Unsupervised Cross-Domain Image Generation

Xinru Hua, Davis Rempe, and Haotian Zhang

Abstract— **We explore the problem of general domain transfer by replicating a recent method presented at ICLR 2017. This method maps a sample from one domain to another using a generative adversarial network (GAN) in an unsupervised fashion. We attempt to replicate this method in two visual application areas - digits and faces - and perform additional analysis on various components of the approach. We achieve similar visual results for digits but not faces, finding that the training procedure is crucial to a successful GAN implementation.**

I. INTRODUCTION AND MOTIVATION

This project delves into a fundamental machine learning problem - generalized domain transfer - by reproducing and analyzing ”Unsupervised Cross-Domain Image Generation” [1]. This work from Facebook AI Research studies the task of ”transferring a sample in one domain to an analog sample in another domain.” They present a method to successfully do so in visual domains using two different application examples.

The work of [1] deals with the problem of what they call ”general analogy synthesis.” This is an instance of what is usually referred to as domain transfer, which seeks to map samples from some domain S to a related domain T. In particular, the paper wants to learn a generative mapping that produces a plausible representation in T given a sample in S, and to do so in a completely unsupervised fashion. This task, though easily and intuitively done by humans, is extremely difficult for machine learning models. The introduced method achieves unsupervised domain transfer through a modified generative adversarial network (GAN) architecture which takes advantage of a pre-trained feature-encoding block f and a ternary discriminator. Importantly, training uses a loss function which pushes the network to keep f invariant to the generative transform. After the network has been trained, the generator takes in a sample from S (i.e. an image of a house number digit) and outputs the representation in the target domain T (i.e. the same digit in handwritten form). The model is explained in more detail in the Methods section.

Though the problem of domain transfer has been extensively explored recently, especially in visual domains, [1] focuses on constructing a more general solution that can transfer between any two domains. There is no reason to believe their method won’t work for the general problem, but the applications to which they apply the new model are restricted to visual domains.

For our project, we seek to confirm the results in [1] by implementing the Domain Transfer Network (DTN) for the two applications presented in the paper: 1) transferring style between two different image domains representing digits, and 2) generating an emoji given a face image. Furthermore, we experiment with a modified loss function for digit transfer, and explore a wide range of GAN training techniques since the original paper offers absolutely no indication to the training methods used to achieve their results.

TABLE I

DATASETS USED IN THE PROJECT.

Application	Name	Training Size Used	Type
Digit Transfer	SVHN	531,131	RGB
	MNIST	60,000	Grayscale
Face Transfer	MS-Celeb-1M	912,224	RGB
	Bitmoji	1,000,000	RGB

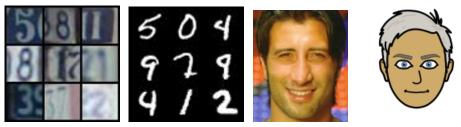

Fig. 1. Examples of SVHN, MNIST, MS-Celeb, and Bitmoji data.

II. DATASETS

We use four different image datasets in the project which are laid out in Table I: a source and target domain for each of the two application areas. Examples for each dataset are shown in Figure 1.

For digit transfer, we use images from the Street View House Numbers (SVHN) dataset [2] and MNIST database of handwritten digits [3]. These are the same sets used in the original paper. SVHN contains a training split with 73,257 images, an extra training split with 531,131 images, and a test split with 26,032 images. The MNIST training set is size 60,000 and has a test split of 10,000 additional digit images. All images in these two datasets are resized to (32, 32) and SVHN images are normalized to [-1, 1].

For the face transfer source domain, the original paper uses ”a set **s** of one million random images without identity information”, but neglects to provide too many additional details. For our implementation we use a subset of size 912,224 from Microsoft’s MS-Celeb-1M [4] which provides face images of one million celebrities. The dataset also contains 500 development images. We use the dataset version that comes pre-cropped and aligned by Microsoft’s algorithm.

