A SEED- AUGMENT-TRAIN FRAMEWORK FOR UNIVERSAL DIGIT CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose a Seed-Augment-Train/Transfer (SAT) framework that contains a synthetic seed image dataset generation procedure for languages with different numeral systems using the freely available open font file datasets. This seed dataset of images is then augmented to create a purely synthetic training dataset, which is in turn used to train a deep neural network and test on held-out real world handwritten digits dataset spanning five Indic scripts, Kannada, Tamil, Gujarati, Malayalam and Devanagari . We showcase the efficacy of this approach both qualitatively , by training a BGAN that generates realistic digit images in the five languages, and also qualitatively by testing a CNN trained on the synthetic data on the real-world datasets. This we hope not just establishes an interesting nexus between the font-datasets-world and transfer learning, but also provides a recipe for universal-digit classification in any script.

1 INTRODUCTION

Transfer learning from the *synthetic realm* to the real-world has elicited a lot of attention in the machine learning community recently (See [1; 2; 3]). This typically entails three steps. Generating large volumes of synthetic training data (which is often, a relatively *cheap* process), synthesizing it using a domain specific recipe to address the *reality gap* [4] and training real-world deployment-worthy machine learning models. As seen in [1; 2; 3], deep generative models such as variational autoencoders (VAE) and Generative adversarial Networks (GAN) are oft deployed during the synthesizing process. This paper fits squarely into this category of work, where we tackle of problem of absence of MNIST scale datasets for the Indic scripts to achieve real-world high accuracy digit-classification using synthetic datasets generated by harnessing the Open Font License [1] (OFL) font files freely available on the internet.
The main contributions of this paper are:

1. New real-world MNIST-styled *Indic-digits* datasets for the following five languages: Kannada, Tamil, Gujarati, Malayalam and Devanagari with 1280 images each.
2. The Seed - Augment - Transfer (SAT) framework for the real world digit classification problem. This framework can be extended in a myriad of ways to potentially cover all the digit representations in different language scripts.
3. Successfully trained digit-GAN models for the Indic languages listed above
4. Open sourced scripts and code[2] to reproduce the results disseminated in this paper.

The rest of the paper is organized as follows. In section 2, we cover in detail the real-world dataset generation process. In section 3, we introduce the SAT framework. The classification and the generative model results are presented in Section 4 and we conclude the paper in Section-5 and cover some of the directions that we are currently working on.

[1] https://en.wikipedia.org/wiki/SIL_Open_Font_License
[2] https://anonymous.4open.science/repository/3246a9f1-f5fa-4a74-9822-0dd34b3f8b16/

2 DATASET PREPARATION

We begin with a handwritten grid of 32 x 40 digits on a commercial Mead Cambridge Quad Writing Pad, 8-1/2" x 11", Quad Ruled, White, 80 Sheets/Pad book with a black ink Z-Grip Series — Zebra Pen. We then scan the sheet(s) using a Dell - S3845cdn scanner (specific details here: `https://goo.gl/fykK8k`) and use an image segmentation script to slice the scanned image grid to generate 1280 28 x 28 *mnist-ized* digit images. An example of the raw scanned image for the *Devanagri -Hindi* script is as shown in Fig 1.

