

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

# BÁO CÁO ĐỒ ÁN

MÔN HỌC: MÁY HỌC (MACHINE LEARNING)

ĐỀ TÀI: PHÂN LOẠI CHỮ VIẾT TAY TIẾNG VIỆT CÓ DẤU

**Giảng viên hướng dẫn:** Lê Đình Duy

Phạm Nguyễn Trường An

**Sinh viên thực hiện:** Đỗ Trọng Khánh – 19521676

Võ Phạm Duy Đức – 19521383

Trịnh Công Danh – 19521326

**Lớp:**

CS114.L22.KHCL

CS114.L21.KHCL

# NỘI DUNG BÁO CÁO

01

GIỚI THIỆU ĐỀ TÀI

02

CÁC NGHIÊN CỨU  
TRƯỚC

03

MÔ TẢ BỘ DỮ LIỆU

04

XỬ LÝ DỮ LIỆU  
VÀ TRÍCH XUẤT  
ĐẶC TRƯNG

05

TRAINING VÀ ĐÁNH GIÁ  
CÁC MODEL

06

ỨNG DỤNG VÀ  
HƯỚNG CẢI THIỆN

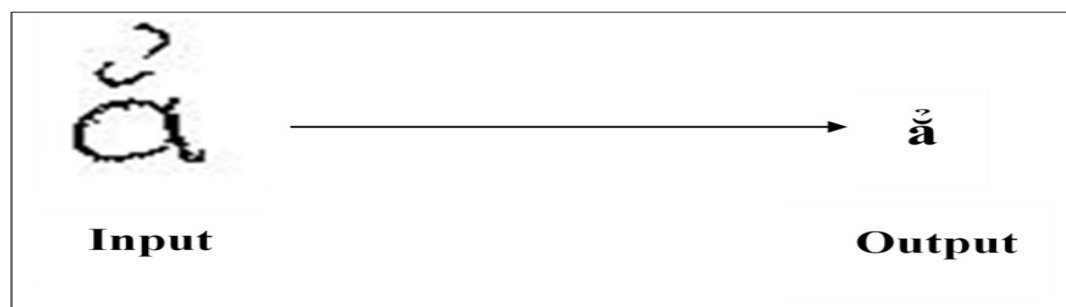
# 1. GIỚI THIỆU ĐỀ TÀI

## Tổng quan về đề tài

Phân loại chữ cái viết tay là đề tài nghiên cứu khá phổ biến. Nhưng chữ cái Tiếng Việt hiện nay vẫn chưa được nghiên cứu nhiều. Đó là lí do nhóm quyết định thực hiện đề tài này.

## Mô tả bài toán

- Bài toán 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 chứa một chữ cái tiếng Việt viết thường.
- Đầu ra là kết quả dự đoán chữ cái tương ứng với tấm ảnh đó.



## 2. CÁC NGHIÊN CỨU TRƯỚC

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

#### 1. Mô tả bộ dữ liệu

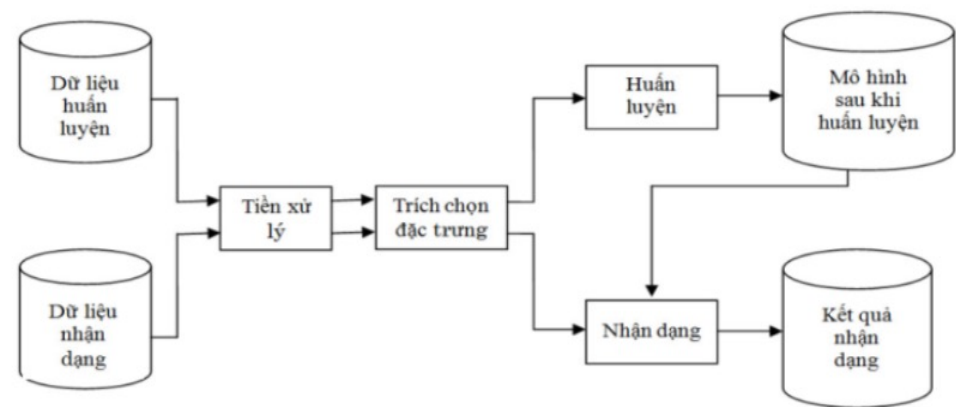
- Bộ dữ liệu chuẩn MNIST: Bộ dữ liệu MNIST bao gồm 60.000 mẫu huấn luyện và 10.000 mẫu khác để nhận dạng, mỗi mẫu là một ảnh kích thước 28 x 28.

