VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

An Internship Project Report on

"PNEUMONIA DETECTION USING CYCLIC GAN"

Submitted in partial fulfillment of the requirements as a part of the

AI/ML INTERNSHIP

(NASTECH)

For the award of degree of

Bachelor of Engineering
in
Information Science and Engineering

Submitted by

ADITYA D
1RN19IS011

GURUPRASAD HS
1RN19CS053

Internship Project Coordinators

Dr. R Rajkumar
Associate Professor
Dept. of ISE, RNSIT

Mrs. Sunitha K
Assistant Professor
Dept. of ISE, RNSIT

Department of Information Science and Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post,
Bengaluru – 560 098

2022 -2023

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post,

Bengaluru – 560 098

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project report entitled ***PNEUMONIA DETECTION USING CYCLE GAN*** has been successfully completed by **ADITYA D** bearing USN **1RN19IS011** and **GURUPRASAD HS** bearing USN **1RN19CS053**, presently VI semester students of **RNS Institute of Technology** in partial fulfillment of the requirements as a part of the ***AI/ML Internship (NASTECH)*** for the award of the degree of ***Bachelor of Engineering in InformationScience and Engineering*** under **Visvesvaraya Technological University, Belagavi** during academic year **2022 – 2023**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report and deposited in the departmental library. The internship project report has been approved as it satisfies the academic requirements as a part of AI/ML Internship.

Dr. R Rajkumar	**Mr. Nirmalkumar S Benni**	**Dr. Suresh L**
Coordinator	Guide	Professor and HoD
Associate Professor	Assistant Professor	Dept of ISE
Dept of ISE	Dept of ISE	

Dr M K Venkatesha

Principal

RNSIT

External Viva

Name of the Examiners	**Signature with date**
1. _____	_____
2. _____	_____

NASTECH

Microsoft Partner Network

CERTIPORT

AUTHORIZED TESTING CENTER

CERTIFICATE

This is to certify that **Aditya D** bearing USN: **1RN19IS011** from RNSIT has taken part in internship training on the **Artificial Intelligence & Machine Learning and successfully completed** project work" **Pneumonia Detection with chest x-rays using cyclic GAN** "in New Age Solutions Technologies (NASTECH) from March 2022 to June 2022.

Aditya D has shown great enthusiasm and interest in the project. The incumbents' conduct and performance were found satisfactory during all phases of the project.

Thank you very much.

Sincerely Yours,

Deepak Garg

Founder

Address: Block No.10, Kailashnagar, Highland Residency, Kolshet Road, Thane (W)- 400608, Mumbai, Maharashtra.
Mobile Number: 8591132820| E-mail: deeprgarg@nastech.in

ABSTRACT

This study is aimed at evaluating the effectiveness of the state-of-the-art pretrained Generative Adversarial Networks (GANs) on the automatic diagnosis of Pneumonia from chest X-rays (CXRs). The dataset used in the experiments consists of 1200 CXR images from individuals with Pneumonia, 1345 CXR images from individuals with viral Pneumonia, and 1341 CXR images from healthy individuals. In this paper, the effectiveness of artificial intelligence (AI) in the rapid and precise identification of Pneumonia from CXR images has been explored based on different pretrained deep learning algorithms and fine-tuned to maximise detection accuracy to identify the best algorithms. The results showed that deep learning with X-ray imaging is useful in collecting critical biological markers associated with Pneumonia infections. The basic idea is we have generators for both of the mappings with a U-Net style architecture (which forces them to learn something from the original pixels). We then have discriminators which determine if the images we have created are real or fake. Finally we have the cycle-consistent loss of using both the forward and backward back-to-back which should give us the original images. The outstanding performance of Cyclic GAN can significantly improve the speed and accuracy of Pneumonia diagnosis. However, a larger dataset of Pneumonia X-ray images is required for a more accurate and reliable identification of Pneumonia infections when using deep transfer learning.

ACKNOWLEDGMENT

The fulfillment and rapture that go with the fruitful finishing of any assignment would be inadequate without the specifying the people who made it conceivable, whose steady direction and support delegated the endeavors with success.