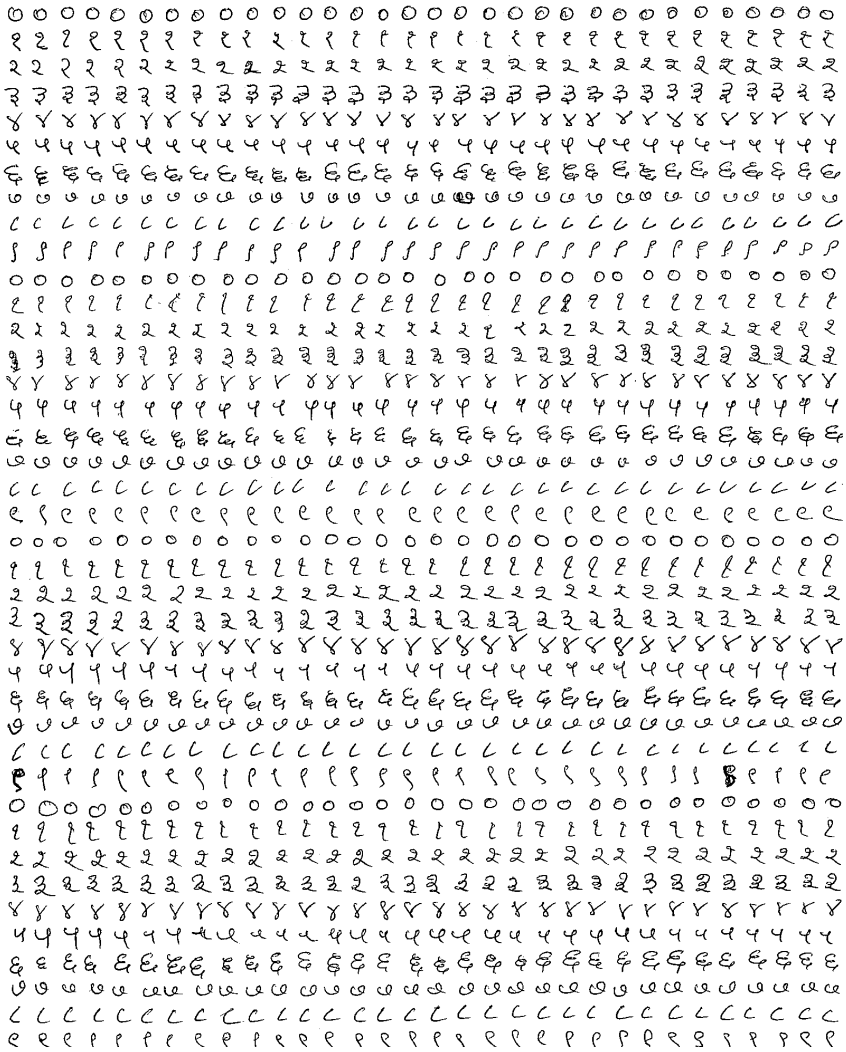

Figure 1: The raw scanned image for the Devanagari handwritten-digits dataset

The class-wise means of the handwritten digit images for the five languages are as shown in Fig 2.

We now perform a novel 2-step sanity check for MNIST compatibility for the languages in the following manner.

2.1 COMMONALITY OF THE ZERO DIGIT

Firstly, we pick images belonging to class-0 and pass it through a CNN pre-trained on the MNIST dataset. Given that the representation of 0 is the same in all the Indic languages and the *standard-MNIST*, we should expect to get high accuracy predictions for this class.

Figure 2: Mean of the 0-9 digits for the languages in the written dataset

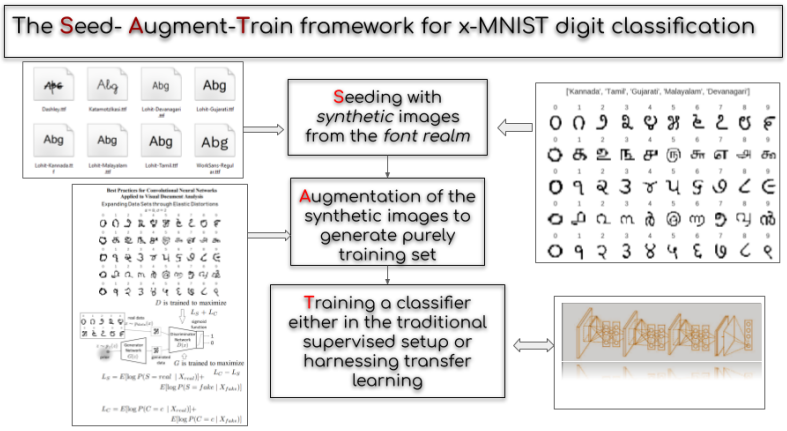

Figure 3: The Seed-Augment-Train/Transfer framework

2.2 Morphological similarity of certain digits with regards to the standard-MNIST digits

Given the Indian roots of the modern HinduArabic numeral system [5], it should come as no surprise that we find morphological similarity in the shape of certain digits between the modern digit system (used in MNIST) and the Indic languages considered in this work. We exploit this to perform our second sanity check, which is showcased with a specific example below.
If we refer to Fig 2, we see that the shapes of the digits for 3 and 7 in Kannada look similar to 2 in the modern script for the Hindu-Arabic numeral system. Therefore, passing images belonging to class 3 and 7 in the Kannada dataset through the standard MNIST-trained CNN should result in high rates of *correct* mis-classifications as 'class- 2'. For this dataset, we obtained accuracies of 1.0 and (1.0,0.977), respectively, for the two checks above.

