DECOMPOSITION-BASED DOMAIN ADAPTATION FOR REAL-WORLD FONT RECOGNITION

Zhangyang Wang & Thomas S. Huang
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{zwang119, t-huang1}@illinois.edu

Jianchao Yang & Hailin Jin & Eli Shechtman & Aseem Agarwala & Jonathan Brandt
Adobe Research
San Jose, CA, 95110
{jiayang, hljin, elishe, asagarwa, jbrandt}@adobe.com

ABSTRACT

We present a domain adaption framework to address a domain mismatch between synthetic training and real-world testing data. We demonstrate our method on a challenging fine-grain classification problem: recognizing a font style from an image of text. In this task, it is very easy to generate lots of rendered font examples but very hard to obtain real-world labeled images. This real-to-synthetic domain gap caused poor generalization to new real data in previous font recognition methods (Chen et al. (2014)). In this paper, we introduce a Convolutional Neural Network decomposition approach, leveraging a large training corpus of synthetic data to obtain effective features for classification. This is done using an adaptation technique based on a Stacked Convolutional Auto-Encoder that exploits a large collection of unlabeled real-world text images combined with synthetic data preprocessed in a specific way. The proposed DeepFont method achieves an accuracy of higher than 80% (top-5) on a new large labeled real-world dataset we collected.

1 INTRODUCTION

While machine learning has in recent years achieved significant success in classification, regression, and clustering, most of these successes depend critically on the assumption that the training and test data are drawn from the same feature space with the same distribution. When the distributions of training and testing data differ, the result is domain mismatch (Pan & Yang (2010)), where the learned models tend to perform poorly because they do not generalize well to the testing domain (Ben-David et al. (2010)).

Domain mismatch particularly arises when labeled data for the target domain is scarce or expensive to obtain, and other, more readily obtainable, data is used for training. Domain adaption (Pan & Yang (2010)) is the problem of reducing the impact of domain mismatch. That is, given a source training domain having sufficient labeled data for training, and a target testing domain, having insufficient labeled data and a distribution that differs from the source domain, the problem is to effectively utilize the model trained on the source domain to achieve good performance on the target domain. In this paper, we study a specific domain adaption problem which arises in the context of font recognition, i.e. identifying a particular typeface given an image of a text fragment. To apply machine learning to this problem, we require realistic text images with ground truth font labels. However, such data is scarce and expensive to obtain, since it requires a high level of domain expertise which is out of reach of most people. Moreover, the training data requirement is vast, since there are hundreds of thousands of fonts in use for Roman characters alone. Therefore, it is infeasible to collect a sufficient set of real-world training images.

One way to overcome the training data challenge is to synthesize the training set by rendering text fragments for all the necessary fonts. However, to attain effective models with this strategy, we must
face the domain mismatch between synthetic and real-world text images (Chen et al. (2014)). For example, it is common for designers to edit the spacing, aspect ratio or alignment of text, to make the text fit other design components. The result is that characters in real-world images are spaced, stretched and distorted in numerous ways. For example, Fig. 1 (a) and (b) depict typical examples of character spacing and aspect ratio differences between synthetic and real-world images. Other perturbations, such as background clutter, perspective distortion, noise, and blur, are also ubiquitous. A few more challenging real-world examples are displayed in Fig. 1 (c). In Chen et al. (2014), the authors tried to overcome this difficulty by adding different degradations, e.g., noise, blur and geometric distortions, to synthetic data. In the end, introducing all possible real-world degradations into the training data is infeasible. Consequently, the domain mismatch problem remains, despite heroic attempts to synthesize realistic training data.

We address this domain mismatch problem in font recognition, by leveraging a large training corpus of synthetic data to obtain effective features for the classification task, while introducing an adaptation technique based on Stacked Convolutional Auto-Encoder (SCAE) with the help of a large corpus of unlabeled real-world text images. The framework employs a multi-layer Convolutional Neural Network (CNN) architecture which is decomposed into two sub-networks: a "shared" low-level sub-network which is learned from the composite set of synthetic and real-world data, and a second high-level sub-network that learns a deep classifier using the low-level features. The proposed algorithm, named DeepFont, reaches an impressive performance on a large collection of real-world test images, covering an extensive variety of font categories. Notably, it correctly recognizes all of the challenging examples in Fig. 1. Our contributions in this paper are summarized as follows:

1. A learning framework to leverage both synthetic and real-world data, despite domain mismatch, using a multi-layer CNN decomposition and SCAE-based domain adaptation. Specifically, the framework is successfully applied to font recognition.
2. A DeepFont system that achieves a recognition accuracy of higher than 80% (top-5) on our collected dataset, which surpasses the state-of-the-arts (Chen et al. 2014) significantly.
3. A large set of real-world labeled data and a large corpus of unlabeled real-world images for studying the font domain adaptation problem, covering 2,383 font classes, which will be made publicly available soon.

