Semantic Image Inpainting with a Novel Twist

Ling He
Columbia University
lh2916@columbia.edu

Gerardo Antonio Lopez Ruiz
Columbia University
gal2140@columbia.edu

Andrea Navarrete Rivera
Columbia University
an2886@columbia.edu

***Abstract*— Generative adversarial networks were introduced by Ian Goodfellow et al. back in 2014 [5]. Despite the big success these networks currently have, it is particularly challenging to inpaint images when the missing regions regarding said image are very large. We are proposing a different approach to improve the accuracy: searching in the latent image manifold the closest encoding of the corrupted image and then passing it through a DCGAN to generate the missing content. Our experiments show promising results.**

I. INTRODUCTION

While classical inpainting or completion focuses on correcting false visual data in corrupted images based on a single image, semantic inpainting [14] infers arbitrary large missing regions in images based on the context of surrounding pixels. This task is significantly more difficult than classical inpainting due to the prediction of high-level context. Nevertheless, its usefulness is evident in applications such as the restoration of damaged paintings [4] where there are large regions missing and the semantics of images play an important role in the inpainting.

Our objective is to predict the detailed content of a large missing region using semantic inpainting. Prior works on image inpainting such as total variation (TV), low rank (LR), and PatchMatch (PM), tend to focus on hand-designed matching and editing, and usually work better for correcting small holes in images, but machine learning-based methods [11], [19], [7], [13] have shown more favorable results.

Instead of using a learning-based method like the Context Encoder (CE) by Pathak et al. [14], which takes in a mask indicating the missing regions, and predicts the missing content using a trained neural network that encodes the context information, we decided to use a generative adversarial network (GAN) model-based approach [5], [16].

General adversarial networks simultaneously train two neural networks, a discriminator and a generator. The generator produces images as real as possible to fool the discriminator. In order to do this, its input is a matrix mapped from a random vector z that is sampled from a prior distribution $p(z)$ [20], it then gives us a fake image that tries to resemble a real one by determining the features that make an image true. The discriminator takes real images and fake images created by the generator as inputs and it outputs the probability that said image is real. We can see then that each network have opposite objectives, whilst the generator wants to create images that fool the discriminator, the discriminator tries to improve its capacities of discerning between fake and real images and thus, preventing being fooled by the generator.

We think that semantic inpainting can be regarded as an image generation problem with prior knowledge. With the recent advances in generative modeling, we have the hypothesis that utilizing GANs to train a generative model on uncorrupted images and thus inferring the missing regions from said corrupted images will generate better results.

We evaluated our DCGAN model using the three data sets: CelebA [10], SVHN [12] and Stanford Cars [8]. The experiment on CelebA achieved the most favorable results.

II. SUMMARY OF THE ORIGINAL PAPER

A. Methodology of the Original Paper

Yeh et al.s paper "Semantic Image Inpainting with Deep Generative Models"[21] is the original paper from were we are basing our work. The authors consider semantic inpainting as an image generation problem with constraints. The hypothesis from this paper is that if G is capable of representing a image, then an image that is not part of the corrupted data will not lie on the learned encoding manifold. Because of this, they aimed to obtain $\hat{z}$, the 'closest' encoding of the corrupted image that is constrained to the manifold.

Afterwards, the encoding is used to reconstruct (paint) the the corrupted image using the trained generator. The 'closest' is defined by a weighted context loss to condition on the corrupted image, with a prior loss to penalize unrealistic images.

The context loss is given by:

$$\mathcal{L}_c(z|y, M) = \|\mathcal{W} \cdot (G(z) - y)\|$$

Where:

$$\mathcal{W}_i = \begin{cases} \sum_{j \in N(i)} (\frac{(1-M_i)}{|N(i)|} & \text{if } M_i \neq 0 \\ 0 & \text{if } M(i) = 0 \end{cases}$$

and the prior loss is given by:

$$\mathcal{L}_p(z) = \lambda log(1 - D(G(z)))$$

Thus, the construction for this encoding is given by:

$$\hat{z} = \underset{z}{\operatorname{argmin}}\{\mathcal{L}_c(z|y, M) + \mathcal{L}_p(z)\}$$

Where $\mathcal{L}_c$ is the context loss, y the input corrupted image and M the mask. $\mathcal{L}_p$ is the prior loss (that penalizes unrealistic images) [20].