For the emoji target domain, [1] says that ”the set **t** consists of assorted facial avatars (*emoji*) created by an online service (bitmoji.com),” but offers no indication of emoji creation methodology and how many emojis were necessary to generate. This left us to come up with our own pipeline for emoji generation that is constrained using only hints we could extrapolate from the original paper. To create

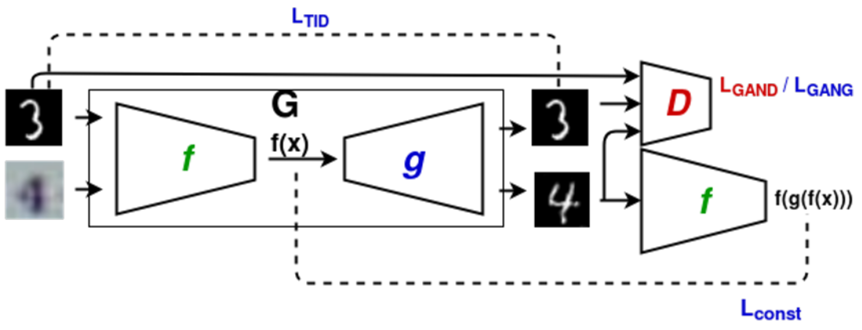

Fig. 2. Outline of the domain transfer network and related loss functions.

a large and diverse set of emojis, we wrote a script that randomly chooses a reasonable set of emoji attributes then calls Bitmoji's API to create the avatar and save an image. The controlled attributes include emoji gender, face shape, hair/eye color, hair length and style, jaw shape, eyebrows, eyes, nose, ears, facial hair, glasses, and lipstick. Using this method we created our own dataset of 1,000,000 emoji images. All face and emoji images are resized to (96, 96) and normalized to [-1, 1].

III. METHODS

The model architecture is shown in Figure 2. Block f is the feature encoder, block g is the generator, and block D is the discriminator.

Unlike other style transfer algorithms, the paper introduces a function f to a typical generative adversarial network (GAN). The function f should remain unchanged with the generative transformation: $f(x) = f(G(x))$ where $G = (g \circ f)$. f is pretrained and remains constant during training our DTN. g takes in the feature representation from f and outputs a generated image. That is, given an image from source dataset, the model could generate an image that has the same features as the given image and has the same style as other images in target dataset. As shown in Figure 2, the green f block and blue g block form the G.

The discriminator (D) returns a ternary classification, which is more robust than the typical binary classifier used in regular GANs. For an image, D returns classfication for three classes: 1) a source sample that has been through the generator, 2) a target sample that has been through the generator, and 3) a target sample.

The loss functions contain generator's loss and discriminator's loss. The generator's loss function has four terms: $\mathrm{L_{GANG}}$, $\mathrm{L_{CONST}}$, $\mathrm{L_{TID}}$, $\mathrm{L_{TV}}$. The equation of L_G is:

$$L_G = L_{GANG} + \alpha L_{Const} + \beta L_{TID} + \gamma L_{TV}$$

where:

$$L_{GANG} = -\sum_{x\in\mathbf{s}} \log D_3(g(f(x))) - \sum_{x\in\mathbf{t}} \log D_3(g(f(x)))$$

$$L_{CONST} = \sum_{x\in\mathbf{s}} d(f(x), f(g(f(x))))$$

$$L_{TID} = \sum_{x\in\mathbf{t}} d_2(x, G(x))$$

$$L_{TV}(z) = \sum_{i,j}((z_{i,j+1} - z_{i,j})^2 + (z_{i+1,j} - z_{i,j})^2)^{\frac{B}{2}}$$

$\mathrm{L_{GANG}}$ measures how well generator tricks the discrimiantor for source and target images. $\mathrm{L_{CONST}}$ wants to keep $f(G(s))$ and $f(s)$ identical, so they have the same encoded features. $\mathrm{L_{TID}}$ wants G to remain the identity mapping when it takes in some t from the target domain. There is also the $\mathrm{L_{TV}}$ term which smooths the generated images by pixel. The loss is defined on the generated image $z = [z_{i,j}] = G(x)$. $\mathrm{L_{TV}}$ is not used in digit transfer model, but it is used in face transfer model.

The discriminator's loss function is the sum of its classification mistakes in each class:

$$L_D = -\sum_{x\in\mathbf{s}} \log D_1(g(f(x))) \\ -\sum_{x\in\mathbf{t}} \log D_2(g(f(x))) - \sum_{x\in\mathbf{t}} \log D_3(x)$$

In the paper, both d and d_2 are MSE loss. For calculating $\mathrm{L_{GANG}}$ and $\mathrm{L_D}$, we used cross entropy loss.

