

Counting Strokes in Chinese Characters

Yien Xu

yien.xu@wisc.edu

Yuqi Lin

ylin273@wisc.edu

Scott Lai

qlai5@wisc.edu

Abstract

We use machine learning algorithms to count stroke number of Chinese characters that are in image form. As Chinese international students who are proud of our language, we hope to find a good method that helps more people learn Chinese. Hence we make a stroke number recognition system for Chinese character that if we input a 28×28 pixels image of a Chinese character, we are able to tell how many stroke(s) it has. To unify the definition of stroke features, we use the definition in Unicode as one of our databases, which includes 36 stroke features in total. We also divide our dataset into 90% training and 10% testing sets. The models we use include k -Nearest Neighbors algorithm (k NN), Logistic Regression and Convolutional Neural Network (CNN), combined with k -Fold Cross Validation (k -Fold CV). We use Euclidean Distance for distance calculation and choose $k = 15$ for the Cross Validation in k NN algorithm, and find the test accuracy is about 20%. We also try Logistic Regression model by first inverting the image into black background, and then applying L1 regularization and using the optimizer SAGA [4] to train. We obtain a test accuracy lower than k NN. By adding Bagging to Logistic Regression, we improve the accuracy a little bit. Logistic Regression with Max-Pooling gives the lowest test accuracy among all the models we use, and the highest test accuracy we gain is about 50% by using CNN.

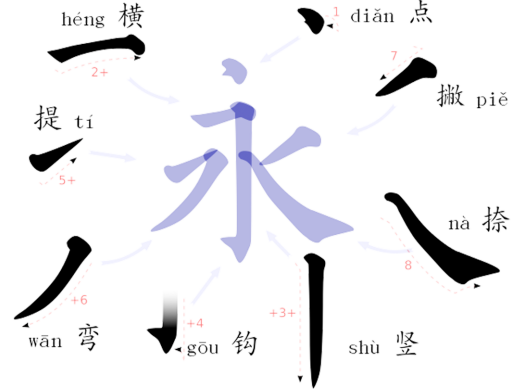
1. Introduction

As one of the oldest known language systems in the world, Chinese is the only prolific writing system that is still in use today [10]. Unlike the alphabet or syllabary, Chinese characters are comprised of different strokes, and each character is like a single word in English which has a meaning itself. There are five basic stroke features [2], including Héng (Horizontal Stroke), Shù (Vertical Stroke), Piě (Down Stroke to The Left), Diǎn (Dot), Zhé (Horizontal Stroke with A Vertical Turn) (See Figure 1). Under each basic stroke, there are subordinate stroke features. In total there are 36 stroke features (See Figure 9 of all 36 stroke features in Appendix). Take Figure 2 as an example [1].



Figure 1. Five basic stroke features of Chinese characters. From left to right: Héng, Shù, Piě, Diǎn, Zhé [6].

This character has 5 strokes in total and means "eternity" in English. These simple stroke features compose almost a hundred thousands of Chinese characters with complex forms.



Eight Principle strokes extracted from 永, means "eternity".
The total number of strokes of 永 is 5.

Figure 2. Yǒng [1]

As Chinese international students, we hope to create a method that helps people who are interested in Chinese to learn this magical language. Stroke, as one of the most crucial parts of Chinese characters, is a good place to get start with. There is an order to write each stroke in a Chinese character and when looking up a character in the dictionary, we need to first know how many strokes its radical has, and then find the place of the character based on the rest (except radical) of stroke number the character has. Most of the online systems for Chinese character can only count the stroke number of type-in Chinese character. What if the learner only knows how the word looks like but does not know how to type it in? Hence, in this project, our goal is to use machine learning algorithms to count the total num-

ber of strokes of a Chinese character depicted as an image. To find out the best method to count strokes, we conduct an experiment based on three main models that are primarily used in today’s machine learning field, including k -Nearest Neighbors algorithm (k NN), Logistic Regression and Convolutional Neural Network (CNN). We divide our experiment into three parts: the first part of the experiment is to fit the training data to make the machine learn how many strokes each Chinese character has; the second part is to use the learned knowledge to predict the number of strokes of a given image of a Chinese character; the third part is to evaluate our models to see how well our models perform on counting the number of strokes of Chinese characters that are not presented in our training data.