We would like to profoundly thank **Management** of **RNS Institute of Technology** for providing such a healthy environment to carry out this AI/ML Internship Project.

We would like to express our thanks to our Principal **Dr. M K Venkatesha** for his support and for inspiring us towards the attainment of knowledge.

We wish to place on record our words of gratitude to **Dr. Suresh L,** Professor and Head of the Department, Information Science and Engineering, for having accepted to patronize us in the right direction with all his wisdom.

We like to express our profound and cordial gratitude to our Internship Project Coordinator**, Dr. R Rajkumar,** Associate Professor, Department of Information Science and Engineering for their valuable guidance, constructive comments, continuous encouragement throughout the Internship Work and guidance in preparing report.

We would like to thank all other teaching and non-teaching staff of Information Science & Engineering who have directly or indirectly helped us to carry out the Mini Project Work.

Also, we would like to acknowledge and thank our parents who are source of inspiration and instrumental in carrying out this Mini Project Work.

ADITYA D
USN:1RN19IS011

GURUPRASAD HS
USN:1RN19CS053

TABLE OF CONTENTS

LIST OF FIGURES AND EQUATIONS

Figures

Equations

Chapter 1

INTRODUCTION

1.1 ORGANIZATION/INDUSTRY

1.1.1 COMPANY PROFILE

NASTECH is formed with the purpose of bridging the gap between Academia and Industry Nastech is one of the leading Global Certification and Training service providers for technical and management programs for educational institutions. We collaborate with educational institutes to understand their requirements and form a strategy in consultation with all stakeholders to fulfill those by skilling, reskilling and upskilling the students and faculties on new age skills and technologies.

1.1.2 DOMAIN/TECHNOLOGY

The domain chosen for our project is AI/ML. Machine learning, the fundamental driver of AI, is possible through algorithms that can learn themselves from data and identify patterns to make predictions and achieve your predefined goals, rather than blindly following detailed programmed instructions, like in traditional computer programming. This technology allows the machine to perceive, learn, reason and communicate through observation of data, like a child that grows up and acquires knowledge from examples. Machines also have the advantage of not being limited by our inherent biological limitations. With machine learning, manufacturing companies have increased production capacity up to 20%, while lowering material consumption rates by 4%.

Nowadays, the revolutionary AI technology evolved from rule-based expert systems to machine learning and more advanced subcomponents such as deep learning (learning representations instead of tasks), artificial neural networks (inspired by animal brains) and reinforcement learning (virtual agents rewarded if they made good decisions).

The AI can master the complexity of the intertwining industrial processes to enhance the whole flow of production instead of isolated processes. This enormous cognitive capacity gives the AI the ability to consider the spatial organization of plants and the timing constraints of live production. Another key advantage is the capability of AI algorithms to think

probabilistically, with all the subtlety this allows in edge cases, instead of traditional rule basedmethods that require rigid theories and a full comprehension of problems.

1.1.3 Department

R.N.Shetty Institute of Technology (RNSIT) established in the year 2001, is the brain-child of the Group Chairman, Dr. R. N. Shetty. The Murudeshwar Group of Companies headed by Sri. R. N. Shetty is a leading player in many industries viz construction, manufacturing, hotel, automobile, power & IT services and education. The group has contributed significantly to the field of education. A number of educational institutions are run by the

R. N. Shetty Trust, RNSIT being one amongst them. With a continuous desire to provide quality education to the society, the group has established RNSIT, an institution to nourish and produce the best of engineering talents in the country. RNSIT is one of the best and top accredited engineering colleges in Bengaluru.

1.2 PROBLEM STATEMENT

1.2.1 Existing System and their Limitations

The Current system uses x-rays for diagnosis but analyzing an x-ray is not easy the readings may be inaccurate as well. Models Using Traditional Methods are not highly accurate but accuracy matters the most in terms of medical uses especially in x-rays.

1.2.2 Proposed Solution

To eliminate the drawbacks stated above, Cyclic Generative Adversarial Networksalgorithm can be used to implement the model which helps in detecting Pneumonia using the chest X-Rays.

