**Phần 1\_5\_Cách ra quyết định cho từng bài toán**

**Notes:**

* *Về format cấu trúc soạn như sau:*
  + *Lý thuyết…*
  + *Bộ code mấu/ ví dụ …*
  + *Ứng dụng (nếu có)...*
* *Mems làm nhớ note tên để mn dễ contact*

**Mục lục**

[**I. Nội dung chính 1**](#_ypdook5qmr5d)

[**II. Nội dung biên soạn chi tiết 2**](#_2ha3nsql82dj)

[1. Bài toán hồi quy 2](#_winn3q26a07z)

[1.1 Lý thuyết 2](#_8dhhku4qp0o)

[1.2 Code - Linear regression 4](#_5gh5hk5zy88n)

[1.2.1 Numpy array 4](#_f6k2ekxfx36v)

[1.2.2 Tensorflow v2 6](#_d93opkallv79)

[1.3 Ứng dụng 7](#_g0xkhwpwhmn7)

[2. Bài toán phân loại 8](#_oyjrqeg2aq7i)

[2.1 Lý thuyết 8](#_pvur0r79j8z4)

[2.2 Code - Logistic regression 15](#_bc85cb1u6t8t)

[2.2.1 Numpy array - Phân loại nhị phân 15](#_pdebcj42rcs6)

[2.2.2 Tensorflow v2 - Phân loại nhiều class 17](#_anfopqytawcz)

[2.3 Ứng dụng 18](#_o0toe8g2m6ma)

[3. Bài toán phân đoạn 19](#_1ju9mbtghhzl)

[3.1. Lý thuyết 19](#_89a8ydkjvrq1)

[3.2. Code 21](#_1uj72rvpfl7j)

[3.3. Ứng dụng 25](#_70e8kq34q1er)

[4. Bài toán phát hiện 26](#_aszzbyqrdvyt)

[4.1 Lý thuyết 26](#_7l540ao8x156)

[4.2. Code - YOLO v1 30](#_89ldaq8fb01l)

[4.3. Ứng dụng 33](#_cqn4od7z1a8f)

[5. Bài toán sinh ảnh 33](#_lw9bqhb05q7s)

[5.1 Lý thuyết 33](#_fsp8cywqrv9y)

[5.2 Code 35](#_12z3xnk1zzjx)

[5.3 Ứng dụng 38](#_9p4tlvvjwpgy)

## 

## **I. Nội dung chính**

Về dạng bài toán hay công việc (tasks): **phân loại**, **hồi quy**, **phát hiện**, **phân** **đoạn**, **sinh ảnh**. Cho mỗi dạng bài toán, cần hiểu rõ:

* + (5) Cách ra quyết định cho từng bài toán
    - * Ví dụ: với hồi quy: tính ra đáp bằng tính kỳ vọng từ phân phối (đầu ra của mô hình)
      * Ví dụ: với phân loại: **softmax** và sau đó là **argmax** hay **top-k** lớp có confidence cao nhất
      * Ví dụ: với phân đoạn: **softmax** và sau đó là **argmax**
      * Ví dụ: với phát hiện: **non-maxima-suppression**
      * Ví dụ: với sinh ảnh: **sample latent variable** + [**decoder/gan/reverse**]

## **II. Nội dung biên soạn chi tiết**

### **1. Bài toán hồi quy**

#### **1.1 Lý thuyết**

<https://www.linkedin.com/pulse/regression-analysis-machine-learning-explained/>

Phân tích hồi quy cho phép chúng ta khám phá mối quan hệ giữa các biến và đưa ra dự đoán dựa trên những mối quan hệ đó. Phân tích hồi quy là một phương pháp thống kê được sử dụng để mô phỏng mối quan hệ giữa một **biến phụ thuộc** và một hoặc nhiều b**iến độc lập**.

**Biến phụ thuộc** đại diện cho kết quả chúng ta muốn dự đoán hoặc hiểu, trong khi **biến độc lập** là các yếu tố có thể ảnh hưởng đến biến phụ thuộc. Bằng cách phân tích những mối quan hệ này, phân tích hồi quy giúp chúng ta ước lượng giá trị cho biến phụ thuộc dựa trên các giá trị đã biết của biến độc lập.

Có nhiều loại phân tích hồi quy thường được sử dụng trong máy học. Một trong những loại đó là **hồi quy tuyến tính** (**linear regression**), giả sử mối quan hệ tuyến tính giữa biến phụ thuộc và biến độc lập. Hồi quy tuyến tính nhằm tìm ra một đường thẳng phù hợp nhất với các điểm dữ liệu bằng cách giảm thiểu tổng bình phương của sự khác biệt giữa các giá trị quan sát và giá trị dự đoán.

Một loại khác là **hồi quy đa thức** (**polynomial regression**), cho phép mối quan hệ phi tuyến giữa các biến bằng cách giới thiệu các hạng tử bậc cao vào phương trình mô hình. Hồi quy đa thức có thể nắm bắt những mô hình phức tạp hơn trong dữ liệu so với hồi quy tuyến tính, nhưng có thể dễ bị quá mức khi không được kiểm soát đúng cách.

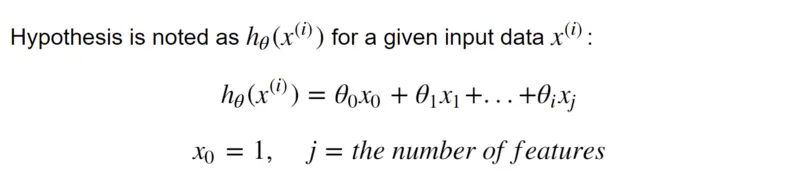
**Hồi quy tuyến tính đa biến** (**Multiple linear regression**) liên quan đến nhiều biến độc lập và nhằm tìm ra mối quan hệ tuyến tính giữa chúng và biến phụ thuộc. Nó xem xét cách mỗi biến độc lập đóng góp vào việc dự đoán hoặc giải thích sự thay đổi trong biến phụ thuộc.

Hiểu biết về những loại hồi quy khác nhau này là quan trọng vì chúng cho phép chúng ta chọn thuật toán phù hợp dựa trên tập dữ liệu và vấn đề cụ thể. Mỗi thuật toán có những ưu điểm và hạn chế riêng, vì vậy việc chọn một thuật toán phù hợp là quan trọng để có dự đoán chính xác.

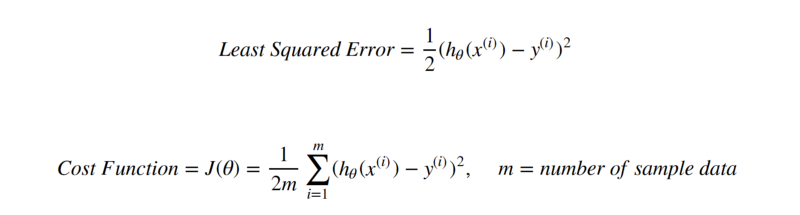
Các thành phần cơ bản của một mô hình hồi quy điển hình bao gồm:

1. **Biến phụ thuộc**: Đây là biến chúng ta muốn dự đoán hoặc giải thích. Trong trường hợp của chúng ta, nó có thể là bất cứ điều gì từ giá cổ phiếu đến giá nhà hoặc thậm chí tỷ lệ mất khách hàng.
2. **Biến độc lập**: Đây là các yếu tố có thể ảnh hưởng đến biến phụ thuộc. Chúng có thể bao gồm giá trị số như tuổi hoặc thu nhập, biến phân loại như giới tính hoặc địa điểm, hoặc sự kết hợp giữa cả hai.
3. **Hệ số**: Đây là các tham số đại diện cho mối quan hệ giữa biến độc lập và biến phụ thuộc. Chúng chỉ ra làm thế nào một sự thay đổi đơn vị trong biến độc lập ảnh hưởng đến biến phụ thuộc.

<https://towardsdatascience.com/optimization-of-supervised-learning-loss-function-under-the-hood-df1791391c82>

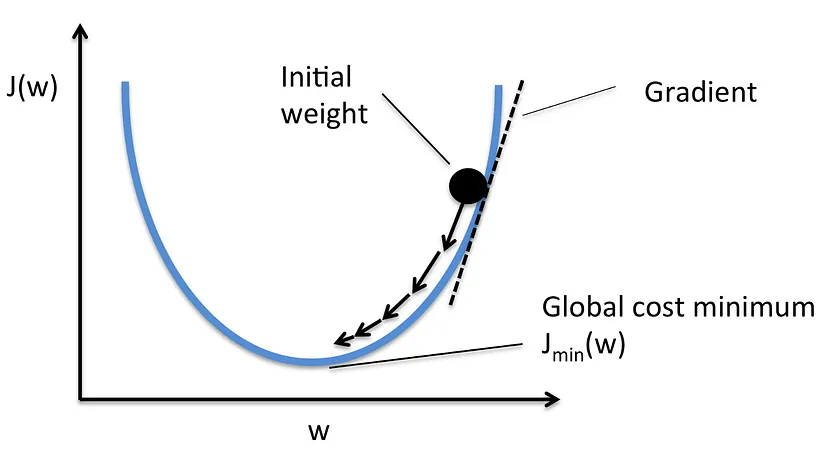


Trên là hàm giả thuyết cho mô hình hồi quy tuyến tính. Với

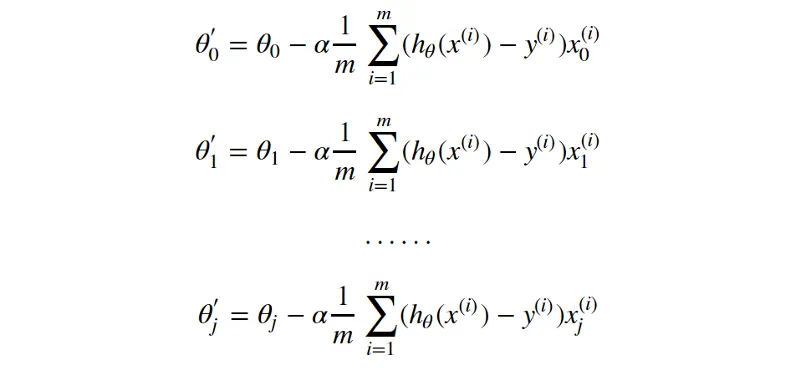


Và loss function của nó

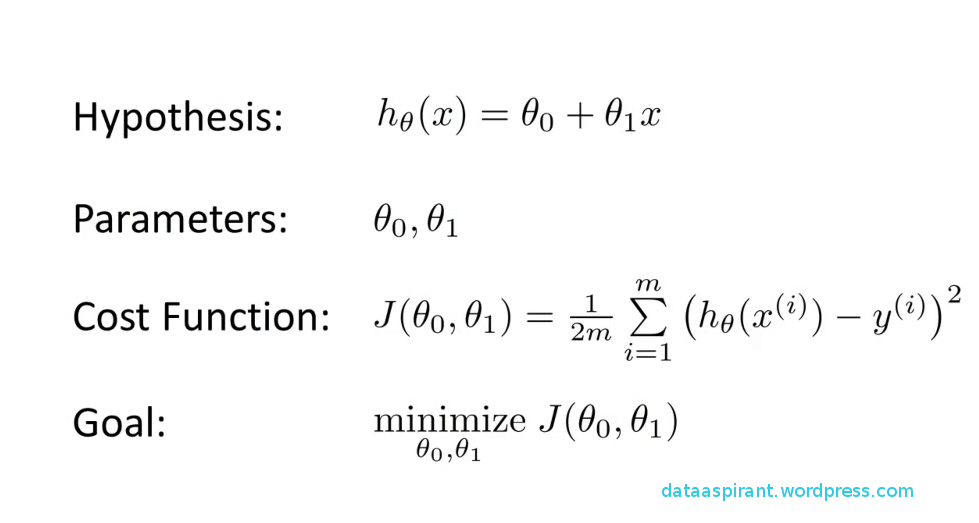




Như chúng ta có thể thấy từ công thức, loss function là một đường cong parabol. Để giảm thiểu nó, chúng ta cần tìm đỉnh của nó. Nó có thể được giải phân tích hoặc bằng cách sử dụng các thuật toán lập trình. Để tìm vị trí cực tiểu của hàm loss này ta sẽ sử dụng Gradient Descent. Gradient Descent được sử dụng rộng rãi trong các mô hình khác nhau, thực hiện các bước dựa trên một tốc độ học α, di chuyển về phía đỉnh của đường cong parabol để tìm ra giá trị tối thiểu toàn cục cho Hồi quy Tuyến tính, cũng được biết đến là điểm có độ dốc bằng 0. Vì vậy, để có độ dốc, chúng ta lấy đạo hàm của hàm chi phí đối với từng hệ số θ.



