**CƠ SỞ LÝ THUYẾT VỀ OCR**

**1. GIỚI THIỆU VỀ OCR**

Optical Character Recognition (OCR) hay Nhận dạng ký tự quang học là công nghệ cho phép chuyển đổi các tài liệu dạng hình ảnh như tài liệu được scan, ảnh chụp văn bản, PDF dạng ảnh thành dữ liệu văn bản có thể chỉnh sửa và tìm kiếm được. Công nghệ này đóng vai trò quan trọng trong việc số hóa dữ liệu, tự động hóa quá trình nhập liệu và xử lý tài liệu.

Lịch sử phát triển của OCR đã trải qua nhiều giai đoạn từ những hệ thống cơ học đơn giản đến các hệ thống hiện đại dựa trên trí tuệ nhân tạo và deep learning. Các kỹ thuật OCR hiện đại có thể nhận dạng nhiều ngôn ngữ khác nhau, các phông chữ đa dạng và thậm chí cả chữ viết tay với độ chính xác cao.

Tuy hướng tới cùng một mục tiêu, nhưng OCR có thể được phân loại thành nhiều dạng khác nhau:

* **OCR truyền thống**: Sử dụng các phương pháp xử lý ảnh và mẫu so khớp cơ bản
* **ICR (Intelligent Character Recognition)**: Chuyên dụng cho nhận dạng chữ viết tay
* **OMR (Optical Mark Recognition)**: Nhận dạng dấu tick, chấm tròn trên các biểu mẫu
* **OCR dựa trên AI**: Sử dụng deep learning và neural networks

**2. QUY TRÌNH TỔNG QUAN CỦA OCR**

Một hệ thống OCR hoàn chỉnh thường bao gồm các bước sau:

1. **Thu nhận ảnh**: Quá trình chụp ảnh hoặc scan tài liệu để chuyển đổi thành dạng số.
2. **Tiền xử lý ảnh**: Các phương pháp xử lý ảnh để tăng chất lượng và chuẩn bị cho các bước tiếp theo.
3. **Phân đoạn ký tự**: Xác định và tách các vùng văn bản, dòng và ký tự riêng lẻ.
4. **Nhận dạng ký tự**: Chuyển đổi hình ảnh của từng ký tự thành mã số hoặc văn bản.
5. **Hậu xử lý**: Sửa lỗi, kiểm tra ngữ pháp, từ điển để cải thiện kết quả nhận dạng.

Mỗi bước trong quy trình này đều có những thách thức và kỹ thuật riêng, đòi hỏi sự kết hợp của nhiều lĩnh vực từ xử lý ảnh, học máy đến ngôn ngữ học.

**3. TIỀN XỬ LÝ ẢNH**

Tiền xử lý ảnh là một trong những bước quan trọng nhất trong quy trình OCR, quyết định đến chất lượng của các bước tiếp theo. Mục đích chính của tiền xử lý là loại bỏ các yếu tố nhiễu, tăng cường chất lượng ảnh, và chuẩn hóa dữ liệu đầu vào.

**3.1. Chuyển đổi grayscale**

Chuyển đổi ảnh màu thành ảnh xám là bước đầu tiên trong hầu hết các hệ thống OCR. Công thức phổ biến để chuyển đổi ảnh RGB sang grayscale là:

Grayscale = 0.299\*R + 0.587\*G + 0.114\*B

Trong đó R, G, B lần lượt là giá trị kênh màu đỏ, xanh lá, và xanh dương. Việc chuyển sang grayscale giúp:

* Giảm độ phức tạp tính toán
* Loại bỏ thông tin màu sắc không cần thiết cho nhận dạng ký tự
* Chuẩn bị cho các thuật toán phân ngưỡng

**3.2. Khử nhiễu**

Nhiễu trong ảnh có thể đến từ quá trình scan, chụp ảnh hoặc nguồn gốc của tài liệu. Các phương pháp khử nhiễu phổ biến bao gồm:

* **Lọc trung vị (Median filtering)**: Hiệu quả trong việc loại bỏ nhiễu muối tiêu (salt-and-pepper noise)
* **Lọc Gaussian**: Làm mờ và giảm nhiễu Gaussian
* **Lọc song phương (Bilateral filtering)**: Giữ lại các cạnh trong khi khử nhiễu
* **Biến đổi sóng con (Wavelet transformation)**: Phân tích ảnh ở nhiều tỷ lệ để phát hiện và loại bỏ nhiễu

**3.3. Điều chỉnh độ tương phản**

Việc tăng cường độ tương phản giúp phân biệt rõ ràng hơn giữa văn bản và nền. Các kỹ thuật phổ biến bao gồm:

* **Cân bằng histogram (Histogram equalization)**: Phân phối lại các mức xám để tăng độ tương phản
* **Kéo dãn tương phản (Contrast stretching)**: Mở rộng phạm vi giá trị pixel
* **Điều chỉnh gamma**: Điều chỉnh độ sáng phi tuyến tính

**3.4. Phân ngưỡng (Thresholding)**

Phân ngưỡng là quá trình chuyển đổi ảnh xám thành ảnh nhị phân (chỉ có đen và trắng), giúp tách biệt văn bản khỏi nền. Các phương pháp phân ngưỡng bao gồm:

* **Phân ngưỡng toàn cục (Global thresholding)**: Sử dụng một ngưỡng cho toàn bộ ảnh
  + Phương pháp Otsu: Tự động tìm ngưỡng tối ưu dựa trên phân phối histogram
  + Ngưỡng cố định: Sử dụng một giá trị ngưỡng định sẵn
* **Phân ngưỡng cục bộ (Local/Adaptive thresholding)**: Sử dụng các ngưỡng khác nhau cho các vùng khác nhau của ảnh
  + Phương pháp Niblack: Tính ngưỡng dựa trên trung bình và phương sai cục bộ
  + Phương pháp Sauvola: Cải tiến của Niblack, thích ứng tốt hơn với độ tương phản thay đổi

