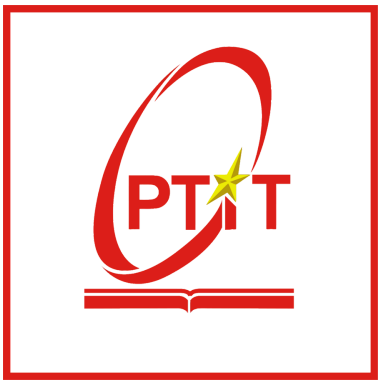
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1
****



BÀI TẬP LỚN XỬ LÝ ẢNH

Đề tài: Nhận dạng chữ viết và hình dạng đơn giản bằng mạng neural

Giảng viên hướng dẫn: Phạm Hoàng Việt

Lớp chuyên ngành: D22CNPM02 - Nhóm BTL: 7

Thành viên	Mã sinh viên
Trần Mai Hương	B22DCCN424
Nguyễn Thị Khánh Vân	B22DCCN892

Hà Nội, ngày 01/12/2025

MỤC LỤC

PHẦN I. MỞ ĐẦU & CƠ SỞ LÝ THUYẾT

1. Giới thiệu tổng quan

1.1. Đặt vấn đề

- Trong thời đại số hóa ngày nay, việc tự động hóa quá trình nhận dạng thông tin từ ảnh đóng vai trò quan trọng trong nhiều ứng dụng thực tế.
- Đối với ***nhận dạng chữ viết hay ORC (Optical Character Recognition)***, công nghệ này hỗ trợ số hóa tài liệu lịch sử, xử lý biểu mẫu y tế và giáo dục, giúp giảm thời gian thủ công và tăng độ chính xác. Ví dụ, trong thư viện quốc gia, hàng triệu trang tài liệu viết tay có thể được chuyển đổi thành văn bản có thể tìm kiếm. Tuy nhiên, thách thức lớn là độ phức tạp và tính biến thiên cao của chữ viết cá nhân, kích cỡ, hình dạng, độ nghiêng, nhiễu ảnh và chuỗi ký tự liên tục mà không có khoảng cách rõ ràng.
- Tương tự, ***nhận dạng hình dạng đơn giản (Simple Shape Recognition)*** là nền tảng cho các hệ thống thị giác máy tính, như robot tự hành phát hiện vật cản (hình vuông đại diện cho hộp) hoặc ứng dụng giáo dục phân loại hình học, xử lý ảnh y tế, tự động hoá. Thách thức bao gồm biến đổi góc quay, kích thước và nhiễu nền, đòi hỏi mô hình phải trích xuất đặc trưng hình học mạnh mẽ.
- Đề tài này tập trung vào việc sử dụng mạng neural (Deep Learning) để giải quyết hai bài toán trên, tận dụng sức mạnh của Convolutional Neural Network (CNN) cho đặc trưng không gian và Recurrent Neural Network (RNN) cho chuỗi thời gian.

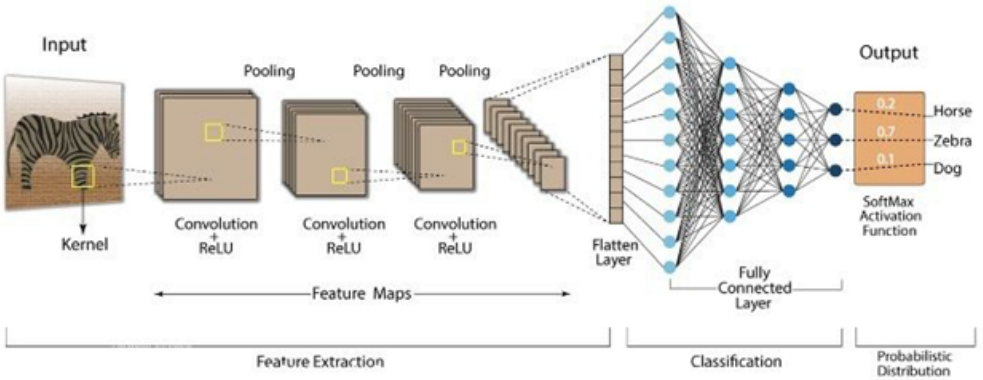
1.2. Mục tiêu đề tài

- Xây dựng ***mô hình CRNN + CTC*** cho nhận dạng chữ viết tay tiếng Anh, tối ưu hóa tốc độ và độ chính xác ở cấp độ từ.
- Xây dựng ***mô hình CNN*** cho phân loại hình dạng 5 lớp cơ bản, nhấn mạnh vào việc trích xuất đặc trưng hình học.
- Tích hợp hai mô hình vào một Tool Demo thống nhất cho mục đích trình diễn và đánh giá.
- Đánh giá chi tiết hiệu suất, đặc biệt là phân tích lỗi để đề xuất cải tiến.

2. Cơ sở lý thuyết về mạng neural (Nền tảng chung)

2.1. Mạng neural tích chập (Convolutional Neural Network - CNN)

- **Vai trò:** Trích chọn đặc trưng thị giác (Feature Extraction). CNN mô phỏng não bộ con người trong việc nhận diện mẫu hình ảnh qua các bộ lọc học được.
- **Kiến trúc tổng quan của CNN:** thường gồm 4 nhóm lớp chính
    - **Convolutional Layer (Lớp tích chập):** chịu trách nhiệm chính trong việc trích xuất đặc trưng từ ảnh đầu vào. Áp dụng các bộ lọc kernel để quét ảnh, thực hiện phép tích chập tạo feature map.
    - **Activation Layer (Lớp kích hoạt):** quyết định liệu một nơron có nên được kích hoạt hay không, tạo ra tính phi tuyến tính, cho phép mô hình học và biểu diễn các mẫu dữ liệu phức tạp.
    - **Pooling Layer (Lớp gộp):** giúp giảm kích thước không gian của bản đồ đặc trưng nhưng vẫn giữ lại thông tin quan trọng và tăng khả năng chống nhiễu (invariance).
    - **Fully Connected Layer (Lớp kết nối đầy đủ):** lớp này thực hiện vai trò phân loại hoặc hồi quy tùy theo bài toán cụ thể
- **Hình minh hoạ:**

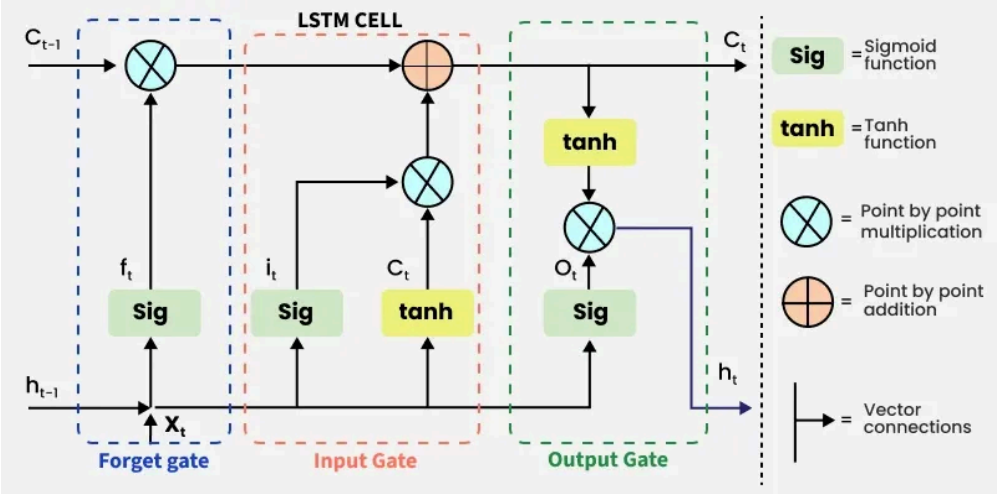


2.2. Học sâu Tuần tự (Sequence Learning) – LSTM và Bi-LSTM

❖ **LSTM (Long Short-Term Memory):** một biến thể cải tiến của RNN, được thiết kế nhằm khắc phục vấn đề "vanishing gradient" (độ dốc biến mất) khi huấn luyện mạng trên các chuỗi dài.

- Thay vì chỉ duy trì một trạng thái ẩn như RNN, LSTM bổ sung một bộ nhớ trong (cell state) và ba cổng điều khiển (gates) để lưu trữ, thêm, hoặc loại bỏ thông tin theo thời gian một cách có chọn lọc.
    - Cell state được truyền gần như không thay đổi (chỉ nhân với các gate ≈ 1)
    - Forget Gate – quên thông tin không quan trọng
    - Input Gate – thêm thông tin mới
    - Output Gate – quyết định phần nào dùng để tạo output
- Nhờ đó, LSTM có khả năng ghi nhớ dài hạn (long-term memory) mà RNN thông thường không làm được.

- **Hình minh hoạ:**



- Cách xác định thông tin cho từng cổng, được dựa trên công thức toán học chuẩn của LSTM. Giả sử tại một thời điểm ***t***, ta có:
    - x_t: Input tại thời điểm t.
    - h_{t-1}: Hidden state từ thời điểm trước.
    - C_{t-1}: Cell state từ thời điểm trước.
- Công thức chung cho các cổng sử dụng các ma trận trọng số (W) và bias (b), được học qua backpropagation.
- **Forget Gate: Xác định thông tin cần quên (thông tin cũ không quan trọng):**
    - **Mục đích:** "Quên" phần nào của cell state cũ (C_{t-1}) để tránh tích lũy nhiễu hoặc thông tin lỗi thời.
    - **Cách xác định:**
        - Tính toán một vector sigmoid từ input hiện tại (x_t) và hidden state trước (h_{t-1}).
        - Giá trị gần 0: Quên hoàn toàn (thông tin cũ không quan trọng).
        - Giá trị gần 1: Giữ nguyên (thông tin cũ vẫn quan trọng).
    - **Công thức:**

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

        - σ: Hàm sigmoid (0 đến 1).
        - [h_{t-1}, x_t]: Nối hai vector.
        - Sau đó, cập nhật cell state: C_t' = f_t ⊙ C_{t-1} (⊙ là nhân element-wise).
    - **Ví dụ minh họa:** Nếu chuỗi là "The cat sat on the mat", forget gate có thể "quên" từ "The" ở vị trí sau vì nó không quan trọng cho ngữ cảnh hiện tại.
- **Input Gate: Xác định thông tin mới cần thêm:**
    - **Mục đích:** Quyết định phần input mới nào đáng tin cậy để thêm vào cell state, tạo "candidate values" mới.
    - **Cách xác định:**

        - Input Gate (i_t): Sigmoid quyết định "mức độ thêm" (0: không thêm, 1: thêm đầy đủ).
        - Candidate values (˜C_t): Tanh tạo giá trị mới tiềm năng (từ -1 đến 1, giúp kiểm soát độ lớn).
        - Kết hợp: Chỉ thêm những phần input mới "quan trọng" dựa trên ngữ cảnh.
    - **Công thức:**

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

        - **Cập nhật cell state:**

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t.$$

    - **Ví dụ minh họa:** Trong câu trên, input gate có thể thêm "cat" như thông tin mới quan trọng, nhưng bỏ qua từ lặp lại nếu không cần.
- **Output Gate: Xác định thông tin dùng cho đầu ra (output):**
    - **Mục đích:** Lọc cell state để tạo hidden state (h_t) làm đầu ra, chỉ "xuất" phần liên quan đến dự đoán hiện tại.
    - **Cách xác định:**
        - Sigmoid quyết định "mức độ xuất" từ cell state đã cập nhật (C_t).
        - Nhân với tanh của C_t để nén giá trị về -1 đến 1.
        - Giá trị gần 0: Ẩn thông tin (không dùng cho output).
        - Giá trị gần 1: Xuất đầy đủ (thông tin quan trọng cho output).
    - **Công thức:**

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

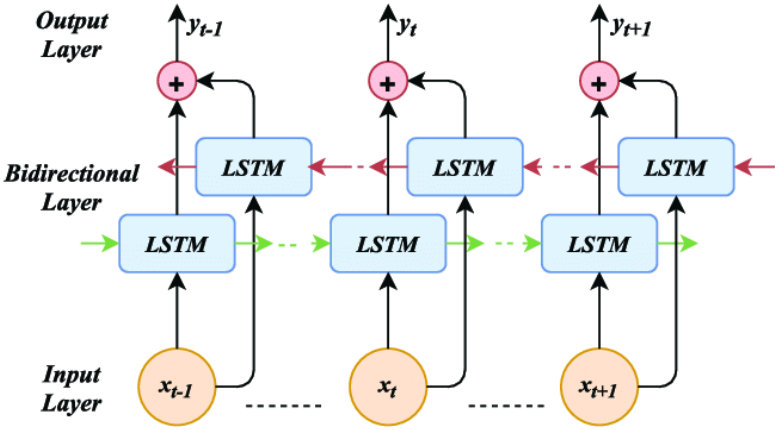
$$h_t = o_t \odot \tanh(C_t)$$

        - h_t sẽ được dùng làm input cho bước tiếp theo hoặc dự đoán cuối cùng.
    - **Ví dụ minh họa:** Output gate có thể chỉ "xuất" "sat on the mat" để dự đoán hành động tiếp theo, bỏ qua chi tiết không liên quan.