Tiếp theo, liên tục cập nhật mỗi θ với một lượng lớn lặp cùng với quan sát về giảm giá trị của hàm chi phí cho đến khi nó đạt đến một đường ngang. Trong thực tế, không thể đạt được một giá trị chính xác của độ dốc bằng 0 do kích thước bước đã chọn, nhưng nó có thể gần với 0 càng nhiều tùy thuộc vào siêu tham số.



Tóm lại quy trình sẽ như trên

#### **1.2 Code - Linear regression**

##### **1.2.1 Numpy array**

import numpy as np

m, n = X\_train.shape

ones = np.ones((m, 1))

X\_train = np.concatenate((X\_train, ones), axis=1)

theta = np.random.normal(size=X\_train.shape[1]).reshape((X\_train.shape[1], 1))

y\_train = y\_train.values.reshape(-1, 1)

Code trên cho thấy các features được chứa trong X\_train, tuy nhiên để sử dụng trong thuật toán thì X\_train này cần phải concatenate cho vector 1.



Số 1 này sẽ đại diện cho việc nhân cho theta0.

from sklearn.utils import shuffle

learning\_rate = 0.01

def hypo(X, theta):

return X.dot(theta)

def cost(X, y, theta, m):

return 1/(2\*m)\*np.sum((hypo(X,theta)-y)\*\*2)

def SGD(X, y, theta, learning\_rate, epochs):

m = len(X)

for i in range(epochs):

X,y = shuffle(X,y)

for k in range(m):

error = (hypo(X[k],theta) - y[k])[0]

for j in range(len(theta)):

theta[j] = theta[j] - learning\_rate\*error\*(X[k][j])

if i % 100 == 0:

print('Loss là: {}' .format(cost(X, y, theta, m)))

Trên là code biểu thị cách cập nhật theta trong mô hình hồi quy tuyến tính.

Hàm cost dựa theo công thức loss function đề cập trong phần lý thuyết

SGD là thuật toán stocastic gradient descent, tại mỗi epoch thì feature X và label y sẽ bị xáo trộn. Sau đó sẽ lặp qua từng điểm dữ liệu, tại mỗi điểm dữ liệu ta sẽ tính toán chênh lệch giữa giá trị dự đoán và thực tế sau đó cho vào biến error. Sau đó sẽ tiến hành cập nhật.

##### **1.2.2 Tensorflow v2**

<https://www.theclickreader.com/linear-regression-in-tensorflow-2-0/>

class Model:

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

# Initializing variables weight(W) and bias(b)

self.W = tf.Variable(16.0)

self.b = tf.Variable(10.0)

def \_\_call\_\_(self, x):

return self.W \* x + self.b

Phương thức init khởi tạo trọng số và độ lệch cho mô hình hồi quy tuyến tính. Phương thức gọi trả về giá trị dự đoán, theo phương trình $y = W\*x + b$.

def loss(y, y\_pred):

return tf.reduce\_mean(tf.square(y - y\_pred))

Trong mô hình trên, trọng số và độ lệch được khởi tạo ngẫu nhiên. Chúng ta sẽ đào tạo mô hình này để xác định giá trị tốt nhất cho trọng số và độ lệch. Để làm điều đó, chúng ta cần một hàm mất mát sẽ chỉ ra mức độ hiệu suất của mô hình. Trong trường hợp này, chúng ta sẽ sử dụng hàm mất mát mean square error (sai số bình phương trung bình).

def train(model, x, y, lr=0.1):

with tf.GradientTape() as t:

# Get prediction

y\_hat = model(x)

# Compute the loss

current\_loss = loss(y, model(x))

# Find the gradient

grad\_W, grad\_b = t.gradient(current\_loss, [model.W, model.b])

# Update weight

model.W.assign\_sub(lr \* grad\_W)

# Update Bias

model.b.assign\_sub(lr \* grad\_b)

Bây giờ, cho mỗi vòng lặp (epoch) trong quá trình huấn luyện mô hình, chúng ta cần:

Tính toán độ dốc của các tham số của mô hình đối với hàm mất mát: Phương thức tf.GradientTape() ghi lại tất cả các hoạt động đang được thực hiện bên trong ngữ cảnh quản lý. Điều này là cần thiết khi tính toán độ dốc.

Cập nhật các tham số của mô hình: Sau khi tính toán độ dốc của $W$ và $b$, chúng ta nhân độ dốc với một tỷ lệ học và trừ kết quả này từ giá trị hiện tại của $W$ và $b$.



Sau khi có W và b ta có thể dùng nó để dự đoán Y=W\*X+b.

#### **1.3 Ứng dụng**

<https://www.knowledgehut.com/blog/data-science/linear-regression-for-machine-learning#beginner-projects%C2%A0to-try-out-linear%C2%A0regression%C2%A0> ,

**Dự Án 1: Dự Đoán Nợ Vay**

Ngân hàng sử dụng Máy học để dự đoán nợ vay và quyết định về đơn đăng ký vay. Việc cho vay dưới dạng khoản vay là một nguồn thu nhập chính cho các ngân hàng, hợp tác xã tín dụng và tổ chức tài chính khác, chiếm một phần lớn trong tài sản của ngân hàng. Tuy nhiên, khi những khoản vay này không đảm bảo, các tổ chức tài chính phải đối mặt với hậu quả nặng nề. Tải về Cơ sở dữ liệu Dự đoán Nợ vay.

**Dự Án 2: Dự Đoán Giá Nhà**

Dự đoán giá nhà có thể giúp xác định giá bán của một tài sản ở một địa điểm cụ thể và xác định thời điểm tốt nhất để mua nhà. Tải về Cơ sở dữ liệu Dự đoán Giá nhà.

**Dự Án 3: Dự Đoán Thị Trường Chứng Khoán**

Phân tích và dự báo thị trường chứng khoán là công việc rất khó khăn. Giá cổ phiếu thay đổi liên tục và bị ảnh hưởng bởi nhiều yếu tố. Khi dự báo cổ phiếu, hầu hết các chứng khoán sử dụng phân tích phương pháp và cơ bản và phân tích chuỗi thời gian. Tải về Cơ sở dữ liệu Dự đoán Thị trường Chứng khoán.

**Dự Án 4: Dự Báo Doanh Số Thị Trường**

Dự báo doanh số bán hàng quan trọng vì nó giúp các công ty khám phá xem chiến lược nào hoạt động hiệu quả và nơi nào cần điều chỉnh chiến lược cụ thể để đảm bảo thành công trong tương lai. Tải về Cơ sở dữ liệu Dự Báo Doanh số thị trường.

**Dự Án 5: Quảng cáo**

Các doanh nghiệp thường quảng cáo sản phẩm của họ qua trang web và các kênh truyền thông xã hội. Tuy nhiên, thách thức chính của họ nằm ở việc tìm ra đối tượng phù hợp để tập trung vào tiếp thị trực tuyến. Vì quảng cáo rất đắt đỏ, việc chạm mục tiêu quảng cáo vào một khách hàng không có khả năng mua sản phẩm có thể là một mất mát cho công ty. Tải về Cơ sở dữ liệu Quảng cáo.

### **2. Bài toán phân loại**

#### **2.1 Lý thuyết**

<https://machinelearningmastery.com/types-of-classification-in-machine-learning/>

Trong máy học, phân loại đề cập đến một vấn đề mô hình dự đoán, trong đó một nhãn lớp được dự đoán cho một ví dụ cụ thể của dữ liệu đầu vào.

Các ví dụ về vấn đề phân loại bao gồm:

1. Cho một ví dụ, phân loại xem nó có phải là thư rác hay không.

2. Cho một ký tự viết tay, phân loại nó là một trong các ký tự đã biết.

3. Dựa trên hành vi người dùng gần đây, phân loại xem đó là việc rời bỏ dịch vụ (churn) hay không.

Từ góc độ mô hình hóa, phân loại đòi hỏi một bộ dữ liệu đào tạo với nhiều ví dụ về dữ liệu đầu vào và đầu ra từ đó mô hình có thể học.

Một mô hình sẽ sử dụng bộ dữ liệu đào tạo và sẽ tính toán cách tốt nhất để ánh xạ các ví dụ của dữ liệu đầu vào vào các nhãn lớp cụ thể. Do đó, bộ dữ liệu đào tạo phải đại diện đầy đủ vấn đề và có nhiều ví dụ cho mỗi nhãn lớp.

Nhãn lớp thường là giá trị chuỗi, ví dụ: "thư rác," "không phải thư rác," và phải được ánh xạ thành giá trị số trước khi được cung cấp cho một thuật toán để mô hình hóa. Điều này thường được gọi là mã hóa nhãn, nơi một số nguyên duy nhất được gán cho mỗi nhãn lớp, ví dụ: "thư rác" = 0, "không phải thư rác" = 1.

Có nhiều loại thuật toán phân loại khác nhau để mô hình hóa các vấn đề dự đoán phân loại.

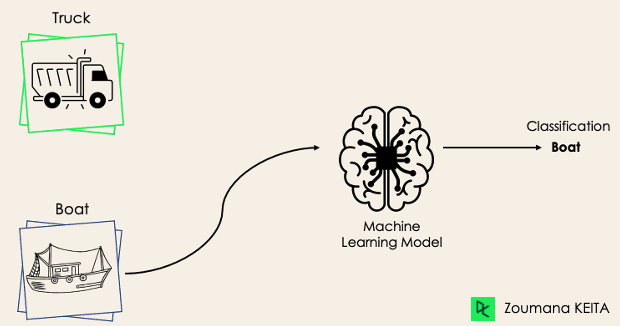
Không có lý thuyết tốt về cách ánh xạ các thuật toán vào loại vấn đề; thay vào đó, thường khuyến khích người thực hành sử dụng thí nghiệm kiểm soát và khám phá xem thuật toán nào và cấu hình thuật toán mang lại hiệu suất tốt nhất cho một nhiệm vụ phân loại cụ thể.

Các thuật toán dự đoán phân loại được đánh giá dựa trên kết quả của chúng. Độ chính xác phân loại là một độ đo phổ biến được sử dụng để đánh giá hiệu suất của mô hình dựa trên các nhãn lớp dự đoán. Độ chính xác phân loại không hoàn hảo nhưng là một điểm xuất phát tốt cho nhiều nhiệm vụ phân loại.

Thay vì các nhãn lớp, một số nhiệm vụ có thể yêu cầu dự đoán xác suất của việc thuộc một lớp cho mỗi ví dụ. Điều này cung cấp sự không chắc chắn bổ sung trong dự đoán mà một ứng dụng hoặc người dùng sau đó có thể giải thích. Một công cụ phổ biến để đánh giá xác suất dự đoán là Đường cong ROC.

