Exploring the Efficiency of Capsule Networks in GANs

Nitin Kishore Sai Samala
University of Massachusetts Amherst
nsamala@cs.umass.edu

Sruthi Chilamakuri
University of Massachusetts Amherst
schilamakuri@cs.umass.edu

Abstract

A GAN (Generative Adversarial Network) is a very popular machine learning technique created just two years ago that has found uses in almost every field. It has 2 components which are trained at the same time as adversaries in a minimax game: a generator that attempts to transform samples drawn from a prior distribution to samples from a complex data distribution with much higher dimensionality and a discriminator that decides whether the given sample is real or generated. Traditionally, GANS use CNNs but the internal data representation of a CNN doesn't capture important spatial hierarchies between simple and complex objects while Capsules are groups of neurons that are locally invariant and learn to recognize entities and output activation vectors that represent both the presence and their properties relevant to the visual task. Capsule networks were recently shown to outperform CNNs, we attempt designing a GAN using Capsule networks and employ several techniques devised to stabilize and make training more robust.

1. Introduction

Colloquially, GANs are known to be popular for generating very realistic fake images but a more fascinating use for gans is to learn features from data without any labels (unsupervised). They were initially proposed in [6] and have gone through several modifications and evolutions that it is now hard to keep track of them. The concept is fairly simple. The two components of a GAN, a generator and a discriminator are trained simultaneously so they both improve by competing against one another. GANs have a lot of promising research potential based on the wide range of applications they are employed in, such as face or font generation,painting [27],image completion, content-aware filling, synthetic data generation [24], drug discovery through generation of sample drug candidates, image generation from text description([4], [28], [20]), domain transfer(e.g. style-transfer, pix2pix, sketch2image)([17], [12], [10]), etc. There have been a lot of publications in this field making it a very active research problem.

Recently a new type of neural network called Capsule network [7] was introduced which makes use of dynamic routing by agreement between capsules [21] for learning. The cornerstone of deep learning's surge in popularity undoubtedly must be convolutional neural networks [15] but they do not exist without flaws. The convolutional layer is responsible for learning important high-level features as combinations of simpler features learnt from the lower layers. The setup of a CNN involves using a succession of convolution, non-linear activation and pooling layers repeatedly to reduce the spatial dimensions and the widen the scope of the higher-level neurons that detect higher order features. This gave results outperforming the state-of-the-art at the time albeit losing valuable information from images. The fact remains that orientation and relative spatial relation between components in an image are of less consequence to a CNN. The CNN's internal data representation doesnt capture spatial hierarchies between simple and complex objects effectively.

Contrary to producing an image from an internal representation of objects, the human brain deconstructs the visual stimuli into a hierarchical representation matching it with existing learned patterns and relationships. This approach is modeled by a Capsule network to preserve hierarchical pose (translation plus rotation) relationships between object parts. The intuition behind Capsule networks is that a model has to understand and discriminate between different views of the same object i.e., the representation should not depend on viewing angle. Capsule networks can achieve an understanding of 3D space and do it using less data than a CNN requires, which makes them a very desirable alternative.

The fundamental unit of this network is called a capsule. Instead of outputting a single scalar value like a neuron, a capsule outputs a vector that encapsulates not only presence but also information about the state of the detected feature under consideration [8]. The probability of detection of this feature is encoded as the length of the vector while its orientation gives us the state of the feature. This makes feature detection pose invariant and allows for more powerful representational capabilities. Currently Capsule Networks are

slower than most models and have been shown to do well on simpler datasets.

In this project, we attempt to take the evolution of GANs a bit further chronologically and aim to use Capsule networks to learn features from images. However, GANs are notorious for being difficult to train due to various factors such as minimax optimization, vanishing gradients causing unstable training, mode collapse, etc. Hence our goal is to implement several training hacks outlined in [22], [25], [1], [23], [26] to improve the fundamental stability of a "CapsuleGAN" and make training more robust. We also hope to generate better results on more complex image datasets than MNIST such as CIFAR-10 and celebrities face dataset for this project.