❖ **Bi-LSTM (Bidirectional long short term memory):** là cải tiến của LSTM thông thường,

- Sử dụng kết hợp hai luồng LSTM xử lý song song:
    - Forward (tiến): Xử lý sequence từ đầu → cuối
    - Backward (lùi): Xử lý sequence từ cuối → đầu
    - Kết hợp output từ cả 2 chiều để tạo final representation

- **Lý do sử dụng:** Trong nhận dạng chữ viết, Bi-LSTM giúp mô hình hiểu rõ một ký tự bằng cách xem xét cả ngữ cảnh trước và sau nó trong từ, tăng độ chính xác 5-10% so với LSTM một chiều.
- **Hình minh hoạ:**



2.3. Kết hợp CNN-RNN (CRNN) và CTC Loss

- **CRNN:** Mô hình lai mạnh mẽ, kết hợp khả năng trích xuất đặc trưng không gian của CNN với khả năng mô hình hóa chuỗi của RNN.
  - Luồng: **CNN** (Ảnh → Feature Maps) → **Reshape** (Feature Maps → Chuỗi Time-Steps) → **Bi-LSTM** (Mô hình hóa chuỗi).
- **CTC Loss (Connectionist Temporal Classification):** Thuật toán then chốt cho nhận dạng chuỗi từ ảnh. Giải quyết alignment giữa input ảnh và output chuỗi mà không cần label vị trí ký tự.
  - **Ưu điểm:** Cho phép huấn luyện mô hình không cần nhãn phân đoạn (segmentation-free). CTC sử dụng ký tự Blank để xử lý các khoảng cách và ký tự lặp, tự động căn chỉnh (align) chuỗi dự đoán với nhãn thực tế.
  - **Ví dụ**: "A-- B" sẽ align thành "AB"
  - **Công thức loss:**

$$\mathcal{L}_{CTC} = -\log P(\mathbf{y}|\mathbf{x}) = -\log \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{y})} P(\pi|\mathbf{x})$$

    - y: Nhãn thực (chuỗi ký tự).
    - x: Output model (sequence prob).
    - B: Mapping path (thêm blank, lặp) → align tự động.
    - Bỏ 2 timestep đầu: Tránh bias từ đầu sequence.

2.4. CTC Decoding (Giải mã trong quá trình dự đoán)

- **Mục đích:** Tìm ra chuỗi ký tự y^* có xác suất cao nhất từ ma trận xác suất đầu ra của mô hình.
- **Công thức:**

$$y^* = \text{argmax}_y \, P(y|x).$$

- **Các thuật toán phổ biến:**

- **Greedy Search (Tham lam):** Nhanh nhưng không tối ưu về xác suất toàn cục.
- **Beam Search (Tìm kiếm theo chùm):** Chậm hơn nhưng hiệu quả hơn. Thuật toán này giữ lại k chuỗi có xác suất cao nhất tại mỗi bước thời gian (k là Beam Width), giúp tìm ra chuỗi tối ưu trên toàn bộ độ dài.

2.5. Hàm Kích hoạt (Activation Functions)

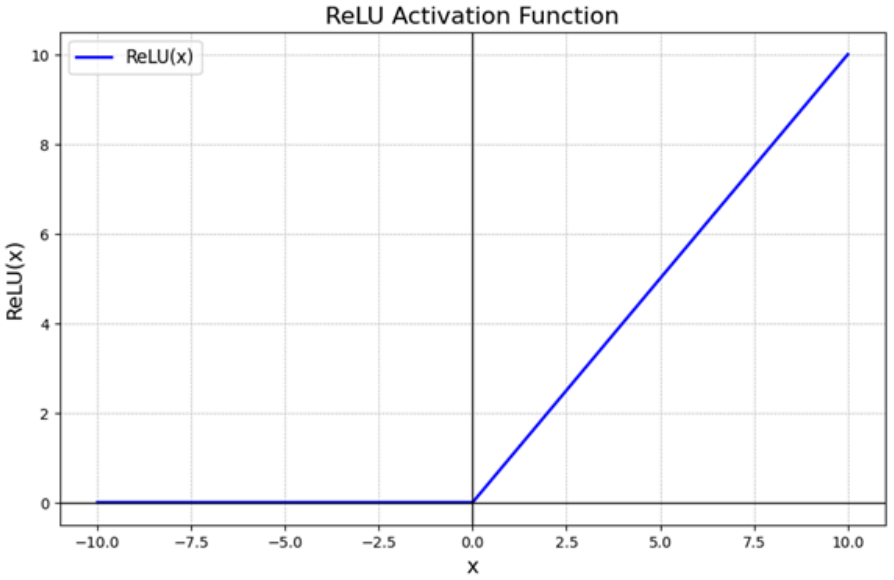
- Hàm kích hoạt là lớp phi tuyến tính được áp dụng cho đầu ra của mỗi nơ-ron. Chúng cho phép mạng neural học các mối quan hệ phức tạp, phi tuyến tính trong dữ liệu.

2.5.1. Relu (Rectified Linear Unit):

- **Khái niệm:** Hàm ReLU là hàm tuyến tính từng phần, có công thức:

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

- **Cách hoạt động:**
  - Nếu đầu vào x > 0, ReLU giữ nguyên giá trị x.
  - Nếu x ≤ 0, ReLU trả về 0.
  - f′(x)=1 nếu x > 0; f′(x)=0 nếu x < 0.
- **Đồ thị:**



- **Ưu điểm:**
  - Không gây vanishing gradient khi x > 0
  - Tính toán nhanh (chỉ so sánh và max, không exp).
  - Tính toán nhẹ – phù hợp GPU
  - Hoạt động tốt cho CNN, MLP, AutoEncoder…
- **Nhược điểm:**
  - Dead ReLU: nếu x luôn âm → neuron chết (gradient = 0 mãi, không học được).
  - Không zero-centered
  - Không mượt ở x = 0.

- Đầu ra không bị giới hạn → có thể gây exploding ở vài mô hình đặc biệt.

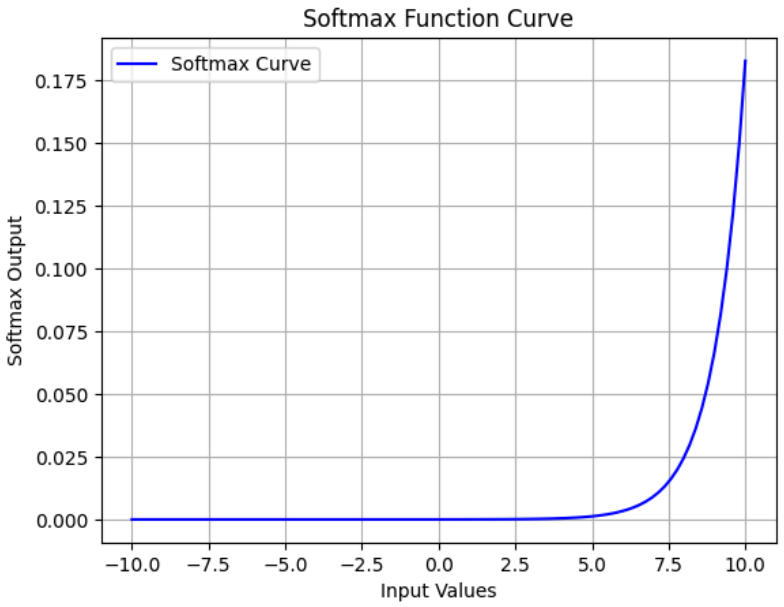
- **Ứng dụng:**
  - Phổ biến trong các hidden layer của mạng nơ-ron sâu (deep neural networks).
  - Giúp huấn luyện nhanh hơn, giải quyết Vanishing Gradient: Giúp giảm thiểu vấn đề đạo hàm tiến về 0 (vanishing gradient) so với các hàm như Sigmoid hay Tanh..
  - Không dùng ở output layer cho bài toán phân loại, mà chỉ dùng ở tầng ẩn.

2.5.2. Softmax (Exponential Normalization)

- **Định nghĩa:** Hàm Softmax được sử dụng ở tầng đầu ra (Output Layer) của các mô hình phân loại đa lớp (Multi-class Classification).
- **Công thức:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

  - với z_i là Logits, C là số lượng lớp
- **Cơ chế:** Biến đổi một vector các giá trị Logits (giá trị thực) thành một phân phối xác suất (Probability Distribution). Tổng xác suất của tất cả các lớp đầu ra bằng 1.
- **Đồ thị:**



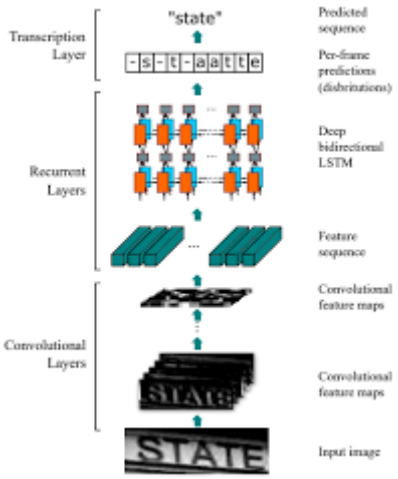
- **Ứng dụng:** Chuẩn hóa đầu ra thành các giá trị xác suất, giúp việc giải thích dự đoán trở nên trực quan, và là đầu vào bắt buộc cho hàm mất mát Cross-entropy (hoặc CTC Loss).

2.6. Bộ Tối ưu hóa (Optimizers)

- **Adam (Adaptive Moment Estimation):** Adam là bộ tối ưu hóa kết hợp các ưu điểm của Momentum và RMSProp. Nó là một trong những thuật toán tối ưu hóa mặc định và mạnh mẽ nhất hiện nay.

- **Cơ chế: Adam duy trì hai ước tính động:**
    - **m (Momentum):** Ước tính trung bình động của gradient bậc 1 (giống như động lượng, giúp tăng tốc theo hướng nhất quán).
    - **v (RMSProp):** Ước tính trung bình động của gradient bậc 2 (bình phương gradient, giúp điều chỉnh tốc độ học riêng cho từng tham số).
- **Ứng dụng:**
    - **Adaptive Learning Rate (Tốc độ học thích ứng):** Cung cấp tốc độ học riêng cho từng trọng số, giúp mô hình hội tụ nhanh hơn và ổn định hơn.
    - **Hiệu suất cao:** Thường cho hiệu suất tốt hơn các phương pháp cổ điển (như SGD) trên các tập dữ liệu lớn và kiến trúc phức tạp (Như CRNN).

