Recurrent Capsule Network for Image Generation

Srikumar Sastry
DA-IICT
Gandhinagar

Abstract

We have already seen state-of-the-art image generation techniques with Generative Adversarial Networks (Goodfellow et al. 2014), Variational Autoencoder and Recurrent Network for Image generation (K. Gregor et al. 2015). But all these architectures fail to learn object location and pose in images. In this paper, I propose Recurrent Capsule Network based on variational auto encoding framework which can not only preserve equivariance in images in the latent space but also can be used for image classification and generation. For image classification, it can recognise highly overlapping objects due to the use of capsules (Hinton et al. 2011), considerably better than convolutional networks. It can generate images which can be difficult to differentiate from the real data.

1. Introduction

The objective of deep learning is to represent and predict probability distribution over various kinds of data such as natural language, images, audio and so on. Most of the models do so in *one shot,* meaning they predict or generate these distribution at once. In case of generative models, it can mean that all datapoints are conditioned on a single distribution. This is the “one shot” approach. The recurrent capsule network architecture, creates parts of images which are independent from each other similar to the DRAW network. Additionally, it uses capsules to perform image segmentation and preserve the actual pose of objects in the image. It uses the variational encoding framework with separate margin loss for the capsules and variational upper bound on the log-likelihood loss for the recurrent network.

The recurrent network generates images piece by piece which mimics human when drawing an image. When a human is asked to draw, he/she will do so in an iterative and sequential manner, focussing on a single area at a time while ignoring other parts of the image. It will not suffer from mode collapse unlike the generative adversarial networks and much more easy to train. *The preservation of position of objects in images is an important aspect for image generation*. For instance, to generate an image of a face, we require objects such as eyes, nose, mouth to be present inside the face and not elsewhere. Not only we should ensure these objects being inside face, but we also require them to be at correct position inside the face, for instance eyes being above nose.

2. Related work for image generation

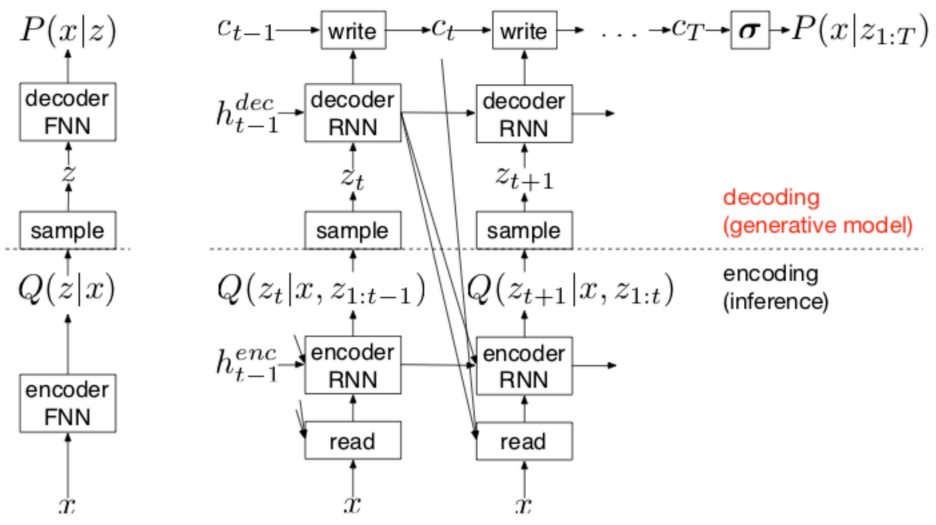


FIGURE 1. LEFT: STANDARD VARIATIONAL AUTOENCODER. IT USES FEED FORWARD OR CONVOLUTIONAL NETWORKS IN THE ENCODER AND DECODER SECTION. Z IS THE LATENT VARIABLE WHICH ENCODES IMAGES. RIGHT: DRAW NETWORK. SAME STRUCTURE AS OF THE VARIATIONAL AUTO ENCODER BUT USES RECURRENT NETWORKS IN THE ENCODER AND DECODER SECTION WHICH GENERATES IMAGES PIECE BY PIECE. AT EVERY TIME STEP, PREVIOUS ENCODER AND DECODER RESULTS ARE PASSED TO ENCODER RNN AS INPUTS.