2. Related Work

Stroke order and number recognition of Chinese characters with complete relational graphs as input and model characters was presented in 2000 [9]. As there was no suitable algorithms available for graph matching, a transformation from graph-matching to two-layer assignment was made through Hungarian method and was found to be efficient. Similar to their work, our project also aims to find out a good stroke number recognition method for Chinese characters. While their choice of images were not restricted and hence required a graph matching process, we define our image pictures to be in required size and resolution, which makes the work easier.

There have been many previous studies about character recognition in different languages. Starting from basic stroke recognition to whole character recognition, from typing words to handwritten words, different recognition methods were created, experimented and improved. An approach to characterize online handwritten alphanumeric character by a sequence of dominant points in strokes and a sequence of writing directions between consecutive dominant points was innovated by Li and Yeung in 1995 [8]. A recognition experiment was conducted to test the efficiency of the approach and 91% recognition rate was reached. Another handwritten character recognition method about Kanji published in 1996 sought to solve the one-to-one stroke correspondence problem with both the stroke-number and stroke-order variations common in cursive Japanese handwriting by using excessive mapping and deficient mapping [14]. In the recognition test, high recognition rates of 99.5%, 97.6% and 94.1% were obtained for training data and two test data separately. Compared with the alphanumeric character recognition, Japanese Kanji character recognition is more related to our work, as both Japanese and Chinese characters are based on stroke and radical system rather than alphabet one.

3. Proposed Method

Processing image data requires methods that are slightly different from that of regular text labeled datasets. In detail, every single pixel in an image can be treated as a feature of a machine learning model. Hence, we think of each pixel as a dimension added to our model.

In this project, we fit three different kinds of models to count the number of strokes that a Chinese character has. First, we use a simple k NN algorithm as a micro-benchmark. Then, we fit a Logistic Regression model to see how much it can improve compared to k NN. Next, we apply a Max-Pooling kernel to each image before fitting them to a Logistic Regression model. Finally, we train a CNN so that image spatial information can be best absorbed by the model. Accuracy is used as our performance metric throughout the three types of models, and is defined as

$$ACC = \frac{\# \text{ Correct Predictions}}{\# \text{ Total Predictions}} \quad (1)$$

3.1. k -Nearest Neighbor

We select the k NN algorithm as a beginning point; that is, a micro-benchmark to be compared with other methods that are more advanced.

The k NN algorithm categorizes a data point depending on a plurality voting of its k neighbors. To determine the validity of a neighbor, a distance function must be provided. We use the Euclidean distance function here, which is defined as

$$d(p^{[i]}, p^{[j]}) = \sqrt{\sum_{i=1}^N (p_i^{[i]} - p_i^{[j]})^2} \quad (2)$$

where $p^{[i]}$ and $p^{[j]}$ are two data points in an N -dimensional space. Note that each data point p represents a character, and its label is the number of strokes it has. The algorithm selects the first k neighbors of the test data point p and it determines the predicted label of point p by performing a plurality voting among the k neighbors.

We perform 5-fold cross validation to determine the best value of k in the k NN algorithm, so as to avoid overfitting. The training dataset is split into 5 sub-partitions. As a result, the model is trained using 4 of the sub-partitions and validated using the remaining part of the data. A cross validation score is reported as an average of the scores computed during the process. Meanwhile, the best value k of the k NN algorithm is determined.

We apply dimensionality reduction to the data before feeding them to the k NN model because k NN is particularly susceptible to the curse of dimensionality. We carry out Principal Component Analysis (PCA) to cope with this problem. Through PCA, we transform the data into directions of their maximum variance. The number of compo-

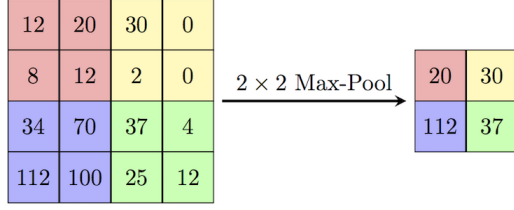


Figure 3. Max-Pooling Example: A 2×2 Max-Pool kernel is applied to a 4×4 input matrix. It is equivalent to selecting the maximum elements among the four 2×2 square sub-matrices.

nents to be obtained is also determined by 5-fold cross validation.

