CapsuleGAN: Using Capsule Network as Discriminator and using ClassCaps as input of Generator in GAN

Rahul Kishorbhai Pipaliya
Student ID: 1153308
Dept. of Computer Science
Lakehead University
Thunderbay, Ontario
Email: rpipaliy@lakeheadu.ca

***Abstract*—The field of image processing has seen massive growth in the last decade, the Main models being Convolution Neural Nets(CNNs) and Generative Adversarial Networks(GANs), which are based on CNN. Many models have been based on CNN like AlexNet, ResNet, DenseNet, etc. However, a few years ago, Geoffery Hinton introduced capsule Network(CapsNet), and it shows the state-of-the-art performance for the baseline dataset of MNIST. In this paper, I am using Capsule Network as a Discriminator and Class Capsules as input to the Generator to generate new Images, and I will then use a DVM-Car dataset to generate new car images using the designed Model.**

***Index Terms*—CapsNet, GANs, Caonvolutional Neural Networks, Capsules**

I. Introduction

The first-ever convolution neural network model was LeNet [9], introduced back in the 90s. It was a breakthrough in the area of image classification. The CNNs got popular after AlexNet [8] acquired state-of-the-art classification results in the ImageNet dataset in 2012. The main reason for that was that it introduced Dropout as regularization, and also it uses ReLU as an activation function. After AlexNet, CNNs became very popular. We can see VGGNet, which had a deeper network to achieve even better results. Then there has been DenseNet [16] which uses densely connected blocks. Around the same time, Generative Adversarial Network(GAN) [10] was also being introduced in 2014. GANs had such a unique concept of Generator and Discriminator models working against each other to generate New Images. The Generator will generate fake data using feedback from the Discriminator. DCGAN [12] is the famous successor of the original GAN model, which performed very well using the CIFAR-10 Dataset.

Recently in 2017, Geoffery Hinton and his colleagues published the paper named "Dynamic Routing Between Capsules" [14], which introduced using capsules and dynamic routing between capsules instead of Pooling used in CNN. Authors have argued about many flows of CNNs, mainly about its subsampling, which loses the spatial relationships between the different objects in the image and about "ViewPoint Invariance" – the ability to make the model invariant to changes in viewpoint. CapsNet is the state-of-the-art model for the MNIST dataset to date.

In this paper, First, I will discuss how Convolutional neural network(CNNs) works and also I'll discuss some of the issues with CNNs, mainly how routing works in CNNs and the advantage of using Capsule Network over CNNs, and also I will discuss a few papers on GANs that have used CapsNet as Discriminators. Then I will discuss the Methodology that I will be using.

II. CNNs

Convolutional Neural Networks are composed of 2 Main parts, Feature Learning and Classification[Fig 1.]. Convolutional layers, pooling layers, fully connected layers, and flattening layers are used for the whole process. A nxm kernel (n ¿ 0 and m ¿ 0) automatically searches the input picture during convolutions to extract features. The picture is imposed on the filter, which is then moved over the image depending on a stride value to build a feature map. The feature map will be smaller as the stride increases. When the stride is higher than or equal to 2, the convolution loses certain picture characteristics. Various distinct kernels are employed to obtain several feature maps in order to keep as many features as feasible. ReLU is used shortly after the convolutional step to increase nonlinearity and minimize computational model complexity. The feature map is pooled to ensure that the CNN detects the same object in pictures of various shapes while simultaneously reducing the model's memory needs. It incorporates spatial invariance into CNNs, which ultimately proves to be one of their primary flaws after that pooled feature map is flattened into a column matrix to feed it to an Artificial neural network (ANN) which is made of Input, hidden, and output layers. We use ANN to get the final classification output.

