ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

o0o

Nhóm 8

Môn học: Xử lý ảnh (INT3404E 20)

Giảng viên: Lê Thanh Hà

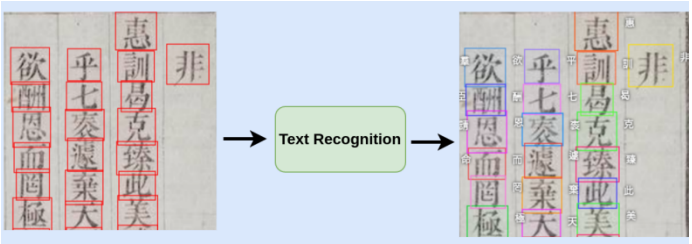
Thành viên : **Lê Văn Bảo - 21020171**
Tống Minh Trí - 21020249
Nguyễn Văn Thuyên - 21020237
Nguyễn Ngọc Tuấn - 21020247

Hà Nội - 2024

Mục lục

1 Bài toán nhận diện chữ Hán-Nôm

Nhận diện chữ Hán-Nôm là một thách thức lớn trong lĩnh vực ngôn ngữ học máy tính và nhận diện ký tự quang học (OCR). Chữ Hán-Nôm, bao gồm chữ Hán và chữ Nôm, được sử dụng để viết chữ Hán cổ điển và tiếng Việt trước khi hệ chữ cái Latinh được áp dụng vào đầu thế kỷ 20. Đặc trưng của chữ Hán-Nôm là bộ ký tự đa dạng với hàng ngàn ký tự, cùng nhiều biến thể lịch sử và khu vực, làm phức tạp thêm quá trình nhận diện. Nhiều văn bản chữ Hán-Nôm tồn tại dưới dạng các bản thảo lịch sử, thường bị hư hỏng hoặc phai mờ, đòi hỏi hệ thống OCR phải giải mã chính xác từ hình ảnh chất lượng kém.



Hình 1: Nhận diện chữ Hán-Nôm

Độ chính xác (**Accuracy**) là tiêu chí quan trọng trong nhận diện chữ Hán-Nôm vì nó đảm bảo việc tái tạo trung thực các văn bản, hỗ trợ nghiên cứu ngôn ngữ học và phục vụ các ứng dụng thực tiễn như giáo dục và số hóa di sản văn hóa. Độ chính xác cao giúp bảo tồn các tác phẩm văn học và tài liệu lịch sử của Việt Nam, tạo điều kiện thuận lợi cho các nhà nghiên cứu phân tích ngôn ngữ và tiến hành các nghiên cứu sâu hơn về tiến hóa chính tả và ngôn ngữ học so sánh. Ngoài ra, nhận diện chính xác cũng mang lại lợi ích thực tế trong việc phát triển các công cụ giáo dục và thư viện số, giúp cộng đồng tiếp cận dễ dàng hơn với các văn bản cổ.

$$\textbf{Độ chính xác} = \frac{\textbf{Số lượng dự đoán đúng}}{\textbf{Tổng số dự đoán}}$$

Các hệ thống nhận diện chữ Hán-Nôm hiện đại cần giải quyết nhiều thách thức kỹ thuật. Sự khan hiếm của các tập dữ liệu được gán nhãn đòi hỏi sử dụng các kỹ thuật như học chuyển giao, tăng cường dữ liệu và tạo dữ liệu tổng hợp để cải thiện hiệu suất của mô hình. Các kỹ thuật trích xuất đặc trưng tiên tiến, bao gồm mạng nơ-ron tích chập (CNN) và mạng nơ-ron hồi quy (RNN), giúp phân biệt giữa các ký tự có hình dạng tương tự. Ngoài ra, việc kết hợp các quy tắc ngôn ngữ và thông tin ngữ cảnh thông qua xử lý hậu kỳ có thể nâng cao

độ chính xác của nhận diện ký tự. Tóm lại, những tiến bộ trong công nghệ học máy và OCR mang lại hy vọng cải thiện độ chính xác và độ tin cậy của các hệ thống nhận diện chữ Hán-Nôm, góp phần vào việc bảo tồn và đánh giá cao di sản văn hóa quan trọng này.