<https://www.datacamp.com/blog/classification-machine-learning>

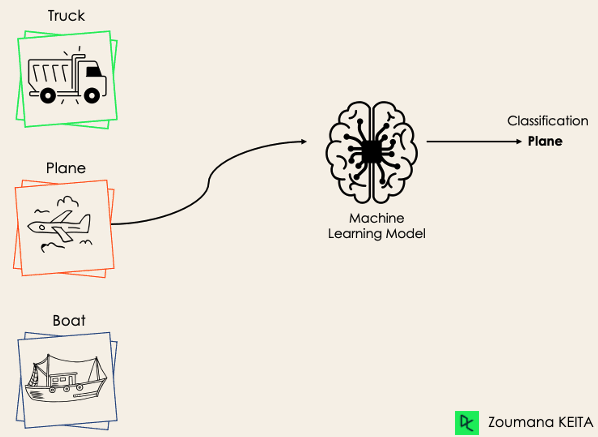
Có bốn loại nhiệm vụ phân loại chính bạn có thể gặp phải; chúng là:



1. **Phân loại nhị phân (Binary Classification)**: đề cập đến những nhiệm vụ phân loại có hai nhãn lớp.

Thuật toán thường dùng:

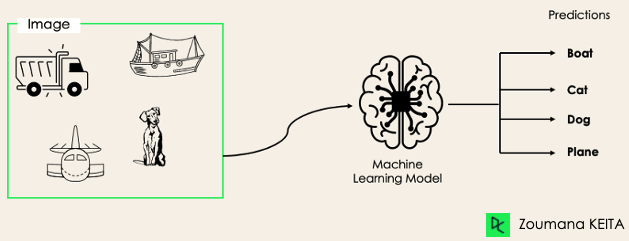
* Logistic Regression
* k-Nearest Neighbors
* Decision Trees
* Support Vector Machine
* Naive Bayes



2. **Phân loại đa lớp (Multi-Class Classification)**: đề cập đến những nhiệm vụ phân loại có nhiều hơn hai nhãn lớp.

Thuật toán thường dùng:

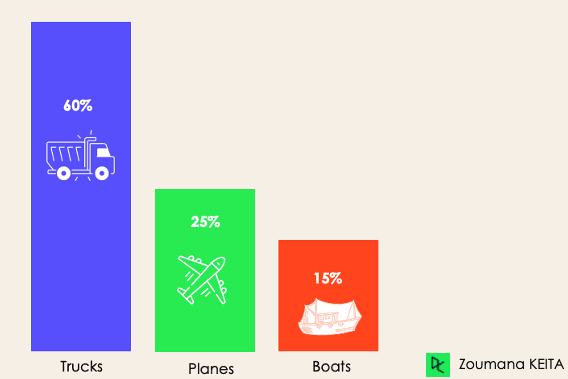
* k-Nearest Neighbors.
* Decision Trees.
* Naive Bayes.
* Random Forest.
* Gradient Boosting.



3. **Phân loại đa nhãn (Multi-Label Classification)**: đề cập đến những nhiệm vụ phân loại có hai hoặc nhiều nhãn lớp, trong đó một hoặc nhiều nhãn lớp có thể được dự đoán cho mỗi ví dụ.

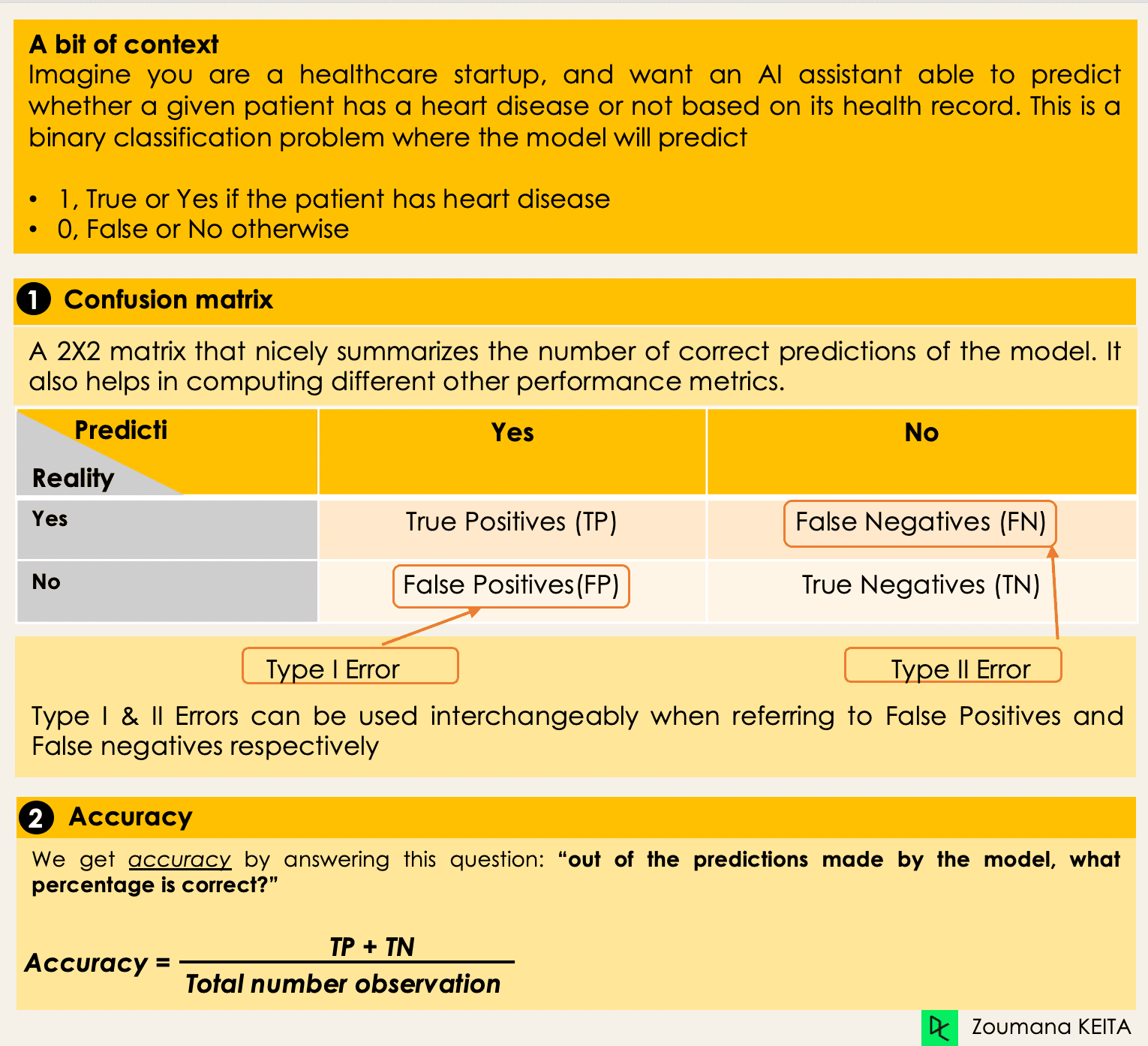
Thuật toán thường dùng:

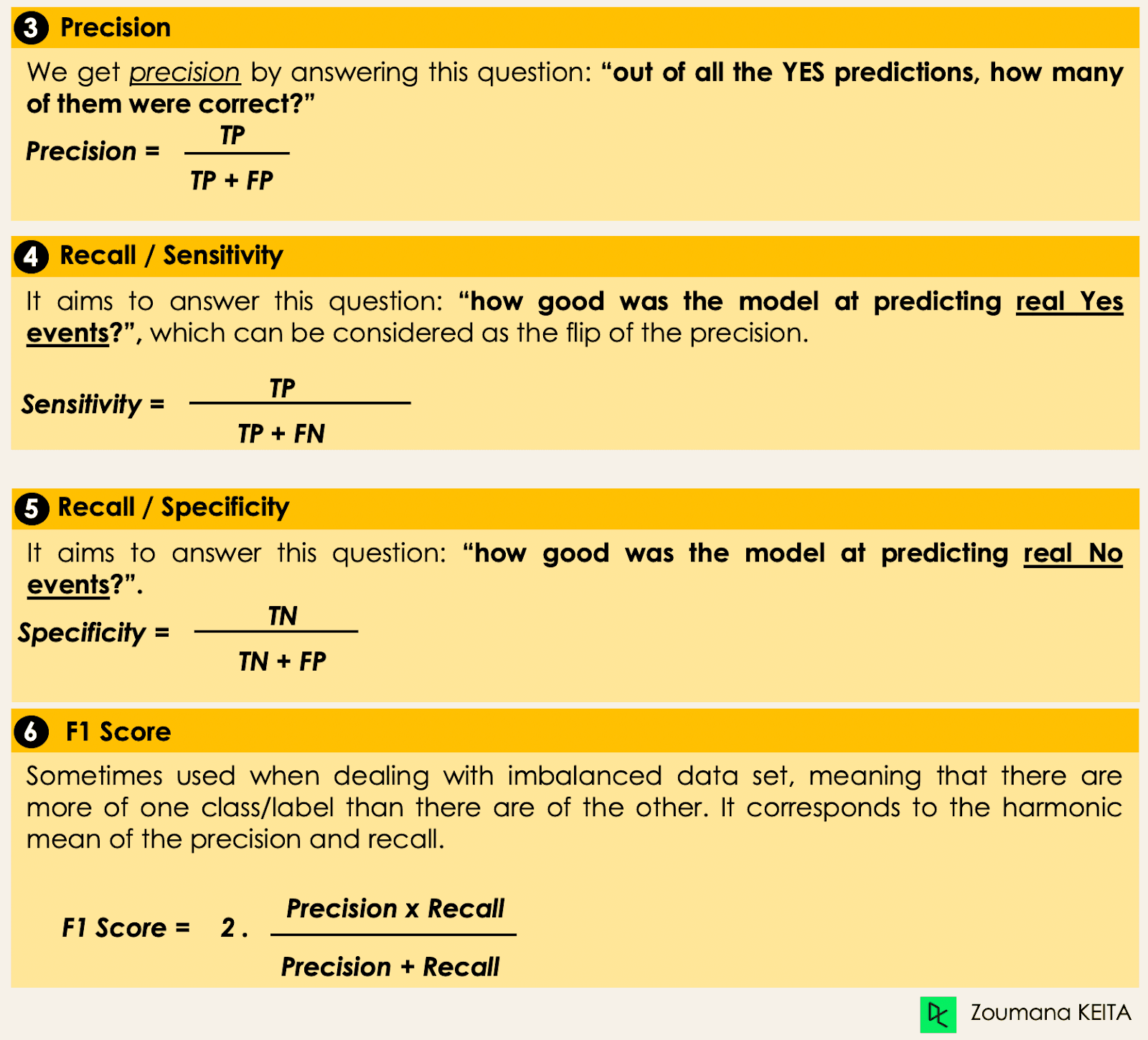
* Multi-label Decision Trees
* Multi-label Random Forests
* Multi-label Gradient Boosting

