TextCaps : Handwritten Character Recognition with Very Small Datasets

Vinoj Jayasundara
University of Moratuwa
vinojjayasundara@gmail.com

Sandaru Jayasekara
University of Moratuwa
sandaruamashan@gmail.com

Hirunima Jayasekara
University of Moratuwa
nhirunima@gmail.com

Jathushan Rajasegaran
University of Moratuwa
brjathu@gmail.com

Suranga Seneviratne
University of Sydney
suranga.seneviratne@sydney.edu.au

Ranga Rodrigo
University of Moratuwa
ranga@uom.lk

Abstract

Many localized languages struggle to reap the benefits of recent advancements in character recognition systems due to the lack of substantial amount of labeled training data. This is due to the difficulty in generating large amounts of labeled data for such languages and inability of deep learning techniques to properly learn from small number of training samples. We solve this problem by introducing a technique of generating new training samples from the existing samples, with realistic augmentations which reflect actual variations that are present in human hand writing, by adding random controlled noise to their corresponding instantiation parameters. Our results with a mere 200 training samples per class surpass existing character recognition results in the EMNIST-letter dataset while achieving the existing results in the three datasets: EMNIST-balanced, EMNIST-digits, and MNIST. We also develop a strategy to effectively use a combination of loss functions to improve reconstructions. Our system is useful in character recognition for localized languages that lack much labeled training data and even in other related more general contexts such as object recognition. [1]

1. Introduction

Handwritten character recognition is a nearly solved problem for many of the mainstream languages thanks to the recent advancements in deep learning models [1]. Nonetheless, for many other languages, handwritten digit recognition remains a challenging problem due to the lack of sufficiently large labeled datasets that are essential to train deep learning models [2]. While conventional models such as linear classifiers, K-nearest neighbors, non-linear classifiers, and Support Vector Machines (SVM) [3] can be used for this task, they are not able to achieve the near human level performances provided by deep learning models. Convolutional Neural Networks (CNN) have achieved state-of-the-art results due to their ability to encode deep features and spatial understanding. Although CNNs are good at understanding low level and high level features in images, by doing so, they lose valuable information at pooling layers. CNNs require large number of training samples (usually in the scale of thousands or tens of thousands per class) to train and classify images successfully. As a result, there is a strong interest in training CNNs with a lesser number of training samples.

In this paper, we propose a technique which tackles this problem of the labeled dataset being small in size, with the aid of Capsule Networks (CapsNets) [4]. We exploit their ability to augment data just by manipulating the instantiation parameters [5]. CapsNets learn the properties of an image—in this case a character—in addition to its existence. This makes them useful in learning to recognize characters with a less amount of labeled data. Our architecture is based on the CapsNet architecture proposed by Sabour *et al.* [4], which comprises a capsule network and a fully connected decoder network. We replace the decoder network with a deconvolutional network while doing minor alterations to the capsule network. By adding a controlled amount of noise to the instantiation parameters that represent the properties of an entity, we transform the entity to characterize actual variations that happen in reality. This results in a novel data generation technique, much more realistic than augmenting data with affine transformations. As the reconstruction accuracy is also important in many contexts, we present an empirically appropriate strategy of combining loss functions which significantly improves the reconstruction. Our system achieves results that are on-par with the state-of-the-art with just 200 data points per class, while achieving even better results with larger volumes of training data.

[1] https://github.com/vinojjayasundara/textcaps

The key contributions of this paper are as follows:

- We outperform the state-of-the-art results in EMNIST-letters, EMNIST-balanced and EMNIST-digits character datasets, by training our system on all the training samples available.
- We evaluate the proposed architecture on a non-character dataset, Fashion-MNIST, to ensure the flexibility and robustness. We achieve very good results with 200 training samples and achieve the state-of-the-art with the full dataset.
- We propose a novel technique for training capsule networks with small number of training samples, as small as 200 per class, and keeping the same set of test samples, while achieving the state-of-the-art performance. Our method require only 10% of data necessary for a state-of-the-art system, to produce similar results.
- We propose and evaluate several variations to the decoder network and analyze its performance with different loss functions to provide a strategy to select a suitable combination of loss functions.

Rest of the paper is organized as follows: in Section 2 we discuss the related works, and in Section 3 we explain our methodology. Subsequently, we discuss our results in Section 4.

2. Related Work

MNIST [6] is the widely used benchmark for the handwritten digit recognition task. Multiple works [7, 8, 9, 10, 11] have used CNN models on MNIST dataset and have achieved results in excess of 99% accuracy. Apart from digit recognition, several attempts [12, 13] have been made in handwritten character recognition with EMNIST datasets [12]. A bidirectional neural network is introduced in [14] which is capable of performing both image classification and image reconstruction by adding a style memory to the output layer of the network.

The idea of a capsule was introduced in 2011 by [5], as a transforming autoencoder. With a three layered CapsNets architecture, and by training the network using dynamic routing, authors of [4] have achieved 0.25% error rate on the MNIST dataset. This architecture consists of a primary capsule, which was built by stacking convolutional layers, and a fully connected capsule, which uses routing by agreement between higher level capsules and lower level capsules. We draw intuition for this paper from the concept of instantiation parameters proposed in [5], while emphasizing that our work is significantly novel and different from [5].

