TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Nhóm 1 - Lớp 24TNT1

Project: Super-Resolution Using CNN

ĐỒ ÁN MÔN THỰC HÀNH GIỚI THIỆU NGÀNH TRÍ TUỆ NHÂN TẠO
CHƯƠNG TRÌNH CHÍNH QUY

Tp. Hồ Chí Minh, tháng 11/2024

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Nhóm 1 - 24TNT1
Project: Super-Resolution Using CNN

Nguyễn Tuấn Lâm - 24122006
Đỗ Lê Phong Phú - 24122010
Quách Lê Nhật Huy - 24122016
Phan Bảo Đức Phát - 24122020
Nguyễn Thanh Tuấn - 24122025
Trần Trọng Tú - 24122024

ĐỒ ÁN MÔN THỰC HÀNH GIỚI THIỆU NGÀNH TRÍ TUỆ NHÂN TẠO
CHƯƠNG TRÌNH CHÍNH QUY

GIÁO VIÊN HƯỚNG DẪN

ThS. Nguyễn Trần Duy Minh
ThS. Phạm Trọng Nghĩa
TS. Lê Ngọc Thành

Tp. Hồ Chí Minh, tháng 11/2024

Lời cảm ơn

Nhóm 1 xin chân thành cảm ơn sự giúp đỡ tận tình của giáo viên hướng dẫn đã hỗ trợ nhóm trong lúc làm đồ án. Sự chỉ dẫn của thầy là thành phần không thể thiếu cho đồ án của nhóm chúng em.

Mục lục

Danh sách hình

Danh sách bảng

Chương 1

Đánh Giá Thành Viên

Member	Đánh giá (Progress Bar)
Lâm	(10/10)
Phú	(10/10)
Huy	(10/10)
Phát	(10/10)
Tuấn	(10/10)
Tú	(10/10)

Bảng 1.1: Đánh thành viên theo tiến độ

Chương 2

Đánh Giá Mức Độ Hoàn Thành Cho Từng Yêu Cầu

Yêu cầu	Đánh Giá (Progress Bar)
Phân công công việc	(10/10)
Lập trình và mô hình A.I	(10/10)
Báo cáo các mục:	(10/10)
Introduction	(10/10)
Methodology	(10/10)
Conclusion	(10/10)
Lưu source code trên Github	(10/10)
Slide	(10/10)
Video	(10/10)

Bảng 2.1: Đánh thành viên theo tiến độ

Chương 3

Nội Dung Chính

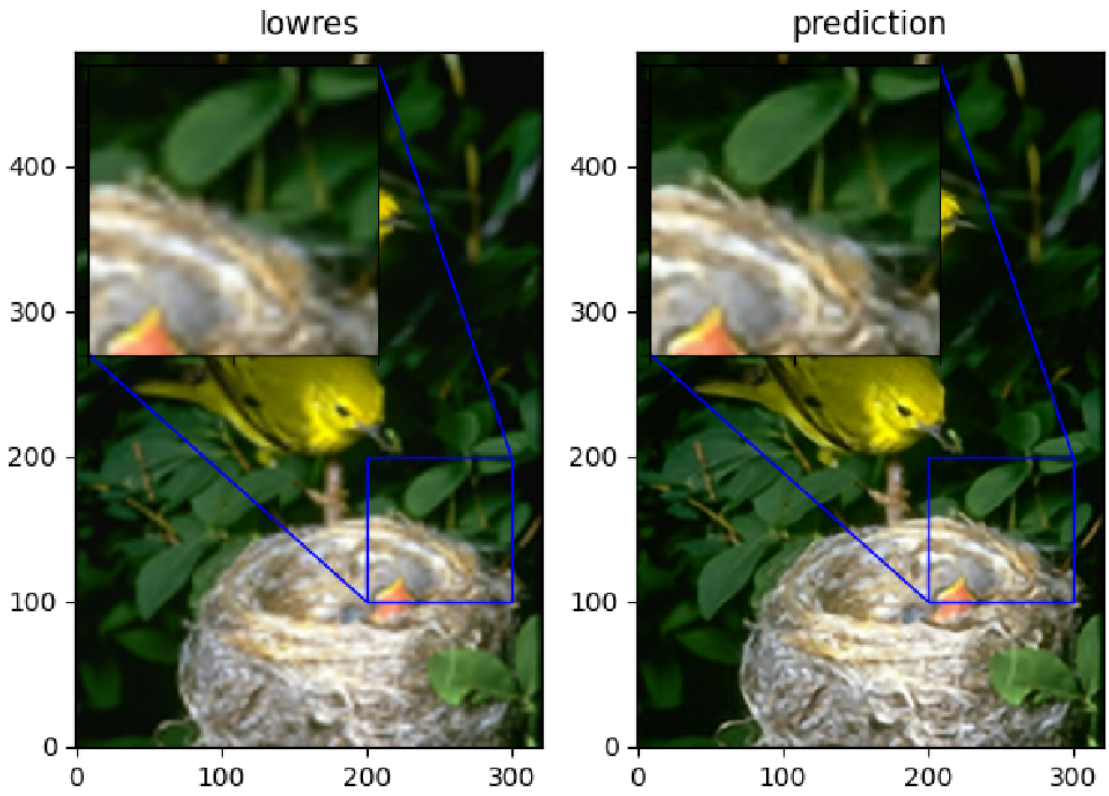
1. Introduction: Giới Thiệu Bài Toán

Trong dự án này, chúng ta sẽ xây dựng một ứng dụng web thực hiện chức năng **nâng cao chất lượng hình ảnh (super-resolution)**, cụ thể hơn là làm rõ các chi tiết trong hình ảnh, biến hình ảnh có độ phân giải thấp thành hình ảnh sắc nét, chi tiết hơn. Chức năng nâng cấp hình ảnh được thực hiện bằng mô hình AI sử dụng thuật toán **CNN (Convolutional Neural Network)**, một công nghệ AI mạnh mẽ trong lĩnh vực xử lý hình ảnh giúp ta xây dựng mô hình có độ chính xác cao. Thuật toán CNN sẽ được ta giải thích kĩ hơn trong phần giới thiệu mô hình.

Super-resolution là một kỹ thuật xử lý ảnh bằng cách làm sắc nét các chi tiết, giúp ảnh nhìn rõ hơn mà không nhất thiết phải tăng kích cỡ của ảnh. Đây là một công cụ rất quan trọng, đặc biệt trong thời đại ngày nay, khi ảnh kỹ thuật số được sử dụng rộng rãi trong hầu như mọi lĩnh vực. Công nghệ super-resolution được sử dụng rộng rãi trong các lĩnh vực như:

- **Thương mại điện tử:** nhu cầu sử dụng hình ảnh sắc nét để quảng cáo sản phẩm.
- **Nghệ thuật số (digital art):** nâng cao chất lượng ảnh giúp người nghệ sĩ và các nhà thiết kế tạo ra sản phẩm cuối cùng có chất lượng cao hơn.
- **Điện ảnh:** việc nâng cấp chất lượng hình ảnh có thể giúp cải thiện chất lượng video cũ, giúp chúng trở nên hấp dẫn hơn với khản giả hiện đại.

