A Project Report on

Detection of COVID-19 through chest X-ray using WGAN-GP

Submitted by

Saurabh Mangalvedhekar (Roll no. 09)

Karthik Manthitta (Roll no. 10)

Hitarthi Mokashi (Roll no. 13)

Aditya Nayak (Roll no. 17)

in partial fulfillment for the award of the degree

BACHELOR OF ENGINEERING

in

Electronics and Telecommunication Engineering

Under the Guidance of

Mr. Santosh Chapaneri

St. Francis Institute of Technology, Mumbai

University of Mumbai

2020 - 2021

CERTIFICATE

This is to certify that Saurabh Mangalvedhekar, Karthik Manthitta, Hitarthi Mokashi, Aditya Nayak are the bonafide students of St. Francis Institute of Technology, Mumbai. They have successfully carried out the project titled "Detection of COVID-19 through chest X-ray using WGAN-GP" in partial fulfilment of the requirement of B.E. Degree in Electronics and Telecommunication Engineering of Mumbai University during the academic year 2020-2021. The work has not been presented elsewhere for the award of any other degree or diploma prior to this.

_____

(Mr. Santosh Chapaneri)

_____ _____

(Dr. Gautam Shah)
EXTC HOD

(Dr. Sincy George)
Principal

Project Report Approval for B.E.

This project entitled ***'Detection Of COVID-19 through chest X-ray using WGAN-GP'*** by **Saurabh Mangalvedhekar, Karthik Manthitta, Hitarthi Mokashi, Aditya Nayak** is approved for the degree of Bachelor of Engineering in Electronics and Telecommunication from University of Mumbai.

Examiners

1. - - - - - - - - - - - - - - - - - - -

2. - - - - - - - - - - - - - - - - - - -

Date:

Place:

ACKNOWLEDGEMENT

We are thankful to a number of individuals who have contributed towards our final year project and without their help, it would not have been possible. Firstly, we offer our sincere thanks to our project guide, Mr. Santosh Chapaneri for his constant and timely help and guidance throughout our preparation.

We are grateful to all project co-ordinators for their valuable inputs to our project. We are also grateful to the college authorities and the entire faculty for their support in providing us with the facilities required throughout this semester.

We are also highly grateful to Dr. Gautam A. Shah, Head of Department (EXTC), Principal, Dr. Sincy George, and Director Bro. Jose Thuruthiyil for providing the facilities, a conducive environment and encouragement.

Signatures of all the students in the group

(Saurabh Mangalvedhekar)

(Karthik Manthitta)

(Hitarthi Mokashi)

(Aditya Nayak)

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included; we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in this submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signatures of all the students in the group

(Saurabh Mangalvedhekar)

(Karthik Manthitta)

(Hitarthi Mokashi)

(Aditya Nayak)

Abstract

Generative Adversarial Networks (GAN) are powerful generative models but have many drawbacks related to convergence and mode collapse and suffer from instability. The proposed plan is to apply Wasserstein GAN (WGAN) with Gradient Penalty (GP) in order to expand the database of chest X-ray images and produce high quality plausible images, as WGAN has been proven to have better stability and efficiency in comparison to its predecessors. These images would then be given to a CNN which would then classify the images and indicate the probable presence or absence of COVID-19 in the patient.

Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
CNN	Convolution Neural Network
COVID	Corona Virus Disease
GAN	Generative Adversarial Network
JS	Jensen-Shannon
KL	Kullback-Leibler Divergence
ML	Machine learning
PA	Posteroanterior
ReLU	Rectified Linear Unit
VAE	Variational Auto Encoders
WGAN-GP	Wasserstein Generative Adversarial Network with gradient penalty
WGAN	Wasserstein Generative Adversarial Networks

Chapter 1

Introduction

1.1 Motivation

As we all know the COVID-19 virus is one of the most devastating viruses till date. It is the virus that causes inflammation in lungs which leads to pneumonia. Several deaths have been reported due to this virus around the globe. The development of a GAN which is an intricate part of machine learning will help us achieve in the detection of COVID 19 virus present in the lungs. ML is the study of computer algorithms that improves automatically through experience. A definition given by Mr. Tom Mitchell is that *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T as measured by P improves with experience E."* [1]. Machine learning is the ability of a computer system to observe, learn and gain experience from lots of data, and use this experience to predict future results. Machine learning is basically of four types namely:

1. Supervised Learning
2. Unsupervised learning
3. Semi-supervised learning
4. Reinforcement Learning

In supervised learning, the goal is to learn the mapping (the rules) between a set inputs and outputs. If the model is over-trained, which causes over-fitting to

the examples used and the model would be unable to adapt to new, previously unseen inputs. Supervised learning is limited in a variety of sense such that it cannot handle some of the complex tasks in machine learning. Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labelled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of density estimation in statistics, such as finding the probability density function. Unsupervised learning also encompasses other domains involving summarizing and explaining data features. This model will be then passed on to a convolutional neural network (CNN). A CNN is a foundation of most computer vision technologies. It uses two operations called the pooling and convolution which consists of different types of layers which are fully connected.

1.2 Problem Statement

To generate COVID-19 chest X-ray images using Wasserstein Generative Adversarial Network-Gradient Penalty (WGAN-GP). Given the chest X-ray of the patients, to detect the following respiratory infections using Convolution Neural Network (CNN):

1. COVID-19 (Primary)
2. Pneumonia

1.3 Methodology

Patients having symptoms of COVID-19 have to get an X-ray of their chest done and show it to their respective doctors to determine if they have COVID-19 or pneumonia present in their lungs. The intensity of the virus inside the lungs can be determined majorly by examining the chest X-ray of the patient. After knowing the intensity of the infection, the doctors can start with the treatment of the particular

patient. Bearing in mind the crowded hospitals and test centres, we have created a model to classify the images which will provide the patients with an early intimation of the possibility of the infection.

1.4 Organization of Project Report

This project report is organized as follows:

Chapter 2 presents the literature survey on the existing techniques
In the initial stages of the conceptualization of this project, we present a survey of literature to study the work that has already been done in the field of Machine Learning and its various applications.

- Supervised learning: Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. It has drawbacks like overfitting and it cannot cluster or classify data by discovering its features on its own.

- Unsupervised learning: The drawbacks are satisfied with unsupervised learning. In unsupervised learning, only input data is provided in the examples. There are no labelled example outputs to aim for. To find the interesting structures in unlabeled data, we use density estimation.

- Convolutional Neural networks: A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

- Generative adversarial network: Generative Adversarial Networks, or GANs for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks .Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the

model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

- Deep Convolutional Generative Adversarial Network: DCGAN is one of the popular and successful network design for GAN. It mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the down-sampling and the up-sampling.

- Wasserstein Generative Adversarial Network: The Wasserstein Generative Adversarial Network, or Wasserstein GAN, is an extension to the generative adversarial network that both improves the stability when training the model and provides a loss function that correlates with the quality of generated images.

- Wasserstein Generative Adversarial Network Gradient penalty: WGAN-GP, is a generative adversarial network that uses the Wasserstein loss formulation plus a gradient norm penalty to achieve Lipschitz continuity.

Chapter 3 provides a brief explanation of the theoretical background of GAN, DCGAN, WGAN and WGAN-GP. It describes how a generative model works, the working of GAN, various loss functions that are being used in our achitecture. The working of DCGAN, the working of WGAN and the drawbacks of each architecture which lead us to WGAN-GP. It also describes about the dataset of the chest X-ray images of normal patients as well as patients having COVID-19 and pneumonia.

Chapter 4 is dedicated to the simulation and experimental results.

Chapter 5 presents the conclusions and future scope for this project.

Chapter 2

Literature Survey

1. *"Generative Adversarial Networks"* by Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio. [2]

   The authors have discussed in brief about generative models and adversarial networks and how they work. They have also elaborated on the minimax equation and discussed how the discriminator and generator learn simultaneously and the final ideal state where discriminator accuracy stands at 50% and it is unable to differentiate images as real or fake. The algorithm of GAN working has been illustrated. The equations and state of convergence have been discussed. They have worked on CIFAR-10, MNIST and many others. The advantages included backpropagation to update weights of the generator network through discriminator through gradients and not by directly changing the parameters of the generator. The disadvantages of the existing model were that the generator model ran into "mode collapse" which hampered the productivity of the model.

2. *"Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"* by Alec Radford, Luke Metz and Soumith Chintala. [3]

The authors have discussed the use of convolutional layers (which uses convolutional filters that move over the input image) in place of dense layers as in GAN and have used this architecture on various datasets and proved how use of said convolution layers is beneficial in image processing and extraction of important features from the images. The use of GANs as a feature extractor to classify CIFAR-10 has been discussed. The problem of mode collapse still persisted as filters moved to a single oscillating mode which led to instability in the model. They succeeded in reducing the number of trainable parameters in the generative model by making use of convolutional layers in place of dense layers.

