Interactive Image Generation Using a Conditional DCGAN

Lakshay Bhandari
ECE Department, NCSU
lrbhanda@ncsu.edu

Max Gordon
ECE Department, NCSU
mjgordo3@ncsu.edu

Abhayjeet Singh Juneja
ECE Department, NCSU
asjuneja@ncsu.edu

***Abstract*—The objective of this project is to develop a model which can generate images for a label given as input by the user. It is planned to develop a Deep Convolutional Generative Adversarial Network (DCGAN) which is capable of generating an image as close as possible to a real image corresponding to one of the classes of CIFAR-10 data-set.**

I. Introduction

Recently, machine learning models such as Generative Adversarial Models (GANs) [1] and their variants and derivatives have been developed to create images similar to real example images. In particular, Deep Convolutional GANs (DCGANs) are well-suited to generating images. GANs can also be modified to produce outputs similar to one class of a multi-class training set by adding an input of desired class to the generator, making the network a Conditional GAN (c-GAN) [2]. This can be combined with a DCGAN to produce a c-DCGAN, a conditional GAN well-suited to producing images [3].

In this project, we have developed a c-DCGAN architecture and trained it on the CIFAR-10 data-set. We have improvised the c-DCGAN architecture using some other useful metrics while keeping the goal of our project preserved. We then used this network to generate images of a particular class given a class label as input.

II. Methods

A. Data

To train our model, we used the CIFAR-10 dataset [4], which contains $60,000$ $32x32$ pixel color images, separated into 10 classes.

B. Architecture

Our network consists of two stages: a generator network to create candidate images and a discriminator network produce the probability of the image being real or fake, i.e., from the real data (CIFAR-10) or generated from the generator model. We will refer to Discriminator as D and Generator as G throughout the report. The generator network can be thought of as a reverse convolutional network (sometimes called deconvolution, but is actually the transpose or gradient of convolution rather than actual deconvolution) where instead of down-sampling, we up-sample the output of a previous layer to produce the input to the next layer. The inputs to the generator network are the image class to generate and a vector of random numbers to build the image from.

In our first approach, which is the vanilla c-DCGAN, we used the discriminator model to generate the sigmoid probabilities of the images being real or fake. The discriminator in this case is trained to decrease the distance between probabilities for real and fake images.

Inherently, the generator in this type of GAN is used to minimize the JS Divergence between the real and generated data distributions [1].

However, training G and D networks using the objective function in Eq.3 relies solely on the log probabilities of images being real or fake, which is incapable of distinguishing between two data distributions. This poses a problem especially when the real data is multi-modal, as in our case (CIFAR10 - 10 classes). Conventional GANs often fail to learn the multiple modes [5] from the real data distribution, generating images of similar 'mode' for different classes. This problem in GANs is generally referred to as Mode Collapse. Also, it is shown in [5] that the JS or KL divergence are not great choices for cost functions when learning distributions supported by low dimensional manifolds. In section Section III, we describe the use of Earth Mover (EM) distance or Wasserstein distance as suggested in [5] as the metric to distinguish between the real and generated data distributions. The Wasserstein distance between P and Q intuitively indicates the work done to move a certain amount from P to Q or vice versa in order to make them similar.