A. *DIGIT TRANSFER NETWORK*

We trained our f starting from training a classifier of digits on SVHN dataset. The network structure of the classifier (f) consists of four blocks of convolutional layers and a ReLU nonlinearity layer. For each convolution layer, we used a stride of 2 to shrink the size of the image by half in each layer, instead of using max pooling layer after each convolutional layer. After obtaining the $128 \times 1 \times 1$ feature representation, we used a fully-connected layer to shrink it to the output size of 10 (total number of labels for the digits).

After the classifier is trained, we took the first 7 layers out of 9 as the f block in our digit model, so that it encodes the features from the images.

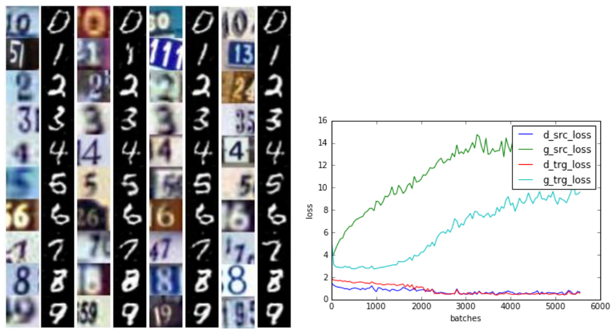

Fig. 3. Best results of digit transfer and loss curves during training

TABLE II
EVALUATION ACCURACY ON DIFFERENT MODELS

	Training set	Test set
Our best model	81.50%	74.50%
No L_{const}	22.06%	18.85%
Cross entropy for L_{const}	96.92%	91.09%
Referenced paper	N/A	90.66%

Generator g maps f's feature vector to a 32×32 grayscale image. g employs four blocks of transposed convolution, batch-normalization, and ReLU. Except the first layer, all transposed convolution layers use kernel size of 4, stride of 2 and padding of 1, so that the input is upscaled by 2. We added a Tanh layer at the end, because we want to normalize g's output in $[-1, 1]$ range.

The discriminator D takes in the 32×32 image from the generator and returns 3 possibilities of the image belongs to each 3 class. D employs 5 blocks of convolution, batch-normlization, and Leaky ReLU.

B. *FACE TRANSFER NETWORK*

In the original paper, the network for face image transfer builds upon the representation layer of the DeepFace network [5] (from Facebook) for the important f block, which is not publicly available. We were therefore left to find a viable alternative to use for f. Because DeepFace was meticulously trained on an in-house set of four million labeled face images, any open-source/pre-trained alternative we find is most likely less capable. This may place an upper bound on the quality of domain transfer we can achieve.

The DeepFace representation layer produces a 256d representation of the input face image, trained with the goal of classifying identity, so we looked for pre-trained alternatives similar to this. We found and experimented with two open-source networks for f that have pre-trained implementations in PyTorch, the same framework we use for the entirety of our implementation. The first is OpenFace [6], a Torch implmentation of a face recognition network introduced in [7] that was ported to PyTorch [8]. We used the nn4.small2.v1 version of the network which was trained on the FaceScrub [9] and CASIA-WebFace [10] datasets (about 600,000 images total), and is reported to have reached about 93% accuracy on the Labeled Faces in the Wild (LFW) dataset [11]. OpenFace outputs a 128d vector representation of the input image and is trained so that similar faces are closer together in this feature space. The second is SphereFace [12], which was implemented and trained in PyTorch by [13]. SphereFace outputs a 512d feature vector for each face image, such that faces can be easily compared by measuring the angle between their features (i.e. with cosine similiarity). The pre-trained PyTorch model was trained on CASIA and reaches 99.22% accuracy on LFW.

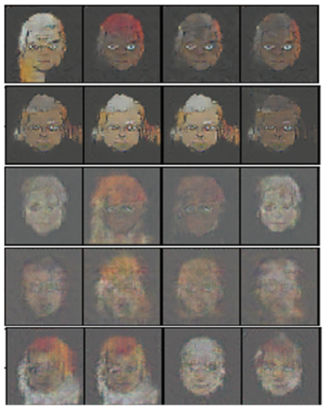

Fig. 4. Example of better results for face to emoji transfer.

