Segmatation by color algorithm:

****General idea****:

Since the characters in our CAPTCHA image are mostly in different colors, we can classify pixels based on their colors. Then, we copy pixels of the same color to a new image with a white background.

Version 1: based on the length of filename (known of how many characters)

1. **Color Extraction (`extract_colors`):**

- Scans the image pixel-by-pixel, ignoring a specified background color (white).
- Counts occurrences of each unique non-background color.
- Sorts colors based on frequency (most frequent first).

2. **Color Sorting by Horizontal Position (`sort_colors_by_position`):**

- For each color, calculates the average horizontal position (x-coordinate) of pixels matching that color.
- Sorts colors from left to right based on their average horizontal positions.

3. **Processing Each Target File:**

- Reads image data and determines the intended number of distinct colors based on the file's base name length.
- Extracts and sorts colors by frequency and position.
- Maps each extracted color to characters in the image's base name for naming output files.

4. **Creating and Saving Individual Color Images:**

- For each identified color:
    - Generates a new image with a white background.
    - Transfers pixels of the current color to this new image.
    - Saves the generated image separately with a filename derived from the base name, associated character, and occurrence count.


wtvaivr-r-1


wtvaivr-v-2


wtvaivr-i-1


wtvaivr-a-1


wtvaivr-v-1


wtvaivr-t-1


wtvaivr-w-1

Version 2: based on the frequency of the pixels with the same color (unknown of how many characters)

Color Selection Criteria:

- First algorithm - version 1:
    - The number of colors extracted is based solely on the length of the image's filename.
    - It extracts the most frequent colors up to that predetermined number, without considering sudden drops in color frequency.
- Second algorithm – version 2:
    - Dynamically selects colors based on their frequencies, specifically looking for **a significant frequency drop** (a sudden decrease by more than half).

```
[(np.uint8(58), np.uint8(249), np.uint8(158)), (np.uint8(228), np.uint8(174),
[489, 401, 399, 374, 348, 282, 151, 150, 55, 48, 47, 43, 42, 30, 29, 28, 22,
Saved ./try/0024miih-0-0-1.png
Saved ./try/0024miih-0-0-2.png
Saved ./try/0024miih-0-2-1.png
Saved ./try/0024miih-0-4-1.png
Saved ./try/0024miih-0-m-1.png
Saved ./try/0024miih-0-i-1.png
Saved ./try/0024miih-0-i-2.png
Saved ./try/0024miih-0-h-1.png
```

    - Always selects at least 4 colors, but no more than 8 colors.
    - If no sharp drop is found, it defaults to extracting the top 8 colors.

Accuracy:

- Train:
    - More than real length: 937
    - Less than real length: 328

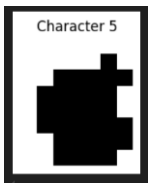
    - **accuracy = 1 – (937+328)/7832 = 83.85%**
- Test:
    - More than real length: 245
    - Less than real length: 75
    - **accuracy = 1 – (245+75)/1967 = 83.73%**

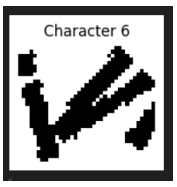
Summary Table of Differences:

Aspect	First Algorithm	Second Algorithm
Color Selection Method	Filename length	Frequency drop (>50%) with min/max bounds
Number of Colors	Fixed (filename-dependent)	Dynamic (4 to 8, frequency-dependent)
Robustness	Less robust, dependent on naming	More robust, adaptive to image data

****Advantages: Compare with CFS and Drop-Fall Algorithm****:

1. Characters like "i" and "j" will not be mistakenly split into two images just because their pixels are not connected.





2. When characters overlap, segmentation issues won't cause parts of one character to end up in another character's segment.

3. Even though some parts of a character might be lost due to overlapping with others, the overall shape remains recognizable.