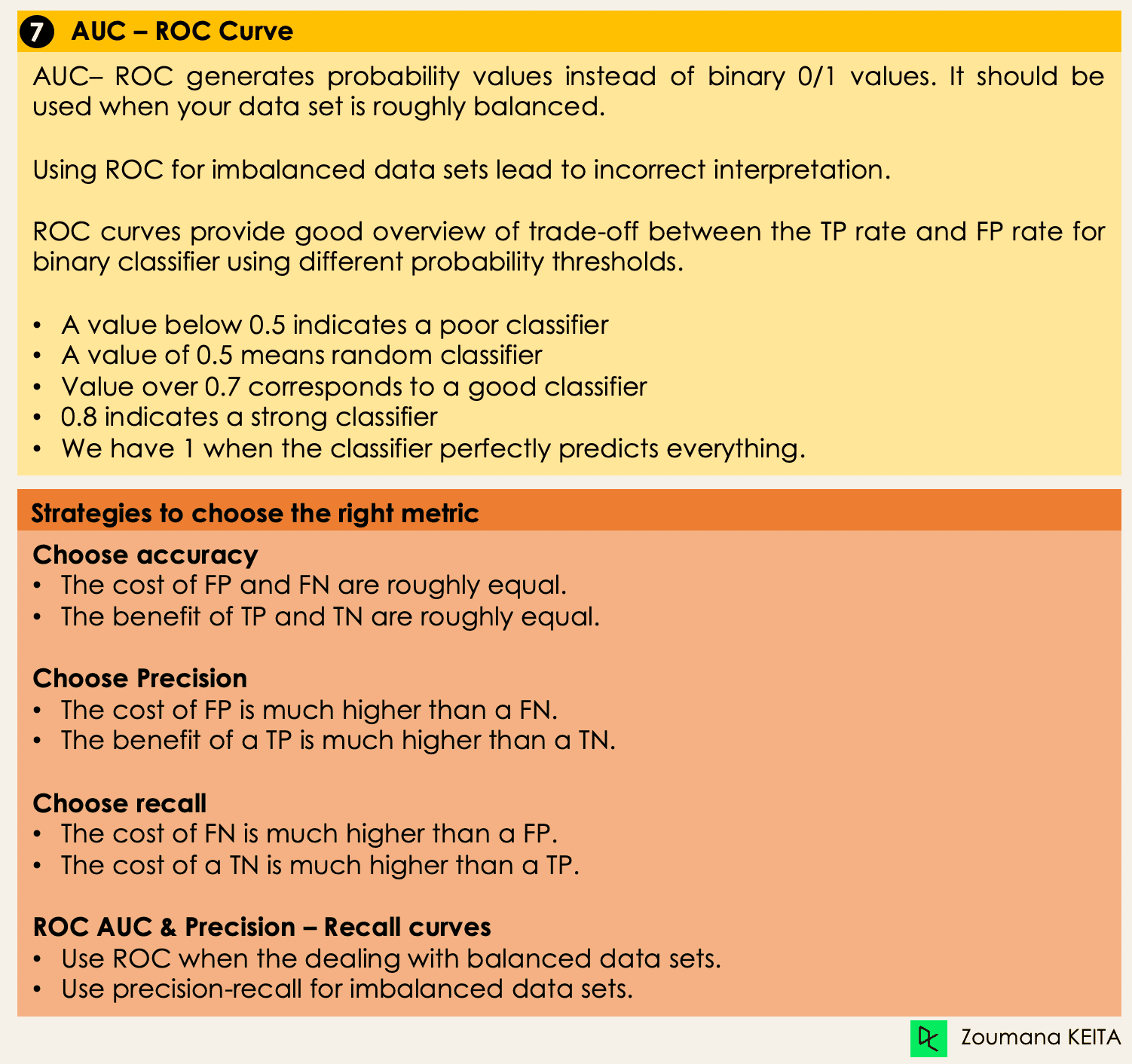


4. **Phân loại mất cân bằng (Imbalanced Classification)**: đề cập đến các nhiệm vụ phân loại trong đó số lượng mẫu trong mỗi lớp được phân bổ không đồng đều. Thông thường, nhiệm vụ phân loại không cân bằng là nhiệm vụ phân loại nhị phân trong đó phần lớn các mẫu trong tập dữ liệu huấn luyện thuộc về lớp bình thường và một số ít mẫu thuộc về lớp bất thường.

Các chỉ số được dùng để đánh giá độ phù hợp của mô hình:

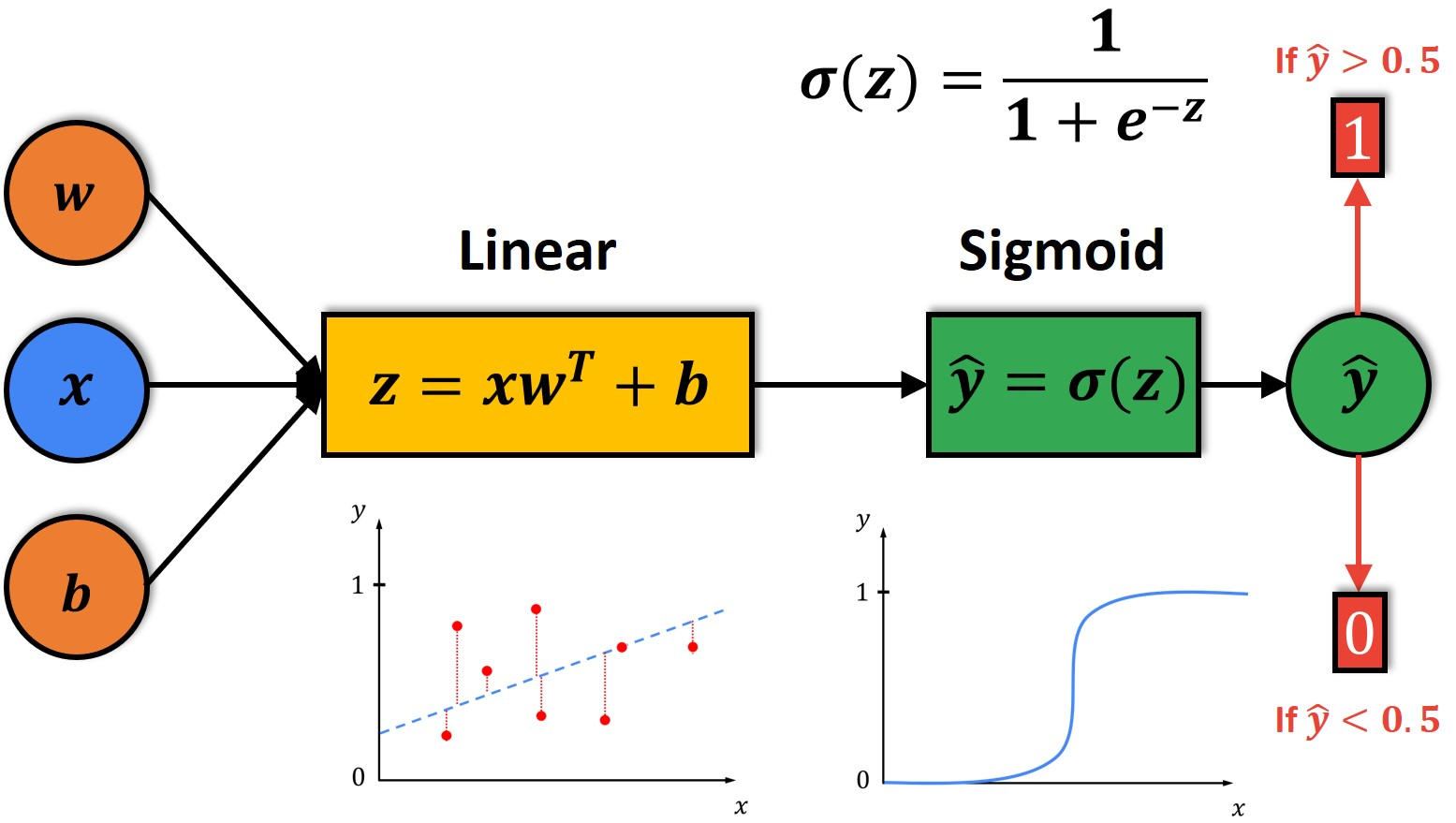




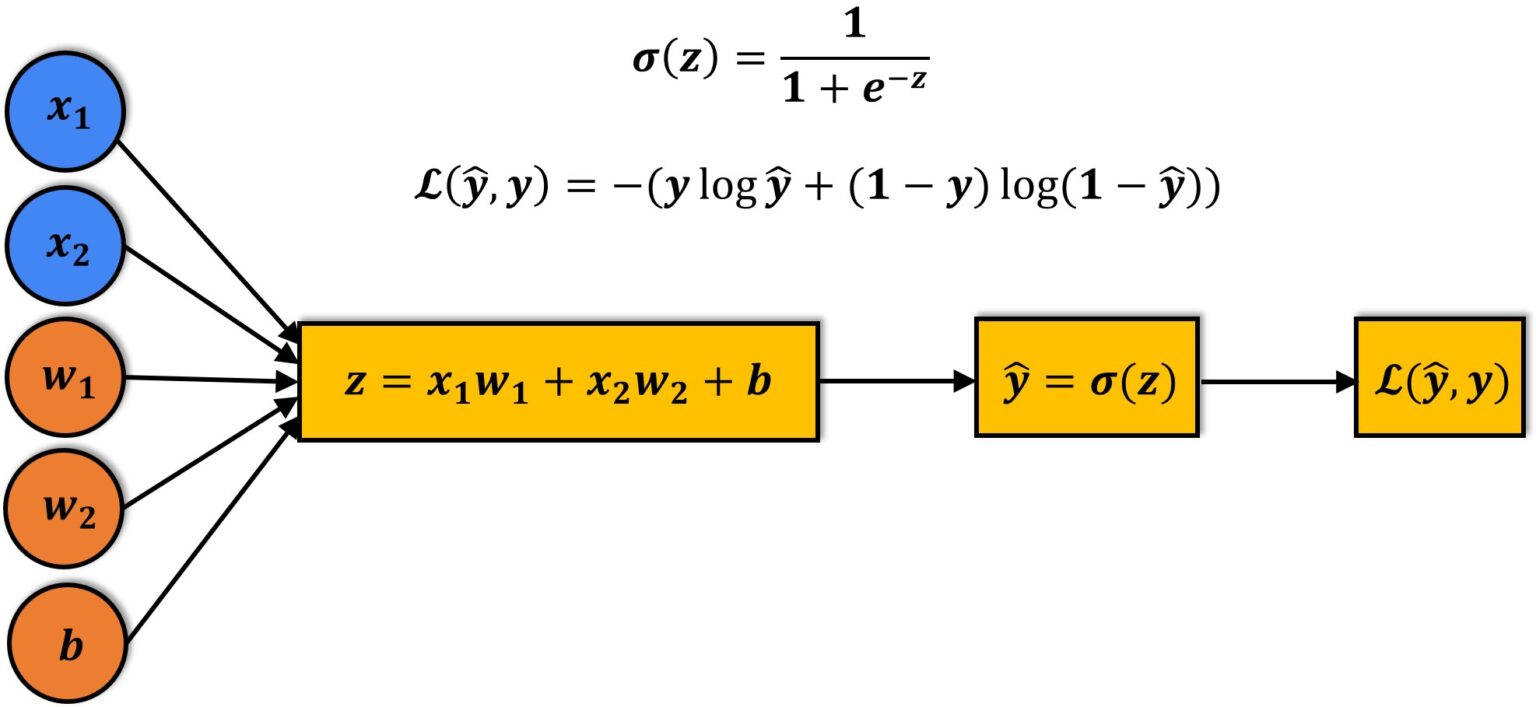


#### **2.2 Code - Logistic regression**

##### **2.2.1 Numpy array - Phân loại nhị phân**



Công thức hàm hồi quy logistic



Hàm mất mát

def cal\_z(x, theta):

return x.dot(theta)

Hàm tính z value

def sigmoid(z):

return 1/(1+np.exp(-z))

Hàm sigmoid

def cal\_h\_theta(x, theta):

return sigmoid(cal\_z(x, theta))

Hàm cal\_h\_theta là hàm hợp của cal\_z và sigmoid

def cost\_func(x, y, theta):

loss = -1\*(np.sum(y\*np.log(cal\_h\_theta(x, theta))+(1-y)\*np.log(1-cal\_h\_theta(x, theta))))

return np.mean(loss)

Loss function của logistic regression

def gradient(X, y, theta):

return X.T.dot((cal\_h\_theta(X, theta)-y))

Hàm tính toán gradient

epochs = 1000

learning\_rate = 0.001

theta = np.random.normal(size=3).reshape((3, 1))

for e in range(epochs):

grad = gradient(trainX,trainY,theta)

theta = theta-learning\_rate\*grad

total\_cost = cost\_func(trainX,trainY,theta)

if e % 100 == 0:

print(total\_cost)

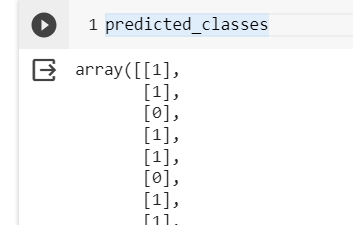
print('Training finished')

Trên là đoạn code dùng Gradient Descent để cập nhật theta

Đầu tiên sẽ khởi tạo tham số theta

Tiếp theo đó sẽ lặp qua từng epoch dữ liệu. Mỗi epoch thì toàn bộ dữ liệu X,Y và theta sẽ được dùng để tính toán gradient và cho vào biến grad

Theta sẽ được cập nhật



predicted\_classes = (cal\_h\_theta(testX, theta) >= 0.5).astype(int)

Có thể dùng các đặc trưng trong X nhân với theta để ra xác suất dự đoán và đưa ra một ngưỡng, như trên 0.5 cho 1 (tức True) và nhỏ hơn 0.5 cho 0 (tức False).

