# P1:

**Giải thích đoạn code:**

* **train\_dir = os.path.join('/content/drive/MyDrive/hhtqd/dataset/train'):**
  + Dòng này xác định đường dẫn đến thư mục chứa tập dữ liệu huấn luyện.
  + os.path.join() là một hàm trong Python giúp kết hợp các phần của đường dẫn một cách chính xác, tránh lỗi khi các hệ điều hành khác nhau có cách biểu diễn đường dẫn khác nhau.
  + '/content/drive/MyDrive/hhtqd/dataset/train' là đường dẫn cụ thể đến thư mục chứa các dữ liệu dùng để huấn luyện mô hình.
* **labels = ['001000', '002000', '005000', '010000', '020000','050000','100000','200000','500000']:**
  + Dòng này tạo một danh sách các nhãn (labels) hoặc các lớp (classes) mà dữ liệu có thể thuộc về.
  + Các nhãn này thường là các chuỗi ký tự đại diện cho các loại khác nhau của dữ liệu. Trong trường hợp này, các nhãn có thể đại diện cho các mức độ khác nhau của một đại lượng nào đó (ví dụ: cường độ, kích thước).
* **validation\_dir = os.path.join('/content/drive/MyDrive/hhtqd/dataset/validation'):**
  + Tương tự như dòng đầu tiên, dòng này xác định đường dẫn đến thư mục chứa tập dữ liệu kiểm định.
  + Tập dữ liệu kiểm định được sử dụng để đánh giá hiệu suất của mô hình sau khi đã được huấn luyện trên tập dữ liệu huấn luyện.

**Tại sao cần cả tập huấn luyện và tập kiểm định:**

Việc chia tập dữ liệu thành tập huấn luyện và tập kiểm định là một quy trình tiêu chuẩn trong học máy. Nó giúp đảm bảo rằng mô hình học được các đặc trưng chung của dữ liệu, chứ không chỉ đơn thuần học thuộc lòng tập dữ liệu huấn luyện.

* **Tập huấn luyện:**
  + Được sử dụng để huấn luyện mô hình.
  + Mô hình sẽ học cách tìm ra mối quan hệ giữa các đặc trưng đầu vào và nhãn tương ứng.
  + Mô hình sẽ cố gắng tìm ra các quy tắc chung để dự đoán nhãn cho các dữ liệu mới chưa từng thấy.
* **Tập kiểm định:**
  + Được sử dụng để đánh giá hiệu suất của mô hình sau khi huấn luyện.
  + Mô hình sẽ được đưa vào các dữ liệu trong tập kiểm định mà nó chưa từng thấy trước đó.
  + Sau đó, ta so sánh kết quả dự đoán của mô hình với nhãn thực tế để đánh giá độ chính xác của mô hình.

**Lợi ích của việc chia tập dữ liệu:**

* **Ngăn ngừa hiện tượng quá khớp (overfitting):** Nếu chỉ sử dụng một tập dữ liệu để cả huấn luyện và kiểm định, mô hình có thể học thuộc lòng quá kỹ tập dữ liệu đó, dẫn đến hiệu suất kém khi áp dụng vào dữ liệu mới.
* **Đánh giá khách quan hiệu suất của mô hình:** Tập kiểm định cung cấp một cách đánh giá khách quan và độc lập về khả năng tổng quát hóa của mô hình.
* **Điều chỉnh siêu tham số:** Tập kiểm định giúp ta điều chỉnh các siêu tham số của mô hình để đạt được hiệu suất tốt nhất.

**labels = ['001000', '002000', ...]**

**Dòng code này đang tạo ra một danh sách các nhãn (labels) hoặc các lớp (classes) được sử dụng để phân loại dữ liệu.**

Trong ngữ cảnh của bài toán học máy, mỗi nhãn đại diện cho một loại hoặc một nhóm dữ liệu cụ thể. Trong trường hợp này, các nhãn '001000', '002000', ... được hiểu là các mệnh giá tiền Việt Nam.

**Ý nghĩa cụ thể:**

* **Danh sách các nhãn:** Danh sách labels này chứa tất cả các mệnh giá tiền mà mô hình sẽ cần phải phân loại.
* **Mục đích:** Các nhãn này sẽ được sử dụng để gắn cho từng dữ liệu đầu vào. Ví dụ, nếu một hình ảnh chứa tờ tiền 500.000 đồng, thì nhãn tương ứng của hình ảnh đó sẽ là '500000'.
* **Trong quá trình huấn luyện:** Mô hình sẽ học cách phân biệt các hình ảnh chứa các mệnh giá tiền khác nhau dựa trên các đặc trưng hình ảnh.
* **Trong quá trình dự đoán:** Khi đưa một hình ảnh mới vào mô hình, mô hình sẽ dự đoán nhãn (mệnh giá tiền) tương ứng cho hình ảnh đó.

# P2:

#checking the size of data available to us for training out model

for label in labels:

  directory1 = os.path.join(train\_dir,label)

  print("Images of label \"" + label + "\":\t", len(os.listdir(directory1)))

**Mục đích:** Đoạn code này nhằm kiểm tra số lượng ảnh có sẵn cho từng nhãn trong tập dữ liệu huấn luyện.

**Phân tích chi tiết:**

1. **for label in labels::**
   * Vòng lặp này lặp qua từng nhãn trong danh sách labels. Mỗi nhãn đại diện cho một loại dữ liệu cụ thể.
2. **directory1 = os.path.join(train\_dir, label):**
   * Dòng này tạo ra đường dẫn đến thư mục chứa các ảnh thuộc nhãn hiện tại.
   * train\_dir là đường dẫn đến thư mục gốc của tập dữ liệu huấn luyện.
   * label là nhãn đang được xét trong vòng lặp.
3. **print("Images of label \"" + label + "\":\t", len(os.listdir(directory1))):**
   * Dòng này in ra thông tin về số lượng ảnh của nhãn hiện tại.
   * os.listdir(directory1) trả về một danh sách các tên tệp trong thư mục directory1.
   * len(os.listdir(directory1)) tính độ dài của danh sách này, tức là số lượng ảnh trong thư mục.

**Ý nghĩa tổng thể:**

Đoạn code này giúp chúng ta hiểu rõ số lượng ảnh có sẵn cho mỗi nhãn trong tập dữ liệu huấn luyện. Điều này rất quan trọng vì nó cho phép chúng ta đánh giá sự cân bằng của dữ liệu và có thể điều chỉnh các kỹ thuật xử lý dữ liệu hoặc chọn thuật toán phù hợp. Nếu một nhãn có số lượng ảnh quá ít so với các nhãn khác, có thể dẫn đến hiện tượng overfitting hoặc underfitting.

# P3:

|  |
| --- |
| #plotting images of different review for understanding the dataset  plt.figure(figsize=(30,14))  for i in range(9):    directory = os.path.join(train\_dir, labels[i])    for j in range(10):      path = os.path.join(directory, os.listdir(directory)[j])      img = mpimg.imread(path)      plt.subplot(9, 10, i\*10 + j + 1)      plt.imshow(img)      if j == 0:        plt.ylabel(labels[i], fontsize=20)  plt.setp(plt.gcf().get\_axes(),xticks=[],yticks=[]);  plt.tight\_layout()  plt.show() |