3. *"Wasserstein GAN"*, by Martin Arjovsky, Soumith Chintala and Leon Bottou. [4]

The authors have provided a comprehensive theoretical analysis of how the Earth Mover (EM) distance behaves in comparison to popular probability distances and divergences used in the context of learning distributions.They have defined a form of GAN called Wasserstein-GAN that minimizes a reasonable and efficient approximation of the EM distance and theoretically shown that the corresponding optimization problem is valid. Also empirically shown is how WGANs cure the main training problems of GANs. In particular, training WGANs does not require maintaining a careful balance in training of the discriminator and the generator, and does not require a careful design of the network architecture either. The mode collapse phenomenon that is typical in GANs is also drastically reduced. One of the benefits of WGANs is the ability to continuously estimate the EM distance by training the discriminator to optimality. Plotting these learning curves is not only useful for debugging and hyperparameter searches, but also correlate with the observed sample quality.

4. *"Improved Training of Wasserstein GANs"*, by Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin and Aaron Courville. [5]

   The authors propose gradient penalty (WGAN-GP), which does not suffer from the problems caused due to critic weight clipping which was implemented in [4] and these experiments on gradient penalty were performed on LSUN dataset. They have also demonstrated stable training of varied GAN architectures, performance improvements over weight clipping and high-quality image generation.

2.1 Our contribution

- We have implemented WGAN-GP on chest X-ray P.A. view images of COVID-19.
- We made use of our own simple model architecture for generative model and were able to generate images that bore resemblance to the original dataset.
- We built a successful classifier model with 98% accuracy and the architecture used was simple without use of a deep neural net.

2.2 Literature review on the existing technology

2.2.1 Google Colab

Google Colab is an online Python compiler mainly used to solve Machine Learning and data analysis problems. It is hosted by Jupyter notebook service that requires no setup to use. Colab provides different cells which makes possible to execute every line of the program individually. It also helps in debugging. One can also export external datasets in Colab and it also supports many popular Machine Learning libraries such as Matplotlib, Pandas, Numpy etc.

2.2.2 Augmentor

Augmentor is a package in Python for image augmentation. It helps us to generate similar images as of the input image. This is mostly used to resolve many Machine Learning problems. This package has a pipeline-based approach. All the images are passed through the pipeline, where each defined operation is added sequentially. We have used multiple operations such as flip, rotate, tilt to create a larger dataset. A user defined value of probability is taken and the given number of images are selected randomly and augmented. This helps us to train the discriminator model on more images and increase its capability to differentiate fake images from real. The example given below is a sample output of Augmentor. Fig. 2.2, Fig. 2.3, Fig. 2.4 shows the rotated, sheared and flipped images respectively. In this example the shearing and rotation is done in the range -10 to 10 degrees. Images of our dataset are sheared and rotated in the range -2 to 2. The rotation range is small because the images of the entire dataset should be somewhat similar to train the discriminator accurately.

Figure 2.1: Original image

Figure 2.2: Rotated image

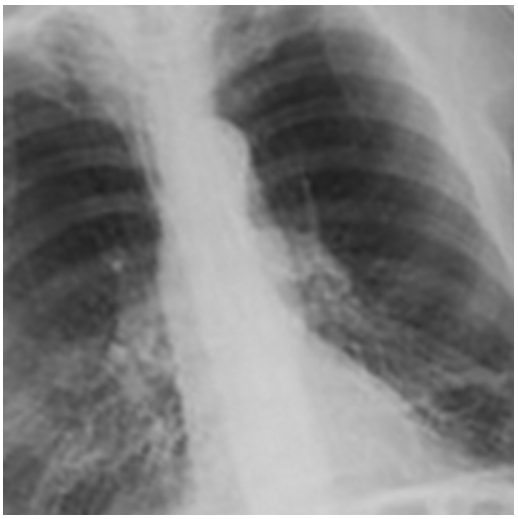


Figure 2.3: Sheared image

Figure 2.4: Flipped image

Chapter 3

Theoretical Background and Design Methodology

3.1 Meaning And Examples

Generative adversarial networks (GANs) are an exciting recent innovation in Machine Learning. GANs are generative models, they create new data instances that resemble your training data. For example, GANs can create images that look like photographs of human faces, even though the faces don't belong to any real person. These images shown in Fig. 3.1, were created by a GAN:

Figure 3.1: Sample Images Produced By GAN
[6]

GANs achieve this level of realism by pairing a generator, which learns to produce the target output, with a discriminator, which learns to distinguish true data from the output of the generator. The generator tries to fool the discriminator, and the discriminator tries to keep from being fooled.

3.2 What is Generative Modelling?

A generative model could generate new photos that look like real photos, while a discriminative model could differentiate between the classes of the photos. GANs are just one kind of generative model.

- Generative models can generate new data instances.
- Discriminative models discriminate between different kinds of data instances.

More formally, given a set of data instances X and a set of labels Y:

- Generative models capture the joint probability $p(X, Y)$, or just $p(X)$ if there are no labels.
- Discriminative models capture the conditional probability $p(Y|X)$. Discriminative models try to draw boundaries in the data space, while generative models try to model how data is placed throughout the space. For example, the Fig. 3.2, shows discriminative and generative models of handwritten digits:

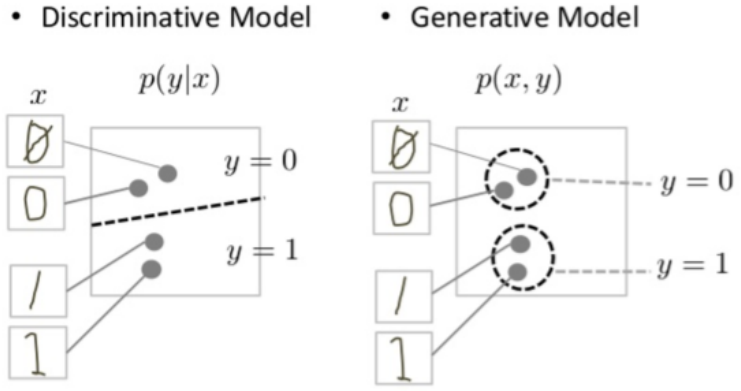


Figure 3.2: Basic discriminative and generative model for handwritten digits [7]

The discriminative model tries to tell the difference between handwritten 0's and 1's by drawing a line in the data space. If it gets the line right, it can distinguish 0's from 1's without ever having to model exactly where the instances are placed in the data space on either side of the line.

In contrast, the generative model tries to produce convincing 1's and 0's by generating digits that fall close to their real counterparts in the data space. It has to model the distribution throughout the data space.
GANs offer an effective way to train such rich models to resemble a real distribution.

3.3 Working Of GAN

3.3.1 The Discriminator

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying.

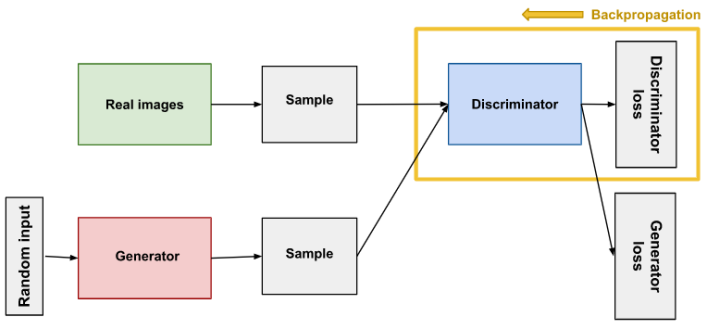


Figure 3.3: Block diagram of Basic GAN
[8]

The discriminator's training data comes from two sources:

1. Real data instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.

2. Fake data instances created by the generator. The discriminator uses these instances as negative examples during training.

In Fig 3.3, the two "Sample" boxes represent these two data sources feeding into the discriminator. During discriminator training the generator does not train. Its weights remain constant while it produces examples for discriminator to train on.

Training the Discriminator:
The discriminator connects to two loss functions. During discriminator training, we lock the parameters of the generator neural network for discriminator training so that the generator parameters are not changed while the discriminator is classifying the real and the fake data.
During discriminator training:

1. The discriminator classifies both real data and fake data from the generator.
2. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

3.3.2 The Generator

In generator training we lock the parameters of the discriminator and we use the discriminator to train the generator so that our generator parameters are only changed.
The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real. Generator training requires tighter integration between the generator and the discriminator than discriminator training. The portion of the GAN that trains the generator includes:

- Random Input
- Generator network, which transforms the random input into a data instance.
- Discriminator network, which classifies the generated data
- Discriminator output
- Generator loss, which penalizes the generator for failing to fool the discriminator

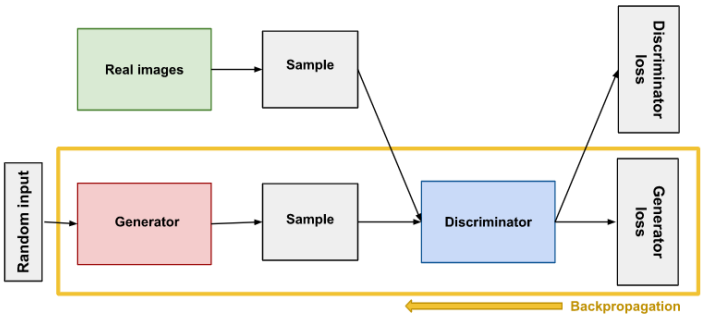


Figure 3.4: Backpropagation in generator training
[9]

Random Input:

Neural networks need some form of input. Normally we input data that we want to do something with, like an instance that we want to classify or make a prediction about.

