Graph-Based Method for Oracle Bone Inscriptions Recognition

Haibin Lai
12211612@mail.sustech.edu.cn
Southern University of Science and Technology
Shenzhen, Guangdong, China

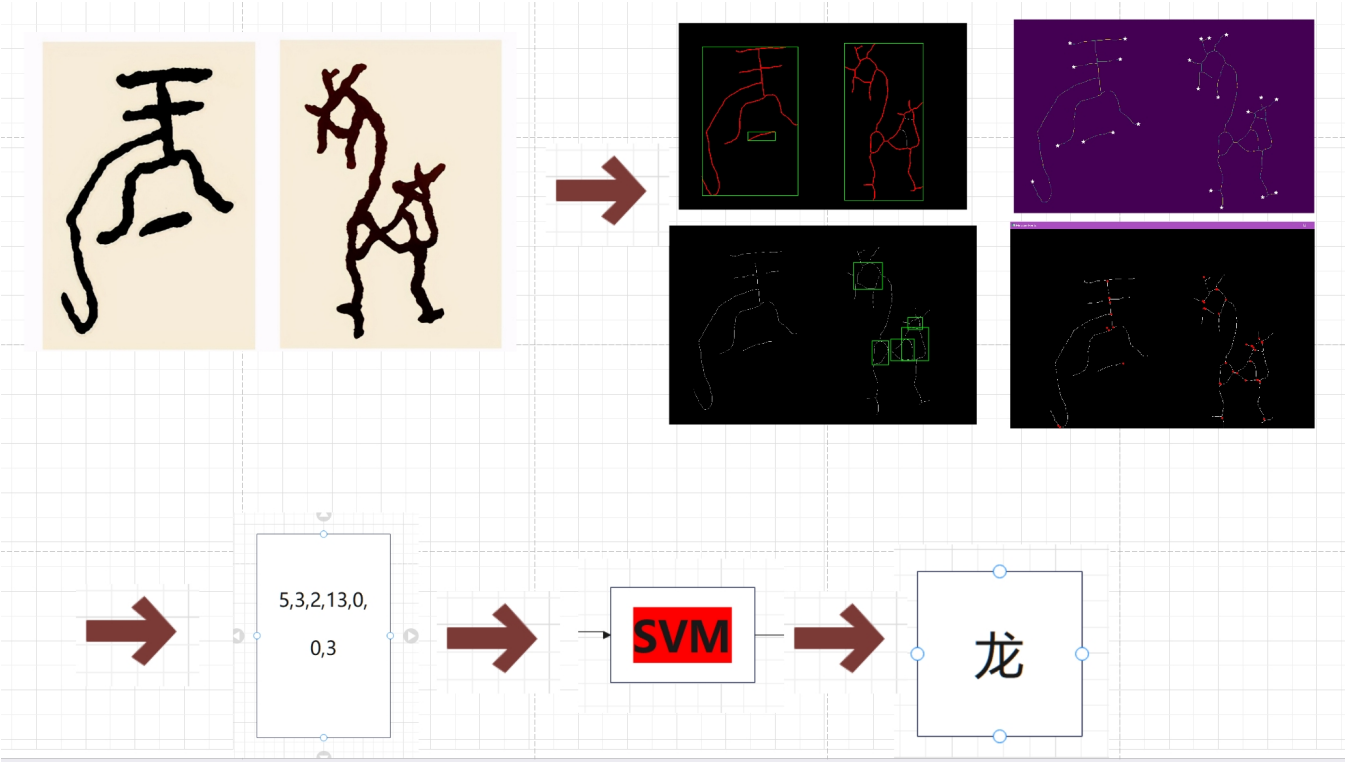

Figure 1. Oracle Bone Inscriptions Recognition

Abstract

In this *Discrete Mathematics(H)* project, we recognize Oracle as **undirected graph** based on its graph characteristics by designing a python program. After inputting pictures of Oracle, it will first pre-process it into skeleton picture with the help of **OpenCV** library.