**3.5. Hiệu chỉnh độ nghiêng**

Các tài liệu được scan có thể bị nghiêng, gây khó khăn cho quá trình phân đoạn và nhận dạng. Các kỹ thuật phát hiện và sửa độ nghiêng bao gồm:

* **Biến đổi Hough**: Phát hiện các đường thẳng để xác định góc nghiêng
* **Phân tích phân phối chiều cao hàng**: Xác định góc nghiêng dựa trên khoảng cách giữa các dòng văn bản
* **Moment of Inertia**: Tính toán moment quán tính để xác định hướng chính của văn bản

**3.6. Resize và chuẩn hóa kích thước**

Việc điều chỉnh kích thước ảnh giúp:

* Tối ưu hóa thời gian xử lý
* Chuẩn hóa đầu vào cho các mô hình nhận dạng
* Giữ tỷ lệ phù hợp để không làm biến dạng ký tự

Các phương pháp nội suy như song tuyến (bilinear), láng giềng gần nhất (nearest neighbor), hoặc bicubic được sử dụng tùy thuộc vào yêu cầu về tốc độ và chất lượng.

**4. PHÂN ĐOẠN KÝ TỰ**

Phân đoạn là quá trình chia nhỏ ảnh đầu vào thành các thành phần có ý nghĩa: vùng văn bản, đoạn văn, dòng, từ và ký tự. Đây là bước quan trọng và cũng là bước khó khăn trong OCR, đặc biệt đối với tài liệu có bố cục phức tạp.

**4.1. Phân đoạn vùng văn bản (Text Region Segmentation)**

Phương pháp phân đoạn vùng văn bản giúp tách biệt các vùng văn bản khỏi hình ảnh, biểu đồ và các thành phần phi văn bản khác:

* **Phương pháp dựa trên RLSA (Run Length Smoothing Algorithm)**: Kết nối các pixel gần nhau để tạo thành các khối
* **Phân tích thành phần kết nối (Connected Component Analysis)**: Gom nhóm các pixel kết nối thành các đối tượng
* **Phương pháp dựa trên histogram chiều ngang và dọc**: Phát hiện vùng văn bản dựa trên sự tập trung của pixel

**4.2. Phân đoạn dòng (Line Segmentation)**

Sau khi xác định vùng văn bản, bước tiếp theo là tách văn bản thành các dòng riêng biệt:

* **Phương pháp dựa trên histogram chiều ngang**: Tìm các "thung lũng" trong histogram biểu thị khoảng trắng giữa các dòng
* **Phương pháp Hough Transform**: Phát hiện các đường thẳng đại diện cho dòng văn bản
* **Phương pháp dựa trên clustering**: Nhóm các thành phần kết nối thành các dòng

Một thuật toán phân đoạn dòng cơ bản có thể được mô tả như sau:

1. Tính histogram chiều ngang (đếm số pixel đen theo mỗi hàng)
2. Áp dụng làm mịn (smoothing) để giảm nhiễu trong histogram
3. Tìm các "thung lũng" (giá trị thấp) trong histogram, đại diện cho khoảng trắng giữa các dòng
4. Cắt ảnh theo các vị trí được xác định

**4.3. Phân đoạn từ (Word Segmentation)**

Sau khi tách dòng, mỗi dòng được phân tích để tách thành các từ:

* **Phương pháp dựa trên khoảng cách**: Xác định khoảng trắng giữa các từ
* **Histogram chiều dọc**: Tìm các khoảng trống dọc trong mỗi dòng
* **Phân tích khoảng cách giữa các thành phần kết nối**: Phân loại khoảng cách thành khoảng cách trong từ và giữa các từ

**4.4. Phân đoạn ký tự (Character Segmentation)**

Đây là bước cuối cùng trong quá trình phân đoạn, tách mỗi từ thành các ký tự riêng lẻ:

* **Phân tích thành phần kết nối**: Xác định các đối tượng riêng biệt trong từ
* **Phương pháp dựa trên đường viền (Contour-based methods)**: Tìm đường viền của mỗi ký tự
* **Phương pháp cắt dọc (Vertical Projection)**: Tìm các vị trí có thể cắt dựa trên histogram chiều dọc

Các thách thức chính trong phân đoạn ký tự:

* **Ký tự liền nhau**: Các ký tự chạm vào nhau, khó tách biệt
* **Ký tự bị đứt đoạn**: Một ký tự bị tách thành nhiều phần
* **Nhiễu và méo mó**: Làm thay đổi hình dạng và cấu trúc của ký tự

Đối với các ngôn ngữ phức tạp như tiếng Việt, với các dấu thanh và dấu phụ, việc phân đoạn càng trở nên khó khăn hơn và thường đòi hỏi các thuật toán chuyên biệt.

**5. NHẬN DẠNG KÝ TỰ**

Sau khi phân đoạn, mỗi ký tự được nhận dạng để chuyển từ hình ảnh sang mã số hoặc văn bản. Có nhiều phương pháp nhận dạng ký tự, từ các phương pháp truyền thống đến các kỹ thuật học sâu hiện đại.

