# **MỤC LỤC**

[**MỤC LỤC**](#_naicst5dzv6l) **1**

[**PHẦN MỞ ĐẦU**](#_h3ncjck60x3j) **3**

[**CHƯƠNG 1: TỔNG QUAN VỀ MẠNG VGGNET**](#_mtnssmyxdry7) **4**

[1.Tiến trình phát triển của CNN](#_6xniigpsfpw3) 4

[1.1. Giới thiệu chung](#_j0m6g55lc7o) 4

[1.2. Các dấu mốc quan trọng](#_7redsjs847zx) 4

[2. Mạng sử dụng Khối (VGG)](#_xcvfj6l3v93s) 5

[2.1 Khối VGG](#_xitacpjz76rs) 6

[2.2 Mạng VGG](#_eeiwy5pu8p5e) 6

[3. Code minh họa](#_2qr5l6hlrdk0) 9

[**CHƯƠNG 2: BÀI TOÁN HỌC MÁY**](#_kefnk3poovkl) **11**

[1. Bài toán phân loại quần áo, giày dép trên bộ dữ liệu Fashion MNIST](#_u9v38p8abk5y) 11

[1.1. Mô tả dữ liệu](#_hali3i1qk3l2) 11

[1.2. Mô tả bài toán học máy](#_o4virxazcufw) 12

[1.3. Áp dụng bài toán học máy](#_vue7cn50am0) 13

[1.4. Kết luận](#_cwsze07e6rgj) 17

[2. Bài toán dự đoán cân nặng của cá](#_psphopb2c49f) 18

[2.1 Mô tả dữ liệu](#_8vux66xzxy0z) 18

[2.2 Mô tả bài toán học máy](#_xmipta8wi1c9) 18

[2.3 Code bài toán](#_ldk3188j5jbt) 18

[2.4 Kết luận](#_ldk3188j5jbt) 19

[3. Bài toán phân loại ảnh chó mèo](#_chqmr2u4xri9) 20

[3.1. Mô tả dữ liệu](#_cszzoa5h809m) 20

[3.2. Mô tả bài toán học máy](#_tmtwxrl83df7) 21

[3.3. Mô tả bài toán học máy](#_k8qlnhvoes5q) 21

[3.4 Kết luận](#_iym4y8wm3n5v) 27

# 

# **PHẦN MỞ ĐẦU**

# 

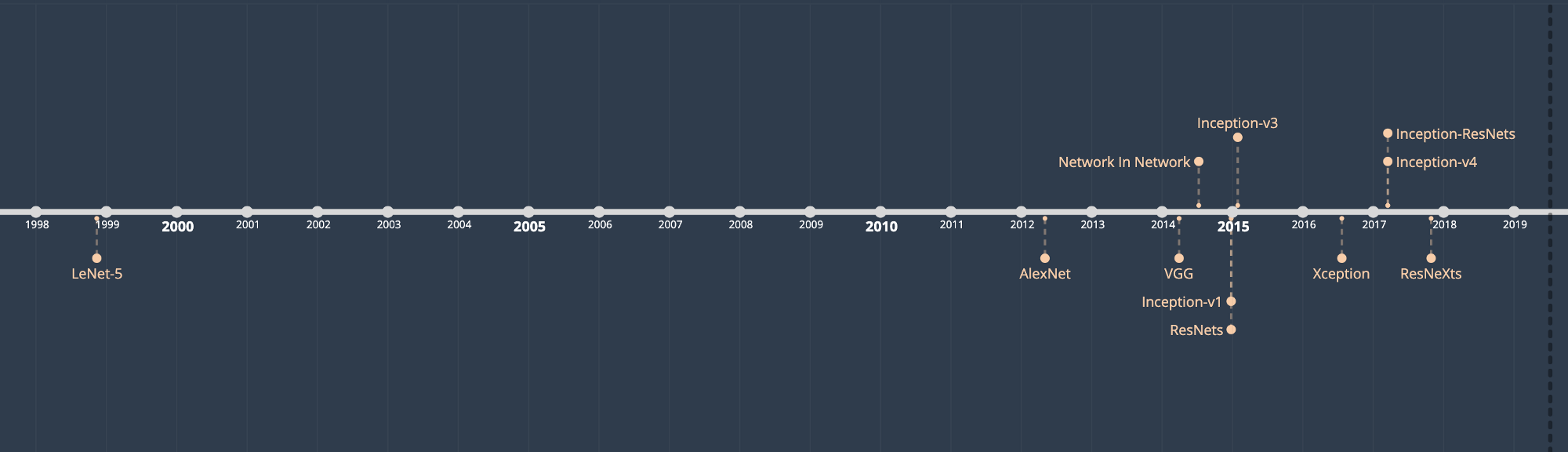
# **CHƯƠNG 1: TỔNG QUAN VỀ MẠNG VGGNET**