1.2.3 Problem formulation

A **generative adversarial network** (**GAN**) is a class of machine learning frameworks in which two neural networks contest with each other in the form of a zero-sum game, where one agent's gain is another agent's loss. The core idea of a GAN is based on the "indirect" training through the discriminator, another neural network that can tell how "realistic" the input seems, which itself is also being updated dynamically. This means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator.

Chapter 2

REQUIREMENT ANALYSIS, TOOLS &TECHNOLOGIES

2.1 Hardware and Software Requirements

2.1.1 Hardware Requirements:

- Processor: Pentium IV or above
- RAM: 8 GB or more
- Hard Disk: 2GB or more

2.1.2 Software Requirements:

- Operating System: Windows 7 or above
- IDE: Google Colab/Kaggle Notebook(recommended)/Jupyter Notebook (Make sure you have enabled gpu)

2.2 Tools/Languages/Platforms

- Python

CHAPTER 3

Design And Implementation

3.1 Problem Statement

This study is aimed at evaluating the effectiveness of the state-of-the-art pretrained Generative Adversarial Networks (GANs) on the automatic diagnosis of PNEUMONIAfrom chest X-rays (CXRs)

Cyclic Generative Adversarial Method (Cycle GAN) has been used for this project

The following functions have been defined and used:

1. build_generator: U-Net Generator
2. conv2d: Layers used during downsampling
3. deconv2d: Layers used during upsampling
4. _ds_pipe: downsampling
5. build_discriminator: U-Net Discriminator

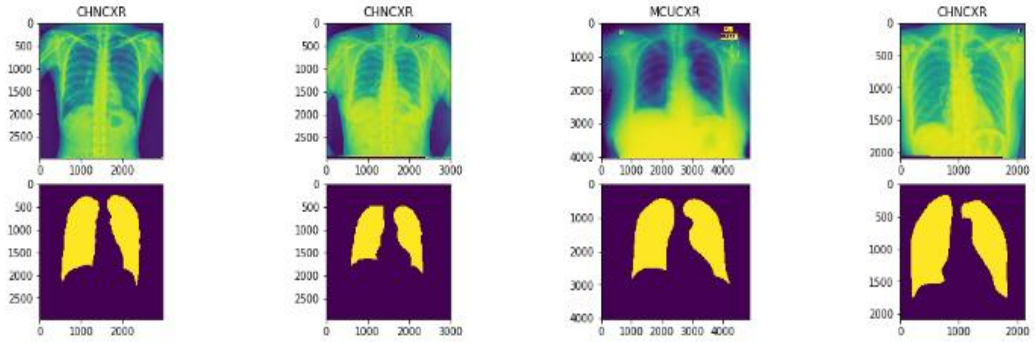


Fig 3.1 CXR Dataset

Out[7]:

modality	source	image_id	CXR_png	masks
96	CHNCXR	0097	../input/chest-xray-masks-and-labels/data/Lung...	../input/chest-xray-masks-and-labels/data/Lung...
654	CHNCXR	0655	../input/chest-xray-masks-and-labels/data/Lung...	../input/chest-xray-masks-and-labels/data/Lung...
392	CHNCXR	0393	../input/chest-xray-masks-and-labels/data/Lung...	../input/chest-xray-masks-and-labels/data/Lung...