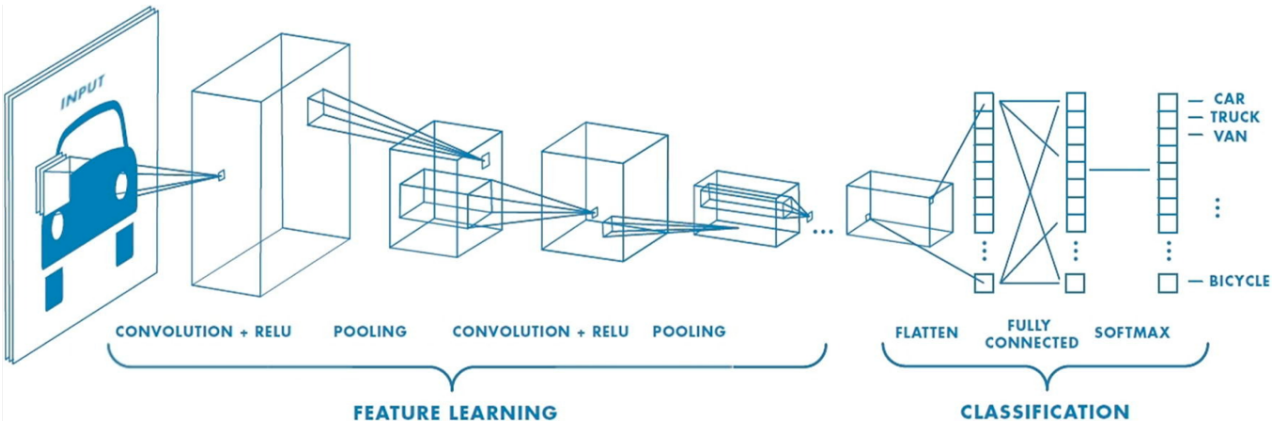

Fig. 1: Convolutional Neural Network (CNN) Architecture

Problem with CNNs

CNNs are invariant because they are not equivariant and so lack equivalence since they do not recognize the posture, texture, and deformations of an image or sections of an image because of Pooling Operation. Image reconstruction in CNN is much more difficult than reconstruction in CapsNet due to max pooling. Also, because the Pooling Operation loses a lot more information(We can see it in [Fig. 2]), it requires a lot more training data. And CNNs are more vulnerable to adversarial attacks like pixel perturbations [17], which result in incorrect classifications.

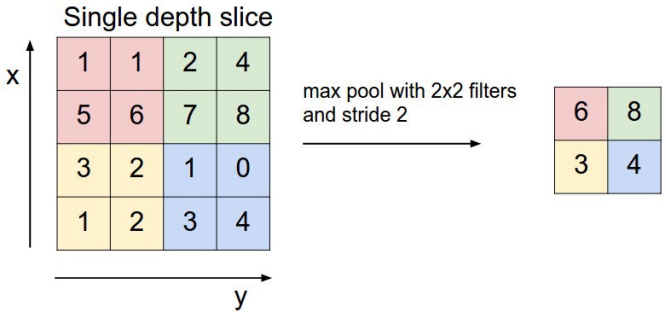

Fig. 2: A simple illustration of Max Pooling [3]

Another example given by [2] is both fundamental and non-technical. Consider a face [Fig. 3]. What exactly are the features? The face is oval, with two eyes, a nose, and a mouth. For a CNN, the sheer existence of these objects can be a powerful signal that the picture contains a face. A CNN is unconcerned about the orientational and relative spatial connections between these components.

It is critical to understand that higher-level characteristics integrate lower-level data as a weighted sum: activations from the previous layer are multiplied by the weights of the neuron in the next layer and combined before being transmitted to activation nonlinearity. There is no pose (translational and rotational) link between basic features that make up a higher-level feature in this design. To address this issue, CNN

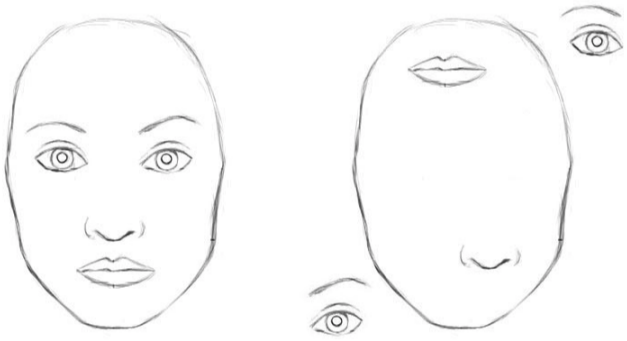

Fig. 3: Face Features Example [2]

employs max pooling or consecutive convolutional layers, which lower the spatial dimension of the data flowing through the network; thus, CNN sees all the features of the face and does not consider its pose and classifies it as a face.