In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, we can get the GAN to produce a wide variety of data, sampling from different places in the target distribution.

Experiments performed by the author as mentioned in [2] suggest that the distribution of the noise doesn't matter much, so we can choose something that's easy to sample from, like a uniform distribution. For convenience the space from which the noise is sampled is usually of smaller dimension than the dimensionality of the output space.

3.3.3 Loss function in GAN

As we can see that the geneartor is trying to fool the discriminator and discriminator is trying to differentiate better, they are playing a minmax game and trying to get better than the other and also improving themselves at the same time.

The loss function in GANs is called Minimax Loss because the generator tries to minimize the following function while the discriminator tries to maximize it:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data(x)}}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (3.1)$$

- E refers to expected value.

- $P_{data(x)}$ is the probability distribution of the real data.
- $D(x)$ is the discriminator's estimate of the probability that real data instance x is real.
- $p_z(z)$ is the probability distribution of the generated data.
- E_x is the expected value over all real data instances.
- $G(z)$ is the generator's output when given noise z.
- E_z is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances $G(z)$).

3.3.4 Using the Discriminator to train the Generator:

1. During generator training, the discriminator parameters are freezed so that the parameters are not backpropagated and modified which may affect the generator training.
2. The generator block is fed with a sample random Gaussian noise which is generated by creating a latent space of random integer points.
3. This noise is then upsampled to create a 'fake' image.
4. This image is then passed to a trained discriminator where it is classified as either 'fake' or 'real' i.e., class 0 or 1 respectively.
5. The generator loss penalizes the generator for producing a sample that the discriminator network classifies as 'fake'.

3.4 Problems faced by GAN

3.4.1 Hard to find Nash Equilibrium

GAN is based on a zero sum non-cooperative game which says that if one wins, the other loses. It is also called as minimax which implies that the opponent wants to maximise its actions and your actions are to minimize them. Here each model

updates its cost independently. Nash equilibrium is the only state where the opponents actions will not change the outcome of the game. When the generator and the discriminator reaches a Nash equilibrium, the GAN model converges.
Consider a two player which controls the value of x and y where the player A wants to maximise the value of xy and B wants to minimize the value of xy.

$$\min_G \max_D V(D,G) = xy \tag{3.2}$$

The Nash equilibrium is $x = y = 0$. In this state any actions of the opponents‘ will not change the outcome. To find out the Nash equilibrium using gradient descent, we update the x and y parameters based on the gradient of the value of the function V.

$$\Delta x = \alpha \frac{\partial(xy)}{\partial(x)} \tag{3.3}$$

$$\Delta y = -\alpha \frac{\partial(xy)}{\partial(y)} \tag{3.4}$$

Here α is the learning rate.
While plotting x, y, xy we can see that our solution does not converge.

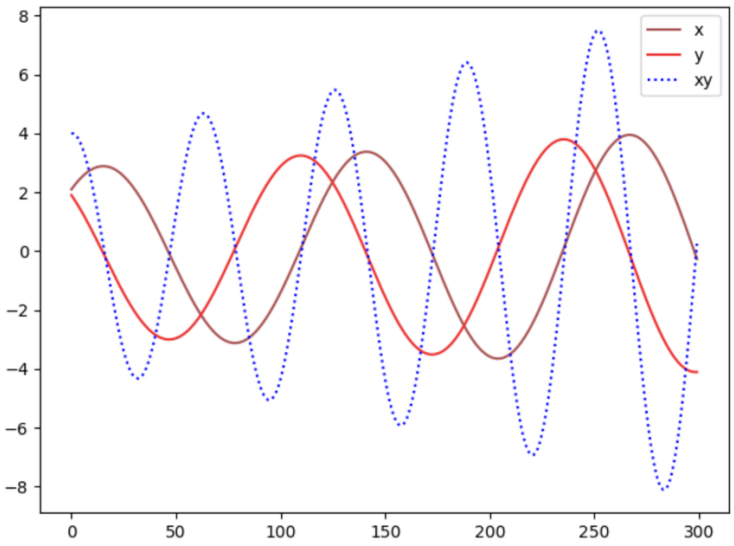


Figure 3.5: Nash Equilibrium
[10]

Here the opponent always countermeasures which makes the model hard to find the nash equilibrium and the model is hard to converge.

3.4.2 Non-Convergence

As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction.
This progression poses a problem for convergence of the GAN as a whole since the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback and its own quality may collapse.
For a GAN, convergence is often a fleeting, rather than stable state.

3.4.3 Mode Collapse

During the training, there are high chances of a generator to collapse which means the generator produces the same output. This happens when the generator learns how to fool the discriminator and it continues to produce outputs of that same class and hence the model fails to generate varied output and we obtain output of the only class which was able to bypass the discriminator as 'real'. At such a stage, it can be said that the generator model has failed.

3.4.4 Vanishing Gradient

When we have a perfect discriminator $D(x) = 1, \forall x \in p_r$ and $D(x) = 0, \forall x \in p_g$. So the loss function L becomes zero and there is no gradient to update the information of loss during the iterations.
Here is a graph which shows that when the discriminator gets better, the gradient vanishes.

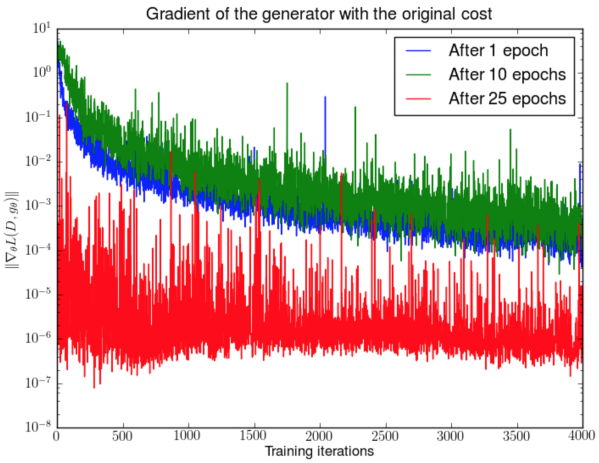


Figure 3.6: Vanishing Gradient
[11]

3.5 Deep Convolutional Generative Adversarial Network (DCGAN)

The above basic GAN had drawbacks like non-convergence, mode collapse, diminished gradient which leads us to deep convolutional generative adversarial network (DCGAN). To start with DCGAN, the dense layers are replaced with convolutional layers. Dense layers use only neurons in all the layers which leads to more training time. Convolutional layers uses weight sharing because of which number of parameters decreases by large number and leads to faster training time. With layers we also use batch normalization and pooling layers. Common Machine Learning benchmarking datasets such as the MNIST database of handwritten digits are suitable for most forms of ANN, due to its relatively small image dimensionality of just (28 × 28). With this dataset a single neuron in the first hidden layer will contain 784 weights ((28× 28× 1) where 1 denotes that MNIST is normalized to just black and white values), which is manageable for most forms of ANN. If we consider a more substantial colored image input of (64 × 64), the number of weights on just a single neuron of the first layer increases substantially to 12,288. Also to be considered is the fact that to deal with this scale of input, the network will also need to be a lot larger than one used to classify color normalized MNIST digits and then the drawbacks of using such models is understood.

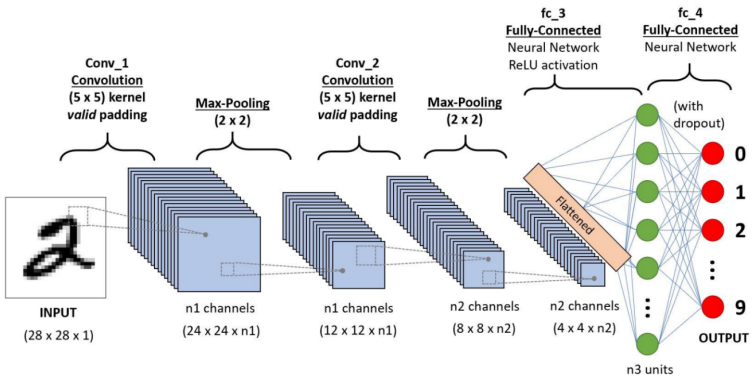


Figure 3.7: Basic Convolutional Neural Network
[12]

- Fig. 3.7 shows the architecture of convolution layers used in DCGAN.
- INPUT [28×28×1] will hold the raw pixel values of the image, in this case an image of width 28, height 28 and with a single grayscale channel.
- **CONV layer:**

  Convolutional layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size $M \times M$. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter ($M \times M$).

