

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NGOẠI NGỮ - TIN HỌC
THÀNH PHỐ HỒ CHÍ MINH**



ĐỒ ÁN MÔN HỌC: THỊ GIÁC MÁY TÍNH
TÌM HIỂU VỀ THUẬT TOÁN VGG16

GVHD: Nguyễn Thanh Bình

SV: Dương Thái Bảo – MSSV: 20DH110422

SV: Huỳnh Thanh Vạn – MSSV: 22DH114808

SV: Ưông Thanh Trung – MSSV: 22DH113985

SV: Tạ Ngọc Thiện – MSSV: 22DH114748

TP. HỒ CHÍ MINH 12/2023

MỤC LỤC

Chương 1: Giới thiệu	3
1.1 Giới thiệu đề tài	3
1.2 Mục tiêu và nội dung đề tài.....	3
1.3 Giới hạn đề tài	4
Chương 2: Cơ sở lý thuyết	7
2.1 Cấu trúc của VGG16	7
2.2 Đặc điểm nổi bật của VGG16	8
2.3 Nguyên tắc hoạt động của thuật toán VGG16	8
Chương 3: Phương Pháp	11
3.1 Yêu cầu bài toán	11
3.2 Giải thuật thực hiện	12
2.4 . Phương pháp đánh giá:	15
Chương 4: Hiện thực kết quả	17
4.1 Yêu cầu hệ thống và tập dữ liệu thực nghiệm.	17
4.1.1 Yêu cầu hệ thống.....	17
4.1.2 Tập dữ liệu thực nghiệm	17
4.2 Kết quả thực nghiệm.....	17
Chương 5: Kết luận	19
5.1 Kết quả đạt được	19
5.2 Ưu và nhược điểm của thuật toán VGG16	19
5.2.2 Ưu điểm:	19
5.2.3 Nhược điểm	19
5.3 Hướng mở rộng tương lai.....	19
TÀI LIỆU THAM KHẢO	20
PHỤ LỤC	21

Chương 1: Giới thiệu

1.1 Giới thiệu đề tài

- VGG16 là một kiến trúc mạng nơ-ron tích chập (Convolutional Neural Network - CNN) nổi tiếng, được phát triển bởi nhóm nghiên cứu Visual Geometry Group (VGG) tại Đại học Oxford, do Simonyan và Zisserman đề xuất trong bài báo "Very Deep Convolutional Networks for Large-Scale Image Recognition" (2014). VGG16 là một trong những mạng CNN phổ biến nhất trong lĩnh vực nhận dạng hình ảnh và đã đạt được kết quả ấn tượng trên các bộ dữ liệu lớn như ImageNet.

1.2 Mục tiêu và nội dung đề tài

- Mục tiêu chính của thuật toán VGG16 là xây dựng một mô hình mạng nơ-ron tích chập (CNN) có khả năng học và nhận dạng các đặc trưng phức tạp trong hình ảnh một cách hiệu quả, với một kiến trúc đơn giản nhưng mạnh mẽ. Mục tiêu này được thể hiện qua các điểm sau:
 - Cải thiện khả năng nhận dạng hình ảnh: VGG16 được thiết kế để phân loại hình ảnh vào các lớp cụ thể, đạt hiệu suất rất cao trong các bài toán nhận dạng hình ảnh trên bộ dữ liệu lớn như ImageNet. Mô hình này đã đạt được độ chính xác cao trong các thách thức nhận dạng hình ảnh và giúp phát triển các ứng dụng trong nhiều lĩnh vực như y tế, giao thông, và bảo mật.
 - Khám phá hiệu quả của mạng sâu (deep networks): VGG16 là một trong những mạng sâu nổi bật (với 16 lớp) trong thời kỳ ra mắt, và nó đã chứng minh rằng việc tăng độ sâu của mạng nơ-ron có thể cải thiện khả năng học các đặc trưng phức tạp và trừu tượng hơn từ dữ liệu hình ảnh.
 - Đơn giản hóa cấu trúc mô hình: VGG16 sử dụng một cấu trúc mạng nơ-ron đơn giản nhưng hiệu quả, với các lớp tích chập 3x3 và pooling, giúp dễ dàng hiểu và triển khai. Điều này cũng giúp giảm thiểu sự phức tạp trong việc điều chỉnh và tối ưu hóa mô hình.
 - Phát triển mô hình học sâu (Deep Learning) có thể ứng dụng trong các tình huống thực tế: Thuật toán VGG16 không chỉ tập trung vào nghiên cứu lý thuyết mà còn có mục tiêu ứng dụng vào các hệ thống nhận dạng hình ảnh

thực tế, từ nhận diện đối tượng trong ảnh, phân tích hình ảnh y tế cho đến các hệ thống xe tự lái.