We identified two main solutions in the literature to the low data issue, namely one-shot learning and new data generation. An example of the former is the Siamese networks as proposed in [15]. A one-shot learning deep model was proposed by Bertinetto *et al.*[16], where they used a second network to predict the parameters of the first network. Since one-shot learning solutions are mostly application-specific, we turn to a new data generation approach.

Existing literature offers several successful data generation techniques. Although GANs [17] can be used to generate data, a basic form of a GAN network will not be sufficient to augment the dataset for training, since it can not generate labelled data, unless a separate GAN is trained per class. Another potential choice which has image generation capabilities, Variational Autoencoders (VAEs) [18] have similar problems. VAEs represent all the images as 1D vectors, whereas capsule networks have dedicated dimensions for each class. As a result, when VAE's 1D vectors are perturbed, there is a high probability that those changes affect multiple classes. Data augmentation techniques such as jittering and flipping (not suitable for characters) offer limited amount of simple augmentation. Thus, they can not offer complex and subtle variations that are more closer to the human variations. A comprehensive comparison of our results with these techniques are provided in Section 4.2.

3. Methodology

This sections outlines our approach. Prior to experimenting with reduced datasets, in Section 3.1, we attempt to surpass the state-of-the-art results for several hand written digit databases including EMNIST balanced, EMNIST letters and EMNIST digits, with the use of all the training samples provided. Subsequently, we attempt to achieve the state-of-the-art performance using a limited number of training samples, as low as 200 training samples per class, as opposed to, for example, 2400 data points per class in EMNIST balanced and 4800 data points per class in EMNIST letters.

In order to address the drawbacks faced when training the classifier with low number of training samples, we propose a novel technique for increasing the number of training samples in Section 3.2. We perform a comprehensive analysis of the effect of the loss function in reconstruction, in Section 3.3.

3.1. Character Recognition with Capsule Networks

For the character recognition task, we propose an architecture comprising of a capsule network and a decoder network, as illustrated in Fig. 1 and Fig. 2.

In the capsule network, the first three layers are convolutional layers with 64 3×3 kernels with stride 1, 128 3×3 kernels with stride 1 and 256 3×3 kernels with stride 2 respectively. The fourth layer is a primary capsule layer with 32 channels of 8-dimensional capsules, with each primary capsule containing 8 convolutional units with a 9×9 kernel and a stride of 2. The fifth layer, termed as the character capsule layer, is a fully connected capsule layer with a 16-dimensional capsule per class, resulting in M capsules for a dataset with M number of classes. We use dynamic routing

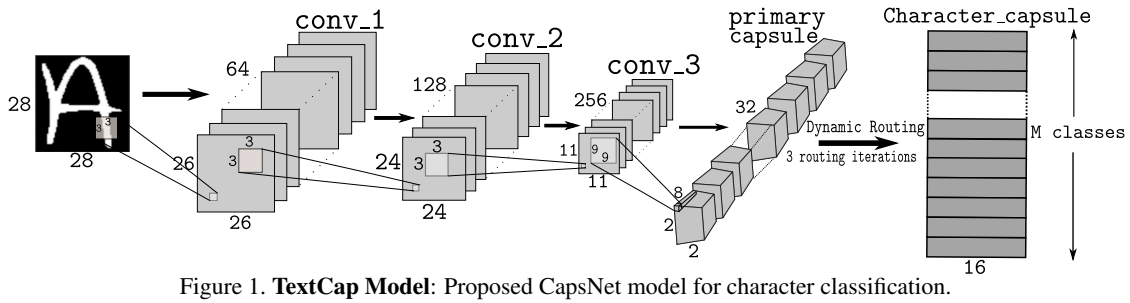


Figure 1. **TextCap Model**: Proposed CapsNet model for character classification.

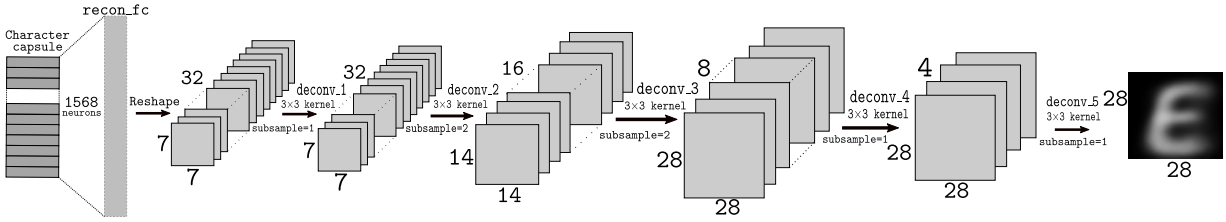


Figure 2. **TextCap Decoder**: Decoder network for the character reconstruction. Input to this network is obtained by masking the DigitCaps layer of the TextCap classifier

between the primary capsule layer and the character capsule layer, as proposed by [4], with 3 routing iterations. The input to the capsule network is a set of J, 28×28 images and the output is a $J \times M \times 16$ dimensional tensor C, containing the corresponding instantiation parameters, where each $C_j, j \in [J]$ is the instantiation parameter matrix of the j^{th} training sample.