2 Các phương pháp sử dụng

Nhóm em sử dụng 3 model khi thực hiện:

1. Mô hình 1:

```
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(len(class_folders), activation='softmax')
])
```

Mô hình CNN (Convolutional Neural Network) trong đoạn mã trên được xây dựng theo cấu trúc tuần tự. Đầu tiên, mô hình khởi tạo với lớp tích chập 2D đầu tiên có 32 bộ lọc kích thước 3x3, sử dụng hàm kích hoạt ReLU, và đầu vào có kích thước (chiều rộng, chiều cao, số kênh màu là 3). Tiếp theo là lớp gộp tối đa 2D với cửa sổ 2x2 để giảm kích thước không gian của đặc trưng đầu vào. Mô hình tiếp tục với lớp tích chập 2D thứ hai có 64 bộ lọc 3x3 và lớp gộp tối đa 2x2. Sau đó là lớp tích chập 2D thứ ba cũng với 64 bộ lọc 3x3. Các lớp tích chập này giúp trích xuất các đặc trưng quan trọng từ ảnh đầu vào. Sau khi qua các lớp tích chập, đặc trưng 2D được chuyển đổi thành vector 1D bằng lớp làm phẳng (Flatten). Tiếp theo là lớp kết nối đầy đủ (Dense) với 64 đơn vị đầu ra và hàm kích hoạt ReLU, giúp mô hình học các đặc trưng phức tạp hơn. Cuối cùng, lớp kết nối đầy đủ với số lượng đơn vị đầu ra bằng số lớp trong tập dữ liệu và hàm kích hoạt Softmax chuyển đổi các giá trị đầu ra thành xác suất cho các lớp. Mô hình này kết hợp các lớp tích chập, gộp tối đa, làm phẳng và kết nối đầy đủ để trích xuất đặc trưng và phân loại ảnh đầu vào một cách hiệu quả.

2. Mô hình 2:

```
def chu_nom_classifier(in_):
  model_ = Conv2D(32, (3, 3), activation = 'relu', padding = 'same')(in_)
  model_ = BatchNormalization()(model_)
  model_ = Conv2D(32, (3, 3), activation = 'relu')(model_)
  model_ = BatchNormalization()(model_)
  model_ = Conv2D(32, 5, strides = 2, padding = 'same', activation = 'relu')(mode
  model_ = MaxPooling2D((2, 2))(model_)
  model_ = BatchNormalization()(model_)
  model_ = Dropout(0.4)(model_)
  model_ = Conv2D(64, (3, 3), strides = 2, padding = 'same', activation = 'relu')
  model_ = MaxPooling2D(pool_size = (2, 2))(model_)
  model_ = BatchNormalization()(model_)
  model_ = Conv2D(64, kernel_size = (3, 3), strides = 2, padding = 'same', activa
  model_ = Dropout(0.4)(model_)
  model_ = Flatten()(model_)
  model_ = Dense(128, activation = 'relu')(model_)
  model_ = Dense(len(class_folders), activation = 'softmax')(model_)

  return model_
```

