Bộ giáo dục và đào tạo

Trường Đại học Ngoại ngữ - Tin học TP.HCM

Đề tài:

Phân vùng ảnh dựa trên mạng nơron nhân tạo (Artificial Neural Network Based Segmentation)

GVHD: PGS. TS Nguyễn Thanh Bình

SVTH: 22DH113672 - Nguyễn Đình Thượng
22DH114572 – Bành Vĩnh Khang

Tháng 10 Năm 2024

Mục lục

Chương 1: Giới thiệu đề tài

1.1 Giới thiệu

Thị giác máy tính là một lĩnh vực quan trọng trong trí tuệ nhân tạo, chuyên nghiên cứu và phát triển các phương pháp giúp máy tính có thể hiểu và xử lý hình ảnh giống như con người. Trong bối cảnh này, phân vùng ảnh dựa trên trí tuệ nhân tạo (AI) trở thành một công cụ mạnh mẽ, giúp máy tính phân tích và phân loại các vùng trong ảnh dựa trên các đặc trưng học được từ dữ liệu.

Đề tài này tập trung vào việc ứng dụng trí tuệ nhân tạo trong phân vùng ảnh, sử dụng các thuật toán học máy và học sâu để tự động phân chia ảnh thành các vùng có đặc điểm tương đồng. Kỹ thuật phân vùng ảnh dựa trên AI không chỉ giúp xác định các vùng có tính chất giống nhau trong ảnh mà còn cung cấp những thông tin quan trọng về cấu trúc và phân bố của chúng trong không gian hình ảnh. Điều này làm cho phương pháp trở thành một công cụ quan trọng trong nhiều ứng dụng thực tế, từ công nghiệp, y tế đến nghiên cứu khoa học.

Mục tiêu của đề tài không chỉ là khám phá lý thuyết và các phương pháp hiện đại trong phân vùng ảnh dựa trên trí tuệ nhân tạo mà còn áp dụng chúng vào thực tế.

Chúng em mong muốn tìm ra các giải pháp hiệu quả, đồng thời cung cấp kiến thức thực tế về việc ứng dụng AI trong các lĩnh vực như phân tích hình ảnh y học, nhận diện đối tượng trong công nghiệp, và kiểm soát chất lượng sản phẩm. Qua đề tài này, chúng em hy vọng sẽ chia sẻ những khám phá, thử nghiệm và tiến bộ trong việc ứng dụng trí tuệ nhân tạo vào thị giác máy tính, góp phần thúc đẩy sự phát triển không ngừng của lĩnh vực này.

1.2 Mục tiêu và nội dung đề tài

1.2.1 Mục tiêu

Mục tiêu chính của đề tài là tìm hiểu và ứng dụng kỹ thuật phân vùng ảnh dựa trên mạng nơ-ron nhân tạo (Artificial Neural Network - ANN) vào việc phân tích và phân loại các vùng trong ảnh. Đề tài tập trung vào các mục tiêu cụ thể như sau:

• Nghiên cứu: Khám phá các phương pháp và công nghệ phân vùng ảnh dựa trên mạng nơ-ron nhân tạo, với trọng tâm là lý thuyết cơ bản và các thuật toán quan trọng trong việc ứng dụng mạng nơ-ron vào phân vùng ảnh. Đề tài sẽ nghiên cứu về kiến trúc U_net và mạng nơ-ron sâu (Deep Learning), cũng như cách thức học và tối ưu hóa trong bối cảnh phân vùng ảnh.

• Ứng dụng: Triển khai phương pháp phân vùng ảnh dựa trên mạng nơ-ron nhân tạo vào xử lý ảnh, nhằm phân chia các vùng có đặc điểm tương đồng. Đề tài sẽ tập trung vào việc lựa chọn và tinh chỉnh các tham số trong mô hình nơ-ron nhân tạo, tối ưu hóa thuật toán để đạt hiệu quả cao trong các ứng dụng phân vùng ảnh phức tạp.

• Thử nghiệm và Đánh giá: Thực hiện thử nghiệm để đánh giá hiệu suất của phương pháp phân vùng dựa trên mạng nơ-ron nhân tạo trong các tình huống ứng dụng cụ thể. Đề tài sẽ phân tích độ chính xác và khả năng tổng quát hóa của mô hình trong các bối cảnh và dữ liệu khác nhau, từ đó đưa ra các giải pháp cải tiến hiệu quả hơn cho các bài toán phân vùng ảnh thực tế.

1.2.2 Giới hạn của đề tài

Đề tài chỉ dùng để phân vùng ảnh chó mèo với các mặt nạ nhị phân cho mỗi loài. Đề tài chỉ sử dụng kiến trúc U_net tiêu chuẩn, chưa có mở rộng và thay đổi kiến trúc. Chưa tối ưu được thời gian huấn luyện mô hình. Đề tài chỉ sử dụng các kiến thức cơ bản như resize, augmentation, và normalize để xử lý dữ liệu

1.2.2 Nội Dung

Nội dung của đề tài gồm các bước sau:

+ Thu thập dữ liệu là hình ảnh chó, mèo
+ Trực quan hóa dữ liệu bằng biểu đồ
+ Hiển thị thông tin bộ huấn luyện và test
+ Tăng cường dữ liệu
+ Chuẩn hóa dữ liệu hình ảnh
+ Xây dựng kiến trúc U_net
+ Hiển thị kết quả phân vùng

1.3 Cấu trúc báo cáo

Chương 1:

- Trình bày phần giới thiệu đề tài. Đưa ra mục tiêu và nội dung của đề tài

Chương 2:

- Trình bày mô hình U_net được dử dụng trong đề tài

Chương 3:

- Trình bày về yêu cầu bài toán, nội dung giải thuật đã sử dụng trong đề tài
- Từ đó đưa ra các mô hình đánh giá

Chương 4:

- Trình bày về yêu cầu của hệ thống và tập dữ liệu đã thực nghiệm và kết quả thực nghiệm trong đề tài

Chương 5:

- Trình bày về các kết quả đạt được, ưu - nhược điểm của mô hình và hướng mở rộng tương lai

Chương 2: Cơ sở lý thuyết

2.1 Kiến trúc U_Net

2.1.1 Tóm tắt về kiến trúc

U_Net là mạng nơ ron nhân tạo được thiết kế cho các bài toán phân đoạn ảnh. U_net được ứng dụng hiệu quả trong lĩnh vực y tế

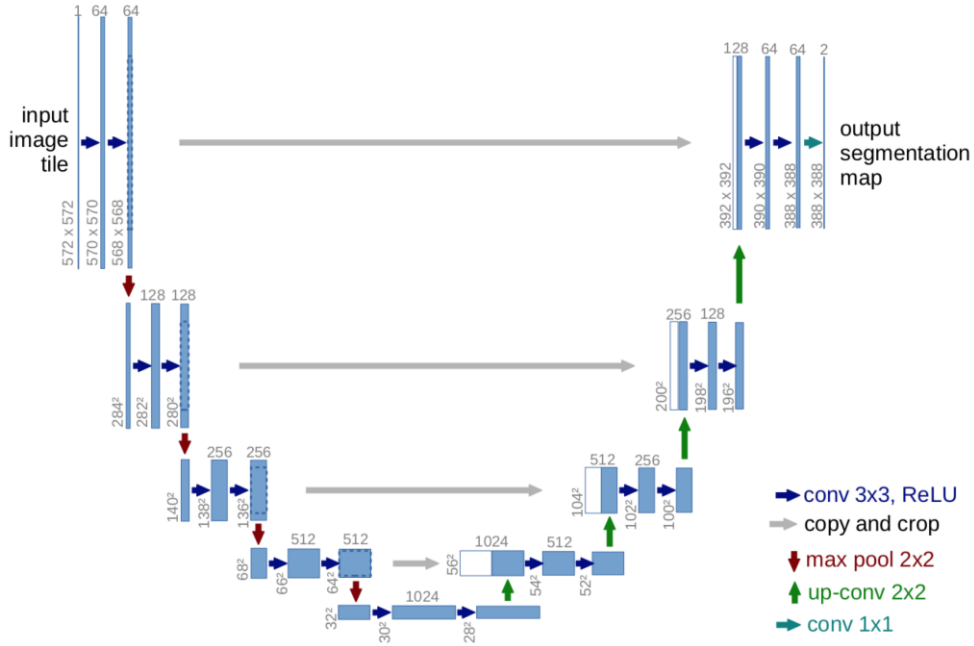
Cấu trúc của U_net:

Kiến trúc đối xứng: U-Net có hình dạng giống như hình chữ U, gồm hai phần chính là encoder và decoder

Phần thu hẹp: Có các lớp tích chập và các lớp giảm kích thước, giúp trích xuất các đặc trưng từ ảnh đầu vào

Phần mở rộng: Sử dụng các lớp giải nén để tái tạo lại ảnh kích thước, kết hợp với các thông tin từ các lớp phần thu hẹp thông qua các kết nối nhảy

Kiến trúc U-Net được thể hiện trong hình sau:



Hình 1. Kiến trúc của mạng U-Net

2.1.2 Các bước thực hiện của U_net

Bước 1: Encoder (Mã hóa)

Mục đích: Trích xuất đặc trưng từ ảnh đầu vào.