Also, as indicated in [5], Wasserstein distance is a good

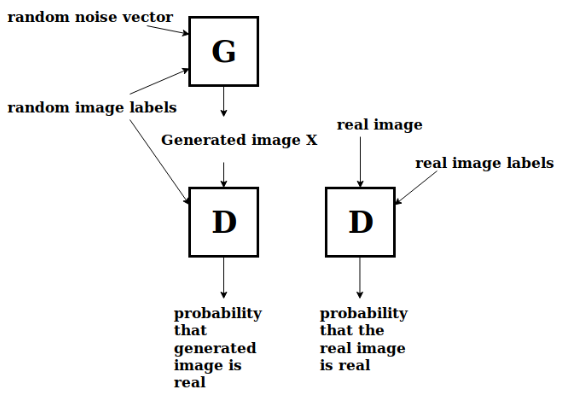

Fig. 1. Vanilla c-DCGAN Architecture Overview

choice as it is differentiable everywhere in contrast with JS Divergence. This offers an advantage in the optimization of the algorithm. Using the Wasserstein Distance as our loss function, we essentially convert the previously proposed Conditional DCGAN into a Conditional WDCGAN. In this new approach, the Discriminator, D does not produce the probabilities of the images being real or fake. This said, the loss is not calculated as a difference between log of these probablities. Rather, the final layer in G uses a tanh activation function to squash the output in the range [-1,1]. Using the tanh function in G is advantageous because as the training progresses, it produces negative values for fake images and positive values for real images. This increases the distance between the the $D(\boldsymbol{x})$ and $D(G(\boldsymbol{z},\mathbf{1}))$, where $\boldsymbol{x}$ represents the real data, G is the generator and $G(\boldsymbol{z},\mathbf{1})$ is the generated image and $\boldsymbol{z} \sim \mathcal{N}(0, I)$ is the normally distributed latent data provided to G along with the conditional input $V = \mathbf{1}$ which is a one-hot decoded label vector corresponding to 10 classes of CIFAR-10. Generator, G is trained to reduce $D(G(\boldsymbol{z},\mathbf{1}))$. To sum up, G is trained to minimize output activation of D given $G(\boldsymbol{z},\mathbf{1})$ and D learns to maximize the distance between the activations $D(.)$ for real and generated images, and is trained to reduce the difference between the two. Dwelling on this approach, training of WGANs can be improved using a gradient penalty as explained in [6]. We use this approach coupled with WGANs in this task. The metrics will be discussed in Section III.

The architectures in the final approach shown below

- Discriminator, D
  - Conv2d: Filters: 64; Kernel size: 5; Stride: 2
  - Conv2d: Filters: 128; Kernel size: 5; Stride: 2
  - Conv2d: Filters: 256; Kernel size: 5; Stride: 2
  - Conv2d: Filters: 512; Kernel size: 5; Stride: 2
  - Conv2d: Filters: 1; Kernel size: 4; Stride: 1

We use Leaky ReLU activations to preserve the activations in the negative region. As suggested in [5], no Batch Normalization [7] is used between the layers in D.

- Generator, G
  - Concatenate : $(\boldsymbol{z},\mathbf{1})$
  - Dense: 8192
  - Transpose Conv2d: Filters: 512; Kernel size: 5; Stride: 2
  - Transpose Conv2d: Filters: 256; Kernel size: 5; Stride: 2
  - Transpose Conv2d: Filters: 128; Kernel size: 5; Stride: 2
  - Transpose Conv2d: Filters: 3; Kernel size: 5; Stride: 1

We use Identity activation function, which essentially is $a(x) = x$ in all the layers except for the last, where we use a tanh activation to squash the pixel values of the images produced by G in the range [-1,1].

C. Implementation

We implemented this Conditional WDCGAN using the TensorFlow [8] library in Python. We used the following key layers provided by the library to construct the network.

- Conv2D : $tf.contrib.layers.conv2d$ [9]
- Conv2DTranspose : $tf.contrib.layers.conv2d\_transpose$ [10]
- Concatenate : $tf.concat$ [11]
- Fully Connected : $tf.contrib.layers.fully\_connected$ [12]

In addition to these layers, we also the following activation functions:

- Sigmoid : $tf.math.sigmoid$ [13]
- Tanh : $tf.math.tanh$ [14]
- LeakyReLU : $tf.nn.leaky\_relu$ [15]

and the Adam optimizer

- Adam : $tf.train.AdamOptimizer$ [16]

TensorFlow offers easy and efficient implementation of the two member networks G and D and caters to the training needs of two networks where one is learning from the other.