3 The Seed-Augment-Train/Transfer framework

In the sub-sections below, we present the Seed-Augment-Train/Transfer framework (See Fig 3).

3.1 Seeding with true-font images

There exists a vast body of hand-written, synthetic and artistically inscribed textual data freely available on the internet in the form of *font files*. In fact, we found a staggering 961 Open Font License

Figure 4: The Lohit seed dataset portraying 0-9 digits for the five Indic scripts

(OFL) fonts in the *Google fonts* repository [3] alone, each with intra-font variants such as italicized, bold etc. For the Indic scripts, Red Hat [4] released the *Lohit* [5] font family that covers 11 languages: Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Punjabi, Tamil, Telugu. In this paper, we used the Lohit True Type Font (TTF)[6] files (that can be used on both Mac and Windows platforms) to create a ***seed dataset*** for each of the languages.

This is done by combining the text insertion ability of the Python Imaging Library [6] with the textual glyphs generated using the Lohit font family. The python code is shown below:

```
import numpy as np

def num2image(i):
   image = Image.new('L', (28, 28))
   draw = ImageDraw.Draw(image)
   # Use a truetype font
   font = ImageFont.truetype("K.ttf",
   25)
   draw.text((7, 1),i, font=font,
   fill=(255))
   return np.array(image)
```

In Fig 4 we see the 5×10 array of the 28 x 28 images thus generated using the script above. We can similarly create a plethora of seed image datasets using a vast array of other fonts from sources such as Google fonts[7] and Baraha [8].

[3] https://github.com/google/fonts/tree/master/ofl
[4] https://en.wikipedia.org/wiki/Red_Hat
[5] https://en.wikipedia.org/wiki/Lohit_fonts
[6] https://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=iws-chapter08
[7] https://fonts.google.com/
[8] https://www.baraha.com/

Figure 5: The Lohit seed dataset portraying 0-9 digits for the five Indic scripts

3.2 AUGMENTATION

One can argue that the glyph in a seed image represents a puritanical version of the digit should look like. But if our goal is to perform human handwritten digit classification, we need to smartly distort these seed images in order to generate a synthetic dataset *realistic* enough to transfer well into the real-world data domain. This problem of *realistic* augmentation for digits has approaches in many ways, most notably by Simard et al [7], where they proposed the *Elastic distortion* method that they hypothesized did correspond well to *'uncontrolled oscillations of the hand muscles, dampened by inertia. '* This constitutes our Shallow augmentation approach. Once we generate the elastic-distortion augmented dataset, we also trained an AC-GAN [8] whose samples constitute the *deep augmented* synthetic training dataset.

3.2.1 SHALLOW AUGMENTATION

Classic [7] image augmentation techniques include augmenting by additive noise, intensity shifting/scaling, random rotation and/or translation, random flipping and warping. Here, as stated above we focus on the Elastic distortion method proposed in [7]. It is a bi-parametric method with the two control parameters being:

- α: The random-field parameter
- σ: The Gaussian-convolution standard deviation parameter

As recommended in [7], we set $\alpha = 8$ for each of the five Indic scripts and vary the σ parameter and tune it till the perturbed images look *realistic*. For these experiments, we used the implementation in *imgaug* [9] python library.
In Fig 5, we see a grid of random images generated using the five Indic seed datasets, with $\alpha = 8$ and σ varying from 0.01 to 3. As seen, the realistic looking image distortions mimicking the variations brought about by the human hand occur in the interval of $\sigma \in [1.5, 2.5]$. We use this heuristic to generated the Elastic distorted datasets for all the five languages by sampling from $\sigma \sim U[1.5, 2.5]$

3.2.2 DEEP AUGMENTATION - ACGAN

We complement classic image augmentation techniques using synthetic data generation, where data is generated and sampled from deep generative models such as Generative Adversarial Networks (GANs). GANs can implicitly learn the true data distribution and generate promising samples, such that the samples from the learned distribution resemble the true underlying data distribution p_{data}.