The domain mismatch between synthetic and real-world data on the lower-level statistics can occur in more scenarios, such as real-world face recognition from rendered images or sketches, recognizing characters in real-scene with synthetic training, human pose estimation with synthetic images generated from 3D human body models. We conjecture that our framework can be applicable to those scenarios as well, where labeled real-world data is scarce but synthetic data can be easily rendered.

2 RELATED WORK

The authors in Dai et al. (2007) treated a problem of this type as a standard semi-supervised problem by ignoring the domain difference and considering the source instances as labeled data and the target
ones as unlabeled data. The authors in [Nguyen et al. (2013)] performed adaptations on a hierarchy of features instead of a single hand-crafted feature. In [Raina et al. (2007)], the proposed transfer learning approach relies on the unsupervised learning of representations, which related itself closely to Deep Learning (DL) methods. It is well known that the internal layers of the deep networks can act as a generic extractor of image representation, which can be pre-trained on one dataset and then re-used on other target tasks [Glorot et al. (2011); Bengio et al. (2011)].

For instances coming from an overlapping but different distribution, Bengio et al. hypothesized in [Bengio (2009)] that more levels of representation can give rise to more abstract, more general features of the raw input, and that the lower layers of the predictor compute a hierarchy of features that can be shared across variants of the input distribution. As a specific example of utilizing “shared” features across domains, the method in [Glorot et al. (2011)] learns a shared feature extraction in an unsupervised fashion from all the available domains using a Stacked Auto Encoder (SAE). A linear SVM is trained on the transformed labeled data from the source domain using SAE, and to be tested on the target domain. Interestingly, they found out that transfer learning across domains will always lead to performance improvements, even those domains are not directly correlated. A theoretical analysis of generalization improvements due to sharing of intermediate features across different tasks already points towards that explanation [Baxter (1995)]. However, this category of problem still remains not very well studied.

In the literature of text analysis, there have been certain efforts touching the problem the real-to-synthetic domain adaptation and feature sharing, a part of which are closely relevant to deep learning. In [Wang et al. (2012)], the authors trained two CNNs for text detection and recognition tasks respectively. The training adopts synthetic data to augment the insufficient real-world dataset, but the differences between the two data domains were not considered. A deep learning-based text spotting method proposed in [Jaderberg et al. (2014)] shared the first two layers among several CNN-based classifiers along the working pipeline. It improves performances in classification tasks with less informative labels (text/no-text classification), or tasks with fewer training examples (case-sensitive character classification, bigram classification).

3 Domain Adaption Approach

3.1 Basic CNN Architecture

Our basic CNN architecture is similar to the popular ImageNet CNN structure in [Krizhevsky et al. (2012)], as depicted in Fig. 2. The numbers along with the network pipeline specify the dimensions of outputs of corresponding layers. The input is a $105 \times 105$ patch sampled from a “normalized” image, since the aspect (height-to-width) ratio of most standard-sized characters are close to one, whereas a square window usually does 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, such as ligatures. The normalization of the input images starts with cropping each word with a tight bounding box, and then cut off all parts under the descender line for each word (for most characters, their main recognizable features lie above the descender line). The cropped images are then scaled to fixed height of 105 pixels. We then introduce the *squeezing* operation, that scales the width of the height-normalized image to be of a constant ratio relative to the height (2.5 in all our experiments). Note that the squeezing operation is equivalent to producing “long” rectangular input patches. More training details of CNN are included in the supplementary material.

Figure 2: The CNN architecture in the DeepFont system, and its decomposition marked by different colors ($N=8, K=2$).
3.2 CNN Decomposition

When the CNN model trained fully on a synthetic dataset, it witnesses a significant performance drop when testing on real-world data, compared to when applied to another synthetic validation set (see Section 5). This also happens with other models such as in Chen et al. (2014), which uses training and testing sets of similar properties to ours. This alludes to discrepancies between the distributions of synthetic and real-world examples.