  The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image. This may result in volume such as [24×24×12] also depending upon the size of the filter used if we decided to use 12 filters.
- **RELU layer:**

  Fig. 3.8 represents the graph for ReLU activation. This function outputs the input directly if it is positive otherwise it will put zero. This layer will apply an elementwise activation function, such as the max (0, x) thresholding at zero. This leaves the size of the volume unchanged ([24×24×12]).

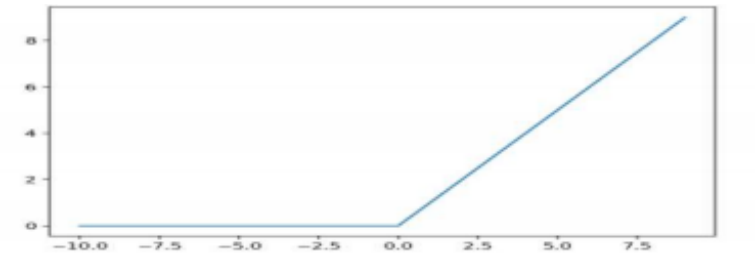


Figure 3.8: ReLU activation function
[13]

- **POOL layer:**

  A Convolutional Layer is followed by a Pooling Layer. It is used to decrease the size of the convolved feature map to reduce the computational costs. In Max Pooling, the largest element is taken from feature map. This is performed by decreasing the connections between layers and independently operates on each feature map. It will then perform a down-sampling operation along the spatial dimensions (width, height), resulting in volume such as [12×12×12].

- Fully Connected layer will compute the class scores, resulting in volume of size [1×1×n] where n denotes the total number of classes present. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

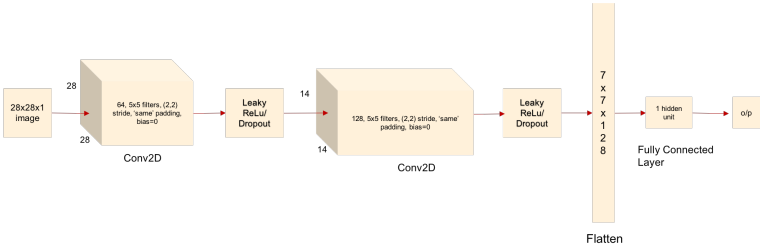


Figure 3.9: Discriminator structure
[14]

In generator we perform up-sampling for which we use a layer called Convolutional2dtranspose which performs the same operations as a convolution layer but in reverse. To be exact, it inserts features into the image to contribute to

upsampling and to produce the final image as given in Fig. 3.10. This layer converts noise to a large data instance (image).

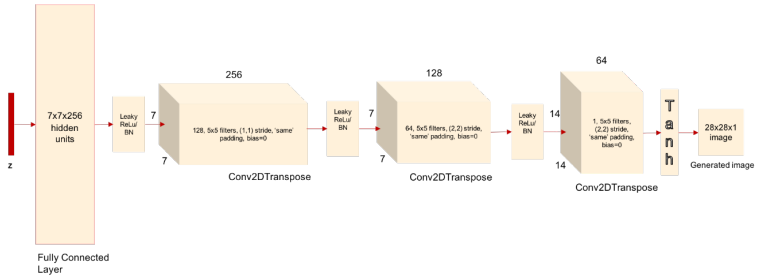


Figure 3.10: Generator structure
[15]

3.5.1 Tanh, LeakyReLU activation functions

Tanh

Tanh is like logistic sigmoid but better. The range of the Tanh function is from (-1 to 1). Tanh is also Sigmoidal(For probability that exists only between the range of 0 and 1, sigmoid is used) (s - shaped).

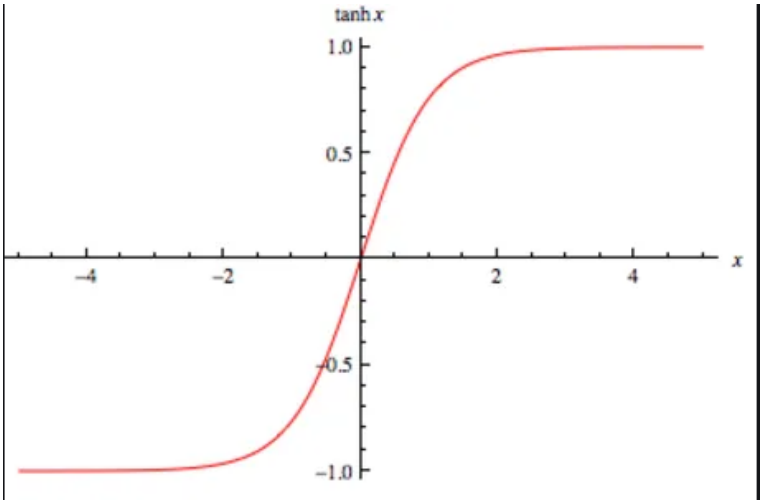


Figure 3.11: Tanh activation function
[16]

LeakyReLU

LeakyReLU stands for Leaky Rectified Linear Unit. It is a non-linear activation function. It is an improved version of the Rectified Linear Unit function in which

the gradient is 0 for $x < 0$, which would deactivate the neurons in less than 0 region. Mathematical expression :-

$$f(x) = \begin{pmatrix} 0.01x, x < 0 \\ x, x \geq 0 \end{pmatrix} \tag{3.5}$$

Instead of defining the ReLU function as 0 for negative values of x, it is defined as an extremely small linear component of x. The gradient of the left side of the graph comes out to be a non zero value after this modification. So, we don't encounter dead neurons in that region.

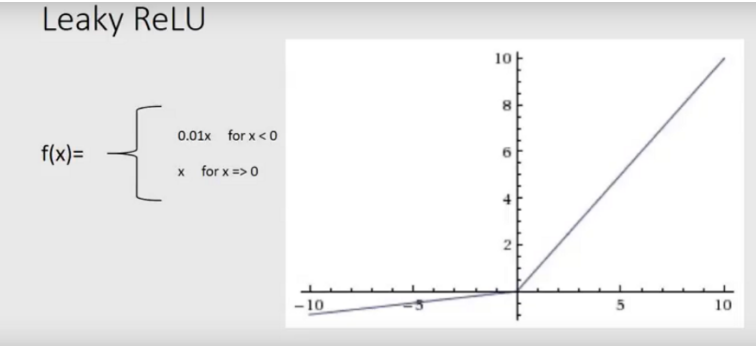


Figure 3.12: LeakyReLU activation function
[17]

3.5.2 Batch Normalization

Training deep neural networks with multiple layers is challenging as they can be sensitive to the initial random weights and configuration of the learning algorithm. One possible reason for this difficulty is the distribution of the inputs to layers deep in the network may change after each mini-batch when the weights are updated. This can cause the learning algorithm to forever chase a moving target. Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

3.5.3 Binary Cross Entropy loss function

Binary Cross entropy compares each and every predicted probability to the real class values which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. This tells us how close or far our predicted value is from the actual value.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1 - y_i) log(1 - p(y_i)) \tag{3.6}$$

Here y is the label (0 for class of fake image and 1 for class of real image of X-ray) and $p(y)$ is the predicted probability of the point being real for all N images. From this formula, for each real image ($y = 1$), equation adds $log(p(y))$ to the loss, that is, the log probability of it being real. It also adds $log(1 - p(y))$, that is, the log probability of it being fake, for each fake image ($y = 0$).

3.6 Statistical Distance

It is a general way of calculating the difference between two different probability distributions for a random variable. There are many instances where we want to compare two probability distributions in which we may have a single variable and two different probability distributions for the variable. One approach is to calculate the distance or calculate the divergence between the two probability distributions.

A divergence is a measure but its not symmetrical. Divergence is a scoring of how one probability distribution is different from the other. Divergence while calculating the distance between two probability distributions, say P and Q is different than the divergence while calculating the distance between Q and P in some cases. Divergence plays a very important role in calculations and in Machine Learning. The divergence is also used to understand the complex GAN model.