PHẦN II. BÀI TOÁN 1 - NHẬN DIỆN CHỮ VIẾT TAY



1. Phân tích và Tiền xử lý dữ liệu chữ viết tay

1.1. Bộ dữ liệu

- **Nguồn dataset**: Sử dụng **Written Name Recognition Dataset** (Kaggle), một tập con đã được xử lý từ các nguồn chữ viết tay lớn hơn (ví dụ: IAM Database).
- **Đặc điểm:** Mặc dù bộ dữ liệu gốc có thể chứa đến 400,000 mẫu, dự án này nhóm chỉ sử dụng tập dữ liệu đã được phân chia sẵn ở cấp độ từ để đảm bảo tính khả thi trong môi trường tính toán giới hạn.
    - **Nguồn dữ liệu:** các file: *written_name_train_v2.csv, validation_v2.csv, test_v2.csv* chứa đường dẫn ảnh ở cấp độ từ và nhãn tương ứng.
    - **Kích thước mẫu:** khoảng 34.000 mẫu, là chuỗi văn bản chữ viết tay tên riêng tiếng Anh. Ảnh ban đầu không đồng nhất với sự đa dạng cao về nét chữ, độ nghiêng và kích thước, nhưng đã được xử lý về định dạng ảnh Grayscale.
    - **Tập từ vựng:** 29 ký tự (A-Z, -, ',) + 1 ký tự *blank* cho CTC, tổng cộng có 30 lớp.
    - **Phân chia dữ liệu:**
        - Tập Train: ~ 27.000 ảnh (80% tổng).
        - Tập Validation: ~ 3.000 ảnh (9% tổng).
        - Tập Test: ~ 4.000 ảnh (11% tổng).

1.2. Các bước tiền xử lý đặc trưng

1.2.1. Tiền xử lý Dữ liệu nhãn (Labels): Quá trình này diễn ra trong hàm *parse_data và get_new_label:*

- Chuẩn hóa và lọc nhãn:
    - Xóa các dòng thiếu nhãn
    - Loại bỏ các dòng có nhãn là “UNREADABLE” (không thể đọc được)
    - Chuyển tất cả nhãn về chữ in hoa.
- Mã hoá nhãn (Label encoding):

- Sử dụng **alphabet_dict** để chuyển đổi từng ký tự trong nhãn (ví dụ: 'A', 'B', ' ') thành một chỉ mục số nguyên tương ứng (ví dụ: 0, 1, 28).

- Đệm nhãn (Label padding):
  - Tạo một ma trận nhãn có kích thước cố định, tất cả các chuỗi nhãn đều có cùng độ dài là 34.
  - Vì ban đầu các chuỗi có số ký tự khác nhau nên cần đệm thêm giá trị (-1) nếu chuỗi ngắn hơn kích thước cố định.
  - ```
    labels = np.ones([len(df),34]) * (-1)
    ```

1.2.2. Tiền xử lý ảnh (Image processing): Quá trình này diễn ra trong hàm *process_data*:

- Đọc file ảnh từ đường dẫn và ***chuyển về ảnh Grayscale*** (ảnh trắng đen):
  - Việc này bảo đảm ảnh có 1 kênh màu, giúp giảm tham số mô hình
  - ```
    image = tf.image.decode_image(image, channels=1, expand_animations=False)
    ```
- **Cắt ảnh (Cropping):**
  - Đặt kích thước mục tiêu cho ảnh là 64x256
  - Để tránh ảnh quá to làm chậm mô hình, nhưng giữ nguyên tỷ lệ (không méo). Tập trung vào phần chữ chính (thường ở trên-trái).
  - Nếu ảnh gốc lớn hơn kích thước mục tiêu, cắt bỏ đi phần dư thừa từ góc trên-trái (offset=0).
- **Đệm ảnh (Padding):**
  - Để đảm bảo tất cả ảnh có kích thước đồng nhất, dễ xử lý theo batch.
  - Ta đệm thêm vào ảnh giá trị 255 (màu trắng), ở bên dưới và bên phải ảnh, không thêm bên trên/trái để giữ chữ ở vị trí gốc.
  - Nền trắng phù hợp chữ viết tay (tránh nhiễu đen). Không dùng resize (kéo giãn) để tránh làm chữ mờ/méo.
  - ```
    if pad_height > 0 or pad_width > 0:
        image = tf.pad(image, paddings=[[0, pad_height], [0, pad_width], [0, 0]], constant_values=255)
    ```
- **Xoay ảnh (Rotation):**
  - Dataset handwriting thường ảnh ngang, nhưng mô hình CRNN cần input dọc (height nhỏ, width lớn) để xử lý thông tin tuần tự theo chiều ngang của chữ viết.
  - Xoay ảnh 270 độ theo chiều kim đồng hồ (k=3 nghĩa là xoay 3 lần 90° clockwise, tương đương 90° ngược chiều).
  - ```
    image = tf.image.rot90(image, k=3)
    ```
- **Chuẩn hoá giá trị pixel (Normalization):**

- Mô hình neural network học tốt hơn với giá trị nhỏ [0,1], tránh overflow số lớn, ổn định gradient.
- Chuyển giá trị pixel từ dải [0, 255] về dải [0, 1].

→ **Kết quả ảnh sau quá trình tiền xử lý ảnh:** Tensor 256x64x1 (float32, [0,1]), sẵn sàng cho mô hình.

1.2.3. **Tối ưu hoá Data pipeline:** Quá trình này diễn ra trong hàm *create_dataset*:

- **Xử lý song song:**
  - ```
    dataset = dataset.map(process_data, num_parallel_calls=tf.data.AUTOTUNE)
    ```
  - Sử dụng num_parallel_calls để cho phép CPU xử lý đồng thời nhiều ảnh, tận dụng tài nguyên tối đa.
- **Gộp Lô (Batching):**
  - ```
    dataset = dataset.batch(BATCH_SIZE)
    ```
  - Gộp các mẫu đã xử lý thành lô (120 mẫu) trước khi đưa vào huấn luyện, giúp tận dụng tính toán song song của GPU.
- **Tải trước (Prefetching):**
  - ```
    dataset = dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
    ```
  - Tải trước lô dữ liệu tiếp theo vào bộ nhớ khi GPU đang xử lý lô hiện tại, loại bỏ tình trạng "GPU đói" và tối đa hóa hiệu suất huấn luyện.

2. Kiến trúc và Huấn luyện Mô hình CRNN

2.1. Kiến trúc Mô hình (CRNN)

- Mô hình CRNN được thiết kế để xử lý tuần tự (sequence-to-sequence), nhận đầu vào là ảnh và trả về chuỗi ký tự.
- Kiến trúc được xây dựng theo mô hình lai (Hybrid) với 3 khối chính: CNN (Feature Extraction) → Reshape → Bi-LSTM (Sequence Modeling).
- **Đầu vào:** kích thước ảnh là 256x64x1, đây là ảnh đã được xoay (256 là chiều Time-step, 64 là chiều Feature).

```
input_data = Input(shape=(256, 64, 1), name='input')
```

- **CNN Backbone (Trích chọn Đặc trưng):** Sử dụng 2 khối Conv2D kết hợp BatchNormalization và Activation('relu').
  - **Block 1 (Extraction):**

```
inner = Conv2D(32, (3, 3), padding='same', name='conv1', kernel_initializer='he_normal')(input_data)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max1')(inner)
```

    - **Conv2D (32 filters, 3x3):** dùng để trích xuất low-level features (cạnh, nét chữ). Lọc đặc trưng cơ bản (cạnh, nét). *padding='same' để* giữ kích thước. *he_normal* init tránh vanishing gradient. → đầu ra ảnh 256x64x32.
```

- **BatchNormalization():** Ổn định hóa quá trình huấn luyện và tăng tốc độ hội tụ bằng cách chuẩn hóa đầu vào của tầng tiếp theo (ReLU)
- **Relu:** Tăng tính phi tuyến cho mô hình, cho phép mạng học các mối quan hệ phức tạp.
- **MaxPooling2D(pool_size=(2, 2))**: Giảm kích thước không gian xuống còn 1/4 (128x32), giúp mô hình giảm số lượng tham số và tăng tính bất biến với sự dịch chuyển nhỏ của nét chữ. → đầu ra ảnh 128x32x32.

○ **Block 2 (Deep Extraction & Regularization):**

```
inner = Conv2D(64, (3, 3), padding='same', name='conv2', kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max2')(inner)
inner = Dropout(0.3)(inner)
```

- **Conv2D(64 filters, 3x3):** Tiếp tục trích xuất đặc trưng mid-level features (các hình dạng ký tự phức tạp). Tăng số lượng bộ lọc từ 32 lên 64. → đầu ra ảnh 128x32x64.
- **MaxPooling2D(pool_size=(2, 2)):** Tiếp tục giảm kích thước không gian lần nữa. Đầu ra cuối cùng của CNN trước khi chuyển sang RNN. → đầu ra ảnh 64x16x64.
- **Dropout(0.3):** Kỹ thuật điều chỉnh (Regularization). Ngẫu nhiên vô hiệu hóa 30% nơ-ron, buộc mạng phải học các đặc trưng mạnh mẽ hơn và ngăn ngừa Overfitting.

- **Chuyển đổi và Nén Đặc trưng (Bridge Layers):**
  - Đây là khối quan trọng để chuyển đổi dữ liệu hình ảnh 2D sang chuỗi 1D mà RNN có thể xử lý

```
inner = Reshape(target_shape=((64, 1024)), name='reshape')(inner)
inner = Dense(64, activation='relu', kernel_initializer='he_normal', name='dense1',)(inner)
```

  - **Reshape:** Chuẩn bị cho RNN xử lý theo sequence (mỗi cột ảnh là timestep).
    - Biến đổi chiều dữ liệu. Từ (64, 16, 64) → (64, 1024) – Flatten theo chiều cao.
    - 16x64=1024 thành chiều feature (đặc trưng).
    - 64 trở thành chiều Time-Steps, đại diện cho độ dài chuỗi (tối đa 64 ký tự).
  - **Dense(64, activation='relu'):** Giảm chiều, giúp LSTM dễ học.
    - Nén chiều đặc trưng. Giảm số lượng Feature từ 1024 → 64.
    - Việc nén này giúp giảm độ phức tạp đầu vào cho tầng LSTM mà không mất đi quá nhiều thông tin cốt lõi.
    - Đầu ra ảnh 64x64.
- **RNN Block (Học Chuỗi):**

```
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm1')(inner)
inner = Bidirectional(LSTM(256, return_sequences=True), name = 'lstm2')(inner)
```

- Khối này xử lý các đặc trưng đã được trích chọn như một chuỗi thời gian. Mô hình hóa phụ thuộc ký tự (ví dụ như 'th' thường đi theo nhau).
- **2 khối Bi-LSTM:** Học ngữ cảnh theo chuỗi
  - LSTM hai chiều (forward + backward) học ngữ cảnh toàn chuỗi, nắm bắt mối quan hệ ngữ cảnh của ký tự, nhớ dài hạn.
  - *return_sequences=True* để output mỗi timestep.
  - 256 đơn vị x 2 chiều = 512 đơn vị → đầu ra ảnh 64x512, giúp tăng cường khả năng học chuỗi sâu hơn.

- **Output Layer:**

```
inner = Dense(30, kernel_initializer='he_normal',name='dense2')(inner)
y_pred = Activation('softmax', name='softmax')(inner)
```

  - **Dense(30):** Tạo Logits. Chuyển đổi đầu ra của LSTM thành 30 giá trị Logits (xác suất thô) tại mỗi Time-step. (30 = 29 ký tự + 1 Blank). → đầu ra ảnh 64x30.
  - **Activation('softmax'):** Tạo xác suất. Chuẩn hóa Logits thành xác suất, đảm bảo tổng xác suất của 30 lớp tại mỗi Time-step bằng 1.