2. Related Work and Background

Over the past few years a lot of relevant progress has been made and different types of GANs were proposed. The first significant enhancement on vanilla GAN that went on to become a solid baseline to implement and compare future models with, is the Deep Convolutional GAN. Deep Convolutional GAN (DCGAN) puts more emphasis on using ReLu activations and Batchnorm layers while avoiding fully-connected hidden layers and pooling layers in the convolutions. This architecture was further improved in [22] allowing generation of better high-resolution images by addressing the issues of training stability and convergence. Some of these enhancements are what we hope to apply to our model as well. DCGANs do not suffer from mode collapse [19] when generating complex images similar to that of the CIFAR-10 dataset which propelled more widespread adoption of GANs in novel applications and implementations.

Zi Yi Dou, (2017) proposed a new framework to train discriminators dynamically based on the distance between generated samples and real samples, called Metric Learning based Generative Adversarial Network (MLGAN) [5]. MLGANs update the generator under the newly learned metric and avoid the instability issues caused by the sigmoid cross entropy loss function used in traditional GAN discriminators. This approach has been found to empirically increase the stability of training in GANs. Another modification to the loss function involves including the Wasserstein distance [2] which correlates with image quality. This distance between points from one distribution to another, replaces Jensen-Shannon divergence as an objective function that greatly improves training stability. This approach also improves convergence by offering a numerical value that interprets how well the parameters are tuned and thus eliminates the need to constantly evaluate the generated samples for signs of convergence. Our approach is to use Capsule Networks instead of CNN for discriminators in our GAN.

3. Methodology

A discriminative model involves evaluating the conditional probability of a target random variable Y given an observed random variable X, while a generative model entails evaluating the joint probability P(X,Y). We can interpret images as samples from a probability distribution. In a generative adversarial network, both the discriminator and the generator are competitors trained simultaneously in a minimax game. Initially, the generator samples a vector noise Z from a simple distribution, say Gaussian, and then up-samples this vector up to an image which is predominantly noise at first. As the generator keeps learning, the discriminator classifies the generated images and the real images. This result is propagated backwards in a feedback loop enabling the generator's distribution to get as close as possible to the distribution of real images while the discriminator is trained to identify the fakes. This process continues until the probability of classification of an image becomes 0.5, indicating that the generated image can pass for a real one.

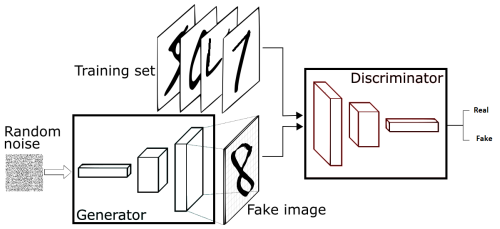


Figure 1: GAN architecture

3.1. Model

3.2. Capsule Network

Several components of the Capsule Network are described in detail in the subsequent sections.

3.2.1 Capsule Architecture

A capsule is the most basic unit of Capsule networks that takes in a weighted sum of output vectors from the capsules in the layer below and produces an output vector. For a capsule s_j the input is given by:

$$s_j = \sum_i c_{ij} \hat{u}_{j|i} \;\; where \;\; \hat{u}_{j|i} = W_{ij} u_i$$

$\hat{u}_{j|i}$ are the output vectors from capsules in the layer below obtained by computing dot product with the weight matrix.

c_{ij} is the coupling coefficient between capsule s_j and each of the i capsules in the previous layer. The coupling

coefficients sum to one and are the softmax probabilities of logits b_{ij}. Initial values for b_{ij} can all be zero indicating that all capsules have equal affinity to all parent capsules or can be initialized to values from a prior distribution that captures affinity between capsule i and parent j.

The output vector also known as the prediction vector v_j is computed using the following non linearity (a method termed squashing):

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

Once the prediction vector is computed, the logits b_{ij} are updated with the agreement: the product $v_j \hat{u}_{j|i}$. If the product is high, then the low level features detected by i capsules are important to parent capsule j and naturally the coupling coefficients b_{ij} must be increased proportionally to propagate relevant information to the parent through routing by agreement.

3.2.2 Routing by Agreement

The dynamic routing by agreement algorithm in [21] is presented below:

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 $\mathbf{v}_j$

Figure 2: Routing by Agreement. Sabour, Frosst et al."Dynamic Routing Between Capsules."(2017)

3.2.3 Loss Function

A separate margin loss is used for each of the k classes in the final layer of the Capsule network.

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

T_k is an indicator variable which is equal to 1 if object of class k exists in the image and 0 otherwise. m^+ and m^- are hyper-parameters and are set to 0.9 and 0.1 respectively. As mentioned in [21] the second term of the loss function concerning absent object classes helps prevent shrinking of the lengths of prediction vectors for these classes during initial stages of learning. Thus the total loss is the sum of k margin losses, one for each class.