Ta có thể map các giá trị 1 và 0 này tới một dict để lấy class tương ứng của chúng

##### **2.2.2 Tensorflow v2 - Phân loại nhiều class**

<https://www.theclickreader.com/logistic-regression-in-tensorflow-2-0/>

class Model:

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

self.W = tf.Variable(tf.ones([784, 10]), name="weight")

self.b = tf.Variable(tf.zeros([10]), name="bias")

def \_\_call\_\_(self, x):

return tf.nn.softmax(tf.matmul(x, self.W) + self.b)

Bây giờ, chúng ta định nghĩa mô hình logistic regression như một lớp Python với hai phương thức: init và call. Tương tự như mô hình hồi quy tuyến tính, trọng số và độ lệch cho mô hình logistic regression được định nghĩa trong phương thức init trong khi công thức được định nghĩa trong phương thức call.

Vì vectơ đặc trưng đầu vào có 784 giá trị pixel và 10 lớp (số từ 0-9), nên trọng số sẽ có hình dạng [784, 10] và độ lệch sẽ là một vectơ 1 chiều có 10 giá trị. Sau đó, chúng ta sẽ nhân vectơ đầu vào với trọng số và cuối cùng thêm một độ lệch để thu được logits. Cuối cùng, chúng ta sẽ áp dụng hàm softmax để chuẩn hóa logits thành một phân phối xác suất.

def loss(y\_pred, y\_true):

# Encode label to a one hot vector

y\_true = tf.one\_hot(y\_true, depth=10)

# Clip prediction values to avoid log(0) error

y\_pred = tf.clip\_by\_value(y\_pred, 1e-9, 1.)

# Compute cross-entropy

return tf.reduce\_mean(-tf.reduce\_sum(y\_true \* tf.math.log(y\_pred),1))

# Stochastic gradient descent optimizer.

optimizer = tf.optimizers.SGD(lr = 0.1)

Bây giờ, chúng ta sẽ truyền logits được thu được từ mô hình vào một hàm mất mát để đánh giá hiệu suất của mô hình. Trước hết, chúng ta sẽ mã hóa one-hot cho các đầu ra bằng cách sử dụng hàm one\_hot() của TensorFlow.

Sau đó, chúng ta tính mất mát entropy chéo giữa giá trị dự đoán và nhãn one-hot được mã hóa thực tế. Một hàm khác sẽ tính độ chính xác của mô hình của chúng ta. Để cập nhật trọng số và độ lệch ở mỗi lần lặp (epoch), chúng ta sẽ sử dụng bộ tối ưu hóa Stochastic Gradient Descent (SGD).

def train(model, x, y):

with tf.GradientTape() as t:

pred = model(x)

current\_loss = loss(pred, y)

# Compute gradients

gradients = t.gradient(current\_loss, [model.W, model.b])

# Update W and b following gradients.

optimizer.apply\_gradients(zip(gradients, [model.W, model.b]))

Đối với mỗi lần lặp (epoch) trong quá trình đào tạo mô hình, chúng ta cần:

1. Tính gradient của các tham số mô hình đối với hàm mất mát: Phương thức GradientTape() ghi lại tất cả các hoạt động được thực hiện bên trong ngữ cảnh quản lý. Điều này là cần thiết khi tính gradient.

2. Cập nhật các tham số của mô hình: Sau khi tính gradient của $W$ và $b$, cập nhật chúng bằng cách sử dụng bộ tối ưu hóa SGD.

def accuracy(y\_pred, y\_true):

# Predicted class is the index of highest score in prediction vector (i.e. argmax).

correct\_prediction = tf.equal(tf.argmax(y\_pred, 1), tf.cast(y\_true, tf.int64))

return tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

Ta sẽ dùng armax để từ vector one-hot predict chọn index có giá trị lớn nhất và so với y\_true từ đó xác định bao nhiêu giá trị dự đoán là đúng

#### **2.3 Ứng dụng**

Một số đã đề cập trong phần lý thuyết

### **3. Bài toán phân đoạn**

[Nguyễn Trường Thành]

#### **3.1. Lý thuyết**

Trong bài toán object detection, mô hình có nhiệm vụ đi tìm các box bounding quanh các đối tượng trong ảnh và sau đó phân loại các bouding box. Tuy nhiên là các bounding box thì không biểu thị được đúng hình dạng của đối tượng và có nhiều nhiễu ở trong bounding box đấy ví dụ như trong bounding box màu đỏ có cả một phần của cây thông cũng như cái gối.

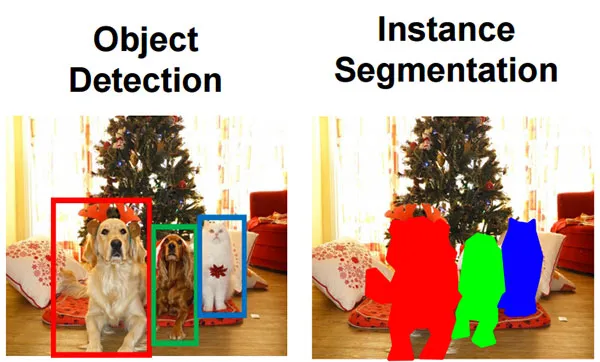
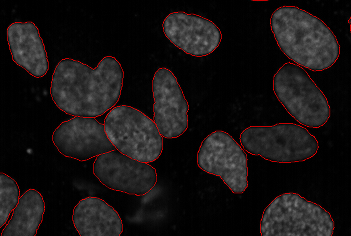


Image segmentation ra đời để chia ảnh thành nhiều vùng khác nhau hay tìm được đúng hình dạng của các đối tượng.

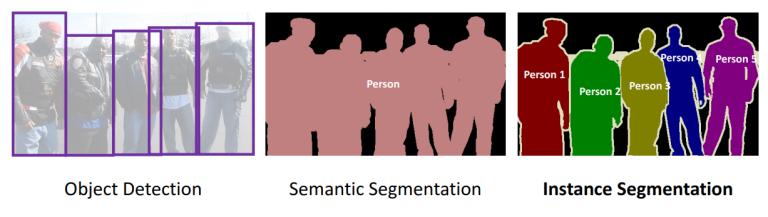
Lấy ví dụ vì sao cần image segmentation. Ung thư là một căn bệnh hiểm nghèo và cần được phát hiện sớm để điều trị. Vì hình dạng của các tế bào ung thư là một trong những yếu tố quyết định độ ác tính của bệnh, nên ta cần image segmentation để biết được chính xác hình dạng của các tế bào ung thư để có các chẩn đoán xác định. Rõ ràng object detection ở đây không giải quyết được vấn đề.



**Phân loại bài toán image segmentation**

Bài toán image segmentation được chia ra làm 2 loại:

* **Semantic segmentation:** Thực hiện segment với từng lớp khác nhau, ví dụ: tất cả người là 1 lớp, tất cả ô tô là 1 lớp.
* **Instance segmentation:** Thực hiện segment với từng đối tượng trong một lớp. Ví dụ có 3 người trong ảnh thì sẽ có 3 vùng segment khác nhau cho mỗi người.



Cần áp dụng kiểu segmentation nào thì phụ thuộc vào bài toán. Ví dụ: cần segment người trên đường cho ô tô tự lái, thì có thể dùng sematic segmentation vì không cần thiết phải phân biệt ai với ai, nhưng nếu cần theo dõi mọi hành vi của mọi người trên đường thì cần instance segmentation thì cần phân biệt mọi người với nhau.

Cấu trúc và nguyên tắc của Phân đoạn Hình ảnh:

**Recognition + Localization**

The image segmentation model first extracts features from the image to segment the image into [ Recognition ]. Recognize the object and [ Localization ] Determine the location. The features we extract here are things like color, texture, and shape. The model used for image segmentation can use these features to partially organize the image, creating a segment map and defining information based on pixel-by-pixel boundaries.

After the image is segmented, the model classifies each segment as either an object or a background. This is done using the Classification feature trained on a dataset of labeled images as objects contained in the image.

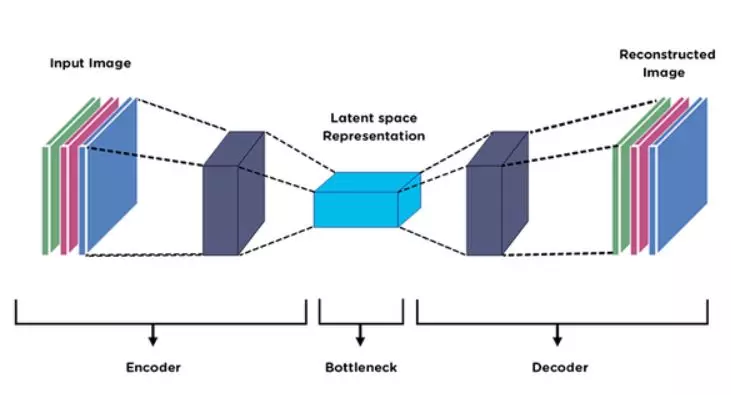
Finally, the model locates the object in the image by identifying the bounding box around the object. A bounding box is a set of four coordinates that define the boundaries of an object.

**Encoder + Decoder**

In the field of computer vision, most image segmentation models consist of an Encoder - Decoder structure. The segment map that comes out of the Decoder can be thought of as a kind of map that represents the location of each object in the image.

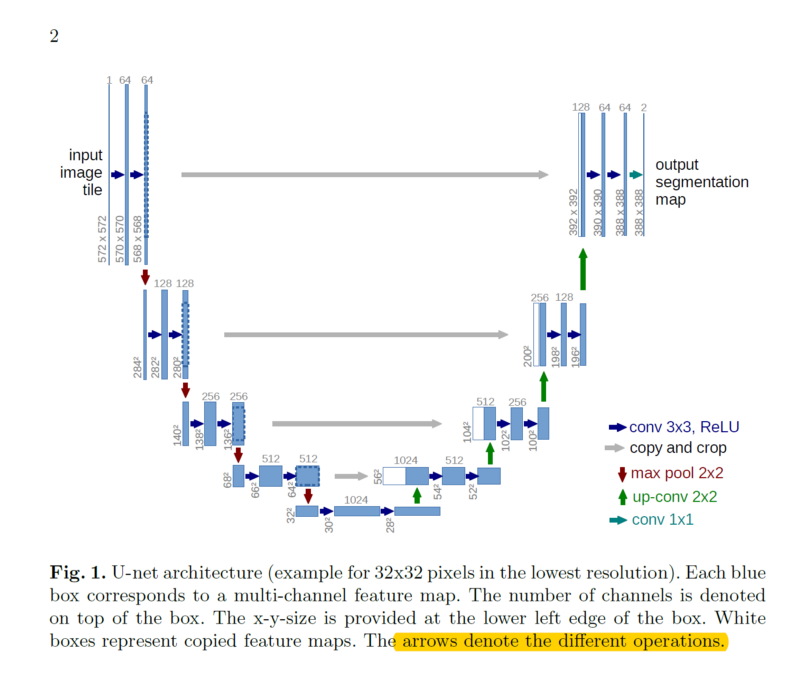
**Encoder:** A layer that extracts images through a series of increasingly narrow and deep filters.