Các sản phẩm của dự án được lưu tại **link Github.**



Hình 3.1: Hình ảnh trước dự đoán (trái) và sau dự đoán (phải)

Slides PowerPoint được lưu tại: Slides
Dự án được xây dựng và chỉnh sửa từ tài liệu hướng dẫn [2].

2. Methodology: Giải thích phần xử lý dữ liệu, giới thiệu mô hình, giải thích mô hình

Quá trình thực hiện dự án sẽ được mô tả qua những bước sau:

i. Xác định mục tiêu:

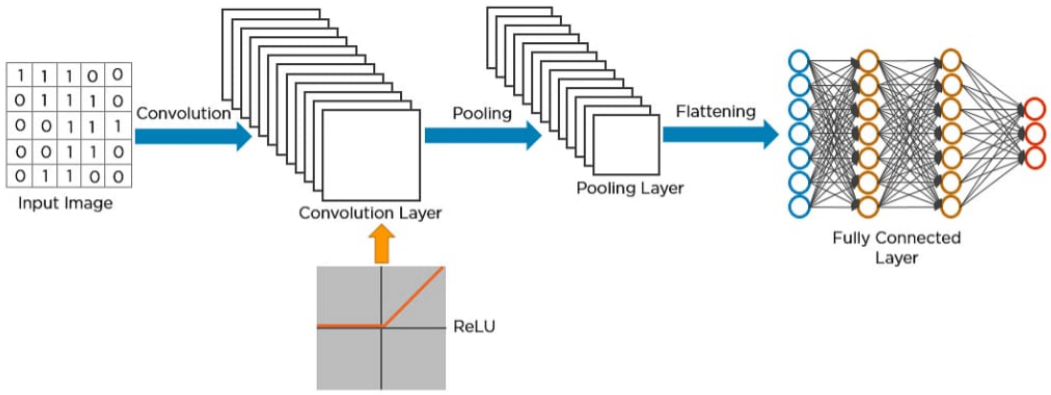
Mục tiêu của dự án là xây dựng một mô hình có khả năng làm sắc nét hình ảnh sử dụng công nghệ CNN. Sau đó áp dụng mô hình này để xây dựng một ứng dụng web đơn giản bằng Flask nhận một hình ảnh từ người dùng, thực hiện dự đoán từ mô hình và trả về hình ảnh đã được làm rõ cho người dùng.

ii. Giới thiệu mô hình:

- **Giới thiệu thuật toán CNN:**

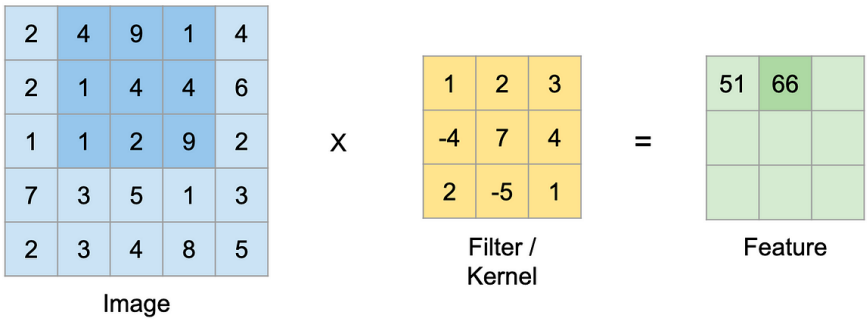
  Super-resolution là quá trình tăng độ phân giải hình ảnh bằng cách tái tạo các chi tiết bị mất. Về cơ bản, đó là quá trình phân tích và dự đoán để thêm các pixel mới vào xung quanh các pixel gốc, vì vậy nó không chỉ thay đổi kích thước mà còn thêm thông tin.

  Mô hình mạng neuron tích chập (Convolutional Neural Network - CNN) là một thuật toán chuyên xử lý và phân tích dữ liệu có dạng lưới như hình ảnh hoặc âm thanh, bao gồm nhiều lớp liên kết phức tạp, trong đó lớp quan trọng nhất là lớp tích chập. Mỗi lớp tích chập sử dụng một hay nhiều **bộ lọc (kernel)**, thường là một ma trận nhỏ 3x3 hoặc 5x5 chứa các trọng số quan trọng, để "trượt" qua toàn bộ hình ảnh để trích xuất đặc trưng (quá trình cụ thể bao gồm việc nhân từng phần tử của bộ lọc với các pixel tương ứng trong ảnh, sau đó cộng các giá trị lại). Sau khi hoàn thành việc quét qua hình ảnh sẽ tạo ra một **bản đồ đặc trưng (feature map)** thể hiện các đặc trưng mà bộ lọc phát hiện.[3]



Hình 3.2: Cấu trúc khái quát của thuật toán CNN

Ưu điểm nổi bật nhất của CNN nằm ở khả năng tự động trích xuất các đặc trưng quan trọng từ dữ liệu một cách hiệu quả bằng cách sử dụng bộ lọc kernel. Trong dự án của chúng ta, CNN chủ yếu được dùng để phân tích ảnh đầu vào, dự đoán và tái tạo các pixel bị thiếu và tạo ra ảnh có độ phân giải cao.



Hình 3.3: Kernel trong quá trình quét qua ma trận hình ảnh

Lưu ý rằng trong chương trình mà chúng ta xây dựng, ảnh đầu vào được thu nhỏ trước khi làm rõ và phóng to bằng mô hình. Điều này đảm bảo ảnh đầu ra cùng kích thước với ảnh đưa vào nhưng cũng hạn chế bớt khả năng làm rõ do với làm việc với cùng số pixel.

- **Giới thiệu khái niệm PSNR:**

  PSNR (Peak Signal-to-Noise Ratio) là thông số được dùng để đánh giá chất lượng hình ảnh tái tạo (trong một quá trình nào đó, có thể là nén, khử nhiễu, nâng độ phân giải...) so với phiên bản gốc. Cụ thể, PSNR thể hiện tỉ số giữa tín hiệu cao nhất (đại diện cho giá trị pixel lớn nhất có trong hình ảnh, đối với ảnh được chuẩn hóa sẽ là 1.0) và cường độ nhiễu (lỗi tạo ra trong quá trình xử lý) của hình ảnh tái tạo so với hình ảnh gốc. Đây là một thước đo phổ biến được dùng để đánh giá độ hiệu quả của các chương trình xử lý hình ảnh và cũng sẽ được sử dụng cho mô hình của chúng ta.

  PSNR được tính bằng công thức:

$$PSNR = 10.\log_{10}(\frac{MAX_I^2}{MSE})$$

  Trong đó:

  - MAX_I^2: giá trị pixel lớn nhất có trong hình ảnh, trong thuật toán của ta sẽ làm việc với các ảnh đã chuẩn hóa nên MAX_I^2 mang giá trị 1.

– MSE: viết tắt cho *mean squared error*, thể hiện chênh lệch bình phương trung bình giữa các pixel trong ảnh tái tạo và ảnh gốc. Thông số này sẽ được tìm hiểu kĩ hơn ở phần gọi hàm mất mát.

