

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



UIT

TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

MÔN HỌC: MÁY HỌC

ĐỀ TÀI

PHÂN LOẠI CHỮ VIẾT TAY TIẾNG VIỆT CÓ DẤU

Giảng viên hướng dẫn: Lê Đình Duy

Phạm Nguyễn Tường An

Sinh viên thực hiện: Đỗ Trọng Khánh - 19521676
Võ Phạm Duy Đức – 19521383
Trịnh Công Danh - 19521326

Lớp: CS114.L21.KHCL

CS114.L22.KHCL

Thành phố Hồ Chí Minh, ngày 10 tháng 7 năm 2021

MỤC LỤC

<i>I. Tổng quan</i>	- 2 -
1. Tổng quan về đề tài	- 2 -
2. Mô tả bài toán	- 2 -
<i>II. Các nghiên cứu trước</i>	- 3 -
<i>III. Xây dựng bộ dữ liệu.....</i>	- 5 -
1. Cách thu thập dữ liệu.....	- 5 -
2. Phân chia dữ liệu	- 10 -
<i>III. Xử lý dữ liệu và trích xuất đặc trưng.....</i>	- 11 -
1. Tiền Xử lý dữ liệu	- 11 -
2. Xử lý dữ liệu	- 12 -
2.1 Sử dụng kỹ thuật HOG để trích xuất đặc trưng của ảnh.....	- 12 -
2.2 Các bước tính HOG	- 13 -
<i>IV. Training và đánh giá các model</i>	- 16 -
1. Thực nghiệm trên Logistic Regression	- 16 -
2. Thực nghiệm trên Support vector machine (SVM).....	- 18 -
3. Thực nghiệm trên Multi layer Perceptron (MLPClassifier)	- 20 -
<i>V. Ứng dụng và hướng cải thiện</i>	- 22 -
1. Ứng dụng	- 22 -
2. Các hướng cải thiện bài toán	- 23 -
<i>VI. Tài liệu tham khảo</i>	- 24 -

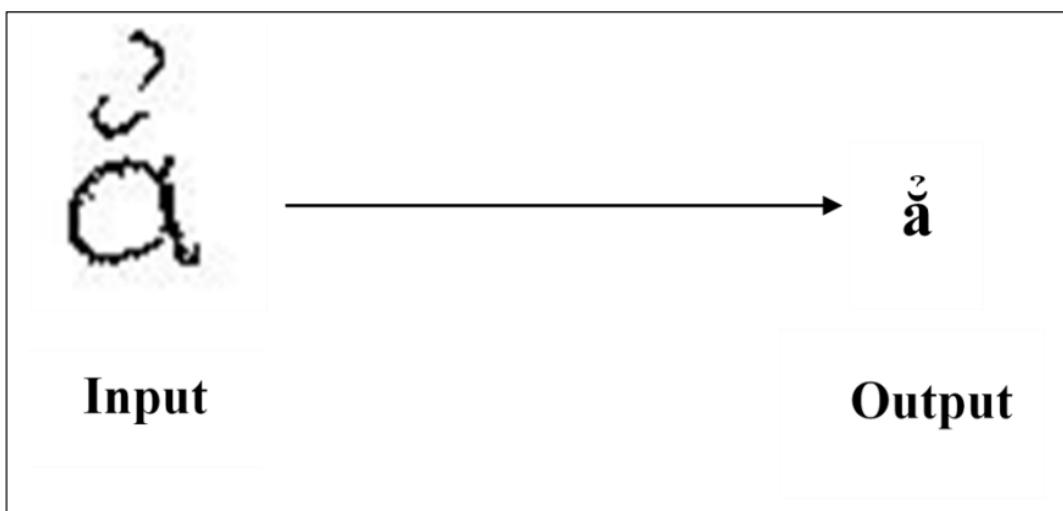
I. Tổng quan

1. Tổng quan về đề tài

Bài toán nhận diện chữ cái viết tay mang lại một lợi ích rất lớn cho con người trong các hoạt động thường ngày. Nhưng trước khi giải quyết được bài toán lớn đó thì chúng ta phải giải quyết được một bài toán nhỏ khác đó là phân loại chữ cái viết tay. Đã có khá nhiều nghiên cứu về đề tài phân biệt chữ viết tay nhưng nhận diện chữ cái Tiếng Việt viết tay hiện nay vẫn chưa nhiều. Đó là lý do nhóm quyết định thực hiện đề tài này.

2. Mô tả bài toán

- Bài toán này thuộc lớp bài toán phân loại, có tổng cộng 89 lớp đại diện cho 89 chữ cái tiếng Việt viết thường bao gồm cả các dấu phụ (sắc, huyền, hỏi, ngã, nặng).
- Đầu vào của bài toán là một tấm ảnh trong đó có chứa đúng một chữ cái tiếng Việt viết thường.
- Đầu ra là kết quả dự đoán chữ cái tương ứng với tấm ảnh đó.

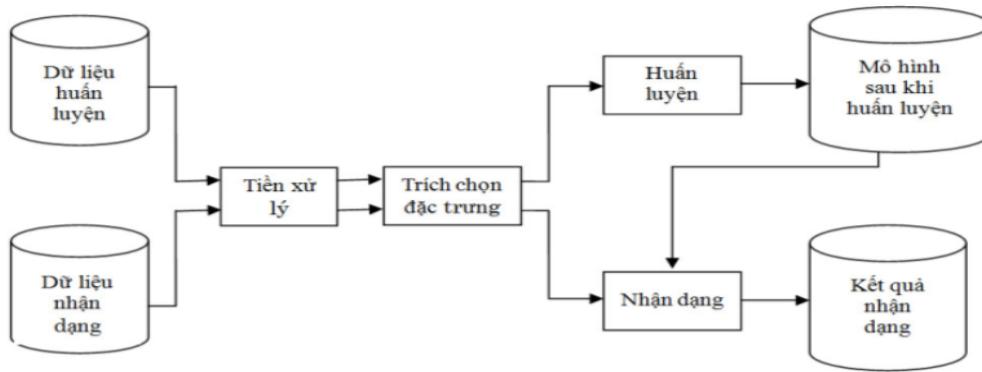


Hình 1: Mô phỏng bài toán

II. Các nghiên cứu trước

Nghiên cứu của các giảng viên trường đại học Duy Tân:

- **Link tham khảo:** https://tailieuxanh.com/vn/tlid1711621_nhan-dang-chu-viet-tay-roi-rac-tren-co-so-phuong-phap-may-vec-to-tua.html
- **Mô tả bộ dữ liệu:**
 - Bộ dữ liệu chuẩn MNIST: Bộ dữ liệu MNIST bao gồm 60.000 mẫu huấn luyện và 10.000 mẫu khác để nhận dạng, mỗi mẫu là một ảnh kích thước 28x28.
 - Bộ dữ liệu chữ viết tay tiếng Việt: Bộ dữ liệu chữ viết tay tiếng Việt (**VietData**) bao gồm 89 lớp chữ cái in hoa, mỗi lớp chọn ra 200 mẫu, như vậy bộ dữ liệu VietData tổng cộng 17.800 mẫu.
- **Phương pháp nghiên cứu:**
 - Tác giả xây dựng mô hình nhận dạng chữ viết tay rời rạc dựa trên phương pháp phân lớp SVM - Support Vector Machines. Công việc được thực hiện dựa trên 2 bước:
 - Bước 1: Xây dựng mô hình huấn luyện:
 - Tập dữ liệu huấn luyện sau khi qua các khâu tiền xử lý và trích chọn đặc trưng sẽ được đưa vào máy huấn luyện phân lớp SVM. Sau khi kết thúc quá trình huấn luyện, hệ thống sẽ lưu lại giá trị các tham số của hàm quyết định phân lớp để phục vụ cho việc nhận dạng sau này.
 - Bước 2: Phân lớp nhận dạng.
 - Dựa vào giá trị các tham số của hàm quyết định thu được ở Bước 1, một mẫu mới x sau khi đã qua các khâu tiền xử lý và trích chọn đặc trưng sẽ được đưa vào tính toán thông qua hàm quyết định để xác định lớp của mẫu x .



