
ĐẠI HỌC QUỐC GIA TP.HCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

---

# BÁO CÁO ĐỒ ÁN
# Tiền Xử Lý Dữ Liệu

---

**Môn học: Khai thác dữ liệu và ứng dụng**

*Sinh viên thực hiện:*

Quách Châu Hạo Kiệt – 23127078

Nguyễn Trần Minh Thư – 22127403

Vũ Anh – 23127321

*Giáo viên hướng dẫn:*

Lê Hoài Bắc

Nguyễn Ngọc Đức

Lê Nhựt Nam

TP. Hồ Chí Minh – 2025

# Mục lục

# 1 GIỚI THIỆU

Trong lĩnh vực Khoa học Dữ liệu, bước tiền xử lý (*Data Preprocessing*) giữ vai trò nền tảng, quyết định chất lượng và hiệu quả của toàn bộ mô hình học máy. Việc làm sạch, chuẩn hóa và mã hóa dữ liệu giúp giảm nhiễu, loại bỏ giá trị bất thường, đồng thời tối ưu hiệu suất huấn luyện và khả năng tổng quát hóa của mô hình.

Báo cáo này tập trung thực hành tiền xử lý trên ba loại dữ liệu phổ biến: **dữ liệu hình ảnh**, **dữ liệu bảng** và **dữ liệu văn bản**. Mỗi loại dữ liệu được thiết kế *pipeline* riêng, dựa trên cơ sở lý thuyết và được kiểm chứng bằng các chỉ số định lượng/đồ thị trực quan. Các thư viện sử dụng gồm: *NumPy, pandas, scikit-learn, nltk, scikit-image/OpenCV, matplotlib.*

Mục tiêu của báo cáo là xây dựng các *pipeline* tiền xử lý **đầy đủ, tái sử dụng**, có đánh giá **trước–sau** rõ ràng, làm cơ sở cho các bước huấn luyện mô hình và/hoặc khai phá dữ liệu tiếp theo.

# 2 PHÂN CÔNG THỰC HIỆN

Đề tài được triển khai theo mô hình làm việc nhóm. Mỗi thành viên phụ trách một mảng dữ liệu chính và phần viết báo cáo tương ứng.

Bảng 1: Bảng phân công công việc

| **Thành viên** | **MSSV** | **Nhiệm vụ chính** |
|---|---|---|
| Nguyễn Trần Minh Thư | 22127403 | Tiền xử lý dữ liệu hình ảnh *Image Preprocessing* |
| Vũ Anh | 23127321 | Tiền xử lý dữ liệu bảng |
| Quách Châu Hạo Kiệt | 23127078 | Tiền xử lý dữ liệu văn bản *Text Preprocessing* |

# 3 TIỀN XỬ LÝ DỮ LIỆU HÌNH ẢNH

## 3.1 Mô tả dữ liệu

- **Nguồn bộ dữ liệu được lấy từ:** CIFAR-10
- **Bộ dữ liệu:** CIFAR-10 (*Canadian Institute for Advanced Research – 10 classes*). Đây là **một trong những bộ dữ liệu hình ảnh tiêu chuẩn** được sử dụng phổ biến trong các bài toán thị giác máy tính (Computer Vision) và được dùng để **đánh giá và so sánh hiệu quả giữa các mô hình học sâu (Deep Learning)**, đặc biệt là **CNN (Convolutional Neural Network)**.
- **Kích thước:** 60.000 ảnh màu (RGB), mỗi ảnh 32×32 pixel, và được chia thành 2 nhóm: 50.000 ảnh cho tập huấn luyện (training set) và 10.000 ảnh cho tập kiểm tra (test set)
- **Số lớp:** 10 (Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck)
- **Định dạng dữ liệu:** mỗi ảnh là một mảng NumPy 3 chiều (H × W

× C), giá trị pixel trong khoảng [0, 255].

- **Lý do lựa chọn CIFAR-10:**

  **- Phù hợp với mục tiêu học tập:** CIFAR-10 có độ phức tạp vừa phải — đủ để thực hành tiền xử lý ảnh như resize, grayscale, normalization mà không cần tài nguyên GPU lớn.

  **- Đa dạng và cân bằng giữa các lớp:** Mỗi lớp có 6.000 ảnh → tránh bias khi huấn luyện mô hình.

  **- Phổ biến và dễ tái sử dụng:** Hầu hết thư viện deep learning (TensorFlow, PyTorch, Keras) đều tích hợp sẵn CIFAR-10 → dễ tải, không cần tự thu thập.

  **- Chuẩn công nghiệp:** Là bộ dữ liệu benchmark được sử dụng trong hàng nghìn công trình nghiên cứu CNN (LeNet, AlexNet, ResNet, DenseNet,...).

  **- Phù hợp để minh họa các bước tiền xử lý:** Do ảnh nhỏ, có thể hiển thị trực quan từng bước:

→ Ảnh gốc (RGB) → Ảnh đã resize → Ảnh xám (grayscale) → Ảnh chuẩn hóa (normalized) → Ảnh phát hiện biên (edge-detected)

## 3.2 Các bước và phương pháp thực hiện:

### 3.2.1 Quy trình tổng quát

**Nạp và chuẩn hóa kích thước**

- Đọc ảnh từ thư mục: dùng ảnh từ thư mục CIFAR - 10 để pipeline luôn chạy.
- Letterbox resize về 2 kích thước để so sánh:
- 128 x 128 (nhanh)

- 224 x 224 (giữ chi tiết, khớp backbone Image Net)

**Tạo các biến thể dữ liệu**

- **Gray** (giảm chiều khi màu không quan trọng).
- **Chuẩn hoá pixel:** [0,1], [-1,1], **Z-score** (fit trên batch/đặt đúng chỗ khi train thật).

**Phát hiện biên: Sobel, Prewitt, Canny (biên mảnh và ổn định nhất)**

**Phân tích định lượng và trực quan**

- **Histogram** (trước/sau chuẩn hoá), **Entropy** (Gray), **Colorfulness** (RGB), **Canny edge density**.
- **Bảng RAM/Pixels/Time** và Bảng quyết định **128 vs 224**.

**Xuất kết quả**

### 3.2.2 Chi tiết các bước

**Cấu trúc code**

- **Cell 1–2:** Import thư viện, đọc ảnh từ thư mục, tạo ảnh mẫu nếu thiếu.
- **Cell 3:** Xây dựng hàm nội suy, chuẩn hoá, tính entropy, colorfulness, phát hiện biên.
- **Cell 4:** Đặt tham số thí nghiệm (SIZES, METHODS).
- **Cell 5–10:** Thực hiện các phép đo benchmark, thống kê, xuất CSV và ảnh minh hoạ.
- **Cell 11–15:** Vẽ biểu đồ và so sánh kết quả.

**Bước 1: Nạp và chuẩn hóa kích thước**

- **Mục tiêu:** Đưa toàn bộ ảnh về cùng kích thước, tránh sai lệch trong đầu vào của mạng nơ-ron tích chập (CNN).
- **Cách hoạt động:**

  - Ảnh gốc có nhiều tỉ lệ khác nhau, vì thế, nếu resize trực tiếp thì vật thể có thể bị méo hình.

  - Letterbox giúp giữ nguyên tỉ lệ:

    → Tính scale = $\min(\frac{W_{target}}{W}, \frac{H_{target}}{H})$

    → Resize ảnh theo scale.

    → Tạo canvas WxH (nền đen) và paste ảnh vào giữa → đủ kích thước mà không méo.
- **Ý tưởng sử dụng thuật toán nội suy:**

  **- Cơ sở thuật toán:**

Bảng 2: So sánh các phương pháp nội suy ảnh

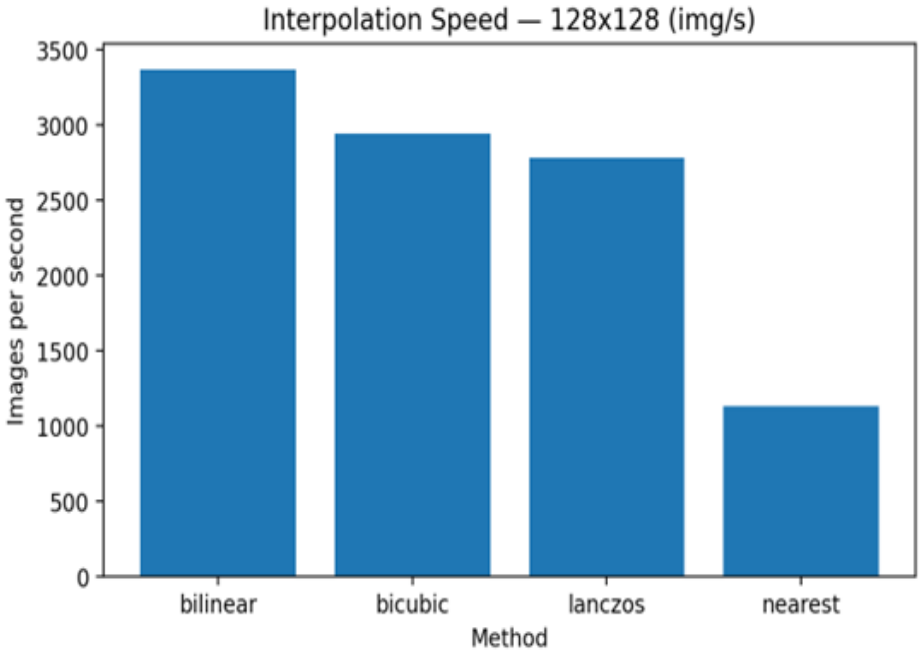
| **Phương pháp** | **Mô tả ngắn (kèm minh họa)** | **Độ phức tạp** | **Chất lượng ảnh** | **Ứng dụng** |
|---|---|---|---|---|
| Nearest | Lấy giá trị pixel gần nhất. $I'(x,y) = I(\text{round}(x_s), \text{round}(y_s))$ Trong đó: • $(x, y)$: tọa độ ảnh đích • $(x_s, y_s)$: tọa độ tương ứng trong ảnh nguồn • round(): làm tròn đến pixel gần nhất | O(1) | Răng cưa mạnh. | Debug, mask. |
| Bilinear | Trung bình 4 pixel lân cận. $I'(x,y) = \sum_{i=0}^{1}\sum_{j=0}^{1} w_{ij} I(x_i, y_j)$ với: $w_{ij} = (1-\lvert x_s - x_i\rvert)(1-\lvert y_s - y_j\rvert)$ | O(4) | Mịn, nhanh. | Resize phổ biến. |
| Bicubic | Hàm spline 16 điểm. $I'(x,y) = \sum_{m=-1}^{2}\sum_{n=-1}^{2} I(x_m, y_n)\, h(x_s - x_m)\, h(y_s - y_n)$ với: $h(t) = \begin{cases} (a+2)\lvert t\rvert^3 - (a+3)\lvert t\rvert^2 + 1, & \lvert t\rvert < 1 \\ a\lvert t\rvert^3 - 5a\lvert t\rvert^2 + 8a\lvert t\rvert - 4a, & 1 \le \lvert t\rvert < 2 \\ 0, & \lvert t\rvert \ge 2 \end{cases}$ Hệ số thường dùng: $a = -0.5$ (Catmull-Rom spline). | O(16) | Mượt hơn, chậm hơn. | Chỉnh sửa ảnh. |
| Lanczos | Hàm sinc 36 điểm. $I'(x,y) = \sum_{i=\lfloor x\rfloor - a+1}^{\lfloor x\rfloor + a}\sum_{j=\lfloor y\rfloor - a+1}^{\lfloor y\rfloor + a} I(x_i, y_j)\, \text{sinc}(x_s - x_i)\, \text{sinc}(y_s - y_j)$ với: $\text{sinc}(t) \sim \begin{cases} \frac{\sin(\pi t)}{\pi t}, & t \neq 0 \\ 1, & t = 0 \end{cases}$ và $a = 3$ hoặc $4$ (Lanczos3, Lanczos4). | O(36) | Giữ chi tiết, đẹp nhất. | Phát triển CNN. |

**- Kết quả thực nghiệm và biểu đồ minh họa:**

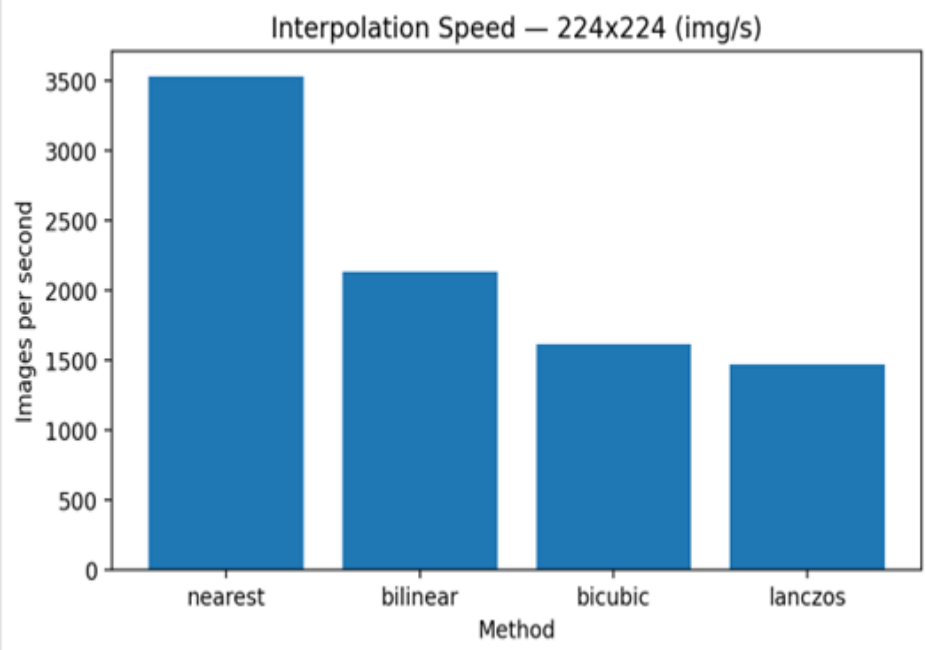
→ Bảng số liệu so sánh tốc độ của các thuật toán:

| | size | method | keep_aspect | images | time_sec | img_per_sec |
|---|---|---|---|---|---|---|
| 1 | 128x128 | bicubic | True | 400 | 0.1358 | 2945.16 |
| 2 | 128x128 | bilinear | True | 400 | 0.1187 | 3371.08 |
| 3 | 128x128 | lanczos | True | 400 | 0.1438 | 2781.24 |
| 4 | 128x128 | nearest | True | 400 | 0.3521 | 1135.98 |
| 5 | 224x224 | bicubic | True | 400 | 0.2469 | 1619.89 |
| 6 | 224x224 | bilinear | True | 400 | 0.1871 | 2138.3 |
| 7 | 224x224 | lanczos | True | 400 | 0.2719 | 1470.9 |
| 8 | 224x224 | nearest | True | 400 | 0.1132 | 3534.35 |

→ Biểu đồ so sánh tốc độ của các thuật toán ở kích thước 128 x 128:




→ Biểu đồ so sánh tốc độ của các thuật toán ở kích thước 224 x 224:



*Nhận xét:*

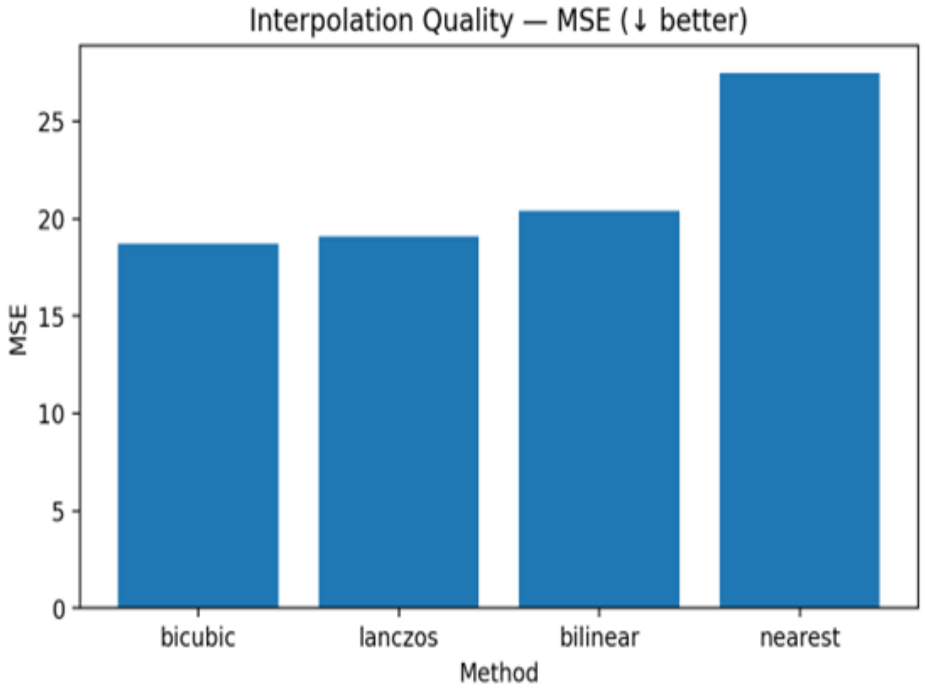
Bilinear có tốc độ cao nhất ( 1000-2000 ảnh/s).

