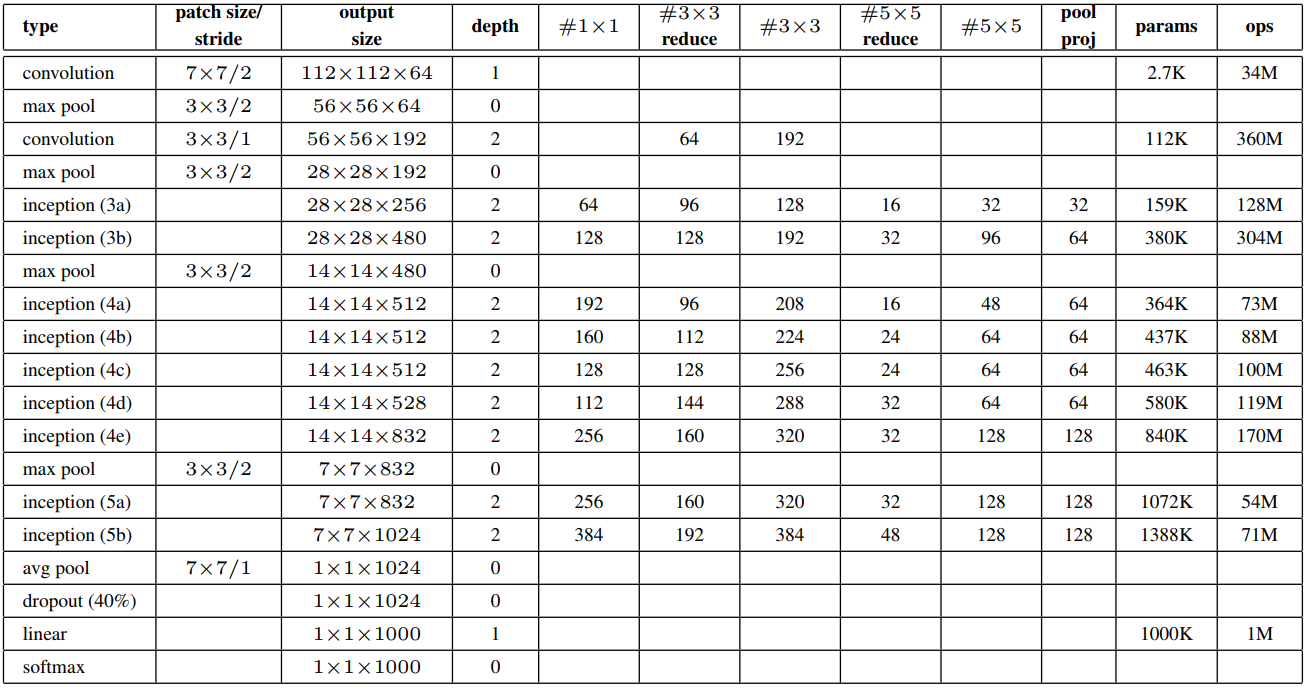
Ngoài ra trong bài báo gốc, các tác giả còn đề xuất sử dụng 2 bộ phân loại hỗ trợ (Auxiliary classifiers) giữa các tầng tích chập cụ thể được nối vào song song với khối Inception thứ 4 và 7 trong mô hình để giải quyết vấn đề xảy ra trong quá trình huấn luyện là hiện tượng gradient vanishing và gradient exploding. Với auxiliary classifier, quá trình tính toán gradient được lan truyền qua tầng phân loại, giúp cho các giá trị gradient không bị mất hoặc bị tràn và qua đó giúp cho huấn luyện mô hình tốt hơn. Bộ phân loại này có kiến trúc giống nhau với các tầng Convolutional và Pooling, tiếp theo là 2 tầng Fully Connected và cuối cùng là Softmax để phân loại.

A picture containing text, diagram, screenshot, font

Description automatically generated



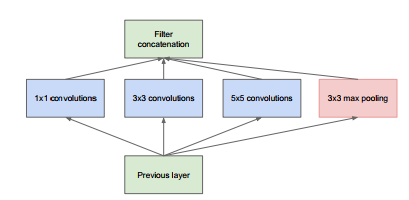
Thông số chi tiết của các tầng như sau:



## 3. Khối Inception

Như đã đề cập, khối tích chập cơ bản trong mô hình GoogLeNet được gọi là Inception, được đặt tên dựa theo cảm hứng của câu nói “Chúng ta cần đi sâu hơn” (“We Need To Go Deeper”) trong bộ phim Inception.

Ý tưởng chính của kiến trúc Inception là tìm cách tối ưu cấu trúc thưa thớt trong mạng tích chập bằng các thành phần dày đặc. Các khối xây dựng tích chập được lặp lại trong không gian và được sắp xếp thành các cụm có tương quan cao. Các cụm này hình thành các đơn vị của tầng tiếp theo và được kết nối với các đơn vị trong tầng trước đó. Các đơn vị tương ứng với các khu vực trong ảnh đầu vào và được nhóm thành bộ lọc. Kiến trúc của một khối Inception như sau:



Kiến trúc ngây thơ

A diagram of a flowchart

Description automatically generated with low confidence

Kiến trúc đầy đủ

Như mô tả ở hình trên, khối inception bao gồm bốn nhánh song song với nhau. Theo kiến trúc ngây thơ ban đầu của các tác giả, ba nhánh đầu sử dụng các tầng tích chập với kích thước kernel lần lượt là 1×1, 3×3, và 5×5 để trích xuất thông tin từ các vùng không gian có kích thước khác nhau. Nhánh thứ tư sử dụng một tầng maxpool kích thước 3×3. Tuy nhiên trong kiến trúc cuối cùng có thêm các tầng tích chập 1x1 ở hai nhánh giữa để giảm số kênh đầu vào, từ đó giảm độ phức tạp của mô hình. Nhánh thứ tư tương tự cũng sử dụng một tầng tích chập 1×1 để thay đổi số lượng kênh. Cả bốn nhánh sử dụng phần đệm phù hợp để đầu vào và đầu ra của khối có cùng chiều cao và chiều rộng. Cuối cùng, các đầu ra của mỗi nhánh sẽ được nối lại theo chiều kênh để tạo thành đầu ra của cả khối. Các tham số thường được tinh chỉnh của khối Inception là số lượng kênh đầu ra mỗi tầng.

Sự hiệu quả của khối Inception là do chúng khám phá hình ảnh trên các vùng có kích thước khác nhau. Tức là những chi tiết ở những mức độ khác nhau sẽ được nhận diện một cách hiệu quả bằng các bộ lọc kích thước khác nhau. Đồng thời, chúng ta có thể phân bổ số lượng tham số khác nhau cho những vùng có phạm vi khác nhau.

## 4. Hàm Loss và các hyperparameter

Hàm loss được sử dụng trong mô hình GoogleNet là Softmax Cross-entropy loss.

Các Hyperparameter được tác giả sử dụng trong mô hình:

* Optimizer: asynchronous stochastic gradient descent
* Momentum: 0.9
* Learning rate: không đề cập chính xác tuy nhiên được giảm xuống 4% mỗi 8 epoch
* Drop\_out: 0.7

Ngoài ra trong mô hình gốc, các tác giả còn sử dụng một số kỹ thuật trong quá trình huấn luyện như:

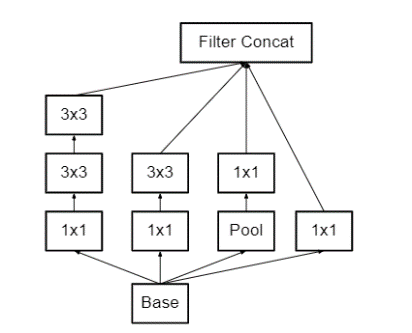
* Polyak averaging: là một kỹ thuật regularization (chính quy hóa) để có được một mô hình có độ chính xác cao và tránh hiện tượng overfitting (quá khớp)
* Photometric distortions: là các biến đổi được áp dụng lên các ảnh để làm thay đổi độ sáng, màu sắc và độ tương phản trong ảnh. Trong mô hình, các tác giả sử dụng Photometric distortions để xử lý vấn đề overfitting.
* Một số kỹ thuật nội suy (interpolation) như bilinear, area, nearest neighbor and cubic

## 5. Cải tiến

Việc Inception v1 sử dụng các tầng tích chập 5x5 đã gây giảm kích thước đầu vào một cách đáng kể. Điều này dẫn đến mất mát độ chính xác của mạng neural nếu kích thước đầu vào giảm đột biến. Ngoài ra các khối này cũng khiến độ phức tạp lớn hơn so với khi sử dụng các khối 3x3. Hạn chế trên của mô hình Inception v1 đã dẫn đến sự ra đời của các mô hình cải tiến như Inception v2, Inception v3,…

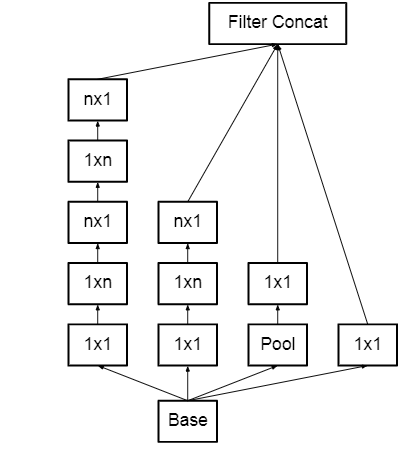
### 5.1 Inception v2

Thay đổi tổng thể đáng chú ý của mô hình Inception v2 so với GoogleNet là việc phân chia 3 đoạn Inception 3, 4, 5 đã được thay đổi số khối Incpetion từ 2-5-2 thành 3-5-2 tức là thêm một khối Inception 3c vào thành 10 khối tất cả. Trong mô hình Inception v2, do hạn chế đã được đề cập ở trên nên các khối tích chập 5x5 đã được thay thế bằng khối 3x3 tương ứng ở các khối 3a, 3b và 3c

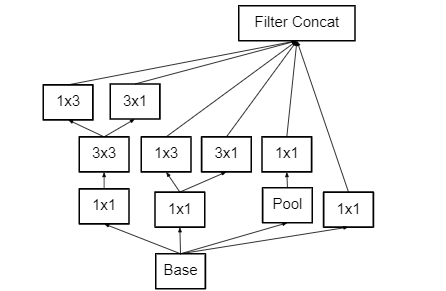


Mặc dù độ phức tạp đã được giảm khi thay như trên tuy nhiên ta có thể thu gọn convolution 3x3 thành hai convolution không đối xứng 1x3 và 3x1. Điều này tương đương với việc trượt một mạng hai lớp có vùng nhận tính toán tương đương với convolution 3x3 nhưng độ phức tạp thấp hơn 33%.

Các tác giả cũng cho rằng điều này có thể được áp dụng với bất kì khối nxn bất kì. Như vậy các khối 4a, 4b, 4c, 4d, 4e có kiến trúc như sau, trong bài báo đề xuất Inception v2, các tác giả sử dụng n = 7



Tuy nhiên để tránh vấn đề representational bottleneck, khi lớp trung gian giữa các lớp đầu vào và đầu ra của một mạng neural không đủ rộng hoặc chưa đủ sâu để có thể chứa đựng đầy đủ thông tin của dữ liệu đầu vào, hai khối 1x3 và 3x1 sẽ được mở rộng thành 2 nhánh thay vì chồng lên nhau ở 2 khối 5a và 5b.



Cuối cùng, các tác giả cho rằng các bộ phân loại hỗ trợ được sử dụng trong mô hình gốc không thực sự có tác dụng như được đề xuất, thay vào đó các bộ phân loại này có thể được sử dụng như regularizer để giảm thiểu các hiện tượng overfitting với việc sử dụng thêm lớp batch-normalized hoặc sử dụng Drop\_out.

### 5.2 Inception v3

Kiến trúc của Inception v3 có thay đổi tương đồng với kiến trúc Inception v2, cộng với một số thay đổi như sau:

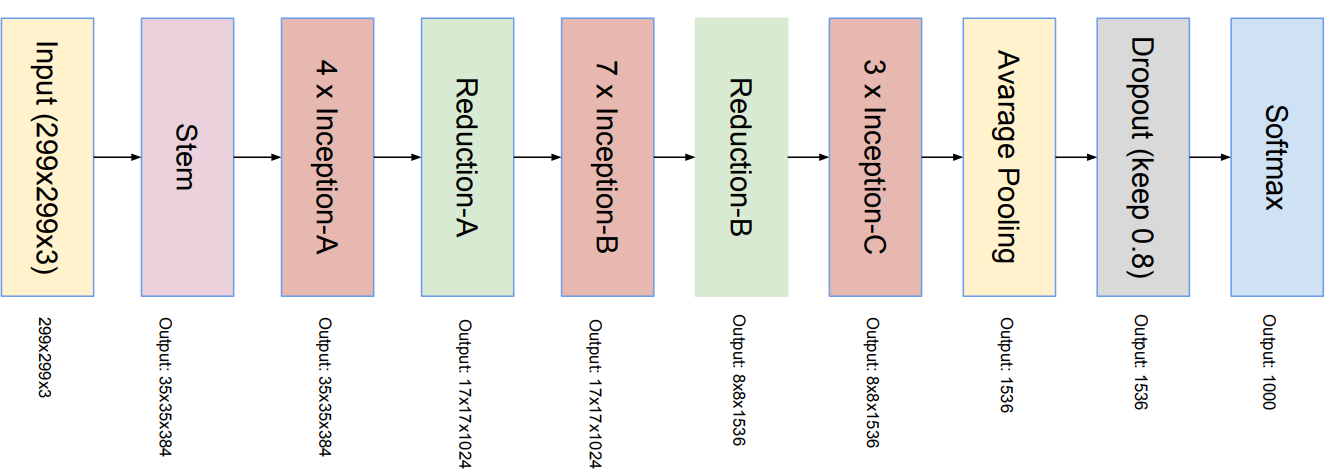
* Sử dụng bộ tối ưu hóa RMSprop.
* Sử dụng Batch Normalization trong lớp fully connected của bộ phân loại phụ.
* Phần thu giảm kích thước của các kernel 5 × 5 và 3 × 3 được chia thành hai kernel một chiều (1x7 và 7x1, 1x3 và 3x1, tương ứng).
* Label Smoothing Regularization: là một kỹ thuật regularization được sử dụng để cải thiện độ chính xác và độ ổn định của bộ phân loại. Nó giúp bộ phân loại tránh dự đoán một nhóm quá tự tin. Việc thêm Label Smoothing cải thiện 0,2% từ tỉ lệ lỗi.

Ngoài ra, kích thước đầu vào của mạng đã thay đổi từ 224 × 224 thành 299 × 299.

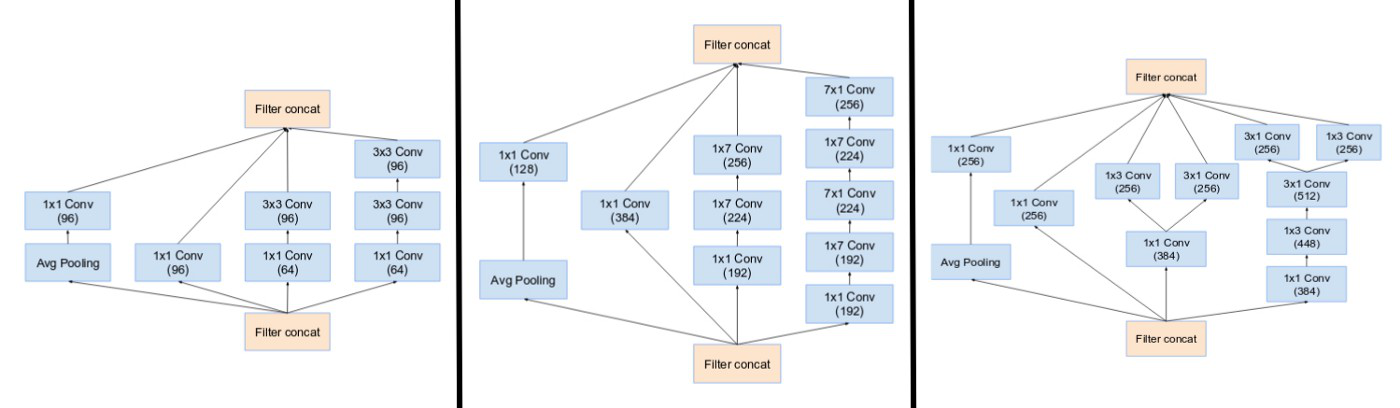
### 5.3 Inception v4

Inception v4 được đề xuất nhằm giảm độ phức tạp xuống so với Inception v3 trước đó.