- 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.

#### 2. 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
- + Bước 2: Phân lớp nhận dạng



## 2. CÁC NGHIÊN CỨU TRƯỚC

### 3. 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 vector tựa).

Chiến lược	Số vector tựa	Thời gian huấn luyện	Thời gian Test	Độ chính xác
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).

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

### 4. 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.
- Hạn chế khác của SVM là huấn luyện đòi hỏi không gian nhớ lớn
- 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.

### 3. MÔ TẢ BỘ DỮ LIỆU

- Dữ liệu được thu thập từ hơn 30 người tình nguyện. Nhóm sẽ góp chung dữ liệu với nhóm bạn **Đặng Văn Minh** để làm *Training set* và *Validation set*. Sau đó sẽ thu thập thêm dữ liệu để làm tập *Test set* dành riêng cho nhóm để đánh giá độ chính xác của mô hình.
- Nhóm sẽ chuẩn bị những mẫu giấy A4 và sẽ nhờ người viết tay những con chữ vào các ô giấy.

[illegible]

## Mẫu dữ liệu

[illegible]

Mẫu dữ liệu đã được viết

# 3. MÔ TẢ BỘ DỮ LIỆU

## Các bước thực hiện

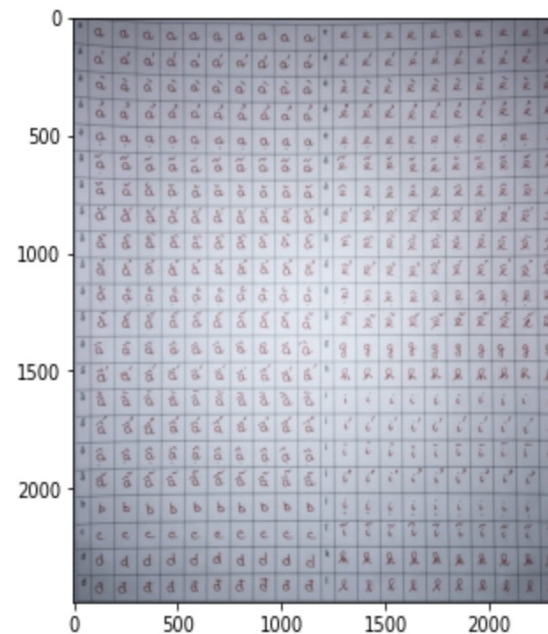
**Bước 1:** Sử dụng **cv2 edge detection** để cắt gọn những khoảng trắng dư thừa để thuận tiện trong việc lọc các ô chữ.

### ✦ 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 sau khi được cắt gọn các khoảng trắng  
image = Cut(image)  
plt.figure(figsize=(5,10))  
plt.imshow(image, cmap='gray')

☐ <matplotlib.image.AxesImage at 0x7f584d2733d0>





### 3. MÔ TẢ BỘ DỮ LIỆU

**Bước 2:** Lọc từng ô chữ sau khi đã được cắt gọn.

```
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 = []
```
















### 3. MÔ TẢ BỘ DỮ LIỆU

**Bước 3:** 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.














```
# Lặp 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
```

### 3. MÔ TẢ BỘ DỮ LIỆU

**Bước 4:** Sau khi lọc và cắt từng tấm ảnh chỉ chứa 1 chữ cái riêng biệt thì 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
	ã
	ä
	å
	ä
	å
	ä
	å
	ä
	å
	ä

### 3. MÔ TẢ BỘ DỮ LIỆU

- Sau khi phân loại và gán nhãn cho dữ liệu, có 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** với **20.740** mẫu, các mẫu từ training set và validation set được thu thập từ nhiều người viết khác nhau.

**Validation set** với **5.229** mẫu, ***không*** được dùng để huấn luyện mô hình mà dùng để đánh giá mô hình sau khi train.