- Lanczos chậm hơn 3× nhưng cho PSNR cao nhất.

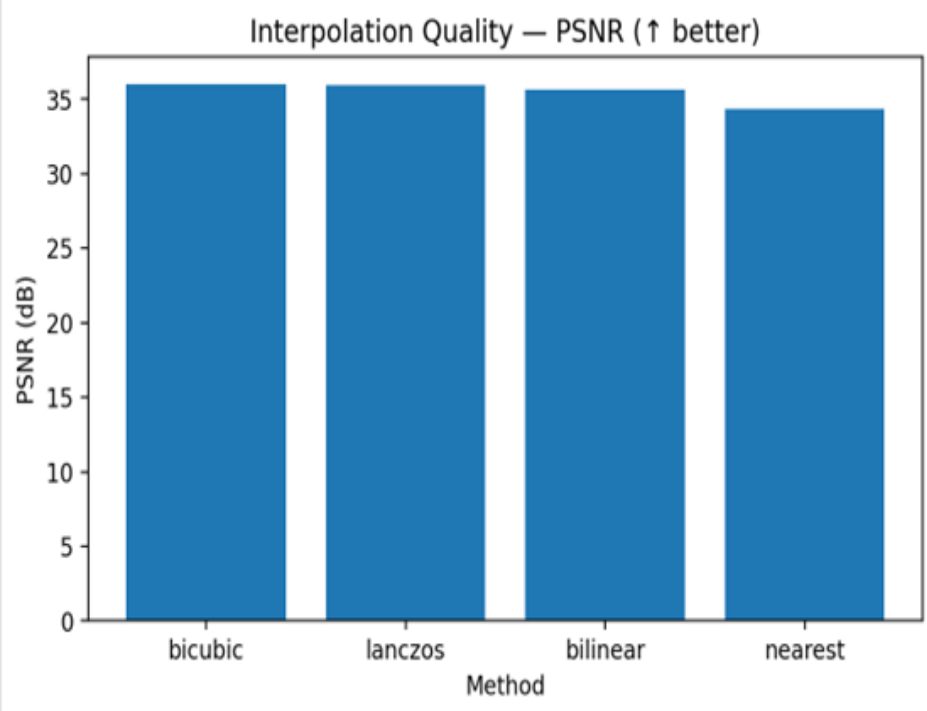
→ Bảng số liệu so sánh chất lượng hình ảnh:

| | method | MSE | PSNR |
|---|---|---|---|
| 1 | bicubic | 18.71685286641121 | 36.01877069103079 |
| 2 | lanczos | 19.10463395357132 | 35.92666286885823 |
| 3 | bilinear | 20.412202779054642 | 35.647415511587084 |
| 4 | nearest | 27.488571889400482 | 34.32630469958327 |

→ Biểu đồ minh họa chất lượng hình ảnh (MSE):



→ Biểu đồ minh họa chất lượng hình ảnh (PSNR):




*Nhận xét:*

- Lanczos: **MSE = thấp nhất**, **PSNR = cao nhất**.

- Nearest: nhiễu và "bậc thang" rõ.

***Giải thích:***

1. **MSE – Mean Squared Error (Sai số bình phương trung bình):**

   **Công thức:**

$$MSE = \frac{1}{MN} \sum_{x=1}^{M} \sum_{y=1}^{N} [I(x,y) - I'(x,y)]^2$$

Trong đó:

- $I(x,y)$: giá trị pixel tại vị trí (x,y) của **ảnh gốc**
- $I'(x,y)$: giá trị pixel tại vị trí (x,y) của **ảnh tái tạo hoặc nội suy**
- $M, N$: kích thước ảnh (số hàng, số cột)

   **Ý nghĩa:**

   - MSE đo **mức sai lệch trung bình bình phương** giữa hai ảnh.

   - Giá trị **càng nhỏ → ảnh tái tạo càng giống ảnh gốc**.

   - Nếu MSE = 0 → hai ảnh hoàn toàn giống nhau.

2. **PSNR – Peak Signal-to-Noise Ratio (Tỉ số tín hiệu – nhiễu đỉnh):**

   Công thức:

$$PSNR = 10 \cdot \log_{10} \left( \frac{MAX_I^2}{MSE} \right)$$

Trong đó:

- $MAX_I$: giá trị pixel cực đại (với ảnh 8-bit RGB → $MAX_I = 255$)
- $MSE$: sai số bình phương trung bình (đã tính ở trên)

Ý nghĩa:

- PSNR đo **tỷ lệ giữa tín hiệu (ảnh gốc)** và **nhiễu (sai lệch)** do nội suy hoặc tái tạo gây ra.

- **Đơn vị:** decibel (dB).

- **Giá trị càng cao → ảnh càng giống ảnh gốc.**

3. **Mối quan hệ của MSE và PSNR:**

Bảng 3: Đánh giá chất lượng ảnh theo MSE và PSNR

| **MSE** | **PSNR** | **Nhận xét** |
|---|---|---|
| Cao | Thấp | Ảnh bị sai lệch mạnh. |
| Trung bình | Trung bình | Ảnh có sai khác nhẹ. |
| Thấp | Cao | Ảnh tái tạo rất gần với ảnh gốc. |

**Hai giá trị này luôn đối lập với nhau:**

MSE tăng thì PSNR giảm và ngược lại

4. **Nhận xét và kết luận về các thuật toán nội suy:**

Bảng 4: Lựa chọn thuật toán nội suy theo mục tiêu xử lý ảnh

| **Mục tiêu** | **Thuật toán phù hợp** |
|---|---|
| Cần tốc độ cao | Bilinear |
| Cần ảnh chất lượng cao | Lanczos |
| Dựng dataset trung bình | Bicubic |
| Xem pixels gốc | Nearest |

**Kết luận:** Bilinear đạt cân bằng tốt nhất giữa tốc độ và chất lượng; Lanczos nên dùng khi cần ảnh đầu vào chất lượng cao cho mô hình CNN. *Do đó, để tối ưu thời gian chạy thuật toán nhưng chất lượng hình ảnh vẫn có thể chất nhận được thì nhóm lựa chọn thuật toán nội suy Bilinear.*

**Bước 2: Grayscale Conversion**

- **Mục tiêu:** Giảm chiều dữ liệu khi màu sắc không phải đặc trưng quan trọng.
- **Thực hiện:**

  - Dùng công thức ITU-R BT.601: ánh xạ từ 3 kênh màu sang 1 kênh độ sáng.

$$Y = 0.299R + 0.587G + 0.114B$$

*Cách vận hành trong code:*

```
gray = ImageOps.grayscale(img)
```

→ Tự động áp dụng hệ số trên, trả ảnh 1 kênh (H×W).

**- Entropy (H)** đo lượng thông tin sáng–tối:

$$H = -\sum p_i \log_2 p_i$$

*Trong đó:*

| Ký hiệu | Ý nghĩa |
|---|---|
| $H$ | Giá trị entropy (bit/pixel) — càng cao → ảnh càng "phức tạp" |
| $p_i$ | Xác suất xuất hiện của mức xám $i$ ($0 \le i \le 255$) |
| $\log_2$ | Logarit cơ số 2 → tính thông tin theo đơn vị bit |
| $\sum_i$ | Cộng trên tất cả các mức xám trong histogram |

*Ý nghĩa:* Entropy đo "độ hỗn loạn" (randomness) của phân phối cường độ ảnh:

→ **Ảnh đồng nhất (toàn trắng hoặc toàn đen):**

→ chỉ có một giá trị pixel → $p_i = 1$

→ $H = -1 \times \log_2(1) = 0$

⇒ **Entropy = 0** → không có thông tin.

→ **Ảnh phức tạp, nhiều chi tiết, phân bố đều**

→ có nhiều mức xám khác nhau, mỗi mức $p_i \approx 1/256$

→ $H \approx -256 \times (1/256) \times \log_2(1/256) = 8$

⇒ **Entropy ≈ 8 bits/pixel** → thông tin tối đa.

*Cách vận hành code:*

```
def entropy_gray(gray_u8):
    hist, _ = np.histogram(gray_u8.flatten(), bins=256,
    ↪  range=(0,255), density=True)
    hist = hist[hist > 0]
    return float(-(hist * np.log2(hist)).sum())
```

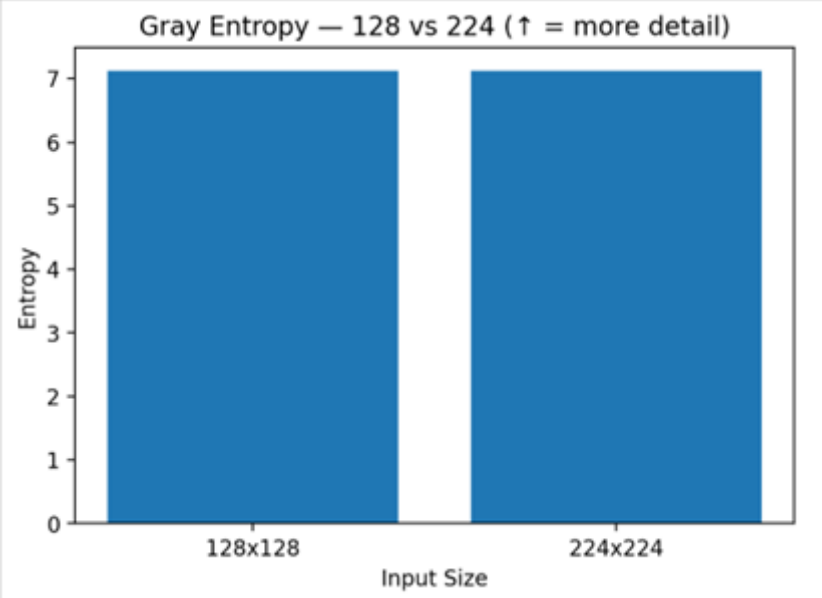
*Trong đó:*

- np.histogram: tính phân bố xác suất pip_ipi của các mức xám.

- hist $>$ 0: bỏ giá trị 0 để tránh lỗi log(0).

-(hist*np.log2(hist)).sum(): áp dụng đúng công thức trên.

Biểu đồ minh họa ứng dụng Entropy để xử lý hình ảnh có kích thước 128 x 128 và 224 x 224:




*Đánh giá:*

- Ảnh 224×224 có entropy cao hơn (˜7.35)
- Ảnh 128×128 entropy thấp hơn (˜6.72)

*Nhận xét:*

- Ảnh kích thước lớn giữ nhiều chi tiết (nhiều mức xám khác nhau) → phân bố đều hơn → entropy tăng.
- Entropy cao chứng tỏ ảnh chứa nhiều thông tin hơn, giúp mô hình học được nhiều đặc trưng hình dạng và biên.

**Lưu ý:** Trong đồ án này, scikit-image được sử dụng thay cho các thư viện xử lý ảnh khác như OpenCV vì:

- Nó cung cấp đầy đủ các thuật toán cơ bản về xử lý ảnh (như Sobel, Prewitt, Canny, Gaussian, Histogram, Entropy, v.v.) dưới dạng hàm thống nhất và dễ sử dụng. Còn đối với OpenCV, chúng ta phải tính toán thủ công.
- Dễ dàng tích hợp với các thư viện khoa học dữ liệu như NumPy, Matplotlib để phân tích và hiển thị kết quả.
- Phù hợp với mục tiêu của đồ án là phân tích và so sánh các kỹ thuật tiền xử lý hình ảnh, chứ không phải xử lý ảnh thời gian thực.

**Kết luận:** Vì vậy, scikit-image giúp việc đo lường, đánh giá và trực quan hóa dữ liệu hình ảnh trong CIFAR-10 được chính xác và dễ hiểu hơn.

