ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

ĐỒ ÁN TỔNG HỢP
HƯỚNG TRÍ TUỆ NHÂN TẠO

Đề tài

Ứng dụng thuật toán OCR trong bài toán nhận diện chữ viết tay

---

Giảng viên hướng dẫn: Vũ Văn Tiến

Nhóm sinh viên thực hiện: Trương Hoàng Nguyên Vũ - 2112673
Nguyễn Đức Bình - 2112899
Nguyễn Đình Quang - 2112096
Mai Phương Nhã - 2111892

TP. HỒ CHÍ MINH, THÁNG 11/2023

Mục lục

1 Giới thiệu đề tài

1.1 Lý do chọn đề tài

Ngày nay, cùng với sự phát triển của Trí tuệ nhân tạo (AI), vai trò của nó cũng trở nên ngày càng lớn trong thế giới hiện đại. Đi đôi với sự phát triển này là sự cải tiến không ngừng trong lĩnh vực **Thị giác máy tính** (Computer Vision). Đây là một lĩnh vực của Trí tuệ nhân tạo (AI) với mục đích làm cho máy tính hay hệ thống có khả năng trích xuất ra thông tin hữu ích từ ảnh kĩ thuật số, video và những nguồn hiển thị nào khác; sau đó sẽ dùng những thông tin trích xuất được mà thực hiện các hành động hay đưa ra các quyết định[1]. Thị giác máy tính là một chủ đề rất rộng với nhiều mảng liên quan, một trong những chủ đề đang rất được quan tâm hiện nay đó chính là vấn đề **Nhận diện trong thị giác máy tính**.

Sự phát triển của Trí tuệ nhân tạo đã tạo động lực rất lớn thúc đẩy quá trình số hóa nói chung và đặc biệt là vấn đề **số hóa các văn bản viết tay với mục đích bảo tồn và lưu trữ** nói riêng; việc này sẽ giúp con người lưu trữ thông tin từ những văn bản viết tay quý giá có thể bị hư hỏng qua thời gian nhằm phục vụ các mục đích bảo tồn, nghiên cứu và phục vụ đời sống với độ tin cậy và độ sẵn sàng cao hơn. Hiện tại đã rất nhiều phần mềm, phương pháp được đưa ra có thể kể đến như Tesseract, OCRopus, GOCR,... giúp giải quyết vấn đề này; tuy vậy vẫn chưa có **phần mềm hay phương pháp nào thật sự tỏ ra vượt trội hơn so với phần còn lại**. Do đó chủ đề này vẫn còn đang dành được sự quan tâm của nhiều các nhà nghiên cứu trên thế giới với mục tiêu cải tiến thêm, tìm kiếm các phương pháp mới giải quyết vấn đề này.

Với những lý do trên cùng thực trạng như vậy, trong Đồ án tổng hợp hướng trí tuệ nhân tạo lần này, chúng em quyết định chọn đề tài **Ứng dụng thuật toán OCR trong bài toán nhận diện chữ viết tay** để làm chủ đề nghiên cứu của mình.

1.2 Giới thiệu đề tài

Nhận diện chữ viết tay từ lâu đã là chủ đề nhận được sự quan tâm rất lớn từ cộng đồng nghiên cứu Trí tuệ nhân tạo nói chung và cộng đồng nghiên cứu Thị giác máy tính nói riêng. Vì lý do trên, người ta đã cho ra đời rất nhiều mô hình và sản phẩm thị trường nhằm phục vụ giải quyết bài toán này cách nhanh chóng; tuy vậy vẫn còn nhiều nhược điểm có thể chỉ ra được từ những sản phẩm này. Tận dụng

[1] IBM, *What is computer vision ?*. Truy cập từ https://www.ibm.com/topics/computer-vision.

lợi thế sẵn có từ những nghiên cứu đi trước, nhóm chúng em có mục tiêu **xây dựng một hệ thống nhận diện chữ viết tay áp dụng các phương pháp phù hợp để có thể cải thiện được hiệu quả nhận diện chữ so với các sản phẩm hiện hành**.

Đối với đề tài **Ứng dụng thuật toán OCR trong bài toán nhận diện chữ viết tay** lần này, nhóm chúng em sẽ xây dựng một mô hình đầu cuối (end-to-end system) nhận dữ liệu đầu vào là hình ảnh chụp một từ tiếng anh được viết tay và kết quả đầu ra là từ tiếng anh được nhận diện chứa bên trong hình ảnh đó với độ tin cậy và chính xác cao.

2 Xác định bài toán và phương pháp giải quyết bài toán

2.1 Xác định bài toán

Với miêu tả đề tài đã đề cập ở trên, ta có thể xác định đây là một bài toán nhận diện thuộc lĩnh vực Thị giác máy tính của Trí tuệ nhân tạo. Riêng với bài toán nhận diện chữ viết tay này, có một mảng học thuật liên quan là *OCR (Optical Character Recognition)* hay còn được gọi là *Nhập diện kí tự quang học*. Sự phát triển nhanh chóng của các thuật toán OCR hiện nay đi đôi với nhu cầu ngày càng lớn về nhận diện kí tự cùng sự xuất hiện của những mô hình đang tỏ ra hiệu quả trong nhiều tác vụ thực tiễn; vì lý do đó nhóm chúng em sẽ áp dụng các thuật toán OCR vào chủ đề này và xác định cụ thể đây là bài toán ứng dụng các thuật toán OCR để nhận diện một từ tiếng anh viết tay từ hình ảnh chụp từ viết tay đó

2.2 Phương pháp giải quyết bài toán

Trong rất nhiều các hệ thống nhận diện kí tự quang học nổi tiếng hiện nay, việc nhận diện chữ viết tay không phải là một công việc đơn duy nhất mà được tách thành nhiều các bước nhỏ hơn phụ trách từng phần của công việc. Lấy ví dụ như một trong các hệ thống OCR vô cùng nổi tiếng, Tesseract, các bước thực hiện cho một tác vụ nhận diện văn bản như sau[1]:

- **Tiền xử lý dữ liệu (Data preprocessing)**: Xử lý dữ liệu đầu vào như khử nhiễu, làm mượt, chuẩn hóa, ... và bao gồm cả việc phân tích các vùng văn bản
- **Tìm kiếm từ và dòng (Line and word finding)**: Tìm kiếm dòng và xác định các từ trong văn bản. Ở bước này còn một bước nhỏ hơn đó là tách các

kí tự có quãng cố định (fixed pitch) trong một từ. Nếu từ nào có quãng kí tự là cố định, Tesseract sẽ tách từng kí tự trong từ đó ra và vô hiệu hóa bước tiếp theo trên từ đó

- **Nhận diện từ (Word recognition)**: Ở bước này có hai bước nhỏ hơn quan trọng đó là *Tách các kí tự dính liền (Chopping joined characters)* và *Liên kết các kí tự bị hỏng (Associating broken characters)*

- **Phân loại kí tự tĩnh (Static character classification)**: Nhận diện và phân loại các dữ liệu kí tự đã được tách ra trước đó

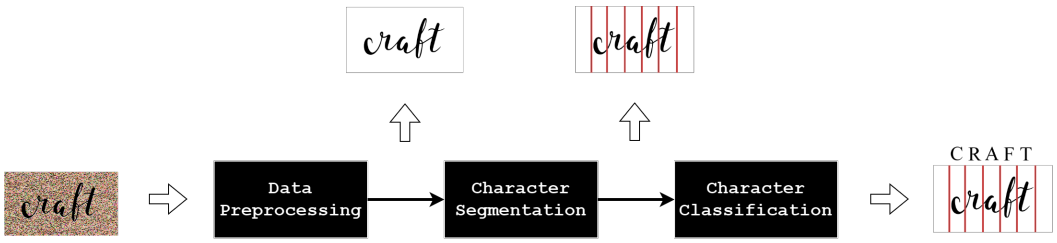
Các hệ thống nổi tiếng hiện nay với các phương pháp xử lý bài toán chữ viết tay hiệu quả là những nguồn tham khảo rất tốt cho việc xây dựng mô hình của nhóm. Sau khi xem xét và kế thừa từ nhiều các phương pháp phù hợp khác nhau, nhóm chúng em đã quyết định đưa ra mô hình giải quyết tổng quát cho bài toán với ba module đảm nhiệm ba công việc chính (được minh họa ở Hình 1) của hệ thống được liệt kê như sau:

- ***Module 1 - Data Preprocessing (tiền xử lý dữ liệu)***: Ở bước này, dữ liệu đầu vào là hình ảnh sẽ được xử lý như là khử nhiễu, làm mượt hay chuẩn hóa, ...