3.2. Logistic Regression

Apart from using a lazy learning algorithm, we choose to fit a multiclass Logistic Regression model, which serves as another micro-benchmark. A multiclass Logistic Regression model uses probabilities to represent how likely a label is to be selected. The probability function is defined as

$$\Pr(y^{[i]} = c) = \frac{\exp(\beta_c \mathbf{x}^{[i]})}{\sum_{k=1}^K \exp(\beta_k \mathbf{x}^{[i]})} \quad (3)$$

where $\mathbf{x}^{[i]}$ is a vector of pixels of image i , $y^{[i]}$ is the number of strokes the character has, K is the number of classes to classify, c is a specific class, and β_c are the weights of the model corresponding to that class.

L1 regularization of weights is applied to the Logistic Regression model to prevent the model from overfitting. We append a regularization term, which is an L1 norm of the weights, to the loss function, with a coefficient λ . Cross Validation methods are used here to determine the best value of λ too.

In addition, bagging is applied to the training process of the model. Bagging is a learning algorithm where the same type of model is fit using different subsets of the dataset. Here, different subsets are chosen by bootstrapping samples. The variance of the model is therefore reduced through bagging.

3.3. Logistic Regression with Max Pooling

We believe that regular Logistic Regression might also suffer from the curse of dimensionality. Besides, Logistic Regression treats every single pixel as a feature. There is no inter-connection between the pixels in an image. Each pixel is treated as an independent predictor rather than as a whole integrated system. Thus, we apply a Max-Pool kernel over an image so that dimensionality is reduced and spacial information can be relatively preserved.

Max-pooling is a process that down-samples an input image. It is done by applying a max filter (i.e., selection of

the maximum value) to non-overlapping sub-regions of an image. Figure 3 shows an example of a process of the Max-Pooling kernel.

3.4. Convolutional Neural Network

Convolutional Neural Networks (CNN) are best known for recognizing simple objects with high shape variability [7]. We train a CNN so that the model is able to recognize the variety of strokes, and thus make more accurate predictions than classical models. Table 1 describes the network architecture that we use.

We use categorical cross entropy as our loss function, and ADADELTA [15] as our optimizer.

Layer	Property	Kernel Size	Activation
Conv2D	filter=64	(3,3)	ReLU
Conv2D	filter=128	(3,3)	ReLU
MaxPooling 2D	N/A	(2,2)	N/A
Dropout	prob=0.3	N/A	N/A
Dense	len=256	N/A	ReLU
Dropout	prob=0.6	N/A	N/A
Dense	len=29	N/A	softmax

Table 1. CNN Network Architecture

4. Experiments

4.1. Dataset

The original dataset we find contains 15 million 28×28 PNG files of 52,835 Chinese characters with different fonts [11]. This dataset is created based on 36 stroke features. Burkimsher gets this dataset by generating Chinese fonts from Chinese Font Design, a website containing a great amount of Chinese font collection. Another dataset we find is from Unicode, which is a txt file containing most of the Chinese characters and their stroke counts [6]. We use it to look up stroke count information for both training and testing sections. The reason we decide to use the Unicode dataset is that it provides encoding of all characters used in the world's written languages. More importantly, the image dataset of Chinese characters we find is also based on Unicode dataset.

For the data cleaning part, we first pick a font called Sim Hei for this project, as it supports a large number of Chinese characters and can be easily recognizable compared with some other Chinese fonts with join-up writing which would make the recognition of strokes much harder. Then we delete characters with extreme counts (stroke count = 1 and ≥ 31). The final dataset we use consists of 27,746 PNG files with a resolution of 28×28 . For each pixel of the picture,

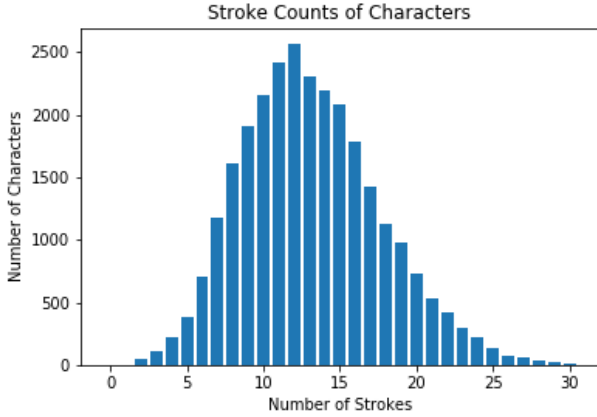


Figure 4. A distribution table of the number of Chinese characters categorized by stroke number.

we use an interval $[0, 1]$ to define the extent of greyness, 0 represents the darkest blackness and 1 represents the lightest whiteness. We also divide the number of strokes into 29 classes, with numbers 2 to 30 for each different character. Figure 4 shows the distribution of character counts categorized by the number of strokes.