- Lựa chọn thuật toán huấn luyện phân lớp:
 - Trong phần cài đặt thực nghiệm, tác giả áp dụng thuật toán SMO để huấn luyện phân lớp SVM nhị phân, sử dụng và kế thừa một số chức năng của phần mềm mã nguồn mở LibSVM [1] để phát triển ứng dụng nhận dạng chữ viết tay rác.
- Thuật toán nhận dạng chữ viết tay:
 - Tác giả sử dụng hai chiến lược OVO và OVR để cài đặt thử nghiệm thuật toán nhận dạng đối với dữ liệu chữ viết tay rác.
- Kết quả thực nghiệm:
 - Trên bộ dữ liệu MNIST:
 - Mô hình SVM được sử dụng với hàm nhân RBF và các tham số $C = 10$ (tham số hàm phạt), Cache = 1000 (kích thước vùng nhớ để lưu trữ các vectơ tựa).

<i>Chiến lược</i>	<i>Số vectơ tựa</i>	<i>Thời gian huấn luyện</i>	<i>Thời gian Test</i>	<i>Độ chính xác</i>
OVR	8542	> 9 giờ	~ 3 phút	96,1%
OVO	31280	~ 2 giờ	~ 5 phút	97,2%

- Trên bộ dữ liệu chữ viết tay tiếng Việt:
 - Việc thực nghiệm trên dữ liệu chữ viết tay tiếng Việt được tiến hành theo phương thức thẩm định chéo (Cross-Validation). Bộ dữ liệu

VietData được chia thành k phần (ở đây k được chọn =10), sau đó sử dụng k-1 phần để huấn luyện và 1 phần còn lại để nhận dạng, quá trình được này được lặp đi lặp lại k lần.

<i>Chiến lược</i>	<i>Thời gian huấn luyện</i>	<i>Thời gian Test</i>	<i>Độ chính xác</i>
OVR	~ 49 phút	~ 2 phút	82.7%
OVO	~ 16 phút	~ 6 phút	83.6%

- **Nhận xét:**

- SVM là một phương pháp học máy tiên tiến có cơ sở toán học chặt chẽ và đạt độ chính xác phân lớp cao. Tuy nhiên, tốc độ phân lớp của SVM là chậm, tùy thuộc vào số lượng vectơ tựa thu được sau khi huấn luyện.
- Hạn chế khác của SVM là huấn luyện đòi hỏi không gian nhớ lớn, vì vậy việc huấn luyện đối với các bài toán có số lượng mẫu lớn sẽ gặp trở ngại trong vấn đề lưu trữ.
- Bản chất của phương pháp SVM là phân lớp nhị phân nên việc mở rộng khả năng của SVM để giải quyết các bài toán phân loại nhiều lớp là vấn đề khó và cần rất nhiều nghiên cứu.

III. Xây dựng bộ dữ liệu

1. Cách thu thập dữ liệu

- Dữ liệu được nhóm thu thập từ hơn 30 người tình nguyện ở thành phố Quảng Ngãi và Quảng Nam. Nhóm sẽ góp chung dữ liệu với nhóm bạn **Đặng Văn Minh** để làm *Trainning set* và *Validation set*.
- Nhóm sẽ thu thập thêm dữ liệu để làm tập *Test set* dành riêng cho nhóm để đánh giá độ chính xác của mô hình.

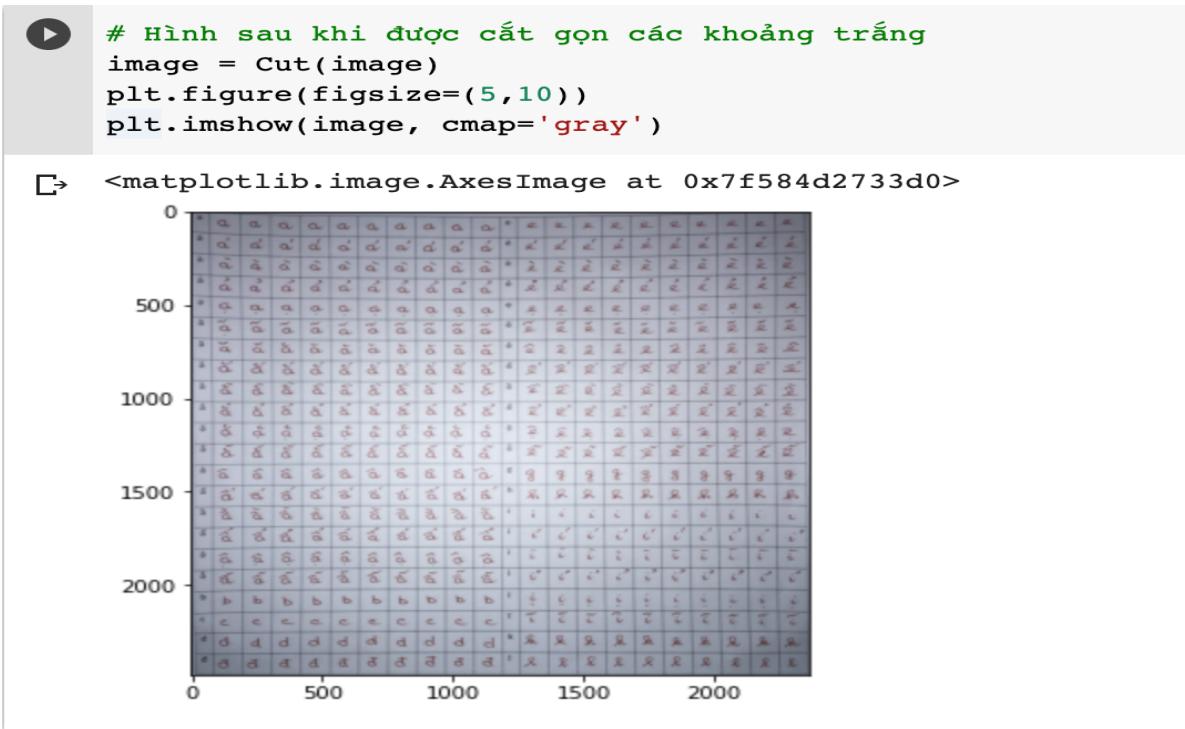
▼ Cắt gọn những khoảng trắng dư thừa

```
[ ] def Cut(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 57, 5)
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
    max = -1
    L = []
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        if cv2.contourArea(cnt) > max:
            x_max, y_max, w_max, h_max = x, y, w, h
            max = cv2.contourArea(cnt)
    table = image[y_max:y_max+h_max, x_max:x_max+w_max]
    return table
```

Hình 4: Code cắt các khoảng trắng dư thừa

Mục đích của đoạn code trên là tìm ra được hình chữ nhật lớn nhất có trong hình. Đầu tiên dùng hàm **cv2.findContours** để lấy được danh sách các contours có trong bức ảnh nhị phân. Mỗi countours được lưu trữ dưới dạng vector các điểm.