- Hình ảnh đầu vào sẽ đi qua các lớp tích chập (Convolution) để trích gọn các đặc trưng quan trọng, giúp mô hình hiểu rõ hơn về cấu trúc của ảnh.

- Tiếp theo, ảnh sẽ qua lớp Max Pooling để giảm kích thước không gian, cô đọng thông tin và giảm số lượng tham số, sau đó tiếp tục qua các lớp tích chập để trích xuất thêm đặc trưng.

Bước 2: Bottleneck (Phần kết nối giữa Encoder và Decoder)

- **Mục đích**: Tạo ra các đặc trưng sâu hơn cho ảnh.
- Ảnh sau khi qua Encoder sẽ đi qua phần kết nối (Bottleneck), giúp tạo ra các đặc trưng phức tạp nhất, tuy nhiên kích thước của ảnh ở phần này sẽ nhỏ nhất.
- Phần này đóng vai trò là vùng tập trung đặc trưng trước khi ảnh được mở rộng lại trong quá trình khôi phục.

Bước 3: Decoder (Giải mã)

- **Mục đích**: Khôi phục kích thước ảnh ban đầu và tạo ra mặt nạ phân đoạn.
- Sau khi đi qua Encoder và phần Bottleneck, ảnh sẽ đi qua phần Decoder để mở rộng lại kích thước không gian của ảnh về dạng ban đầu.
- Phần Decoder sử dụng Upsampling (tăng kích thước không gian) qua các lớp deconvolution hoặc transposed convolution.
- Đồng thời, các **Skip Connections** được sử dụng để kết nối với các tầng tương ứng trong Encoder nhằm kết hợp các đặc trưng chi tiết đã được trích xuất ở giai đoạn mã hóa, giúp phục hồi thông tin chi tiết và cải thiện độ chính xác phân đoạn.

Bước 4: Tổng hợp kết quả phân đoạn

- **Lớp Convolution 1x1**: Cuối cùng, lớp tích chập 1x1 được áp dụng để giảm độ sâu của ảnh và phân loại mỗi pixel vào một lớp nhất định, tạo ra mặt nạ phân đoạn chính xác với kích thước tương ứng với ảnh đầu vào.
- **Đầu ra**: Kết quả cuối cùng là một ảnh phân đoạn, trong đó mỗi vùng được gán nhãn với một lớp cụ thể

Chương 3: Phương pháp

3.1 Yêu cầu bài toán

Yêu cầu của bài toán là huấn luyện một mạng nơ-ron nhân tạo để phân vùng ảnh chó mèo, giúp mô hình có thể nhận diện và phân tách các vùng chứa chó và mèo trong hình ảnh. Để giải quyết bài toán này, chúng em sử dụng kiến trúc U-Net, một mô hình mạng nơ-ron sâu đặc biệt cho phân vùng ảnh.

U-Net là một mạng nơ-ron có kiến trúc encoder-decoder, được thiết kế để phân tích và tái tạo lại các đối tượng trong ảnh một cách chi tiết. Mô hình này bao gồm một phần encoder và một phần decoder, kết hợp với các skip connections giữa các lớp encoder và decoder. Các kết nối này giúp bảo tồn thông tin chi tiết từ các lớp đầu vào, giúp mạng phân biệt chính xác các vùng trong ảnh, đặc biệt là các vùng mờ hoặc nhỏ.

Đối với bài toán phân vùng ảnh chó mèo, mô hình U-Net sẽ học cách phân biệt giữa hai đối tượng chính là chó và mèo, từ đó tạo ra một bản đồ phân vùng, với mỗi pixel trong ảnh được gán nhãn là "chó", "mèo" hoặc "background". Quá trình huấn luyện sẽ sử dụng tập dữ liệu có nhãn, giúp mạng học cách nhận diện các đặc trưng của chó và mèo, từ đó áp dụng vào ảnh mới để phân vùng chính xác.

Với kiến trúc U-Net, việc phân vùng ảnh chó mèo sẽ được thực hiện hiệu quả

3.1.1 Đầu vào , đầu ra

Đầu vào của mô hình bao gồm hình ảnh có kích thước chiều cao, chiều rộng và kênh màu, cùng với mặt nạ phân vùng (input_mask) có kích thước chiều cao và chiều rộng. Cả hai đều được chuẩn hóa, với giá trị pixel của hình ảnh trong khoảng [0, 1] và mặt nạ có giá trị nhãn bắt đầu từ 0. Đầu ra của mô hình là hình ảnh đã được phân vùng

Đầu ra của mô hình là các dự đoán từ mạng, bao gồm giá trị phân vùng cho từng pixel trong ảnh, thường dưới dạng một mặt nạ phân vùng (output_mask) có kích thước tương tự như mặt nạ đầu vào, với các giá trị đại diện cho các lớp phân vùng dự đoán

3.2 Giải thuật thực hiện

3.2.1 Giải thuật U_net

Bộ mã hóa (Encoder):

Convolutional Layers: Mỗi khối trong bộ mã hóa thường bao gồm hai hoặc ba lớp tích chập liên tiếp. Mỗi lớp áp dụng một bộ lọc để trích xuất các đặc trưng từ hình ảnh.

Activation Function: Thường sử dụng hàm kích hoạt ReLU (Rectified Linear Unit) sau mỗi lớp tích chập để tăng tính phi tuyến.

Max Pooling: Sau mỗi khối tích chập, một lớp max pooling được áp dụng để giảm kích thước không gian (spatial dimensions) của hình ảnh, giúp giảm số lượng tham số và tính toán.

Bottleneck:

Đây là tầng giữa, nơi mà kích thước hình ảnh đạt tối thiểu. Tại đây, U-Net thực hiện các phép tích chập sâu hơn để khai thác các đặc trưng cao cấp của hình ảnh.

Bộ giải mã (Decoder):

Up-sampling: Mỗi khối trong bộ giải mã bắt đầu bằng việc tăng kích thước của hình ảnh bằng cách sử dụng các phương pháp như transposed convolution (hay còn gọi là deconvolution).

Skip Connections: Ở mỗi bước, đầu ra từ bộ mã hóa (tầng có cùng kích thước) được kết hợp với đầu ra từ bộ giải mã. Việc này giúp đưa thông tin chi tiết từ bước trước vào trong bước hiện tại, giữ lại những đặc trưng không gian quan trọng.

Convolutional Layers: Sau khi kết hợp, các lớp tích chập được áp dụng để tinh chỉnh và giảm số lượng kênh.

Đầu ra:

Tầng đầu ra thường là một lớp tích chập với kích thước đầu ra bằng kích thước đầu vào, sử dụng hàm kích hoạt sigmoid (cho phân đoạn nhị phân) hoặc softmax (cho phân đoạn đa lớp) để tạo ra bản đồ phân đoạn.

3.2.2 Các bước thực hiện

- Input:

+ Đầu vào của mô hình là những hình ảnh kích cỡ (128, 128, 3) - chiều cao = 128, chiều rộng = 128 với 3 kênh màu RGB.