**5.1. Trích xuất đặc trưng (Feature Extraction)**

Trước khi nhận dạng, các đặc trưng quan trọng của ký tự cần được trích xuất:

* **Đặc trưng hình dạng (Shape features)**:
  + Moment bất biến (Invariant moments)
  + Tỷ lệ khung hình (Aspect ratio)
  + Đặc trưng Zoning: Chia ký tự thành các vùng nhỏ và tính đặc trưng cho mỗi vùng
* **Đặc trưng cấu trúc (Structural features)**:
  + Số lượng và vị trí các nét thẳng, cong
  + Điểm giao nhau, điểm cuối
  + Vòng tròn và lỗ (loops and holes)
* **Đặc trưng thống kê (Statistical features)**:
  + Histogram của hướng gradient (HOG)
  + Mô tả Fourier
  + Biểu đồ chiếu (Projection histograms)

**5.2. Phương pháp so khớp mẫu (Template Matching)**

Đây là phương pháp đơn giản nhất, so sánh trực tiếp hình ảnh ký tự với các mẫu đã biết:

* **So khớp trực tiếp**: So sánh pixel-by-pixel
* **So khớp tương quan (Correlation-based matching)**: Tính độ tương quan giữa hai hình ảnh
* **Biến đổi khoảng cách (Distance Transform matching)**: So sánh dựa trên bản đồ khoảng cách

Phương pháp này đơn giản nhưng kém linh hoạt, dễ bị ảnh hưởng bởi sự thay đổi về phông chữ, kích thước, và độ nghiêng.

**5.3. Phương pháp cấu trúc và cú pháp (Structural and Syntactic Approaches)**

Phương pháp này phân tích cấu trúc của ký tự:

* **Biểu diễn cây (Tree representation)**: Biểu diễn ký tự dưới dạng cây với các thành phần cơ bản
* **Biểu diễn đồ thị (Graph matching)**: Biểu diễn ký tự dưới dạng đồ thị và so khớp đồ thị
* **Phân tích nét (Stroke analysis)**: Phân tích các nét cơ bản và mối quan hệ của chúng

**5.4. Phương pháp Machine Learning truyền thống**

Các thuật toán học máy được sử dụng rộng rãi trong OCR:

* **k-Nearest Neighbors (k-NN)**: So sánh với k mẫu gần nhất trong không gian đặc trưng
* **Support Vector Machines (SVM)**: Tìm siêu phẳng tối ưu phân tách các lớp ký tự
* **Random Forests**: Kết hợp nhiều cây quyết định
* **Hidden Markov Models (HMM)**: Mô hình xác suất cho chuỗi quan sát

**5.5. Deep Learning trong OCR**

Các phương pháp học sâu đã mang lại bước đột phá lớn trong OCR:

* **Convolutional Neural Networks (CNN)**: Hiệu quả trong việc trích xuất đặc trưng từ hình ảnh
  + LeNet, AlexNet, VGG, ResNet: Các kiến trúc CNN phổ biến được điều chỉnh cho OCR
* **Recurrent Neural Networks (RNN)**: Xử lý chuỗi thời gian, tốt cho nhận dạng chuỗi ký tự
  + LSTM (Long Short-Term Memory): Xử lý phụ thuộc dài hạn
  + Bidirectional RNNs: Xử lý thông tin từ cả hai hướng
* **Connectionist Temporal Classification (CTC)**: Kỹ thuật quan trọng cho phép nhận dạng chuỗi mà không cần phân đoạn ký tự rõ ràng
* **Seq2Seq và Attention**: Mô hình encoder-decoder với cơ chế attention, hiệu quả cho OCR end-to-end
* **Transformer**: Kiến trúc dựa trên cơ chế self-attention, hiệu quả cho cả nhận dạng và hiệu chỉnh

**5.6. Một số kiến trúc OCR phổ biến**

* **Tesseract**: Mã nguồn mở, ban đầu dựa trên phương pháp truyền thống, phiên bản mới nhất (v4+) sử dụng LSTM
* **CRNN (Convolutional Recurrent Neural Network)**: Kết hợp CNN và RNN, phổ biến trong OCR hiện đại
* **EAST (Efficient and Accurate Scene Text Detector)**: Phát hiện văn bản trong cảnh phức tạp
* **FOTS (Fast Oriented Text Spotting)**: Kết hợp phát hiện và nhận dạng trong một mô hình end-to-end

**Mạng LSTM và ứng dụng trong OCR**

**1. Giới thiệu**

Nhận dạng ký tự quang học (Optical Character Recognition - OCR) là quá trình chuyển đổi hình ảnh văn bản thành văn bản kỹ thuật số có thể chỉnh sửa. Công nghệ này đã phát triển mạnh mẽ trong những năm gần đây nhờ vào sự phát triển của các mô hình học sâu, đặc biệt là mạng nơ-ron hồi quy dài-ngắn hạn (Long Short-Term Memory - LSTM).

Báo cáo này tập trung vào mạng LSTM và ứng dụng của nó trong lĩnh vực OCR, đặc biệt là trong công cụ Tesseract - một trong những hệ thống OCR mã nguồn mở phổ biến nhất hiện nay. Chúng tôi sẽ phân tích kiến trúc LSTM, ưu điểm của nó so với các mạng nơ-ron hồi quy truyền thống (RNN), và cách LSTM được tích hợp vào quy trình nhận dạng văn bản từ hình ảnh.

**2. Mạng LSTM: Lý thuyết và Kiến trúc**

**Vấn đề với RNN truyền thống**

Mạng nơ-ron hồi quy (Recurrent Neural Networks - RNN) là một loại mạng nơ-ron được thiết kế đặc biệt để xử lý dữ liệu tuần tự như văn bản, âm thanh, hoặc chuỗi thời gian. Ý tưởng cơ bản của RNN là sử dụng thông tin từ các bước trước đó để tác động đến quyết định tại bước hiện tại.

