Improving Generative Adversarial Networks with CapsuleNET

Arnab Karmakar
Indian Institute of Space Science and Technology
Thiruvananthapuram, Kerala
arnabkarmakar.001@gmail.com

Abstract

This project describes Generative Adversarial Capsule Network (CapsuleGAN), a framework that uses capsule networks (CapsuleNets) instead of the standard convolutional neural networks (CNNs) as discriminators within the generative adversarial network (GAN) setting, while modeling image data. We designed CapsuleNet discriminators with the updated GAN objective function, which incorporates the CapsuleNet margin loss for training CapsuleGAN models. We show that CapsuleGAN outperforms convolutional-GAN at modeling image data distribution on MNIST dataset.

1. Introduction

Generative Adversarial Networks (GANs)[2] employ two deep learning architectures as adversaries Generator and Discriminator to promote the Generator to create data that belong to a particular dataset. GANs have proven to be able to generate images that look at least superficially authentic to human eyes.

For a Generator to be able to learn the mappings from a latent variable space to a particular data distribution, it is imperative for the Discriminator to be a stronger model, that is, a stronger classifier. A strong Discriminator will force the Generator to find the features relevant to the data distribution, thus, improving the Generator's performance.

Traditionally, a Deep Convolutional GAN (DCGAN)[6] features a deconvolutional neural network that generates data (the Generator) and a convolutional neural network model that classifies whether an input image is a fake or a real image. Recently, Geoffrey E. Hinton et al. (2017)[1] from Google Brain released a paper explaining the shortcomings of a convolutional neural network. Their paper titled as 'Dynamic Routing Between Capsules'[7] tries to achieve an architecture that tries to address the issues a CNN faces. Their results have demonstrated their architecture of Capsule Networks to be more powerful and resilient in comparison to CNN[4].

Now with this improved version of classifiers[3], we were inspired to see what effects it has when plugged into a GAN setting. Here we have kept the rest of GAN architecture unchanged with only, Dynamically routed capsule working as the discriminator for the network. Our expectations were that this new architecture will provide us better generative abilities than the traditional DCGANs.

2. Theory

2.1. Problems with Convolutional Neural Networks

Convolutional Neural Networks (CNN)[5] are one of the reasons deep learning is so popular today. The main component of a CNN is a convolutional layer. Its job is to detect important features in the image pixels. Layers that are deeper (closer to the input) will learn to detect simple features such as edges and color gradients, whereas higher layers will combine simple features into more complex features[9]. Finally, dense layers at the top of the network will combine very high level features and produce classification predictions.

An important thing to understand is that higher-level features combine lower-level features as a weighted sum: activations of a preceding layer are multiplied by the following layer neurons weights and added, before being passed to activation nonlinearity. Nowhere in this setup there is pose (translational and rotational) relationship between simpler features that make up a higher level feature. CNN approach to solve this issue is to use max pooling or successive convolutional layers that reduce spacial size of the data flowing through the network and therefore increase the "field of view" of higher layers neurons, thus allowing them to detect higher order features in a larger region of the input image. Max pooling is a crutch that made convolutional networks work surprisingly well, achieving superhuman performance in many areas[8]. But, while CNNs work better than any model before them, max pooling nonetheless is losing valuable information.

Hinton himself stated that the fact that max pooling is working so well is a big mistake and a disaster[1]

> *"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster."*

Although, even without the max pooling layer one can good results with traditional CNNs, but they still do not solve the key problem:

> *Internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects.*

Let us consider a very simple and non-technical example. Imagine a face. As the features We have the face oval, two eyes, a nose and a mouth. For a CNN, a mere presence of these objects can be a very strong indicator to consider that there is a face in the image. Orientational and relative spatial relationships between these components are not very important to a CNN (Figure 1).

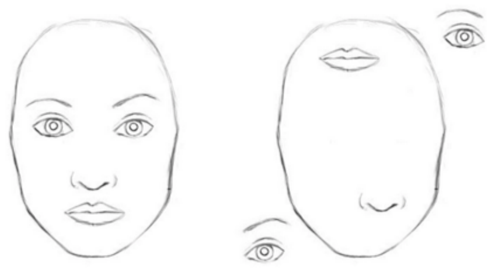


Figure 1: To a CNN, both pictures are similar, since they both contain similar elements

In the example above, a mere presence of 2 eyes, a mouth and a nose in a picture does not mean there is a face, we also need to know how these objects are oriented relative to each other.