Then, we consider the skeleton picture as **undirected graph** and recognize its feature like endpoints (vertex that degree is 1), blocks and so on. Since these features are marvelously similar to the features in fingerprint recognition, we use some algorithm from **Fingerprint Recognition** like *Gabor Filter* to improve our recognition in oracle skeleton.

After that, as an example, we store the feature message of 5 Oracle as vectors. Then we implemented **Support Vector Machine** to help distinguish whether the new picture input has the probability in the 5 Oracle characters.

CCS Concepts: • **Mathematics of computing** → **Graph theory**; • **Applied computing** → **Arts and humanities**; • **Computing methodologies** → *Computer vision.*

Keywords: Oracle Bone Inscriptions, Graph Theory, Image Recognition

ACM Reference Format:
Haibin Lai. 2023. Graph-Based Method for Oracle Bone Inscriptions Recognition. In *Proceedings of SUSTech (Discrete Mathematics(H) Project).* ACM, New York, NY, USA, 8 pages.

1 Introduction

Oracle Bone Inscriptions represent the earliest mature writing system discovered in China. They serve as the origin of Chinese characters, and their significant cultural value and historical importance have gained recognition worldwide. As a literal, Oracle is combined with exact "line" and "dot",

which has formed a stable topological structure inside an Oracle.

Delving into the realm of ancient Oracle bone inscriptions, our project employs a approach that consider Oracle as an undirected graph in order to recognize it. Leveraging the OpenCV library, we preprocess images of Oracle bones into skeletonized graphs. Inspired by fingerprint recognition, we employ algorithms like the Gabor Filter to identify key features.

We take key features from 5 Oracle bone inscriptions into vectors, and a pivotal role is played by Support Vector Machine, SVM. This SVM model aids in distinguishing whether a new image shares characteristics with the predefined Oracle characters.

Our approach merges ancient artifacts with modern computational methods, contributing to the advancement of oracle bone studies, holding substantial implications for the field of digital humanities and ancient Chinese culture research.

2 Oracle Picture Choosing

We mainly choose our Oracle pictures from the following database:

1. China Character Resource System (bnu.edu.cn) A China Character Database developed by Beijing Normal University.

Figure 2. China Character Resource System

2. Fudan Oracle Database (fdgwz.org.cn) Fudan University is the earliest university in China to initiate research on Oracle. The database is one of the earliest and most authoritative Oracle database developed by Fudan University.

Figure 3. Fudan Oracle Database

3. GUOXUE Master (guoxuedashi.net) Guoxue Master is a website developed by Li Yong, an adjunct researcher at the Zhejiang Yue Culture Research Center and the Yue Culture Research Institute of Shaoxing Wenli College. He is also a researcher at the China Xi Shi Culture Research Center, a member of the expert database in philosophy and social sciences in Shaoxing, and a member of the Zhuji Writers Association.

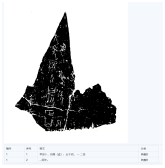

Figure 4. GUOXUE Master

3 Pre-processing

After taking oracle pictures, we first need to turn them into **skeleton** pictures to help algorithm to recognize them. The following figure shows the process of Oracle 'San'.

