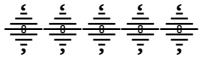
TRƯỜNG ĐẠI HỌC PHENIKAA

KHOA CÔNG NGHỆ THÔNG TIN



PHENIKAA
UNIVERSITY

BÁO CÁO BÀI TẬP LỚN

CHỦ ĐỀ

Học phần : Thị giác máy tính -1-1-22(N03)
Giảng viên : Nguyễn Văn Tới
Sinh viên : Hòa Thị Thu Trang - Mã SV : 20010799
Nguyễn Chí Công - Mã SV : 20010846
Nguyễn Tuấn Hiệp - Mã SV : 2001859
Nguyễn Thị Nga - Mã SV : 20010809

HÀ NỘI, THÁNG 11 NĂM 2022

MỤC LỤC

Thêm Tiêu đề (Định dạng > Kiểu đoạn). Khi đó, chúng sẽ xuất hiện trong mục lục.

DANH SÁCH HÌNH ẢNH

Bảng phân công công việc

Họ và tên	**Công việc**
Nguyễn Chí Công	-
Nguyễn Thị Nga	
Nguyễn Tuấn Hiệp	
Hòa Thị Thu Trang	-

MỞ ĐẦU

NỘI DUNG

A. Ứng dụng mạng nơ ron tích chập (Convolution Neural Network) trong bài toán Image Classification

I. Tổng quan

1. Giới thiệu

Convolutional Neural Networks (CNN) là một trong những mô hình deep learning phổ biến nhất và có ảnh hưởng nhiều nhất trong cộng đồng Computer Vision. CNN được dùng trong trong nhiều bài toán như nhận dạng ảnh, phân tích video, ảnh MRI, hoặc cho bài các bài của lĩnh vực xử lý ngôn ngữ tự nhiên,và hầu hết đều giải quyết tốt các bài toán này.

2. Image Classification

Phân loại ảnh là một bài toán quan trọng bậc nhất trong lĩnh vực Computer Vision. Đã có rất nhiều nghiên cứu để giải quyết bài toán này bằng cách rút trích các đặc trưng rất phổ biến như SIFT, HOG rồi cho máy tính học nhưng những cách này tỏ ra không thực sự hiệu quả. Nhưng ngược lại, đối với con người, chúng ta lại có bản năng tuyệt vời để phân loại được những đối tượng trong khung cảnh xung quanh một cách dễ làm.

Dữ liệu đầu vào của bài toán là một bức ảnh. Một ảnh được biểu ảnh bằng ma trận các giá trị. Mô hình phân lớp sẽ phải dự đoán được lớp của ảnh từ ma trận điểm ảnh này, ví dụ như ảnh đó là con mèo, chó, hay là chim.

157	48	240	49	2	78	229	64	207	32
159	54	94	218	126	97	60	163	60	69
128	201	202	100	53	5	4	131	49	209
199	132	202	121	119	238	49	220	76	149
192	63	21	129	16	226	104	32	255	15
124	225	229	180	141	155	153	100	20	252
90	17	238	232	34	209	64	187	197	210
212	244	86	30	192	160	85	195	36	111
226	221	201	223	161	170	114	154	9	68
252	217	1	107	127	126	50	26	151	193

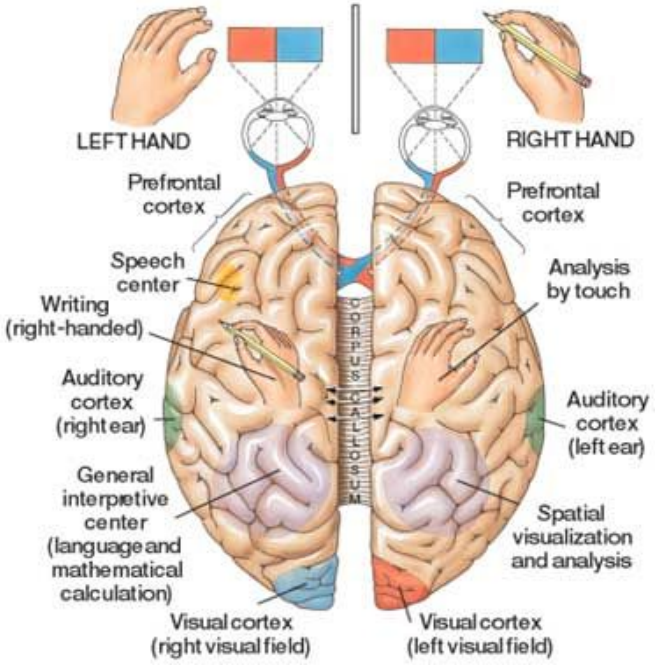
Chúng ta nhìn thấy

Máy tính thấy

Để biểu diễn một bức ảnh 256x256 pixel trong máy tính thì ta cần ma trận sẽ có kích thước 256x256 chiều, và tùy thuộc vào bức ảnh là có màu hay ảnh xám thì ma trận này sẽ có số kênh tương ứng, ví dụ với ảnh màu 256x256 RGB, chúng ta sẽ có ma trận 256x256x3 để biểu diễn ảnh này.

