

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.K21.KHCL

# NỘI DUNG BÁO CÁO

01 | GIỚI THIỆU ĐỀ TÀI

02 | MÔ TẢ BỘ DỮ LIỆU

03 | XỬ LÝ DỮ LIỆU

04 | ĐÁNH GIÁ KẾT QUẢ

05 | KẾT LUẬN

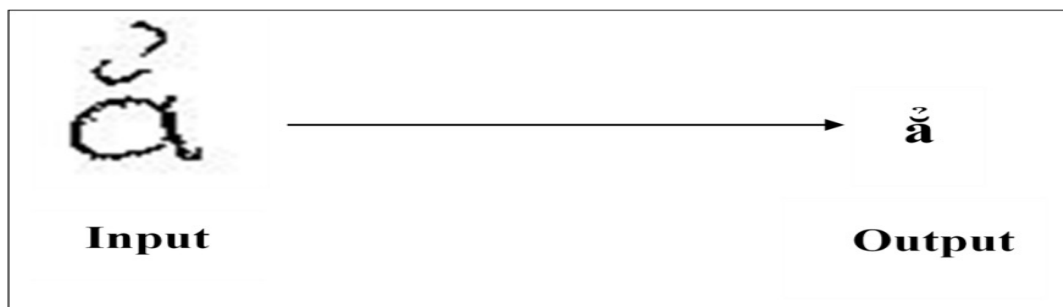
# 1. GIỚI THIỆU ĐỀ TÀI

## 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. 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.

a								e						
á								é						
à								è						
ǎ								ě						
ạ								ẹ						
ä								ë						
â								ê						
ă								ē						
ā								ē						
Ḃ								ĕ						
Ḅ								ê						
Ḇ								ē						
Ḉ								ĕ						
Ḑ								ĕ						
Ḓ								ĕ						
Ḕ								ĕ						
Ḗ								ĕ						
Ḙ								ĕ						
Ḛ								ĕ						
Ḝ								ĕ						
Ḟ								ĕ						
Ḡ								ĕ						
Ḣ								ĕ						
Ḥ								ĕ						
Ḧ								ĕ						
Ḩ								ĕ						
Ḫ								ĕ						
Ḭ								ĕ						
Ḱ								ĕ						
Ḳ								ĕ						
Ḵ								ĕ						
Ḷ								ĕ						
Ḹ								ĕ						
Ṁ								ĕ						
Ṃ								ĕ						
Ṅ								ĕ						
Ṇ								ĕ						
Ṉ								ĕ						
Ṱ								ĕ						
Ṳ								ĕ						
Ṵ								ĕ						
Ṷ								ĕ						
Ṹ								ĕ						
Ṻ								ĕ						
Ṽ								ĕ						
ṽ								ĕ						
Ẁ								ĕ						
Ẃ								ĕ						
Ẅ								ĕ						
Ẇ								ĕ						
Ẹ								ĕ						
Ẹ̃								ĕ						
Ẹ̇								ĕ						
Ẹ̈́								ĕ						
Ẹ̊								ĕ						
Ẹ̋								ĕ						
Ẹ̌								ĕ						
Ẹ̍								ĕ						
Ẹ̎								ĕ						
Ẹ̏								ĕ						
Ẹ̐								ĕ						
Ẹ̑								ĕ						
Ẹ̒								ĕ						
Ẹ̓								ĕ						
Ẹ̔								ĕ						
Ẹ̕								ĕ						
Ẹ̖								ĕ						
Ẹ̗								ĕ						
Ẹ̘								ĕ						
Ẹ̙								ĕ						
Ẹ̚								ĕ						
Ẹ̛								ĕ						
Ẹ̜								ĕ						
Ẹ̝								ĕ						
Ẹ̞								ĕ						
Ẹ̟								ĕ						
Ẹ̠								ĕ						
Ẹ̡								ĕ						
Ẹ̢								ĕ						
Ẹ̣								ĕ						
Ẹ̤								ĕ						
Ẹ̥								ĕ						
Ẹ̦								ĕ						
Ȩ̣								ĕ						
Ę̣								ĕ						