- Sau đó dùng hàm **cv2.boudingRect** để lấy được toạ độ các hình chữ nhật có trong hình. Trong đó x, y là toạ độ trọng tâm của hình chữ nhật và w, h là chiều dài và chiều rộng của hình chữ nhật đó.
- Sau khi tìm được toạ độ của hình chữ nhật lớn nhất, nhóm thực hiện cắt và được hình như bên dưới.

**Hình 5: Hình ảnh dữ sau khi được cắt****Bước 2:**

- Lọc từng ô chữ sau khi đã được cắt gọn ở *hình 5*.

```
cnts = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
for c in cnts:
    area = cv2.contourArea(c)
    if area < 1000:
        cv2.drawContours(thresh, [c], -1, (0,0,0), -1)

# Xoá các yếu tố gây nhiễu
vertical_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1,5))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, vertical_kernel, iterations=9)
horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,1))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, horizontal_kernel, iterations=4)

# Sắp xếp theo hàng trên xuống dưới và từng hàng từ trái sang phải
invert = 255 - thresh
cnts = cv2.findContours(invert, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
(cnts, _) = contours.sort_contours(cnts, method="top-to-bottom")

data_rows = []
row = []
for (i, c) in enumerate(cnts, 1):
    area = cv2.contourArea(c)
    if area < 50000:
        row.append(c)
        if i % 9 == 0:
            (cnts, _) = contours.sort_contours(row, method="left-to-right")
            data_rows.append(cnts)
            row = []
```

Hình 6: Code lọc từng ô chữ trong mẫu dữ liệu

- Sau khi đã có được vị trí của các hàng và vị trí của các từng ô trong mỗi hàng. Nhóm tiến hành duyệt từng ô chữ và lưu vào drive.

```
# Lắp lại từng ô
count = 1
for row in data_rows:
    for c in row:
        mask = np.zeros(image.shape, dtype=np.uint8)
        cv2.drawContours(mask, [c], -1, (255,255,255), -1)
        result = cv2.bitwise_and(image, mask)
        result[mask==0] = 255
        img_result = result
        try:
            final = Cut(img_result)
            final = cv2.cvtColor(final, cv2.COLOR_BGR2GRAY)
            final = final[10:110, 10:110]
            final = cv2.cvtColor(final, cv2.COLOR_GRAY2RGB)
            cv2.imwrite('/content/gdrive/My Drive/My Data/image_' + str(count) + '.JPG' , final)
            count += 1
        except:
            continue
```

Hình 7: Code mô tả duyệt từng ô và lưu vào drive**Bước 3:**

Sau khi thực hiện việc lọc và cắt từng tấm ảnh chỉ chứa 1 chữ cái riêng biệt sau đó phân loại các tấm ảnh về thành những thư mục riêng.

	image_1071.JPG	me
	image_1097.JPG	me
	image_1098.JPG	me
	image_1167.JPG	me
	image_1192.JPG	me
	image_1193.JPG	me
	image_1224.JPG	me
	image_1235.JPG	me
	image_1238.JPG	me
	image_1246.JPG	me
	image_1248.JPG	me
	image_1259.JPG	me
	image_1260.JPG	me

Phân loại và
đưa vào đúng
thư mục

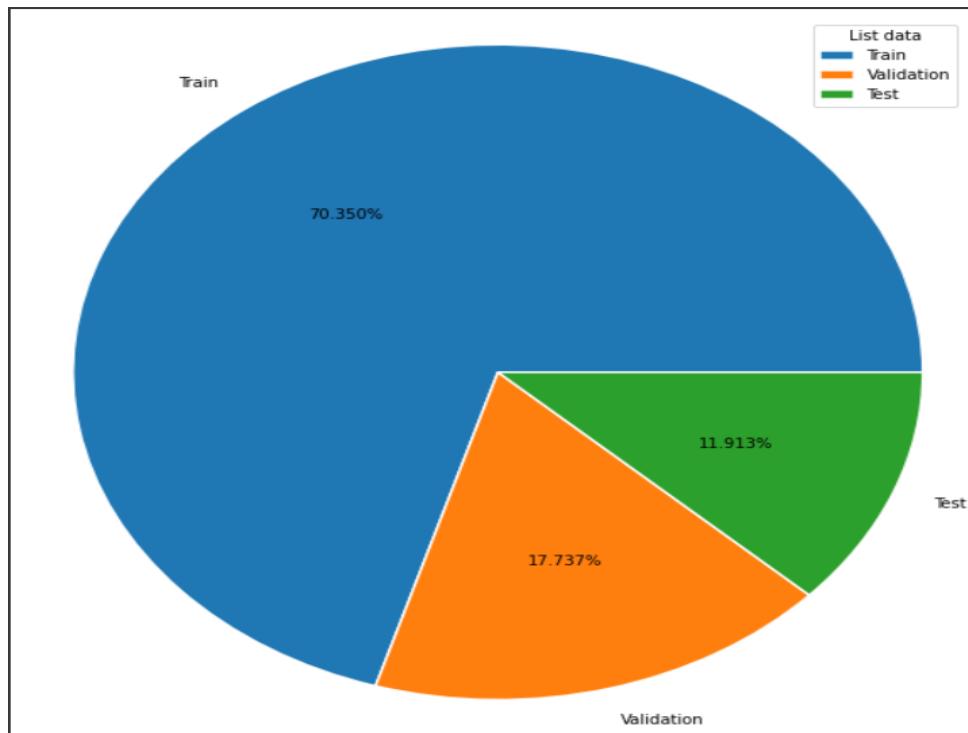


	a
	á
	à
	ă
	â
	ã
	å
	ä
	ë
	ö
	œ

- Khó khăn trong việc thu thập dữ liệu: Vì hình được chụp không được cố định vị trí giống nhau sau khi lọc và cắt ảnh, thứ tự các hình được cắt ra không như mong muốn, do đó việc gán nhãn cho từng hình tốn nhiều thời gian.

2. Phân chia dữ liệu

- Sau khi phân loại và gán nhãn cho dữ liệu, nhóm thống kê được tổng cộng **29.211** mẫu với **89** class, trung bình mỗi class sẽ có khoảng **328** tấm ảnh.
- Nhóm chia dữ liệu thu thập được thành 3 tập training set, validation set và test set với số lượng như sau:
 - **Training set** với **20.740** mẫu, các mẫu từ training set và validation set được thu thập từ nhiều người viết khác nhau và mỗi người chỉ được viết tờ thu thập dữ liệu tối đa là hai lần.
 - **Validation set** với **5.229** mẫu, dùng để đánh giá mô hình sau khi train. Các mẫu ở tập này không được dùng để huấn luyện mô hình.
 - **Test set** với **3.512** mẫu được thu thập riêng biệt với hai tập trên.