Moreover, we inverted every single image of the font Sim Hei, as shown in Figure 5. In other words, we define the new image as

$$\text{new_img} = 1 - \text{img} \quad (4)$$

where new_img and img are matrices containing all the pixels of an image in our dataset. After inversion, images render black in the background and white in the character strokes. Hence, these images are now represented by sparse matrices, given that most pixel values are 0.

After image inversion, we split the image dataset into two separate parts - for training and testing. We apply stratified sampling to our dataset to maintain the original class proportion in resulting subsets, using the function `train_test_split` from scikit-learn [12]. The training set consists of 90% of the images, while the test set contains the rest 10%.

4.2. k -Nearest Neighbor

In general, we use a pipeline that connects two estimators, PCA and `KNeighborsClassifier`, pass them into `GridSearchCV`, and perform 5-fold cross validation to determine the best hyper-parameters, $n_components$ and C (the inverse of λ), as described previously.

Specifically, each 28×28 pixels image is treated as a single data point in a 784-dimensional space. So, we choose a list of parameters, 100, 200, 300, ..., 700, and 784 as $n_components$ used in PCA. Furthermore, we pick a list

ranging from 5 to 555, with step size 50 as a set of parameters for $n_neighbors$ used in k NN. The exhaustive search algorithm with cross validation implemented will pick the best pair of parameters among all the combinations.

Because the step sizes of both the parameter lists are relatively large (100 and 50, respectively), we run the algorithm a second time with narrower range and smaller step sizes. After another pair of parameters is selected, we test the model on the test dataset and thereby obtain the final test score.

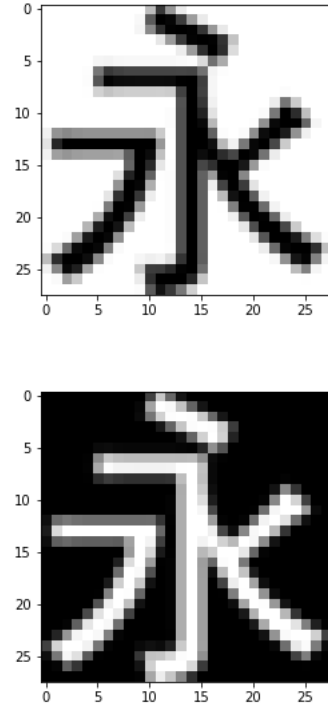


Figure 5. The image of Yǒng before inversion, which is a black character in white background (Top); The image of Yǒng after inversion, which is a white character in black background (Bottom).

4.3. Logistic Regression

To train a Logistic Regression model, we need to specify a few parameters, including a regularization metric, a solver, the maximum number of iterations and a stopping criteria. We apply L1 regularization to our model because our image dataset is sparse. Most of the pixel values in the images are 0 due to the black background color. Next, We select the SAGA [4] solver because it is the only solver in scikit-learn that supports both L1 regularization and multi-class classification. The only hyper-parameter we tune is C , which is the regularization strength. Hence, we use the estimator `LogisticRegressionCV` here, with C values ranging from 10^{-4} to 10^4 in a logarithmic scale. In the

hyper-parameter tuning stage, we set maximum iterations to 500 and stopping tolerance to 0.05 to avoid long tuning time. After an optimal C is chosen, we switch them to 10,000 and 0.001, respectively.

In addition to that, we apply bagging to reduce the variance of the model. We construct 10 Logistic Regression estimators by applying bootstrap sampling method.

4.4. Logistic Regression with Max-Pooling

We construct a Max-Pooling operation estimator on our own, because scikit-learn does not support it currently. Internally, we use the `block_reduce` function provided by scikit-image [13] to perform the Max-Pooling operation. The Max-Pool kernel size is selected by `GridSearchCV` from (2,2), (3,3) and (4,4).

Similar to a classical Logistic Regression, we also use Cross Validation method to pick the best regularization strength. All other parameters are set to the same value as the Logistic Regression model mentioned above.

4.5. Convolutional Neural Network

We build our CNN model using Keras [3], which is a famous Python deep learning library. Please refer to Table 1 for the network architecture in detail. Instead of using a data pipeline, we load our dataset directly into memory. We choose a batch size of 128, and let the optimizer iterate through the dataset for 50 epochs.