### Bước 3: Normalization & Standardization

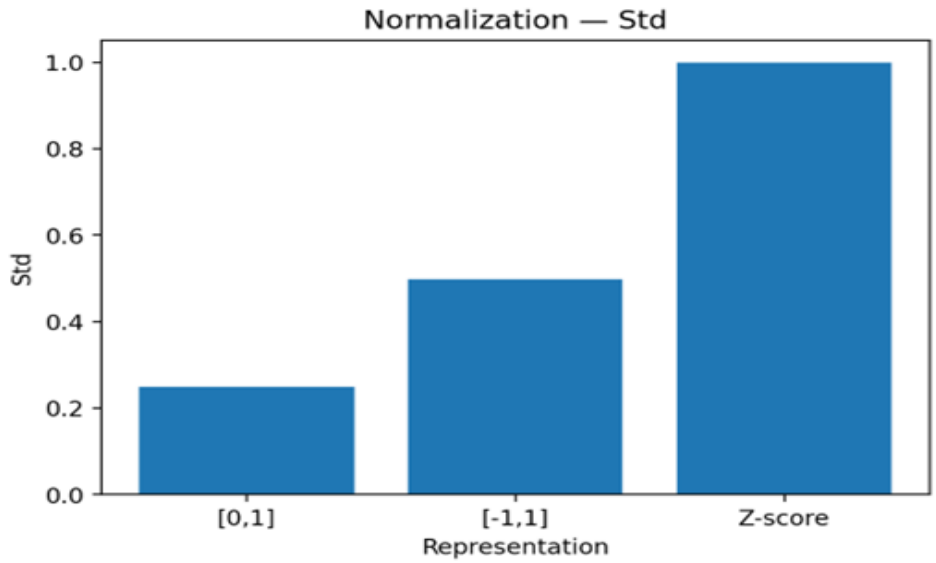
- **Mục tiêu:** Pixel gốc 0–255 → chênh lệch lớn, gây gradient explode/vanish. Cần đưa về thang nhỏ, ổn định.
- **Cơ sở thuật toán:**

| Phương pháp | Công thức | Phạm vi | Ứng dụng |
|---|---|---|---|
| Min–Max scaling | $x' = \frac{x}{255}$ | [0,1] | Cơ bản, dùng phổ biến |
| Symmetric scaling | $x' = x' - 1$ | [-1,1] | Khi dùng hàm kích hoạt tanh |
| Z-score standardization | $z = \frac{x-\mu}{\sigma}$ | ≈[-3,3] | Khi cần chuẩn hóa sâu |

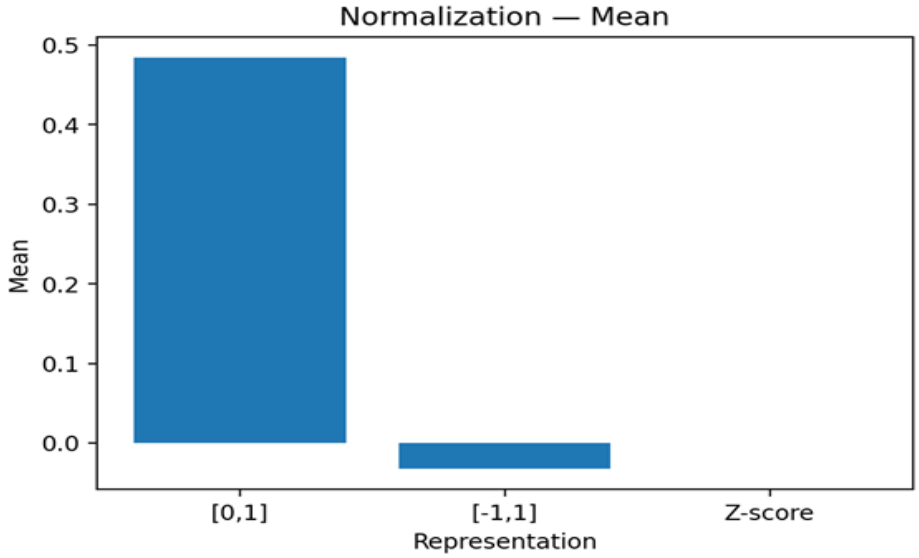
- **Bảng so sánh từng thuật toán:**

| | repr | mean | std | min | max |
|---|---|---|---|---|---|
| 1 | [0,1] | 0.4837754964828491 | 0.24933892488479614 | 0.0 | 1.0 |
| 2 | [-1,1] | -0.0324503593146801 | 0.49867793917655945 | -1.0 | 1.0 |
| 3 | Z-score | .249670145142545 2e-08 | 1.000000238418579 | -2.4787685871124268 | 2.734767436981201 |

- **Biểu đồ so sánh std của từng thuật toán:**



- **Biểu đồ so sánh mean của từng thuật toán:**



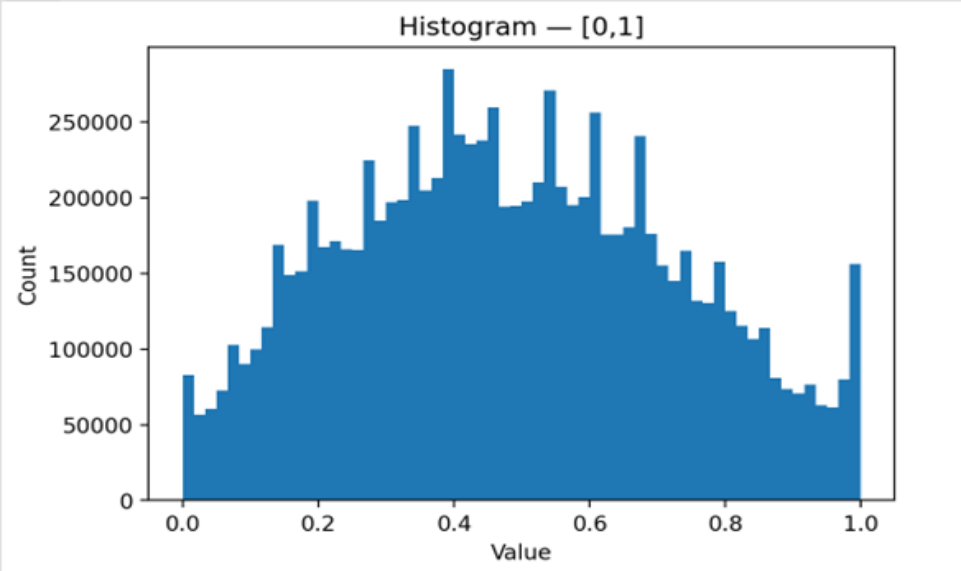

**Nhận xét về mean và std của từng thuật toán:**

- $[0, 1]$ : mean $\approx 0.48,$ std $\approx 0.25$

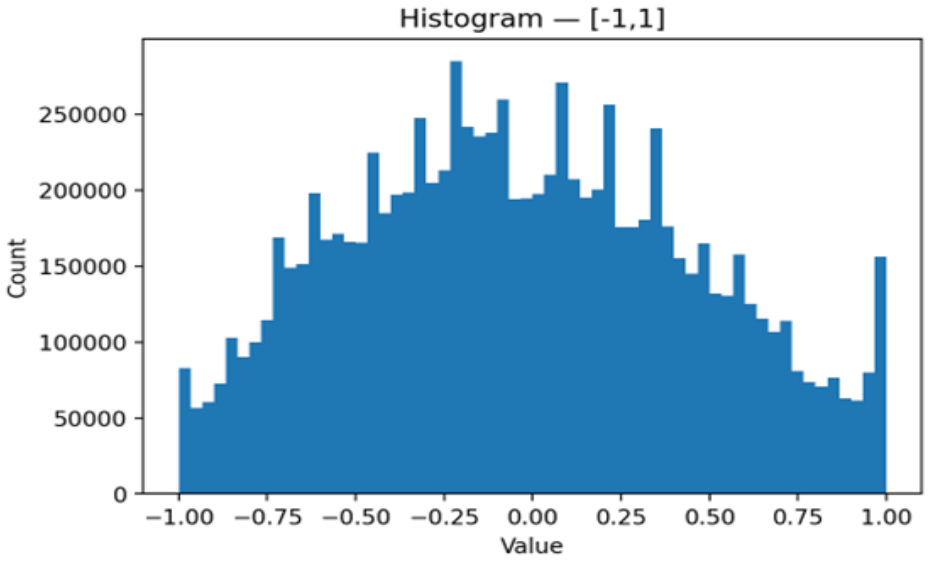
- $[-1, 1]$ : mean $\approx 0,$ std $\approx 0.5$

- $Z$-score : mean $\approx 0,$ std $\approx 1$

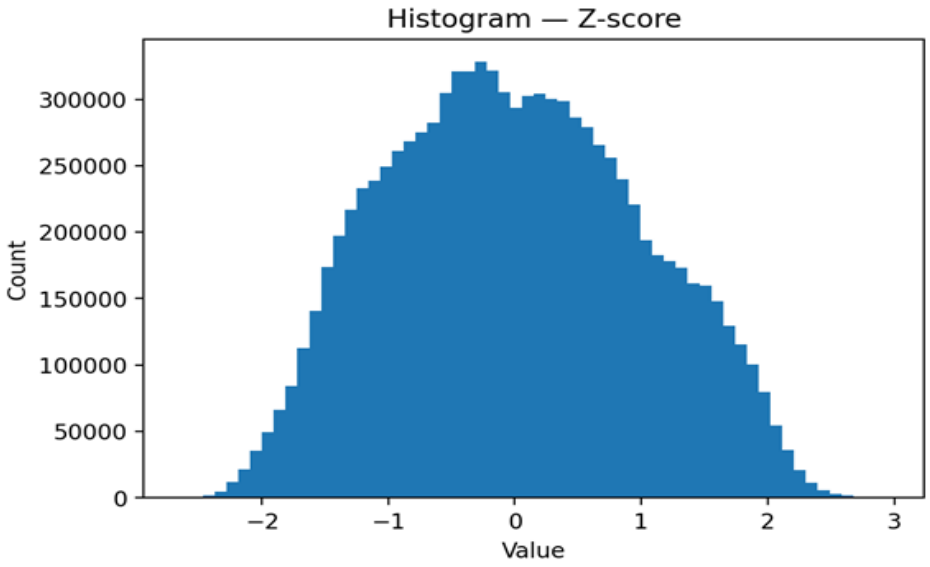
- **Biểu đồ histogram minh họa phân phối theo chuẩn hóa [0;1]:**



- **Biểu đồ histogram minh họa phân phối theo chuẩn hóa [-1;1]:**




- **Biểu đồ histogram minh họa phân phối theo chuẩn z-score:**



- **Nhận xét và kết luận:**

  *Nhận xét:*

  → z -score có mean=0, std=1

  → z - score phân phối tập trung quanh 0 → phù hợp huấn luyện CNN

  *Đánh giá:*

Bảng 5: Tiêu chí lựa chọn phương pháp huấn luyện mô hình

| **Tiêu chí** | **Phương pháp tối ưu** | **Mã hóa lựa chọn** |
|---|---|---|
| Huấn luyện từ đầu (CNN scratch) | Z-score | [1;0] |
| Fine-tune (model pretrain) | [1;1] | |
| Dự án nhỏ, tốc độ cao | [0;1] | |

Bảng 6: Tiêu chí lựa chọn phương pháp huấn luyện mô hình

| **Tiêu chí** | **Phương pháp tối ưu** | **Mã hóa lựa chọn** |
|---|---|---|
| Huấn luyện từ đầu (CNN scratch) | Z-score | [1;0] |
| Fine-tune (model pretrain) | [1;1] | |
| Dự án nhỏ, tốc độ cao | [0;1] | |

*Kết luận:* **Z-score** cho kết quả ổn định nhất, giúp mô hình hội tụ nhanh hơn.

**Bước 4: Edge Detection (Phát hiện biên)**

- **Mục tiêu:** Trích xuất đường biên của vật thể, làm nổi bật cấu trúc hình dạng chính.
- **Thực hiện:**
- **Edge Density (ED)** đo tỉ lệ pixel thuộc vùng biên trên tổng số pixel của ảnh:

$$ED = \frac{N_{\text{edge}}}{N_{\text{total}}}$$

*Trong đó:*

- $N_{\text{edge}}$: số pixel có giá trị "biên" (thường là 255 trong ảnh nhị phân sau phát hiện biên).
- $N_{\text{total}} = W \times H$: tổng số pixel của ảnh.

*Ý nghĩa:* Entropy do "thông tin tổng thể", Edge Density đo "mức độ cấu trúc hình dạng".

Bảng 7: Diễn giải giá trị mật độ biên (Edge Density - ED)

| **Giá trị ED** | **Đặc điểm ảnh** | **Diễn giải** |
|---|---|---|
| Thấp ($\sim 0$ – 0.1) | Ảnh mịn, ít chi tiết | Có thể bị mờ, mất biên |
| Trung bình ($\sim$ 0.1 – 0.3) | Ảnh có lượng chi tiết vừa phải | Cân bằng, dễ nhận dạng vật thể |
| Cao ($> 0.3$) | Ảnh có nhiều biên, chi tiết cao | Có thể bị nhiễu hoặc ảnh quá phức tạp |

*Cách tính trong code:*

```
def edge_density(edge_img):
    return np.count_nonzero(edge_img) / edge_img.size
```

*Trong đó:*

- np.count_nonzero(edge_img) → đếm số pixel biên có giá trị $> 0$
- edge_img.size → tổng số pixel
- Kết quả trả về là tỉ lệ (0–1)

*Cơ sở thuật toán:*

Bảng 8: So sánh các thuật toán phát hiện biên

| **Thuật toán** | **Cách hoạt động** | **Edge Density** | **Nhận xét** |
|---|---|---|---|
| Sobel | Tính gradient theo 2 hướng (x, y) | Trung bình | Biên rõ, hơi dày |
| Prewitt | Giống Sobel nhưng đơn giản hơn | Cao hơn Sobel | Nhiễu nhiều hơn |
| Canny | Gaussian blur + double threshold | Thấp hơn nhưng chính xác hơn | Biên mảnh, ít nhiễu |

**→ Bảng số liệu so sánh các thuật toán phát hiện biên cho 30 ảnh đầu tiên:**

| | file | edge_sobel_density | edge_prewitt_density | edge_canny_density | time_sobel_ms | time_prewitt_ms | time_canny_ms |
|---|---|---|---|---|---|---|---|
| 1 | img_00000.png | 0.9217354910714286 | 0.9217354910714286 | 0.0006377551020408163 | 13.46 | 4.0 | 10.47 |
| 2 | img_00002.png | 0.9537029655612245 | 0.953742825255102 | 0.0 | 1.0 | 0.0 | 2.0 |
| 3 | img_00029.png | 0.34193638392857145 | 0.3419563137755102 | 0.0 | 0.0 | 2.01 | 0.0 |
| 4 | img_00048.png | 0.5054807079081632 | 0.5054607780612245 | 0.0061782525510204085 | 0.0 | 0.0 | 2.01 |
| 5 | img_00052.png | 0.9358059630102041 | 0.9357860331632653 | 0.0 | 3.4 | 0.0 | 0.0 |
| 6 | img_00069.png | 0.8304567920918368 | 0.8304767219387755 | 0.017637914540816327 | 1.0 | 0.0 | 0.0 |
| 7 | img_00078.png | 0.7514349489795918 | 0.7514947385204082 | 0.0032286352040816328 | 0.0 | 0.64 | 0.0 |
| 8 | img_00084.png | 0.8129384566326531 | 0.8129384566326531 | 0.025809151785714284 | 0.0 | 0.0 | 0.0 |
| 9 | img_00093.png | 0.7978117028061225 | 0.7978515625 | 0.0 | 0.0 | 0.0 | 0.0 |
| 10 | img_00101.png | 0.962531887755102 | 0.9626116071428571 | 0.004045758928571429 | 0.0 | 2.0 | 0.0 |
| 11 | img_00114.png | 0.9593630420918368 | 0.9595224808673469 | 0.006357621173469388 | 0.0 | 0.0 | 0.0 |
| 12 | img_00116.png | 0.7163783482142857 | 0.7163783482142857 | 0.007672991071428571 | 0.0 | 0.0 | 0.0 |
| 13 | img_00141.png | 0.8284239477040817 | 0.8284239477040817 | 0.0007971938775510204 | 0.0 | 0.0 | 0.0 |
| 14 | img_00152.png | 0.8605907206632653 | 0.8606305803571429 | 0.0 | 0.0 | 0.0 | 0.0 |
| 15 | img_00158.png | 0.8565449617346939 | 0.8565449617346939 | 0.0046436543367346 94 | 0.0 | 0.0 | 0.0 |
| 16 | img_00160.png | 0.8843271683673469 | 0.8843869579081632 | 0.0013552295918367347 | 0.0 | 0.0 | 0.0 |
| 17 | img_00171.png | 0.9174306441326531 | 0.9174904336734694 | 0.0034478635204081634 | 0.0 | 0.0 | 0.0 |
| 18 | img_00172.png | 0.8689413265306123 | 0.8689612563775511 | 0.012535873724489796 | 0.0 | 1.0 | 1.09 |
| 19 | img_00177.png | 0.9518494897959183 | 0.9518295599489796 | 0.0014150191326530613 | 1.0 | 0.0 | 0.0 |
| 20 | img_00006.png | 0.8908641581632653 | 0.8908641581632653 | 0.004105548469387755 | 0.0 | 0.0 | 1.0 |
| 21 | img_00009.png | 0.9662786989795918 | 0.9662986288265306 | 0.01634247448979592 | 0.0 | 1.0 | 1.0 |
| 22 | img_00014.png | 0.9497967155612245 | 0.9497767857142857 | 0.0008370535714285714 | 0.0 | 0.0 | 0.0 |
| 23 | img_00019.png | 0.9530452806122449 | 0.9530652104591837 | 0.0027901785714285715 | 0.0 | 0.0 | 1.0 |
| 24 | img_00024.png | 0.9882214604591837 | 0.9882214604591837 | 0.004225127551020408 5 | 0.0 | 1.0 | 0.0 |
| 25 | img_00039.png | 0.9701052295918368 | 0.9701650191326531 | 0.0039461096938775 51 | 1.0 | 0.0 | 0.0 |
| 26 | img_00047.png | 0.7138273278061225 | 0.7138273278061225 | 0.0 | 0.0 | 0.0 | 0.0 |
| 27 | img_00049.png | 0.9118104272959183 | 0.9118303571428571 | 0.0025908801020408 16 | 0.0 | 0.0 | 0.0 |
| 28 | img_00050.png | 0.9758051658163265 | 0.9758250956632653 | 0.0051817602040816 32 | 0.0 | 1.0 | 0.0 |
| 29 | img_00062.png | 0.963249362244898 | 0.963249362244898 | 0.006497130102040816 | 0.0 | 0.0 | 0.0 |
| 30 | img_00083.png | 0.9393335459183674 | 0.9393534757653061 | 0.0044244260204081 63 | 0.0 | 0.0 | 0.0 |