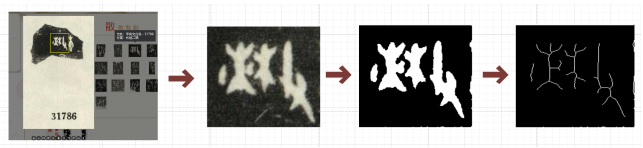

Figure 5. Pre-processing steps: Loading, threshing,skeletonizing

First, the picture is download from the internet. Then it's Binarize into Binarization picutre so that the Skeleton Algorithm can easily recognize the bone. If the picture contains noise that deffer the recognizing result, optionally we can use Convolution Kernel Algorithm to lower the noise of the picture and help with our recognition. After that, we can implement Skeletonization on picture to get pre-processed picture like 'San'. The pictures 6, 7, 8, 9 are the steps that we do in pre-processing.

```python
import requests


# Download the picture
def download(file_path, picture_url):
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 "
                      "(KHTML, like Gecko) Chrome/63.0.3239.132 "
                      "Safari/537.36 QIHU 360SE",
    }
    r = requests.get(picture_url, headers=headers)
    with open(file_path, 'wb') as f:
        f.write(r.content)


# Step1 load
url = 'https://qxk.bnu.edu.cn/pic_server//images/chinesedatabase/tp-zb-cropper-img/'

jpg = '91f241991f164cacb7450afe7f2fee82.jpg!70!10000000'

url = url+jpg
filePath = 'D:/Python/JiaGuWen/JiaGuWen/JiaGuWen.png'
download(filePath, url)
```

Figure 6. Downloading Pictures

```python
img = cv2.imread(filePath)
cv2.imshow('img', img)
cv2.waitKey()

# Step2 Grey Picture
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Step3 Turn Picture into binary part
_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

cv2.imshow('img', binary)
cv2.waitKey()
```

Figure 7. Image Segmentation by Binarization

```python
# Step4 take out skeleton
binary[binary == 255] = 1
skeleton0 = morphology.skeletonize(binary)
skeleton = skeleton0.astype(np.uint8) * 255
cv2.imshow('img', skeleton)
cv2.waitKey()
# save your skeleton picture
cv2.imwrite("skeletonWithKernelM028_2.png", skeleton)
```

Figure 8. Skeletonize pictures

```python
#  ##################### Optional Choice: Lower the noise of image#####################
#  Convolution Kernel
kernel = np.ones((3, 3), np.float32) / 10
# Convolution, make the tunnel the same as original picture
dst = cv2.filter2D(img, -1, kernel)


#  read your image that is in low noise
img = cv2.imread('1.png', 0)  # read image
# _,binary = cv2.threshold(img, 200, 255, cv2.THRESH_BINARY_INV)  # thresh binary inv
cv2.imwrite("binary.png", binary)  # save your binary thresh png

cv2.imshow('img', binary)
cv2.waitKey()
#  ###############################################################################
```

Figure 9. Lower the noise of picture using Convolution Kernel

4 Features Extraction

To recognize and store our Oracle, we need to find graph features of Oracle. We first give definition of some graph features of the Oracle referencing passage[1].

Oracle is in the embryo time of Chinese Character, so it doesn't have strokes of a Chinese character like Horizontal, Vertical, Dot, and Hook. However, as a literal, Oracle is combined with exact "line" and "dot", which has formed a stable **topological structure** inside an Oracle. So, we consider three classes of features to identify Oracle: *Primary Identification, Secondary Identification.*

As an example, we use the Oracle literal "Long" (Means Oriental Dragon) and "Lin"(Represent the scale of the Dragon) as an dealing example (figure10). With two character as comparison, we can also test that whether the feature choosing is plausible, while in data loading part we will use the pre-processed picture from Oracle database.

Figure 10. Oracle literal "Long" and "Lin"

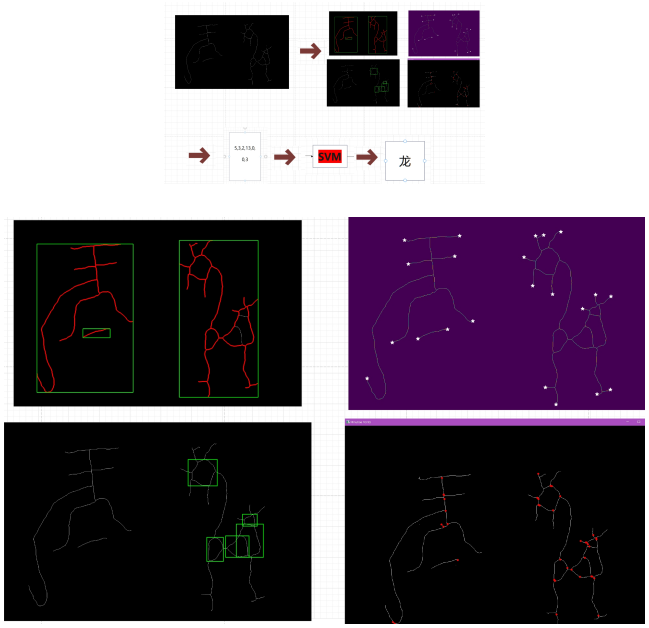

Figure 11. Primary Feature of "Long" and "Lin"