The generator g and discriminator D were implemented following the rough outline laid out in the original paper. g takes in the feature vector from f and outputs a 64x64 RGB image. This is bilinearly upsampled to 96x96 and then fed into the discriminator D which outputs 3 "scores", one for each class described above. g consists of 5 blocks, each containing a stride 2 transposed convolution followed by batch normalization and a ReLU. As suggested by the paper, a 1x1 convolution was added after each block in an attempt to lower final L_{CONST} values. After these 5 blocks, a final transposed convolution is performed followed by a Tanh output layer to ensure outputs between -1 and 1. The number of filters for each convolutional layer varied through experimentation, but usually started at 512 in the first layer and slowly decreased to 3 in the final output. D similarly contains 5 blocks, each of which contain a stride 2 convolution, batch normalization, and leaky ReLU non-linearity with $\alpha = 0.2$. The final output is a convolution with 3 filters. Again the number of filters in convolutional layers varied, but generally the first convolution has few filters (like 32, 62, or 128), increased to 4 times more in the middle, then decreases to reach 3 in the output.

IV. RESULTS

We implemented the DTN using PyTorch and ran all the experiments on Google Cloud (with 8 Intel Boardwell CPUs and 1 NIVIDIA Tesla P100/K80 GPU). In this section, our best results in the two applications are shown, with some implementation details.

A. *DIGIT TRANSFER*

For digit transfer, we first pretrained the classifier (f) using the extra training split of SVHN. We just added another fully-

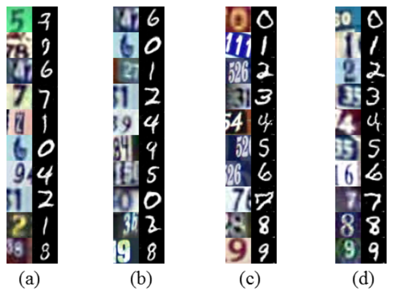
Fig. 5. (a) (b) digit transfer results without L_{const}; (c) (d) digit transfer results using cross entropy loss for L_{const}

connected layer after the encoder and calculate cross entropy loss with original labels. Our classifier achieved accuracy of 93.5% on the test split, which is enough for our DTN model as the author said in the referenced paper.

We trained our best digit transfer model for 12 epochs using the extra training split of SVHN dataset and the training split of MNIST dataset, and used the test split of SVHN dataset for evaluation. Our best model uses loss weight $\alpha = 0.1$, $\beta = 15$ and a constant learning rate of $1e-3$. Adam optimizer is used for both generator and discriminator. Some of the results are shown in Figure 3 with original SVHN images and their transferred images. The transferred digits are clear and indistinguishable from actual MNIST images. These results are visually identical to those achieved in the paper. We also showed our loss curve during training, where you can see the generator's loss going up and discriminator's loss slowly going down as desired.

For quantitative evaluation, we trained an MNIST classifier using the training split of MNIST dataset to achieve 99.2% accuracy on the test split. We judged our transferred results by feeding them into this MNIST classifier with the original labels of SVHN images. The accuracy of our best model and referenced paper are shown in Table II. We achieved 74.5% accuracy compared to 90.66% in the paper. The disparity in accuracy is due to a few problematic digits for our implementation (6 vs. 5 and 1 vs. 7).

B. FACE TRANSFER

We found it extremely difficult to arrive at successful results for face to emoji transfer, and were not able to reach the quality that is seen in the original publication. Because of this, there is not a single "best" model, as we saw similar results with a number of different hyperparameters, architectures, and training schemes. Examples of results from some of our better configurations are shown in Figure 4. Only the network output (emojis) are shown in this picture, as they do not correspond well with the model input (face images) for reasons explained below in the Discussion section. The best correspondence we noticed was transfer of expression like smiles or frowns. Note that these results show only small variation, many are very similar and contain a faint or blurry blend of attributes like skin and hair color.

Though there was not one best model, a number of settings did stay the same across the better-performing configurations.

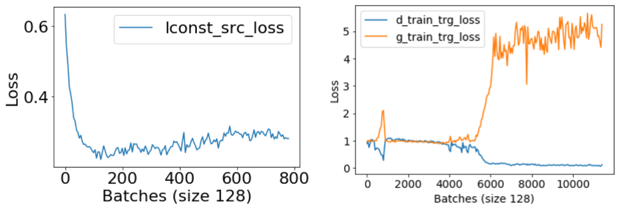

Fig. 6. Problematic loss curves during face transfer training.