3. Mối liên kết giữa CNN và thị giác

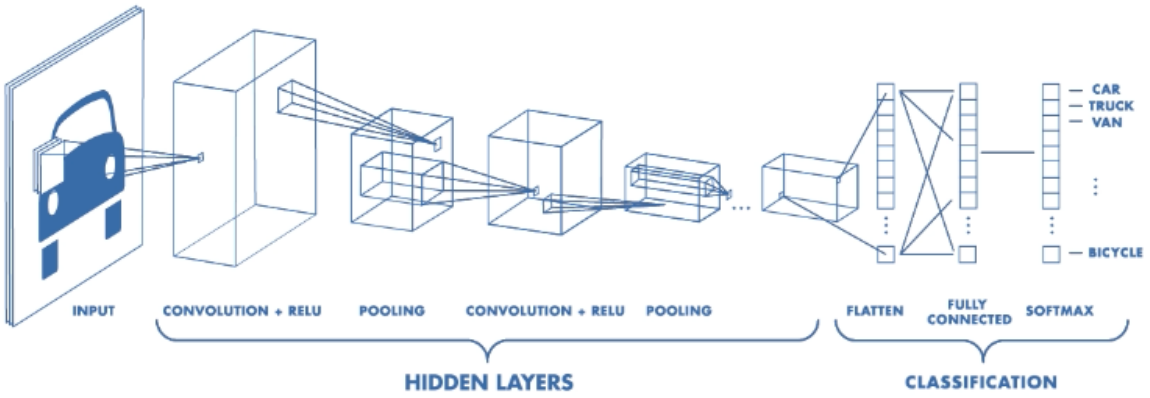
CNN có mối liên kết chặt chẽ với sinh học, cụ thể là của vỏ não thị giác, nơi xử lý thông tin liên quan đến hình ảnh từ các tế bào cảm thụ ánh sáng nằm ở mắt người. Mỗi neuron được thiết lập để phản ứng lại một số đặc điểm cố định của neuron đó.



4. Cấu trúc CNN

Mạng Nơron tích chập (Convolutional Neural Network - CNN) là một mô hình học sâu có thể xây dựng được các hệ thống phân loại với độ chính xác cao. Ý tưởng của CNN được lấy cảm hứng từ khả năng nhận biết thị giác của bộ não con người – một khả năng nhận biết và xử lý hình ảnh mạnh mẽ và tự nhiên.

Cấu trúc cơ bản của mô hình CNN gồm hai bộ chính: Bộ trích xuất đặc trưng (Feature Learning) và bộ phân lớp (Classification). Bộ trích xuất đặc trưng là tập hợp các lớp tích chập (Convolution Layer), lớp tổng hợp (Pooling Layer) và lớp phi tuyến ReLU chồng lên nhau. Bộ phân lớp của mô hình CNN gồm lớp kết nối đầy đủ (Fully Connected Layer) và kết quả Output là xác suất phân lớp của mô hình.



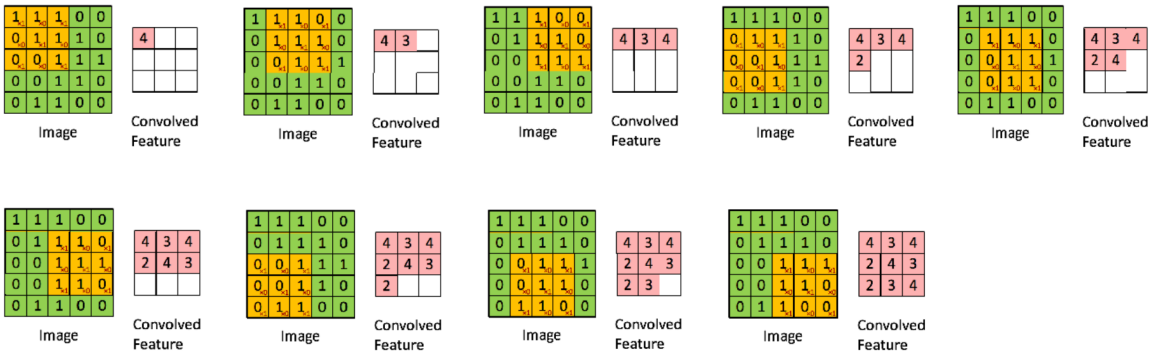
Hình 4.1: Kiến trúc mô hình mạng nơron tích chập

(https://master-engineer.com/2020/11/05/convolution-networks/)

Các lớp trong CNN được liên kết với nhau thông qua cơ chế tích chập. Dữ liệu đầu vào của lớp tiếp theo là kết quả tích chập từ lớp trước đó, mỗi lớp được áp đặt các bộ lọc khác nhau. Lớp Pooling dùng để lọc các thông tin hữu ích hơn bằng cách loại bỏ các thông tin nhiễu. Trong suốt quá trình huấn luyện, CNN sẽ tự động học các tham số cho các lớp. Lớp cuối cùng là lớp kết nối đầy đủ (Fully Connected Layer) liên kết các đặc trưng đã được trích xuất từ những lớp trước và thực hiện phân lớp.