## **1.Tiến trình phát triển của CNN**

### ***1.1. Giới thiệu chung***

Mạng CNN ra đời đã thúc đẩy quá trình phát triển của ngành computer vision. Hiện tại có rất nhiều các kiến trúc mạng CNN khác nhau và các kiến trúc mới vẫn đang tiếp tục được khám phá ngày qua ngày. Nhưng ít ai biết rằng đằng sau những khám phá đó là một tiến trình khoa học lâu dài và bền bỉ trong gần 20 năm. Với sự kết hợp đồng bộ của phát triển kiến trúc mạng, khả năng tính toán của máy tính và các phương pháp tối ưu hóa. Bài viết này mình sẽ giới thiệu tới các bạn lược sử hình thành của các kiến trúc CNN tiêu biểu và những đóng góp mang tính cải tiến của những kiến trúc mạng này so với trước đó. Thông qua bài viết bạn đọc sẽ hình dung được lộ trình hình thành và phát triển cho tới ngày nay của những mạng CNN và đồng thời hiểu rõ được đặc trưng trong kiến trúc của từng mạng. Những ưu nhược điểm và cải tiến đã thực hiện so với những kiến trúc mạng trước đó.

### ***1.2. Các dấu mốc quan trọng***



**Hình 1:** Các cột mốc phát triển của mạng CNN. Source: [Illustrated: 10 CNN Architectures - Raimi Karim](https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d)

Tiến trình phát triển của các kiến trúc CNN có thể được khái quát qua những dẫu mốc quan trọng sau đây:

* 1998: Yan Lecun lần đầu tiên sử dụng mạng tích chập trong tác vụ phân loại chữ số viết tay và đạt hiệu quả cao. Tuy nhiên vào thời điểm đó do chưa có sự phát triển của dữ liệu và khả năng tính toán nên mạng CNN vẫn chưa có cơ hội phát triển.
* 2009: Bộ dữ liệu ImageNet được giới thiệu vào năm 2009 là một trong những bộ dữ liệu tạo ra tiếng vang trong cộng đồng computer vision. Đây là bộ dữ liệu lớn nhất so với các bộ dữ liệu từng có từ trước đến thời điểm đó. Với 1 triệu ảnh của 1000 nhãn, mỗi nhãn bao gồm 1000 ảnh, các mô hình được huấn luyện trên ImageNet dường như có thể chuyển giao tới rất nhiều những domain dữ liệu khác nhau. Kể từ thời điểm đó, bắt đầu từ năm 2010, ImageNet trở thành cuộc thi đo đạc sự phát triển của các thuật toán học có giám sát trong thị giác máy tính.
* 2012: Mạng AlexNet lần đầu tiên vượt qua các phương pháp tạo đặc trưng thủ công theo truyền thống và đạt độ chính xác cách biệt trong cuộc thi ImageNet. Khởi đầu cho xu hướng nghiên cứu của các chuyên gia đầu ngành trong computer vision trong việc ứng dụng CNN.
* Liên tiếp vào những năm sau đó, chỉ trong vòng 8 năm, rất nhiều các kiến trúc mới của CNN đã được phát triển và cải tiến qua các năm như VGG Net, GoogleNet, ResNet, DenseNet,… mà chúng ta sẽ tìm hiểu qua bài viết này. Qúa trình phát triển của các kiến trúc mạng song hành cùng với sự phát triển phần cứng máy tính mạnh hơn trước. Các GPU có tốc độ tính toán nhanh hơn, các frame work hỗ trợ deep learning hình thành và phát triển, các bộ dữ liệu lớn ngày một nhiều hơn và kéo theo đó là sự mở rộng của cộng đồng những người theo đuổi và nghiên cứu về AI cho tới thời điểm hiện tại.

## **2. Mạng sử dụng Khối (VGG)**

Mặc dù AlexNet đã chứng minh rằng các mạng nơ-ron tích chập có thể đạt được kết quả tốt, nó lại không cung cấp một khuôn mẫu chung để định hướng nghiên cứu sau này trong việc thiết kế các mạng mới. Trong các phần tiếp theo, chúng tôi sẽ giới thiệu một số khái niệm dựa trên thực nghiệm được sử dụng rộng rãi khi thiết kế mạng học sâu.

Sự phát triển trong lĩnh vực này có nét tương đồng tiến triển trong ngành thiết kế vi xử lý, chuyển từ việc sắp đặt các bóng bán dẫn, đến các phần tử logic và các khối logic.

Tương tự như vậy, việc thiết kế kiến trúc các mạng nơ-ron đã phát triển theo hướng ngày một trừu tượng hơn. Điển hình là việc các nhà nghiên cứu đã thay đổi suy nghĩ từ quy mô các nơ-ron riêng lẻ sang các tầng, và giờ đây là các khối chứa các tầng lặp lại theo khuôn mẫu.

Ý tưởng sử dụng các khối lần đầu xuất hiện trong mạng VGG, được đặt theo tên của nhóm [VGG](http://www.robots.ox.ac.uk/~vgg/) thuộc Đại học Oxford. Sử dụng bất kỳ các framework học sâu hiện đại nào với vòng lặp và chương trình con để xây dựng các cấu trúc lặp lại này là tương đối dễ dàng.