What is needed is not positional invariance, but equivariance between the features. This leads to the requirement of an architecture that understands meta-features of the sub-features that help build up a more complex feature structure. It is this problem that Capsule Networks (as described in 'Dynamic Routing between Capsules' by Geoffrey E. Hinton et al.) try to address. A 'routing' mechanism is proposed which decides the flow of data through the network of 'Capsules'. These Capsules are structures that represent a particular feature and help understand the meta-features like orientation, velocity etc. regarding a particular feature.

2.2. Capsule Networks

The paper compares to the current state-of-the-art CNNs. In this paper the authors project that a human brain has modules called "capsules". These capsules are particularly good at handling different types of visual stimulus and encoding things like pose (position, size, orientation etc.), deformation, velocity, albedo, hue, texture etc. The brain must have a mechanism for "routing" low level visual information to what it believes is the best capsule for handling it.

The paper proposes an architecture for creating these capsules which will replace the neurons in a neural network setting. The capsules consist of a nested layer of neurons. A capsule outputs an activation vector v_j. v_j is defined such that its length describes the probability of the existence of the entity represented by its capsule. Shorter vectors are squashed near 0 and the longer ones are valued a little under 1 using a non-linear activation function defined by the paper:

$$v_j = \frac{||s_j||^2}{1 + ||s_j||^2} * \frac{s_j}{||s_j||^2} \quad (1)$$

Where v_j is the output vector and s_j is the input of the activation function.

Let u_i be the output of the previous capsule layer, which acts as the input for the current layer j. Now, the outputs of the previous layers are multiplied by a weight matrix W_{ij} which results into what is called as a prediction vector $u_{j|i}$.

$$u_{j|i} = W_{ij} u_i \quad (2)$$

Now, this prediction vector is then multiplied by the coupling coefficients between the capsules of the layers i and j : c_{ij}, to give the resultant s_j.

$$s_j = \sum_i c_{ij} u_{j|i} \quad (3)$$

Here, the sum of all coupling coefficients for a capsule is equal to 1. It is determined by the process of 'routing softmax' where the logits, b_{ij} are the log probabilities that a capsule i couples with a capsule j.

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \quad (4)$$

These log priors are then learnt discriminatively at the same time as that of the other parameters. The coupling coefficients are then learnt iteratively by monitoring the agreement between the capsules of the different layers. This agreement is measured by the scalar multiplication of the output of the capsule of the given layer (v_j) and that of the input prediction vector coming from the previous layer ($u_{j|i}$).

$$a_{ij} = v_j u_{j|i} \tag{5}$$

This value of a_{ij} is treated as log likelihood and is added to the initial value of b_{ij}. This is then used to calculate the new values of coupling coefficients using the routing softmax method.

In the final layer, the length of the of output vectors defines the class of the given input data. For multiple classes (say for k classes), the loss will be as follows:

$$L_k = T_k max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) max(0, ||v_k|| - m^-)^2 \tag{6}$$

Where $T_k = 1$ in case of the presence of a particular class and $T_k = 0$ in case of its absence. λ is the weighing down factor that stops initial learning from shrinking the lengths of the activity vectors of all final layer capsules in case the target class is absent. Following are the values for the other parameters used: $m^+ = 0.9$, $m^- = 0.1$, $\lambda = 0.5$. The total loss is the sum of losses of each class.

Also, a reconstruction loss is used to enable the capsules to encode the instantiation parameters of the input into the network. The activity vector of the correct class is used to reconstruct the input image. The decoder is a fully connected neural network that outputs the pixel densities. Least square error between the pixel values of the input data and the reconstructed image multiplied by a scaling factor forms the reconstruction loss. The scaling factor helps control the influence of the reconstruction loss during the optimization process over the marginal loss discussed in the previous paragraph.

2.2.1 Advantages

Hinton's CapsuleNET has outperformed the traditional CNNs in many ways:

- cut error rate by 45% as compared to the previous state of the art.
- only using a fraction of the data that a CNN would use
- preserve hierarchical pose relationships between object parts - numerically as a 4D pose matrix.
- Translation, rotation, scale invariant.
- Capsules encapsulate all important information about the state of the feature they are detecting, in vector form.