D. Hyperparameter Selection

As mentioned in the section II, our final architecture is based on the [6]. We chose the our loss function from JS-Divergence to Wasserstein Distance between the real and generated data distributions. Further, with the Wasserstein distance as our loss function, we again modified our loss function by adding a gradient penalty (GP) according to [6]. This gives us 3 iterations of our GAN model network. For CIFAR-10 data, the final model was selected based on the following: (MC refers to Mode Collapse)

	c-DCGAN	c-DCWGAN	c-DCWGAN+GP
MC	More	Less	Least
Performance	Worse	+ Unclear Img	+ Clear Img

Therefore, we chose the GAN network with Wasserstein distance as our loss function modified with a gradient penalty.

One of the other things we played with while building the network were providing the latent features to the Generator G. We noticed that the performance of any GAN network was apt when the random-normal distributed noise vector was given to G using numpy package [18] in python. The performance were not expedient with random-uniform distributed noise vector.

For our Optimization function, we chose Adam optimizer [19] offered in TensorFlow package with the parameters as specified in [20]. One interesting experiment would be to test the performance with RMSProp [21] as the optimizer which could not be done now because of time constraints.

III. Evaluation

As stated in Section II, our initial network is a Conditional DCGAN which inherently uses the JS divergence to distinguish between the real and generated data distributions. JS Divergence between P and Q is

$$JSD(P||Q) = \frac{1}{2}KL(P||M) + \frac{1}{2}KL(Q||M) \qquad (1)$$

where P, Q represent real and generated data distributions, respectively, $KL(P||Q)$ represents the KL Divergence [17] between P and Q, and

$$M = \frac{1}{2}(P + Q) \tag{2}$$

The objective function of this GAN architecture is as follows

$$\begin{aligned} min_{\boldsymbol{G}} max_{\boldsymbol{D}} V(D, G) = E_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[log D(\boldsymbol{x})] \\ + E_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[log(1 - D(G(\boldsymbol{z})))] \end{aligned} \tag{3}$$

where the first term in RHS works to maximize the log probability of a real image being real, while the second term in the RHS works to minimize the probability of the image being real when fed a fake image, which sums up the min-max game that a GAN is. Training with JS divergence, it became difficult for the GAN network to learn all the classes present in the dataset. In out case, the 10 classes belonging to CIFAR10 were not learned by the GAN when trained with JS Divergence. We specifically noted, as was also shown in the initial progress presentation, that this network produced images mainly corresponding to Automobiles. Additionally, we also noted that the D training loss kept increasing although G training loss was decreasing. We concluded that D network was incapable teach the G network.

This may have arisen because D produced log probabilities for the images being real or fake. D network was then trained to minimize the distance between

$$\mathcal{L} = D(\boldsymbol{x}) - D(G(\boldsymbol{z}, \mathbf{1})) \tag{4}$$

where these computations are done over a batch (one training step). $\mathbf{1}$ denotes the one-hot encoded conditional input to G now and hereafter, $D(.)$ in this GAN network produced probability

$$\mathcal{P}(Image) = Real/Fake \tag{5}$$

To optimize Eq.4, we used an Adam Optimizer [19] as mentioned in [20]. Since Eq.4 suggests that the distance between real and generated data distributions is small, it was difficult to optimize it and hence difficult for D to distinguish between the two kinds of images. This poor learning trend resulted in the problem of Mode Collapse [5]. The results generated by this Conditional DCGAN can be seen in Fig.8. Each column represents a single class of CIFAR10 dataset.

