Abstract
Embedding is a foundational technique in modern natural language processing. It is necessary to convert text to vectors before feeding to neural networks for further processing. However, current approaches are predominantly developed on alphabetical languages, where individual characters have no meaning, and the meaning and the appearance of text are unrelated. In contrast, Chinese is logographic. Each character in Chinese is meaningful, and the meaning is correlated to its appearance. Therefore, this project proposes OceanText, a visual embedding system of Chinese, which utilizes the visual features of Chinese characters to improve the accuracy of embedding. The project team produced an embedding library and a preprocessing library for Chinese corpora and conducted extensive experiments to validate the effectiveness of OceanText. The algorithm set a new state-of-the-art for character embedding quality on wordsim-297.
Table of Contents

Abstract ........................................................................................................................................ 1

1. Introduction ................................................................................................................................ 4

2. Related Works ................................................................................................................................ 7
   2.1. Word Embedding .................................................................................................................... 7
   2.2. Chinese Embedding ................................................................................................................ 7
   2.3. Visual Embedding of Chinese .............................................................................................. 8

3. Method ........................................................................................................................................... 10
   3.1. Generic Formulation of Language Embedding .................................................................... 10
   3.2. The OceanText Algorithm .................................................................................................. 11

4. Implementation ............................................................................................................................ 13
   4.1. Implementation Choices ....................................................................................................... 13
   4.2. Embedding library ................................................................................................................. 13
   4.3. Preprocessing Library ......................................................................................................... 15

5. Experimental Analysis .................................................................................................................. 16
   5.1. Experimental Setup .............................................................................................................. 16
   5.2. Initial Validation .................................................................................................................... 16
   5.3. Backbone Network ............................................................................................................... 17
   5.4. Pooling Method .................................................................................................................... 18
   5.5. ImageNet pretraining .......................................................................................................... 18
   5.6. Font of Rendering ................................................................................................................ 19
   5.7. Simplified and Traditional Characters .................................................................................. 19
   5.8. Comparison with the State-of-the-Art .................................................................................. 20

6. Conclusion ..................................................................................................................................... 21
1. Introduction

*Deep learning* has been a central technique for recent advancements in natural language processing (NLP). The mathematical form of deep neural networks requires text to be converted to vector representations before being fed into the networks. This process is language embedding. As a necessary step for any deep learning-based NLP system, language embedding has been a foundational technique in modern NLP. The quality of embeddings has a significant impact on all subsequent systems that use them. However, existing approaches have been predominantly developed on alphabetical languages like English. Chinese, as a language with a logographic writing system, has two fundamental differences compared to alphabetical languages. Firstly, a single Chinese character is associated with meaning, unlike letters in alphabetical languages. For instance, 火 (*huo*) means fire, 人 (*ren*) means person, and 电 (*dian*) means electricity. All three examples are individual characters isolated from any context, but they retain their meanings. In comparison, the English letter *l* , when isolated from any context, is not associated with any meaning. Secondly, the appearance of a Chinese character is related to the meaning it expresses. As Figure 1 illustrates, 海 (*hai*, sea), 湖 (*hu*, lake), and 河 (*he*, river) all share the radical\(^1\) \(\克里斯多夫\) (known as sandianshui, or three-dotted water) and all have meanings related

---

\(^1\) A radical is a technical term for a recognized recurring and semantic constituent of Chinese characters. There are 214 radicals in the Taiwan standard.
to water. These distinct properties indicate a potential to incorporate the appearance information of Chinese characters into language embedders.

The research community has made various attempts to utilize the unique characteristics of the Chinese writing system to build more powerful embedders. Chen et al. (2015) proposed to utilize the semanticity of individual Chinese characters by jointly training a character and a word embedder. Yu et al. (2018) manually specified a set of decomposition rules to examine the internal structures of Chinese characters. Cao et al. (2018) designed \textit{cw2vec}, representing Chinese characters as stroke\textsuperscript{2} n-grams\textsuperscript{3}. Models using the embeddings generated by \textit{cw2vec} currently holds the accuracy records on a variety of NLP tasks. However, the stroke sequence lacks information about the variations in the realization of a stroke. As Figure 2 shows,

![Figure 2](image.png)

\textbf{Figure 2.} Two Chinese characters with the same stroke sequence but different realizations and meanings.

characters \(\pm\) (\textit{shi}, a respected person) and \(\pm\) (\textit{tu}, soil) have the same character sequence but vastly different meanings. They differ only by the relative lengths of the two hengs\textsuperscript{4}, which is not captured by their stroke n-grams. To inform the embedding model of the realization details of the

\textsuperscript{2} A \textit{stroke} is a constituent of Chinese characters that is written with the pen tip continuously touching the paper. There are 37 fine-grained types of common strokes and 5 coarse-grained types.