+ Các batch dữ liệu hình ảnh được chuẩn bị từ train_dataset và test_dataset với kích thước batch = 64 và dùng trong quá trình huấn luyện.

+ Các hình ảnh được đưa qua bộ tiền xử lý để chia thành các batch, được trộn (shuffle), lưu vào bộ nhớ đệm (cache), và thực hiện tiền tải (prefetch).

- Encoder (Contraction Path):

+ Downsample Blocks: Mô hình U-Net sử dụng các khối downsampling (thu nhỏ kích thước) để trích xuất các đặc trưng từ ảnh đầu vào:

Mỗi khối downsampling bao gồm:

+ Double Convolution: Thay vào đó, hai lớp Conv2D với kích thước bộ lọc 3x3 và hàm kích hoạt là ReLU.

+ Max Pooling: sử dụng lớp MaxPool2D để giảm kích thước ảnh

+ Dropout: Dropout được áp dụng để tránh overfitting.

+ Các khối downsampling lần lượt trích xuất các đặc trưng ở các mức độ khác nhau và giảm kích thước của ảnh qua mỗi bước:

- Khối 1: Sử dụng 64 bộ lọc.
- Khối 2: Sử dụng 128 bộ lọc.
- Khối 3: Sử dụng 256 bộ lọc.
- Khối 4: Sử dụng 512 bộ lọc .

- Bottleneck:

+ Double Convolution: Sau khối downsampling cuối cùng, hình ảnh được chuyển qua một khối tích chập chứa 1024 bộ lọc để xử lý các đặc trưng trừu tượng ở mức thấp nhất.

- Decoder (Expansion Path):

+ Upsample Blocks: Để phục hồi kích thước của ảnh về độ phân giải ban đầu, mô hình sử dụng các khối upsampling (tăng kích thước ảnh):

+ Conv2DTranspose: Sử dụng lớp Conv2DTranspose để tăng kích thước ảnh.

+ Concatenate: Các đặc trưng từ phần encoder được ghép với ảnh sau khi upsampling để bảo toàn thông tin chi tiết.

+ Dropout: Nhằm giảm thiểu hiện tượng overfitting.

+ Double Convolution: Áp dụng lại các lớp Conv2D với activation ReLU.

+ Các khối upsampling sẽ tăng dần số lượng bộ lọc:

- Khối 6: Sử dụng 512 bộ lọc.
- Khối 7: Sử dụng 256 bộ lọc.
- Khối 8: Sử dụng 128 bộ lọc.
- Khối 9: Sử dụng 64 bộ lọc.

- Lớp Output:

+ Conv2D (1x1): Cuối cùng, một lớp Conv2D với kích thước bộ lọc 1x1 được áp dụng để giảm số lượng kênh xuống 3 (cho 3 lớp phân vùng trong trường hợp phân vùng đa lớp).

Softmax: Hàm kích hoạt Softmax trong lớp cuối cùng thường được sử dụng để phân loại từng pixel vào các lớp tương ứng trong bài toán phân vùng. Nói cách khác, mỗi pixel sẽ được phân loại vào lớp 0, 1, hoặc 2.

- Mô Hình Huấn Luyện:

+ Tối ưu hóa và Loss Mô hình được biên dịch bằng cách sử dụng bộ tối ưu Adam và hàm mất mát sparse_categorical_crossentropy để huấn luyện mô hình.

Huấn luyện: Mô hình được huấn luyện trên train_batches với 20 epochs. Các bước huấn luyện được tính toán từ tổng số dữ liệu chia cho kích thước batch. Mỗi epoch mô hình sẽ thực hiện việc cập nhật trọng số dựa trên loss và các đặc trưng học được từ dữ liệu huấn luyện.

- Đầu Ra (Output):

+ Output của mô hình là output mask có kích thước (128, 128, 3), nơi mỗi pixel có xác suất thuộc về một trong các lớp phân vùng - ví dụ 3 lớp trong bài toán phân vùng ba lớp. Mô hình phân loại từng pixel của ảnh vào các lớp tương ứng dựa trên đặc trưng đã học.

3.3 Phương pháp đánh giá

- Phương pháp:

Chún em sài hàm Jaccard Index để đánh giá mô hình, hàm Jaccard Index được tính bằng công thức giao của hai điểm A và B chia cho hội của hai điểm A và B:

$$\text{Jaccard Index} = \frac{A \cap B}{A \cup B}$$

Với bài toán phân vùng ảnh A là phân vùng được dự đoán và B là phân vùng thực tế

Trong đó:

$A \cap B$: là số lượng pixel có giá trị **1** ở cả hai ảnh, tức là những pixel chung của A và B

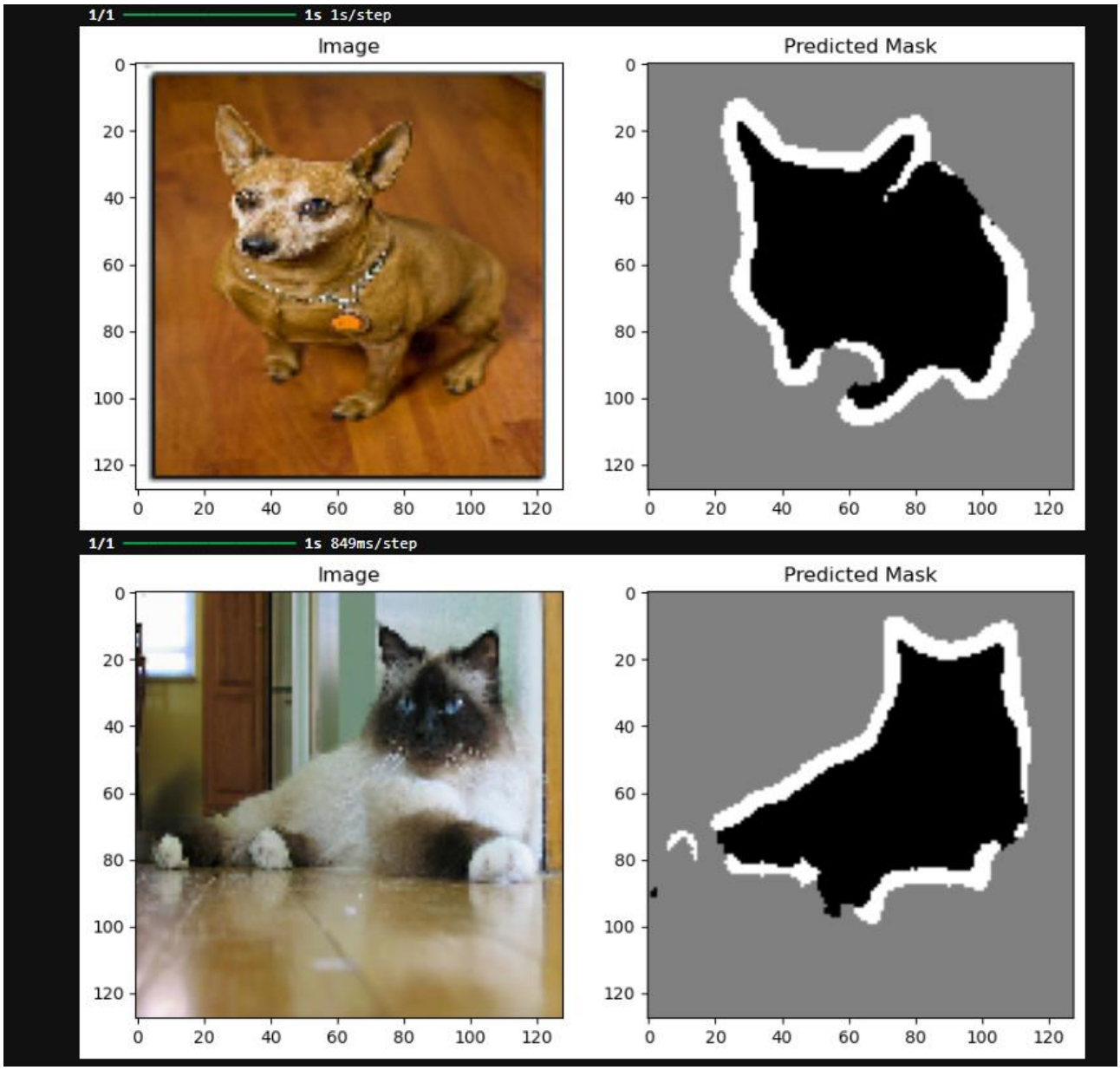
$A \cup B$: là tập hợp các phần tử có mặt trong ít nhất một trong hai tập hợp

