ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo cái bài thực hành 2

Đề tài: Hồi quy Softmax cho bài toán nhận diện chữ số viết tay

---

Môn học: Nhập môn học máy

Sinh viên thực hiện:

Tô Hữu Danh (23127336)

Nguyễn Đăng Hưng (23127050)

Lê Phú Cường (23127164)

Nguyễn Bá Đăng Khoa (23127392)

Giáo viên hướng dẫn:

Thầy Võ Nhật Tân

Ngày 10 tháng 12 năm 2025

Mục lục

Danh sách bảng

Danh sách hình vẽ

1 Mô tả và tiền xử lý dữ liệu

1.1 Mô tả dữ liệu

Hệ thống được xây dựng để vận hành trên bộ cơ sở dữ liệu MNIST, một tập dữ liệu chuẩn mực trong lĩnh vực thị giác máy tính bao gồm các hình ảnh chữ số viết tay. Dữ liệu được tổ chức lưu trữ cục bộ trong thư mục `data` và được truy xuất thông qua các đường dẫn tệp tĩnh được định nghĩa trong cấu hình hệ thống.

1.1.1 Cấu trúc tập tin

Bộ dữ liệu bao gồm bốn tệp nhị phân riêng biệt theo định dạng IDX, tương ứng với hai tập dữ liệu con: tập huấn luyện (training set) và tập kiểm thử (test set). Cụ thể bao gồm:

- **Tập huấn luyện (train):** Bao gồm tệp hình ảnh (`train-images.idx3-ubyte`) và tệp nhãn tương ứng (`train-labels.idx1-ubyte`).
- **Tập kiểm thử (test):** Bao gồm tệp hình ảnh (`t10k-images.idx3-ubyte`) và tệp nhãn tương ứng (`t10k-labels.idx1-ubyte`).

1.1.2 Định dạng nội tại và Metadata (Internal Format and Metadata)

Dữ liệu không được lưu trữ dưới dạng các tệp ảnh thông thường (như PNG hay JPEG) mà dưới dạng chuỗi byte (byte stream). Quá trình đọc dữ liệu được thực hiện bằng cách giải mã (unpack) cấu trúc nhị phân của tệp header:

- **Số định danh (Magic Number):** Mỗi tệp bắt đầu bằng 4 byte đầu tiên chứa "magic number" để xác thực định dạng. Hệ thống kiểm tra giá trị này để đảm bảo tính toàn vẹn: giá trị 2049 xác định tệp chứa nhãn và giá trị 2051 xác định tệp chứa dữ liệu hình ảnh.
- **Thông tin kích thước:** Ngay sau magic number, header cung cấp thông tin về số lượng mẫu (size), số hàng (rows) và số cột (cols). Các thông số này được đọc thông qua module `struct` để thiết lập kích thước mảng dữ liệu chính xác.

1.1.3 Đặc tả chi tiết dữ liệu

Sau khi giải mã, dữ liệu được chuyển đổi thành các cấu trúc đại số tuyến tính để phục vụ tính toán:

- **Dữ liệu hình ảnh (Image Data):**
  - **Kích thước gốc:** Mỗi mẫu dữ liệu thô sau khi đọc được tái cấu trúc (reshape) thành một ma trận hai chiều với kích thước cố định là 28×28 pixel.
  - **Kiểu dữ liệu:** Trong quá trình nạp vào bộ nhớ chính (RAM), các ma trận này được ép kiểu sang định dạng `numpy.uint8` (số nguyên không dấu 8-bit). Điều này đồng nghĩa mỗi điểm ảnh (pixel) mang một giá trị nguyên nằm trong khoảng $[0, 255]$, đại diện cho độ đậm nhạt của nét viết.
- **Dữ liệu nhãn (Label Data):**
  - Chứa các giá trị số nguyên tương ứng với chữ số trong hình ảnh.
  - Hệ thống được cấu hình tham số `n_classes = 10`, xác định không gian đầu ra bao gồm 10 lớp phân loại (tương ứng các chữ số từ 0 đến 9).

1.2 Tiền xử lý dữ liệu

Quy trình xử lý dữ liệu được thiết kế theo mô hình đường ống (pipeline), bắt đầu từ việc đọc dữ liệu nhị phân thô, chuẩn hóa, trích xuất đặc trưng và cuối cùng là phân chia tập dữ liệu để phục vụ việc huấn luyện mô hình hồi quy Softmax.

1.2.1 Thu thập và đọc dữ liệu thô (Data Loading)

Dữ liệu đầu vào là bộ dataset MNIST được lưu trữ dưới định dạng tệp nhị phân IDX. Module `MnistDataloader` chịu trách nhiệm đọc và giải mã các tệp này.

- **Kiểm tra tính toàn vẹn:** Quá trình đọc tệp sử dụng thư viện `struct` để giải mã phần header. Hệ thống thực hiện kiểm tra "magic number" (số định danh tệp) để đảm bảo tính hợp lệ của dữ liệu: giá trị 2049 đối với tệp nhãn (label) và 2051 đối với tệp hình ảnh.
- **Định dạng dữ liệu ban đầu:** Dữ liệu hình ảnh được đọc dưới dạng mảng byte liên tục, sau đó được tái cấu trúc (reshape) thành các ma trận kích thước 28×28 pixel.
- **Chuyển đổi dữ liệu:** Tại hàm `load_mnist_data`, dữ liệu thô được chuyển đổi sang định dạng `numpy.array` với kiểu dữ liệu `uint8` (số nguyên không dấu 8-bit) nhằm tối ưu hóa bộ nhớ lưu trữ trước khi bước vào giai đoạn xử lý đặc trưng.

1.2.2 Chuẩn hoá và trích xuất đặc trưng (Feature Extraction)

Giai đoạn này được thực hiện bởi lớp `FeatureExtractor`, bao gồm bước chuẩn hóa chung và ba chiến lược trích xuất đặc trưng riêng biệt nhằm đa dạng hóa đầu vào cho mô hình.

Chuẩn hoá dữ liệu (Normalization): Trước khi áp dụng bất kỳ thuật toán trích xuất nào, phương thức `_normalize` được áp dụng để chuyển đổi giá trị pixel từ thang đo số nguyên $[0, 255]$ sang thang đo số thực $[0.0, 1.0]$. Phép toán này sử dụng kiểu dữ liệu float32 để đảm bảo độ chính xác tính toán.

Các chiến lược trích xuất đặc trưng: Hệ thống triển khai ba phương pháp trích xuất đặc trưng chính:

- **Đặc trưng điểm ảnh (Pixel Features):** Phương thức `get_pixel_features` thực hiện duỗi phẳng (flatten) ma trận ảnh 28×28 thành vector đặc trưng một chiều có kích thước 784 ($N \times 784$). Đây là phương pháp giữ nguyên toàn bộ thông tin gốc của ảnh.
- **Đặc trưng biên dạng (Edge Detection Features):** Phương thức `get_edge_features` sử dụng thuật toán Canny (thông qua thư viện OpenCV) để phát hiện các cạnh của chữ số. Các ngưỡng (threshold) được thiết lập với `min_val=100` và `max_val=200`. Kết quả là các vector đặc trưng đại diện cho cấu trúc hình học của chữ số, loại bỏ các chi tiết nền không cần thiết.
- **Đặc trưng trung bình khối (Block Averaging Features):** Phương thức `get_block_features` thực hiện giảm chiều dữ liệu bằng kỹ thuật gộp (pooling). Ảnh được chia thành các khối kích thước 4×4 không chồng lấn, sau đó tính giá trị trung bình cho từng khối. Kỹ thuật này giảm kích thước vector đặc trưng từ 784 xuống còn 49 ($N \times 49$), giúp giảm tải tính toán.

1.2.3 Phân chia dữ liệu (Data Splitting)

Sau khi trích xuất đặc trưng, dữ liệu được phân chia thành tập huấn luyện (training set) và tập kiểm định (validation set) thông qua hàm `train_val_split`.