\textsuperscript{3} An \textit{n-gram} is a subsequence of length \(n\). For example, the 3-grams in the word \textit{hello} are \textit{hel}, \textit{ell}, and \textit{llo}.

\textsuperscript{4} \textit{Heng} is the terminology for a horizontal stroke.
sequences and other information like the relative positions of the strokes, an end-to-end approach with each character encoded as an image input to the model might prove helpful.

Several works have focused on augmenting Chinese embedders with rendered images of characters. The GWE (Su & Lee, 2017) uses an unsupervised convolutional autoencoder to visually encode characters. Then, for each word, the model adds the visual encodings of all constituent characters to an embedding of the word generated by a conventional embedder. However, this approach does not utilize the self-supervising information provided by the context, since the visual encoding is generated by a convolutional auto-encoder, which does not have access to the context. Additionally, the GWE is not end-to-end trainable, since the visual embedder and the conventional embedder must be trained separately. Costa-Jussa et al. (2017) proposed a system that directly flattens the pixels of the image of a character and enhances a conventional embedding by concatenating it with the pixel vector. This method ignores contextual self-supervision and is not end-to-end. Further, the flattening operation destroys most of the structural information of the character image. The ID+CNN embedder (Dai & Cai, 2017) is the method closest to ours. Different from the previous two approaches, it is a character embedder and utilizes contextual self-supervision. However, they only tested a two-layer and a one-layer CNN, whose representational power is significantly inferior compared to any other embedding architectures. Further, the study did not conduct any experiments on widely-used large-scale datasets.

This report proposes a novel visual embedder of Chinese. Unlike existing methods, it is the first to satisfy all the following:

1. It incorporates appearance information via rendered character images into the embedding process;
2. It preserves the structural information of characters;
3. It is end-to-end trainable;
4. It utilizes contextual self-supervision;
5. It uses latest neural network architectures in computer vision;
6. It will be trained and evaluated on widely-used large-scale datasets.
2. Related Works

This section introduces various relevant works. Section 2.1 presents previous works on word embedding. Section 2.2 discusses works specifically on the embedding of Chinese. Section 2.3 presents works on Chinese embedding that uses visual information, which most closely relate to this work.

2.1. Word Embedding

Word embedding is the process of converting each word in a piece of text to a vector representation. *word2vec* (Mikolov et al., 2013) was the first successful embedding algorithm that enabled the currently predominant deep learning-based NLP techniques. It represents each word by a one-hot vector and uses a 2-layer fully-connected feedforward neural network to predict the appearance context words around it. They defined a context word as a word within a fixed-sized window around the target word. However, since for every word in the corpus, every possible word not present in the context window qualifies as a negative example, the number of negative examples is prohibitively large. The authors proposed a negative example sampling technique called noisy contrastive estimation to mitigate this.

*fastText* (Bojanowski et al. 2016) improved upon *word2vec* by incorporating sub-word information into the embedding process. The authors observed that words in alphabetical languages like English consists of recurring and semantic character sequences, known as word roots or morphs. For illustration, the English word *representation* consists of morphs *re-*-, *pre-*-, *-s-*-, *-ent*, and *-ation*. They utilize the recurrence and semanticity of morphs by extracting all character n-grams with n in a specified range from each word and represent a word by an n-hot vector indicating both its ID and its character n-grams before feeding into the network.

2.2. Chinese Embedding

Chen et al. (2015) observed that individual Chinese characters are semantic, unlike the letters in alphabetical languages. Hence, they proposed to utilize the semanticity of individual Chinese characters by jointly training a character and a word embedder. Similarly to *fastText*’s (Bojanowski et al. 2016) utilization of letter n-grams, CWE represents each Chinese word by an n-hot vector indicating both the IDs of its constituent characters and its own ID and feeds this into the embedding network.
However, there are over 50,000 distinct Chinese characters, far exceeding the total number of English or Latin letters. This results in the extreme rarity of the least common characters. The contextual supervision provided by even a large corpus might be insufficient for training effective representations for those characters. To remedy this, the JWE (Yu et al., 2017) manually specified a set of rules to decompose each Chinese character to a set of common sub-character constituents. The reduced sparsity ensures enough supervision signals for all sub-character constituents.