Generative Adversarial Networks [GAN] (Goodfellow et al. 2014) proposed adversarial training procedure alternatively minimising loss of the generator and discriminator. It is hard to train a GAN as it requires to solve the MINI-MAX optimization problem. Also, it suffers from mode collapse and does not preserve spatial information of objects in images. The proposed Deep Convolutional GAN [DCGAN] fails to encode object orientation and position and the generated images are not related to the real data.

Variational Autoencoders are tricky to train and suffer over pruning problems. Suppose, we are trying to reconstruct over some probability distribution $p(x)$ with $x \in R^k$. Let the latent dimension variable be z. So we can represent the reconstruction of $p(x)$ as :

$$p(x) = \int p(x|z)p(z)dz$$

The problem with latent variable modelling is that we cannot compute the above reconstruction expression directly. So, we resort to techniques such as monte carlo sampling or the importance sampling to approximate such expression. Variational autoencoders provide a way to model such latent variable models.

Finally, the DRAW network[1] (K. Gregor et al. 2015) which uses recurrent variational auto encoding framework to generate image piece by piece. It also fails to segment image and preserve poses.

3. Recurrent Capsule Network

The structure of RCN is similar to that of DRAW network: encoder which consists of RNN which encodes images over latent codes and a decoder which also consists of RNN which decodes the latent codes and generates images piece by piece. The main difference is the addition of primary capsules in the encoder and decoder just before the RNN layer. The decoder and encoder outputs at previous step are added as inputs in the current time step similar to the DRAW network, and also the input from capsules are taken into consideration.

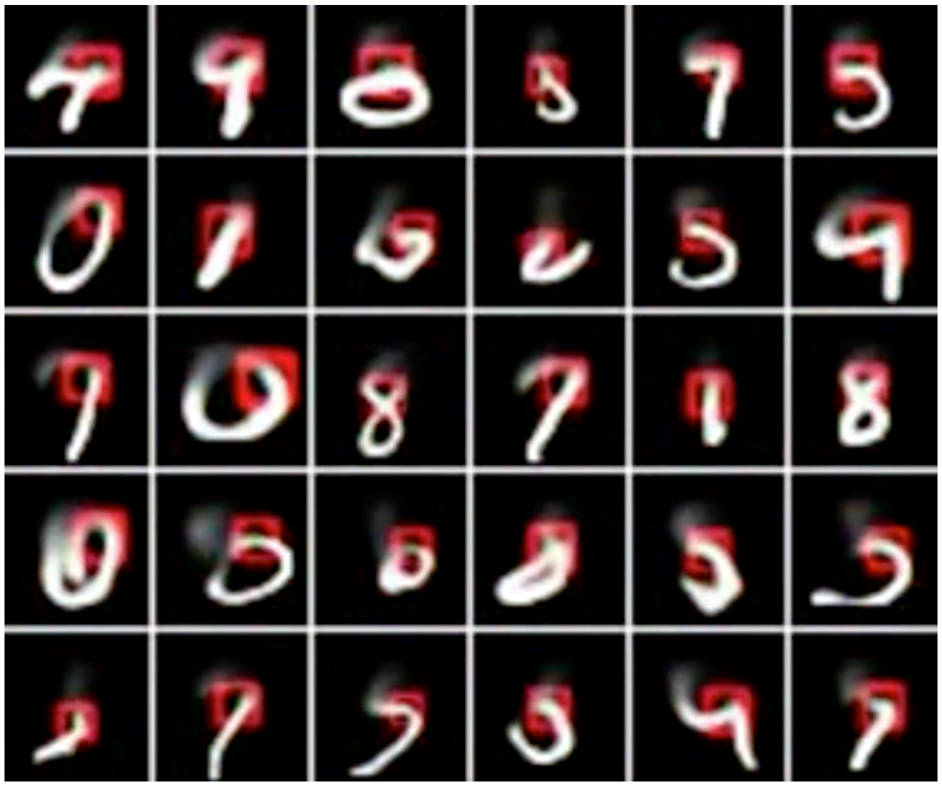


FIGURE 2. TRAINED RCN DRAWING IMAGE PIECE BY PIECE. EACH TIME IT GENERATES IMAGES NOT PRESENT IN THE DATASET. THE RED RECTANGLE REPRESENTS THE AREA OF IMAGE CURRENTLY FOCUSSED BY THE NETWORK.