### ***2.1 Khối VGG***

Khối cơ bản của mạng tích chập cổ điển là một chuỗi các tầng sau đây: (i) một tầng tích chập (với phần đệm để duy trì độ phân giải), (ii) một tầng phi tuyến như ReLU, (iii) một tầng gộp như tầng gộp cực đại. Một khối VGG gồm một chuỗi các tầng tích chập, tiếp nối bởi một tầng gộp cực đại để giảm chiều không gian. Trong bài báo gốc của VGG [[Simonyan & Zisserman, 2014]](https://d2l.aivivn.com/chapter_references/zreferences.html#simonyan-zisserman-2014), tác giả sử dụng tích chập với các hạt nhân 3×3 và tầng gộp cực đại 2×2 với sải bước bằng 2 (giảm một nửa độ phân giải sau mỗi khối). Trong mã nguồn dưới đây, ta định nghĩa một hàm tên vgg\_block để tạo một khối VGG. Hàm này nhận hai đối số num\_convs và num\_channels tương ứng lần lượt với số tầng tích chập và số kênh đầu ra.

**from** **d2l** **import** mxnet **as** d2l

**from** **mxnet** **import** np, npx

**from** **mxnet.gluon** **import** nn

npx.set\_np()

**def** vgg\_block(num\_convs, num\_channels):

blk = nn.Sequential()

**for** \_ **in** range(num\_convs):

blk.add(nn.Conv2D(num\_channels, kernel\_size=3,

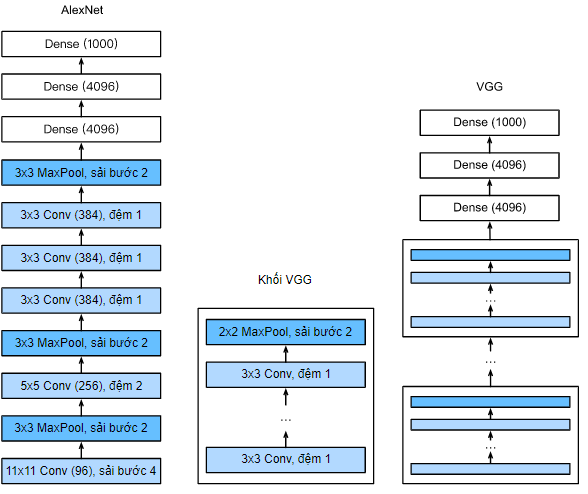
padding=1, activation='relu'))

blk.add(nn.MaxPool2D(pool\_size=2, strides=2))

**return** blk

### ***2.2 Mạng VGG***

Giống như AlexNet và LeNet, mạng VGG có thể được phân chia thành hai phần: phần đầu tiên bao gồm chủ yếu các tầng tích chập và tầng gộp, còn phần thứ hai bao gồm các tầng kết nối đầy đủ. Phần tích chập của mạng gồm các mô-đun vgg block kết nối liên tiếp với nhau. Trong [Fig. 7.2.1](https://d2l.aivivn.com/chapter_convolutional-modern/vgg_vn.html#fig-vgg), biến conv\_arch bao gồm một danh sách các tuples (một tuple cho mỗi khối), trong đó mỗi tuple chứa hai giá trị: số lượng tầng tích chập và số kênh đầu ra, cũng chính là những đối số cần thiết để gọi hàm vgg block. Mô-đun kết nối đầy đủ giống hệt với mô-đun tương ứng trong AlexNet.



*Thiết kế mạng từ các khối cơ bản*

Mạng VGG gốc có 5 khối tích chập, trong đó hai khối đầu tiên bao gồm một tầng tích chập ở mỗi khối, ba khối còn lại chứa hai tầng tích chập ở mỗi khối. Khối đầu tiên có 64 kênh đầu ra, mỗi khối tiếp theo nhân đôi số kênh đầu ra cho tới khi đạt giá trị 512. Vì mạng này sử dụng 8 tầng tích chập và 3 tầng kết nối đầy đủ nên nó thường được gọi là VGG-11.

conv\_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))

Đoạn mã nguồn sau đây lập trình mạng VGG-11. Ở đây ta chỉ đơn thuần thực hiện vòng lặp for trên biến conv\_arch.

**def** vgg(conv\_arch):

net = nn.Sequential()

*# The convolutional layer part*

**for** (num\_convs, num\_channels) **in** conv\_arch:

net.add(vgg\_block(num\_convs, num\_channels))

*# The fully connected layer part*

net.add(nn.Dense(4096, activation='relu'), nn.Dropout(0.5),

nn.Dense(4096, activation='relu'), nn.Dropout(0.5),

nn.Dense(10))

**return** net

net = vgg(conv\_arch)

Tiếp theo, chúng ta sẽ tạo một mẫu dữ liệu một kênh với chiều cao và chiều rộng là 224 để quan sát kích thước đầu ra của mỗi tầng.

net.initialize()

X = np.random.uniform(size=(1, 1, 224, 224))

**for** blk **in** net:

X = blk(X)

print(blk.name, 'output shape:**\t**', X.shape)

sequential1 output shape: (1, 64, 112, 112)

sequential2 output shape: (1, 128, 56, 56)

sequential3 output shape: (1, 256, 28, 28)

sequential4 output shape: (1, 512, 14, 14)

sequential5 output shape: (1, 512, 7, 7)

dense0 output shape: (1, 4096)

dropout0 output shape: (1, 4096)

dense1 output shape: (1, 4096)

dropout1 output shape: (1, 4096)

dense2 output shape: (1, 10)

Như bạn thấy, ta đã giảm chiều cao và chiều rộng đi một nửa sau mỗi khối, cuối cùng kích thước của các biểu diễn chỉ còn là 7 trên mỗi chiều trước khi được trải phẳng ra để tiếp tục xử lý trong tầng kết nối đầy đủ.