Tradition approaches to handle this gap include pre-processing steps applied on the training and/or testing data (Chen et al. (2014)), semi-supervised learning by using the target domain as unlabeled data (Weston et al. (2012)). The domain adaptation method in Bengio et al. (2011) extracts low-level features that represent both the synthetic and real-world data. Our goal is to build on the strength of modern convolutional networks and to apply domain adaption directly within the CNN learning framework, without increasing its model complexity. In order to keep the model structure and number of parameter fixed, we propose to “decompose” the $N$ basic CNN layers into two sub-networks to be learned sequentially:

- **Unsupervised cross-domain sub-network** $C_u$, which consists of the first $K$ layers of CNN. It accounts for extracting low-level visual features shared by both synthetic and real-world data domains. $C_u$ will be trained in an unsupervised way, using unlabeled data from both domains. It constitutes a crucial step that minimizes the low-level feature gap.

- **Supervised domain-specific sub-network** $C_s$, which consists of the remaining $N - K$ layers. It accounts for learning higher-level discriminative features for classification, 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.

We show an example of the proposed CNN decomposition in Fig. 2. The $C_u$ and $C_s$ parts are marked by red and green colors, respectively, with $N = 8$ and $K = 2$.

We are not the first to look into an essentially “hierarchical” deep architecture for domain adaption. The authors in Glorot et al. (2011) used data from the union of all the domains to learn their shared features, which is different from many previous domain adaptation methods that focus on learning features in a unsupervised way from the target domain only. However, their entire network hierarchy is learned in a unsupervised fashion, except for a simple linear classifier trained on top of the network, i.e., $K = N - 1$. In Wang et al. (2012), the CNN learned a set of filters from raw images as the first layer, and those low-level filters are fixed when training higher layers of the same CNN, i.e., $K = 1$. In other words, they either adopt a simple feature extractor ($K = 1$), or apply a shallow classifier ($K = N - 1$). Our CNN decomposition is different from prior work in that:

- Our feature extractor $C_u$ and classier $C_s$ are both deep sub-networks with more than one layer (both $K$ and $N - K$ are larger than 1), which means that both are able to perform more sophisticated learning. We analyze the tradeoffs in applying various values of $K$ in Section 5.2, and conclude that the choice $K = 2$ achieves the best overall performance.

- We learn “shared-feature” convolutional filters rather than a fully-connected network such as in Glorot et al. (2011), the former of which is more suitable for visual feature extractions. Those filters are not crafted from raw data as in Wang et al. (2012), but are learned by a deep architecture.

### 3.2.1 Unsupervised cross-domain learning using Auto-Encoders

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 (Bengio et al. (2007); Weston et al. (2012)). However, they rely mostly on fully-connected models and ignore the 2D image structure. In Masci et al. (2011), 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 Vincent et al. (2008), 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.
Figure 3: The Stacked Convolutional Auto-Encoder (SCAE) architecture.

plots the SCAE architecture for $K = 2$ layers in our case. Note its first and second half are mirror-symmetrical, with our belief that every convolutional layer needs to have a corresponding deconvolutional layer to reconstruct the input. Note also that the first two convolutional layers have an identical topology to the first two layers in Fig. 2. The max-pooling layer not only pools within feature maps, but also records the locations of the maxima in each pooling region. The unpooling layer takes the elements from input and places them in the recorded maxima locations, while the remaining elements are set zero. After SCAE is learned, its Conv. Layers 1 and 2 are imported to the CNN in Fig. 2 as the $C_u$ sub-network. The $C_s$ sub-network, based on the shared features output by $C_u$, are then trained in a supervised manner similarly as described in Section 3.1.

While there are multiple SCAE implementations available, we adopt the implementation by Paine et al. (2014) as it has shown some improvements over the one in Masci et al. (2011) on a common benchmark (CIFAR-10). Their model is similar to the network in Zeiler et al. (2011) but without using sparse coding, and introducing zero-biases (Memisevic et al. (2014)) and ReLUs in the convolutional layers.