We have stated that [5] suggests that choosing Wasserstein distance as a loss function in training GANs is more suitable than using JS or KL Divergences. Thus, to avoid the difficulties of our previous approach, we chose to change the network from c-DCGAN to c-WDCGAN [20]. Our goal is still the same as before. Although there are a number of ways to prevent Mode Collapse in GANs [22], we chose the approach of using a better loss metric, Wasserstein distance. Also, [6] brings forward the problem of capacity underuse and states that the optimal WGAN discriminator d has unit gradient norm almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$. So, we observe that our neural network architecture which tries to attain its maximum gradient norm k ends up learning extremely simple functions. It also explains the problem of exploding and vanishing gradients that makes the WGAN optimization process difficult due to complex interactions between the weight constraint and the cost function. To overcome this, [6] suggests a method of gradient penalty. It states that a differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere, so it considers directly constraining the gradient norm of the discriminator d output with respect to its input. For this, a soft version of the constraint with a penalty on the gradient norm is enforced for random samples $\widehat{x} \sim \mathbb{P}_{\widehat{x}}$ and that make the new objective as

$$\begin{aligned} L = E_{(\widetilde{x} \sim \mathbb{P}_g)}[D(\widetilde{x})] - E_{(\widetilde{x} \sim \mathbb{P}_r)}[D(x)] + \\ \lambda E_{(\widehat{x} \sim \mathbb{P}_{\widehat{x}})}[(\|\nabla \widehat{x} D(\widehat{x})\|_2 - 1)^2 \end{aligned} \tag{6}$$

where λ is the penalty coefficient and is chosen to be equal to 10 in our case. We have also implemented a no critic batch normalization technique mentioned in [6]. Usually, GAN implementations use batch normalization in both the generator and the discriminator which is expected to stabilize the training process. But, batch normalization changes the form of the discriminators problem from mapping a single input to a single output, to mapping from an entire batch of inputs to a batch of outputs. This is not valid for the new penalized objective function. To resolve this, batch normalization is not used with the discriminator.

With little tweaks in our initial model, we arrived at the final model as shown in II-B which is just the c-WDCGAN with gradient penalty term (GP). With this approach, G is trained such that the pixel values of the generated images is fairly toward -1. Recall that G has a $tanh$ activation function in the last layer. This helps D to distinguish between real and generated images, which contributes to its fair learning trend. The definition of Wasserstein distance or Earth Mover (EM) distance in [5] is given as:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} E_{(x,y) \sim \gamma}\big[\, \|x - y\| \,\big] \tag{7}$$

where $\Pi(P_r, P_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively P_r and P_g. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions P_r into the distribution P_g.

The loss function for D can be thought of as Eq.4 in this case as well. But in this case, $D(.)$ does not produce probabilities of the images being fake of real. Rather, output of $D(x)$ and $D(G(z, \mathbf{1}))$ is now far apart (Wasserstein Distance) and hence is fairly optimizable. We used the same Adam optimizer for this architecture as well. It was seen from the training loss of D and G that although the loss in G was high, there was a point during training where loss of D went to zero and the loss for G had reached the minimum. This was achieved after 100 epochs. We continued training for more epochs, but the loss for both the networks kept increasing. Thus, we chose the saved checkpoint for G after 100^{th} epoch for generating results. This can be seen in Fig.3. Each column corresponds to each class in CIFAR-10.

IV. Results

We show the results generated by the network in our initial approach (JS Divergence) and in the final approach, where we show the results of c-WDCGAN implementation. The ground truth, which is the CIFAR-10 Dataset in this case is shown in Fig.2 It can be seen in Fig.3 that the images generated are clearer and almost distinctively represent their respective classes in contrast with Fig.8. Training error of c-WDCGAN implementation for G and D is shown in Fig.7 and Fig.6 respectively. As mentioned before, we chose the model checkpoint where losses associated with both G and D is the least. In contrast, Fig.5 and Fig.4 represent training loss for G and D in c-DCGAN implementation, where it can be seen that D fails to teach G as the loss associated with it increases over training duration. We report the result for this GAN model after 35^{th} epoch as D loss kept increasing and results kept deteriorating. Fig.9 through Fig.16 show comparison images generated by c-DCGAN and c-WDCGAN with GP. It can be seen that while each image from c-WDCGAN represnts its class distinguishably, the images generated by c-DCGAN fail to some extent to do so.