1.3 Giới hạn đề tài

1.3.1 Phần nhóm chưa làm

1. Nhận dạng khuôn mặt (Face Recognition)

VGG16 có thể được sử dụng để nhận dạng khuôn mặt trong các ứng dụng như an ninh, hệ thống kiểm soát ra vào, hay các ứng dụng điện thoại di động.

+ **An ninh và giám sát:** Nhận diện và xác thực khuôn mặt người trong các hệ thống bảo mật.

+ **Ứng dụng tiêu dùng:** Các ứng dụng điện thoại thông minh dùng để nhận diện khuôn mặt người dùng (như mở khóa điện thoại).

2. Phân tích cảm xúc (Emotion Recognition)

Mô hình VGG16 có thể được sử dụng để nhận diện cảm xúc của con người từ các bức ảnh khuôn mặt. Mô hình này có thể phân loại các biểu cảm khuôn mặt thành các nhóm cảm xúc như vui vẻ, buồn bã, giận dữ, sợ hãi, vv.

+ **Ứng dụng trong giáo dục và chăm sóc sức khỏe:** Phân tích cảm xúc của học sinh, bệnh nhân hoặc người cao tuổi trong các ứng dụng giáo dục hoặc chăm sóc sức khỏe từ xa.

3. Phân tích y tế (Medical Image Analysis)

VGG16 có thể được áp dụng trong phân tích hình ảnh y tế, chẳng hạn như nhận diện các vấn đề liên quan đến bệnh lý từ hình ảnh X-quang, CT scan, MRI, hoặc ảnh chụp tế bào.

+ **Chẩn đoán hình ảnh:** Phát hiện và phân loại các dấu hiệu của bệnh trong hình ảnh y tế như phát hiện ung thư, u não, bệnh tim mạch, vv.

+ **Hệ thống hỗ trợ chẩn đoán:** Giúp bác sĩ xác định các đặc điểm bất thường trong hình ảnh y tế để đưa ra quyết định chính xác hơn.

4. Xử lý ảnh nghệ thuật (Artistic Image Generation)

VGG16 có thể được sử dụng trong các mô hình tạo ra hình ảnh nghệ thuật hoặc chuyển đổi phong cách (style transfer). Mô hình có thể học các đặc trưng của phong cách nghệ thuật cụ thể (như của các họa sĩ nổi tiếng) và áp dụng chúng lên các hình ảnh mới.

+ **Style Transfer:** Sử dụng VGG16 để chuyển phong cách của một bức tranh lên một bức ảnh khác, ví dụ như chuyển phong cách tranh của Van Gogh lên ảnh chụp của bạn.

+ **Tạo ảnh nghệ thuật:** Chuyển các bức ảnh thành các tác phẩm nghệ thuật với những đặc trưng đặc trưng từ các nghệ sĩ nổi tiếng.

5. Tìm kiếm hình ảnh (Image Retrieval)

VGG16 có thể được sử dụng trong các hệ thống tìm kiếm hình ảnh để trích xuất đặc trưng của các bức ảnh và so sánh chúng với những bức ảnh khác trong cơ sở dữ liệu.

+ **Tìm kiếm hình ảnh theo nội dung (Content-Based Image Retrieval - CBIR):** Tìm kiếm hình ảnh trong cơ sở dữ liệu dựa trên các đặc trưng hình ảnh (ví dụ: màu sắc, hình dạng, kết cấu) thay vì chỉ dựa trên từ khóa.

6. Đảm bảo chất lượng sản phẩm trong công nghiệp (Quality Control)

Trong các ngành công nghiệp sản xuất, VGG16 có thể được áp dụng để phát hiện các lỗi trong sản phẩm qua hình ảnh, chẳng hạn như các vết nứt, trầy xước, hoặc các khuyết tật khác.