A generalized comparison between CapsuleNET and traditional CNNs are shown in Figure 3

2.3. Generative Adversarial Networks

The model uses noise variables z that act as latent variables to generate the data that needs to belong to data x. The Generators probability distribution, p_g, is defined over the data x with a prior on z as $p_z(z)$. The Generator function $G(z;\theta_g)$ represents a mapping from the latent variable space to the data space. The Discriminator function $D(x_d, \theta_d)$ maps the data space to a single scalar which represents the probability that x_d belongs to the actual dataset x or to p_g.

The Discriminator is trained to maximize the probability of classifying an image whether it comes from the Generator or belongs to the original dataset correctly. The Generator is in turn trained to fool the Discriminator that is to maximize the probability of having a generated image being classified as belonging to the original dataset.

2.3.1 DCGAN - CNN as Generator

DCGAN has proveb to be the most stable and robust architecture in the GAN perspective. Also, it provides for faster learning which in turn enables our CapsuleGAN for faster convergence.

2.3.2 CapsuleNET as Discriminator

In our architure, we intend to incorporate a strong discriminator compared to the generator so that the discriminator can learn wuickly and aid the generator for accurate and fast learning. Therefore, CapsuleNET has been the preferred choice. Also, CapsuleNET is less time and resource consuming when used as a discriminator than the generator. CapsuleNET as the generator has been much slower and the discriminator outperforms the generator every time; hence the learning of discriminator is not proper.

Hence the most effective choice is DCGAN-CNN as Generator alongwith CapsuleNET as Discriminator.

3. Experiment

3.1. Dataset

We trained the Capsule Network architecture on MNIST dataset with 60000 images, 6000 images per class for 10 classes.

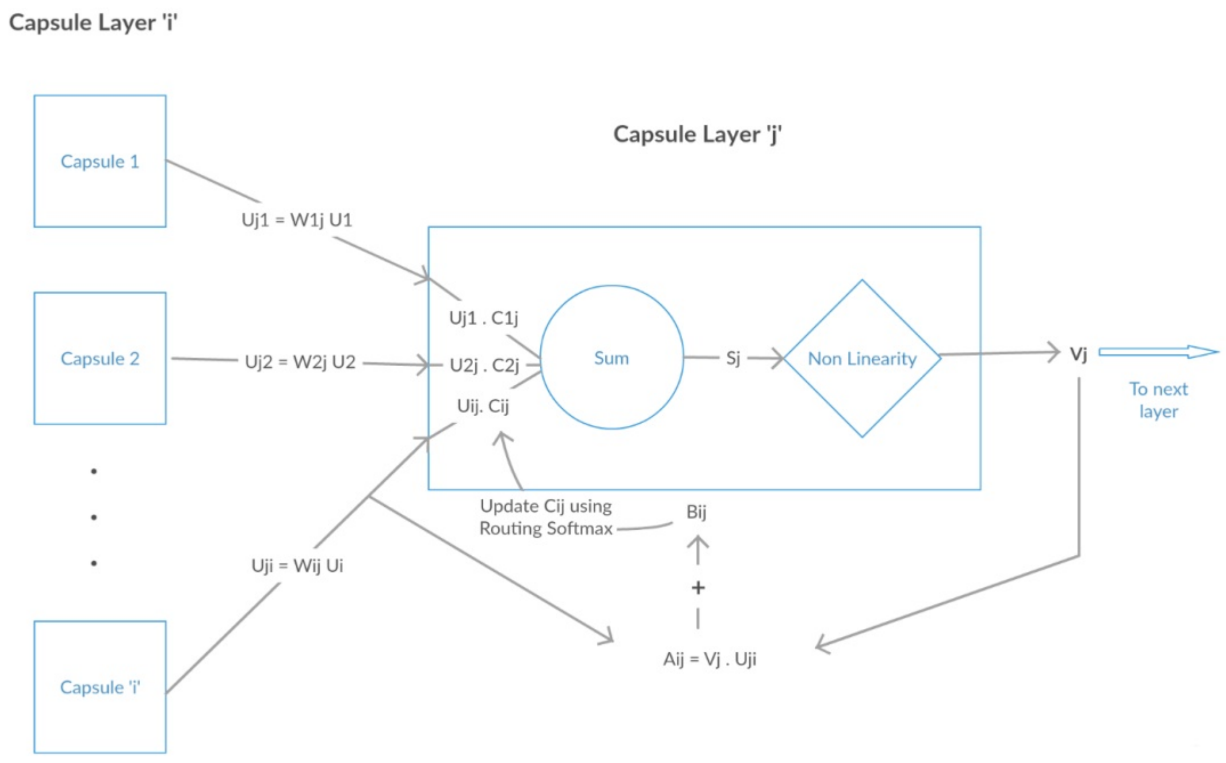
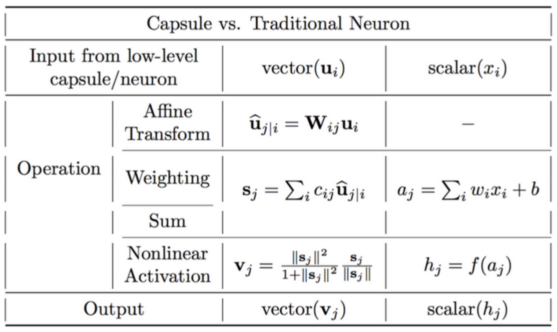