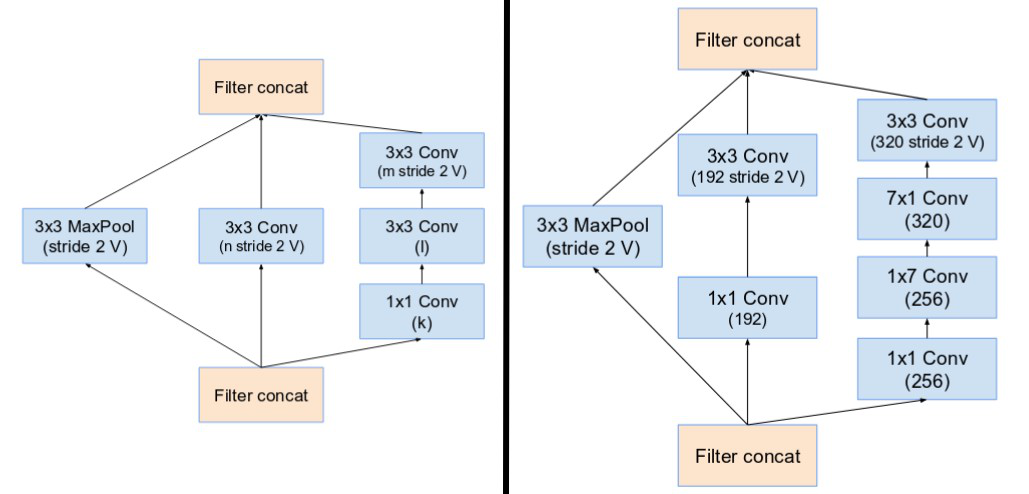
Kiến trúc tổng thể của Inception v4:



Có thể thấy các tác giả đã thêm các khối Inception trong mô hình với tổng cộng 14 khối Inception chia thành Inception A với 4 khối, Incption B với 7 khối và Inception C với 3 khối với kiến trúc các khối A, B, C lần lượt như sau:



Ngoài ra, giữa các khối Inception sẽ là một khối reduction module  được sử dụng để giảm kích thước không gian của đầu vào giúp tăng tốc độ tính toán và giảm độ sâu của mô hình. Sau khi qua Reduction Module, đầu ra của mô hình sẽ có kích thước nhỏ hơn, điều này giúp giảm thiểu số lượng tham số trong mô hình và tăng tốc độ tính toán.

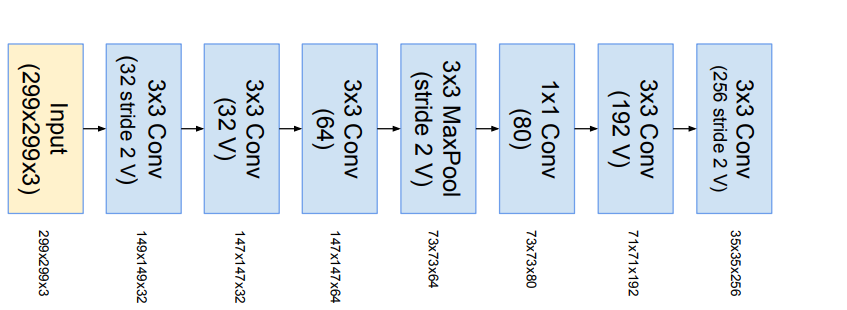
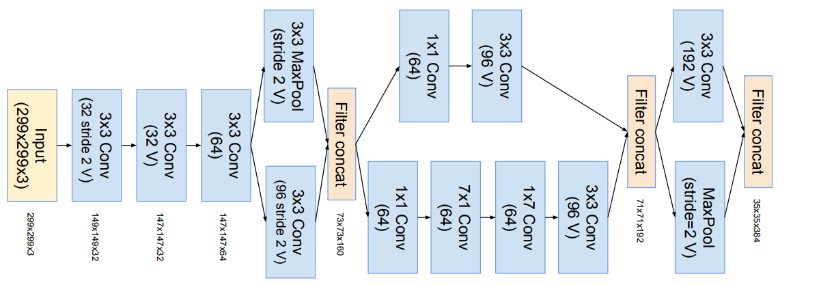


Kiến trúc khối Reduction A và Reduction B tương ứng

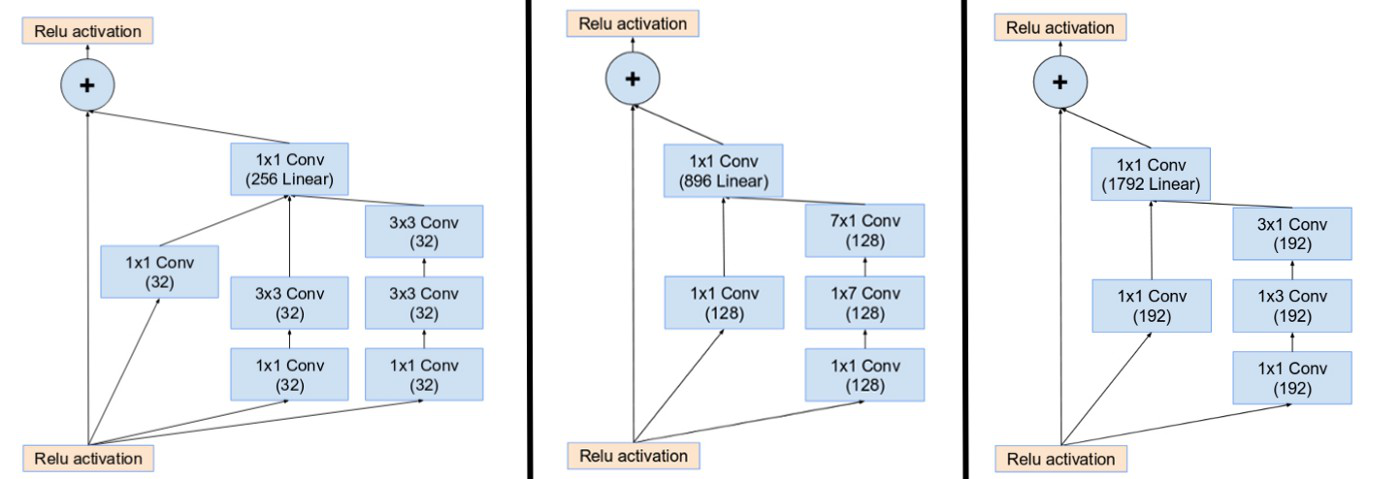
### 5.4 Inception ResNet

Cũng trong bài báo đề xuất mô hình Inception v4, bị ấn tượng bởi hiệu năng mà mạng tích chập ResNet đạt được, các tác giả đã đề xuất 2 mô hình kết hợp giữa mô hình Inception và mô hình ResNet gọi là  Inception ResNet V1 và Inception ResNet V2.

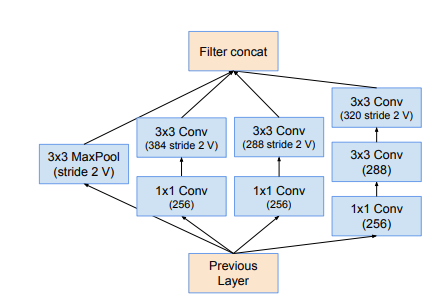
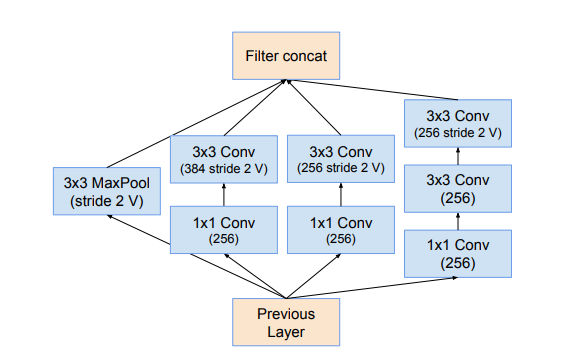
Cấu trúc của 2 mô hình này về tổng thể không khác so với mô hình Inception v4 đã đề cập tuy nhiên số lượng khối Inception được tăng lên và chia theo tỉ lệ 5-10-5, ngoài ra khối Stem ở đầu cũng có sự khác nhau như sau:

Đối với khối Inception ở cả 2 mô hình, khối Inception ở mô hình Inception v4 đã được thay thế bằng một khối Inception mới đã được kết hợp với kiến trúc khối Residual của mô hình ResNet. Kiến trúc tương ứng kiến trúc các khối A, B, C lần lượt như sau:



So với mô hình Inception v4, khối Reduction Block B ở cả 2 mô hình cũng có sự khác biệt như sau:



# **II. Thực nghiệm với datasets CIFAR-10**

Cài đặt và huấn luyện 4 mô hình sau:

* Mô hình CNN cơ bản
* Mô hình FullyConnected cơ bản
* Mô hình Resnet-18
* Mô hình Resnet-50

Với mô hình CNN cơ bản và mô hình mạng FC vì kiến trúc khá đơn giản nên em quyết định thử nghiệm chạy thêm và sử dụng Drop\_out.

Các hyperparameter chung dùng để huấn luyện:

* Hàm loss: CrossEntropy
* Optimizer: SGD
* Learning rate: 0,01
* Momentum: 0.9
* Epoch: 20

## 2.1 Bộ dữ liệu Cifar-10

CIFAR-10 (Canadian Institute for Advanced Research) là một bộ dữ liệu hình ảnh được sử dụng rộng rãi trong lĩnh vực học sâu và máy học. Bộ dữ liệu này bao gồm một tập hợp 60.000 ảnh kích thước 32x32 pixels RGB chia thành 2 tập huấn luyện và kiểm tra, Tập dữ liệu bao gồm 10 lớp với mỗi lớp có 6.000 ảnh (5.000 ảnh trong tập huấn luyện và 1.000 ảnh trong tập kiểm tra). Các lớp này bao gồm:

|  |  |
| --- | --- |
| * Airplane * Automobile * Bird * Cat * Deer | * Dog * Frog * Horse * Ship * Truck |

Tập dữ liệu CIFAR-10 thường được sử dụng phổ biến trong việc kiểm tra các mô hình phân lớp.

A collage of images of animals and vehicles

Description automatically generated with low confidence

## 2.2 Mô hình CNN

### 2.2.1 Kiến trúc tổng quát:

* Một lớp Convolutional Layer với 16 filters, kích thước kernel là 3x3, độ dịch chuyển là 1, padding là 1.
* Một lớp kích hoạt ReLU được áp dụng sau Convolutional Layer đầu tiên.
* Một lớp Dropout với xác suất bỏ qua 25%, được áp dụng trước MaxPooling Layer đầu tiên để giảm quá trình quá khớp (overfitting).
* Một lớp MaxPooling Layer với kích thước kernel là 2x2, độ dịch chuyển là 2.
* Một lớp Convolutional Layer khác với 32 filters, kích thước kernel là 3x3, độ dịch chuyển là 1, padding là 1.
* Một lớp kích hoạt ReLU được áp dụng sau Convolutional Layer thứ hai.
* Một lớp Dropout khác với xác suất bỏ qua 50%, được áp dụng trước MaxPooling Layer thứ hai để giảm quá trình quá khớp.
* Một lớp MaxPooling Layer thứ hai với kích thước kernel là 2x2, độ dịch chuyển là 2.
* Một lớp Linear FC Layer với 10 nơ-ron đầu ra, được kết nối với đầu ra của MaxPooling Layer thứ hai.
* Cuối cùng là một lớp Dropout, được áp dụng trước khi đầu ra được trả về.

Lưu ý: Các lớp Dropout chỉ được kích hoạt khi giá trị tham số Drop\_out = True.

- Với mô hình không sử dụng Drop\_out:

A screenshot of a computer program

Description automatically generated with low confidence

- Với mô hình sử dụng Drop\_out:

A screenshot of a computer program

Description automatically generated with low confidence

### 2.2.2 Thực nghiệm

- Với mô hình không sử dụng Drop\_out:

Tổng thời gian huấn luyện: *302.92*s, trung bình *15.15*s mỗi epoch

Độ chính xác: *68.93*%, cao nhất là *69.60*% ở epoch 8

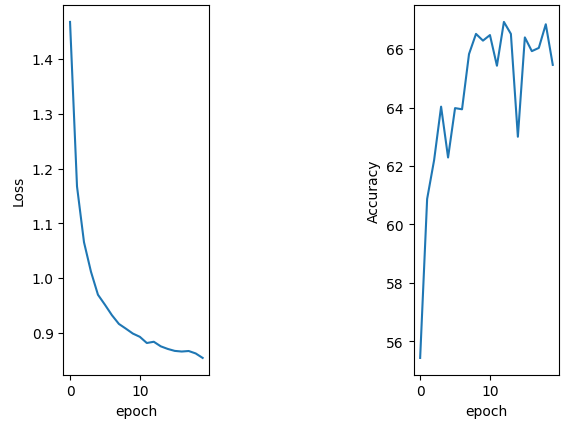
A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated with low confidence

- Với mô hình sử dụng Drop\_out:

Tổng thời gian huấn luyện là *297.91*s, trung bình *14.9*s mỗi epoch

Độ chính xác *65.46*%, cao nhất là *66.93*% ở epoch 13



## 2.3 Mô hình mạng Fully Connected

### 2.3.1 Kiến trúc tổng quát

* Lớp đầu tiên là một Fully Connected Layer với 512 đơn vị ẩn, kết nối với đầu vào với kích thước là 32323. `nn.Linear(32\*32\*3, 512)`
* Tiếp theo, một lớp Dropout với xác suất bỏ qua 50% được áp dụng sau dense layer đầu tiên để giảm quá trình quá khớp (overfitting). `self.dropout1 = nn.Dropout(p=0.5)`
* Lớp thứ hai có 256 đơn vị ẩn, kết nối với lớp trước đó. `nn.Linear(512, 256)`
* Một lớp Dropout khác với xác suất bỏ qua 50% cũng được áp dụng trước lớp Linear Layer thứ hai. `self.dropout2 = nn.Dropout(p=0.5)`
* Lớp thứ ba có 128 đơn vị ẩn kết nối với lớp trước đó. `nn.Linear(256, 128)`
* Lớp Dropout thứ ba với xác suất bỏ qua là 25%. `self.dropout3 = nn.Dropout(p=0.25)`
* Lớp cuối cùng là một Linear Layer với 10 đơn vị đầu ra, kết nối với lớp trước đó. `nn.Linear(128, 10)`
* Cuối cùng, hàm kích hoạt ReLU được áp dụng trên tất cả các lớp Linear trừ lớp cuối cùng. Chúng được áp dụng là `self.relu = nn.ReLU(inplace=True)`.

Lưu ý: Các lớp Dropout chỉ được kích hoạt khi giá trị tham số Drop\_out = True.