**Test set** với **3.512** mẫu được thu thập riêng biệt với hai tập trên.

## 4. XỬ LÝ DỮ LIỆU VÀ TRÍCH XUẤT ĐẶC TRƯNG

### Tiền Xử lý dữ liệu

- Các ảnh trong tập train và tập validation đều được chuyển thành ảnh nhị phân (trắng đen) và xử lý nhiễu.
- Cắt bớt các khoảng trắng dư thừa xung quanh chữ và thống kê được min của width và height là **(4, 15)**.
- Tiếp theo tiến hành thử resize về kích thước **(4, 15)** một vài hình, nhận thấy hình không còn được rõ. Do đó nhóm quyết định xoá các hình có **width < 14** và **height = 15**.

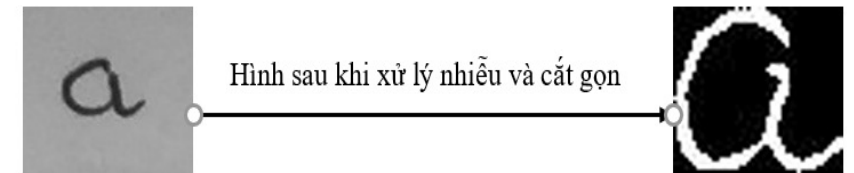
```
[ ] # Hàm cắt gọn ảnh bằng cách xác định các countours
def crop_images(img):
    blur = cv2.GaussianBlur(img,(7,7),0)
    thresh = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,7,7)
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]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



- Sau đó resize về kích thước 14x16 để tránh một số chữ như y, h không được rõ và bị hư.

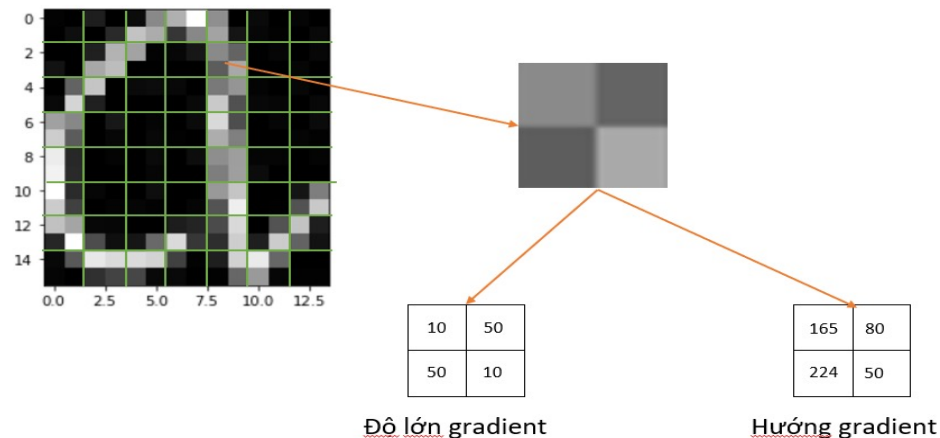
## 4. XỬ LÝ DỮ LIỆU VÀ TRÍCH XUẤT ĐẶC TRƯNG

### Xử lý dữ liệu

- Sử dụng phương pháp trích xuất đặc trưng **HOG** (Histogram of oriented gradient)
- Hình ảnh được chia thành các ô nhỏ nối tiếp nhau, mỗi ô có kích thước 2 x 2 pixel. Sẽ có 4 giá trị hướng và 4 giá trị độ lớn ứng với mỗi ô
- Mỗi ô vuông trong ảnh có kích thước 2x2 pixel và mỗi ô sẽ có 4 giá trị hướng và 4 giá trị độ lớn như sau:

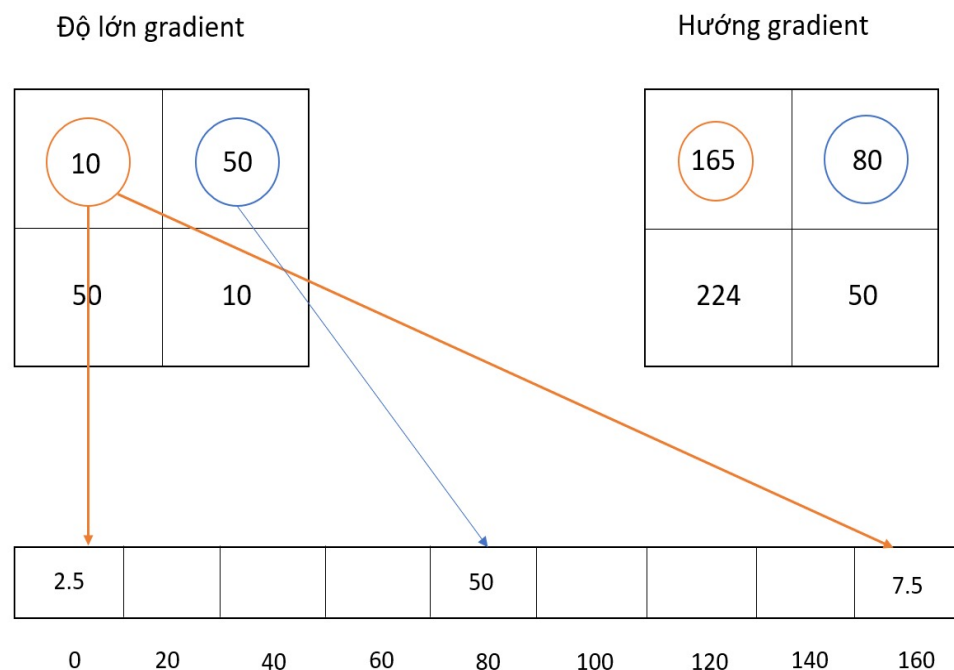
```
1 # Calculate gradient gx, gy
2 gx = cv2.Sobel(gray, cv2.CV_32F, dx=0, dy=1, ksize=3)
3 gy = cv2.Sobel(gray, cv2.CV_32F, dx=1, dy=0, ksize=3)