Prior to passing C as the input to the decoder network, the corresponding instantiation parameters should be masked with zeros for all the classes except the true class. Hence, the masked tensor $\widehat{C}$ is still a $J \times M \times 16$ dimensional matrix, yet containing only the instantiation parameters corresponding to the true class as the non-zero values.

The decoder network comprises one fully connected layer, followed by five deconvolutional layers [19] with parameters as shown by Fig. 2. The input to the decoder is the masked matrix $\widehat{C}$, and the output of the decoder is the set of reconstructed 28×28 images. Except for the final deconvolution layer, which has sigmoid activation, the fully connected layer and the other deconvolution layers have ReLU activation.

First, we train the proposed model with the full training sets and evaluate its performance. Subsequently, in an effort to address the issue of lack of high number of training samples in character recognition and similar tasks as elaborated in section 1, we attempt to achieve the state-of-the-art performance using 200 training samples per class, using the same network.

By examining the results of the above section with low number of training samples, as elaborated in Section 4.1, we observed that even though the capsule network performance achieved the state-of-the-art, the decoder network fails to achieve acceptable reconstruction. The most obvious solution to enhancing the performance of the decoder network is to increase the number of training samples, by generating new training samples from the samples available in the original (reduced) training set. Therefore, we propose a novel method of generating new training samples by augmenting original training samples with the aid of the concept of instantiation parameters in the CapsNets, as described by the following Section 3.2

3.2. Proposed Technique for Image Data Generation Using Perturbation of Instantiation Parameters

From the concept of instantiation parameters in capsule networks, we can represent any character using a 16 dimensional vector [4]. With a pre-trained decoder network, we can successfully reconstruct the original image, by using only this instantiation parameter vector. The intuition behind our proposed perturbation algorithm is that by adding controlled random noise to the values of the instantiation vector, we can create new images, which are significantly different from the original images, effectively increasing the size of the training dataset. Fig. 3 illustrates the variations of an image, when one particular instantiation parameter is changed thusly.

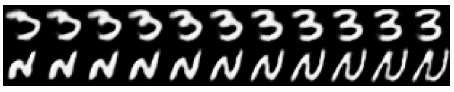
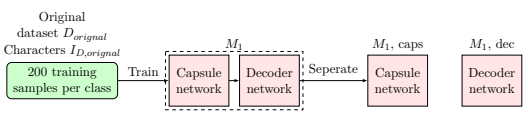


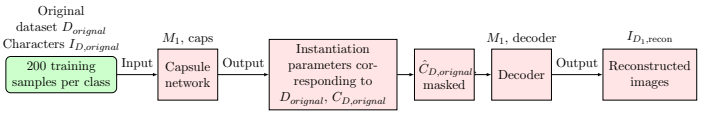
Figure 3. Variation in characters with the perturbation of instantiation parameters

Similarly, each of the instantiation parameter is responsible for a certain property of the image, individually or collectively. Hence, we propose a novel technique of generating a new dataset, from a dataset with limited amount of training samples, as illustrated by Fig. 4 and Algorithm 1.

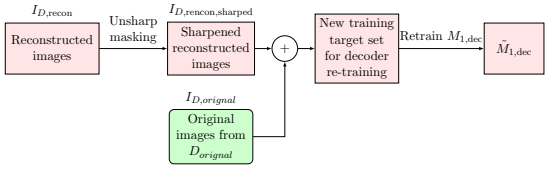
First, as illustrated by Fig. 4(a), we train the network



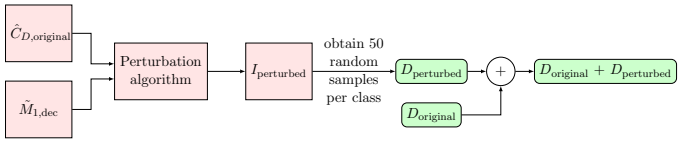
(a) Initial Training of M_1



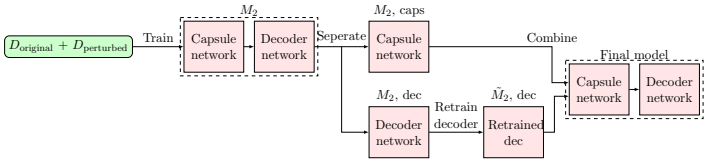
(b) Generating instantiation parameters and reconstructed images



(c) Proposed decoder re-training technique



(d) New image data generation by the proposed technique



(e) Training model M_2 afresh with the new dataset

Figure 4. The overall methodology for improving the decoder performance

proposed in Section 3.1, M_1, with dataset, D_{original}, containing 200 training samples per class and all testing samples. Without loss of generality, we choose the first 200 training samples in each class in the dataset. Subsequently, we consider the trained capsule network, $M_{1,\text{caps}}$, and the trained decoder network, $M_{1,dec}$, separately. Next, as illustrates by Fig. 4(b), we obtain the instantiation parameters, $C_{D,\text{original}}$, corresponding to the training images, $I_{D,\text{original}}$, in D_{original} as the output of the capsule network $M_{1,\text{caps}}$, which can be masked as $\widehat{C}_{D,\text{original}}$ and passed as the input of the decoder network $M_{1,\text{dec}}$. We can obtain the corresponding reconstructed images $I_{D,\text{recon}}$ as the output of $M_{1,\text{dec}}$. Fig. 5 shows several such reconstructed images.