3.3. GAN Architecture

Please refer Figures 3 and 4. [1]

[1]Code for this project will be maintained on https://github.com/snknitin/DeepfakeCapsuleGAN

GENERATOR:

Layer (type)	Output Shape	Param #
input_15 (InputLayer)	(None, 100)	0
dense_21 (Dense)	(None, 8192)	827392
reshape_9 (Reshape)	(None, 8, 8, 128)	0
batch_normalization_28 (Batc	(None, 8, 8, 128)	512
up_sampling2d_15 (UpSampling	(None, 16, 16, 128)	0
conv2d_22 (Conv2D)	(None, 16, 16, 128)	147584
leaky_re_lu_13 (LeakyReLU)	(None, 16, 16, 128)	0
batch_normalization_29 (Batc	(None, 16, 16, 128)	512
up_sampling2d_16 (UpSampling	(None, 32, 32, 128)	0
conv2d_23 (Conv2D)	(None, 32, 32, 64)	73792
leaky_re_lu_14 (LeakyReLU)	(None, 32, 32, 64)	0
batch_normalization_30 (Batc	(None, 32, 32, 64)	256
conv2d_24 (Conv2D)	(None, 32, 32, 3)	1731
activation_24 (Activation)	(None, 32, 32, 3)	0

Total params: 1,051,779
Trainable params: 1,051,139
Non-trainable params: 640

Figure 3: Generator summary

DISCRIMINATOR:

Layer (type)	Output Shape	Param #	Connected to
input_14 (InputLayer)	(None, 32, 32, 3)	0	
conv1 (Conv2D)	(None, 24, 24, 256)	62464	input_14[0][0]
leaky_re_lu_9 (LeakyReLU)	(None, 24, 24, 256)	0	conv1[0][0]
batch_normalization_26 (BatchNo	(None, 24, 24, 256)	1024	leaky_re_lu_9[0][0]
primarycap_conv2 (Conv2D)	(None, 8, 8, 256)	5308672	batch_normalization_26[0][0]
primarycap_reshape (Reshape)	(None, 2048, 8)	0	primarycap_conv2[0][0]
primarycap_squash (Lambda)	(None, 2048, 8)	0	primarycap_reshape[0][0]
batch_normalization_27 (BatchNo	(None, 2048, 8)	32	primarycap_squash[0][0]
flatten_3 (Flatten)	(None, 16384)	0	batch_normalization_27[0][0]
uhat_digitcaps (Dense)	(None, 160)	2621600	flatten_3[0][0]
softmax_digitcaps1 (Activation)	(None, 160)	0	uhat_digitcaps[0][0]
dense_17 (Dense)	(None, 160)	25760	softmax_digitcaps1[0][0]
multiply_7 (Multiply)	(None, 160)	0	uhat_digitcaps[0][0] dense_17[0][0]
leaky_re_lu_10 (LeakyReLU)	(None, 160)	0	multiply_7[0][0]
softmax_digitcaps2 (Activation)	(None, 160)	0	leaky_re_lu_10[0][0]
dense_18 (Dense)	(None, 160)	25760	softmax_digitcaps2[0][0]
multiply_8 (Multiply)	(None, 160)	0	uhat_digitcaps[0][0] dense_18[0][0]
leaky_re_lu_11 (LeakyReLU)	(None, 160)	0	multiply_8[0][0]
softmax_digitcaps3 (Activation)	(None, 160)	0	leaky_re_lu_11[0][0]
dense_19 (Dense)	(None, 160)	25760	softmax_digitcaps3[0][0]
multiply_9 (Multiply)	(None, 160)	0	uhat_digitcaps[0][0] dense_19[0][0]
leaky_re_lu_12 (LeakyReLU)	(None, 160)	0	multiply_9[0][0]
dense_20 (Dense)	(None, 1)	161	leaky_re_lu_12[0][0]

Total params: 8,071,233
Trainable params: 8,070,705
Non-trainable params: 528

Figure 4: Discriminator summary

4. Dataset

We have three datasets of interest in this project. The first one is MNIST, a large collection of black and white handwritten digit images, widely popular and used for generating baselines in many academic research papers. Baseline and results in [21] were computed on the MNIST dataset. MNIST consists of 60,000 training images and 10,000 test-