In order to generate synthetic data with corresponding labelled classes, we introduced the usage of auxiliary classifier GAN (ACGAN) where the generator input can be condition to sample from a

[9] `https://imgaug.readthedocs.io`

target class [8]. ACGAN is trained with the discriminator producing two outputs, one to discriminate between real and fake images X, and the other to classify X in terms of its class, c.

During training, we explicitly samples X_c from the generator during training for each class $c \sim p_c$, with the generator parameters adjusted to maximize the superposition of the two components in the objective function, L_s and L_c as follows. Here, L_s refers to the log-likelihood of the correct source, whereas L_c refers to the log-likelihood of the correct class.

$$L_s = \mathbb{E}\left[\log P(S = real \mid X_{real})\right] + \mathbb{E}\left[\log P(S = fake \mid X_{fake})\right] \tag{1}$$

$$L_c = \mathbb{E}\left[\log P(C = c \mid X_{real})\right] + \mathbb{E}\left[\log P(C = c \mid X_{fake})\right] \tag{2}$$

The discriminator, D, is trained to maximize $L_s + L_c$ whereas the generator, G, is trained to maximize $L_c - L_s$. In our experiments, we used learning rate l with Adam Optimizer ($l = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999$), with dropout probability $p_{dropout} = 0.3$ in the discriminator and batch normalization in the generator as regularization techniques. The generator taking in Hadamard product between the latent space and the class conditional embedding as an input.

ACGAN thus shows promising results in deep class-condition augmentation, as the discriminator now produces a probability distribution over the sources and also a probability distribution over the class labels. With the introduction of class probabilities, p_c, more structure is present in the GAN latent space which helps producing higher-quality training samples and improving stability during training. Nonetheless, we chose ACGAN as a key deep generative model in the *augment* step as it generate samples with corresponding class labels, as the class is crucial in measuring classification accuracy for the generated output with regards to the corresponding label. Fig 13 showcases the snapshots of the images produced by sampling the ACGAN generator after 10 epochs for all the five languages. For each language, we see two blocks distanced by the white separator space. The images in the left *column* are the generated images and the images in the right column are *real* shallow-synthetic images that the GAN was trained on.

4 RESULTS

Through this paper, we wanted to showcase the efficacy of the SAT framework using two approaches. The first approach is more qualitative where the focus is on training a deep-generative model (BGAN) using only the shallow synthetic data and getting it to generate realistic looking handwritten digit imagery for all the languages. The second approach is quantitative where we train a Convolutional Neural Network (CNN) purely on the synthetic training data and testing it with real-world data and also *doping* the training dataset with a small amount of real-world data ($\sim 20\%$) and computing the classification accuracies in both scenarios.

4.0.1 BGAN TRAINING AND SAMPLE GENERATION

GANs face the limitation where the generated samples have to be completely differentiable with regards to the parameters of the generator, θ, and do not work for discrete data where the gradient is zero almost everywhere, besides other issues such as training stability. Boundary-seeking GAN (BGAN) allows training of GANs with discrete data by using a policy gradient based on the KL-divergence with importance weights as a reward signal [9].

In a discrete setting, the policy gradient with importance weighting acts as a reward signal, and estimate unique rewards for each latent sample z with a low-variance gradient estimator. In our case, we focus more on the convexity of the objective function of BGAN.