Despite the reduced sparsity, the decomposition rules in the JWE (Yu et al., 2017) is manually designed and might not be optimal. Cao et al. (2018) used a learning approach to improve the decomposition of Chinese characters. Their method classifies each stroke into one of the five coarse-grained categories and represents each Chinese character as a set of stroke n-grams, for n ranging from 3 to 12. The authors empirically tested their approach and achieved record accuracies on word similarity, word analogy, text classification, and named entity recognition in Chinese. However, the stroke sequence lacks information about the variations in the realization of a stroke.

2.3. Visual Embedding of Chinese

The GWE (Su & Lee, 2017) attempted to augment Chinese embedders with visual features. The authors trained a 10-layer convolutional autoencoder using rendered images of characters that produces vectors of a fixed dimensionality. When embedding a word, the model obtains a vector of the same dimensionality from a conventional word embedder. Then, the model extracts a visual feature vector from the image of each constituent character. Finally, the model computes the output embedding as the sum of the conventional embedding and all visual features.

The bitmap-enhanced embedder (Costa-Jussà et al., 2017) uses the flattened vector of the pixels of the image of a character as its visual feature. The method then concatenates the visual feature with a conventional embedding and then returns the concatenated vector. The authors validated their approach on Spanish-Chinese translation. The model outperformed previous neural translators but was inferior to the most accurate phrase-based models.

Unlike the GWE (Su & Lee, 2017) and the bitmap-enhanced embedder (Costa-Jussà et al., 2017), The ID+CNN embedder (Dai & Cai, 2017) is a character embedder. The authors replaced the 2-layer feedforward network in a conventional embedder by a visual architecture. The
architecture consists of 1 or 2 convolutional layers followed by 2 fully connected layers. It takes in the image of a character and outputs an embedding of it. By only substituting the neural network in the embedding pipeline, the model can utilize contextual self-supervision like conventional embedders. The authors trained the networks on the Chinese Gigaword dataset. Then, they conducted evaluation on the same dataset for language modeling and on the MSR and the PKU datasets for word segmentation. Since the Chinese Gigaword dataset is proprietary and close-sourced, it is not possible to reproduce their work.
3. Method
This section formally describes the proposed OceanText algorithm. Section 3.1 provides a generic formulation of language embedding that captures existing approaches. Then, Section 3.2 presents the formal definition of the OceanText algorithm.

3.1. Generic Formulation of Language Embedding
This section presents a generic formulation of language embedding. Language embedding is a mapping from the space of a type of linguistic units (e.g. characters, words, sentences, etc.) to a vector space. It generally consists of 3 steps, namely encoding, embedding, and supervision.

![Figure 3. Flow chart demonstrating the three steps of embedding for word2vec.](image)

Figure 3 illustrates the 3 steps for word2vec. The encoding process of word2vec transforms a word into a one-hot vector indicating the ID of the word via a dictionary look-up. Then, the embedding process projects that one-hot vector to an embedding vector by a linear layer. Finally, the supervision module predicts context words using a softmax classifier and computes a cross entropy loss as the supervision signal. Then, the module computes the gradients for parameters in the model with respect to the loss using the backpropagation algorithm (Linnainmaa, 1970) and performs a parameter update using an optimizer (e.g. SGD (Robins & Monroe, 1985) or Adam (Kingma & Ba, 2015).
3.2. The OceanText Algorithm

The proposed method is shown in Figure 4. The encoding process renders a character into an image. The image is then used as input to the embedding process. The embedding module consists of a convolutional neural network and a linear layer. The linear layer has the same capacity as the one in a conventional embedder and ensures that the embedding module possesses at least the representational power of a conventional linear embedding module. Lastly, the supervision module is not modified compared to word2vec for a fair comparison. Table 1 presents a detailed comparison of OceanText and existing methods.

![Figure 4](image_url) Flow chart demonstrating the three steps of embedding for OceanText.