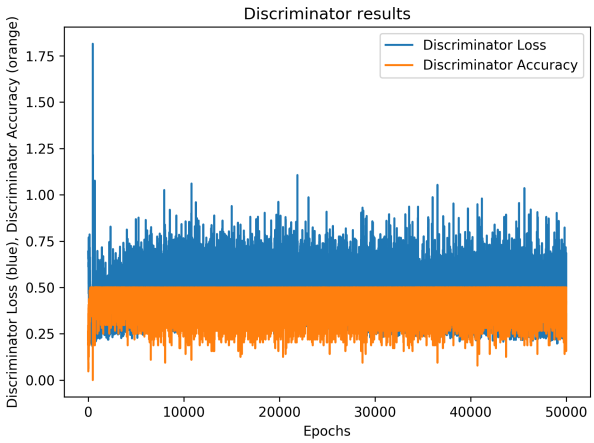
ing images.

Another dataset we will test our model on is the CIFAR-10 dataset which consists of 32x32 color images of 10 classes (airplane, automobile, bird, cat, deer, dog, horse, frog, ship and truck) with 6000 images per class. The dataset statistics are similar to MNIST with 50000 train and 10000 test images. CIFAR-10 is well recognized and has been used for computing baselines in several publications involving convolutional neural networks and computer vision. [14]

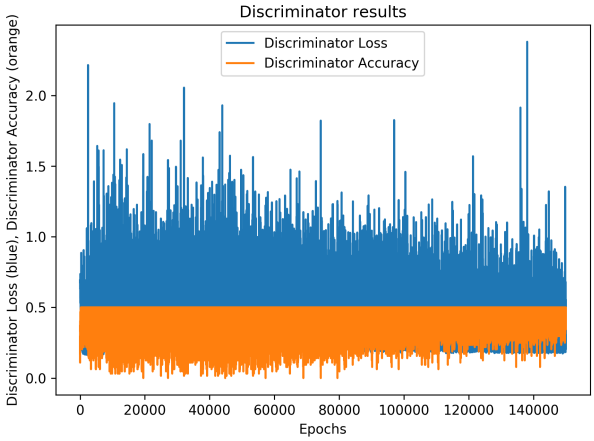
The third dataset is the large-scale CelebFaces Attributes (CelebA) dataset [16]. It contains 202,599 diverse celebrity face images that are cropped and aligned. Researchers at Nvidia recently used this dataset to progressively train a GAN by using low-resolution images first and gradually increasing image resolution and network size until they were able to generate very realistic faces [11]. We processed these aligned and cropped images to have a (32,32,3) shape to avoid complicating our architecture by having multiple generators for different dimensions. CelebA offers a challenge and it would be interesting to see how Capsule networks do on such a complex dataset.

5. Training techniques

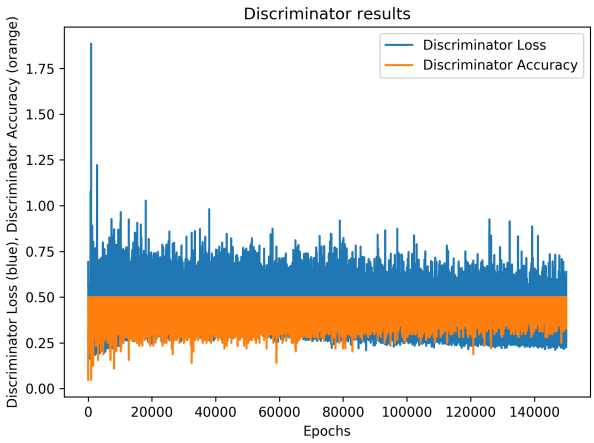
It is recommended to use three routing iterations for training Capsule networks. Training a GAN involves several moving parts which make it difficult to identify or validate whether the generator and discriminator work and changing these parts collapses the GAN. Hence we used several hacks and heuristics to stabilize the GAN game and avoid the usual suspects. To get a reasonable experience in training gans, we follow some steps outlined in [22] and normalize the images to be between -1 to +1. Normalization was done by subtracting and dividing the image pixel intensity values by 127.5 . The latent z variable given as noise was sampled from normal distribution instead of uniform distribution [26]. In addition to this, we also add noise to inputs and decay it over time. Batch statistics help the discriminator get stronger at the beginning of the adversarial game. As seen in the architecture, batchnorm is used in both the discriminator and generator with different batches for real and fake image. We don't mix a batch with fake and real images included in it. Another trick from [22] that we employed was, to make the labels soft and noisy for the discriminator, instead of a hard constraint of 1 for real and 0 for fake images, by giving a stochastic value to the fake labels and introduce a noise which implies that we flip the label with some probability to inject a constructive stochasticity into the network. Typically, using Relu activations or Max pooling leads to sparse gradients which aren't conducive to the stability of a GAN, especially for a generative model. We avert this problem by using Leaky Relu (with alpha = 0.03) and strided convolution when Upsampling and