[1] https://arxiv.org/pdf/1502.04623.pdf

4. Capsule Network

Capsule network consists of layers of capsules. Each layer is divided into groups of neurons called capsules. Capsules will be considered to be active depending on the activation of neurons. Every active capsule will select another capsule present in the next layer using a routing algorithm.

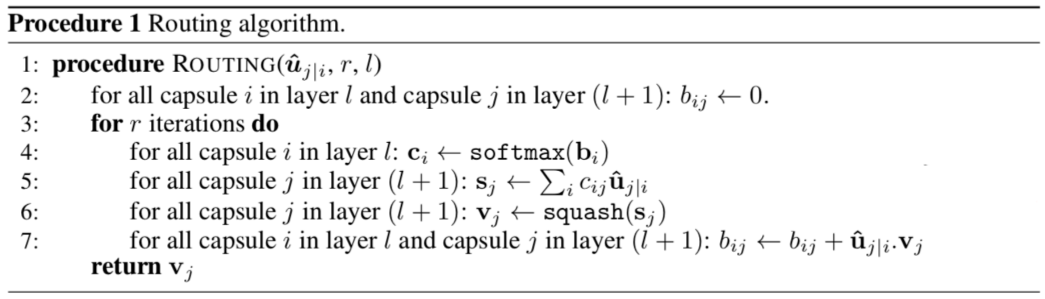
The activation of neurons in the capsule will depend on various properties of the image such as colour, pose, position, texture etc. Thus, each capsule will capture some property of objects in the image. So, a capsule will be active if the object represented by that capsule is present in the image.

Output of a capsule will be a vector. We want this vector to represent some sort of probability. Thus we need the output vector to lie between 0 and 1. We can achieve it using some non-linearity operators. The original paper[2] used squashing function which ensures small output vectors shrunk close to zero and large vectors shrunk close to 1.

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||^2}$$

Where, v_j is output vector of capsule j and s_j is its total input.

Routing algorithm for choosing the capsule in next layer is defined in the paper:



Procedure 1 Routing algorithm.

```
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
5:         for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij} \hat{\mathbf{u}}_{j|i}$
6:         for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
       return $\mathbf{v}_j$
```

Margin loss is used to determine the existence of an object :

$$L_k = T_k max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) max(0, ||v_k|| - m^-)^2$$

where, total number of possible objects in the image is k. T_k = 1 iff a digit of class k is present and m^+ = 0.9 and m^- = 0.1.

CapsNet architecture consists of convolutional layers which are fed with the input images. The output of these convolutional layers are fed into *primary capsules* which perform the inverse graphics process that is inverse rendering process. The primary capsule layer is connected to another capsule layer which then outputs the probability of presence of an object in the image.