2.2. Hàm mất mát (CTC Loss)

- CTC Loss được triển khai thông qua Lambda layer vì nó cần tính toán đặc biệt (không phải là phép toán tuyến tính).

```
ctc_loss = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([y_pred, labels, input_length, label_length])
```

- **Điểm quan trọng:**

```
y_pred = y_pred[:, 2:, :]
```

  - Dòng code này cắt bỏ 2 Time-steps đầu tiên của đầu ra RNN.
  - Lý do là đầu ra của RNN (đặc biệt là LSTM) ở những bước thời gian đầu tiên thường không ổn định và có xu hướng là "rác" (garbage), việc loại bỏ chúng có thể giúp cải thiện độ chính xác tổng thể.
- Hàm tính loss: *ctc_batch_cost()*

2.3. Quá trình huấn luyện

- **Mô hình huấn luyện (*model_final*):**

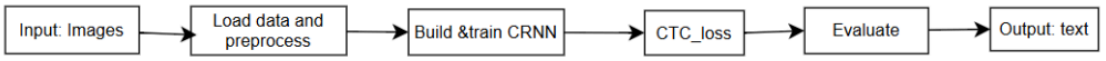
```
model_final = Model(inputs=[input_data, labels, input_length, label_length], outputs=ctc_loss)

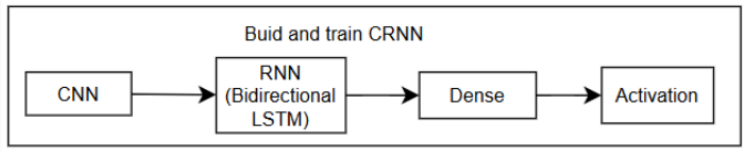
model_final.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=Adam(learning_rate = 0.0001),metrics=["accuracy"])
history=model_final.fit(train_ds, validation_data=val_ds, epochs=25)
```

  - Mô hình này nhận 4 đầu vào (input_data, labels, input_length, label_length).
  - Đầu ra duy nhất là giá trị của CTC Loss (ctc_loss)
- **Optimizer**: Adam(learning_rate = 0.0001)
  - Học sâu với CTC Loss thường đòi hỏi Learning Rate rất nhỏ để đảm bảo quá trình hội tụ diễn ra mượt mà và không bị phân kỳ.
- **Epoch:** Huấn luyện trên 25 Epochs.

- **Monitor:** History keys: ['loss', 'accuracy', 'val_loss', 'val_accuracy'].

2.4. Tổng kết luồng hoạt động

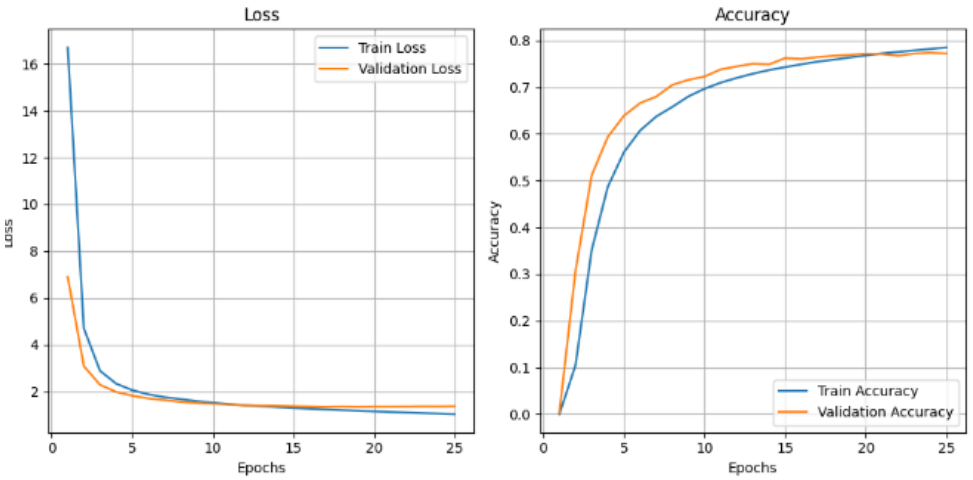




3. Kết quả và Đánh giá hiệu suất mô hình Chữ viết Tay

3.1. Trực quan hóa Quá trình huấn luyện

- Việc trực quan hóa bằng biểu đồ cho Loss và Accuracy là cần thiết để đánh giá chất lượng hội tụ của mô hình.
- **Kết quả thu được:**



- **Nhận xét:**
  - **Biểu đồ Loss**
    - Cả Train Loss và Validation Loss đều giảm ổn định qua 25 Epochs.
    - Kết luận: Sự giảm đều của Validation Loss cho thấy mô hình đang học được các đặc trưng tốt và không bị Overfitting quá mức.
  - **Biểu đồ Accuracy:**
    - accuracy ở đây là độ chính xác được tính theo phương pháp truyền thống (so sánh giá trị argmax) và chỉ mang tính tham khảo, không phải là độ chính xác thực tế theo CTC (Word/Character Accuracy).
    - Tuy nhiên, sự tăng trưởng ổn định cho thấy mô hình đang hoạt động đúng hướng

3.2. Đánh giá hiệu suất mô hình

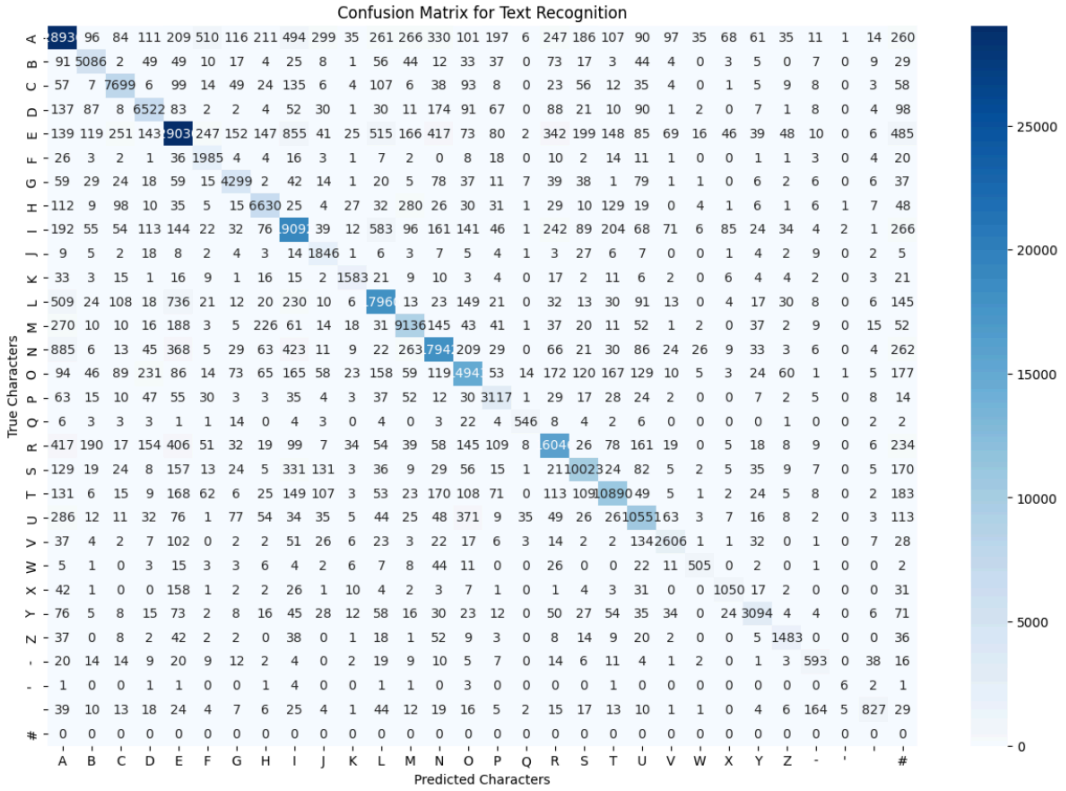
- Hiệu suất (evaluation) của mô hình nhận dạng chữ viết tay được đánh giá trên tập Test và Validation bằng cách giải mã CTC (Greedy Search) và so sánh với nhãn thực tế.

```
    preds = model.predict(image)
    decoded = K.get_value(K.ctc_decode(preds, input_length=np.ones(preds.shape[0])*preds.shape[1], greedy=Tru
e)[0][0])
```

  - Sử dụng *Greedy Search* để chuyển đầu ra xác suất thành chuỗi ký tự.
- Trên tập Validation:
  - Correct characters predicted : 87.03%
  - Correct words predicted : 76.86%
- Trên tập Test:
  - Correct characters predicted : 86.59%
  - Correct words predicted : 76.30%
- **Nhận xét:**
  - Độ chính xác cấp độ từ thấp hơn cấp độ ký tự là điều bình thường. Việc một từ dài bị sai dù chỉ một ký tự cũng bị tính là sai từ, làm giảm đáng kể Word Accuracy.

3.3. Phân tích và đánh giá chung

- **Ma trận Nhầm lẫn (Confusion Matrix):** Trực quan hóa số lần mô hình nhầm lẫn ký tự thực tế (True) với ký tự dự đoán (Predicted).



- **Báo cáo Phân loại:**
  - **Ký tự Mạnh (F1-Score cao):** 'C' (0.90), 'L' (0.89), 'A' (0.87), 'E' (0.88). Đây là các ký tự có độ phân biệt tốt.
  - **Ký tự Yếu (F1-Score thấp):**

- Ký tự có Support thấp ('): F1-Score chỉ 0.31 (do chỉ có 23 mẫu).
- Ký tự có Support thấp ('F', 'J', 'W'): Có F1-Score thấp hơn (0.76 - 0.78), thường do sự biến đổi lớn trong nét viết hoặc số lượng mẫu huấn luyện ít hơn các ký tự phổ biến khác.

○ **Kết quả:**

	precision	recall	f1-score	support
A	0.88	0.86	0.87	33474
B	0.87	0.89	0.88	5718
C	0.90	0.90	0.90	8566
D	0.86	0.85	0.86	7631
E	0.89	0.86	0.88	33895
F	0.65	0.91	0.76	2183
G	0.86	0.87	0.87	4936
H	0.87	0.87	0.87	7631
I	0.85	0.87	0.86	21955
J	0.68	0.92	0.78	2004
K	0.86	0.87	0.87	1819
L	0.89	0.89	0.89	20249
M	0.87	0.87	0.87	10456
N	0.90	0.86	0.88	20892
O	0.89	0.87	0.88	17164
P	0.78	0.85	0.81	3653
Q	0.87	0.85	0.86	642
R	0.90	0.87	0.88	18449
S	0.90	0.88	0.89	11378
T	0.91	0.87	0.89	12497
U	0.88	0.88	0.88	12022
V	0.86	0.83	0.84	3141
W	0.83	0.74	0.78	687
...				
accuracy			0.87	270272
macro avg	0.80	0.80	0.80	270272
weighted avg	0.88	0.87	0.87	270272

PHẦN III. BÀI TOÁN 2 - NHẬN DẠNG HÌNH ẢNH

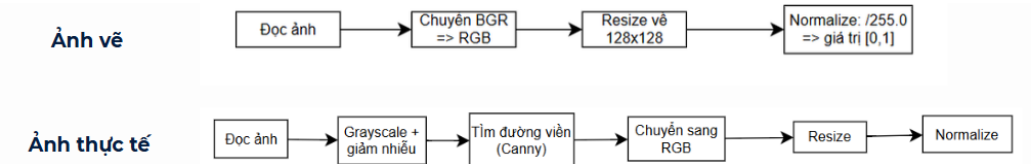
1. Phân tích và Tiền xử lý dữ liệu

1.1. Bộ dữ liệu

- **Nguồn dataset:** Sử dụng **Geometric Shapes Dataset** (Kaggle), một bộ dữ liệu synthetic đơn giản cho nhận dạng hình dạng 2D đơn giản, phù hợp với bài toán classification cơ bản.
- **Đặc điểm:**
  - Dataset bao gồm ảnh PNG 128x128 pixels (RGB), nền trắng, hình đen rõ nét, không nhiễu phức tạp, với đa dạng góc quay và kích thước để tăng tính tổng quát.
  - **Số lớp:** 3 (Circle - Tròn, Square - Vuông, Triangle - Tam giác).
  - **Tổng số mẫu:** 30.000 ảnh, chia đều 10.000 ảnh cho mỗi lớp.
  - **Phân chia tập dữ liệu:**
    - Tập Train: 19,200 mẫu (64%)
    - Tập Validation: 4,800 mẫu (16%)
    - Tập Tes: 6,000 mẫu (20%)

1.2. Các bước tiền xử lý đặc trưng

- Tiền xử lý tập trung vào chuẩn hóa kích thước, tăng cường dữ liệu (augmentation) để xử lý biến đổi (rotation, flip), và edge detection để nhấn mạnh đường viền hình dạng – giúp CNN tập trung vào đặc trưng hình học thay vì pixel raw.