- **Xáo trộn ngẫu nhiên (Shuffling):** Để đảm bảo tính khách quan và tránh hiện tượng mô hình học thuộc thứ tự dữ liệu, chỉ số dữ liệu được xáo trộn ngẫu nhiên với một `seed` cố định (42) nhằm đảm bảo kết quả có thể tái lập.
- **Tỷ lệ phân chia:** Dựa trên cấu hình hệ thống (`CONFIG`), 10% dữ liệu từ tập huấn luyện gốc

được tách ra để làm tập kiểm định (`val_split`: 0.1), phục vụ việc đánh giá chéo trong quá trình huấn luyện.

2 Nền tảng toán học và triển khai mô hình

2.1 Nền tảng toán học

2.1.1 Mô hình Softmax Regression

Mô hình *Softmax Regression* được sử dụng để giải quyết bài toán phân loại đa lớp (*multi-class classification*) trên tập dữ liệu MNIST. Với một mẫu đầu vào được biểu diễn bởi vector đặc trưng $x \in \mathbb{R}^d$ (trong đó d là số chiều của không gian đặc trưng), mô hình trước tiên tính toán điểm số tuyến tính (logits) tương ứng với K lớp thông qua phép biến đổi sau:

$$z = Wx + b.$$

Trong đó:

- $W \in \mathbb{R}^{K \times d}$ là ma trận trọng số;
- $b \in \mathbb{R}^K$ là vector bias;
- $z \in \mathbb{R}^K$ là vector điểm số.

Để chuyển các điểm số tuyến tính z thành một phân phối xác suất hợp lệ trên không gian nhãn, mô hình sử dụng hàm kích hoạt *softmax*. Xác suất để mẫu x thuộc lớp k được xác định bởi:

$$\hat{y}_k = P(y = k|x) = softmax(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

Trong đó, z_k là điểm số ứng với lớp k, và mẫu số $\sum_{j=1}^{K} e^{z_j}$ đảm bảo rằng tổng các xác suất bằng 1, tạo thành một phân phối xác suất chuẩn hóa trên K lớp.

2.1.2 Hàm mất mát (Loss Function)

Để huấn luyện mô hình, ta sử dụng hàm mất mát *Cross-Entropy* nhằm đo lường mức độ khác biệt giữa phân phối xác suất dự đoán và nhãn thực tế. Đồng thời, nhằm hạn chế hiện tượng quá khớp (*overfitting*), một thành phần điều chuẩn L2 (*L2 Regularization*) được bổ sung vào hàm mất mát tổng quát. Cụ thể, hàm mất mát được định nghĩa như sau:

$$J(W, b) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{i,k} \log \hat{y}_{ik} + \frac{\lambda}{2} \sum_{l} \sum_{m} W_{lm}^2.$$

Trong đó:

- N là số lượng mẫu trong một batch;
- $y_{i,k}$ là thành phần của vector one-hot label (bằng 1 nếu mẫu i thuộc lớp k, ngược lại bằng 0);
- λ (ký hiệu là `reg` trong mã nguồn) là hệ số điều chuẩn.

Trong biểu thức trên, hạng tử cross-entropy (hạng tử thứ nhất) thúc đẩy mô hình tối đa hóa xác suất dự đoán đúng, trong khi hạng tử điều chuẩn (hạng tử thứ hai) giúp giới hạn độ lớn của các trọng số, từ đó nâng cao khả năng tổng quát hóa của mô hình.

2.1.3 Giải thuật tối ưu Gradient Descent

Quá trình huấn luyện mô hình được thực hiện bằng thuật toán *Mini-batch Gradient Descent*. Ở mỗi bước lặp, ta tính gradient của hàm mất mát đối với các tham số W và b. Nhờ đặc tính giải tích thuận lợi của hàm softmax khi kết hợp với hàm mất mát cross-entropy, các biểu thức đạo hàm thu được có dạng đơn giản và hiệu quả như sau:

$$\frac{\partial J}{\partial z} = \hat{y} - y,\ \frac{\partial J}{\partial W} = \frac{1}{N} X^T(\hat{y} - y) + \lambda W,\ \frac{\partial J}{\partial b} = \frac{1}{N} \sum (\hat{y} - y).$$

Trong đó $X \in \mathbb{R}^{N \times d}$ là ma trận dữ liệu đầu vào của mini-batch gồm N mẫu.
Quy tắc cập nhật tham số với tốc độ học α (*learning rate*) được thể hiện như sau:

$$W \leftarrow W - \alpha \frac{\partial J}{\partial W},\ b \leftarrow b - \alpha \frac{\partial J}{\partial b}.$$

Nhờ việc sử dụng mini-batch, quá trình tối ưu vừa duy trì sự ổn định của gradient, vừa đảm bảo hiệu năng tính toán phù hợp cho các tập dữ liệu lớn.
Trước khi đi vào quy trình huấn luyện lặp, ta cần xác định cách khởi tạo tham số và chi tiết các bước tính toán giá trị mất mát cũng như đạo hàm tại mỗi bước lặp. Các hàm này tương ứng với phương thức `__init__` và `compute_loss_and_grads` trong mã nguồn cài đặt.

Algorithm 1 Khởi tạo và Tính toán Loss/Gradient

function INITIALIZE($n\_features, n\_classes, lr, reg$)
  $self.lr \leftarrow lr$
  $self.reg \leftarrow reg$
  ▷ Khởi tạo ngẫu nhiên nhỏ để phá vỡ tính đối xứng
  $W \leftarrow 0.01 \times \mathcal{N}(0, 1, (n\_classes, n\_features))$
  $b \leftarrow \vec{0}_{n\_classes}$
end function

function COMPUTELOSSANDGRADS(X, y)
  $N \leftarrow \text{rows}(X)$
  1. Forward Pass:
  $scores \leftarrow XW^\top + b$
  $probs \leftarrow \text{Softmax}(scores)$
  ▷ Sử dụng max-subtraction để ổn định số học
  2. Compute Loss (Data + Regularization):
  $correct\_logprobs \leftarrow -\log(probs[\text{range}(N), y] + \epsilon)$
  $data\_loss \leftarrow \text{Mean}(correct\_logprobs)$
  $reg\_loss \leftarrow 0.5 \cdot reg \cdot \sum(W \circ W)$
  $loss \leftarrow data\_loss + reg\_loss$
  3. Backward Pass (Gradient):
  $dscores \leftarrow probs$
  $dscores[\text{range}(N), y] \leftarrow dscores[\text{range}(N), y] - 1$
  $dscores \leftarrow dscores/N$
  $\nabla_W \leftarrow dscores^\top X$
  if $reg > 0$ **then**
    $\nabla_W \leftarrow \nabla_W + reg \cdot W$
  end if
  $\nabla_b \leftarrow \text{Sum}(dscores, \text{axis} = 0)$
  return $(loss, \nabla_W, \nabla_b)$
end function

Đoạn mã giả trên mô tả chi tiết hai quá trình:

- **Initialize**: Thiết lập ma trận trọng số W với các giá trị ngẫu nhiên nhỏ theo phân phối chuẩn và bias b bằng 0. Việc này giúp mô hình hội tụ tốt hơn so với khởi tạo bằng 0.
- **ComputeLossAndGrads**: Thực hiện tính toán xuôi (forward) để lấy xác suất dự đoán, sau đó tính toán ngược (backward) để xác định gradient. Lưu ý rằng gradient của trọng số ∇_W bao gồm cả thành phần đạo hàm từ *Regularization* ($reg \cdot W$).

2.1.4 Quy trình huấn luyện mô hình

- Xáo trộn (*shuffle*) tập huấn luyện ở đầu mỗi epoch bằng một hoán vị ngẫu nhiên các chỉ số.