[2] https://arxiv.org/pdf/1710.09829.pdf

5. Recurrent Capsule Network Architecture

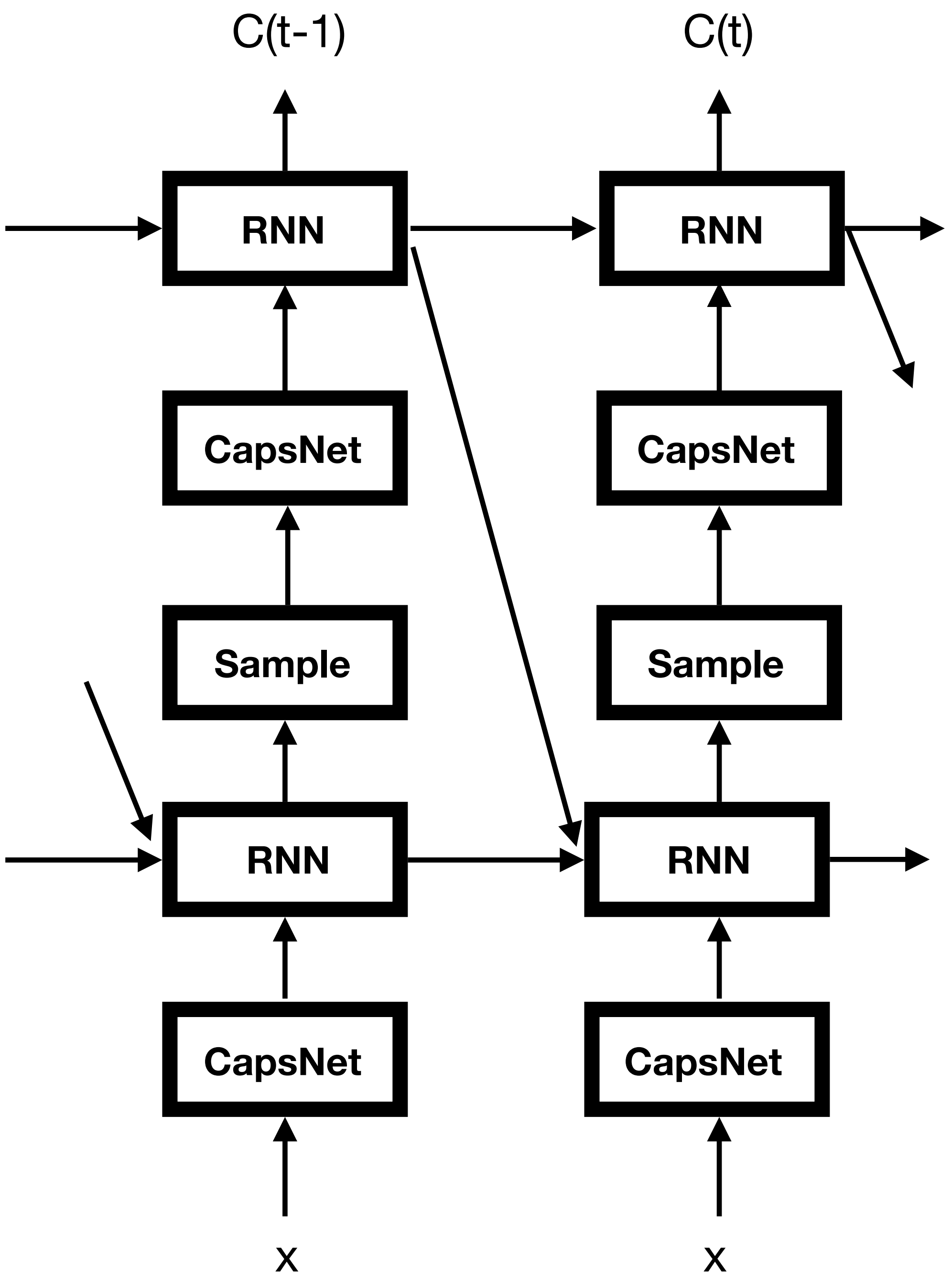


FIGURE 3. RCN ARCHITECTURE. THE NETWORK BELOW SAMPLE PHASE IS THE ENCODER NETWORK AND THE NETWORK ABOVE IS THE DECODER OR THE GENERATOR NETWORK.

RCN architecture will use images as inputs which are properly preprocessed. Inputs will be fed to a capsule network consisting of primary capsules and second capsule layer. It will not have the final connected layers as opposed to the original CapsNet architecture.