3.2.2 Reducing Domain Gap with Synthetic Data Preparation

In deep learning literature, it is popular to artificially enlarge the dataset using label-preserving transformations to reduce overfitting to the training data. In Krizhevsky et al. (2012), the authors applied image translations and horizontal reflections to the training images, as well as altering the intensities of their RGB channels. In DeepFont, we explore specific data preprocessing steps to further attain a better model generalization ability.

Training Stage: Chen et al. (2014) added moderate distortions and corruptions to the synthetic text images, including:

- 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.

The above pre-processing covers standard perturbations for general images, and we adopt the same convention in processing our training data too. However, as a very particular type of images, text images have various real-world appearances caused by a few more kinds of specific handlings. In Fig. 3(a) and (b), we display the implications of Character Spacing and Aspect Ratio that differentiate real world images from synthetic ones. More quantitative analysis can be found in the supplementary material. Based on these observations, we add two additional font-specific preprocessing steps to our training data preparation:

- 5. Variable Character Spacing: when rendering synthetic images, we set the character spacing (by pixel) in each image to be a Gaussian random variable of mean 10 and standard deviation 40. The spacing variable value is also bounded by [0, 50].
- 6. Variable Aspect Ratio: Before cropping each image into a input patch, the image, with height fixed, is squeezed in width by a random ratio, drawn from a uniform distribution between $\frac{5}{6}$ and $\frac{7}{6}$. Note this operation is independent from the squeezing operation introduced in Section 3.1 - the purpose of the former is to vary the aspect ratio for each patch.
Note that these steps are not useful for the method in [Chen et al. (2014)] because it exploits very localized features. However, as we show in our experiments, these steps lead to significant performance improvements in our DeepFont system. Overall, our data preparation includes steps 1-6.

**Testing Stage** We adopt multi-scale multi-view testing to improve the result robustness. 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 $105 \times 105$ 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.

### 4 Dataset

Due to the limited amount of labeled real-world font images, one needs to rely on synthetic images to obtain a large part of the training data. In [Chen et al. (2014)], the authors selected 2,420 fonts and rendered long English words sampled from a large corpus. We removed some script fonts, ending up with a subset of 2,383 font classes. We follow the same procedure in [Chen et al. (2014)] to generate tightly cropped, gray-scale text images. We cut off all parts under the baseline for each word, since for most characters, their major parts with recognizable features lie above the baseline. Finally, we scale all the images to have 105 pixels in height. For each class, we assign 1,000 images for training and 100 for validation.

Collecting and labeling real-world examples is notoriously hard and thus a real-world labeled dataset has been absent for long. In [Chen et al. (2014)], the authors collected a small dataset "VFRWild325" ("VFR" short for "visual font recognition"), consisting of 325 real-world test images, covering 93 classes. However, the small dataset size makes it difficult to properly evaluate the effectiveness of recognition algorithms. We collected 201,780 text images from various typography forums, such as myfonts.com, where people post these images seeking help from experts to identify the fonts. Most of them come with a hand-annotated font label which may be inaccurate. Unfortunately, only a very small portion of them falls into our list of 2,383 fonts. All images are first converted into gray scale. Those images with our target class labels are then selected and inspected if the labels are correct, by visually comparing each candidate image with the synthetic image rendered with the same label font and text. Images with correct verified class labels are then manually cropped with tight bounding boxes and normalized in the same way as for the synthetic data. We obtain 4,384 real-world test images with reliable labels, covering 617 classes (out of 2,383). Compared to the synthetic data, these images typically have much larger appearance variations caused by scaling, background clutter, lighting, noise, perspective distortions, and compression artifacts. Removing the 4,384 labeled images from the full set, we are left with 197,396 unlabeled real-world images which we denote VFR_real_u.

### 5 Experiments

#### 5.1 Analysis of Domain Mismatch

We first analyze the domain mismatch between synthetic and real-world data, and qualitatively examine the merits of our synthetic data preparation, by a cross-validation experiment. First we define five dataset variations generated from VFR_syn_train and VFR_real_u, which will be used for various comparison experiments hereinafter. These are denoted by the letters N, S, F, R and FR and are explained in Table 1 ("preparation" refers to the process in Section 3.2.2).