Entropy is the measure of uncertainty which says how disordered a system is or the degree of randomness of in a set of data. When we have a higher entropy, there are less chances of finding the pattern is the data or we can say less predictability. [18]

3.6.1 Kullback-Leibler (KL) Divergence

The KL divergence score, quantifies how much one probability distribution differs from another probability distribution. It is also referred to as relative entropy. It is a key component of Gaussian Mixture models. The equation of the KL Divergence is as follows:

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx \tag{3.7}$$

If the probability distribution p and q are in the same finite set then $D_{KL}(p||q)$ is the expected amount of information gained when the observer considered probability distribution as q, when in reality it is p. If the probability distributions are not distinct than each other then the divergence is 0. If you have a lower value of the divergence the approximation or the closeness of the model is better. [19]

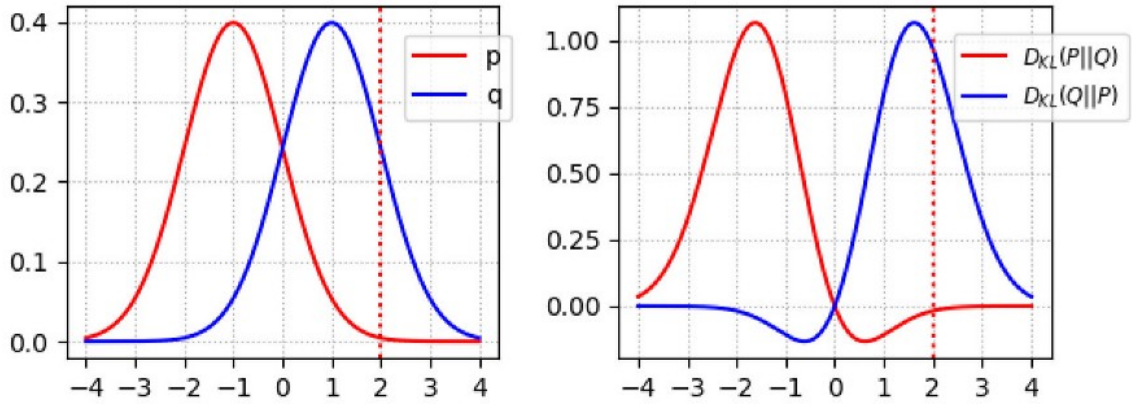


Figure 3.13: KL Divergence
[20]

Application:

- KL divergence has its direct application in the VAE and also in deep learning applications, for example in the image reconstruction or regeneration of images similar to the input images.

Drawbacks:

- As mentioned earlier, the KL divergence indicates the divergence or the score

to be zero if the entropies are strictly equal this says the the KL divergence does not have a well defined upperbound.

- It is does not satisfy the triangle inequality.
- It is non-linear.
- It varies in the range of zero to infinity.

3.6.2 Jensen-Shannon divergence

It is an another way of calculating the distance between two probability distribution. JS divergence is an extension of the KL divergence. It calculates the symmetrical score and the distance measure from one probability to another. It is symmetric i.e., $JS(P||Q) = JS(Q||P)$. JS divergence can be calculated as follows:

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2}) \tag{3.8}$$

It provides a more smoothed and normalised version of KL divergence. It has the score ranging from 0-1.

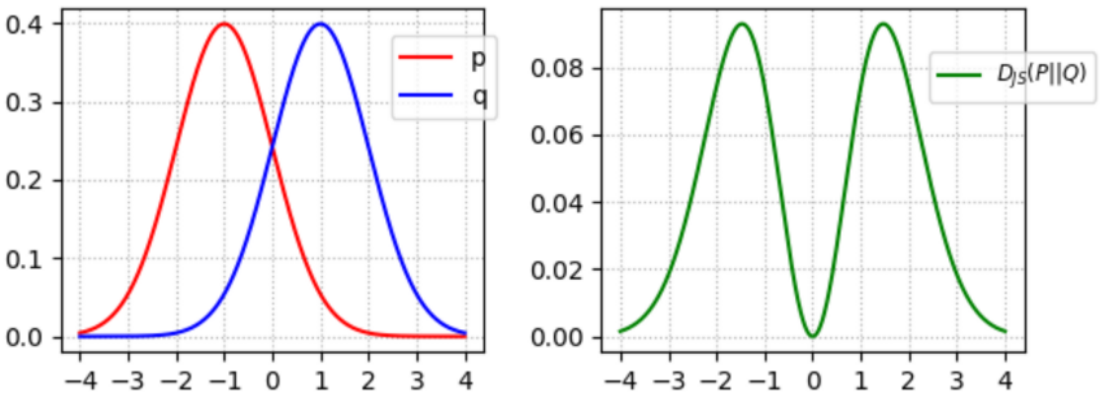


Figure 3.14: JS Divergence
[21]

Drawbacks:

- It gives a constant value of log 2 when the distributions are unlike. For example, if there is a point on p for which the corresponding point on q has a

value of 0, the substitution of these produces a JS divergence value of log 2. As the derivative of the constant value is 0, the generator does not get trained.

Hence we move on to Wasserstein distance which horizontally calculates the distance between the two probability distributions. [22], [23]

3.6.3 Wasserstein distance

Wasserstein distance is also the measure of the distance between two probability density functions horizontally. It as known as Earth Mover's distance. It can be formally interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of other distribution.

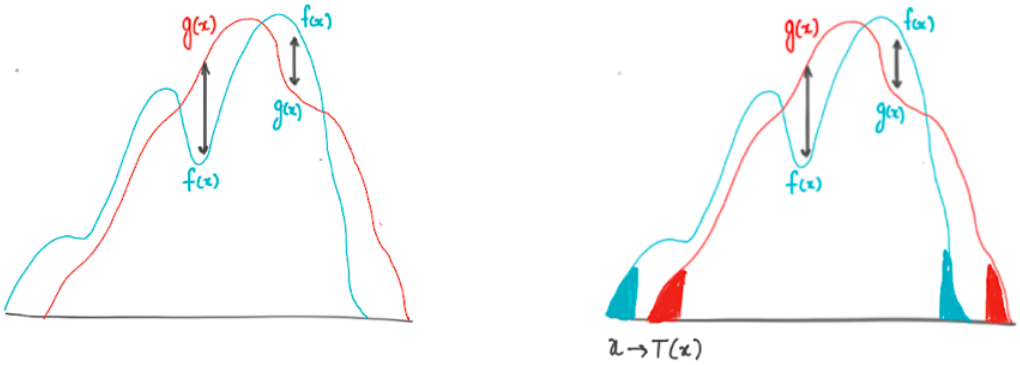


Figure 3.15: Wasserstein distance
[24]

$$W(\mathbb{P}_r.\mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r.\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\Big[\ ||\ x - y\ ||\ \Big] \tag{3.9}$$

- $\mathbb{P}_r$ - Real probability distribution.
- $\mathbb{P}_g$ - Generated Probability distribution.
- $\Pi(\mathbb{P}_r.\mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$.
- inf - Infimum.

- $\mathbb{E}$ - expected value.
- $\mathbb{E}_{(x,y)}$ - expected value of x , y sampled from distribution gamma.

$\gamma(x, y)$ indicates how much 'mass' must be transported from x to y in order to transform the distributions P_r into the distribution P_g. It calculates the distance between two points (x and y) in the horizontal axis. But these points are sampled from γ and γ belongs to the transport plan which gives the minimum cost of all the transport plans available so we call it infimum.
In order to match the two probability distributions, there are various ways to shift the probability points and one such shifting plan is called transport plan. There are various such transport plans and the minimum(infimum) value out of these is selected as the cost of the loss function.

3.7 Wasserstein Generative Adversarial Network (WGAN)

It is an extension of the GAN that seeks an alternate way of training the generator model to better approximate the distribution of data observed in a given training dataset.
Instead of using a discriminator to classify or predict the probability of generated images as being real or fake, the WGAN changes or replaces the discriminator model with a critic that scores the realness or fakeness of a given image.
This change is motivated by a theoretical argument that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples.
The benefit of the WGAN is that the training process is more stable and less sensitive to model architecture and choice of hyperparameter configurations. Perhaps most importantly, the loss of the discriminator appears to relate to the quality of images created by the generator.
The equation for the Wasserstein distance is highly intractable. Using the

Kantorovich Rubinstein duality and simplifying it further, we get equation as shown in Eq. 3.10.

$$W(p_r, p_g) = \frac{1}{K} \sup_{||f||_L \leq K} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)] \tag{3.10}$$

Here, *sup*(Supremum) is the least upper bound, K is known as Lipschitz constant for function f and f is a K-Lipschitz function following this constraint

$$| f(x_1) - f(x_2) | \leq K | x_1 - x_2 | \tag{3.11}$$

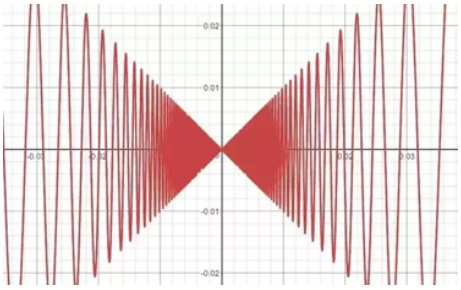


Figure 3.16: Lipschitz Constraint [25]