- Mô hình không sử dụng Drop\_out:

A picture containing text, screenshot, number, font

Description automatically generated

- Mô hình sử dụng Drop\_out:

A screenshot of a computer

Description automatically generated with medium confidence

### 2.3.2 Thực nghiệm

- Với mô hình không sử dụng Drop\_out:

Tổng thời gian huấn luyện là *294.06*s, trung bình *14.7*s mỗi epoch

Độ chính xác: *53.92*%, cao nhất là *54.95*% ở epoch 9

A picture containing text, diagram, line, plot

Description automatically generated

- Với mô hình sử dụng Drop\_out:

Tổng thời gian huấn luyện là *316.5*s, trung bình *15.83*s mỗi epoch

Độ chính xác: *48.72*% (cao nhất)

A picture containing text, diagram, line, plot

Description automatically generated

## 2.4 Mạng Resnet-18

### 2.4.1 Kiến trúc của mô hình ResNet-18:

* Lớp Convolutional đầu tiên với 64 filters kích thước kernel là 3x3, stride là 1, padding là 1.
* 4 nhóm các lớp Residual Block (BasicBlock), mỗi nhóm bao gồm 2 hoặc 3 khối.
* Mỗi block bao gồm 2 lớp Convolutional với số filters là 64 và kích thước kernel là 3x3.
* Các lớp Convolutional có thể có padding 1 nếu số filters đầu ra khác số filters đầu vào.
* Lớp kích hoạt ReLU được áp dụng giữa các lớp Convolutional.
* Một shortcut connection với lớp Convolutional đầu tiên của block được thêm vào nếu số filters đầu ra của block khác với số filters đầu vào của block.
* Shortcut connection này có thể là một lớp Convolutional 1x1 để giảm bớt kích thước hoặc là một lớp trực tiếp để giữ không đổi kích thước.
* Một Lớp Average Pooling với kích thước kernel là 7x7 và stride 1.
* Một lớp Fully Connected Layer với 1000 nơ-ron đầu ra.

A picture containing text, receipt, font, screenshot

Description automatically generated

### 2.4.2 Thực nghiệm

Tổng thời gian huấn luyện là *864.33*s, trung bình *43.22*s mỗi epoch

Độ chính xác: *83.70*%,cao nhất là *84.09*% ở epoch 19

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated with low confidence

## 2.4 Mạng Resnet-50

### 2.4.1 Kiến trúc của ResNet-50

* Lớp Convolutional đầu tiên với 64 filters kích thước kernel là 7x7, stride là 2, padding là 3, được theo sau bởi một lớp MaxPooling với kích thước kernel là 3x3 và stride là 2.
* Ba nhóm các lớp Residual Block (Bottleneck Block):
* Mỗi bottlenecks block bao gồm 3 lớp Convolutional có số lượng filters tăng dần từ 64 đến 256 theo thứ tự 64, 64, 256 với kích thước kernel lần lượt là 1x1, 3x3, 1x1 và stride bằng 1. Các lớp Convolutional có thể có padding 1 nếu số filters đầu ra khác số filters đầu vào.
* Lớp kích hoạt ReLU được áp dụng giữa các lớp Convolutional.
* Một shortcut connection với lớp Convolutional đầu tiên của block được thêm vào nếu số filters đầu ra của block khác với số filters đầu vào của block.
* Shortcut connection này là một lớp Convolutional 1x1 để giảm bớt kích thước.
* Một Lớp Average Pooling với kích thước kernel là 7x7 và stride 1.
* Một lớp Fully Connected Layer với 1000 nơ-ron đầu ra.

### 2.4.2 Thực nghiệm

Tổng thời gian huấn luyện là *3240.71*s , trung bình *162.04*s một epoch

Độ chính xác: *82.47*% (cao nhất)

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated with low confidence

## 2.5 Tổng quát

Ta có bảng kết quả sau:

|  |  |  |  |
| --- | --- | --- | --- |
| Mô hình | Parameters | Thời gian trung bình mỗi epoch | Thông số sau khi huấn luyện |
| CNN cơ bản | 25,578 | 15.15s | Loss: 0.6089 |
| Accuracy: 68.93% |
| CNN cơ bản sử dụng Drop\_out | 25,578 | 14.9s | Loss: 0.8543 |
| Accuracy: 66.93% |
| Mạng kết nối đầy đủ (FC) | 1,738,890 | 14.70s | Loss: 0.4012 |
| Accuracy: 53.92% |
| Mạng kết nối đầy đủ (FC) sử dụng Drop\_out | 1,738,890 | 15.83s | Loss: 1.3304 |
| Accuracy: 48.72% |
| ResNet-18 | 11,220,132 | 43.22s | Loss: 0.0061 |
| Accuracy: 83.70% |
| ResNet-50 | 23,705,252 | 162.04s | Loss: 0.0279 |
| Accuracy: 82.47% |

2.6 Nhận xét

Dịch Pytorch

Phần này chạy qua API dành cho các tác vụ phổ biến trong máy học. Tham khảo các liên kết trong mỗi phần để tìm hiểu sâu hơn.

1. Làm việc với dữ liệu

PyTorch có hai [nguyên mẫu để làm việc với dữ liệu](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) : torch.utils.data.DataLoadervà torch.utils.data.Dataset. Datasetlưu trữ các mẫu và nhãn tương ứng của chúng, đồng thời DataLoaderbao bọc một lần lặp xung quanh tệp Dataset.

**import** **torch**

**from** **torch** **import** **nn**

**from** **torch.utils.data** **import** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)

**from** **torchvision** **import** **datasets**

**from** **torchvision.transforms** **import** [**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)

PyTorch cung cấp các thư viện dành riêng cho miền như [TorchText](https://pytorch-org.translate.goog/text/stable/index.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) , [TorchVision](https://pytorch-org.translate.goog/vision/stable/index.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) và [TorchAudio](https://pytorch-org.translate.goog/audio/stable/index.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) , tất cả đều bao gồm bộ dữ liệu. Đối với hướng dẫn này, chúng tôi sẽ sử dụng bộ dữ liệu TorchVision.

Mô torchvision.datasets-đun chứa Datasetcác đối tượng cho nhiều dữ liệu tầm nhìn trong thế giới thực như CIFAR, COCO. Trong hướng dẫn này, chúng tôi sử dụng bộ dữ liệu FashionMNIST. Mỗi TorchVision Datasetbao gồm hai đối số: transformvà target\_transformđể sửa đổi mẫu và nhãn tương ứng.

*# Download training data from open datasets.*

[**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST) **=** [**datasets.FashionMNIST**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**(**

**root=**"data"**,**

**train=True,**

**download=True,**

**transform=**[**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**(),**

**)**

*# Download test data from open datasets.*

[**test\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST) **=** [**datasets.FashionMNIST**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**(**

**root=**"data"**,**

**train=False,**

**download=True,**

**transform=**[**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**(),**

**)**

Chúng tôi chuyển the Datasetlàm đối số cho DataLoader. Điều này kết thúc một lần lặp lại trên tập dữ liệu của chúng tôi và hỗ trợ tải dữ liệu theo nhóm, lấy mẫu, xáo trộn và đa xử lý tự động. Ở đây, chúng tôi xác định kích thước lô là 64, tức là mỗi phần tử trong trình tải dữ liệu có thể lặp lại sẽ trả về một lô gồm 64 tính năng và nhãn.

**batch\_size** **=** 64

*# Create data loaders.*

[**train\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader) **=** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**(**[**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**,** **batch\_size=batch\_size)**

[**test\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader) **=** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**(**[**test\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**,** **batch\_size=batch\_size)**

**for** [**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **y** **in** [**test\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**:**

print**(**f"Shape of X [N, C, H, W]: *{*[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.shape***}*"**)**

print**(**f"Shape of y: *{***y.shape***}* *{***y.dtype***}*"**)**

**break**

Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])

Shape of y: torch.Size([64]) torch.int64

Đọc thêm về [tải dữ liệu trong PyTorch](https://pytorch-org.translate.goog/tutorials/beginner/basics/data_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) .

Tạo mô hình

Để xác định mạng thần kinh trong PyTorch, chúng tôi tạo một lớp kế thừa từ [nn.Module](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) . Chúng tôi xác định các lớp của mạng trong \_\_init\_\_hàm và chỉ định cách dữ liệu sẽ truyền qua mạng trong forwardhàm. Để tăng tốc hoạt động trong mạng thần kinh, chúng tôi chuyển nó sang GPU hoặc MPS nếu có.

*# Get cpu, gpu or mps device for training.*

**device** **=** **(**

"cuda"

**if** [**torch.cuda.is\_available**](https://pytorch-org.translate.goog/docs/stable/generated/torch.cuda.is_available.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.cuda.is_available)**()**

**else** "mps"

**if** [**torch.backends.mps.is\_available**](https://pytorch-org.translate.goog/docs/stable/backends.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.backends.mps.is_available)**()**

**else** "cpu"

**)**

print**(**f"Using *{***device***}* device"**)**

*# Define model*

**class** **NeuralNetwork(**[**nn.Module**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module)**):**

**def** \_\_init\_\_**(**self**):**

super**().**\_\_init\_\_**()**

self**.flatten** **=** [**nn.Flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten)**()**

self**.linear\_relu\_stack** **=** [**nn.Sequential**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Sequential.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Sequential)**(**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**28**\***28**,** 512**),**

[**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**(),**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**512**,** 512**),**

[**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**(),**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**512**,** 10**)**

**)**

**def** **forward(**self**,** [**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**):**

[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** self**.flatten(**[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

**logits** **=** self**.linear\_relu\_stack(**[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

**return** **logits**

**model** **=** [**NeuralNetwork**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module)**().to(device)**

print**(model)**

Using cuda device

NeuralNetwork(

(flatten): Flatten(start\_dim=1, end\_dim=-1)

(linear\_relu\_stack): Sequential(

(0): Linear(in\_features=784, out\_features=512, bias=True)

(1): ReLU()

(2): Linear(in\_features=512, out\_features=512, bias=True)

(3): ReLU()

(4): Linear(in\_features=512, out\_features=10, bias=True)

)

)

Đọc thêm về [cách xây dựng mạng thần kinh trong PyTorch](https://pytorch-org.translate.goog/tutorials/beginner/basics/buildmodel_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) .

Tối ưu hóa các tham số mô hình

Để huấn luyện một mô hình, chúng ta cần một [hàm mất mát](https://pytorch-org.translate.goog/docs/stable/nn.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#loss-functions) và một [trình tối ưu hóa](https://pytorch-org.translate.goog/docs/stable/optim.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) .

[**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss) **=** [**nn.CrossEntropyLoss**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**()**

[**optimizer**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD) **=** [**torch.optim.SGD**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD)**(**[**model.parameters**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.parameters)**(),** **lr=**1e-3**)**

Trong một vòng huấn luyện duy nhất, mô hình đưa ra các dự đoán trên tập dữ liệu huấn luyện (được cung cấp cho nó theo đợt) và lan truyền ngược lỗi dự đoán để điều chỉnh các tham số của mô hình.

**def** **train(dataloader,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**,** [**optimizer**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD)**):**

**size** **=** len**(dataloader.dataset)**

[**model.train**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.train)**()**

**for** **batch,** **(**[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **y)** **in** enumerate**(dataloader):**

[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **y** **=** [**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.to(device),** **y.to(device)**

*# Compute prediction error*

[**pred**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** **model(**[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

**loss** **=** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**(**[**pred**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **y)**

*# Backpropagation*

**loss.backward()**

[**optimizer.step**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.Optimizer.step.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.Optimizer.step)**()**

[**optimizer.zero\_grad**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD.zero_grad)**()**

**if** **batch** **%** 100 **==** 0**:**

**loss,** **current** **=** **loss.item(),** **(batch** **+** 1**)** **\*** len**(**[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

print**(**f"loss: *{***loss***:*>7f*}* [*{***current***:*>5d*}*/*{***size***:*>5d*}*]"**)**

Chúng tôi cũng kiểm tra hiệu suất của mô hình so với tập dữ liệu thử nghiệm để đảm bảo mô hình đang học.

**def** **test(dataloader,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**):**

**size** **=** len**(dataloader.dataset)**

**num\_batches** **=** len**(dataloader)**

[**model.eval**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.eval)**()**

**test\_loss,** **correct** **=** 0**,** 0

**with** [**torch.no\_grad**](https://pytorch-org.translate.goog/docs/stable/generated/torch.no_grad.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.no_grad)**():**

**for** [**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **y** **in** **dataloader:**

[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **y** **=** [**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.to(device),** **y.to(device)**

[**pred**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** **model(**[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

**test\_loss** **+=** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**(**[**pred**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **y).item()**

**correct** **+=** **(**[**pred**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.argmax(**1**)** **==** **y).type(**[**torch.float**](https://pytorch-org.translate.goog/docs/stable/tensor_attributes.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.dtype)**).sum().item()**

**test\_loss** **/=** **num\_batches**

**correct** **/=** **size**

print**(**f"Test Error: **\n** Accuracy: *{***(**100**\*correct)***:*>0.1f*}*%, Avg loss: *{***test\_loss***:*>8f*}* **\n**"**)**

Quá trình đào tạo được tiến hành qua nhiều lần lặp lại ( *kỷ nguyên* ). Trong mỗi kỷ nguyên, mô hình sẽ học các tham số để đưa ra dự đoán tốt hơn. Chúng tôi in độ chính xác và mất mát của mô hình ở mỗi kỷ nguyên; chúng tôi muốn thấy độ chính xác tăng lên và tổn thất giảm đi sau mỗi kỷ nguyên.

**epochs** **=** 5

**for** **t** **in** range**(epochs):**

print**(**f"Epoch *{***t+**1*}***\n**-------------------------------"**)**

**train(**[**train\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**,** [**optimizer**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD)**)**

**test(**[**test\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**)**

print**(**"Done!"**)**

Epoch 1

-------------------------------

loss: 2.303494 [ 64/60000]

loss: 2.294637 [ 6464/60000]

loss: 2.277102 [12864/60000]

loss: 2.269977 [19264/60000]

loss: 2.254234 [25664/60000]

loss: 2.237145 [32064/60000]

loss: 2.231056 [38464/60000]

loss: 2.205036 [44864/60000]

loss: 2.203239 [51264/60000]

loss: 2.170890 [57664/60000]

Test Error:

Accuracy: 53.9%, Avg loss: 2.168587

Epoch 2

-------------------------------

loss: 2.177784 [ 64/60000]

loss: 2.168083 [ 6464/60000]

loss: 2.114908 [12864/60000]

loss: 2.130411 [19264/60000]

loss: 2.087470 [25664/60000]

loss: 2.039667 [32064/60000]

loss: 2.054271 [38464/60000]

loss: 1.985452 [44864/60000]

loss: 1.996019 [51264/60000]

loss: 1.917239 [57664/60000]

Test Error:

Accuracy: 60.2%, Avg loss: 1.920371

Epoch 3

-------------------------------

loss: 1.951699 [ 64/60000]

loss: 1.919513 [ 6464/60000]

loss: 1.808724 [12864/60000]

loss: 1.846544 [19264/60000]

loss: 1.740612 [25664/60000]

loss: 1.698728 [32064/60000]

loss: 1.708887 [38464/60000]

loss: 1.614431 [44864/60000]

loss: 1.646473 [51264/60000]

loss: 1.524302 [57664/60000]

Test Error:

Accuracy: 61.4%, Avg loss: 1.547089

Epoch 4

-------------------------------

loss: 1.612693 [ 64/60000]

loss: 1.570868 [ 6464/60000]

loss: 1.424729 [12864/60000]

loss: 1.489538 [19264/60000]

loss: 1.367247 [25664/60000]

loss: 1.373463 [32064/60000]

loss: 1.376742 [38464/60000]

loss: 1.304958 [44864/60000]

loss: 1.347153 [51264/60000]

loss: 1.230657 [57664/60000]

Test Error:

Accuracy: 62.7%, Avg loss: 1.260888

Epoch 5

-------------------------------

loss: 1.337799 [ 64/60000]

loss: 1.313273 [ 6464/60000]

loss: 1.151835 [12864/60000]

loss: 1.252141 [19264/60000]

loss: 1.123040 [25664/60000]

loss: 1.159529 [32064/60000]

loss: 1.175010 [38464/60000]

loss: 1.115551 [44864/60000]

loss: 1.160972 [51264/60000]

loss: 1.062725 [57664/60000]

Test Error:

Accuracy: 64.6%, Avg loss: 1.087372

Done!