<table>
<thead>
<tr>
<th>Method</th>
<th>word2vec</th>
<th>CWE</th>
<th>JWE</th>
<th>GWE</th>
<th>cw2vec</th>
<th>Bitmap-enhanced embedder</th>
<th>ID+CNN embedder</th>
<th>OceanText</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses visual information</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Uses character semanticity</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Uses character structure information</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Uses image inputs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>End-to-end trainable</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Uses contextual self-supervision</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Uses latest network architectures in computer vision</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Validated on widely-used large-scale datasets</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes*</td>
</tr>
</tbody>
</table>

Table 1. Comparison of notable embedders for Chinese.

*OceanText* will be validated on widely-used large-scale datasets. However, the experiments have not be conducted yet.
4. Implementation

This section describes the implementation of the project. Subsection 4.1 introduces the choices and tradeoffs for the implementation. Subsection 4.2 describes the embedding library the team produced. Subsection 4.3 presents the preprocessing library for Chinese corpora the team implemented.

4.1. Implementation Choices

The team implemented the OceanText algorithm in PyTorch. PyTorch is an open-source library for heterogeneous scientific computing and deep learning. It provides a diversity of popular operations supporting deep learning algorithms, including convolutions, linear layers, long short-term memory cells, and softmax operations, which are important constituents of OceanText. Additionally, it provides automatic differentiation, which alleviates the need to implement the tedious and error-prone differentiation and backpropagation mechanisms. Compared to competing deep learning libraries like TensorFlow, MXNet, Caffe 2, and CNTK, PyTorch has two additional advantages. First, it features a simple API and intuitive semantics with tight integration with the Python programming language. This creates a smooth learning curve assuming prior knowledge of Python. Furthermore, it uses dynamic computation graphs, i.e. it dynamically generates the graph depicting the computational relations between different variables when computing them. This allows flexible debugging and expedites research explorations.

4.2. Embedding library

The project team decided to build its own code base upon existing an open-source word embedding library in PyTorch. This approach would minimize the efforts spent to produce a working baseline that OceanText can build upon and compare with. Among openly available libraries that satisfy the needs, pytorch_word2vec\(^5\) (bamtercelboo, 2017) has the largest number of users and the most active community. The active community endorses its correctness and effectiveness. Therefore, the team chose it as the basis of the OceanText code base.

---

\(^5\) https://github.com/bamtercelboo/pytorch_word2vec.
However, `pytorch_word2vec` has two major deficiencies. First, its speed is a major concern. Using the Wikipedia Chinese corpus, the standard dataset for Chinese NLP, it would take 82 days to train a single model. This speed is significantly slower than the theoretical calculation and unaffordable for research purposes. Additionally, the code quality of the library was poor. It featured copy-and-pasted code, commented-out lines, oversized classes and functions, and other symptoms indicating unsatisfactory code quality.

To improve `pytorch_word2vec` and make it practical for research purposes, the team decided to refactor it to improve its code quality, since high-quality code is more analyzable, modifiable, testable, and maintainable. After the refactoring, the resultant code base offered the same functionality and performance with less than 300 lines of code, down from the over 1500 lines of the original code base. Furthermore, it eliminated all identified symptoms of poor code quality, including the ones mentioned in the previous paragraph.

With a high-quality code base, an investigation quickly pinpointed a defect that caused the significant slowdown of the library. After fixing the defect, the refactored code base runs over 70 times faster than the original. It now takes slightly more than a day to train a model, which makes the training of word embedder on widely-used large-scale corpora like Wikipedia Chinese viable. The resultant library is named `OceanEmbedding`. Table 2 summarizes the major differences between `OceanEmbedding` and `pytorch_word2vec`.

<table>
<thead>
<tr>
<th>Metric</th>
<th>OceanEmbedding</th>
<th>pytorch_word2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training time (single model on Wikipedia Chinese)</td>
<td>Lower the better</td>
<td>28.1 hours</td>
</tr>
<tr>
<td>Logical lines of code</td>
<td>Lower the better</td>
<td>287</td>
</tr>
<tr>
<td>Average cyclomatic complexity</td>
<td>Lower the better</td>
<td>2.54</td>
</tr>
</tbody>
</table>

Table 2. Comparison between `OceanEmbedding` and `pytorch_word2vec`. Cyclomatic complexity is a widely-used metric to measure code complexity.
4.3. Preprocessing Library

Working with Chinese text requires a preprocessor to format the text appropriately for subsequent processing. Many open-source libraries are available for this task. OpenCC is the most widely used. It was adopted by most of the previous works on Chinese embedding (Cao et al. 2018; Chen et al., 2015; Su & Lee, 2017; Yu et al. 2015) that this work improves upon. The project team follows them and uses OpenCC as the main preprocessor.