III. CapsNet

The idea of capsules was first introduced in 2011 by Hinton et al. and his colleagues [4]. In this Paper they mentioned that Artificial neural networks should use local ”capsules” that perform some complicated internal computations on their inputs and then encapsulate the results of these computations into a small vector of highly informative outputs. Each capsule learns to recognize an implicitly defined visual entity over a limited domain of viewing conditions and deformations, and it outputs both the probability that the entity is present within its finite domain and a set of ”instantiation parameters”.

Assume a capsule recognizes a face in a picture and produces a 3D vector with a length of 0.99. Then we begin to move the face over the picture. The vector will spin in its space, depicting the changing condition of the discovered face, but its length will remain constant, indicating that the capsule is still confident it has identified a face. Hinton refers to this as activities equivariance: neuronal activity alters when an

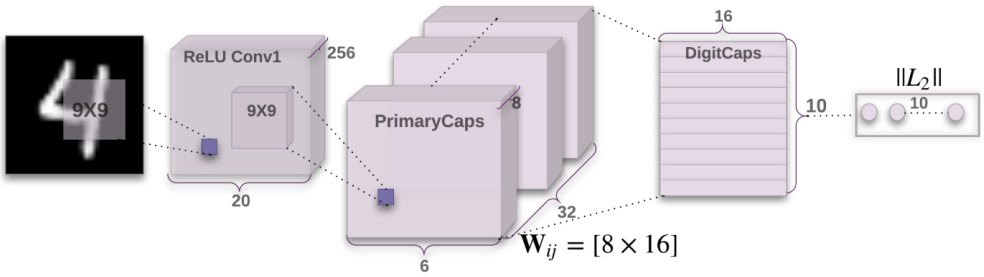

Fig. 4: CapsNet Encoder [14]

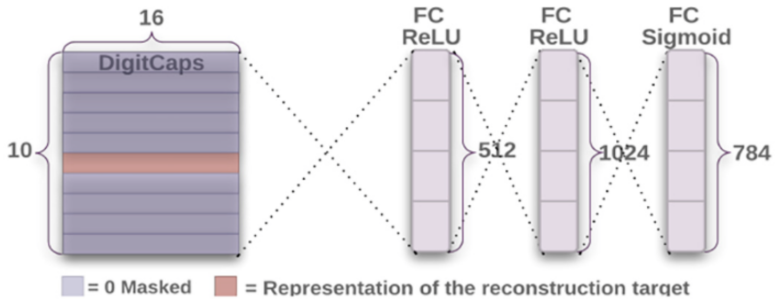

Fig. 5: CapsNet Decoder [14]

Capsule vs. Traditional Neuron			
Input from low-level Capsule / Neuron		vector(u_i)	scalar(x_i)
Operations	Linear/Affine Transformation	$\hat{u}_{j\|i} = W_{ij} u_i$	-
	Weighting	$s_j = \sum_i c_{ij} \hat{u}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Summation		
	Non-linearity Activation	$v_j = \frac{\|\|s_j\|\|^2}{1 + \|\|s_j\|\|^2} \frac{s_j}{\|\|s_j\|\|}$	$h_j = f(a_j)$
Output		vector(v_j)	scalar(h_j)

TABLE I: Difference Between Capsule and Traditional Neuron

item in the vision "moves around the manifold of conceivable appearances." At the same time, the detection probabilities stay constant, which is the sort of invariance we should strive for rather than the type provided by CNNs with max pooling. We can see the difference between Capsule and Traditional Neuron in the [Table 1]

So finally, this study led to the vision of Capsule Network, which was introduced in 2017 by Hinton et al. and his colleagues [14]. CapsNet is made of mainly two parts: Encoder[Fig. 4] and Decoder[Fig. 5]. The Encoder is a network of different capsules. Capsules are groups of different neurons that generate vectors representing each component's location and orientation in the image. The Decoder is just a bunch of Fully Connected layers to reconstruct the original image based on the output of Encoder. A simple CapsNet architecture is shallow with only two convolutional layers and one fully-connected layer. The first convolutional layer converts pixel intensities to the activities of local feature detectors that are then passed as inputs to the primary capsules, a group of convolutional layers. then instead of Pooling, CapsNet uses something called "routing-by-agreement." This parallel attention mechanism allows each capsule at one level to attend to some active capsules at the level below and ignore others, and using this technique; the authors were able to classify overlapping digits also of the MultiMNIST test dataset.