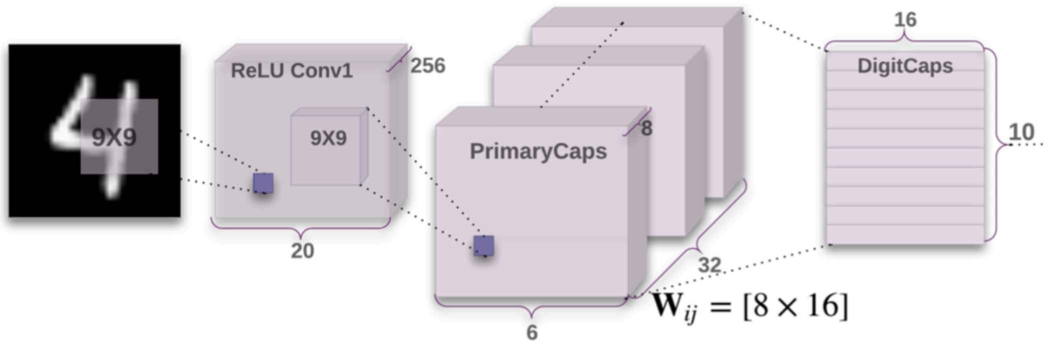


FIGURE 4. CAPSNET ARCHITECTURE USED IN RCN. NOTE THAT THE LAST CONNECTED LAYER IS NOT USED.

The output of the final capsule layer will be fed to the first RNN in the encoder. This RNN will generate the latent space of the autoencoder. The RNN architecture used is *Long Short Term Memory* (LSTM) (Hochreiter & Schmidhuber (1997)). LSTM will make training easier and help solve vanishing gradient problem. The encoder RNN will receive 3 inputs from i) Final Capsule Layer, ii) Decoder RNN at previous time step iii) Encoder RNN at previous time step.

The latent distribution is *Gaussian*. At each time step, a sample drawn from the latent distribution will be passed to the next capsule network. This capsule network will be composed of 2 *deconvolutional layers* followed by primary capsule layer and a final capsule layer. The output of final capsule layer will be passed to the decoder RNN which once again uses LSTM architecture. This RNN layer will construct the real image from the sample latent space.

6. Loss function

The loss function for optimising the capsules is the margin loss as described earlier.

The reconstruction loss function for both the RNN is the same as used by the DRAW architecture. It is defined as the negative log probability :

$$L^x = -\, log D(x \mid c_T)$$

The latent loss L^z for a sequence of latent distributions Q(Z |h^{enc}) is defined as the summed Kullback-Leibler divergence of some latent prior P (Z) from Q(Z |h^{enc}) :

$$L^z = \sum_{t=1}^{T} KL(Q(Z \mid h^{enc}) \mid\mid P(Z_t))$$

This loss depends on Z which in turn depends on input x. $Q(Z \mid h^{enc})$ can be calculated using prior latent distribution for example Gaussian.
The total loss is then defined as :

$$L = \langle L^x + L^z \rangle_{z \sim Q}$$

7. Experiments

MNIST

The MNIST dataset consists of handwritten digits 28x28 in size. The dataset has 60K and 10K images for training and testing respectively. The CNN layers in CapsNet has 256 and 128 channels respectively. Each has 5x5 kernels and 1 stride. Deconvolutional layers 128 and 256 channels respectively. RCN model gets a test error of **3.12**% on this dataset.