(a) MNIST Training



(b) CIFAR-10 Training



(c) CelebA Training

Figure 5: Discriminator Losses and Accuracies

Downsampling, and have tanh as the final activation layer for the generator output. [19] prompted us to us Adam (with learning rate = 2e-4) since it is the most popular and stable choice among optimizers for GANs. The dimensions in the architecture of the generators and the discriminators have been designed to handle both the image dimensions we are dealing with, based on the dataset chosen.

Our code outputs a 5x5 grid of 25 generated images every 1000 epochs and saves the generator every 2000 iterations. We ran the models on NVIDIA Titan X GPUs on the gypsum cluster using Keras and TensorFlow. The MNIST model was executed for 50,000 epochs, while the Cifar-10 and CelebA models were executed for 150,000 epochs with a batch size being 64 for all. Figures 5 and 6 show the training of the generator and the discriminator over different number of epochs for each dataset.

6. Evaluation

Several methods have been proposed to evaluate GANs. In the beginning, Goodfellow et al. (2014) evaluated GANs using a comparison of the generated data sample and its nearest neighbors in the data. We want to identify improvements in GANs both in terms of training stability and quality of the samples generated so we initially considered using Inception Score as an evaluation metric [22] for the Cifar10 model.

Inception Score is computed by applying a pre-trained neural network to the generated samples and calculating the KL divergence between the conditional class distribution and the marginal class distribution at the final layer as described below.

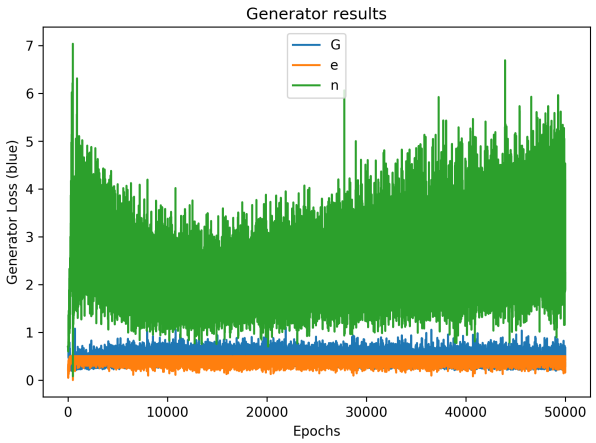
$$IS(G) = exp\,(\,\mathbb{E}_{x\sim p_g}\,[D_{KL}(\,p(y|x)\,||\,p(y)\,)]\,)$$

p_g is the probability distribution the generator tries to model. $p(y|x)$ is the conditional class distribution and $p(y)$ is the marginal class distribution of the generated sample with respect to the pre-trained network representing distribution p.

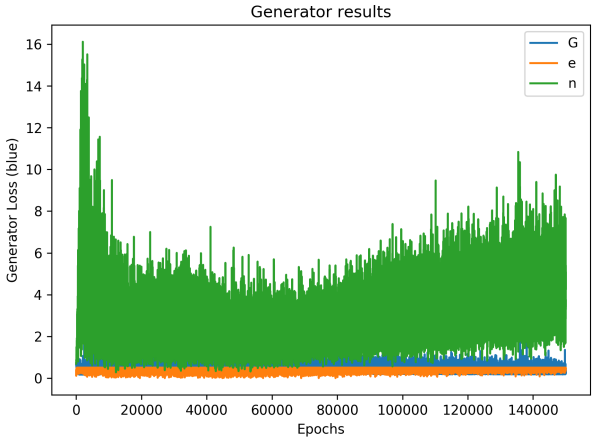
However since we cannot actually compute P_g but can only draw samples from the distribution, both the expectation term in the definition and P_g in $p(y) = \int_x p(y|x)\,p_g(x)$ will be computed using Monte Carlo approximation. We generate N samples and define $\hat{p}(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|x_i)$ and the approximated Inception Score as below.