Đọc thêm về [Đào tạo mô hình của bạn](https://pytorch-org.translate.goog/tutorials/beginner/basics/optimization_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) .

Lưu mô hình

Một cách phổ biến để lưu một mô hình là tuần tự hóa từ điển trạng thái bên trong (chứa các tham số của mô hình).

[**torch.save**](https://pytorch-org.translate.goog/docs/stable/generated/torch.save.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.save)**(**[**model.state\_dict**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.state_dict)**(),** "model.pth"**)**

print**(**"Saved PyTorch Model State to model.pth"**)**

# **2.TENXƠ**

Tenor là một cấu trúc dữ liệu chuyên biệt rất giống với mảng và ma trận. Trong PyTorch, chúng tôi sử dụng tenxơ để mã hóa đầu vào và đầu ra của một mô hình, cũng như các tham số của mô hình.

Tenors tương tự như ndarray [của NumPy](https://translate.google.com/website?sl=en&tl=vi&hl=vi&client=webapp&u=https://numpy.org/) , ngoại trừ việc tensors có thể chạy trên GPU hoặc các bộ tăng tốc phần cứng khác. Trên thực tế, các mảng tensors và NumPy thường có thể chia sẻ cùng một bộ nhớ cơ bản, loại bỏ nhu cầu sao chép dữ liệu (xem [Bridge with NumPy](https://pytorch-org.translate.goog/tutorials/beginner/blitz/tensor_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#bridge-to-np-label) ). Các tenxơ cũng được tối ưu hóa để tự động phân biệt (chúng ta sẽ xem thêm về điều đó sau trong phần [Autograd](https://pytorch-org.translate.goog/tutorials/beginner/basics/autogradqs_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) ). Nếu bạn đã quen thuộc với ndarrays, thì bạn sẽ thấy quen thuộc với Tensor API. Nếu không, hãy làm theo!

**import** **torch**

**import** **numpy** as **np**

## Khởi tạo một Tenor

Tenor có thể được khởi tạo theo nhiều cách khác nhau. Hãy xem các ví dụ sau:

**Trực tiếp từ dữ liệu**

Các tenxơ có thể được tạo trực tiếp từ dữ liệu. Kiểu dữ liệu được tự động suy ra.

**data** **=** **[[**1**,** 2**],[**3**,** 4**]]**

[**x\_data**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.tensor**](https://pytorch-org.translate.goog/docs/stable/generated/torch.tensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.tensor)**(data)**

**Từ một mảng NumPy**

Các thang đo có thể được tạo từ các mảng NumPy (và ngược lại - xem [Bridge with NumPy](https://pytorch-org.translate.goog/tutorials/beginner/blitz/tensor_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#bridge-to-np-label) ).

**np\_array** **=** **np.array(data)**

[**x\_np**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.from\_numpy**](https://pytorch-org.translate.goog/docs/stable/generated/torch.from_numpy.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.from_numpy)**(np\_array)**

**Từ một tensor khác:**

Tenxơ mới giữ lại các thuộc tính (hình dạng, kiểu dữ liệu) của tenxơ đối số, trừ khi được ghi đè rõ ràng.

[**x\_ones**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.ones\_like**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones_like.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones_like)**(**[**x\_data**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)** *# retains the properties of x\_data*

print**(**f"Ones Tensor: **\n** *{*[**x\_ones**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}* **\n**"**)**

[**x\_rand**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand\_like**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand_like.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand_like)**(**[**x\_data**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **dtype=**[**torch.float**](https://pytorch-org.translate.goog/docs/stable/tensor_attributes.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.dtype)**)** *# overrides the datatype of x\_data*

print**(**f"Random Tensor: **\n** *{*[**x\_rand**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}* **\n**"**)**

Ones Tensor:

tensor([[1, 1],

[1, 1]])

Random Tensor:

tensor([[0.8823, 0.9150],

[0.3829, 0.9593]])

**Với các giá trị ngẫu nhiên hoặc không đổi:**

shapelà một bộ kích thước tensor. Trong các chức năng dưới đây, nó xác định kích thước của tensor đầu ra.

**shape** **=** **(**2**,**3**,)**

[**rand\_tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand)**(shape)**

[**ones\_tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.ones**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones)**(shape)**

[**zeros\_tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.zeros**](https://pytorch-org.translate.goog/docs/stable/generated/torch.zeros.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.zeros)**(shape)**

print**(**f"Random Tensor: **\n** *{*[**rand\_tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}* **\n**"**)**

print**(**f"Ones Tensor: **\n** *{*[**ones\_tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}* **\n**"**)**

print**(**f"Zeros Tensor: **\n** *{*[**zeros\_tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

Random Tensor:

tensor([[0.3904, 0.6009, 0.2566],

[0.7936, 0.9408, 0.1332]])

Ones Tensor:

tensor([[1., 1., 1.],

[1., 1., 1.]])

Zeros Tensor:

tensor([[0., 0., 0.],

[0., 0., 0.]])

## Các thuộc tính của Tenor

Các thuộc tính tensor mô tả hình dạng, kiểu dữ liệu và thiết bị mà chúng được lưu trữ.

[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand)**(**3**,**4**)**

print**(**f"Shape of tensor: *{*[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.shape***}*"**)**

print**(**f"Datatype of tensor: *{*[**tensor.dtype**](https://pytorch-org.translate.goog/docs/stable/tensor_attributes.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.dtype)*}*"**)**

print**(**f"Device tensor is stored on: *{*[**tensor.device**](https://pytorch-org.translate.goog/docs/stable/tensor_attributes.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.device)*}*"**)**

Shape of tensor: torch.Size([3, 4])

Datatype of tensor: torch.float32

Device tensor is stored on: cpu

## Hoạt động trên Tensor

Hơn 100 phép toán tensor, bao gồm số học, đại số tuyến tính, thao tác ma trận (chuyển vị trí, lập chỉ mục, cắt lát), lấy mẫu, v.v. được mô tả toàn diện [tại đây](https://pytorch-org.translate.goog/docs/stable/torch.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) .

Mỗi thao tác này có thể chạy trên GPU (ở tốc độ thường cao hơn so với trên CPU). Nếu bạn đang sử dụng Colab, hãy phân bổ GPU bằng cách chuyển đến Thời gian chạy > Thay đổi loại thời gian chạy > GPU.

Theo mặc định, các tenxơ được tạo trên CPU. Chúng ta cần chuyển một cách rõ ràng các tenxơ sang GPU bằng .tophương pháp (sau khi kiểm tra tính khả dụng của GPU). Hãy nhớ rằng việc sao chép các tenxơ lớn trên các thiết bị có thể tốn kém về thời gian và bộ nhớ!

*# We move our tensor to the GPU if available*

if [**torch.cuda.is\_available**](https://pytorch-org.translate.goog/docs/stable/generated/torch.cuda.is_available.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.cuda.is_available)**():**

[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.to(**"cuda"**)**

Hãy thử một số hoạt động từ danh sách. Nếu bạn đã quen thuộc với API NumPy, bạn sẽ thấy API Tensor rất dễ sử dụng.

**Lập chỉ mục và cắt lát giống như tiêu chuẩn:**

[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.ones**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones)**(**4**,** 4**)**

print**(**f"First row: *{*[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**[**0**]***}*"**)**

print**(**f"First column: *{*[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**[:,** 0**]***}*"**)**

print**(**f"Last column: *{*[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**[...,** **-**1**]***}*"**)**

[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**[:,**1**]** **=** 0

print**(**[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

First row: tensor([1., 1., 1., 1.])

First column: tensor([1., 1., 1., 1.])

Last column: tensor([1., 1., 1., 1.])

tensor([[1., 0., 1., 1.],

[1., 0., 1., 1.],

[1., 0., 1., 1.],

[1., 0., 1., 1.]])

**Nối các tenxơ** Bạn có thể sử dụng torch.catđể nối một chuỗi các tenxơ dọc theo một chiều nhất định. Xem thêm [torch.stack](https://pytorch-org.translate.goog/docs/stable/generated/torch.stack.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) , một toán tử nối tensor khác một cách tinh tế với torch.cat.

[**t1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.cat**](https://pytorch-org.translate.goog/docs/stable/generated/torch.cat.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.cat)**([**[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**],** **dim=**1**)**

print**(**[**t1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

tensor([[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],

[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],

[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.],

[1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1.]])

**Các phép tính toán học**

*# This computes the matrix multiplication between two tensors. y1, y2, y3 will have the same value*

*# ``tensor.T`` returns the transpose of a tensor*

[**y1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **@** [**tensor.T**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

[**y2**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.matmul(**[**tensor.T**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

[**y3**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand\_like**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand_like.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand_like)**(**[**y1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

[**torch.matmul**](https://pytorch-org.translate.goog/docs/stable/generated/torch.matmul.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.matmul)**(**[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**tensor.T**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **out=**[**y3**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

*# This computes the element-wise product. z1, z2, z3 will have the same value*

[**z1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **\*** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

[**z2**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.mul(**[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

[**z3**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand\_like**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand_like.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand_like)**(**[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

[**torch.mul**](https://pytorch-org.translate.goog/docs/stable/generated/torch.mul.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.mul)**(**[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **out=**[**z3**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

tensor([[1., 0., 1., 1.],

[1., 0., 1., 1.],

[1., 0., 1., 1.],

[1., 0., 1., 1.]])

**Tenxơ một phần tử** Nếu bạn có tenxơ một phần tử, chẳng hạn bằng cách tổng hợp tất cả các giá trị của tenxơ thành một giá trị, bạn có thể chuyển đổi nó thành giá trị số Python bằng cách sử dụng item():

[**agg**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.sum()**

**agg\_item** **=** [**agg**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.item()**

print**(agg\_item,** type**(agg\_item))**

12.0 <class 'float'>

**Thao tác tại chỗ** Các thao tác lưu trữ kết quả vào toán hạng được gọi là tại chỗ. Chúng được biểu thị bằng một \_hậu tố. Ví dụ: x.copy\_(y), x.t\_(), sẽ thay đổi x.

print**(**f"*{*[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}* **\n**"**)**

[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.add\_(**5**)**

print**(**[**tensor**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

tensor([[1., 0., 1., 1.],

[1., 0., 1., 1.],

[1., 0., 1., 1.],

[1., 0., 1., 1.]])

tensor([[6., 5., 6., 6.],

[6., 5., 6., 6.],

[6., 5., 6., 6.],

[6., 5., 6., 6.]])

GHI CHÚ

Các hoạt động tại chỗ tiết kiệm một số bộ nhớ, nhưng có thể gặp vấn đề khi tính toán các công cụ phái sinh do mất lịch sử ngay lập tức. Do đó, việc sử dụng chúng không được khuyến khích.

## Cầu nối với NumPy

Các tenxơ trên các mảng CPU và NumPy có thể chia sẻ các vị trí bộ nhớ cơ bản của chúng và việc thay đổi cái này sẽ thay đổi cái kia.

### **Tensor đến mảng NumPy**

[**t**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.ones**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones)**(**5**)**

print**(**f"t: *{*[**t**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

**n** **=** [**t**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.numpy()**

print**(**f"n: *{***n***}*"**)**

t: tensor([1., 1., 1., 1., 1.])

n: [1. 1. 1. 1. 1.]

Một thay đổi trong tensor phản ánh trong mảng NumPy.

[**t**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.add\_(**1**)**

print**(**f"t: *{*[**t**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

print**(**f"n: *{***n***}*"**)**

t: tensor([2., 2., 2., 2., 2.])

n: [2. 2. 2. 2. 2.]

### **Mảng NumPy thành Tenor**

**n** **=** **np.ones(**5**)**

[**t**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.from\_numpy**](https://pytorch-org.translate.goog/docs/stable/generated/torch.from_numpy.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.from_numpy)**(n)**

Các thay đổi trong mảng NumPy phản ánh trong tensor.

**np.add(n,** 1**,** **out=n)**

print**(**f"t: *{*[**t**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

print**(**f"n: *{***n***}*"**)**

t: tensor([2., 2., 2., 2., 2.], dtype=torch.float64)

n: [2. 2. 2. 2. 2.]

# **BỘ DỮ LIỆU & TRÌNH TẢI DỮ LIỆU**

Mã để xử lý các mẫu dữ liệu có thể trở nên lộn xộn và khó bảo trì; lý tưởng nhất là chúng tôi muốn mã tập dữ liệu của mình được tách rời khỏi mã đào tạo mô hình của chúng tôi để có thể đọc và mô đun hóa tốt hơn. PyTorch cung cấp hai dữ liệu gốc: torch.utils.data.DataLoadervà torch.utils.data.Datasetđiều đó cho phép bạn sử dụng bộ dữ liệu được tải sẵn cũng như dữ liệu của riêng bạn. Datasetlưu trữ các mẫu và nhãn tương ứng của chúng, đồng thời DataLoaderbọc một vòng lặp có thể lặp lại xung quanh Datasetđể cho phép dễ dàng truy cập vào các mẫu.

Các thư viện miền PyTorch cung cấp một số bộ dữ liệu được tải sẵn (chẳng hạn như FashionMNIST) để phân lớp con torch.utils.data.Datasetvà triển khai các chức năng dành riêng cho dữ liệu cụ thể. Chúng có thể được sử dụng để tạo nguyên mẫu và đánh giá mô hình của bạn. Bạn có thể tìm thấy chúng ở đây: [Bộ dữ liệu hình ảnh](https://pytorch-org.translate.goog/vision/stable/datasets.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) , [Bộ dữ liệu văn bản](https://pytorch-org.translate.goog/text/stable/datasets.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) và [Bộ dữ liệu âm thanh](https://pytorch-org.translate.goog/audio/stable/datasets.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp)

## Đang tải một tập dữ liệu

Dưới đây là một ví dụ về cách tải bộ dữ liệu [Fashion-MNIST](https://translate.google.com/website?sl=en&tl=vi&hl=vi&client=webapp&u=https://research.zalando.com/project/fashion_mnist/fashion_mnist/) từ TorchVision. Fashion-MNIST là tập dữ liệu các hình ảnh bài viết của Zalando bao gồm 60.000 ví dụ huấn luyện và 10.000 ví dụ kiểm tra. Mỗi ví dụ bao gồm một hình ảnh thang độ xám 28×28 và nhãn được liên kết từ một trong 10 lớp.

Chúng tôi tải [Bộ dữ liệu FashionMNIST](https://pytorch-org.translate.goog/vision/stable/datasets.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#fashion-mnist) với các tham số sau:

* rootlà đường dẫn lưu trữ dữ liệu đào tạo/kiểm tra,
* trainchỉ định tập dữ liệu huấn luyện hoặc kiểm tra,
* download=Truetải xuống dữ liệu từ internet nếu không có sẵn tại root.
* transformvà target\_transformchỉ định chuyển đổi tính năng và nhãn

**import** **torch**

**from** **torch.utils.data** **import** [**Dataset**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.Dataset)

**from** **torchvision** **import** **datasets**

**from** **torchvision.transforms** **import** [**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)

**import** **matplotlib.pyplot** as **plt**

[**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST) **=** [**datasets.FashionMNIST**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**(**

**root=**"data"**,**

**train=True,**

**download=True,**

**transform=**[**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**()**

**)**

[**test\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST) **=** [**datasets.FashionMNIST**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**(**

**root=**"data"**,**

**train=False,**

**download=True,**

**transform=**[**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**()**

**)**

## Lặp lại và trực quan hóa bộ dữ liệu

Chúng tôi có thể lập chỉ mục Datasetsthủ công như một danh sách: training\_data[index]. Chúng tôi sử dụng matplotlibđể trực quan hóa một số mẫu trong dữ liệu đào tạo của mình.