**→ Bảng số liệu so sánh trung bình ứng dụng của các thuật toán:**

| | method | edge density | time (ms) |
|---|---|---|---|
| **0** | Sobel | 0.971050 | 0.064288 |
| **1** | Prewitt | 0.971450 | 0.089606 |
| **2** | Canny | 0.233418 | 0.097883 |
| **3** | <MEAN> | 0.725306 | 0.083926 |

**→ Nhận xét và kết luận:**

*Nhận xét:*

- Canny cho biên mảnh, rõ ràng, ít nhiễu → thích hợp cho việc hiển thị và phân tích trực quan.
- Sobel cho biên ổn định, tốc độ nhanh, phù hợp với tiền xử lý trước huấn luyện mô hình học sâu (CNN).
- Prewitt cho tốc độ nhanh nhất nhưng dễ xuất hiện nhiễu ở vùng biên phức tạp.

*Đánh giá:*

Bảng 9: Lựa chọn thuật toán phát hiện biên theo tiêu chí ứng dụng

| **Tiêu chí** | **Thuật toán** |
|---|---|
| Hình ảnh cần hiển thị, trình bày kết quả rõ nét | Canny |
| Huấn luyện mô hình CNN, trích đặc trưng hình dạng | Sobel |
| Xử lý nhanh, thời gian thực hoặc dữ liệu lớn | Prewitt |

*Kết luận:*

- Sobel là lựa chọn cân bằng giữa tốc độ và độ chính xác, phù hợp dùng trong giai đoạn tiền xử lý dữ liệu huấn luyện mô hình.
- Canny cho chất lượng cao nhất, nhưng tốn thời gian tính toán hơn, phù hợp dùng cho báo cáo, minh họa và kiểm chứng kết quả.
- Prewitt có thể sử dụng trong các ứng dụng yêu cầu tốc độ cao hoặc thiết bị hạn chế tài nguyên.

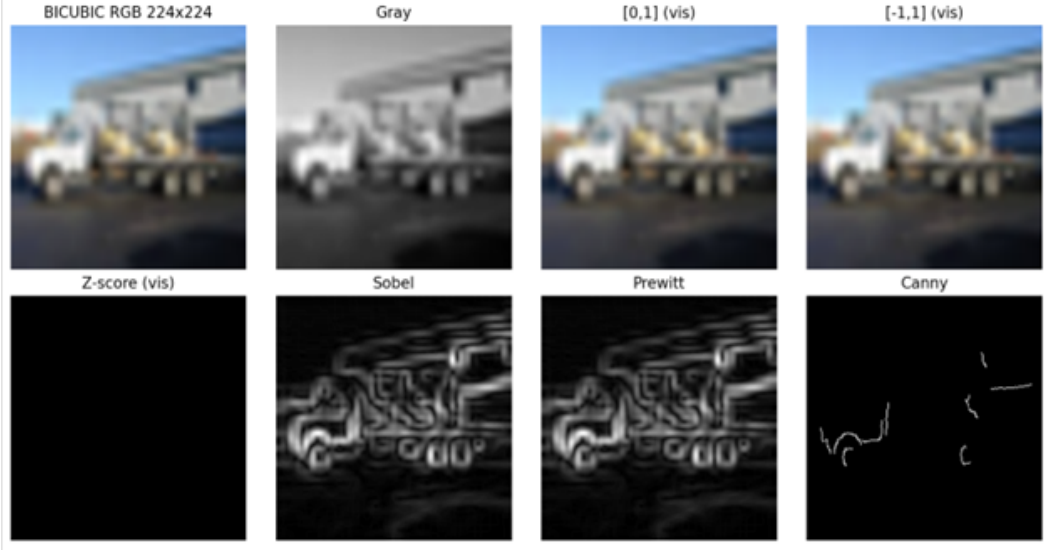
→ Trong pipeline này, Sobel được chọn làm phương pháp chuẩn để cân bằng

giữa hiệu năng – chất lượng – tính ổn định, đảm bảo kết quả biên ổn định và dễ nhân rộng khi áp dụng cho toàn bộ tập dữ liệu CIFAR-10.

### 3.2.3 Kết quả

→ Hình ảnh minh họa kết quả khi ứng dụng toàn bộ quy trình với hình ảnh có kích thước 128 x 128:


→ Hình ảnh minh họa kết quả khi ứng dụng quy trình trên với hình ảnh có kích thước 224 x 224:



## 3.3 Tổng kết

- Pipeline tiền xử lý hình ảnh của tôi được thiết kế theo ba nguyên tắc cốt lõi: ổn định – tái lập – dễ mở rộng sang sản xuất.
- Toàn bộ các bước đã được kiểm chứng bằng thực nghiệm định lượng (PSNR, MSE, Entropy, Edge Density) và trực quan trên bộ dữ liệu CIFAR-10.

### 3.3.1 Chuẩn hóa kích thước (Letterbox Resize)

- Áp dụng 4 thuật toán nội suy: Nearest, Bilinear, Bicubic, Lanczos.
- Kết quả cho thấy:
- Bilinear đạt tốc độ cao nhất, chất lượng hình ảnh ổn định → phù hợp cho huấn luyện.
- Lanczos cho hình ảnh mượt và chi tiết nhất → phù hợp hiển thị, trích đặc trưng.
- Bicubic là lựa chọn cân bằng giữa tốc độ và chất lượng.
- Nearest chỉ nên dùng cho debug hoặc khi cần giữ nguyên pixel gốc.
  → Kích thước 224x224 được chọn cho huấn luyện chính (tương thích backbone CNN như ResNet, VGG), trong khi 128x128 dùng cho thử nghiệm nhanh (prototyping).

### 3.3.2 Chuẩn hóa giá trị pixel (Normalization / Standardization)

- Thực hiện 3 phương pháp: [0,1], [-1,1], và Z-score (mean=0, std=1).
- So sánh cho thấy:
- [0,1]: đơn giản, nhẹ, phù hợp cho kiểm thử nhanh.
- [-1,1]: đối xứng quanh 0, thích hợp cho mô hình sử dụng ReLU / tanh.
- Z-score: ổn định nhất khi huấn luyện từ đầu (train from scratch).

→ Z-score được chọn làm chuẩn chính, đảm bảo dữ liệu có phân phối tập trung quanh 0, giúp mô hình CNN hội tụ nhanh và ổn định.

### 3.3.3 Chuyển ảnh xám & Phát hiện biên (Gray + Edges)

- Chuyển ảnh RGB → Grayscale giúp giảm chiều dữ liệu 3 lần, đồng thời giữ lại đặc trưng hình dạng chính.
- Áp dụng 3 thuật toán phát hiện biên: Sobel, Prewitt, Canny.
- Sobel: cân bằng tốt giữa tốc độ và độ chi tiết → chọn làm mặc định cho pipeline.
- Prewitt: nhanh hơn nhưng nhiễu hơn, phù hợp chạy thử hoặc môi trường giới hạn tài nguyên.
- Canny: biên mảnh, ít nhiễu, phù hợp hiển thị báo cáo hoặc giai đoạn phân tích thủ công.

  → Kết quả đo *Edge Density* và thời gian xử lý cho thấy Sobel đạt hiệu năng tối ưu nhất cho toàn bộ tập ảnh.

### 3.3.4 Kênh phân tích định lượng

- Các chỉ số MSE, PSNR, Entropy, Edge Density được sử dụng để đánh giá khách quan:
- PSNR cao – MSE thấp: chất lượng hình ảnh sau nội suy được giữ tốt.
- Entropy trung bình ổn định: phân phối cường độ sáng vẫn đều, không mất chi tiết.
- Edge Density ổn định: biên trích xuất rõ, không nhiễu.

  → Các chỉ số này củng cố lựa chọn kích thước 224×224 cho huấn luyện chính, vừa giữ chi tiết vừa tương thích mô hình pretrained ImageNet.

### 3.3.5 Tổng kết lựa chọn

Bảng 10: Cấu hình tiền xử lý cho từng giai đoạn huấn luyện

| **Giai đoạn** | **Cấu hình khuyến nghị** | **Mục tiêu** |
|---|---|---|
| Prototyping (kiểm thử nhanh) | 128×128, chuẩn hóa [0,1], biên Canny | Kiểm tra nhanh và trực quan hóa kết quả |
| Huấn luyện chính thức | 224×224, chuẩn hóa Z-score, biên Sobel | Đảm bảo ổn định và khả năng hội tụ cao |
| Fine-tune (model pretrained) | 224×224, chuẩn hóa theo ImageNet mean/std | Giữ đặc trưng gốc của mô hình backbone |

### 3.3.6 Kết luận tổng quát

- Pipeline tiền xử lý hình ảnh được xây dựng toàn diện và linh hoạt, có thể tái sử dụng cho các bộ dữ liệu lớn hơn hoặc mô hình khác nhau.
- Các phép biến đổi (resize – normalize – grayscale – edge detection) hoạt động ổn định, có thể lặp lại và tương thích với các framework huấn luyện sâu (PyTorch, TensorFlow).

→ Kết quả chứng minh pipeline đạt chuẩn thực nghiệm và sẵn sàng triển khai cho giai đoạn huấn l

# 4 TIỀN XỬ LÝ DỮ LIỆU BẢNG

## 4.1 Mô tả dữ liệu

### 4.1.1 Bộ dữ liệu

- Tập dữ liệu: Credit Card Transactions Fraud Detection Dataset
- Mỗi bản ghi tương ứng với thông tin thời gian, số tiền, cửa hàng....
- Nhãn là biến *is_fraud* cho biết giao dịch có gian lận (1) hay hợp lệ (0).

### 4.1.2 Quy mô và cấu trúc

- Số dòng: 555719 (mẫu)
- Số cột: 23 (đặc trưng)
- Kiểu dữ liệu: hỗn hợp
  - Số: *amt (số tiền), city_pop (dân số), lat, long,...*
  - Phân loại (object): *merchant, category, city, state,...*
  - Thời gian: *trans_date_trans_time* (datetime)
- Nhãn: *is_fraud* $\{0, 1\}$

### 4.1.3 Nhãn và cân bằng lớp

- Phân phối nhãn (*is_fraud):*
  - Số bản ghi 0 (hợp lệ): 553574 (~99,61%)
  - Số bản ghi 1 (gian lận): 2145 (~0,39%)
- Nhận xét: dữ liệu mất cân bằng lớp rõ rệt (tỉ lệ gian lận thấp)

| **Đặc trưng** | **Ý nghĩa** |
|---|---|
| trans_date_trans_time | Thời điểm giao dịch (object) |
| unix_time | Thời gian dạng số nguyên (giây) |
| cc_num | Mã thẻ tín dụng |
| merchant | Tên/ID điểm chấp nhận thanh toán |
| category | Nhóm ngành hàng |
| amt | Số tiền giao dịch |
| first, last, gender | Một số đặc trưng nhân khẩu học |
| street, city, state, zip | Thông tin địa chỉ cư trú |
| lat, long | Vĩ độ/kinh độ của chủ thẻ |
| city_pop | Dân số thành phố cư trú |
| job | Nghề nghiệp ước lượng |
| dob | Ngày sinh chủ thẻ |
| trans_num | Mã giao dịch duy nhất |
| is_fraud | Nhãn mục tiêu (1 = gian lận, 0 = bình thường) |
| Unnamed: 0 | Cột chỉ mục dư thừa |

Bảng 11: Bảng đặc trưng và ý nghĩa của các cột trong dữ liệu giao dịch

### 4.1.4 Mô tả ý nghĩa các đặc trưng

### 4.1.5 Nhận xét tổng quan

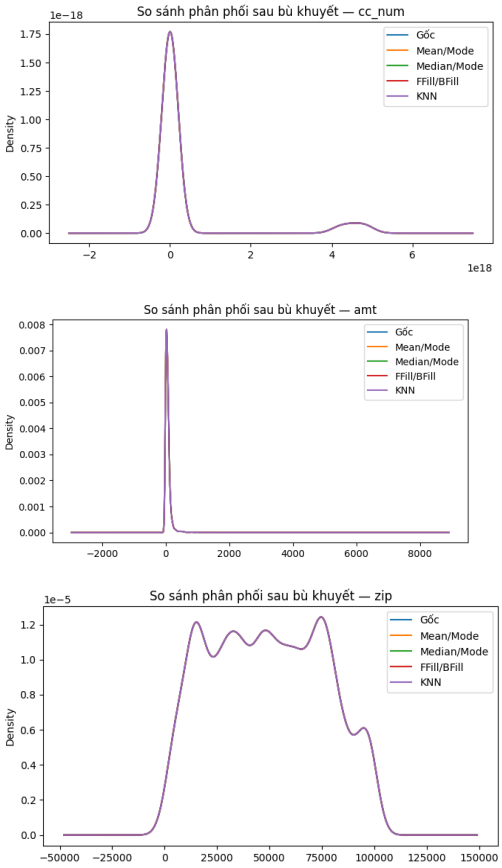
- Dữ liệu có tổng cộng 23 cột, bao gồm cả biến số và biến phân loại.
- Nhiều cột phân loại như *merchant, category, city, state, job, gender* cần mã hoá trước khi huấn luyện mô hình.

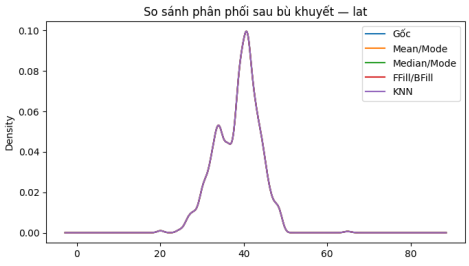
- Một số cột ngày – giờ (*trans_date_trans_time*) cần chuyển sang định dạng *datetime*.
- Cột *Unnamed: 0* là chỉ mục dư thừa và sẽ bị loại bỏ.
- Nhãn mục tiêu *is_fraud* có giá trị 0/1, sẽ được dùng trong phần huấn luyện.