- ***Module 2 - Character Segmentation (tách kí tự)***: Ở bước này, từ dữ liệu đã được xử lý, các kí tự của từ được chụp trong hình ảnh sẽ được tách ra riêng từng kí tự một để phục vụ bước tiếp theo

- ***Module 3 - Character Classification (phân loại kí tự)***: Ở bước này, các kí tự đã được tách ra sẽ được đi qua bộ nhận diện kí tự để có thể nhận diện được, kết quả đầu ra là kí tự đã được nhận diện thành công

Với mỗi module được liệt kê ở trên sẽ đảm nhiệm các giai đoạn cần thiết trong quá trình xử lý bài toán nhận diện chữ viết tay và hiệu quả của mỗi module sẽ ảnh hưởng đến kết quả cuối cùng của toàn bộ hệ thống. Cũng ở mỗi module, các phương pháp phù hợp khác nhau sẽ được tìm hiểu và áp dụng vào mô hình chung giúp tăng hiệu quả tổng thể cho cả hệ thống nhận diện chữ viết tay này. Về phương pháp thực hiện cho mỗi module sẽ được nhóm em làm rõ ở phần sau của báo cáo.



Hình 1: *Các bước thực hiện bài toán và kết quả sau các bước thực hiện*

3 Phương pháp xử lí từng module

3.1 Module Data Preprocessing

Một số bước cơ bản của kỹ thuật Preprocessing có thể kể đến đó là:

- Binarization (nhị phân hóa)
- Skew Correction (khử nghiêng)
- Thining and Skeletionization (làm mỏng và khung hóa)
- Noise Removal (khử nhiễu)

Hãy đi qua từng kỹ thuật Preprocessing một cách cụ thể:

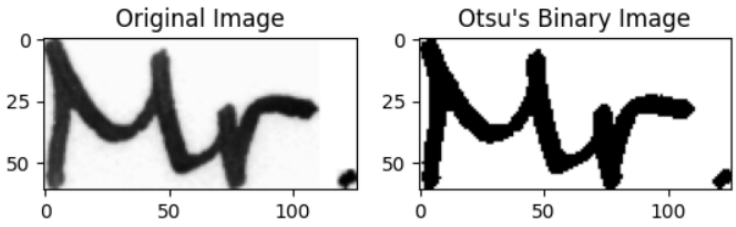
3.1.1 Binarization

Nhị phân hóa là một quá trình chuyển đổi một hình ảnh màu thành một hình ảnh chỉ bao gồm các điểm ảnh màu đen và trắng (Điểm ảnh màu đen có giá trị 0 và điểm ảnh màu trắng có giá trị 255). Theo một quy tắc cơ bản, điều này có thể thực hiện được bằng cách **thiết lập một threshold**.

Tuy nhiên, nếu chúng ta thiết lập threshold = 127 (vì đó là chính giữa của phạm vi giá trị của điểm ảnh từ 0 đến 255, giá trị lớn hơn threshold được coi là điểm ảnh màu trắng, ngược lại được coi là điểm ảnh màu đen). thì cách này lại không mang lại kết quả mong muốn. Trong trường hợp mà điều kiện thiếu ánh sáng không đồng đều trên hình ảnh, phương pháp này sẽ không hiệu quả. Vì vậy, ta sẽ dùng phương pháp khác đó chính là **Otsu' Binarization**.

Otsu' Binariztio là phương pháp cung cấp một threshold cho toàn bộ hình ảnh, xem xét các đặc điểm khác nhau của toàn bộ hình ảnh (như điều kiện ánh sáng, độ

tương phản, độ sắc nét,...) và threshold đó được sử dụng để thực hiện việc nhị phân hóa hình ảnh.



Hình 2: *Ví dụ về Otsu' Binarization*

3.1.2 Skew Correction

Khi quét một chữ, đôi khi hình ảnh có thể nghiêng một chút (hình ảnh không được căn chỉnh hoàn toàn theo một góc nào đó so với ngang). Trong quá trình trích xuất hình ảnh đã quét, việc phát hiện và chỉnh sửa sự nghiêng này rất quan trọng.

Có nhiều kỹ thuật được sử dụng để chỉnh sửa sự nghiêng, một trong những phương pháp hiệu quả nhất đó là **phương pháp chiếu (projection profile method)**.

Trong phương pháp này, đầu tiên chúng ta sẽ lấy hình ảnh nhị phân và sau đó thực hiện phép chiếu theo chiều ngang (tính tổng số điểm ảnh trên từng hàng của ma trận hình ảnh) để thu được biểu đồ của số điểm ảnh theo chiều cao của hình ảnh, tức là số điểm ảnh nền trên mỗi hàng.

Tiếp theo, hình ảnh sẽ được quay ở các góc khác nhau (Với một khoảng góc nhỏ gọi là δ), và sự khác biệt giữa các đỉnh sẽ được tính toán (Phương sai cũng có thể được sử dụng làm một trong các chỉ số). Góc mà tại đó sự khác biệt lớn nhất giữa các đỉnh (Hoặc phương sai) được tìm thấy, góc tương ứng đó sẽ là góc nghiêng cho hình ảnh.

Sau khi tìm thấy góc nghiêng, chúng ta có thể chỉnh độ nghiêng bằng cách quay hình ảnh một góc bằng với góc nghiêng theo hướng ngược lại của sự nghiêng.

3.1.3 Thinning and Skeletonization

Vì chúng ta sử dụng hệ thống OCR cho văn bản viết tay, phong cách các người viết khác nhau và do đó độ rộng nét chữ cũng sẽ khác nhau. Vì vậy để làm cho nét chữ đồng đều, chúng ta phải làm mảnh nó.

3.1.4 Noise Removal

Mục tiêu chính của giai đoạn loại bỏ nhiễu là làm mịn hình ảnh bằng cách loại bỏ các điểm/ đám chấm nhỏ có độ sáng cao hơn so với phần còn lại của hình ảnh.

3.2 Module Character Segmentation

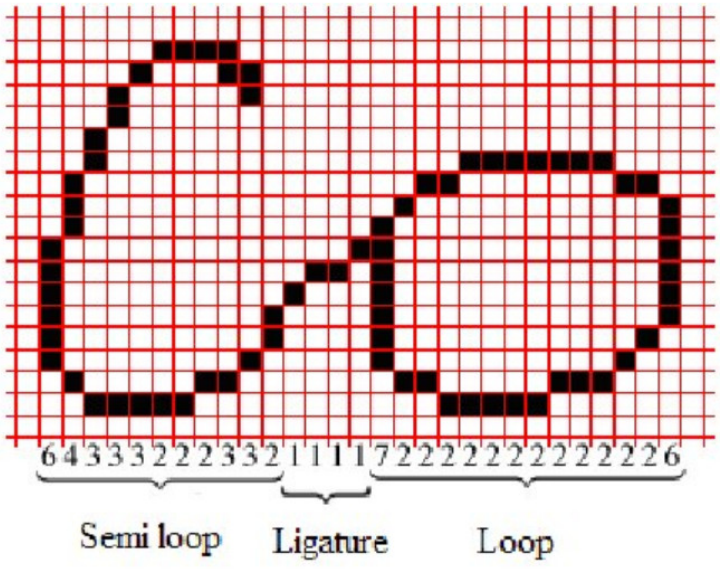
Ở module này, nhóm chúng em sẽ áp dụng phương pháp **Vertical Histogram Projection** để giải quyết hiệu quả bài toán phân tách kí tự.

3.2.1 Giới thiệu phương pháp

Điều kiện: Hình ảnh từ viết tay trước khi được đưa vào phân tách cần được thực hiện việc tiền xử lý

- Sau bước này, hình ảnh chữ viết tay được xoay đúng (slant correction, skew correction), các nét được làm mảnh (thinning) theo đúng quy chuẩn (độ rộng nét chữ không vượt quá một ô pixel)
- Hình ảnh sau khi xử lý là một ma trận h x w, với h là chiều cao, w là chiều rộng. Ma trận này được đưa về dạng nhị phân (0,1) để dễ tính toán.
  - Mỗi pixel có nét chữ đi qua có giá trị 1 (pixel "nổi")
  - Pixel trống có giá trị 0 (pixel "nền")

Bước 1: Tính tổng của những pixel "nổi" trong một cột. Những cột nào có giá trị tổng này bằng 0 hoặc 1 được lưu là những cột chia tiềm năng (PSC). Ở bước này, ta gặp vấn đề phân chia quá mức từ hai nguyên nhân. Đầu tiên là do những nét nối giữa những chữ cái với nhau. Thứ hai là những nét nối trong chính những chữ cái như 'm', 'n', 'h', 'w', 'u',... . Ở đây những chữ cái này được gọi là chữ cái mở (không có vòng - no loop). Những chữ cái có vòng (loop) hay bán vòng (semi loop) được gọi là chữ cái đóng, ví dụ như a, b, c, e,... . Những chữ cái mở, khi tính tổng giá trị các pixel sẽ cho giá trị 1, nên cũng trở thành những cột phân chia tiềm năng.