V. Discussion

We developed a c-DCGAN modified to c-WDCGAN with gradient penalty (GP) and trained it on the CIFAR-10 dataset. This network was then used to generate examples of several classes of photos as labelled in the CIFAR-10 dataset. We were able to generate images resembling elements of the CIFAR-10 classes using this network and also controlling the generation with conditional input. By using the modified versions of GAN, we were able to reduce the problem of Mode Collapse substantially. By substituting the Wasserstein distance for the standard loss function, we were able to improve the performance of this model for generating new images.

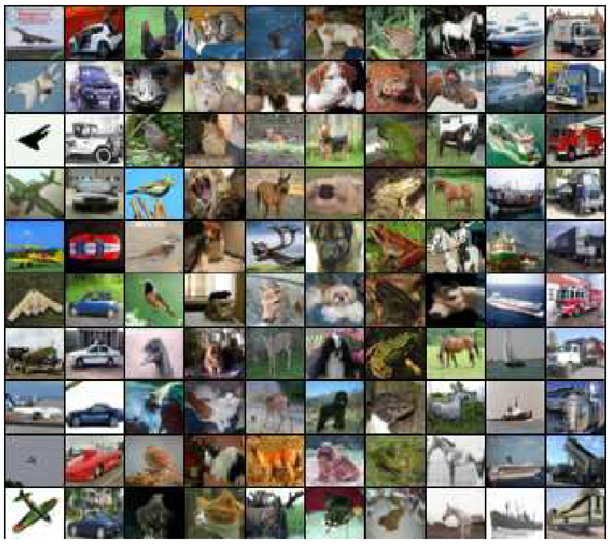

Fig. 2. CIFAR-10 Dataset (Ground-Truth)

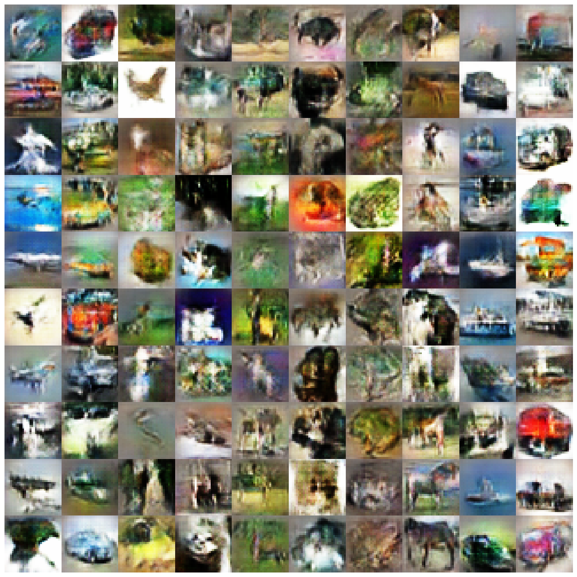

Fig. 3. CIFAR-10 results after 100 epochs (Wasserstein)