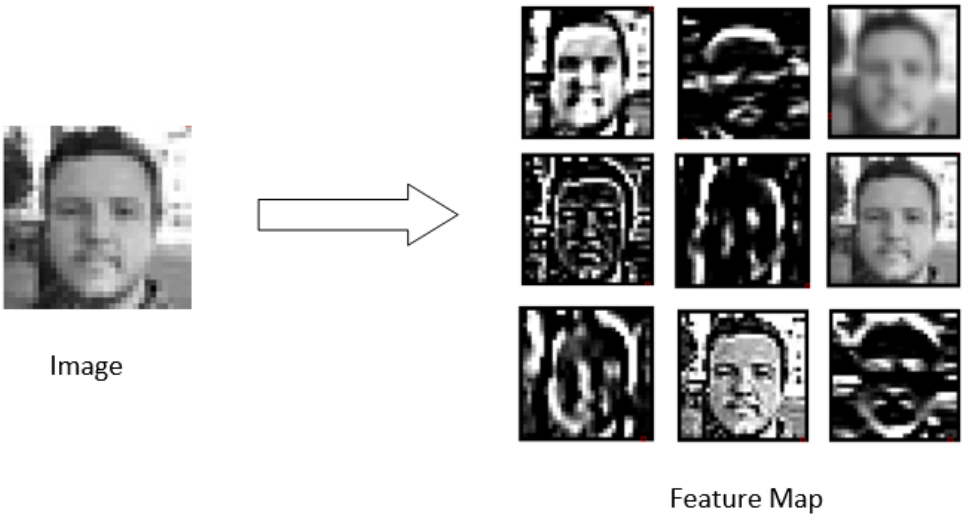
Convolution Layer (lớp tích chập)

Lớp Convolution là lớp đầu tiên trong mạng CNN dùng để trích xuất các đặc trưng từ một hình ảnh. Lớp Convolution duy trì mối quan hệ giữa các pixel bằng cách sử dụng một bộ các bộ lọc (filter - là một ma trận và thường có kích thước nhỏ hơn ma trận đầu vào) áp lên một vùng của ma trận đầu vào và tiến hành tích chập (nhân filter với vùng áp lên có cùng kích thước). Các filter sẽ dịch chuyển chạy dọc theo ma trận đầu vào từ trái sang phải sau đó quay trở lại bên trái hình ảnh, quá trình này lặp lại đến khi filter đã quét qua toàn bộ hình ảnh.



Hình 4.2: Quá trình filter 3x3 quét ảnh

Sự kết hợp của một hình ảnh với các filter khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ, làm sắc nét,... Từ đó giúp mô hình trích xuất các đặc trưng của ảnh. Kết quả của quá trình tích chập hình ảnh với các filter khác nhau là một tập các đặc trưng của hình ảnh đó (Feature Map).

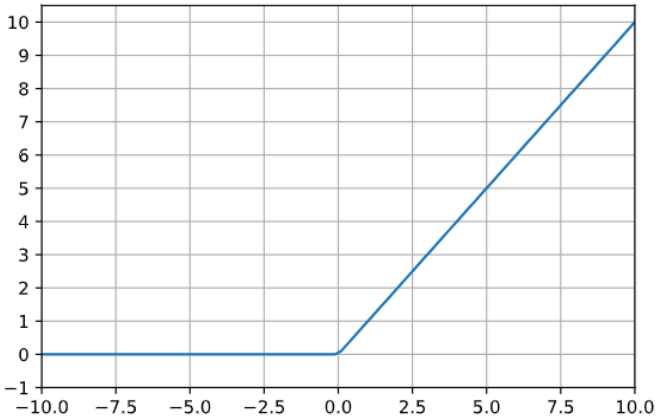


Hình 4.3: Feature Map của một bức ảnh xám được tạo từ 9 filter

ReLU Layer (lớp phi tuyến ReLU)

Output của Convolution Layer sẽ phải qua hàm kích hoạt (activation function) trước khi trở thành input của Convolution Layer tiếp theo. Hàm kích hoạt thường được sử dụng cho mô hình CNN là hàm ReLU vì một số ưu điểm nổi bật: tốc độ hội tụ nhanh, chi phí tính toán thấp và hiệu năng cao hơn so với hàm Tanh hoặc Sigmoid.

Công thức toán học của hàm ReLU: f (x) = max(0,x)



Hình 4.4: Hàm ReLU

Pooling Layer (lớp tổng hợp)

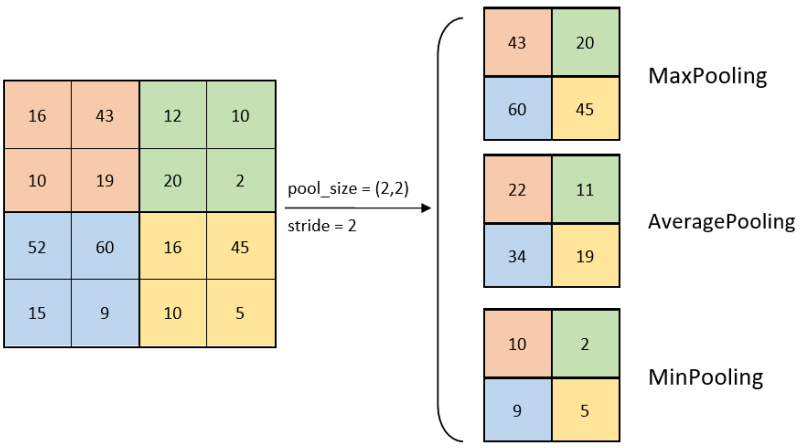
Lớp Pooling thường được sử dụng ngay sau lớp convolution layer để đơn giản hóa thông tin và giảm bớt số lượng neuron. Các phương thức phổ biến trong lớp Pooling:

+ MaxPooling: trả về giá trị lớn nhất trong các phần tử được bao bởi bộ lọc.