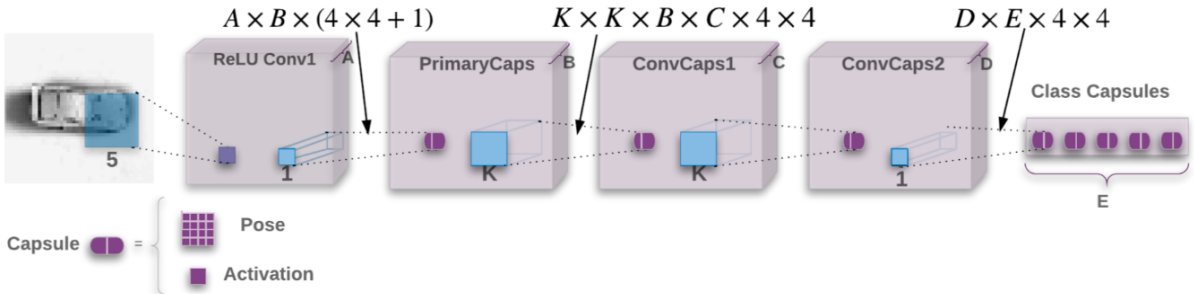

Fig. 6: Discriminator Architecture in CapsGan as Proposed in [15]

IV. Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) consist of two networks: Discriminator and Generator, playing minimax game with each other. Generative modeling is an unsupervised learning task in machine learning that involves discovering and learning the regularities or patterns in data. It can be used to generate or output new examples that plausibly could have been drawn from the original dataset. The GAN architecture was first described in the 2014 paper by Ian Goodfellow, et al. [10]. A standardized approach called Deep Convolutional Generative Adversarial Networks, or DCGAN, that led to more stable models was later formalized by Alec Radford, et al. in the 2015 [12]

The generator tries to minimize the following function while the discriminator tries to maximize it:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

Where, $D(x)$ is the discriminator's estimate of the probability that real data instance x is real. $E_x$is the expected value over all real data instances. And $G(z)$ is the generator's output when given noise z.

A. CapsNet in GANs

Not many researchers have tried to use CapsNet in the GANs. The author of [15] has compared Capsule Network GAN with DCGAN. The author used CapsNet model as a discriminator, and they have used MNIST and Small-NORB datasets for this comparison. The proposed discriminator architecture[Fig 6.] consists of an initial Convolutional Layer followed by a Primary capsule and two more convolutional capsule layers, and the final capsule layer consists of one capsule per output class. And they have used Generator from DCGAN. Using CapsNet makes it hard to fool Discriminator, and in terms, the Generator will try to generate a better image.

Another proposed method stated that it is necessary to keep the number of parameters low in the CapsNet Discriminator because CapsNet is very hard to fool, and it might even fail Generator entirely or make them struggle so much. also authors suggested using margin loss instead of the convolutional binary cross-entropy loss for the training. Then authors compared the results of convolutional GAN and Capsule GAN using MNIST and CIFAR-10 dataset, and they concluded that the error rate was low in Capsule GAN compared to convolutional GAN.

In another paper [13], the Author compared Capsule Network GAN with DCGAN and Vanilla GAN. The Author used the original capsnet encoder as a discriminator but changed the digit caps layer a little bit; the Author uses leakyRelu instead of the Squash function in the DigitCaps layer and then flattening the results 16x10(160) neurons and then finally connected with one sigmoid output layer. And Author is using the same generator model as DCGAN. The Author got results favorable in capsule network gan on MNIST dataset.