## **3. Code minh họa**

from numpy import asarray

from numpy import unique

from numpy import argmax

from tensorflow.keras.datasets.mnist import load\_data

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import MaxPool2D

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dropout

# load dữ liệu

(x\_train, y\_train), (x\_test, y\_test) = load\_data()

#định hình dữ liệu

x\_train = x\_train.reshape((x\_train.shape[0], x\_train.shape[1], x\_train.shape[2], 1))

x\_test = x\_test.reshape((x\_test.shape[0], x\_test.shape[1], x\_test.shape[2], 1))

# xác định ảnh đầu vào

in\_shape = x\_train.shape[1:]

# xác định số lớp

n\_classes = len(unique(y\_train))

print(in\_shape, n\_classes)

# chuẩn hóa dữ liệu

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

#xác định mô hình

model = Sequential()

model.add(Conv2D(32, (3,3), activation='relu', kernel\_initializer='he\_uniform', input\_shape=in\_shape))

model.add(MaxPool2D((2, 2)))

model.add(Flatten())

model.add(Dense(100, activation='relu', kernel\_initializer='he\_uniform'))

model.add(Dropout(0.5))

model.add(Dense(n\_classes, activation='softmax'))

# xác định mất mát và tối ưu

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# khớp model

model.fit(x\_train, y\_train, epochs=10, batch\_size=128, verbose=0)

# đánh giá mô hình

loss, acc = model.evaluate(x\_test, y\_test, verbose=0)

print('Accuracy: %.3f' % acc)

# dự đoán

image = x\_train[0]

yhat = model.predict(asarray([image]))

print('Predicted: class=%d' % argmax(yhat))

# 

# **CHƯƠNG 2: BÀI TOÁN HỌC MÁY**

## **1. Bài toán phân loại quần áo, giày dép trên bộ dữ liệu Fashion MNIST**

### ***1.1. Mô tả dữ liệu***

Tập dữ liệu về quần áo và giày dép từ https://github.com/zalandoresearch/fashion-mnist, chứa khoảng 70,000 ảnh đen trắng phân thành 10 loại. Mỗi một ảnh là một loại quần áo hoặc giày dép với độ phân giải thấp (28 by 28 pixel), như hình minh hoạ bên dưới:



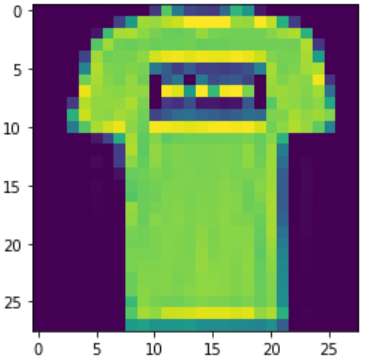
Bộ data này gồm 70,000 ảnh, trong đó training set có 60,000 ảnh, test set có 10,000 ảnh được phân vào 10 class như sau:

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

### ***1.2. Mô tả bài toán học máy***

Bài toán phân lớp nhiều lớp dựa vào ảnh để phân loại quần áo, giày dép

Bộ dữ liệu có 60.000 ảnh trong tập huấn luyện, với mỗi ảnh được biểu diễn theo dạng 28x28 pixel:



Kỹ thuật sử dụng: Kỹ thuật học máy được áp dụng: Neural Network (C[onvolutional Neural Network](https://nttuan8.com/bai-6-convolutional-neural-network/))

### ***1.3. Áp dụng bài toán học máy***

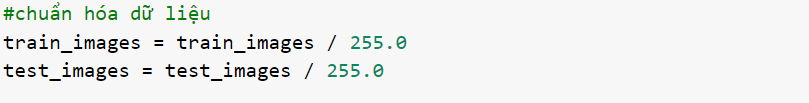


*Tiền xử lý dữ liệu*

Dữ liệu cần được tiền xử lý trước khi được dùng để huấn luyện mạng neuron. Phân tích ảnh đầu tiên trong tập dữ liệu, chúng ta sẽ thấy các pixel có giá trị từ 0 đến 255.



Tiền xử lý để mỗi một điểm ảnh có giá trị từ 0 đến 1. Để làm điều này, chúng ta chỉ cần lấy giá trị của pixel chia cho 255

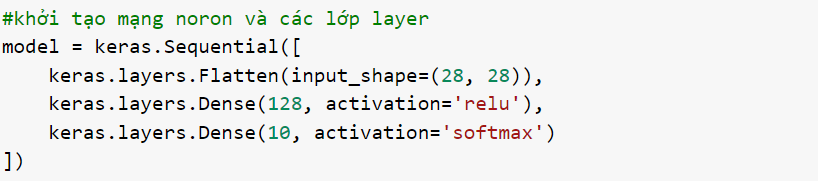


Để xây dựng mạng neuron, chúng tay cần cấu hình các layer của mô hình, và sau đó biên dịch mô hình.