We train five separate SCAEs, all of the same architecture as in Fig. 3, using the above five training data variants. The training and testing errors are all measured by relative MSEs (normalized by the total energy) and compared in Table 1. The testing errors are evaluated on both the unprepared synthetic dataset N and the real-world dataset R. Ideally, the better the SCAE captures the features from a domain, the smaller the reconstruction error will be on that domain.
Table 1: Comparison of Training and Testing Errors (%) of Five SCAEs (K=2)

| Methods | Training Data | Train | Test |
|---------|---------------|-------|------|
| SCAE N  | N: VFR_syn_train, no data preparation | 0.02  | 3.54 | 31.28 |
| SCAE S  | S: VFR_syn_train, standard preparation (1-4) | 0.21  | 2.24 | 19.34 |
| SCAE F  | F: VFR_syn_train, full preparation (1-6) | 1.20  | 1.67 | 15.26 |
| SCAE R  | R: VFR_real_u, real unlabeled dataset | 9.64  | 5.73 | 10.87 |
| SCAE FR | FR Combination of data from F and R | 6.52  | 2.02 | 14.01 |

As revealed by the training errors, real-world data contains rich visual variations and is more difficult to fit. The sharp performance drop from N to R of SCAE N indicates that the convolutional features for synthetic and real data are quite different. This gap is reduced in SCAE S, and further in SCAE F, which validates the effectiveness of adding font-specific data preparation steps. SCAE R fits the real-world data best, at the expense of a larger error on N. SCAE FR achieves an overall best reconstruction performance of both synthetic and real-world images.

5.2 OPTIMAL CNN DECOMPOSITION

Given a fixed network complexity (N layers), one may ask about how to best decompose the hierarchy to maximize the overall classification performance on real-world data. Intuitively, we should have large enough amount of lower-level feature extractor layers for a good reconstruction performance as well as enough subsequent layers for good classification of labeled data. Thus, K (the size of the unsupervised SCAE based sub-network) should neither be too small nor too large.

Table 2 shows that while the classification training error increases with K, the testing error does not vary monotonically. The best performance is obtained with K = 2 (3 slightly above), where smaller or larger values of K give substantially worse performance. When K = 5 for example, all layers are learned using SCAE, leading to the worst results. Rather than learning all hidden layers by unsupervised training, as suggested in Glorot et al. (2011) and other DL-based transfer learning work, our CNN decomposition reaches its optimal performance when higher-layer convolutional filters are still trained by supervised data. Visual inspection of reconstruction results of a real-world example, using SCAE FR with Different K values (see Fig. 7 in the supplementary material), shows that a larger K causes less information loss during feature extraction and leads better reconstruction. However this comes at the expense of worse classification results since features that capture noise and irrelevant high frequency details (e.g. texture) might hamper recognition performance in the latter stages. The optimal K = 2 value, corresponds to a proper “content-aware” smoothening, filtering out “noisy” details while keeping recognizable structural properties of the font style.

5.3 PERFORMANCE COMPARISONS ON SYNTHETIC AND REAL-WORLD DATASETS

We implemented and evaluated the local feature embedding-based algorithm (LFE) in Chen et al. (2014) as a baseline, and include the four different DeepFont models as specified in Table 3. The first two models are trained in a fully supervised manner on F, without any decomposition applied. For
Table 2: Top-5 testing Errors (%) for different CNN decompositions, varying $K$ with $N = 8$

| $K$ | 0    | 1    | 2    | 3    | 4    | 5    |
|-----|------|------|------|------|------|------|
|     | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test | Train| VFR_real_test |
| 0   | 8.46 | 20.72 | 9.88 | 20.31 | 11.23 | 18.21 | 12.54 | 18.96 | 15.21 | 22.52 | 17.88 | 25.97 |

Table 3: Comparison of Training and Testing Errors on Synthetic and Real-world Datasets (%)

| Methods              | Training Data | Training Error | VFR_syn_val | VFRWild325 | VFR_real_test |
|----------------------|---------------|----------------|-------------|-------------|---------------|
|                      | $C_u$ | $C_s$ | Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 | Top-1 | Top-5 |
| LFE                  | /    | /    | /    | 26.50 | 6.55 | 44.13 | 30.25 | NA   | NA   |
| DeepFont S           | /    | F    | 0.84 | 1.03  | 0    | 64.60 | 57.23 | 57.51 | 50.76 |
| DeepFont F           | /    | F    | 8.46 | 7.40  | 0    | 43.10 | 22.47 | 33.30 | 20.72 |
| DeepFont CAE.FR      | FR   | F    | 11.23| 6.58  | 0    | 38.15 | 20.62 | 28.58 | 18.21 |
| DeepFont CAE.R       | R    | F    | 13.67| 8.21  | 1.26 | 44.62 | 29.23 | 39.46 | 27.33 |