Figure 5. Original and reconstructed Image pairs with Model M_1

From Fig. 5, we clearly observe that training with such low number of training samples result in poor reconstruction performance. The subtle variations in the input characters are absent from the reconstructions, in addition to being blurred. Hence, we cannot directly apply the concept of perturbation and create new training samples from a such poorly trained model. First, we attempt to eliminate the blurriness in the decoder network output, by proposing the following technique, illustrated by Fig. 4(c). For each reconstructed image in $I_{D,\text{recon}}$, we perform unsharp masking [20] with $radius = 1$, $threshold = 1$ and $unsharp\ strength = 10\ times$, which sharpens the reconstructed images. Then we combine the new sharpened image set $I_{D,\text{recon,sharped}}$ with the initial $I_{D,\text{recon}}$ set, in order to create a new target set for the decoder $M_{1,\text{dec}}$. Subsequently, we re-train the decoder for 10 epochs with this new target set, in order to obtain an improved decoder $\widetilde{M}_{1,\text{dec}}$ which provides sharper reconstructions than $M_{1,\text{dec}}$.

After training, there can be training samples which are not properly learned, and hence are wrongly reconstructed. If these wrongly reconstructed samples are considered for perturbing and creating new samples, it may result in missclassified samples in the newly generated training dataset. Therefore, prior to applying perturbation and creating new training samples, it is necessary to remove such training samples, after the model is trained.

Subsequently, we perform new data generation by perturbation, as illustrated by Fig. 4(d). For non-zero instantiation parameters in $\widehat{C}_{D,\text{original}}$, we add random controlled noise (Algorithm 1). Here, we perturb only one instantiation parameter at a time to generate new samples to avoid distortions. Hence, a method of selection of which instantiation parameter to perturb is required. We observed that, for a given class, there exists a relationship between the variance of an instantiation parameter and the actual physical variations in the generated images. Higher the variance, we observe rapid variations and vice versa. Hence, for each instantiation parameter $k \in [0, 15]$ in each class $m \in M$, we calculate the variance, $\sigma_{m,k}$, across all the training samples that belongs to m, and sort in the descending order. We have 16 choices for the value of a, with $a = 0$ representing the instantiation parameter with the highest variance and $a = 15$ representing that with lowest variance. For this study, we generate two datasets with $a = 0$ and $a = 1$.

For a given a, we calculate the noise value to add for each instantiation parameter considered. The adjusted value of the instantiation parameter should not exceeded the max-

imum value that it can take for a given class. Adding noise to instantiation parameters without any restrictions will lead to various distortions in the reconstructed images, as illustrated in Fig. 6 below. The **H** has visually changed classes to **A**, and **a** is visually unrecognizable anymore. Such distortions are detrimental for a data generation technique. Therefore, we propose a carefully designed control mechanism to avoid such distortions, without manual elimination by visual inspection.

Figure 6. Distortions caused by adding uncontrolled noise

Hence, for each instantiation parameter k of the class m, we calculate the maximum noise that can be added, $\tau_{m,k}$. We further constrain the noise by τ_k, which is the average maximum noise that can be added for the instantiation parameter k across all the classes, even though $\tau_{m,k}$ allows for higher values. This is to prevent sudden high variations occurring after perturbations. Hence, the noise value added for any k is capped at τ_k. Subsequently, we obtain the new reconstructed images, $I_{D,\text{perturbed}}$, which are significantly different from the original training images, $I_{D,\text{original}}$, by passing the perturbed instantiation parameter tensor to the decoder $\widetilde{M}_{1,\text{dec}}$.

Algorithm 1 Image data generation using perturbation

Input: Instantiation parameters $\widehat{C}$, a^{th} highest variance, Decoder Network model ($\widetilde{M}_{dec}$).
Output: Perturbed images $I_{\text{perturbed}}$

Calculate class variance $\sigma_{m,k} = \text{var}_j(\widehat{C}_{m,j,k})$.
Get $\tilde{\sigma}_{m,k'} \leftarrow sort_k(\sigma_{m,k})$ descending.
Get $\hat{k} = k$ corresponding to $k' = a$.
$\tau_{m,k} \leftarrow \frac{\max_j(\widehat{C}_{m,j,k}) - \min_j(\widehat{C}_{m,j,k})}{2}$
get $\tau_k \leftarrow \text{avg}_i(\tau_{m,k})$
for each $\hat{j} \in [j]$ **do**
    if $\widehat{C}_{m,\hat{j},\hat{k}} > 0$ **then**
        $\widehat{C}_{m,\hat{j},\hat{k}} \leftarrow \widehat{C}_{m,\hat{j},\hat{k}} + \min(\tau_{m,\hat{k}}, \tau_{\hat{k}})$
    else
        $\widehat{C}_{m,\hat{j},\hat{k}} \leftarrow \widehat{C}_{m,\hat{j},\hat{k}} - \min(\tau_{m,\hat{k}}, \tau_{\hat{k}})$
$I_{\text{perturbed}} \leftarrow \widetilde{M}_{\text{dec}}(\widehat{C})$

Finally, we have two new sets of training samples generated with $a = 0$ and $a = 1$. We combine these two sets and obtain 50 random samples per class (user's discretion), to formulate the new dataset $D_{\text{perturbed}}$. We combine $D_{\text{perturbed}}$ and D_{original}, which will effectively increase the number of training samples, solving our target problem. Subsequently, as illustrated by Fig. 4(e), we train a new model, M_2 with the new $D_{\text{original}} + D_{\text{perturbed}}$ dataset, re-train the decoder with the proposed re-training technique and obtain the final model for character classification.