4.1 Primary Feature Identification Code

In Primary Feature Identification, we extract the most significant features of Oracle. They are:

1. **Blocks** Count
2. **Endpoints** Count
3. **Mesh** Count
4. **Trifurcation** Point Count

And we will explain them with the example figure 11.

(1) **Blocks** Count: A connected **subgraph** is referred to as a 'block.' The number of blocks containing blocks in an undirected graph is called the block count of the graph. For the character 'Long' , there are 2 connected subgraphs (as the top left figure on11 shows), and its block count is 2. For the character 'Lin' , there is one connected subgraph, and its block count is 1.

(2) **Endpoints** Count: In an undirected graph, a **vertex** with a degree of 1 is called a Endpoints. The number of vertices in a graph is referred to as the vertex count of the graph. In the upper-right diagram, the oracle bone script character 'Long' has 9 vertices (marked with asterisks), so the vertex count is 9. The oracle bone script character 'Lin' has 13 vertices, resulting in a vertex count of 13.

(3) **Mesh** Count: In an undirected graph, a closed region surrounded by edges is referred to as a mesh (mesh is a concept in planar graphs, and graphs constructed with oracle bone script characters are planar graphs). The number of meshes contained in a graph is called the mesh count of the graph. As shown in the bottom-left diagram, 'Long' does not have any meshes, so the mesh count is 0. 'Lin' has 5 meshes, resulting in a mesh count of 5.

(4) **Trifurcation Point Count**: A point with a degree of 3 is called a trifurcation point. The number of trifurcation points in a graph is referred to as the trifurcation point count of the graph. Using the trifurcation point count can provide a good understanding of the topological structure of oracle bone scripts.

Of course, we can also define quadrifurcation point count, quintifurcation point count, hexifurcation point count, and so on. However, in practical detection, there are not as many occurrences of these cross-points. Trifurcation points can effectively reflect the intersections of heads within oracle bone scripts, intersections of loops within oracle bone scripts and other radicals, etc., making it an excellent parameter. However, with excellence comes many recognition challenges, and the recognition of trifurcation points is particularly challenging.

Here, we have employed a recognition algorithm for cross-points used in fingerprint recognition, resulting in some errors (which will be explained in the following sections). In the bottom-right diagram, the character 'long' has 5 trifurcation points. However, the algorithm identified a total of 9 trifurcation points. The character 'lin' has 19 trifurcation points, but it was identified with 20 trifurcation points. Fortunately, despite these discrepancies, and after storing the trifurcation point count in a database, the recognition algorithm can still correctly identify that new images of 'lin' and 'long' respectively have 9 and 19 trifurcation points.

4.1.1 Definition 1: Oracle Graph Code Type A.

Oracle Graph Code Type A (D_1): The sequence obtained by arranging the six graph features mentioned above in the order of block count, mesh count, vertex count, and trifurcation point count is referred to as the Oracle Graph Code Type A. It is denoted as D_1.

For example, for "Long", its Oracle Graph Code Type A is :
$D_L = (2, 9, 0, 9)$
For "Lin" , its Oracle Graph Code Type A is :
$D_I = (1, 13, 5, 19)$

Oracle Graph Code Type A gives serveral features that Oracle contains, so that it has a high level of discrimination between each of the Oracle script. And when we compare most of the Oracle, they can be divided or spotted by Type A code without the following precedure.