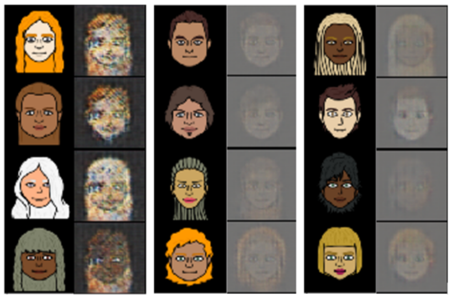
Fig. 7. Results of reconstructing target images using f encoding using MSE loss (left), cosine similarity (middle), and weighted combination of these (right).

First, we ran into issues using the anisotropic total variation loss L_{TV} as defined in the paper. Because of the square root in the equation, the loss was very unstable often causing gradients to NaN. Instead of taking the square root of the sum squared differences, we simply took the absolute value of each difference and summed them. Second, we found the SphereFace network more effective as f than OpenFace, as the 512d output is able to pass more information about face images to g. Lastly, we found a number of parameters to consistently behave better than others; in particular, the Adam optimization algorithm with a constant learning rate of 2e-4, weight decay of 1e-6, and 128 batch size.

V. DISCUSSION AND ADDITIONAL ANALYSIS

The hardest part of this project was training the GAN. The referenced paper did not provide any information of their training strategies and did not release their source code. Therefore, we spent a huge amount of time and energy exploring how to train the model successfully. In this section, we will discuss our results with some additional analysis in the two applications, and summarize some useful training strategies we found during experiments.

A. DIGIT TRANSFER

We got some ideas of training strategies from a successful tensorflow implementation [14]. We seperated loss functions for source images and target images, and trained our model on them individually. In order to balance the generator and discriminator, we also experimented with training schedule for g and D. For the weight of L_{Const}, we found that $\alpha = 15$ as suggested in the referenced paper will generate much noisier images. After exploring different weight, $\alpha = 0.1$ gives our best results.

Under our unsupervised training setting, the loss term L_{Const} is extremely important for maintaining the digit correspondence during transfer. In order to demonstrate the effectiveness of L_{Const}, we designed two additional experiments.

1) Without L_{Const}: We first tried to train the model without adding L_{Const} to the loss function. Some results and evaluation accuracy are shown in Figure 5(a)(b) and Table II. As you can see, generated digits are clear but they obviously lack correspondence during transfer. Through this experiment, we found that removing L_{Const} for several batches at the end of training can help us obtain more clear transferred results.

2) Cross entropy loss for L_{Const}: The original L_{Const} is designed to minimize the distance between the encoded features before and after g. For the digit domain, the digit label is the ideal feature, so we can simplify this loss by comparing the MNIST classification of the transferred image with its original SVHN label. Therefore, we designed a new form a L_{Const} as the cross entropy loss of the MNIST classification and the SVHN labels. Some results and evaluation accuracy are also shown in Figure 5(c)(d) and Table II. We easily achieved excellent digit transfer performance even better than the referenced paper. However, this modification contradicts our initial idea of an "unsupervised method".

B. FACE TRANSFER

Multiple factors contributed to the difficulty in obtaining high quality results for face to emoji transfer. With no guidance from the original paper, the biggest contributor by far was attempting to derive a stable and successful GAN training technique for this specific application. Two of the biggest problems we ran into during this are shown in Figure 6. The left plot shows a pattern for the value of L_{CONST} that was often seen during training: dropping at the beginning, but then quickly leveling off or increasing as the generator starts to focus on optimizing adversarial loss rather than L_{CONST}. This results in emojis with few unique qualities, as we saw in Figure 4. This tends to happen more when the α hyperparameter weighting L_{CONST} is low. So why not turn it up? Well this often leads to our second problem, shown on the right side of Figure 6. When the generator is less focused on optimizing adversarial loss, it becomes too weak to properly combat the discriminator during training making things very unstable. This usually results in the discriminator loss plummeting to near 0, effectively killing any chance of training further.

During this balancing act, the primary parts of the training configuration that we varied were: number of times to train D vs. G and loss thresholding schemes, weights of component loss functions, using cosine similarity for d in L_{CONST} rather than MSE, and the number of filters in convolutional layers of D and g. Though some combinations did improve results, we were still limited by the following factors which are forcibly different from the original paper: our open-source f, the distribution of our emoji dataset, and quality of face image training set.