Hình 3: *Tổng các pixel nổi*

Bước 2: Trước hết, để giải quyết việc phân chia quá mức do những nét nối, ta sẽ lấy giá trị trung bình của những điểm chia tiềm năng (PSC) liên tiếp mà khoảng cách giữa chúng nhỏ hơn giá trị *threshold*. Trong đó, *threshold* là giá trị chiều ngang lớn nhất không thể chứ một chữ cái (ví dụ: i, l, j, ...). Giá trị *threshold* được lựa chọn theo kinh nghiệm từ nhiều lần thực hành trên bộ dữ liệu.

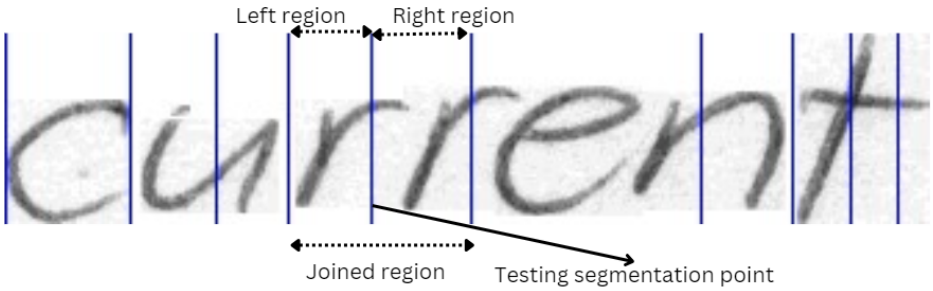
Bước 3: Tiếp theo, để xử lý vấn đề phân chia quá mức do nét nối trong những chữ cái mở, ta xét tiếp đến những đặc điểm khác. Trước hết, xem xét giá trị tổng pixel - vertical histogram tại vị trí điểm chia tiềm năng đang xét. Nếu giá trị đó bằng 0 thì cột đó là điểm phân chia mà không cần xem xét đến những vấn đề sau. Nếu giá trị đó lớn hơn 0 thì tiến hành xác thực thông qua một mô hình ANN.

Ở đây, ta có hai cách tiếp cận để xác thực điểm phân chia:

- **Cách 1**: Mô hình ANN được train để phân biệt giữa những điểm phân chia hợp lệ và không hợp lệ. Để train được mô hình này, trước tiên ta cần trích xuất một số thông tin về điểm chia đó (ví dụ như ma trận mật độ - density matrix, transition feature, direction feature, ...) và label **đúng** hoặc **sai** được phân loại thủ công. Những điểm phân chia, sau khi được xác thực bằng mô hình, trả về kết quả sai sẽ bị loại khỏi danh sách những điểm phân chia.

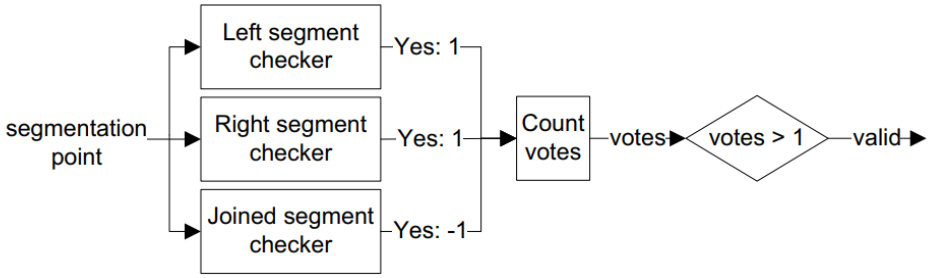
- **Cách 2**: Ta tận dụng mô hình CNN đã được train để phân loại ký tự ở module phân loại. Mô hình này, bên cạnh việc phân loại các chữ cái, cần nhận biết

được ký tự "rác", tức là phần không đủ là một ký tự, hoặc nhiều hơn một ký tự. Tại một điểm phân chia bất kỳ, lần lượt kiểm tra vùng phân chia bên trái - left segment, vùng phân chia bên phải - right segment, và vùng phân chia ghép - join segment (tức là lấy vùng phân chia có rìa bên trái và phải lần lượt là các điểm phân chia trái và phải liền kề nó).



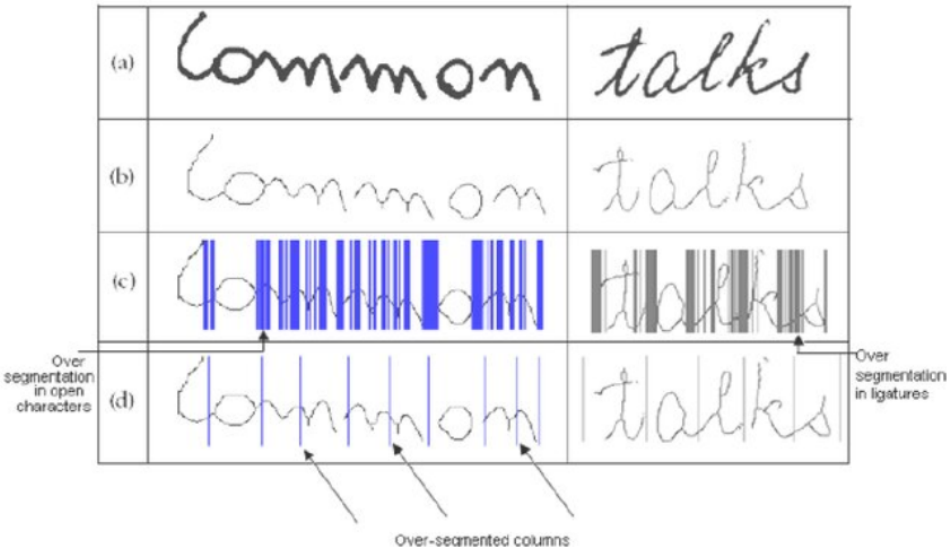
Hình 4: *Các vùng phân tách*

– Điểm phân tách được kiểm tra theo cấu trúc sau:



Hình 5: *Cấu trúc kiểm tra điểm phân tách*

– Cụ thể, vùng phân tách bên trái và bên phải, nếu sau khi được kiểm tra mà trả về giá trị là một trong những chữ cái hợp lệ thì cộng 1 vào giá trị **votes**. Vùng phân tách ghép, nếu trả về giá trị hợp lệ, thì trừ 1 vào giá trị **votes**. Ngược lại, nếu sau khi kiểm tra các vùng này mà trả về giá trị là ký tự 'rác' thì không ảnh hưởng đến **vote**. Sau các bước này, nếu **votes** lớn hơn 1 thì kết luận điểm phân chia đó hợp lệ. Trường hợp ngược lại, ta loại bỏ điểm đó khỏi danh sách điểm phân chia.



Hình 6: *Các bước phân tách*

3.2.2 Pseudo Code

Một hình ảnh có chiều cao h và chiều rộng w được biểu diễn bởi một ma trận P, P $\in \{0, 1\}$

$$p_{i,j} \in P \begin{cases} i = 1, 2, 3, ..., h \\ j = 1, 2, 3, ..., w \end{cases}$$

1. **Tính tổng các pixel nổi cho từng cột**

$$a_j = \sum_{i=1}^{h} p_{i,j}, \quad j = 1, 2, ..., w$$

2. **Tìm ra những cột chia tiềm năng (PSC - Potential Segmentation Column)**

$$PSC = \{psc \in a_j | a_j \leq 1, j = 1, 2, ...w\}$$

3. **Loại bỏ những PSC dư thừa trong những PSC liên tiếp gần nhau (thresholding)**

Khởi tạo m, n, sum = 0
for k = 1..length(PSC) - 1
  $if(PSC_{k+1} - PSC_k \leq threshold)$ then
    $sum = sum + PSC_k$
  $n = n + 1$
  else
    if $(n > 0)$ then $SC_m = round(sum/n)$
    else $SC_m = CSC_k$
    $m = m + 1, sum = 0, n = 0$
endfor

4. **Xác thực các cột phân chia, trừ cột đầu và cuối**
   $validate\_SC_1 = SC_1$
   $k = 2, m = 2$
   while $(m < length(SC))$
     $save\_segment\_col = true$
     $jump = false$
     if $histogram(SP_m) > 0$ then
       extract $feature(SP_m)$ then feed to ANN validation model
       if ANN return "incorrect" then
         $save\_segment\_col = false$
       else
         $m = m + 1$
         $jump = true$
       end
     end
   end while
   $validate\_SP_k = SP_m$

3.3 Module Character Classification

3.3.1 Mô hình Convolutional Neural Network (CNN)

Mạng neuron tích chập (CNN) là một tập hợp các lớp convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Trong mô hình mạng truyền ngược (feedforward

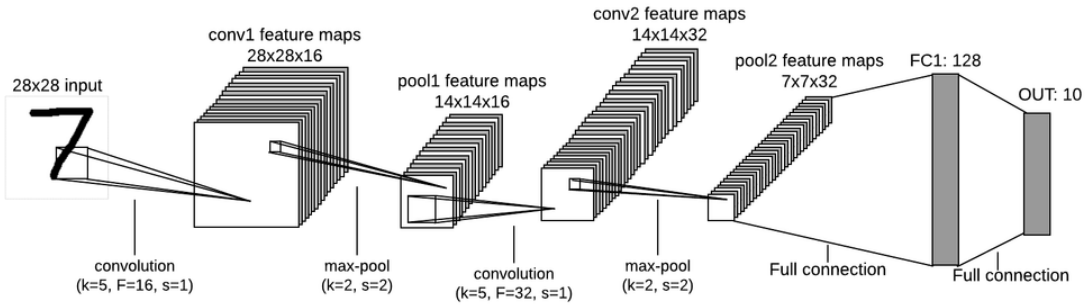
neural network) thì mỗi neural đầu vào (input node) cho mỗi neural đầu ra trong các lớp tiếp theo.