**Đoạn code này thực hiện việc hiển thị các hình ảnh minh họa cho từng loại nhãn (label) trong tập dữ liệu huấn luyện.**

**Phân tích từng phần:**

1. **Nhập các thư viện cần thiết:**
   * **matplotlib.pyplot:** Thư viện này được sử dụng để vẽ đồ thị và hiển thị hình ảnh.
   * **mpimg:** Thư viện này được dùng để đọc các file hình ảnh.
2. **Tạo một hình vẽ với kích thước cụ thể:**
   * plt.figure(figsize=(30,14)): Tạo một hình vẽ có chiều rộng 30 inch và chiều cao 14 inch. Kích thước này có thể điều chỉnh tùy theo số lượng hình ảnh và kích thước màn hình.
3. **Vòng lặp qua các nhãn:**
   * for i in range(9):: Vòng lặp này sẽ lặp qua 9 nhãn đầu tiên trong danh sách labels.
   * **Tìm đường dẫn đến thư mục chứa hình ảnh của nhãn hiện tại:** directory = os.path.join(train\_dir, labels[i])
   * **Lấy danh sách các tên tệp hình ảnh trong thư mục:** filenames = os.listdir(directory)
4. **Vòng lặp qua các hình ảnh trong mỗi nhãn:**
   * for j in range(10):: Vòng lặp này sẽ lặp qua 10 hình ảnh đầu tiên trong thư mục của nhãn hiện tại.
   * **Tạo đường dẫn đầy đủ đến tệp hình ảnh:** path = os.path.join(directory, filenames[j])
   * **Đọc hình ảnh:** img = mpimg.imread(path)
   * **Tạo một subplot:** plt.subplot(9, 10, i\*10 + j + 1): Tạo một subplot trong một lưới 9x10. Vị trí của subplot được tính toán dựa trên giá trị của i và j.
   * **Hiển thị hình ảnh:** plt.imshow(img): Hiển thị hình ảnh vào subplot vừa tạo.
   * **Thêm nhãn cho cột đầu tiên (tùy chọn):** if j == 0: plt.ylabel(labels[i], fontsize=20): Nếu đây là hình ảnh đầu tiên trong cột, sẽ thêm nhãn của lớp đó vào trục y.
5. **Loại bỏ các nhãn trên trục:** plt.setp(plt.gcf().get\_axes(), xticks=[], yticks=[]): Loại bỏ các nhãn trên trục x và trục y để hình ảnh được hiển thị rõ ràng hơn.

**xticks=[] và yticks=[]:**

* xticks và yticks là các thuộc tính của axes, đại diện cho các nhãn trên trục x và trục y.
* Bằng cách gán giá trị [] (một danh sách rỗng) cho các thuộc tính này, chúng ta đang nói rằng chúng ta không muốn hiển thị bất kỳ nhãn nào trên các trục đó.

1. **Điều chỉnh bố cục:** plt.tight\_layout(): Điều chỉnh bố cục của các subplot để tránh chồng chéo.
2. **Hiển thị hình ảnh:** plt.show(): Hiển thị hình vẽ cuối cùng.

**Ý nghĩa tổng thể:**

Đoạn code này giúp chúng ta trực quan hóa dữ liệu bằng cách hiển thị các ví dụ hình ảnh cho từng loại nhãn. Điều này giúp chúng ta hiểu rõ hơn về đặc điểm của từng loại dữ liệu và đánh giá chất lượng của tập dữ liệu.

**Các điểm cần lưu ý:**

* **Kích thước hình vẽ:** Có thể điều chỉnh kích thước hình vẽ để phù hợp với số lượng hình ảnh và kích thước màn hình.
* **Số lượng hình ảnh hiển thị:** Có thể thay đổi số lượng hình ảnh hiển thị cho mỗi nhãn bằng cách điều chỉnh giá trị của range trong các vòng lặp.
* **Các nhãn trên trục:** Có thể tùy chỉnh việc hiển thị hoặc ẩn các nhãn trên trục x và trục y.

**Ứng dụng:**

Đoạn code này thường được sử dụng trong các bài toán xử lý ảnh, ví dụ như phân loại hình ảnh, phát hiện đối tượng. Nó giúp chúng ta kiểm tra xem dữ liệu có đủ đại diện cho các lớp khác nhau hay không, và có bất kỳ vấn đề gì về chất lượng hình ảnh hay không.

# P4:

|  |
| --- |
| from tensorflow.keras.preprocessing.image import ImageDataGenerator  train\_data\_gen = ImageDataGenerator(rescale=1/255,)  validation\_data\_gen = ImageDataGenerator(rescale=1/255) |

**Mục đích:** Đoạn code này tạo ra hai đối tượng ImageDataGenerator để chuẩn bị dữ liệu hình ảnh cho quá trình huấn luyện và kiểm định mô hình học sâu.

**Phân tích chi tiết:**

1. **from tensorflow.keras.preprocessing.image import ImageDataGenerator:**
   * Import thư viện ImageDataGenerator từ thư viện tensorflow.keras.preprocessing.image. Đây là một lớp tiện ích giúp chuẩn bị dữ liệu hình ảnh cho mô hình học sâu, bao gồm các thao tác tiền xử lý như tăng cường dữ liệu, chuẩn hóa, và chia dữ liệu thành batches.
2. **train\_data\_gen = ImageDataGenerator(rescale=1/255):**
   * Tạo một đối tượng ImageDataGenerator cho tập dữ liệu huấn luyện.
   * **rescale=1/255:** Đặt tham số rescale thành 1/255. Điều này sẽ chuẩn hóa giá trị pixel của các hình ảnh từ khoảng 0-255 (mặc định cho ảnh 8-bit) về khoảng 0-1. Đây là một bước chuẩn hóa phổ biến trong xử lý hình ảnh cho các mô hình học sâu.
3. **validation\_data\_gen = ImageDataGenerator(rescale=1/255):**
   * Tạo một đối tượng ImageDataGenerator cho tập dữ liệu kiểm định.
   * **rescale=1/255:** Đặt tham số rescale tương tự như đối tượng train\_data\_gen. Điều này đảm bảo rằng cả tập huấn luyện và kiểm định đều được chuẩn hóa theo cùng một cách.

**Tại sao lại chọn 1/255?**

* **Giá trị pixel:** Trong hình ảnh 8-bit, mỗi pixel có thể có giá trị từ 0 đến 255, tương ứng với mức độ sáng tối khác nhau của màu sắc.
* **Chuẩn hóa về khoảng 0-1:** Chia tất cả các giá trị pixel cho 255 sẽ đưa chúng về khoảng từ 0 đến 1. Khoảng này là một khoảng phổ biến được sử dụng trong nhiều thuật toán học máy và học sâu.

**Tổng kết:**

Đoạn code này tạo ra hai đối tượng ImageDataGenerator để chuẩn bị dữ liệu hình ảnh cho quá trình huấn luyện và kiểm định mô hình. Việc chuẩn hóa giá trị pixel bằng rescale=1/255 là một bước quan trọng trong xử lý hình ảnh để đảm bảo tính nhất quán và hiệu suất của mô hình.

# P5:

|  |
| --- |
| train\_generator = train\_data\_gen.flow\_from\_directory(train\_dir,                                                      target\_size=(144, 144),                                                      batch\_size=32,                                                      class\_mode='binary')  validation\_generator = validation\_data\_gen.flow\_from\_directory(validation\_dir,                                                      target\_size=(144, 144),                                                      batch\_size=32,                                                      class\_mode='binary') |

**Mục đích:** Đoạn code này tạo ra hai generator (bộ sinh dữ liệu) để cung cấp dữ liệu hình ảnh cho quá trình huấn luyện và kiểm định mô hình học sâu.

**Phân tích chi tiết:** train\_generator và validation\_generator

* **train\_data\_gen** và **validation\_data\_gen**: Đây là hai đối tượng ImageDataGenerator được tạo trước đó, chúng chứa các thông số cấu hình cho việc tăng cường dữ liệu và chuẩn hóa hình ảnh.
* **.flow\_from\_directory():** Phương thức này được sử dụng để tạo một generator từ một thư mục chứa các hình ảnh.

### Các tham số:

* **train\_dir** và **validation\_dir**: Đường dẫn đến thư mục chứa các hình ảnh huấn luyện và kiểm định tương ứng.
* **target\_size=(144, 144)**: Kích thước mong muốn của các hình ảnh. Tất cả các hình ảnh sẽ được resize thành kích thước này trước khi được đưa vào mô hình.
* **batch\_size=32**: Kích thước của một batch dữ liệu. Một batch là một nhóm các hình ảnh được xử lý cùng lúc trong quá trình huấn luyện.
* **class\_mode='binary'**: Chỉ định loại đầu ra của generator. Trong trường hợp này, vì là bài toán phân loại nhị phân (binary classification), nên đầu ra của generator sẽ là một mảng nhị phân (0 hoặc 1) tương ứng với lớp dự đoán của mỗi hình ảnh.