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



## 4. XỬ LÝ DỮ LIỆU VÀ TRÍCH XUẤT ĐẶC TRƯNG

- Tiếp theo, thực hiện 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.



### 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  $[x_0, x_1]$ ).

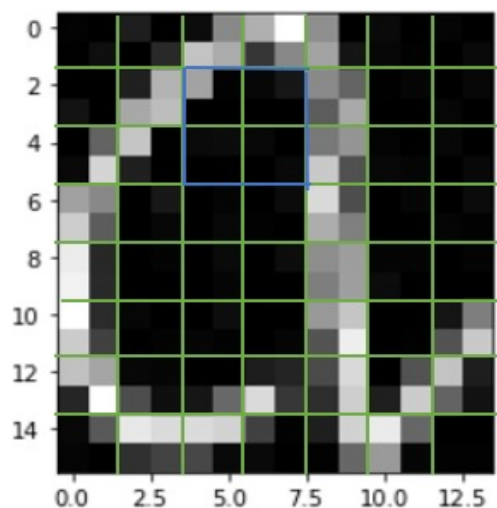
Khi đó:

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

- Tại bin thứ  $l$ :

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

## 4. XỬ LÝ DỮ LIỆU VÀ TRÍCH XUẤT ĐẶC TRƯNG



- Chuẩn hóa vector histogram theo block gồm 4 ô, mỗi ô 2 x 2 pixel.

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

- 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 khối 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 x 6 x 7 = 1512 phần tử. Đây là vector **HOG** đại diện cho toàn bộ hình ảnh