Nhận xét: Bảo toàn các đặc trưng được phát hiện; được sử dụng thường xuyên

+ AveragePooling: trả về giá trị trung bình của các phần tử được bao bởi bộ lọc.

Nhận xét: Giảm kích thước feature map; được sử dụng trong mạng LeNet

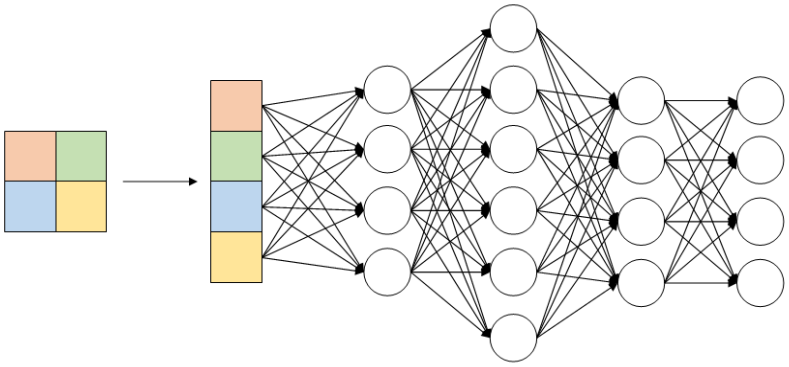


Hình 4.5: Các phương pháp phổ biến trong lớp Pooling

Công dụng của lớp Pooling là giảm kích thước dữ liệu. Các tầng trong CNN chồng lên nhau có lớp Pooling ở cuối mỗi tầng giúp giảm kích thước dữ liệu nhưng vẫn giữ được các đặc trưng để lấy mẫu, ngoài ra MaxPooling cũng hoạt động như một công cụ khử nhiễu, thực hiện việc loại bỏ nhiễu song song với giảm kích thước. Từ đó cũng giúp giảm số lượng tham số của mạng, làm tăng tính hiệu quả và kiểm soát hiện tượng Overfitting.

Fully Connected Layer (lớp kết nối đầy đủ)

Fully Connected Layer nhận đầu vào là các dữ liệu đã được làm phẳng (trong mô hình phân lớp CNN là các giá trị được tính toán từ lớp trước), mỗi đầu vào đó sẽ được kết nối đến tất cả noron. Hình ảnh dưới đây là ví dụ là lớp Fully Connected Layer trong mạng CNN.



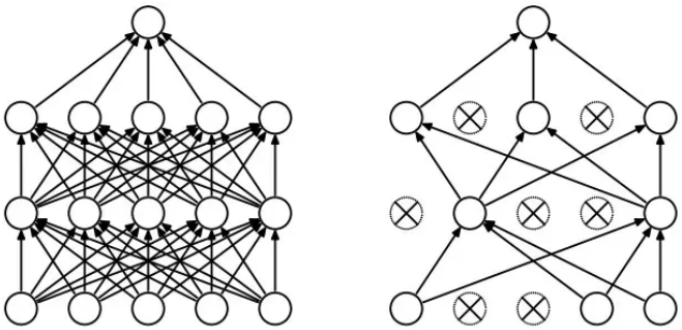
Hình 4.6: Lớp Fully Connected trong mạng CNN

Kết quả phân lớp của mô hình CNN là tham số của lớp noron cuối cùng trong Fully Connected Layer kết hợp với hàm kích hoạt là sigmoid function (nếu là phân lớp nhị phân) hoặc softmax function, trả về xác suất phân lớp của dữ liệu.

Dropout

Nếu Fully Connected Layer có quá nhiều noron, các nơron lại quá liên kết và phụ thuộc lẫn nhau trong quá trình huấn luyện sẽ làm hạn chế sức mạnh của mỗi nơron, dẫn đến hiện tượng Overfitting - hiện tượng mô hình huấn luyện đưa ra kết quả phân lớp tốt đối với tập dữ liệu training nhưng với dữ liệu testing lại cho kết quả tồi (biểu hiện: Độ chính xác khi phân lớp của tập train cao hơn gấp nhiều lần tập test,...)

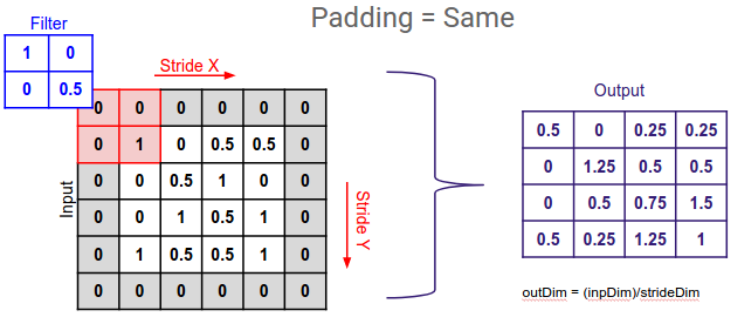
Một trong những kỹ thuật giúp tránh Overfitting là Dropout - vô hiệu hóa hoạt động của một số noron ngẫu nhiên. Kỹ thuật này giúp các noron bỏ qua một số tính năng không cần thiết để học thêm một số tính năng mạnh mẽ khác. Mặc dù Dropout khiến mô hình phải tăng lượng epoch cần thiết để hội tụ nhưng nó cũng làm giảm thời gian thực hiện của mỗi epoch.