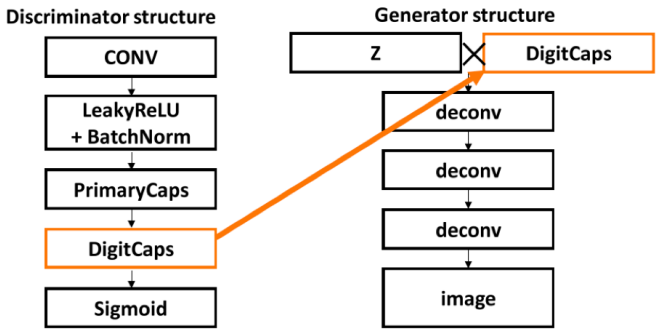

Fig. 7: Using Digitcaps layer into Generator [11]

Most interesting paper was by Kanako Marusaki and Hiroshi Watanabe [11]. They proposed Capsule GAN that uses Capsule Network in both the discriminator and the generator architecture of GAN. They used two approaches for that. One is to use the DigitCaps layer of the discriminator for the generator[Fig. 7]. DigitCaps layer is the output layer of the Capsule Network. The other is to use Capsule Network in the generator, such as CNNs. They used the Inception Score to measure the quality of generated images. The first approach

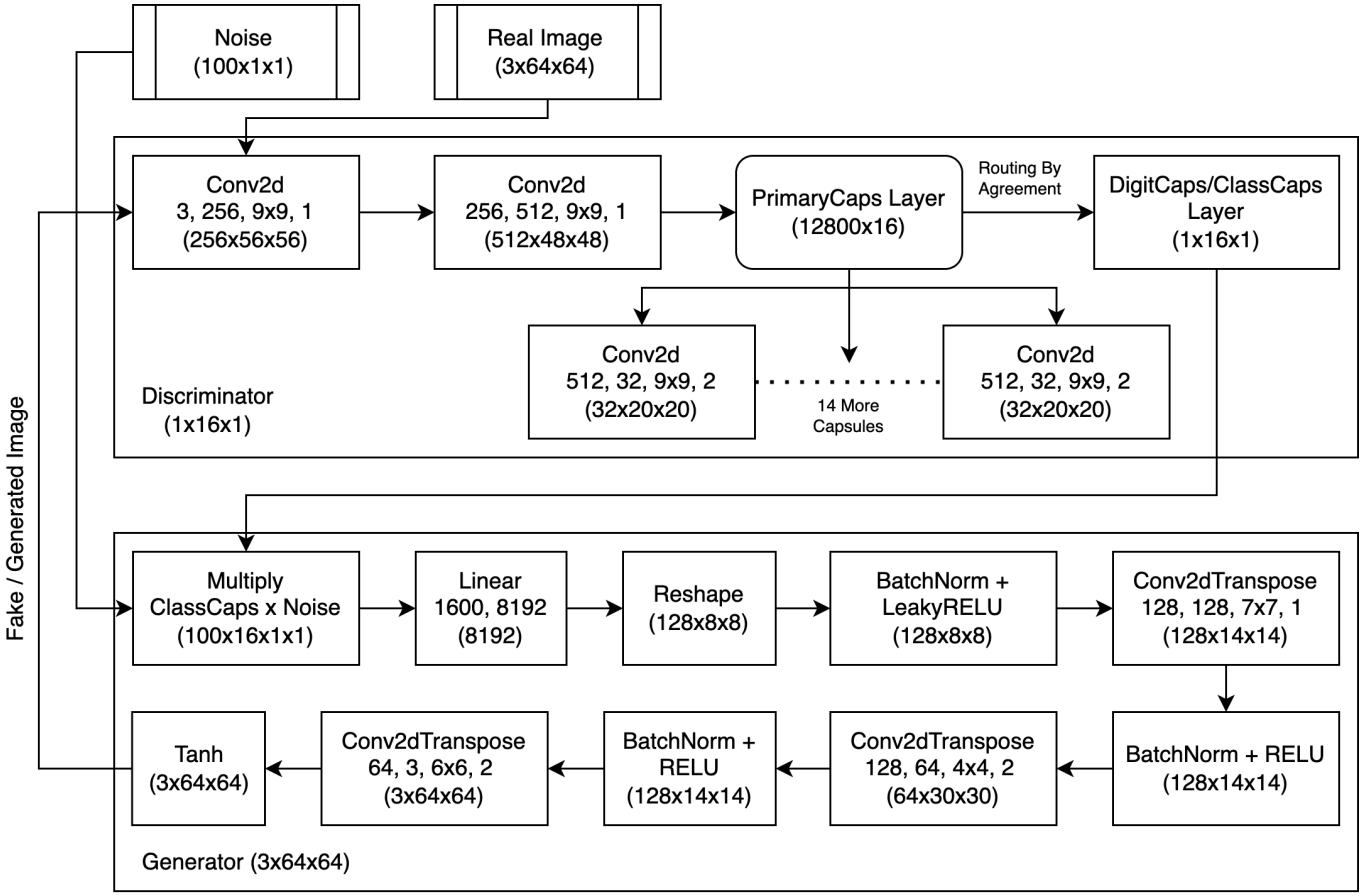

Fig. 8: Proposed GAN Model with each layer's input and output

showed promising results in the CAT dataset with three color channels. The second approach showed good results in MNIST and FashionMNIST, which uses only black and white colors. In this Paper, they have used a vanilla capsule network, though.

V. Methodology

I am taking inspiration from Kanako Marusaki and Hiroshi Watanabe and using CapsNet as Discriminator and using the classCaps layer as Input to the Generator along with noise[Fig. 8]. I'm using a slightly modified version of the DCGAN generator with the number of filters changed and added one flatten layer to flatten classCaps Input. Also, I'm using a more complex network as I'm using three channels compared to only one used by them. And finally, I am using Margin Loss instead of Binary Cross Entropy.

As a dataset, I'm using the DVM Car dataset [5], a Large-Scale Dataset for Automotive Applications. This dataset is divided into mainly two parts,

- Sale and Specification data
- Image data

I'm more interested in Image data, so I'll use that. It consists of 1,451,784 images from 899 UK market car models from different angles of cars. All of these images are 300x300 resolutions with the background removed.

Fig. 9: Front View

Fig. 10: Front Side View

As this dataset combines all images in one large zip file,

Fig. 11: Back View

Fig. 12: Side View