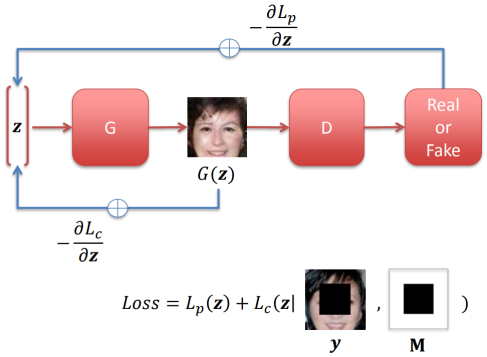

Fig. 1. Image obtained from [20]. Yeh et al.'s proposed framework for updating z to find $\hat{z}$.

B. Implementation Details of the Original Paper

The paper used the DCGAN model architecture from Radford et al. [17]. For the generative model it took a random 100 dimensional vector that originates from a random vector with values between [-1,1] and generated a 64 x 64 x 3 image using transpose convolutional layers and a final tanh activation function. The discriminator input consisted in a 64 x 64 x 3 image with a series of convolutional layers next. The output for the discriminator was a two class softmax. Furthermore, they utilized a training procedure found in [16], used Adam for optimization and data augmentation of random horizontal flipping on the training images [20].

For inpainting, they found $\hat{z}$ in the latent space using back-propagation, terminating after 1500 iterations. The inpainting is the one given by Perez et al. in [15]. All the parameter tuning was the same for all testing data sets and masks.

C. Key Results of the Original Paper

The state-of-the-art method for semantic inpainting is the Context Encoder (CE) [14], which is an autoencoder conditioned on the corrupted images [20]. It works best while the structure of the holes is fixed during the training and inference steps. However, it isn't very good when the regions deleted are not fixed.

Overall, the results for their model was superior. Contending against the CE, they found out that CE needed to be re-trained in order to achieve satisfactory results, their method however, could be applied to random masks without re-training the network, which they considered a huge advantage, argument with which we agree. Also, their capabilities were evidently superior with arbitrary masking (as their initial hypothesis stated). Figure 2 shows us a couple of successful examples.

Fig. 2. Image obtained from [20]. Some successful examples. Left to right is the real image, the input, the test with CE and the paper's test.

It is also important to notice that even though the context encoder was the main contender, there were more models that were compared with the paper's new method. These are: nearest neighbors filling, total variation [18], [1], low rank [6] and PatchMatch [3]. The new method showed superior results in almost every test against these models, being CE the only real contender.

However, not every result was successful. There were cases where the networks unsuccessfully inpainted the image, creating unrealistic and blurry images, or simply attaching a subset of a training image into the corrupted image. Figure 3 shows some less favorable examples.

Fig. 3. Image obtained from [20]. Not so successful examples

III. Methodology

We followed the footsteps of Yeh et al [20]. and took a step further to improve the inpainting results through various image augmentation methods and modifications to the adversarial network. We evaluated and compared our approach on three data sets: CelebA [10], SVHN [12] and Stanford Cars [8], with various types of missing regions. Our

inpainting results show that our approach can fill the missing regions with realistic information.

A. Objectives and Technical Challenges

The objective of this paper is to replicate the model proposed by Yeh et al. We aspire to achieve similar results and if time allows, tweaking the parameters of the DCGAN architecture and parts of the algorithms to improve the performance of the model. We encountered several technical challenges which could be enclosed in 3 key aspects:

- Data manipulation.
- Architecture and memory limitations.
- Image inferring and inpainting.