## 5. 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)
```

- Thời gian train: 2m 23s.
- Thời gian test (tập validation và tập test): 39s.

## 5. TRAINING VÀ ĐÁNH GIÁ CÁC MODEL

- 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	0.63	3512
weighted avg	0.65	0.63	0.63	3512

→ Accuracy: 63%

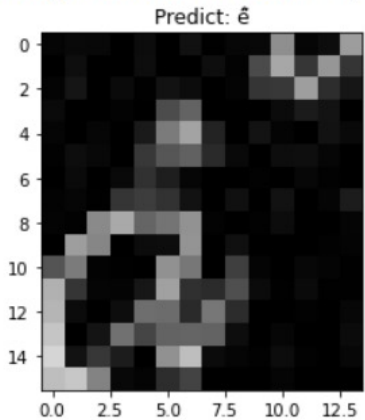
- Nhận xét:

Các mẫu dữ liệu mà model dự đoán sai (dựa trên điểm f1-score): ã(39%), ả(34%), ố(36%), y (27%).

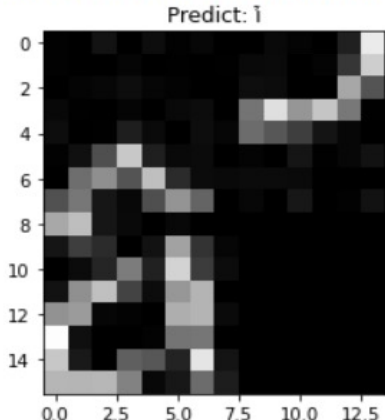
Ảnh minh họa cho các trường hợp class bị dự đoán sai:

## 5. TRAINING VÀ ĐÁNH GIÁ CÁC MODEL

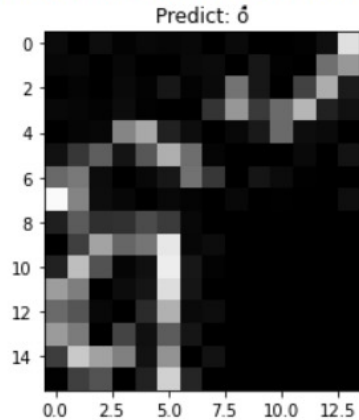
Text(0.5, 1.0, 'Predict: ě')



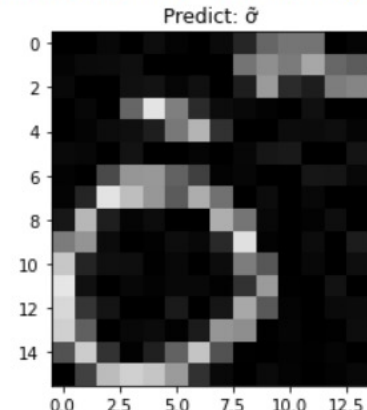
Text(0.5, 1.0, 'Predict: ĭ')



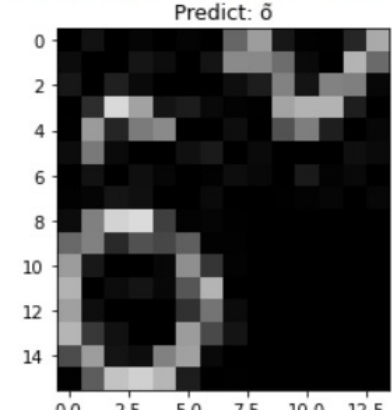
Text(0.5, 1.0, 'Predict: ő')



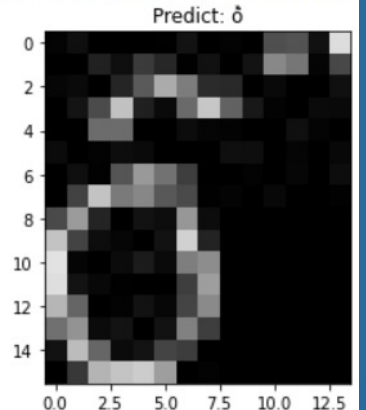
Text(0.5, 1.0, 'Predict: őr')



Text(0.5, 1.0, 'Predict: ős')



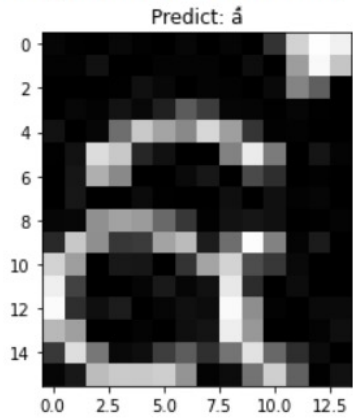
Text(0.5, 1.0, 'Predict: öd')



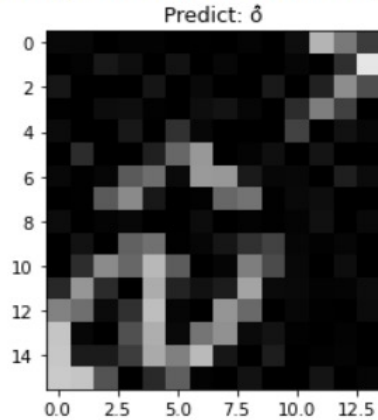
*Dự đoán class 'ă' bằng model Logistic Regression*