3.3. Various Reconstruction Loss Functions

We investigate the effect on reconstruction based on the loss function used for reconstruction in a capsule network, in order to identify a well-suited reconstruction loss function for the TextCaps model. First, we study the variations on reconstruction with different loss functions for different number of training samples on the EMNIST-Balanced dataset, and then we extend our analysis to various combinations of loss functions. In this analysis, both spatial and structural similarity measures are used as reconstruction loss functions.

Since different loss functions we use produce outputs in different scales, it is not possible compare the losses directly. Therefore, we use Peak Signal-to-Noise Ratio (PSNR) given by (1), as an independent (of reconstruction loss functions) measure, to determine the quality of reconstructed images.

$$PSNR = 10 \log_{10} \left(\frac{MAX_i^2}{MSE} \right) \tag{1}$$

where MAX_i is the maximum possible pixel value of the image (1 in our case) and MSE is the Mean Squared Error between the test and reconstructed images.

Let $x(p)$ be the intensity of the p^{th} reconstructed pixel and $y(p)$ be the intensity of the p^{th} true input pixel and N be the total number of pixels.

3.3.1 MSE

Following Sabour *et al.* [4], we use MSE, as the loss function for reconstruction, defined by,

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (x(p) - y(p))^2 \tag{2}$$

3.3.2 L_1 Norm

To remove artifacts introduced by MSE, we consider L_1 norm as a loss function which is defined by,

$$L_1 = \frac{1}{N} \sum_{i=1}^{N} |x(p) - y(p)| \tag{3}$$

3.3.3 SSIM

L_1 and MSE do not capture the spatial relationship between pixels. We use $SSIM$ proposed in [21] to capture spatial relationship between the input image and reconstructed image. $SSIM$ for x, y and the loss function for $SSIM$, structural dissimilarity ($DSSIM$), are defined by,

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)}{(\mu_x^2 + \mu_y^2 + C_1)} \cdot \frac{(2\sigma_{xy} + C_2)}{(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4}$$

$$DSSIM = \frac{1}{N} \sum_{i=1}^{n} 1 - SSIM(p) \tag{5}$$

where μ_x, μ_y and σ_x^2, σ_y^2 are the means and variances and σ_{xy}^2 is the covariance of reconstructed and true input pixel intensities. $C_1 = (K_1 L)^2$ and $C_2 = (K_2 L)^2$ where L is the dynamic range of the pixel values (typically, $2^m - 1$, where m is the number of bits per pixel) and K_1, K_2 are small constants ($K_1 = 0.01$ and $K_2 = 0.03$).

3.3.4 Binary Cross-Entropy (BCE)

BCE is often used as a measure to identify the difference between two distributions, which is defined by,

$$BCE = -\frac{1}{N}\sum_{i=1}^{n}[y(p)\log(x(p))+(1-y(p))\log(1-x(p))] \quad (6)$$

3.3.5 Combinations of Loss Functions

To combine two loss functions, rather than linearly combining two loss equations mathematically, we propose a method which combines two reconstructed images together. We slightly modify the CapsNet decoder network using two decoders, one for each loss function and generate two separate reconstruction outputs. Then we compare the absolute values of the difference of each pixel value between the two reconstructed outputs and the test images independently, and assign the pixel value which is closer to the test image to the final reconstructed output. Different loss function combinations we use here for two decoders are L_1 & $DSSIM$, L_1 & BCE, MSE & $DSSIM$, MSE & BCE and BCE & $DSSIM$.

4. Experiments and Results

For each dataset in Table 1, we train TextCaps on 200 training samples from the training set and test using the whole test set. In order to test the performance of the TextCaps architecture, we also evaluate it by training it on full training sets and testing on full test sets.

Table 1. Five datasets used to evaluate TextCaps

Dataset	Classes	Train samp/class	Train size	Test size
EMNIST-Balanced[12]	47	2,400	112,800	18,800
EMNIST-Letters[12]	26	4,800	124,800	20,800
EMNIST-Digits[12]	10	24,000	240,000	40,000
MNIST[6]	10	6,000	60,000	10,000
Fashion MNIST[22]	10	6,000	60,000	10,000

4.1. Handwritten Character Classification

Table 2 compares our results to the state-of-the-art. We include the results that we obtained with the full training sets, as well as using only 200 training samples per class. In both instances, we have used the full testing sets shown in Table 1, to report the average accuracies. We use a combination of marginal loss and the reconstruction loss for training as proposed in [4], and further, the training procedure followed for every experiment in this paper is similar to [4]. For each dataset, we use ensembling to improve our model accuracy, and to avoid over fitting. We use cyclic learning rates for each 30 epochs, giving us 3 ensemble models with 90 epochs [23].