**Decoder:** A layer mask that scales the output of the encoder to a segmentation mask similar to the pixel resolution of the input image.



#### **3.2. Code**

Mạng Unet



inputs = Input(input\_size)

conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(inputs)

conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv1)

pool1 = MaxPooling2D(pool\_size=(2, 2))(conv1)

conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(pool1)

conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv2)

pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)

conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(pool2)

conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv3)

pool3 = MaxPooling2D(pool\_size=(2, 2))(conv3)

conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(pool3)

conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv4)

drop4 = Dropout(0.5)(conv4)

pool4 = MaxPooling2D(pool\_size=(2, 2))(drop4)

conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(pool4)

conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv5)

drop5 = Dropout(0.5)(conv5)

up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(UpSampling2D(size = (2,2))(drop5))

merge6 = concatenate([drop4,up6], axis = 3)

conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(merge6)

conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv6)

up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(UpSampling2D(size = (2,2))(conv6))

merge7 = concatenate([conv3,up7], axis = 3)

conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(merge7)

conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv7)

up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(UpSampling2D(size = (2,2))(conv7))

merge8 = concatenate([conv2,up8], axis = 3)

conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(merge8)

conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv8)

up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(UpSampling2D(size = (2,2))(conv8))

merge9 = concatenate([conv1,up9], axis = 3)

conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(merge9)

conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv9)

conv9 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel\_initializer = 'he\_normal')(conv9)

conv10 = Conv2D(1, 1, activation = 'sigmoid')(conv9)

model = Model(input = inputs, output = conv10)

model.compile(optimizer = Adam(lr = 1e-4), loss = 'binary\_crossentropy', metrics = ['accuracy'])

Trên là mô hình

**import** glob2

test\_paths **=** glob2.glob('unet/data/membrane/test/\*.png')

test\_paths **=** [path **for** path **in** test\_paths **if** 'predict' **not** **in** path]

rand\_ind **=** np.random.randint(5)

path **=** test\_paths[rand\_ind]

**def** **\_predict\_path**(path, figsize **=** (16, 8)):

img **=** cv2.imread(path)

img **=** cv2.resize(img, (512, 512), cv2.INTER\_LINEAR)

img\_expand **=** img[np.newaxis, ...]

img\_pred **=** model3.predict(img\_expand).reshape(512, 512)

img\_pred[img\_pred **<** 0.5] **=** 0

img\_pred[img\_pred **>=** 0.5] **=** 1

plt.subplots(figsize **=** figsize)

plt.subplot(122)

plt.title('Predict Image')

plt.imshow(img\_pred)

plt.subplot(121)

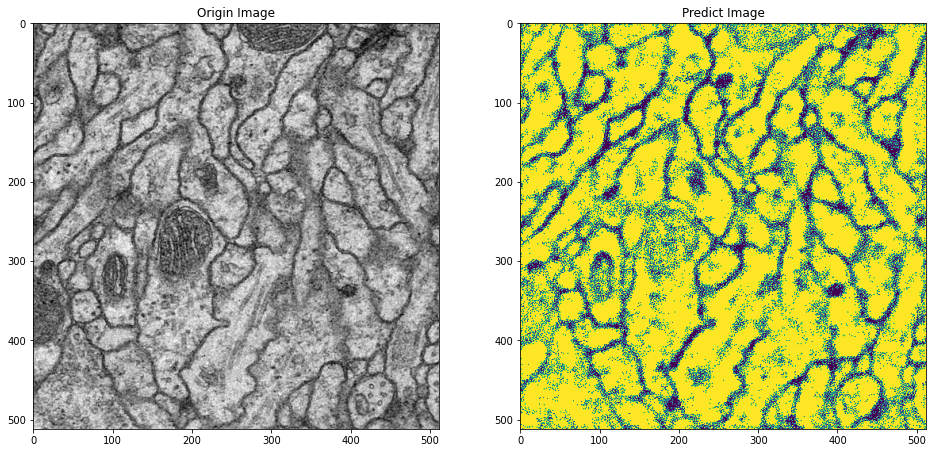
plt.title('Origin Image')

plt.imshow(img)

\_predict\_path(path)

Unet sẽ dự đoán một hình ảnh mới với kích thước tương tự, mỗi pixel sẽ chứa xác suất thuộc một label (như trên label có đối tượng là 1, và label nền là 0)

Như vậy xác suất dưới ngưỡng 0.5 sẽ xem như nền và cho 0 và trên ngưỡng là đối tượng và cho 1.



Tác dụng như trên

#### **3.3. Ứng dụng**

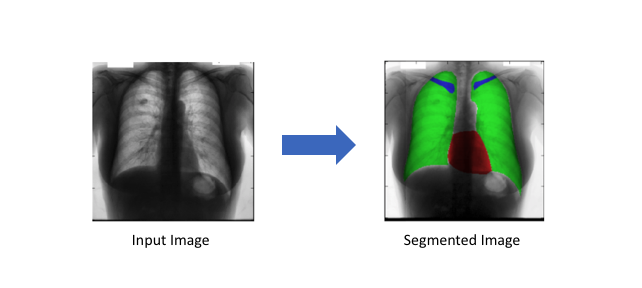
**Ô tô tự lái:**

Segmentation dùng để xác định đường, các xe ô tô, người đi bộ,… để hỗ trợ cho ô tô tự lái



**Chẩn đoán trong y học:**

Segmentation được ứng dụng rất nhiều trong y học để hỗ trợ việc chẩn đoán bệnh. Ví dụ phân tích ảnh X-quang.



**Phát hiện đối tượng và hiểu biết về bối cảnh:**

Phân đoạn cải thiện độ chính xác của các mô hình phát hiện đối tượng. Bằng cách xác định và phân đoạn từng đối tượng riêng lẻ trong một hình ảnh, các kỹ thuật phân đoạn giúp nâng cao độ chính xác của các mô hình phát hiện đối tượng. Bằng cách xác định nội dung ngữ nghĩa của một cảnh, hiểu biết về bối cảnh có thể được cải thiện, và khả năng phân đoạn đối tượng có thể được sử dụng cho việc theo dõi, nhận dạng và hiểu biết về mối quan hệ.



### **4. Bài toán phát hiện**

#### **4.1 Lý thuyết**

<https://machinelearningmastery.com/object-recognition-with-deep-learning/>

**Nhận diện đối tượng** (**Object recognition**) là một thuật ngữ tổng quát để mô tả một tập hợp các nhiệm vụ thị giác máy tính (**Computer Vision**) liên quan đến việc xác định các đối tượng trong các bức ảnh kỹ thuật số.

**Phân loại hình ảnh** (**Image classification**) liên quan đến việc dự đoán lớp của một đối tượng trong một hình ảnh.

**Định vị đối tượng** (**Object localization**) đề cập đến việc xác định vị trí của một hoặc nhiều đối tượng trong một hình ảnh và vẽ một hộp giới hạn xung quanh chúng.

**Phát hiện đối tượng** (**Object detection**) kết hợp cả hai nhiệm vụ này và xác định và phân loại một hoặc nhiều đối tượng trong một hình ảnh.

Khi người sử dụng hoặc người thực hành đề cập đến "**nhận diện đối tượng**", họ thường có nghĩa là "**phát hiện đối tượng**".

Do đó, chúng ta có thể phân biệt giữa ba nhiệm vụ thị giác máy tính này:

**Phân loại hình ảnh**: Dự đoán type hoặc class của một đối tượng trong một hình ảnh.

* Input: Một hình ảnh với một đối tượng, VD: một bức ảnh.
* Output: Một class label.

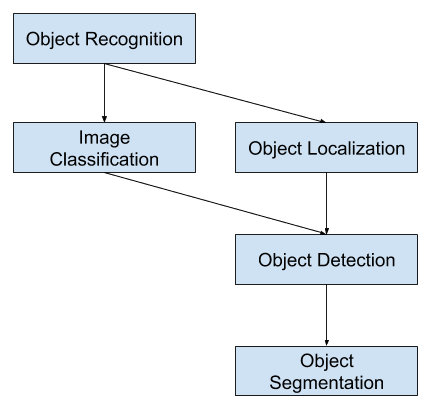
**Định vị đối tượng**: Xác định sự hiện diện của các đối tượng trong một hình ảnh và chỉ định vị trí của chúng với một hộp giới hạn.

* Input: Một hình ảnh với một hoặc nhiều đối tượng, VD: một bức ảnh.
* Output: Một hoặc nhiều bounding box

**Phát hiện đối tượng**: Xác định sự hiện diện của các đối tượng với một bounding box và type hoặc class của các đối tượng được xác định trong một hình ảnh.

* Input: Một hình ảnh với một hoặc nhiều đối tượng, VD: một bức ảnh.
* Output: Một hoặc nhiều boudnging box, và một class label cho mỗi hộp giới hạn.

Từ bảng phân tích này, chúng ta có thể thấy rằng nhận dạng đối tượng đề cập đến một loạt các nhiệm vụ thị giác máy tính đầy thách thức.



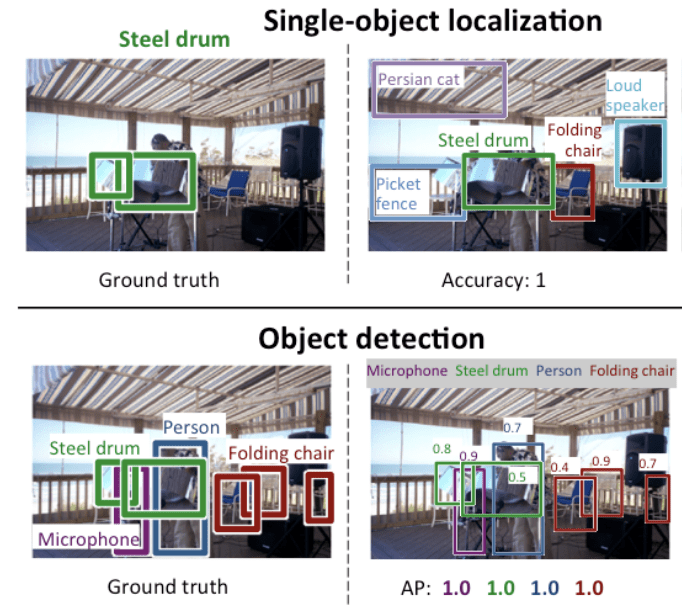
Hầu hết các đổi mới gần đây trong các vấn đề nhận diện hình ảnh đến từ sự tham gia vào các nhiệm vụ ILSVRC.

Đây là một cuộc thi học thuật hàng năm với một thách thức riêng biệt cho mỗi loại vấn đề trong ba loại vấn đề này, với ý định thúc đẩy sự độc lập và cải thiện riêng biệt ở mỗi cấp độ có thể được tận dụng rộng rãi hơn. Ví dụ, hãy xem danh sách ba loại nhiệm vụ tương ứng dưới đây lấy từ bài đánh giá ILSVRC năm 2015:

* **Phân loại hình ảnh**: Các thuật toán tạo ra một danh sách các loại đối tượng xuất hiện trong hình ảnh.
* **Định vị đối tượng đơn (Single-object localization)**: Các thuật toán tạo ra một danh sách các loại đối tượng xuất hiện trong hình ảnh, cùng với một hộp giới hạn hướng trục chỉ ra vị trí và tỷ lệ của một trường hợp của mỗi loại đối tượng.
* **Phát hiện đối tượng**: Các thuật toán tạo ra một danh sách các loại đối tượng xuất hiện trong hình ảnh cùng với một hộp giới hạn hướng trục chỉ ra vị trí và tỷ lệ của mỗi trường hợp của mỗi loại đối tượng.