$$IS(G) = exp\,(\,\frac{1}{N}\sum_{i=1}^{N} D_{KL}(\,p(y|x_i)\,||\,\hat{p}(y)\,)\,)$$

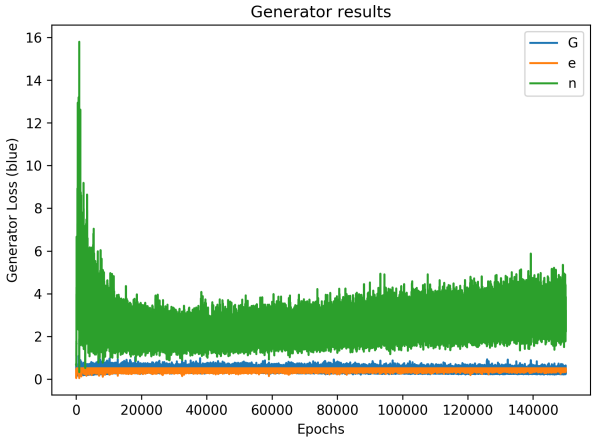
The Inception Score method of evaluation is not without flaws. Efficacy and pitfalls of Inception Score are explored in depth in [3]. One of the major disadvantages of this method is the reliance on a pre-trained network and the



(a) MNIST Training

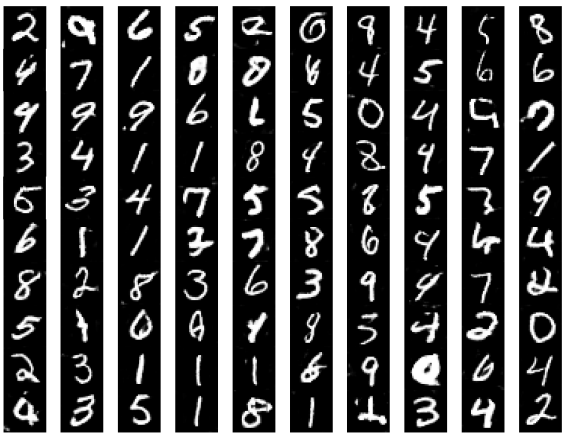


(b) CIFAR-10 Training

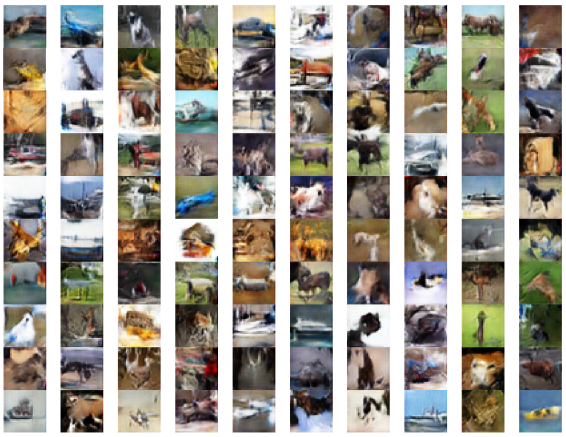


(c) CelebA Training

Figure 6: Generator Losses



(a) MNIST generated images (after 50,000 epochs)



(b) CIFAR-10 generated images (after 150,000 epochs)

(c) CelebA generated images (after 150,000 epochs)

Figure 7: Images generated by the CapsuleGAN

existence of a requisite dataset whose classes coincide with classes of the generated samples used to train the pre-trained network. Further, as demonstrated in [3], minor variations in the weights of the pre-trained network result in vastly different Inception Scores even though each of the disparate networks report the same classification accuracy. This is undesirable as we would want a good evaluation metric to be invariant under changes to factors that do not influence

Dataset	Evaluation method	Baseline	Our Result
MNIST	Semi-supervised classification Error rate	0.0702	0.0638
CIFAR-10	Inception Score	10.852±0.181 (for 50000 samples)	1.0015±0.004 (for 500 samples)
CelebA Faces	Human Evaluation(Discriminator accuracy)	NA	38.56%

Table 1: Results

the quality of the generated samples.

We computed the Inception Score for generated CIFAR-10 images using the default Tensorflow implementation of Inception v3, a network pre-trained using ImageNet. Due to the time constraint of this project and the number of epochs it took to train the generator to create good images of Cifar, we were unable to match the standard of generating 50000 samples to compare with the baseline from [3].