**labels\_map** **=** **{**

0**:** "T-Shirt"**,**

1**:** "Trouser"**,**

2**:** "Pullover"**,**

3**:** "Dress"**,**

4**:** "Coat"**,**

5**:** "Sandal"**,**

6**:** "Shirt"**,**

7**:** "Sneaker"**,**

8**:** "Bag"**,**

9**:** "Ankle Boot"**,**

**}**

**figure** **=** **plt.figure(figsize=(**8**,** 8**))**

**cols,** **rows** **=** 3**,** 3

for **i** **in** range**(**1**,** **cols** **\*** **rows** **+** 1**):**

**sample\_idx** **=** [**torch.randint**](https://pytorch-org.translate.goog/docs/stable/generated/torch.randint.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.randint)**(**len**(**[**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**),** **size=(**1**,)).item()**

[**img**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**[sample\_idx]**

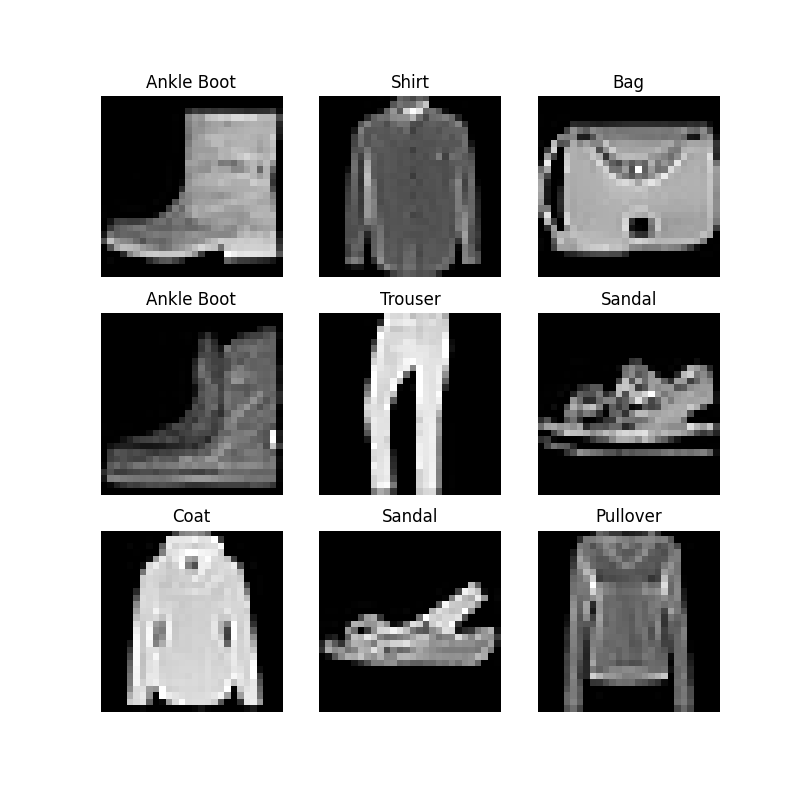
**figure.add\_subplot(rows,** **cols,** **i)**

**plt.title(labels\_map[**[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**])**

**plt.axis(**"off"**)**

**plt.imshow(**[**img**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.squeeze(),** **cmap=**"gray"**)**

**plt.show()**



## Tạo Tập dữ liệu tùy chỉnh cho các tệp của bạn

Lớp Bộ dữ liệu tùy chỉnh phải triển khai ba chức năng: \_\_init\_\_ , \_\_len\_\_ và \_\_getitem\_\_ . Hãy xem triển khai này; hình ảnh FashionMNIST được lưu trữ trong một thư mục img\_dirvà nhãn của chúng được lưu trữ riêng trong tệp CSV annotations\_file.

Trong các phần tiếp theo, chúng ta sẽ chia nhỏ những gì đang xảy ra trong mỗi chức năng này.

**import** **os**

**import** **pandas** as **pd**

**from** **torchvision.io** **import** [**read\_image**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.io.read_image.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.io.read_image)

class **CustomImageDataset(**[**Dataset**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.Dataset)**):**

def \_\_init\_\_**(**self**,** **annotations\_file,** **img\_dir,** **transform=None,** **target\_transform=None):**

self**.img\_labels** **=** **pd.read\_csv(annotations\_file)**

self**.img\_dir** **=** **img\_dir**

self**.transform** **=** **transform**

self**.target\_transform** **=** **target\_transform**

def \_\_len\_\_**(**self**):**

return len**(**self**.img\_labels)**

def \_\_getitem\_\_**(**self**,** **idx):**

**img\_path** **=** **os.path.join(**self**.img\_dir,** self**.img\_labels.iloc[idx,** 0**])**

**image** **=** [**read\_image**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.io.read_image.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.io.read_image)**(img\_path)**

[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** self**.img\_labels.iloc[idx,** 1**]**

if self**.transform:**

**image** **=** self**.transform(image)**

if self**.target\_transform:**

[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** self**.target\_transform(**[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

return **image,** [**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

### **\_\_trong đó\_\_**

Hàm \_\_init\_\_ được chạy một lần khi khởi tạo đối tượng Bộ dữ liệu. Chúng tôi khởi tạo thư mục chứa hình ảnh, tệp chú thích và cả hai phép biến đổi (được đề cập chi tiết hơn trong phần tiếp theo).

Tệp nhãn.csv trông giống như:

**tshirt1.jpg,** 0

**tshirt2.jpg,** 0

**......**

**ankleboot999.jpg,** 9

def \_\_init\_\_**(**self**,** **annotations\_file,** **img\_dir,** **transform=None,** **target\_transform=None):**

self**.img\_labels** **=** **pd.read\_csv(annotations\_file)**

self**.img\_dir** **=** **img\_dir**

self**.transform** **=** **transform**

self**.target\_transform** **=** **target\_transform**

### **\_\_len\_\_**

Hàm \_\_len\_\_ trả về số lượng mẫu trong tập dữ liệu của chúng tôi.

Ví dụ:

def \_\_len\_\_**(**self**):**

return len**(**self**.img\_labels)**

### **\_\_getitem\_\_**

Hàm \_\_getitem\_\_ tải và trả về một mẫu từ tập dữ liệu tại chỉ mục đã cho idx. Dựa trên chỉ mục, nó xác định vị trí của hình ảnh trên đĩa, chuyển đổi vị trí đó thành tensor bằng cách sử dụng read\_image, truy xuất nhãn tương ứng từ dữ liệu csv trong self.img\_labels, gọi các hàm biến đổi trên chúng (nếu có) và trả về hình ảnh tensor và nhãn tương ứng trong một tuple.

def \_\_getitem\_\_**(**self**,** **idx):**

**img\_path** **=** **os.path.join(**self**.img\_dir,** self**.img\_labels.iloc[idx,** 0**])**

**image** **=** [**read\_image**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.io.read_image.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.io.read_image)**(img\_path)**

[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** self**.img\_labels.iloc[idx,** 1**]**

if self**.transform:**

**image** **=** self**.transform(image)**

if self**.target\_transform:**

[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** self**.target\_transform(**[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

return **image,** [**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

## Chuẩn bị dữ liệu của bạn để đào tạo với DataLoaders

Truy Datasetxuất các tính năng và nhãn của tập dữ liệu của chúng tôi mỗi lần một mẫu. Trong khi đào tạo một mô hình, chúng tôi thường muốn chuyển các mẫu trong “minibatch”, chia sẻ lại dữ liệu ở mọi giai đoạn để giảm tình trạng thừa mô hình và sử dụng Python multiprocessingđể tăng tốc độ truy xuất dữ liệu.

DataLoaderlà một lần lặp tóm tắt sự phức tạp này cho chúng tôi trong một API dễ dàng.

**from** **torch.utils.data** **import** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)

[**train\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader) **=** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**(**[**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**,** **batch\_size=**64**,** **shuffle=True)**

[**test\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader) **=** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**(**[**test\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**,** **batch\_size=**64**,** **shuffle=True)**

## Lặp lại thông qua DataLoader

Chúng tôi đã tải tập dữ liệu đó vào DataLoadervà có thể lặp qua tập dữ liệu nếu cần. Mỗi lần lặp bên dưới trả về một lô train\_featuresvà train\_labels(chứa batch\_size=64các tính năng và nhãn tương ứng). Bởi vì chúng tôi đã chỉ định shuffle=True, nên sau khi chúng tôi lặp lại tất cả các lô, dữ liệu sẽ bị xáo trộn (để kiểm soát chi tiết hơn đối với thứ tự tải dữ liệu, hãy xem [Bộ lấy mẫu](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#data-loading-order-and-sampler) ).

*# Display image and label.*

[**train\_features**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**train\_labels**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** next**(**iter**(**[**train\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**))**

print**(**f"Feature batch shape: *{*[**train\_features**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.size()***}*"**)**

print**(**f"Labels batch shape: *{*[**train\_labels**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.size()***}*"**)**

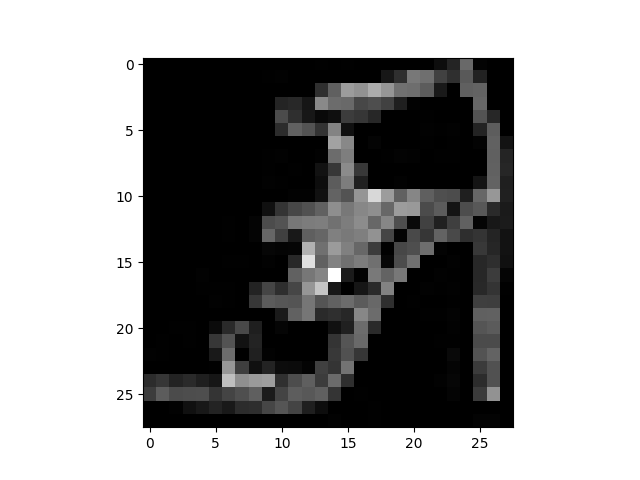
[**img**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**train\_features**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**[**0**].squeeze()**

[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**train\_labels**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**[**0**]**

**plt.imshow(**[**img**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** **cmap=**"gray"**)**

**plt.show()**

print**(**f"Label: *{*[**label**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**



Feature batch shape: torch.Size([64, 1, 28, 28])

Labels batch shape: torch.Size([64])

Label: 5

# **4.BIẾN ĐỔI**

Dữ liệu không phải lúc nào cũng ở dạng được xử lý cuối cùng cần thiết để huấn luyện các thuật toán học máy. Chúng tôi sử dụng **các phép biến đổi** để thực hiện một số thao tác với dữ liệu và làm cho nó phù hợp để đào tạo.

Tất cả bộ dữ liệu TorchVision đều có hai tham số - transformđể sửa đổi các tính năng và target\_transformsửa đổi nhãn - chấp nhận các lệnh gọi có chứa logic chuyển đổi. Mô -đun [torchvision.transforms](https://pytorch-org.translate.goog/vision/stable/transforms.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) cung cấp một số biến đổi thường được sử dụng ngay lập tức.

Các tính năng của FashionMNIST ở định dạng Hình ảnh PIL và các nhãn là số nguyên. Để đào tạo, chúng tôi cần các tính năng như tenxơ được chuẩn hóa và nhãn là tenxơ được mã hóa một lần. Để thực hiện các phép biến đổi này, chúng tôi sử dụng ToTensorvà Lambda.

**import** **torch**

**from** **torchvision** **import** **datasets**

**from** **torchvision.transforms** **import** [**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**,** [**Lambda**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.Lambda.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.Lambda)

[**ds**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST) **=** [**datasets.FashionMNIST**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**(**

**root=**"data"**,**

**train=True,**

**download=True,**

**transform=**[**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**(),**

[**target\_transform**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.Lambda.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.Lambda)**=**[**Lambda**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.Lambda.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.Lambda)**(**lambda **y:** [**torch.zeros**](https://pytorch-org.translate.goog/docs/stable/generated/torch.zeros.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.zeros)**(**10**,** **dtype=**[**torch.float**](https://pytorch-org.translate.goog/docs/stable/tensor_attributes.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.dtype)**).scatter\_(**0**,** [**torch.tensor**](https://pytorch-org.translate.goog/docs/stable/generated/torch.tensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.tensor)**(y),** **value=**1**))**

**)**

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-idx3-ubyte.gz

0%| | 0/26421880 [00:00<?, ?it/s]

0%| | 65536/26421880 [00:00<01:12, 361065.74it/s]

1%| | 229376/26421880 [00:00<00:38, 677544.94it/s]

3%|3 | 917504/26421880 [00:00<00:09, 2590764.52it/s]

7%|7 | 1867776/26421880 [00:00<00:06, 3905731.07it/s]

18%|#7 | 4751360/26421880 [00:00<00:02, 10395582.99it/s]

25%|##5 | 6619136/26421880 [00:00<00:01, 11899455.27it/s]

31%|###1 | 8290304/26421880 [00:01<00:01, 12059214.93it/s]

43%|####2 | 11239424/26421880 [00:01<00:00, 16345464.64it/s]

50%|####9 | 13172736/26421880 [00:01<00:00, 16125603.81it/s]

56%|#####6 | 14909440/26421880 [00:01<00:00, 16417974.53it/s]

63%|######3 | 16711680/26421880 [00:01<00:00, 15777922.42it/s]

75%|#######5 | 19824640/26421880 [00:01<00:00, 19591474.75it/s]

83%|########2 | 21889024/26421880 [00:01<00:00, 18537222.05it/s]

90%|######### | 23822336/26421880 [00:01<00:00, 17193512.16it/s]

100%|##########| 26421880/26421880 [00:01<00:00, 13793965.78it/s]

Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-idx1-ubyte.gz

0%| | 0/29515 [00:00<?, ?it/s]

100%|##########| 29515/29515 [00:00<00:00, 325507.89it/s]

Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz

0%| | 0/4422102 [00:00<?, ?it/s]

1%|1 | 65536/4422102 [00:00<00:12, 360833.50it/s]

5%|5 | 229376/4422102 [00:00<00:06, 679408.53it/s]

21%|## | 917504/4422102 [00:00<00:01, 2606987.90it/s]

39%|###9 | 1736704/4422102 [00:00<00:00, 3569611.33it/s]

97%|#########7| 4292608/4422102 [00:00<00:00, 9336380.52it/s]

100%|##########| 4422102/4422102 [00:00<00:00, 5970037.26it/s]

Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz

0%| | 0/5148 [00:00<?, ?it/s]

100%|##########| 5148/5148 [00:00<00:00, 52792853.28it/s]

Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw

## ToTensor()

[ToTensor](https://pytorch-org.translate.goog/vision/stable/transforms.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor) chuyển đổi hình ảnh PIL hoặc NumPy ndarraythành tệp FloatTensor. và chia tỷ lệ các giá trị cường độ điểm ảnh của hình ảnh trong phạm vi [0., 1.]

## Biến đổi Lambda

Biến đổi Lambda áp dụng bất kỳ hàm lambda nào do người dùng xác định. Ở đây, chúng tôi định nghĩa một hàm để biến số nguyên thành tenxơ được mã hóa một lần nóng. Trước tiên, nó tạo một tenxơ bằng 0 có kích thước 10 (số lượng nhãn trong tập dữ liệu của chúng tôi) và gọi hàm [scatter\_](https://pytorch-org.translate.goog/docs/stable/generated/torch.Tensor.scatter_.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) gán a value=1trên chỉ mục như được cho bởi nhãn y.