Mô hình này gọi là mạng kết nối đầy đủ (fully connected layer) hay mạng toàn vẹn (affine layer). Còn trong mô hình CNNs thì ngược lại. Các layer liên kết được với nhau thông qua cơ chế convolution.

Layer tiếp theo là kết quả convolution từ layer trước đó, nhờ vậy mà ta có được các kết nối cục bộ. Như vậy mỗi neuron ở lớp kế tiếp sinh ra từ kết quả của filter áp đặt lên một vùng ảnh cục bộ của neuron trước đó.

Mỗi một lớp được sử dụng các filter khác nhau thông thường có hàng trăm hàng nghìn filter như vậy và kết hợp kết quả của chúng lại. Ngoài ra có một số layer khác như pooling/subsampling layer dùng để chắt lọc lại các thông tin hữu ích hơn (loại bỏ các thông tin nhiễu).

Trong quá trình huấn luyện mạng (traning) CNN tự động học các giá trị qua các lớp filter dựa vào cách thức mà bạn thực hiện. Ví dụ trong bài toán nhận diện kí tự, mô hình CNN bao gồm những bước sau: Input image -> Convolutional layer (Conv) + Pooling layer (Pool) -> Fully connected layer (FC) -> Output.



Hình 7: *Mô hình CNN trong tác vụ phân lớp ảnh.*

3.3.2 Convolution Layer

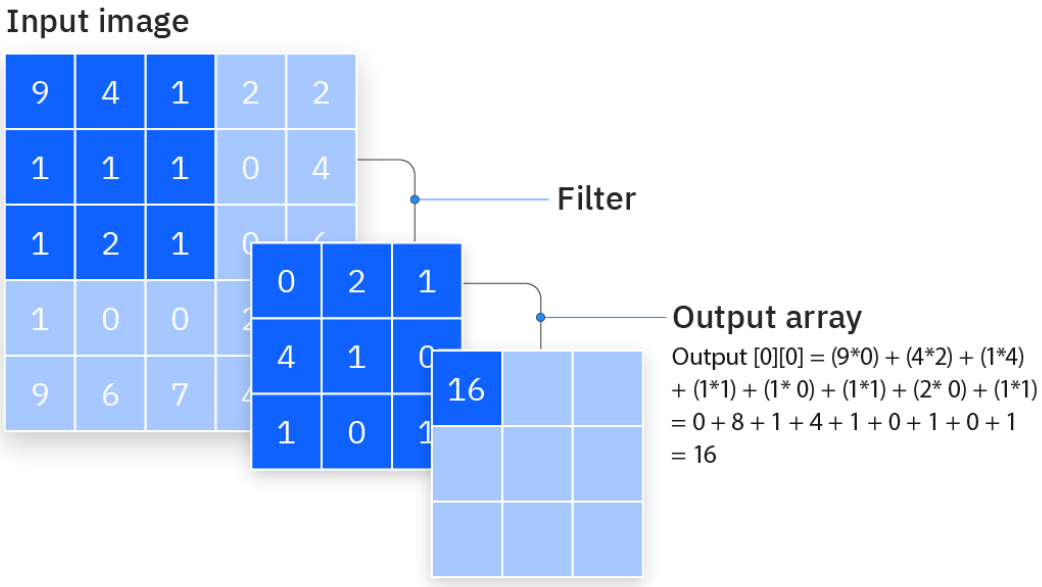
Convolution layer là khối xây dựng chính của CNN. Nó chịu phần lớn công việc tính toán của network.

Giả sử rằng đầu vào là một hình ảnh màu, được tạo thành từ một ma trận các điểm ảnh trong không gian 3D. Điều này có nghĩa rằng đầu vào sẽ có ba chiều - chiều cao, chiều rộng và chiều sâu - tương ứng với màu sắc RGB trong một hình ảnh. Chúng ta cũng có một bộ feature detector, còn được gọi là kernel hoặc bộ lọc, sẽ di chuyển qua các vùng thu nhận trên hình ảnh để kiểm tra xem đặc trưng có tồn tại

không. Quá trình này được gọi là convolution.

Feature detector là một mảng hai chiều (2-D) chứa các trọng số, đại diện cho một phần của hình ảnh. Mặc dù chúng kích thước có thể linh hoạt, kích thước của bộ lọc thường là một ma trận 3x3; điều này cũng xác định kích thước của vùng thu nhận. Bộ lọc sau đó được áp dụng cho một khu vực trên hình ảnh, tích vô hướng được tính toán giữa các điểm ảnh đầu vào và bộ lọc. Tích vô hướng này sau đó được đưa vào một mảng đầu ra. Sau đó, bộ lọc dịch chuyển 1 stride, lặp lại quá trình cho đến khi kernel đã quét qua toàn bộ hình ảnh. Đầu ra cuối cùng từ loạt tích vô hướng từ đầu vào và bộ lọc này được gọi là một activation map.

Sau mỗi phép convolution, mạng CNN áp dụng hàm ReLU vào activation map để tạo tính phi tuyến tính cho mô hình.



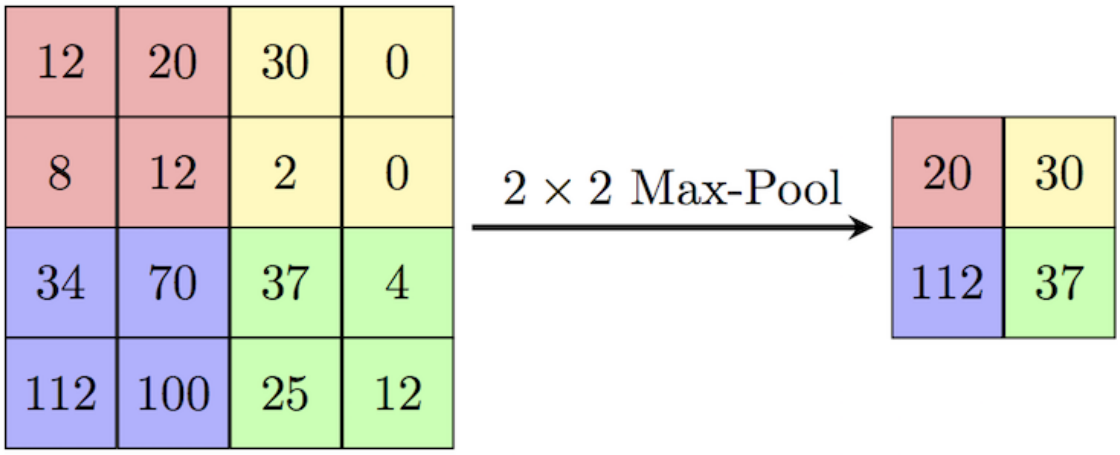
Hình 8: *Convolution layer*

3.3.3 MaxPooling Layer

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

Gọi pooling size kích thước K*K. Input của pooling layer có kích thước H*W*D, ta tách ra làm D ma trận kích thước H*W. Với mỗi ma trận, trên vùng kích thước K*K trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh. Nhưng hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên.

Có 2 loại pooling layer phổ biến là: max pooling và average pooling. Cụ thể trong đề tài này, ta chọn MaxPooling làm phép pooling chính.