Most of the works on Chinese embedding follow the preprocessing procedure by Chen et al. (2015). Therefore, this work identified an opportunity to automate the procedure with a program that encapsulates OpenCC and other relevant libraries. The implemented program has an intuitive interface, taking in a Wikipedia dump file and producing a plain text file where each line stores the Chinese portion of a Wikipedia item.
5. Experimental Analysis
This section presents the experimental design, outcomes, and analysis.

5.1. Experimental Setup
The experiments trained the models on the Wikipedia Chinese dataset. The batch size was 50 for all models, following the word2vec_pytorch library (bamtercelboo, 2017). Each model trained until convergence or maximally for 24 hours on a single NVIDIA GeForce GTX TITAN X.

The evaluation of the models used the wordsim-297 dataset. A model under test first generates an embedding vector for each character in a word, then aggregates the character embeddings via a pooling method to produce a word embedding. A standard evaluator takes in the word embeddings and computes the similarity scores between each pair of words in the wordsim-297 dataset by dot product. Finally, the evaluator computes the Spearman’s rho, a measure of correlation, between the predicted similarities and annotated similarities the dataset provides, as a metric reflecting the effectiveness of the embeddings.

5.2. Initial Validation
The experiment for initial validation trained a model with a NanoNet backbone for character embedding. NanoNet is a convolutional neural network the team designed for this task. It aims at
minimizing the computational cost to shorten the time to validate the approach. It consists of 3 convolutional layers with 3x3 kernels, a global average pooling layer, and a fully-connected layer. The convolutional layers use Kaiming uniform initialization (He et al., 2015), and the fully-connected layer uses Xavier uniform initialization (Glorot and Bengio, 2010). Figure 5 illustrates the architecture of NanoNet.

![Figure 5. Architectural illustration of NanoNet.](image)

The experiment then combined the character embeddings with global average pooling to form word embeddings. Finally, it tested the word embeddings on the Chinese wordsim-297 dataset. Table 3 presents the result.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Pooling</th>
<th>Spearman’s rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>NanoNet</td>
<td>Average</td>
<td>8.31</td>
</tr>
</tbody>
</table>

*Table 3. Result of the initial validation of the proposed method.*

### 5.3. Backbone Network

This section presents the comparative experiments for different backbone networks. The experiments tested 4 different convolutional backbone networks, NanoNet, ResNet-18 (He et al. 2016), GlyphNet (Dai and Cai, 2017), and TianzegeCNN (Wu et al., 2019). Table 4 presents the results.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Pooling</th>
<th>Spearman’s rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>NanoNet</td>
<td>Average</td>
<td>8.31</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Average</td>
<td>37.6</td>
</tr>
<tr>
<td>GlyphNet</td>
<td>Average</td>
<td>8.08</td>
</tr>
<tr>
<td>TianzegeCNN</td>
<td>Average</td>
<td>9.70</td>
</tr>
</tbody>
</table>

*Table 4. Experiments for different backbone networks.*

As in Table 4, the three shallow networks have relatively similar and poor performance. The deep architecture, ResNet-18, performs significantly better. This confirms the hypothesis in Section 1 that modern, deep architectures in computer vision might provide a significant performance boost to visual embedding systems.
5.4. Pooling Method

This section presents experiments contrasting different pooling methods to combine character embeddings to word embeddings. The experiments tried 4 methods, average pooling, minimum pooling, maximum pooling, and minimax pooling. Average pooling, minimum pooling, and maximum pooling return the average, minimum, and maximum of the values in each channel for all characters constituting a word. Minimax pooling generates the minimum-pooling and maximum-pooling results and concatenates them along the channels dimension. Table 5 reports the results.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Pooling</th>
<th>Spearman’s rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>Average</td>
<td>37.6</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Maximum</td>
<td>26.0</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Minimum</td>
<td>18.8</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Minimax</td>
<td>25.0</td>
</tr>
</tbody>
</table>

Table 5. Experiments for different pooling methods.

As Table 5 shows, the performance of average pooling is the best. This is likely due to that average pooling appropriately aggregates the semantics of all characters in a word. The other three methods are all extremum pooling methods. Suppose in a channel of the embeddings, the response is high for one character and low for another. Both are strong semantic signals. However, extremum pooling would shadow all other values with one extremum, which might not be representative of the semantics of the whole word.