Algorithm 2 Hàm huấn luyện `fit`

function FIT($X_{train}, y_{train}, X_{val}, y_{val}, epochs, batch\_size$)
  $N \leftarrow \text{rows}(X_{train})$
  $history.train\_loss \leftarrow [\,]$
  $history.val\_acc \leftarrow [\,]$
  for $epoch \leftarrow 1$ to $epochs$ **do**
    $indices \leftarrow \text{RandomPermutation}(\{0, \ldots, N-1\})$
    $X_{train} \leftarrow X_{train}[indices]$
    $y_{train} \leftarrow y_{train}[indices]$
    $epoch\_loss \leftarrow 0$
    $n\_batches \leftarrow 0$
    for $start \leftarrow 0$ to N step $batch\_size$ **do**
      $end \leftarrow start + batch\_size$
      $X_{batch} \leftarrow X_{train}[start : end]$
      $y_{batch} \leftarrow y_{train}[start : end]$
      $(L, \nabla_W, \nabla_b) \leftarrow \text{ComputeLossAndGrads}(X_{batch}, y_{batch})$
      $epoch\_loss \leftarrow epoch\_loss + L$
      $n\_batches \leftarrow n\_batches + 1$
      $W \leftarrow W - lr \cdot \nabla_W$
      $b \leftarrow b - lr \cdot \nabla_b$
    end for
    $avg\_loss \leftarrow epoch\_loss / \max(n\_batches, 1)$
    $history.train\_loss.\text{append}(avg\_loss)$
    if $X_{val} \neq \emptyset$ and $y_{val} \neq \emptyset$ **then**
      $\hat{y}_{val} \leftarrow \text{Predict}(X_{val})$
      $val\_acc \leftarrow \text{Mean}(\hat{y}_{val} = y_{val})$
      $history.val\_acc.\text{append}(val\_acc)$
    end if
  end for
  return *history*
end function

- Chia dữ liệu đã xáo trộn thành các mini-batch có kích thước `batch_size`.
- Với mỗi mini-batch, gọi `ComputeLossAndGrads` để tính giá trị hàm mất mát và gradient theo W, b.
- Cập nhật tham số theo quy tắc gradient descent: $W \leftarrow W - \text{lr} \cdot \nabla_W$, $b \leftarrow b - \text{lr} \cdot \nabla_b$.
- Tính loss trung bình của toàn bộ các mini-batch trong một epoch và lưu vào `history.train_loss`.
- Nếu có tập validation, tính độ chính xác trên X_{val} và lưu vào `history.val_acc`.

2.1.5 Quy trình suy diễn và dự đoán

Algorithm 3 Hàm suy diễn `predict_proba` và `predict`

function PREDICT_PROBA(X)
  $Z \leftarrow XW^\top + b$
  $Z' \leftarrow Z - \max(Z \text{ theo từng hàng})$
  $expZ \leftarrow \exp(Z')$
  $P \leftarrow expZ / \sum(expZ \text{ theo từng hàng})$
  return P
end function

function PREDICT(X)
  $P \leftarrow \text{predict\_proba}(X)$
  $\hat{y} \leftarrow \arg\max(P \text{ theo từng hàng})$
  return $\hat{y}$
end function

- `predict_proba`:
  - Tính điểm số tuyến tính $Z = XW^\top + b$ cho tất cả mẫu và tất cả lớp.
  - Chuẩn hoá Z theo từng hàng để ổn định số học, sau đó áp dụng hàm softmax để thu được ma trận xác suất P.
- `predict`:
  - Gọi lại `predict_proba` để lấy phân phối xác suất trên các lớp cho từng mẫu.
  - Chọn lớp có xác suất lớn nhất bằng toán tử $\arg\max$ theo chiều lớp, thu được vector nhãn dự đoán $\hat{y}$.

3 Thiết kế các feature vector

3.1 Feature vector 1: Đặc trưng cường độ điểm ảnh (Raw Pixel Intensity)

Theo [**lecun1998gradient**], phương pháp biểu diễn dữ liệu trực quan nhất là sử dụng trực tiếp giá trị cường độ sáng (pixel intensity) của ảnh làm vector đặc trưng. Phương pháp này, được triển khai trong hàm `get_pixel_features`, coi mỗi điểm ảnh là một biến độc lập, bỏ qua mối quan hệ không gian cục bộ giữa các điểm ảnh lân cận.

3.1.1 Biểu diễn vector hoá (Vectorization)

Dữ liệu gốc của bộ MNIST bao gồm các ảnh xám có kích thước 28×28 pixel. Tuy nhiên, các mô hình học máy truyền thống như Softmax Regression yêu cầu đầu vào là các vector một chiều cố định. Do đó, kỹ thuật duỗi phẳng (flattening) được áp dụng để chuyển đổi ma trận ảnh $I \in \mathbb{R}^{H \times W}$ thành vector đặc trưng $x \in \mathbb{R}^D$, với $D = H \times W$.
Trong ngữ cảnh đồ án này:

$$x = \text{flatten}(I) \in \mathbb{R}^{784}.$$

Quá trình này biến đổi cấu trúc dữ liệu từ $(N, 28, 28)$ thành $(N, 784)$. Mặc dù đơn giản, cách tiếp cận này vẫn giữ lại toàn bộ thông tin gốc của ảnh và đã được Yann LeCun cùng cộng sự sử dụng làm mốc cơ sở (baseline) trong các nghiên cứu đầu tiên về MNIST.

3.1.2 Chuẩn hoá dữ liệu (Data Normalization)

Trước khi đưa vào mô hình, dữ liệu điểm ảnh thô (thường nằm trong khoảng $[0, 255]$ đối với ảnh 8-bit) cần được chuẩn hóa. Trong phương thức `_normalize`, hệ thống thực hiện phép biến đổi tuyến tính để đưa giá trị về khoảng $[0, 1]$:

$$x_{norm} = \frac{x_{raw}}{255.0}.$$

Thao tác này được thực hiện bằng cách ép kiểu dữ liệu sang `float32` và chia cho 255.0. Việc chuẩn hoá này đóng vai trò quan trọng trong việc tối ưu hoá mô hình Gradient Descent:

- Đồng nhất thang đo: Giúp các đặc trưng đóng góp bình đẳng vào hàm mất mát.

- Ổn định số học: Ngăn chặn hiện tượng bão hòa hoặc tràn số khi tính toán hàm mũ trong lớp Softmax.
- Tăng tốc độ hội tụ: Giúp thuật toán tối ưu tìm được cực trị nhanh hơn so với dữ liệu thô chưa chuẩn hóa.

3.2 Feature vector 2: Trích xuất đặc trưng biên (Edge feature)

Trong khuôn khổ của hệ thống nhận dạng chữ viết tay, bên cạnh việc sử dụng trực tiếp giá trị cường độ điểm ảnh (pixel intensity), phương thức trích xuất đặc trưng thứ hai được đề xuất là khai thác thông tin cấu trúc hình học thông qua kỹ thuật phát hiện biên (Edge Detection). Kỹ thuật này được hiện thực hóa trong phương thức `get_edge_features` của lớp `FeatureExtractor`.

3.2.1 Nguyên lý toán học của thuật toán Canny [canny1986computational]

Phương pháp này sử dụng thuật toán Canny (Canny Edge Detector), một trong những kỹ thuật phát hiện biên kinh điển và hiệu quả nhất trong xử lý ảnh. Về mặt toán học, biên của một đối tượng trong ảnh số được định nghĩa là nơi có sự thay đổi đột ngột về cường độ sáng.
Thuật toán Canny hoạt động dựa trên việc tính toán đạo hàm của cường độ sáng (gradient) tại mỗi điểm ảnh. Với một đầu vào $I(x, y)$, gradient tại toạ độ (x, y) được xác định bởi vector:

$$\nabla I = \begin{bmatrix} G_x \\ G_y \end{bmatrix}.$$

Trong đó G_x và G_y là đạo hàm riêng theo hướng ngang và dọc. Biên độ (magnitude) của gradient, ký hiệu là G, đại diện cho cường độ của biên tại điểm đó:

$$G = \sqrt{G_x^2 + G_y^2}.$$

3.2.2 Kỹ thuật phân ngưỡng trễ (Hysteresis Thresholding)

Điểm đặc biệt trong triển khai của phương thức `get_edge_features` là việc áp dụng cơ chế phân ngưỡng trễ (Hysteresis Thresholding) để lọc nhiễu và xác định các đường biên thực sự. Cơ chế này sử dụng hai giá trị ngưỡng: ngưỡng dưới (T_{min}) và ngưỡng trên (T_{max}).
Trong đồ án, hai tham số này được thiết lập mặc định như sau:

- Ngưỡng dưới (`min_val`): 100.

- Ngưỡng trên (`max_val`): 200.