Mô hình mạng nơ-ron tích chập (CNN) để phân loại ảnh. Mô hình bắt đầu với lớp tích chập (Conv2D) có 32 bộ lọc 3x3, hàm kích hoạt ReLU và padding để giữ nguyên kích thước đầu ra, sau đó là lớp chuẩn hóa lô (BatchNormalization) để tăng tốc độ huấn luyện và ổn định mô hình. Tiếp theo là một lớp tích chập khác với 32 bộ lọc 3x3 và chuẩn hóa lô. Sau đó, mô hình thêm lớp tích chập với 32 bộ lọc 5x5, sử dụng stride 2 để giảm kích thước đầu ra, theo sau là lớp gộp tối đa (MaxPooling2D) với cửa sổ 2x2, chuẩn hóa lô, và dropout với tỷ lệ 0.4 để giảm thiểu overfitting. Mô hình tiếp tục với lớp tích chập có 64 bộ lọc 3x3, stride 2 và padding, sau đó là lớp gộp tối đa và chuẩn hóa lô. Một lớp tích chập khác với 64 bộ lọc 3x3, stride 2 và padding được thêm vào, tiếp theo là dropout. Các đặc trưng 2D sau đó được làm phẳng thành vector 1D bằng lớp Flatten, tiếp theo là lớp kết nối đầy đủ (Dense) với 128 đơn vị và hàm kích hoạt ReLU. Cuối cùng, lớp kết nối đầy đủ với số lượng đơn vị đầu ra bằng số lớp cần phân loại và hàm kích hoạt Softmax để phân loại đầu ra thành xác suất.

3. Mô hình 3:

```
def cbr(x, out_layer, kernel, stride):
    x = Conv2D(out_layer, kernel_size=kernel, strides=stride, padding="same")(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(alpha=0.1)(x)
    return x

def resblock(x_in, layer_n):
    x = cbr(x_in, layer_n, 3, 1)
    x = cbr(x, layer_n, 3, 1)
    x = Add()([x, x_in])
    return x

def create_classification_model(in_):
    x = cbr(in_, 64, 3, 1)
    x = resblock(x, 64)
    x = resblock(x, 64)
    x = cbr(x, 128, 3, 2)
    x = resblock(x, 128)
    x = resblock(x, 128)
    x = cbr(x, 256, 3, 2)
    x = resblock(x, 256)
    x = resblock(x, 256)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.4)(x)  # Tăng Dropout để tránh overfitting
    out = Dense(len(class_folders), activation="softmax")(x)

    return out
```

Hàm cbr (Convolution-BatchNorm-ReLU) thực hiện ba bước liên tiếp trên đầu vào: áp dụng lớp tích chập (Conv2D) với số lượng bộ lọc, kích thước bộ lọc và stride được chỉ định, sau đó là lớp chuẩn hóa lô (BatchNormalization) để tăng tốc độ huấn luyện và ổn định mô hình, cuối cùng là hàm kích hoạt Leaky ReLU với hệ số alpha là 0.1. Tiếp theo, hàm resblock (residual block) tạo một khối dư bằng cách áp dụng hàm cbr hai lần, rồi cộng kết quả đầu ra với đầu vào ban đầu để tạo kết nối dư, giúp mô hình học được các đặc trưng phức tạp hơn. Cuối cùng, hàm create classification model xây dựng mô hình phân loại ảnh bằng cách kết hợp nhiều lớp cbr và resblock. Đầu tiên, hàm áp dụng lớp cbr với 64 bộ lọc và kích thước 3x3, sau đó thêm hai khối dư với 64 bộ lọc. Tiếp theo, một lớp cbr với 128 bộ lọc và stride 2 được thêm vào, sau đó là hai khối dư với 128 bộ lọc. Tiếp tục, hàm thêm một lớp cbr với 256 bộ lọc và stride 2, sau đó là hai khối dư với 256 bộ lọc. Mô hình kết thúc với lớp gộp trung bình toàn cục (GlobalAveragePooling2D) để giảm kích thước đầu ra thành vector 1D, lớp Dropout với tỷ lệ 0.4 để tránh overfitting, và cuối cùng là lớp kết nối

đầy đủ (Dense) với số lượng đơn vị đầu ra bằng số lớp và hàm kích hoạt Softmax để phân loại đầu ra thành xác suất.

- Nhóm em thử nghiệm các tỉ lệ chia dữ liệu cho tập train và validate cho mô hình như 0.8-0.2, 0.9-0.1, 0.7-0.3, 0.85-0.15.