+ **Phát hiện lỗi sản phẩm:** Đánh giá hình ảnh của các sản phẩm trong dây chuyền sản xuất để phát hiện lỗi mà không cần sự can thiệp của con người.

1.3.2 Phân Nhóm đã làm:

1. Phân loại hình ảnh (Image Classification)

VGG16 đã được huấn luyện trên bộ dữ liệu ImageNet, một trong những bộ dữ liệu phân loại hình ảnh lớn nhất, với hàng nghìn lớp đối tượng. Mô hình này có thể phân loại các hình ảnh vào các nhóm đối tượng khác nhau, chẳng hạn như nhận diện động vật, phương tiện, đồ vật, vv. Ví dụ:

+ **Phân loại các loài động vật:** Phân loại các loài động vật khác nhau như chó, mèo, chim, ngựa, vv.

+ **Phân loại các đối tượng công nghiệp:** Phân loại các sản phẩm trong các ngành công nghiệp, chẳng hạn như phân biệt các loại linh kiện điện tử hoặc ô tô.

2. Nhận diện đối tượng (Object Detection)

Mặc dù VGG16 chủ yếu được sử dụng cho phân loại, nhưng nó cũng có thể được sử dụng trong các mô hình nhận diện đối tượng. Các mô hình như **Faster R-CNN** hoặc **YOLO (You**

Only Look Once) có thể sử dụng các mạng CNN như VGG16 để trích xuất đặc trưng từ ảnh và sau đó xác định vị trí và loại đối tượng trong ảnh.

+ **Ứng dụng nhận diện đối tượng:** Trong các hệ thống giám sát video, nhận diện đối tượng có thể giúp xác định các vật thể cụ thể (như người, xe cộ, hành lý, vv) trong cảnh quay.

3. Tự động chú thích hình ảnh (Image Captioning)

VGG16 có thể được sử dụng như một phần trong hệ thống tạo chú thích tự động cho hình ảnh. Các mô hình như **Show and Tell** kết hợp VGG16 (hoặc các mô hình tương tự) với các mạng RNN (Recurrent Neural Networks) để mô tả nội dung của hình ảnh bằng câu văn tự nhiên.

+ **Ứng dụng cho người khiếm thị:** Tạo các chú thích mô tả hình ảnh để hỗ trợ người khiếm thị trong việc nhận diện và hiểu các bức ảnh.

+ **Tự động chú thích hình ảnh trên mạng xã hội:** Các ứng dụng trên mạng xã hội hoặc các nền tảng chia sẻ hình ảnh có thể sử dụng VGG16 để tạo ra các mô tả cho ảnh.

1.4 Cấu trúc báo cáo

- Chương 1: Giới thiệu : Giới thiệu tổng quan về VGG16,
- Chương 2: Cơ sở lý thuyết : Trình bày về các khái niệm lý thuyết liên quan đến cấu trúc và nguyên lý hoạt động
- Chương 3: Phương Pháp : Mô tả các phương pháp được sử dụng để triển khai VGG16
- Chương 4: Hiện thực kết quả : Trình bày các kết quả đạt được từ việc triển khai VGG16
- Chương 5: Kết luận : Tóm tắt các phát hiện chính từ nghiên cứu về VGG16,

Chương 2: Cơ sở lý thuyết

2.1 Cấu trúc của VGG16

- VGG16 bao gồm 16 lớp học học (layers), trong đó có 13 lớp tích chập (convolutional layers) và 3 lớp fully connected (FC) ở cuối. Các lớp trong mạng VGG16 được tổ chức theo nguyên lý rất đơn giản nhưng hiệu quả: sử dụng các lớp tích chập với kích thước bộ lọc 3x3 và sử dụng bước nhảy (stride) bằng 1, kết hợp với các lớp max-pooling 2x2 với bước nhảy 2.
- Cấu trúc chi tiết của VGG16 như sau:
 1. Các lớp Convolutional:
 - VGG16 sử dụng 13 lớp tích chập, với các bộ lọc có kích thước 3x3, bước nhảy là 1.
 - Các lớp tích chập này được tổ chức thành các nhóm (blocks), mỗi nhóm có 2 hoặc 3 lớp liên tiếp.
 - Các lớp Max Pooling:
 - Sau mỗi nhóm lớp tích chập, mạng sử dụng một lớp max-pooling với kích thước 2x2 và bước nhảy 2 để giảm kích thước không gian (spatial dimensions) của dữ liệu.
 2. Các lớp Fully Connected (FC):
 - Cuối cùng, các đặc trưng đã được trích xuất qua các lớp tích chập và pooling được "phẳng" (flatten) lại và kết nối với 3 lớp fully connected, trong đó lớp cuối cùng là lớp phân loại (classification layer).
 - Lớp fully connected đầu tiên có 4096 neuron, lớp thứ hai cũng có 4096 neuron và lớp thứ ba có 1000 neuron (tương ứng với số lượng lớp trong bộ dữ liệu ImageNet).
 3. Lớp Softmax:
 - Lớp cuối cùng của VGG16 là một lớp softmax, dùng để phân loại dữ liệu đầu vào vào một trong các lớp (class) xác định