*Thiết lập các layers*

Thành phần cơ bản của một mạng neuron là các *layer*. Các layer trích xuất các điểm đặc biệt từ dữ liệu mà chúng đón nhận. Khi thực hiện tốt, những điểm đặc biệt này mang nhiều ý nghĩa và phục vụ cho toán của chúng ta.

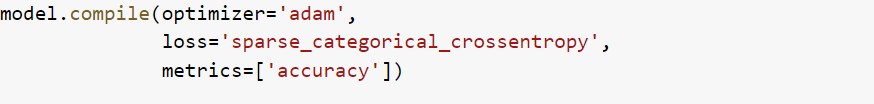
Đa số các mô hình deep learning đều chứa các layer đơn giản được xâu chuỗi lại với nhau. Đa số các layer, ví dụ tf.keras.layers.Dense, đều có các trọng số sẽ được học trong quá trình huấn luyện.



*Biên dịch mô hình*

Trước khi mô hình có thể được huấn luyện, chúng ta cần thêm vài chỉnh sửa. Các chỉnh sửa này được thêm vào trong bước *biên dịch* của mô hình:

* *Hàm thiệt hại* — dùng để đo lường mức độ chính xác của mô hình trong quá trình huấn luyện. Chúng ta cần giảm thiểu giá trị của hạm này "điều khiển" mô hình đi đúng hướng (thiệt hại càng ít, chính xác càng cao).
* *Trình tối ưu hoá* — Đây là cách mô hình được cập nhật dựa trên dữ liệu huấn luyện được cung cấp và hàm thiệt hại.
* *Số liệu* — dùng để theo dõi các bước huấn luyện và kiểm thử. Ví dụ sau dùng *accuracy*, tỉ lệ ảnh được phân loại chính xác.



*Huấn luyện mô hình*

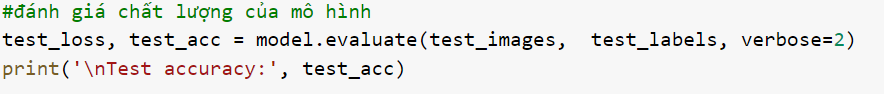
Huấn luyện mô hình mạng neuron cần các bước sau:

1. Cung cấp dữ liệu huấn luyện cho mô hình. Trong ví dụ này, dữ liệu huấn luyện năm trong 2 mảng train\_images và train\_labels
2. Mô hình sẽ học cách liên kết ảnh với nhãn.
3. Chúng ta sẽ yêu cầu mô hình đưa ra dự đoán từ dữ liệu của tập kiểm thử, trong ví dụ này là mảng test\_images, sau đó lấy kết quả dự đoán đối chiếu với nhãn trong mảng test\_labels.

Để bắt đầu huấn luyện, gọi hàm model.fit. Hàm này được đặt tên fit vì nó sẽ "fit" ("khớp") mô hình với dữ liệu huấn luyện:



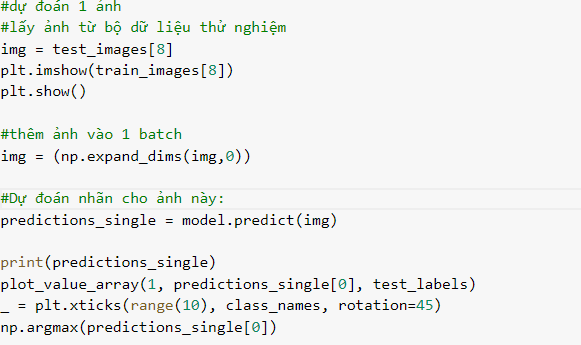
*Đánh giá chất lượng mô hình*



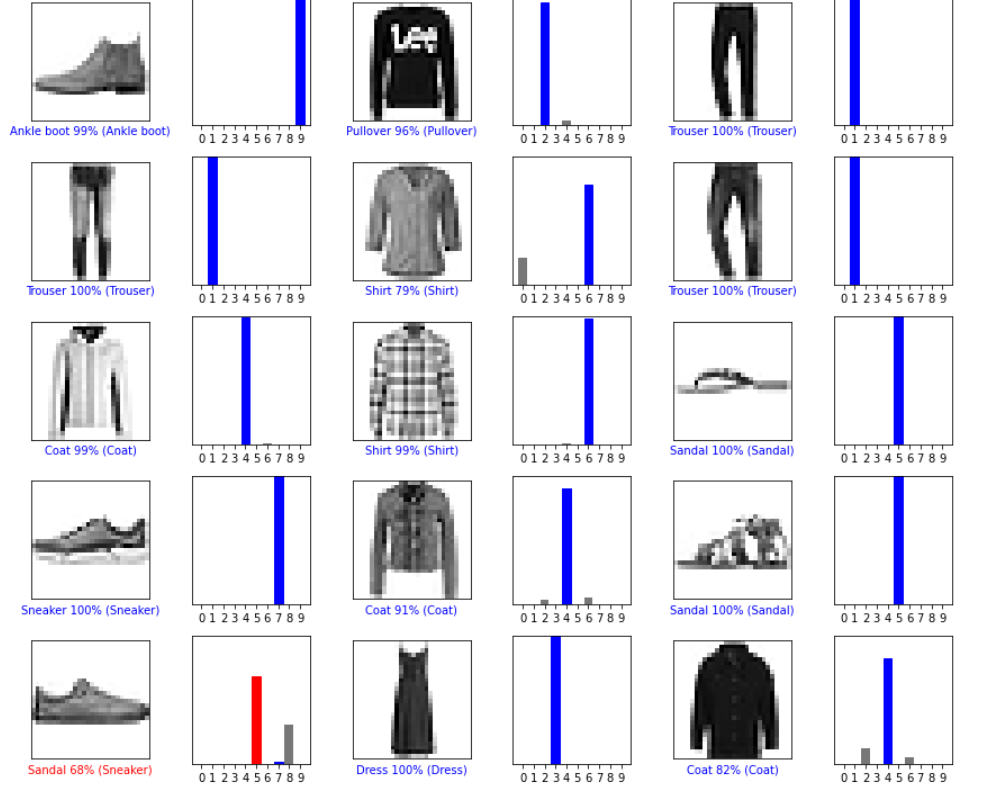
*Đưa ra dự đoán*

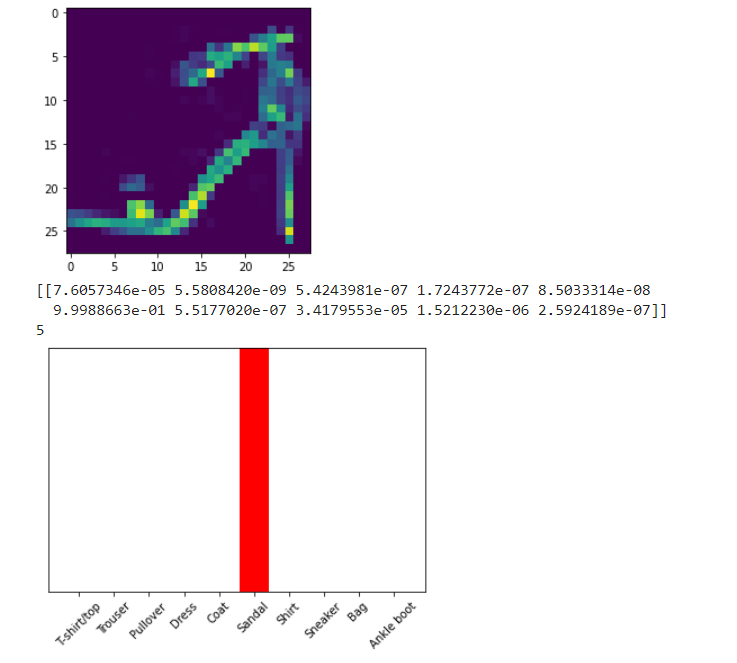


Dùng mô hình để đưa ra dự đoán về một ảnh duy nhất.



Kết quả chương trình sau khi chạy





### ***1.4. Kết luận***

Chúng ta đã hoàn thành bài toán phân lớp từ load data, tạo model và tiến hành huấn luyện mô hình cho tập dữ liệu Fashion MNIST với độ chính xác là 88,4 % tương đối cao. Chính vì thế mà ngày nay việc áp dụng kỹ thuật này đang trở nên vô cùng phổ biến cũng như tầm quan trọng của nó

* Chuẩn đoán ảnh X-ray của bệnh nhân có bị ung thư hay không
* Phân loại, nhận diện được các chữ, số viết tay => tự động đọc được biển số xe, văn bản.
* Phân loại được các biển báo giao thông => hỗ trợ cho ô tô tự lái

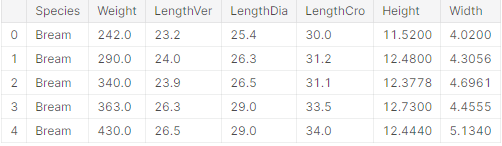
## **2. Bài toán dự đoán cân nặng của cá**

### ***2.1 Mô tả dữ liệu***

Bộ dữ liệu cá được tải tại: <https://www.kaggle.com/aungpyaeap/fish-market>

Bộ dữ liệu có 7 loài cá khác nhau phổ biến trong một cửa hàng cá. Với bộ dữ liệu này, ta có thể thực hiện mô hình dự đoán và ước lượng được cân nặng của cá.

Bộ dữ liệu gồm 159 con cá với 7 cột dữ liệu

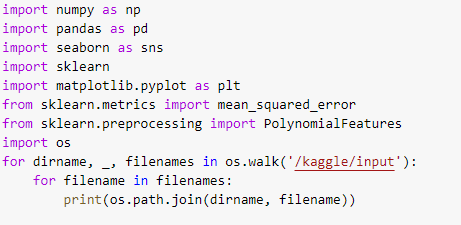


### ***2.2 Mô tả bài toán học máy***

Bài toán hồi quy sử dụng kỹ thuật học máy hồi quy tuyến tính Linear Regression để dự đoán cân nặng của cá dựa vào (loài, chiều cao, chiều dài đường chéo,chiều dài chéo, chiều rộng đường chéo, chiều dài dọc)