5. Results

For the k NN model, we find that it gives us the best result when $k = 15$. The test accuracy we gain is 24.86%. For Logistic Regression, we obtain a test accuracy of 19.47%. By adding Bagging into our Logistic Regression model, we improve the accuracy by 2% and make it to 21.68%. We also try Logistic Regression with Max-Pooling model but discover that the test accuracy drops to 13.58%, which is not what we expected. Finally, we apply CNN model and successfully improve our test accuracy to 47.43%.

Model Name	Condition	Test Accuracy
k NN	$k = 15$	24.86%
Logistic Regression	regular	19.47%
	Bagging	21.68%
	Max-Pooling	13.58%
CNN	regular	47.43%

Table 2. Test Accuracy for All Three Models

5.1. k -Nearest Neighbor

The k NN model achieves a test accuracy of 24.86%. The optimal choice of k is 15, and the optimal `n_components` in PCA is 300, according to the output of `GridSearchCV`. It took nearly 24 hours for the algorithm to find the optimal pair of hyper-parameters from the lists that we provided. Due to time limitation, we are not able to dig further to find the true optimal hyper-parameter. Also, it is not worth going further because of the low accuracy that k NN algorithm provides.

5.2. Logistic Regression

The Logistic Regression model achieves a test accuracy of 19.47%. The optimal choice of C is 10. When bagging is applied, test accuracy rises by 2.21%, resulting in 21.68%. Besides, the optimal choice of C becomes 10^2 . Still, the test accuracy is not ideal either, and it is even lower than that of the k NN model.

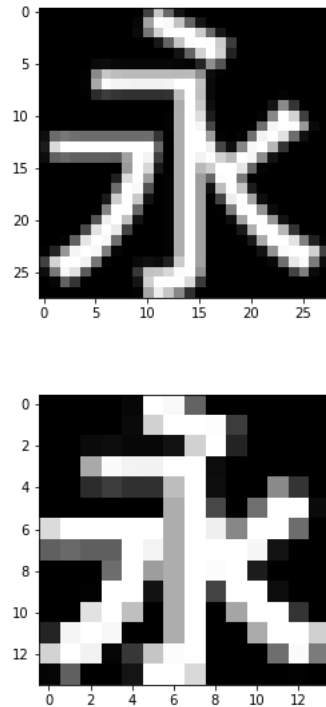


Figure 6. Yǒng (Top) before the Max-Pooling and Yǒng (Bottom) after the Max-Pooling

5.3. Logistic Regression with Max-Pooling

When Max-Pool operations are applied to every image, we observe a 8.1% decrease in test accuracy, comparing to Logistic Regression with Bagging. The optimal Max-Pool kernel size is (2,2), and the best value of C is 1.

Prior to evaluating this model, we believe that spacial information is relatively preserved during the process of Max-Pooling. However, some detailed information about the image is lost in tandem too. For example, in Figure 6, the Chinese character yǒng is composed of 5 strokes. One can easily recognize that there are three unconnected partitions in the image. However, when Max-Pool operation is applied to the image, the previously unconnected parts are glued together; thus, it becomes hard for the model to separate the strokes. Therefore, it is unavoidable that the accuracy of the new model decreases.

5.4. Convolutional Neural Network

Not surprisingly, our CNN model performs the counting task best among the three models, with a test accuracy of 47.73%. As we can see, CNN preserves spatial information relatively well, so it can easily recognize some patterns that appear in the images.

5.5. Discussion

Both classical models, k NN and Logistic Regression, perform poorly in counting the number of strokes. The test accuracy reported by the CNN model is also not very ideal as it is less than 50%. We believe that this results from the labeling method of our dataset. Recall that we label each image the number of strokes it has, so the label is a simple integer ranging from 2 to 30. However, sometimes it might not be easy to tell how many strokes there are in a Chinese character at a first glance. For example, in Figure 7, the character shì consists of 5 strokes. People usually overestimate or underestimate this quantity because the four strokes underneath the dot are all connected together. Take another example, zhuāng in Figure 8, which is composed of 12 strokes. The dot in the middle seems to be connected with the structure at the bottom. Nevertheless, it counts as a separate stroke. Our models might be confused with situations like this, and thereby they produce poor results.