Ẹ̩								ĕ						
Ẹ̪								ĕ						
Ẹ̫								ĕ						
Ẹ̬								ĕ						
Ẹ̭								ĕ						
Ẹ̮								ĕ						
Ẹ̯								ĕ						
Ẹ̰								ĕ						
Ẹ̱								ĕ						
Ẹ̲								ĕ						
Ẹ̳								ĕ						
Ẹ̴								ĕ						
Ẹ̵								ĕ						
Ẹ̶								ĕ						
Ẹ̷								ĕ						
Ẹ̸								ĕ						
Ẹ̹								ĕ						
Ẹ̺								ĕ						
Ẹ̻								ĕ						
Ẹ̼								ĕ						
Ẹ̽								ĕ						
Ẹ̾								ĕ						
Ẹ̿								ĕ						
Ẹ̀								ĕ						
Ẹ́								ĕ						
Ệ								ĕ						
Ẹ̃								ĕ						
Ẹ̄								ĕ						
Ẹ̅								ĕ						
Ẹ̆								ĕ						
Ẹ̇								ĕ						
Ẹ̈								ĕ						
Ẹ̉								ĕ						
Ẹ̊								ĕ						
Ẹ̋								ĕ						
Ẹ̌								ĕ						
Ẹ̍								ĕ						
Ẹ̎								ĕ						
Ẹ̏								ĕ						
Ẹ̐								ĕ						
Ẹ̑								ĕ						
Ẹ̒								ĕ						
Ẹ̓								ĕ						
Ẹ̔								ĕ						
Ẹ̕								ĕ						
Ẹ̖								ĕ						
Ẹ̗								ĕ						
Ẹ̘								ĕ						
Ẹ̙								ĕ						
Ẹ̚								ĕ						
Ẹ̛								ĕ						
Ẹ̜								ĕ						
Ẹ̝								ĕ						
Ẹ̞								ĕ						
Ẹ̟								ĕ						
Ẹ̠								ĕ						
Ẹ̡								ĕ						
Ẹ̢								ĕ						
Ẹ̣								ĕ						
Ẹ̤								ĕ						
Ẹ̥								ĕ						
Ẹ̦								ĕ						
Ȩ̣								ĕ						
Ę̣								ĕ						
Ẹ̩								ĕ						
Ẹ̪								ĕ						
Ẹ̫								ĕ						
Ẹ̬								ĕ						
Ẹ̭								ĕ						
Ẹ̮								ĕ						
Ẹ̯								ĕ						
Ẹ̰								ĕ						
Ẹ̱								ĕ						
Ẹ̲								ĕ						
Ẹ̳								ĕ						
Ẹ̴								ĕ						
Ẹ̵								ĕ						
Ẹ̶								ĕ						
Ẹ̷								ĕ						
Ẹ̸								ĕ						
Ẹ̹								ĕ						
Ẹ̺								ĕ						
Ẹ̻								ĕ						
Ẹ̼								ĕ						
Ẹ̽								ĕ						
Ẹ̾								ĕ						
Ẹ̿								ĕ						
Ẹ̀								ĕ						
Ẹ́														

## Mẫu dữ liệu

[illegible]

Mẫu dữ liệu đã được viết

## 2. MÔ TẢ BỘ DỮ LIỆU

Các bước thực hiện

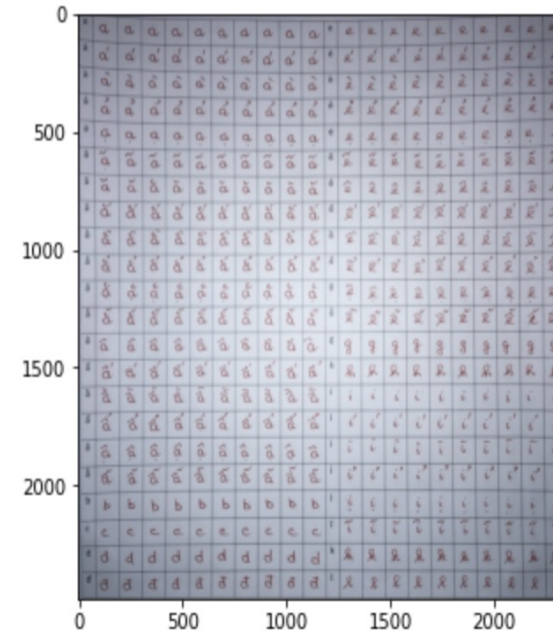
**Bước 1:** Sử dụng **cv2 edge detection** để cắt gọn những khoảng trắng dư thừa để thuận tiện trong việc lọc các ô chữ.