Từ công thức có thể thấy, khi giá trị PSNR **càng cao** thì tỉ lệ giữa cường độ tín hiệu và độ nhiễu càng lớn, nghĩa là hình ảnh tái tạo **càng gần** với ảnh gốc hơn (độ nhiễu càng nhỏ hoặc giá trị pixel trong ảnh càng lớn) và được xem là tốt hơn vì độ nhiễu có thể xem là một thước đo hữu ích cho khả năng tái tạo ảnh chất lượng cao.

iii. Xây dựng mô hình:

Phần này sẽ đi giải thích code và phân tích quá trình xây dựng và huấn luyện mô hình trong file *training/super_resolution.ipynb*, có thể theo dõi trong link Github đã gửi trên.

Thiết lập môi trường:

- **Cài đặt các công cụ cần thiết:**

  Ta bắt đầu quá trình xây dựng mô hình bằng việc kết nối với Drive, tạo thư mục của dự án và clone repository từ Github, sau đó tải các thư viện cần thiết trong file *requirements.txt*.

- **Cài đặt thư viện:**

  Import các thư viện đã tải. Các thư viện được sử dụng bao gồm: os, math, PIL (Pillow), tensorflow (bao gồm Keras), numpy, matplotlib, keras.applications.vgg16, mpl_toolkits.axes_grid1, IPython.display.

Thu thập và xử lý dữ liệu:

- **Thu thập dữ liệu:**

  Dữ liệu trong dự án bao gồm 500 hình ảnh được tải từ nguồn *Berkeley Segmentation Dataset 500 (BSDS500)* [1]. Đây là một tập dữ liệu hình ảnh phổ biến thường được dùng trong lĩnh vực thị giác máy tính.

Các file được tải về và giải nén, sau đó được lưu và tạo đường dẫn thích hợp bằng root_path.

- **Phân loại dữ liệu:**

  Ta quy định các thông số quan trọng, bao gồm: kích thước chuẩn hóa ảnh đầu vào input_size và ảnh đầu ra target_size mặc định lần lượt là 100x100 và 300x300, với hệ số phóng đại scale_factor tương ứng là 3. Thông số batch_size là 8 quy định số lượng ảnh được xử lý trong một lô. Việc quy định các thông số này ngay từ ban đầu giúp việc xử lý hình ảnh của ta được thống nhất và hiệu quả hơn.

  Dữ liệu hình ảnh khi tải về gồm các tệp khác nhau. Để tiện, ta sẽ gộp hết dữ liệu có sẵn và phân chia lại làm dữ liệu huấn luyện (training dataset) và dữ liệu kiểm tra (validation dataset). Dữ liệu đã phân loại đều được chuẩn hóa về kích thước 300x300 như đã nói trên. Tỉ lệ giữa dữ liệu huấn luyện và kiểm tra là 4:1 (400 ảnh huấn luyện và 100 ảnh kiểm tra).

- **Xử lý dữ liệu:**

  Dữ liệu đã được phân chia tiếp tục được xử lý qua những bước sau:

  - **Chuẩn hóa pixel:** Chuẩn hóa giá trị pixel của mỗi hình ảnh từ [0, 255] (do là hình ảnh RGB) về [0, 1].
  - **Tiền xử lý chi tiết:**

    Ở đây ta sẽ định nghĩa hai hàm quan trọng là process_input và process_target. Hàm process_input chuyển đổi ảnh từ không gian màu RGB sang YUV, trong đó Y là kênh độ sáng, chứa phần lớn thông tin chi tiết của ảnh và hai kênh U, V chứa thông tin màu sắc. Hàm thực hiện tách và trả về kênh Y của hình ảnh sau khi thực hiện thêm bước chỉnh lại kích thước hình theo kích cỡ đưa vào. Hàm process_target tương tự như process_input nhưng không có bước chỉnh lại kích thước.

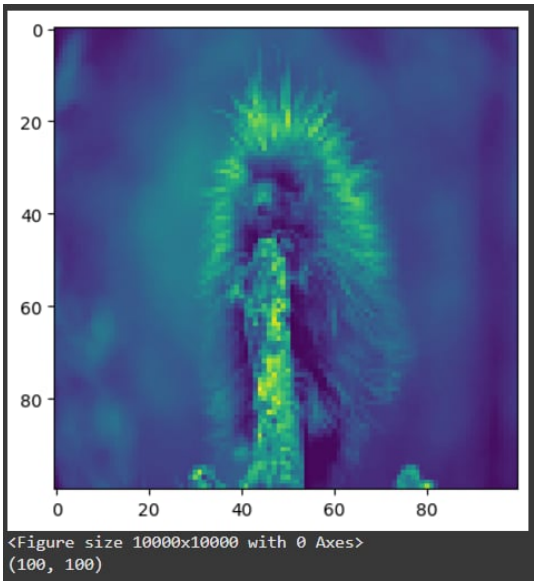
    Vai trò của hai hàm process_input và process_target là biến hình ảnh đã có thành hình ảnh đầu vào có kích thước 100x100 và hình ảnh đầu ra 300x300, cả hai hình ảnh chỉ giữ lại kênh Y

chứa những chi tiết quan trọng và bỏ các kênh màu không cần thiết.

Ta thực hiện áp dụng hai hàm lên từng hình ảnh trong dữ liệu đã có và phân mỗi ảnh thành một ảnh đầu vào thu nhỏ và một ảnh đầu ra tiêu chuẩn để phục vụ cho việc huấn luyện. Đồng thời tăng tốc độ nạp dữ liệu bằng hàm prefetch (nạp trước 32 mẫu dữ liệu trong bộ nhớ). Kết quả ta có hai bộ dữ liệu huấn luyện train_ds và dữ liệu kiểm tra valid_ds đã được xử lý.

- **Hiển thị và kiểm tra:**

  Ở bước này, ta dùng thư viện **matplotlib** để hiển thị hình ảnh đầu tiên từ tập dữ liệu huấn luyện train_ds dưới dạng một ảnh đầu vào thu nhỏ 100x100 và một ảnh đầu ra 300x300 kèm theo khung **figure** và kích cỡ ảnh.



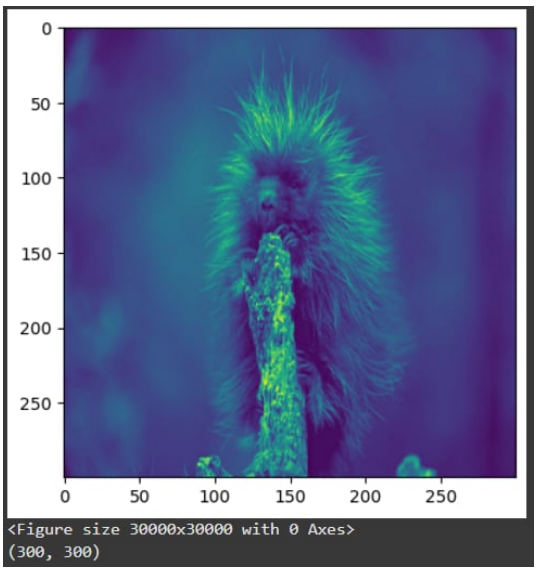
Hình 3.4: Ảnh đầu vào tiêu chuẩn

Xây dựng và huấn luyện mô hình:

- **Xây dựng mô hình cơ sở:**

  Trong phần này, ta xây dựng một mô hình học sâu cơ sở để tiến hành huấn luyện. Cụ thể như sau:

  - **Xây dựng lớp DepthToSpaceLayer:**



Hình 3.5: Ảnh đầu ra tiêu chuẩn

Lớp DepthToSpaceLayer là một **lớp tùy chỉnh (custom layer)**, được xây dựng bằng cách kế thừa lớp cơ bản tensorflow.keras.lay--ers.Layer trong Keras. Mục đích chính của hàm là biến đổi thông tin trong các chiều dữ liệu trong các **tensor** (tên gọi cấu trúc mảng trong tensorflow) từ chiều sâu (chiều thông tin) sắp xếp lại thành chiều không gian (chiều dài và chiều rộng của ảnh). Nói đơn giản hơn, hàm thực hiện phóng to hình ảnh theo một tỉ lệ block_size bằng cách giải nén thông tin từ các kênh chiều sâu, kết quả là một bức ảnh được phóng to về không gian mà tổng số dữ liệu vẫn không thay đổi.

– **Xây dựng hàm tạo mô hình:**

Hàm này tiến hành xây dựng một mô hình học sâu cơ sở sử dụng công nghệ **CNN (mạng neuron tích chập)** mà ta đã giải thích ở phần giới thiệu mô hình qua hàm get_model.

Biến conv_args chứa các thông số cài đặt chung cho các lớp tích chập, cụ thể là:

∗ **activation = "relu":** Hàm kích hoạt relu (Rectified Linear Unit) là một hàm kích hoạt phổ biến trong các mạng neuron học sâu, được định nghĩa bằng công thức:

$$f(x) = max(0, x)$$

Nghĩa là nếu giá trị đầu vào là âm, hàm sẽ trả về 0, nếu là dương thì giữ nguyên. Vai trò của hàm là giúp mô hình học được các mẫu phức tạp bằng cách thêm tính phi tuyến vào mô hình.

* **kernel_initializer = "Orthogonal":** Các trọng số ban đầu của các lớp tích chập được khởi tạo theo phương pháp **Orthogonal**, các ma trận trọng số được tạo ra có đặc điểm là các hàng và cột vuông góc với nhau. Điều này giúp cải thiện hiệu suất học và giúp mô hình hội tụ nhanh hơn khi huấn luyện.
* **padding = "same":** Kích thước đầu ra được giữ nguyên so với kích thước đầu vào sau khi thực phép tích chập (bằng cách thêm các pixel giá trị 0 vào viền ảnh đầu ra).

Tiếp đến là luồng xử lý dữ liệu qua mô hình. Đầu vào inputs có chiều dài và chiều rộng bất kì nhưng số kênh cố định là 1. Sau đó dữ liệu đầu vào được đưa qua một loạt các lớp tích chập được thiết kế với các thông số trong conv_args:

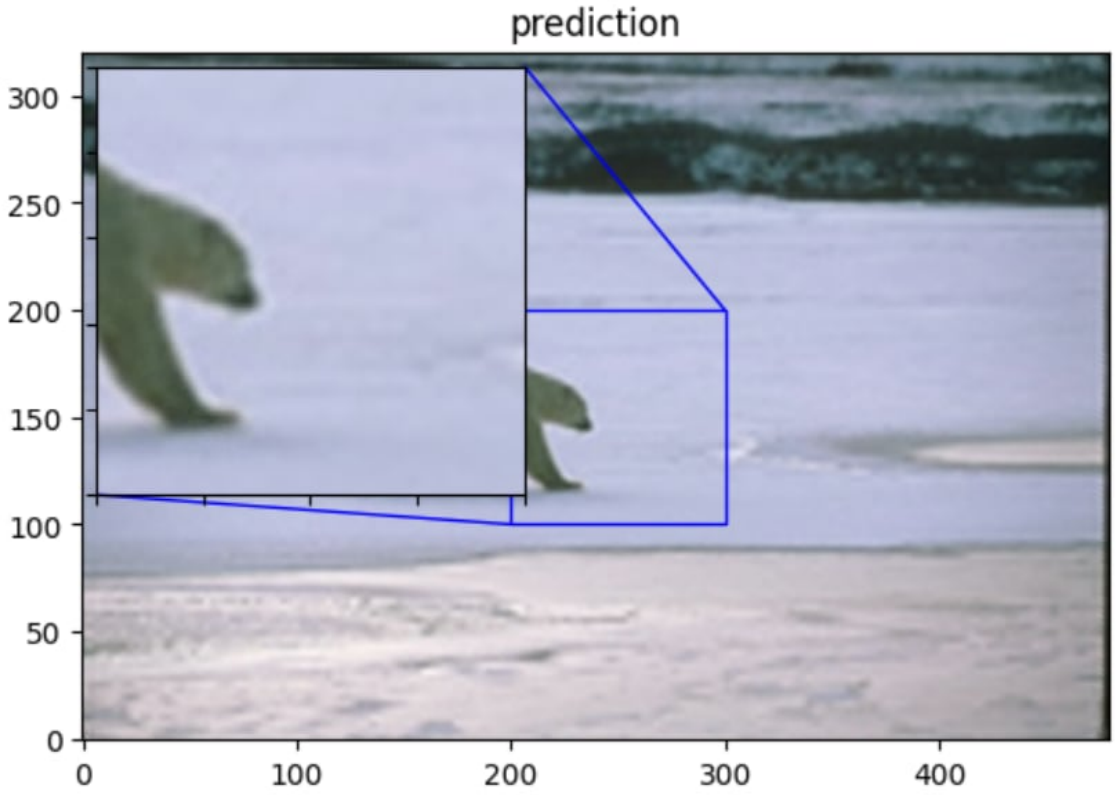
* **Lớp 1:** Có 64 kernel lớn 5x5 để thu được nhiều thông tin cơ bản từ ảnh đầu vào.
* **Lớp 2 và 3:** Lần lượt có 64 và 32 kernel kích thước 3x3 để học chi tiết hơn từ các đặc trưng đã trích xuất.
* **Lớp 4:** Lớp quan trọng chuẩn bị dữ liệu cho lớp DepthToSpaceLayer mà ta đã tạo trên. Tạo ảnh đầu ra với số kênh thích hợp dựa trên upscale_factor và channels cho việc phóng đại hình ảnh (ở đây là 9 kênh).

Kết quả cuối cùng sau khi qua các lớp tích chập chứa các đặc trưng quan trọng trong các kênh chiều sâu. Được tái tổ chức qua DepthToSpaceLayer từ chiều sâu sang chiều không gian, tạo hình ảnh có kích thước lớn hơn.

Như vậy qua hàm get_model ta có một mô hình học sâu sử dụng công nghệ CNN đã được định nghĩa đầy đủ, cho ra một ảnh có độ phân giải cao từ ảnh độ phân giải thấp. Mô hình đóng vai trò là mô

hình cơ sở để tiến hành huấn luyện và tìm ra các trọng số tối ưu cho các lớp tích chập.