4.2 Implementation of Primary Feature Identification Code

(1) Implementation of Blocks Count

In this part, we use Contour Detection in OpenCV library to detect the Blocks of the Oracle. The core codes are shown on figure 12.

```python
# 执行轮廓检测
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
cv2.imshow('Image with Stars', img)
cv2.waitKey(0)
contours, hierarchy = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

print(len(contours))

cv2.drawContours(img, contours, -1, (0,0,255), 3)

# 遍历轮廓
for contour in contours:
    # 假设 contour 是你的轮廓

    # 计算轮廓的周长
    perimeter = cv2.arcLength(contour, True)
    # 近似轮廓，其中epsilon是从原始轮廓到近似轮廓的最大距离
    epsilon = 0.5 * perimeter
    approx = cv2.approxPolyDP(contour, epsilon, True)

    # 如果近似的轮廓是闭合的，说明是一个闭环
    # 获取轮廓的边界框
    x, y, w, h = cv2.boundingRect(contour)
    # 绘制边界框
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

# 显示结果
cv2.imshow('Detected Closed Loop', img)
cv2.imwrite("output.png", img)
```

Figure 12. Contour Detection For Blocks

(2) Implementation of Endpoints Count

Endpoints is a significant features in Oracle recognition. Detecting the endpoint is one of the hardest during the whole research since we need to master OpenCV well and few people have deal Endpoints in Oracle. Luckily, one day I consult Professor Shiqi Yu, who taught me the class Principle of Database(H) on fall 2023. Professor Yu introduced me with Fingerprint Recognition Algorithm on OpenCV, which gave me inspiration. Then some research on fingerprint was done.

As Jing Chen said in his paper, we can find endpoints and even Trifurcation Point by using a 3×3 kernel. That gives the basic idea of Endpoints Counting Algorithm.

After that, we get learn from Skelecton spots and cross points recognition (zhihu.com) Which introduce that for the following situation, we can say that the centural white point is a endpoint.

And here we get the core code of Endpoint Counting as figure 15 shows.

(3) Implementation of Mesh Count

For Mesh Count, we use the SimpleBlobDetector algorithm in OpenCV. SimpleBlobDetector is a part of the OpenCV

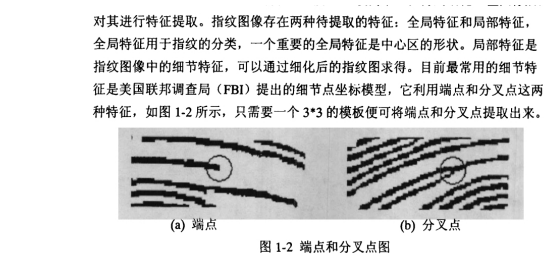

Figure 13. Endpoints Count and Trifurcation Points Count on Finger Point Recognition

Figure 14. Endpoints Recognition from https://zhuanlan.zhihu.com/p/653911423

```python
def endpoints_of_skeleton(center_line_img):
    W, H = center_line_img.shape
    img = center_line_img
    endpoints = []
    x, y = np.where(img > 0)
    for i, j in zip(x, y):
        if i - 1 < 0 or j - 1 < 0 \
                or i + 1 >= W or j + 1 >= H:
            continue
            # 条件A 该点的四领域内仅有一个点与之连接
        c_a = (img[i - 1, j] + img[i + 1, j] + img[i, j - 1] + img[i, j + 1]) == 255
        # 条件B 该点的对角领域内仅有一个点与之连接
        c_b = (img[i - 1, j - 1] + img[i - 1, j + 1] + img[i + 1, j - 1] + img[i + 1, j + 1]) == 255
        # 该点的八领域内仅有一个点
        if (img[i - 1:i + 2, j - 1:j + 2] == 255).sum() <= 2:
            endpoints.append([i, j])
        # 同时满足两个条件AB，且设置八种其余的端点坐标情况，如：
        '''
           ##0  0##
           0#0  0#0   ...
           000  000
        '''
        if c_a and c_b:
            if (img[i - 1, j - 1:j + 2] == 255).sum() == 2 \
                    or (img[i + 1, j - 1:j + 2] == 255).sum() == 2 \
                    or (img[i - 1:i + 2, j - 1] == 255).sum() == 2 \
                    or (img[i - 1:i + 2, j + 1] == 255).sum() == 2:
                endpoints.append([i, j])
    return endpoints
```

Figure 15. Endpoint Counting Algorithm

library, which is a popular computer vision library in Python. This algorithm is designed for detecting and extracting blobs or keypoints in an image.