**Ý nghĩa của số 144:**

* **Kích thước chiều rộng và chiều cao của hình ảnh:** Con số 144 ở đây có nghĩa là tất cả các hình ảnh trong tập dữ liệu của bạn sẽ được resize (thay đổi kích thước) thành hình vuông có chiều rộng và chiều cao đều là 144 pixel.
* **Chuẩn hóa kích thước:** Việc đưa ra một kích thước cố định như vậy giúp đảm bảo rằng tất cả các hình ảnh đầu vào sẽ có cùng một kích thước, điều này rất quan trọng cho quá trình huấn luyện mô hình. Nếu các hình ảnh có kích thước khác nhau, mô hình sẽ gặp khó khăn trong việc học các đặc trưng chung.

**Tại sao lại chọn kích thước 144x144?**

Kích thước 144x144 được chọn dựa trên một số yếu tố:

* **Đặc điểm của dữ liệu:** Kích thước này có thể phù hợp với kích thước trung bình của các hình ảnh trong tập dữ liệu của bạn.
* **Khả năng tính toán:** Kích thước hình ảnh càng lớn, yêu cầu về bộ nhớ và khả năng tính toán càng cao. Việc chọn kích thước 144x144 là một sự cân nhắc giữa độ chính xác của mô hình và khả năng tính toán của hệ thống.
* **Kiến trúc của mô hình:** Một số kiến trúc mô hình học sâu yêu cầu kích thước đầu vào cụ thể.

## Giải thích tham số batch\_size=32

**Tham số batch\_size=32 trong ImageDataGenerator nghĩa là gì?**

Khi huấn luyện một mô hình học sâu, dữ liệu không được đưa vào mô hình một lần mà được chia thành các nhóm nhỏ gọi là **batch**. Mỗi batch chứa một số lượng nhất định các mẫu dữ liệu. Trong trường hợp này, batch\_size=32 có nghĩa là mỗi lần cập nhật trọng số của mô hình, ta sẽ sử dụng một batch gồm 32 hình ảnh để tính toán gradient và cập nhật các tham số.

**Tại sao lại chọn batch\_size=32?**

Việc lựa chọn kích thước batch là một quyết định quan trọng, ảnh hưởng đến hiệu suất và độ ổn định của quá trình huấn luyện. Số 32 thường được chọn vì một số lý do sau:

* **Ổn định:** Một batch size quá nhỏ có thể dẫn đến việc cập nhật trọng số quá đột ngột, làm cho quá trình huấn luyện trở nên không ổn định. Ngược lại, một batch size quá lớn lại làm giảm tốc độ hội tụ của mô hình. Batch size bằng 32 thường là một giá trị cân bằng giữa độ ổn định và tốc độ hội tụ.
* **Bộ nhớ:** Kích thước batch ảnh hưởng trực tiếp đến lượng bộ nhớ cần thiết để lưu trữ dữ liệu trong quá trình huấn luyện. Nếu batch size quá lớn, máy tính có thể bị quá tải.
* **Tính toán:** Việc tính toán gradient và cập nhật trọng số cho một batch lớn sẽ tốn nhiều thời gian hơn so với một batch nhỏ. Tuy nhiên, việc sử dụng GPU có thể giúp tăng tốc quá trình này.
* **Khái quát hóa:** Một batch size nhỏ hơn có thể giúp mô hình học được các đặc trưng cụ thể hơn của dữ liệu, trong khi một batch size lớn hơn có thể giúp mô hình học được các đặc trưng chung hơn.

**Các yếu tố ảnh hưởng đến việc chọn batch\_size:**

* **Kích thước của tập dữ liệu:** Nếu tập dữ liệu nhỏ, batch size cũng nên nhỏ.
* **Khả năng tính toán của phần cứng:** Máy tính có bộ nhớ lớn và GPU mạnh mẽ có thể xử lý các batch size lớn hơn.
* **Kiến trúc của mô hình:** Một số kiến trúc mô hình có thể hoạt động tốt hơn với các batch size cụ thể.
* **Bài toán:** Các bài toán khác nhau có thể yêu cầu các batch size khác nhau.

# P6:

|  |
| --- |
| from keras.models import Sequential  from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten  model = tf.keras.models.Sequential()  #lớp CNN1  model.add(Conv2D(32,(3,3), activation='relu', input\_shape=(144,144,3)))  model.add(MaxPooling2D(2,2))  #lớp CNN2  model.add(Conv2D(64,(3,3), activation='relu'))  model.add(MaxPooling2D(2,2))  #lớp CNN3  model.add(Conv2D(128,(3,3), activation='relu'))  model.add(MaxPooling2D(2,2))  #chuyển dữ liệu đầu ra của mạng CNN từ mảng 2 chiều về mảng 1 chiều  model.add(Flatten())  #lớp ẩn  model.add(Dense(512,activation=tf.nn.relu))  #lớp output  model.add(Dense(9,activation=tf.nn.softmax))  model.summary() |

**Mục đích:** Đoạn code này xây dựng một mô hình mạng nơ-ron tích chập (Convolutional Neural Network - CNN) để phân loại hình ảnh.

**Phân tích chi tiết:**

1. **Import thư viện:**
   * from keras.models import Sequential: Import lớp Sequential từ thư viện Keras để tạo một mô hình tuần tự.
   * from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten: Import các lớp Dense, Conv2D, MaxPooling2D và Flatten từ thư viện Keras để xây dựng các lớp của mô hình.
2. **Tạo mô hình:**
   * model = tf.keras.models.Sequential(): Tạo một đối tượng Sequential để lưu trữ các lớp của mô hình.
3. **Xây dựng các lớp CNN:**
   * **Lớp CNN1:**
     + model.add(Conv2D(32,(3,3), activation='relu', input\_shape=(144,144,3))): Thêm một lớp tích chập 2D với 32 bộ lọc, kích thước bộ lọc là 3x3, hàm kích hoạt là ReLU và kích thước đầu vào là 144x144x3 (chiều cao, chiều rộng, số kênh màu).
     + model.add(MaxPooling2D(2,2)): Thêm một lớp pooling tối đa với kích thước cửa sổ 2x2.
   * **Lớp CNN2:**
     + model.add(Conv2D(64,(3,3), activation='relu')): Thêm một lớp tích chập 2D với 64 bộ lọc, kích thước bộ lọc là 3x3 và hàm kích hoạt là ReLU.
     + model.add(MaxPooling2D(2,2)): Thêm một lớp pooling tối đa với kích thước cửa sổ 2x2.
   * **Lớp CNN3:**
     + model.add(Conv2D(128,(3,3), activation='relu')): Thêm một lớp tích chập 2D với 128 bộ lọc, kích thước bộ lọc là 3x3 và hàm kích hoạt là ReLU.
     + model.add(MaxPooling2D(2,2)): Thêm một lớp pooling tối đa với kích thước cửa sổ 2x2.
4. **Chuyển dữ liệu về mảng 1 chiều:**
   * model.add(Flatten()): Chuyển đổi đầu ra của các lớp CNN từ mảng 2 chiều thành mảng 1 chiều.
5. **Thêm lớp ẩn và lớp đầu ra:**
   * model.add(Dense(512,activation=tf.nn.relu)): Thêm một lớp ẩn với 512 nơ-ron và hàm kích hoạt ReLU.
   * model.add(Dense(9,activation=tf.nn.softmax)): Thêm lớp đầu ra với 9 nơ-ron (tương ứng với 9 lớp phân loại) và hàm kích hoạt softmax để đưa ra xác suất cho từng lớp.
6. **In thông tin mô hình:**
   * model.summary(): In thông tin về cấu trúc của mô hình, bao gồm số lượng tham số, kích thước đầu vào và đầu ra của mỗi lớp.

## Tổng kết:

Mô hình này được thiết kế để phân loại hình ảnh thành 9 lớp. Nó sử dụng 3 lớp tích chập để trích xuất các đặc trưng từ hình ảnh, sau đó chuyển dữ liệu thành mảng 1 chiều và sử dụng hai lớp ẩn để thực hiện phân loại. Cuối cùng, lớp đầu ra đưa ra xác suất cho từng lớp phân loại.