We formulate BGAN in our setting as follows: we have $G_\theta : \mathcal{Z} \to \mathcal{X}$ be a generator that takes in a latent variable drawn from a prior, $z \sim p_z$. We also have a neural network $\mathcal{F}_\phi : \mathcal{X} \to \mathbb{R}$ parameterized by ϕ. Generally, we have $\hat{\theta} = \arg\min_\theta F_\phi(G_\theta(z))^2$, and we choose to use log in our BGAN objective as

$$\hat{\theta} = \arg\min_\theta \mathbb{E}_{z \sim p_z(z)} \left[\frac{1}{2}\left(\log D(x) - \log(1 - D(x))\right)^2\right] \tag{3}$$

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: The plain-vanilla CNN-MNIST architecture

Typical generator loss of continuous data is a concave function with poor convergence properties as it relies on continuous optimization of the discriminator to stabilize learning. Drawing upon the discrete approach of BGANs, we reformulate the concave optimization as a convex one by training the generator to train for the decision boundary of the discriminator, which helps improving stability. In implementation, we also use Adam Optimizer with learning rate l $(l = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999)$. The results are as shown in Fig 14 for epoch numbers 200,1000,2000 and 5000. We see the *learning* setting in as the epochs increase with realistic looking imagery emerging after as few as 5000 epochs.

4.1 Classification results

For the test-accuracy computations over the real-world dataset, we trained an off-the-shelf *plan-vanilla* CNN model[10] with cross-entropy loss and the ada-delta optimizer. Fig 7 showcases the results obtained. When we used a purely synthetic training dataset with 140000 training samples of which 80000 came from the elastic distortion procedure from section-3 and 60000 came from the ACGAN, we obtained overall accuracies varying from 60% to 76% for the five different languages (The `Purely synthetic` bars in the bar-plot). When we augmented this training dataset with merely $\sim 20\%$ of the real-world dataset (a 280-1000 train-test split), the accuracies shot up by a significant amount. This observation, we believe has two important facets to it besides the *reality-gap*: The limited size of the test-set and some idiosyncratic morphological changes in the digits' representation in the lohit-font and the colloquial representation. For example, both in Gujarati and Kannada (See Fig 11 and Fig 10 respectively), we see that six is represented rather differently in the synthetic set and in the test set. This seems to get tackled when you introduce small amount of real-world data into the training data. In the case of Gujarati, the accuracy for digit 6 that we just discussed increases from 0.7% to 95%! As seen in Fig 7, the overall test accuracy increases for every