- Tiêu chí:

  Chúng em dung tiêu chí tính trung bình của các giá trị Jaccard Index cho tất cả các lớp trong bài toán phân vùng ảnh

Chương 4: Hiện thực kết quả

4.1 Kết quả thực nghiệm



Chương 5: Kết luận

5.1 Kết quả đạt được

Kết quả sau khi sử dụng mô hình U-Net phân vùng ảnh là hệ thống tự động phân tách các khu vực dựa trên đặc trưng và cấu trúc của đối tượng trong hình ảnh. Cụ thể là khả năng U-Net học được phân vùng ảnh chính xác và hiệu quả, từ đó tạo ra các nhãn cho từng vùng đối tượng trong ảnh. Điều này giúp tách biệt các phần của ảnh theo từng lớp (class) hoặc khu vực (region) cụ thể, phục vụ cho các ứng dụng như phân đoạn ảnh y tế, phân tích hình ảnh, nhận diện đối tượng, và nhiều tác vụ thị giác máy tính khác.

Mô hình U-Net với kiến trúc encoder-decoder kết hợp với các lớp convolution và upsampling đã cho phép mô hình có thể tái tạo lại được các đặc trưng quan trọng của ảnh trong quá trình phân vùng, đồng thời giữ được chi tiết tinh vi ngay cả ở những tầng sâu. Do khả năng dùng dữ liệu huấn luyện có độ chính xác cao mà U-Net cho ra những kết quả phân vùng chính xác, rõ ràng, tạo ra những phân vùng ảnh sắc nét giúp ích cho những nhiệm vụ phân tích và xử lý ảnh sau này.

Trong tổng quát, phân vùng ảnh đã đưa U-Net vào để có được những kết quả xuất sắc, đặc biệt là trong các bài toán phân đoạn ảnh phức tạp nơi mà việc giữ lại các chi tiết và cấu trúc ảnh là rất quan trọng.

5.2 Ưu và nhược điểm

- Ưu điểm:

Hiệu suất phân vùng ảnh cao: U-Net rất hiệu quả trong các bài toán phân đoạn ảnh, đặc biệt trong các ứng dụng như phân tích ảnh y tế, nhận diện đối tượng, và phân vùng các chi tiết phức tạp. Khả năng phân vùng ảnh chính xác của U-Net giúp mô hình có thể tái tạo các đặc trưng ảnh chi tiết và tinh vi.

Khả năng giữ lại thông tin: Kiến trúc skip connections (kết nối bỏ qua) giúp U-Net duy trì thông tin quan trọng từ các tầng thấp (input) đến các tầng cao, giúp mô hình không mất đi các chi tiết quan trọng trong quá trình phân vùng.

Khả năng xử lý ảnh nhỏ và nhanh: Mô hình U-Net có thể hoạt động hiệu quả với các ảnh có kích thước nhỏ (như 128x128) và có thể áp dụng các kỹ thuật tăng cường dữ liệu (data augmentation) như lật, xoay, và thay đổi kích thước, giúp cải thiện độ chính xác và tính tổng quát của mô hình.

Tiết kiệm tài nguyên: U-Net yêu cầu ít tài nguyên tính toán so với các mô hình phân vùng ảnh phức tạp hơn như Mask R-CNN, nhờ vào việc giảm bớt số lượng tham số và kết cấu đơn giản nhưng hiệu quả.

- Nhược điểm:

Chỉ phù hợp cho phân vùng ảnh có cấu trúc rõ ràng: U-Net hoạt động rất tốt với các hình ảnh có cấu trúc và đặc trưng rõ ràng (như phân đoạn ảnh y tế), nhưng khi đối mặt với các ảnh có độ phức tạp cao hoặc đối tượng không rõ ràng, kết quả phân vùng có thể không chính xác.

Khó khăn khi xử lý ảnh lớn: U-Net có thể gặp khó khăn khi xử lý các ảnh có độ phân giải cao hoặc ảnh có kích thước lớn, do yêu cầu bộ nhớ và tài nguyên tính toán lớn. Các chiến lược như cropping hoặc resizing có thể ảnh hưởng đến chất lượng phân vùng. Cần dữ liệu huấn luyện đủ lớn: Để mô hình hoạt động tối ưu, U-Net yêu cầu một lượng dữ liệu huấn luyện đủ lớn để học được các đặc trưng ảnh chính xác. Nếu dữ liệu huấn luyện không đủ phong phú hoặc bị thiếu, mô hình có thể bị overfitting hoặc không đạt được hiệu quả tốt.

5.3 Hướng mở rộng tương lai

Trong tương lai chúng em sẽ có các hướng mở rộng hơn nữa để phát triển mô hình như:

+ Sử dụng U_net++ để cải thiện hiệu suất phân vùng trong các trường hợp phức tạp, giúp mô hình phân vùng chính xác hơn trong các tình huống mà đối tượng và bối cảnh có sự chồng lấn hoặc độ phức tạp cao

+ Kết hợp U-Net với các mạng nơ ron khác như ResNet, DenseNet hoặc VGG để cải thiện khả năng nhận diện đặc trưng sâu hơn trong ảnh. Kết hợp các mô hình này có thể cải thiện chất lượng phân vùng bằng cách tận dụng các đặc trưng hình ảnh ở nhiều cấp độ khác nhau

Đề tài này chúng em đang phân vùng ảnh chó hoặc hoặc mèo trong tương lai chúng em có thể áp dụng mô hình hình U_net vào trong các ứng dụng thực tế như:

+ Y học: U-Net có thể được áp dụng mạnh mẽ trong phân tích ảnh y tế, đặc biệt là trong việc phân vùng các tổn thương, khối u, hoặc các mô học quan trọng trong ảnh MRI, CT Scan, hoặc X-ray. Việc mở rộng mô hình với các dữ liệu y tế phức tạp hơn có thể mang lại nhiều tiến bộ trong chẩn đoán tự động.

+ Phân vùng ảnh vệ tinh: Trong những bài toán phân vùng ảnh vệ tinh, mô hình U-Net có thể được sử dụng để phân biệt các lớp như rừng, nước, thành phố, đất đai. Việc cải thiện độ phân giải ảnh và kết hợp với các phương pháp học máy khác có thể mở rộng khả năng ứng dụng trong việc giám sát môi trường, lập bản đồ hoặc phân tích thay đổi đất đai.

+ Nhận diện đối tượng: Bên cạnh phân vùng ảnh, U-Net có thể được kết hợp với các mạng nhận diện đối tượng để phát hiện và phân loại các đối tượng trong ảnh, tạo ra các ứng dụng như nhận dạng đối tượng trong ảnh, phát hiện vật thể trong video, v.v.