2.2 Đặc điểm nổi bật của VGG16

- Kiến trúc đơn giản nhưng mạnh mẽ: VGG16 sử dụng các bộ lọc có kích thước nhỏ 3x3 thay vì những bộ lọc lớn hơn (như 5x5 hay 7x7), điều này giúp giảm số lượng tham số trong mạng và tăng khả năng tổng quát (generalization).
- Sử dụng các lớp Convolutional có bước nhảy (stride) bằng 1: Điều này giúp duy trì kích thước không gian (spatial size) của dữ liệu trong suốt quá trình lan truyền qua mạng.
- Sự tập trung vào việc tăng độ sâu: VGG16 là một trong những mạng có độ sâu rất lớn trong thời kỳ của nó, với 16 lớp, giúp mạng có khả năng học các đặc trưng phức tạp từ dữ liệu.

2.3 Nguyên tắc hoạt động của thuật toán VGG16

Convolution

Là một phép toán toán học giữa một bộ lọc (kernel) và dữ liệu đầu vào (thường là hình ảnh), nhằm trích xuất các đặc trưng (features) từ hình ảnh đó.

Kernel (hoặc filter) là một ma trận nhỏ (thường 3x3) được sử dụng để quét qua hình ảnh. Mỗi kernel sẽ học được một đặc trưng cụ thể. Khi kernel di chuyển (hoặc "trượt") qua hình ảnh, nó sẽ thực hiện phép toán nhân ma trận với các pixel trong khu vực tương ứng, tạo ra một giá trị mới cho mỗi vị trí.

Max Pooling

Max pooling hoạt động bằng cách chia đầu vào thành các vùng nhỏ hơn (thường là hình vuông) và chọn giá trị lớn nhất trong mỗi vùng. Trong vgg16, một kernel (hoặc filter) với kích thước 2x2, sẽ trượt qua đầu vào và tại mỗi vị trí, nó sẽ ghi lại giá trị lớn nhất trong vùng tương ứng. Điều này giúp giảm số lượng kích thước hình ảnh đi 1 nửa, từ đó giảm độ phức tạp tính toán và số lượng tham số của mô hình.

Fully Connected

Lớp Dense kết nối tất cả các neuron của lớp trước đó với từng neuron trong lớp Dense. Điều này có nghĩa là mỗi neuron trong lớp Dense nhận đầu vào từ tất cả các neuron của lớp trước, cho phép mạng học được các mối quan hệ phức tạp giữa các đặc trưng

Mỗi neuron trong lớp Dense tính toán đầu ra bằng cách áp dụng một phép toán trọng số lên đầu vào. Cụ thể, đầu vào sẽ được nhân với một ma trận trọng số, cộng với một bias (hệ số điều chỉnh), và sau đó được đưa vào một hàm kích hoạt (thường là ReLU hoặc softmax cho phân loại).



Sơ đồ hoạt động của thuật toán VGG16

1. Input Layer (Lớp đầu vào):

- Hình ảnh đầu vào có kích thước $224 \times 224 \times 3$ (224 pixel chiều rộng, 224 pixel chiều cao, và 3 kênh màu RGB).