[10] `https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py`

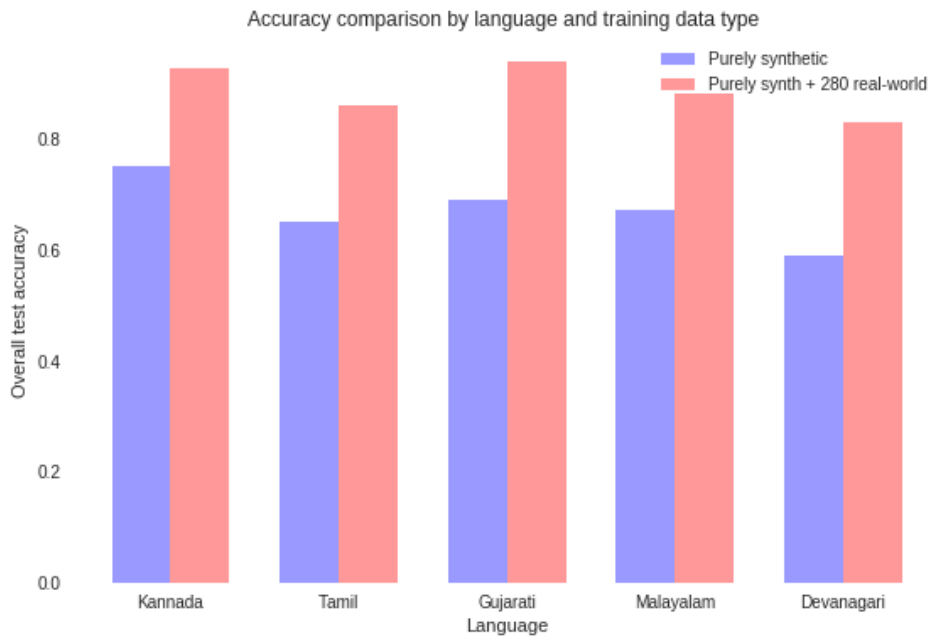

Figure 7: The accuracy comparisons for the five scripts

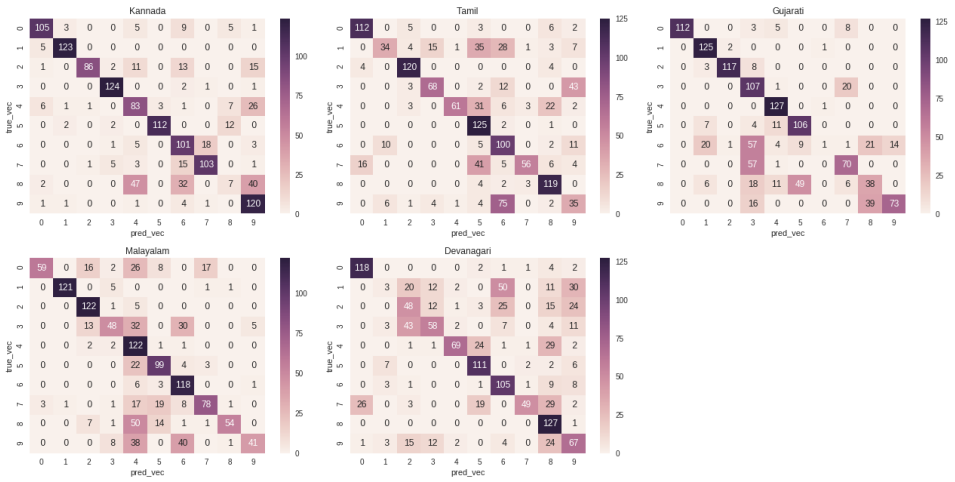

Figure 8: Class-wise confusion matrix for all the five Indic scripts obtained after training on a purely synthetic *hybrid* dataset

language by a substantial margin for the 'Purely synth + 280 real-world training' dataset option. The confusion matrices for the two training scenarios are as shown in Fig 9 and Fig 12 respectively.

5 Conclusion and Future Work

In this paper, we introduced 5 new real-world digit datasets in Indic languages and also a transfer learning framework, which we term as Seed-Augment-Train/Transfer (SAT), to perform real-world handwritten digit classification in these languages. Our goal is to also draw attention of the machine learning community to this potentially attractive nexus between the work happening in the synthetic-to-real transfer learning domain and a veritably rich reservoir of semi-synthetic textual glyphs training data freely available in the form of *Font files*. We showcased the efficacy of this

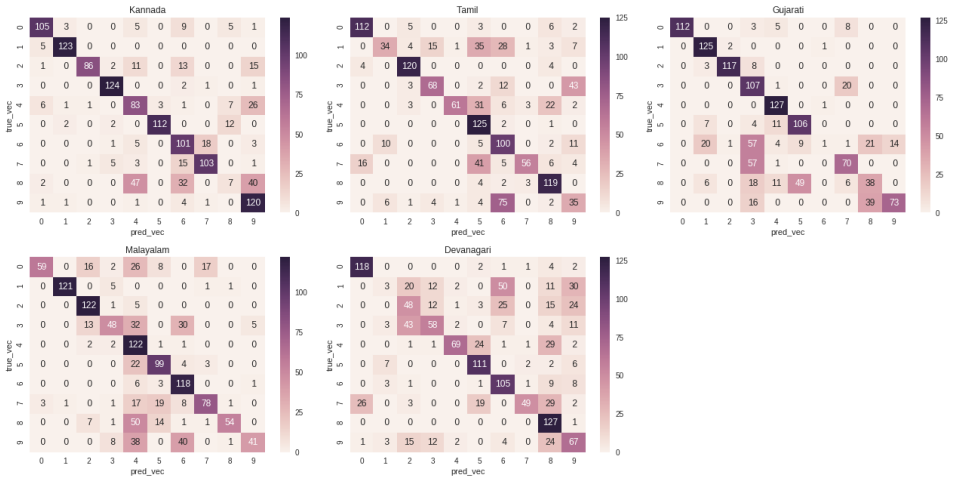

Figure 9: Class-wise confusion matrix for all the five Indic scripts obtained after training on a purely synthetic *hybrid* dataset