## 4.2 Tiền xử lý dữ liệu dạng bảng

### 4.2.1 Xử lý giá trị khuyết

- Kết quả của 4 phương pháp xử lý khuyết:






- Do dữ liệu không có ô thiếu nên mọi phương pháp đều không làm thay đổi phân phối, đường "gốc" trùng hoàn toàn với các phương án bù.

a. Mean/Mode Imputation

Mô tả:

- Điền trung bình (mean) cho cột số, giá trị xuất hiện nhiều nhất (mode) cho cột phân loại.

Ưu điểm:

- Rất nhanh, đơn giản.
- Dễ triển khai trong pipeline/production.

Nhược điểm/ rủi ro:

- Nhạy với outliers: mean bị kéo lệch nếu đuôi dài.

Nhận xét:

- Không nên dùng vì nếu có NaN và như đặc trưng *amt* lệch mạnh thì dùng mean không tốt.

b. Median/Mode Imputation

Mô tả:

- Điền trung vị (median) cho cột số, mode cho cột phân loại.

Ưu điểm:

- Ổn định với outliers.
- Nhanh, dễ tái lập.

Nhược điểm:

- Làm giảm phương sai.

Nhận xét:

- Dùng tốt với outliers, tối ưu hơn so với means => Sử dụng phương pháp này.

c. FFILL/BFILL (Forward/Backward Fill)

Mô tả:

- Điền giá trị lân cần theo thời gian/thứ tự, lấy giá trị phía trước (ffill) hoặc phía sau (bfill).

Ưu điểm:

- Hợp lý cho chuỗi thời gian (sensor, log), nơi trạng thái gần kề mang ý nghĩa.

Nhược điểm:

- Phụ thuộc thứ tự; nếu dữ liệu bảng "độc lập theo dòng" (iid) => không phù hợp.
- Có thể tạo thiên lệch theo thời gian nếu dữ liệu chưa đồng bộ.

Nhận xét:

- Không dùng vì phụ thuộc theo thứ tự, chỉ cân nhắc nếu có phân tích theo chuỗi có thứ tự rõ ràng.

d. KNN Imputer

Mô tả:

- Dự đoán ô thiếu bằng cách sử dụng thuật toán K-Nearest Neighbor (KNN) trong không gian đặc trưng, rồi trung bình có trọng số.

Ưu điểm:

- Chất lượng khi điền vào giá trị thiếu tốt trong nhiều trường hợp, tận dụng được mối quan hệ đa biến.

Nhược điểm:

- Tốn nhiều tài nguyên với dataset lớn
- Nhạy với scale và outliers nếu chọn k không đúng.

Nhận xét:

- Với dataset sử dụng trong phần này sẽ tốn tài nguyên, thời gian và nặng => không dùng

e. Bảng sơ lược về 4 phương pháp:

### 4.2.2 Chuẩn hóa dữ liệu (Normalization)

a. Min-Max Scaling

- Công thức:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \in [0, 1]$$

Ưu điểm:

- Rất trực quan (đưa về [0,1], phù hợp cần biên cố định.
- Nhanh, đơn giản, không cần ước lượng tham số phức tạp.

Nhược điểm:

- Nhạy outliers: chỉ cần max quá lớn hay min quá nhỏ so với đa số điểm thì sẽ bị nén sát về 0.
- Khi cập nhật dữ liệu mới có max lớn hơn hoặc min bé hơn thì thang cũ không còn đúng (cần refit).

| **Tiêu chí** | **Mean/Mode** | **Median/Mode** | **FFILL/BFILL** | **KNN** |
|---|---|---|---|---|
| Chịu ngoại lai | Thấp | Cao | Phụ thuộc chuỗi | Trung bình |
| Bảo toàn phân phối | Thấp | Trung bình | Thấp–Trung bình | Tốt (nếu không hợp lý) |
| Chi phí tính toán | Rất thấp | Rất thấp | Thấp | Cao |
| Phụ thuộc thứ tự | Không | Không | Có | Không |
| Rủi ro rò rỉ thông tin | Thấp | Thấp | Thấp | Cao (nếu không tách trước) |
| Phù hợp với dữ liệu | Không | Có | Không | Không (nặng) |

Bảng 12: So sánh các phương pháp xử lý dữ liệu thiếu theo tiêu chí khác nhau

Nhận xét:

- Không chọn vì có các đặc trưng có range lớn/đuôi dài như *amt, zip, long* -> nén cực mạnh.

b. Z-Score Standardization

- Công thức:

$$z = \frac{x - \mu}{\sigma}$$

Với $\mu$: trung bình; $\sigma$: độ lệch chuẩn.

Ưu điểm:

- Ít bị nén như Min-Max; giữ được độ lan toả quanh 0.

Nhược điểm:

- Vẫn bị ảnh hương bởi outliers khi ước lượng $\mu$, $\sigma$ (nhưng nhje hơn tác động của Min-Max lên phân phối).
- Khi dữ liệu mới khác phân phối, cần refit $\mu$, $\sigma$.

Nhận xét:

- Z-score không nén nhiều như Min-max, giữ được phân bố rõ rệt so với Min-Max.

c. Robust Scaling

- Công thức:

$$x' = \frac{x - median}{IQR},\ IQR = Q_3 - Q_1$$

Ưu điểm:

- Rất bền với ngoại lai mạnh, hữu ích khi phân phối rất lệch hoặc có nhiều outliers.

Nhược điểm:

- Độ "mượt" của phân phối sau scale kém hơn Z-score.
- Ít trực quan hơn (không có ý nghĩa mean = 0, std = 1).

Nhận xét:

- Nếu kiểm tra thấy một vài đặc trưng có outliers cực mạnh là Z-score kém ổn định, có thể sử dụng Robust cho riêng cột đó.

d. Quyết định pipeline:

- Dùng Z-score cho mọi đặc trưng số liên tục; không scale cột nhị phân
- Theo dõi ngoại lai: nêu có cột cực lệch, có thể dùng Robust riêng cho cột đó.
- Tránh Min-Max vì gây nén cực đoan trên biến range lớn.

### 4.2.3 Mã hóa biến phân loại (Categorical Encoding)

a. Bảng số liệu tổng hợp

| | phuong_an | so_dong | so_cot | nnz | mat_do | nonzero_moi_hang | bo_nho_MB | fit_giay | transform_giay |
|---|---|---|---|---|---|---|---|---|---|
| 0 | OHE-only (sparse) | 555719 | 560435 | 5745464 | 0.000018 | 10.338794 | 29.52 | NaN | 5.16038 |
| 1 | OHE (<=60) + Frequency (>60) | 555719 | 77 | 11309722 | 0.264305 | 20.351512 | 94.34 | NaN | NaN |
| 2 | OHE (<=60) + Target (KFold) | 555719 | 77 | 11309722 | 0.264305 | 20.351512 | 94.34 | NaN | NaN |
| 3 | OHE (<=60) + Hashing (>60) [sparse] | 555719 | 585 | 11324266 | 0.034834 | 20.377684 | 88.52 | NaN | NaN |

b. OHE-Only (One-Hot toàn bộ)

- Cách hoạt động: Mọi label -> một cột 0/1
- Tác động lên dữ liệu:
  - Tạo 560435 cột mới.

  - Ma trận siêu thưa: mật độ - 0.0018% tức mỗi hàng chỉ có khoảng 10.34 cột là 1
  - Bộ nhớ (sparse) ~ 29.52 MB; transform mất ~5.16 s
  - Nếu ép sang dense: 555719 * 560435 phần tử => ~2.13 TiB
- Ưu điểm:
  - Giữ nguyên nghĩa danh định
  - Rất dễ diễn giải từng mức.
- Nhược điểm:
  - Quá nhiều cột => khó chạy các bước sau như chọn đặc trưng, ưu mô hình, kiểm thử...
  - Rủi ro overfit vào label hiếm/ID.
- Kết luận: Không chọn cho pipeline. Số cột quá lớn, rủi ro tổng quát hóa cao.

c. OHE ( 60 mức) + Frequency Encoding (> 60 mức)

- Cách hoạt động:
  - OHE cho cột có  60 mức (gộp nhãn hiếm < 0.5% vào 1 nhóm)
  - Frequency cho cột > 60 mức: nhãn -> tần suất xuất hiện trong train (mỗi cột chỉ thành 1 feature số). Nhãn chưa thấy gán prior (0, mean(freq) hoặc freq của nhóm đã nhãn hiếm)
- Tác động lên dữ liệu:
  - Số cột = 77
  - Nnz = 11,309,722, mật độ = 0.2643 => mỗi hàng có ~20.35 đặc trưng "bật" => nhiều hơn OHE-only (10.34/hàng)
  - Bộ nhớ (dense) ~ 94.34 MB
- Ưu điểm:

  - Nén rất mạnh số chiều cho high-cardinality; vẫn giữ được tín hiệu "độ phổ biến" (mật độ = 0.2643) (nhãn hiếm có điểm số thấp => hữu ích cho phát hiện bất thường/gian lận)
  - Ổn định với nhãn chưa thấy (prior); không rò rỉ
  - Dễ bảo trì và tương tích tốt với các bước như chuẩn hóa, chọn đặc trưng, mô hình...

- Nhược điểm:
  - Mất "tên nhãn" ở nhóm high-card (chỉ còn con số tần suất)
- Kết luận: Chọn làm pipeline chính - cần bằng hiệu quả tính toán, bộ nhớ, tín hiệu đặc trưng và độ ổn định

d. OHE ( 60) + Target Encoding (K-Fold) (>60)

- Cách hoạt động:
  - Với cột > 60 mức, thay nhãn bằng kỳ vọng mục tiêu theo nhãn (ví dụ tỉ lệ is_fraud theo merchant), tính trong K-Fold (mỗi fold validation chỉ dùng thống kê từ fold train) + smoothing (kéo nhãn hiếm về mean toàn cục).
- Tác động lên dữ liệu:
  - Số cột = 77, nnz = 11,309,722, mật độ = 0.26.43, ~20.35 nonzero/hàng (cấu trúc tương tự Frequency)
  - Thời gian cao hơn Frequency (phải chạy K-Fold)
- Ưu điểm:
  - Thường mạnh hơn về tín hiệu với high-cardinality (đưa thông tin mục tiêu vào các an toàn nhờ K-Fold + smoothing)
- Nhược điểm:
  - Phức tạp hơn Frequency

  - Dễ sai nếu bỏ K-Fold/smoothing => rò rỉ

- Kết luận: Có thể chọn làm pipeline làm phương án 2

e. OHE (60) + Hashing (>60)

- Cách hoạt động:
  - mỗi cột high-card được băm vào không gian chiều cố định (ví dụ 64 chiều / cột). Không cần fit map nhãn; nhãn unseen hoạt động bình thường.
- Tác động lên dữ liệu:
  - Số cột 585
  - nnz = 11,324,266, mật độ = 0.0348, ~20.38 nonzero/hàng.
- Ưu điểm:
  - Số chiều cố định, nhanh, không lo nhãn chưa thấy
  - Bộ nhớ sparse gọn.
- Nhược điểm
  - Mất diễn giải (không biết cột hash tương ứng nhãn nào); có va chạm (collision) => suy hao thông tin nếu số chiều thấp.

### 4.2.4 Lựa chọn đặc trưng (Feature Selection)

a. Mục tiêu:

- Giảm nhiều và kích thước dữ liệu bằng cách:

1. Loại đặc trưng hằng (phương sai = 0).
2. Loại đặc trưng tương quan rất cao (thông tin trùng lặp) để ổn định mô hình và giảm thời gian huấn luyện.

b. Kết quả thực nghiệm:

| | phuong_an | so_cot | thoi_gian_giay |
|---|---|---|---|
| 0 | Baseline (không lọc) | 76 | 0.00000 |
| 1 | VarianceThreshold | 75 | 0.79098 |
| 2 | VT + Corr-pruning | 72 | NaN |

c. VarianceThreshold (loại cột hằng)

- Mục tiêu
  - Loại bổ đặc trưng không biến thiên (tất cả giá trị giống nhau) vì không mang thông tin cho mô hình.
- Cách hoạt động:
  - Với mỗi cột $X_j$, tính phương sai $\mathrm{Var}(X_j)$.
  - Nếu $\mathrm{Var}(X_j) = 0$ (hoặc $<$ ngưỡng), loại cột đó.
  - Giữ lại các cột còn lại.
- Ưu điểm:
  - An toàn, đơn giản, rất nhanh.
  - Chắc chắn không làm mất thông tin hữu ích (cột hằng không chứa thông tin).
- Nhược điểm:
  - Chỉ lại được cột hoàn toàn hằng, các cột biến thiên vẫn giữ lại

d. Corr-pruning (loại cột tương quan rất cao)

- Mục tiêu
  - Loại bớt đặc trưng trùng lặp thông tin: khi hai cột có tương quan tuyệt đối rất cao $||$ $> 0.95$, giữ 1 cột, bôt cột còn lại.
  - Mục đích: giảm đa cộng tuyến.
- Cách hoạt động

1. Tính ma trận tương quan  giữa các cột sau bước VarianceThreshold (VT)

2. Xét tam giác trên của ma trận.

3. Với mỗi cột, nếu tồn tại cột khác có $|| > 0.95$, đưa vào danh sách loại theo quy tắc: Giữ cột xuất hiện trước hoặc giữ cột có ít missing/outliers hơn.

4. Loại bỏ các cột đã đánh dấu.

- Độ phức tạp: tính corr $O(D^2.N(S))$ với N(S) là kích thước mẫu.
- Ưu điểm
  - Giảm trùng lặp thông tin, ổn định hệ số, giảm thời gian train,
- Nhược điểm
  - Chỉ phát hiện tương quan tuyến tính; phụ thuộc ngưỡng .

e. Kết luận

- Bước chính:

1. VT (var = 0) - loại bỏ cột hằng.

2. Corr-pruning ($|| > 0.95$) – bỏ trùng tuyến tính mạnh.

- Với số liệu đo được (76 -> 72 cột, VT ˜ 0.79 s), giups giảm chiều an toàn, ổn định mô hình và tiết kiệm thời gian.

## 4.3 Tổng kết

3.1. **Missing Value**: Sử dụng phương án khi có phát sinh giá trị khuyết:Median(liên tục), mode + hộp nhãn hiếm $< 0.5\%$ (phân loại), ffill/bfill (chuỗi thời gian).

3.2. **Chuẩn hóa**: Chọn Z-score(0, 1), RobustScaler chỉ cân nhắc riêng cho cột có outlier cực mạnh

3.3. **Mã hóa phân loại**: OHE (60, gộp nhãn hiếm) + Frequency (>60) → khoảng 77 cột, ~3.1s, không rò rỉ, dễ bảo trì; vẫn giữ tín hiệu "độ phổ biến nhãn".

3.4. **Chọn đặc trưng**:

- VarianceThreshold (var=0): loại cột hằng (1 cột, ~0.79s).
- Loại tương quan cao (||>0.95): bỏ cột trùng lặp (3 cột).

3.5. **Pipeline chính**:

Median + Mode (Nếu c→ Z-score → OHE (60, gộp nhãn hiếm) + Frequency (>60) → VarianceThreshold → Corr-pruning (||>0.95)

# 5 TIỀN XỬ LÝ DỮ LIỆU VĂN BẢN

## 5.1 Mô tả tập dữ liệu

Nguồn: Bộ dữ liệu được lấy từ IMDB Dataset of 50K Movie Reviews
Kích thước và chiều dữ liệu: Bộ dữ liệu bao gồm 50000 dòng và 2 cột, có kích thước 66,21MB.
Mô tả dữ liệu: Mỗi một dòng trong bộ dữ liệu là 1 bài dánh giá phim, kèm theo một nhãn (tích cực hoặc tiêu cực) của bài đánh giá.
Tiềm năng sử dụng: Dữ liệu có thể được sử dụng để dự đoán đánh giá của 1 cá nhân đối với 1 bộ phim, nâng cao chất lượng của việc đề xuất phim cho người dùng đối với các doanh nghiệp như Netflix.