2. Convolutional Layers (Lớp tích chập):

- Các lớp tích chập sử dụng các bộ lọc 3×3 , áp dụng nhiều lần để trích xuất đặc trưng từ hình ảnh.
- Sau mỗi lớp tích chập, kích thước không gian của dữ liệu giảm dần, nhưng số lượng kênh (channels) tăng lên. Ví dụ:
- Từ $224 \times 224 \times 3$ (hình ảnh đầu vào) đến $224 \times 224 \times 64$.

- Sau đó giảm kích thước không gian và tăng số lượng kênh từ $112 \times 112 \times 128$ đến $14 \times 14 \times 512$.
 - Sau mỗi lớp tích chập là hàm kích hoạt ReLU để tăng tính phi tuyến tính.
3. Max Pooling Layers (Lớp gộp tối đa):
- Lớp max pooling (màu đỏ) có kích thước 2×2 giúp giảm kích thước không gian (width và height) xuống một nửa, trong khi vẫn giữ nguyên số kênh.
 - Max pooling giúp giảm độ phức tạp tính toán, đồng thời tránh overfitting bằng cách lấy đặc trưng quan trọng nhất trong mỗi vùng nhỏ.
4. Fully Connected Layers (Lớp kết nối đầy đủ):
- Sau khi các đặc trưng đã được trích xuất qua các lớp tích chập và max pooling, dữ liệu đi qua các lớp fully connected.
 - Có 2 lớp fully connected đầu tiên với 4096 nơ-ron mỗi lớp và một lớp cuối cùng để tổng hợp các đặc trưng cho việc phân loại.
5. Softmax Layer (Lớp Softmax):
- Lớp softmax cuối cùng sẽ tính xác suất cho mỗi lớp đầu ra (class), chọn lớp có xác suất cao nhất làm dự đoán cuối cùng của mô hình.

Chương 3: Phương Pháp

3.1 Yêu cầu bài toán

Input

1. Huấn luyện

- Để mô hình VGG16 học tốt, bộ dữ liệu hình ảnh cần có số lượng hình ảnh lớn và đa dạng cho mỗi loại (chó và mèo). Ở đây nhóm sử dụng một dataset phổ biến Cats vs Dogs có khoảng 25.000 hình ảnh, chia đều cho mỗi lớp (chó và mèo).
- Bộ hình ảnh đa dạng về góc chụp (chụp từ nhiều góc khác nhau, với nhiều phong nền và điều kiện ánh sáng). Ngoài ra, nên có nhiều giống chó và mèo khác nhau để mô hình có thể phân biệt chính xác dù hình ảnh khác nhau.
- Hình ảnh nên có độ phân giải và chất lượng đủ tốt, giúp mô hình nhận diện đặc trưng của mỗi loài dễ dàng hơn. Tuy nhiên, trước khi đưa vào mô hình, các hình ảnh cần được resize về kích thước 224x224, là yêu cầu của VGG16.
- Hình ảnh cần được dán nhãn rõ ràng và chính xác, với mỗi hình ảnh có nhãn tương ứng là "chó" hoặc "mèo". Việc này giúp mô hình phân loại biết được đâu là hình ảnh của chó và đâu là hình ảnh của mèo trong quá trình huấn luyện.

2. Dự đoán

- Kích thước hình ảnh: Hình ảnh đầu vào cần được resize về kích thước 224x224 pixels. Đây là kích thước cố định mà VGG16 yêu cầu để có thể xử lý đầu vào qua các lớp convolution.
- Số lượng kênh màu: VGG16 yêu cầu hình ảnh có 3 kênh màu (RGB). Điều này có nghĩa là hình ảnh phải là hình màu, không phải ảnh đen trắng (nếu ảnh đen trắng, cần chuyển sang RGB).
- Chuẩn hóa giá trị pixel: Thông thường, giá trị pixel của ảnh đầu vào sẽ được chuẩn hóa (normalize) về khoảng phù hợp, chẳng hạn chia giá trị pixel cho 255 để đưa vào khoảng $[0, 1]$, hoặc trừ đi trung bình của tập huấn luyện ImageNet (mà VGG16 ban đầu được huấn luyện) để đưa về giá trị xung quanh $[-1, 1]$.
- Bố cục hình ảnh: Hình ảnh đầu vào cần được chuyển đổi thành dạng tensor với cấu trúc (batch_size, 224, 224, 3) trong TensorFlow hoặc (batch_size, 3, 224, 224) trong


PyTorch, với `batch_size` là số lượng ảnh trong một batch (thường có thể là 1 nếu bạn dự đoán một ảnh).