Given the fact that the data sets we used for evaluation are large and complex, we created a bucket in Google cloud for us to access it through the instance. However, it took close to an entire day to upload and the running time necessary for downloading the data into the instance during training is almost two hours. Therefore, in parallel, we decided to move the data sets to the actual instance instead of having to call it every time. It was imperative to have the entire data set for training in order to get desired results, since using subsets of data sometimes just creates silhouettes but not complete images.

Finally, we had 32 x 32 images, making the resize a little tricky. Fortunately, openCV library can do this automatically. Moreover, the results from the different data sets was very different given the quality and amount of data we have from each data set.

In terms of architecture, DCGAN is very complex and difficult to implement. There are multiple design choices on hyper parameters, layers and activation functions that can significantly impact the training results. Also, different from other neural networks such as CNN (convolutional neural network) that do not require a different model in the inference phase, we need a separate model for inpainting. Due to the size of our trained networks, we also ran into resource exhaustion/memory allocation issues when we tried to restore our model. Therefore, we had to increase our virtual instances' memory to be able to read and manipulate the network graphs.

Last but not the least, image inferring and inpainting is also challenging. After training and storing the DCGAN model, we ran into many complications when restoring and graph and using it for the testing and inpainting. We were not able to recover the last image generated by the model, so that our current inpainting model is not able to reconstruct the inpainted image from a test image using the trained model. With more time and understanding of Tensorflow, we could have been able to overcome this challenge.

B. Problem Formulation and Design

Our objective of is to solve the image inpainting problem as an image generation problem, especially for corrupted images with large missing regions. We aim to replicate the model proposed by Yeh et al [20], and potentially improve the performance of the model by experimenting with hyper parameters of the DCGAN architecture.

We utilize three data sets to evaluate our model: the Stanford Cars Dataset [8], the CelebFaces Attributes Dataset (CelebA) [10], and the Street View House Numbers (SVHN) [12]. It is beneficial to see and compare the results among different data sets.

- The Stanford Cars data set contains 16,185 images of 196 classes of cars. It was already splitted into train and test sets, where the training set contained 8,144 images, and the test set contained 8,041 images. We did not use any labels in either training or inference phase.
- The CelebA data set contains 202,599 face images with coarse alignment [10]. From here we separated 2,000 images for the testing phase.
- The SVHN data set contains 99,289 RGB cropped images house numbers, splitted by train and test.

All the images were resized to 64 x 64 to fit the DCGAN model architecture.

IV. Implementation

We implemented a DCGAN model, which is trained using uncorrupted data from the three data sets, and we then implemented an inpainting model that uses the trained DCGAN to infer missing regions given corrupted images [21]. While our overall architecture is similar to the work of Radford et al.[17], we experimented with a number of new designs and parameters, during the training of the network.

A. Deep Learning Network

We replicated the DCGAN model architecture from Radford et al.[17], and followed the implementation designs from Brandon Amos [2].

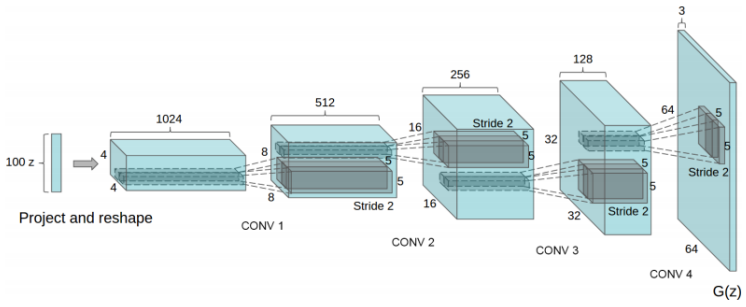

Fig. 4. Image obtained from [17]. The generator G(z) with a DCGAN.

The generative model, G, whose architecture is shown in fig. 4, takes a random 100 dimensional vector drawn

from either a uniform or normal distribution between [-1,1], and generates a 64 x 64 x 3 image. It has four hidden transpose convolution layers with two strides and a 5 x 5 kernel, each follow by a batch normalization and a ReLU activation function except for the output layer, which uses Tanh. The architecture of this CNN can be seen in Fig.4.