Figure 3.2 Description of dataset

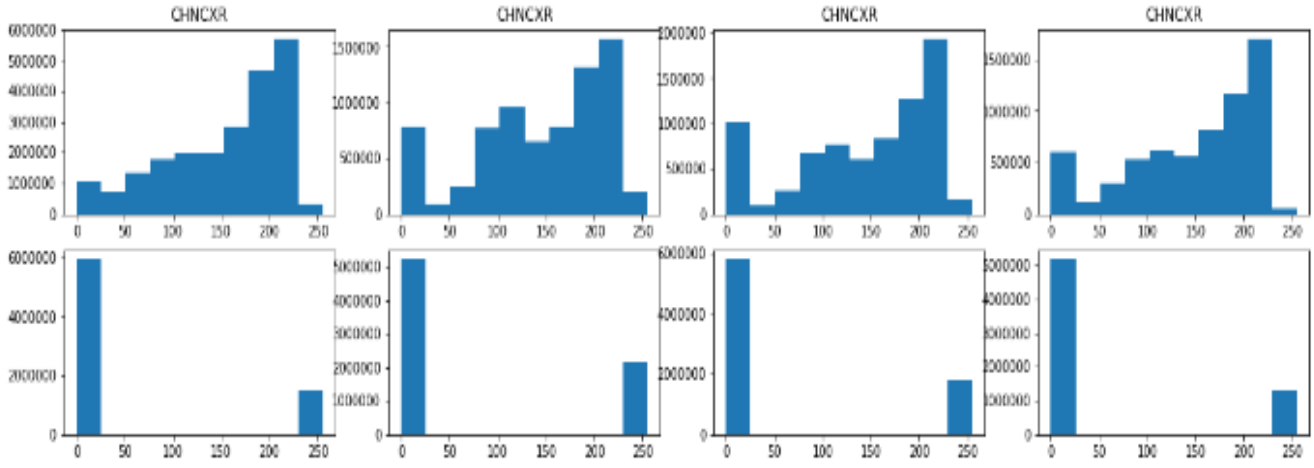


Figure 3.3 Mapping Graph of X-Rays

The figure 3.1 gives us the dataset that is the cxr xrays. It consists of 1200 CXR images from individuals with COVID-19, 1345 CXR images from individuals with viral Pneumonia, and 1341 CXR images from healthy individuals

The figure 3.2 gives us the the description of the dataset in a tabular form which we will be using during mapping

The figure 3.3 gives us the mapping of image in the form of a histogram

The graph will help us in mapping the points while defining and training the generators and Discriminator.

3.2 Algorithm

Generative Adversarial Networks(GANs)

A **generative adversarial network** (**GAN**) is a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in June 2014. Two neural networks contest with each other in the form of a zero-sum game, where one agent's gain is another agent's loss.

Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of generative model for unsupervised learning, GANs have also proved useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

The core idea of a GAN is based on the "indirect" training through the discriminator, another neural network that can tell how "realistic" the input seems, which itself is also being updated dynamically. This means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

GANs are similar to mimicry in evolutionary biology, with an evolutionary arms race between both networks.

The original GAN is defined as the following game:

- Each probability space(Ω,μ_{ref}) defines a GAN game.
- There are 2 players: generator and discriminator.
- The generator's strategy set is P(Ω), the set of all probability measures μ_G on(Ω) .
- The discriminator's strategy set is the set of Markov kernels μ_D: $\Omega \to$P[0,1], where P[0,1] is the set of probability measures on [0,1].

$$L(\mu_G, \mu_D) := \mathbb{E}_{x\sim\mu_{ref}, y\sim\mu_D(x)}[\ln y] + \mathbb{E}_{x\sim\mu_G, y\sim\mu_D(x)}[\ln(1-y)]$$

Equation 3.1 Objective Function Of GAN

The generator aims to minimize the objective, and the discriminator aims to maximize the objective.

Cyclic Generative Adversarial Networks(Cycle GANs)

CycleGAN is an architecture for performing translations between two domains, such as between photos of horses and photos of zebras, or photos of night cities and photos of day cities.

The CycleGAN game is defined as follows:

- There are two probability spaces $(\Omega_X, \mu_X), (\Omega_Y, \mu_Y)$, corresponding to the two domains needed for translations fore-and-back.
- There are 4 players in 2 teams: generators $G_X : \Omega_X \to \Omega_Y, G_Y : \Omega_Y \to \Omega_X$ and discriminators

$D_X : \Omega_X \to [0,1], D_Y : \Omega_Y \to [0,1]$.

$$L(G_X, G_Y, D_X, D_Y) = L_{GAN}(G_X, D_X) + L_{GAN}(G_Y, D_Y) + \lambda L_{cycle}(G_X, G_Y)$$