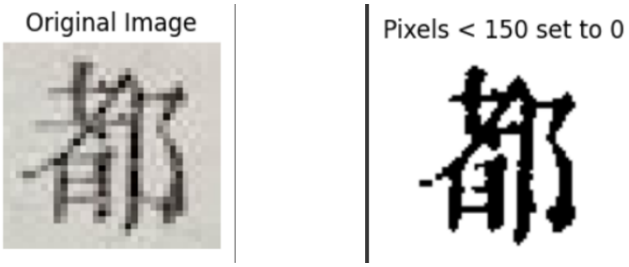
- Trong quá trình thực nghiệm nhóm em có sử dụng 2 hàm loss là:
  - Sparse categorical crossentropy: Hàm loss sparse categorical crossentropy sử dụng cho các bài toán phân loại nhiều lớp khi nhãn lớp là các số nguyên. Hàm loss này đo lường sự khác biệt giữa phân phối xác suất dự đoán của mô hình và phân phối xác suất thật. Trong bài toán phân loại nhiều lớp, mô hình dự đoán một vector xác suất cho mỗi mẫu đầu vào, trong đó mỗi phần tử của vector biểu thị xác suất của mẫu thuộc về một lớp nhất định. Hàm los này giúp giúp đơn giản hóa việc xử lý nhãn lớp và vẫn đảm bảo tính toán chính xác sự khác biệt giữa phân phối xác suất dự đoán và phân phối xác suất thật, làm cho quá trình huấn luyện mô hình trở nên dễ dàng và hiệu quả hơn.
  - categorical crossentropy: Chúng em sử dụng bởi trong quá trình tìm hiểu bài toán, chúng em thấy đa số mọi người dùng hàm los này. Hàm loss categorical crossentropy được sử dụng khi các nhãn đầu ra được mã hóa dạng one-hot. One-hot encoding là một phương pháp biểu diễn các nhãn dưới dạng vector nhị phân, trong đó chỉ có một phần tử có giá trị 1 và các phần tử còn lại đều là 0.

- Do nhận thấy ảnh được cung cấp hơi mờ hoặc có nhiều chỗ không được sắc nét và rõ ràng nên nhóm em có thử áp dụng các phương pháp như:
  - Sharpen

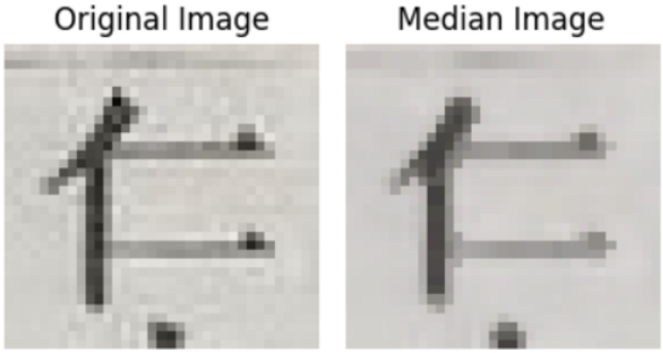
```
def sharpen_image(image_path):
  # Đọc ảnh từ đường dẫn
  img = cv2.imread(image_path)

  # Scale ảnh về kích thước 64x64
  new_height, new_width = 64, 64
  interpolated_img = cv2.resize(img, (new_width, new_height),
  interpolation=cv2.INTER_LINEAR)

  # Làm nổi bật chi tiết bằng Gaussian Blur
  blurred_img = cv2.GaussianBlur(interpolated_img, (0, 0), 80)
  sharpened_img = cv2.addWeighted(interpolated_img, 1.5, blurred_img, -0.5,0)
```

```
sharpened_img_gray = cv2.cvtColor(sharpened_img, cv2.COLOR_BGR2GRAY)

# Khởi tạo ảnh chỉnh sửa
modified_img = np.copy(sharpened_img_gray)

# Đặt các pixel có giá trị độ xám nhỏ hơn threshold bằng 0
threshold = 150
modified_img[sharpened_img_gray < threshold] = 0
modified_img[sharpened_img_gray >= threshold] = 255
return modified_img
```

Sharpening giúp tăng cường các chi tiết nhỏ và các cạnh trong hình ảnh và làm giảm tác động của sự mờ và cải thiện chất lượng của dữ liệu đầu vào.