Output

- Sau khi huấn luyện mô hình sẽ lưu lại mô hình đã huấn luyện dưới dạng một file để sử dụng lại sau mà không cần huấn luyện lại. File này thường có định dạng HDF5 (.h5)
- Xác suất dự đoán cho mỗi lớp: Khi đầu vào là một ảnh, VGG16 sẽ tính toán xác suất cho từng lớp mà ảnh đó có thể thuộc về (ví dụ: "chó" hoặc "mèo"). Kết quả này có dạng một mảng chứa các xác suất. Từ các xác suất trên, lớp có xác suất cao nhất sẽ được chọn làm kết quả dự đoán cuối cùng.
- Khi bạn áp dụng mô hình trên tập kiểm thử, output còn bao gồm các chỉ số đánh giá hiệu suất như độ chính xác (accuracy), độ nhạy (recall), độ đặc hiệu (specificity), và F1-score. Những chỉ số này giúp đánh giá mức độ chính xác của mô hình khi phân loại đúng giữa chó và mèo.

3.2 Giải thuật thực hiện

- Mô hình được khởi tạo với lớp Sequential cho phép thêm các lớp theo thứ tự.

```
 model = Sequential()
```

- Thêm lớp Conv2D đầu tiên với 64 bộ lọc, sử dụng kích thước kernel 3x3 và padding "same". Lớp này nhận đầu vào là hình ảnh có kích thước 224x224 pixel với 3 kênh màu RGB. Hàm kích hoạt ReLU được sử dụng để tăng cường khả năng phi tuyến của mô hình.

```
model.add(Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu"))
```

- Tiếp theo, một lớp Conv2D thứ hai cũng với 64 bộ lọc được thêm vào, tiếp tục trích xuất các đặc trưng từ đầu ra của lớp trước mà không cần chỉ định lại kích thước đầu vào.

```
model.add(Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu"))
```

- Sau khi hai lớp convolutional, một lớp MaxPooling2D được thêm vào để giảm kích thước của đầu ra, giúp giảm thiểu số lượng tham số và tính toán trong mô hình. Lớp

pooling này sử dụng kích thước vùng 2x2 và bước di chuyển 2, giúp giảm kích thước đầu ra một nửa.

```
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

- Quá trình này tiếp tục với việc thêm hai lớp Conv2D tiếp theo, nhưng lần này tăng số lượng bộ lọc lên 128. Cấu trúc cũng giống như trước, với hai lớp convolutional tiếp theo được theo sau bởi một lớp MaxPooling2D để tiếp tục giảm kích thước đầu ra.

```
[ ] model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

- Khi mô hình tiếp tục phát triển, số lượng bộ lọc được tăng lên 256 cho ba lớp Conv2D tiếp theo, tiếp tục theo sau bởi một lớp MaxPooling2D. Sự gia tăng số lượng bộ lọc này giúp mô hình học được các đặc trưng phức tạp hơn từ hình ảnh.

```
[ ] model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

- Sau đó, ba lớp Conv2D với 512 bộ lọc được thêm vào, mỗi lớp cũng được theo sau bởi một lớp MaxPooling2D. Việc sử dụng nhiều lớp convolutional với số lượng bộ lọc lớn hơn cho phép mô hình trích xuất được các đặc trưng chi tiết và sâu sắc hơn từ dữ liệu đầu vào.

```
[ ] model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

- Sau đó, ba lớp Conv2D với 512 bộ lọc được thêm vào, mỗi lớp cũng được theo sau bởi một lớp MaxPooling2D

```
[ ] model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
```

```
[ ] model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

- Cuối cùng, sau khi trải qua nhiều lớp convolutional và pooling, dữ liệu đầu ra được chuyển đổi từ định dạng 2D sang 1D thông qua lớp Flatten. Điều này rất quan trọng trước khi thêm các lớp Dense, nơi mô hình sẽ thực hiện phân loại.

```
[ ] model.add(Flatten())
```

- Hai lớp Dense với 4096 đơn vị được thêm vào, sử dụng hàm kích hoạt ReLU để xử lý các đặc trưng đã được trích xuất.