****Limitations****

1. Designed for specific dataset (project) only.

Data preprocessing

Step 0: Original image

Step 1: Convert to grayscale

Step 2: Binarization

Step 3: Find contours

Step 4: Get bounding box and crop

Step 5: Pad image to a square shape

Step 6: Resize to standard dimensions (32x32)

Step 7: Normalize pixel values

```
normalized = resized / 255.0
```

example_0_original

example_1_gray

example_2_binary

example_3_contours

example_4_bbox

example_5_cropped

example_6_padded

example_7_resized

Model

1.SVM

```
# === 2. Train SVM model ===
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)

clf = SVC(
    kernel='rbf',
    C=1.0,
    gamma='scale',
    probability=True,
    random_state=42
)
clf.fit(X_train, y_train_encoded)
```

```
SVC
SVC(probability=True, random_state=42)
```

Training dataset:

	precision	recall	f1-score	support
0	0.72	0.71	0.72	1255
1	0.60	0.72	0.65	1271
2	0.92	0.89	0.90	1184
3	0.92	0.90	0.91	1275
4	0.88	0.89	0.88	1246
5	0.92	0.87	0.89	1229
6	0.92	0.89	0.90	1243
7	0.87	0.89	0.88	1208
8	0.91	0.86	0.89	1215
9	0.92	0.86	0.89	1250
a	0.90	0.84	0.87	1231
b	0.92	0.86	0.89	1267

c	0.87	0.90	0.89	1234
d	0.92	0.86	0.89	1292
e	0.93	0.86	0.89	1275
f	0.89	0.87	0.88	1275
g	0.91	0.85	0.88	1283
h	0.80	0.89	0.84	1288
i	0.56	0.81	0.66	1266
j	0.81	0.84	0.82	1203
k	0.85	0.87	0.86	1276
l	0.76	0.67	0.71	1246
m	0.93	0.91	0.92	1250
n	0.89	0.88	0.88	1289
o	0.72	0.68	0.70	1249
p	0.91	0.89	0.90	1283
q	0.85	0.84	0.84	1306
r	0.92	0.81	0.86	1261
s	0.85	0.84	0.85	1253
t	0.80	0.84	0.82	1266
u	0.86	0.90	0.88	1239
v	0.91	0.82	0.86	1255
w	0.95	0.91	0.93	1233
x	0.90	0.88	0.89	1293
y	0.71	0.89	0.79	1232
z	0.94	0.88	0.91	1241
accuracy			0.85	45162
macro avg	0.86	0.85	0.85	45162
weighted avg	0.86	0.85	0.85	45162

Test dataset:

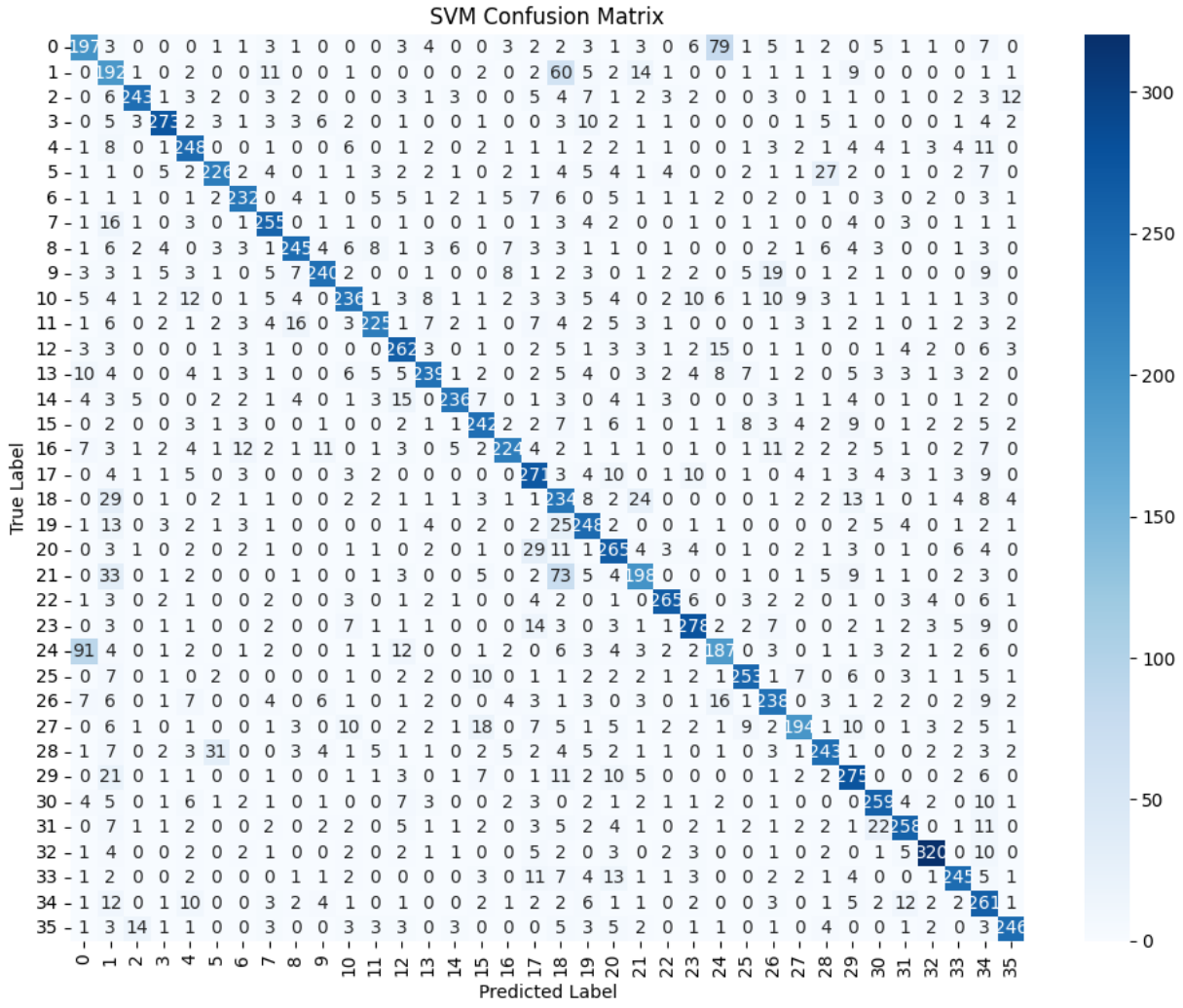
Classification Report:

	precision	recall	f1-score	support
0	0.57	0.59	0.58	335
1	0.44	0.62	0.51	308
2	0.88	0.77	0.82	314
3	0.87	0.82	0.84	334
4	0.73	0.79	0.76	313
5	0.80	0.72	0.76	315
6	0.83	0.78	0.80	297
7	0.78	0.84	0.81	304
8	0.82	0.74	0.78	329
9	0.85	0.73	0.79	327
a	0.77	0.67	0.72	350
b	0.84	0.72	0.78	312
c	0.74	0.80	0.77	327
d	0.81	0.71	0.76	336
e	0.88	0.77	0.82	308
f	0.76	0.77	0.76	315
g	0.83	0.70	0.76	322
h	0.67	0.78	0.72	347
i	0.45	0.67	0.54	350
j	0.70	0.76	0.73	325
k	0.70	0.76	0.73	349
l	0.69	0.56	0.62	351
m	0.87	0.84	0.86	316
n	0.80	0.79	0.80	350

o	0.58	0.54	0.56	344
p	0.84	0.80	0.82	315
q	0.71	0.73	0.72	326
r	0.78	0.65	0.71	297
s	0.75	0.72	0.73	339
t	0.71	0.78	0.74	354
u	0.79	0.80	0.80	322
v	0.81	0.75	0.78	344
w	0.90	0.86	0.88	370
x	0.81	0.78	0.80	313
y	0.58	0.77	0.66	341
z	0.86	0.80	0.83	309
accuracy			0.74	11808
macro avg	0.76	0.74	0.75	11808
weighted avg	0.75	0.74	0.74	11808

Accuracy: 0.7412771002710027

Confusion Matrix:



**** Captcha accuracy = 0**

2.CNN

Version 1: (base)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	320
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 128)	295,040
dense_1 (Dense)	(None, 36)	4,644

Total params: 318,500 (1.21 MB)

Trainable params: 318,500 (1.21 MB)

Non-trainable params: 0 (0.00 B)

Version 2:

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 30, 30, 32)	320
batch_normalization_3 (BatchNormalization)	(None, 30, 30, 32)	128
max_pooling2d_5 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_6 (Conv2D)	(None, 13, 13, 64)	18,496
batch_normalization_4 (BatchNormalization)	(None, 13, 13, 64)	256
max_pooling2d_6 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_7 (Conv2D)	(None, 4, 4, 128)	73,856
batch_normalization_5 (BatchNormalization)	(None, 4, 4, 128)	512
max_pooling2d_7 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_2 (Flatten)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 36)	9,252

Total params: 234,148 (914.64 KB)

Trainable params: 233,700 (912.89 KB)

Non-trainable params: 448 (1.75 KB)

Using batch:

```
history = model.fit(
    X_train,
    y_train_encoded,
    epochs=50,
    batch_size=64,
    validation_split=0.1,
    callbacks=[early_stopping, timing]
)
```

Data processing:

A method of ImageDataGenerator in Keras that calculates **certain normalization statistics** before data augmenntation

```
datagen = ImageDataGenerator(
    rotation_range=10,           # random rotate degree
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,              # random zoom range
    horizontal_flip=False        # not using horizontal_flip in char
    featurewise_center=True,
    featurewise_std_normalization=True
)

datagen.fit(X_train)
```

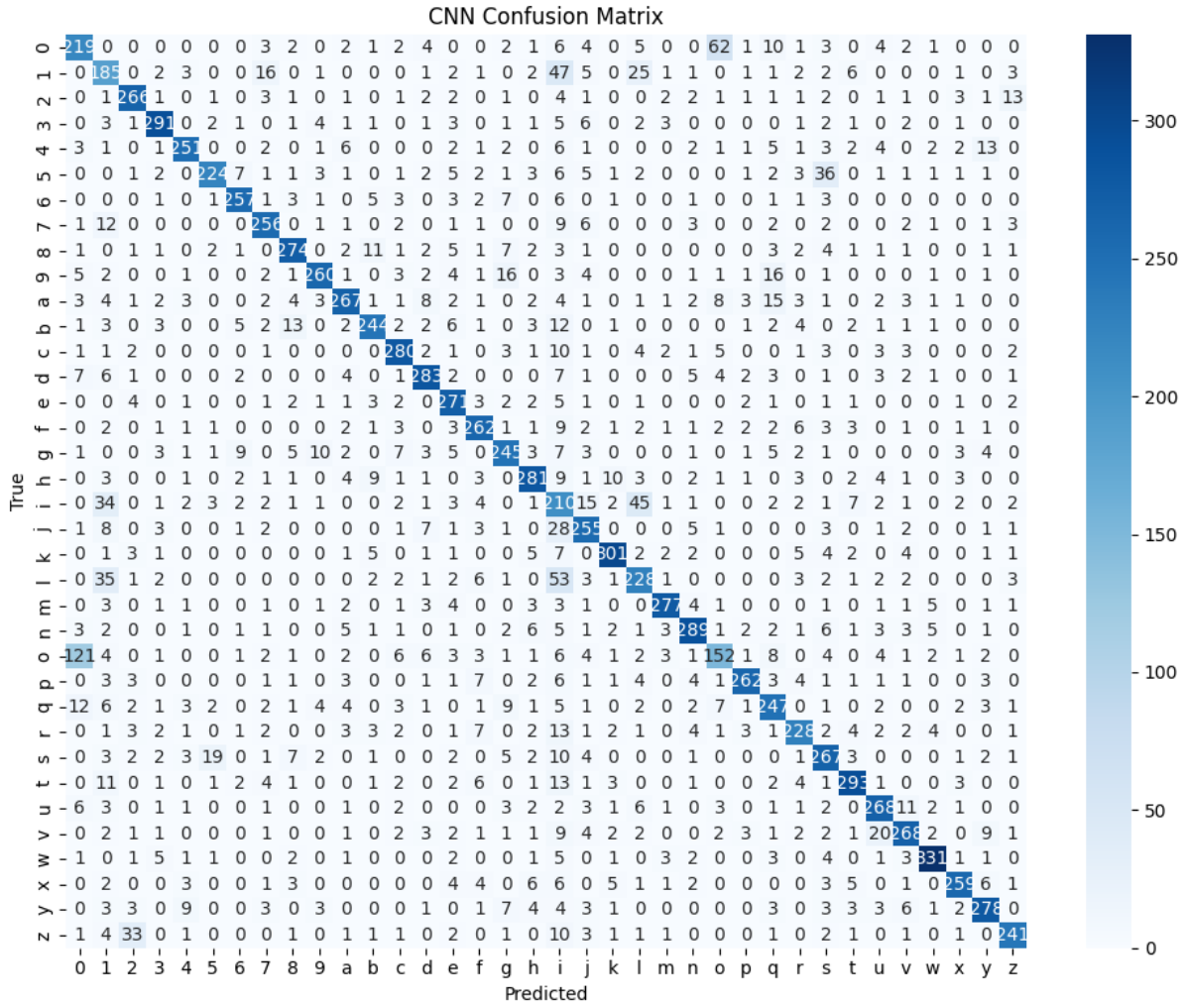
Classification Report: (Test)

	precision	recall	f1-score	support
0	0.57	0.65	0.61	335
1	0.53	0.60	0.56	308
2	0.81	0.85	0.83	314
3	0.88	0.87	0.88	334
4	0.87	0.80	0.84	313
5	0.87	0.71	0.78	315
6	0.88	0.87	0.87	297
7	0.81	0.84	0.83	304
8	0.84	0.83	0.84	329
9	0.88	0.80	0.83	327
a	0.83	0.76	0.80	350
b	0.84	0.78	0.81	312
c	0.84	0.86	0.85	327
d	0.84	0.84	0.84	336
e	0.78	0.88	0.83	308
f	0.81	0.83	0.82	315