1.2.1. Tiền xử lý Dữ liệu nhãn (Labels)

- **Chuẩn hóa và lọc nhãn:**
  - Tự động map thư mục thành labels (ví dụ: 'Circle/' → class 0).
  - Loại bỏ file không hợp lệ qua class_mode='categorical'
- **Mã hoá nhãn:**
  - Tự động chuyển nhãn (Circle, Square, Triangle) thành định dạng **One-Hot Encoding** (ví dụ: [1, 0, 0]).

1.2.2. Tiền xử lý ảnh (Image processing)

- **Đọc file ảnh từ đường dẫn và chuyển về ảnh RGB chuẩn:**
  - Load bằng OpenCV (cv2.imread), giữ RGB (3 channels) để tương thích CNN.
  - Việc này đảm bảo ảnh có định dạng nhất quán, giúp tăng tốc decode.
- **Trích xuất viền (Edge Detection - Kỹ thuật cốt lõi):**
  - Chuyển ảnh sang grayscale → dùng *cv2.cvtColor(..., COLOR_BGR2GRAY)* để loại bỏ thông tin màu, chỉ giữ lại độ sáng – phù hợp cho phân tích biên dạng

- Làm mờ bằng Gaussian Blur để giảm nhiễu, tránh phát hiện những cạnh giả không cần thiết.
- Phát hiện đường viền cạnh bằng Canny
  - Ngưỡng thấp = 50 → nhận cạnh yếu
  - Ngưỡng cao = 150 → nhận cạnh mạnh
  - Kết quả là một ảnh dạng nhị phân chỉ gồm các đường viền trắng trên nền đen.
- Chuyển ảnh viền sang dạng 3 kênh (RGB) → *cv2.cvtColor(edges, COLOR_GRAY2BGR)* để phù hợp khi đưa vào CNN (mặc định đầu vào 3 channel).

- **Resize:** Data Generator đọc ảnh từ thư mục tải lên và tự động resize tất cả ảnh về kích thước cố định, 128x128 pixels.
- **Chuẩn hóa giá trị pixel (Normalization):**
  - Resize về (128,128) (cv2.resize hoặc generator)
  - Kỹ thuật: rescale=1./255 (Chia giá trị pixel cho 255)
  - Mục đích: Chuyển giá trị pixel từ dải [0, 255] về dải [0, 1]. Điều này giúp ổn định gradient và tăng tốc độ hội tụ của mô hình.
- **Kết quả ảnh sau quá trình tiền xử lý ảnh**: Tensor 128x128x3 (float32, [0,1]), viền nổi bật, sẵn sàng cho CNN.

2. Kiến trúc và Huấn luyện mô hình

2.1. Kiến trúc Mô hình (CNN)

- Mô hình được xây dựng theo kiến trúc Sequential CNN gồm 3 khối Tích chập sâu và một khối Phân loại (Dense Head): CNN (Feature Extraction) → Flatten → Dense (Classification).
- **Đầu vào:** ảnh 128x128x3 (Cao, Rộng, Kênh RGB).
- **CNN Backbone (Trích chọn Đặc trưng):** Sử dụng 3 khối Conv2D kết hợp BatchNormalization, ReLU, MaxPooling và Dropout.
  - **Block 1 (Low-level Extraction)**

```
            # Block 1
            layers.Conv2D(32, (3, 3), activation='rel
u', input_shape=self.input_shape),
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.25),
```

    - **Conv2D (32 filters, 3x3):** Trích xuất đặc trưng cấp thấp. 32 bộ lọc bắt các cạnh và đường nét cơ bản của hình dạng. → Đầu ra ảnh 126x126x32 (valid padding mặc định)
    - **BatchNormalization()**: Chuẩn hóa, ổn định quá trình huấn luyện.
    - **MaxPool(2, 2):** Giảm kích thước xuống ¼ → 63x63x32.
    - **Dropout(0.25):** ngăn ngừa overfitting

- **Block 2 (Mid-level Extraction & Regularization)**

```
            # Block 2
            layers.Conv2D(64, (3, 3), activation='rel
u'),
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.25),
```

  - **Conv2D(64 filters, 3x3):** Trích xuất đặc trưng trung cấp (góc, đường cong, texture). Tăng số lượng bộ lọc lên 64→ Đầu ra ảnh 61x61x64.
  - **MaxPool(2, 2):** Giảm kích thước → 30x30x64
- **Block 3 (High-level Extraction)**

```
            # Block 3
            layers.Conv2D(128, (3, 3), activation='re
lu'),
            layers.BatchNormalization(),
            layers.MaxPooling2D((2, 2)),
            layers.Dropout(0.25),
```

  - **Conv2D(128 filters, 3x3):** Trích xuất đặc trưng cấp cao (tổng thể hình dạng). Tăng số lượng bộ lọc lên 128 → Đầu ra ảnh 28x28x128.
  - **MaxPool(2, 2):** Giảm kích thước → 14x14x128

- **Tầng Phân loại (Dense Head):**

```
            # Dense layers
            layers.Flatten(),
            layers.Dense(256, activation='relu'),
            layers.BatchNormalization(),
            layers.Dropout(0.5),
            layers.Dense(128, activation='relu'),
            layers.Dropout(0.5),
            layers.Dense(self.num_classes, activation
='softmax')
```

  - **Flatten:** Chuyển đổi Feature Map 3D (14x14x128) thành Vector 1D (25.088 dims) để đưa vào mạng Dense..
  - **Dense(256, activation='relu'):** Tầng ẩn. Học các tổ hợp phi tuyến tính của các đặc trưng trích xuất → 256 dims.
  - **BatchNormalization() + Dropout(0.5):** Normalize + drop 50%.

- **Dense(128, activation='relu') + Dropout(0.5):** Tinh chỉnh và giảm chiều.
- **Output Layer: Dense(3) + Softmax:** Tầng đầu ra. 3 đơn vị tương ứng với 3 lớp. Softmax chuyển đổi Logits thành xác suất, tổng xác suất bằng 1.

2.2. Quá trình huấn luyện

- **Mô hình huấn luyện (model):** Sequential, nhận input ảnh, output softmax 3 lớp.

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

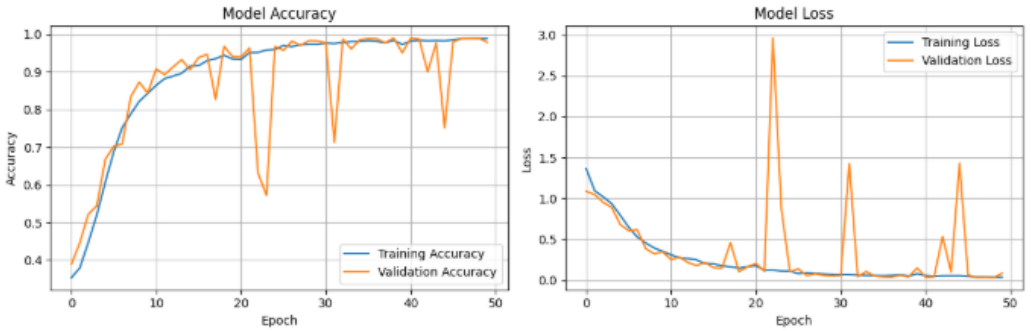
- **Optimizer:** Adam (lr=0.001 mặc định).
- **Epoch**: Huấn luyện trên 30 Epochs.
- **Batch size:** 32
- **Monitor**: History keys: ['loss', 'accuracy', 'val_loss', 'val_accuracy'].
- **Hàm tính loss:** categorical_crossentropy() từ Keras

2.3. Tổng kết luồng hoạt động

3. Kết quả và Đánh giá hiệu suất mô hình Hình dạng

3.1. Trực quan hóa Quá trình Huấn luyện

- **Kết quả thu được:**

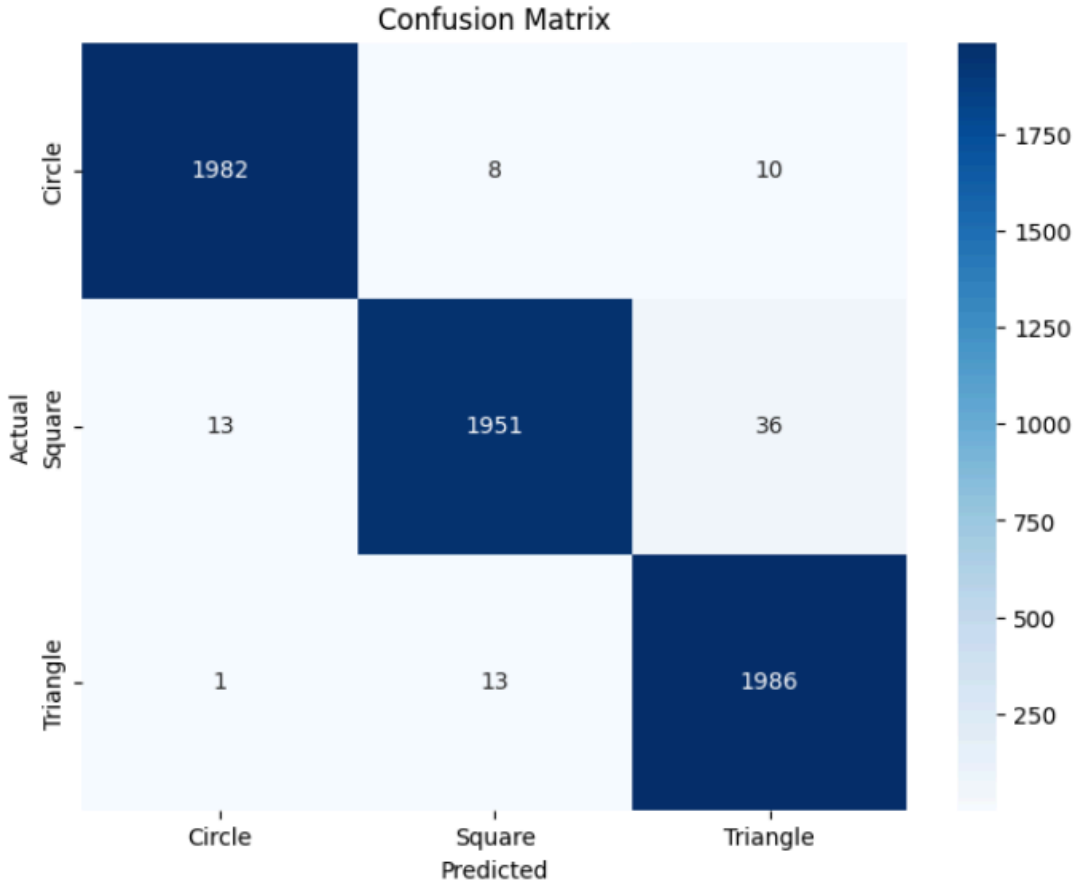


- **Nhận xét:**
  - **Tốc độ Hội tụ:** Mô hình đạt độ chính xác cao rất nhanh (trên 95% chỉ sau khoảng 10 Epochs) do tính chất đơn giản của bộ dữ liệu (ảnh sạch, ít biến đổi).
  - **Biểu đồ Loss:** Train Loss và Validation Loss giảm nhanh và ổn định.
  - **Biểu đồ Accuracy:** Train Accuracy và Validation Accuracy tăng nhanh và duy trì ở mức rất cao (trên 98%).

- Sự chênh lệch giữa Train Loss và Validation Loss là rất nhỏ. Chứng tỏ mô hình được điều chỉnh tốt bằng BatchNorm và Dropout, đạt khả năng khái quát hóa tốt và không bị overfitting.