Hình 8: Biểu đồ pie-chart thể hiện sự phân bố của các tập data set

III. Xử lý dữ liệu và trích xuất đặc trưng

1. Tiền Xử lý dữ liệu

- Mỗi tấm ảnh trong tập train và tập validation đều được chuyển thành ảnh nhị phân (trắng đen) và xử lý nhiều.
- Cắt bớt các khoảng trắng dư thừa xung quanh chữ.

```
[ ] # Hàm cắt gọn ảnh bằng cách xác định các countours
def crop_images(img):
    blur = cv2.GaussianBlur(img,(7,7),0)
    thresh = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,7,7)
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
    x_min = 10**9
    x_max = 0
    y_min = 10**9
    y_max = 0

    for cnt in contours:
        x, y, width, height = cv2.boundingRect(cnt)
        if cv2.contourArea(cnt) > 0:
            x_min = min(x_min, x)
            y_min = min(y_min, y)

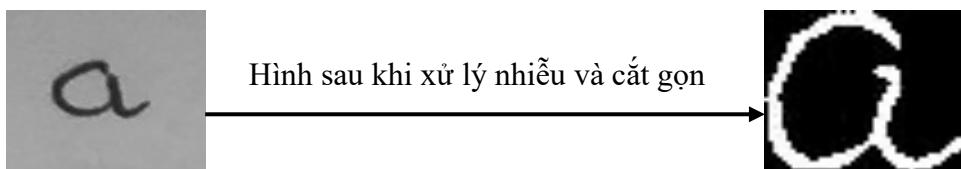
            if x + width > x_max:
                x_max = x + width

            if y + height > y_max:
                y_max = y + height

    table = thresh[y_min: y_max, x_min:x_max]
    return table
```

Hình 9: Code cắt bớt các khoảng trắng dư thừa xung quanh chữ

- Sau khi nhóm cắt bớt các khoảng trắng xung quanh chữ, nhóm đã thống kê được min của width và height là **(4, 15)**.
- Tiếp theo nhóm tiến hành thử resize về kích thước **(4, 15)** của một vài hình thì nhận thấy hình không còn được rõ. Do đó nhóm quyết định xoá các hình có **width < 14**.



- Sau đó resize về khích thước 14x16. Nhóm quyết định resize về kích thước 14x16 là vì những chữ y, h khi nhóm thử về kích thước 16x16 thì hình không còn được rõ và bị hư.

2. Xử lý dữ liệu

2.1 Sử dụng kỹ thuật HOG để trích xuất đặc trưng của ảnh

a. Giới thiệu về thuật toán HOG

- Thuật toán **HOG** (Histogram of oriented gradient) là một thuật toán tuy có điểm nhung cũng rất đơn giản và hiệu quả trong việc xử lý ảnh. Thuật toán này sẽ tạo ra các bộ mô tả đặc trưng (feature descriptor) với mục đích phát hiện vật thể (object detection). Từ một bức ảnh ta sẽ lấy ra 2 ma trận quan trọng giúp lưu thông tin ảnh đó là độ lớn **gradient** (gradient magnitude) và phương của **gradient** (gradient orientation). Bằng cách kết hợp 2 thông tin này vào một biểu đồ phân phối **histogram**, cuối cùng chúng ta sẽ thu được một **vector** mang những đặc trưng **HOG** (các số thực) đại diện cho một ảnh.

b. Tính toán gradient

- Trong xử lý ảnh, độ dốc (tức **gradient**) đang nói đến ở đây chính là độ dốc về mức sáng. Hay nói cách khác chính là sự thay đổi các giá trị **pixel** trong ảnh.
- Tính toán **gradient** theo trực **Ox(Gx)** và **Oy (Gy)** bằng hàm có sẵn trong **OpenCV**.

```

1 # Calculate gradient gx, gy
2 gx = cv2.Sobel(gray, cv2.CV_32F, dx=0, dy=1, ksize=3)
3 gy = cv2.Sobel(gray, cv2.CV_32F, dx=1, dy=0, ksize=3)

```

- Kích thước của **Gx** và **Gy** đều bằng ảnh gốc. Như vậy mỗi **pixel** trên **Gx** sẽ ứng với 1 **pixel** ở tọa độ tương ứng trên **Gy**. Vì vậy **giá trị độ lớn gradient và phương gradient** được tạo ra từ 2 **gradient Gx** và **Gy** sẽ xác định được độ lớn và phương tại mỗi **pixel**.

- Công thức tính độ lớn của gradient:

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

- Công thức tính phương của gradient:

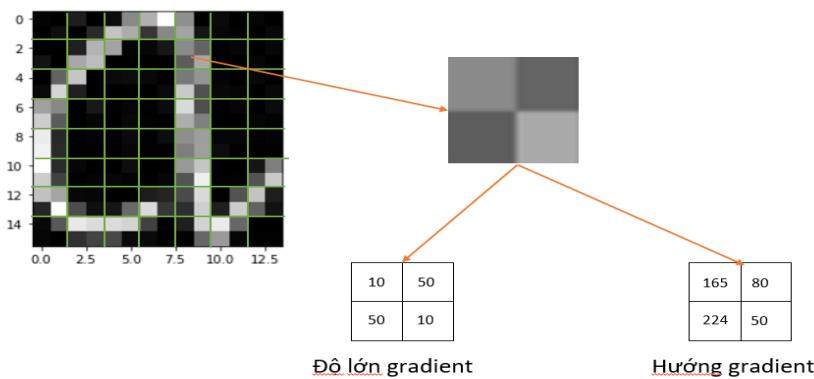
$$\theta = \text{arctan}\left(\frac{G_y}{G_x}\right)$$

- Sử dụng hàm có sẵn trong **OpenCV** để tính độ lớn và phương của gradient:

```
1 | # Python Calculate gradient magnitude and direction ( in degrees )
2 | mag, angle = cv2.cartToPolar(gx, gy, angleInDegrees=True)
```

2.2 Các bước tính HOG

- Ta nhận thấy đặc trưng của mỗi bức ảnh được biểu diễn thông qua 2 thông số đó là mức độ thay đổi cường độ màu sắc (ma trận gradient magnitude) và hướng thay đổi cường độ màu sắc (ma trận gradient direction). Do đó chúng ta cần tạo ra được một bộ mô tả (feature descriptor) sao cho biến đổi bức ảnh thành một véc tơ mà thể hiện được cả 2 thông tin này.
- Để làm được như vậy thì hình ảnh được chia thành một lưới ô vuông, mỗi ô vuông có kích thước 2 x 2 pixels. Như vậy ta sẽ có 4 giá trị hướng và 4 giá trị độ lớn ứng với mỗi ô như ma trận bên dưới.