Hình 4.7: Mô hình mạng nơron áp dụng kỹ thuật Dropout

Padding (Đường viền)

Lớp tích chập trong mạng CNN có nhiệm vụ trích xuất đặc trưng từ dữ liệu, tuy nhiên việc sử dụng các filter để học các đặc trưng cũng khiến cho mô hình có xu hướng mất các pixel ở viền hình ảnh (kích thước filter tăng thì lượng pixel bị mất cũng tăng), điều này có thể khiến mô hình bỏ qua một số đặc trưng quan trọng. Một giải pháp đơn giản cho vấn đề này là thêm các pixel phụ xung quanh viền của dữ liệu đầu vào, các pixel phụ thường có giá trị = 0. Hình ảnh dưới đây mô tả hoạt động của lớp Convolution mà không làm giảm số chiều dữ liệu:



Hình 4.8: Hoạt động của lớp Convolution với padding = 1

(https://analyticsindiamag.com/guide-to-different-padding-methods-for-cnn-models/)

Việc thêm vào các pixel phụ làm tăng sự đóng góp của các pixel ở đường viền của hình ảnh gốc bằng cách đưa chúng vào giữa hình ảnh được. Do đó, thông tin trên các đường viền được giữ nguyên cũng như thông tin ở giữa hình ảnh.

Cách chọn tham số cho CNN

Số các convolution layer: càng nhiều các convolution layer thì hiệu năng càng được cải thiện. Sau khoảng 3 hoặc 4 layer, các tác động được giảm một cách đáng kể

Filter size: thường filter theo size 5×5 hoặc 3×3

Pooling size: thường là 2×2 hoặc 4×4 cho ảnh đầu vào lớn

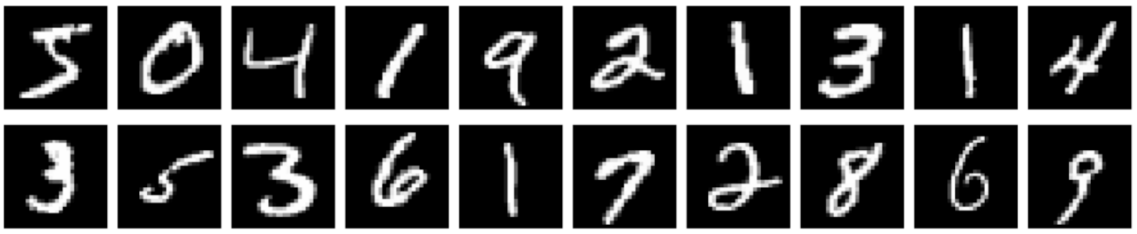
Cách cuối cùng thực hiện nhiều lần train test để chọn ra được param tốt nhất.

II. Xây dựng mô hình

1. Mô tả cơ sở dữ liệu

Nhận dạng là lĩnh vực được các nhà khoa học rất quan tâm để giải quyết các yêu cầu trong cuộc sống hiện nay, có nhiều lĩnh vực như nhận dạng tín hiệu, nhận dạng tiếng nói hay nhận dạng ảnh. Vấn đề nhận dạng chữ viết tay nói chung và nhận dạng chữ số viết tay nói riêng là một thách thức lớn đối với các nhà nghiên cứu.

Chữ số viết tay xuất hiện ở hầu hết các công việc của các cơ quan, nhà máy, xí nghiệp, trường học. Vấn đề đặt ra là mỗi người đều có một kiểu chữ viết tay khác nhau: về cỡ chữ, kiểu chữ, … nên việc nhận diện chữ viết sao cho máy tính có thể hiểu độ chính xác cao là khá phức tạp và yêu cầu tốn thời gian. Vì vậy trong bài này, nhóm em trình bày về phương pháp nhận dạng chữ số viết tay sử dụng tập dữ liệu MNIST - gồm 70.000 hình ảnh của các chữ số viết tay (từ 0 đến 9), mỗi ảnh là một mảng có kích thước 28x28 và dữ liệu được chia thành 2 phần: training set: 60.000 ảnh và test set: 10.000 ảnh.



2. Các bước thực hiện

Import các thư viện và data

```
import numpy as np
import tensorflow
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout,Flatten, Input, Reshape
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

```
[2] from tensorflow.keras.datasets.mnist import load_data
    (X_train, y_train), (X_test, y_test) = load_data()
```

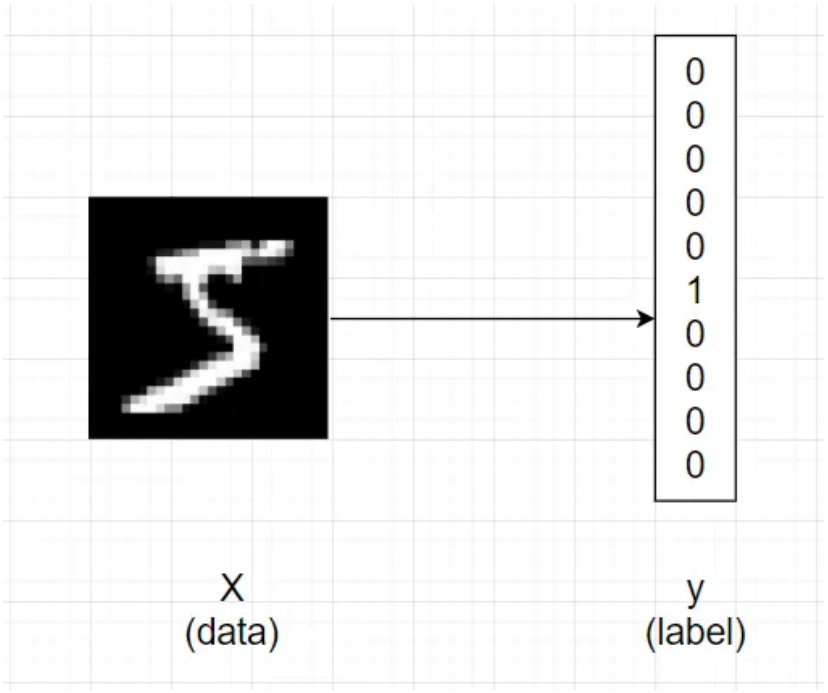
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

Chuẩn hóa dữ liệu