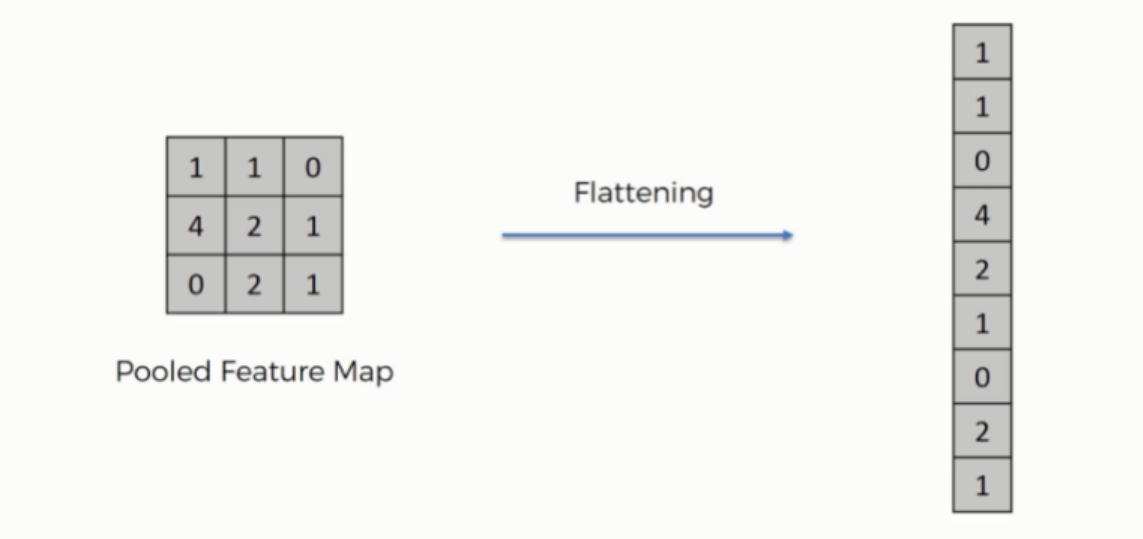


Hình 9: *MaxPooling layer*

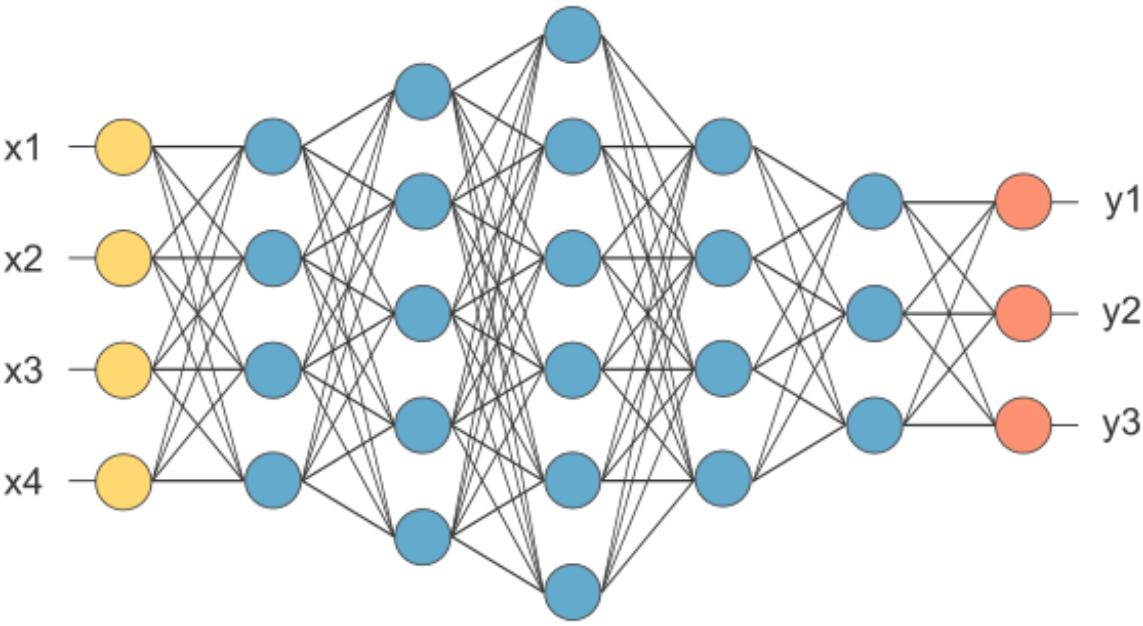
3.3.4 Fully connected Layer

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,...) thì tensor của output của layer cuối cùng, kích thước H*W*D, sẽ được chuyển về 1 vector kích thước (H*W*D)

Với FC layer được kết hợp với các tính năng lại với nhau để tạo ra một mô hình. Cuối cùng sử dụng softmax hoặc sigmoid để phân loại đầu ra.



Hình 10: *Flatten Layer*

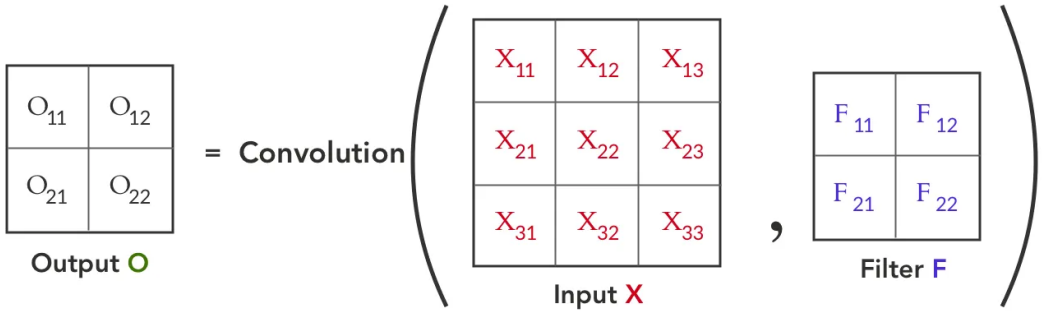


Hình 11: *Fully Connected Layer*

3.3.5 Back Propagation

Giả sử hàm f là hàm convolution giữa input X và filter F. Input X là một ma trận 3x3, còn filter F là một ma trận 2x2 như hình dưới. Convolution giữa input X và filter F cho ta output O.

Ta sẽ tính toán mức độ ảnh hưởng của input X và filter F đến output O, từ đó



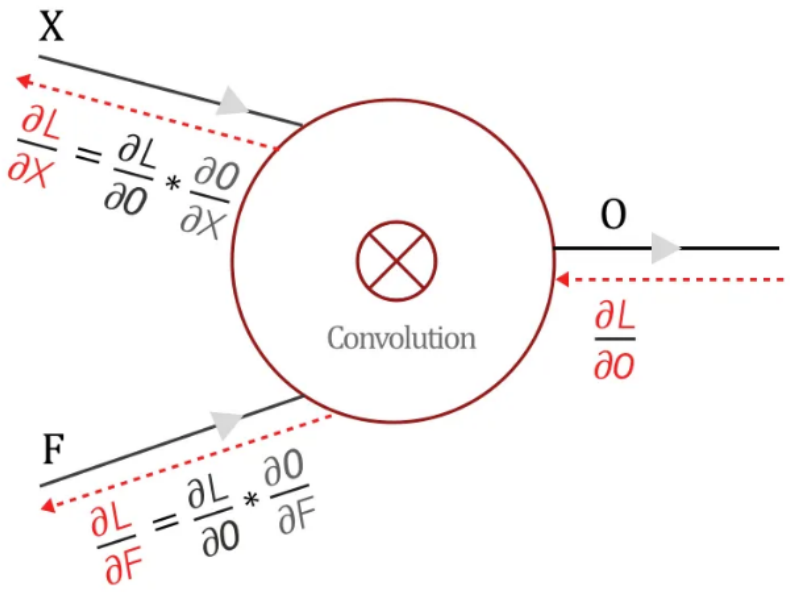
Hình 12: *Hàm convolution giữa input X và filter F cho ra output O*

ta có thể cập nhật lại filter F và dùng mức ảnh hưởng của input X làm output của layer trước đó. Cụ thể, mức ảnh hưởng ở đây chính là gradient descent. Ta tính toán được gradient của output O đối với loss funtion L $\frac{\partial L}{\partial O}$, từ đó, ta sẽ tính toán được $\frac{\partial L}{\partial X}$ và $\frac{\partial L}{\partial F}$ dựa vào Chain rule:

Trong đó:

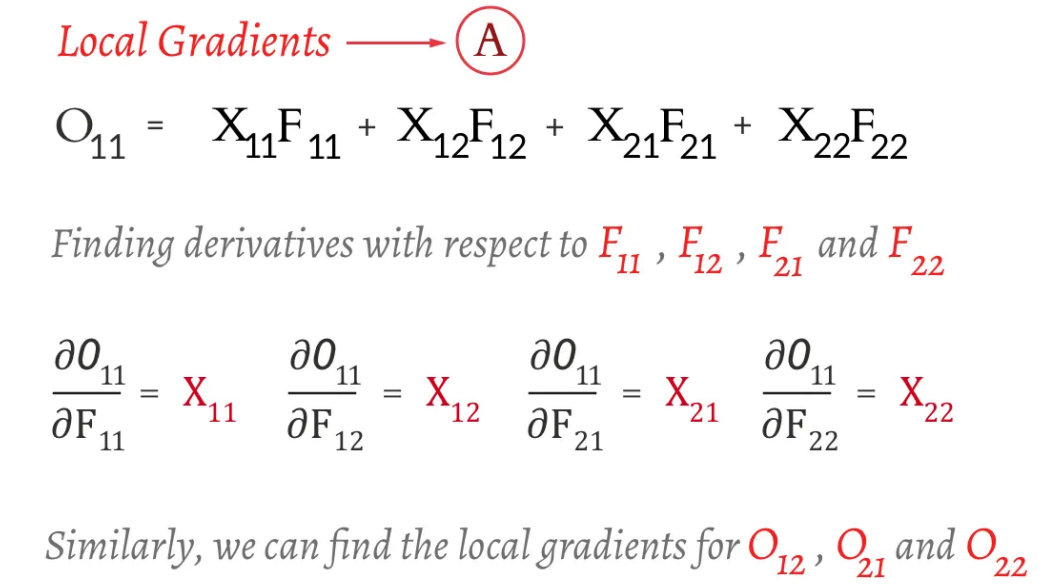
- $\frac{\partial O}{\partial F}$ và $\frac{\partial O}{\partial X}$ là local gradient.
- $\frac{\partial L}{\partial O}$ là loss từ layer sau lan truyền về layer trước