Therefore, the three types of extremum pooling all have inferior performance compared to average pooling. Among the three extremum pooling methods, maximum pooling performs the best. This shows that the presence of certain semantic features within a word might be more predictive than the lack thereof. In congruence with the expectation, the performance of minimax pooling is between maximum pooling and minimum pooling.

5.5. ImageNet pretraining

ImageNet pretraining has become the standard method to initialize a neural network for a task without an excessive amount of data. This section presents experiments exploring the effect of ImageNet pretraining on visual embedding of Chinese. The experiments tried ImageNet pretraining on ResNet-18. Table 6 shows the experimental outcomes.
### Table 6. Experiments for the effect of ImageNet pretraining. All experiments in the table used average pooling.

As Table 6 shows, pretrained ResNet-18 performs worse than the randomly initialized one. This might be due to the large domain gap between natural images, which ImageNet contains, and character renderings. Therefore, ImageNet pretraining is detrimental for visual embedding.

### 5.6. Font of Rendering

This section explores whether the font that the system uses to render the characters has any impact on the performance. The experiments evaluated several popular fonts, including Noto, Fangsong, Kaiti, Song, Xingshu, and Caoshu. Table 7 presents the outcomes.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Font</th>
<th>Spearman’s rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>Noto Sans</td>
<td>37.6</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Fangsong</td>
<td>20.5</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Kaiti</td>
<td>17.2</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Song</td>
<td>20.4</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Xingshu</td>
<td>26.1</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Caoshu</td>
<td>18.9</td>
</tr>
</tbody>
</table>

As Table 7 shows, Noto Sans lead to the best performance. A plausible explanation is that various substructures of Chinese characters are more regular in Noto Sans than in other font families. The font families, by decreasing structural regularity, are roughly Noto Sans, Song and Fangsong, Kaiti, Xingshu, and Caoshu.

### 5.7. Simplified and Traditional Characters

This section analyzes the difference between simplified and traditional characters as input to the system. Table 8 presents the outcomes.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Chinese character type</th>
<th>Spearman’s rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>Simplified</td>
<td>37.6</td>
</tr>
</tbody>
</table>
As Table 8 shows, simplified Chinese is more effective than traditional ones. A hypothesis is that, although traditional characters have richer semantic features than simplified ones, their complicated structures are more difficult to distinguish in low-resolution images. Due to resource limitations, the project did not explore inputs larger than 28x28. The low resolution of the inputs might explain the lower effectiveness of traditional characters.

5.8. Comparison with the State-of-the-Art

This section contrasts OceanText with existing state-of-the-art methods. Table 9 presents the performance metrics of the methods.

<table>
<thead>
<tr>
<th>Model</th>
<th>Visual</th>
<th>Spearman’s rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skip-gram</td>
<td>No</td>
<td>15.6</td>
</tr>
<tr>
<td>GlyphNet</td>
<td>Yes</td>
<td>8.08</td>
</tr>
<tr>
<td>TianzigeCNN</td>
<td>Yes</td>
<td>9.70</td>
</tr>
<tr>
<td>OceanText</td>
<td>Yes</td>
<td>37.6</td>
</tr>
</tbody>
</table>

Table 9. Comparison of OceanText and existing state-of-the-art methods.

As Table 9 shows, OceanText outperforms all other state-of-the-art methods, including visual and non-visual models. It is the only model that surpasses skip-gram, the best non-visual model. This demonstrates the effectiveness of OceanText.
6. Conclusion

This report proposes a novel visual embedding method for Chinese. It improves upon existing embedding systems for Chinese by incorporating visual information more effectively. It captures the decompositions of Chinese characters and their spatial relations and realizations from raw pixels of image renderings of the characters. The project produced an embedding library and a Chinese preprocessing library to facilitate research explorations. The team conducted extensive experiments on the Wikipedia Chinese and wordsim-297 datasets to explore and justify the design choices of the model. The project has three major findings. First, applying convolutional neural networks on raw image renderings of Chinese characters is an effective method to utilize the visual information in Chinese characters. Second, modern, deep architectures in computer vision provide a significant boost in performance in comparison to the shallow architectures popular in visual embedding currently. Third, the proposed OceanText algorithm is the current best algorithm for character embedding on the wordsim-297 benchmark. The team hopes visual embeddings, state-of-the-art computer vision networks, and OceanText help boost progress for various Chinese language processing research.
REFERENCES:


