FontRecognition: Identify Your Font from An Image

Tan Xiao, Yang Chengdong, Cai Lingyun
Department of Electrical Information
Wuhan University
{2017301200387, 2017301200155, 2017301200105}@whu.edu.cn

***Abstract*—As font is one of the core design concepts, automatic English font identification and similar font suggestion from an image or photo has been on the wish list of many designers. We present a system for the English font recognition for natural scene images. Real text images of large corpus AdobeVFR are used for training, and the model trained by Convolutional Neural Network (CNN) is used for recognition. A front-end is built to realize the interaction with users, forming a simple and easy-to-use system. The system can operate under a range of scene conditions, and achieves an accuracy of higher than 95**

***Index Terms*—font recognition; AdobeVFR; CNN; SCAE**

I. Introduction

Recognizing English fonts in pictures by taking photos in natural scenes is a challenging problem with many practical applications. For graphic designers, we-media workers and other people who have high requirements for the use of fonts, they need to timely record the desired fonts they see. To help them solve the problem, we present an easy-to-use system that consists of three main stages: (a)Preprocessing, (b)Training, (c)Identification.

Additionally, AdobeVFR is a large-scale visual font recognition database consisting of a composite training set and a real test set. The compositing training set consists of 2383 fonts, each with 1000 compositing images. The real test set included 4383 tagged images covering 671 of 2383 fonts[2]. The download address is http://www.atlaswang.com/deepfont.html. Fig.1 and Fig.2 respectively show some composite samples and real samples in the AdobeVFR data set.

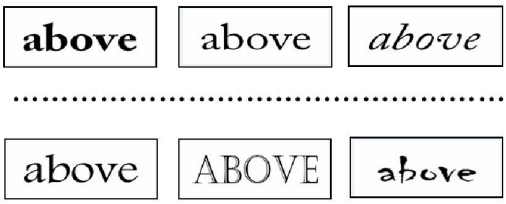


Fig. 1. Synthetic samples in AdobeVFR dataset

Fig. 2. Real sample in AdobeVFR dataset

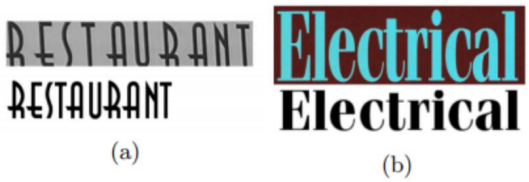
In Section 2, we describe the algorithms used for training, identification, and interactions on the WeChat applet. In Section 3, we describe the implementation process. In Section 4, we present the results. We conclude the paper in the fifth Section.

II. Algorithm

A. Preprocessing

Before feeding synthetic data into model training, it is popular to artificially augment training data using label-preserving transformations to reduce overfitting. In [7], the authors applied image translations and horizontal reflections to the training images, as well as altering the intensities of their RGB channels. The authors in [8] added moderate distortions and corruptions to the synthetic text images:

1) Noise: a small Gaussian noise with zero mean and standard deviation 3 is added to input.
2) Blur: a random Gaussian blur with standard deviation from 2.5 to 3.5 is added to input.
3) Perspective Rotation: a randomly-parameterized affine transformation is added to input.
4) Shading: the input background is filled with a gradient in illumination.

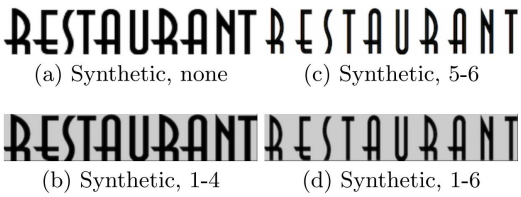


(a) (b)

Fig. 3. (a) the different characters spacings between a pair of synthetic and real-world images. (b) the different aspect ratio between a pair of synthetic and real-world image.

According to the observations in figure 3, in [1], the author added two additional font-specific enhancement steps to the training data:

5) Variable Character Spacing: when rendering each synthetic image, we set the character spacing (by pixel) to be a Gaussian random variable of mean 10 and standard deviation 40, bounded by [0, 50].
6) Variable Aspect Ratio: Before cropping each image into a input patch, the image, with heigh fixed, is squeezed in width by a random ratio, drawn from a uniform distribution between 56 and 76 .



(a) Synthetic, none (c) Synthetic, 5-6

(b) Synthetic, 1-4 (d) Synthetic, 1-6

Fig. 4. The effects of data augmentation steps. (a)-(d): synthetic images of the same text but with different data augmentation ways.

To leave a visual impression, the real-world image Fig. 3 (a), and synthesize a series of images in Fig. 4, are all taken with the same text but with different data augmentation ways. Specially, (a) is synthesized with no data augmentation; (b) is (a) with standard augmentation 1-4 added; (c)is synthesized with spacing and aspect ratio customized to be identical to those of Fig. 3 (a); (d) adds standard augmentation 1-4 to (c). We input images (a)-(d) through the trained DeepFont model. For each image, its layer-wise activations are campared with those of the real image Fig. 3 (a) feeding through the same model, by calculating the normalized MSEs. A few synthetic patches after full data augmentation 1-6 are displayed in Fig. 5. It is observable that they possess a much more visually similar appearance to real-world data.

Fig. 5. Examples of synthetic training 105 × 105 patches after pre-processing steps 1-6.

B. Training

In the section, multi-layer CNN decomposition and SCAE-based domain adaptation are used to extract low-level features of test data. A Convolutional Neural Network (CNN) architecture is employed, which is further decomposed into two sub-networks[1]: a "shared" low-level sub-network which is learned from the training data, and a high-level sub-network that learns a deep classifier from the low-level features.

The basic CNN architecture is similar to the popular ImageNet structure [3], as in Fig.6. The numbers along with the network pipeline specify the dimensions of outputs of corresponding layers. The input is a 105 × 105 patch sampled from a normalized image. Since a square window may not capture sufficient discriminative local structures, and is unlikely to catch high-level combinational features when two or more graphemes or letters are joined as a single glyph (e.g., ligatures), we introduce a squeezing operation, that scales the width of the height-normalized image to be of a constant ratio relative to the height [1]. Note that the squeezing operation is equivalent to producing "long" rectangular input patches.

The N CNN layers are decomposed into two sub-networks to be learned sequentially:(1)Unsupervised cross-domain sub-network C_u, which consists of the first K layers of CNN. C_u will be trained in a unsupervised way, using unlabeled data from both domains. (2)Supervised domain-specific sub-network C_s, which consists of the remaining $N-K$ layers. It accounts for learning higher-level discriminative features for classi-fication, based on the shared features from C_u. C_s will be trained in a supervised way, using labeled data from the synthetic domain only.

An example of the proposed CNN decomposition is shown in Fig.6. The C_u and C_s parts are marked by red and green colors, respectively, with $N = 8$ and $K = 2$. Note that the low-level shared features are implied to be independent of class labels. Therefore in order to address the open-ended problem of font classes, one may keep re-using the C_u sub-network, and only re-train the C_s part.

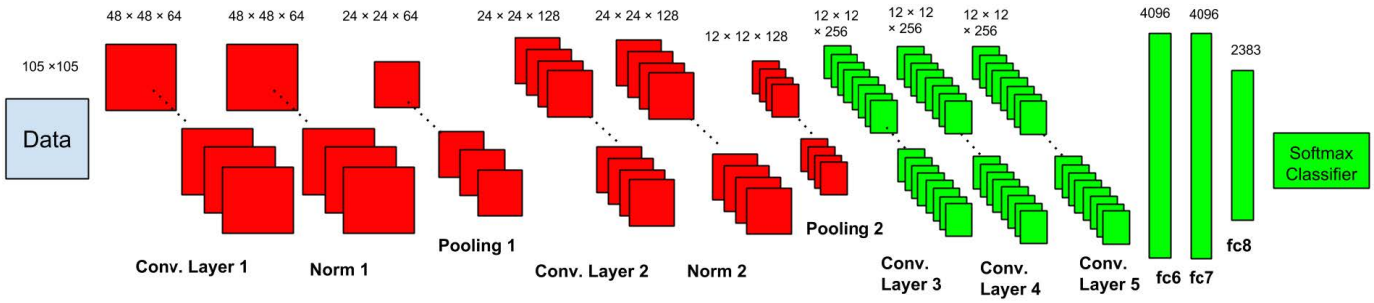


Fig. 6. The CNN architecture in the DeepFont system, and its decomposition marked by different colors(N=8, K=2).

Learning C_u from SCAE Representative unsupervised feature learning methods, such as the Auto-Encoder and the Denoising Auto-Encoder, perform a greedy layer-wise pre-training of weights using unlabeled data alone followed by supervised fine-tuning ([3]). However, they rely mostly on fully-connected models and ignore the $2D$ image structure. In [4], a Convolutional Auto-Encoder (CAE) was proposed to learn non-trivial features using a hierarchical unsupervised feature extractor that scales well to high-dimensional inputs. The CAE architecture is intuitively similar to the conventional auto-encoders in [5], except for that their weights are shared among all locations in the input, preserving spatial locality. CAEs can be stacked to form a deep hierarchy called the Stacked Convolutional Auto-Encoder (SCAE), where each layer receives its input from a latent representation of the layer below. Fig.7 plots the SCAE architecture for our $K = 2$ case.

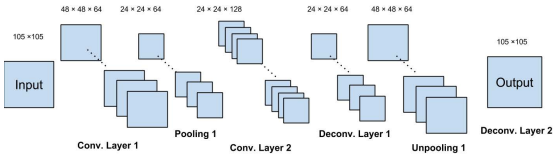


Fig. 7. The Stacked Convolutional Auto-Encoder (SCAE) architecture.

We first train the SCAE on both synthetic and real-world data in a unsupervised way[1], with a learning rate of 0.01 (we do not anneal it through training). Mean Squared Error (MSE) is used as the loss function. After SCAE is learned, its Conv. Layers 1 and 2 are imported to the CNN in Fig.6, as the C_u sub-network and fixed. The C_s sub-network, based on the output by C_u, is then trained in a supervised manner. We start with the learning rate at 0.014, and follow a common heuristic to manually divide the learning rate by 10 when the validation error rate stops decreasing with the current rate. The "dropout" technique is applied to fc6 and fc7 layers during training. Both C_u and C_s are trained with a default batch size of 128, momentum of 0.9 and weight decay of 0.0005. The network training is implemented using the CUDA ConvNet package [3], and runs on a workstation with 12 Intel Xeon 2.67GHz CPUs and 1 GTX680 GPU. It takes around 1 day to complete the entire training pipeline.

C. Identification

Multi-scale and multi-view testing is adopted to improve the result robustness[1]. For each test image, it is first normalized to 105 pixels in height, but squeezed in width by three different random ratios, all drawn from a uniform distribution between 1.5 and 3.5, matching the effects of squeezing and variable aspect ratio operations during training. Under each squeezed scale, five 105×105 patches are sampled at different random locations. That constitutes in total fifteen test patches, each of which comes with different aspect ratios and views, from one test image. As every single patch could produce a softmax vector through the trained CNN, we average all fifteen softmax vectors to determine the final classification result of the test image.

III. Experiments

Sufficient layers of lower-level feature extractors as well as enough subsequent layers are required for good classification of labeled data. In order to maximize the overall classification performance on real-world data, the depth K of C_u should neither be too small nor too large.

From the research conclusions of Wang[1] et al, we know the classification training error increases with K, but the testing error does not vary monotonically. The best performance is obtained with $K = 2$. The optimal $K = 2$ corresponds to a proper "content-aware" smoothening, filtering out "noisy"

details while keeping recognizable structural properties of the font style. The scheme flow is shown in Fig.8.

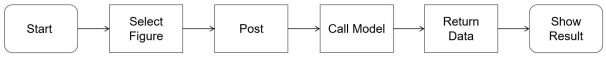


Fig. 8. The scheme flow

Based on the DeepFont system, the measures of font similarity are able to be established. The 4096×1 outputs of the fc7 layer is used as the high-level feature vectors describing font visual appearances. Such features are extracted from all samples in VFR syn val Dataset, and 100 feature vectors per class are obtained. Next for each class, the 100 feature vectors is averaged to a representative vector. Finally, the Euclidean distance between the representative vectors of two font classes is calculated as their similarity measure. Visualized examples are demonstrated in Fig.9. Note that although the result fonts can belong to different font families from the query, they share identifiable visual similarities by human perception.

Fig. 9. Visualized examples

On the basis of this network structure, the image font recognition accuracy increases with the increase of training times, as shown in the Table 1.

TABLE I
RECOGNITION ACCURACY

Epochs	Train	Test
25	52.7%	37.4%
50	64.5%	51.9%
75	78.6%	59.3%
100	85.2%	64.8%
125	91.3%	75.6%
150	95.6%	83.4%
175	99.8%	92.1%
200	99.9%	95.5%

IV. RESULT

To achieve interaction with users, we call a JavaScript in the WeChat developer tool to identify and display the results.

Our system provides a way to upload images from a local folder. It can also be used for natural scenes and scanning text images. The GUI interface is shown in Fig.10, and the font recognition results are displayed in the middle of the screen.

Fig. 10. GUI interface

V. CONCLUSION

We provide a simple and easy-to-use system to solve the problem of English font recognition in natural scenes. This current system has proved effective in simple scene. However, the test scenarios are still limited. It is also to be noted that there are some fonts that the system cannot accurately identify due to the size limitations of our dataset. This system not only is effective for font recognition, but can also produce a font similarity measure for font selection and suggestion.

REFERENCES

[1] Wang Z., Yang J., Jin H., et al. DeepFont: Identify Your Font from An Image [C]. Proceedings of ACM Conference on Multimedia Conference, 2015: 451-459.

[2] Shuye Z., Depth model and its application in visual text analysis[D].South China University of Technology,2016.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, pages 1097–1105, 2012.

[4] J. Masci, U. Meier, D. Cire¸san, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In ICANN, pages 52–59. 2011.

[5] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In ICML, pages 1096–1103. ACM, 2008.

[6] G. Chen, J. Yang, H. Jin, J. Brandt, E. Shechtman, A. Agarwala, and T. X. Han. Large-scale visual font recognition. In CVPR, pages 3598–3605. IEEE, 2014.

[7] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy ayer-wise training of deep networks. NIPS, 19:153, 2007.

[8] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In NIPS, pages 1269–1277, 2014.