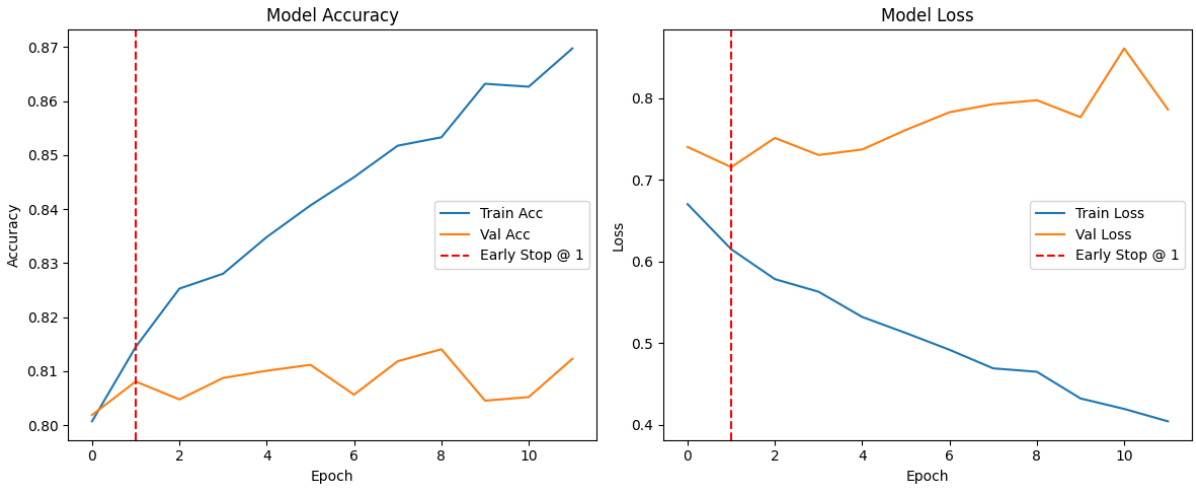
g	0.77	0.76	0.76	322
h	0.83	0.81	0.82	347
i	0.38	0.60	0.47	350
j	0.74	0.78	0.76	325
k	0.89	0.86	0.88	349
l	0.67	0.65	0.66	351
m	0.91	0.88	0.89	316
n	0.85	0.83	0.84	350
o	0.60	0.44	0.51	344
p	0.90	0.83	0.87	315
q	0.72	0.76	0.74	326
r	0.81	0.77	0.79	297
s	0.71	0.79	0.75	339
t	0.86	0.83	0.84	354
u	0.80	0.83	0.82	322
v	0.82	0.78	0.80	344
w	0.92	0.89	0.91	370
x	0.89	0.83	0.86	313
y	0.84	0.82	0.83	341
z	0.86	0.78	0.82	309
accuracy			0.79	11808
macro avg	0.80	0.79	0.79	11808
weighted avg	0.80	0.79	0.79	11808

Accuracy: 0.7850609756097561

Confusion Matrix:



Model accuracy / loss plot:



**** Captcha accuracy: 0.7851** (segmatation version 1: known length)

💡 **Key Observations**

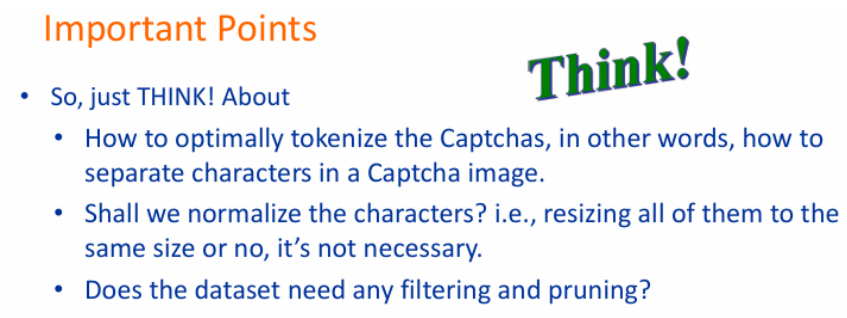
- **CNN consistently outperforms SVM** in most metrics: accuracy, precision, recall, and F1. (likely due to CNN's ability to learn hierarchical features, which traditional SVMs cannot do as effectively)
- **Hard-to-classify characters** like i, o, 1, and 0 have **low performance across both models**, possibly due to **visual similarity**.
- CNN shows **significantly better performance** on complex, high-variation characters such as 4, 6, 9, e, and w.