[**target\_transform**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.Lambda.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.Lambda) **=** [**Lambda**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.Lambda.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.Lambda)**(**lambda **y:** [**torch.zeros**](https://pytorch-org.translate.goog/docs/stable/generated/torch.zeros.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.zeros)**(**

10**,** **dtype=**[**torch.float**](https://pytorch-org.translate.goog/docs/stable/tensor_attributes.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.dtype)**).scatter\_(dim=**0**,** **index=**[**torch.tensor**](https://pytorch-org.translate.goog/docs/stable/generated/torch.tensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.tensor)**(y),** **value=**1**))**

# **5.XÂY DỰNG MẠNG LƯỚI THẦN KINH**

Mạng nơ-ron bao gồm các lớp/mô-đun thực hiện các thao tác trên dữ liệu. Không gian tên [torch.nn](https://pytorch-org.translate.goog/docs/stable/nn.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) cung cấp tất cả các khối xây dựng mà bạn cần để xây dựng mạng thần kinh của riêng mình. Mỗi mô-đun trong PyTorch phân lớp [nn.Module](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) . Mạng thần kinh là một mô-đun bao gồm các mô-đun (lớp) khác. Cấu trúc lồng nhau này cho phép xây dựng và quản lý các kiến ​​trúc phức tạp một cách dễ dàng.

Trong các phần tiếp theo, chúng ta sẽ xây dựng một mạng thần kinh để phân loại hình ảnh trong bộ dữ liệu FashionMNIST.

**import** **os**

**import** **torch**

**from** **torch** **import** **nn**

**from** **torch.utils.data** **import** **DataLoader**

**from** **torchvision** **import** **datasets,** **transforms**

## Nhận thiết bị để đào tạo

Chúng tôi muốn có thể đào tạo mô hình của mình trên bộ tăng tốc phần cứng như GPU hoặc MPS, nếu có. Hãy kiểm tra xem có [torch.cuda](https://pytorch-org.translate.goog/docs/stable/notes/cuda.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) hoặc [torch.backends.mps](https://pytorch-org.translate.goog/docs/stable/notes/mps.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) không, nếu không thì chúng ta sử dụng CPU.

**device** **=** **(**

"cuda"

if [**torch.cuda.is\_available**](https://pytorch-org.translate.goog/docs/stable/generated/torch.cuda.is_available.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.cuda.is_available)**()**

else "mps"

if [**torch.backends.mps.is\_available**](https://pytorch-org.translate.goog/docs/stable/backends.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.backends.mps.is_available)**()**

else "cpu"

**)**

print**(**f"Using *{***device***}* device"**)**

Using cuda device

## Xác định lớp

Chúng tôi xác định mạng thần kinh của mình bằng cách phân lớp nn.Modulevà khởi tạo các lớp mạng thần kinh trong \_\_init\_\_. Mỗi nn.Modulelớp con thực hiện các thao tác trên dữ liệu đầu vào trong forwardphương thức.

class **NeuralNetwork(**[**nn.Module**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module)**):**

def \_\_init\_\_**(**self**):**

super**().**\_\_init\_\_**()**

self**.**[**flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten) **=** [**nn.Flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten)**()**

self**.linear\_relu\_stack** **=** [**nn.Sequential**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Sequential.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Sequential)**(**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**28**\***28**,** 512**),**

[**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**(),**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**512**,** 512**),**

[**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**(),**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**512**,** 10**),**

**)**

def **forward(**self**,** **x):**

**x** **=** self**.**[**flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten)**(x)**

[**logits**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** self**.linear\_relu\_stack(x)**

return [**logits**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

Chúng tôi tạo một thể hiện của NeuralNetworkvà di chuyển nó đến devicevà in cấu trúc của nó.

**model** **=** [**NeuralNetwork**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module)**().to(device)**

print**(model)**

NeuralNetwork(

(flatten): Flatten(start\_dim=1, end\_dim=-1)

(linear\_relu\_stack): Sequential(

(0): Linear(in\_features=784, out\_features=512, bias=True)

(1): ReLU()

(2): Linear(in\_features=512, out\_features=512, bias=True)

(3): ReLU()

(4): Linear(in\_features=512, out\_features=10, bias=True)

)

)

Để sử dụng mô hình, chúng tôi truyền cho nó dữ liệu đầu vào. Điều này thực thi mô hình forward, cùng với một số [hoạt động nền](https://translate.google.com/website?sl=en&tl=vi&hl=vi&client=webapp&u=https://github.com/pytorch/pytorch/blob/270111b7b611d174967ed204776985cefca9c144/torch/nn/modules/module.py%23L866) . Đừng gọi model.forward()trực tiếp!

Gọi mô hình trên đầu vào trả về một tenxơ 2 chiều với dim=0 tương ứng với mỗi đầu ra gồm 10 giá trị dự đoán thô cho mỗi lớp và dim=1 tương ứng với các giá trị riêng lẻ của từng đầu ra. Chúng tôi nhận được xác suất dự đoán bằng cách chuyển nó qua một phiên bản của nn.Softmaxmô-đun.

[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand)**(**1**,** 28**,** 28**,** **device=device)**

[**logits**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** **model(**[**X**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

[**pred\_probab**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**nn.Softmax**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Softmax.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Softmax)**(dim=**1**)(**[**logits**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

[**y\_pred**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**pred\_probab**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.argmax(**1**)**

print**(**f"Predicted class: *{*[**y\_pred**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

Predicted class: tensor([7], device='cuda:0')

## Lớp mô hình

Hãy chia nhỏ các lớp trong mô hình FashionMNIST. Để minh họa, chúng tôi sẽ lấy một lô nhỏ mẫu gồm 3 hình ảnh có kích thước 28x28 và xem điều gì sẽ xảy ra với nó khi chúng tôi chuyển nó qua mạng.

[**input\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand)**(**3**,**28**,**28**)**

print**(**[**input\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.size())**

torch.Size([3, 28, 28])

### **nn.Làm phẳng**

Chúng tôi khởi tạo lớp [nn.Flatten](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) để chuyển đổi từng hình ảnh 2D 28x28 thành một mảng liền kề có giá trị 784 pixel ( kích thước minibatch (ở độ mờ = 0) được duy trì).

[**flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten) **=** [**nn.Flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten)**()**

[**flat\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten)**(**[**input\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

print**(**[**flat\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.size())**

torch.Size([3, 784])

### **nn.Tuyến tính**

Lớp [tuyến tính](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) là một mô-đun áp dụng phép biến đổi tuyến tính trên đầu vào bằng cách sử dụng các trọng số và độ chệch được lưu trữ của nó.

[**layer1**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear) **=** [**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(in\_features=**28**\***28**,** **out\_features=**20**)**

[**hidden1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**layer1**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**[**flat\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

print**(**[**hidden1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.size())**

torch.Size([3, 20])

### **nn.ReLU**

Kích hoạt phi tuyến tính là thứ tạo ra ánh xạ phức tạp giữa đầu vào và đầu ra của mô hình. Chúng được áp dụng sau các phép biến đổi tuyến tính để giới thiệu tính phi tuyến tính , giúp mạng lưới thần kinh tìm hiểu nhiều hiện tượng khác nhau.

Trong mô hình này, chúng tôi sử dụng [nn.ReLU](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) giữa các lớp tuyến tính của mình, nhưng có các kích hoạt khác để giới thiệu tính phi tuyến tính trong mô hình của bạn.

print**(**f"Before ReLU: *{*[**hidden1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}***\n\n**"**)**

[**hidden1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**()(**[**hidden1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

print**(**f"After ReLU: *{*[**hidden1**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

Before ReLU: tensor([[ 0.4158, -0.0130, -0.1144, 0.3960, 0.1476, -0.0690, -0.0269, 0.2690,

0.1353, 0.1975, 0.4484, 0.0753, 0.4455, 0.5321, -0.1692, 0.4504,

0.2476, -0.1787, -0.2754, 0.2462],

[ 0.2326, 0.0623, -0.2984, 0.2878, 0.2767, -0.5434, -0.5051, 0.4339,

0.0302, 0.1634, 0.5649, -0.0055, 0.2025, 0.4473, -0.2333, 0.6611,

0.1883, -0.1250, 0.0820, 0.2778],

[ 0.3325, 0.2654, 0.1091, 0.0651, 0.3425, -0.3880, -0.0152, 0.2298,

0.3872, 0.0342, 0.8503, 0.0937, 0.1796, 0.5007, -0.1897, 0.4030,

0.1189, -0.3237, 0.2048, 0.4343]], grad\_fn=<AddmmBackward0>)

After ReLU: tensor([[0.4158, 0.0000, 0.0000, 0.3960, 0.1476, 0.0000, 0.0000, 0.2690, 0.1353,

0.1975, 0.4484, 0.0753, 0.4455, 0.5321, 0.0000, 0.4504, 0.2476, 0.0000,

0.0000, 0.2462],

[0.2326, 0.0623, 0.0000, 0.2878, 0.2767, 0.0000, 0.0000, 0.4339, 0.0302,

0.1634, 0.5649, 0.0000, 0.2025, 0.4473, 0.0000, 0.6611, 0.1883, 0.0000,

0.0820, 0.2778],

[0.3325, 0.2654, 0.1091, 0.0651, 0.3425, 0.0000, 0.0000, 0.2298, 0.3872,

0.0342, 0.8503, 0.0937, 0.1796, 0.5007, 0.0000, 0.4030, 0.1189, 0.0000,

0.2048, 0.4343]], grad\_fn=<ReluBackward0>)

### **nn.Tuần tự**

[nn.Sequential](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Sequential.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) là một bộ chứa các mô-đun được sắp xếp theo thứ tự. Dữ liệu được truyền qua tất cả các mô-đun theo thứ tự như đã xác định. Bạn có thể sử dụng các vùng chứa tuần tự để kết hợp một mạng nhanh như seq\_modules.

[**seq\_modules**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Sequential.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Sequential) **=** [**nn.Sequential**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Sequential.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Sequential)**(**

[**flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten)**,**

[**layer1**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**,**

[**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**(),**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**20**,** 10**)**

**)**

[**input\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.rand**](https://pytorch-org.translate.goog/docs/stable/generated/torch.rand.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.rand)**(**3**,**28**,**28**)**

[**logits**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**seq\_modules**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Sequential.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Sequential)**(**[**input\_image**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

### **nn.Softmax**

Lớp tuyến tính cuối cùng của mạng thần kinh trả về nhật ký - giá trị thô trong [-infty, infty] - được chuyển đến mô-đun [nn.Softmax](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Softmax.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) . Nhật ký được chia tỷ lệ thành các giá trị [0, 1] đại diện cho xác suất dự đoán của mô hình cho mỗi lớp. dimtham số cho biết thứ nguyên theo đó các giá trị phải có tổng bằng 1.

[**softmax**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Softmax.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Softmax) **=** [**nn.Softmax**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Softmax.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Softmax)**(dim=**1**)**

[**pred\_probab**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**softmax**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Softmax.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Softmax)**(**[**logits**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

## Thông số mô hình

Nhiều lớp bên trong mạng thần kinh được tham số hóa , nghĩa là có các trọng số và độ lệch liên quan được tối ưu hóa trong quá trình đào tạo. Phân lớp nn.Moduletự động theo dõi tất cả các trường được xác định bên trong đối tượng mô hình của bạn và làm cho tất cả các tham số có thể truy cập được bằng mô hình parameters()hoặc named\_parameters()phương thức của bạn.

Trong ví dụ này, chúng tôi lặp lại từng tham số và in kích thước của tham số cũng như bản xem trước các giá trị của tham số đó.

print**(**f"Model structure: *{***model***}***\n\n**"**)**

for **name,** [**param**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.parameter.Parameter.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.parameter.Parameter) **in** [**model.named\_parameters**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.named_parameters)**():**

print**(**f"Layer: *{***name***}* | Size: *{*[**param**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.parameter.Parameter.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.parameter.Parameter)**.size()***}* | Values : *{*[**param**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.parameter.Parameter.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.parameter.Parameter)**[:**2**]***}* **\n**"**)**

Model structure: NeuralNetwork(

(flatten): Flatten(start\_dim=1, end\_dim=-1)

(linear\_relu\_stack): Sequential(

(0): Linear(in\_features=784, out\_features=512, bias=True)

(1): ReLU()

(2): Linear(in\_features=512, out\_features=512, bias=True)

(3): ReLU()

(4): Linear(in\_features=512, out\_features=10, bias=True)

)

)

Layer: linear\_relu\_stack.0.weight | Size: torch.Size([512, 784]) | Values : tensor([[ 0.0273, 0.0296, -0.0084, ..., -0.0142, 0.0093, 0.0135],

[-0.0188, -0.0354, 0.0187, ..., -0.0106, -0.0001, 0.0115]],

device='cuda:0', grad\_fn=<SliceBackward0>)

Layer: linear\_relu\_stack.0.bias | Size: torch.Size([512]) | Values : tensor([-0.0155, -0.0327], device='cuda:0', grad\_fn=<SliceBackward0>)

# **6. KHÁC BIỆT HÓA TỰ ĐỘNG VỚITORCH.AUTOGRAD**

Khi đào tạo các mạng thần kinh, thuật toán được sử dụng thường xuyên nhất là **lan truyền ngược** . Trong thuật toán này, các tham số (trọng số của mô hình) được điều chỉnh theo độ **dốc** của hàm mất mát đối với tham số đã cho.

Để tính toán các độ dốc đó, PyTorch có một công cụ phân biệt tích hợp có tên là torch.autograd. Nó hỗ trợ tự động tính toán độ dốc cho bất kỳ biểu đồ tính toán nào.

Hãy xem xét mạng nơ-ron một lớp đơn giản nhất, với đầu vào x, tham số wvà bvà một số hàm mất mát. Nó có thể được định nghĩa trong PyTorch theo cách sau:

**import** **torch**

[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.ones**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones)**(**5**)** *# input tensor*

[**y**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.zeros**](https://pytorch-org.translate.goog/docs/stable/generated/torch.zeros.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.zeros)**(**3**)** *# expected output*

[**w**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.randn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.randn.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.randn)**(**5**,** 3**,** **requires\_grad=True)**

[**b**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.randn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.randn.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.randn)**(**3**,** **requires\_grad=True)**

[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.matmul**](https://pytorch-org.translate.goog/docs/stable/generated/torch.matmul.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.matmul)**(**[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**w**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)+**[**b**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

[**loss**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.nn.functional.binary\_cross\_entropy\_with\_logits**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.functional.binary_cross_entropy_with_logits.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.functional.binary_cross_entropy_with_logits)**(**[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**y**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

## Tenxơ, hàm và đồ thị tính toán

**Mã này xác định biểu đồ tính toán** sau :

A picture containing black, darkness

Description automatically generated

Trong mạng này, wvà blà **các tham số** mà chúng ta cần tối ưu hóa. Vì vậy, chúng ta cần có khả năng tính toán độ dốc của hàm mất mát đối với các biến đó. Để làm được điều đó, chúng tôi đặt requires\_gradthuộc tính của các tenxơ đó.

GHI CHÚ

Bạn có thể đặt giá trị requires\_gradkhi tạo tensor hoặc sau này bằng cách sử dụng x.requires\_grad\_(True)phương thức.