Ngoài ra trong phạm vi đề tài này chúng em chỉ làm trên môi trường python trong tương lai chúng em có thể triển khai mô hình này trên các nền tảng khác nhau như di động, web, giúp mở rộng khả năng ứng dụng thực tế của mô hình, từ việc phân vùng ảnh trên các thiết bị di động đến các ứng dụng giám sát trong môi trường thực tế

Phụ lục code demo

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np
```

1 Import các thư viện

2 Tải dữ liệu huấn luyện train và test, in thông tin dữ liệu

- Tải dữ liệu train, test

```python
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

```python
print(info)
```

- Thông tin của dữ liệu

```
tfds.core.DatasetInfo(
    name='oxford_iiit_pet',
    full_name='oxford_iiit_pet/3.2.0',
    description="""
    The Oxford-IIIT pet dataset is a 37 category pet image dataset with roughly 200
    images for each class. The images have large variations in scale, pose and
    lighting. All images have an associated ground truth annotation of breed.
    """,
    homepage='http://www.robots.ox.ac.uk/~vgg/data/pets/',
    data_dir='C:\\Users\\thuong\\tensorflow_datasets\\oxford_iiit_pet\\3.2.0',
    file_format=tfrecord,
    download_size=773.52 MiB,
    dataset_size=774.69 MiB,
    features=FeaturesDict({
        'file_name': Text(shape=(), dtype=string),
        'image': Image(shape=(None, None, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=37),
        'segmentation_mask': Image(shape=(None, None, 1), dtype=uint8),
        'species': ClassLabel(shape=(), dtype=int64, num_classes=2),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=3669, num_shards=4>,
        'train': <SplitInfo num_examples=3680, num_shards=4>,
    },
    citation="""@InProceedings{parkhi12a,
      author       = "Parkhi, O. M. and Vedaldi, A. and Zisserman, A. and Jawahar, C.~V.",
      title        = "Cats and Dogs",
      booktitle    = "IEEE Conference on Computer Vision and Pattern Recognition",
      year         = "2012",
    }""",
)
```

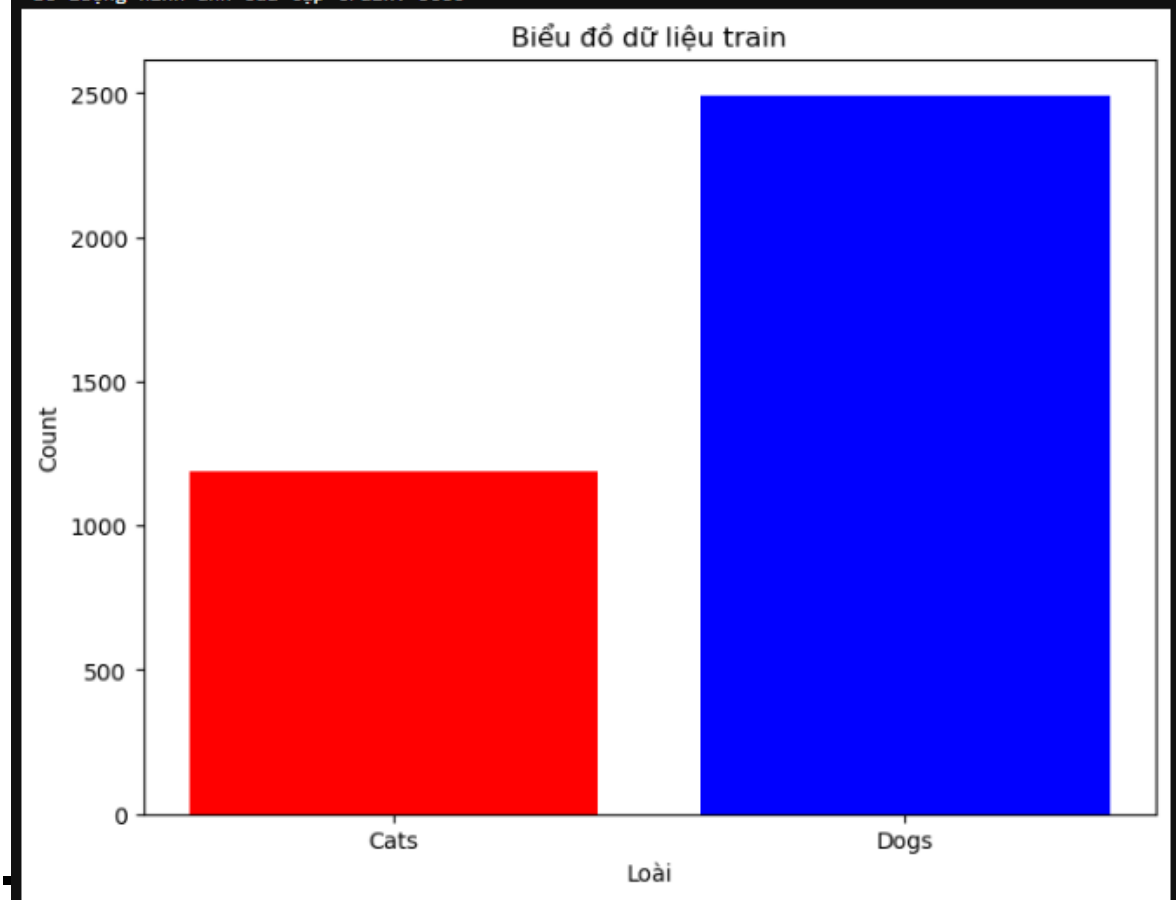
```python
def visualize_data_distribution(dataset, dataset_name):
    num_cats = 0
    num_dogs = 0

    # Iterate over the dataset to count cats and dogs
    for example in dataset:
        species = example['species'].numpy()  # Assuming 'species' is a tensor
        if species == 0:  # 0 for cat
            num_cats += 1
        elif species == 1:  # 1 for dog
            num_dogs += 1

    # Plot the results
    labels = ['Cats', 'Dogs']
    counts = [num_cats, num_dogs]

    plt.figure(figsize=(8, 6))
    plt.bar(labels, counts, color=['red', 'blue'] if dataset_name == 'train' else ['yellow', 'blue'])
    plt.xlabel('Loài')
    plt.ylabel('Count')
    plt.title(f'Biểu đồ dữ liệu {dataset_name}')
    plt.show()
```

3 Hàm để trực quan hóa dữ liệu train và test

- Trực quan hóa hàm train:

```python
# Visualize the training data
train_ds = dataset['train']
print('Số lượng hình ảnh của tập train:', len(train_ds))
visualize_data_distribution(train_ds, 'train')
```

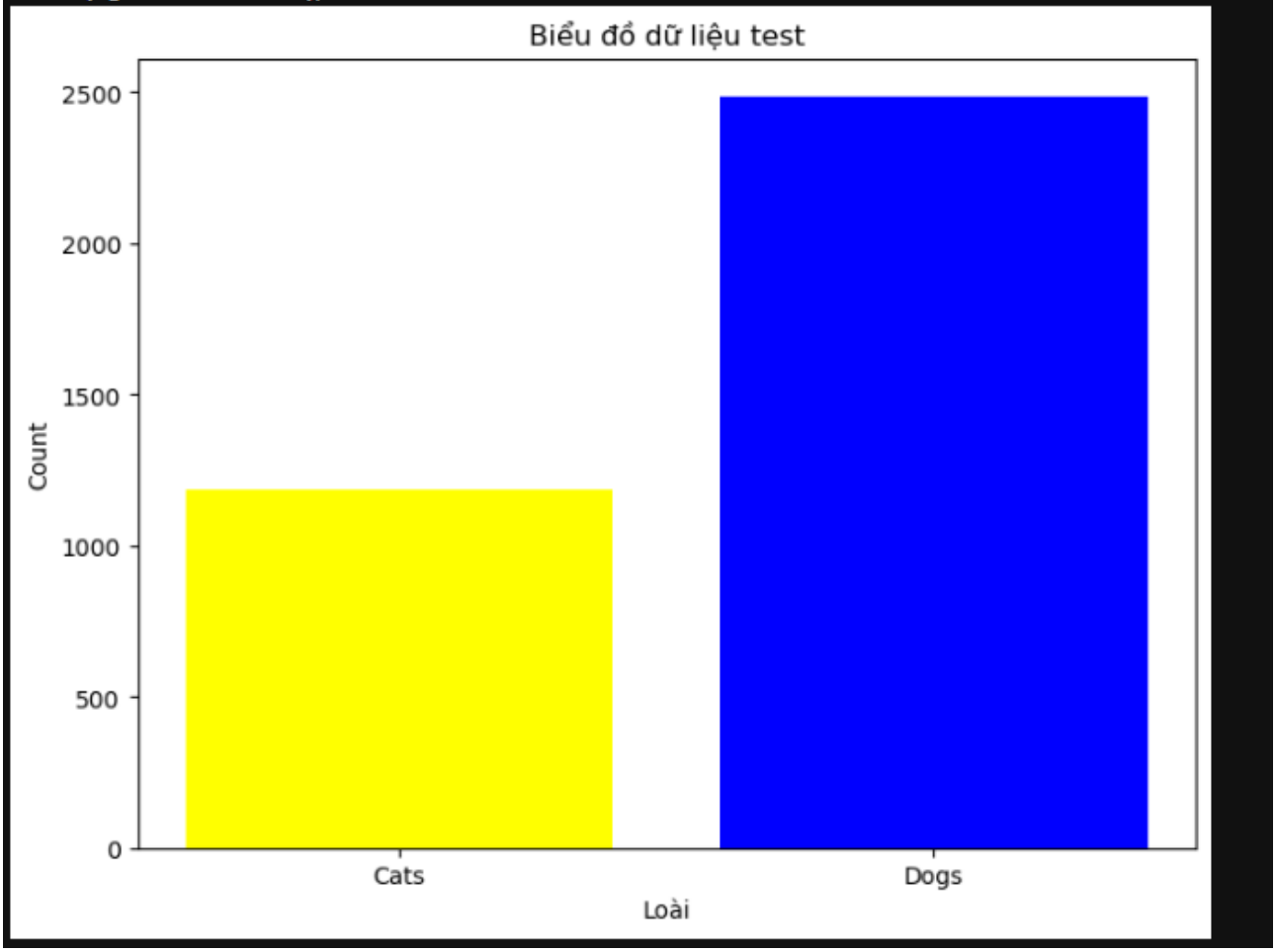
```
Số lượng hình ảnh của tập train: 3680
```



- Trực quan hóa hàm test

```
# Visualize the test data
test_ds = dataset['test']
print('Số lượng hình ảnh của tập test:', len(test_ds))
visualize_data_distribution(test_ds, 'test')
```

```
Số lượng hình ảnh của tập test: 3669
```



4 Thông tin về bộ nhớ của dữ liệu train và test

```
# Lấy thông tin về số lượng ảnh trong train và test
train_size = info.splits['train'].num_examples
test_size = info.splits['test'].num_examples

# Kiểm tra kích thước của một ảnh
image_shape = info.features['image'].shape

# Kiểm tra nếu image_shape có giá trị None, lấy giá trị từ ảnh mẫu
if None in image_shape:
    # Lấy một ảnh mẫu từ dataset để lấy kích thước
    sample_image = next(iter(dataset['train']))['image']
    image_shape = sample_image.shape

# Lấy chiều cao, chiều rộng và số kênh từ image_shape
image_height, image_width, num_channels = image_shape
bytes_per_pixel = 4  # Giả sử ảnh là float32, tức là mỗi pixel chiếm 4 bytes

# Tính bộ nhớ của một ảnh
memory_per_image_bytes = image_height * image_width * num_channels * bytes_per_pixel
memory_per_image_gb = memory_per_image_bytes / (1024 ** 3)  # Chuyển đổi sang GB

# Tính bộ nhớ tổng cho tập train và test
train_memory_gb = train_size * memory_per_image_gb
test_memory_gb = test_size * memory_per_image_gb

# In kết quả
print(f"Bộ nhớ của tập train: {train_memory_gb:.2f} GB")
print(f"Bộ nhớ của tập test: {test_memory_gb:.2f} GB")
```

```
Bộ nhớ của tập train: 10.28 GB
Bộ nhớ của tập test: 10.25 GB
```

5 Điều chỉnh kích thước và tăng cường dữ liệu hình ảnh đầu vào

```
#điều chỉnh kích thước hình ảnh về 128 128
def resize(input_image,input_mask):
    input_image = tf.image.resize(input_image,(128,128),method="nearest")

    input_mask = tf.image.resize(input_mask,(128,128),method = "nearest")

    return input_image,input_mask
```

```
# tăng cường dữ liệu bằng cách lật theo chiều ngang
def agument(input_image,input_mask):
    if tf.random.uniform(()) > 0.5:
        #Lật ngẫu nhiên
        input_image = tf.image.flip_left_right(input_image)

        input_mask = tf.image.flip_left_right(input_mask)

    return input_image,input_mask
```

6 Chuẩn hóa dữ liệu hình ản đầu vào và hình ảnh đã đươc phân vùng

```
#Chuẩn hóa dữ liệu chia hình ảnh nằm trong khoảng [-1,1] và đưa mặt nạ nhị phân về 1

def normalize(input_image,input_mask):
    input_image = tf.cast(input_image,tf.float32)/255.0
    input_mask -= 1
    return input_image,input_mask
```

7 Tiền xử lý trên tập huấn luyện

```
#tiền xử lý dữ liệu trên tập huấn luyện
def load_image_train(datapoint):
    input_image = datapoint["image"]
    input_mask = datapoint["segmentation_mask"]

    input_image,input_mask = resize(input_image,input_mask)
    input_image,input_mask = agument(input_image,input_mask)
    input_image,input_mask = normalize(input_image,input_mask)

    return input_image,input_mask
```

8 Tiền xử lý trên tập kiểm tra

```
#tiền xử lý trên tập test
def load_image_test(datapoint):
    input_image = datapoint["image"]
    input_mask = datapoint["segmentation_mask"]

    input_image,input_mask = resize(input_image,input_mask)
    input_image,input_mask = normalize(input_image,input_mask)

    return input_image,input_mask
```

9 Khởi tạo và chuẩn bị dữ liệu huấn luyện và kiểm tra

```
#Xây dựng hàm để khởi tạo
train_dataset = dataset["train"].map(load_image_train, num_parallel_calls=tf.data.AUTOTUNE)
test_dataset = dataset["test"].map(load_image_test,num_parallel_calls=tf.data.AUTOTUNE)
```

```
Batch_size = 64
Buffer_size = 1000
train_batches = train_dataset.cache().shuffle(Buffer_size).batch(Batch_size).repeat()
train_batches = train_batches.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

validation_bathches = test_dataset.take(3000).batch(Batch_size)
test_batches = test_dataset.skip(3000).take(669).batch(Batch_size)
```

10 Hiển thị hình ảnh mẫu trong dữ liệu huấn luyện

```
def display(display_list):
 plt.figure(figsize=(15, 15))
 title = ["Input Image", "True Mask"]
 for i in range(len(display_list)):
   plt.subplot(1, len(display_list), i+1)
   plt.title(title[i])
   plt.imshow(tf.keras.utils.array_to_img(display_list[i]))
   plt.axis("off")
 plt.show()
```

```
sample_batch = next(iter(train_batches))
random_index = np.random.choice(sample_batch[0].shape[0])
sample_image, sample_mask = sample_batch[0][random_index], sample_batch[1][random_index]
display([sample_image, sample_mask])
```

11 Hàm xây dựng mạng CNN để trích gọn đặc trưng

```
def double_conv_block(x,n_fillters): #x đầu vào cho khối tích chập, số lượng bộ lọc, 3 kích thước

    #Conv2D then RELU activation
    x = layers.Conv2D(n_fillters, 3, padding = "same", activation= "relu", kernel_initializer="he_normal")(x)
    x = layers.Conv2D(n_fillters, 3, padding = "same", activation= "relu", kernel_initializer="he_normal")(x)
    return x
```

12 Hàm downsampling để giảm kích thước của hình ảnh

```
def downsample_block(x, n_fillters):
    f = double_conv_block(x,n_fillters)
    p = layers.MaxPool2D(2)(f)
    p = layers.Dropout(0.3)(p)

    return f,p
```

13 Hàm upsampling tang kích thước của hình ảnh

```
def upsample_block(x,conv_features, n_fillters):

    #upsample
    x = layers.Conv2DTranspose(n_fillters, 3, 2, padding="same")(x)
    # concatenate
    x = layers.concatenate([x,conv_features])
    #dropout
    x = layers.Dropout(0.3)(x)
    # Conv2D twice with ReLU activation
    x = double_conv_block(x,n_fillters)

    return x
```

