**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---🙢🕮🙠---



**BÁO CÁO**

**MACHINE LEARNING**

**XÂY DỰNG HỆ THỐNG GIẢI CAPTCHA**

**SVTH** : Dương Tăng Lực **MSSV**: 61133916

**Lớp** : 61.CNTT-CLC

**GVHD** : Đinh Đồng Lưỡng

*Khánh Hòa, tháng 10 năm 2022*

Mục lục

[1 Tổng quan đề tài 3](#_Toc117879216)

[1.1 Lý do chọn đề tài 3](#_Toc117879217)

[1.2 Kết quả đạt được 3](#_Toc117879218)

[2 Chuẩn bị dữ liệu 4](#_Toc117879219)

[2.1 Thu thập dữ liệu 4](#_Toc117879220)

[2.2 Mô tả dữ liệu 4](#_Toc117879221)

[2.3 Xử lý dữ liệu 4](#_Toc117879222)

[3 Train dữ liệu 12](#_Toc117879223)

[3.1 SVM Model 12](#_Toc117879224)

[3.2 Random Forest Classifier 12](#_Toc117879225)

[3.3 KNeighbors Classifier 13](#_Toc117879226)

[4 Xây dựng hệ thống giải captcha 15](#_Toc117879227)

[4.1 Tái sử dụng 15](#_Toc117879228)

[4.2 Giải captcha 15](#_Toc117879229)

[5 References 17](#_Toc117879230)

1. Tổng quan đề tài

Lý do chọn đề tài

Một CAPTCHA (ˈkæptʃə, đọc giống như "[capture](https://vi.wiktionary.org/wiki/capture)") là một loại kiểm thử [dạng hỏi đáp](https://vi.wikipedia.org/w/index.php?title=X%C3%A1c_minh_d%E1%BA%A1ng_h%E1%BB%8Fi-%C4%91%C3%A1p&action=edit&redlink=1) được dùng trong [máy tính](https://vi.wikipedia.org/wiki/M%C3%A1y_t%C3%ADnh) để xác định xem người dùng có phải là [con người](https://vi.wikipedia.org/wiki/Lo%C3%A0i_ng%C6%B0%E1%BB%9Di) hay không. "CAPTCHA" là chữ viết tắt của "Completely Automated Public Turing test to tell Computers and Humans Apart" ([Phép thử Turing](https://vi.wikipedia.org/wiki/Ph%C3%A9p_th%E1%BB%AD_Turing) công cộng hoàn toàn tự động để phân biệt [máy tính](https://vi.wikipedia.org/wiki/M%C3%A1y_t%C3%ADnh) với [người](https://vi.wikipedia.org/wiki/Ng%C6%B0%E1%BB%9Di)), được trường [Đại học Carnegie Mellon](https://vi.wikipedia.org/wiki/%C4%90%E1%BA%A1i_h%E1%BB%8Dc_Carnegie_Mellon) cố gắng đăng ký thương hiệu nhưng đã bị bác bỏ. Đây là một quá trình một máy tính ([máy chủ](https://vi.wikipedia.org/wiki/M%C3%A1y_ch%E1%BB%A7)) yêu cầu một người dùng hoàn tất một kiểm tra đơn giản mà máy tính có thể dễ dàng tạo ra và đánh giá, nhưng không thể tự giải nó được. Vì máy tính không thể giải quyết CAPTCHA, bất kỳ người dùng nào nhập vào lời giải đúng sẽ được xem là con người.

Có nhiều loại CAPTCHA khác nhau, tuy nhiên ở để tài này, em lựa chọn dạng captcha ký tự [a-z0-9] bị làm nhiễu và uống cong. Bài toán khá tương tự nhận diện chữ viết tay, tuy nhiên ở mức độ khó hơn

Kết quả đạt được

* Kỹ năng phân tích dữ liệu cho việc học máy
* Biết cách trích đặc trưng từ dữ liệu
* Biết lựa chọn kĩ thuật học máy (học có giám sát và không giám sát)
* Lựa chọn phương pháp training model
* Nhận xét so sánh các kết quả, biết chọn lọc kết quả tối ưu
* Có những bước đầu làm quen, tiếp cận với học máy
* Giải captcha

1. Chuẩn bị dữ liệu

Thu thập dữ liệu

Bộ dữ liệu được thu thập trên Internet

Mô tả dữ liệu

Tập dữ liệu là bộ ảnh gồm 12,131 Files ảnh được cắt ra từ ảnh captcha. Được đặt tên theo quy tắc ‘<X>-<uuid>.jpg’.

* X ở đây là nhãn của anh – ảnh số 0 thì lable = ‘0’
* uuid – ký tự ngẫu nhiên để không bị trùng tên file ảnh

Ảnh có kích thước 30x36, vì vậy có 1080 pixel

Xử lý dữ liệu

Denoising image

C:\Users\Admin\Desktop\captcha\data\raw\0037.jpg

Figure - Ảnh captcha raw

Quan sát captcha, ta thấy rằng những ký tự bị các đường, chấm xanh, vàng, đỏ... đè lên gây nhiễu

Ta giảm nhiễu hình bằng hàm cv2.fastNlMeansDenoisingColored() trong thư viện nguồn mở OpenCV

Cài đặt thư viện: pip install opencv-python

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| void cv::fastNlMeansDenoisingColored | | ( | [**InputArray**](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#ga353a9de602fe76c709e12074a6f362ba) | src, |
|  | |  | [**OutputArray**](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#gaad17fda1d0f0d1ee069aebb1df2913c0) | dst, |
|  | |  | float | h = 3, |
|  | |  | float | hColor = 3, |
|  | |  | int | templateWindowSize = 7, |
|  | |  | int | searchWindowSize = 21 |
|  | | ) |  |  |
| h | | Parameter regulating filter strength for luminance component. Bigger h value perfectly removes noise but also removes image details, smaller h value preserves details but also preserves some noise | | | | |
| hColor | | The same as h but for color components. For most images value equals 10 will be enough to remove colored noise and do not distort colors | | | | |

Theo định nghĩa thì ta thử h, hColor từ 3. Tiếp tục tăng giá trị chúng lên và thử nhiều lần thì thấy giá trị h, hColor = 50, 50 thì giảm nhiễu tốt nhất

|  |  |
| --- | --- |
| 1  2 | img = cv2.imread(file\_path)  dst = cv2.fastNlMeansDenoisingColored(img,None,50,50,7,21) |

Sau đó ta chuyển hình thành trắng đen, pixel nào < 128 thì thành 0(đen) ngược lại 255(trắng)

|  |  |
| --- | --- |
| 1  2  3 | cv2.imwrite(file\_path, dst)  img = Image.open(file\_path).convert('L')  img = img.point(**lambda** x: 0 **if** x<128 **else** 255, '1') |

* **Kết quả:**



Figure - Ảnh captcha trước và sau giảm nhiễu

* **Nhận xét:** Bản chất của denoising là làm cho dữ liệu ảnh đã bị mất mát. Em đã thử tăng giá trị h, hColor lên rất cao và hình ảnh trong hình bị xóa mất toàn bộ(gần như ảnh màu trắng). Tuy nhiên, ở trường hợp các giá trị hợp lý trên các ký tự trong hình khá rõ ràng và chấp nhận được

Cắt từng ký tự captcha

1. Tìm lề trái phải

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | part = 0  img = Image.open(file\_path)  p = img.convert('P')  w, h = p.size  letters = []  left, right= -1, -1  found = False  **for** i **in** range(w):  in\_letter = False  **for** j **in** range(h):  **if** p.getpixel((i,j)) == 0:  in\_letter = True  **break**  **if** **not** found **and** in\_letter:  found = True  left = i  **if** found **and** **not** in\_letter **and** i-left > 25:  found = False  right = i  letters.append([left, right])  origin = file\_path.split('/')[-1].split('.')[0]  **for** [l,r] **in** letters:  **if** r-l < 40:  bbox = (l, 0, r, h)  crop = img.crop(bbox)  crop = crop.resize((30,60))  crop.save(join(out\_directory, '{0:04}\_{1}.jpg'.format(part, origin)))  part += 1 |

Mục đích đoạn code trên: Tìm lề trái – phải của từng ký tự trong hình

Giải thích: Tìm lề trái và lề phải của từng chữ cái trong hình captcha. Chuyển ảnh captcha thành matrix 2 chiều, quét theo chiều x(chiều dài của ảnh) nếu gặp pixel 0(đen) thì tìm thấy lề trái của ký tự (found) và đang trong ký tự (in\_letter), tăng x lên nếu gặp pixel 255(trắng) thì not in\_letter, tức lề phải. Tiếp tục đến hết chiều dài hình thì dừng. Ta cắt những cặp lề trái phải và thu được từng ký tự riêng lẻ

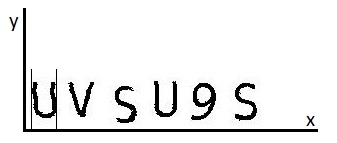


Figure - Ảnh mô tả việc tìm lề trái và phải của các ký tự trong hình

* **Nhận xét:** Thuật toán chưa tối ưu, vì một số hình các chữ cái dính liền nhau nên sẽ không tách ra được

1. Tìm lề trên dưới

Sau khi xử lý lề trái phải, ta xử lý lề trên dưới

|  |  |
| --- | --- |
| 1  2  3  4  5 | **def** is\_column\_lower\_value(column, val):  **for** e **in** column:  **if** e <= val:  **return** True  **return** False |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | BLACK = 0  *# load the image*  image = Image.open(file\_path)  *# convert image to numpy array*  data = np.asarray(image)  *# check first bottom row is black*  i = len(data) - 1  **while** i > 0:  **if**(is\_row\_lower\_value(data[i], BLACK)):  data = data[:i]  **break**  i-=1  *# check first top row is black*  **for** i **in** range(len(data)):  **if** is\_row\_lower\_value(data[i], BLACK):  data = data[i:]  **break**  image = Image.fromarray(data)  image = image.resize((30, 36)) |
|  |  |

Ý tưởng thuật toán là check pixel đầu tiên ở bottom là 0 (lề dưới), pixel đầu tiên ở top là 0(lề trên), sau đó cắt hình và resize về (30,36)

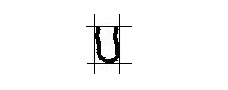


Figure - Mô tả việc tìm lề trên dưới của ký tự

* **Kết quả:**

C:\Users\Admin\Desktop\captcha\data\chars\U-87NDR4.jpg

Figure - Ký tự sau khi được cắt ra

* **Nhận xét:**
  + Ảnh cắt các ký tự khá là ổn định, ít mất mát và giữ được các thông tin của nó
  + Ảnh cắt ra sẽ thể mất mát nếu như ký tự quá dài và bị dính liền với ký tự khác. Ta có thể quan sát ảnh ở dưới, có thể thấy từ ký tự ‘o’ đã trở thành ký tự ‘c’

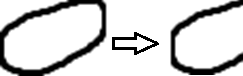


Figure - Ảnh cắt ra ở trường hợp ký tự quá dài và bị dính liền

Dán nhãn cho từng ký tự

Có thể thực hiện bằng tay, máy, api… Trong bài toán này, em sẽ dán nhãn cho 36 ký tự đầu tiên bằng tay sau đó đặt tên theo định dạng ‘<X>-<uuid>.jpg’, trong đó <X> là nhãn chính xác của hình [0-9][a-z], <uuid> unique id