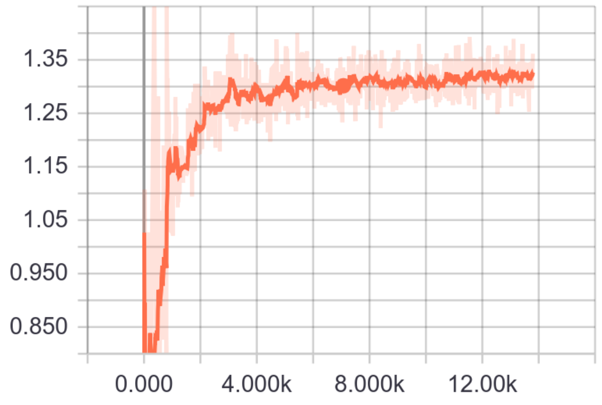

Fig. 4. Discriminator Loss over training steps

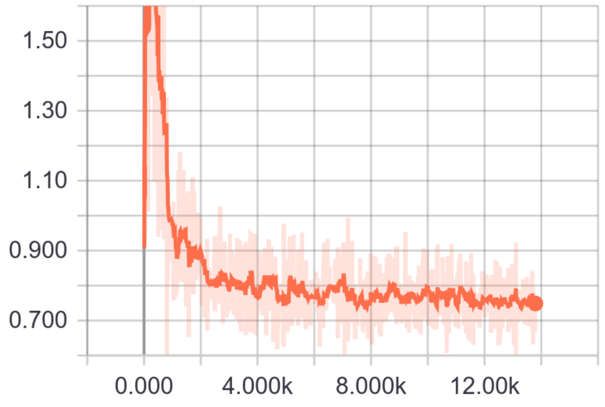

Fig. 5. Generator Loss over training steps

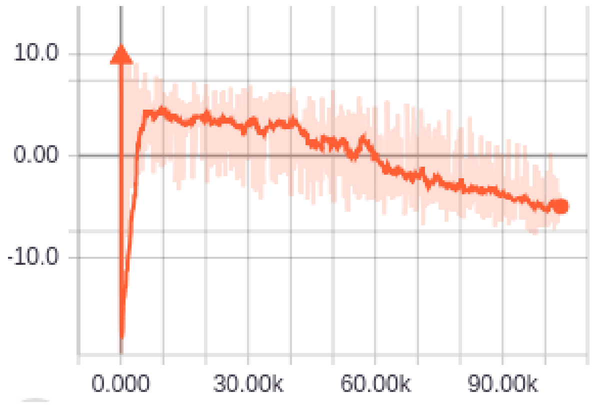

Fig. 6. Discriminator Loss over training steps (Wasserstein)

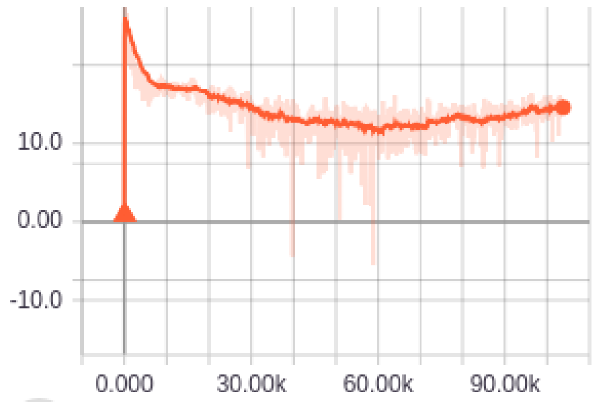

Fig. 7. Generator Loss over training steps (Wasserstein)

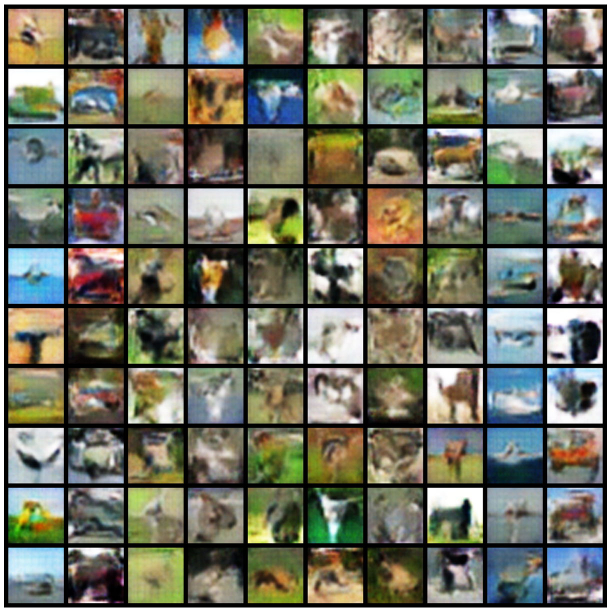

Fig. 8. CIFAR-10 results after 35 epochs (non Wasserstein)