Chúng ta có thể thấy rằng "**Định vị đối tượng đơn**" là một phiên bản đơn giản hóa của "**Định vị đối tượng**" có định hình lại nhiệm vụ định vị cho các đối tượng của một loại trong một hình ảnh, mà ta có thể giả định là một nhiệm vụ dễ dàng hơn.

Dưới đây là một ví dụ so sánh giữa định vị đối tượng đơn và phát hiện đối tượng, được lấy từ bài báo ILSVRC. Lưu ý sự khác biệt trong kỳ vọng về thực tế trong mỗi trường hợp.



Hiệu suất của một mô hình phân loại hình ảnh được đánh giá bằng cách sử dụng sai số phân loại trung bình trên các nhãn lớp được dự đoán. Hiệu suất của một mô hình định vị đối tượng đơn được đánh giá bằng cách sử dụng khoảng cách giữa hộp giới hạn được dự đoán và kỳ vọng cho lớp mong đợi. Trong khi đó, hiệu suất của một mô hình nhận diện đối tượng được đánh giá bằng cách sử dụng độ chính xác và độ nhớ giữa mỗi hộp giới hạn tốt nhất tương ứng với các đối tượng đã biết trong hình ảnh.

**Các thuật toán phát hiện đối tượng**

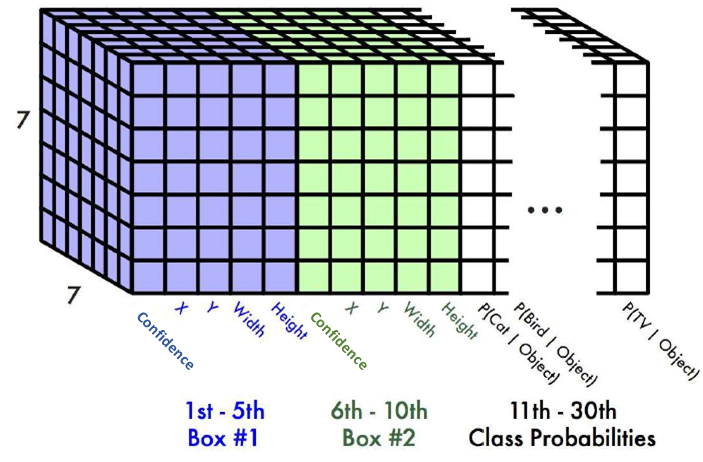
**Họ phương pháp R-CNN** đề cập đến R-CNN, có thể là viết tắt của "Regions with CNN Features" hoặc "Region-Based Convolutional Neural Network," được phát triển bởi Ross Girshick và đồng nghiệp.

Nó bao gồm các kỹ thuật R-CNN, Fast R-CNN và Faster R-CNN được thiết kế và chứng minh cho việc định vị và nhận diện đối tượng.

Một họ mô hình nhận diện đối tượng phổ biến khác được gọi chung là **YOLO** hoặc "**You Only Look Once**," được phát triển bởi Joseph Redmon và đồng nghiệp.

Các mô hình R-CNN có thể nói chung là chính xác hơn, nhưng họ mô hình YOLO lại nhanh chóng, nhanh hơn nhiều so với R-CNN, đạt được việc phát hiện đối tượng trong thời gian thực.

#### **4.2. Code - YOLO v1**



Output của YOLO theo paper gốc thì sẽ là một tensor 7x7x30 chứa các thông tin gồm:

* Số lượng ô grid là 7x7=49 ô
* Mỗi ô grid sẽ chứa 30 tham số
* Trong 30 tham số thì có B=2 (số lượng bounding box tối đa dự đoán trong ô đó là 2) và số lượng class C = 20

Vậy sử dụng đầu ra này thế nào ?

def model\_test(filename):

try:

test\_path=COCO\_PATH+filename

img=cv2.resize(cv2.imread(test\_path),(H,W))

image=tf.io.decode\_jpeg(tf.io.read\_file(test\_path))

image=tf.image.resize(image, [H,W])

output=YOLO.predict(np.expand\_dims(image, axis = 0)) # Vector (1,7,7,30)

print(output.shape)

THRESH=.25

object\_positions=tf.concat(

[tf.where(output[...,0]>=THRESH),tf.where(output[...,5]>=THRESH)],axis=0) # Bounding box có score hơn THRESH mới lấy

print(object\_positions) # Grid chứa đối tượng

selected\_output=tf.gather\_nd(output,object\_positions) # Lấy thông tin trong grid đó

print(selected\_output)

final\_boxes=[]

final\_scores=[]

for i,pos in enumerate(object\_positions):

print('index',i)

print('position',pos)

for j in range(2):

if selected\_output[i][j\*5]>THRESH:

# pos[0] vào ảnh, pos[1] vào hàng, pos[2] vào cột => Sau đó lấy 4 tham số box

output\_box=tf.cast(output[pos[0]][pos[1]][pos[2]][(j\*5)+1:(j\*5)+5],dtype=tf.float32)

# Lấy stt grid + tọa độ x,y => ra được tọa độ grid => sau đó đem tọa độ grid nhân cho 32 => ra được tọa độ thực

x\_centre=(tf.cast(pos[1],dtype=tf.float32)+output\_box[0])\*32

y\_centre=(tf.cast(pos[2],dtype=tf.float32)+output\_box[1])\*32

# Chiều rộng và chiều cao thì nhân cho chiều rộng và chiều cao ảnh thực tức 224 => ra được chiều rộng và chiều cao thực

x\_width,y\_height=tf.math.abs(H\*output\_box[2]),tf.math.abs(W\*output\_box[3])

# Tính ra tọa độ min-max

x\_min,y\_min=int(x\_centre-(x\_width/2)),int(y\_centre-(y\_height/2))

x\_max,y\_max=int(x\_centre+(x\_width/2)),int(y\_centre+(y\_height/2))

if(x\_min<=0):x\_min=0

if(y\_min<=0):y\_min=0

if(x\_max>=W):x\_max=W

if(y\_max>=H):y\_max=H

# Dùng argmax trong class để chọn ra được tên class phù hợp

final\_boxes.append(

[x\_min,y\_min,x\_max,y\_max,

str(classes[tf.argmax(selected\_output[...,10:],axis=-1)[i]])])

final\_scores.append(selected\_output[i][j\*5])

print(final\_scores)

print('finalboxes',final\_boxes)

final\_boxes=np.array(final\_boxes)

object\_classes=final\_boxes[...,4]

nms\_boxes=final\_boxes[...,0:4]

nms\_output=tf.image.non\_max\_suppression(

nms\_boxes,final\_scores,max\_output\_size=100,iou\_threshold=0.2,

score\_threshold=float('-inf')

)

print(nms\_output)

for i in nms\_output:

cv2.rectangle(

img,

(int(final\_boxes[i][0]),int(final\_boxes[i][1])),

(int(final\_boxes[i][2]),int(final\_boxes[i][3])),(0,0,255),1)

cv2.putText(

img,

final\_boxes[i][-1],

(int(final\_boxes[i][0]),int(final\_boxes[i][1])+15),

cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,1,(2,225,155),1

)

cv2.imwrite('/content/outputs/'+filename[:-4]+'\_det'+'.jpg',cv2.resize(img,(384,384)))

except:

print("NO object found !!!")

Sau khi cho image đã được resize thành HxW thì image này sẽ đóng vai trò là input khi cho vào mô hình YOLOv1

=> Đầu ra sẽ là output có dạng là (1,7,7,30)

**Sử dụng**:

* Tại tham số thứ 0 và thứ 5 sẽ xác định xem 2 bounding box dự đoán có lớn hơn ngưỡng THRESH hay không. Nếu có sẽ concat lại để lấy index của vị trí grid đảm bảo thông tin này trong 49 grid. VD: [0 3 3] tức hình thứ 0, grid hàng thứ 3 và cột thứ 3
* Chọn các grid thông qua index đảm bảo điều kiện và cho vào selected\_output.
* Triển khai vòng lặp trong objec\_position ta sẽ lấy thông tin các bounding box của 2 bounding box được dự đoán gồm x,y,w,h
* Vì x và y đang là hệ tọa độ tâm trong 1 grid có phạm vi 0-1. Ta sẽ cộng với index của grid đó để ra được hệ tọa độ grid 0-7. Sau đó nhân kích thước 1 grid là 32 pixel (hình ảnh trong vd là 224/7=32) => Thu được x,y thuộc hệ tọa độ ảnh 0-224.
* w, h chỉ là scale của chiều cao, rộng của bounding box với kích thước ảnh thực tế, nên chỉ cần nhân cho W, H sẽ ra được kích thước trong ảnh có hệ tọa độ 0-224
* Tiếp đó, thông qua x,y,w,h ta sẽ tính toán được x min, y min, x max, y max.
* Dùng argmax để lấy 20 tham số cuối và chọn ra index (đại diện cho class) có điểm số cao nhất. Sau đó map với class để lấy ra label.
* Ta sẽ dùng các thông tin x min, y min, x max, y max.để vẽ bounding box, và thông tin label để xác định đối tượng nằm trong bounding box đó là gì

#### **4.3. Ứng dụng**

<https://blog.roboflow.com/object-detection/>

**Trong giao thông vận tải**

Nhà cung cấp dịch vụ vận tải có thể sử dụng phát hiện đối tượng để đảm bảo an toàn trên mạng lưới của họ, ví dụ, bằng cách xác định các vật cản trên đường sắt hoặc kiểm tra xem có ai đó có mặt trong khu vực hạn chế trên công trường làm việc. Thực sự, phát hiện đối tượng - và thị giác máy tính nói chung - có thể đóng một vai trò quan trọng trong an toàn giao thông vận tải.

**Trong sản xuất thực phẩm**

Nhà sản xuất thực phẩm có thể sử dụng phát hiện đối tượng để đảm bảo tính nguyên vẹn của sản phẩm trước khi đóng gói và gửi đi. Ví dụ, bạn có thể sử dụng thị giác để đảm bảo rằng kem không tan chảy trên băng tải trong quá trình sản xuất.

**Trong hệ thống xe tự lái**

Trong thập kỷ qua, đã có nỗ lực đáng kể để tạo ra các ô tô có thể tự lái một cách an toàn. Đằng sau sự đổi mới này là thị giác máy tính. Xe cần có khả năng phát hiện một số đối tượng nhất định (ví dụ: người đi bộ, đèn giao thông, biển cảnh báo giao thông) để có thể đưa ra quyết định về những gì nên làm và điều hướng nên đi.

### **5. Bài toán sinh ảnh**

[Nguyễn Trường Thành]

#### **5.1 Lý thuyết**

Bài toán sinh ảnh (image generation) là một lĩnh vực trong trí tuệ nhân tạo mà mục tiêu chính là tạo ra hình ảnh mới một cách tự động từ dữ liệu đầu vào. Có nhiều phương pháp và mô hình khác nhau được phát triển để giải quyết bài toán này, và một số trong những mô hình phổ biến nhất được thiết kế dựa trên deep learning.