Ex: U-20262\_3427.jpg

Tiền xử lý dữ liệu

1. Cài đặt HOG features extraction

* **derivAperture, winSigma, histogramNormType, L2HysThreshold, gammaCorrection and nlevels**: những tham số được coi là mặc định. Một số bài viết có nói rằng hầu như mỗi khi sử dụng phương pháp này đều cài đặt thông số như vậy
* winSize: size của ký tự được cắt (30x36)
* nbins: Số lượng bins trong biểu đồ histogram, tác giả của HOG paper đề nghị sử dụng giá trị 9. Độ lớn của phương gradient sẽ nằm trong khoảng [0, 180] nên mỗi bins sẽ có độ dài là 20
* cellSize: ảnh của chúng ta có 30x36, cho nên số pixel là 1080. Vì thế size của **descriptor** phải nhỏ hơn 1080
* blockSize: Kích thước của một block (đơn vị pixels) mà trên đó ta chuẩn hóa véc tơ histogram tổng hợp. Thông thường sẽ bằng 2 lần cellSize
* winStride: Số bước stride (đơn vị pixels) khi di chuyển window trên ảnh gốc để tính véc tơ histogram trên mỗi block của ảnh.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9 | | winSize = (30, 36)  nbins = 9  derivAperture = 1  winSigma = -1.  histogramNormType = 0  L2HysThreshold = 0.2  gammaCorrection = 1  nlevels = 64  signedGradients = True |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | | **def** process\_image(image\_path, cellSize):      blockSize = (cellSize[0]\*2, cellSize[1]\*2)      blockStride = cellSize      img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)      hog = cv2.HOGDescriptor(winSize, blockSize, blockStride,                              cellSize, nbins, derivAperture,                              winSigma, histogramNormType, L2HysThreshold,                              gammaCorrection, nlevels, signedGradients)      descriptor = hog.compute(img)  **return** np.array(descriptor) | | |
|  | |  | | |

1. Cài đặt trích đặc trưng HOG features từ ký tự và lấy nhãn

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | chars\_list = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'  chars\_dict = {c: chars\_list.index(c) **for** c **in** chars\_list}  **def** process\_data(directory, cellSize):  images = []  labels = []  *# lấy path tất cả hình \*.jpg*  image\_list = process\_directory(directory)  **for** image\_path **in** image\_list:  images.append(process\_image(image\_path, cellSize))  *# lấy ký tự đầu tiên của tên hình làm nhãn*  labels.append(chars\_dict[image\_path.split('/')[-1].split('-')[0]])  **return** np.array(images), np.array(labels) |