Tuy nhiên, RNN truyền thống gặp phải vấn đề nghiêm trọng khi xử lý chuỗi dài: vấn đề mất mát gradient (vanishing gradient problem). Khi thông tin phải truyền qua nhiều bước thời gian, gradient có xu hướng trở nên rất nhỏ, khiến mạng không thể học được các phụ thuộc dài hạn. Trong OCR, điều này có nghĩa là RNN khó có thể liên kết các ký tự ở vị trí xa nhau trong một từ dài hoặc câu.

**Kiến trúc LSTM**

LSTM (Long Short-Term Memory) được giới thiệu bởi Hochreiter và Schmidhuber vào năm 1997 như một giải pháp cho vấn đề mất mát gradient. LSTM là một loại đặc biệt của RNN được thiết kế để nhớ thông tin trong thời gian dài.

Điểm khác biệt cơ bản của LSTM so với RNN thông thường là việc sử dụng một cơ chế phức tạp hơn để cập nhật và kiểm soát luồng thông tin. Trong khi RNN chỉ có một trạng thái ẩn (hidden state), LSTM có cả trạng thái ẩn và một trạng thái tế bào (cell state) riêng biệt.

**Các thành phần chính của LSTM**

Một khối LSTM tiêu chuẩn bao gồm các thành phần sau:

1. **Cell State (C\_t)**: Là "băng chuyền" của LSTM, cho phép thông tin chảy qua mạng với ít thay đổi. Cell state hoạt động như bộ nhớ dài hạn của mạng.
2. **Hidden State (h\_t)**: Đại diện cho trạng thái ẩn tại thời điểm t, tương tự như trong RNN truyền thống.
3. **Forget Gate (f\_t)**: Quyết định thông tin nào nên được loại bỏ khỏi cell state. Nó xem xét đầu vào hiện tại và trạng thái ẩn trước đó để tạo ra một vector các giá trị từ 0 đến 1, với 1 có nghĩa là "giữ hoàn toàn" và 0 là "quên hoàn toàn".
4. **Input Gate (i\_t)**: Quyết định thông tin mới nào sẽ được lưu trữ trong cell state. Nó bao gồm hai phần:
   * Một lớp sigmoid quyết định giá trị nào cần được cập nhật.
   * Một lớp tanh tạo ra một vector các giá trị mới có thể được thêm vào cell state.
5. **Output Gate (o\_t)**: Quyết định phần nào của cell state sẽ được đưa ra làm đầu ra. Đầu ra dựa trên cell state, nhưng là một phiên bản được lọc.

Mỗi LSTM cell có thể được biểu diễn bằng các phương trình sau:

* Forget gate: f\_t = σ(W\_f · [h\_(t-1), x\_t] + b\_f)
* Input gate: i\_t = σ(W\_i · [h\_(t-1), x\_t] + b\_i)
* Cell update: C̃\_t = tanh(W\_C · [h\_(t-1), x\_t] + b\_C)
* Cell state update: C\_t = f\_t \* C\_(t-1) + i\_t \* C̃\_t
* Output gate: o\_t = σ(W\_o · [h\_(t-1), x\_t] + b\_o)
* Hidden state: h\_t = o\_t \* tanh(C\_t)

Trong đó:

* σ là hàm sigmoid
* tanh là hàm tanh
  + là phép nhân Hadamard (element-wise)
* [h\_(t-1), x\_t] là phép nối vector
* W và b là các tham số có thể học được của mạng

**Ưu điểm của LSTM so với RNN**

LSTM có một số ưu điểm quan trọng so với RNN truyền thống, đặc biệt trong bối cảnh OCR:

1. **Khả năng học các phụ thuộc dài hạn**: Cơ chế gating trong LSTM cho phép nó nhớ thông tin trong thời gian dài, giúp nó nhận dạng các mẫu phụ thuộc xa trong chuỗi ký tự.
2. **Giải quyết vấn đề mất mát gradient**: Cấu trúc của LSTM giúp gradient có thể chảy qua mạng mà không bị suy giảm nhanh chóng, cho phép học hiệu quả hơn.
3. **Khả năng chọn lọc thông tin**: LSTM có khả năng "quên" thông tin không liên quan và chỉ giữ lại thông tin quan trọng, giúp xử lý hiệu quả các chuỗi dài.
4. **Khả năng xử lý nhiễu**: Trong OCR, hình ảnh văn bản thường chứa nhiễu. LSTM có khả năng lọc nhiễu tốt hơn nhờ cơ chế gating.
5. **Hiệu suất cao với dữ liệu ngữ cảnh**: LSTM có thể học và sử dụng thông tin ngữ cảnh (như các ký tự xung quanh) để cải thiện độ chính xác của nhận dạng.

**3.LSTM trong Tesseract OCR**

**Tesseract OCR: Tổng quan**

Tesseract là một công cụ OCR mã nguồn mở được phát triển ban đầu bởi HP và sau đó được Google tiếp quản. Phiên bản Tesseract 4.0 (phát hành năm 2018) đã đánh dấu một bước ngoặt quan trọng với việc tích hợp mạng LSTM vào kiến trúc cốt lõi của nó, thay thế cho phương pháp dựa trên đặc trưng trong các phiên bản trước.

Tesseract được thiết kế để nhận dạng văn bản từ hình ảnh, với các bước chính bao gồm:

1. Phân tích bố cục trang
2. Phân đoạn dòng và từ
3. Nhận dạng ký tự
4. Hậu xử lý và xuất văn bản

Với việc tích hợp LSTM, hiệu suất của Tesseract đã được cải thiện đáng kể, đặc biệt đối với các ngôn ngữ phức tạp và các tình huống có nhiều nhiễu.