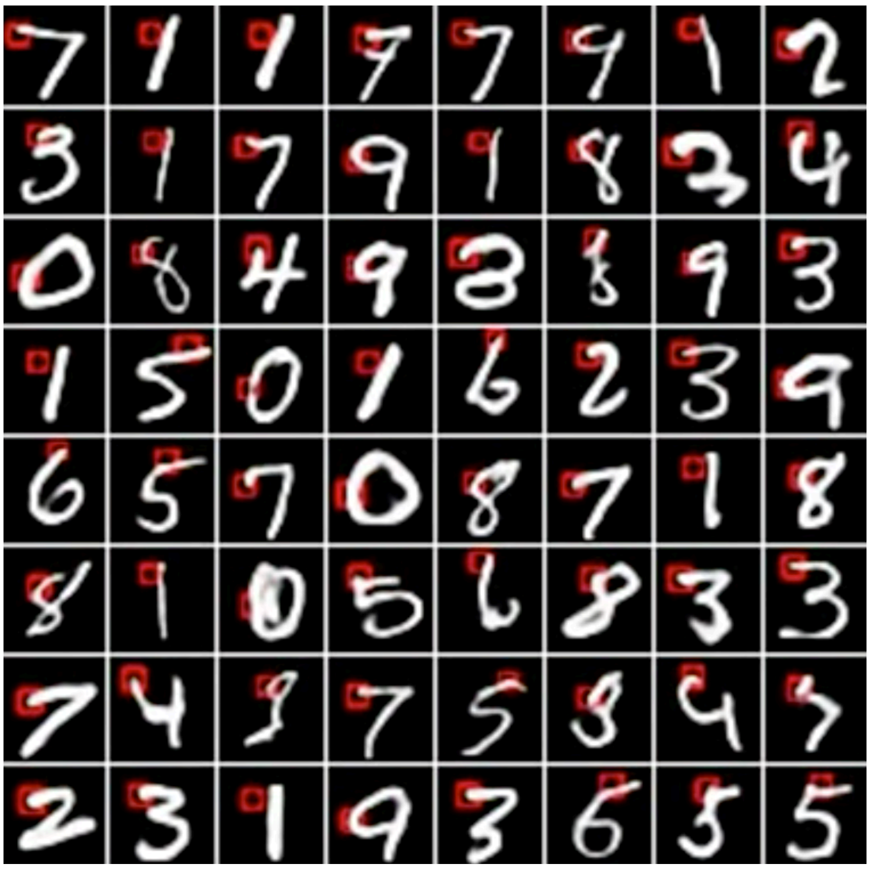


FIGURE 5. GENERATED IMAGES OF HANDWRITTEN DIGITS USING RCN. NEW IMAGES ARE GENERATED WHICH ARE NOT PRESENT IN THE REAL DATA. L2 NORM OF DIFFERENCE BETWEEN THE PIXELS OF GENERATED IMAGES AND REAL DATA IS QUITE SMALL AS EXPECTED.

CIFAR-10

The Cifar-10 dataset consists of handwritten digits 32x32 in size. The dataset has 50K and 10K images for training and testing respectively. The CNN layers in

CapsNet has 256 and 128 channels respectively. Each has 5x5 kernels and 1 stride. Deconvolutional layers 128 and 256 channels respectively. RCN model gets a test error of **13.1**% on this dataset.