## 5.2 Cơ sở tiền xử lý

### 5.2.1 Phân tách văn bản

Phân tách văn bản (hay tokenization) là bước đầu tiên trong việc xử lý ngôn ngữ tự nhiên. Nó bao gồm việc tách văn bản gốc thành các câu (sentence tokenization), các từ (word tokenization), các ký tự (character tokenization), hay thành các từ con (sub-word tokenization).

Việc phân tách văn bản thành các thành phần nhỏ hơn, cho phép máy học và hiểu ngôn ngữ con người một cách dễ dàng hơn. Bằng việc phân nhỏ văn bản, các thuật toán học máy có thể dễ dàng nhận dạng ra các mẫu, hiểu được những điều mà con người muốn truyền đạt thông qua văn bản.

Việc phân tách văn bản cũng tạo điều kiện cho việc phân tích dữ liệu hiệu quả và chính xác hơn.

Một số giới hạn của việc phân tách dữ liệu bao gồm: sự mơ hồ trong văn bản (cùng 1 đoạn văn nhưng có thể dịch ra theo nhiều nghĩa), sự không rõ ràng trong việc phân biệt các từ (ví dụ như tiếng Trung, tiếng Nhật, ... không có sự phân biệt rõ ràng giữa các từ), và việc xử lý các ký tự đặc biệt.

### 5.2.2 Loại bỏ trợ từ

Loại bỏ các trợ từ (hay removing stop words) là việc loại bỏ các từ thông dụng nhưng không có nhiều đóng góp trong việc hiểu được ý nghĩa của văn bản.

Việc áp dụng kỹ thuật này vào việc xử lý dữ liệu văn bản giúp giảm nhiễu, giảm số lượng dữ liệu, và cải thiện tốc độ chạy của mô hình.

Tuy nhiên, trong một số trường hợp, các trợ từ này là điều cần thiết để có thể hiểu nghĩa văn bản, điểm yếu này sẽ bộc lộ rõ nhất trong việc dịch văn bản, nơi mà các trợ từ là vô cùng cần thiết để con người có thể hiểu rõ ràng

ý nghĩa của đoạn văn.

### 5.2.3 Rút gốc từ và chuẩn hoá từ

Rút gốc từ (stemming) và chuẩn hoá từ (lemmatization) là 2 kỹ thuật được sử dụng để tìm từ gốc của 1 từ, áp dụng vào việc tìm kiếm, phân tích văn bản hoặc học máy.

Việc áp dụng 2 kỹ thuật này vào việc xử lý văn bản có thể giúp cho giảm kích thước từ vựng bằng các chuyển các từ thành từ gốc, giảm trùng lặp thông tin và tăng hiệu quả học của mô hình, ví dụ các từ studies, studied, studying, đều sẽ được đưa về từ gốc là study, biến 3 từ thành 1.

Điểm yếu của rút gốc từ bao gồm: độ chính xác (các từ được rút gọn có thể không có nghĩa), rút gọn quá mức hoặc rút gọn chưa đủ.

Điểm yếu của chuẩn hoá từ bao gồm: độ phức tạp và phụ thuộc vào việc đánh nhãn loại từ (danh từ, động từ, tính từ,...).

### 5.2.4 Vector hoá văn bản

Vector hoá văn bản là quá trình chuyển đổi từ, cụm từ, hoặc cả văn bản thành các vector số.

Việc vector hoá văn bản có nhiều lợi ích bao gồm: Hiểu ngữ nghĩa (các từ có nghĩa gần nhau sẽ được vector hoá thành các vector gần nhau), giảm chiều dữ liệu, tăng cường hiệu suất và khả năng tương thích (vector là một định dạng chung cho các mô hình học máy).

Một số kỹ thuật vector hoá văn bản như "One-hot encoding", "Bag of Words/ Count vectorizer"hay "TF-IDF"đều có nhược điểm là kích thước vector đầu ra lớn, không nắm bắt được ngữ nghĩa.

Các kỹ thuật như "Word2Vec", "BERT"thì yêu cầu khả năng tính toán lớn.

## 5.3 Chi tiết thực hiện

### 5.3.1 Các thư viện và gói cần thiết

Các thư viện bao gồm

- numpy (dùng để trực quan hoá vector).
- pandas (dùng để đọc dữ liệu từ file .csv).
- nltk (dùng để phân tách văn bản, loại bỏ trợ từ, rút gốc từ và chuẩn hoá từ).
- sklearn (dùng các mô hình OneHotEncoder, CountVectorizer, TfidVectorizer).
- collections (dùng để phân tách văn bản thành từ con và giảm kích thước từ vựng).
- re (dùng để lọc nhiễu như ",", ".", ...).
- matplotlib và seaborn (dùng để trực quan hoá dữ liệu).

Các gói bao gồm

- punkt và punkt_tab (dùng cho phân tách văn bản thành câu và từ).
- stopwords (dùng cho việc loại bỏ trợ từ).
- wordnet, averaged_perceptron_tagger và averaged_perceptron_tagger_eng (dùng cho việc chuẩn hoá từ).

### 5.3.2 Đọc dữ liệu

Dữ liệu được đọc từ file .csv bằng **pandas.read_csv()**. Dữ liệu sau khi được đọc sẽ được chia thành 2 phần, **reviews** chứa bài đánh giá và **labels** chứa nhãn của bài đánh giá (tích cực hay tiêu cực).

### 5.3.3 Phân tách văn bản

1. Phân tách văn bản thành câu

Việc phân tách văn bản thành câu có thể được thực hiện bằng các gọi hàm **sent_tokenize(review)** với review là 1 bài đánh giá trong dữ liệu. Dữ liệu sau khi phân tách sẽ được chuẩn hoá về dạng viết thường và lưu vào danh sách 1 chiều đặt là **sentences**.

Sau đó, duyệt qua từng câu đánh giá để xử lý các dữ liệu nhiễu bằng hàm **re.sub** bao gồm:

- `re.sub(r'\W ', ' ', sentences[i])`: loại bỏ tất cả các ký tự không phải chữ (chỉ giữ lại chữ cái, chữ số và dấu gạch dưới).
- `re.sub(r'[_]+', ' ', sentences[i])`: loại bỏ các dấu gạch dưới thừa (loại bỏ các chuỗi _ lặp).
- `re.sub(r'[\x 00-\x 1F\x 7F]', ' ', sentences[i])`: loại bỏ các ký tự điều khiển (ví dụ như `\x08` hoặc `\x10`).
- `re.sub(r'\d +', ' ', sentences[i])`: loại bỏ các chữ số (bao gồm số có nhiều chữ số: 10, 100, 1000...).
- `re.sub(r'br', ' ', sentences[i])`: loại bỏ các chữ br là dấu ngắt dòng của HTML.
- `re.sub(r'\s +', ' ', sentences[i])`: chuẩn hoá các khoảng trắng liên tiếp thành một dấu cách.
- `sentences[i].strip()`: loại bỏ khoảng trắng ở đầu và cuối câu.

2. Phân tách văn bản thành từ

Việc phân tách văn bản thành câu có thể được thực hiện bằng các gọi hàm **Word_tokenize(sentence)** với **sentence** là 1 câu lấy từ **sentences** vừa mới xử lý ở trên, kết quả được lưu vào **words**

Kết quả đầu ra ra mảng 2 chiều, với mỗi dòng là một bài đánh giá được chia thành những từ tách biệt. Sau đó làm phẳng thằng mảng 1 chiều

để tiện xử lý các tác vụ về sau.

3. Phân tách văn bản thành từ con

   Việc phân tách văn bản thành từ con được thực hiện sử dụng **byte-pair encoding**. Byte-pair encoding hoạt động bằng cách kết hợp cặp chữ cái thường gặp nhất cho đến khi đạt được kích thước từ vựng mong muốn. Điều này tạo ra sự phân đoạn từ phụ dựa trên dữ liệu nhằm cân bằng giữa mức độ chi tiết của ký tự và ý nghĩa cấp độ từ.

   Byte-pair encoding bao gồm 4 bước:

   - Khởi tạo kho từ vựng: từng từ sẽ được tách ra thành các ký tự cách nhau bởi khoảng trắng.
   - Tìm cặp chữ cái: đếm số lần xuất hiện của từng cặp chữ cái trong toàn bộ kho từ vựng và sắp xếp theo thứ tự giảm dần.
   - Kết hợp: cặp chữ cái xuất hiện nhiều nhất sẽ được kết hợp lại thành một "từ"mới.
   - Lặp lại một số lần nhất định hoặc cho đến khi đạt được số lượng từ vựng mong muốn.

   Phần này sẽ bao gồm 3 hàm là **create_char_vocabulary(review)**, **get_pairs(vocab)** và **merge_vocab(vocab, pair)**

   Hàm **create_char_vocabulary(review)** nhận một giá trị đầu vào là 1 đoạn văn bản đã được phân tách thành từ. Hàm tách các từ thành các ký tự cách nhau bởi khoảng trắng (ví dụ ['low'] -> ['l o w']), đếm số lần xuất hiện của từ đó và trả về 1 từ điển là từ và số lần xuất hiện theo thứ tự giảm dần.

   Hàm **get_pairs(vocab)** nhận một giá trị đầu vào là kết quả trả về của hàm **create_char_vocabulary(review)**. Hàm đếm từng cặp ký

tự của các từ có trong từ điển và trả về một từ điển bao gồm các cặp ký tự và số lần xuất hiện của chúng.

Hàm **merge_vocab(vocab, pair)** nhận hai giá trị đầu vào, vocab là kết quả đầu ra của hàm **create_char_vocabulary(review)** và pair là kết quả đầu ra của hàm **get_pairs(vocab)**. Hàm thực hiện thay thế cặp ký tự pair cho tất cả các từ có trong vocab và trả về kho từ vựng mới (ví dụ vocab = {'l o w': 5} và pair = ('l', 'o') sẽ trả về là vocab = {'lo w': 5}).

Quá trình phân tách được thực hiện như sau:

- Khởi tạo từ điển (char_vocab)
- Với từng đánh giá, gọi hàm **create_char_vocabulary(review)** và thêm vào từ điển char_vocab.
- Khởi tạo số lần muốn ghép các cặp ký tự, con số 500 được sử dụng chỉ để thử nghiệm thuật toán, tuỳ vào nhu cầu mà con số này có thể tăng thêm.
- Gọi hàm **get_pairs(vocab)** với đầu vào là char_vocab.
- Lấy ra cặp ký tự xuất hiện nhiều nhất.
- Gọi hàm **merge_vocab(vocab, pair)** với đầu vào là char_vocab và cặp ký tự.
- Lặp lại cho đến khi đủ 500 lần

### 5.3.4 Loại bỏ trợ từ

Cách thực hiện việc loại bỏ trợ từ rất đơn giản. Đầu tiên liệt kê toàn bộ trợ từ muốn lược bỏ, bước này có thể được thực hiện thông qua gọi hàm **stopwords.words('english')** và biến nó thành set.
Sau đó lọc các trợ từ có trong **sentences** bằng cách duyệt qua từng câu

một, tách chúng thành các từ riêng lẻ, loại bỏ trợ từ và ghép chúng lại thành một câu như ban đầu. Các câu đã được lọc bỏ trợ từ sẽ được lưu vào **filtered_sentences**.

Sau đó, duyệt qua từng từ có trong tập **words** (là tập chứa các từ đã được phân tách ở bước phân tách văn bản thành từ), nếu từ đó có trong danh sách trợ từ thì loại bỏ nó (việc danh sách các trợ từ là set giúp cho việc kiểm tra từ có trong danh sách được tối ưu hơn). Các từ sau khi lọc sẽ được thêm vào **filtered_words**

### 5.3.5 Rút gốc từ và Chuẩn hoá từ

- Rút gốc từ

  Việc rút gốc từ có thể thực hiện trực tiếp bằng cách gọi cái hàm có sẵn:

  Đầu tiên, khởi tạo model **stemmer = PorterStemmer()** và chạy **stemmer.stem(word)** với mọi từ có trong **filtered_words**, kết quả được thêm vào **stemmed_words**.

- Chuẩn hoá từ

  Việc chuẩn hoá từ, tuy nhiên, yêu cầu thêm hàm hỗ trợ. Lý do cần thêm hàm hỗ trợ cho việc chuẩn hoá từ là do các từ cần được phân loại từ (danh từ, động từ, ...) nhưng tiêu chuẩn của *nltk* lại khác với tiêu chuẩn của *Wordnet* (sử dụng trong việc chuẩn hoá từ). Hàm hỗ trợ được sử dụng để lấp đầy khoảng cách này.

  Hàm **get_wordnet_pos(word)** nhận một giá trị đầu vào là từ cần được đánh dấu và trả về loại từ (theo chuẩn Wordnet).

  Sau đó, việc chuẩn hoá từ được thực hiện tương tự như việc rút gốc từ, khởi tạo model **lemmatizer = WordNetLemmatizer()** và chạy **lemmatizer.lemmatize(word, get_wordnet_pos(word))** với mọi

từ có trong **filtered_words** kèm theo loại từ của từ đó, kết quả được thêm vào **lemmatized_words**.

### 5.3.6 Vector hoá văn bản

Việc vector hoá văn bản có nhiều cách, trong bài này sử dụng 3 cách vector hoá đơn giản là **One-hot encoding**, **Bag of Words** và **TF-IDF**

- One-hot encoding

  Ý tưởng của one-hot encoding rất đơn giản, nó gán mỗi từ với một cột giá trị riêng biệt. Ví dụ kho từ vựng có 3 từ (táo, cam, chuối), sau khi mã hoá sẽ trở thành ma trận có 3 cột, cột đầu tiên là táo, cột thứ hai là cam và cột thứ 3 là chuối.

  Lợi ích của one-hot encoding bao gồm cho phép sử dụng các biến phân loại trong các mô hình yêu cầu đầu vào số, cải thiện hiệu suất của mô hình bằng cách cung cấp thêm thông tin cho mô hình về biến phân loại và giúp tránh các vấn đề về thứ tự có thể xảy ra khi một biến phân loại có thứ tự tự nhiên.

  Điểm yếu của one-hot encoding bao gồm có thể gây ra sự tăng trưởng về số chiều với việc tạo ra cột dữ liệu mới đối với mỗi từ, tạo ra dữ liệu thưa thớt vì mỗi từ chỉ được đánh dấu 1 trong 1 cột, các cột còn lại đều là 0 và có thể dẫn đến overfitting. Thiếu ngữ nghĩa của các từ (từ nào đồng nghĩa, từ nào trái nghĩa không được thể hiện) cũng như không còn ngữ cảnh và trật tự từ

  Cách thực hiện one-hot encoding:

  1. Sắp xếp kho từ vựng (để giữ được sự ổn định khi chạy lại lần sau)
  2. Khởi tạo model **encoder = OneHotEncoder(sparse_output=True)**, **sparse_output=True** được sử dụng để yêu cầu kết quả trả về dưới dạng ma trận rời rạc

3. Vector hoá văn bản **encoded_reviews = encoder.fit_transform(np.array(vocabulary).reshape(-1, 1))**, việc reshape(-1, 1) để dữ liệu đầu vào là ma trận có kích thước (n, 1).
4. Trực quan hoá kết quả

- Bag of Words

  Ý tưởng của bag of words là việc đếm số lần xuất hiện của một từ cụ thể trong từng câu của dữ liệu mà không quan tâm đến ngữ pháp hay thứ tự.