### ***2.3 Code bài toán***

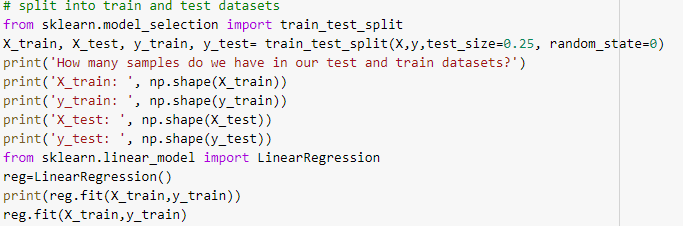
Cài đặt các thư viện cần dùng trong bài toán



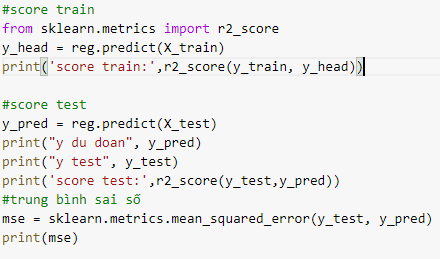
Tải bộ dữ liệu và đọc 10 dòng đầu tiên của bộ



Chia tập dữ liệu thành train và test, sử dụng kỹ thuật hồi quy tuyến tính để dự đoán trọng lượng của cá.



Các phần trăm train và test, sai số thực tế sau khi dự đoán.



### ***2.4 Kết luận***

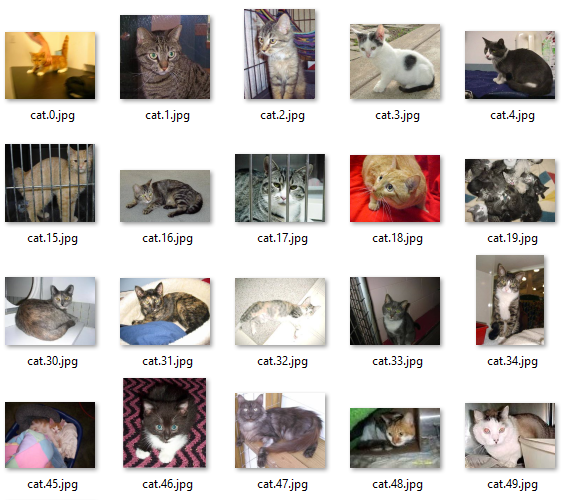
Trong toán học, quy hoạch tuyến tính là bài toán tối ưu hóa. Vì vậy, sử dụng để dự đoán cân nặng của cá sẽ có độ chính xác cao. Tuy nhiên, chúng ta phải xác nhận lại độ sai số thực tế để loại trừ những dữ liệu nằm ngoài phạm vi để có được độ chính xác nhất.

## **3. Bài toán phân loại ảnh chó mèo**

### ***3.1. Mô tả dữ liệu***

Bộ dữ liệu chó mèo tại nguồn: <https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip>

chứa hơn 5000 ảnh chó mèo màu khác nhau với hàng là 32, size ảnh là (160; 160)



|  |  |
| --- | --- |
| **label** | **class** |
| 0 | cat |
| 1 | dog |

### ***3.2. Mô tả bài toán học máy***

Bài toán sử dụng phân lớp nhị phân dựa vào đó phân loại ảnh chó mèo

Kỹ thuật sử dụng: Transfer learning và fine-tuning

### ***3.3. Mô tả bài toán học máy***

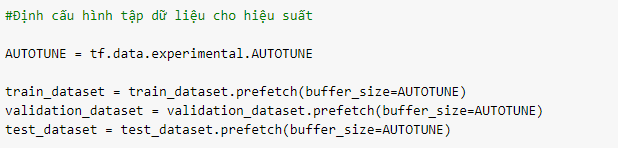


Mô hình được đào tạo trước là một mạng đã lưu đã được đào tạo trước đó trên một tập dữ liệu lớn, thường là trong một nhiệm vụ phân loại hình ảnh quy mô lớn. Bạn có thể sử dụng mô hình được đào tạo trước như hiện tại hoặc sử dụng học chuyển giao để tùy chỉnh mô hình này cho một nhiệm vụ nhất định.

Trực giác đằng sau việc học truyền để phân loại hình ảnh là nếu một mô hình được đào tạo trên một tập dữ liệu đủ lớn và tổng quát, thì mô hình này sẽ đóng vai trò là một mô hình chung của thế giới hình ảnh một cách hiệu quả. Sau đó, bạn có thể tận dụng các bản đồ đặc trưng đã học này mà không cần phải bắt đầu lại từ đầu bằng cách đào tạo một mô hình lớn trên một tập dữ liệu lớn.

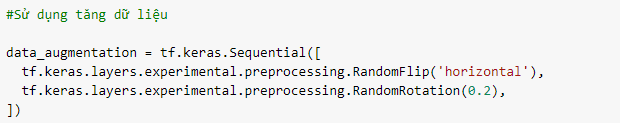
Định cấu hình tập dữ liệu cho hiệu suất

Sử dụng tìm nạp trước có bộ đệm để tải hình ảnh từ đĩa mà không bị chặn I / O. Để tìm hiểu thêm về phương pháp này, hãy xem hướng dẫn [hiệu suất dữ liệu](https://www.tensorflow.org/guide/data_performance) .