14 Hàm xây dựng mô hình

```
#inputs
def build_unet_model():
    inputs = layers.Input(shape=(128,128,3))

    # encoder: contracting path - downsample
    # 1 - downsample
    f1,p1 = downsample_block(inputs,64)

    # 2 - downsample
    f2,p2 = downsample_block(p1,128)

    # 3 - downsample
    f3,p3 = downsample_block(p2,256)

    f4,p4 = downsample_block(p3,512)

    # 5 - bottleneck
    bottleneck = double_conv_block(p4,1024)

    # decoder: expanding path - upsample
    # 6 - upsample
    u6 = upsample_block(bottleneck, f4, 512)

    # 7 - upsample
    u7 = upsample_block(u6,f3,256)

    # 8 - upsample
    u8 = upsample_block(u7,f2,128)

    # 9 - upsample
    u9 = upsample_block(u8,f1,64)

    outputs = layers.Conv2D(3, 1, padding="same", activation="softmax")(u9)

    u_net_model = tf.keras.Model(inputs,outputs,name="U-Net")
    return u_net_model
```

15 Khởi tạo các tham số

```
unet_model = build_unet_model()
```

```
NUM_EPOCHS = 20

TRAIN_LENGTH = info.splits["train"].num_examples
STEPS_PER_EPOCH = TRAIN_LENGTH // Batch_size


VAL_SUBSPLITS = 5
TEST_LENTH = info.splits["test"].num_examples
VALIDATION_STEPS = TEST_LENTH // Batch_size // VAL_SUBSPLITS
```

16 Biên dịch và huấn luyện mô hình

```
NUM_EPOCHS = 20

TRAIN_LENGTH = info.splits["train"].num_examples
STEPS_PER_EPOCH = TRAIN_LENGTH // Batch_size

|
VAL_SUBSPLITS = 5
TEST_LENTH = info.splits["test"].num_examples
VALIDATION_STEPS = TEST_LENTH // Batch_size // VAL_SUBSPLITS
```

```
# Compile the model
unet_model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

model_history = unet_model.fit(
    train_batches,
    epochs=NUM_EPOCHS,
    steps_per_epoch=STEPS_PER_EPOCH,
    validation_data=test_batches,  # Sử dụng test_dataset làm validation
    validation_steps=VALIDATION_STEPS,
)
```

17 Lưu mô hình sau khi đã huấn luyện

```
# Lưu mô hình đã huấn luyện
unet_model.save('unet_model.keras')
```

```
from tensorflow.keras.models import load_model

# Tải mô hình đã huấn luyện
unet_model = load_model('unet_model.keras', compile=False)
```

18 Xây dựng hàm đánh giá Jaccard Indexsau khi huấn luyện

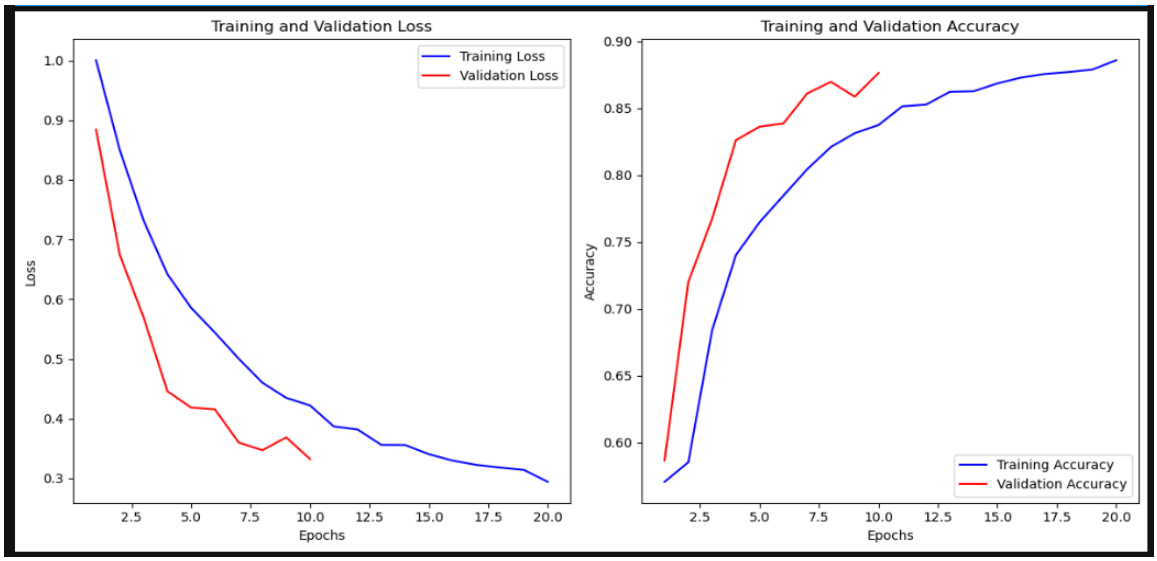
```
# Hàm đánh giá Mean IoU (Jaccard Index)
def evaluate_jaccard_index(model, dataset, binary=True):
    jaccard_scores = []

    # Thiết lập `num_classes` dựa trên số lớp của `true_masks`
    for batch_data in dataset:
        input_images, true_masks = batch_data
        num_classes = 2 if binary else int(tf.reduce_max(true_masks)) + 1
        mean_iou_metric = tf.metrics.MeanIoU(num_classes=num_classes)

        true_masks = tf.cast(true_masks, dtype=tf.int32)  # MeanIoU yêu cầu nhãn là kiểu int

        # Dự đoán mặt nạ cho batch
        pred_masks = model.predict(input_images)

        if not binary:  # Phân đoạn đa lớp
            pred_masks = tf.argmax(pred_masks, axis=-1)
        else:  # Phân đoạn nhị phân
            pred_masks = tf.round(pred_masks)  # Làm tròn cho phân đoạn nhị phân

        pred_masks = tf.cast(pred_masks, dtype=tf.int32)  # Chuyển về kiểu int

        # Cập nhật giá trị cho metric MeanIoU
        mean_iou_metric.update_state(true_masks, pred_masks)

    # Lấy giá trị MeanIoU cuối cùng
    mean_jaccard = mean_iou_metric.result().numpy()
    mean_iou_metric.reset_state()  # Reset trạng thái metric cho lần đánh giá tiếp theo

    return mean_jaccard
# Đánh giá Mean IoU trên tập kiểm tra
mean_jaccard_test = evaluate_jaccard_index(unet_model, test_batches, binary=False)
```

```
print(f"Kết quả đánh giá: {mean_jaccard_test:.4f}")
```

```
Kết quả đánh giá: 0.7002
```

19 Vẽ biểu đồ thể hiện hàm loss và accuracy

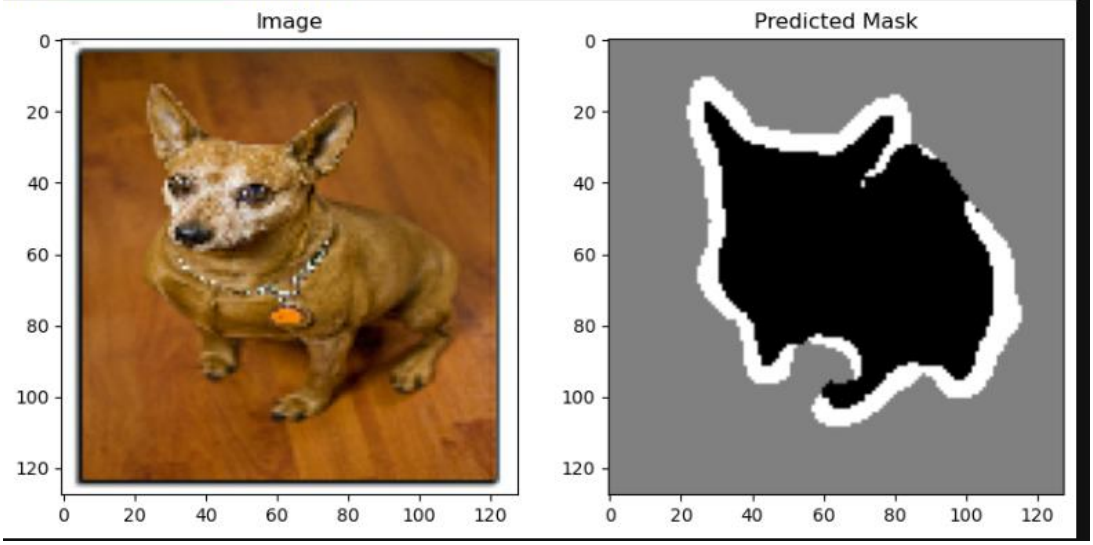
```python
import matplotlib.pyplot as plt
# not compute Mul as input #1(zero-based) was e

# Lấy giá trị loss và accuracy từ model_history
loss = model_history.history['loss']
val_loss = model_history.history.get('val_loss', [])  # Sử dụng .get để tránh lỗi nếu

accuracy = model_history.history['accuracy']
val_accuracy = model_history.history.get('val_accuracy', [])

epochs = range(1, len(loss) + 1)
val_epochs = range(1, len(val_loss) + 1)  # Epochs cho validation nếu khác nhau

# Vẽ đồ thị Loss
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(epochs, loss, 'b', label='Training Loss')
if val_loss:  # Chỉ vẽ nếu có dữ liệu validation
    plt.plot(val_epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Vẽ đồ thị Accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, accuracy, 'b', label='Training Accuracy')
if val_accuracy:  # Chỉ vẽ nếu có dữ liệu validation
    plt.plot(val_epochs, val_accuracy, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



20 Hiển thị kết quả phân vùng dựa trên tập test