## Tại sao sử dụng 3 lớp CNN trong mô hình?

Việc sử dụng 3 lớp CNN trong mô hình trên là một thiết kế khá phổ biến trong các mạng CNN. Mỗi lớp CNN đóng vai trò khác nhau trong việc trích xuất các đặc trưng từ hình ảnh. Dưới đây là một số lý do:

### 1. **Trích xuất các đặc trưng ở các cấp độ khác nhau:**

* **Lớp đầu tiên:** Thường tập trung vào việc trích xuất các đặc trưng cơ bản như các cạnh, góc, texture đơn giản.
* **Lớp thứ hai:** Trích xuất các đặc trưng phức tạp hơn như các hình dạng đơn giản, các bộ phận của vật thể.
* **Lớp thứ ba:** Trích xuất các đặc trưng trừu tượng hơn, liên quan đến toàn bộ đối tượng và có thể giúp phân biệt các lớp khác nhau.

### 2. **Tăng khả năng học các đặc trưng phức tạp:**

* **Sâu hơn:** Mạng càng sâu, khả năng học các đặc trưng phức tạp càng cao. Mỗi lớp CNN bổ sung thêm một cấp độ trừu tượng, giúp mô hình hiểu sâu hơn về dữ liệu.
* **Tăng biểu diễn:** Với nhiều lớp, mô hình có thể học được nhiều biểu diễn khác nhau của dữ liệu, giúp cải thiện độ chính xác.

### 3. **Giảm hiện tượng overfitting:**

* **Pooling:** Các lớp pooling giúp giảm kích thước của feature map, loại bỏ một số nhiễu và giảm nguy cơ overfitting.
* **Nhiều lớp:** Việc sử dụng nhiều lớp giúp phân tán độ phức tạp của mô hình, giảm khả năng mô hình học thuộc lòng dữ liệu huấn luyện.

### 4. **Tùy biến:**

* **Số lượng lớp:** Số lượng lớp CNN có thể thay đổi tùy thuộc vào độ phức tạp của bài toán và kích thước của tập dữ liệu.
* **Số lượng filter:** Số lượng filter trong mỗi lớp cũng có thể điều chỉnh để tăng hoặc giảm khả năng biểu diễn của mô hình.

### **Ví dụ cụ thể:**

Trong mô hình bạn đưa ra:

* **Lớp CNN đầu tiên:** Có thể học các đường viền, màu sắc cơ bản của hình ảnh.
* **Lớp CNN thứ hai:** Học các hình dạng đơn giản như hình tròn, hình vuông, các bộ phận của đối tượng (ví dụ: mắt, mũi trong hình ảnh khuôn mặt).
* **Lớp CNN thứ ba:** Học các đặc trưng trừu tượng hơn, giúp phân biệt các loại đối tượng khác nhau (ví dụ: chó, mèo, xe hơi).

**Lưu ý:**

* **Không có quy tắc cứng nhắc:** Số lượng lớp CNN và cấu trúc của mô hình có thể thay đổi tùy thuộc vào từng bài toán cụ thể.
* **Thử nghiệm:** Việc chọn số lượng lớp và cấu trúc mạng thường đòi hỏi thử nghiệm và điều chỉnh để tìm ra mô hình tốt nhất.

**Các yếu tố khác ảnh hưởng đến việc lựa chọn số lượng lớp CNN:**

* **Kích thước của tập dữ liệu:** Với tập dữ liệu lớn, ta có thể sử dụng nhiều lớp hơn.
* **Độ phức tạp của bài toán:** Bài toán càng phức tạp, ta cần nhiều lớp hơn để trích xuất các đặc trưng phức tạp.
* **Khả năng tính toán:** Số lượng lớp ảnh hưởng đến thời gian huấn luyện và tài nguyên máy tính.

**Tóm lại:**

Việc sử dụng 3 lớp CNN trong mô hình trên giúp mô hình học được các đặc trưng ở nhiều cấp độ khác nhau, tăng khả năng phân loại và giảm nguy cơ overfitting. Tuy nhiên, việc lựa chọn số lượng lớp CNN phù hợp là một quyết định quan trọng và cần được cân nhắc kỹ lưỡng dựa trên từng bài toán cụ thể.

## Tại sao số lượng filters trong các lớp Conv2D tăng dần?

**Bạn hoàn toàn đúng khi nhận thấy số lượng filters trong các lớp Conv2D thường tăng dần theo các lớp trong một mạng CNN. Đây là một thiết kế phổ biến và có lý do khoa học đằng sau nó.**

### Lý giải:

* **Độ phức tạp của các đặc trưng:**
  + **Các lớp đầu:** Các lớp Conv2D đầu tiên thường học các đặc trưng đơn giản như các cạnh, góc, texture. Với số lượng filters ít, các lớp này tập trung vào việc trích xuất những đặc trưng cơ bản nhất.
  + **Các lớp sau:** Khi đi sâu vào mạng, các lớp tiếp theo sẽ kết hợp các đặc trưng đơn giản từ các lớp trước để tạo ra các đặc trưng phức tạp hơn, chẳng hạn như các hình dạng, cấu trúc. Để biểu diễn được sự đa dạng và phức tạp của các đặc trưng này, cần một số lượng filters lớn hơn.
* **Tăng khả năng biểu diễn:**
  + **Số lượng filters:** Số lượng filters càng lớn, khả năng biểu diễn của lớp đó càng cao. Điều này có nghĩa là lớp đó có thể học được nhiều đặc trưng khác nhau hơn.
  + **Tăng dần:** Bằng cách tăng dần số lượng filters theo các lớp, mô hình có thể học được các đặc trưng ở nhiều cấp độ trừu tượng khác nhau, từ đơn giản đến phức tạp.
* **Giảm mất mát thông tin:**
  + **Pooling:** Các lớp pooling giúp giảm kích thước của feature map, nhưng đồng thời cũng làm mất đi một số thông tin.
  + **Tăng filters:** Bằng cách tăng số lượng filters ở các lớp sau, mô hình có thể bù lại phần thông tin bị mất do pooling, giúp bảo toàn tốt hơn các đặc trưng quan trọng.

### Ví dụ:

* **Lớp đầu tiên:** Có thể có 32 filters để học các cạnh đơn giản.
* **Lớp thứ hai:** Có thể có 64 filters để học các hình dạng đơn giản, các kết cấu phức tạp hơn.
* **Lớp thứ ba:** Có thể có 128 filters để học các đặc trưng trừu tượng hơn, liên quan đến toàn bộ đối tượng.

### Tóm lại:

Việc tăng số lượng filters trong các lớp Conv2D theo chiều sâu của mạng là một cách hiệu quả để giúp mô hình học được các đặc trưng ngày càng phức tạp và cải thiện khả năng phân loại. Tuy nhiên, việc lựa chọn số lượng filters cụ thể cho từng lớp phụ thuộc vào nhiều yếu tố như:

## Số 3 trong input\_shape=(144,144,3) nghĩa là gì?

Trong ngữ cảnh của mạng nơ-ron tích chập (CNN) và cụ thể là tham số input\_shape trong lớp Conv2D của Keras, số 3 đại diện cho **số kênh màu** của hình ảnh đầu vào.

### Giải thích chi tiết:

* **(144, 144):** Hai số này lần lượt biểu thị **chiều cao** và **chiều rộng** của hình ảnh, tức là hình ảnh của bạn có kích thước 144x144 pixel.
* **3:** Số 3 cho biết hình ảnh đó là hình ảnh màu, được biểu diễn dưới dạng 3 kênh màu cơ bản: **Đỏ (Red), Lục (Green) và Lam (Blue)**. Mỗi pixel trong hình ảnh sẽ có 3 giá trị tương ứng với cường độ của mỗi kênh màu này.

### Tại sao lại là 3 kênh màu?

* **Hình ảnh màu:** Hầu hết các hình ảnh chúng ta nhìn thấy hàng ngày đều là hình ảnh màu. Để máy tính hiểu được hình ảnh màu, ta cần phân tách hình ảnh thành 3 kênh màu cơ bản.
* **Biểu diễn:** Mỗi pixel trong hình ảnh sẽ được biểu diễn bởi một vector có 3 phần tử, mỗi phần tử tương ứng với giá trị của một kênh màu. Ví dụ, một pixel có màu đỏ tươi sẽ có giá trị lớn ở kênh đỏ và gần bằng 0 ở kênh xanh và lam.