First, we describe the results we obtained with full training sets and compare with the state-of-the-art. On EMNIST-letters, we significantly outperform the state-of-the-art Wiyatno *et al.* [14] by 4.09%. An average accuracy of 90.46% was achieved by our system for the EMNIST-balanced dataset, which outperforms the state-of-the-art Dufourq *et al.* [13] by 2.16%. For EMNIST-digits dataset, TextCaps was able to surpass the state-of-the-art achieved by Dufourq *et al.* [13] by 0.49%. For MNIST and Fashion-MNIST, our system produced sub-state-of-the-art accuracy. Yet, our results are on-par.

Table 2. Comparison of TextCaps with state-of-the-art results, the mean and the standard deviation from 3 trials are shown

EMNIST-Letters		
Implementation	With full train set	With 200 samp/class
Cohen *et al.* [12]	85.15%	-
Wiyatno*et al.*[14]	91.27%	-
TextCaps	**95.36 ± 0.30%**	**92.79 ± 0.30%**
EMNIST-Balanced		
Implementation	With full train set	With 200 samp/class
Cohen *et al.* [12]	78.02%	-
Dufourq *et al.* [13]	88.3%	-
TextCaps	**90.46 ± 0.22%**	87.82 ± 0.25%
EMNIST-Digits		
Implementation	With full train set	With 200 samp/class
Cohen *et al.* [12]	95.90%	-
Dufourq *et al.* [13]	99.3%	-
TextCaps	**99.79 ± 0.11%**	98.96 ± 0.22%
MNIST		
Implementation	With full train set	With 200 samp/class
Sabour *et al.* [4]	99.75%	-
Cireşan *et al.* [8]	99.77%	-
Wan *et al.* [24]	**99.79%**	-
TextCaps	99.71 ± 0.18%	98.68 ± 0.30%
Fashion MNIST		
Implementation	With full train set	With 200 samp/class
Xiao *et al.* [22]	89.7%	-
Bhatnagar *et al.* [25]	92.54%	-
Zhong *et al.* [26]	**96.35%**	-
TextCaps	93.71 ± 0.64%	85.36 ± 0.79%

Subsequently, we describe and compare the results we obtained with only 200 training samples per class. On EMNIST-letters, we exceed the state-of-the-art results by 1.52%. However for EMNIST-balanced, EMNIST-digits, MNIST we were able to achieve the state-of-the-art results. Even though our system did not surpass the state-of-the-art performance, we highlight that we were able to achieve a near state-of-the-art performance using only 8-10% of the training data.

4.2. Results of the Proposed Image Data Generation Technique

In this section, we present the results of the decoder retraining technique and the new image data generation by perturbation technique proposed in Section 3.2. We evaluate the success as well as the limitations of the two techniques by referring to these results.

Fig.7 (a) illustrates several sample images from the

EMNSIT-balanced test set and (b) illustrates the corresponding reconstructed images by $M_{1,dec}$, which was trained using 200 training samples per class. Fig. 7 (c) illustrates the corresponding reconstructions by $\widetilde{M}_{1,dec}$, which was obtained by re-training $M_{1,dec}$ using the proposed technique. It is evident that reconstructions by $\widetilde{M}_{1,dec}$ are much sharper than those by $M_{1,dec}$. Therefore, our proposed decoder re-training technique is highly successful in sharpening the character reconstruction.

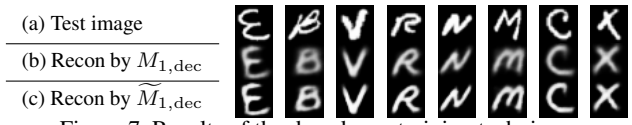


Figure 7. Results of the decoder re-training technique

Even though the proposed decoder re-training technique is highly succesful in sharpening the reconstruction, it still does not capture the subtle variations of the input images expected from a succesful reconstruction. Hence, we perform the new image data generation by perturbation technique on the images reconstructed by $\widetilde{M}_{1,\mathrm{dec}}$. Fig. 8 illustrates our results. Fig. 8 (a) and (b) are identical to that of Fig. 7 (a) and (b), which we include for comparison. Fig. 8 (c) illustrates the corresponding results obtained by $\widetilde{M}_{2,\mathrm{dec}}$, which was trained afresh using the new dataset generated by the proposed technique. It is evident that reconstructions by $\widetilde{M}_{2,\mathrm{dec}}$ now captures the subtle variations that we required —curvature of E, slanting of B, asymmetry of X—, and are much closer to the test image than the reconstruction by $M_{1,\mathrm{dec}}$. We observed similar improvements in approximately 80% of the test images, rendering the proposed technique significantly successful in improving the decoder performance with small number of training samples.