$\frac{\partial L}{\partial O}$ ở layer cuối cùng được tính toán bằng cách tính đạo hạm của hàm loss L theo O. Như vậy để tìm $\frac{\partial L}{\partial X}$ và $\frac{\partial L}{\partial F}$, ta cần tìm $\frac{\partial O}{\partial X}$ và $\frac{\partial O}{\partial F}$. Đầu tiên, để tìm $\frac{\partial O}{\partial X}$, ta tính đạo hàm của output O đối với filter F, dựa vào phép convolution đã biết, ta tính được phần tử đầu tiên O^{11} như sau:

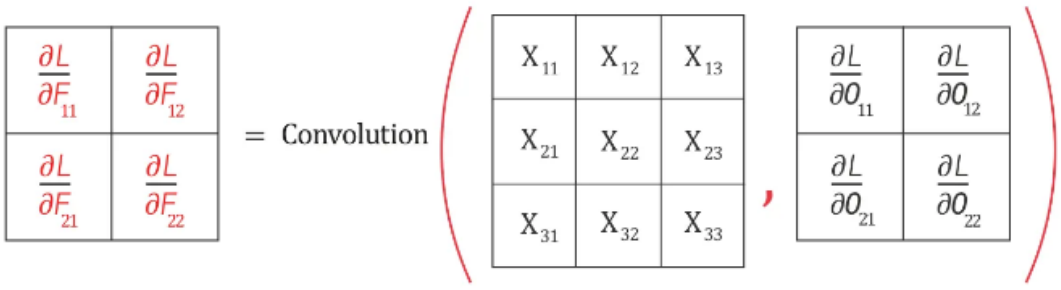


Hình 13: *backward propagation*

Sau khi tính toán được $\frac{\partial O}{\partial F}$, ta sẽ sử dụng Chain rule để tính toán $\frac{\partial L}{\partial X}$ bằng công thức $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial X}$. Trong đó $\frac{\partial O}{\partial X}$ vừa tính được ở trên, còn $\frac{\partial L}{\partial O}$ là gradient của layer sau lan truyền ngược về layer trước, $\frac{\partial L}{\partial O}$ của layer cuối cùng được tính toán bằng cách tính đạo hàm của hàm loss L đối với output O. Như vậy, sau khi tính toán được $\frac{\partial O}{\partial F}$, ta có công thức tính $\frac{\partial F}{\partial X}$ được biểu diễn dưới dạng ma trận cụ thể như sau:

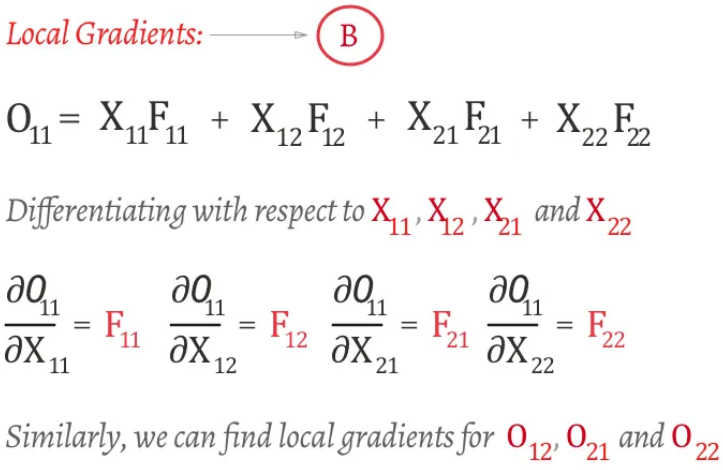


Hình 14: *tính toán* $\frac{\partial O}{\partial F}$

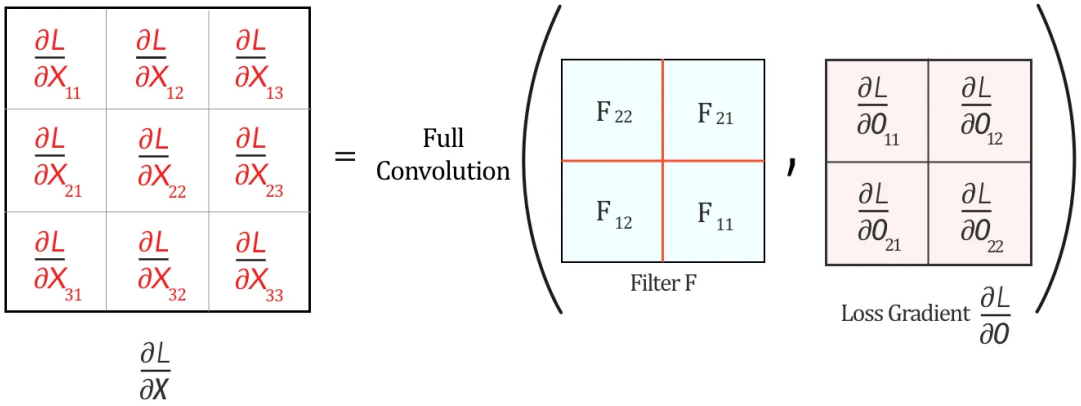


Tiếp theo ta cần tính toán $\frac{\partial O}{\partial X}$. Tương tự như cách tính $\frac{\partial O}{\partial F}$, ta có công thức tính $\frac{\partial O}{\partial X}$ như sau:

Sau khi tính được $\frac{\partial O}{\partial X}$, ta dễ dàng tính toán được $\frac{\partial L}{\partial X}$ sử dụng chain rule,



sau khi tìm được công thức tính $\frac{\partial L}{\partial X}$, ta nhận thấy công thức đó có thể biểu diễn dưới dạng ma trận "Full convolution" như sau:



Để ý thấy filter F trong công thức trên đã bị xoay 180 độ so với filter F ban đầu. Từ những phân tích trên, ta có thể kết luận công thức tính gradient của những convolution layer như sau:

Backpropagation in a Convolutional Layer of a CNN

Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X, \text{Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^\circ \text{rotated Filter } F, \text{Loss Gradient } \frac{\partial L}{\partial O}\right)$$

Trong đó, $\frac{\partial L}{\partial F}$ dùng để cập nhật lại filter F bằng công thức $F_{new} = F - \alpha\frac{\partial F}{\partial X}$. α là learning rate, tạm hiểu là một phần tỷ lệ của một bước dịch chuyển trọng số mô hình được cập nhật theo các mini-batch truyền vào. Còn $\frac{\partial L}{\partial X}$ sẽ đóng vai trò như $\frac{\partial O}{\partial X}$ cho layer trước đó.

3.3.6 Hiện thực mô hình

Từ ý tưởng xử lí module phân loại kí tự, nhóm em đã hiện thực mô hình CNN như sau:

Sau khi xử lí ảnh để đưa về ảnh trắng đen có kích thước (40x40)px, ta xây dựng

```
model = models.Sequential()
# Convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(40, 40, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))

# Flatten the output for the fully connected layers
model.add(layers.Flatten())

# Fully connected layers
model.add(layers.Dense(165, activation='relu'))
model.add(layers.Dropout(0.5))  # Dropout layer to reduce overfitting
model.add(layers.Dense(52, activation='softmax'))  # Adjust 'num_classes' as needed

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

Hình 15: *CNN Model*

layer đầu tiên của mô hình bằng cách thêm 32 filter, mỗi filter có kích thước 3x3, sử dụng hàm activation ReLU, với input đầu vào có kích thước 40x40 px. Sau đó thêm layer MaxPooling với size=(2,2) để trích xuất những đặc điểm quan trọng nhất của input. Trong mô hình này, ta thêm dropout layer với xác suất 0.2 để giảm rủi ro bị overfitting.

Ta xây dựng layer thứ 2 và thứ 3 tương tự với layer đầu tiên. Ở layer thứ 2, ta sử dụng 64 filter với kích thước 3x3, ReLU activation, max pooling với size 2x2, dropout 0.2. Ở layer thứ 3, ta sử dụng 128 filter với kích thước 3x3, ReLU activation, max pooling với size 2x2, dropout 0.2.

Sau khi xây dựng xong convolution layer, ta làm phẳng output để làm input cho fully connected layer. Ở layer này, ta xây dựng layer đầu tiên với 165 neuron và dùng ReLU activation. Sau đó thêm dropout layer 0.5 để giảm thiểu overfitting. Cuối cùng, ta thêm layer cuối với 52 neuron cùng hàm activation softmax.

Note: Các hàm activation được sử dụng

- ReLU: f(x)=max(0,x) : nếu đầu vào lớn hơn 0 thì sẽ trả về đầu vào, nếu đầu vào nhỏ hơn 0 thì sẽ trả về 0.
- softmax: $a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}, \quad \forall i = 1, 2, \ldots, C$: giá trị đầu ra là một số dương trong khoảng [0,1] và tổng các giá trị đầu ra bằng 1

4 Độ đo (metric) cho từng module

4.1 Độ đo cho module tiền xử lí hình ảnh

Trong tiền xử lý hình ảnh, có một số độ đo quan trọng mà ta có thể sử dụng để đánh giá hiệu suất hoạt động. Dưới đây là một số độ đo phổ biến:

- **Thời gian xử lý (Processing Time):** Đây là thời gian mà module tiền xử lý mất để xử lý một hình ảnh. Điều này quan trọng để đảm bảo rằng xử lý hình ảnh xảy ra trong khoảng thời gian chấp nhận được để không gây ảnh hưởng các giai đoạn phía sau.
- **Độ chính xác (Accuracy):** Đánh giá khả năng của module xử lý hình ảnh trong việc nhận diện và phân loại đối tượng hoặc đặc trưng trong hình ảnh. Độ chính xác cao là mục tiêu quan trọng.
- **Sai số (Error Rate):** Đây là tỷ lệ giữa số lượng hình ảnh được xử lý sai và tổng số hình ảnh đầu vào. Sai số càng thấp càng tốt.
- **Độ phủ (Coverage):** Xác định tỷ lệ của hình ảnh đầu vào được xử lý thành công. Điều này quan trọng trong các ứng dụng y tế hoặc kiểm tra hình ảnh.

4.2 Độ đo cho module tách các kí tự

Để đánh giá một mô hình phân tách chữ cái, ta xét đến hai yếu tố là *Độ phức tạp tính toán (Computational Complex)* và *Hiệu suất của mô hình (Performance)*:

- **Độ phức tạp tính toán**: Thường được xác định thông qua các trọng số như độ lớn bộ dữ liệu và thời gian cần thiết để train model, số lượng các thao tác tính toán,... Tuy nhiên yếu tố này là tổng quát đối với các module model

training cũng như các bài toán model training, ta sẽ xét sâu hơn đến yếu tố thứ hai.

- **Đánh giá hiệu năng**: Phân tách là quá trình quan trọng trong nhận diện chữ viết tay. Ta chỉ có thể đạt được tỷ lệ nhận diện cao nếu các chữ cái được phân tách đúng. Vì vậy, việc cải thiện độ chính xác của quá trình này là thiết yếu.

Hiệu suất của quá trình phân tách chữ cái trong từ viết tay có thể được đánh giá thông qua một số độ đo như sau:

- **Độ chính xác (Accuracy)**: Hay segmentation rate, là độ đo phổ biến nhất để đánh giá hiệu quả của kỹ thuật phân tách. Chỉ số này được tính toán bằng cách chia số ký tự được phân tách đúng cho tổng số ký tự.
- **Tỷ lệ lỗi phân tách (Segmentation Error Rate – SER)**: Ngược lại với Độ chính xác (Accuracy), tức là chỉ số này được tính bằng cách chia những ký tự bị phân tách sai cho tổng số ký tự. Tỷ lệ này gồm ba chỉ số
  - Tỷ lệ phân tách quá mức (Over-segmentation rate): Tính dựa trên số lượng những điểm phân tách quá mức (tại những điểm không nên có điểm phân tách)
  - Tỷ lệ phân tách thiếu (Missed-segmentation rate): Tính dựa trên số lượng những điểm phân tách không được phát hiện ra
  - Tỷ lệ phân tách xấu (Bad-segmentation rate): Tính dựa trên những điểm phân tách không chia hai chữ cái một cách hợp lý

Không giống với chữ in trên máy hay chữ in viết tay, việc phân tách chữ cái trong chữ viết tay thường gặp những vấn đề dưới đây:

- Những ký tự khi viết có thể bị dính nhau làm quá trình phân tách trở lên phức tạp hơn
- Những phần cao, trồi lên (ví dụ các chữ cái **h, l, k, b, d**) hoặc thấp, trồi xuống của (ví dụ ở các chữ cái **g, p, j, y,** ...) các chữ cái có thể chồng lên các ký tự trước hoặc sau nó gây khó khăn trong việc phân tách

Những vấn đề trên gây ra hai lỗi thường gặp của quá trình phân tách là *phân tách quá đà (Over–segmentation)* hoặc *phân tách chưa đủ (Under–segmentation)*. Vì vậy độ đo **Accuracy** được xem là phù hợp nhất cho bài toán phân tách chữ viết tay. Giá trị **Accuracy** càng lớn thì phân tách càng chính xác.

Table 2
Comparative study of segmentation results.

Author	Segmentation approach	Segmentation rate (%)	Database
Verma and Gader [14]	Neural based method	76.52	CEDAR subset
Blumenstein and Verma [15]	Neural conventional method	78.85	CEDAR subset
Verma [16]	Feature based & neural assistance	84.87	CEDAR subset of 300 words
Verma [20]	Heuristic approach	81.08	CEDAR subset
Cheng et al. [17]	Neural based method	95.27	CEDAR subset of 317 words
Cheng and Blumenstein [18]	Neural based method	84.19	CEDAR subset of 317 words
Lee and Verma [21]	Neuro-heuristic approach	83.46	CEDAR subset of 311 words
Proposed approach	Geometric features based & neural assistance	88.08	CEDAR test set of 317 words

Hình 16: *Các độ chính xác được ghi nhận [2]*

Ta có thể tham khảo độ chính xác của bài toán phân tách chữ cái trong các nghiên cứu trước đó để đánh giá hiệu suất mô hình của nhóm.

4.3 Độ đo cho module phân loại kí tự

Độ chính xác của một mô hình được định nghĩa bằng số lần dự đoán phân loại kí tự đúng chia cho tổng số lần dự đoán phân loại kí tự.

Công thức này cung cấp một định nghĩa dễ hiểu cho bài toán phần loại nhị

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Hình 17: *Công thức độ chính xác của mô hình*

phân. Tuy nhiên, phân loại kí tự là một bài toán phân loại đa lớp, vậy nên ta sẽ phải dùng công thức tính độ chính xác cho bài toán phân loại đa lớp.
trong đó:

$$Multiclass\ Accuracy(y_i, z_i) = \frac{1}{n}\sum_{1}^{n}[[y_i == z_i]]$$

Hình 18: *Độ chính xác cho bài toán phân loại đa lớp*

- n: số lượng mẫu
- [[...]]: là dấu ngoặc Iverson, dấu ngoặc sẽ trả về 1 nếu như biểu thức trong ngoặc là đúng và trả về 0 nếu biểu thức trong ngoặc sai.

- y_i và z_i lần lượt là output dự đoán và nhãn của mẫu.

Xét ví dụ sau, ma trận sau chỉ ra những giá trị đúng và giá trị dự đoán của bài toán phân loại 3 lớp.

		True values			
	Classes	A	B	C	Total
Predictions	A	6	0	7	13
	B	2	4	1	7
	C	5	0	2	7
	Total	13	4	10	27

Hình 19: *confusion matrix*

Ta tính toán độ chính xác bằng cách chia số lần dự đoán chính xác (đường chéo chính) cho tổng số mẫu như sau:

$$Multiclass\ Accuracy = \frac{6+4+2}{27} = 0.44$$

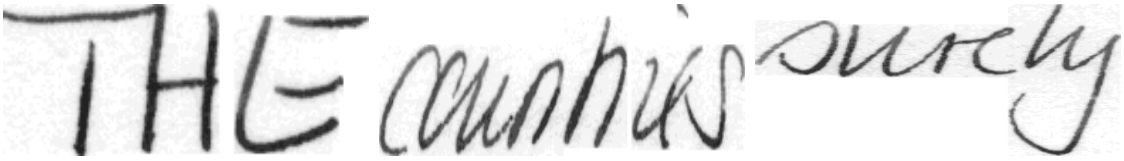
Hình 20: *tính toán độ chính xác*

Kết quả trên chỉ ra rằng mô hình có độ chính xác 44% trong bài toán phân loại đa lớp.

5 Bộ dữ liệu được sử dụng

Với hệ thống đầu cuối này, nhóm chúng em đã tách riêng các module ra với mục đích thực hiện hiệu quả hơn việc hiện thực bên cạnh đó giúp tránh việc phụ thuộc giữa các module và rắc rối trong quá trình thực hiện đồ án. Cũng vì lý do này mà ở mỗi module chúng em sẽ có thể sử dụng các bộ dữ liệu khác nhau sao cho phù hợp hơn với nhiệm vụ từng module.

Về bộ dữ liệu chính để đánh giá hệ thống đầu cuối, chúng em sẽ sử dụng bộ dữ liệu **IAM Handwriting Word** có thể dễ dàng được truy cập ở đây. Đây là bộ dữ liệu các từ tiếng anh được tách ra từ bộ dữ liệu viết tay IAM nổi tiếng, bộ dữ liệu sử dụng chứa khoảng **115320** ảnh chưa được xử lý của các từ được tách ra từ các văn bản tiếng anh của bộ dữ liệu gốc và được sắp xếp vào các thư mục của từng trang, từng dòng văn bản cụ thể chứa hình ảnh từ đó; bộ dữ liệu cũng đi kèm với file chứa nội dung của các từ được trích xuất ra đã được kiểm tra thủ công. Một số ảnh của từ trong bộ dữ liệu được cho ở hình 2.



(a) THE (b) countries (c) surely

Hình 21: *Một số hình ảnh của bộ dữ liệu*

5.1 Mô tả dataset từng module

5.1.1 Module tiền xử lý hình ảnh và module chia tách kí tự

Hai module tiền xử lý hình ảnh và module chia tách kí tự sẽ sử dụng bộ dữ liệu **IAM Handwriting Word** như đã được giới thiệu trước đó.

5.1.2 Module phân loại kí tự

Module phân loại kí tự xử dụng bộ dữ liệu **Digit classifier CNN** có thể được truy cập ở đây. Bộ dữ liệu này gồm **3410** hình ảnh trắng đen đã qua xử lý với mỗi hình ảnh là 1 ký tự cụ thể. Bộ dữ liệu đi kèm một file chứa nhãn kí tự chính xác của từng ảnh trong bộ dữ liệu.

(a) l (b) m

Hình 22: *Một số hình ảnh của bộ dữ liệu*

6 Kết quả và đánh giá

6.1 Module Character Segmentation

(a) Từ trước phân chia

(b) Các ký tự sau khi được phân tách chuyển cho module phân loại

Hình 23: *Ví dụ về kết quả của module phân tách*

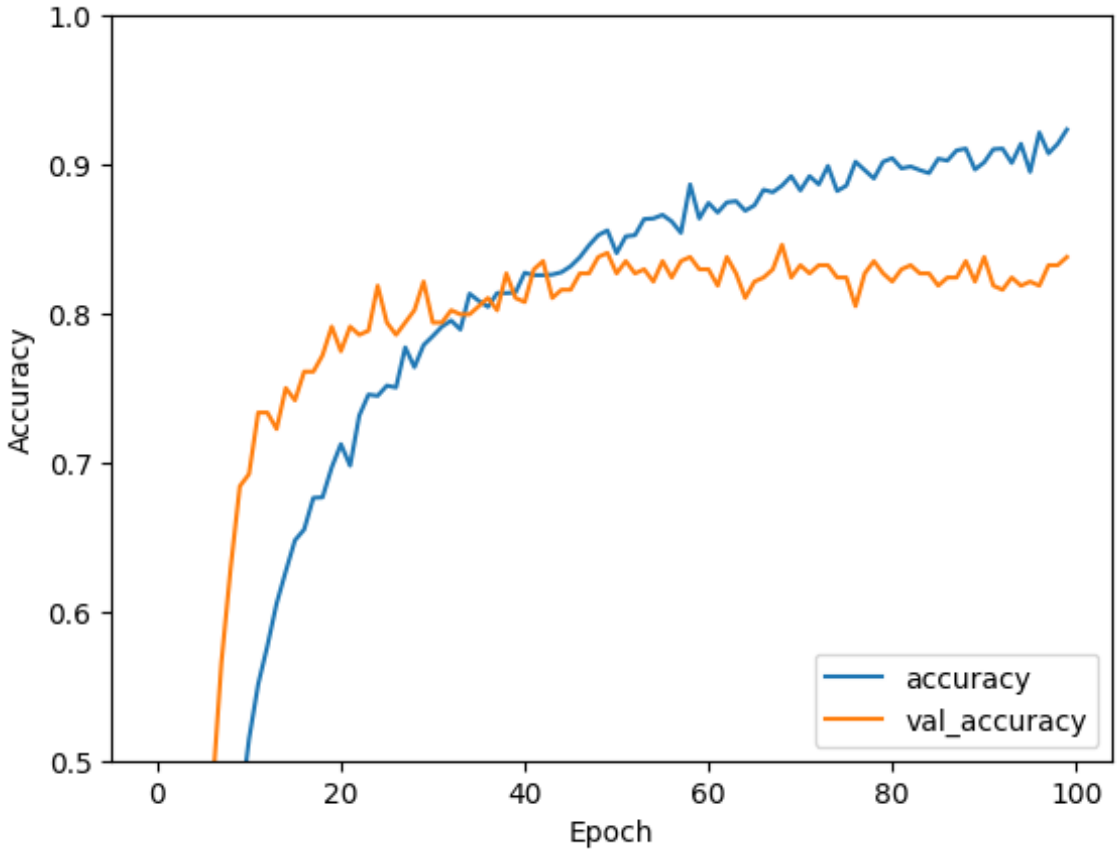
Ở giai đoạn đầu, tỷ lệ phân tách còn khá thấp (khoảng 58%) do nhiều nguyên nhân như:

- Over - segmentation: Tỷ lệ này còn khá cao do chưa được xác thực bằng ANN.
- Missed - segmentation: do các nét cao, thấp các chữ cái làm chồng lên những nét nối. Ta cần giải quyết bằng cách cải thiện ở các bước preprocessing (skew correction, slant correction, ...).

6.2 Module Character Classification