So to calculate the Wasserstein distance, we just need to find a K-Lipschitz function. Like other deep learning problem, we can build a deep network to learn it. Indeed, this network is very similar to the discriminator D, just without the sigmoid function and outputs a scalar score rather than a probability. This score can be interpreted as how real the input images are. In reinforcement learning, we call it the value function which measures how good a state (the input) is. We rename the discriminator to critic to reflect its new role. Suppose this function f comes from a family of K-Lipschitz continuous functions $\{f_\omega\}_{\omega \in W}$ parameterized by ω. In the modified Wasserstein-GAN, the "discriminator" model is used to learn ω to find a good fω and the loss function is configured as measuring the Wasserstein distance between P_g and P_r .

$$L(p_r, p_g) = W(p_r, p_g) = max_{\omega \in W} E_{x \sim P_r}[f_\omega(x)] - E_{z \sim P_r(z)}[f_\omega[g_\theta(z)]] \tag{3.12}$$

Thus the ”discriminator” is not a direct critic of telling the fake samples apart

from the real ones anymore. Instead, it is trained to learn a K-Lipschitz continuous function to help compute Wasserstein distance. As the loss function decreases in the training, the Wasserstein distance gets smaller and the generator model's output grows closer to the real data distribution. However, there is one major thing missing f has to be a K-Lipschitz function.

3.7.1 Weight clipping

To enforce the constraint, WGAN applies a very simple clipping to restrict the maximum weight value in f, i.e., the weights of the discriminator must be within a certain range controlled by the hyperparameters c.

$$\omega \leftarrow +\alpha.(\omega, g_\omega) \tag{3.13}$$

$$\omega \leftarrow clip(\omega, -c, c) \tag{3.14}$$

The difficulty in WGAN is to enforce the Lipschitz constraint. Clipping is simple but it introduces some problems.

Problems of WGAN:

- It usually leads to vanishing or exploding gradients depending upon the value of c. Too small a value leads to vanishing gradients while too large a value may lead to exploding gradient.
- Clipping usually leads to the weights being pushed towards the two extremes i.e., the extremes of the clipping range.
- Weight clipping leads to capacity underuse as the critic models only basic approximations to the optimal function and clipping ignores higher moments of the data distribution.

The model may still produce poor quality images and does not converge, in particular when the hyperparameter c is not tuned correctly. The model

performance is very sensitive to this hyperparameter. The weight clipping behaves as a weight regulation. It reduces the capacity of the model f and limits the capability to model complex functions [26].

3.8 Wasserstein GAN with gradient penalty (WGAN-GP)

WGAN-GP uses gradient penalty instead of the weight clipping to enforce the Lipschitz constraint.

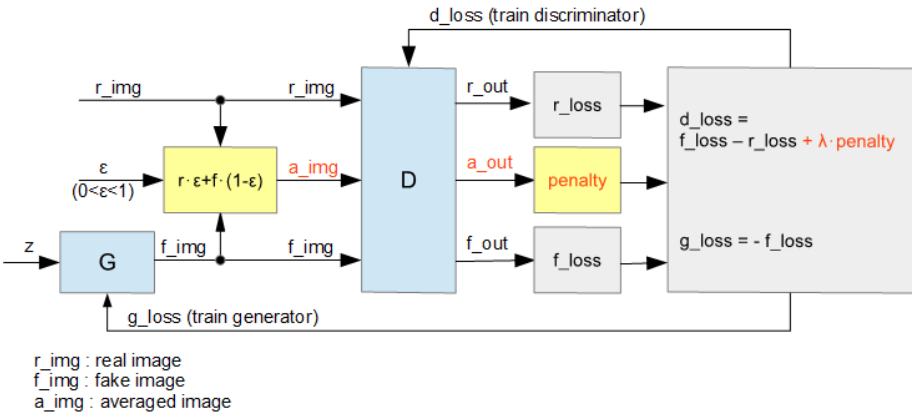


Figure 3.17: WGAN-GP Architechture [27]

The block diagram of WGAN-GP clearly shows its working. The critic is fed with 3 inputs namely:

1. Real image(real data).
2. Generated/fake image(generated by generator).
3. Interpolated/averaged image.

The averaged image contains a part of real image and a part of generated image and this partition is decided by the variable ϵ which has a range of 0 to 1. For example, an ϵ value of 0.6 indicates that averaged image has 60% of real image and remaining 40% of fake image. This ensures that the critic doesn‘t purely learn to distinguish between real and fake which may lead to overfitting but also is able

to learn to differentiate between the two simultaneously. The loss output by the averaged image is the penalty term which is the basis of this learning technique. The final critic loss is the difference between loss for fake and real added with the penalty term which is multiplied with a trainable hyperparamter λ.

3.8.1 Gradient penalty

A differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere. Points interpolated between the real and generated data should have a gradient norm of 1 for f. So instead of applying clipping, WGAN-GP penalizes the model if the gradient norm moves away from its target norm value 1.

$$L = \underbrace{E_{\tilde{x}\sim P_g}[D(\tilde{x})] - E_{x\sim P_r}[D(x)]}_{\text{Original critic loss}} + \underbrace{\lambda E_{\hat{x}\sim P_{\hat{x}}}[(||\ \nabla_{\hat{x}} D(\hat{x})\ ||_2 - 1^2)]}_{\text{Gradient Penalty}} \tag{3.15}$$

where $\hat{x}$ is sampled from $\tilde{x}$ and x as explained in section 3.8 above and t uniformly sampled between 0 and 1

λ is set to 10. The point x used to calculate the gradient norm is any points sampled between the P_g and P_r .

Batch normalization is avoided for the critic (discriminator). Batch normalization creates correlations between samples in the same batch. It impacts the effectiveness of the gradient penalty which is confirmed by the experiments performed by the author. [5]

Some new cost functions add gradient penalty to the cost function. Some are purely based on empirical observation that models misbehave when the gradient increases. However, gradient penalty adds computational complexity that may not be desirable but it does produce some higher-quality images. The computational complexity can be attributed to the fact that the loss function and backpropagation along with averaging is applied to every image instead of computation after a complete batch unlike GANs and DCGANs. The major advantage of WGAN-GP is its convergency. It makes training more stable and therefore easier to train. As WGAN-GP helps models to converge better, we can use a more complex model like a deep ResNet for generator and discriminator.

3.9 Dataset

We have used the PA view X-ray images of patients having COVID-19 and pneumonia. Also we have taken the X-ray images of patients have no lung disease. The images were of size 256×256. The original images were taken from Git Hub and a combination chest X-ray images from Kaggle and the COVID-19 Chest X-ray dataset collected by Dr. Joseph Paul Cohen of the University of Montreal.The original dataset consisted of 555 chest X-ray images of COVID-19, 3100 chest X-ray images of pneumonia and 2800 normal chest X-ray images [28]. As we had a shortage of chest X-ray images of COVID-19 we passed it through an augmentor. As mentioned earlier we can increase the number of images by passing it in an augmmentor. We tilted the images from 3 to 5 degrees from left to right and also horizontally flipped the image. After augmentation we had a total of 2055 chest X-ray images of COVID-19.

Figure 3.18: COVID-19 image 1

Figure 3.19: COVID-19 image 2

Figure 3.20: Normal

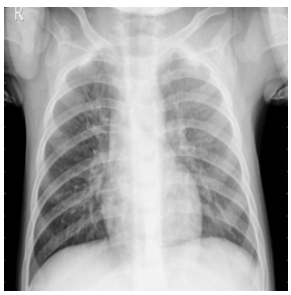


Figure 3.21: Pneumonia

Chapter 4

Simulation and Experimental Results

4.1 Model Architecture

4.1.1 The Generator

The generator model consists of four convolutional layers represented using the Conv2D function. The four layers contains 256, 128, 64, 32 neurons respectively. The input to generator model are the latent points i.e random gaussian noise are given as the input. Using UpSampling2D function we double the input dimension and it helps us to avoid the checkerboard artifact. A batch normalization is used after each convolutional layer and before the activation function using BatchNormalization function. It is used to make the neural network more faster and more stable. LeakyReLU is used as the activation function in all the four pair of layers. LeakyReLU outputs the input if input is positive and for negative inputs it gives a small negative slope. Lastly using the Cropping2D function the generated images are cropped to the original size i.e., 256*256. In the output layer Tanh activation function is used. Graph of Tanh activation is shown in Fig. 3.11. In Tanh the positive, negative and zero inputs are strongly matched with positive, negative and zero respectively, just similar to sigmoid activation.