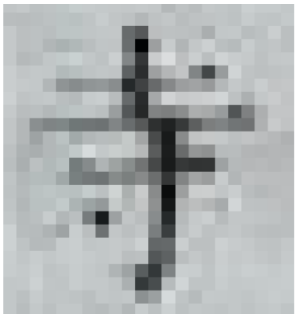
– Median: Xử lý ảnh bằng bộ lọc trung vị (median filter) là một phương pháp hiệu quả để giảm nhiễu mà không làm mất chi tiết của ảnh. Bộ lọc trung vị hoạt động bằng cách thay thế mỗi điểm ảnh với giá trị trung vị của các điểm ảnh lân cận trong một cửa sổ kích thước xác định. Phương pháp này đặc biệt hữu ích trong việc loại bỏ nhiễu xung và giữ lại các cạnh sắc nét.



– Threshold:

```
def threshold(image_path):
  img = cv2.imread(image_path)
  img = cv2.resize(img, (H, W))
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  ret, th1 = cv2.threshold(img, 175, 255, cv2.THRESH_BINARY_INV)
  return th1
```

Hàm này sẽ đọc hình ảnh từ đường dẫn đầu vào và lưu hình ảnh đó vào biến img. Sau đó, hình ảnh sẽ được thay đổi kích thước về kích thước cố định với chiều cao và chiều rộng lần lượt là H và W. Tiếp theo, hình ảnh được chuyển từ không gian màu BGR sang không gian màu xám (grayscale). Sau khi chuyển đổi, hàm áp dụng một ngưỡng nhị phân đảo (THRESH BINARY INV) với giá trị ngưỡng là 175 lên hình ảnh xám. Điều này có nghĩa là các điểm ảnh có giá trị xám lớn hơn hoặc bằng 175 sẽ được chuyển thành màu đen (0), trong khi các điểm ảnh có giá trị nhỏ hơn 175 sẽ được chuyển thành màu trắng (255). Kết quả cuối cùng là một hình ảnh nhị phân đảo được lưu vào biến th1, và biến này sẽ được trả về khi hàm kết thúc. Hàm threshold này giúp chuyển đổi một hình ảnh màu thành một hình ảnh nhị phân với ngưỡng cụ thể, đồng thời đảo ngược màu sắc của hình ảnh đó.

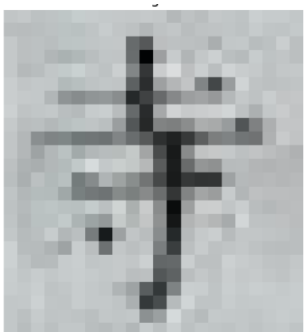


– Denoising + Ben's Preprocessing + threshold

```
def read_image(image):
    return cv2.cvtColor(cv2.imread(image), cv2.COLOR_BGR2RGB)

def apply_ben_preprocessing(image):
    return cv2.addWeighted(image,4,cv2.GaussianBlur(image,(0,0),10),-4,128)

def apply_denoising(image):
    return cv2.fastNlMeansDenoisingColored(image, None, 20, 20, 7, 21)
```

```
def threshold(image_path):
  img = read_image(image_path)
  before = apply_ben_preprocessing(img)
  after = apply_denoising(before)
  img = cv2.resize(after, (H, W))
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  ret, th1 = cv2.threshold(img, 130, 255, cv2.THRESH_BINARY_INV)
  return th1
```