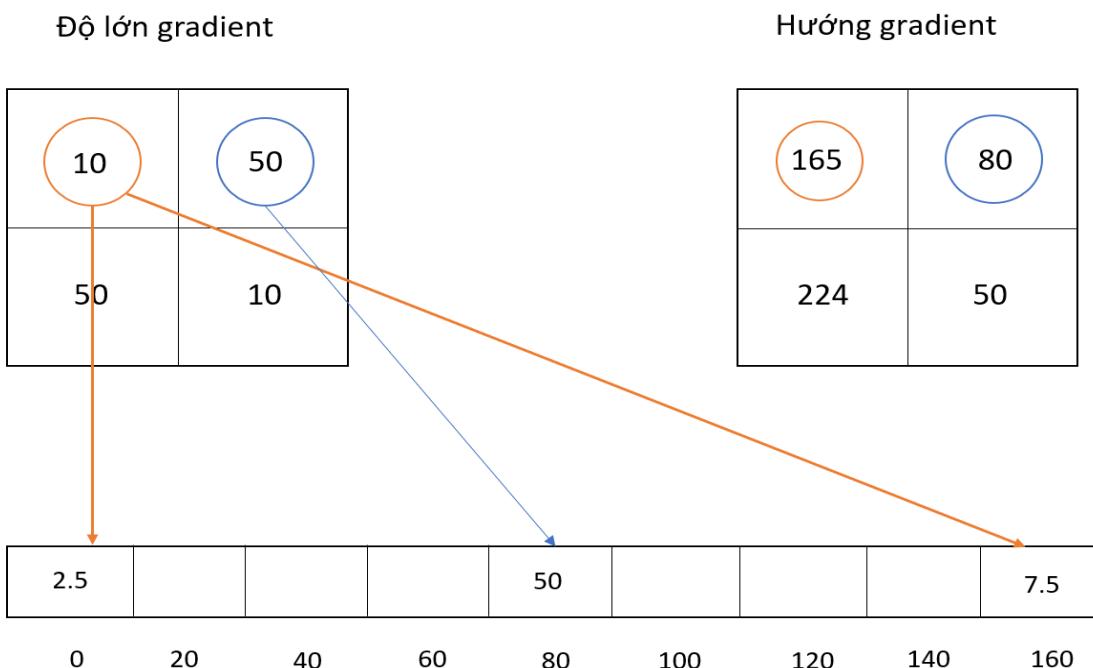


- Sau khi tìm được **hướng** và **độ lớn** của các khối nhỏ 2×2 **pixels**, ta sẽ vote giá trị độ lớn của mỗi **pixel** vào khoảng hướng có cùng vị trí tọa độ vào 1 trong 9 bin sau khi xác định được hướng của **pixel** thuộc **pin** tương ứng.
- Trong trường hợp **độ lớn của phương gradients** không chia hết cho đầu mút (nghĩa là không chia hết cho 0, 20, 40...), ta sẽ sử dụng công thức **linear interpolation** để phân chia độ lớn gradient về 2 biên liền kề.
 - Công thức **linear interpolation**:
 - Giá trị phương gradient bằng x tương ứng với độ lớn gradient y có cùng vị trí tọa độ (x thuộc $[x_0, x_1]$). Khi đó:
 - Tại **bin** thứ $l - 1$:

$$x_{l-1} = \frac{(x_1 - x)}{x_1 - x_0} * y$$

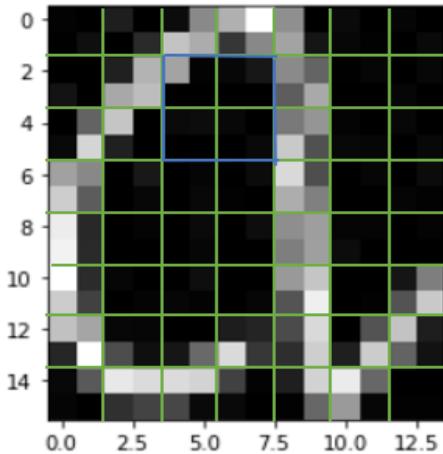
- Tại **bin** thứ l :

$$x_l = \frac{(x - x_0)}{x_1 - x_0} * y$$



Hình 9: Biểu diễn phương và độ lớn các pixels vào các bin

- Tiếp theo, ta sẽ chuẩn hóa **vector histogram** theo block gồm 4 ô, mỗi ô **2x2 pixel**.



- Quá trình chuẩn hóa sẽ được thực hiện chuẩn hóa một **block** có kích thước **2 x 2** đầu tiên, mỗi block chứa 4 ô, mỗi ô có kích thước **2 x 2 pixel**, như vậy chúng ta sẽ có 4 **vector histogram** kích thước **1 x 9** và khi ghép nối tiếp nhau sẽ được một **vector** có 36 phần tử. Sau đó ta chuẩn hóa theo công thức bên dưới.

$$\text{L2-norm, } \mathbf{v} \rightarrow \mathbf{v}/\sqrt{\|\mathbf{v}\|_2^2 + \epsilon^2};$$

L2 – norm: sau chuẩn hóa, độ dài của vector bằng 1.

- Sau đó dịch block đó sang 1 ô và ta sẽ thực hiện chuẩn hóa cho block đó. Đầu vào là một ảnh có kích thước 14x16, áp dụng thuật toán tính HOG với kích thước cells là 2x2 chúng ta sẽ thu được một lưới ô vuông có kích thước $14/2 = 7$ ô theo chiều rộng và $16/2 = 8$ ô theo chiều dài. Sau khi khôi block có kích thước 2x2 trải qua 6 bước theo chiều rộng và 7 bước theo chiều và ghép nối tiếp các vector có 36 phần tử lại với nhau ta sẽ có một vector có $36 \times 6 \times 7 = 1512$ phần tử. Đây là vector **HOG** đại diện cho toàn bộ hình ảnh.

IV. Training và đánh giá các model

Với bài toán phân loại chữ viết tay tiếng việt, nhóm em sẽ dùng các model sau để huấn luyện:

- Logistic Regression
- Support vector machine (SVM)
- Multi layer Perceptron (MLPClassifier)

Để đánh giá kết quả, chúng em sẽ sử dụng cách tính accuracy, accuracy càng cao thì mô hình càng tốt.

1. Thực nghiệm trên Logistic Regression

```
[16] from sklearn.linear_model import LogisticRegression
    model_LG = LogisticRegression(C = 0.1, max_iter=1000)
    model_LG.fit(X_train, Y_train)
    y_pred_vali = model_LG.predict(X_vali)
    y_pred_test = model_LG.predict(X_test)
```

- Các thông số:
 - C: Thông số dùng để ngăn chặn sự overfit của mô hình, C càng nhỏ thì mô hình sẽ ít bị overfit (nhưng quá nhỏ sẽ bị underfit).
 - Max_iter: Số lần lặp tối đa để thuật toán có thể tìm ra local minimum của bài toán.
- Thời gian train: 2m 23s.
Thời gian test (tập validation và tập test): 39s.
- Đánh giá kết quả:
 - Kết quả trên tập validation:

accuracy			0.67	5229
macro avg	0.66	0.65	0.65	5229
weighted avg	0.67	0.67	0.66	5229

➔ Accuracy: 67%

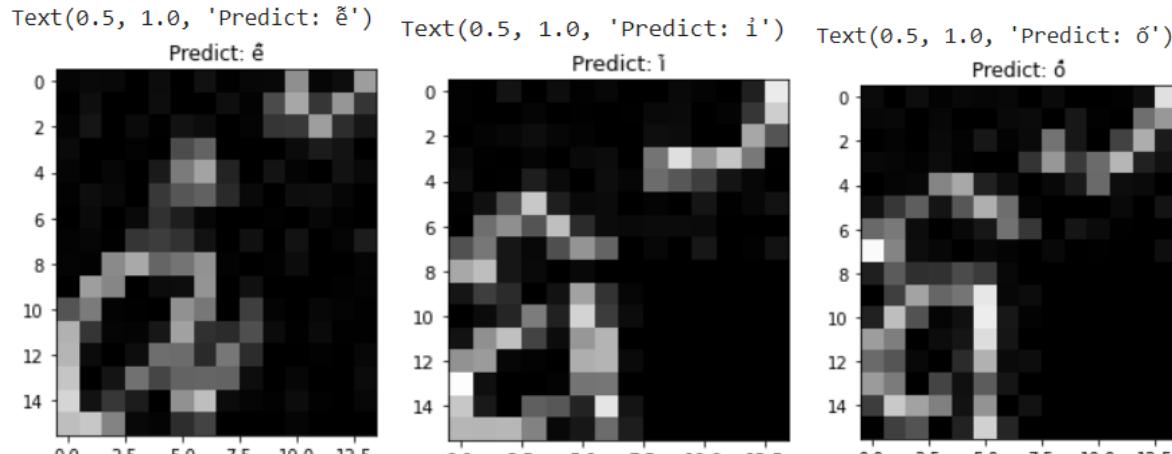
- Kết quả trên tập test:

accuracy		0.63	3512
macro avg	0.65	0.63	3512
weighted avg	0.65	0.63	3512

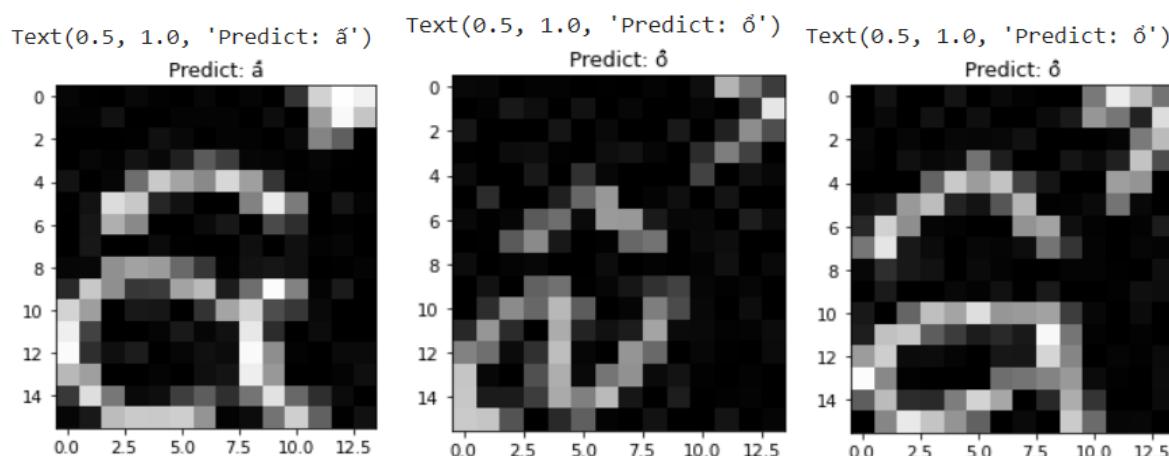
➔ Accuracy: 63%

- Nhận xét:

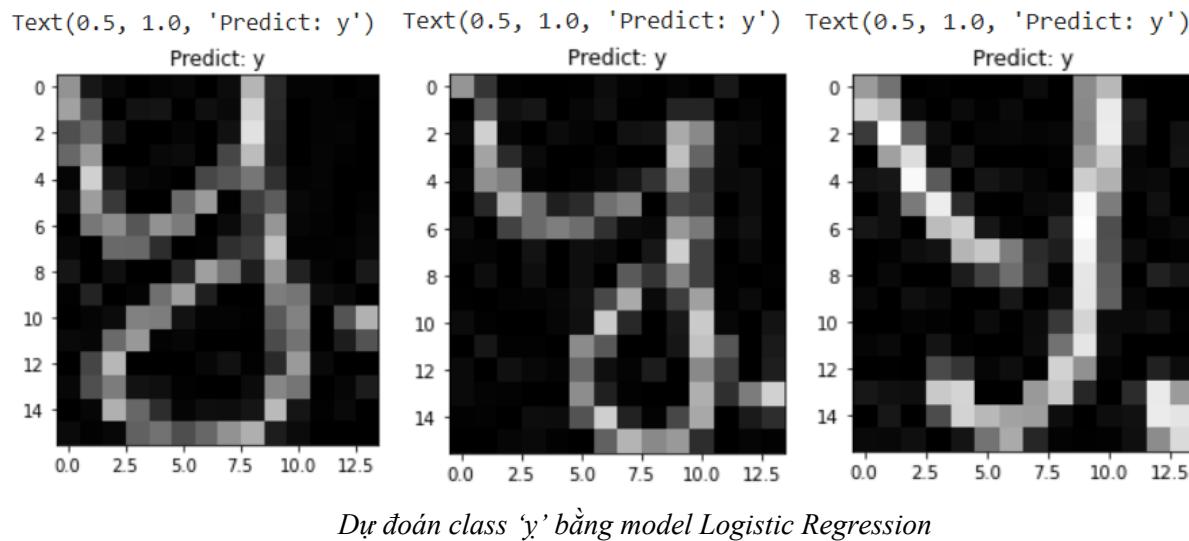
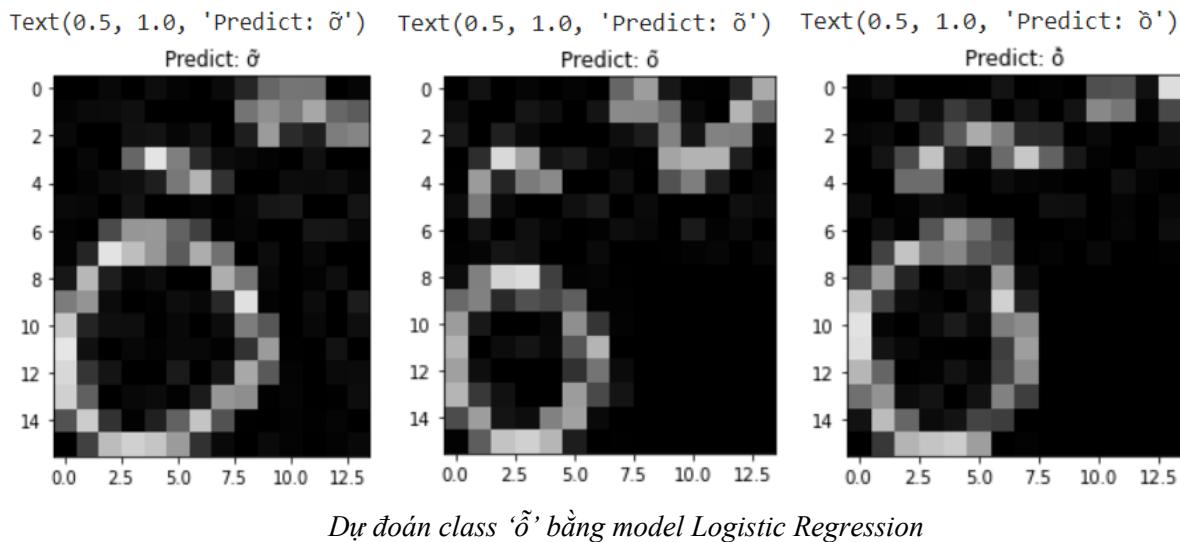
- Các mẫu dữ liệu mà model dự đoán sai (dựa trên điểm f1-score): ā(39%), ā̂(34%), õ(36%), y (27%).
- Ảnh minh họa cho các trường hợp class bị dự đoán sai:



Dự đoán class ‘ā’ bằng model Logistic Regression



Dự đoán class ‘ā’ bằng model Logistic Regression



2. Thực nghiệm trên Support vector machine (SVM)

- Nhóm em thực hiện training sử dụng model SVM với 3 kernel: linear, polynomial và rbf. Kết quả cho thấy accuracy trên tập validation và tập test nếu sử dụng kernel rbf là cao nhất.

```
[ ] from sklearn.svm import SVC
model_SVM_rbf = SVC(C=1000, kernel = 'rbf', gamma=0.001)
model_SVM_rbf.fit(X_train, Y_train)
```

- Các thông số:

- C (Regularization): Tham số điều chỉnh việc có nên bỏ qua các điểm dữ liệu bất thường trong quá trình tối ưu mô hình SVM. Nếu tham số này có giá trị lớn, quá trình tối ưu sẽ chọn một siêu phẳng sao cho siêu phẳng này phân cách tất cả các điểm dữ liệu một cách tốt nhất, từ đó khoảng cách giữa siêu phẳng tới các điểm dữ liệu của các lớp sẽ có giá trị nhỏ (small-margin).
- Gamma: Tham số gamma xác định việc sử dụng bao nhiêu điểm dữ liệu cho việc xây dựng siêu phẳng phân cách. Với giá trị gamma nhỏ, các điểm dữ liệu nằm xa đường phân cách sẽ được sử dụng trong việc tính toán đường phân cách. Ngược lại, với giá trị gamma lớn, chỉ những điểm nằm gần đường phân cách mới được sử dụng để tính toán.