- **Xây dựng công cụ theo dõi và đánh giá quá trình huấn luyện:**
  - **Xây dựng các hàm cần thiết:**
    - **plot_results(img, prefix, title):**
      Mục đích chính của hàm là hiển thị hình ảnh với một vùng phóng to để dễ dàng theo dõi độ hiệu quả của mô hình trong quá trình huấn luyện. Hàm thực hiện từng bước từ việc chuyển đổi hình ảnh thàng mảng số, chuẩn hóa giá trị pixel tới việc tạo các khung phóng đại và vẽ khung liên kết bằng mark_inset.



Hình 3.6: Hình ảnh được hiển thị

    - **get_lowres_image(img, upscale_factor):**]
      Hàm nhận một hình ảnh gốc và tạo một ảnh có độ phân giải thấp dùng làm đầu vào cho mô hình, sử dụng phương pháp nội suy song khối của PIL để giữ chất lượng ảnh sau

khi thu nhỏ tốt nhất có thể. Ảnh thu nhỏ có kích cỡ bằng 1/upscale_factor ảnh gốc (ở đây là 1/3).

* **upscale_image(model, img):**
  Đây là một hàm quan trọng thực hiện việc áp dụng mô hình để thực hiện việc nâng cao chất lượng hình ảnh. Hàm thực hiện bằng cách chuyển hình ảnh đầu vào sang không gian màu **YCbCr**, tách kênh Y chứa thông tin độ sáng ra khỏi các kênh màu sắc không quan trọng. Sau đó kênh Y được chuyển từ hình ảnh thành mảng số và được chuẩn hóa pixel về thang [0, 1] trước khi đưa qua mô hình để thực hiện dự đoán. Kết quả trả về từ mô hình được chuyển pixel lại về thang [0, 255] và được chuyển từ mảng số thành hình ảnh. Các kênh Cb và Cr được chỉnh lại kích cỡ bằng nội suy song khối để phù hợp với kích thước đầu ra của kênh Y, sau đó kết hợp lại để chuyển đổi về không gian RGB.
  Kết quả trả về là một ảnh đã được làm rõ trong không gian màu RGB.

– **Xây dựng các cơ chế kiểm soát quá trình huấn luyện:**
  Trong phần này ta sẽ xây dựng các hàm kiểm soát để phục vụ cho quá trình huấn luyện mô hình, thường được gọi là các **callback** trong Keras:

  * **ESPCNCallback:**
    Lớp ESPCNCallback dùng để tích hợp các hàm trên và theo dõi độ hiệu quả của mô hình trong quá trình huấn luyện. Bản chất của lớp là một tùy chỉnh của lớp Callback trong Keras.
    Ta bắt đầu bằng việc thiết lập đường dẫn đến thư mục *test* chứa các ảnh kiểm tra trong biến test_img_paths. Các hình ảnh trong thư mục được sắp xếp theo tên để đảm bảo thứ tự xử lý được nhất quán.
    Lớp ESPCNCallback của chúng ta bao gồm những hàm quan trọng sau:

    · **__init__**:

Kế thừa lớp keras.callbacks.Callback, đồng thời lấy ảnh từ thư mục test_img_paths và thu nhỏ bằng hàm get_lowres_ image làm ảnh kiểm tra độ hiệu quả của mô hình.

- **on_epoch_begin**:
  Được gọi tại đầu mỗi **epoch** (mỗi lần mô hình được huấn luyện qua toàn bộ dữ liệu) để tạo danh sách lưu các giá trị PSNR. Định nghĩa PSNR đã được nói kĩ ở phần giới thiệu khái niệm PSNR.
- **on_test_batch_end**:
  Được gọi sau mỗi **batch** (số lượng ảnh được huấn luyện trước mỗi lần cập nhật trọng số) để tính và lưu giá trị PSNR cho batch đó vào danh sách đã tạo trên.
- **on_epoch_end**:
  Được gọi tại cuối mỗi epoch, có nhiệm vụ tính và hiển thị giá trị PSNR trung bình cho toàn bộ epoch. Đồng thời cứ sau mỗi 20 epoch thì sẽ thực hiện chạy mô hình trên ảnh kiểm tra bằng hàm upscale_image và hiển thị kết quả bằng plot_results để trực quan hóa kết quả, giúp người dùng quan sát được mức độ hiệu quả hiện tại của mô hình.

* **early_stopping_callback:**
  Dừng việc huấn luyện sớm nếu mô hình không có cải thiện như mong muốn khi huấn luyện. Cụ thể, hàm theo dõi giá trị của hàm mất mát loss và nếu nó không giảm sau 10 epochs liên tiếp, quá trình sẽ dừng lại để tránh việc phí thời gian huấn luyện vô ích.
* **model_checkpoint_callback:**
  Lưu lại trạng thái của mô hình vào đường dẫn filepath sau mỗi epoch nếu mô hình đạt được hiệu quả tốt nhất (dựa trên kết quả hàm loss).

- **Huấn luyện mô hình:**

Sau khi đã có đủ dữ liệu huấn luyện và các công cụ cần thiết, ta thực hiện bước tổng kết cuối cùng và bắt đầu huấn luyện mô hình:

- **Tổng kết các công cụ cần thiết:**
  - Lấy mô hình cơ sở từ hàm get_model và hiển thị thông tin về mô hình qua hàm summary.
  - Tổng hợp các callback đã xây dựng trong biến callbacks, bao gồm ESPCNCallback, early_stopping_callback và model_checkpoint_callback.
  - Lấy hàm mất mát từ Keras để tính toán **độ sai lệch** giữa ảnh dự đoán từ mô hình và ảnh gốc với công thức:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

    Trong đó:
    - MSE (Mean Squared Error): Giá trị đo sai số bình phương trung bình giữa giá trị thực và giá trị dự đoán.
    - N: Số lượng mẫu dữ liệu.
    - y_i: Giá trị thực tại mẫu i.
    - $\hat{y}_i$: Giá trị dự đoán tại mẫu i.
  - Lấy bộ tối ưu hóa **Adam (Adaptive Moment Estimation)** áp dụng cho mô hình huấn luyện. Đây là thuật toán giúp mô hình "học" từ các dữ liệu bằng cách điều chỉnh tham số. Tốc độ học được quy định là 0.001.
- **Huấn luyện và lưu mô hình:**
  - **Biên dịch mô hình:** Để chuẩn bị cho mô hình trước khi huấn luyện. Mô hình được biên dịch với bộ tối ưu hóa Adam và hàm mất mát như đã nêu trên.
  - **Huấn luyện mô hình:** Mô hình được huấn luyện trên tập dữ liệu train_ds và valid_ds với các callback đã quy định và số epoch bằng 300 (quá trình huấn luyện thực tế được

dừng sớm hơn). Chỉ số verbose=2 quy định hiển thị các chỉ số huấn luyện như loss và val_loss sau mỗi epoch.

* **Lưu các trọng số:**
  Trọng số tốt nhất được lưu lại trong quá trình huấn luyện được chọn ra và tải lên mô hình hiện có. Mô hình giờ đây đã hoàn thành quá trình huấn luyện.

– **Lưu mô hình:**

Tạo đường dẫn *Models/image_upscale_model* và lưu mô hình với toàn bộ trọng số và thông tin cấu hình vào file *my_model.keras*. Mô hình đã lưu có thể được tái sử dụng và áp dụng vào các hệ thống khác mà không cần phải huấn luyện lại từ đầu.