**Vai trò của LSTM trong Tesseract**

Trong Tesseract 4.0 trở lên, LSTM đóng vai trò trung tâm trong quá trình nhận dạng ký tự. Thay vì sử dụng các đặc trưng được định nghĩa thủ công, Tesseract sử dụng mạng LSTM hai chiều (bidirectional LSTM) để "đọc" hình ảnh văn bản theo cả hai hướng (từ trái sang phải và từ phải sang trái), sau đó kết hợp kết quả để đưa ra dự đoán về ký tự.

Quy trình xử lý của Tesseract với LSTM bao gồm:

1. **Trích xuất đặc trưng**: Chuyển đổi hình ảnh đầu vào thành chuỗi các vector đặc trưng, thường dựa trên các cửa sổ trượt dọc theo dòng văn bản.
2. **Mạng LSTM hai chiều**: Vector đặc trưng được đưa vào mạng LSTM hai chiều, học các phụ thuộc theo cả hai hướng của văn bản.
3. **Lớp Connectionist Temporal Classification (CTC)**: Đầu ra của LSTM được xử lý thông qua lớp CTC để ánh xạ chuỗi dự đoán sang chuỗi ký tự thực tế, giải quyết vấn đề căn chỉnh giữa đầu vào và đầu ra.
4. **Hậu xử lý**: Áp dụng các kỹ thuật hậu xử lý như kiểm tra từ điển để cải thiện kết quả.

**Xử lý chuỗi ký tự liên tục**

Một trong những thách thức lớn trong OCR là nhận dạng văn bản có ký tự liên tục hoặc chữ viết tay, khi ranh giới giữa các ký tự không rõ ràng. LSTM trong Tesseract giải quyết vấn đề này một cách hiệu quả nhờ các đặc điểm sau:

1. **Phân tích theo chuỗi thời gian**: LSTM xử lý hình ảnh như một chuỗi thời gian, trong đó mỗi cột của hình ảnh được coi như một bước thời gian. Điều này cho phép nó nhận dạng các mẫu trong chuỗi thay vì các ký tự riêng lẻ.
2. **CTC Decoding**: Tesseract sử dụng giải thuật CTC để giải quyết vấn đề căn chỉnh giữa đầu vào (chuỗi các vector đặc trưng) và đầu ra (chuỗi ký tự). CTC cho phép mạng học cách ánh xạ từ đầu vào dài (nhiều frame) sang đầu ra ngắn hơn (ít ký tự hơn) mà không cần thông tin rõ ràng về ranh giới ký tự.
3. **Xử lý ngữ cảnh**: LSTM hai chiều có thể xem xét ngữ cảnh từ cả trước và sau vị trí hiện tại, giúp nó hiểu được các mẫu phức tạp trong chữ viết liên tục.
4. **Mô hình ngôn ngữ**: Tesseract sử dụng mô hình ngôn ngữ kết hợp với LSTM để cải thiện độ chính xác bằng cách cân nhắc xác suất của các chuỗi từ trong ngôn ngữ đích.

Nhờ những khả năng này, LSTM trong Tesseract có thể xử lý hiệu quả các tình huống khó như:

* Chữ viết tay nối liền
* Font chữ trang trí với các ký tự chồng chéo
* Văn bản có khoảng cách không đều
* Văn bản bị méo hoặc biến dạng

**4.Mô hình học sâu trong nhận dạng văn bản**

**Input (Ảnh)**

Quá trình OCR bắt đầu với hình ảnh chứa văn bản. Xử lý đầu vào là một bước quan trọng ảnh hưởng đến hiệu suất của hệ thống nhận dạng:

1. **Tiền xử lý hình ảnh**:
   * Chuyển đổi sang ảnh xám (grayscale) hoặc nhị phân (binary)
   * Loại bỏ nhiễu bằng các kỹ thuật lọc
   * Điều chỉnh độ tương phản
   * Xoay và căn chỉnh văn bản
   * Loại bỏ các yếu tố không liên quan
2. **Phân đoạn**:
   * Phân đoạn trang thành các khối văn bản, bảng, hình ảnh
   * Phân đoạn khối văn bản thành dòng
   * Phân đoạn dòng thành từ (nếu cần)
3. **Trích xuất đặc trưng**:
   * Trong các hệ thống OCR truyền thống: Trích xuất đặc trưng thủ công như histogram của gradient, đặc trưng Haar, v.v.
   * Trong hệ thống dựa trên học sâu như Tesseract với LSTM: Hình ảnh được chuyển thành chuỗi các vector đặc trưng, thường bằng cách quét hình ảnh theo từng cột hoặc cửa sổ trượt.

Trong Tesseract, quá trình xử lý đầu vào bao gồm:

* Chuẩn hóa kích thước
* Chuẩn hóa độ tương phản
* Trích xuất đặc trưng dựa trên cửa sổ trượt
* Chuẩn hóa đặc trưng

**Hidden Layers**

Kiến trúc mạng học sâu trong OCR hiện đại thường bao gồm nhiều lớp ẩn phức tạp. Trong trường hợp của Tesseract với LSTM:

1. **Lớp tích chập (Convolutional Layer)**:
   * Trước khi đưa vào LSTM, đầu vào thường được xử lý qua các lớp tích chập (CNN) để trích xuất đặc trưng không gian cục bộ.
   * Các lớp CNN có thể phát hiện các đặc điểm như cạnh, góc, và các mẫu cục bộ khác trong hình ảnh văn bản.
2. **Lớp LSTM**:
   * Mạng LSTM hai chiều là thành phần cốt lõi, xử lý chuỗi đặc trưng theo cả hai hướng.
   * Tesseract sử dụng nhiều lớp LSTM chồng lên nhau để tăng khả năng học các biểu diễn phức tạp.
   * Mỗi lớp LSTM có thể có hàng trăm đơn vị (nodes), tùy thuộc vào cấu hình cụ thể.
3. **Lớp kết nối đầy đủ (Fully Connected Layer)**:
   * Đầu ra của LSTM được đưa qua một hoặc nhiều lớp kết nối đầy đủ để ánh xạ sang không gian đặc trưng phù hợp với số lượng lớp đầu ra (số ký tự trong từ điển).
4. **Lớp Dropout và Batch Normalization**:
   * Các lớp này được sử dụng để ngăn chặn overfitting và cải thiện tốc độ và độ ổn định của quá trình huấn luyện.
5. **Lớp CTC (Connectionist Temporal Classification)**:
   * Không phải là một lớp ẩn theo nghĩa thông thường, nhưng đóng vai trò quan trọng trong việc chuyển đổi đầu ra của LSTM thành chuỗi ký tự.
   * CTC giải quyết vấn đề căn chỉnh giữa đầu vào và đầu ra, cho phép mạng học mà không cần thông tin rõ ràng về ranh giới ký tự.

**Output (Text)**

Quá trình tạo đầu ra văn bản từ mạng LSTM trong OCR bao gồm:

1. **Dự đoán ký tự**:
   * Đầu ra của lớp cuối cùng là một ma trận xác suất, với mỗi cột biểu thị xác suất của mỗi ký tự có thể (bao gồm cả ký tự trống) tại một vị trí cụ thể.
2. **Giải mã CTC**:
   * Thuật toán giải mã CTC chuyển đổi chuỗi xác suất thành chuỗi ký tự thực tế.
   * Có hai chiến lược giải mã chính:
     + Giải mã tham lam (Greedy Decoding): Chọn ký tự có xác suất cao nhất tại mỗi vị trí, sau đó loại bỏ các ký tự trùng lặp liên tiếp và ký tự trống.
     + Giải mã beam search: Tìm kiếm chuỗi có xác suất cao nhất bằng cách xem xét nhiều khả năng cùng lúc.
3. **Hậu xử lý**:
   * Áp dụng kiểm tra từ điển để sửa lỗi chính tả phổ biến.
   * Sử dụng mô hình ngôn ngữ (n-gram models) để cải thiện độ chính xác dựa trên ngữ cảnh.
   * Áp dụng các quy tắc đặc thù cho ngôn ngữ để cải thiện kết quả.
4. **Định dạng đầu ra**:
   * Kết quả cuối cùng có thể được định dạng theo nhiều cách khác nhau (văn bản thuần túy, XML, JSON, PDF có thể tìm kiếm, v.v.) tùy thuộc vào yêu cầu ứng dụng.

Trong Tesseract, đầu ra có thể bao gồm không chỉ văn bản nhận dạng mà còn:

* Thông tin về vị trí của mỗi ký tự, từ, dòng trong hình ảnh gốc
* Độ tin cậy của mỗi nhận dạng
* Các lựa chọn thay thế cho mỗi ký tự với điểm số tương ứng

# Cơ chế huấn luyện trong Tesseract

## Giới thiệu

Tesseract OCR từ phiên bản 4.0 trở đi đã chuyển sang sử dụng mạng nơ-ron LSTM (Long Short-Term Memory) làm động cơ nhận dạng chính, thay thế cho phương pháp dựa trên đặc trưng trong các phiên bản trước. Sự thay đổi này không chỉ cải thiện đáng kể độ chính xác mà còn làm thay đổi hoàn toàn cách thức huấn luyện Tesseract cho các ngôn ngữ hoặc font chữ mới.

Trong báo cáo này, chúng tôi sẽ tập trung vào cơ chế huấn luyện trong Tesseract, bao gồm cấu trúc dữ liệu huấn luyện, các file cấu hình cần thiết, và các lệnh sử dụng trong quá trình huấn luyện. Hiểu rõ những thành phần này là chìa khóa để có thể huấn luyện và tinh chỉnh mô hình Tesseract cho các ứng dụng cụ thể.

## Tổng quan về quá trình huấn luyện

Trước khi đi vào chi tiết, chúng ta cần hiểu tổng thể quy trình huấn luyện Tesseract:

1. **Chuẩn bị dữ liệu huấn luyện**: Tạo hoặc thu thập hình ảnh văn bản (.tif) và file ground truth tương ứng (.gt.txt).
2. **Tiền xử lý dữ liệu**: Chuyển đổi dữ liệu thành các định dạng phù hợp (.box và .lstmf) để huấn luyện.
3. **Chuẩn bị file cấu hình**: Chuẩn bị các file cấu hình như font\_properties, wordlist, và các file trong thư mục langdata.
4. **Huấn luyện mô hình**: Sử dụng các công cụ như text2image, tesseract, và lstmtraining để tiến hành huấn luyện mô hình LSTM.
5. **Đánh giá và tinh chỉnh**: Kiểm tra hiệu suất mô hình và tinh chỉnh nếu cần.
6. **Đóng gói và triển khai**: Tạo file traineddata cuối cùng để sử dụng trong các ứng dụng OCR.

Bây giờ, chúng ta sẽ đi sâu vào từng phần của quy trình này.