Dưới đây là một số khái niệm lý thuyết quan trọng liên quan đến bài toán sinh ảnh:

1. **Generative Models (Mô hình tạo sinh):** Đây là các mô hình có khả năng tạo ra dữ liệu mới từ dữ liệu đầu vào. Trong bài toán sinh ảnh, generative models được sử dụng để tạo ra hình ảnh mới.
2. **Variational Autoencoder (VAE):** Là một dạng generative model sử dụng deep learning. VAE tạo ra không gian biểu diễn tiếp tục và có khả năng sinh ra các mẫu mới từ không gian này.
3. Generative Adversarial Network (GAN): Một loại generative model khác, GAN bao gồm hai mô hình là Generator và Discriminator, đấu tranh với nhau trong quá trình huấn luyện. Generator cố gắng tạo ra dữ liệu mới một cách thật giống với dữ liệu thực, trong khi Discriminator cố gắng phân biệt giữa dữ liệu thật và dữ liệu được tạo ra. Quá trình này giúp cả hai mô hình ngày càng hoàn thiện.
4. **Conditional Generative Models:** Trong bài toán sinh ảnh, mô hình có thể được điều chỉnh để tạo ra hình ảnh dựa trên điều kiện nào đó. Ví dụ, có thể tạo ra hình ảnh của một lớp đối tượng cụ thể.
5. **Evaluation Metrics:** Để đánh giá chất lượng của hình ảnh được sinh ra, cần sử dụng các độ đo như Inception Score, FID (Fréchet Inception Distance), hoặc Perceptual Metrics như SSIM (Structural Similarity Index).
6. **Transfer Learning:** Các mô hình đã được huấn luyện trên một lượng lớn dữ liệu có thể được sử dụng để tạo ra hình ảnh mới trong bài toán sinh ảnh thông qua quá trình chuyển giao học.

Bài toán sinh ảnh có nhiều ứng dụng trong thực tế như tạo ảnh nghệ thuật, tái tạo hình ảnh mất thông tin, và phát triển dữ liệu mở rộng trong các lĩnh vực như nhận dạng ảnh và thị giác máy tính.

**Mô hình tạo sinh Generative Adversarial Network (GAN):**

**Generator (Người tạo):** Tạo ra hình ảnh từ không gian tiếp tục thông thường. Nhiệm vụ của nó là "đánh lừa" Discriminator bằng cách tạo ra hình ảnh sao cho Discriminator không thể phân biệt được giữa hình ảnh được tạo ra và hình ảnh thực.

**Discriminator (Người phân biệt):** Điều này đánh giá xem một hình ảnh là thực tế hay được tạo ra bởi Generator. Nó được huấn luyện để phân loại chính xác giữa hình ảnh thực và hình ảnh giả.

**Training Process (Quá trình huấn luyện):** Trong quá trình huấn luyện, Generator và Discriminator được cập nhật xen kẽ. Generator cố gắng tạo ra hình ảnh tốt nhất để "đánh lừa" Discriminator, trong khi Discriminator cố gắng ngày càng phân biệt giữa hình ảnh thực và giả.

**Mô hình tạo sinh Variational Autoencoder (VAE):**

**Encoder (Bộ mã hóa):** Chuyển đổi hình ảnh thành không gian biểu diễn tiếp tục (latent space). Các điểm trong không gian này biểu thị đặc trưng của hình ảnh.

**Decoder (Bộ giải mã):** Chuyển đổi các điểm trong không gian biểu diễn tiếp tục trở lại hình ảnh ban đầu. Mục tiêu là tạo ra một không gian tiếp tục sao cho việc lấy mẫu từ đó tạo ra hình ảnh giống với dữ liệu huấn luyện.

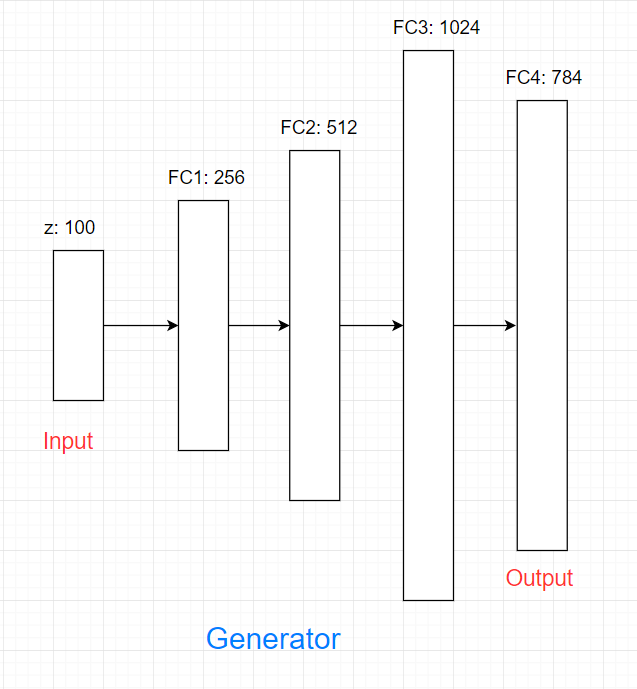
**Loss Function (Hàm mất mát):** VAE sử dụng hai thành phần chính cho hàm mất mát: phần lỗi tái tạo (reconstruction loss) để đảm bảo hình ảnh được tạo ra giống với hình ảnh đầu vào, và phần lỗi độ biến động (variational loss) để đảm bảo rằng không gian biểu diễn tiếp tục được tạo ra là liên tục và có cấu trúc.

**Transfer Learning trong Bài toán Sinh ảnh:**

**Pre-trained Models:** Các mô hình được huấn luyện trước trên tập dữ liệu lớn có thể được sử dụng để tạo sinh hình ảnh mới. Bằng cách chuyển giao học (transfer learning), mô hình có thể được fine-tuned hoặc sử dụng như là một bộ sinh khả năng sinh ảnh chất lượng cao.

#### **5.2 Code**

GAN:



# Mô hình Generator

g = Sequential()

g.add(Dense(256, input\_dim=z\_dim, activation=LeakyReLU(alpha=0.2)))

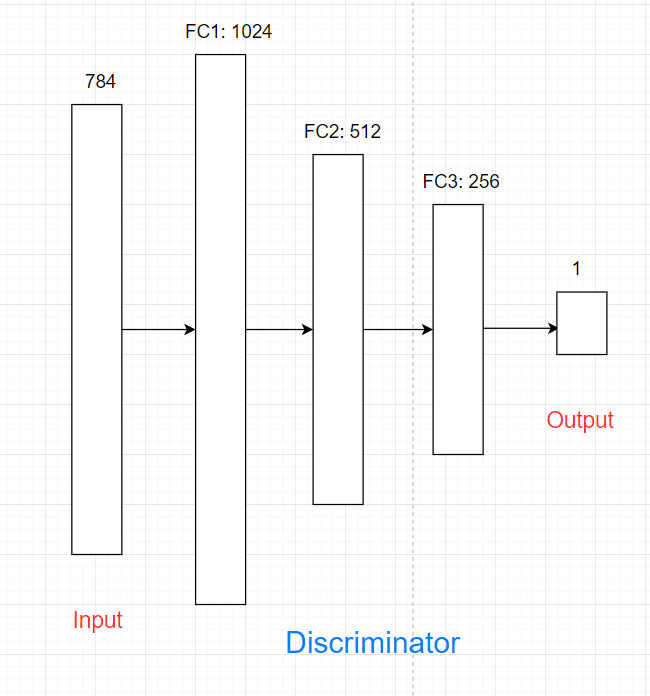
g.add(Dense(512, activation=LeakyReLU(alpha=0.2)))

g.add(Dense(1024, activation=LeakyReLU(alpha=0.2)))

# Vì dữ liệu ảnh MNIST đã chuẩn hóa về [0, 1] nên hàm G khi sinh ảnh ra cũng cần sinh ra ảnh có pixel value trong khoảng [0, 1] => hàm sigmoid được chọn

g.add(Dense(784, activation='sigmoid'))

g.compile(loss='binary\_crossentropy', optimizer=adam, metrics=['accuracy'])



# Mô hình Discriminator

d = Sequential()

d.add(Dense(1024, input\_dim=784, activation=LeakyReLU(alpha=0.2)))

d.add(Dropout(0.3))

d.add(Dense(512, activation=LeakyReLU(alpha=0.2)))

d.add(Dropout(0.3))

d.add(Dense(256, activation=LeakyReLU(alpha=0.2)))

d.add(Dropout(0.3))

# Hàm sigmoid cho bài toán binary classification

d.add(Dense(1, activation='sigmoid'))

X\_eval, y\_eval = generate\_fake\_samples(g\_model=gan\_model.layers[0], latent\_dim=100, n\_samples=25)

\_plot(X\_eval[:, :, :, 0])

Để ra quyết định tức đưa ra ảnh mới thì sau khi training ta sẽ dùng mô hình sinh ảnh fake để sử dụng

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

*# Hàm sinh batch input là noise véc tơ cho generator*

**def** **generate\_latent\_points**(latent\_dim, n\_samples):

*# khởi tạo input ngẫu nhiên*

x\_input **=** np.random.randn(latent\_dim **\*** n\_samples)

*# reshape về batch*

x\_input **=** x\_input.reshape(n\_samples, latent\_dim)

**return** x\_input

*# Hàm sinh batch cho fake sample, nhãn của fake sample là 0*

**def** **generate\_fake\_samples**(g\_model, latent\_dim, n\_samples):

*# sinh batch là các noise véc tơ*

x\_input **=** generate\_latent\_points(latent\_dim, n\_samples)

*# dự báo outputs từ g\_model*

X **=** g\_model.predict(x\_input)

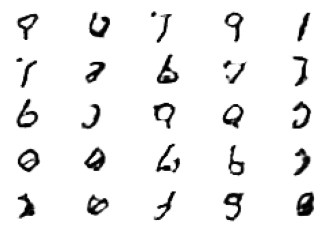
*# khởi tạo y = 0 với nhãn fake*

y **=** np.zeros((n\_samples, 1))

**return** X, y

Hàm generate\_fake\_samples ban đầu sẽ dùng hàm generate\_latent\_points để tạo mẫu với giá trị ngẫu nhiên và cho vào x\_input

x\_input này sẽ được cho vào g\_model tức mô hình GAN đã được train để cho ra X



Các giá trị X này chính là ảnh ngẫu nhiên được sinh ra

#### **5.3 Ứng dụng**

**Tạo Ảnh Nghệ Thuật:**

Sử dụng mô hình sinh ảnh để tạo ra tranh nghệ thuật, hình ảnh số với các yếu tố sáng tạo và nghệ thuật.

**Tăng cường Dữ liệu Huấn luyện:**

Sinh ảnh mới để mở rộng dữ liệu huấn luyện cho các mô hình máy học và deep learning, giúp cải thiện hiệu suất của mô hình trên các tập dữ liệu nhỏ.

**Tạo Ảnh Chân Dung Từ Mô tả:**

Tự động sinh ảnh chân dung dựa trên mô tả văn bản, có thể được sử dụng trong các ứng dụng như phát hiện tội phạm hoặc xây dựng hồ sơ người sử dụng.

**Tạo Ảnh Số Hóa Thế Giới Thực:**

Chuyển đổi ảnh chụp từ thế giới thực thành ảnh số hóa với các yếu tố nghệ thuật hoặc biến đổi.

**Khôi phục ảnh cũ:**

Sử dụng sinh ảnh để khôi phục và cải thiện chất lượng của ảnh cũ, hư hại hoặc mờ.

**Tạo Ảnh Phong Cảnh:**

Tạo sinh ảnh phong cảnh mới dựa trên dữ liệu đầu vào hoặc yêu cầu người dùng.

**Đồng Bộ Hóa Không Gian và Thời Gian:**

Sinh ảnh để đồng bộ hóa không gian và thời gian trong video, giúp tái tạo hoặc tạo ra cảnh quay mới.

**Hiệu Ứng và Lọc Ảnh:**

Tạo sinh ảnh để áp dụng hiệu ứng và bộ lọc ảnh tự động.

**Tạo Ảnh Dựa Trên Nội Dung:**

Sinh ảnh dựa trên nội dung của hình ảnh khác, có thể được sử dụng trong việc tạo ra hình ảnh mới từ các yếu tố đã biết.

**Tạo Ảnh Đa Dạng Cho Ứng dụng Thị giác Máy tính:**

Sinh ảnh để tạo ra các biến thể và dạng mới của đối tượng để cải thiện đa dạng trong dữ liệu huấn luyện của các mô hình nhận dạng.