The SimpleBlobDetector is used for identifying and extracting blobs, which are regions in an image that share common properties such as color or intensity. It's also useful for us to detect the Mesh in Oracle.

Here we introduce some of the key features of SimpleBlobDetector:

1. **Thresholding:** The algorithm typically involves thresholding the image to segment it into regions of interest.
2. **Filtering:** It applies filtering based on certain characteristics such as size, circularity, and color to identify blobs.
3. **Centroid and Size:** The algorithm provides information about the centroid (center) and size of each detected blob.
4. **Parameters:** SimpleBlobDetector allows you to set parameters like threshold values, filter by color, and specify the minimum and maximum size of blobs.
5. **Easy Integration:** Being part of OpenCV, it seamlessly integrates with other image processing and computer vision functions available in the library.

And the Key Code of Mesh Count is shown as figure 16.

```python
# Set up the detector with default parameters.
detector = cv2.SimpleBlobDetector_create()

# Detect blobs.
keypoints = detector.detect(im)

# Draw detected blobs as red circles.
# cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the circle corresponds to the size of blob
im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]),
                                      (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

Figure 16. Mesh Count Algorithm using SimpleBlobDetector

(4) Implementation of Trifurcation Point Count

After we find out the powerful of Fingerprint Recognition, we tried the algorithm that use kernal to detect Trifurcation point. However, the result is not as well as we expect. So we went back to the Fingerprint algorithm to find out if we can get some of the algorithm to work. Soon we find that we mainly come across trouble in pictures' changing during our pre-processing. So we need to find an algorithm to deal picture in a suitable way. And that we finally found: Gabor filtering.

The 2D Gabor filter, introduced by Daugman in 1980, has found widespread applications in texture classification, texture segmentation, and biometric feature recognition.

The Gabor filter is a mathematical model inspired by the human visual system's ability to analyze spatial frequencies. It is particularly effective in capturing texture information in images. The filter is defined by a complex sinusoidal function modulated by a Gaussian envelope. The Gabor filter is characterized by two main parameters: frequency (controls the number of oscillations) and orientation (determines the preferred orientation of the filter).

The mathematical representation of the 2D Gabor filter can be expressed as follows:

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

Where:

$$x' = x\cos(\theta) + y\sin(\theta)$$
$$y' = -x\sin(\theta) + y\cos(\theta)$$

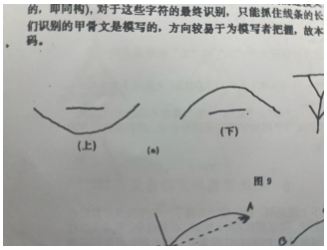

Figure 18. Shang and Xia are isomorphism but different

λ is the wavelength of the sinusoidal factor, θ is the orientation of the normal to the parallel stripes of the Gabor function, ψ is the phase offset, σ is the standard deviation of the Gaussian envelope, and γ is the spatial aspect ratio.

In fingerprint recognition, Gabor filters are utilized to enhance the ridge and valley structures of fingerprints. This helps in creating distinctive feature representations for matching fingerprints.

So we first implement Gabor filter like fingerprint recognition, then we use contours algorithm to find the Trifurcation point as figure shows 17.

```python
def extract_minutiae(img_path):
    # 读取图像
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

    # 使用Gabor滤波增强图像的ridge特征
    gabor_kernel = cv2.getGaborKernel((10, 10), 10.0, np.pi/4, 11.0, 0.6, 0, ktype=cv2.CV_32F)
    img_filtered = cv2.filter2D(img, cv2.CV_8UC3, gabor_kernel)

    # 查找图像中的轮廓
    contours, _ = cv2.findContours(img_filtered, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    minutiae_list = []

    # 遍历轮廓，提取Minutiae特征点
    for contour in contours:
        for point in contour:
            x, y = point[0]
            if img[y, x] == 255:  # 仅处理二值化后的白色区域
                minutiae_list.append((x, y))

    return minutiae_list
```

Figure 17. Trifurcation Point using Gabor filter

5 Secondary Feature Identification Code

Since some of the Oracle are isomorphism and only different in direction, we use fingerprint Similarity Algorithm to find if they are the same literal or the literal like 'shang' and 'xia'18.

This is inspired by the fingerprint algorithm in[2] as figure19 shows.

Using SVM to aid Oracle Recognition

Support Vector Machine (SVM) is a powerful machine learning algorithm that can be used for classification problems, and its application in oracle bone script recognition highlights its outstanding performance. SVM seeks to find an optimal decision boundary between different classes by maximizing the margin between this boundary and samples of each class, enabling accurate classification of new samples.

```python
def extract_features(image):
    # 灰度化
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # 计算梯度
    gradient_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    gradient_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
    # 计算梯度幅值和方向
    magnitude = np.sqrt(gradient_x ** 2 + gradient_y ** 2)
    direction = np.arctan2(gradient_y, gradient_x)

    # 对方向场进行平滑处理
    smoothed_direction = cv2.GaussianBlur(direction, (0, 0), 2)

    # 提取特征，可以根据具体需求选择不同的特征
    features = smoothed_direction.flatten()

    return features

def fingerprint_matching(given_image, database_images):
    # 提取给定图片的特征
    given_image_features = extract_features(given_image)

    # 提取数据库中的特征
    database_features = [extract_features(img) for img in database_images]

    # 计算余弦相似度
    similarity_scores = cosine_similarity([given_image_features], database_features)

    return similarity_scores[0]
```

Figure 19. Direction dectet

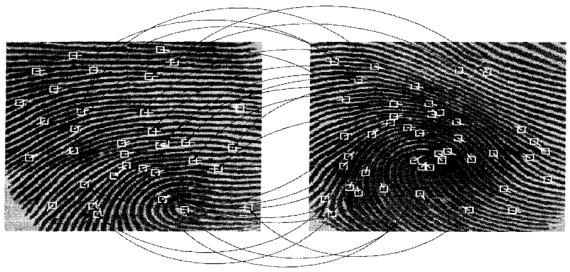

Figure 16.7 Fingerprint matching.

Figure 20. Direction dectet on fingerprint

In oracle bone script recognition, we leverage the SVC (Support Vector Classification) module provided by the scikit-learn library. By selecting an appropriate kernel function for similarity calculation, we enhance the model's classification performance. The choice of the kernel function plays a crucial role in oracle bone script image processing, as it determines the mapping of samples in a high-dimensional space, making data separation more feasible in the new feature space.

Through the use of SVM, we can effectively differentiate the features of different characters in oracle bone script, facilitating accurate recognition. The following codes on figure 21 are the SVM implementation in scikit-learn.

6 Experiment Result

In the experiment we done the program with process in figure 22 We do 5 Character into the codes and do SVM to distribute them. Then we input a new graph and try to find

```python
# our vector
X = np.read('D:\Python\JiaGuWen\JiaGuDetect\Loop\output.np', 4)  # 100个样本，每个样本4维向量

y = np.read.choice('D:\Python\JiaGuWen\JiaGuDetect\Loop\y.np' ,size=20)  # 二分类标签

# train and test lib
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# build SVM divider
svm_classifier = SVC(kernel='linear')  # using linear kernel