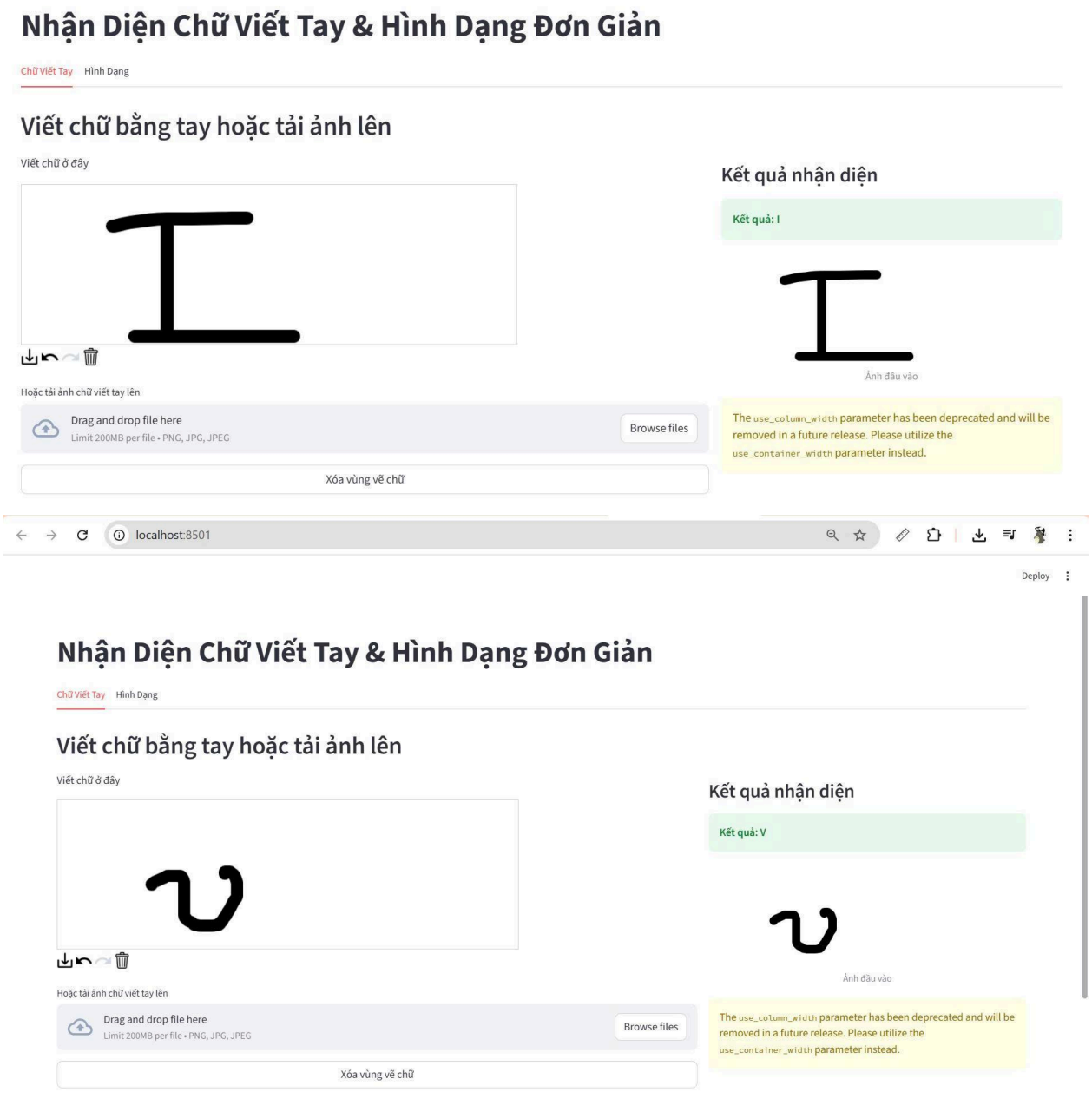
3.2. Đánh giá hiệu suất mô hình

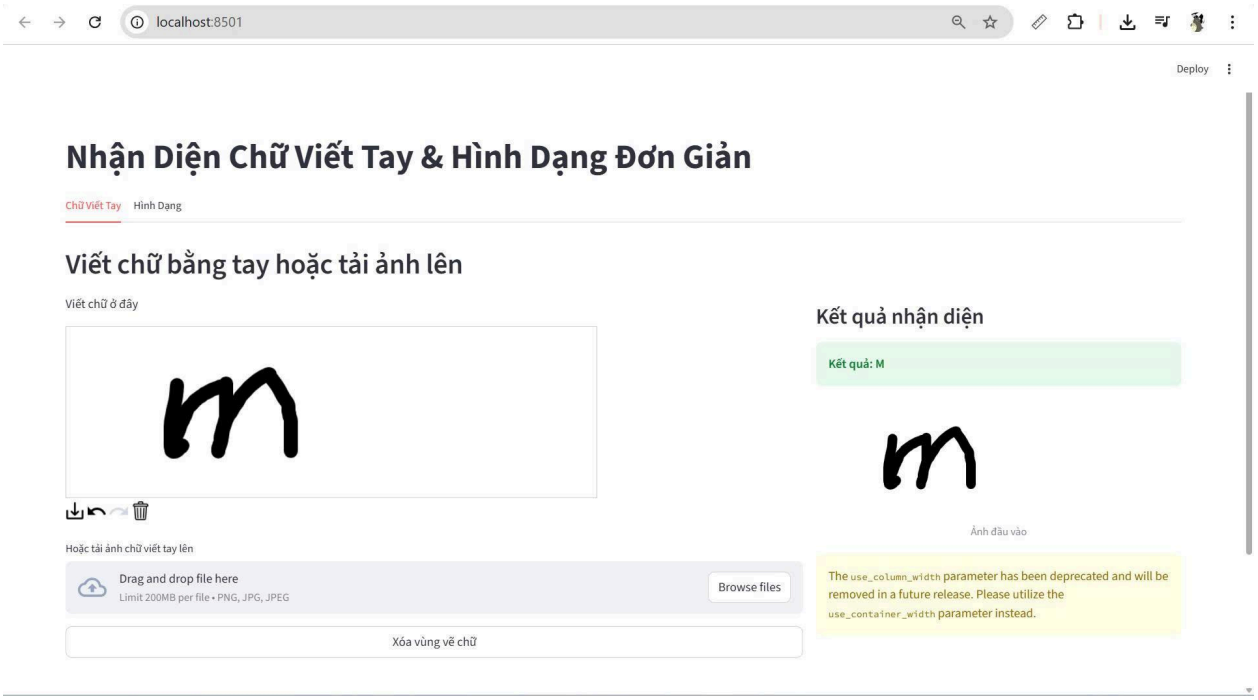
- **Ma trận Nhầm lẫn (Confusion Matrix):** diagonal mạnh, ít nhầm lẫn