Quy tắc lọc biên hoạt động như sau:

- Các điểm ảnh có gradient $G \geq T_{max}$ (200) được xem là biên chắc chắn (strong edeges).
- Các điểm ảnh có gradient $G < T_{min}$ (100) bị loại bỏ.
- Các điểm ảnh nằm trong khoảng $T_{min} \leq G < T_{max}$ được xem là biên yếu (weak edges) và chỉ được giữ lại nếu chúng có liên kết hình học với một biên chắc chắn. Điều này giúp đảm bảo tính liên tục của các nét chữ viết tay và loại bỏ các nhiễu rời rạc.

3.2.3 Quy trình tiền xử lý và biểu diễn đặc trưng

Trước khi áp dụng thuật toán Canny, dữ liệu ảnh đầu vào được chuyển đổi sang định dạng `uint8` (số nguyên 8-bit không dấu) để tương thích với thư viện OpenCV:

$$I_{uint8} = \text{cast}(I_{raw}, \text{uint8}).$$

Việc chuyển đổi này được thực hiện trong vòng lặp xử lý từng mẫu dữ liệu i thông qua lệnh `images[i].astype(np.uint8)`.
Kết quả đầu ra của bộ phát hiện biên là một ma trận nhị phân biểu thị vị trí các nét vẽ. Để đồng nhất với đầu vào của mô hình Softmax Regression, ma trận này được chuẩn hoá về khoảng $[0, 1]$ và duỗi phẳng (flatten) thành một vector đặc trưng $x \in \mathbb{R}^{784}$:

$$x_{edge} = \text{flatten}\left(\frac{\text{Canny}(I_{uint8})}{255.0}\right).$$

Quy trình này đảm bảo mỗi ảnh được biểu diễn bởi một vector đặc trưng có kích thước $(N, 784)$, trong đó các giá trị khác 0 đại diện cho cấu trúc hình học của chữ số.

3.2.4 Ý nghĩa

Việc sử dụng đặc trưng biên giúp mô hình tập trung vào hình dáng và cấu trúc tô pô của chữ số thay vì phụ thuộc vào độ nhạt của nét bút (vốn có thể thay đổi tuỳ thuộc vào người viết hoặc dụng cụ viết). Đây là một bước trích xuất đặc trưng bậc cao giúp tăng cường khả năng bất biến của mô hình đối với các biến thể về độ sáng của ảnh đầu vào.

3.3 Feature vector 3: Giảm dimension bằng block averaging

Ý tưởng chính: Ở hai thiết kế trước đó, mô hình Softmax Regression làm việc trực tiếp với tập dữ liệu, dẫn đến việc vector đặc trưng có số chiều khá lớn, khiến quá trình huấn luyện chậm hơn và mô hình dễ bị ảnh hưởng bởi nhiễu.
Vì vậy, feature vector thứ ba (**Block Averaging**) được giới thiệu nhằm giải quyết vấn đề này bằng cách thay thế 1 block với dimension cho trước (vd: 4×4) thành 1 giá trị duy nhất là trung bình của các pixel trong block này.

Cơ chế chuyển đổi ảnh 28×28 thành vector đặc trưng.

1. Chia ảnh 28×28 thành các block con không chồng lắp có kích thước cố định, ví dụ 4×4.
2. Mỗi block được rút gọn thành một giá trị duy nhất bằng cách tính trung bình cường độ sáng của các pixel trong block đó.
3. Các giá trị trung bình của các block con được xếp lại theo thứ tự của block 4×4 cha, tạo thành một vector đặc trưng có số chiều nhỏ hơn rất nhiều so với vector pixel gốc.

Cụ thể, với ảnh MNIST kích thước 28×28 và kích thước block $(b_h, b_w) = (4, 4)$:

- Chiều cao ảnh: $H = 28$, chiều rộng ảnh: $W = 28$;
- Số block theo chiều dọc: $H' = \frac{H}{b_h} = \frac{28}{4} = 7$;
- Số block theo chiều ngang: $W' = \frac{W}{b_w} = \frac{28}{4} = 7$;
- Tổng số block: $H' \cdot W' = 7 \cdot 7 = 49$.

Sau bước Block Averaging, mỗi ảnh được biểu diễn bằng vector 7×7 (thay vì 28×28).

Biểu diễn toán học. Ký hiệu ảnh đã chuẩn hoá (chia cho 255) là ma trận $X \in [0, 1]^{28 \times 28}$. Với kích thước block (b_h, b_w), ta định nghĩa ma trận đặc trưng sau khi khử chiều là $F \in \mathbb{R}^{H' \times W'}$, với:

$$H' = \frac{H}{b_h}, \quad W' = \frac{W}{b_w}.$$

Phần tử $F_{r,c}$ (block ở hàng r, cột c) được tính bằng trung bình cường độ sáng trên block tương ứng của ảnh gốc:

$$F_{r,c} = \frac{1}{b_h \cdot b_w} \sum_{i=0}^{b_h-1} \sum_{j=0}^{b_w-1} X_{r \cdot b_h + i,\, c \cdot b_w + j}, \quad 0 \le r < H',\ 0 \le c < W'. \tag{1}$$

Cuối cùng, ta vector hoá ma trận F theo thứ tự hàng–cột:

$$\phi_{\text{block}}(X) = \text{vec}(F) \in \mathbb{R}^{H' \cdot W'}.$$

Trong thực nghiệm với $(b_h, b_w) = (4, 4)$, ta có:

$$\phi_{\text{block}}(X) \in \mathbb{R}^{49}.$$

Quy trình trích xuất đặc trưng trong code:

1. **Chuẩn hoá**: chuyển ảnh từ kiểu `uint8` về `float32` và chia cho 255 để đưa cường độ pixel về khoảng $[0, 1]$:
$$X_{\text{norm}} = \frac{X}{255}.$$

2. **Reshape để gom block**: với N ảnh đầu vào có dạng $(N, 28, 28)$, ta reshape thành dạng:
$$(N, H', b_h, W', b_w) = (N, 7, 4, 7, 4),$$
sao cho mỗi "ô" (r, c) tương ứng đúng một block 4×4 của ảnh ban đầu.

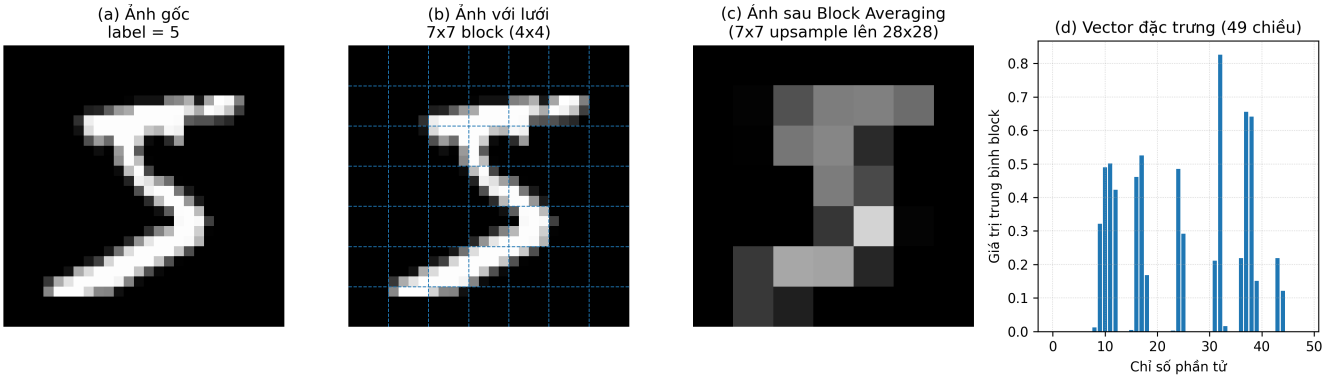
3. **Trung bình theo block**: tính trung bình trên hai trục ứng với kích thước block (b_h, b_w) để thu được tensor dạng:
$$(N, H', W') = (N, 7, 7).$$

4. **vector hoá**: flatten từng ma trận 7×7 về vector chiều 49 để đưa vào mô hình Softmax Regression.

Trong mã nguồn, các bước này tương ứng với hàm:

```
FeatureExtractor.get_block_features(images, block_size=(4,4))
```

trả về mảng NumPy kích thước $(N, 49)$.



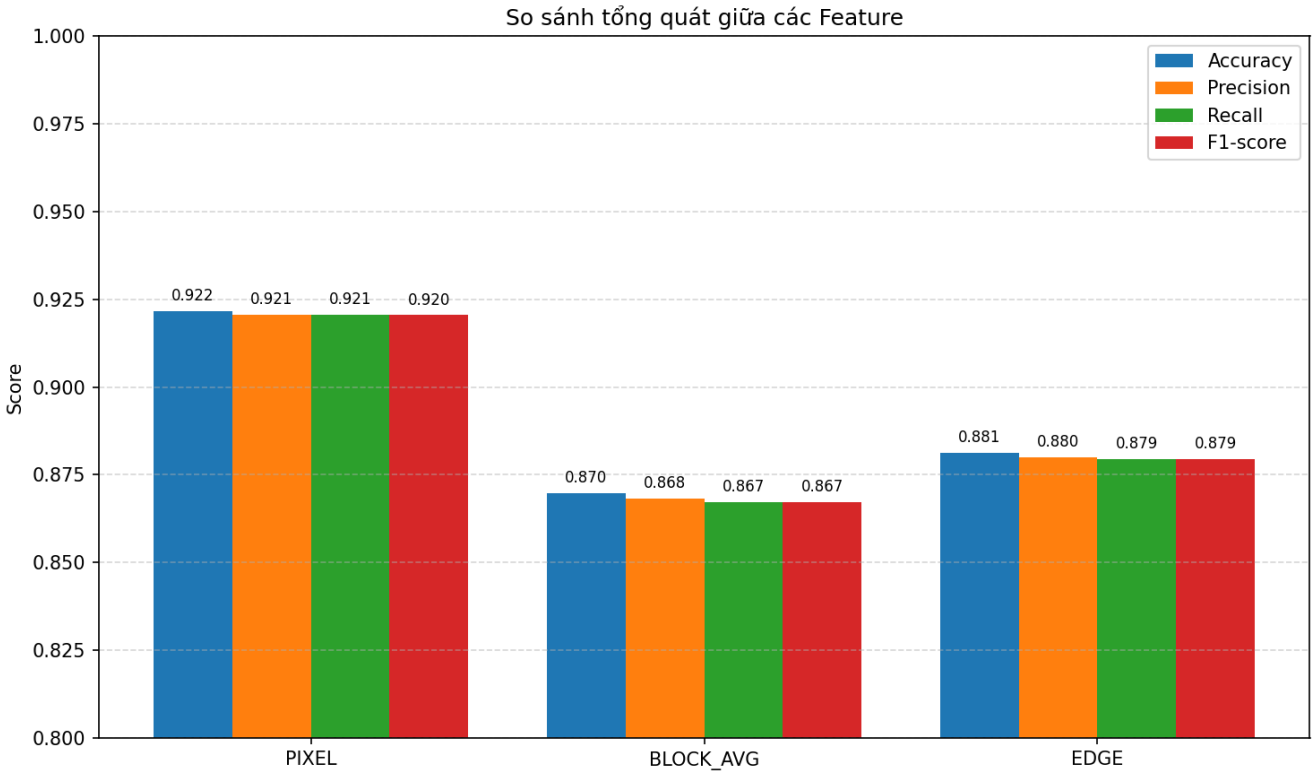
Hình 1: Quy trình trích xuất đặc trưng Block Averaging: từ ảnh chữ số gốc 28×28 sang ảnh đã được khử chiều 7×7 và vector đặc trưng chiều 49.

Đánh giá sơ bộ về ưu nhược điểm trước khi train

- **Ưu điểm:**
  - Giảm số chiều đặc trưng rất mạnh (từ 784 xuống 49), qua đó:
    - Mô hình Softmax Regression có ít tham số hơn, huấn luyện nhanh hơn
    - Giảm nguy cơ overfitting trên những nhiễu rất cục bộ.
  - Giữ lại được cấu trúc tổng thể của chữ số: các vùng pixel "sáng" (nét chữ) khi được trung bình hoá vẫn tạo thành hình dạng gần giống chữ số ban đầu ở cấp độ thô.
  - Trung bình hoá trong block hoạt động như một bộ lọc thông thấp (low-pass filter), làm trơn các biến thiên nhỏ trong block và giúp mô hình ít nhạy với nhiễu.
- **Nhược điểm:**
  - Mất đi nhiều chi tiết quan trọng, đặc biệt là các nét mảnh hoặc cấu trúc tinh tế giúp phân biệt những cặp chữ số khó như $(3, 5)$, $(5, 8)$, $(4, 9)$, ...
  - Do block không chồng lắp, các đường biên rơi đúng ranh giới block có thể bị "bóp méo" nhiều hơn, làm giảm khả năng phân biệt dựa trên hình dạng chính xác.
  - Khi chỉ nhìn ở độ phân giải 7×7, một số chữ số khác nhau có thể trở nên khá giống nhau về mặt phân bố cường độ trung bình, khiến ranh giới quyết định tuyến tính của Softmax Regression khó phân tách hoàn toàn.

4 Đánh giá và phân tích kết quả

4.1 Kết quả dự đoán của mô hình với từng loại feature



Hình 2: So sánh tổng quan Accuracy, Precision, Recall và F1-score giữa các feature

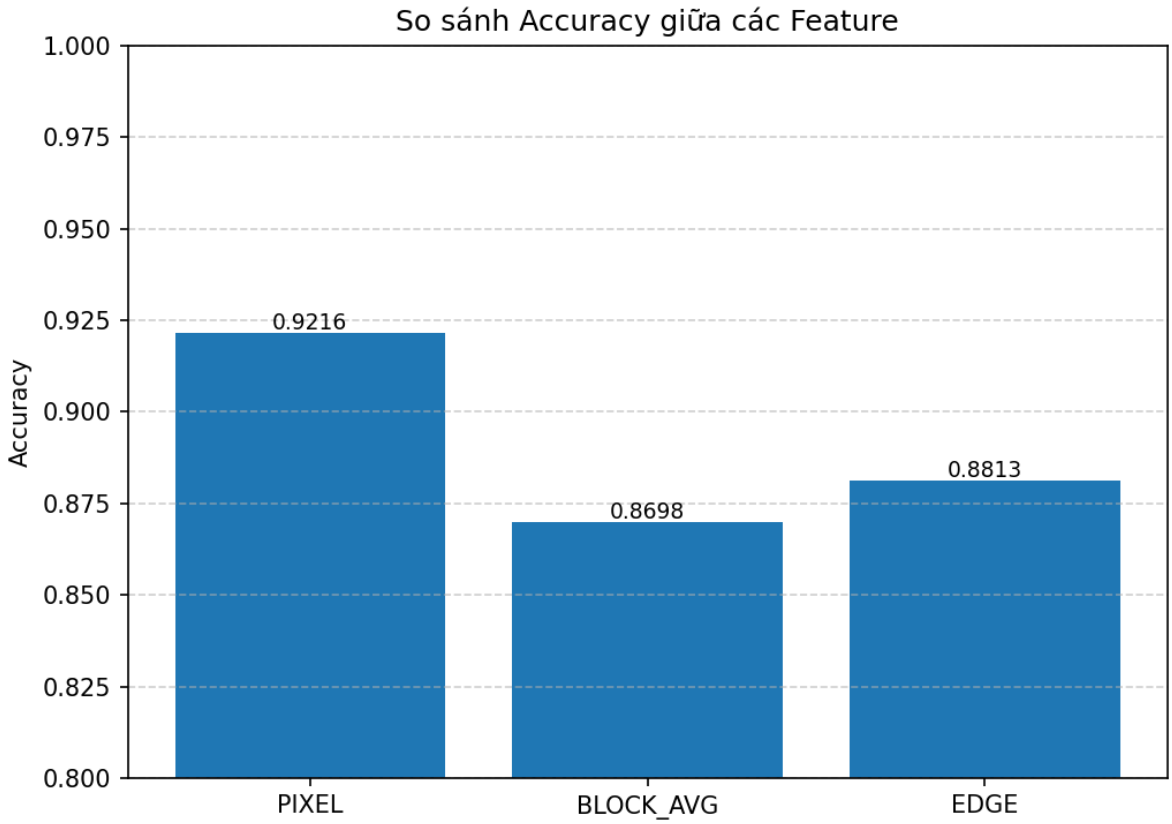
Đầu tiên, ta có thể thấy rằng cả ba dạng feature (PIXEL, BLOCK_AVG, EDGE) đều cho kết quả dự đoán khá tốt, với độ chính xác dao động trong khoảng 0.87 - 0.92. Tuy nhiên, sự khác biệt giữa chúng là rõ ràng:

- **PIXEL** đạt Accuracy cao nhất ≈ 0.922, chứng tỏ mô hình học tốt nhất khi toàn bộ thông tin pixel được giữ nguyên.
- **EDGE** xếp thứ hai với Accuracy ≈ 0.881, thể hiện rằng thông tin về đường biên vẫn mang giá trị phân biệt mạnh.
- **BLOCK_AVG** có Accuracy thấp nhất ≈ 0.870, cho thấy việc giảm chiều mạnh làm mất nhiều chi tiết quan trọng.

=> Mô hình có feature đầy đủ (PIXEL) đạt performance tốt nhất, trong khi hai mô hình còn lại có xu hướng giảm độ chính xác khi thông tin đặc trưng bị giản lược.

4.2 So sánh giữa các feature với nhau

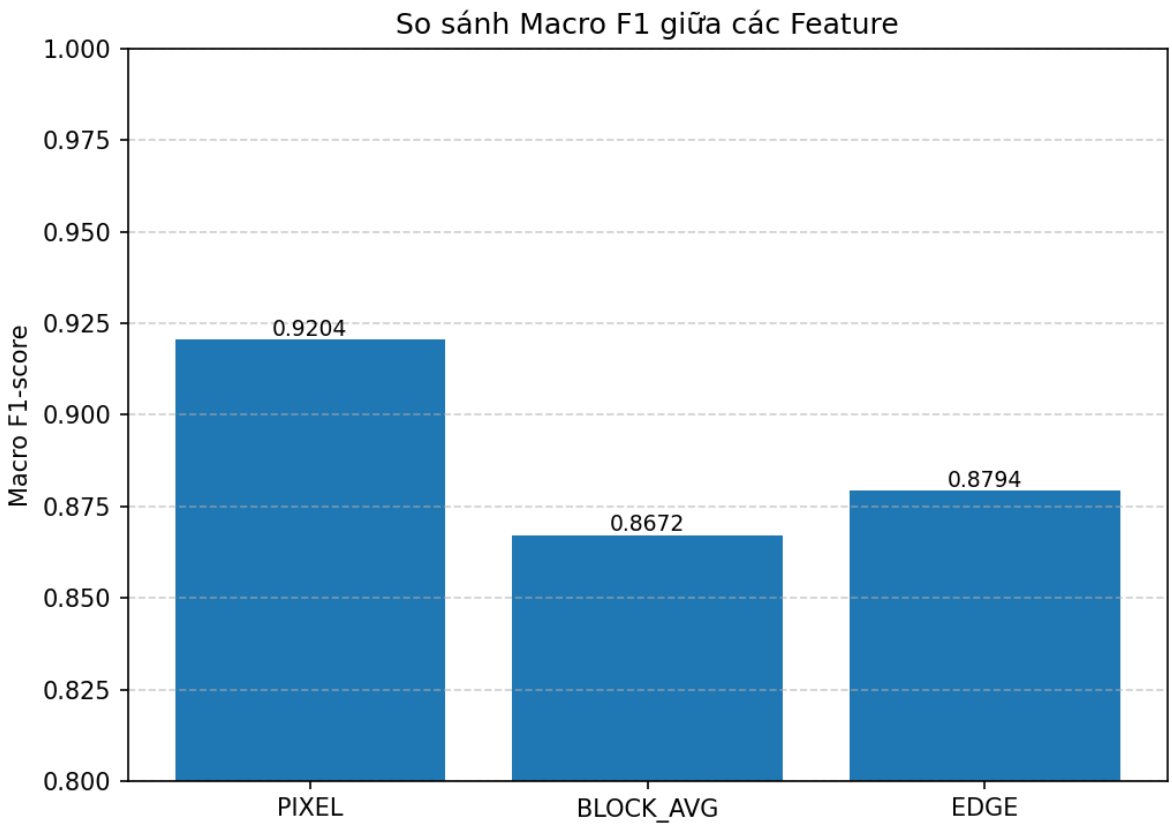
So sánh Accuracy



Hình 3: So sánh Accuracy giữa các feature

Feature PIXEL đạt Accuracy cao nhất (0.922), tiếp theo là EDGE (0.881), và cuối cùng là BLOCK_AVG (0.8698). => Cho thấy mức độ chi tiết của feature ảnh hưởng trực tiếp đến khả năng phân loại của Softmax Regression.

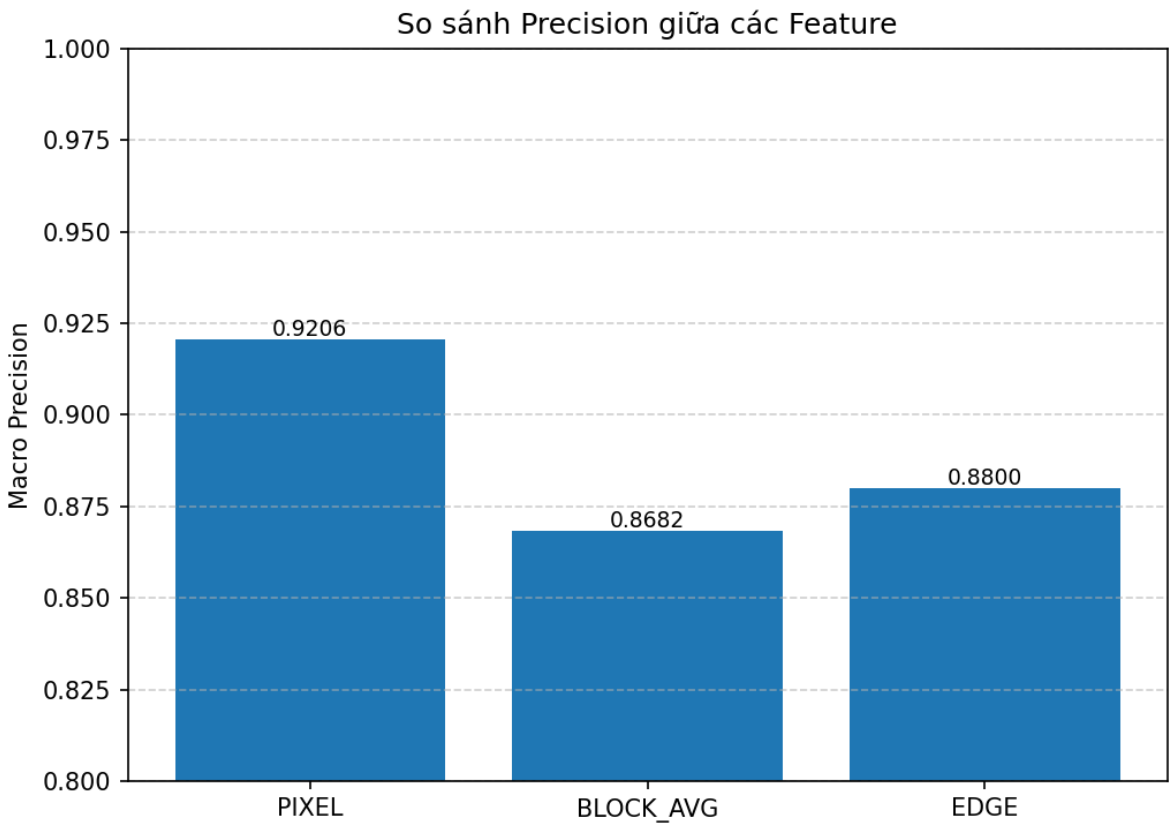
So sánh Macro F1-score



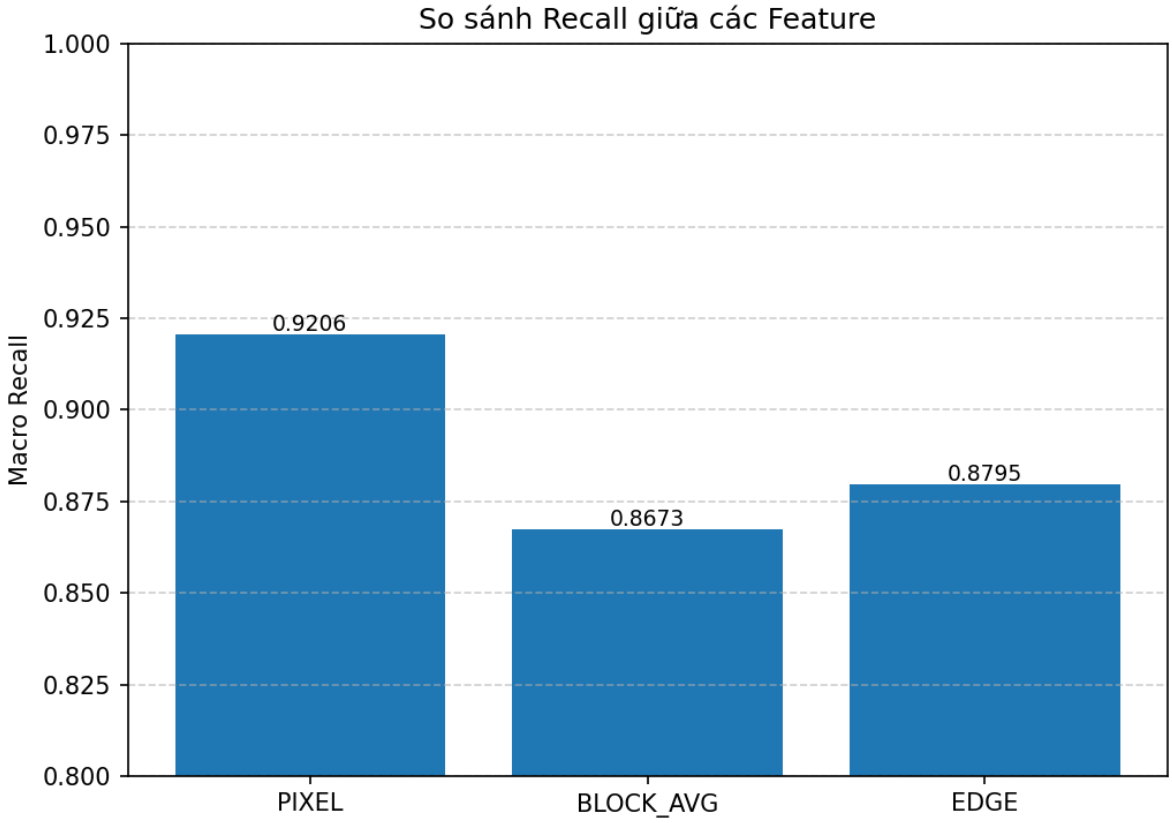
Hình 4: So sánh Macro F1-score giữa các feature

PIXEL tiếp tục đứng đầu với Macro F1 ≈ 0.9204. EDGE đạt ≈ 0.8794 và BLOCK_AVG đạt khoảng 0.8672. => Thứ hạng các feature tương tự Accuracy, chứng tỏ hiệu năng ổn định trên toàn bộ lớp thay vì chỉ tập trung một vài lớp.

So sánh Precision và Recall



Hình 5: So sánh Macro Precision giữa các feature



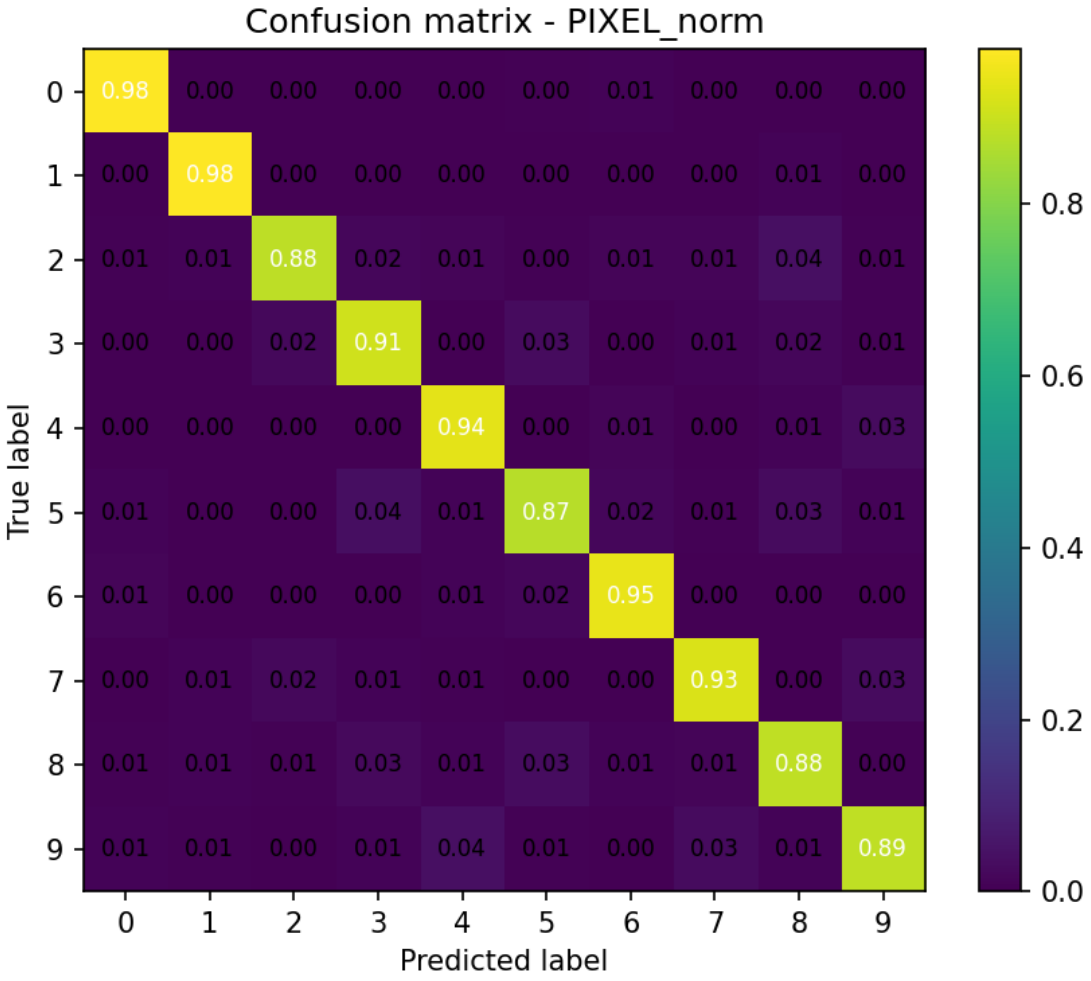
Hình 6: So sánh Macro Recall giữa các feature

Precision và Recall của PIXEL đều trên 0.92, cao nhất trong ba feature. EDGE duy trì mức trung bình ≈ 0.88 và BLOCK_AVG thấp nhất khoảng 0.867. => Cả Precision và Recall đều cho thấy mô hình với feature BLOCK_AVG có xu hướng bỏ sót và nhầm lẫn giữa các lớp nhiều hơn.

4.3 Phân tích chi tiết qua Confusion Matrix

Trong phần này, ta phân tích mức độ nhầm lẫn giữa các lớp dựa trên confusion matrix của từng feature.