- Thời gian train: 18m 12s

Thời gian test (tập validation và tập test): 8m 16s

- Kết quả trên tập validation:

accuracy			0.70	5229
macro avg	0.69	0.69	0.69	5229
weighted avg	0.70	0.70	0.70	5229

➔ Accuracy: 70%

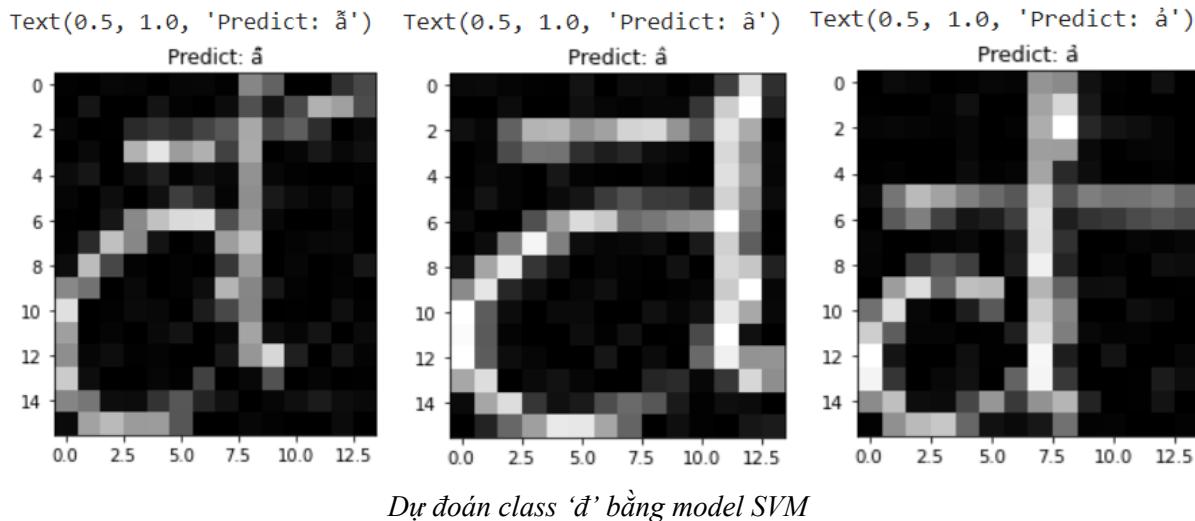
- Kết quả trên tập test:

accuracy			0.67	3512
macro avg	0.67	0.67	0.66	3512
weighted avg	0.68	0.67	0.66	3512

➔ Accuracy: 67%

- Nhận xét:

- Các mẫu dữ liệu mà model dự đoán sai (dựa trên điểm f1-score): đ (36%)
- Ảnh minh họa cho các trường hợp class bị dự đoán sai:



3. Thực nghiệm trên Multi layer Perceptron (MLPClassifier)

```
▶ model_MLP = MLPClassifier(hidden_layer_sizes=(1000, 1000, 1000), max_iter=500)
    model_MLP.fit(X_train, Y_train)
    y_pred_vali = model_MLP.predict(X_vali)
    y_pred_test = model_MLP.predict(X_test)
```

- Các thông số:
 - Hidden_layer_sizes: Số lượng hidden layer và số neuron của mỗi layer.
- Thời gian train: 8m 41s
- Thời gian test (tập validation và tập test): 45s

- Kết quả trên tập validation:

accuracy			0.71	5229
macro avg	0.70	0.70	0.70	5229
weighted avg	0.71	0.71	0.71	5229

➔ Accuracy: 71%

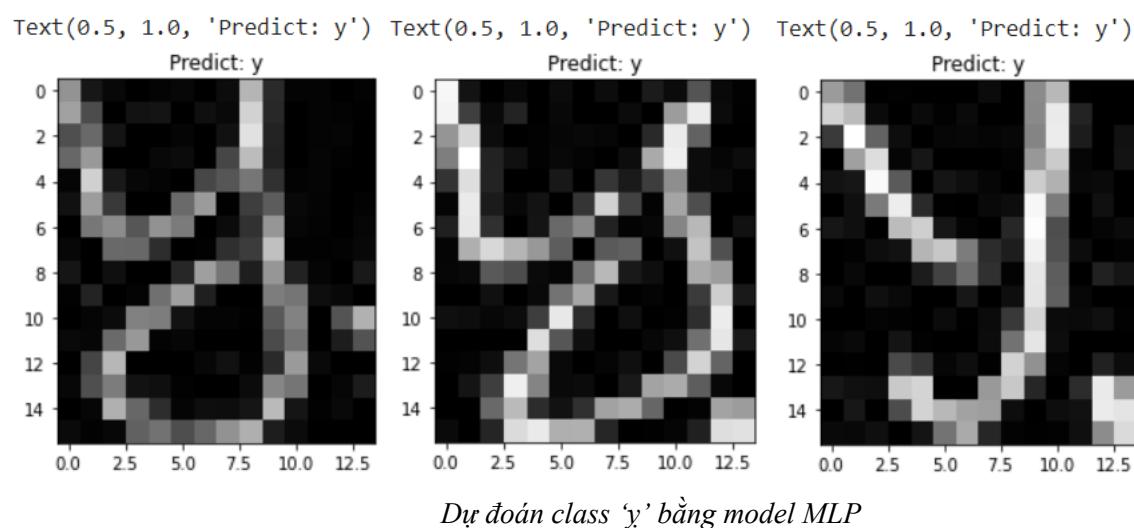
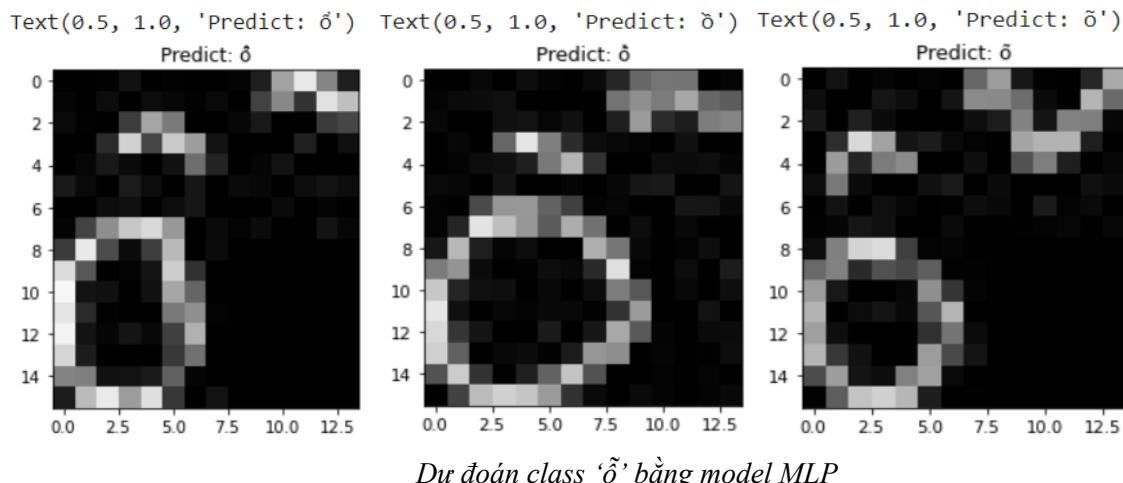
- Kết quả trên tập test:

accuracy			0.68	3512
macro avg	0.69	0.68	0.68	3512
weighted avg	0.69	0.68	0.68	3512