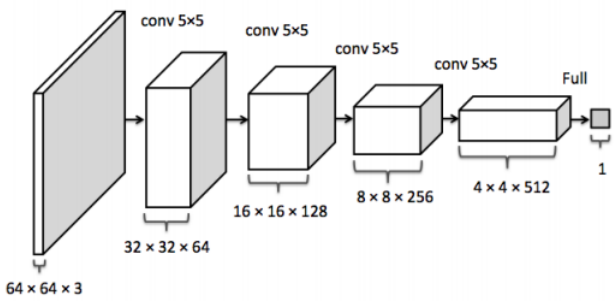

Fig. 5. Image obtained from [21].The discriminator convolutional network.

The discriminator model, D, shown in Fig.5, is designed in reverse order. It has four convolution layers with two strides and a kernel of 5 x 5, each one followed by batch normalization and a leaky relu activation function. The leaky activation function is used in order to fix the "dying ReLU" problem by allowing some positve gradient when units are not active.

The first layer in G and the last layer in D are fully connected hidden layers. LeakyReLU activation is used in D for all layers.

This architecture replaced any pooling layers with strided convolutions in D and fractional-strided convolutions in G. In addition, batch normalization is used in both G and D.

In the training phase, we used mini-batch stochastic gradient descent (SGD) with a mini-batch size of 64 and 128; Adam optimizer for optimization; and a learning rate of 0.0001, instead of the suggested 0.003. We also performed data augmentation of adding Gaussian noise on the training images.

In the inference phase, the inpainting stage, we use 200 iterations of back-propagation to find $\hat{z}$ in the latent space, and we use Adam for optimization and restrict z in to [-1,1].

B. Software Design

Fig.6 demonstrates a top level flow chart of our implementations. We first downloaded our three data sets from their respective links to local machines, and then uploaded them to Google Cloud Bucket and Virtual Instances. We then preprocessed the training images by resizing them into 64 x 64 and normalizing the pixel values to [-1,1].

Once we have the training images ready, we feed them to our network to build and train the DCGAN model,

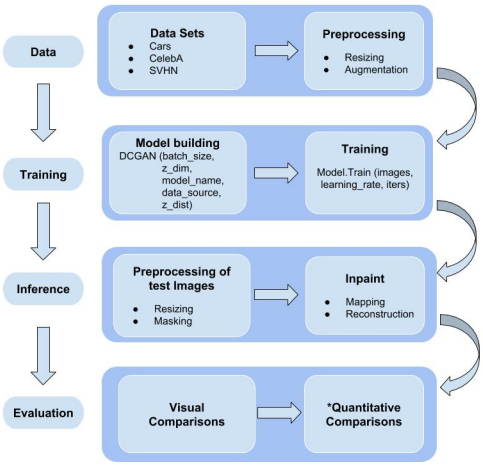

Fig. 6. Flow chart of our implementation.

which is in fact simultaneously training the generator and the discriminator. We used various sets of parameters for different data sets. The pseudo code for the DCGAN training algorithm is shown in Fig.7. The weights of the trained graph are later restored for the inpainting model during the distribution inferral phase.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

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

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

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

end for

Fig. 7. Image obtained from [2]. DCGAN training pseudo code algorithm.

After training, we also display the Tensorflow graph of our networks as shown in Fig.8.

In order to infer the distribution, we preprocessed the test images the same way as we did with the training images. We then used different masks to crop part of the test images so that they become corrupted images. Based on the defined prior and context loss, the corrupted images are mapped to the closest z in the latent space, and we denote it $\hat{z}$. We then feed $\hat{z}$ into our Generator, and obtain the inpainting result by overlaying the uncorrupted pixels from the test images. Lastly, Poisson blending [15] is used to reconstruct our final results. We then evaluate our results by visual comparison, and quantitative comparison if time permits.