Chạy và nhận xét độ hiệu quả của mô hình:

Ta tiến hành áp dụng mô hình đã lưu với một số hình ảnh trong tệp dữ liệu kiểm tra để đánh giá mức độ hiệu quả của mô hình. Đoạn code ở phần này sẽ đi so sánh chỉ số PSNR giữa ảnh thu nhỏ bằng hàm get_lowres_image sử dụng phương pháp nội suy song khối và ảnh gốc với chỉ số PSNR giữa ảnh dự đoán từ mô hình và ảnh gốc. Nếu PSNR của mô hình lớn hơn của nội suy song khối thì được xem là hoạt động tốt hơn.

Để chuẩn bị, ta tải mô hình từ đường dẫn đã lưu với hàm load_model, đồng thời tạo thư mục *predictions* để chứa các dự đoán của mô hình. Hai biến total_bicubic_psnr và total_test_psnr lần lượt chứa tổng PSNR của nội suy song khối và mô hình sau quá trình chạy thử.

Ta tiến hành chạy với một số ảnh trong tệp dữ liệu kiểm tra. Quá trình chạy bao gồm việc áp dụng hàm get_lowres_image lên ảnh gốc để lấy ảnh thu nhỏ và upscale_image lên ảnh thu nhỏ để lấy dự đoán của mô hình. Các ảnh được chỉnh kích cỡ cho bằng ảnh gốc và tiến hành đi tính chỉ số PSNR. Chỉ số PSNR của mỗi lần chạy được in ra màn hình, cũng như được lưu lại để tính giá trị trung bình cho cả quá trình chạy.

Mỗi hình ảnh được chọn để đi thu nhỏ bằng nội suy song tuyến và dự đoán bằng mô hình đều được lưu lại trong thư mục *predictions* để dễ dàng quan sát.

Đó là cách mà chúng ta sẽ chạy thử mô hình. Kết quả cụ thể cũng như số ảnh thử nghiệm sẽ được nói rõ hơn trong phần Experiment.

Tổng kết thuật toán:

Quá trình làm rõ ảnh bằng mô hình được tóm tắt lại như sau:

- **Chuẩn bị ảnh đầu vào:**
  - Ảnh đầu vào được thu nhỏ bằng hàm get_lowres_image sử dụng nội suy song khối. Ảnh thu nhỏ có kích cỡ bằng 1/3 ảnh ban đầu.
  - Ảnh được chuyển không gian màu từ RGB sang YCbCr và tách kênh Y (độ sáng) ra khỏi kênh Cb, Cr, chuyển thành ma trận số rồi chuẩn hóa từ [0, 255] về [0, 1] làm dữ liệu đầu vào cho mô hình.
- **Mô hình thực hiện dự đoán:**
  - Dữ liệu được đưa vào mô hình, đi qua nhiều lớp tích chập Conv2D, mỗi lớp sử dụng bộ lọc (kernel) để trích xuất đặc trưng quan trọng.
  - Cuối quá trình, dữ liệu được đưa qua lớp DepthToSpaceLayer để tái tổ chức thông tin đặc trưng từ các kênh chiều sâu thành chiều không gian. Dữ liệu đầu ra của mô hình là ảnh (dưới dạng mảng số) có độ phân giải gấp 3 lần dữ liệu ban đầu.
- **Xử lý kết quả mô hình:**
  - Kết quả mảng số từ mô hình được chuyển ngược về thang [0, 255] và chuyển thành ảnh.
  - Hai kênh màu Cb, Cr được phóng to bằng nội suy song khối và hợp lại với kênh Y và chuyển lại về ảnh RGB.
  - Kết quả cuối cùng của thuật toán là ảnh có cùng kích thước với ảnh đầu vào nhưng với chi tiết rõ nét hơn nhờ mô hình.

- **Đánh giá:** Độ hiệu quả của mô hình được đánh giá bằng cách so sánh chỉ số PSNR giữa ảnh dự đoán của mô hình với ảnh gốc và chỉ số PSNR giữa ảnh dự đoán bằng nội suy song khối (Bicubic Interpolation) với ảnh gốc. PSNR thì ảnh càng giống với ảnh gốc.

iv. Xây dựng ứng dụng web:

Sau khi đã có được mô hình, ta tiến hành xây dựng ứng dụng web bằng framework Flask. Phần này sẽ đi giải thích cách hoạt động của các file *app.py* và *templates/base.html*, *templates/index.html*.

Tạo file app.py:

- **Sử dụng framework Flask để xây dựng ứng dụng web:**
  - **app = Flask(__name__):** Khởi tạo ứng dụng Flask.
  - **app.secret_key = '24TNT1_nhom1':** Thiết lập secret key cho ứng dụng Flask.
- **Xây dựng các hàm liên quan:**
  - **upscale_model:** Hàm sử dụng mô hình để nâng cao độ phân giải của ảnh. Hàm có cấu trúc và chức năng tương tự với hàm upscale_image được đề cập ở phần xây dựng các hàm cần thiết.
  - **get_lowres_image:** Hàm nhận một hình ảnh và trả về hình ảnh có độ phân giải thấp để làm đầu vào cho mô hình. Hàm get_lowres_image cũng đã được mô tả ở phần xây dựng các hàm cần thiết.
  - **upscale_multiple_times:** Hàm sử dụng mô hình để nâng cao độ phân giải của ảnh nhiều lần bằng cách sử dụng hàm model_upscaling đã được định nghĩa ở trên.
  - **allowed_file:** Kiểm tra file được nhập có định dạng hợp lệ với các loại định dạng được khai báo bao gồm **png**, **jpg**, **jpeg** hay không.

- **index:** Trả về file 'index.html' hiển thị giao diện người dùng và **@app.route('/'):** Định nghĩa **route** (đường dẫn) cho trang chính của ứng dụng.
- **process_image:** Đọc, kiểm tra, xử lý ảnh bằng mô hình và hiển thị trên giao diên người dùng bao gồm ảnh gốc, ảnh sau khi xử lý và độ phân giải của ảnh.

- **Các phương thức được sử dụng: GET** và **POST**
  - Khi phương thức **POST** được gọi:
    * Kiểm tra file có nằm trong request.file và hợp lệ hay không. Nếu không hợp lệ thì gửi thông báo bằng flash và gửi về đường dẫn.
    * Tạo thư mục lưu trữ: Dùng os.makedirs để đảm bảo thư mục static/uploads tồn tại trước khi lưu file.
    * Xử lí ảnh: Tạo ảnh có độ phân giải thấp. Tải mô hình và lớp tuỳ chỉnh DepthToSpaceLayer. Tăng chất lượng ảnh bằng hàm upscale_multiple_times đã được định nghĩa từ trước. Ảnh gốc và ảnh sau khi được xử lý được lưu vào thư mục static/upload.
    * Trả về giao diện người dùng: Tải giao diện 'index.html' và hiển thị trên trình duyệt ảnh gốc, ảnh đã được nâng cao chất lượng và độ phân giải của ảnh.
    * Nếu trong quá trình xử lý xảy ra lỗi thì hiển thị thông báo lỗi bằng flash và chuyển người dùng về trang hiện tại.
  - Khi phương thức **GET** được gọi: Chỉ hiển thị giao diện người dùng mà không xử lý upload.