Equation 3.2 Cyclic GAN Objective Function

3.3 Libraries

- Pandas
- Numpy
- Seaborn
- Matplotlib
- Sklearn

3.4 Building the Cycle GAN model

1) Defining the Cycle GAN Model

```
class CycleGAN():
    def __init__(self,
                 img_rows,
                 img_cols,
                 channels_A,
                 channels_B,
                 parallel_channels=True,
                 ):
        """
        args:
          parallel_channels: process each input and output channel on its own
        """
        # Input shape
        self.img_rows = img_rows
        self.img_cols = img_cols
        self.channels_A = channels_A
        self.channels_B = channels_B
        self.parallel_channels = parallel_channels
        self.img_shape_A = (self.img_rows, self.img_cols, self.channels_A)
        self.img_shape_B = (self.img_rows, self.img_cols, self.channels_B)
        # Calculate output shape of D (PatchGAN)
        patch_r = int(self.img_rows / 2**4)
        patch_c = int(self.img_cols / 2**4)
        self.disc_patch = (patch_r, patch_c, 1)

        # Number of filters in the first layer of G and D
        self.gf = 32
        self.df = 64

        # Loss weights
        self.lambda_cycle = 10.0                    # Cycle-consistency loss
        self.lambda_id = 0.1 * self.lambda_cycle    # Identity loss

        optimizer = Adam(0.0002, 0.5)

        # Build and compile the discriminators
        self.d_A = self.build_discriminator(self.img_shape_A, suffix='A')
        self.d_B = self.build_discriminator(self.img_shape_B, suffix='B')
        self.d_A.compile(loss='mse',
                         optimizer=optimizer,
                         metrics=['accuracy'])
        self.d_B.compile(loss='mse',
                         optimizer=optimizer,
                         metrics=['accuracy'])

        # -------------------------
        # Construct Computational
        #   Graph of Generators
        # -------------------------

        # Build the generators
        self.g_AB = self.build_generator(
            self.img_shape_A, self.img_shape_B, suffix='AB')
        self.g_BA = self.build_generator(
            self.img_shape_B, self.img_shape_A, suffix='BA')

        # Input images from both domains
        img_A = Input(shape=self.img_shape_A, name='ImageA')
        img_B = Input(shape=self.img_shape_B, name='ImageB')

        # Translate images to the other domain
        fake_B = self.g_AB(img_A)
```

```
        fake_A = self.g_BA(img_B)
        # Translate images back to original domain
        reconstr_A = self.g_BA(fake_B)
        reconstr_B = self.g_AB(fake_A)
        # Identity mapping of images
        img_A_id = self.g_BA(img_B)
        img_B_id = self.g_AB(img_A)

        # For the combined model we will only train the generators
        self.d_A.trainable = False
        self.d_B.trainable = False

        # Discriminators determines validity of translated images
        valid_A = self.d_A(fake_A)
        valid_B = self.d_B(fake_B)

        # Combined model trains generators to fool discriminators
        self.combined = Model(inputs=[img_A, img_B],
                              outputs=[valid_A, valid_B,
                                       reconstr_A, reconstr_B,
                                       img_A_id, img_B_id])
        self.combined.compile(loss=['mse', 'mse',
                                    'mae', 'mae',
                                    'mae', 'mae'],
                              loss_weights=[1, 1,
                                            self.lambda_cycle, self.lambda_cycle,
                                            self.lambda_id, self.lambda_id],
                              optimizer=optimizer)
```