6 Conclusion

We introduce a new domain adaption framework that leverages the domain mismatch between synthetic training and real-world testing data. We demonstrate the DeepFont model as its successful application to the challenging font recognition task. To address the domain gap, we introduce CNN decomposition and SCAE-based domain adaptation, as well as font-specific data preparation, that lead to a higher-than-80% top-5 accuracy on a large new dataset. We believe the framework is applicable to a wide range of problems where real-world labeled data is scarce while synthetic data is easy to generate.
7 ACKNOWLEDGEMENT

Our SCAE implementation is in part based on Tom Le Paine, Pooya Khorrami, and Wei Han’s open source neural network library: https://github.com/ifp-uiuc/anna. Their relevant research on unsupervised pre-training can be further checked in Paine et al. (2014). We would also like to thank them for many helpful discussions.

REFERENCES

Baxter, J. Learning internal representations. In Proceedings of COLT, pp. 311–320. ACM, 1995.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. Machine learning, 79(1-2):151–175, 2010.

Bengio, Y. Learning deep architectures for ai. Foundations and trends® in Machine Learning, 2 (1):1–127, 2009.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. Proceedings of NIPS, 19:153, 2007.

Bengio, Y., Bastien, F., et al. Deep learners benefit more from out-of-distribution examples. In Proceedings of AISTATS, pp. 164–172, 2011.

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

Dai, W., Xue, Gu.and Yang, Qiang, and Yu, Y. Transferring naive bayes classifiers for text classification. In Proceedings of AAAI, volume 22, pp. 540, 2007.

Glorot, X., Bordes, A., and Bengio, Y. Domain adaptation for large-scale sentiment classification: A deep learning approach. In Proceedings of ICML, pp. 513–520, 2011.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Deep features for text spotting. In Proceedings of ECCV, pp. 512–528. Springer, 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Proceedings of NIPS, pp. 1097–1105, 2012.

Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. Stacked convolutional auto-encoders for hierarchical feature extraction. In Proceedings of ICANN, pp. 52–59. Springer, 2011.

Memisevic, R., Konda, K., and Krueger, D. Zero-bias autoencoders and the benefits of co-adapting features. arXiv preprint arXiv:1402.3337, 2014.

Nguyen, H. V, Ho, H. T, Patel, V. M., and Chellappa, R. Joint hierarchical domain adaptation and feature learning. IEEE Trans. PAMI, submitted, 2013.

Paine, T., Khorrami, P., Han, W., and Huang, T. S. An analysis of unsupervised pre-training in light of recent advances. arXiv preprint arXiv:1412.6597, 2014.

Pan, S. and Yang, Qi. A survey on transfer learning. IEEE Trans. KDE, 22(10):1345–1359, 2010.

Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. Self-taught learning: transfer learning from unlabeled data. In Proceedings of ICML, pp. 759–766. ACM, 2007.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P-A. Extracting and composing robust features with denoising autoencoders. In Proceedings of ICML, pp. 1096–1103. ACM, 2008.

Wang, T., Wu, David J, Coates, A., and Ng, A. Y. End-to-end text recognition with convolutional neural networks. In Proceedings of ICLR, pp. 3304–3308. IEEE, 2012.

Weston, J., Ratle, F., Mobahi, H., and Collobert, R. Deep learning via semi-supervised embedding. In Neural Networks: Tricks of the Trade, pp. 639–655. Springer, 2012.