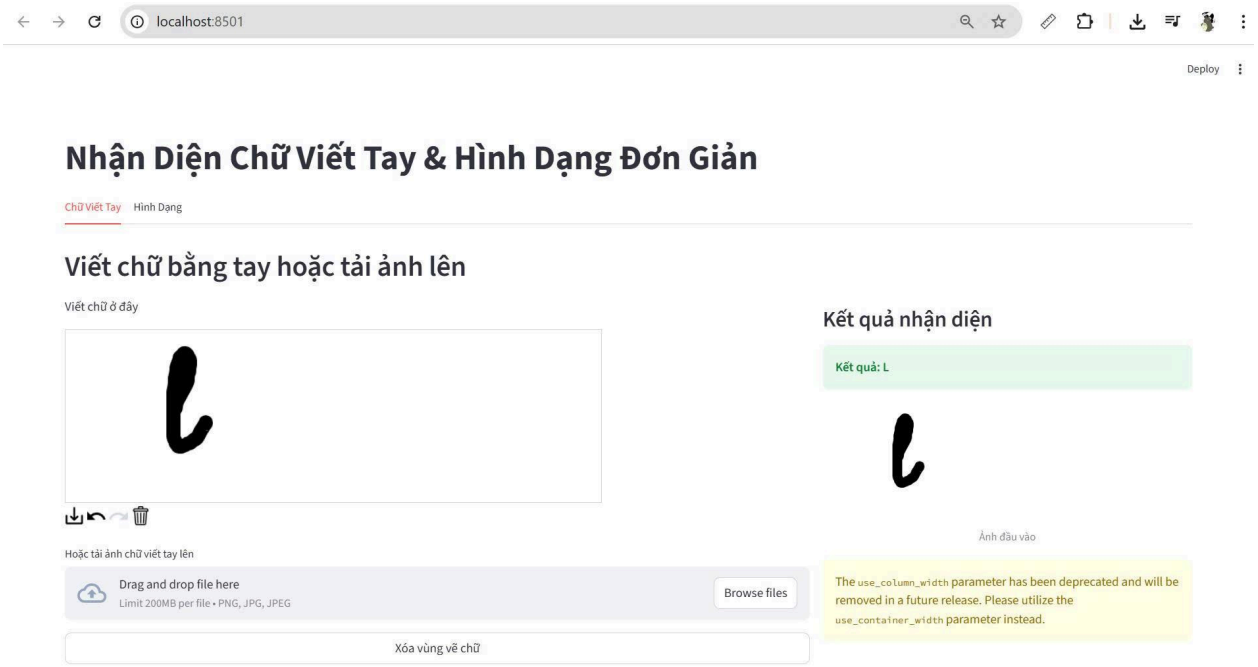


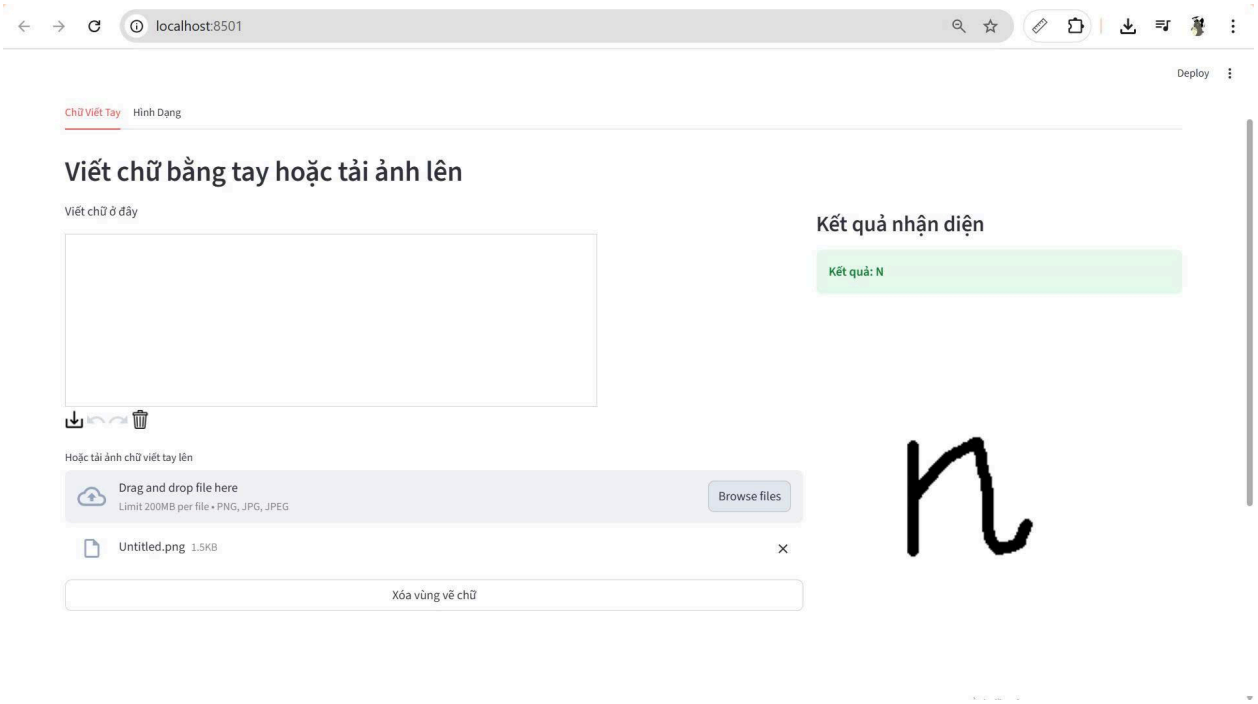
PHẦN IV. XÂY DỰNG TOOL

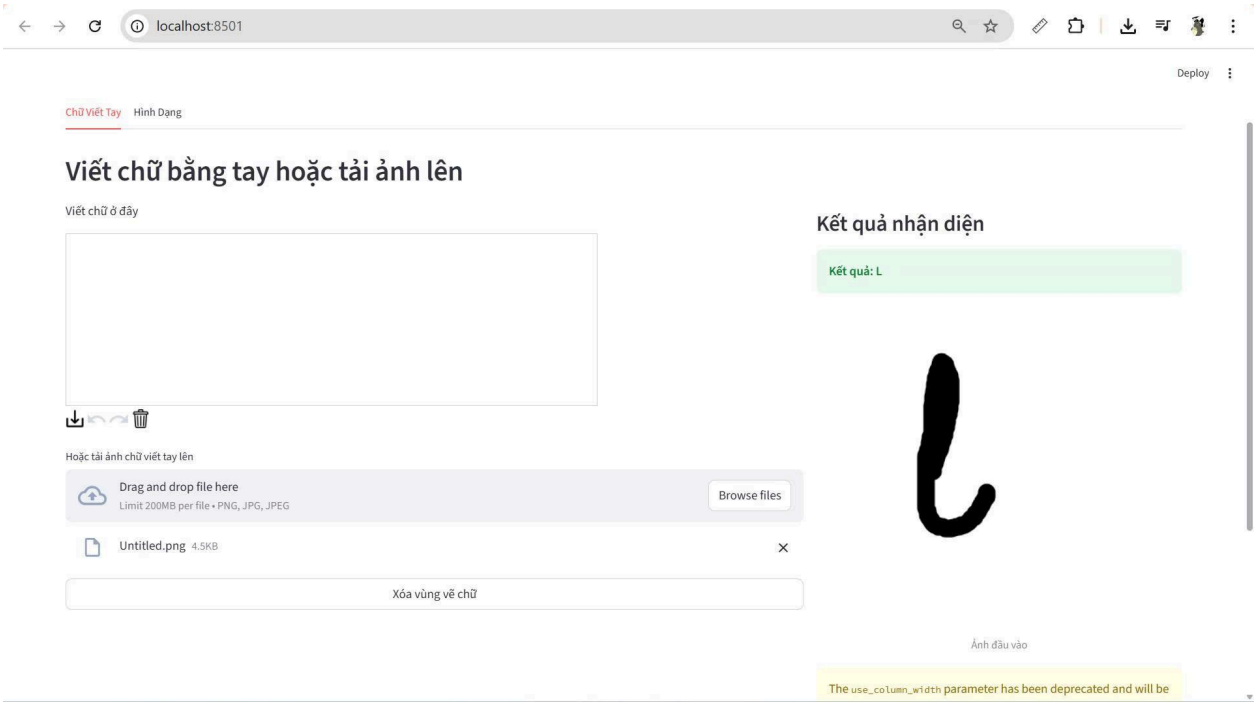
1. Nhận diện chữ viết tay

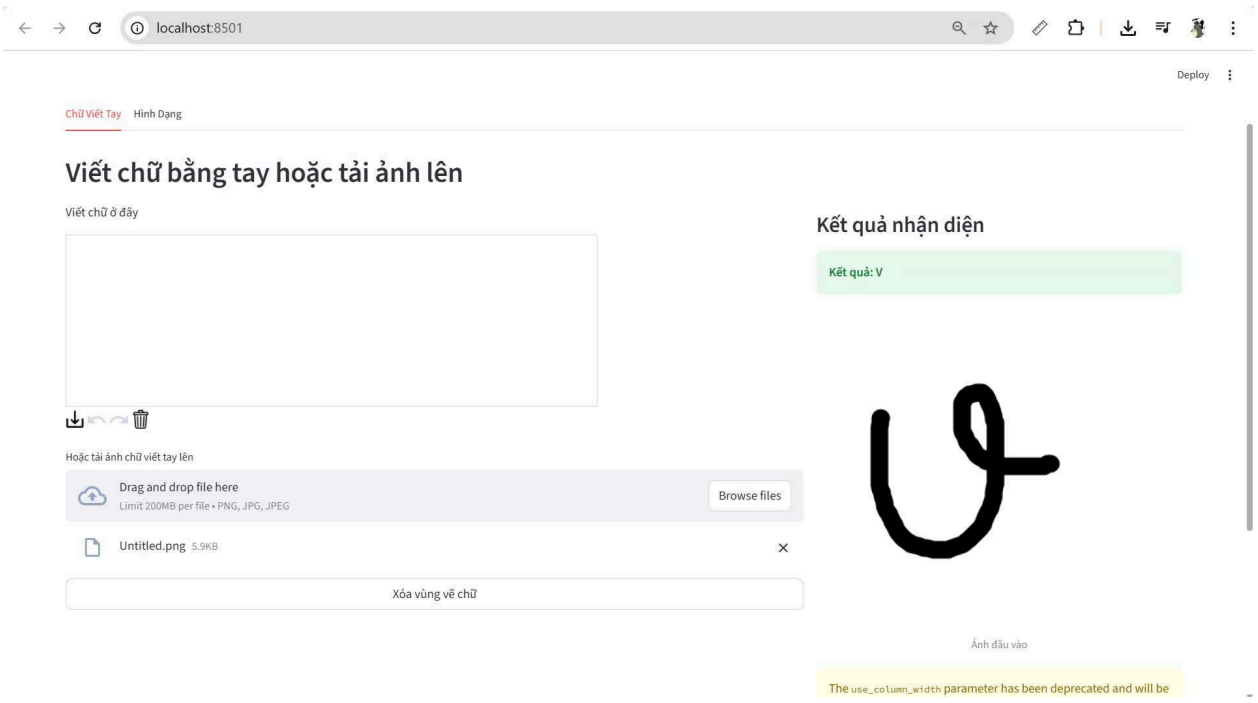



localhost:8501
Deploy
Nhận Diện Chữ Viết Tay & Hình Dạng Đơn Giản
Chữ Viết Tay
Hình Dạng
Viết chữ bằng tay hoặc tải ảnh lên
Viết chữ ở đây
Hoặc tải ảnh chữ viết tay lên
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
Xóa vùng vẽ chữ
Kết quả nhận diện
Kết quả: M
Ảnh đầu vào
The use_column_width parameter has been deprecated and will be removed in a future release. Please utilize the use_container_width parameter instead.

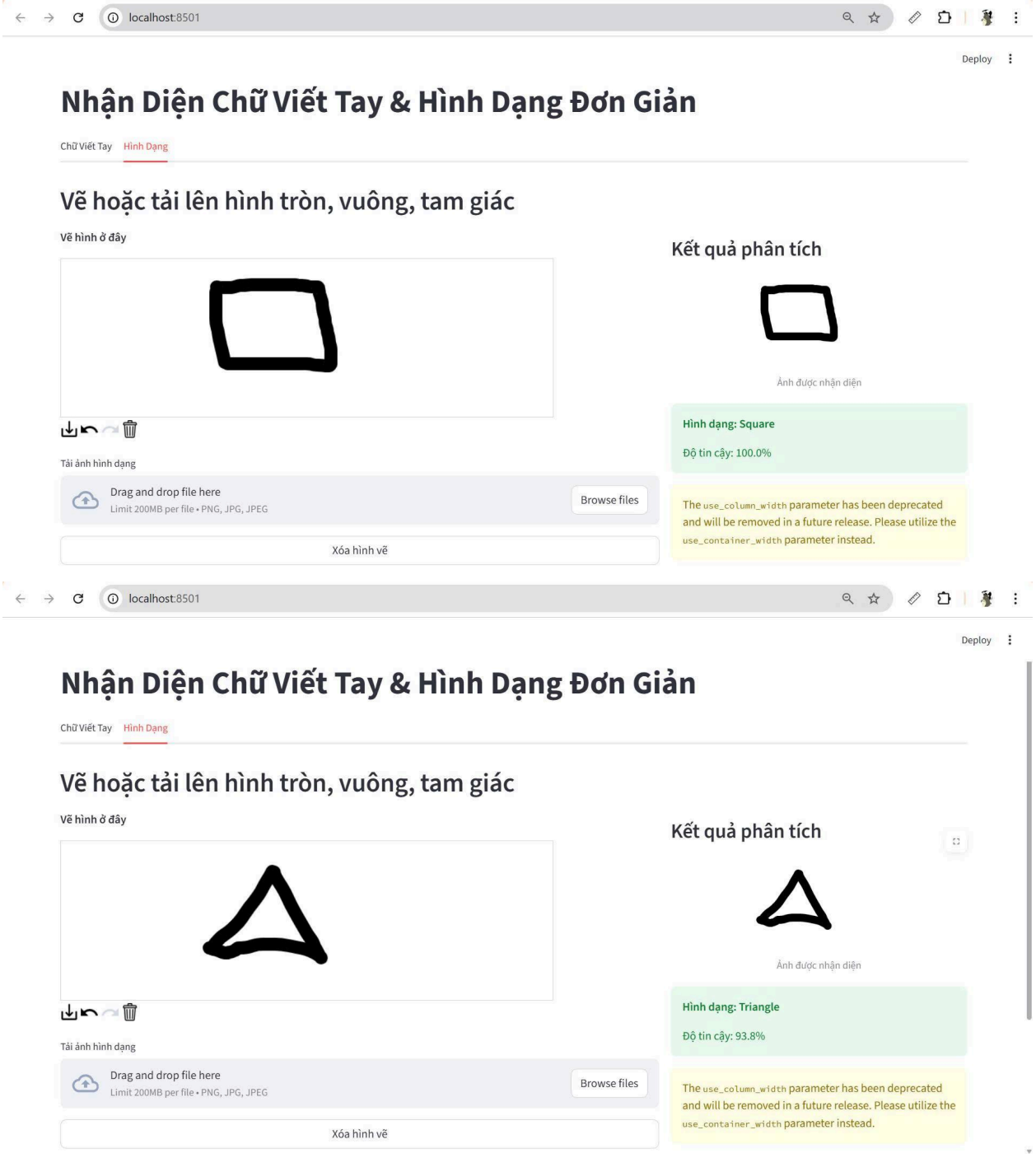

localhost:8501
Deploy
Nhận Diện Chữ Viết Tay & Hình Dạng Đơn Giản
Chữ Viết Tay
Hình Dạng
Viết chữ bằng tay hoặc tải ảnh lên
Viết chữ ở đây
Hoặc tải ảnh chữ viết tay lên
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
Xóa vùng vẽ chữ
Kết quả nhận diện
Kết quả: L
Ảnh đầu vào
The use_column_width parameter has been deprecated and will be removed in a future release. Please utilize the use_container_width parameter instead.

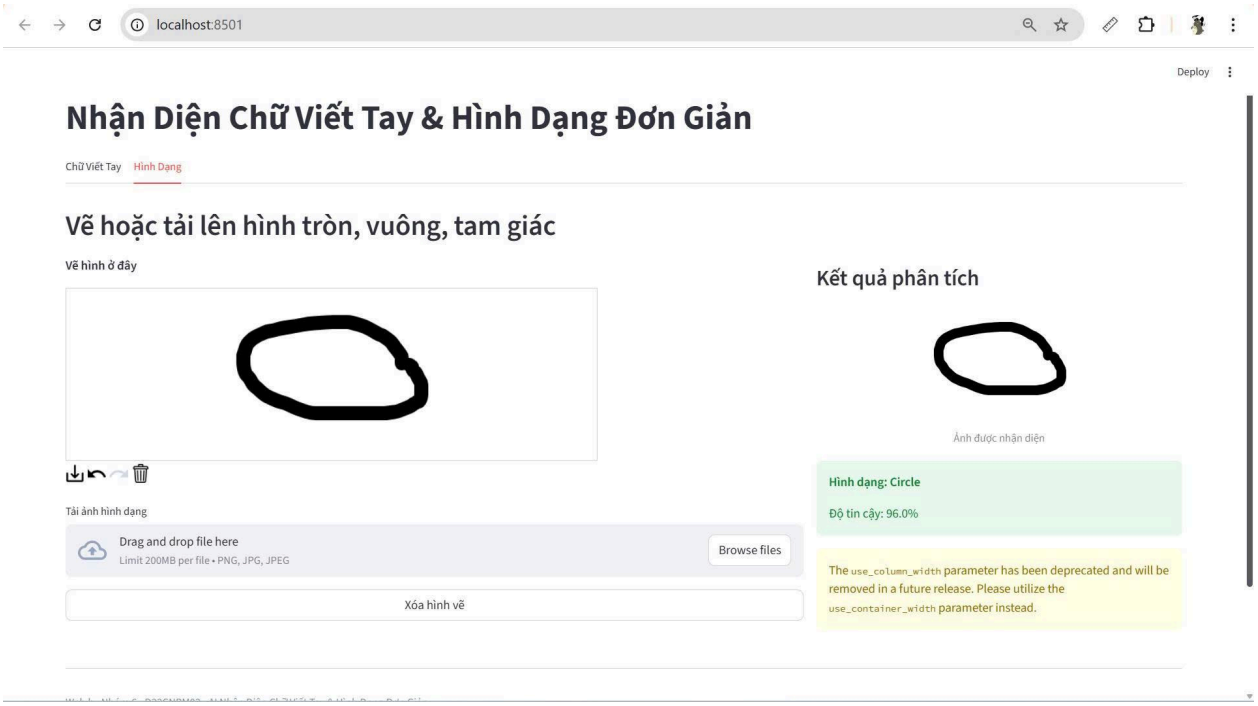

localhost:8501
Deploy
Chữ Viết Tay
Hình Dạng
Viết chữ bằng tay hoặc tải ảnh lên
Viết chữ ở đây
Hoặc tải ảnh chữ viết tay lên
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
Untitled.png 1.5KB
Xóa vùng vẽ chữ
Kết quả nhận diện
Kết quả: N