*Dự đoán class 'õ' bằng model Logistic Regression*

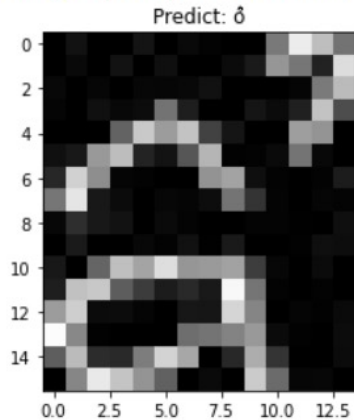
Text(0.5, 1.0, 'Predict: ă')



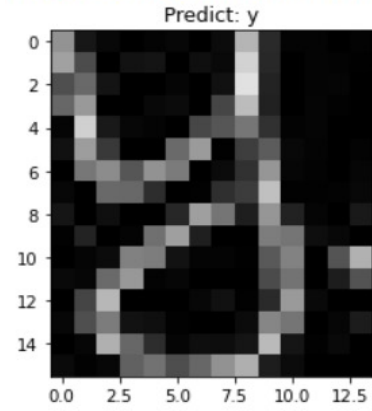
Text(0.5, 1.0, 'Predict: ỗ')



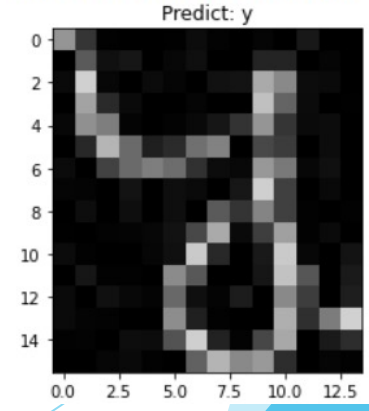
Text(0.5, 1.0, 'Predict: ỗ')



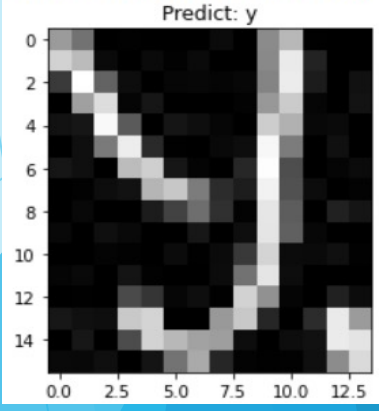
Text(0.5, 1.0, 'Predict: y')



Text(0.5, 1.0, 'Predict: y')



Text(0.5, 1.0, 'Predict: y')



*Dự đoán class 'ă' bằng model Logistic Regression*

*Dự đoán class 'y' bằng model Logistic Regression*

# 5. TRAINING VÀ ĐÁNH GIÁ CÁC MODEL

## 2. Thực nghiệm trên Support vector machine (SVM)

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

- Đánh giá kết quả

+ 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%

- Thời gian train: 18m 12s.

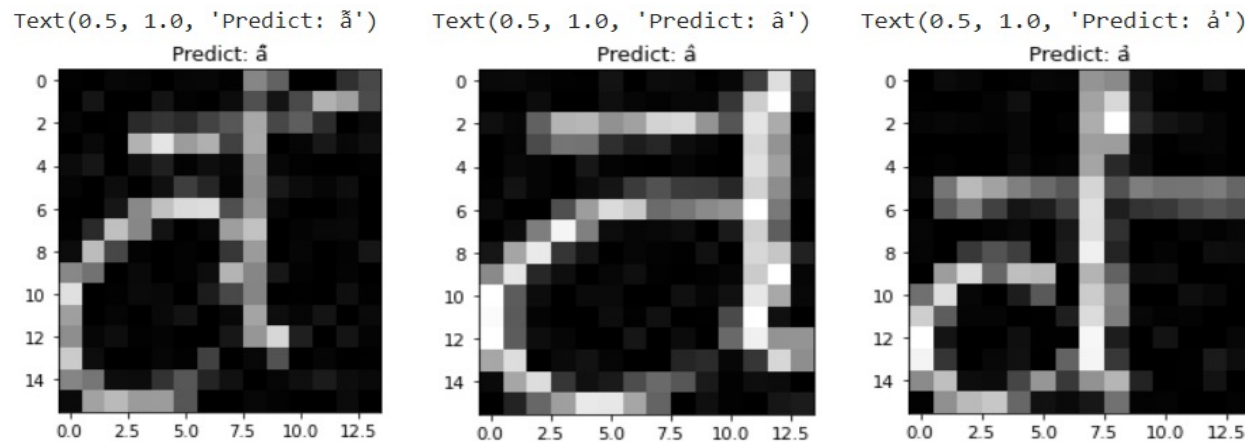
- Thời gian test (tập validation và tập test): 8m 16s.

## 5. TRAINING VÀ ĐÁNH GIÁ CÁC MODEL

- 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:



*Dự đoán class 'đ' bằng model Support vector machine (SVM)*

### 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)
```