### Ví dụ khác:

* **Hình ảnh xám:** Nếu bạn làm việc với hình ảnh xám, chỉ có một kênh màu, vậy input\_shape sẽ là (144, 144, 1).
* **Hình ảnh với nhiều kênh:** Trong một số trường hợp, hình ảnh có thể có nhiều hơn 3 kênh, ví dụ như hình ảnh hồng ngoại, hình ảnh radar. Khi đó, số 3 sẽ được thay thế bằng số kênh tương ứng.

### Tóm lại:

Số 3 trong input\_shape=(144,144,3) cho biết hình ảnh đầu vào là hình ảnh màu và có 3 kênh màu: đỏ, xanh lá và xanh dương. Đây là thông tin quan trọng để mạng CNN hiểu được cấu trúc của dữ liệu đầu vào và tiến hành quá trình trích xuất đặc trưng.

## Giải thích chi tiết về Conv2D(32,(3,3))

Trong ngữ cảnh của mạng nơ-ron tích chập (Convolutional Neural Network - CNN), dòng code Conv2D(32,(3,3)) được sử dụng để thêm một lớp tích chập 2D vào mô hình. Lớp này thực hiện phép tích chập trên dữ liệu đầu vào để trích xuất các đặc trưng.

### Ý nghĩa của các tham số:

* **32:** Đây là số lượng **bộ lọc (filters)** được sử dụng trong lớp này. Mỗi bộ lọc sẽ học một đặc trưng khác nhau từ hình ảnh. Ví dụ, một bộ lọc có thể học cách nhận biết các cạnh thẳng đứng, một bộ lọc khác có thể học cách nhận biết các cạnh ngang, v.v. Số lượng bộ lọc càng lớn, khả năng học các đặc trưng của lớp càng cao.
* **(3,3):** Đây là **kích thước của bộ lọc** (kernel size). Bộ lọc này sẽ quét qua hình ảnh đầu vào với các cửa sổ có kích thước 3x3 pixel. Bằng cách thực hiện phép tích chập giữa bộ lọc và vùng ảnh tương ứng, ta sẽ thu được một giá trị mới tại mỗi vị trí của feature map.

### Cách hoạt động:

1. **Bộ lọc quét qua hình ảnh:** Bộ lọc 3x3 sẽ di chuyển từ trái sang phải và từ trên xuống dưới trên hình ảnh đầu vào.
2. **Tính toán tích chập:** Tại mỗi vị trí, bộ lọc sẽ nhân từng phần tử tương ứng của bộ lọc với vùng ảnh tương ứng và cộng tất cả các tích lại. Kết quả thu được sẽ là một giá trị tại vị trí đó của feature map.
3. **Tạo feature map:** Quá trình này được lặp lại cho tất cả các vị trí của hình ảnh và tất cả các bộ lọc, tạo ra một feature map mới. Feature map này sẽ chứa các đặc trưng được trích xuất từ hình ảnh đầu vào.

### Tại sao sử dụng lớp Conv2D?

* **Trích xuất đặc trưng:** Lớp Conv2D là một trong những công cụ hiệu quả nhất để trích xuất các đặc trưng từ hình ảnh. Nó giúp mạng nơ-ron học được các đặc trưng cục bộ của hình ảnh, chẳng hạn như các cạnh, góc, texture.
* **Giảm số lượng tham số:** So với các lớp fully connected, lớp Conv2D có số lượng tham số ít hơn đáng kể, giúp giảm nguy cơ overfitting.
* **Bất biến dịch chuyển:** Lớp Conv2D có tính bất biến dịch chuyển, nghĩa là nếu một đối tượng trong hình ảnh được dịch chuyển một chút, mạng vẫn có thể nhận ra nó.

**Tóm lại:** Dòng code Conv2D(32,(3,3)) tạo ra một lớp tích chập 2D với 32 bộ lọc, mỗi bộ lọc có kích thước 3x3. Lớp này sẽ học các đặc trưng cơ bản từ hình ảnh đầu vào và là một thành phần quan trọng trong các mạng nơ-ron tích chập.

## Tại sao sử dụng model.add(Flatten()) trong mạng CNN?

**Lớp Flatten** trong Keras có vai trò quan trọng trong việc chuyển đổi dữ liệu đầu ra từ các lớp tích chập (Convolutional Layers) thành một vector một chiều trước khi đưa vào các lớp fully connected.

**Tại sao lại cần chuyển đổi này?**

* **Các lớp fully connected:** Các lớp fully connected (Dense) trong mạng neural network truyền thống hoạt động trên các vector. Mỗi neuron trong lớp fully connected sẽ kết nối với tất cả các neuron ở lớp trước.
* **Đầu ra của các lớp Conv2D:** Đầu ra của các lớp Conv2D là các tensor đa chiều, mỗi tensor đại diện cho một feature map. Feature map này chứa thông tin về các đặc trưng đã được trích xuất từ hình ảnh.

**Vai trò của lớp Flatten:**

* **Chuyển đổi tensor thành vector:** Lớp Flatten sẽ "duỗi thẳng" tensor đa chiều thành một vector một chiều. Điều này giúp cho dữ liệu đầu ra của các lớp Conv2D trở nên phù hợp để làm đầu vào cho các lớp fully connected.
* **Kết nối các lớp:** Lớp Flatten tạo ra một kết nối giữa phần trích xuất đặc trưng (Convolutional part) và phần phân loại (Fully connected part) của mạng.

**Ví dụ:**

Giả sử đầu ra của lớp Conv2D cuối cùng là một tensor có kích thước (7, 7, 64). Điều này có nghĩa là ta có 64 feature map, mỗi feature map có kích thước 7x7. Khi cho qua lớp Flatten, tensor này sẽ được "duỗi thẳng" thành một vector có kích thước 7764 = 3136. Vector này sẽ được đưa vào lớp Dense đầu tiên.

**Tóm lại:**

Lớp Flatten đóng vai trò cầu nối giữa phần trích xuất đặc trưng và phần phân loại trong mạng CNN. Nó đảm bảo rằng dữ liệu đầu vào cho các lớp fully connected có đúng định dạng.

**Khi nào nên sử dụng Flatten?**

* Luôn được sử dụng sau các lớp Conv2D khi bạn muốn chuyển sang các lớp fully connected.
* Có thể được sử dụng để kết nối các phần khác nhau của mạng, ví dụ như giữa các module khác nhau.

**Lưu ý:**

* Vị trí của lớp Flatten trong mạng CNN rất quan trọng. Nếu đặt lớp Flatten quá sớm, mạng có thể mất đi thông tin không gian của hình ảnh.
* Trong một số trường hợp, bạn có thể sử dụng các kỹ thuật khác như Global Average Pooling thay vì Flatten để giảm số lượng tham số và cải thiện khả năng tổng quát của mô hình.

## Giải thích chi tiết về các lớp ẩn và lớp đầu ra

### Lớp ẩn (Dense(512, activation=tf.nn.relu))

* **512 nơ-ron:** Số lượng nơ-ron trong một lớp ẩn quyết định khả năng biểu diễn của mô hình. Một số lượng nơ-ron lớn hơn cho phép mô hình học được những đặc trưng phức tạp hơn. Tuy nhiên, quá nhiều nơ-ron có thể dẫn đến hiện tượng overfitting (mô hình học quá khớp với dữ liệu huấn luyện). Việc chọn 512 nơ-ron trong trường hợp này là một sự cân nhắc giữa khả năng biểu diễn và độ phức tạp của mô hình.
* **Hàm kích hoạt ReLU:** ReLU (Rectified Linear Unit) là một hàm kích hoạt phổ biến trong các mạng neural. Nó có dạng f(x) = max(0, x). Hàm ReLU giúp cho việc tính toán trở nên nhanh hơn và giảm thiểu vấn đề gradient vanishing so với các hàm kích hoạt sigmoid hay tanh.