```
#Input : 28x28x1 chiều, giá trị [0,1]
#Output là một vector 10 chiều , mỗi số biểu diễn khả năng ảnh là các số từ 0-9
X_train_scaled = np.array(X_train) / 255
X_test_scaled = np.array(X_test) / 255
```

Sử dụng OneHotEncoder để chuyển đổi label của ảnh từ giá trị số sang vector cùng kích thước với output của model

Ví dụ:




```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
encoder.fit(y_train.reshape(-1, 1))
y_train = encoder.transform(y_train.reshape(-1,1)).toarray()
y_test = encoder.transform(y_test.reshape(-1,1)).toarray()
print (y_train[2])
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

Xây dựng model

Mô hình chung bài toán CNN: Input image -> Convolutional layer + Pooling layer -> Fully connected layer -> output

```
inp = Input(shape = (28,28,1))
cnn = Conv2D(filters = 16, kernel_size= 5, activation = 'relu')(inp)
pooling = MaxPooling2D(pool_size = (2,2))(cnn)
drop = Dropout (0.25) (pooling)

cnn = Conv2D(filters = 16, kernel_size=5, activation='relu')(drop)
pooling = MaxPooling2D(pool_size = (2,2))(cnn)
drop = Dropout (0.25) (pooling)

f = Flatten() (drop)
fc1 = Dense (units = 32, activation = 'relu')(f)
fc2 = Dense (units = 32, activation = 'relu')(fc1)
out = Dense (units = 10, activation = 'softmax')(fc2)

model = Model(inputs = inp, outputs = out)
model.summary()
```

Train model

$$L = -\sum_{i=1}^{10} y_i * log(\hat{y}_i)$$

Sử dụng hàm loss là "categorical_crossentropy" để phân loại đa lớp có hai hoặc nhiều nhãn đầu ra. Nhãn đầu ra được gán giá trị mã hóa ở dạng 0 và 1. Khi hàm Loss càng nhỏ thì model dự đoán càng gần với giá trị thật.

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

history = model.fit(X_train_scaled, y_train, batch_size=128, epochs = 50, validation_data=(X_test_scaled, y_test))
```

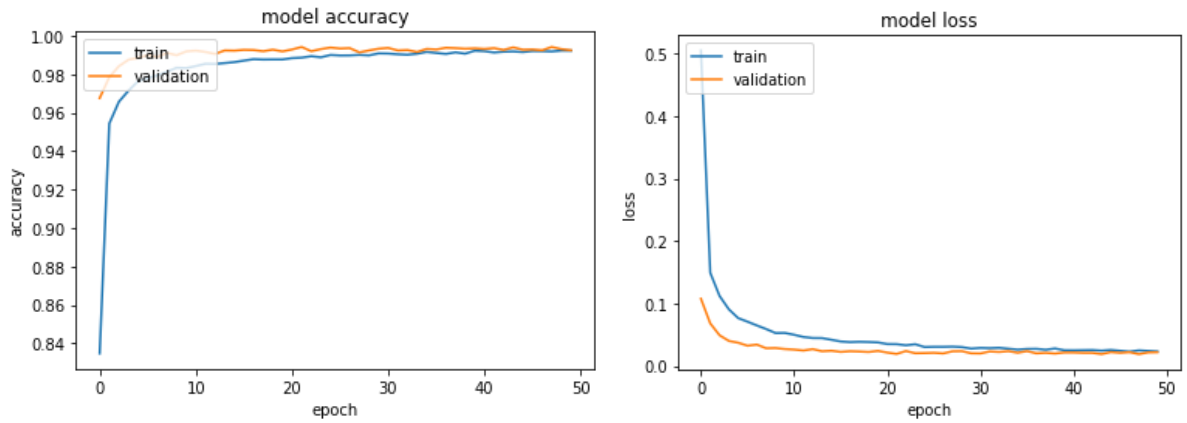
```
Epoch 1/50
469/469 [==============================] - 13s 6ms/step - loss: 0.5049 - accuracy: 0.8347 - val_loss: 0.1077 - val_accuracy: 0.9676
Epoch 2/50
469/469 [==============================] - 2s 4ms/step - loss: 0.1492 - accuracy: 0.9544 - val_loss: 0.0682 - val_accuracy: 0.9785
Epoch 3/50
469/469 [==============================] - 2s 4ms/step - loss: 0.1121 - accuracy: 0.9657 - val_loss: 0.0492 - val_accuracy: 0.9842
Epoch 4/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0907 - accuracy: 0.9715 - val_loss: 0.0401 - val_accuracy: 0.9876
Epoch 5/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0767 - accuracy: 0.9764 - val_loss: 0.0373 - val_accuracy: 0.9885
Epoch 6/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0709 - accuracy: 0.9781 - val_loss: 0.0325 - val_accuracy: 0.9900
Epoch 7/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0649 - accuracy: 0.9794 - val_loss: 0.0343 - val_accuracy: 0.9890
Epoch 8/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0592 - accuracy: 0.9813 - val_loss: 0.0284 - val_accuracy: 0.9911
Epoch 9/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0528 - accuracy: 0.9834 - val_loss: 0.0288 - val_accuracy: 0.9899
Epoch 10/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0530 - accuracy: 0.9832 - val_loss: 0.0272 - val_accuracy: 0.9919
Epoch 11/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0500 - accuracy: 0.9843 - val_loss: 0.0263 - val_accuracy: 0.9923
Epoch 12/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0465 - accuracy: 0.9855 - val_loss: 0.0248 - val_accuracy: 0.9916
Epoch 13/50
469/469 [==============================] - 3s 7ms/step - loss: 0.0448 - accuracy: 0.9854 - val_loss: 0.0271 - val_accuracy: 0.9906
Epoch 14/50
469/469 [==============================] - 4s 8ms/step - loss: 0.0447 - accuracy: 0.9858 - val_loss: 0.0236 - val_accuracy: 0.9924
Epoch 15/50
469/469 [==============================] - 4s 8ms/step - loss: 0.0420 - accuracy: 0.9863 - val_loss: 0.0245 - val_accuracy: 0.9923
Epoch 16/50
469/469 [==============================] - 4s 8ms/step - loss: 0.0393 - accuracy: 0.9871 - val_loss: 0.0226 - val_accuracy: 0.9927
```