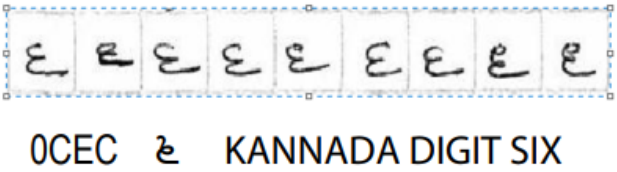

Figure 10: Changes between the Lohit-font representation and the vernacular written representation of the digit 6 in Kannada

SAT-based approach by perform digit-classification and also training GANs for the digits of the languages considered. We are expanding this foundational effort in several directions. The first entails using a small portion of the dataset for transfer learning, which will help tackle cases in which off-the-shelf image augmenters do not effectively capture human handwriting's natural variations. The second involves generative modeling of the ***deviations*** in handwritten digits with regards to the clean seed-synthetic dataset using the extended MNIST dataset (in lieu of the digits themselves). We would then apply this approach to *deep augment* seed datasets across other languages as well, resulting in a universal-MNIST dataset recipe. Thirdly, we are collecting larger volumes of real world handwritten test data for these languages to form MNIST sized datasets.

REFERENCES

[1] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2107–2116, 2017.

[2] E. Santana and G. Hotz, "Learning a driving simulator," *arXiv preprint arXiv:1608.01230*, 2016.

[3] T. Inoue, S. Choudhury, G. De Magistris, and S. Dasgupta, "Transfer learning from synthetic to real images using variational autoencoders for precise position detection," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 2725–2729, IEEE, 2018.

[4] T. Inoue, S. Chaudhury, G. D. Magistris, and S. Dasgupta, "Transfer learning from synthetic to real images using variational autoencoders for precise position detection," *CoRR*, vol. abs/1807.01990, 2018.

[5] S. Gāguli, "The indian origin of the modern place-value arithmetical notation," *The American Mathematical Monthly*, vol. 39, no. 5, pp. 251–256, 1932.

[6] `https://pillow.readthedocs.io/en/stable/`.

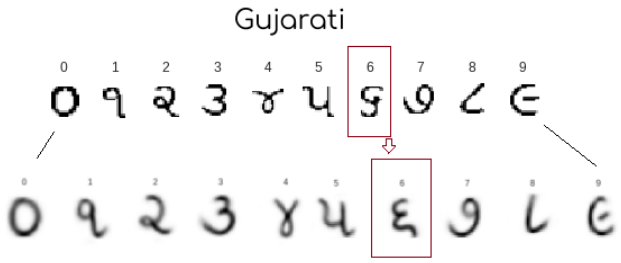

Figure 11: Changes between the Lohit-font representation and the vernacular written representation of the digit 6 in Gujarati

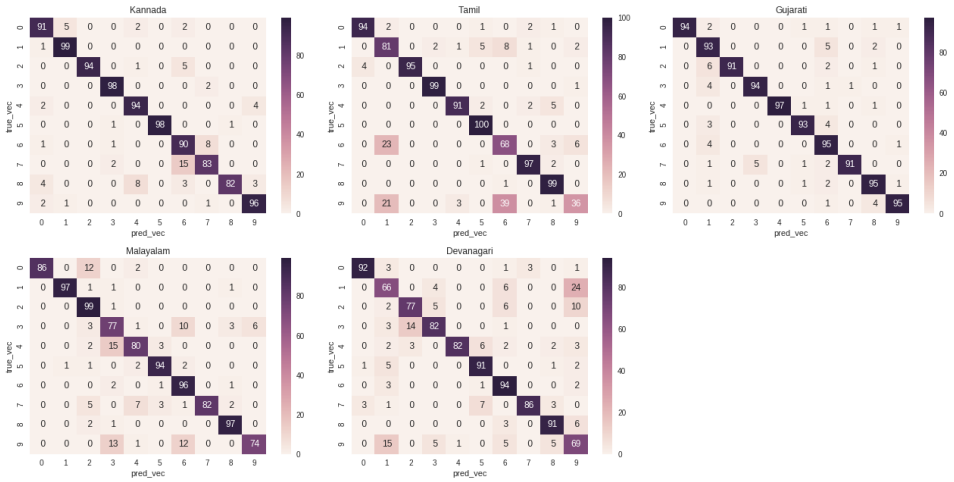

Figure 12: Class-wise confusion matrix for all the five Indic scripts obtained after using 280 training samples from the real-world datasets

[7] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[8] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2642–2651, JMLR. org, 2017.