**Tại sao cần lớp ẩn?** Lớp ẩn giúp mô hình học được các đặc trưng trừu tượng hơn từ dữ liệu đầu vào. Sau khi các lớp tích chập trích xuất các đặc trưng thấp cấp (như cạnh, góc), lớp ẩn sẽ kết hợp các đặc trưng này để tạo thành các đặc trưng cấp cao hơn, có ý nghĩa hơn đối với bài toán phân loại.

### Lớp đầu ra (Dense(9, activation=tf.nn.softmax))

* **9 nơ-ron:** Số lượng nơ-ron trong lớp đầu ra bằng với số lớp cần phân loại. Trong trường hợp này, bạn có 9 lớp phân loại, nên lớp đầu ra có 9 nơ-ron. Mỗi nơ-ron sẽ đại diện cho xác suất thuộc về một lớp cụ thể.
* **Hàm kích hoạt softmax:** Hàm softmax được sử dụng để chuyển đổi vector đầu ra của lớp fully connected thành một phân phối xác suất. Giá trị của mỗi nơ-ron trong vector đầu ra sẽ được chia cho tổng của tất cả các giá trị trong vector đó. Kết quả là một vector các giá trị nằm trong khoảng từ 0 đến 1 và tổng bằng 1. Giá trị lớn nhất trong vector này sẽ tương ứng với lớp mà mô hình dự đoán.

**Tại sao cần lớp đầu ra và hàm softmax?** Lớp đầu ra cung cấp kết quả cuối cùng của mô hình, tức là xác suất mà một mẫu dữ liệu thuộc về mỗi lớp. Hàm softmax đảm bảo rằng tổng các xác suất này bằng 1, giúp ta dễ dàng so sánh và đưa ra quyết định cuối cùng.

**Tóm lại:**

* **Lớp ẩn:** Học các đặc trưng trừu tượng từ dữ liệu, giúp mô hình phân loại chính xác hơn.
* **Lớp đầu ra:** Đưa ra dự đoán cuối cùng về lớp của mẫu dữ liệu.
* **Hàm softmax:** Chuyển đổi đầu ra của lớp cuối cùng thành một phân phối xác suất.

**Các yếu tố ảnh hưởng đến việc lựa chọn số lượng nơ-ron và hàm kích hoạt:**

* **Kích thước của tập dữ liệu:** Tập dữ liệu lớn hơn thường yêu cầu nhiều nơ-ron hơn.
* **Độ phức tạp của bài toán:** Bài toán phức tạp hơn cần nhiều nơ-ron và các lớp ẩn hơn.
* **Khả năng tính toán:** Số lượng nơ-ron quá lớn có thể làm tăng thời gian huấn luyện.

# P7:

|  |
| --- |
| model.compile(loss = 'sparse\_categorical\_crossentropy',                optimizer=keras.optimizers.Adam(learning\_rate = 0.0001),                metrics=['accuracy']) |

## Giải thích đoạn code model.compile()

**Đoạn code này được sử dụng để biên dịch mô hình (model) trước khi bắt đầu quá trình huấn luyện.**

### Ý nghĩa của từng phần:

* **model:** Đây là biến đại diện cho toàn bộ kiến trúc của mạng thần kinh mà bạn đã xây dựng. Nó bao gồm các lớp (layers) như các lớp tích chập (convolutional), lớp pooling, lớp fully connected, v.v...
* **compile:** Đây là một hàm trong Keras dùng để cấu hình quá trình huấn luyện mô hình. Nó xác định cách mô hình sẽ được huấn luyện, bao gồm hàm mất mát, thuật toán tối ưu hóa và các metrics để đánh giá hiệu suất.
* **loss = 'sparse\_categorical\_crossentropy':**
  + **loss:** Đây là hàm mất mát (loss function) được sử dụng để đo lường sự khác biệt giữa giá trị dự đoán của mô hình và giá trị thực tế.
  + **sparse\_categorical\_crossentropy:** Hàm này thường được sử dụng cho các bài toán phân loại đa lớp khi nhãn của các lớp là các số nguyên (ví dụ: 0, 1, 2, ...). Nó sẽ tính toán một giá trị mất mát dựa trên xác suất mà mô hình dự đoán cho lớp đúng và so sánh với nhãn thực tế.
* **optimizer=keras.optimizers.Adam(learning\_rate = 0.0001):**
  + **optimizer:** Thuật toán tối ưu hóa được sử dụng để cập nhật các trọng số của mô hình trong quá trình huấn luyện.
  + **Adam:** Adam là một thuật toán tối ưu hóa phổ biến, thường cho kết quả tốt trong nhiều bài toán.
  + **learning\_rate:** Tốc độ học, xác định mức độ thay đổi của các trọng số trong mỗi bước cập nhật. Giá trị learning\_rate quá lớn có thể khiến mô hình không hội tụ, quá nhỏ có thể làm quá trình huấn luyện chậm.
* **metrics=['accuracy']:**
  + **metrics:** Các chỉ số đánh giá hiệu suất của mô hình.
  + **accuracy:** Độ chính xác, tỷ lệ các mẫu được phân loại đúng.

### Tóm tắt:

Đoạn code trên nói rằng: "Hãy biên dịch mô hình của tôi, sử dụng hàm mất mát là cross-entropy cho các bài toán phân loại đa lớp, sử dụng thuật toán Adam để tối ưu hóa với tốc độ học là 0.0001, và đánh giá hiệu suất bằng độ chính xác."

### Ví dụ minh họa:

Giả sử bạn đang xây dựng một mô hình để phân loại các loại hình ảnh (ví dụ: mèo, chó, chim). Khi chạy đoạn code này, bạn đang nói với máy tính rằng:

* **Mục tiêu:** Dự đoán chính xác một hình ảnh thuộc về lớp nào trong 3 lớp (mèo, chó, chim).
* **Cách đo lường sai số:** Sử dụng hàm cross-entropy để đo mức độ sai lệch giữa dự đoán của mô hình và nhãn thực tế.
* **Cách cập nhật mô hình:** Sử dụng thuật toán Adam để điều chỉnh các tham số của mô hình sao cho giảm thiểu hàm mất mát.
* **Cách đánh giá:** Sử dụng độ chính xác để đánh giá hiệu suất của mô hình trên tập dữ liệu kiểm tra.

**Sau khi biên dịch, bạn có thể bắt đầu huấn luyện mô hình bằng hàm model.fit()**

# P8:

|  |
| --- |
| class myCallback(tf.keras.callbacks.Callback):    def on\_epoch\_end(self, epoch, logs={}):      if(logs.get('accuracy')==1):        print("\nReached  100% accuracy so cancelling training!")        self.model.stop\_training = True  callbacks = myCallback() |

## Giải thích đoạn code

**Mục đích:** Đoạn code này định nghĩa một lớp callback tùy chỉnh (myCallback) để giám sát quá trình huấn luyện mô hình và dừng huấn luyện khi đạt được độ chính xác 100%.

**Phân tích chi tiết:**

1. **class myCallback(tf.keras.callbacks.Callback)::**
   * Tạo một lớp mới tên là myCallback kế thừa từ lớp tf.keras.callbacks.Callback. Điều này cho phép lớp của chúng ta được gọi vào các thời điểm cụ thể trong quá trình huấn luyện.
2. **def on\_epoch\_end(self, epoch, logs={}):**
   * Định nghĩa một phương thức on\_epoch\_end sẽ được gọi vào cuối mỗi epoch (một chu kỳ huấn luyện).
   * Tham số epoch đại diện cho số epoch hiện tại.
   * Tham số logs là một dictionary chứa các chỉ số hiệu suất của mô hình trong epoch đó.
3. **if(logs.get('accuracy')==1)::**
   * Kiểm tra xem độ chính xác (accuracy) trong epoch hiện tại có đạt 100% hay không. Nếu đạt 100%, điều này có nghĩa là mô hình đã học được hoàn toàn dữ liệu huấn luyện.
4. **print("\nReached  100% accuracy so cancelling training!"):**
   * In ra thông báo thông báo rằng đã đạt được 100% độ chính xác và sẽ ngừng huấn luyện.
5. **self.model.stop\_training = True:**
   * Thiết lập thuộc tính stop\_training của mô hình thành True. Điều này sẽ thông báo cho Keras ngừng quá trình huấn luyện.
6. **callbacks = myCallback():**
   * Tạo một đối tượng myCallback. Đối tượng này sẽ được sử dụng để giám sát quá trình huấn luyện.