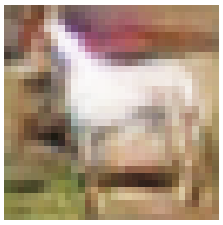

Fig. 9. Horse (Wasserstein)

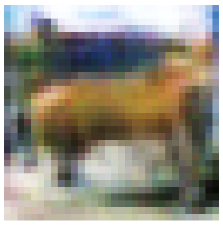

Fig. 10. Horse (Non Wasserstein)

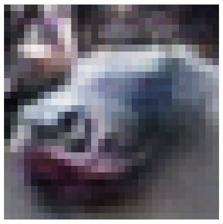

Fig. 11. Car (Wasserstein)

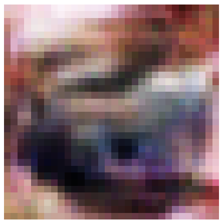

Fig. 12. Car (Non Wasserstein)

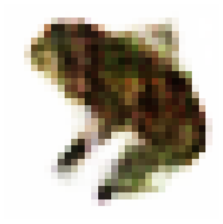

Fig. 13. Frog (Wasserstein)

Fig. 14. Frog (Non Wasserstein)

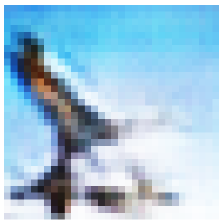

Fig. 15. Airplane (Wasserstein)

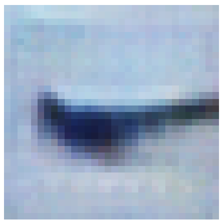

Fig. 16. Airplane (Non Wasserstein)

REFERENCES

[1] J. Hui, “Generative adversarial nets (GAN) , DCGAN, CGAN, InfoGAN”. [Online]. Available: https://jhui.github.io/2017/03/05/Generative-adversarial-models/

[2] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,”arXiv:1411.1784 [cs], Nov. 2014.

[3] U. Desai, “Training a Conditional DC-GAN on CIFAR-10”. [Online]. Available: https://medium.com/@utk.is.here/training-a-conditional-dc-gan-on-cifar-10-fce88395d610

[4] A. Krizhevsky., “Learning multiple layers of features from tiny images,” April 2009.

[5] M. Arjovsky et al., “Wasserstein Generative Adversarial Networks,” arXiv:1701.07875 [stat], Aug. 2017.

[6] I. Gulrajani et al., “Improved Training of Wasserstein GANs,” arXiv:1704.00028 [cs], Dec. 2017.

[7] S. Ioffe, C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, arXiv:1502.03167 [cs], Feb. 2015.

[8] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” Nov. 2015.

[9] “tf.nn.conv2d — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/nn/conv2d.

[10] “tf.nn.conv2d_transpose — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/nn/conv2d_transpose.

[11] “tf.nn.concat — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/nn/concat.

[12] “tf.nn.fully_connected — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/nn/fully_connected.

[13] “tf.math.sigmoid — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/math/sigmoid

[14] “tf.math.tanh — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/math/tanh

[15] “tf.nn.leaky_relu — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/nn/leaky_relu

[16] “tf.train.AdamOptimizer — TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer

[17] S. Kullback, R. Leibler, “On information and sufficiency”, Annals of Mathematical Statistics, 1951.

[18] T. Oliphant. “A guide to NumPy,” Trelgol Publishing, 2006.

[19] D. Kingma, J. Ba, “Adam: A Method for Stochastic Optimization”, arXiv:1412.6980 [cs], Dec. 2014

[20] C. Fabbri, “Conditional Wasserstein Generative Adversarial Networks”. [Online]. Available: https://cameronfabbri.github.io/papers/conditionalWGAN.pdf

[21] G. Hinton “RMSProp.” [Online]. Available: http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

[22] D. Bang, H. Shim, “MGGAN: Solving Mode Collapse using Manifold Guided Training”, arXiv:1804.04391 [cs], Apr. 2018