the first step was to separate all different angles so that I could train them separately. I focused on Four main angles: Front, Back, Side, and Front Side View. After separating these images into different categories, I had different sets of pictures for each class. The front has 240,502 Images, the Back has 231,846 Images, the Side has 195,258 Images, and Front Side has 139,809 Images. Then I had to resize all of them to 64x64 to match the dimension of my Model's requirements. I chose the size of 64 because of resource limitations, and also it gave me enough room to play with different hyperparameters.

For training, I chose a batch size of 128. First, I'm feeding the images to the Discriminator with the class label as true and get the ClassCaps which returns 1x16 (Vector containing Features that represent a car at a particular angle) and calculate discriminator loss using Margin Loss. I then took these vectors and multiplied them with the noise. I'm using the noise of size 100. so I have the final Input of size 100x16 to feed to the Generator. Now in Generator, I'm using one fully connected layer with an output size of 128x8x8(8192), then there is one reshape layer to reshape the feature map to 128x8x8. I'm using BatchNorm and LeakyReLU before feeding it to the TransposedConv2d followed by BatchNorm and ReLU 2 more times, and finally, Tanh activation function to keep the values between (-1,1), and at the end, I have Generated Image of size 3x64x64. Then I feed these generated Images to Discriminator and calculate Margin Loss for Generator. And after all these, there will be BackPropogation using Adam Optimizer with a learning rate of 2e-3.

I'm training this network for two epochs for each car view angle. And I'm also training DCGAN on all different car view angle images for three epochs. And at the end, I'll compare the results for both of them.

VI. Results and Comparisons

After complete training, these are the results; you can see left side CapsuleGAN generated images, right side DCGAN generated Images and real images at the bottom. I am using the Frechet Inception Distance (FID) score metric as a GAN performance measurement of GANs.