Figure 2: Capsule Structure and Data Flow



Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	–
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\\|\mathbf{s}_j\\|^2}{1+\\|\mathbf{s}_j\\|^2}\frac{\mathbf{s}_j}{\\|\mathbf{s}_j\\|}$	$h_j = f(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)

Figure 3: CapsuleNET Architecture vs. traditional CNN Architecture

3.2. Discriminator: CapsuleNET Architecture

The Discriminator in our network is the CapsuleNET architecture.

We have employed a layer of Convolutional Neural Network layer that uses 256 kernels to extract features from the input image of shape 28x28. Parametric ReLU (PReLU) is used to apply non-linearity to the output of this layer. We then pass the output of the CNN to the first Capsule Layer.

There is no routing between the output of the CNN and the Primary Capsule layer the primary capsule layer sees all the filtered channels. The primary capsule layer then runs another CNN with a filter of 7x7 units and a stride of 2 in each direction. The output is then reshaped in a manner in which we have 32 channels of 8-dimensional capsules (PrimaryCaps layer). Each capsule is defined as the vector consisting of pixel values of a particular pixel position over 8 filtered images from the Primary Capsule CNN. These are then routed over to the Secondary Capsule Layer which consists of 16-dimensional capsules (DigitCaps layer).

After the DigitCaps layer, we implemented a Flattening layer. this is an auxiliary layer that helps the vector output of CapsuleNET convert to scalar probability values. Thereafter, we added a fully connected (dense) layer of length 64 and the output is taken from the final dense layer of size 1 which is the prediction of 0 or 1 based on the input is either fake or real. The final dense layer uses a sigmoid activation.

The architecture is shown in Figure 4

The system has been trained for 10000 epochs and the learning rate has been set at 0.0002 with the Adam optimizer.

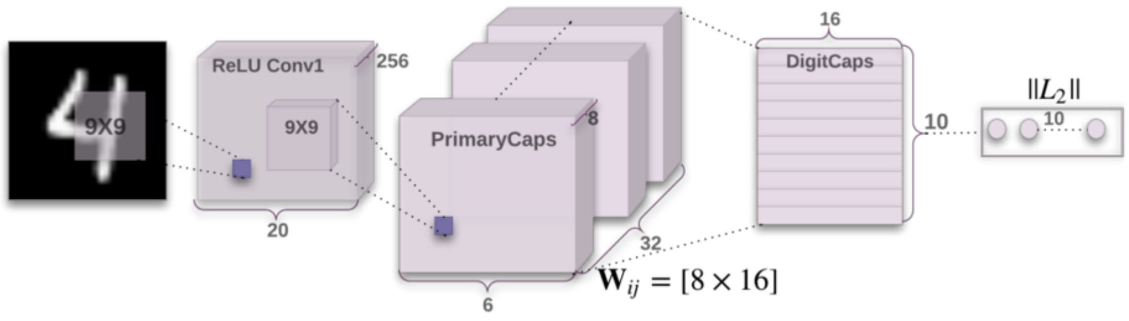


Figure 4: CapsuleNET Architecture

3.2.1 Loss function

We used binary crossentropy as the loss function of the CapsuleNET Discriminator.

3.3. Generator: DCGAN architecture

We use a Deconvolutional Neural Network (Generator) to map a set of 100 latent variables with its values generated randomly to an image of size 28x28. We employ kernels of size 3x3 at each layer. The activation used for non-linearity is Leaky-ReLU. 5 convolution layers have been stacked to make the Generator for the GAN with the following number of kernels in each layer: 128, 64, 32, 3, 1.