Zeiler, M., Taylor, G., and Fergus, R. Adaptive deconvolutional networks for mid and high level feature learning. In Proceedings of ICCV, pp. 2018–2025. IEEE, 2011.
8 Supplementary Material

8.1 Training Details of Basic CNN

The basic CNN contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected, the names of which are denoted in Fig. 2. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer. Conv. Layer 1 filters the input patch with 64 kernels of size $11 \times 11$, with a stride of 4. Conv. Layer 2 takes as input the (response-normalized and pooled) output of the Conv. Layer 1, and filters each channel with 64 kernels of size $5 \times 5$, with a stride of 1. Conv. Layers 3, 4, and 5 are connected to one another without any intervening pooling or normalization layers, to avoid overly downsampling. They all have 256 kernels of size $3 \times 3$, with a stride of 1. Conv. Layers 1 and 2 are followed by a response-normalization and then by max-pooling. After Conv. Layer 5, the three fully-connected layers have 4096, 4096, and 2383 neurons respectively. The output of the last fully-connected layer is fed to a 2,383-way softmax which produces a distribution over the 2,383 class labels. The network maximizes a multinomial logistic regression objective. The training is with a batch size of 128, momentum of 0.9 and weight decay of 0.0005. The weights in all layers are initialized from a zero-mean Gaussian distribution with standard deviation 0.01. We start the learning rate at 0.01, and follow a common heuristic to manually divide the learning rate by 10 when the validation error rate stops decreasing with the current learning rate. The “dropout” technique is applied to the first two fully-connected layers during training.

8.2 Dataset Comparison

Table 4: Comparison of all datasets used in this paper

| Dataset name          | Source | Labeled? | Purpose | Size  | Class |
|-----------------------|--------|----------|---------|-------|-------|
| VFRWild325 (Chen et al. [2014]) | Real   | Y        | Test    | 325   | 93    |
| VFR_real_test         | Real   | Y        | Test    | 4,384 | 617   |
| VFR_real_u            | Real   | N        | Train   | 197,396 |      |
| VFR_syn_train         | Synthetic | Y    | Train   | 2,383,000 | 2,383 |
| VFR_syn_val           | Synthetic | Y    | Test    | 238,300  | 2,383 |

8.3 More Experiments

Figure 5: The effects of data preparation steps. (a)-(d): synthetic images of the same text, with no preparation, standard preparation (1-4), preparations 5-6, and full preparation (1-6), respectively. (e) comparison of relative layer-wise responses of (a)-(d) through DeepFont CAE.FR.

Implications of Spacing and Aspect Ratio Changes We take the real-world example from Fig. 1 (a), and synthesize a series of images, all with the same text, but becoming gradually more visually similar to their real-world counterpart. Specially, (a) is synthesized with standards spacing and
ratio setting and no preparation; (b) is (a) with standard preparation (1-4) added; (c) synthesized with spacing and aspect ratio customized to be identical to those of Fig. 1 (a); (d) adds standard preparation (1-4) to (c). We input images (a)-(d) through DeepFont CAE FR, and obtain the layer-wise intermediate outputs for each image. For each image (a) to (d), we normalize their layer-wise outputs, by calculating the relative MSEs on each layer’s output, to the output of the same layer by feeding Fig. 1 (a) through the same DeepFont model. Fig. 5 (e) shows that the introduced preparations, especially the spacing and aspect ratio changes, reduce significantly the gap between real-world and synthetic features. A few synthetic patches after full data preparation (1-6) are displayed in Fig. 6; they possess a much more visually similar appearance to real-world data.

Figure 6: Examples of synthetic training $105 \times 105$ patches after pre-processing steps 1-6.

Reconstruction Results Using SCAE FR with Different $K$'s In Fig. 7 we train SCAEs corresponding to different $K$ values (1, 2, 4 and 5), in the same manner as we trained SCAE FR ($K = 2$) in Section 5.1. Predictably, we observe that a larger $K$ leads to a better reconstruction. However, increasing $K$ does not always lead to a better recognition, since unwanted features might hamper the performance, e.g., the decoration textures on characters not belonging to the original font style.

![Figure 7: The reconstruction results of a real-world patch using SCAE FR, with different $K$ values.](image)

Failure Examples of DeepFont Fig. 8 lists some failure cases of DeepFont CAE FR. For example, the leftmost image contains extra “fluff” decorations, which is nonexistent in the original font, along text boundaries that will make the algorithm incorrectly map it to some “artistic” fonts. Others are affected by 3-D effects, strong obstacles in foreground, and in background, respectively. Those examples fail mostly because in the DeepFont model, there are neither specific preparation steps taken to handle those effects, nor enough similar examples in VFR_real_u to learn proper features.

![Figure 8: Failure examples using DeepFont CAE FR.](image)