# train model
svm_classifier.fit(X_train, y_train)

# testing on the model
y_pred = svm_classifier.predict(X_test)

# compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Figure 21. SVM

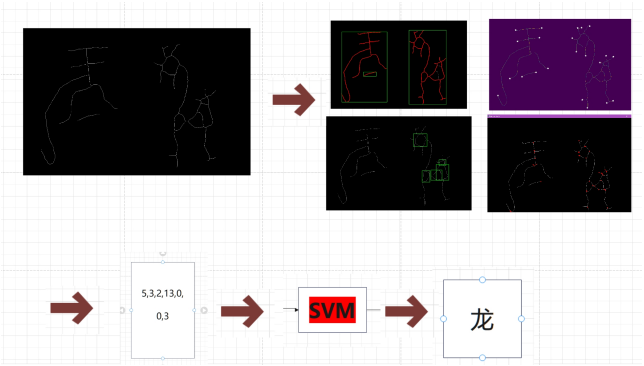

Figure 22. Program's process

Figure 23. Program's result

out if it can recognize it. The result is that it can be done in 5 scripts.

7 More jobs can be done

7.1 Advanced Identification Code

Due to the vast quantity of oracle bone characters, after primary classification, some characters still do not reach final recognition. Upon implementing primary classification coding for 5 of oracle bone characters representing multiple words, we found that although 2 characters share the same features in type A graph codes, they are not isomorphic graphs.

In other words, even though they have the same number of blocks, meshes, endpoints, and trifurcation points, the connections between points and edges are not the same. For these overlapping characters, we propose the concept of 'neighbor point cut sets' and use this as the feature to form *Type B Graph Codes* for secondary recognition features.

Definition of **Adjacent Points**: In an undirected graph, if there is an edge connecting two vertices, then those two vertices are called adjacent points.

Definition of **Neighbor Point Subgraph**: A subgraph formed by two adjacent points and the edges connecting them is called a neighbor point subgraph.

Definition of **Cut Set of Neighbor Point Subgraph**: The minimum set of edges that must be cut to separate the neighbor point subgraph from the graph is referred to as the cut set of the neighbor point subgraph.

7.2 Vectorization and CNN Network Building

Vectorization of Oracle Bone Inscriptions: Explore and implement state-of-the-art vectorization techniques to represent the intricate details of Oracle Bone Inscriptions in a digital format. Develop a systematic approach to convert the visual information in Oracle into vectorized representations for further analysis.

7.2.1 CNN Network Building: Design and construct Convolutional Neural Network architectures tailored for the recognition and interpretation of Oracle Bone Inscriptions. Train the CNN models on the vectorized data to automatically identify characters, patterns, and semantic meanings within the inscriptions. Semantic Analysis:

Conduct semantic analysis on the vectorized and CNN-processed data to decipher meanings, correlations, and context within Oracle Bone Inscriptions. Explore the potential for automating the identification of specific characters or symbols associated with divination and ritualistic practices.

8 Conclusion

In general, we have done a approach that consider Oracle as an undirected graph in order to recognize it. Leveraging the OpenCV library, we preprocess images of Oracle bones into skeletonized graphs. Inspired by fingerprint recognition, we employ algorithms like the Gabor Filter to identify key features.

We take key features from 5 Oracle bone inscriptions into vectors, and do SVM distribution to see what happen to the machine recognition.

9 Acknowledgments

Special Thanks to Qi Wang, who taught me Discrete Mathematics(H) on Fall 2023 semester. His class is very interesting with full of Math and Computer Science knowledge (also he has very good personality which taught me a lot). I hope his textbook on the class can be done soon.

Special Thanks to Shiqi Yu, who gave me many knowledge and inspiration on OpenCV and fingerprint using. Also, database building.

Thanks to my parents, friends and other teachers in SUSTech.

This is my first time writing latex. So I don't know if something would be wrong.

10 Appendices

Source Code: View on github Laihb1106205841/JiaGuWen: Southern University of Science and Technology Discrete Mathmatics(H) Class Project (github.com)

References

[1] Xinlun Zhou Feng Li. 1996. Recognition of JIA GU WEN Based On Graph Theory. *Electronic Engineering* 41 (1996).

[2] Lin Hong. 1998. *Automatic personal identification using fingerprints*. Ph. D. Dissertation. USA. Advisor(s) Jain, Anil. AAI9909316.

A Research Methods

SVM

OpenCV

Fingerprint Algorithm

Graph Theory

Received 21 January 2024