The Discriminator is a Deep Convolutional Neural Network with 5 convolutional layers with Leaky-ReLU activation at each layer except for the last layer where we use sigmoid to produce probabilities for classes.

The architecture is shown in Figure 5

The learning rate for the model is 0.0002 with the Adam optimizer as used previously.

3.3.1 Loss function

We used binary crossentropy as the loss function of the DCGAN (CNN) Generator.

3.4. Integrated CapsuleGAN archhitecture

The final architecture for the combined Generator (DCGAN) and Discriminator (CapsuleNET) model has been shown in Figure

3.5. Total Loss

The following losses have been minimized to train the model: (Figure 7)
where D(x) represents the probability of x coming from the real data rather than the Generator.

4. Training

We use a batch size of 32 for 10000 epochs. Each batch consists of equal number of real images (16) as well as generated images (16) from CNN. one batch is used for training either the generator or the discriminator, i.e. when the discriminator is training, the generator is non-trainable, in the very next batch, the generator training will take place and the discriminator will be non-trainable. The noise vector from the generator is sampled from a gaussian distribution of zero mean and unit variance. We did not use Batch-Normalization while training although we believe it might increase the visual representation of the generated images.

The training for 10000 epochs took around 8-10 hours in CPU with a system consisting intel-Xeon chipset with 32GB of RAM.

5. Results

The loss-accuracy curves for the CapsuleNET Discriminator and the loss curve for DCGAN-CNN generator is shown in Figure 8 and Figure 9. Here we present the results for 1000 epochs because after that the curves remains almost constant (although the visual quality of the generated images improve) and the initial fluctuations of the learning curve has important implications which is clearly seen here.

5.1. Implication

- Discriminator accuracy starts from zero, goes to maximum (100%) quickly, implies Discriminators learning is superior than Generator.
- Strong Discriminator Accuracy goes up in the initial stage, loss goes down. Generator loss goes high simultaneously.
- Discriminator accuracy fluctuates Generator starts learning, Generator loss starts to fall.
- Discriminator accuracy at standstill (50%), reached convergence