```
[ ] model.add(Dense(units=4096,activation="relu"))
```

```
[ ] model.add(Dense(units=4096,activation="relu"))
```

- Lớp cuối cùng là một lớp Dense với 2 đơn vị và hàm kích hoạt softmax, cho phép mô hình phân loại hình ảnh thành hai lớp, như mèo và chó.

```
[ ] model.add(Dense(units=2, activation="softmax"))
```

- Cấu hình mô hình trước khi huấn luyện

```
from keras.optimizers import Adam
# use learning_rate instead of lr
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

- Dùng hai callback của Keras để tối ưu hóa quá trình huấn luyện mô hình VGG16 bằng cách lưu trữ các checkpoint và dừng sớm nếu mô hình không cải thiện thêm.

```
[ ] from keras.callbacks import ModelCheckpoint, EarlyStopping
# use save_freq instead of period and add the .keras extension to the filepath
checkpoint = ModelCheckpoint("cat_dog_fine_tune_model.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_only=False, mode='auto', save_freq=1)
# Set mode to 'max' to tell EarlyStopping to maximize val_acc
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='max')
```

- Sử dụng hàm fit để huấn luyện mô hình model với tập dữ liệu traindata

```
hist = model.fit(traindata,
                  steps_per_epoch=100,
                  validation_data= testdata,
                  validation_steps=10,
                  epochs=10,
                  callbacks=[checkpoint,early])
```

- Sử dụng mô hình vừa huấn luyện để dự đoán hình ảnh

```
from google.colab.patches import cv2_imshow
import numpy as np
import cv2

from keras.preprocessing import image
path = "/content/drive/MyDrive/Colab Notebooks/cats_and_dogs_filtered/validation/dogs/dog.2022.jpg"
img = image.load_img(path,target_size=(224,224))
img = np.asarray(img)
img = np.expand_dims(img, axis=0)
from keras.models import load_model
saved_model = load_model("/content/drive/MyDrive/Colab Notebooks/cat_dog_fine_tune_model.h5")
img1 = cv2.imread(path)
output = saved_model.predict(img)
cv2_imshow(img1)
if output[0][0] > output[0][1]:
    print("cat")
else:
    print('dog')
```

2.4. Phương pháp đánh giá:

1. Độ chính xác (Accuracy):

- Accuracy trên tập huấn luyện và Accuracy trên tập kiểm tra là chỉ số cơ bản để đánh giá hiệu suất của mô hình. Độ chính xác cao trên cả hai tập dữ liệu cho thấy mô hình có thể phân loại chính xác hình ảnh. Trong các tác vụ phân loại, đây thường là chỉ số đầu tiên để đánh giá xem mô hình có đạt được mục tiêu hay không.

2. Hàm mất mát (Loss):

- Loss trên tập huấn luyện và tập kiểm tra cho thấy mức độ sai lệch giữa dự đoán của mô hình và nhãn thực tế. Loss thấp hơn cho thấy mô hình đang học được và đưa ra dự đoán gần đúng hơn. Theo dõi cả loss và accuracy giúp xác định mô hình có đang bị overfitting (khi độ chính xác trên tập huấn luyện cao nhưng trên tập kiểm tra thấp) hoặc underfitting (khi cả hai độ chính xác đều thấp).

Chương 4: Hiện thực kết quả

4.1 Yêu cầu hệ thống và tập dữ liệu thực nghiệm.

4.1.1 Yêu cầu hệ thống

- Phần cứng: Máy tính
- Phần mềm: Sử dụng GoogleColab
- Thư viện:
 - ❖ Keras
 - ❖ os
 - ❖ Numpy
 - ❖ Cv2
 - ❖ Matplotlib

4.1.2 Tập dữ liệu thực nghiệm

Tập dữ liệu Dogs and Cats được lấy từ kaggle

4.2 Kết quả thực nghiệm

Dự đoán được hình ảnh là chó hoặc mèo

1/1 ————— 1s 711ms/step

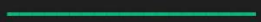


dog

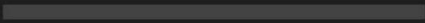
```
loss, accuracy = saved_model.evaluate(testdata)
print('Độ chính xác:', accuracy)
```

Python

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning
self._warn_if_super_not_called()