Phương pháp này bao gồm các bước đọc ảnh, tiền xử lý, khử nhiễu, và áp dụng ngưỡng (thresholding). Đầu tiên, hàm 'read image' đọc một ảnh từ tên file và chuyển đổi nó sang định dạng RGB bằng OpenCV. Tiếp theo, hàm 'apply ben preprocessing' thực hiện tiền xử lý Ben's trên ảnh bằng cách áp dụng trọng số được tính toán dựa trên các pixel của ảnh và sử dụng bộ lọc Gaussian để làm mờ ảnh, tạo ra sự tương phản cao hơn. Sau đó, hàm 'apply denoising' áp dụng khử nhiễu trên ảnh với cường độ gấp đôi so với khuyến nghị từ tài liệu OpenCV, nhằm loại bỏ các nhiễu màu và làm mịn ảnh. Hàm 'threshold' là phần chính của quy trình xử lý, bắt đầu bằng việc đọc ảnh từ đường dẫn file bằng cách sử dụng 'read image'. Sau đó, ảnh được tiền xử lý bằng hàm 'apply ben preprocessing' và tiếp tục khử nhiễu bằng hàm 'apply denoising'. Sau khi khử nhiễu, ảnh được thay đổi kích thước về kích thước cố định (H, W) và chuyển đổi sang ảnh grayscale. Cuối cùng, hàm áp dụng ngưỡng (thresholding) để chuyển đổi ảnh grayscale thành ảnh nhị phân (binary) với ngưỡng 130, nghĩa là các pixel có giá trị thấp hơn 130 sẽ được đặt thành 255 (trắng) và các pixel có giá trị cao hơn sẽ được đặt thành 0 (đen). Kết quả của hàm 'threshold' là ảnh nhị phân đã qua xử lý, sẵn sàng cho các bước phân tích và nhận dạng tiếp theo.



- Chúng em có áp dụng một số phương pháp tăng cường dữ liệu như:
  - rescale, rotation, width shift range, height shift range

    ```
    ImageDataGenerator(rescale=1./255, rotation_range=20,
    width_shift_range=0.1,height_shift_range=0.1)
    ```

    'rescale=1./255': Tham số này chuẩn hóa giá trị pixel của ảnh về khoảng [0, 1] bằng cách chia cho 255. Việc này giúp cải thiện hiệu suất của mô hình huấn luyện bằng cách đưa các giá trị pixel về cùng một phạm vi.'rotation range=20': dải góc xoay ngẫu nhiên (đơn vị là độ) mà ảnh có thể được quay ngẫu nhiên. 'width shift range=0.1' và 'height shift range=0.1' : Các tham số này xác định phạm vi ngẫu nhiên mà ảnh có thể được dịch ngang và dọc.
  - zoom [0.8, 1.3]: Ảnh được đưa vào sẽ được phóng to hoặc thu nhỏ ảnh ngẫu nhiên trong khoảng 80% đến 130%
  - grid distortion, random erasing, elastic transform
- Về việc huấn luyện mô hình, chúng em huấn luyện mô hình với các tham số khác nhau để tìm ra bộ tham số tốt nhất:
  - Kích thước ảnh khi đưa vào mô hình: 64x64, 128x128, 150x150, 192x192, 240x240, 256x256
  - Batch size: 30, 50, 75
  - Epoch: 70, 100, 150, 200, 250

3 Kết quả và phân tích

- Sau đây em sẽ trình bày kết quả theo trình tự thời gian mà nhóm em thực hiện.
- Với mô hình đầu tiên chúng em sử dụng hàm loss sparse categorical crossentropy, áp dụng phương pháp tăng cường dữ liệu: Rescale, rotation, width shift range, height shift range.
  - Với mô hình này chúng em chia tập train và validate theo các tỉ lệ: 0.8-0.2, 0.9-0.1, 0.7-0.3, 0.85-0.15 và trộn ảnh ngẫu nhiên trước khi chia.
  - Và kết quả chúng em thu được trên tập test là
  - Dựa trên kết quả này thì chúng em quyết định sử dụng tỉ lệ train-val là 0.8-0.2 cho các lần sau