Hàm mà chúng ta áp dụng cho tenxơ để xây dựng đồ thị tính toán trên thực tế là một đối tượng của lớp Function. Đối tượng này biết cách tính toán hàm theo hướng thuận và cả cách tính đạo hàm của nó trong bước lan truyền ngược . Một tham chiếu đến hàm lan truyền ngược được lưu trữ trong grad\_fnthuộc tính của một tensor. Bạn có thể tìm thêm thông tin Function [trong tài liệu](https://pytorch-org.translate.goog/docs/stable/autograd.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#function) .

print**(**f"Gradient function for z = *{*[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.grad\_fn***}*"**)**

print**(**f"Gradient function for loss = *{*[**loss**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.grad\_fn***}*"**)**

Gradient function for z = <AddBackward0 object at 0x7fed71a95fc0>

Gradient function for loss = <BinaryCrossEntropyWithLogitsBackward0 object at 0x7fed71a95720>

## Độ dốc điện toán

Để tối ưu hóa trọng số của các tham số trong mạng thần kinh, chúng ta cần tính toán đạo hàm của hàm mất mát đối với các tham số, cụ thể là chúng ta cần và dưới một số giá trị cố định của và . Để tính toán các dẫn xuất đó, chúng tôi gọi , sau đó truy xuất các giá trị từ và :∂tôi���∂�∂ *w*∂ *l oss*​∂tôi���∂�∂ *b*∂ *l oss*​xyloss.backward()w.gradb.grad

[**loss.backward**](https://pytorch-org.translate.goog/docs/stable/generated/torch.Tensor.backward.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor.backward)**()**

print**(**[**w.grad**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

print**(**[**b.grad**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)**

tensor([[0.3313, 0.0626, 0.2530],

[0.3313, 0.0626, 0.2530],

[0.3313, 0.0626, 0.2530],

[0.3313, 0.0626, 0.2530],

[0.3313, 0.0626, 0.2530]])

tensor([0.3313, 0.0626, 0.2530])

GHI CHÚ

* Chúng tôi chỉ có thể lấy gradcác thuộc tính cho các nút lá của biểu đồ tính toán, có requires\_gradthuộc tính được đặt thành True. Đối với tất cả các nút khác trong biểu đồ của chúng tôi, độ dốc sẽ không khả dụng.
* Chúng tôi chỉ có thể thực hiện phép tính độ dốc bằng cách sử dụng backwardmột lần trên biểu đồ nhất định vì lý do hiệu suất. Nếu chúng ta cần thực hiện một số backwardcuộc gọi trên cùng một biểu đồ, chúng ta cần chuyển retain\_graph=Trueđến backwardcuộc gọi.

## Vô hiệu hóa tính năng theo dõi độ dốc

Theo mặc định, tất cả các tenxơ có requires\_grad=Trueđang theo dõi lịch sử tính toán của chúng và hỗ trợ tính toán độ dốc. Tuy nhiên, có một số trường hợp chúng ta không cần phải làm như vậy, ví dụ như khi chúng ta đã huấn luyện mô hình và chỉ muốn áp dụng nó cho một số dữ liệu đầu vào, tức là chúng ta chỉ muốn thực hiện các tính toán chuyển tiếp thông qua mạng . Chúng tôi có thể ngừng theo dõi các tính toán bằng cách bao quanh mã tính toán của mình bằng torch.no\_grad()khối:

[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.matmul**](https://pytorch-org.translate.goog/docs/stable/generated/torch.matmul.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.matmul)**(**[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**w**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)+**[**b**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

print**(**[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.requires\_grad)**

with [**torch.no\_grad**](https://pytorch-org.translate.goog/docs/stable/generated/torch.no_grad.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.no_grad)**():**

[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.matmul**](https://pytorch-org.translate.goog/docs/stable/generated/torch.matmul.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.matmul)**(**[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**w**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)+**[**b**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

print**(**[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.requires\_grad)**

True

False

Một cách khác để đạt được kết quả tương tự là sử dụng detach()phương thức trên tensor:

[**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.matmul**](https://pytorch-org.translate.goog/docs/stable/generated/torch.matmul.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.matmul)**(**[**x**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**,** [**w**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**)+**[**b**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)

[**z\_det**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**z**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.detach()**

print**(**[**z\_det**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.requires\_grad)**

False

Có nhiều lý do bạn có thể muốn tắt tính năng theo dõi độ dốc:

* Để đánh dấu một số tham số trong mạng thần kinh của bạn là **tham số cố định** .
* Để **tăng tốc độ tính toán** khi bạn chỉ thực hiện chuyển tiếp, bởi vì tính toán trên tensors không theo dõi độ dốc sẽ hiệu quả hơn.

## Tìm hiểu thêm về đồ thị tính toán

Về mặt khái niệm, autograd giữ một bản ghi dữ liệu (tenxơ) và tất cả các hoạt động đã thực hiện (cùng với các tenxơ mới thu được) trong biểu đồ tuần hoàn có hướng (DAG) bao gồm các đối tượng [Hàm](https://pytorch-org.translate.goog/docs/stable/autograd.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.autograd.Function) . Trong DAG này, lá là tenxơ đầu vào, rễ là tenxơ đầu ra. Bằng cách truy tìm biểu đồ này từ gốc đến lá, bạn có thể tự động tính toán độ dốc bằng quy tắc dây chuyền.

Trong một lần chuyển tiếp, autograd thực hiện đồng thời hai việc:

* chạy thao tác được yêu cầu để tính toán tensor kết quả
* duy trì chức năng gradient của hoạt động trong DAG.

Đường chuyền ngược bắt đầu khi .backward()được gọi trên gốc DAG. autogradsau đó:

* tính toán độ dốc từ mỗi .grad\_fn,
* tích lũy chúng trong .gradthuộc tính của tensor tương ứng
* sử dụng quy tắc dây chuyền, truyền tất cả các con đường đến các tenxơ lá.

GHI CHÚ

**DAG là động trong PyTorch** Một điều quan trọng cần lưu ý là biểu đồ được tạo lại từ đầu; sau mỗi .backward()cuộc gọi, autograd bắt đầu điền một biểu đồ mới. Đây chính xác là những gì cho phép bạn sử dụng các câu lệnh luồng điều khiển trong mô hình của mình; bạn có thể thay đổi hình dạng, kích thước và thao tác ở mỗi lần lặp lại nếu cần.

## Đọc tùy chọn: Tensor Gradients và Jacobian Products

Trong nhiều trường hợp, chúng ta có một hàm mất mát vô hướng và chúng ta cần tính toán độ dốc đối với một số tham số. Tuy nhiên, có những trường hợp khi hàm đầu ra là một tenxơ tùy ý. Trong trường hợp này, PyTorch cho phép bạn tính toán cái gọi là **tích Jacobian** chứ không phải độ dốc thực tế.

Đối với hàm vectơ , trong đó và , độ dốc của đối với được đưa ra bởi **ma trận Jacobian** :�⃗=�(�⃗)*y*​=*f*(*x*)�⃗=⟨�1,…,��⟩*x*=⟨ *x*1​,…,*xN*​⟩�⃗=⟨�1,…,�tôi⟩*y*​=⟨ *y*1​,…,*ytôi*​⟩�⃗*y*​�⃗*x*

�=(∂�1∂�1⋯∂�1∂��⋮⋱⋮∂�tôi∂�1⋯∂�tôi∂��)*J*=​∂ *x*1​∂ *y*1​​⋮∂ *x*1​∂ *ytôi*​​​⋯⋱⋯​∂ *xN*​∂ *y*1​​⋮∂ *xN*​∂ *ytôi*​​​​

Thay vì tự tính toán ma trận Jacobian, PyTorch cho phép bạn tính toán **Sản phẩm Jacobian** cho một vectơ đầu vào nhất định . Điều này đạt được bằng cách gọi với làm đối số. Kích thước của phải giống với kích thước của tenxơ ban đầu mà chúng ta muốn tính tích:��⋅�*vt*⋅*J*�=(�1…�tôi)*v*=( *v*1​…*vtôi*​)backward�*v*�*v*

[**inp**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** [**torch.eye**](https://pytorch-org.translate.goog/docs/stable/generated/torch.eye.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.eye)**(**4**,** 5**,** **requires\_grad=True)**

[**out**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor) **=** **(**[**inp**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**+**1**).pow(**2**).t()**

[**out.backward**](https://pytorch-org.translate.goog/docs/stable/generated/torch.Tensor.backward.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor.backward)**(**[**torch.ones\_like**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones_like.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones_like)**(**[**out**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**),** **retain\_graph=True)**

print**(**f"First call**\n***{*[**inp.grad**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

[**out.backward**](https://pytorch-org.translate.goog/docs/stable/generated/torch.Tensor.backward.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor.backward)**(**[**torch.ones\_like**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones_like.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones_like)**(**[**out**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**),** **retain\_graph=True)**

print**(**f"**\n**Second call**\n***{*[**inp.grad**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

[**inp.grad**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**.zero\_()**

[**out.backward**](https://pytorch-org.translate.goog/docs/stable/generated/torch.Tensor.backward.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor.backward)**(**[**torch.ones\_like**](https://pytorch-org.translate.goog/docs/stable/generated/torch.ones_like.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.ones_like)**(**[**out**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)**),** **retain\_graph=True)**

print**(**f"**\n**Call after zeroing gradients**\n***{*[**inp.grad**](https://pytorch-org.translate.goog/docs/stable/tensors.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.Tensor)*}*"**)**

First call

tensor([[4., 2., 2., 2., 2.],

[2., 4., 2., 2., 2.],

[2., 2., 4., 2., 2.],

[2., 2., 2., 4., 2.]])

Second call

tensor([[8., 4., 4., 4., 4.],

[4., 8., 4., 4., 4.],

[4., 4., 8., 4., 4.],

[4., 4., 4., 8., 4.]])

Call after zeroing gradients

tensor([[4., 2., 2., 2., 2.],

[2., 4., 2., 2., 2.],

[2., 2., 4., 2., 2.],

[2., 2., 2., 4., 2.]])

Lưu ý rằng khi chúng ta gọi backwardlần thứ hai với cùng một đối số, giá trị của gradient sẽ khác. Điều này xảy ra bởi vì khi thực hiện backwardlan truyền, PyTorch **tích lũy các gradient** , tức là giá trị của các gradient tính toán được thêm vào thuộc gradtính của tất cả các nút lá của đồ thị tính toán. Nếu bạn muốn tính toán độ dốc thích hợp, bạn cần loại bỏ gradthuộc tính trước đó. Trong quá trình đào tạo thực tế, trình tối ưu hóa giúp chúng tôi thực hiện việc này.

GHI CHÚ

Trước đây chúng ta gọi backward()hàm không có tham số. Về cơ bản, điều này tương đương với việc gọi backward(torch.tensor(1.0)), đây là một cách hữu ích để tính toán độ dốc trong trường hợp hàm có giá trị vô hướng, chẳng hạn như mất mát trong quá trình huấn luyện mạng thần kinh.

# **7.TỐI ƯU HÓA THÔNG SỐ MÔ HÌNH**

Bây giờ chúng ta đã có một mô hình và dữ liệu, đã đến lúc đào tạo, xác thực và kiểm tra mô hình của chúng ta bằng cách tối ưu hóa các tham số của mô hình trên dữ liệu của chúng ta. Đào tạo một mô hình là một quá trình lặp đi lặp lại; trong mỗi lần lặp, mô hình đưa ra dự đoán về đầu ra, tính toán lỗi trong dự đoán của nó ( loss ), thu thập đạo hàm của lỗi đối với các tham số của nó (như chúng ta đã thấy trong phần [trước](https://pytorch-org.translate.goog/tutorials/beginner/basics/autograd_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) ) và **tối ưu hóa** các tham số này bằng cách sử dụng hệ số giảm dần . Để có hướng dẫn chi tiết hơn về quy trình này, hãy xem video này về [lan truyền ngược từ 3Blue1Brown](https://translate.google.com/website?sl=en&tl=vi&hl=vi&client=webapp&u=https://www.youtube.com/watch?v%3DtIeHLnjs5U8) .

## Mã điều kiện tiên quyết

Chúng tôi tải mã từ các phần trước trên [Bộ dữ liệu & Trình tải dữ liệu](https://pytorch-org.translate.goog/tutorials/beginner/basics/data_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) và [Mô hình xây dựng](https://pytorch-org.translate.goog/tutorials/beginner/basics/buildmodel_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) .

**import** **torch**

**from** **torch** **import** **nn**

**from** **torch.utils.data** **import** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)

**from** **torchvision** **import** **datasets**

**from** **torchvision.transforms** **import** [**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)

[**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST) **=** [**datasets.FashionMNIST**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**(**

**root=**"data"**,**

**train=True,**

**download=True,**

**transform=**[**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**()**

**)**

[**test\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST) **=** [**datasets.FashionMNIST**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**(**

**root=**"data"**,**

**train=False,**

**download=True,**

**transform=**[**ToTensor**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.transforms.ToTensor.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.transforms.ToTensor)**()**

**)**

[**train\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader) **=** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**(**[**training\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**,** **batch\_size=**64**)**

[**test\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader) **=** [**DataLoader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**(**[**test\_data**](https://pytorch-org.translate.goog/vision/stable/generated/torchvision.datasets.FashionMNIST.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.datasets.FashionMNIST)**,** **batch\_size=**64**)**

class **NeuralNetwork(**[**nn.Module**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module)**):**

def \_\_init\_\_**(**self**):**

super**(**[**NeuralNetwork**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module)**,** self**).**\_\_init\_\_**()**

self**.flatten** **=** [**nn.Flatten**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Flatten.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Flatten)**()**

self**.linear\_relu\_stack** **=** [**nn.Sequential**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Sequential.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Sequential)**(**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**28**\***28**,** 512**),**

[**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**(),**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**512**,** 512**),**

[**nn.ReLU**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.ReLU.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.ReLU)**(),**

[**nn.Linear**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Linear.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Linear)**(**512**,** 10**),**

**)**

def **forward(**self**,** **x):**

**x** **=** self**.flatten(x)**

**logits** **=** self**.linear\_relu\_stack(x)**

return **logits**

**model** **=** [**NeuralNetwork**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module)**()**

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-idx3-ubyte.gz

0%| | 0/26421880 [00:00<?, ?it/s]

0%| | 65536/26421880 [00:00<01:12, 364128.30it/s]

1%| | 229376/26421880 [00:00<00:38, 683715.24it/s]

3%|2 | 688128/26421880 [00:00<00:13, 1866013.56it/s]

6%|6 | 1638400/26421880 [00:00<00:07, 3538214.91it/s]

16%|#6 | 4292608/26421880 [00:00<00:02, 9480228.63it/s]

24%|##4 | 6422528/26421880 [00:00<00:01, 10872893.72it/s]

34%|###4 | 9109504/26421880 [00:01<00:01, 14549805.68it/s]

43%|####2 | 11272192/26421880 [00:01<00:01, 14127811.19it/s]

53%|#####2 | 13959168/26421880 [00:01<00:00, 16838675.37it/s]

61%|######1 | 16187392/26421880 [00:01<00:00, 15740467.14it/s]

71%|#######1 | 18776064/26421880 [00:01<00:00, 17791475.63it/s]

80%|#######9 | 21102592/26421880 [00:01<00:00, 16570251.04it/s]

89%|########9 | 23625728/26421880 [00:01<00:00, 18262503.14it/s]

99%|#########8| 26050560/26421880 [00:01<00:00, 17059993.83it/s]

100%|##########| 26421880/26421880 [00:01<00:00, 13283909.11it/s]

Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-idx1-ubyte.gz

0%| | 0/29515 [00:00<?, ?it/s]

100%|##########| 29515/29515 [00:00<00:00, 329395.02it/s]

Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz

0%| | 0/4422102 [00:00<?, ?it/s]

1%|1 | 65536/4422102 [00:00<00:11, 364702.73it/s]

5%|5 | 229376/4422102 [00:00<00:06, 685977.26it/s]

19%|#8 | 819200/4422102 [00:00<00:01, 2327285.23it/s]

39%|###8 | 1703936/4422102 [00:00<00:00, 3959983.97it/s]

74%|#######4 | 3276800/4422102 [00:00<00:00, 6849332.33it/s]

100%|##########| 4422102/4422102 [00:00<00:00, 6035081.50it/s]

Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz

0%| | 0/5148 [00:00<?, ?it/s]

100%|##########| 5148/5148 [00:00<00:00, 25952256.00it/s]

Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw

## siêu tham số

Siêu tham số là các tham số có thể điều chỉnh cho phép bạn kiểm soát quá trình tối ưu hóa mô hình. Các giá trị siêu tham số khác nhau có thể ảnh hưởng đến tốc độ hội tụ và đào tạo mô hình ( [đọc thêm](https://pytorch-org.translate.goog/tutorials/beginner/hyperparameter_tuning_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) về điều chỉnh siêu tham số)

Chúng tôi xác định các siêu đường kính sau để đào tạo:

* **Số kỷ nguyên** - số lần lặp lại tập dữ liệu
* **Kích thước hàng loạt** - số lượng mẫu dữ liệu được truyền qua mạng trước khi các tham số được cập nhật
* **Tỷ lệ học tập** - mức độ cập nhật các tham số mô hình ở mỗi đợt/kỷ nguyên. Các giá trị nhỏ hơn mang lại tốc độ học chậm, trong khi các giá trị lớn có thể dẫn đến hành vi không thể đoán trước trong quá trình đào tạo.