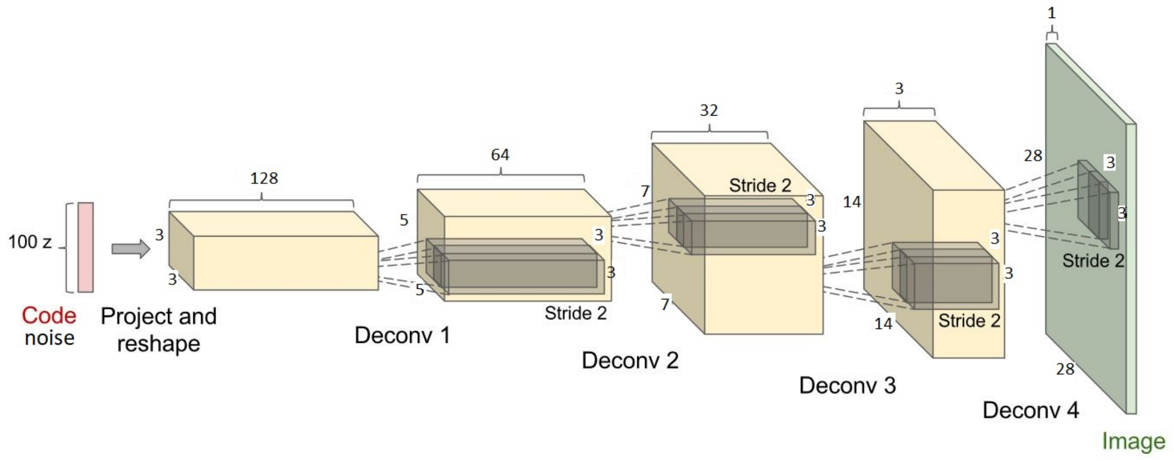


Figure 5: DCGAN Architecture

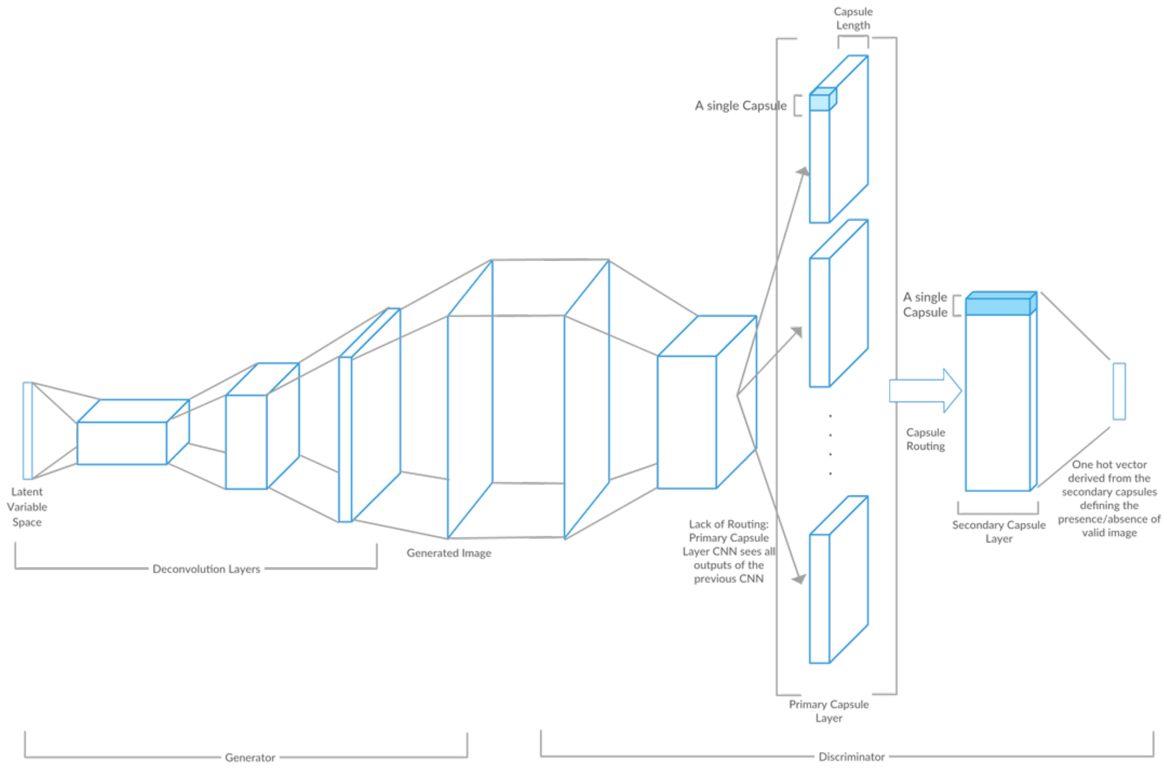


Figure 6: A generalised representation of CapsuleGAN Architecture

- Generator Loss keep on falling, learning better and better parameters to mimic actual data
- Generator loss, Discriminator loss and accuracy at standstill, reached convergence weight updation still going on but on a very minute scale, visual represen-

Discriminator Loss:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

Generator Loss:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Figure 7: Loss function

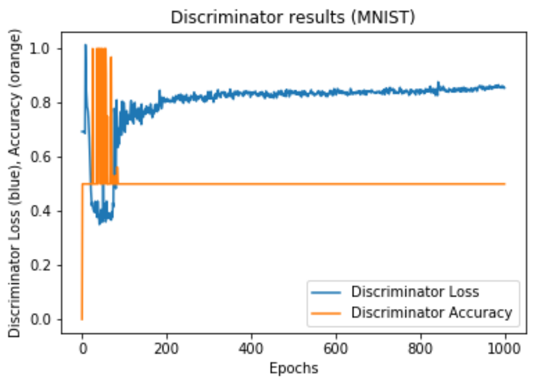


Figure 8: Discriminator Loss-Accuracy curve

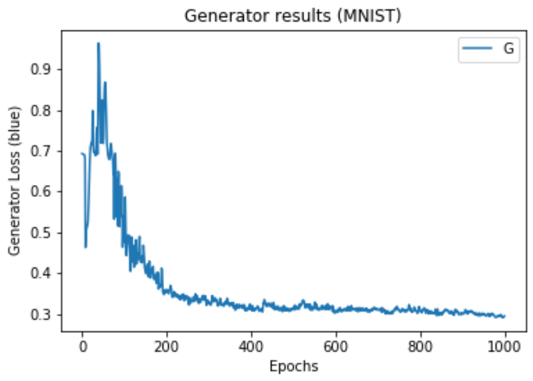


Figure 9: Generator Loss curve

tation of generated data (sharpness, contrast) increases slightly.