Figure 8. Results of the model M_2, trained with the newly generated dataset

Fig. 9 illustrates several instances where the proposed method still failed to capture the required subtle variations. Yet, even in these instances, the reconstructions of $\widetilde{M}_{2,\mathrm{dec}}$ demonstrates significant improvement over the reconstructions of $M_{1,\mathrm{dec}}$.

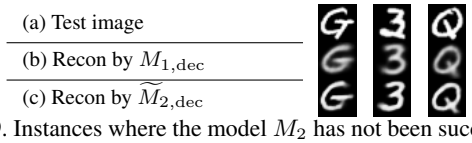


Figure 9. Instances where the model M_2 has not been succesful in capturing the subtle variations of the test image —vertical line of G, bottom part of 3 —

GANs, VAEs and data augmentation techniques are alternatives to our proposed technique. Fig. 10 below, illustrates the generated images from a Conditional GAN (CGAN). However, at the generation phase, the new images are generated from random noise and that does not allow to apply specific perturbations, producing less realistic variations in images in comparison to what we do in the proposed method. Similarly, the reconstructions obtained when using our proposed technique are far better than it's alternatives in the low data regime, as shown by Fig. 11, which contains the reconstructions obtained after training with the respective data augmentation technique. The alternatives produced little or reduced improvement, whereas our method produced visually significant ($\approx$ 1dB PSNR) improvement.


Figure 10. Generated images from CGAN for label 2

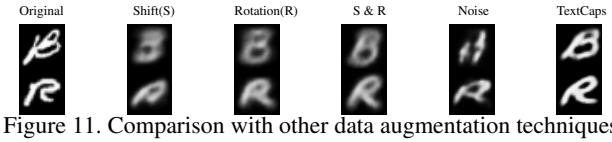


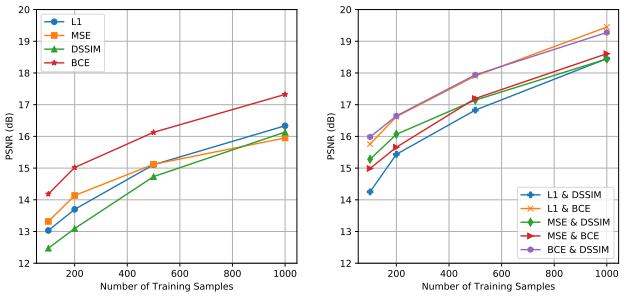
Figure 11. Comparison with other data augmentation techniques

4.3. Results of the Reconstruction Loss Functions and Analysis

Next, we discuss the variations on reconstruction with respect to different loss functions and combinations of loss functions. All the modifications we consider here were tested with varying number of training samples (100, 200, 500 and 1000) per class, from the *EMNIST-Balanced* dataset. We used the CapsNet model proposed in [4] with minor alterations for this analysis, where decoder network consists of fully connected layers, since that architecture is well established.

Fig. 12(a) illustrates the variation of PSNR with the amount of training samples used for different loss functions, which leads to a number of interesting observations. For example, when the number of training samples are small (100 or 200), performance of L_1 is poor compared to MSE, yet for higher number of training samples, L_1 performs better. PSNR values for BCE are the highest regardless of the number of training samples. Hence, we conclude that the most suitable loss function for reconstruction loss in general is BCE. Fig. 13 illustrates the variations in reconstructed images with the use of different loss functions for 200 training samples.

We observed that if we linearly combine two loss functions to design a modified loss function, the obtained reconstructions were relatively poor. Nonetheless, since different loss functions capture different properties of handwritten characters, we used two decoders with two loss functions and combined the two reconstructed images together to get a resultant reconstructed image with improved quality. With this modification, it was interestingly observed



(a) Change in PSNR for different reconstruction loss functions.

(b) Change in PSNR for different combinations of loss functions.

Figure 12. Change in PSNR with the number of training samples.

that individual reconstructions for respective loss functions also improved, due to the effect of the reconstruction loss component in the training loss. However, the amount of improvement of individual PSNRs depends on the loss function combination. Table 3 demonstrates how the individual PSNR values improve with the two-decoder networks for different combinations of reconstruction loss functions.

Table 3. PSNR values for individual reconstructions when different combinations of loss functions are used. Here, we use the two-decoder network model with one loss function per each decoder. For each loss function combination, the PSNR value in the first row of PSNR pairs corresponds to the first reconstruction loss function (used in the first decoder) whereas the second row corresponds to the second loss function (used in the second decoder).

Loss function combination	Number of training samples			
	100	200	500	1000
L_1 & $DSSIM$	13.51 12.89	14.64 14.19	15.95 15.57	17.48 17.03
L_1 & BCE	14.33 14.57	15.26 15.44	16.60 16.71	18.10 18.13
MSE & $DSSIM$	13.87 12.95	14.81 14.06	16.00 15.47	17.29 16.79
MSE & BCE	14.58 14.59	15.19 15.20	16.55 16.56	17.76 17.78
BCE & $DSSIM$	14.62 13.80	15.41 14.77	16.78 16.24	18.08 17.61

With two loss functions, the quality of final reconstructions were much better and PSNR values significantly improved, compared to the single-decoder model with a single loss function. Fig. 12(b) shows the improvement in PSNR of the final output reconstruction, when two loss functions are combined together by a two-decoder network. Fig. 14 shows the variations in the reconstructed images with the use of different loss function combinations for 200 training samples.