2) Defining Generator Function

```
def build_generator(self, in_img_shape, out_img_shape, suffix=''):
        """U-Net Generator"""
        in_chan = in_img_shape[-1]
        out_chan = out_img_shape[-1]
        gf_down = self.gf // in_chan if self.parallel_channels else self.gf
        gf_up = self.gf // in_chan if self.parallel_channels else self.gf

        def conv2d(layer_input, filters, f_size=4):
            """Layers used during downsampling"""
            d = Conv2D(filters, kernel_size=f_size,
                       strides=2, padding='same')(layer_input)
            d = LeakyReLU(alpha=0.2)(d)
            d = InstanceNormalization()(d)
            return d

        def deconv2d(layer_input, skip_input, filters, f_size=4, dropout_rate=0):
            """Layers used during upsampling"""
            u = UpSampling2D(size=2)(layer_input)
            u = Conv2D(filters, kernel_size=f_size, strides=1,
                       padding='same', activation='relu')(u)
            if dropout_rate:
                u = Dropout(dropout_rate)(u)
            u = InstanceNormalization()(u)
            u = Concatenate()([u, skip_input])
            return u

        # Image input
        d0 = Input(shape=in_img_shape)
        # Downsampling

        def _ds_pipe(in_d0):
            d1 = conv2d(in_d0, gf_down)
            d2 = conv2d(d1, gf_down*2)
            d3 = conv2d(d2, gf_down*4)
```

```
            d4 = conv2d(d3, gf_down*8)
            return d1, d2, d3, d4

        if (in_chan > 1) and self.parallel_channels:
            chan_list = []
            for i in range(in_chan):
                c_d0 = layers.Lambda(
                    lambda x: x[:, :, :, i:(i+1)],
                    name='InSel{}_{}'.format(suffix, i))(d0)
                chan_list.append(_ds_pipe(c_d0))
            d1, d2, d3, d4 = [
                layers.concatenate(list(c_outs))
                for c_outs in zip(*chan_list)]
        else:
            d1, d2, d3, d4 = _ds_pipe(d0)

        # Upsampling
        def _us_pipe(chan_count, d1, d2, d3, d4):
            u1 = deconv2d(d4, d3, gf_up*4)
            u2 = deconv2d(u1, d2, gf_up*2)
            u3 = deconv2d(u2, d1, gf_up)

            u4 = UpSampling2D(size=2)(u3)
            output_img = Conv2D(chan_count, kernel_size=4, strides=1,
                                padding='same', activation='sigmoid')(u4)
            return output_img


        if (out_chan > 1) and self.parallel_channels:
            chan_list = []
            for i in range(out_chan):
                chan_list.append(_us_pipe(1, d1, d2, d3, d4))
            output_img = layers.concatenate(chan_list)
        else:
            output_img = _us_pipe(out_chan, d1, d2, d3, d4)


        return Model(d0, output_img, name='Gen{}_{}_{}_{}-{}'.format(suffix, *in_img_shape,
out_img_shape[-1])
```

3) Defining Discriminator Function

```
def build_discriminator(self, img_shape, suffix=''):
        in_chan = img_shape[-1]
        df = self.df // in_chan if self.parallel_channels else self.df

        def d_layer(layer_input, filters, f_size=4, normalization=True):
            """Discriminator layer"""
            d = Conv2D(filters, kernel_size=f_size,
                       strides=2, padding='same')(layer_input)
            d = LeakyReLU(alpha=0.2)(d)
            if normalization:
                d = InstanceNormalization()(d)
            return d

        img = Input(shape=img_shape)

        def _disc_block(in_img):
            d1 = d_layer(in_img, df, normalization=False)
            d2 = d_layer(d1, df*2)
            d3 = d_layer(d2, df*4)
            d4 = d_layer(d3, df*8)
            return d4

        if (in_chan > 1) or not self.parallel_channels:
            chan_list = []
```

```
        for i in range(in_chan):
            c_img = layers.Lambda(
            lambda x: x[:, :, :, i:(i+1)],
            name='DiscSelect{}_{}'.format(suffix, i))(img)
          chan_list.append(_disc_block(c_img))
      d4 = layers.concatenate(chan_list)
  else:
      d4 = _disc_block(img)

  validity = Conv2D(1, kernel_size=4, strides=1, padding='same')(d4)

  return Model(img, validity, name='Disc{}_{}_{}_{}'.format(suffix, *img_shape))
```