Fortunately, we believe this problem can be solved by using a more detailed set of labels. We are currently using labels that only show the number of strokes a character has. There is no detailed information about what kind of strokes it has, nor the order of the strokes displayed as if it were written on a paper. Models might perform better if they are aware of what kind of strokes are included and how many of them are present. In particular, a CNN model will be able to learn how to recognize each kind of strokes in the character. And finally, the stroke count will be a simple summation of every occurrence of each kind of stroke. We are, unluckily, unable to carry out a stroke recognition model because there is limited open-source data regarding stroke details of Chinese characters.

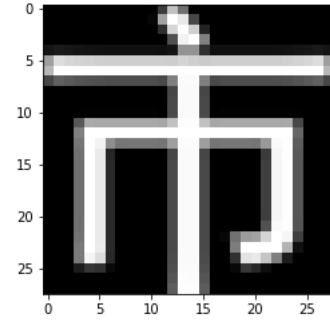


Figure 7. Shì, a character that contains 5 strokes

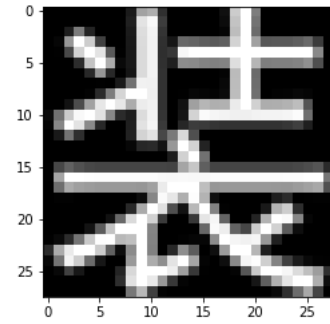


Figure 8. Zhuāng, a character that contains 12 strokes

6. Conclusion

In this project, we demonstrate a variety of methods to count the number of strokes in Chinese characters. We utilize three classes of machine learning algorithms to solve the problem - k NN, Logistic Regression and CNN. Furthermore, other machine learning techniques are applied to achieve better results, such as PCA, bagging, and cross validation. We also show that Max-Pooling operations applied before Logistic Regression do not necessarily help improve model accuracy.

Based on the result we get from all of our models, we believe that the CNN model is the most suitable solution to our research problem. Nonetheless, there is still room of improvement comparing our CNN model to the work people have done previously, though our research goal differs from theirs. In the future, we hope more and more researchers can gather and provide a more detailed database on Chinese characters, especially Chinese strokes. Meanwhile, we hope that a new model that runs on the enhanced database can achieve a higher accuracy than the CNN model we currently train. And finally, we hope our model will be integrated to mobile or web applications in order to help Chinese second language learners establish a solid foundation

on the Chinese stroke system.

7. Contribution

Scott Lai found the original dataset from the web that contains 15 million images. He also coded the k NN model at a very superficial level. Yuqi Lin found the UniHan Database that contains stroke count information of Chinese characters. In addition, she coded the complete version of the k NN model, using PCA and GridSearchCV. She also wrote code for the Logistic Regression model. Yien Xu processed the original dataset and turned it into a single csv file. He also coded the rest of the models, including Logistic Regression with Bagging and Max-Pooling, and the CNN. Both Yien and Yuqi worked together and finished the project report.

References

- [1] Yong zi ba fa.
- [2] *Chinese Character Turning Stroke Standard of GB13000.1 Character Set*, Dec 2001.
- [3] F. Chollet et al. Keras. <https://keras.io>, 2015.
- [4] A. Defazio, F. R. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. *CoRR*, abs/1407.0202, 2014.
- [5] J. J.H., C. R., and L. K. *CJK Strokes*.
- [6] J. J.H., C. R., and L. K. Unicode han database, May 2018. Accessed Dec. 14, 2018.
- [7] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In D. Forsyth, editor, *Feature Grouping*. Springer, 1999.
- [8] X. Li and D.-Y. Yeung. On-line handwritten alphanumeric character recognition using feature sequences. pages 197–204, 1995.
- [9] J. Z. Liu, K. Ma, W. K. Cham, and M. M. Y. Chang. Two-layer assignment method for online chinese character recognition. *IEEE Proceedings - Vision, Image and Signal Processing*, 147(1):47–54, Feb 2000.
- [10] D. Olson. Chinese writing. Mar 2014.
- [11] B. P. Making of a chinese characters dataset, Jun 2018.
- [12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [13] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.
- [14] T. Wakahara, A. Suzuki, N. Nakajima, S. Miyahara, and K. Odaka. Stroke-number and stroke-order free on-line kanji character recognition as one-to-one stroke correspondence problem. pages 529–534, 01 1996.
- [15] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Appendix - CJK Strokes





































	31C	31D	31E
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
A			
B			
C			
D			
E			
F			

Figure 9. 36 Stroke Features [5]