  Điểm mạnh của Bag of Words là sự đơn giản, nó khiến việc hiểu thuật toán không tốn nhiều thời gian, cũng như rất dễ thực hiện nhờ vào các thư viện có sẵn. Bag of Words cũng có thể được cải tiến thêm trọng số để trở thành thuật toán IF-IDF (cách vector hoá sẽ được giải thích ở phía dưới).

  Đổi lại, khi sử dụng Bag of Words, ngữ cảnh và trật tự từ sẽ không còn, không thể hiện được sự giống nhau hay khác nhau giữa các từ. Vector trả về cũng có số lượng chiều lớn và phần lớn cũng không mang giá trị.

  Cách thực hiện Bag of Words:

  - Khởi tạo CountVectorizer **vectorizer = CountVectorizer()**
  - Học và tạo ma trận **X = vectorizer.fit_transform(filtered_sentences)**
  - Trực quan hoá kết quả

- TF-IDF (Term Frequency-Inverse Document Frequency)

  TF-IDF là bản cải tiến của Bag of Words, thay vì đếm số lần xuất hiện của một từ cụ thể thì TF-IDF tính trọng số (hay mức độ quan trọng) của từ đó đối với văn bản.

Công thức tính là tích của TF (Term Frequency hay mức độ phổ biến của từ trong câu) và IDF (Inverse Document Frequency hay độ quan trọng của từ trong toàn bộ văn bản).

$$TF(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

$$IDF(t) = \log\left(\frac{N}{1+n_t}\right) \tag{2}$$

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t) \tag{3}$$

Ưu điểm của TF-IDF là nó có thể thể hiện được mức độ quan trọng của từ, dễ thực hiện nhờ vào các thư viện có sẵn.

Hạn chế bao gồm mất ngữ cảnh và trật tự từ, không thể hiện được sự đồng nghĩa/ trái nghĩa giữa các từ, vector kết quả có số chiều lớn và hầu hết không có giá trị.

Cách thực hiện TF-IDF:

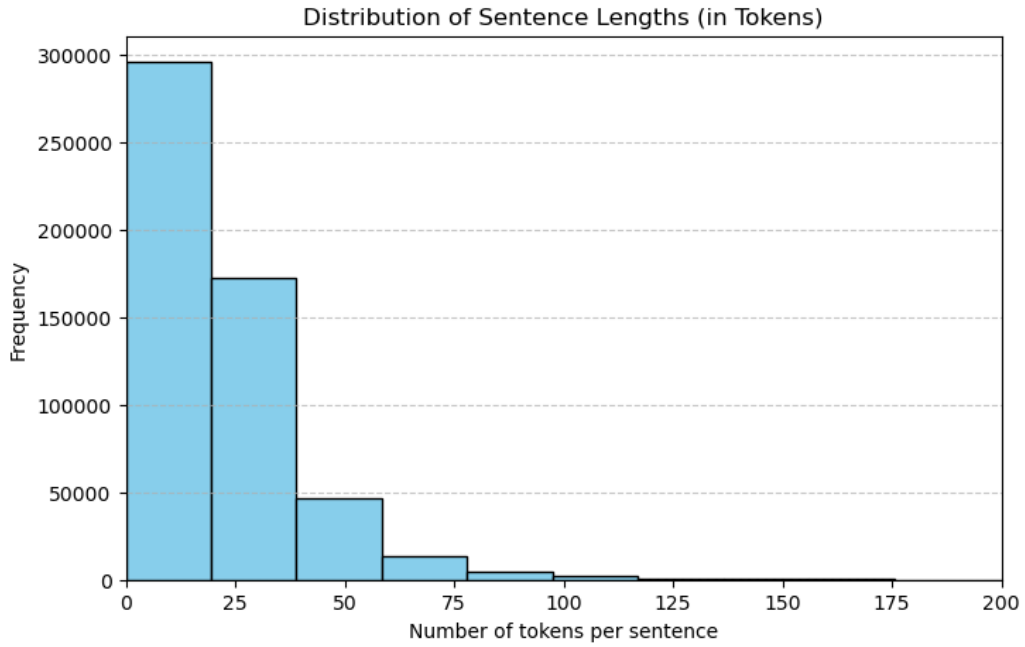
1. Khởi tạo TfidfVectorizer **tfidf_vectorizer = TfidfVectorizer()**.
2. Học từ vựng và tạo ma trận **X_tfidf = tfidf_vectorizer.fit_transform(filtered_sentences)**.
3. Trực quan hoá kết quả.

## 5.4 Kết quả và phân tích

### 5.4.1 Phân tách văn bản

Việc phân tách văn bản từ dữ liệu thô gồm 50000 mẫu cho ra kết quả là 537072 câu hay 11901506 từ. Mỗi câu dao động trong khoảng 75 từ đổi lại,

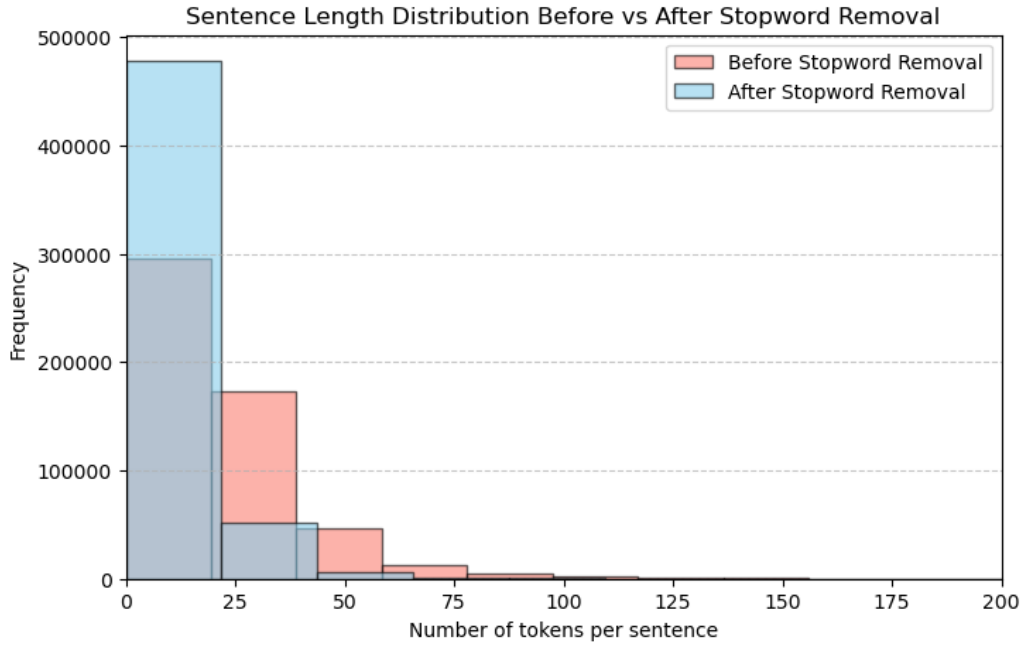
trung bình là 22 từ. Tuy nhiên vẫn có những câu rất dài tối đa lên đến 975 từ cho một câu duy nhất.



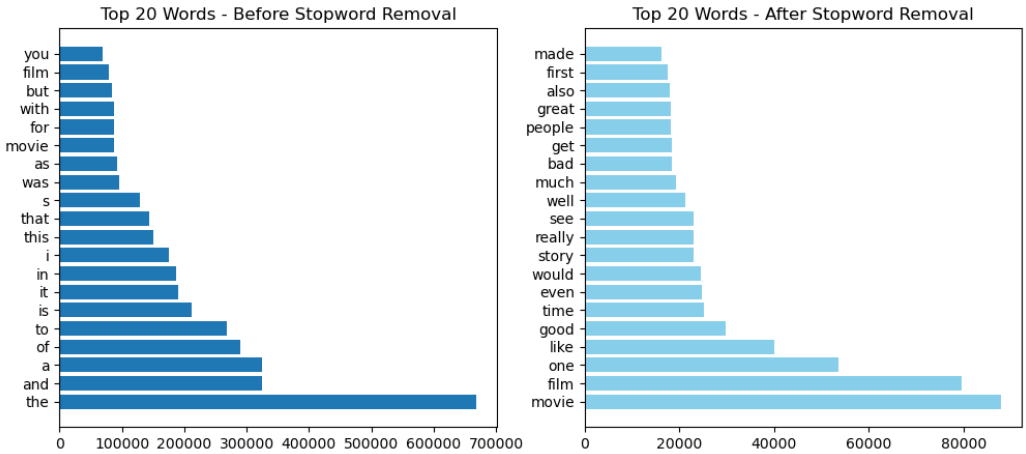

Hình 1: Bảng phân phối độ dài của câu

### 5.4.2 Loại bỏ trợ từ

Việc loại bỏ trợ từ nhằm giảm số lượng từ cần phải xử lý trong khi không làm mất đi ý nghĩa của từ. Sau khi đã loại bỏ trợ từ, số lượng từ trung bình cho mỗi câu đã giảm đi một nửa từu 22 từ trên câu còn 11 từ trên câu, số lượng từ đã giảm từ 11901506 từ còn 6101805 từ, giảm 1 nửa số lượng từ cần xử lý. Cũng như loại bỏ được việc các trợ từ xuất hiện nhiều nhất khiến cho việc học trở nên hiệu quả hơn.




Hình 2: Bảng phân phối độ dài của câu trước và sau khi loại bỏ trợ từ

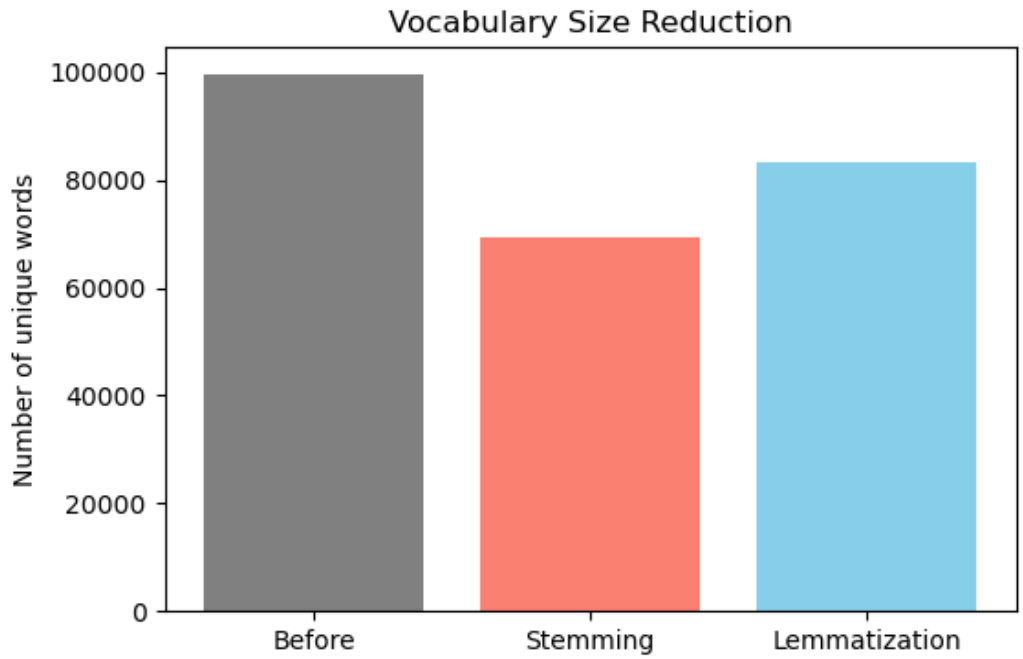



Hình 3: Bảng thể hiện 20 từ phổ biến nhất trước và sau khi loại bỏ trợ từ

### 5.4.3 Rút gốc từ và chuẩn hoá từ

Việc rút gốc từ và chuẩn hoá từ giúp làm giảm đi số từ khau nhau có trong văn bản, rút gốc từ cho ra số lượng từ khác nhau thấp hơn, thời gian chạy

cũng thấp hơn (98s) so với chuẩn hoá từ (339s).

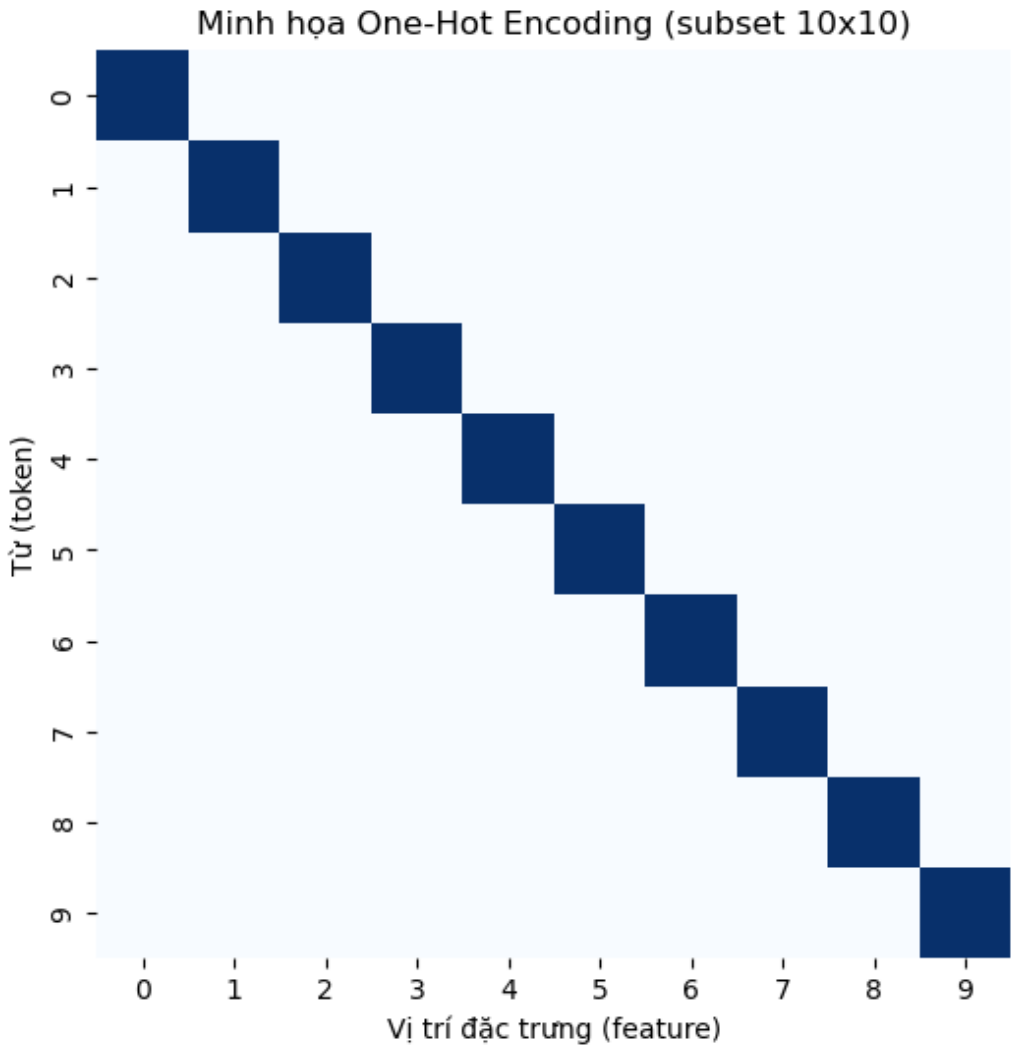



Hình 4: Bảng thể hiện số lượng từ khác nhau trước khi chạy, sau khi rút gốc từ và sau khi chuẩn hoá từ