PIXEL

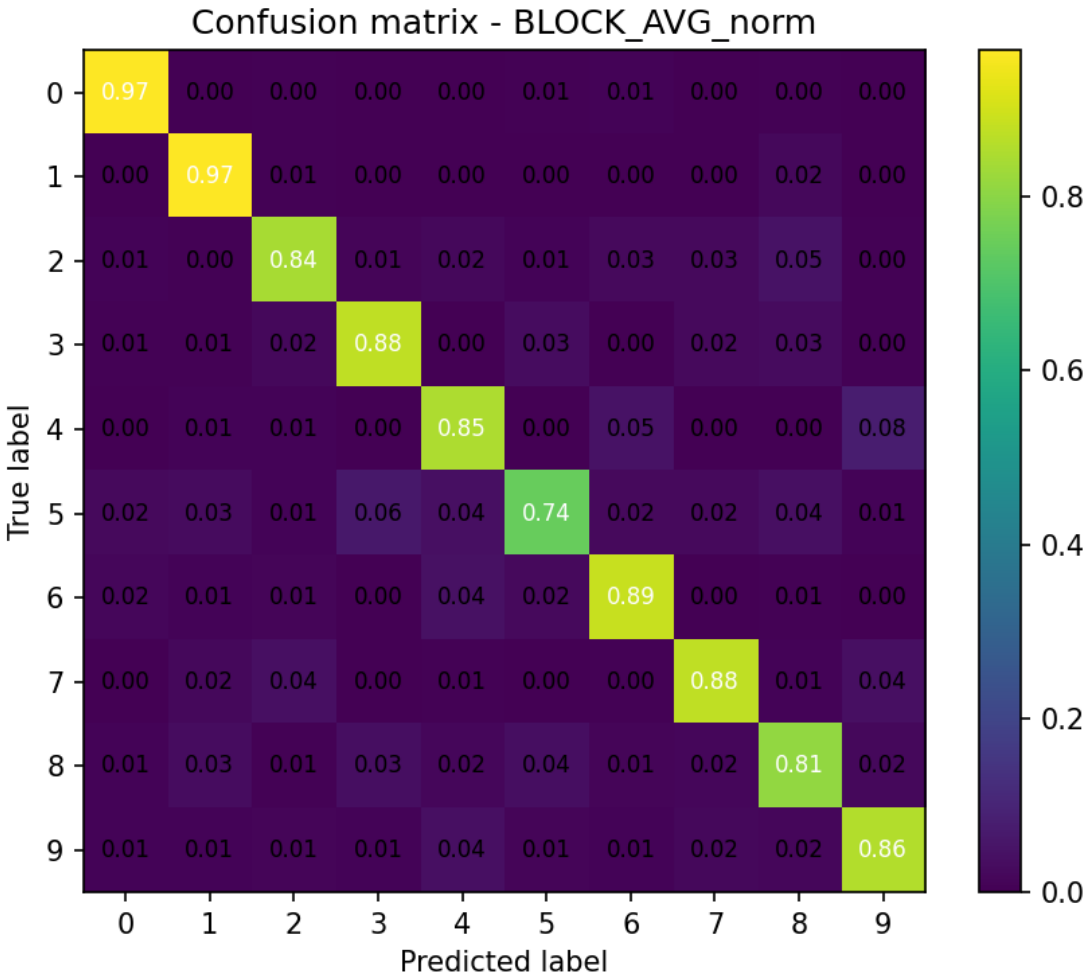


Hình 7: Confusion matrix chuẩn hoá - Feature PIXEL

Mô hình với PIXEL cho thấy đường chéo chính rất đậm:

- Các lớp 0, 1, 6 đạt tỉ lệ dự đoán đúng trên 95%.
- Model có 1 sự nhầm lẫn nhẹ giữa 3 và 5, phản ánh sự tương đồng hình dạng giữa 2 số này.

BLOCK_AVG



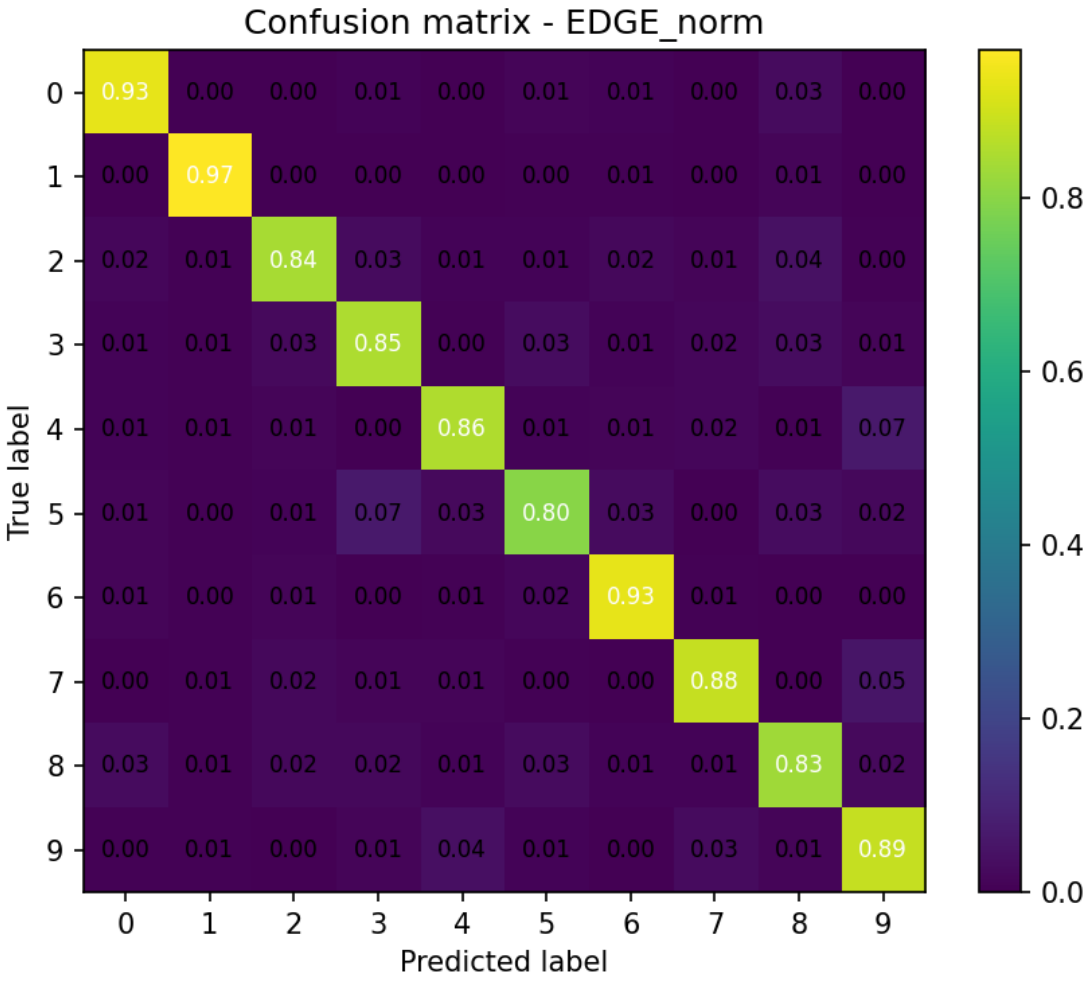
Hình 8: Confusion matrix chuẩn hoá - Feature BLOCK_AVG

BLOCK_AVG có mức nhầm lẫn cao hơn đáng kể:

- Lớp 5 bị nhầm sang lớp 3 khoảng 6%.
- Lớp 8 bị nhầm với 1, 3, và 5 (2-4%).

=> Khi giảm ảnh xuống 7×7, mô hình mắc nhiều sai sót hơn khi phân tách các lớp có cấu trúc tương tự.

EDGE



Hình 9: Confusion matrix chuẩn hoá - Feature EDGE

- Các lớp có hình dạng rõ ràng như 0, 1, 6 được dự đoán chính xác cao.
- Tuy nhiên, tương tự như BLOCK_AVG, EDGE cũng bị nhầm lẫn khi xác định các số còn lại, đặc biệt là 5 và 8.

=> EDGE tốt hơn BLOCK_AVG nhưng chưa vượt được PIXEL vì thiếu thông tin cường độ pixel. Từ toàn bộ số liệu và biểu đồ trên, ta có thể rút ra:

- **PIXEL** là feature mạnh nhất, đạt toàn bộ metric cao nhất (Accuracy, Precision, Recall, F1-score).
- **EDGE** giữ lại cấu trúc biên đủ tốt nên đứng thứ hai.

- **BLOCK_AVG** làm mất nhiều chi tiết quan trọng, dẫn đến performance thấp nhất.

=> Mặc dù BLOCK_AVG và EDGE có thể giúp giảm số chiều và thời gian xử lý, chúng làm giảm đáng kể khả năng phân loại của Softmax Regression.

5 Mô tả ứng dụng

6 Kết luận và nhận định

7 Tài liệu tham khảo