## Cấu trúc dữ liệu huấn luyện

### Định dạng file .tif và .gt.txt

Dữ liệu huấn luyện cơ bản cho Tesseract bao gồm hai thành phần chính:

1. **File hình ảnh (.tif)**:
   * Tesseract ưu tiên sử dụng định dạng TIFF vì nó hỗ trợ lưu trữ hình ảnh không nén và nhiều trang
   * Thông thường là hình ảnh một dòng văn bản trên nền trắng
   * Độ phân giải khuyến nghị là 300 DPI
   * Có thể là hình ảnh nhị phân (đen trắng) hoặc hình ảnh xám (grayscale)
2. **File ground truth (.gt.txt)**:
   * File văn bản thuần túy (plain text) chứa nội dung chính xác của văn bản trong hình ảnh tương ứng
   * Tên file thường là [tên\_file\_hình\_ảnh].gt.txt
   * Nội dung phải khớp chính xác với văn bản trong hình ảnh, bao gồm cả khoảng trắng và ký tự đặc biệt

Ví dụ, nếu có file hình ảnh "line\_01.tif" chứa văn bản "Xin chào thế giới", thì file ground truth tương ứng sẽ là "line\_01.gt.txt" với nội dung "Xin chào thế giới".

### Chuyển đổi sang .box và .lstmf

Các file .tif và .gt.txt cần được chuyển đổi thành các định dạng đặc biệt để huấn luyện LSTM trong Tesseract:

1. **File .box**:
   * Chứa thông tin về vị trí của từng ký tự trong hình ảnh
   * Mỗi dòng đại diện cho một ký tự với định dạng: ký\_tự x\_left y\_bottom x\_right y\_top page\_number
   * Được tạo ra bằng cách chạy lệnh tesseract [image\_file] [output\_base] batch.nochop makebox
2. **File .lstmf (LSTM Feature)**:
   * Đây là định dạng nhị phân chứa đặc trưng đã được trích xuất từ hình ảnh, cùng với thông tin ground truth
   * Mỗi file .lstmf tương ứng với một hình ảnh và ground truth của nó
   * Là định dạng đầu vào trực tiếp cho quá trình huấn luyện LSTM

### Cấu trúc của file .lstmf

Mặc dù file .lstmf là định dạng nhị phân và không thể đọc trực tiếp, nhưng hiểu cấu trúc của nó sẽ giúp chúng ta hiểu rõ hơn về quá trình huấn luyện:

1. **Header**: Chứa thông tin về kích thước hình ảnh, số lượng đặc trưng, và các thông số khác.
2. **Feature Vectors**: Chuỗi các vector đặc trưng được trích xuất từ hình ảnh, thường là kết quả của việc quét hình ảnh theo từng cột.
3. **Ground Truth Text**: Văn bản chính xác tương ứng với hình ảnh.
4. **Bounding Box Information**: Thông tin về vị trí của các ký tự trong hình ảnh.

File .lstmf được tạo ra bằng lệnh tesseract [image\_file] [output\_base] lstm.train khi đã có file .gt.txt tương ứng.

## File cấu hình và dữ liệu phụ trợ

### Thư mục langdata

Thư mục langdata chứa các file cấu hình và dữ liệu phụ trợ cho từng ngôn ngữ:

1. **[lang].training\_text**:
   * Chứa văn bản mẫu để tạo dữ liệu huấn luyện tổng hợp bằng text2image
   * Nên bao gồm đa dạng các ký tự, từ, và cụm từ phổ biến trong ngôn ngữ đó
2. **[lang].punc**:
   * Liệt kê các ký tự dấu câu được sử dụng trong ngôn ngữ đó
3. **[lang].numbers**:
   * Liệt kê các ký tự số được sử dụng
4. **[lang].wordlist**:
   * Danh sách các từ phổ biến trong ngôn ngữ đó, dùng để cải thiện khả năng nhận dạng
5. **[lang].unicharset**:
   * Tập hợp tất cả các ký tự Unicode được sử dụng trong ngôn ngữ đó
6. **[lang].unicharambigs**:
   * Định nghĩa các ký tự dễ nhầm lẫn

### File font\_properties

File font\_properties chứa thông tin về các font chữ được sử dụng trong dữ liệu huấn luyện:

[font\_name] [italic] [bold] [fixed] [serif] [fraktur]

Trong đó:

* [font\_name]: Tên của font chữ (phải khớp với tên font được sử dụng trong lệnh text2image)
* [italic], [bold], [fixed], [serif], [fraktur]: Các giá trị 0 hoặc 1 cho biết font có thuộc tính đó hay không

Ví dụ:

Arial 0 0 0 0 0

Times\_New\_Roman 0 0 0 1 0

Courier\_New 0 0 1 0 0

File này rất quan trọng vì nó giúp Tesseract hiểu được các đặc điểm của font chữ, từ đó cải thiện khả năng nhận dạng.

### File wordlist

File wordlist chứa danh sách các từ được phép xuất hiện trong ngôn ngữ đó:

* Một từ trên mỗi dòng
* Được sử dụng để cải thiện độ chính xác của nhận dạng thông qua kiểm tra từ điển
* Nên bao gồm các từ phổ biến và các biến thể của chúng

Ví dụ cho tiếng Việt:

xin

chào

thế

giới

việt

nam

### Các file cấu hình khác

Ngoài các file đã đề cập, còn có một số file cấu hình khác có thể cần thiết:

1. **[lang].config**:
   * Chứa các thông số cấu hình cho quá trình huấn luyện
   * Có thể định nghĩa các tham số như learning rate, batch size, v.v.
2. **[lang].lstm-unicharset**:
   * Phiên bản đặc biệt của unicharset cho LSTM
   * Được tạo ra trong quá trình huấn luyện
3. **[lang].traineddata**:
   * File cuối cùng sau khi huấn luyện
   * Bao gồm mô hình LSTM và các thông tin cấu hình khác

## Các lệnh và công cụ huấn luyện

### text2image: Tạo dữ liệu huấn luyện tổng hợp

Công cụ text2image được sử dụng để tạo dữ liệu huấn luyện tổng hợp từ text:

text2image --text=[text\_file] --outputbase=[output\_base] --font=[font\_name] --fonts\_dir=[fonts\_directory] --fontconfig\_tmpdir=[tmp\_dir]

Các tham số chính:

* --text: File văn bản đầu vào
* --outputbase: Tiền tố cho các file đầu ra
* --font: Tên font chữ sử dụng
* --fonts\_dir: Thư mục chứa font chữ
* --fontconfig\_tmpdir: Thư mục tạm thời cho fontconfig

Ví dụ:

text2image --text=vie.training\_text --outputbase=vie.Arial --font="Arial" --fonts\_dir=/usr/share/fonts --fontconfig\_tmpdir=/tmp

Lệnh này sẽ tạo ra các file .tif và .box tương ứng với văn bản trong file vie.training\_text, sử dụng font Arial.