One interesting characteristic of the DTN is its inherent asymmetry in that f is completely trained on images from the source domain, but is also expected to pull features from target domain images so that L_{TID} can be minimized. To test how our f (SphereFace) dealt with features from emojis, we weighted L_{TID} at 10x the suggested value and set the weight of L_{CONST} to 0 to allow the network to completely focus on reconstructing target emojis well. We tested both MSE loss and cosine similarity for d_2 and found surprisingly that f could indeed encode emoji features fairly well as seen in Figure 7. Using only MSE loss, f captures skin and hair color well, although does have some trouble with blurry results and some hair shape. Cosine similarity, although faint, actually captures the fine-grained details of the emojis (like hair) very well.

C. GAN TRAINING STRATEGIES

GAN is powerful but training a GAN is painful. Here are some tips which work especially well for our task.

- Balance of discriminator and generator - Keeping this balance is the key in training a GAN. In our task, the generator is usually overpowered by the discriminator. The methods we tried for solving this include: a) Train generator more than discriminator, b) Adjust Hyperparameters like adding higher weight for generator loss, c) Add a lower bound on discriminator loss, and d) Change model architecture.
- Normalization - Instead of normalizing input image into standard normal, normalizing images between $[-1, 1]$. Use Tanh as the last layer of the generator.
- Avoid sparse gradients - Instead of downsampling by maxpooling, use strided convolution. Use LeakyReLU instead of ReLU in discriminator.
- Optimization parameters - Change learning rate schedule. Use SGD for discriminator and Adam for generator. Change weight decay.

VI. CONCLUSION

This project explored the problem of unsupervised domain transfer by attempting to replicate and performing addition analysis on [1]. Using PyTorch to implement the GAN architecture, we were able to achieve the same visual results for digit image style transfer, but could not reach the same accuracy without a modified loss function. For face to emoji transfer we could not replicate the visual quality presented in the paper, but did thoroughly explore effective GAN training procedures and found that the asymmetry of the method might not matter as much as originally expected.

If given more time and computation resources, we would be able to concurrently explore more training strategies to improve the performance, especially for face to emoji transfer.

VII. CONTRIBUTIONS

The main contributions of each team member were as follows:

- Xinru - implemented the digit transfer network in the referenced paper, trained it on Google Cloud and explored training strategies.
- Davis - researched and collected face image data, and wrote data loaders. Characterized Bitmoji API and wrote emoji data-generation script. Researched and tested potential f blocks for face transfer, wrote face DTN architecture and training script and performed all training on said network.
- Haotian - implemented the classifier for SVHN and MNIST, and trained them on Google Cloud. Also helped to debug the digit transfer network and trained it for additional analysis.

References

[1] Y. Taigman, A. Polyak, L. Wolf, Unsupervised Cross-Domain Image Generation, *arXiv preprint arXiv:1611.02200* (2016).

[2] N., Yuval, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning, In *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, p. 5, 2011.

[3] Y. LeCun, The MNIST database of handwritten digits, http://yann. lecun. com/exdb/mnist/ (1998).

[4] J. Gao, L. Zhang, X. He, Y. Guo, Y. Hu, MS-Celeb-1M: A Dataset and Benchmark for Large Scale Face Recognition, *European Conference on Computer Vision*, 2016.

[5] Y. Taigman, M. Yang, M.A. Ranzato, L. Wolf, Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

[6] B. Amos, B. Ludwiczuk, M. Satyanarayanan, Openface: A general-purpose face recognition library with mobile applications. In *CMU-CS-16-118, CMU School of Computer Science, Tech. Rep.*, 2016.

[7] F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering. In *CVPR*, 2015.

[8] https://github.com/thnkim/OpenFacePytorch

[9] H.-W. Ng, S. Winkler, A data-driven approach to cleaning large face datasets. In *Proc. IEEE International Conference on Image Processing (ICIP)*, 2014.

[10] D. Yi, Z. Lei, S. Liao, S.Z. Li, Learning Face Representation from Scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[11] G.B. Huang, M. Ramesh, T. Berg, E. Learned-Miller, Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. In *University of Massachusetts, Amherst, Technical Report 07-49, October*, 2007.

[12] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, L. Song, SphereFace: Deep Hypersphere Embedding for Face Recognition. In *CVPR*, 2017.

[13] https://github.com/clcarwin/sphereface˙pytorch

[14] https://github.com/yunjey/domain-transfer-network