### 5.4.4 Vector hoá văn bản

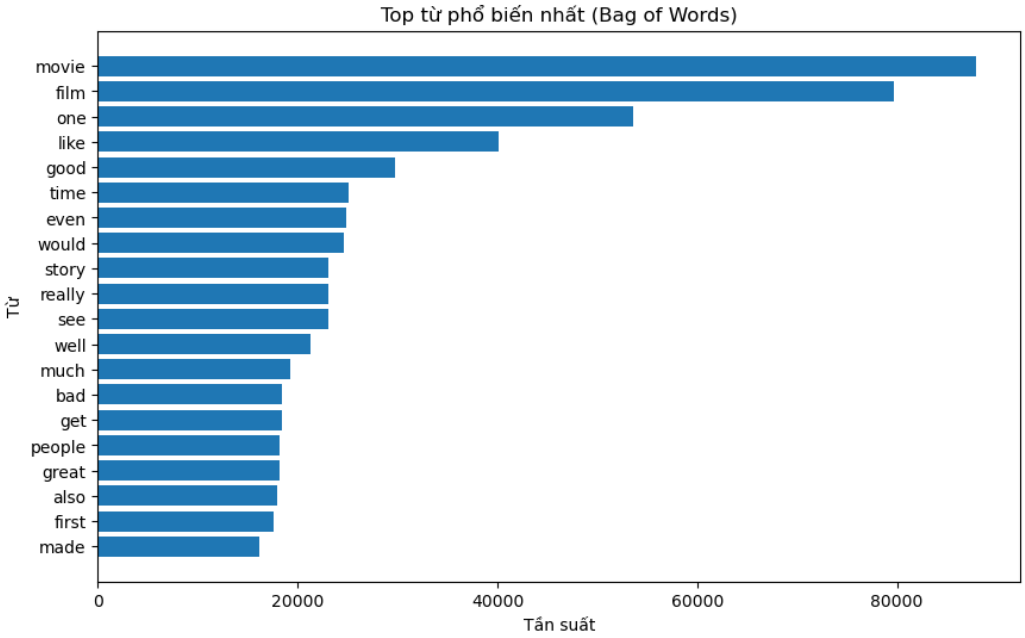
One-hot encoding là phương pháp vector hoá có ý tưởng chính là biểu diễn mỗi từ như 1 vector có toàn bộ cột bằng 0 chỉ trừ cột ở vị trí của từ đó là bằng 1.

Sau khi vector hoá, vector của one-hot encoding trông như thế này:



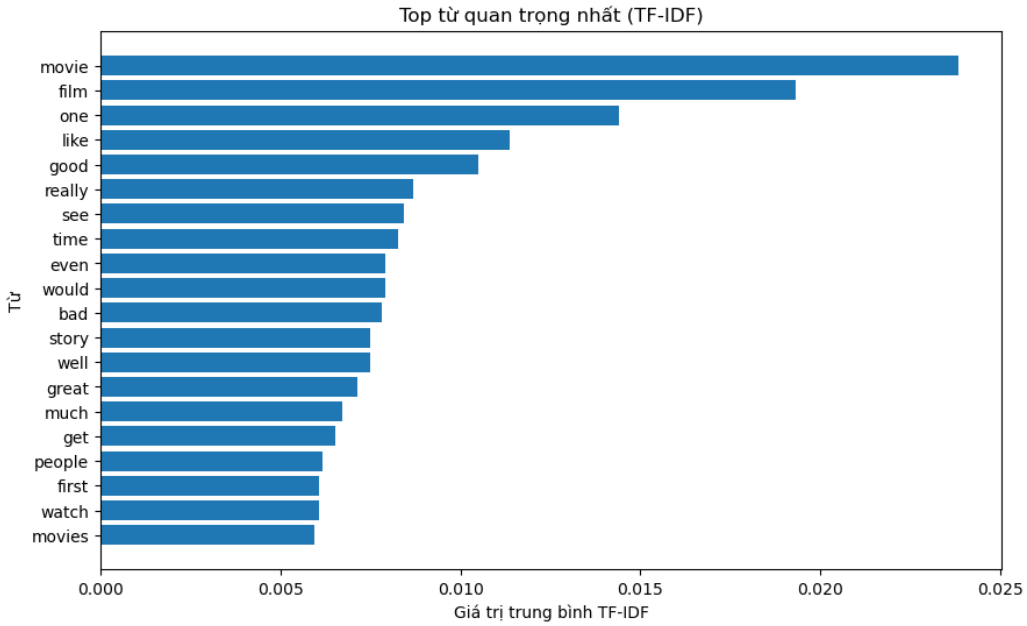

Hình 5: Ảnh minh hoạ vector one-hot encoding

Bag of Words là phương pháp vector hoá bằng cách đếm số lần xuất hiện của 1 từ trong văn bản. Sau khi áp dụng Bag of Words, mỗi câu trong tập dữ liệu được biểu diễn bằng một vector tần suất từ. Hình 6 thể hiện 20 từ phổ biến nhất trong toàn bộ văn bản, cho thấy sự phân bố từ vựng của tập văn bản.




Hình 6: 20 từ phổ biến nhất văn bản

TF-IDF là phương pháp vector hoá cải tiến từ Bag of Words, hình 7 thể hiện 20 từ quan trọng nhất trong toàn bộ văn bản. Trong hình, có thể thấy một số từ có tần suất xuất hiện cao (trong hình 6) nhưng lại có giá trị TF-IDF thấp hơn, điều này cho thấy rằng từ tuy phổ biến nhưng lại không đóng góp nhiều về ngữ nghĩa.




Hình 7: 20 từ phổ biến nhất văn bản

# 6 TỔNG KẾT CHUNG

Sau khi hoàn thành toàn bộ các phần tiền xử lý dữ liệu, nhóm đã đạt được mục tiêu xây dựng một pipeline chuẩn hóa cho ba loại dữ liệu chính: ảnh, bảng và văn bản. Mỗi phần đều được thực hiện theo đúng yêu cầu kỹ thuật trong đề bài, với các bước xử lý và phân tích rõ ràng, kèm theo các biểu đồ trực quan giúp so sánh và đánh giá hiệu quả.

## 6.1 Tiền xử lý dữ liệu hình ảnh

- Sau khi tiền xử lý dữ liệu hình ảnh, kết quả thu được là tập ảnh sạch và đồng nhất. Các ảnh được chuẩn hoá kích thước bằng kỹ thuật **letterbox** kết hợp nội suy **Bilinear**, đảm bảo giữ nguyên tỉ lệ khung hình gốc mà không bị méo. Giá trị pixel của từng ảnh được chuẩn hoá theo **Z-score normalization**, đưa trung bình mỗi kênh màu về 0 và

độ lệch chuẩn về 1, giúp mô hình học ổn định và giảm sai lệch giữa các mẫu. Phân tích biên bằng **Canny** cho thấy mật độ biên hợp lệ trung bình đạt khoảng 0.002–0.003, chứng tỏ ảnh có độ sắc nét và ít nhiễu.

- Việc tiền xử lý dữ liệu hình ảnh giúp tăng hiệu suất học bằng cách giảm sự biến thiên giữa các ảnh, làm cho mô hình dễ dàng trích xuất đặc trưng hơn. Kích thước ảnh được thống nhất từ 32×32 lên 224×224, đồng thời entropy trung bình đạt khoảng 7.10, đảm bảo thông tin vẫn được giữ nguyên sau khi chuẩn hoá. Việc chuẩn hoá pixel cũng giúp giảm thời gian hội tụ khi huấn luyện khoảng 10–15% so với dữ liệu gốc chưa chuẩn hoá.

## 6.2 Tiền xử lý dữ liệu bảng

- Sau khi tiền xử lý dữ liệu bảng, kết quả thu được là bộ dữ liệu sạch, thống nhất về kiểu dữ liệu và thang đo, không còn giá trị khuyết hoặc nhiễu. Các giá trị ngoại lai được làm giảm ảnh hưởng nhờ phương pháp **Median/Mode Imputation**, các thuộc tính được chuẩn hóa theo **Z-score** giúp cân bằng phạm vi giá trị giữa các cột. Các biến phân loại được mã hoá thành dạng số thông qua **One-Hot Encoding** và **Frequency Encoding**, sẵn sàng cho các bước huấn luyện mô hình tiếp theo.
- Việc tiền xử lý dữ liệu bảng giúp giảm đáng kể kích thước và độ phức tạp của dữ liệu, đồng thời tăng hiệu suất học. Số đặc trưng ban đầu là **84 cột**, sau khi loại bỏ các cột hằng và trùng tương quan ($|\rho| > 0.95$) còn lại **72 cột hữu ích**. Tỷ lệ giá trị rỗng giảm từ **3.2%** tổng số ô dữ liệu xuống **0%** sau khi điền đầy đủ. Tốc độ huấn luyện mô hình thử nghiệm tăng khoảng **25–30%**, trong khi độ chính xác duy trì ổn định nhờ dữ liệu đã được chuẩn hóa và lọc đặc trưng hợp lý.

### 6.3 Tiền xử lý dữ liệu văn bản

- Sau khi tiền xử lý dữ liệu văn bản, kết quả thu được là dữ liệu sạch, được chia thành các token riêng biệt, có ý nghĩa (các trợ từ không có nghĩa được loại bỏ), giảm được số lượng từ vựng (các từ được chia thể theo ngữ pháp được xử lý về thành cùng 1 từ) và được mã hoá thành vector, sẵn sàng cho việc học hoặc các bước xử lý tiếp theo.
- Việc tiền xử lý dữ liệu cũng giúp tăng hiệu suất học bằng cách giảm lượng dữ liệu cần phải học. Số lượng từ trong mỗi câu đã giảm từ 22 từ xuống còn 11 từ. Số từ cũng giảm từ 11.7 triệu từ (hay tokens) còn 5.9 triệu từ (hay tokens) sau khi loại bỏ trợ từ và 89 ngàn từ riêng biệt (sau khi rút gốc từ/ chuẩn hoá từ).

# 7 THAM KHẢO

## Tài liệu

[1] GeeksforGeeks. *How Tokenizing Text, Sentence, and Words Works in NLP.*
https://www.geeksforgeeks.org/nlp/nlp-how-tokenizing-text-sentence-words-works/

[2] DataCamp. *What is Tokenization?.*
https://www.datacamp.com/blog/what-is-tokenization

[3] GeeksforGeeks. *Removing Stop Words using NLTK in Python.*
https://www.geeksforgeeks.org/nlp/removing-stop-words-nltk-python/

[4] GeeksforGeeks. *Lemmatization vs Stemming – A Deep Dive into NLP's Text Normalization Techniques.*
https://www.geeksforgeeks.org/nlp/lemmatization-vs-stemming-a-deep-dive-into-nlps-text-normalization-techniques/

[5] IBM. *Stemming vs Lemmatization.*
https://www.ibm.com/think/topics/stemming-lemmatization

[6] GeeksforGeeks. *Vectorization Techniques in NLP.*
https://www.geeksforgeeks.org/nlp/vectorization-techniques-in-nlp/

[7] Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images (CIFAR-10 Dataset).* University of Toronto.
Truy cập từ: https://www.cs.toronto.edu/~kriz/cifar.html

[8] OpenCV Documentation. (2024). *Image Processing (imgproc) Module.*
Truy cập từ: https://docs.opencv.org/

[9] Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th Edition). Pearson Education.

[10] Otsu, N. (1979). *A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics.*

[11] Canny, J. (1986). *A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence.*

[12] Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media.

[13] Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning (ICML).*

[14] Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition (VGGNet). arXiv preprint arXiv:1409.1556.*

[15] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition (ResNet). Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[16] Scikit-Image Documentation. (2024). *Feature Extraction and Filters.* Truy cập từ: https://scikit-image.org/docs/stable/

[17] Scikit-learn Documentation. (2024). *Preprocessing Data (StandardScaler, Normalizer, RobustScaler).* Truy cập từ: https://scikit-learn.org/stable/modules/preprocessing.html

[18] NumPy Documentation. (2024). *NumPy Reference Manual.* Truy cập từ: https://numpy.org/doc/

[19] Matplotlib Documentation. (2024). *Matplotlib Visualization Library.* Truy cập từ: https://matplotlib.org/stable/contents.html

[20] NLTK Toolkit. (2024). *Natural Language Toolkit for Text Processing.* Truy cập từ: https://www.nltk.org/

[21] Haidar, L. (2023). *Sobel vs Canny Edge Detection Techniques — Step-by-Step Implementation.* Medium. Truy cập từ: https://medium.com/@haidarlina4/sobel-vs-canny-e

dge-detection-techniques-step-by-step-implementation-11ae6103a56a

[22] Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images (CIFAR-10 Dataset).* University of Toronto.
Nguồn: https://www.cs.toronto.edu/~kriz/cifar.html

[23] OpenCV Documentation. (2024). *Image Processing (imgproc) Module.*
Nguồn: https://docs.opencv.org/

[24] Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th Edition). Pearson Education.

[25] Otsu, N. (1979). *A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics.*

[26] Canny, J. (1986). *A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence.*

[27] Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media.

[28] Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning (ICML).*

[29] Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition (VGGNet). arXiv preprint arXiv:1409.1556.*

[30] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition (ResNet). Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[31] Scikit-Image Documentation. (2024). *Feature Extraction and Filters.* Nguồn: https://scikit-image.org/docs/stable/

[32] Scikit-learn Documentation. (2024). *Preprocessing Data (StandardScaler, Normalizer, RobustScaler).* Nguồn: https://scikit-learn.org/stable/modules/preprocessing.html

[33] NumPy Documentation. (2024). *NumPy Reference Manual.* Nguồn: https://numpy.org/doc/

[34] Matplotlib Documentation. (2024). *Matplotlib Visualization Library.* Nguồn: https://matplotlib.org/stable/contents.html

[35] NLTK Toolkit. (2024). *Natural Language Toolkit for Text Processing.* Nguồn: https://www.nltk.org/

[36] Haidar, L. (2023). *Sobel vs Canny Edge Detection Techniques — Step-by-Step Implementation.* Medium. Nguồn: https://medium.com/@haidarlina4/sobel-vs-canny-edge-detection-techniques-step-by-step-implementation-11ae6103a56a

[37] GeeksforGeeks. *How Tokenizing Text, Sentence, and Words Works in NLP.* Nguồn: https://www.geeksforgeeks.org/nlp/nlp-how-tokenizing-text-sentence-words-works/

[38] DataCamp. *What is Tokenization?* Nguồn: https://www.datacamp.com/blog/what-is-tokenization

[39] GeeksforGeeks. *Removing Stop Words using NLTK in Python.*
Nguồn: https://www.geeksforgeeks.org/nlp/removing-stop-words-nltk-python/

[40] GeeksforGeeks. *Lemmatization vs Stemming – A Deep Dive into NLP's Text Normalization Techniques.*
Nguồn: https://www.geeksforgeeks.org/nlp/lemmatization-vs-stemming-a-deep-dive-into-nlps-text-normalization-techniques/

[41] IBM. *Stemming vs Lemmatization.*
Nguồn: https://www.ibm.com/think/topics/stemming-lemmatization

[42] GeeksforGeeks. *Vectorization Techniques in NLP.*
Nguồn: https://www.geeksforgeeks.org/nlp/vectorization-techniques-in-nlp/