Tạo file custom_layers.py:

File này chứa lớp DepthToSpaceLayer tương tự như định nghĩa ở trên. Việc tạo một file riêng cho lớp tùy chỉnh giúp tiện cho việc import và xây dựng ứng dụng web hơn.

Tạo thư mục templates:

Thư mục **templates** chứa hai file là **base.html** và **index.html** với mục đích thiết kế giao diện cho ứng dụng web, bao gồm những thành phần cơ bản như **thanh điều hướng (navbar)**, **nội dung chính** và **nơi hiển thị thông báo.**

Tạo file base.html:

Đảm bảo giao diện nhất quán khi file **index.html** kế thừa và có thể tái sử dụng lại đoạn mã mà không viết lại đoạn mã đó nhiều lần.

- **Thẻ head:** có vai trò chính là đặt tiêu đề cho trang web, ở đây tiêu đề là *Super-resolution*, có thêm phần {% block title %}{% endblock title %} cho phép kế thừa và thay đổi tiêu đề từ trang kế thừa.
- **Thẻ body:** gồm ba phần chính đó là nav thanh điều hướng, Flash messages thông báo và main phần chứa nội dung chính của trang.
  - Thanh điều hướng: có vị trí dưới phần head và trên giao diện chính của trang, khi người dùng cuộn xuống thì thanh điều hướng vẫn giữ nguyên vị trí của nó.
  - Thông báo: hiển thị các thông điệp ngắn gọn cho người dùng, thường được dùng để thông báo thành công, hay lỗi khi người dùng thực hiện hành động với trang web.
  - nội dung chính của web: {% block title %}{% endblock title %} cho phép trang con thay đổi hoặc mở rộng nội dung trong phần body.

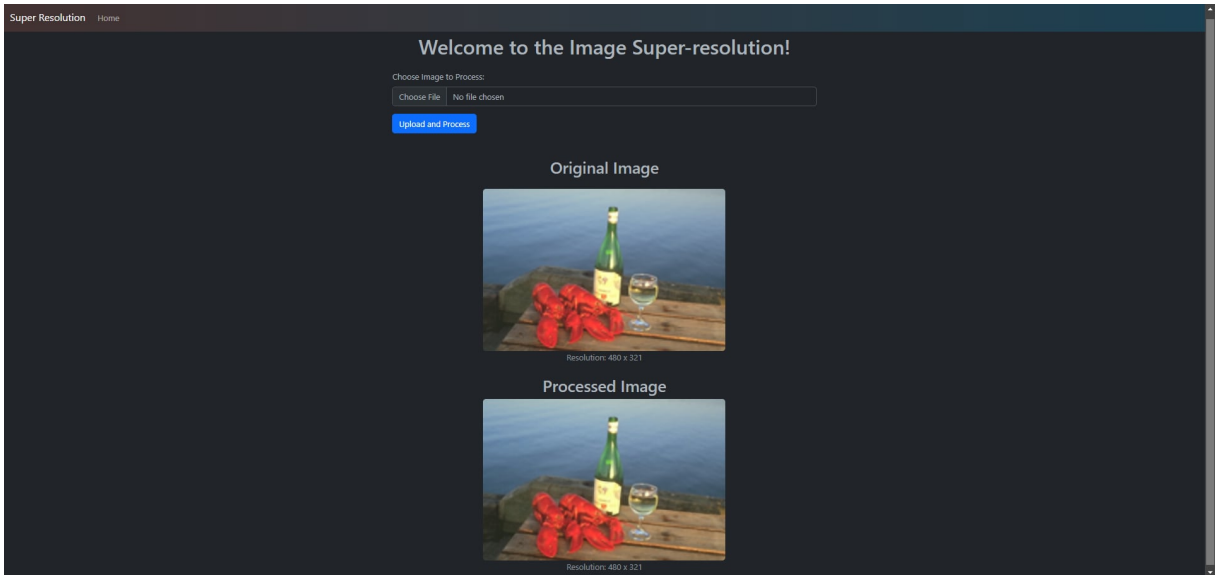
Tạo file index.html:

Mục đích: Tạo giao diện để hiển thị kết quả bao gồm ảnh gốc và ảnh sau khi được nâng cấp chất lượng. File **index.html** sử dụng kế thừa từ file base.html và được cấu thành chính từ ba khối:

- **Khối tiêu đề (block title):** Hiển thị tiêu đề của trang.

- **Khối điều hướng (block nav):** Tạo thanh điều hướng (class="nav-item active") và sử dụng hàm url_for để tạo liên kết đến hàm index trong file Flask app.py (href="{{ url_for('index') }}").
- **Khối thân (block body):**
  - Container chính căn giữa nội dung.
  - Phần hiển thị thanh tiêu đề của giao diện.
  - Cho phép người dùng tải ảnh lên để xử lý ảnh:
    * **action="{{ url_for('process_upload') }}":** định nghĩa đường dẫn nơi dữ liệu được gửi đến, sử dụng hàm url_for để tạo URL.
    * **method="post":** chỉ định phương thức HTTP là **POST**.
    * **<input type="file" id="formFile">:** Định dạng input để người dùng chọn tệp và gán ID để nhận diện input này.
    * **<button type="submit">:** Tạo nút gửi form.
  - Hiển thị ảnh gốc, ảnh sau khi chỉnh sửa và độ phân giải của chúng:
    * **if original_image_url and prediction_image_url:** Kiểm tra điều kiện tồn tại url của ảnh gốc và ảnh sau khi xử lý. Nếu cả 2 url tồn tại, khối HTML bên trong sẽ được thực thi.
    * **src="{{ original_image_url }}"** và **src="{{ prediction_image_url }}":** Đường dẫn đến ảnh gốc và ảnh đã được nâng chất lượng.
    * **Resolution: {{ original_image.width }} x {{ original_image.height }}:** Hiển thị độ phân giải ảnh gốc (width x height), được truyền từ server thông qua biến original_image.
    * Độ phân giải ảnh sau khi xử lý cũng được gọi tương tự, được truyền từ server thông qua biến processed_image.

* **%endif:%**: kết thúc điều kiện if.



Hình 3.7: Giao diện chính thức của ứng dụng web

3. Experiment

Quá trình cụ thể sẽ được chiếu trong video dự án.

Giới thiệu:

- **Mục tiêu:** Kiểm tra và đánh giá mức độ hiệu quả của thuật toán và mô hình. Các thông số được dùng để đánh giá bao gồm PSNR, loss (dựa trên dữ liệu huấn luyện) và val_loss (dựa trên dữ liệu kiểm tra)
- **Môi trường thực thi:**
  - **Phần cứng:** Project được chạy trên nền tảng Google Colaboratory, sử dụng runtime T4-GPU.

- **Framework và thư viện chính:** Tensorflow/Keras, NumPy, PIL (Pillow), Matplotlib, IPython, OS, Math.