### ▼ 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>





## 2. 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 = []
```

## 2. MÔ TẢ BỘ DỮ LIỆU














**Bước 2:** 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
```
















## 2. MÔ TẢ BỘ DỮ LIỆU

**Bước 3:** 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

## 2. 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.

### 3. XỬ LÝ DỮ LIỆU

#### 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 xóa các hình có **width < 14**.

```
[ ] # Hàm cắt gọn ảnh bằng cách xác định các contours
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

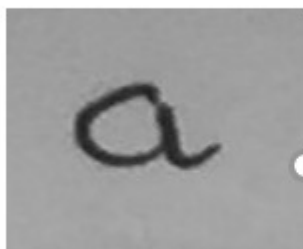
    for cnt in contours:
        x, y, width, height = cv2.boundingRect(cnt)
        if cv2.contourArea(cnt) > 0:
            x_min = min(x_min, x)
            y_min = min(y_min, y)

            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
```

### 3. Xử lý dữ liệu



Hình sau khi xử lý nhiễu và cắt gọn



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

#### **Xử lý dữ liệu**

Sử dụng phương pháp trích xuất đặc trưng HOG (Histogram of oriented gradient)

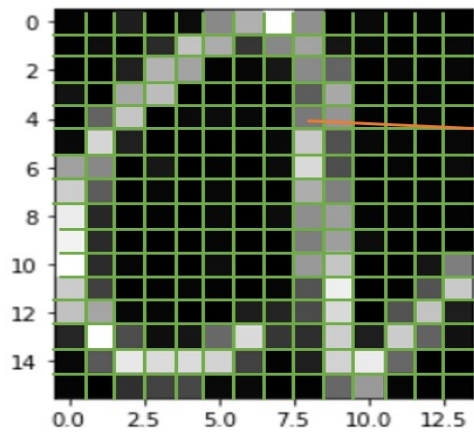
### 3. Xử lý dữ liệu

- Hình ảnh được chia thành các ô nhỏ nối tiếp nhau, mỗi ô có kích thước 2 x 2 pixel. Ta sẽ tìm được độ lớn và phương của mỗi ô.

```
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)
```

- 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:



10	10
10	10

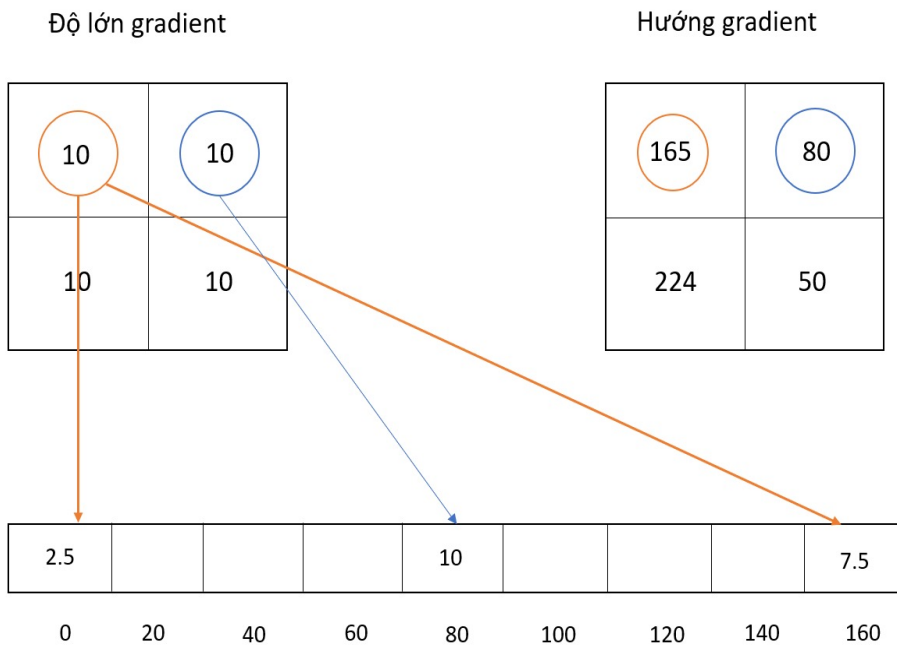
Độ lớn gradient

217	248
224	50

Hướng gradient

### 3. Xử lý dữ liệu

- 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:

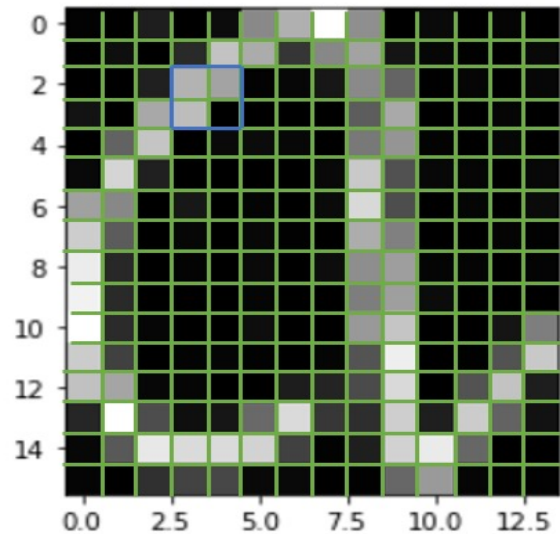
- Tại bin thứ  $l - 1$ : 
$$x_{l-1} = \frac{(x_1 - x)}{x_1 - x_0} * y$$

- Tại bin thứ  $l$ :

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



### 3. Xử lý dữ liệu



- Chuẩn hóa vector histogram theo block gồm 4 ô, mỗi ô 2 x 2 pixel.
- Dịch block đó sang 1 ô và ta sẽ thực hiện chuẩn hóa cho block đó. Sau khi đi qua hết 195 vị trí (vì ảnh có kích thước 14 x 16) và ghép nối tiếp các vector có 36 phần tử lại với nhau ta sẽ có một vector có  $36 \times 13 \times 15 = 7020$  phần tử. Đây là vector HOG đại diện cho toàn bộ hình ảnh.

## 4. ĐÁNH GIÁ KẾT QUẢ

- Sử dụng cách tính accuracy, accuracy càng cao thì mô hình càng tốt.
- Sau quá trình training các model, kết quả đạt được trên tập validation và tập test như sau:

### ❑ Model 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)
```

- 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%

## 4. ĐÁNH GIÁ KẾT QUẢ

### ❑ Model SVM

- 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)
    y_pred_vali = model_SVM_rbf.predict(X_vali)
    y_pred_test = model_SVM_rbf.predict(X_test)
```

- Kết quả trên tập validation:

accuracy			0.70	5229	
macro avg	0.69	0.69	0.69	5229	➔ Accuracy: 70%
weighted avg	0.70	0.70	0.70	5229	

- Kết quả trên tập test:

accuracy			0.67	3512	
macro avg	0.67	0.67	0.66	3512	➔ Accuracy: 67%
weighted avg	0.68	0.67	0.66	3512	

## 4. ĐÁNH GIÁ KẾT QUẢ

### ❑ Model 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)
```

- 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 mô hình đều cho kết quả tốt (đều trên 60%).
- Mô hình MLP-Classifier cho kết quả tốt nhất trên cả 2 tập validation và test.

## 5. KẾT LUẬN

### Kết luận chung

- Mô hình sau khi training cho kết quả tốt, điểm accuracy cao nhất đạt đến 0.71%.
- Các class 'á', 'ã', 'ỉ', 'ô', 'õ', 'ồ', 'ữ', 'y' có f1-score dưới 50%, riêng class 'y' có f1-score 25% và bị dự đoán sang class 'y' khá nhiều, class 'ữ' bị dự đoán sang class 'ứ' và 'ừ'. Nguyên nhân là các dữ liệu trong lúc thu thập và cắt ảnh đã bị nhiễu khá nhiều.

### Các hướng cải thiện bài toán

- Accuracy các model được chọn cho kết quả chung 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 ảnh bị nhiễu khá nhiều.
- Tăng thêm kích thước dữ liệu cho bài toán.
- Phát triển bài toán phân loại 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.

XIN CẢM ƠN!