```
Epoch 17/50
469/469 [==============================] - 3s 7ms/step - loss: 0.0383 - accuracy: 0.9879 - val_loss: 0.0236 - val_accuracy: 0.9926
Epoch 18/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0388 - accuracy: 0.9877 - val_loss: 0.0232 - val_accuracy: 0.9920
Epoch 19/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0384 - accuracy: 0.9877 - val_loss: 0.0225 - val_accuracy: 0.9928
Epoch 20/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0378 - accuracy: 0.9877 - val_loss: 0.0242 - val_accuracy: 0.9919
Epoch 21/50
469/469 [==============================] - 3s 7ms/step - loss: 0.0351 - accuracy: 0.9884 - val_loss: 0.0215 - val_accuracy: 0.9929
Epoch 22/50
469/469 [==============================] - 3s 5ms/step - loss: 0.0350 - accuracy: 0.9887 - val_loss: 0.0194 - val_accuracy: 0.9942
Epoch 23/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0332 - accuracy: 0.9894 - val_loss: 0.0240 - val_accuracy: 0.9920
Epoch 24/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0349 - accuracy: 0.9888 - val_loss: 0.0205 - val_accuracy: 0.9931
Epoch 25/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0304 - accuracy: 0.9901 - val_loss: 0.0207 - val_accuracy: 0.9939
Epoch 26/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0307 - accuracy: 0.9898 - val_loss: 0.0211 - val_accuracy: 0.9934
Epoch 27/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0307 - accuracy: 0.9898 - val_loss: 0.0201 - val_accuracy: 0.9936
Epoch 28/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0310 - accuracy: 0.9901 - val_loss: 0.0235 - val_accuracy: 0.9914
Epoch 29/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0303 - accuracy: 0.9898 - val_loss: 0.0238 - val_accuracy: 0.9924
Epoch 30/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0279 - accuracy: 0.9909 - val_loss: 0.0203 - val_accuracy: 0.9933
Epoch 31/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0290 - accuracy: 0.9908 - val_loss: 0.0201 - val_accuracy: 0.9937
Epoch 32/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0286 - accuracy: 0.9904 - val_loss: 0.0233 - val_accuracy: 0.9924
```

```
Epoch 33/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0291 - accuracy: 0.9902 - val_loss: 0.0225 - val_accuracy: 0.9926
Epoch 34/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0277 - accuracy: 0.9907 - val_loss: 0.0238 - val_accuracy: 0.9916
Epoch 35/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0263 - accuracy: 0.9916 - val_loss: 0.0212 - val_accuracy: 0.9932
Epoch 36/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0273 - accuracy: 0.9911 - val_loss: 0.0238 - val_accuracy: 0.9929
Epoch 37/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0275 - accuracy: 0.9906 - val_loss: 0.0203 - val_accuracy: 0.9938
Epoch 38/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0261 - accuracy: 0.9914 - val_loss: 0.0211 - val_accuracy: 0.9936
Epoch 39/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0281 - accuracy: 0.9907 - val_loss: 0.0200 - val_accuracy: 0.9933
Epoch 40/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0252 - accuracy: 0.9923 - val_loss: 0.0216 - val_accuracy: 0.9935
Epoch 41/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0251 - accuracy: 0.9920 - val_loss: 0.0216 - val_accuracy: 0.9932
Epoch 42/50
469/469 [==============================] - 3s 6ms/step - loss: 0.0253 - accuracy: 0.9913 - val_loss: 0.0211 - val_accuracy: 0.9936
Epoch 43/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0255 - accuracy: 0.9918 - val_loss: 0.0210 - val_accuracy: 0.9927
Epoch 44/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0246 - accuracy: 0.9919 - val_loss: 0.0191 - val_accuracy: 0.9940
Epoch 45/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0257 - accuracy: 0.9916 - val_loss: 0.0223 - val_accuracy: 0.9928
Epoch 46/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0244 - accuracy: 0.9921 - val_loss: 0.0211 - val_accuracy: 0.9929
Epoch 47/50
469/469 [==============================] - 2s 4ms/step - loss: 0.0230 - accuracy: 0.9920 - val_loss: 0.0222 - val_accuracy: 0.9925
Epoch 48/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0250 - accuracy: 0.9919 - val_loss: 0.0190 - val_accuracy: 0.9942
Epoch 49/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0241 - accuracy: 0.9923 - val_loss: 0.0218 - val_accuracy: 0.9931
Epoch 50/50
469/469 [==============================] - 2s 5ms/step - loss: 0.0233 - accuracy: 0.9923 - val_loss: 0.0221 - val_accuracy: 0.9925
```

Vẽ đồ thị loss, accuracy của training set và validation set