Quá trình huấn luyện mô hình:

- **Kiến trúc mô hình**: Mô hình super-resolution sử dụng CNN.
- **Mô tả Dataset:** Dữ liệu 500 hình ảnh được tải từ nguồn [1]. Trong đó dữ liệu huấn luyện và kiểm tra được chia tỉ lệ 4:1 (400 hình ảnh huấn luyện và 100 hình ảnh kiểm tra).
- **Cấu hình tham số:**
  - **Kích cỡ batch:** 8.
  - **Epochs:** 300 (có sử dụng cơ chế dừng sớm nếu hiệu quả mô hình không thay đổi sau 10 epochs).
  - **Learning rate:** 0.001.
  - **Optimizer:** Adam (Adaptive Moment Estimation).
  - **Hàm mất mát:** Mean Squared Error.
- **Kết quả huấn luyện:** Quá trình huấn luyện dừng lại sau **276 epochs**. Chỉ số loss và val_loss tốt nhất được đo lần lượt là 0.0023 và 0.0022.

Quá trình chạy mô hình:

- **Đối tượng chạy thử:** 10 hình ảnh (hình 50 - 59) trong dữ liệu đã tải về (đường dẫn cụ thể: */root/.keras/datasets/BSR/BSDS500/data/i--mages/test*). PSNR của mô hình được so sánh với PSNR của nội suy song khối so với ảnh gốc.
- **Kết quả chạy thử:**
  - **Ảnh 50:**
    * PSNR của nội suy song khối với ảnh gốc: 30.0157
    * PSNR của mô hình với ảnh gốc: 30.8533

- **Ảnh 51:**
  - PSNR của nội suy song khối với ảnh gốc: 25.1103
  - PSNR của mô hình với ảnh gốc: 26.3596
- **Ảnh 52:**
  - PSNR của nội suy song khối với ảnh gốc: 27.7789
  - PSNR của mô hình với ảnh gốc: 28.5077
- **Ảnh 53:**
  - PSNR của nội suy song khối với ảnh gốc: 28.0321
  - PSNR của mô hình với ảnh gốc: 28.3142
- **Ảnh 54:**
  - PSNR của nội suy song khối với ảnh gốc: 25.7853
  - PSNR của mô hình với ảnh gốc: 26.5055
- **Ảnh 55:**
  - PSNR của nội suy song khối với ảnh gốc: 25.9181
  - PSNR của mô hình với ảnh gốc: 26.8834
- **Ảnh 56:**
  - PSNR của nội suy song khối với ảnh gốc: 26.2389
  - PSNR của mô hình với ảnh gốc: 27.3186
- **Ảnh 57:**
  - PSNR của nội suy song khối với ảnh gốc: 23.3281
  - PSNR của mô hình với ảnh gốc: 25.0086
- **Ảnh 58:**
  - PSNR của nội suy song khối với ảnh gốc: 29.9008
  - PSNR của mô hình với ảnh gốc: 30.1139
- **Ảnh 59:**
  - PSNR của nội suy song khối với ảnh gốc: 25.2492
  - PSNR của mô hình với ảnh gốc: 25.8482
- **Tổng kết:**

* PSNR trung bình của nội suy song khối với ảnh gốc: 26.7357
* PSNR trung bình của mô hình với ảnh gốc: 27.5713

Quá trình chạy web:

Được biểu diễn trong Video .

4. Kết luận:

Như vậy qua kết quả huấn luyện, ta thấy mô hình đã học được tương quan giữa ảnh gốc và ảnh độ phân giải thấp. Kết quả chạy thử cho ta thấy chỉ số PSNR trung bình của mô hình cao hơn so với của nội suy song khối. Điều này chứng minh rằng mô hình đã cải thiện được chất lượng hình ảnh tái tạo và mang kết quả vượt trội hơn so với phương pháp nội suy truyền thống. Việc tích hợp mô hình để xây dựng ứng dụng web đã được chạy thử nghiệm và thành công, cho ta quan sát được ứng dụng của mô hình trong việc nâng cao chất lượng hình ảnh.

Tuy nhiên, mô hình vẫn còn có nhiều hạn chế. Qua quan sát trực quan có thể thấy ảnh đã được cải thiện từ ảnh độ phân giải thấp vẫn chưa đạt độ sắc nét của ảnh gốc. Đặc biệt ảnh còn gặp nhiều khó khăn với những ảnh có nhiều chi tiết nhỏ, ảnh có nhiễu và ảnh có độ phân giải quá thấp (ảnh quá mờ)...

Có thể tiếp tục cải thiện mô hình bằng cách tối ưu hóa thuật toán, cải thiện số lượng và chất lượng dữ liệu huấn luyện hoặc thử nghiệm các cấu trúc CNN sâu hơn.

Chương 4

Báo cáo làm việc hàng tuần

<table>
<tr><th>Ngày họp</th><th>Thành viên tham dự</th><th>Phân công</th></tr>
<tr><td>01/12/2024</td><td>Lâm, Phú, Huy, Phát, Tuấn, Tú</td><td>Lâm: Làm Slides giới thiệu đồ án
Phú: Lập trình + Báo cáo đồ án
Huy: Lập trình + Báo cáo đồ án
Phát: Làm Video giới thiệu đồ án
Tuấn: Lập trình + Báo cáo đồ án
Tú: Lập trình + Báo cáo đồ án</td></tr>
<tr><td colspan="3">Thời gian: 01-10/12/2024 Tiến độ: 25%
Công việc đã làm:
- Xác định mục tiêu, phân công công việc, thời hạn ☑
- Lập trình mô hình A.I ☑
- Báo cáo các mục:
+ Introduction ✖
+ Methodology ✖
+ Conclusion ✖
- Hoàn thiện phần mềm, tải lên github sản phẩm hoàn chỉnh ✖
- Powerpoint thuyết minh ✖
- Video giới thiệu và demo đồ án ✖</td></tr>
<tr><td>18/12/2024</td><td>Lâm, Phú, Huy, Phát, Tuấn, Tú</td><td>Lâm: Làm Slides giới thiệu đồ án
Phú: Lập trình + Báo cáo đồ án
Huy: Lập trình + Báo cáo đồ án
Phát: Làm Video giới thiệu đồ án
Tuấn: Lập trình + Báo cáo đồ án
Tú: Lập trình + Báo cáo đồ án</td></tr>
<tr><td colspan="3"></td></tr>
</table>

Thời gian: 01-10/12/2024 Tiến độ: 100%

Công việc đã làm:

- Xác định mục tiêu, phân công công việc, thời hạn ☑
- Lập trình mô hình A.I ☑
- Báo cáo các mục: ☑

+ Introduction ☑

+ Methodology ☑

+ Conclusion ☑

- Hoàn thiện phần mềm, tải lên github sản phẩm hoàn chỉnh ☑
- Powerpoint thuyết minh ☑
- Video giới thiệu và demo đồ án ☑

Tài liệu tham khảo

Tiếng Anh

[1] *Dữ liệu hình ảnh.* URL: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz.

[2] *File Github của mô hình mẫu.* URL: https://github.com/Antverse1/Image-Upscaling-Using-CNN.

[3] *Link mô tả cách hoạt động của CNN.* URL: https://vietnix.vn/cnn-la-gi/.