**Tổng kết:**

Đoạn code này tạo ra một callback tùy chỉnh để tự động dừng quá trình huấn luyện khi mô hình đạt được 100% độ chính xác trên tập dữ liệu huấn luyện. Điều này có thể hữu ích để tiết kiệm thời gian và tài nguyên khi mô hình đã học được hoàn toàn dữ liệu.

**Lưu ý:**

* Đôi khi, đạt được 100% độ chính xác trên tập huấn luyện có thể là dấu hiệu của overfitting, nghĩa là mô hình học quá khớp với dữ liệu huấn luyện và có thể không hoạt động tốt trên dữ liệu mới. Vì vậy, cần kết hợp với các phương pháp khác để đánh giá hiệu suất của mô hình trên tập kiểm định.
* Bạn có thể tùy chỉnh điều kiện dừng huấn luyện theo nhu cầu của mình, ví dụ như dừng khi độ chính xác đạt một mức nhất định hoặc khi mất mát giảm xuống dưới một ngưỡng nào đó.

## Epoch trong huấn luyện mô hình máy học là gì?

**Epoch** là một thuật ngữ quan trọng trong quá trình huấn luyện các mô hình học sâu, đặc biệt là các mạng thần kinh nhân tạo. Nó đại diện cho **một lần duyệt qua toàn bộ tập dữ liệu huấn luyện**.

**Giải thích chi tiết:**

* **Tập dữ liệu huấn luyện:** Đây là tập dữ liệu mà mô hình sử dụng để học các quy tắc và mối quan hệ giữa dữ liệu đầu vào và đầu ra.
* **Một lần duyệt:** Trong mỗi epoch, mô hình sẽ xem xét từng mẫu dữ liệu trong tập huấn luyện một lần để cập nhật các tham số (weight và bias) của mô hình.
* **Cập nhật tham số:** Sau khi xử lý xong một mẫu dữ liệu, mô hình sẽ tính toán lỗi (loss) giữa giá trị dự đoán và giá trị thực tế, sau đó sử dụng thuật toán tối ưu hóa (ví dụ: Gradient Descent) để điều chỉnh các tham số sao cho giảm thiểu lỗi.

**Ví dụ:** Giả sử bạn có một tập dữ liệu huấn luyện gồm 1000 ảnh mèo và chó. Với batch size là 100, mỗi epoch sẽ bao gồm 10 lần cập nhật tham số (1000 ảnh / 100 ảnh/batch = 10 batch). Trong mỗi batch, mô hình sẽ xử lý 100 ảnh và cập nhật các tham số.

**Tại sao lại cần nhiều epoch?**

* **Học sâu:** Các mạng thần kinh nhân tạo có nhiều lớp và tham số. Để học được các đặc trưng phức tạp, mô hình cần được huấn luyện nhiều lần trên cùng một tập dữ liệu.
* **Tối ưu hóa:** Quá trình tối ưu hóa là một quá trình tìm kiếm không gian tham số để tìm ra bộ tham số tốt nhất. Việc lặp đi lặp lại nhiều lần sẽ giúp mô hình tìm được bộ tham số gần với điểm tối ưu nhất.

**Các yếu tố liên quan đến epoch:**

* **Batch size:** Số lượng mẫu dữ liệu được sử dụng để tính toán gradient và cập nhật tham số trong mỗi lần cập nhật.
* **Learning rate:** Tốc độ mà mô hình cập nhật các tham số.
* **Số epoch:** Số lần lặp qua toàn bộ tập dữ liệu.

**Các vấn đề có thể gặp phải khi huấn luyện:**

* **Overfitting:** Mô hình học quá khớp với dữ liệu huấn luyện, dẫn đến hiệu suất kém trên dữ liệu kiểm tra.
* **Underfitting:** Mô hình không học được đủ thông tin từ dữ liệu huấn luyện, dẫn đến hiệu suất kém trên cả tập huấn luyện và kiểm tra.

**Cách chọn số epoch:**

* **Validation set:** Chia tập dữ liệu thành tập huấn luyện và tập kiểm tra. Sau mỗi epoch, đánh giá hiệu suất của mô hình trên tập kiểm tra.
* **Early stopping:** Dừng quá trình huấn luyện khi hiệu suất trên tập kiểm tra không còn cải thiện nữa để tránh overfitting.
* **Hyperparameter tuning:** Thử nghiệm với các giá trị khác nhau của số epoch, batch size, learning rate để tìm ra bộ siêu tham số tốt nhất cho mô hình.

**Tóm lại:**

Epoch là một khái niệm quan trọng trong huấn luyện mô hình máy học. Nó quyết định số lần mô hình được tiếp xúc với toàn bộ tập dữ liệu. Việc chọn số epoch phù hợp là rất quan trọng để đạt được hiệu suất tốt nhất của mô hình.

history = model.fit(train\_generator,

                    epochs = 20, verbose = 1,

                    validation\_data = validation\_generator,

                    callbacks=[callbacks]) đoạn code này dùng để làm gì

Hiện bản thảo

## Giải thích đoạn code model.fit(train\_generator, ...)

Đoạn code này được sử dụng để **huấn luyện mô hình** (model) trong Keras/TensorFlow.

**Phân tích chi tiết:**

* **model.fit(train\_generator, ...):**
  + model: Biến đại diện cho mô hình mà bạn đã xây dựng.
  + train\_generator: Đây là một generator (bộ tạo dữ liệu) cung cấp dữ liệu huấn luyện từng batch cho mô hình. Generator giúp tiết kiệm bộ nhớ bằng cách không tải toàn bộ dữ liệu vào bộ nhớ cùng một lúc.
  + Dấu ba chấm (...) thể hiện các tham số khác có thể được truyền vào hàm fit.

**Các tham số quan trọng:**

* **epochs = 20:** Số lần lặp qua toàn bộ tập dữ liệu huấn luyện (số epoch).
* **verbose = 1:** Kiểu in thông tin:
  + 0: Không in thông tin.
  + 1: In thông tin tóm tắt sau mỗi epoch.
  + 2: In chi tiết hơn về quá trình huấn luyện.
* **validation\_data = validation\_generator:** Tập dữ liệu kiểm định (validation set). Đây là một generator cung cấp dữ liệu kiểm định từng batch để theo dõi hiệu suất của mô hình trong quá trình huấn luyện. Kiểm tra hiệu suất trên tập kiểm định giúp tránh overfitting.
* **callbacks=[callbacks]:** Danh sách các callback (hàm gọi lại). Trong trường hợp này, bạn đang sử dụng callback callbacks được định nghĩa trước đó để dừng huấn luyện khi đạt được 100% accuracy.

**Tóm tắt:**

Đoạn code này bắt đầu quá trình huấn luyện mô hình. Mô hình sẽ được huấn luyện trên dữ liệu được cung cấp bởi train\_generator trong 20 epoch (20 lần lặp qua toàn bộ tập dữ liệu huấn luyện). Hiệu suất của mô hình sẽ được theo dõi trên tập kiểm định được cung cấp bởi validation\_generator. Callback callbacks sẽ được kích hoạt sau mỗi epoch để theo dõi độ chính xác và có thể dừng huấn luyện sớm nếu cần thiết.