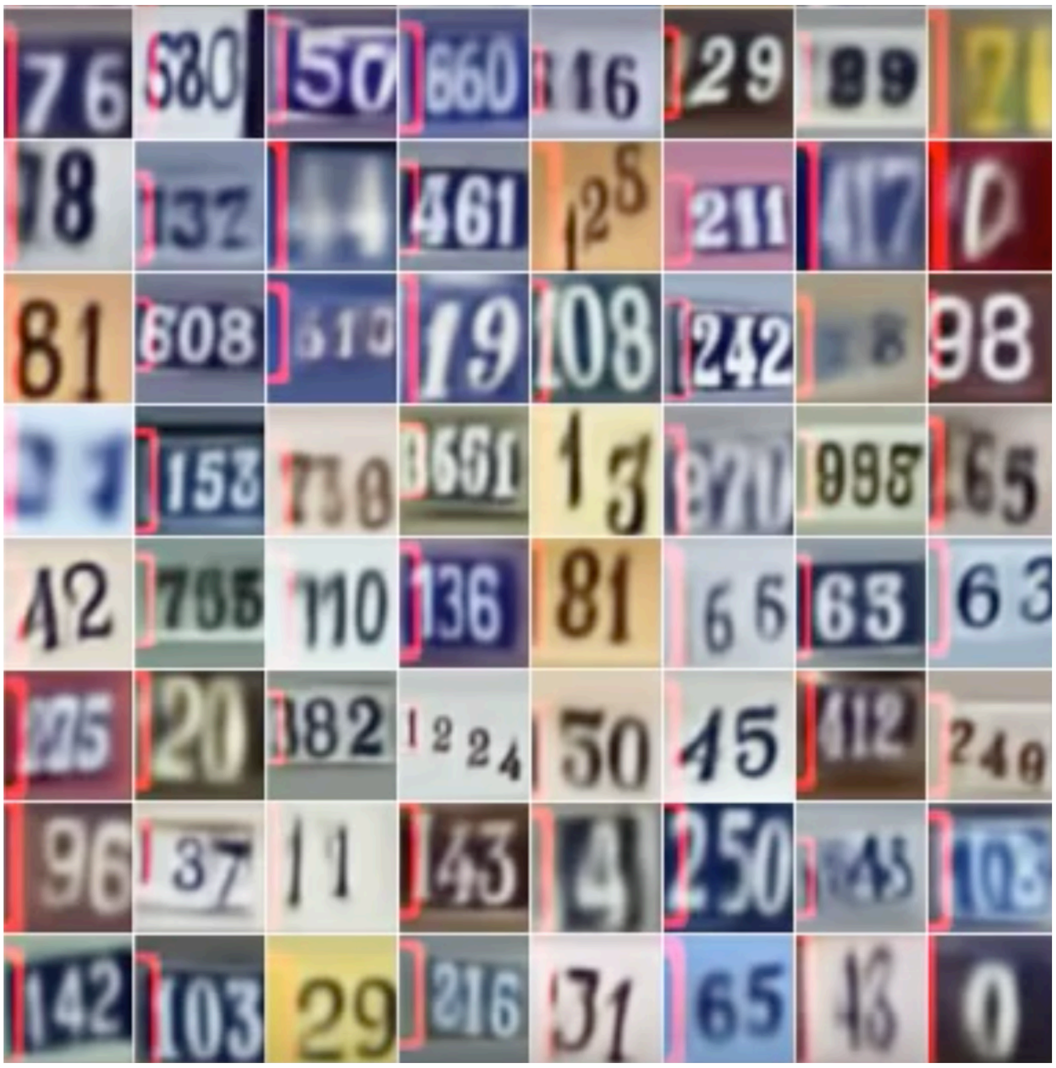


FIGURE 6. GENERATED IMAGES OF CIFAR-10 USING RCN. NEW IMAGES ARE GENERATED WHICH ARE NOT PRESENT IN THE REAL DATA. L2 NORM OF DIFFERENCE BETWEEN THE PIXELS OF GENERATED IMAGES AND REAL DATA IS QUITE SMALL AS EXPECTED.

8. Advantages and disadvantages

This model has its own advantages and disadvantages. It can mimic humans in drawing part by part focussing on single area at a time. The inclusion of capsules makes the network learn spatial location and orientation of objects. Training only requires backdrop and computing gradient. A wide range of images can be generated.

RCN comes with its own disadvantages. RCN is hard to implement by code. Training RCN is also computationally expensive due to minimising two losses : margin loss and the negative log likelihood loss.

9. Conclusion

This architecture can be further adapted :

1. Gated Recurrent Units may be used instead of LSTM which are computationally more efficient.
2. The output of capsules can be used for feature learning.
3. The encoder and decoder networks could be trained on separate GPUs to accelerate training.
4. Decoupled Neural interfaces using synthetic gradients can be used to further reduce training time.

10. References

[1] Aurélien Géron. Capsule Networks (CapsNets) – Tutorial. https://www.youtube.com/watch?v=pPN8d0E3900

[2] Aurélien Géron. extra_capsnets.ipynb. https://github.com/ageron/handson-ml/blob/master/extra_capsnets.ipynb

[3] Sara Sabour, Nicholas Frosst, Geoffrey E Hinton. Dynamic Routing Between Capsules. arXiv:1710.09829

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. arXiv:1406.2661

[5] Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434

[6] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, Daan Wierstra. DRAW: A Recurrent Neural Network For Image Generation. arXiv:1502.04623

[7] Diederik P Kingma, Max Welling. Auto-Encoding Variational Bayes. arXiv:1312.6114v

[8] Carl Doersch. Tutorial on Variational Autoencoders. arXiv:1606.05908