Sử dụng tăng dữ liệu

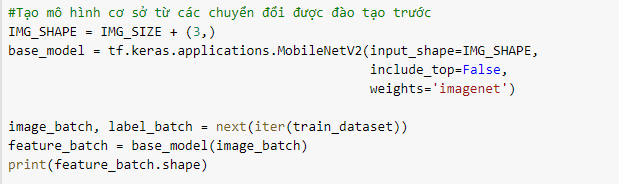
Khi bạn không có tập dữ liệu hình ảnh lớn, bạn nên đưa vào một cách giả tạo tính đa dạng mẫu bằng cách áp dụng các phép biến đổi ngẫu nhiên, nhưng thực tế cho hình ảnh huấn luyện, chẳng hạn như xoay và lật ngang. Điều này giúp mô hình tiếp xúc với các khía cạnh khác nhau của dữ liệu đào tạo và giảm việc trang bị [quá nhiều](https://www.tensorflow.org/tutorials/keras/overfit_and_underfit) . Bạn có thể tìm hiểu thêm về tăng dữ liệu trong [hướng dẫn](https://www.tensorflow.org/tutorials/images/data_augmentation) này .



Lưu ý: Nếu sử dụng hàm khác tf.keras.applications, hãy nhớ kiểm tra tài liệu API để xác định xem chúng có mong đợi pixel trong [-1,1]hoặc [0,1]hoặc sử dụng preprocess\_inputhàm được bao gồm hay không .

Tạo mô hình cơ sở từ các chuyển đổi được đào tạo trước

Bạn sẽ tạo mô hình cơ sở từ mô hình MobileNet V2 được phát triển tại Google. Điều này được đào tạo trước trên tập dữ liệu ImageNet, một tập dữ liệu lớn bao gồm 1,4 triệu hình ảnh và 1000 lớp. ImageNet là một tập dữ liệu đào tạo nghiên cứu với nhiều loại như jackfruitvà syringe. Cơ sở kiến ​​thức này sẽ giúp chúng tôi phân loại chó và mèo từ tập dữ liệu cụ thể của chúng tôi.

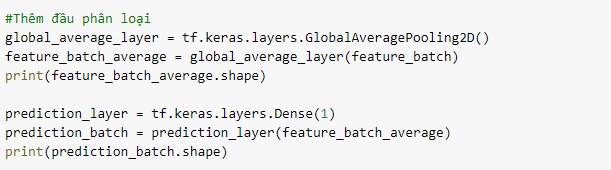


Khai thác tính năng

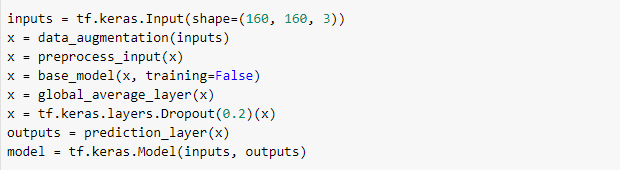
Trong bước này, bạn sẽ đóng băng cơ sở phức hợp được tạo từ bước trước và sử dụng như một trình trích xuất tính năng. Ngoài ra, bạn thêm một bộ phân loại lên trên nó và đào tạo bộ phân loại cấp cao nhất.

Thêm đầu phân loại

Để tạo dự đoán từ khối các đối tượng địa lý, hãy tính trung bình trên các 5x5vị trí không gian vùng, sử dụng một tf.keras.layers.GlobalAveragePooling2Dlớp để chuyển đổi các đối tượng địa lý thành một vectơ 1280 phần tử duy nhất trên mỗi hình ảnh.

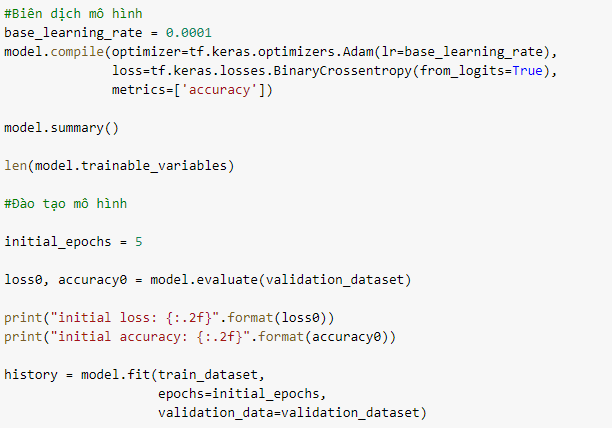


Xây dựng mô hình bằng cách xâu chuỗi các lớp tăng dữ liệu, thay đổi tỷ lệ, base\_model và tính năng với nhau bằng cách sử dụng [API chức năng Keras](https://www.tensorflow.org/guide/keras/functional) . Như đã đề cập trước đây, hãy sử dụng training = False vì mô hình của chúng tôi chứa một lớp BatchNormalization.



Biên dịch mô hình

Biên dịch mô hình trước khi đào tạo nó. Vì có hai lớp, hãy sử dụng suy hao entropy chéo nhị phân from\_logits=True Vì mô hình cung cấp đầu ra tuyến tính.



Đường cong học tập





Đánh giá và dự đoán

Cuối cùng, bạn có thể xác minh hiệu suất của mô hình trên dữ liệu mới bằng cách sử dụng bộ thử nghiệm.

# 

### ***3.4 Kết luận***

**CHƯƠNG 3: KẾT LUẬN**