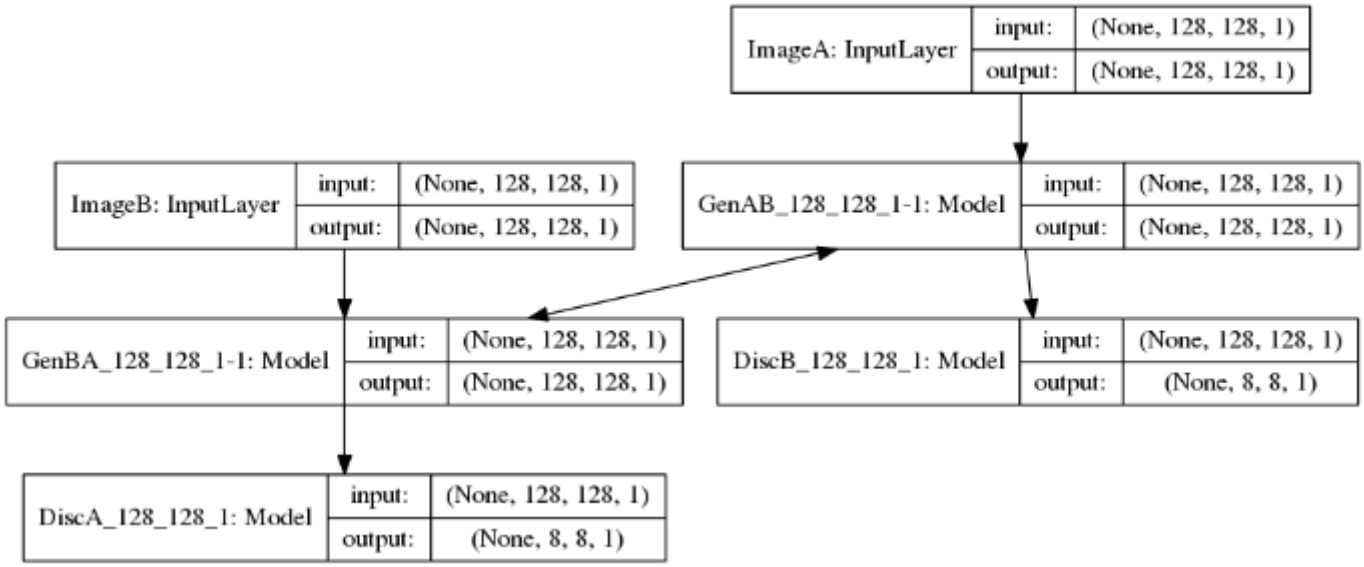
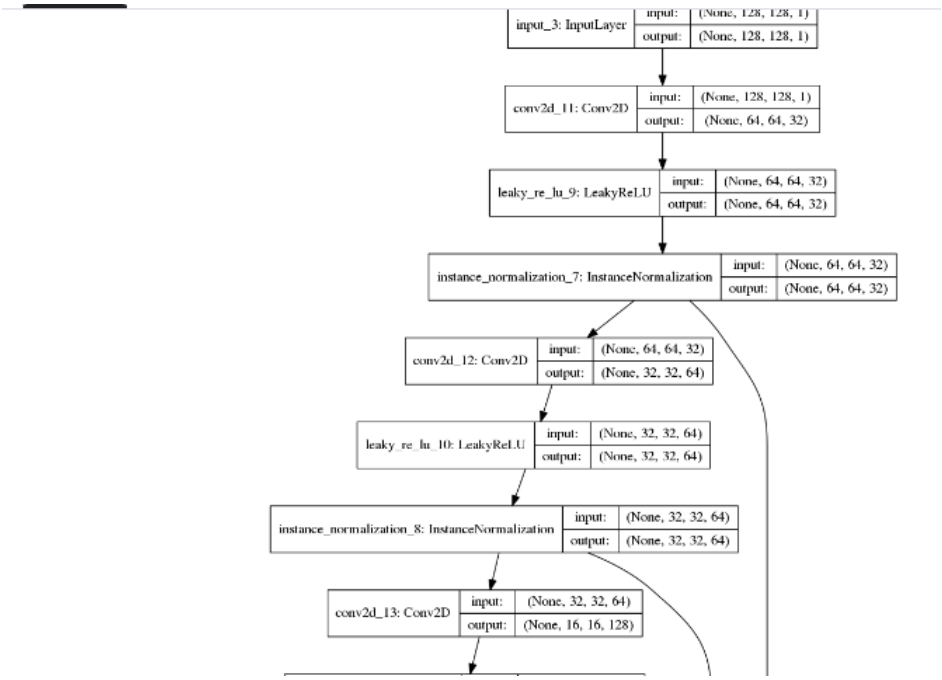