**learning\_rate** **=** 1e-3

**batch\_size** **=** 64

**epochs** **=** 5

## Vòng lặp tối ưu hóa

Khi chúng tôi đặt siêu tham số, chúng tôi có thể đào tạo và tối ưu hóa mô hình của mình bằng vòng lặp tối ưu hóa. Mỗi lần lặp lại của vòng lặp tối ưu hóa được gọi là một **kỷ nguyên** .

Mỗi kỷ nguyên bao gồm hai phần chính:

* **Vòng lặp đào tạo** - lặp lại tập dữ liệu đào tạo và cố gắng hội tụ các tham số tối ưu.
* **Vòng kiểm tra/Xác thực** - lặp lại tập dữ liệu thử nghiệm để kiểm tra xem hiệu suất của mô hình có được cải thiện hay không.

Chúng ta hãy làm quen ngắn gọn với một số khái niệm được sử dụng trong vòng lặp huấn luyện. Tiếp tục để xem [Triển khai đầy đủ](https://pytorch-org.translate.goog/tutorials/beginner/basics/optimization_tutorial.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#full-impl-label) vòng lặp tối ưu hóa.

### **Mất chức năng**

Khi được cung cấp một số dữ liệu đào tạo, mạng chưa được đào tạo của chúng tôi có khả năng không đưa ra câu trả lời chính xác. **Hàm mất mát** đo mức độ không giống nhau của kết quả thu được với giá trị mục tiêu và đó là hàm mất mát mà chúng tôi muốn giảm thiểu trong quá trình đào tạo. Để tính toán tổn thất, chúng tôi đưa ra dự đoán bằng cách sử dụng đầu vào của mẫu dữ liệu đã cho và so sánh nó với giá trị nhãn dữ liệu thực.

Các hàm mất thông thường bao gồm [nn.MSELoss](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.MSELoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.MSELoss) (Lỗi bình phương trung bình) cho các tác vụ hồi quy và [nn.NLLLoss](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.NLLLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.NLLLoss) (Khả năng ghi nhật ký âm) để phân loại. [nn.CrossEntropyLoss](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss) kết hợp nn.LogSoftmaxvà nn.NLLLoss.

Chúng tôi chuyển các bản ghi đầu ra của mô hình cho nn.CrossEntropyLoss, điều này sẽ chuẩn hóa các bản ghi và tính toán lỗi dự đoán.

*# Initialize the loss function*

[**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss) **=** [**nn.CrossEntropyLoss**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**()**

### **trình tối ưu hóa**

Tối ưu hóa là quá trình điều chỉnh các tham số của mô hình để giảm sai số của mô hình trong mỗi bước huấn luyện. **Các thuật toán tối ưu hóa** xác định cách thực hiện quy trình này (trong ví dụ này, chúng tôi sử dụng Giảm dần độ dốc ngẫu nhiên). Tất cả logic tối ưu hóa được gói gọn trong optimizerđối tượng. Ở đây, chúng tôi sử dụng trình tối ưu hóa SGD; ngoài ra, có nhiều [trình tối ưu hóa khác nhau](https://pytorch-org.translate.goog/docs/stable/optim.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp) có sẵn trong PyTorch chẳng hạn như ADAM và RMSProp, hoạt động tốt hơn cho các loại mô hình và dữ liệu khác nhau.

Chúng tôi khởi tạo trình tối ưu hóa bằng cách đăng ký các tham số của mô hình cần được đào tạo và chuyển vào siêu tham số tốc độ học tập.

[**optimizer**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD) **=** [**torch.optim.SGD**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD)**(**[**model.parameters**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.parameters)**(),** **lr=learning\_rate)**

Trong vòng lặp đào tạo, quá trình tối ưu hóa diễn ra theo ba bước:

* Gọi optimizer.zero\_grad()để đặt lại độ dốc của các tham số mô hình. Độ dốc theo mặc định cộng lại; để ngăn việc đếm trùng lặp, chúng tôi sẽ đưa chúng về 0 một cách rõ ràng ở mỗi lần lặp lại.
* Lan truyền ngược tổn thất dự đoán bằng lệnh gọi tới loss.backward(). PyTorch ký gửi độ dốc của tổn thất ghi từng tham số.
* Khi chúng tôi có độ dốc của mình, chúng tôi gọi optimizer.step()để điều chỉnh các tham số theo độ dốc được thu thập trong quá trình quay ngược.

## Thực hiện đầy đủ

Chúng tôi xác định train\_loopvòng lặp đó trên mã tối ưu hóa của mình và test\_loopđánh giá hiệu suất của mô hình dựa trên dữ liệu thử nghiệm của chúng tôi.

def **train\_loop(dataloader,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**,** [**optimizer**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD)**):**

**size** **=** len**(dataloader.dataset)**

*# Set the model to training mode - important for batch normalization and dropout layers*

*# Unnecessary in this situation but added for best practices*

[**model.train**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.train)**()**

for **batch,** **(X,** **y)** **in** enumerate**(dataloader):**

*# Compute prediction and loss*

**pred** **=** **model(X)**

**loss** **=** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**(pred,** **y)**

*# Backpropagation*

**loss.backward()**

[**optimizer.step**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.Optimizer.step.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.Optimizer.step)**()**

[**optimizer.zero\_grad**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD.zero_grad)**()**

if **batch** **%** 100 **==** 0**:**

**loss,** **current** **=** **loss.item(),** **(batch** **+** 1**)** **\*** len**(X)**

print**(**f"loss: *{***loss***:*>7f*}* [*{***current***:*>5d*}*/*{***size***:*>5d*}*]"**)**

def **test\_loop(dataloader,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**):**

*# Set the model to evaluation mode - important for batch normalization and dropout layers*

*# Unnecessary in this situation but added for best practices*

[**model.eval**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.eval)**()**

**size** **=** len**(dataloader.dataset)**

**num\_batches** **=** len**(dataloader)**

**test\_loss,** **correct** **=** 0**,** 0

*# Evaluating the model with torch.no\_grad() ensures that no gradients are computed during test mode*

*# also serves to reduce unnecessary gradient computations and memory usage for tensors with requires\_grad=True*

with [**torch.no\_grad**](https://pytorch-org.translate.goog/docs/stable/generated/torch.no_grad.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.no_grad)**():**

for **X,** **y** **in** **dataloader:**

**pred** **=** **model(X)**

**test\_loss** **+=** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**(pred,** **y).item()**

**correct** **+=** **(pred.argmax(**1**)** **==** **y).type(**[**torch.float**](https://pytorch-org.translate.goog/docs/stable/tensor_attributes.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.dtype)**).sum().item()**

**test\_loss** **/=** **num\_batches**

**correct** **/=** **size**

print**(**f"Test Error: **\n** Accuracy: *{***(**100**\*correct)***:*>0.1f*}*%, Avg loss: *{***test\_loss***:*>8f*}* **\n**"**)**

Chúng tôi khởi tạo hàm mất mát và trình tối ưu hóa, đồng thời chuyển nó tới train\_loopvà test\_loop. Vui lòng tăng số lượng kỷ nguyên để theo dõi hiệu suất cải thiện của mô hình.

[**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss) **=** [**nn.CrossEntropyLoss**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**()**

[**optimizer**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD) **=** [**torch.optim.SGD**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD)**(**[**model.parameters**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.parameters)**(),** **lr=learning\_rate)**

**epochs** **=** 10

for **t** **in** range**(epochs):**

print**(**f"Epoch *{***t+**1*}***\n**-------------------------------"**)**

**train\_loop(**[**train\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**,** [**optimizer**](https://pytorch-org.translate.goog/docs/stable/generated/torch.optim.SGD.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.optim.SGD)**)**

**test\_loop(**[**test\_dataloader**](https://pytorch-org.translate.goog/docs/stable/data.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.utils.data.DataLoader)**,** **model,** [**loss\_fn**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.CrossEntropyLoss.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.CrossEntropyLoss)**)**

print**(**"Done!"**)**

Epoch 1

-------------------------------

loss: 2.298730 [ 64/60000]

loss: 2.289123 [ 6464/60000]

loss: 2.273286 [12864/60000]

loss: 2.269406 [19264/60000]

loss: 2.249603 [25664/60000]

loss: 2.229407 [32064/60000]

loss: 2.227368 [38464/60000]

loss: 2.204261 [44864/60000]

loss: 2.206193 [51264/60000]

loss: 2.166651 [57664/60000]

Test Error:

Accuracy: 50.9%, Avg loss: 2.166725

Epoch 2

-------------------------------

loss: 2.176750 [ 64/60000]

loss: 2.169595 [ 6464/60000]

# **8. LƯU VÀ TẢI MÔ HÌNH**

Trong phần này, chúng ta sẽ xem xét cách duy trì trạng thái mô hình bằng cách lưu, tải và chạy các dự đoán mô hình.

**import** **torch**

**import** **torchvision.models** as **models**

## Lưu và tải trọng lượng mô hình

Các mô hình PyTorch lưu trữ các tham số đã học trong một từ điển trạng thái bên trong, được gọi là state\_dict. Chúng có thể được duy trì thông qua torch.savephương thức:

**model** **=** [**models.vgg16**](https://pytorch-org.translate.goog/vision/stable/models/generated/torchvision.models.vgg16.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.models.vgg16)**(weights=**'IMAGENET1K\_V1'**)**

[**torch.save**](https://pytorch-org.translate.goog/docs/stable/generated/torch.save.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.save)**(**[**model.state\_dict**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.state_dict)**(),** 'model\_weights.pth'**)**

Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /var/lib/jenkins/.cache/torch/hub/checkpoints/vgg16-397923af.pth

0%| | 0.00/528M [00:00<?, ?B/s]

4%|3 | 18.6M/528M [00:00<00:02, 195MB/s]

7%|7 | 38.2M/528M [00:00<00:02, 201MB/s]

11%|# | 57.9M/528M [00:00<00:02, 204MB/s]

15%|#4 | 77.5M/528M [00:00<00:02, 204MB/s]

18%|#8 | 97.1M/528M [00:00<00:02, 205MB/s]

22%|##2 | 117M/528M [00:00<00:02, 206MB/s]

26%|##5 | 137M/528M [00:00<00:01, 206MB/s]

30%|##9 | 156M/528M [00:00<00:01, 207MB/s]

33%|###3 | 176M/528M [00:00<00:01, 207MB/s]

37%|###7 | 196M/528M [00:01<00:01, 207MB/s]

41%|#### | 216M/528M [00:01<00:01, 208MB/s]

45%|####4 | 236M/528M [00:01<00:01, 208MB/s]

49%|####8 | 256M/528M [00:01<00:01, 208MB/s]

52%|#####2 | 276M/528M [00:01<00:01, 209MB/s]

56%|#####6 | 296M/528M [00:01<00:01, 209MB/s]

60%|#####9 | 316M/528M [00:01<00:01, 209MB/s]

64%|######3 | 336M/528M [00:01<00:00, 210MB/s]

67%|######7 | 356M/528M [00:01<00:00, 210MB/s]

71%|#######1 | 376M/528M [00:01<00:00, 210MB/s]

75%|#######5 | 396M/528M [00:02<00:00, 211MB/s]

79%|#######8 | 417M/528M [00:02<00:00, 211MB/s]

83%|########2 | 437M/528M [00:02<00:00, 212MB/s]

87%|########6 | 457M/528M [00:02<00:00, 211MB/s]

90%|######### | 477M/528M [00:02<00:00, 212MB/s]

94%|#########4| 498M/528M [00:02<00:00, 211MB/s]

98%|#########8| 518M/528M [00:02<00:00, 210MB/s]

100%|##########| 528M/528M [00:02<00:00, 208MB/s]

Để tải các trọng số của mô hình, trước tiên bạn cần tạo một phiên bản của cùng một mô hình, sau đó tải các tham số bằng load\_state\_dict()phương thức.

**model** **=** [**models.vgg16**](https://pytorch-org.translate.goog/vision/stable/models/generated/torchvision.models.vgg16.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torchvision.models.vgg16)**()** *# we do not specify ``weights``, i.e. create untrained model*

[**model.load\_state\_dict**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.load_state_dict)**(**[**torch.load**](https://pytorch-org.translate.goog/docs/stable/generated/torch.load.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.load)**(**'model\_weights.pth'**))**

[**model.eval**](https://pytorch-org.translate.goog/docs/stable/generated/torch.nn.Module.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.nn.Module.eval)**()**

VGG(

(features): Sequential(

(0): Conv2d(3, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): ReLU(inplace=True)

(2): Conv2d(64, 64, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(3): ReLU(inplace=True)

(4): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(5): Conv2d(64, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(6): ReLU(inplace=True)

(7): Conv2d(128, 128, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(8): ReLU(inplace=True)

(9): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(10): Conv2d(128, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(11): ReLU(inplace=True)

(12): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(13): ReLU(inplace=True)

(14): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(15): ReLU(inplace=True)

(16): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(17): Conv2d(256, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(18): ReLU(inplace=True)

(19): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(20): ReLU(inplace=True)

(21): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(22): ReLU(inplace=True)

(23): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

(24): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(25): ReLU(inplace=True)

(26): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(27): ReLU(inplace=True)

(28): Conv2d(512, 512, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(29): ReLU(inplace=True)

(30): MaxPool2d(kernel\_size=2, stride=2, padding=0, dilation=1, ceil\_mode=False)

)

(avgpool): AdaptiveAvgPool2d(output\_size=(7, 7))

(classifier): Sequential(

(0): Linear(in\_features=25088, out\_features=4096, bias=True)

(1): ReLU(inplace=True)

(2): Dropout(p=0.5, inplace=False)

(3): Linear(in\_features=4096, out\_features=4096, bias=True)

(4): ReLU(inplace=True)

(5): Dropout(p=0.5, inplace=False)

(6): Linear(in\_features=4096, out\_features=1000, bias=True)

)

)

GHI CHÚ

hãy chắc chắn gọi model.eval()phương thức trước khi suy luận để đặt các lớp chuẩn hóa hàng loạt và loại bỏ thành chế độ đánh giá. Không làm được điều này sẽ mang lại kết quả suy luận không nhất quán.

## Lưu và tải mô hình với hình dạng

Khi tải các trọng số của mô hình, trước tiên chúng ta cần khởi tạo lớp mô hình, vì lớp này xác định cấu trúc của mạng. Chúng ta có thể muốn lưu cấu trúc của lớp này cùng với mô hình, trong trường hợp đó chúng ta có thể chuyển model(và không model.state\_dict()) cho hàm lưu:

[**torch.save**](https://pytorch-org.translate.goog/docs/stable/generated/torch.save.html?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=wapp#torch.save)**(model,** 'model.pth'**)**