Tỉ lệ	Kết quả
0.8-0.2	2.98
0.9-0.1	2.76
0.7-0.3	2.43
0.85-0.15	2.91

- Dựa trên quá trình trực quan ảnh trước và sau khi áp dụng các phương pháp xử lý ảnh là sharpen, median, threshold. Chúng em nhận thấy việc chỉ áp dụng mỗi threshold đưa ra kết quả tốt hơn cũng tốn ít thời gian trong quá tình xử lý ảnh so với 2 phương pháp còn lại. Và sau khi áp dụng threshold với các ảnh trước khi đưa vào quá trình huấn luyện và trước khi đưa vào quá trình dự đoán với mô hình đầu tiên đã cho kết quả trên tập test là 5.14%
- Với mô hình thứ 2, hàm loss vẫn là sparse categorical crossentropy, việc tăng cường dữ liệu chúng em chỉ sử dụng rescale thì kết quả thu được tốt hơn rất nhiều là 30.25%
- Sau quá trình tìm hiểu chúng em thấy rằng với bài toán này thì đa số mọi người sử dụng hàm loss categorical crossentropy nên chúng em quyết định sử dụng hàm loss này. Và với mô hình thứ hai, kết quả chúng em thu được là 46.72%. Từ kết quả trên thì trong các lần huấn sau chúng em sử dụng hàm loss này.
- Với mô hình thứ 2, chúng em huấn luyện mô hình với kích thước batch size, epoch, size image khác nhau:
  - loss: categorical crossentropy
  - augment: zoom [0.5, 2.0]
  - size: 64x64, 128x128, 150x150, 192x192, 240x240, 256x256
  - batch size: 30, 50, 75
  - epoch: 70, 100, 150, 200, 250
  - Kết quả tốt nhất và có thể chạy được: 58.57% với 128x128, epoch 250, batch size 50.

- Trong quá trình thử in ra các ảnh sau khi được áp dụng threshold thì chúng em nhận thấy có nhiều ảnh sau khi bị biến đổi chưa được hợp lý và phù hợp. Và sau khi thử áp dụng với phương pháp xử lý ảnh phương pháp xử lý ảnh Denoising + Ben's Preprocessing + threshold thì đã xử lý được hết các trường hợp chưa được phù hợp ở trên. Hình dưới đây là 1 ví dụ, bên trái là áp dụng mỗi threshold bên phải là áp dụng phương pháp xử lý Denoising + Ben's Preprocessing + threshold. Với mô hình

thứ 2 áp dụng phương pháp xử lý ảnh Denoising + Ben's Preprocessing + threshold với batchsize 50, epoch 250, size: 128x128, chúng em thu được kết quả là 75,58%

- Sau đó chúng em sử dụng mô hình thứ 3 với phương pháp xử lý ảnh Denoising + Ben's Preprocessing + threshold với batchsize 50, epoch 50, kích thước ảnh 128x128. Nhưng do tài nguyên không đủ, bị chết ở epoch 38 và cho kết quả trên tập test là 88.15%.
- Cuối cùng chúng em có thử áp dụng các phương pháp tăng cường dữ liệu khác là grid distortion, random erasing, elastic transform nhưng chưa áp dụng được.

4 Kết luận

Trong 3 mô hình thì mô hình thứ 3 cho kết quả tốt hơn 2 mô hình còn lại. Hàm loss categorical crossentropy phù hợp với bài toán này hơn là Sparse categorical crossentropy. Tỉ lệ chia train-val đem lại kết quả tốt nhất là 0.8-0.2. Phương pháp xử lý ảnh tốt nhất là Denoising + Ben's Preprocessing + threshold. Và cuối cùng khi huấn luyện sử dụng batchsize 50, epoch 250, size: 128x128 đem lại kết quả tốt nhất.