All our source codes can be found in our GitHub Repository [9].

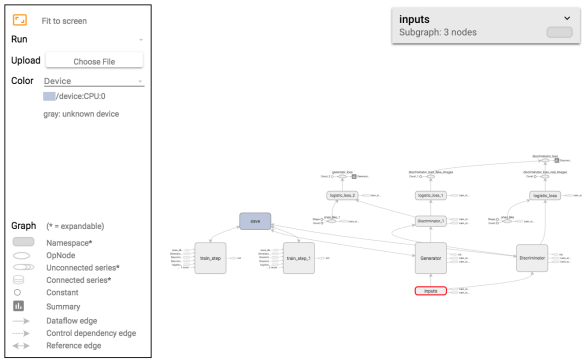

Fig. 8. Tensorflow graph.

Fig. 10. Training results of CelebA. From top left to bottom right, every image is 100 iterations further from the previous one.

V. Results

A. Project Results

We evaluated our DCGAN model using the three data sets: CelebA [10], SVHN [12] and Stanford Cars [8]. The experiment on CelebA achieved the most favorable results, and we will also discuss the limitations of our approach.

CelebA

For the training of the DCGAN, the CelebA dataset was the one generating best fake images. As seen in Fig.9, the images are very homogenous, containing only the faces of people very centered. The original images are not squared, so we lost some quality and definition from the images when resizing them to 64 x 64 pixels, and by adding random normal noise to improve results.

Fig. 9. Original images from CelebA dataset

In Fig.10, we can see the performance of the generator and how it is learning to generate more accurate images to win the discriminator. Starting at the top left we see only random noise, but after 1,500 iterations (bottom right corner), we are able to see the figure of a woman generated by the model.

For this particular model we used a batch size of 128, learning rate of 0.0002 and 1,500 iterations. We tried different approaches as well by changing the size of the random generator and distribution as well as the learning rate. Nevertheless, by increasing the size of the learning rate the results we didn't see improvement.

SVHN

SVHN is a data set that works slightly different than the other two ones. This was also a fact the preliminary evaluations were terrible. We were not capable of finding a cohesive image that would resemble a number.

Because of this, we tried a different approach: trying to infer a distribution of a specific label. Instead of using all our data set, we would for instance take all the images that were labeled as '1' and use them for training.

This intuition comes from the idea that if we fed the model with different numbers, they would stack and we would get blurred results instead of a clear number. By using labels, our manifold would get stronger constraints and define values that are actually a number.

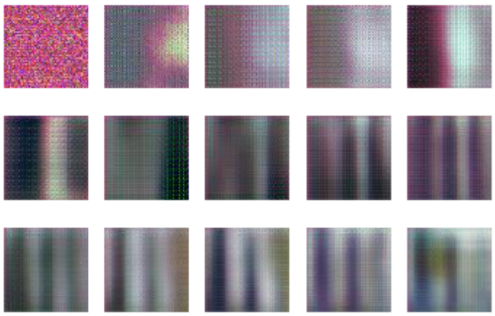

Fig. 11. Training results of SVHN with ones. From top left to bottom right, every image is 100 iterations further from the previous one.

As our results indicate, we can clearly see two number ones in blue in our images. This happens because most of the images have two digits and therefore, the network believes that two digits is a condition for that label.

A interesting idea would be to preprocess the data in order to separate each number in the images and then use that to create our inferences. Also, only taking images with ones might work as well, but the amount of data we would lose makes it an inefficient method.

Stanford Cars

Fig.12 demonstrates some examples of the training data of Stanford Cars after the preprocessing. Since all the images have various dimensions, some of the cars appear to be a bit stretched in some dimension after resizing them to 64 x 64. We lost some quality and definition in the images, but they are still recognizable for our purpose.

Fig. 12. Training results of the Stanford Cars data set.