- Thời gian train: 8m 41s.

- Thời gian test (tập validation và tập test): 45s.

## 5. TRAINING VÀ ĐÁNH GIÁ CÁC MODEL

- Đánh giá kết quả

+ 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%

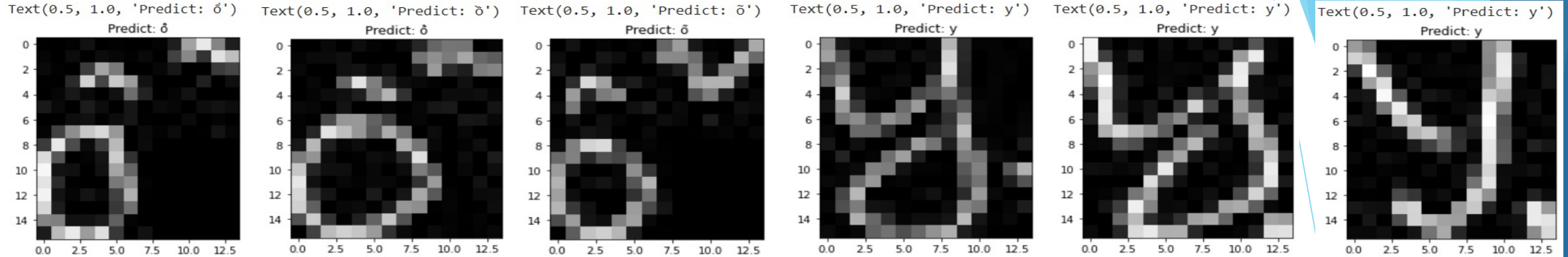
- Nhận xét:

Các mẫu dữ liệu mà model dự đoán sai (dựa trên điểm f1-score): Ồ (34%), y(25%).

Ảnh minh họa cho các trường hợp class bị dự đoán sai:



## 5. TRAINING VÀ ĐÁNH GIÁ CÁC MODEL



Dự đoán class 'ố' bằng model MLP

Dự đoán class 'y' bằng model MLP

### \* Nhận xét chung cho các model:

- Các mô hình đều cho kết quả tốt (đều trên 60%), điểm accuracy cao nhất đạt đến 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 'ỳ' khá nhiều.

	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)
ă	0.58	0.24	0.34	45	(Model logistic regression)
ă	0.75	0.53	0.62	45	(Model MLP)



## 6. HƯỚNG CẢI THIỆN VÀ ỨNG DỤNG

- 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.

- Ứng dụng: ứng dụng thực tế của bài toán nhận dạng chữ viết tay tiếng Việt là chấm bài thi trắc nghiệm.

- 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).
- + Bài toán nhận dạng (Classification).

Questions	Answers	Questions	Answers
1	D	19	B
2	D	20	D
3	C	21	A
4	D	22	D
5	C	23	B
6	B	24	D
7		25	C
8		26	A
9		27	A
10	C	28	D
11	D	29	A
12		30	C
13		31	B
14		32	A
15		33	A
16	C	34	A
17	C	35	B
18	A		

Tờ phiếu trắc nghiệm thông thường

XIN CẢM ƠN!