CNN Highlights:

- Performs **consistently well** across most classes, especially:
  - m: 0.91 precision, 0.88 recall
  - w: 0.92 precision, 0.89 recall
  - k, p, t: f1-scores ≥ 0.84
- Weakest classes:
  - i: 0.38 precision, 0.60 recall → difficult to distinguish
  - o: 0.60 precision, 0.44 recall

SVM Highlights:

- Also strong for m, w, k, and p – but slightly behind CNN on many of these.
- Shows a **notably better recall** for:
  - 1: 0.62 recall vs CNN's 0.60
  - 4: 0.79 recall vs CNN's 0.80
- Generally **lower precision**, meaning more false positives than CNN.



- Segmatation by colors (version 1 and version 2)
- Yes, resizing them to 32x32, using grayscale and binarization to eliminate the noisy generated by colour
- Training dataset does filtering by removing some uncleaned and hard tell Captcha images → accuracy of CNN model increase around 0.03

Folder / Files

Segmentation by Color:

- Summary.pdf
- Preprocess_idea.doc
- comparation_algos_segment.ipynb
  --- comparation of all segmentation algos, eg. CFS, Drop-Fall, version 1, version 2
- dataset
  - train_data_process.ipynb
  - test_data_process.ipynb
  - valid_train --- Captcha images without errors
  - valid_test
  - clean_train --- Captcha images without black lines (training data)
  - clean_test
  - segmat_train_filename ---Characters extracted by version 1 algo (training data)
  - segmat_test_filename ---Characters extracted by version 1 algo (test data)
  - segmat_train_frequency ---Characters extracted by version 1 algo (training data)
    - more --- Characters extracted more than the actual length
    - more_list.txt
      --- A list of Captchas that characters extracted more than actual length
    - less --- Characters extracted less than the actual length
    - less_list.txt
      --- A list of Captchas that characters extracted less than actual length
    - color_extraction_results.csv --- Contains all the information about test data
  - segmat_test_frequency ---Characters extracted by version 1 algo (test data)
    - more
    - more_list.txt
    - less
    - less_list.txt
    - color_extraction_results.csv

1wu0cul-0.pn	1wu0cul	7	[(np.uint8(47), np.uir	[242, 190, 157, 143, 140,	8 more
1x44n7nx-0.pr	1x44n7nx	8	[(np.uint8(72), np.uir	[24, 23, 21, 21, 20, 18, 16,	8 same
1x8ljpi-0.png	1x8ljpi	7	[(np.uint8(133), np.u	[76, 63, 54, 48, 36, 30, 29,	7 same
1xut0w4-0.pn	1xut0w4	7	[(np.uint8(37), np.uir	[89, 84, 63, 59, 47, 44, 42,	8 more
20ka8wmo-0.1	20ka8wmo	8	[(np.uint8(28), np.uir	[292, 273, 256, 238, 222,	8 same
21iyxsn-0.png	21iyxsn	7	[(np.uint8(36), np.uir	[340, 337, 247, 246, 184,	5 less
21z1y-0.png	21z1y	5	[(np.uint8(87), np.uir	[362, 288, 209, 132, 130,	5 same

  - train_dataset --- Character images after regularization (ready for training)
    - X_features.npy
      --- Contains all the features information, can be read for training directly
    - y_labels.npy
      --- Contains all the y labels, can be read for training directly

```
X_train = np.load('train_dataset/X_features.npy')
y_train = np.load('train_dataset/y_labels.npy')
```

- test_dataset --- Character images after regularization (ready for training)
  - X_features.npy
  - y_labels.npy
- data_preprocess_example
  - data_preprocess_Standardized.ipynb
  - preprocess_steps --- Contains the results of each step


example_0_original


example_1_gray


example_2_binary


example_3_contours


example_4_bbox


example_5_cropped


example_6_padded


example_7_resized

- models
  - SVM
    - SVM.ipynb
    - svm.pkl --- SVM model
    - svm_predictions.csv
    - confusion_matrix_svm.png
  - CNN
    - CNN.ipynb
    - cnn_model.h5
    - cnn_predictions.csv
    - confusion_matrix_cnn.png
    - cnn_training_curves.png