### tesseract: Sinh file .lstmf

Công cụ tesseract được sử dụng để tạo các file .lstmf từ các file .tif và .gt.txt:

tesseract [image\_file] [output\_base] lstm.train [config\_file]

Các tham số:

* [image\_file]: File hình ảnh đầu vào (.tif)
* [output\_base]: Tiền tố cho file đầu ra
* lstm.train: Chỉ định chế độ tạo file huấn luyện LSTM
* [config\_file]: File cấu hình tùy chọn

Ví dụ:

tesseract vie.Arial.exp0.tif vie.Arial.exp0 lstm.train

Để tạo file .lstmf từ nhiều file hình ảnh, có thể sử dụng script như sau:

for file in \*.tif; do

base=$(basename "$file" .tif)

tesseract "$file" "$base" lstm.train

done

### lstmtraining: Huấn luyện mô hình LSTM

Công cụ lstmtraining được sử dụng để huấn luyện mô hình LSTM:

lstmtraining --traineddata=[lang].traineddata --train\_listfile=[list\_file] --eval\_listfile=[eval\_list] --continue\_from=[checkpoint] --model\_output=[output\_model] --max\_iterations=[max\_iter]

Các tham số chính:

* --traineddata: File traineddata làm cơ sở
* --train\_listfile: File chứa danh sách các file .lstmf dùng để huấn luyện
* --eval\_listfile: File chứa danh sách các file .lstmf dùng để đánh giá
* --continue\_from: Checkpoint để tiếp tục huấn luyện (nếu có)
* --model\_output: Tiền tố cho file mô hình đầu ra
* --max\_iterations: Số vòng lặp tối đa

Ví dụ:

lstmtraining --traineddata=vie.traineddata --train\_listfile=train\_files.txt --eval\_listfile=eval\_files.txt --continue\_from="" --model\_output=vie/vie --max\_iterations=10000

### Quy trình huấn luyện đầy đủ

Dưới đây là quy trình đầy đủ để huấn luyện một mô hình Tesseract từ đầu:

 **Chuẩn bị dữ liệu huấn luyện**:

mkdir -p data/train data/eval

# Tạo hoặc thu thập các file .tif và .gt.txt

 **Tạo file .lstmf**:

for file in data/train/\*.tif; do

base=$(basename "$file" .tif)

tesseract "$file" "data/train/$base" lstm.train

done

for file in data/eval/\*.tif; do

base=$(basename "$file" .tif)

tesseract "$file" "data/eval/$base" lstm.train

done

 **Tạo file danh sách**:

find data/train -name "\*.lstmf" > train\_files.txt

find data/eval -name "\*.lstmf" > eval\_files.txt

 **Tạo các file cấu hình**:

# Tạo file font\_properties

echo "Arial 0 0 0 0 0" > font\_properties

# Sao chép các file từ langdata

cp langdata/vie/\* .

 **Bắt đầu huấn luyện**:

# Khởi tạo từ một mô hình cơ sở (fine-tuning)

combine\_tessdata -e vie.traineddata vie.lstm

# Huấn luyện mô hình

lstmtraining --traineddata=vie.traineddata \

--train\_listfile=train\_files.txt \

--eval\_listfile=eval\_files.txt \

--continue\_from=vie.lstm \

--model\_output=vie/vie \

--learning\_rate=0.001 \

--max\_iterations=10000

 **Đánh giá và tinh chỉnh**:

# Đánh giá mô hình sau mỗi 1000 vòng lặp

lstmeval --model vie/vie\_checkpoint \

--traineddata vie.traineddata \

--eval\_listfile eval\_files.txt

 **Đóng gói mô hình**:

# Tạo file traineddata mới

combine\_lang\_model --input\_unicharset vie.unicharset \

--script\_dir langdata \

--output\_dir . \

--lang vie \

--pass\_through\_recoder \

--lstm\_trained\_data vie\_checkpoint

## Đánh giá và tinh chỉnh mô hình

Sau khi huấn luyện, việc đánh giá và tinh chỉnh mô hình là rất quan trọng:

1. **Đánh giá độ chính xác**:
   * Sử dụng công cụ lstmeval để đánh giá mô hình trên tập dữ liệu đánh giá
   * Theo dõi các thông số như Character Error Rate (CER) và Word Error Rate (WER)
2. **Tinh chỉnh siêu tham số**:
   * Điều chỉnh learning rate, batch size, số lượng hidden units, v.v.
   * Thử nghiệm với các kiến trúc LSTM khác nhau (ví dụ: một chiều, hai chiều, số lượng lớp)
3. **Mở rộng dữ liệu huấn luyện**:
   * Nếu độ chính xác chưa đạt yêu cầu, có thể bổ sung thêm dữ liệu huấn luyện
   * Tập trung vào các trường hợp mà mô hình đang gặp khó khăn
4. **Đánh giá trên dữ liệu thực tế**:
   * Kiểm tra mô hình trên các hình ảnh thực tế thay vì chỉ trên tập dữ liệu đánh giá
   * Phân tích các lỗi phổ biến và tìm cách khắc phục