```python
import tensorflow as tf
import matplotlib.pyplot as plt

# Hàm hiển thị ảnh và mask dự đoán
def display_predictions(image, pred_mask):
    plt.figure(figsize=(10, 5))

    # Hiển thị ảnh gốc
    plt.subplot(1, 2, 1)
    plt.imshow(tf.squeeze(image))
    plt.title("Image")

    # Hiển thị mask dự đoán
    plt.subplot(1, 2, 2)
    plt.imshow(tf.argmax(pred_mask, axis=-1)[0], cmap='gray')
    plt.title("Predicted Mask")

    plt.show()

def show_test_predictions(test_batches, num=6):
    for images, _ in test_batches.take(1):  # Take one batch of images
        for i in range(num):
            # Select the ith image in the batch
            image = images[i]

            # Expand dimensions to simulate a batch of size 1
            image = tf.expand_dims(image, axis=0)

            # Generate prediction
            pred_mask = unet_model.predict(image)

            # Display the image and prediction
            display_predictions(tf.squeeze(image, axis=0), pred_mask)

# Hiển thị 6 ảnh từ tập test với mask dự đoán sau khi huấn luyện
show_test_predictions(test_batches, num=6)
```

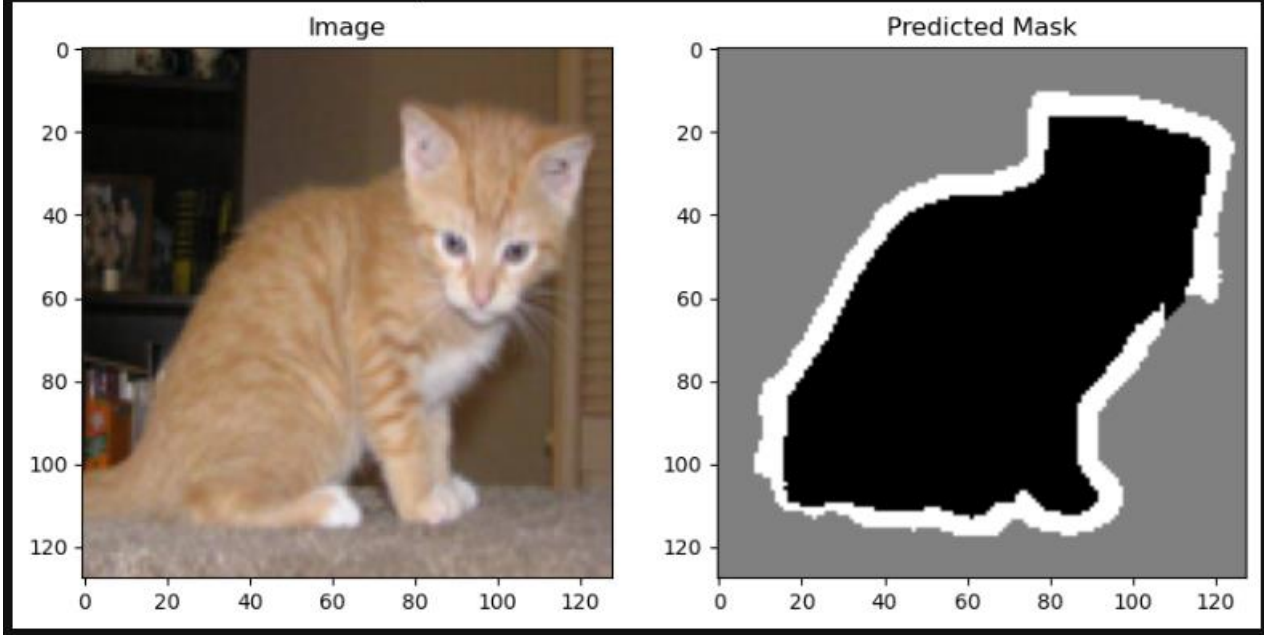
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
```



21 Hiển thị kết quả phân vùng với hình ảnh không có trong tập train và tập test

```python
# Hàm hiển thị ảnh và mask dự đoán
def display_predictions(image, pred_mask):
    plt.figure(figsize=(10, 5))

    # Hiển thị ảnh gốc
    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title("Image")

    # Hiển thị mask dự đoán
    plt.subplot(1, 2, 2)
    plt.imshow(tf.argmax(pred_mask, axis=-1)[0], cmap='gray')
    plt.title("Predicted Mask")

    plt.show()

# Hàm dự đoán cho một ảnh đầu vào
def predict_image(image_path):
    # Đọc ảnh từ đường dẫn và tiền xử lý
    image = Image.open(image_path)
    image = image.resize((128, 128))
    image = np.array(image) / 255.0  # Chuẩn hóa giá trị pixel

    # Thêm chiều batch để có dạng (1, 128, 128, 3) cho đầu vào mô hình
    image = tf.expand_dims(image, axis=0)

    # Dự đoán mask phân vùng
    pred_mask = unet_model.predict(image)

    # Hiển thị ảnh và mask dự đoán
    display_predictions(image[0], pred_mask)

#  sử dụng hàm với một đường dẫn ảnh
image_path ="C:/Users/thuong/Pictures/Dog_Cat/dataset/test_set/cats/cat.4003.jpg"
predict_image(image_path)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step
```



Tài liệu tham khảo

[1] “pyimagesearch,” [Online]. Available: https://pyimagesearch.com/2022/02/21/u-net-image-segmentation-in-keras/#:~:text=U-Net%20is%20a%20semantic%20segmentation%20technique%20originally%20proposed%20for%20medical.

[2] “viblo,” [Online]. Available: https://viblo.asia/p/giai-thuat-jaccard-djeZ1P9GKWz.

[3] “Youtube,” [Online]. Available: https://www.youtube.com/watch?v=NhdzGfB1q74.

[4] “geeksforgeeks,” [Online]. Available: https://www.geeksforgeeks.org/u-net-architecture-explained/.

[5] “Youtube,” [Online]. Available: https://www.youtube.com/watch?v=NgrLIrOATrE&t=18s.

[6] “Youtube,” [Online]. Available: https://www.youtube.com/watch?v=fjPxKahFgTM&t=802s.

[7] “Youtube,” [Online]. Available: https://www.youtube.com/watch?v=lgV1O4fHg6A&t=1322s.

[8] “Youtube,” [Online]. Available: https://www.youtube.com/watch?v=Ejb6utg_mjw.

[9] “Youtube,” [Online]. Available: https://www.youtube.com/watch?v=kWKvNpAU6a4&t=20s.

[10] “Youtube,” [Online]. Available: https://www.youtube.com/watch?v=gi8RReHcXZE&t=240s.