- Discriminator Accuracy never goes below 50% due to the fact that each batch consists of equal number of generated as well as real world images strong discriminator is helpful

- Also implies that Discriminators learning capacity is very high.

5.2. Comparison

A comparative study between the traditional DCGAN (CNN as both generator and discriminator) and our CapsuleGAN (CNN as Generator but CapsuleNET as Discriminator) is shown in Figure 10 at 1000 epochs and 10000 epochs respectively. In our implementation, we see that CapsuleGAN develops a better representation of the MNIST data distribution within 10000 epochs whereas in DCGAN takes around 30000 epochs to deliver similar results.

The model begins to develop similar looking images at a faster rate despite having a reduced number of Discriminator training iterations. This can be attributed to the more powerful classifier model in comparison to the Deep Convolutional Neural Network setting. The images generated by the model over different epochs are shown in Figure 11

6. Challenges

We had limited amount of computing power and memory available, which restricted us to keep the architecture simple enough for the memory constraints to be met.

Also, another barrier was the time required to train the architecture. We would have to wait for at least 5-6 hours to see the effects of every change that we made. This made the whole process very time-consuming.

7. Conclusion

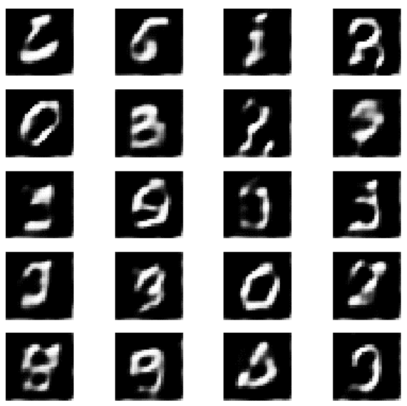
Initial experiments of Capsule Networks on MNIST dataset provided us with results that showed a clear improvement on Convolutional Networks. Then we implemented CapsuleNet in GAN architecture with a Deconvolutional Generator. Despite a very strong Discriminator Model in comparison to the Generator, the model gave better results over a traditional DCGAN setup by generating better MNIST images in significantly lesser number of epochs.

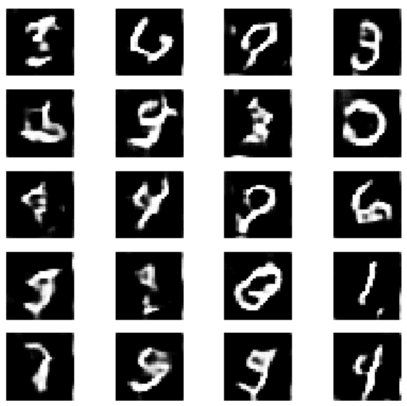
Thus, it can be said that incorporating CapsuleNet in GAN leads to a better discriminator, in turn, making the generator work better.

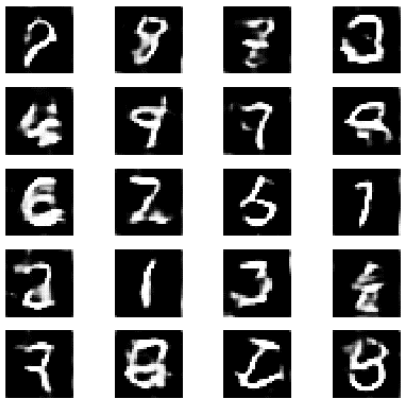
8. Future Work

Looking at the results of the experiment, we want to apply this architecture on different datasets such as fashion-MNIST and notMNIST. Also, we would like to extend out work in RGB images, specifically to train our model with cifar-10 dataset and so son. We hope to achieve improved results on these datasets.

In this experiment, we have implemented the CapsuleNet only as a discriminator in the GAN architecture. We aim to


(a) DCGAN@1000


(b) CapsuleGAN@1000


(c) DCGAN@10000

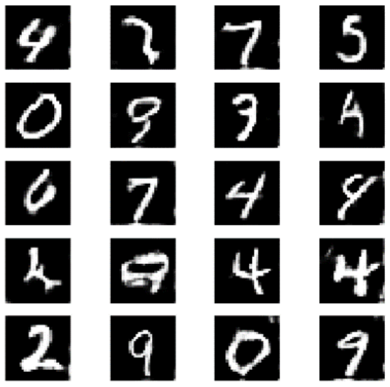

(d) CapsuleGAN@10000

Figure 10: Comparison between DCGAN and CapsuleGAN after 1000 iteration and 10000 iterations respectively

implement it as generator as well which is expected to lead to a better image generation. This is inspired from the fact that CapsuleNet is based on the concept of inverse graphics. Neurons in a capsule represent different instantiation parameters of an image that cant be captured with the ConvNet. Thus, we expect that CapsuleNet as a generator will lead to a better image generation.