**Lưu ý:**

* Quá trình huấn luyện có thể mất một khoảng thời gian tùy thuộc vào kích thước và độ phức tạp của mô hình, kích thước của tập dữ liệu, và tài nguyên tính toán có sẵn.
* Trong quá trình huấn luyện, bạn có thể theo dõi các chỉ số như loss, accuracy trên tập huấn luyện và tập kiểm định để đánh giá hiệu suất của mô hình.

|  |
| --- |
| #đánh giá chất lượng của mô hình và vẽ lại  score = model.evaluate(validation\_generator, verbose=0)  print('Sai số kiểm tra là: ', score[0])  print('Độ chính xác kiểm tra là: ', score[1])  plt.figure(figsize=(10, 6))  plt.plot(history.history['loss'], label='Training Loss')  plt.plot(history.history['accuracy'], label='Training Accuracy')  plt.plot(history.history['val\_loss'], label='Validation Loss')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  plt.title('NhanDienTien\_Training')  plt.xlabel('Epoch')  plt.ylabel('Value')  plt.legend()  plt.show() |

## Giải thích đoạn code và đồ thị

### Đoạn code đánh giá mô hình:

Python

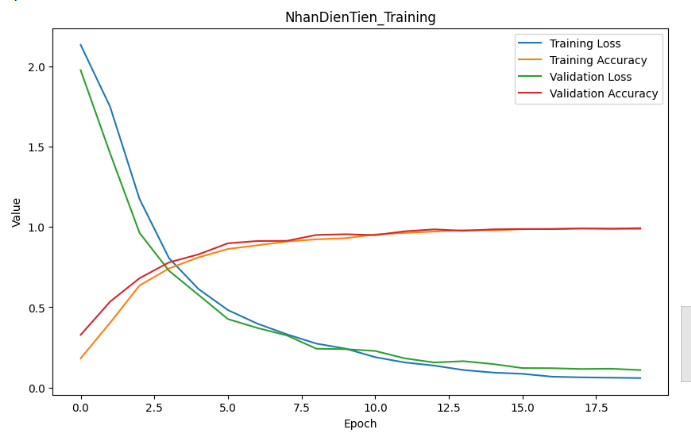
score = model.evaluate(validation\_generator, verbose=0)

print('Sai số kiểm tra là: ', score[0])

print('Độ chính xác kiểm tra là: ', score[1])

* **model.evaluate(validation\_generator, verbose=0):** Dòng này dùng để đánh giá mô hình trên tập dữ liệu kiểm định (validation set) được cung cấp bởi validation\_generator. Tham số verbose=0 nghĩa là không in thông tin chi tiết trong quá trình đánh giá.
* **score:** Đây là một list chứa các giá trị đánh giá. Thông thường, giá trị đầu tiên (score[0]) là sai số (loss) và giá trị thứ hai (score[1]) là độ chính xác (accuracy).
* **In ra sai số và độ chính xác:** Hai dòng print tiếp theo in ra các giá trị này để bạn dễ dàng theo dõi.

### Đoạn code vẽ đồ thị:



Python

plt.figure(figsize=(10, 6))

# ... (các dòng vẽ đồ thị)

Hãy [thận trọng](/faq#coding) khi sử dụng các đoạn mã.

Đoạn code này sử dụng thư viện Matplotlib để vẽ đồ thị biểu diễn sự thay đổi của các chỉ số (loss và accuracy) trên cả tập huấn luyện và tập kiểm định trong quá trình huấn luyện.

* **Các đường trong đồ thị:**
  + **Training Loss:** Đường biểu diễn giá trị loss (sai số) trên tập huấn luyện theo từng epoch.
  + **Training Accuracy:** Đường biểu diễn độ chính xác trên tập huấn luyện.
  + **Validation Loss:** Đường biểu diễn giá trị loss trên tập kiểm định.
  + **Validation Accuracy:** Đường biểu diễn độ chính xác trên tập kiểm định.

### Ý nghĩa của đồ thị

* **Hiểu rõ quá trình huấn luyện:** Đồ thị giúp bạn trực quan hóa cách mô hình học và cải thiện theo thời gian.
* **Phát hiện overfitting:**
  + Nếu đường Training Loss giảm đều và Training Accuracy tăng đều nhưng Validation Loss tăng và Validation Accuracy giảm hoặc không cải thiện, đó là dấu hiệu của overfitting. Mô hình học quá khớp với dữ liệu huấn luyện và sẽ không hoạt động tốt trên dữ liệu mới.
* **Phát hiện underfitting:**
  + Nếu cả Training Loss và Validation Loss đều cao và không giảm nhiều, đó là dấu hiệu của underfitting. Mô hình không học được đủ thông tin từ dữ liệu.
* **Đánh giá hiệu suất:**
  + Bằng cách so sánh giá trị của các đường trên tập huấn luyện và tập kiểm định, bạn có thể đánh giá hiệu suất của mô hình và quyết định xem có cần điều chỉnh các hyperparameter (ví dụ: số epoch, learning rate) hay không.

### Cách sử dụng đồ thị để cải thiện mô hình:

* **Nếu overfitting:**
  + Giảm số lượng epoch.
  + Sử dụng kỹ thuật regularization (L1/L2, dropout).
  + Tăng kích thước của tập dữ liệu.
* **Nếu underfitting:**
  + Tăng số lượng epoch.
  + Sử dụng mô hình phức tạp hơn.
  + Tăng số lượng features.
* **Nếu cả hai:**
  + Điều chỉnh hyperparameter (learning rate, batch size).
  + Sử dụng kỹ thuật augmentation để tăng số lượng dữ liệu.

**Tóm lại:**

Đoạn code này cung cấp một cách hiệu quả để đánh giá hiệu suất của mô hình và theo dõi quá trình huấn luyện. Đồ thị giúp bạn hiểu rõ hơn về cách mô hình hoạt động và từ đó đưa ra các quyết định để cải thiện hiệu suất.

**Mục đích:** Đoạn code này được sử dụng để dự đoán nhãn (mệnh giá tiền) của một hình ảnh được tải lên Colab.

**Phân tích chi tiết:**

1. **Import thư viện:**
   * from google.colab import files: Import thư viện files từ Colab để tải lên tệp.
   * from keras.preprocessing import image: Import thư viện image từ Keras để xử lý hình ảnh.
   * %matplotlib inline: Cấu hình Matplotlib để hiển thị trực tiếp trong notebook.
   * import matplotlib.image as mpimg: Import thư viện mpimg để đọc hình ảnh.
   * import cv2: Import thư viện OpenCV để xử lý hình ảnh.
2. **Tải lên hình ảnh:**
   * uploaded=files.upload(): Cho phép người dùng tải lên một hình ảnh.
3. **Xử lý hình ảnh:**
   * path='/content/'+fn: Tạo đường dẫn đến hình ảnh đã tải lên.
   * plt.imshow(mpimg.imread(path)): Hiển thị hình ảnh đã tải lên.
   * img = load\_img(path, target\_size=(144,144)): Đọc hình ảnh và resize thành kích thước 144x144.
   * plt.imshow(img): Hiển thị hình ảnh đã resize.
   * img=img\_to\_array(img): Chuyển đổi hình ảnh thành một mảng NumPy.
   * img=img.astype('float32'): Chuyển đổi kiểu dữ liệu của mảng thành float32.
   * img=img/255: Chuẩn hóa giá trị pixel về khoảng 0-1.
   * img=np.expand\_dims(img,axis=0): Thêm một chiều mới vào mảng để phù hợp với yêu cầu của mô hình.
4. **Dự đoán nhãn:**
   * result=loaded\_model.predict(img): Dùng mô hình đã được tải (loaded\_model) để dự đoán nhãn của hình ảnh.
   * Kiểm tra giá trị dự đoán cho từng lớp và in ra nhãn tương ứng.

**Tổng kết:**

Đoạn code này thực hiện các bước sau:

1. Tải lên một hình ảnh.
2. Hiển thị hình ảnh.
3. Chuẩn bị hình ảnh cho mô hình (resize, chuyển đổi kiểu dữ liệu, chuẩn hóa).
4. Dùng mô hình để dự đoán nhãn của hình ảnh.
5. In ra kết quả dự đoán.

**Lưu ý:**

* Để sử dụng đoạn code này, bạn cần có một mô hình đã được huấn luyện và tải vào biến loaded\_model.
* Các lớp phân loại được giả định là từ 0 đến 8, tương ứng với các nhãn tiền từ '001000' đến '500000'. Nếu các lớp khác nhau, cần điều chỉnh phần kiểm tra và in nhãn cho phù hợp.
* Đoạn code này chỉ là một ví dụ cơ bản. Bạn có thể tùy chỉnh và mở rộng nó để phù hợp với yêu cầu cụ thể của bài toán của bạn.