Fig.13 shows the results from the training phase. While loading the data from Google Cloud Bucket takes half We used 1,500 iterations and those 15 images were sampled from every 100 iteration. It is evident that the images have been improving through the iterations, indicating that our DCGAN is in fact learning the characteristics of car images. The last image does have some structure and texture of a car image, even though it is not very clear.

There are several potential improvements we could have incorporated in our training. First, we did not use bounding boxes to crop the car images in preprocessing like the original paper suggested. As we can see in Fig.9, the first example image actually contains a good amount of background noise, and cropping could help improving the training results. In addition, Cars images have more variations compared to CelebA or SVHN. Therefore, we could subset the training images into different groups based on features such as the angel of the car (e.g. front, back, and side), and then train a model for each subset.

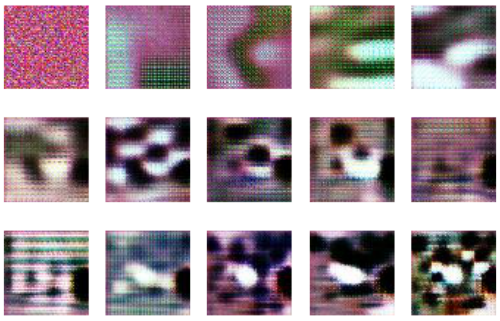

Fig. 13. Training results of the Stanford Cars data set.

Inpainting

Given the trained DCGAN model the process for inapainting consisted on the following:

- Preprocess the test data by reszing and applying a given mask for generating the holes.
- Restore model from the tensorflow graph network to pass new inputs to the generator for testing the images.
- Backpropagation to obtain new fake images (lattent space) for cropped test image.
- Postprocess to apply a poisson blending using binary mask to reconstruct the image.

For this part of the project, we were able to generate the pipeline for generating inpainting given different masks and a trained DCGAN. Nevertheless, we ran into many complications when restoring and graph and using it for the testing and inpainting. We were not able to recover the last image generated by the model, so our current inpainting model is not able to reconstruct the inpainted image from a test image using the trained model. In Fig.14, is it possible to see how the inpainting behaves for each mask given a random generator.

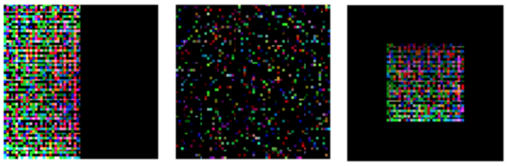

Fig. 14. Inpainting masks with random generator

B. Comparison of results

Out of the three data sets, CelebA achieved the best training results. Thus, we will mainly compare our CelebA results with that from the original paper.

Fig.15 shows some example inpainting results compared with other methods from the original paper. Visually, the results are realistic and impressive. Since we did not reach the final stage in inpaiting, we do not have inpainting results for this comparison. However, Fig.10 demonstrates promising training results from our DCGAN.

In terms of training time, the original paper did not specify how long their training process is. While our data took from half an hour to several hours to load, our training only took from a few minutes to 15 minutes.

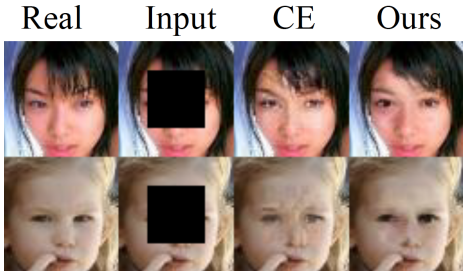

Fig. 15. Image obtained from [20]. Training results with the CelebA data set.

C. Discussion of Insights Gained

We have gained many valuable insights from this project. We not only learned about GAN's architecture and auto encoder, but also obtained knowledge on how various obstacles and limitations can impact the final training and inference results.

From a big picture view, inspired by Yeh et al.[20], we consider semantic inpaiting as an image generation problem with prior knowledge. This is an insightful application of GAN that significantly improved from state-of-art methods in image inpaiting. This application brings out new perspectives on how GAN can be utilized in various types of problems.