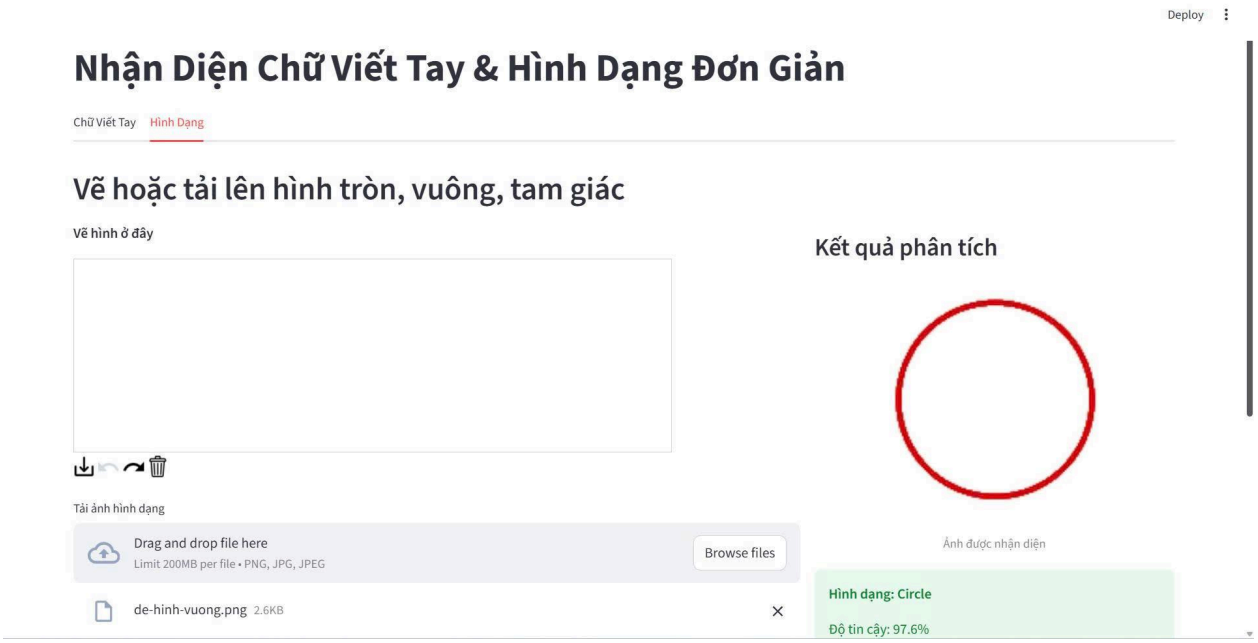

localhost:8501
Deploy
Chữ Viết Tay
Hình Dạng
Viết chữ bằng tay hoặc tải ảnh lên
Viết chữ ở đây
Hoặc tải ảnh chữ viết tay lên
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
Untitled.png 4.5KB
Xóa vùng vẽ chữ
Kết quả nhận diện
Kết quả: L
Ảnh đầu vào
The use_column_width parameter has been deprecated and will be

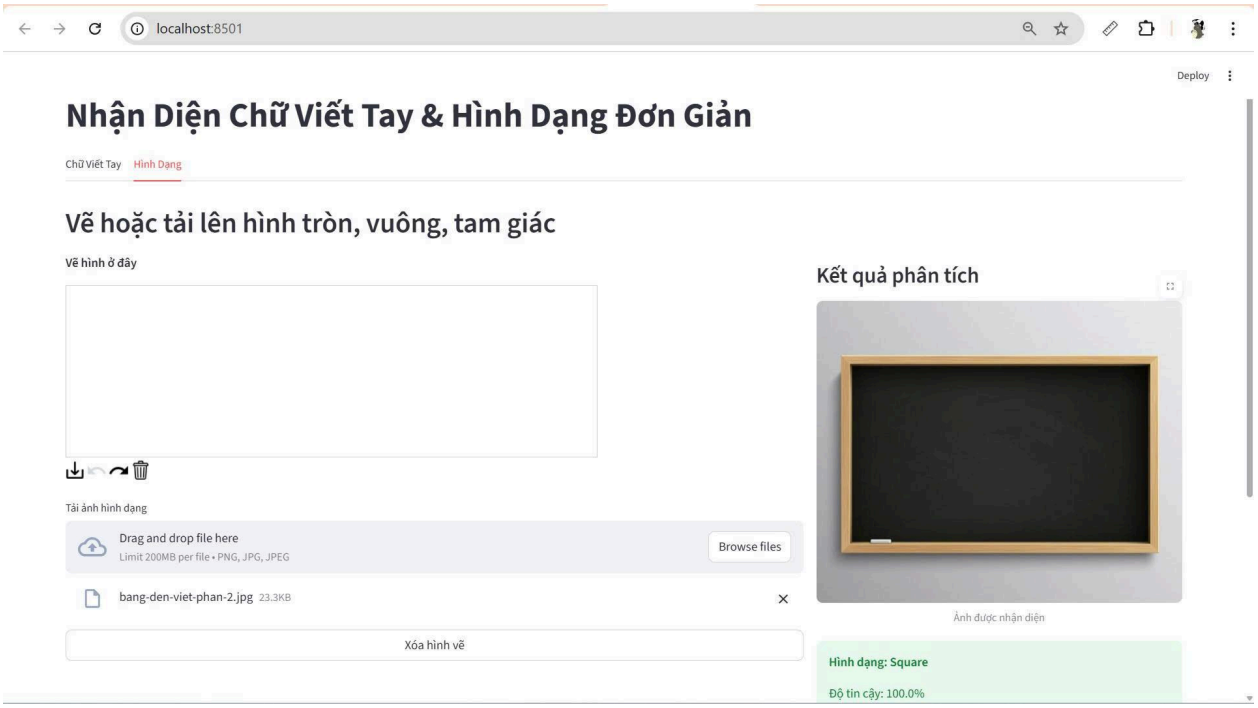

localhost:8501
Deploy
Chữ Viết Tay
Hình Dạng
Viết chữ bằng tay hoặc tải ảnh lên
Viết chữ ở đây
Hoặc tải ảnh chữ viết tay lên
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
Untitled.png 5.9KB
Xóa vùng vẽ chữ
Kết quả nhận diện
Kết quả: V
Ảnh đầu vào
The use_column_width parameter has been deprecated and will be

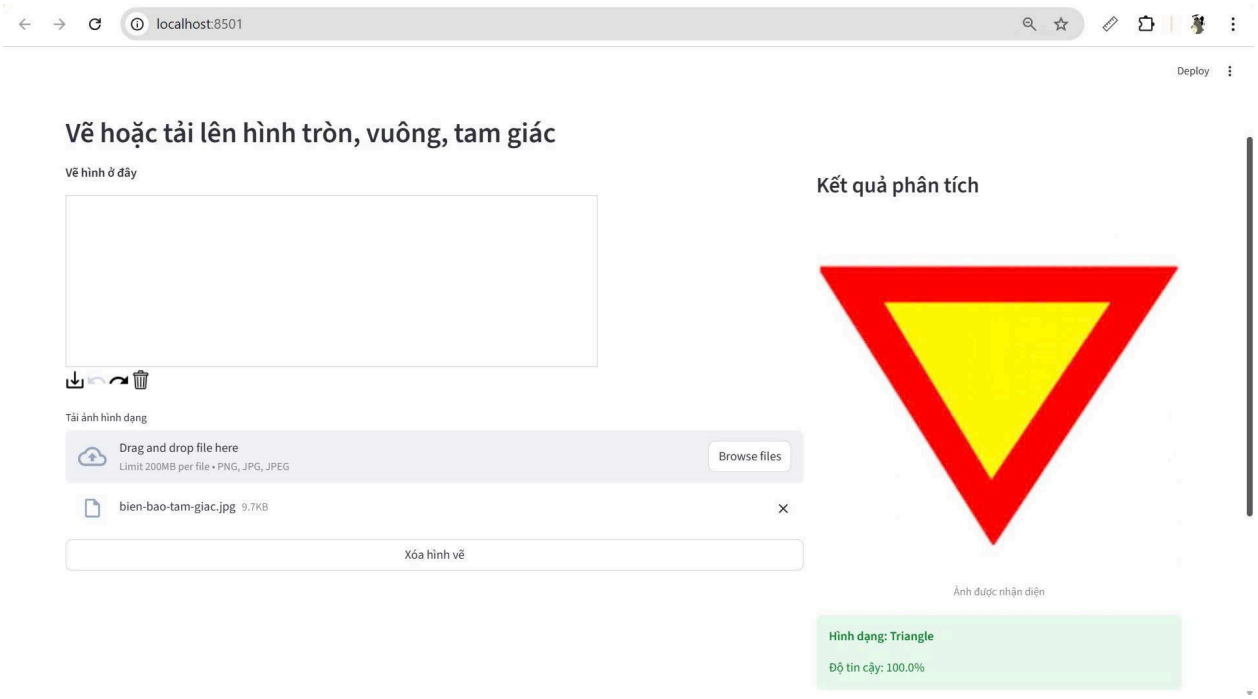
2. Nhận diện hình dạng đơn giản




localhost:8501
Deploy
Nhận Diện Chữ Viết Tay & Hình Dạng Đơn Giản
Chữ Viết Tay
Hình Dạng
Vẽ hoặc tải lên hình tròn, vuông, tam giác
Vẽ hình ở đây
Kết quả phân tích
Ảnh được nhận diện
Hình dạng: Circle
Độ tin cậy: 96.0%
The use_column_width parameter has been deprecated and will be removed in a future release. Please utilize the use_container_width parameter instead.
Tải ảnh hình dạng
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
Xóa hình vẽ


Deploy
Nhận Diện Chữ Viết Tay & Hình Dạng Đơn Giản
Chữ Viết Tay
Hình Dạng
Vẽ hoặc tải lên hình tròn, vuông, tam giác
Vẽ hình ở đây
Kết quả phân tích
Ảnh được nhận diện
Hình dạng: Circle
Độ tin cậy: 97.6%
Tải ảnh hình dạng
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
de-hinh-vuong.png 2.6KB


localhost:8501
Deploy
Nhận Diện Chữ Viết Tay & Hình Dạng Đơn Giản
Chữ Viết Tay
Hình Dạng
Vẽ hoặc tải lên hình tròn, vuông, tam giác
Vẽ hình ở đây
Tải ảnh hình dạng
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
bang-den-viet-phan-2.jpg 23.3KB
Xóa hình vẽ
Kết quả phân tích
Ảnh được nhận diện
Hình dạng: Square
Độ tin cậy: 100.0%


localhost:8501
Deploy
Vẽ hoặc tải lên hình tròn, vuông, tam giác
Vẽ hình ở đây
Tải ảnh hình dạng
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
bien-bao-tam-giac.jpg 9.7KB
Xóa hình vẽ
Kết quả phân tích
Ảnh được nhận diện
Hình dạng: Triangle
Độ tin cậy: 100.0%

TÀI LIỆU THAM KHẢO

[1] CRNN và CTC

[2] Hàm kích hoat

[3] LSTM và BiLSTM

[4] CNN