Car Angle	CapsuleGAN	DCGAN
Front	**120.20968**	147.08056
Side	127.76216	**121.91312**
Back	170.33481	**151.51725**
Front Side	**233.90548**	322.43786

FID Score comparison for CapsuleGAN and DCGAN

Fig. 13: Front View Generated Images

Fig. 14: Front Side View Generated Images

Fig. 15: Back View Generated Images

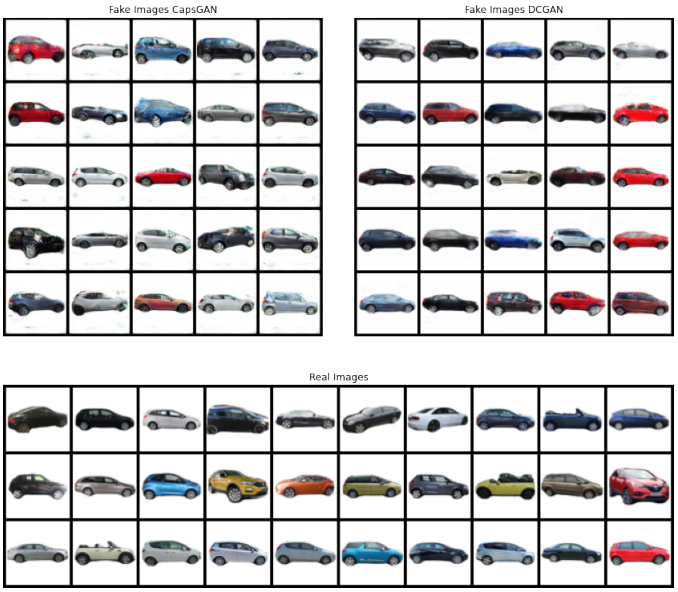

Fig. 16: Side View Generated Images

Now the reason for such a large FID score lies in how FID score works. As we can see from FID scores, CapsuleGAN works best on the Front and Front Side views of the cars, and DCGAN works best on the Side and Back views of the cars. So FID score compares extracted feature map of the Original Image and the Fake Image from the Inception Model. This dataset has different car angles, but they are "PREDICTED," not original. So we can see that there are some images in all four angles dataset which does not belong to that class; thus, there is significant noise in data separation. and that's why when I calculated the FID score, it tried to match even wrong angle image with correctly generated Image resulting into higher FID score.

VII. Discussion and Conclusion

Capsule Network has the potential to overcome CNNs, but it still needs some advanced research. If we don't compare FID scores and just look at how images look, CapsuleGAN-generated images are slightly better than DCGAN-generated images. Also, CapsuleGAN learns the distribution quicker than DCGAN; we can see that by looking at the results step by step.

I am training Front View of 240,502 cars images at a batch size of 128, so the total steps in one epoch will be about (240,502 / 128 = 1879 steps). I compared results at every 50 steps, and I realized that CapsuleGAN started to generate images representing more of Car Front View in just 150 steps compared to DCGAN, which took 500 steps. We can clearly see this in [Fig. 17,18,19,20].