➔ Accuracy: 68%

- Nhận xét:

- Các mẫu dữ liệu mà model dự đoán sai (dựa trên điểm f1-score): ô(34%), y(25%).
- Lý do: Lúc cắt và lọc ảnh, dấu nặng của những ảnh chữ y bị mất nên dữ liệu của chữ y ít hơn so với các nhóm khác (cụ thể là chữ y). Hơn nữa, lúc trích xuất đặc trưng HOG, đặc trưng dấu nặng của chữ y bị mất khá nhiều do quá trình resize ảnh và xóa các yếu tố gây nhiễu.
- Ảnh minh họa cho các trường hợp class bị dự đoán sai:



*** Nhận xét chung cho các model:**

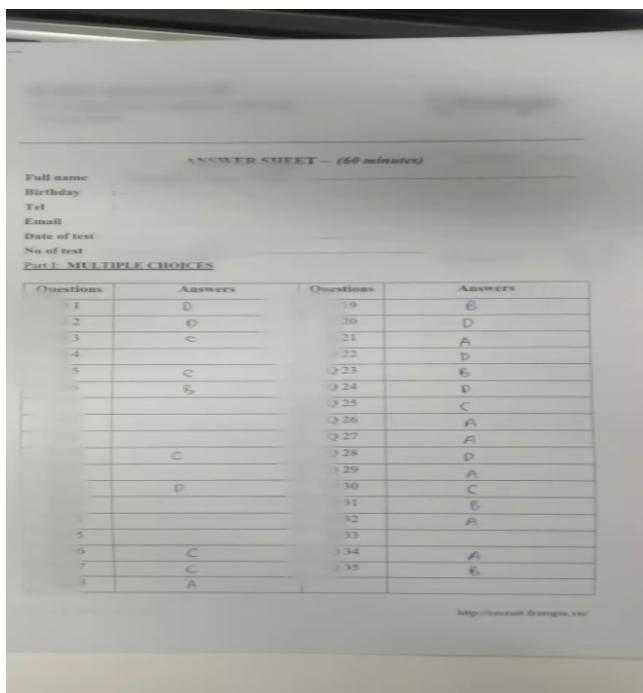
- Các mô hình đều cho kết quả tốt (đều trên 60%), điểm accuracy cao nhất đạt đến 0.71%.
- Mô hình MLP - Classifier cho kết quả tốt nhất trên cả 2 tập validation và test.
- Class ‘y’ bị dự đoán sang class ‘y’ và ‘ÿ’ khá nhiều
- Model SVM dự đoán class ‘đ’ cho kết quả thấp (36%) nhưng model MLP lại cho kết quả tốt hơn (61%). Tương tự, model logistic regression dự đoán class ‘â’ cho kết quả thấp (34%) nhưng model MLP lại cho kết quả tốt hơn (62%)

	precision	recall	f1-score	support	
đ	0.62	0.25	0.36	40	(Model SVM)
đ	0.86	0.47	0.61	40	(Model MLP)
â	0.58	0.24	0.34	45	(Model logistic regression)
â	0.75	0.53	0.62	45	(Model MLP)

V. Ứng dụng và hướng cải thiện

1. Ứng dụng

- Một trong những ứng dụng thực tế của bài toán nhận dạng chữ viết tay tiếng Việt là chấm bài thi trắc nghiệm. Thi trắc nghiệm đã và đang trở thành xu hướng bởi tính khách quan của nó (không phụ thuộc vào người chấm). Tuy nhiên việc chấm số lượng lớn bài trắc nghiệm không phải là một vấn đề đơn giản vì không phải ở đâu cũng có máy chấm trắc nghiệm tự động để khiếu công việc trở nên nhanh chóng nên việc áp dụng mô hình nhận dạng chữ viết tay tiếng Việt vào việc chấm bài thi trắc nghiệm là một ứng dụng rất tốt và cần thiết trong việc giảng dạy và học tập.



Tờ phiếu trắc nghiệm thông thường

- Đối với bài toán thực tế dạng này, bài toán ta sẽ được chia thành 2 bài toán nhỏ hơn:
 - **Bài toán xác định (Detection):** Thực hiện việc tiền xử lý ảnh trên (Áp dụng các phương pháp xử lý ảnh: Resize, làm mờ/mịn, Threshold,...) để xác định vị trí các chữ cái trong hình để làm đầu vào cho bài toán Nhận dạng tiếp theo.
 - **Bài toán nhận dạng (Classification):** Sử dụng các model nhận dạng chữ viết tay để xác định các ảnh đầu vào đó là những chữ cái nào sau đó đến công đoạn so đáp án, thống kê kết quả,...

2. Các hướng cải thiện bài toán

- Accuracy các model nhóm em chọn cho kết quả chung là khá cao nhưng có nhiều class vẫn dự đoán sai, nên phải học và tìm hiểu thêm nhiều model tốt hơn nữa.
- Tìm hiểu thêm các phương pháp rút trích đặc trưng khác ngoài rút trích đặc trưng HOG.
- Cải thiện cách thu thập dữ liệu và tiền xử lý ảnh vì có nhiều ảnh bị nhiễu khá nhiều
- Tăng thêm kích thước dữ liệu cho bài toán.

- Phát triển bài toán nhận diện chữ viết tay Tiếng Việt thành bài toán nhận diện nhiều chữ viết tay Tiếng Việt liền kề nhau, dẫn tới khả năng đọc được nguyên cả một đoạn văn bản và chuyển sang dạng text.

VI. Tài liệu tham khảo

- [1] https://learnopencv.com/histogram-of-oriented-gradients/?fbclid=IwAR3T6jgM_Zx49uGF-kbyAsefEL2pK3GIXYmnaBCTIRhjBoGNxOu3i-H87tA
- [2] <https://minhng.info/tutorials/histograms-of-oriented-gradients.html>
- [3] <https://phamdinhkhanh.github.io/2019/11/22/HOG.html#211-t%C3%ADnh-to%C3%A1n-gradient>
- [4] <https://www.noron.vn/post/gioi-thieu-ve-support-vector-machine-trong-machine-learning-40dxtjcmrdye>
- [5] <https://machinelearningcoban.com/2017/04/09/smv/>
- [6] <https://scikit-learn.org/stable/modules/svm.html#svm-kernels>
- [7] <https://cafedev.vn/tu-hoc-ml-dieu-chinh-sieu-tham-so-svm-bang-gridsearchcv-ml/>
- [8] <https://blog.vietnamlab.vn/danh-gia-model-cua-machine-learning/>
- [9] <https://stackoverflow.com/questions/59182827/how-to-get-the-cells-of-a-sudoku-grid-with-opencv>
- [10] <https://thorphanh.github.io/blog/2018/04/06/Nh%E1%BA%ADn-d%E1%BA%A1ng-ch%E1%BB%AF-s%E1%BB%91-vi%E1%BA%BFt-tay-v%E1%BB%9Bi-sklearn-v%C3%A0-openCV/>