```python
his = history
plt.plot(his.history['accuracy'])
plt.plot(his.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# "Loss"
plt.plot(his.history['loss'])
plt.plot(his.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

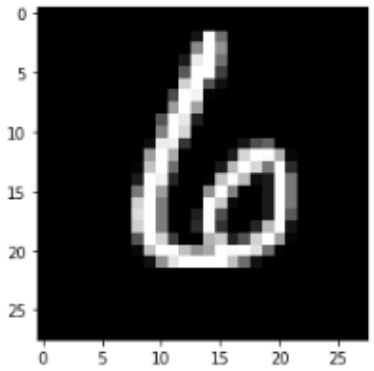


Đánh giá mô hình qua dữ liệu test

```
i = np.random.randint(10000)
plt.imshow(X_test_scaled[i], cmap='gray')

y_predict = model.predict(X_test[i].reshape(1,28,28,1))
print('Giá trị dự đoán: ', np.argmax(y_predict))
```

```
1/1 [==============================] - 0s 17ms/step
Giá trị dự đoán:  6
```



```
score = model.evaluate(X_test_scaled, y_test, verbose=0)
print(score)
```

```
[0.022099260240793228, 0.9933000206947327]
```

Dùng kết quả đánh giá của model với test set để làm kết quả cuối cùng của model. Theo kết quả trên thì model có độ chính xác là 99,33% với tập dữ liệu MNIST.

Lưu mô hình

```
model.save('mnist.h5')
```

3. Đánh giá mô hình, giải thích cách tính parameters

Model có độ chính xác cho tập train là 99,1% và tập test là 99,33% và không xảy ra overfitting.

Cách tính Parameters:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 24, 24, 16)        416

 max_pooling2d (MaxPooling2D  (None, 12, 12, 16)       0
 )

 dropout (Dropout)           (None, 12, 12, 16)        0

 conv2d_1 (Conv2D)           (None, 8, 8, 16)          6416

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 16)         0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 16)          0

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 32)                8224

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 16,442
Trainable params: 16,442
Non-trainable params: 0
```

conv2d (Conv2D): (5*5+1)*16 = 416

conv2d_1 (Conv2D): (5*5*16+1)*16 = 6416

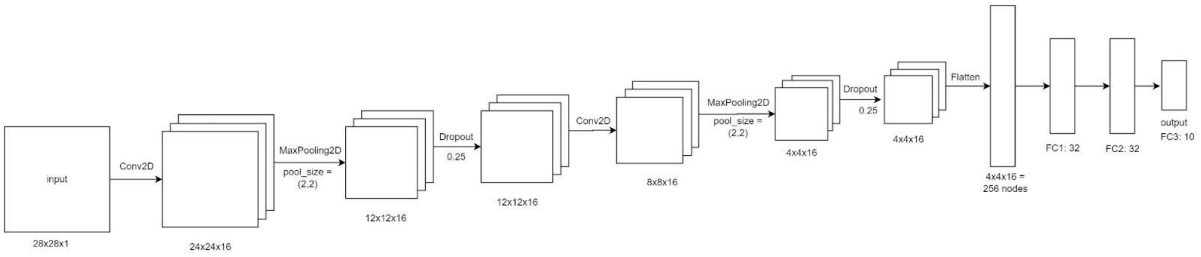
dense (Dense): (256+1) * 32 = 8224

dense_1 (Dense): (32+1) * 32 = 1056

dense_2 (Dense): (31+1) * 10 = 330

Total params: conv2d (Conv2D) + conv2d_1 (Conv2D) + dense (Dense) + dense_1 (Dense) + dense_2 (Dense) = 16422

4. Kiến trúc mạng


input
28x28x1
Conv2D
24x24x16
MaxPooling2D
pool_size =
(2,2)
12x12x16
Dropout
0.25
12x12x16
Conv2D
8x8x16
MaxPooling2D
pool_size =
(2,2)
4x4x16
Dropout
0.25
4x4x16
Flatten
4x4x16 =
256 nodes
FC1: 32
FC2: 32
output
FC3: 10

B. Object detection với mô hình YOLO

I. Phương pháp

1. YOLO v1
2. YOLO v2
3. YOLO v3
4. YOLO v4

5. YOLO v5
6. YOLO v6

C. YOLO v7

2. Phương pháp

3. Cơ sở dữ liệu

4. Thực nghiệm

5. Kết quả