Fig. 17: CapsuleGAN at step : 150

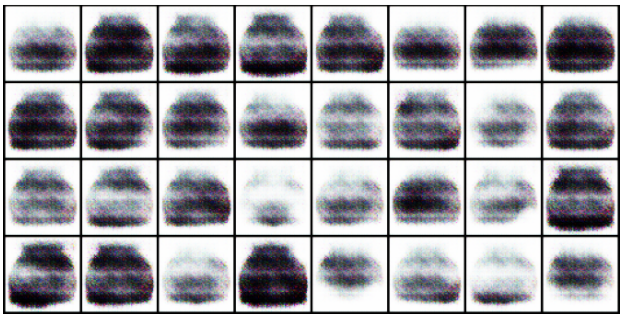

Fig. 18: DCGAN at step : 150

Fig. 19: CapsuleGAN at step : 500

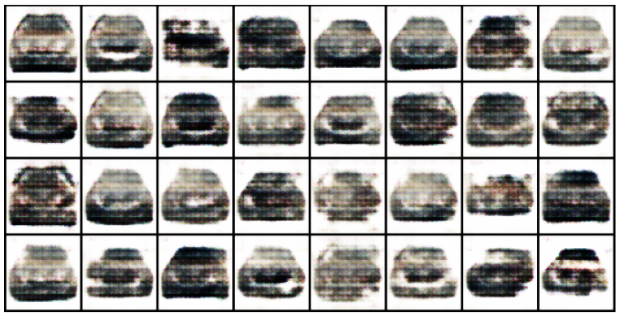

Fig. 20: DCGAN at step : 500

By noticing this, I tried to train the model with only 5000 random images to see how it performed, and to my surprise, it performed reasonably well. I trained the CapsuleGAN for 40 Epochs with the same rest settings.

Fig. 21: CapsuleGAN on only 5000 Images after 40 Epochs

After this, I even trained CapsuleGAN for 50 Epochs with only 1000 images which is a very low amount of data, and still, it was able to generate results that represent the Front View of the car. But if we can train it for more epochs, it might give us even better results.

Fig. 22: CapsuleGAN on only 1000 Images after 50 Epochs

As we discussed earlier, CapsNet does not need much data to train. So we can say that even if we have less data, we can train a CapsuleGAN on it to get some compelling results. And that's what we can see here in action. They are not perfect by any means yet.

VIII. Future Work

Future work definitely will be to correctly separate the dataset into different viewpoints because this dataset has huge potential to perform better on this model. And I can even try different loss functions with different Generator model like presented in boolGAN [6] is a modified version of DCGAN Generator where they added one extra TransposedConv2D layer to convert the image to 3x128x128 and then apply another Conv2D to reduce it to 3x64x64 and they used Wasserstein Loss function instead of BCE or Margin Loss, and they got a pretty good result. Also, I can even try this model on cars196 dataset [7] [1], which is another complex car dataset with different background.

References

[1] https://ai.stanford.edu/˜jkrause/cars/car_dataset.html. https://ai.stanford.edu/ jkrause/cars/car$_{d}ataset.html.(Accessedon04/19/2022)$.

[2] Understanding hinton's capsule networks. part i: Intuition. — by max pechyonkin — ai³ — theory, practice, business — medium. https://medium.com/ai (Accessed on 04/18/2022).

[3] Why pooling is not the answer to every problem — by abhishek chatterjee — medium. https://medium.com/@imdeepmind/why-pooling-is-not-the-answer-to-every-problem-6ad23a8d48f0. (Accessed on 04/18/2022).

[4] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6791 LNCS(PART 1):44–51, 2011.

[5] Jingming Huang, Bowei Chen, Lan Luo, Shigang Yue, and Iadh Ounis. DVM-CAR: A large-scale automotive dataset for visual marketing research and applications. (1), 2021.

[6] Dong Hui Kim. Deep Convolutional GANs for Car Image Generation. 2020.

[7] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[9] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Ha. LeNet. *Proceedings of the IEEE*, (November):1–46, 1998.

[10] Chongxuan Li, Kun Xu, Jun Zhu, and Bo Zhang. Triple generative adversarial nets. *Advances in Neural Information Processing Systems*, 2017-December:4089–4099, 2017.

[11] Kanako Marusaki and Hiroshi Watanabe. Capsule GAN Using Capsule Network for Generator Architecture. 2020.

[12] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–16, 2016.

[13] Kruthi N Raj. Capsule Network GAN vs . DCGAN vs . Vanilla GAN for Apparel Image Generation. pages 1947–1971, 2021.

[14] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *Advances in Neural Information Processing Systems*, 2017-December(Nips):3857–3867, 2017.

[15] Raeid Saqur and Sal Vivona. CapsGAN: Using Dynamic Routing for Generative Adversarial Networks. *Advances in Intelligent Systems and Computing*, 944(Nips):511–525, 2020.

[16] Mary Steen, Soo Downe, Nicola Bamford, and Leroy Edozien. DenseNet:Densely Connected Convolutional Networks arXiv:1608.06993v5. *Arxiv*, 28(4):362–371, 2018.

[17] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.