1. Trích đặt trưng

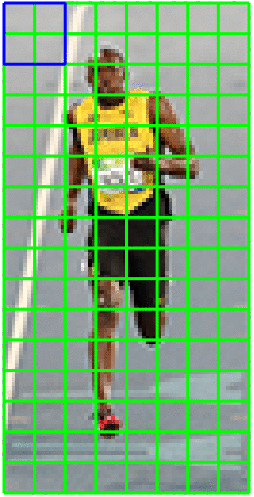


Figure - Ảnh được chia theo cellSize

* Ô vuông xanh dương được gọi là Blocksize. Ở đây là (2, 2), nghĩa là 1 Blocksize có 4 cellSize
* blockStride là bước nhày của Blocksize. Nghĩa là khi ta chuẩn hóa véc tơ histogram tổng hợp, Blocksize sẽ nhảy bao nhiêu pixel để tiếp tục chuẩn hóa véc tơ histogram tổng hợp
* Các ô vuông xanh lá cây được gọi là cellSize. Như đã đề cập ở trên, việc chọn cellSize phải hợp lý để **descriptor** nhỏ hơn 1080. Bởi 1 cellSize ta chia thành 9 bins và blockSize ta lấy (2, 2) nên blockSize sẽ có 9(bins)x4(cellSize)=36(véc tơ histogram)
* Ví dụ ta chọn cellSize (6, 6) quá trình tính toán HOG sẽ di chuyển (30/6-1=4) lượt theo chiều rộng và (36/6-1=5) lượt theo chiều cao. Như vậy sẽ có tổng cộng 4x5=20 patches, mỗi patch tương ứng với 1 véc tơ histograms 36 chiều. Do đó cuối cùng véc tơ HOG sẽ có kích thước là 20x36=720 chiều. Nhỏ hơn 1080 theo ảnh gốc
* Theo cách tính ở trên, để xử lý bài toán, em quyết định trích đặc trưng ảnh theo 2 cellSize là (6, 6) – 720 chiều và (15, 18) – 36 chiều. Để so sánh độ chính xác giữa 36 và 720 đặc trưng
* Trích đặc trưng theo cellSize (6, 6)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | images, labels = process\_data('data/chars/', **(6, 6)**)  # Split data into 80% train and 20% test subsets  images\_train\_0, images\_test\_0, labels\_train\_0, labels\_test\_0 = train\_test\_split(      images, labels, test\_size=0.2, random\_state=42)  images[0].shape **# (720,)** |

* Trích đặc trưng theo cellSize (15, 18)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | images, labels = process\_data('data/chars/', **(15, 18)**)      # Split data into 80% train and 20% test subsets  images\_train\_1, images\_test\_1, labels\_train\_1, labels\_test\_1 = train\_test\_split(      images, labels, test\_size=0.2, random\_state=42)  images[0].shape **# (36,)** |

* Ta thấy, việc trích đặc trưng cho kết quả giống như tính toán ở trên qua shape của vector

1. Train dữ liệu

SVM Model

* Sau khi trích được 2 bộ đặc trưng 720 và 36 thì ta train thử:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # Create a classifier: a support vector classifier 720 features  model\_svm = svm.SVC(kernel="linear", C=10e5)  # Learn the digits on the train subset  model\_svm.fit(images\_train\_0, labels\_train\_0)  model\_svm.score(images\_test\_0, labels\_test\_0) # 0.9958796868562011  # 9.9s excuted  # Create a classifier: a support vector classifier 36 features  model\_svm\_1 = svm.SVC(kernel="linear", C=10e5)  # Learn the digits on the train subset  model\_svm\_1.fit(images\_train\_1, labels\_train\_1)  model\_svm\_1.score(images\_test\_1, labels\_test\_1)# 0.9954676555418212  # 2.2s excuted |

* Ta thấy rằng tỉ lệ chính xác của 2 bộ đặc trưng là tương đương nhau ta, thời gian train là của 36 đặc trưng hiển nhiên nhanh hơn 720 đặc trưng. Ta thấy rằng việc chọn (15, 18) tối ưu nhất
* Dự đoán thử 10 dòng đầu tiên của tập test

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 8 9 | labels\_test\_0[:10]  # Nhãn chính xác  #[30, 11, 27, 22, 23, 30, 28, 33, 16, 21]  model\_svm.predict(images\_test\_0[:10]) # 720 đặc trưng dự đoán #[30, 8, 27, 22, 23, 30, 28, 33, 16, 21] model\_svm\_1.predict(images\_test\_1[:10]) # 36 đặc trưng dự đoán #[30, 8, 27, 22, 23, 30, 28, 33, 16, 21] |

* Ta thấy rằng, độ chính xác là như nhau
* 36 đặc trưng khá chính xác, ổn định và nhanh

[Random Forest Classifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

Bây giờ, ta thử phân lớp bằng một phương pháp khác. Ta vẫn sử dụng 720 và 36 đặc trưng để train và so sánh

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | model\_random\_forest = RandomForestClassifier(criterion='entropy', n\_estimators=100) model\_random\_forest.fit(images\_train\_0, labels\_train\_0) model\_random\_forest.score(images\_test\_0, labels\_test\_0) model\_random\_forest.score(images\_test\_0, labels\_test\_0) # **0.9962917181705809** # 58s excuted model\_random\_forest\_1 = RandomForestClassifier(criterion='entropy', n\_estimators=100) model\_random\_forest\_1.fit(images\_train\_1, labels\_train\_1) model\_random\_forest\_1.score(images\_test\_1, labels\_test\_1) model\_svm\_1.score(images\_test\_1, labels\_test\_1)# **0.9971157807993407** # 10s excuted |