```
39/39 [==============================] - 6s 101ms/step - loss: 4.1178 - accuracy: 0.0188 - val_loss: 4.0825 - val_accuracy: 0.0192
Epoch 2/100
39/39 [==============================] - 3s 89ms/step - loss: 4.0466 - accuracy: 0.0196 - val_loss: 4.0162 - val_accuracy: 0.0220
Epoch 3/100
39/39 [==============================] - 3s 88ms/step - loss: 3.9883 - accuracy: 0.0252 - val_loss: 3.8980 - val_accuracy: 0.0742
Epoch 4/100
39/39 [==============================] - 3s 82ms/step - loss: 3.7562 - accuracy: 0.0653 - val_loss: 3.4960 - val_accuracy: 0.1456
Epoch 5/100
39/39 [==============================] - 3s 83ms/step - loss: 3.4506 - accuracy: 0.1090 - val_loss: 3.0638 - val_accuracy: 0.2143
Epoch 6/100
39/39 [==============================] - 4s 104ms/step - loss: 3.0875 - accuracy: 0.1851 - val_loss: 2.6756 - val_accuracy: 0.3214
Epoch 7/100
39/39 [==============================] - 3s 83ms/step - loss: 2.6606 - accuracy: 0.2664 - val_loss: 2.1328 - val_accuracy: 0.4780
Epoch 8/100
39/39 [==============================] - 4s 104ms/step - loss: 2.3444 - accuracy: 0.3421 - val_loss: 1.8484 - val_accuracy: 0.5659
Epoch 9/100
39/39 [==============================] - 5s 122ms/step - loss: 2.0714 - accuracy: 0.3994 - val_loss: 1.5882 - val_accuracy: 0.6291
Epoch 10/100
39/39 [==============================] - 4s 109ms/step - loss: 1.8424 - accuracy: 0.4647 - val_loss: 1.4079 - val_accuracy: 0.6841
Epoch 11/100
39/39 [==============================] - 4s 108ms/step - loss: 1.6732 - accuracy: 0.5144 - val_loss: 1.2610 - val_accuracy: 0.6923
Epoch 12/100
39/39 [==============================] - 4s 103ms/step - loss: 1.5234 - accuracy: 0.5513 - val_loss: 1.1416 - val_accuracy: 0.7335
Epoch 13/100
...
Epoch 99/100
39/39 [==============================] - 4s 96ms/step - loss: 0.2228 - accuracy: 0.9139 - val_loss: 0.6521 - val_accuracy: 0.8324
Epoch 100/100
39/39 [==============================] - 4s 90ms/step - loss: 0.2164 - accuracy: 0.9235 - val_loss: 0.6507 - val_accuracy: 0.8379
```

Hình 24: *Training result*



Hình 25: *Accuracy*

Tài liệu tham khảo

[1] R. Smith, "An Overview of the Tesseract OCR Engine", *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Curitiba, Brazil, 2007, pp. 629-633.

[2] Amjad Rehman, Tanzila Saba, "Performance analysis of character segmentation approach for cursive script recognition on benchmark database", Science Direct, 2011