64/64  658s 10s/step - accuracy: 0.8720 - loss: 8.2912

Độ chính xác: 0.8633663654327393



Chương 5: Kết luận

5.1 Kết quả đạt được

- Đã tìm hiểu và áp dụng được thuật toán vào bài toán phân loại hình ảnh

5.2 Ưu và nhược điểm của thuật toán VGG16

5.2.2 Ưu điểm:

- Cấu trúc đơn giản dễ hiểu
- Hiệu suất cao
- Khả năng phân loại chính xác

5.2.3 Nhược điểm

- Kích thước mô hình lớn
- Tốc độ chậm trong huấn luyện
- Yêu cầu tài nguyên tính toán cao
- Không tối ưu cho thiết bị di động và đối tượng

5.3 Hướng mở rộng tương lai

Tăng cường dữ liệu (Data Augmentation):

- Áp dụng kỹ thuật như xoay, lật, phóng to/thu nhỏ ảnh để tăng kích thước tập dữ liệu.

Fine-tuning VGG16:

- Sử dụng mô hình VGG16 pre-trained trên ImageNet để cải thiện độ chính xác.
- Tăng số lớp hoặc thử nghiệm mạng khác:
- Thử các kiến trúc CNN hiện đại hơn như ResNet, EfficientNet.

Đánh giá trên bộ dữ liệu lớn hơn:

- Sử dụng tập dữ liệu có nhiều class hơn (không chỉ cats và dogs).

Triển khai mô hình:

- Đưa mô hình vào ứng dụng thực tế để phân loại ảnh tải lên từ người dùng.

TÀI LIỆU THAM KHẢO

1. <https://viso-ai.translate.goog/deep-learning/vgg-very-deep-convolutional-networks>
2. https://d2l.aivivn.com/chapter_convolutional-modern
3. <https://www-digitalocean-com.translate.goog/community/tutorials/popular-deep-learning-architectures-alexnet-vgg-googlenet>

PHỤ LỤC

Source code sử dụng:

- Thư viện

```
import keras, os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
```

- Kết nối Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

- Khởi tạo bộ dữ liệu huấn luyện và kiểm tra

```
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="/content/drive/MyDrive/Colab Notebooks/cats_and_dogs_filtered/train", target_size=(224,224))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(directory="/content/drive/MyDrive/Colab Notebooks/cats_and_dogs_filtered/validation", target_size=(224,224))
```

- Xây dựng mô hình CNN

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3), filters=64, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=4096, activation="relu"))
model.add(Dense(units=4096, activation="relu"))
model.add(Dense(units=2, activation="softmax"))
```

- Compile mô hình

```
from keras.optimizers import Adam
# use learning_rate instead of lr
opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
model.summary()
```

- Cài đặt lưu trữ mô hình và huấn luyện mô hình

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
# use save_freq instead of period and add the .keras extension to the filepath
checkpoint = ModelCheckpoint("vgg16_1.keras", monitor='val_acc', verbose=1, save_best_only=True, save_weights_only=False, mode='auto', save_freq=1)
# Set mode to 'max' to tell EarlyStopping to maximize val_acc
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1, mode='max')
hist = model.fit(traindata,
                  steps_per_epoch=100,
                  validation_data= testdata,
                  validation_steps=10,
                  epochs=10,
                  callbacks=[checkpoint,early])
```

- Dự đoán và hiển thị kết quả

```
from google.colab.patches import cv2_imshow
import numpy as np
import cv2

from keras.preprocessing import image
path = "/content/drive/MyDrive/Colab Notebooks/cats_and_dogs_filtered/validation/dogs/dog.2022.jpg"
img = image.load_img(path,target_size=(224,224))
img = np.asarray(img)
img = np.expand_dims(img, axis=0)
from keras.models import load_model
saved_model = load_model("/content/drive/MyDrive/Colab Notebooks/vgg16_1.keras")
img1 = cv2.imread(path)
output = saved_model.predict(img)
cv2_imshow(img1)
print(output)
if output[0][0] > output[0][1]:
    print("cat")
else:
    print('dog')
```

- Đánh giá mô hình

```
loss, accuracy = saved_model.evaluate(testdata)
```

```
print('Độ chính xác:', accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset
```

```
self._warn_if_super_not_called()
```

64/64 ————— 658s 10s/step - accuracy: 0.8720 - loss: 8.2912

Độ chính xác: 0.8633663654327393