[9] R. D. Hjelm, A. P. Jacob, T. Che, A. Trischler, K. Cho, and Y. Bengio, "Boundary-seeking generative adversarial networks," *arXiv preprint arXiv:1702.08431*, 2017.

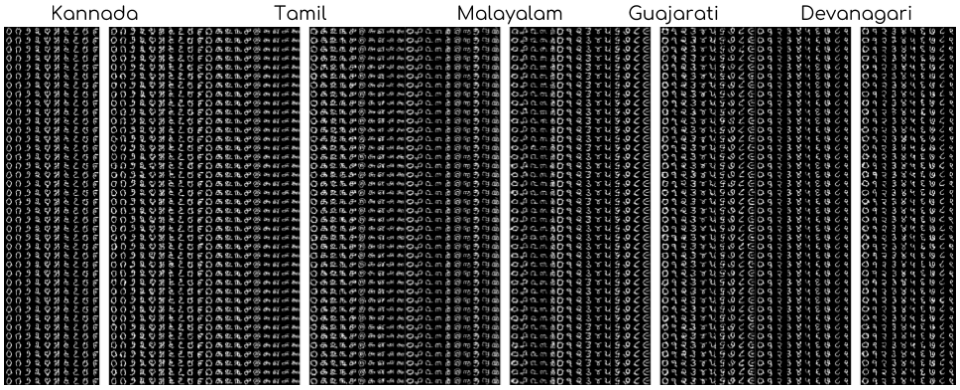

Figure 13: Snapshots of the images produced by sampling the ACGAN generator after 10 epochs for all the five languages.

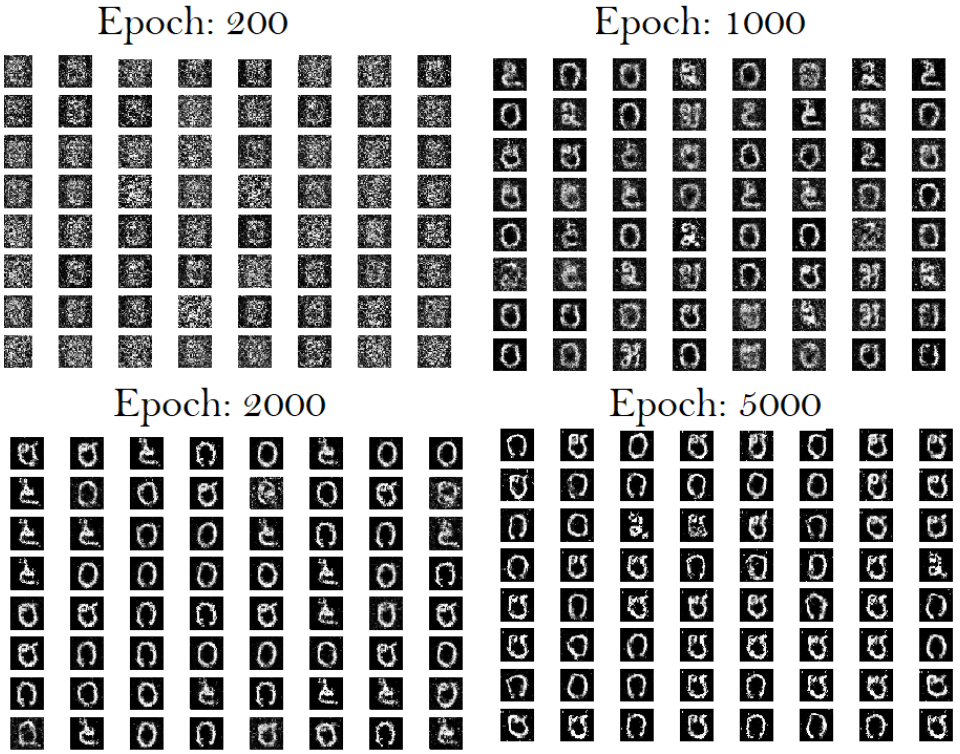

Figure 14: Snapshots of the images produced by sampling the BGAN generator after varying number of epochs