Currently, the main constraints are the slow learning of CapsuleNET as a Generator and the time taken to learn. We hope to improve the perfomance and time constraints by reducing the parameters of CapsuleNET and implementing in multiple GPUs.

References

[1] G. E Hinton. Geoffrey Hinton: What is wrong with convolutional neural nets? `https://www.youtube.com/watch?v=Jv1VDdI4vy4`. Uploaded: 26.09.2017.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[3] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pages 44–51. Springer, 2011.

[4] G. E. Hinton, S. Sabour, and N. Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

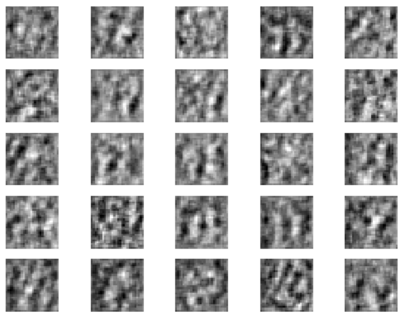
[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[6] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ArXiv e-prints*, Nov. 2015.

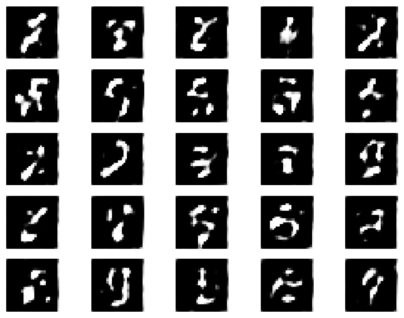
[7] S. Sabour, N. Frosst, and G. E Hinton. Dynamic Routing Between Capsules. *ArXiv e-prints*, Oct. 2017.

[8] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, Sept. 2014.

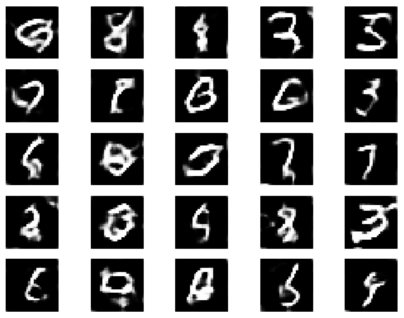
[9] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. *ArXiv e-prints*, Nov. 2013.



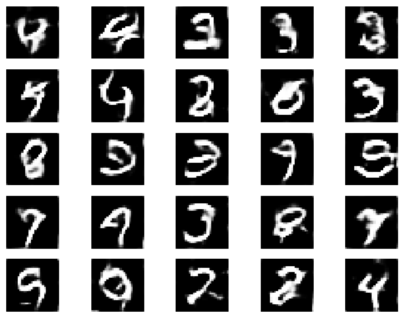
(a) Epoch 50



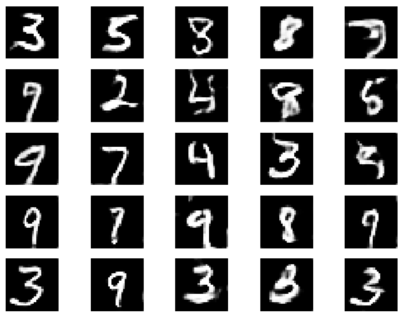
(b) Epoch 250



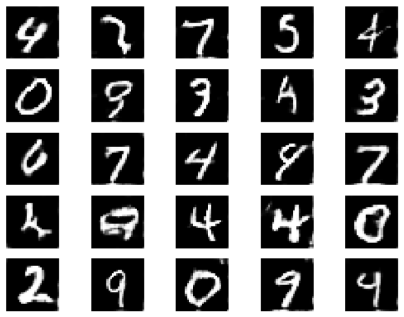
(c) Epoch 1400



(d) Epoch 1650



(e) Epoch 5250



(f) Epoch 9900

Figure 11: The images generated by the CapsuleGAN model over various epochs on the MNIST dataset