For datasets like CIFAR-10 and MNIST, semi-supervised classification can be done by providing raw pixel values to any general classification algorithm using Label Spreading algorithm from [29] with the outputs of our model as the unlabeled examples and some real labeled samples. For MNIST, the state-of-the-art results as per [22] and [13] for semi-supervised classification learning, is an error rate of 0.0702 when using 10,000 real labeled samples and 50,000 generated images. We used sklearn's LabelSpreading package which uses an affinity matrix based on the normalized graph Laplacian and soft clamping across the labels [18].

Apart from this, a qualitative analysis of the generated images should prove to be a better standard to judge the quality of the model. Generated CelebA images were assessed through human evaluation for lack of standardized methods to evaluate this dataset. We acknowledge that human evaluation is not only expensive but also subjective and therefore prone to high variance. Our evaluation process involved a diverse group of 15 volunteers observing specific sets of generated inputs. Individual images are of size 32x32 which would be too tiny and blurred to discriminate. So, we sampled random noise and generated realistic fake images of the CelebA dataset and presented them in a 10x10 grid (100 images) with binary choice questions, varying the option choices, alternating with equal batches of real and fake grids. The images from the CapsuleGAN were deemed realistic 38.56 times on average out of 50 (where a 50% discriminating accuracy implies that you cannot tell the fake images apart from real ones).

7. Results

Please refer Figure 7 and Table 1 for results

8. Conclusion and Future Work

In our project we have shown that the capsule network works well in a GAN setting to generate realistic fake images or samples of not just MNIST but also more complex datasets like Cifar10 and CelebA. We observed more crisp and detailed images generated with fewer epochs of training when compared to regular CNN-GANs, especially in case of CelebA. We avoided mode collapse and vanishing gradients using mini-batch training and LeakyRelu respectively. With a bit more time, we could have had better evaluation results for some cases to make a valid quantitative judgment. Most of the GAN training techniques don't have an explanation attached, as to why they work. They are simply considered as axiomatic instructions for a stable GAN training experience, which helped us train our model. Appealing properties like requiring less training data and the ability to capture hierarchical pose relationships between object component, makes capsule networks a good alternative to CNN as a discriminator in GANS.

Future work includes using Capsule networks for the generator as well.Evaluating the quality of images is very subjective and therefore it is hard to reach a general consensus on what is considered good. However, a GAN is ultimately meant to learn the hidden generative distribution so we can focus on measuring how close the model distribution is to the real distribution. Alternatively generative adversarial metric (GAM) [9] can be considered for evaluation. The idea behind this pairwise metric is to swap discriminators of two GANs and train the generators against their new counterparts in the same minimax fashion. We can compare our model with an improved DCGAN using the GAN values to see if our model outperforms it.One extension that we identified as a potential future project would be to generate a large number of good quality face images and use them in a deep fake to embed the new character in an existing video since capsule networks can achieve a better understanding of 3D space.

References

[1] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] S. Barratt and R. Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

[4] A. Dash, J. C. B. Gamboa, S. Ahmed, M. Z. Afzal, and M. Liwicki. Tac-gan-text conditioned auxiliary classifier generative adversarial network. *arXiv preprint arXiv:1703.06412*, 2017.

[5] Z.-Y. Dou. Metric learning-based generative adversarial network. *arXiv preprint arXiv:1711.02792*, 2017.

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

[7] G. Hinton, N. Frosst, and S. Sabour. Matrix capsules with em routing. 2018.

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

[9] D. J. Im, C. D. Kim, H. Jiang, and R. Memisevic. Generative adversarial metric. 2016.

[10] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

[11] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[12] T. Kim, M. Cha, H. Kim, J. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*, 2017.

[13] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[14] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[17] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. *CoRR, abs/1703.07511*, 2017.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[20] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[21] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

[22] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[23] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883, 2016.

[24] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, page 6, 2017.

[25] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár. Amortised map inference for image super-resolution. *arXiv preprint arXiv:1610.04490*, 2016.

[26] T. White. Sampling generative networks: Notes on a few effective techniques. *arXiv preprint arXiv:1609.04468*, 2016.

[27] R. A. Yeh, C. Chen, T. Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5485–5493, 2017.

[28] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *IEEE Int. Conf. Comput. Vision (ICCV)*, pages 5907–5915, 2017.

[29] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.