Holistically, we aimed to replicate Yeh et al.'s [20] approach, and we used the same data sets to compare our results. However, we experimented with different hyper parameters. For example, the original paper used 0.003 for learning rate and 128 for batch size, and we used 0.0001 for learning and 64 for batch size (except for CelebA). Furthermore, instead of using random horizontal flipping for data augmentation, we used adding Gaussian noise. Also, our Generator takes in a normal distribution rather than a uniform distribution at the beginning. Those modifications in design and hyper parameters yielded more promising results in our training phase.

VI. Conclusion

In this paper, we proposed a new approach to semantic inpainting [14] that infers arbitrary large missing regions. This task is more difficult than classical inpainting because of the prediction of high-level context.

Our goal is replicate the work of Yeh et al. [21] on semantic image inpainting with deep generative models and potentially improve the performance of their model. Their utilization of GAN for this task has significantly improved from previous state-of-the-art methods. We also conducted literature review of other related papers on application and implementation of DCGAN.

We tested our model in three different data sets: the Stanford Cars Dataset [8], the CelebFaces Attributes Dataset (CelebA) [10], and the Street View House Numbers (SVHN) [12]. We experimented with different hyper parameters as well as data augmentation methods, and our training on CelebA achieved the most favorable results. In the future, we would also like to subset the data by some feature and train a model for each subset. We believe that by doing so, the models could achieve more desirable results.

We encountered technical challenges in data manipulation, architecture and memory limitations, as well as image inferring and inpainting. Therefore, due to time and resource limitation, our inpainting model is not able to reconstruct the inpainted image from a test image using the trained model. In the future, we would be able to overcome this challenge with better understanding of Tensorflow. Also, we would like to conduct both qualitative and quantitative evaluations of our final results.

We not only obtained valuable knowledge on DCGAN and auto encoder from this project, but also gained many great insights by implementing the architecture and solving various obstacles. We believe that the GAN-based learning models have the potential to be applied in problems in different fields.

=1

VII. References

[1] M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo. An augmented lagrangian approach to the constrained optimization formulation of imaging inverse problems. *IEEE Transactions on Image Processing*, 2011.

[2] B. Amos. Image completion with deep learning in tensorflow. http://bamos.github.io/2016/08/09/deep-completion/step-1-interpreting-images-as-samples-from-a-probability-distribution, 2016.

[3] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), Aug. 2009.

[4] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, 2000.

[5] I. J. Goodfellow, J. Pouget-abadie, M. Mirza, B. Xu, and D. Warde-farley. Generative Adversarial Nets. pages 1–9, 2014.

[6] Y. Hu, D. Zhang, J. Ye, X. Li, and X. He. Fast and accurate matrix completion via truncated nuclear norm regularization. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 2013.

[7] Q. Y. W. S. JSJ Ren, L Xu. Shepard convolutional neural networks, 2015.

[8] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, 2013.

[9] L. R. A. N. R. Ling He, Gerardo Antonio. Image_inpainting_with_dgm. https://github.com/antoniolruiz/Image_inpainting_with_DGM, 2018.

[10] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. 2014.

[11] J. Mairal, M. Elad, and G. Sapiro. Sparse representation for color image restoration. *Trans. Img.*

Proc., 2008.

[12] W. T. C. A. B. A. W. B. Netzer, Y. and A. Ng.

[13] J. Pan, S. Liu, D. Sun, J. Zhang, Y. Liu, J. Ren, Z. Li, J. Tang, H. Lu, Y.-W. Tai, and M.-H. Yang. Learning dual convolutional neural networks for low-level vision. 2018.

[14] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. 2016.

[15] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. Graph.*, 2003.

[16] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2015.

[17] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. pages 1–16, 2016.

[18] J. Shen and T. F. Chan. Mathematical models for local nontexture inpaintings. *SIAM Journal of Applied Mathematics*, 62:1019–1043, 2002.

[19] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012.

[20] R. A. Yeh, C. Chen, T. Lim, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with perceptual and contextual losses. 2016.

[21] 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.