Fig. 12(a) illustrates that BCE performs better compared to other loss functions when used in either single-decoder or two-decoder network model. Fig. 12(b) illustrates that the combinations BCE & $DSSIM$ and L_1 & BCE perform significantly better than other loss combinations for the two-decoder model. Even though PSNR values for $DSSIM$ loss were not sufficiently significant, it captures the spatial similarity aspects in reconstruction. Hence, the BCE & $DSSIM$ loss combination provides marginally better reconstructions, compared to L_1 & BCE for fewer number of training samples. However, when the number of training samples increase, L_1 & BCE combination produces much better reconstructions.

Figure 13. Variations in reconstruction with different loss functions

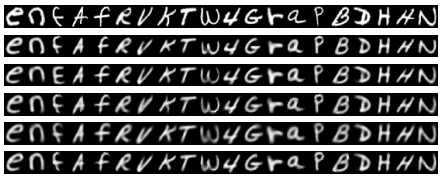


Figure 14. Variations in reconstruction with different loss function combinations

5. Conclusion

In this paper, we introduced a technique for increasing the size of a dataset by exploiting the concepts in CapsNets. We demonstrated the performance of this technique on well-known handwritten character datasets. Our algorithm takes limited amount of training samples, and perturb their corresponding instantiation parameters to create new training samples. In comparison to the conventional data augmentation techniques in the class of affine transformations, our technique generates images with subtle human-like variations to stroke pattern, boldness and other localized transformations. By combining the original dataset and the perturbed dataset as the training set, we achieved state-of-the-art results and better reconstructions of the input images at the decoder. To further improve the image reconstruction, we analysed the use of different loss functions and their combinations.

Our proposed method works well with images of characters. We intend to extend this framework to images on the RGB space, and with higher resolution, such as images from ImageNet and COCO. Further, we intend to apply this framework on regionally localized languages by extracting training images from font files.

6. Acknowledgement

The authors thank the Senate Research Committee of the University of Moratuwa for the financial support through the grant SRC/LT/2016/04 and the Faculty of Information Technology for providing computational resources.

References

[1] Cireşan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep, big, simple neural nets for handwritten digit recognition. Neural Computation **22** (2010) 3207–3220

[2] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS, Lake Tahoe, NV (2012) 1097–1105

[3] Lee, Y.: Handwritten digit recognition using k nearest-neighbor, radial-basis function, and backpropagation neural networks. Neural Computation **3** (1991) 440–449

[4] Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: NIPS, Long Beach, CA (2017) 3856–3866

[5] Hinton, G.E., Krizhevsky, A., Wang, S.D.: Transforming auto-encoders. In: ICANN, Berlin, Heidelberg (2011) 44–51

[6] LeCun, Y., Cortes, C., Burges, C.J.C.: The mnist database of handwritten digits (1998)

[7] Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86** (1998) 2278–2324

[8] Ciresan, D.C., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. CoRR (2012)

[9] Labusch, K., Barth, E., Martinetz, T.: Simple method for high-performance digit recognition based on sparse coding. IEEE Transactions on Neural Networks **19** (2008) 1985–1989

[10] Ranzato, M., Poultney, C., Chopra, S., LeCun, Y.: Efficient learning of sparse representations with an energy-based model. In: Proceedings of NIPS, Cambridge, MA (2006) 1137–1144

[11] Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multi-stage architecture for object recognition? In: ICCV, Kyoto, Japan (2009) 2146–2153

[12] Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: EMNIST: an extension of MNIST to handwritten letters. CoRR (2017)

[13] Dufourq, E., Bassett, B.A.: Eden: Evolutionary deep networks for efficient machine learning. In: PRASA-RobMech, Bloemfontein, South Africa (2017) 110–115

[14] Wiyatno, R., Orchard, J.: Style memory: Making a classifier network generative. CoRR (2018)

[15] Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. (2015)

[16] Bertinetto, L., Henriques, J.a.F., Valmadre, J., Torr, P., Vedaldi, A.: Learning feed-forward one-shot learners. In: NIPS, Barcelona, Spain (2016) 523–531

[17] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. (2014) 2672–2680

[18] Doersch, C.: Tutorial on variational autoencoders. arXiv preprint arXiv:1606.05908 (2016)

[19] Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R.: Deconvolutional networks. In: CVPR, San Francisco, CA (2010) 2528–2535

[20] Polesel, A., Ramponi, G., Mathews, V.J.: Image enhancement via adaptive unsharp masking. IEEE Transactions on Image Processing **9** (2000) 505–510

[21] Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: From error visibility to structural similarity. Trans. Img. Proc. **13** (2004) 600–612

[22] Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. CoRR (2017)

[23] Smith, L.N.: Cyclical learning rates for training neural networks. In: Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on, IEEE (2017) 464–472

[24] Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. ICML **28** (2013) 1058–1066

[25] Bhatnagar, S., Ghosal, D., Kolekar, M.H.: Classification of fashion article images using convolutional neural networks. In: ICIIP, Shimla, India (2017) 1–6

[26] Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. CoRR (2017)