* Dự đoán thử 10 dòng đầu tiên của tập test

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 8 9 | labels\_test\_0[:10]  # Nhãn chính xác  #[30, 11, 27, 22, 23, 30, 28, 33, 16, 21]  model\_svm.predict(images\_test\_0[:10]) # 720 đặc trưng dự đoán #[30, 8, 27, 22, 23, 30, 28, 33, 16, 21] model\_svm\_1.predict(images\_test\_1[:10]) # 36 đặc trưng dự đoán #[30, 8, 27, 22, 23, 30, 28, 33, 16, 21] |

* Ta thấy rằng, độ chính xác là như nhau
* Tuy nhiên, ở Random Forest Classifier ta thấy, độ chính xác của 36 features là nhỉnh hơn so với 720, tốc độ train nhanh hơn gấp 6 lần.

KNeighbors Classifier

Vẫn như trên, nhưng ta sử dụng phương pháp KNeighbors Classifierl

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | model\_KNC\_1 = KNeighborsClassifier(n\_neighbors=36) model\_KNC\_1.fit(images\_train\_1, labels\_train\_1) model\_KNC\_1.score(images\_test\_1, labels\_test\_1) # **0.9645653069633292** # 3.4s excuted  model\_KNC\_1 = KNeighborsClassifier(n\_neighbors=36) model\_KNC\_1.fit(images\_train\_1, labels\_train\_1) model\_KNC\_1.score(images\_test\_1, labels\_test\_1) # **0.934487021013597** #1.9s excuted |

* Ta thấy rằng, độ chính xác là như nhau
* Tuy nhiên, ở KNeighbors Classifier ta thấy, độ chính xác của 36 và 720 features khá là thấp so với các phương pháp trên, tốc độ training thì khá nhanh

1. Xây dựng hệ thống giải captcha

Tái sử dụng

Qua kết quả training, em quyết định lấy kết quả của RandomForestClassifier ở 36 đặc trưng bởi độ chính xác cao nhất và số đặc trưng là ít nhất. Em lưu model ra file ‘rfc.pkl’ để tái sử dụng khi predict ký tự

joblib.dump(model\_random\_forest\_1, "rfc.pkl")

Giải captcha

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | def predict\_char(image\_path):      image = process\_image(image\_path, (15, 18)).reshape(1, -1)      clf = joblib.load("rfc.pkl")      actual = chars\_list[clf.predict(image)[0]]      return actual  def predict\_string(file\_path):      res = ''      out\_path = 'tmp/'      if not os.path.exists(out\_path):          os.makedirs(out\_path)      dir = reduce\_noise(file\_path)      crop(dir, out\_path)      adjust\_dir(out\_path, out\_path)      file\_list = process\_directory(out\_path)      for f in sorted(file\_list):          res += predict\_char(f)      filelist = glob.glob(join(out\_path, "\*"))      for f in filelist:          remove(f)      return res |

* Hàm predict\_char():
  + Trích HOG features (15, 18)
  + Load model Random Forest ‘rfc.pkl’ để predict ký tự được cắt ra
* Hàm predict\_string(), ý tưởng như trình bày ở trên. Cụ thể theo các bước:
  + Denoising
  + Cắt hình thành các ký tự
  + Lưu vào folder tmp(tạm)
  + Đoán từng ký tự được cắt ra
  + Nối chuỗi

1. References

* <https://docs.python.org/3/>
* <https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py>
* <https://viblo.asia/p/breaking-a-captcha-by-machine-learning-bWrZneDbKxw>
* <https://phamdinhkhanh.github.io/2019/11/22/HOG.html>
* https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
* <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
* https://learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/