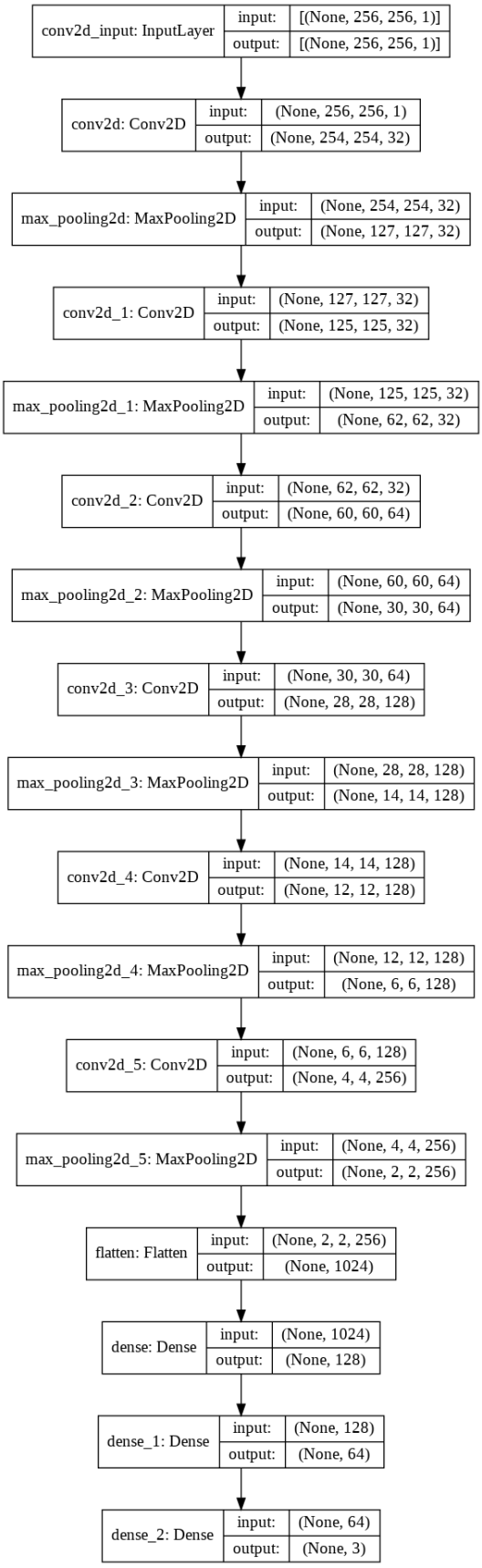


Figure 4.1: Generator Architecture

4.1.2 The Discriminator

The discriminator model shown in Fig. 4.2 is bit similar to the generator model. It performs downsampling. In the discriminator model we have used six pairs of convoultional layers and LeakyReLU activation function. The convolutional layers contain 1, 16, 32, 64, 128, 256 neurons respectively. Input to this model is the output of generator of size 256*256*1. We used ZeroPadding2D function to pad zero's in the input image to maintain the size of the image at the output. A Dropout function is used to randomly drop x % of neurons. This is used so that the neurons won't learn the parameters of the image. It can avoid overfitting. Using Sigmoid activation and one neuron for the output dense layer, the discriminator outputs whether the input is real or fake.

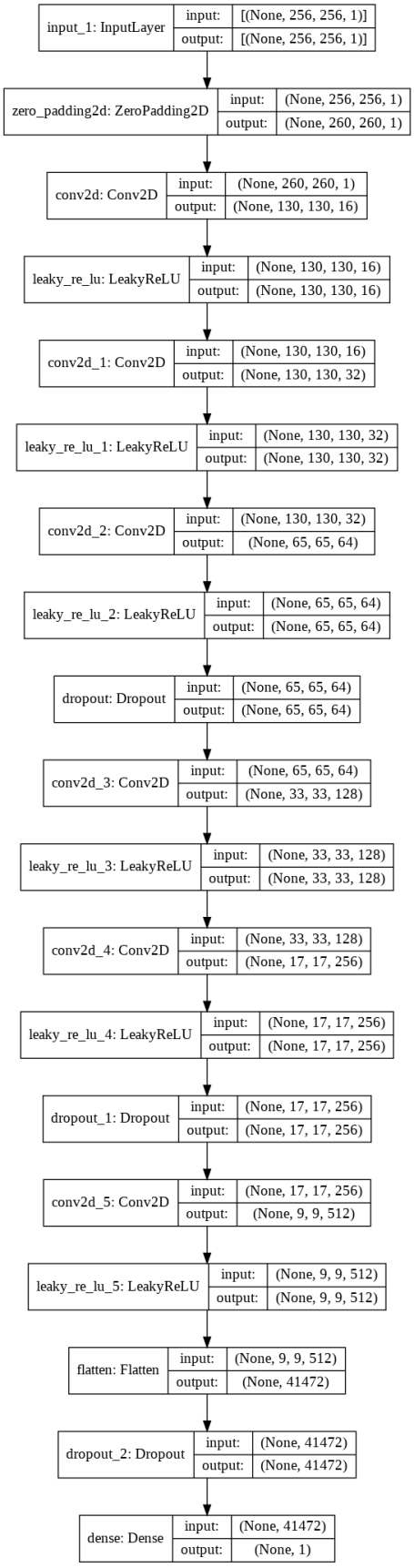


Figure 4.2: Discriminator Architecture

4.2 Output of Generative Models

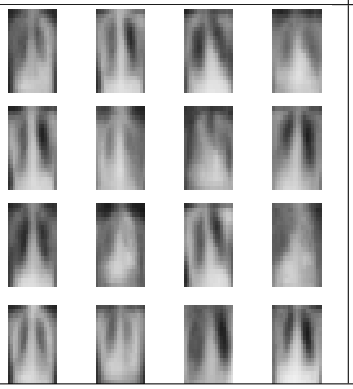
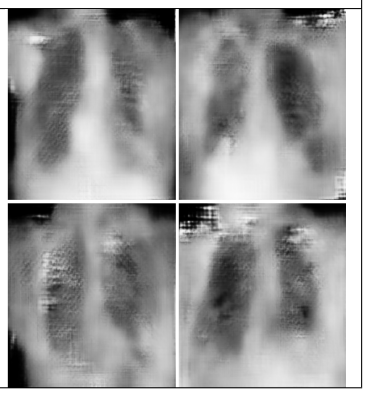
GAN	DCGAN
	
WGAN	**WGAN-GP**
	

Table 4.1: Generative Model output

4.3 Convolution Neural Network

We trained the convolution neural network model using the Chest X-ray dataset, the augmented images and the generated images from WGAN-GP model. All these images were further divided into test set and validation set. Test set is the data which the model has never seen before and validation set is a sample of data held back from training your model that is used to give an estimate of model skill while tuning model's hyperparameters. Further the model consists of three pairs of convolution layers and pooling layers. Pooling layers are used to reduce the number of parameters i.e., it summarizes the features present in a region of the feature map generated by a convolution layer. The three convolution layers have 32, 64 and 128 neurons respectively and Rectified Linear activation function

(ReLU) is used. A dropout function is added to drop 20% of the neurons so that none of the neuron learn the parameters. At last dense layers are used as output layers. An optimizer helps the model to reduce its losses. Over here Adam optimizer is used and also categorical cross-entropy is used as loss function.

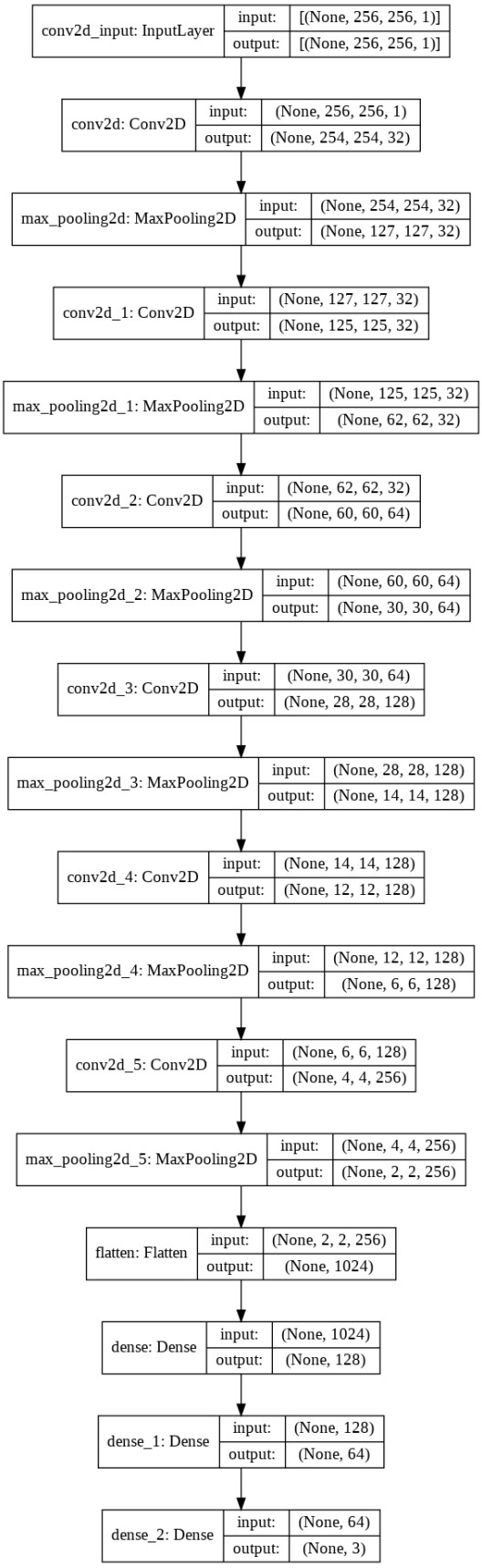


Figure 4.3: CNN Architecture

A confusion matrix helps us to get the performance of the model. It is evaluated on the test set of images. The labels of the images that are going to be predicted are known and are then compared to the label given to each image during prediction. We applied this on our test dataset containing 1000 images of each class. As shown in the Fig. 4.4 below, the predicted images are as follows:

- All the COVID-19 images were predicted correctly.
- 996 normal images were predicted correctly as normal and 4 images were wrongly predicted as Pneumonia.
- 966 Pneumonia images were predicted correctly, 31 were wrongly predicted as normal and 3 were wrongly predicted as COVID-19.

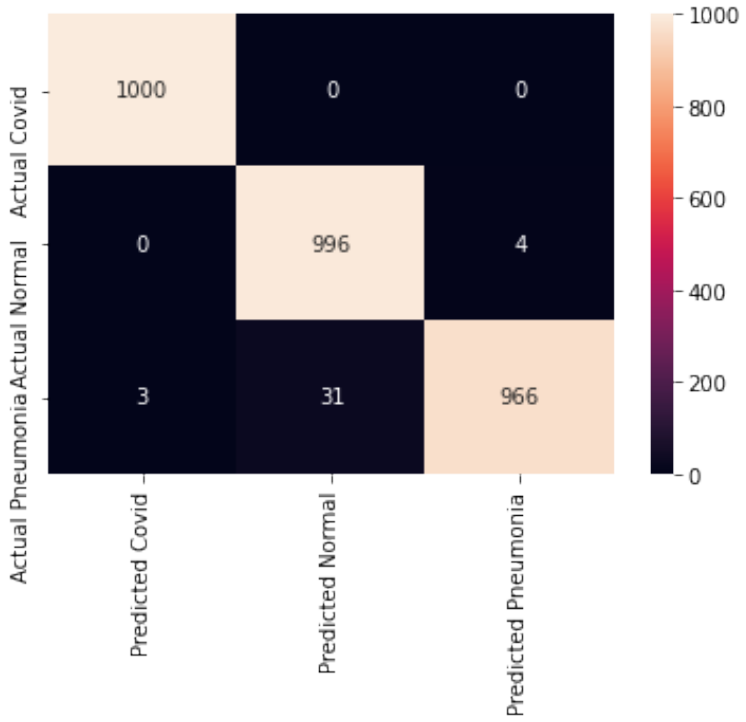


Figure 4.4: Confusion Matrix

Fig. 4.5 and Fig. 4.6 shows the accuracy and loss of the model respectively. According to the graph when validation set graph is about to cross the training set graph then we can conclude that the model is ready for classification. If the validation graph stops far away before crossing the training the graph then the model is about to underfit and if it crosses and continues then model is likely to

overfit.

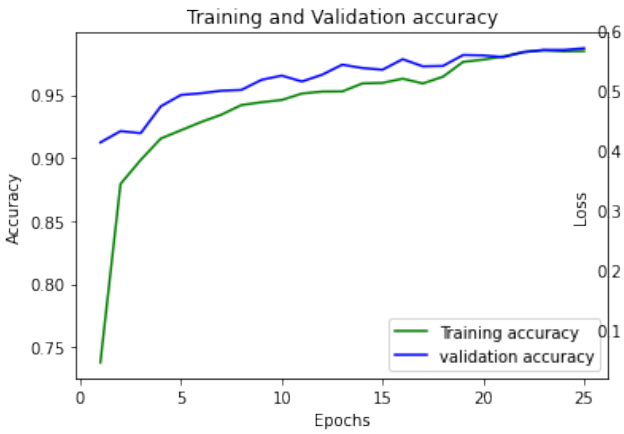


Figure 4.5: Accuracy

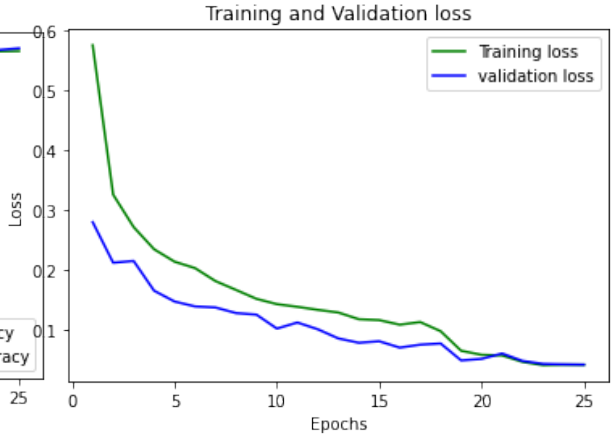


Figure 4.6: Loss

Table 4.2 determines the F1 score of the model. F1 Score is needed when a balance is to be sought between Precision and Recall.

Classes	Precision	Recall	f-1 score
COVID-19	1.00	1.00	1.00
NORMAL	0.97	1.00	0.98
PNEUMONIA	1.00	0.97	0.98

Table 4.2: Precision, Recall and F1 score

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

We learnt the concepts of CNN, GAN, DCGAN, WGAN, WGAN-GP and implemented on various sample dataset and also on our final X-ray dataset. We used our own of architecture in GAN, DCGAN, WGAN, WGAN-GP but received partial output for WGAN-GP and better results can be obtained by using variations of ResNet model. In ResNet model we can overcome the problem of vanishing gradient and improve the WGAN-GP model further. We received 98% accuracy in our CNN model.

5.2 Future Scope

If there is another lung disease that can be detected through chest X-ray of lungs we can modify the architecture of our model and we can predict the infection present in the lungs. This model will be then deployed to a website where the user has to just insert the picture of the chest X-ray and they will find out if they have specific virus present in their lungs.

References

[1] https://towardsdatascience.com

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio, *"Generative Adversarial Networks"*, Jun 2014.

[3] Alec Radford, Luke Metz and Soumith Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, Nov 2015

[4] Martin Arjovsky, Soumith Chintala and LÃ©on Bottou, *"Wasserstein GAN"*, Jan 2017

[5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin and Aaron Courville, *"Improved Training of Wasserstein GANs"*, Mar 2017.

[6] `https://jonathan-hui.medium.com/gan-some-cool-applications-of-gans-4c9ecca35900`

[7] `https://developers.google.com/machine-learning/gan/images/generative_v_discriminative.png`

[8] `https://developers.google.com/machine-learning/gan/discriminator`

[9] `https://developers.google.com/machine-learning/gan/generator`

[10] `https://miro.medium.com/max/3000/1*6TSfFPMoHqkiQpmZYsVulw.png`

[11] https://lilianweng.github.io/lil-log/
assets/images/GAN_vanishing_gradient.png

[12] https://blog.paperspace.com/deploying-
deep-learning-models-flask-web-python/

[13] https://machinelearningmastery.com/rectified-
linear-activation-function-for-deep-learning-neural-networks/

[14] https://medium.com/@ramyahrgowda/dcgan
-implementation-in-keras-explained-e1918fc930ea

[15] https://medium.com/@ramyahrgowda/dcgan
-implementation-in-keras-explained-e1918fc930ea

[16] https://mathworld.wolfram.com/HyperbolicTangent.html

[17] https://towardsdatascience.com/activation
-functions-in-neural-networks-58115cda9c96

[18] https://machinelearningmastery.com

[19] https://analyticsindiamag.com

[20] https://miro.medium.com/max/3000/1*34g2vpW_G_CO4pX41gaGsw.jpeg

[21] https://miro.medium.com/max/875/1*Gx-h9i3oaTUAjFUGXvw9dg.png

[22] https://yongchaohuang.github

[23] https://lilianweng.github

[24] https://kowshikchilamkurthy.medium.com/wasserstein-
distance-contraction-mapping-and-modern-rl-theory-93ef740ae867

[25] https://www.quora.com/What-does-Lipschitz-continuity-imply-and-how-does-
a-Lipschitz-continuous-function-differ-from-a-normal-continuous-function

[26] https://machinelearningmastery.com

[27] https://www.kaggle.com/amanooo/wgan-gp-keras

[28] https://github.com/ieee8023/covid-chestxray-dataset