Figure 3.4 Cycle GAN Model Architecture



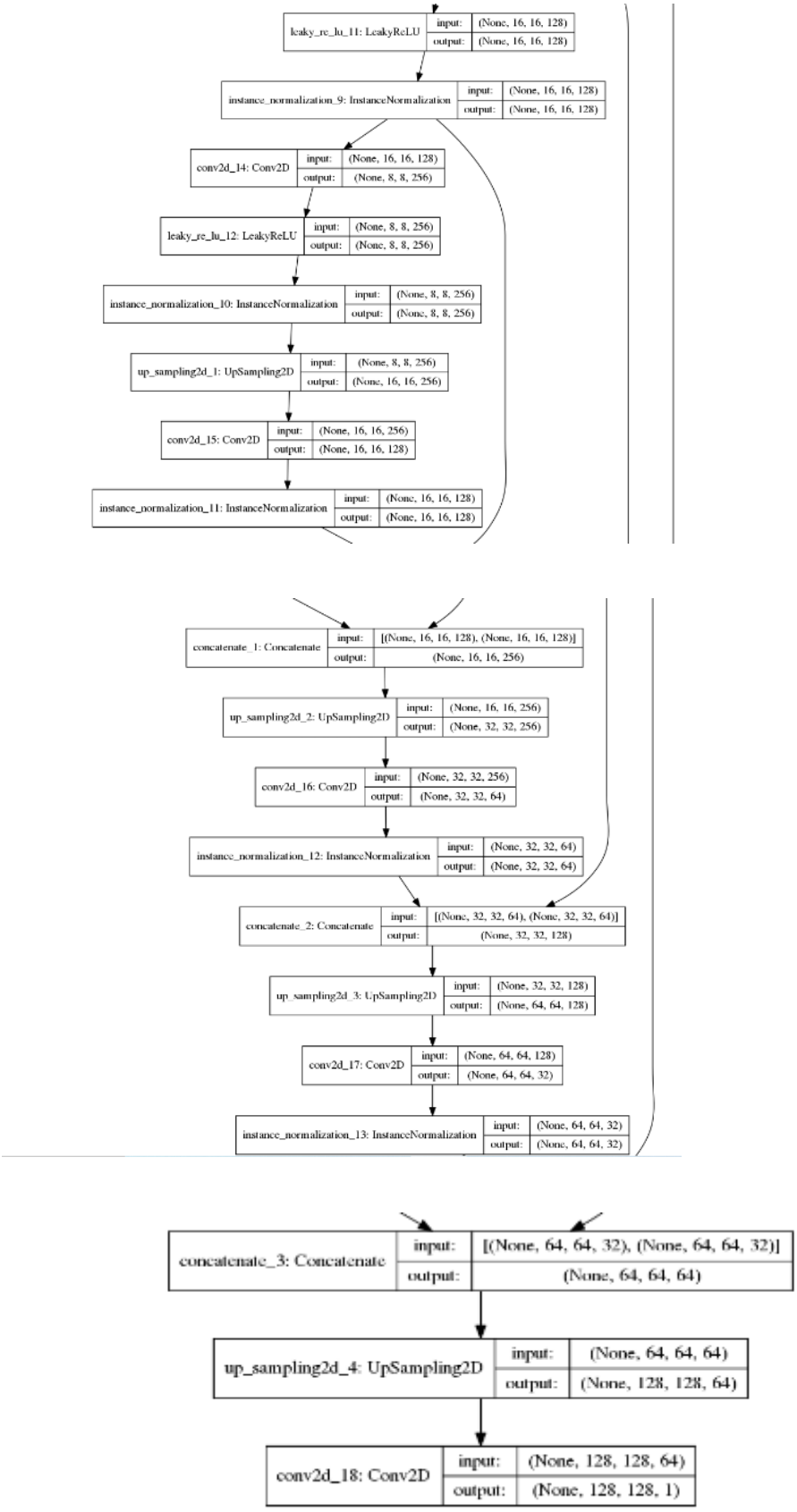


Fig 3.5 Generator Architecture

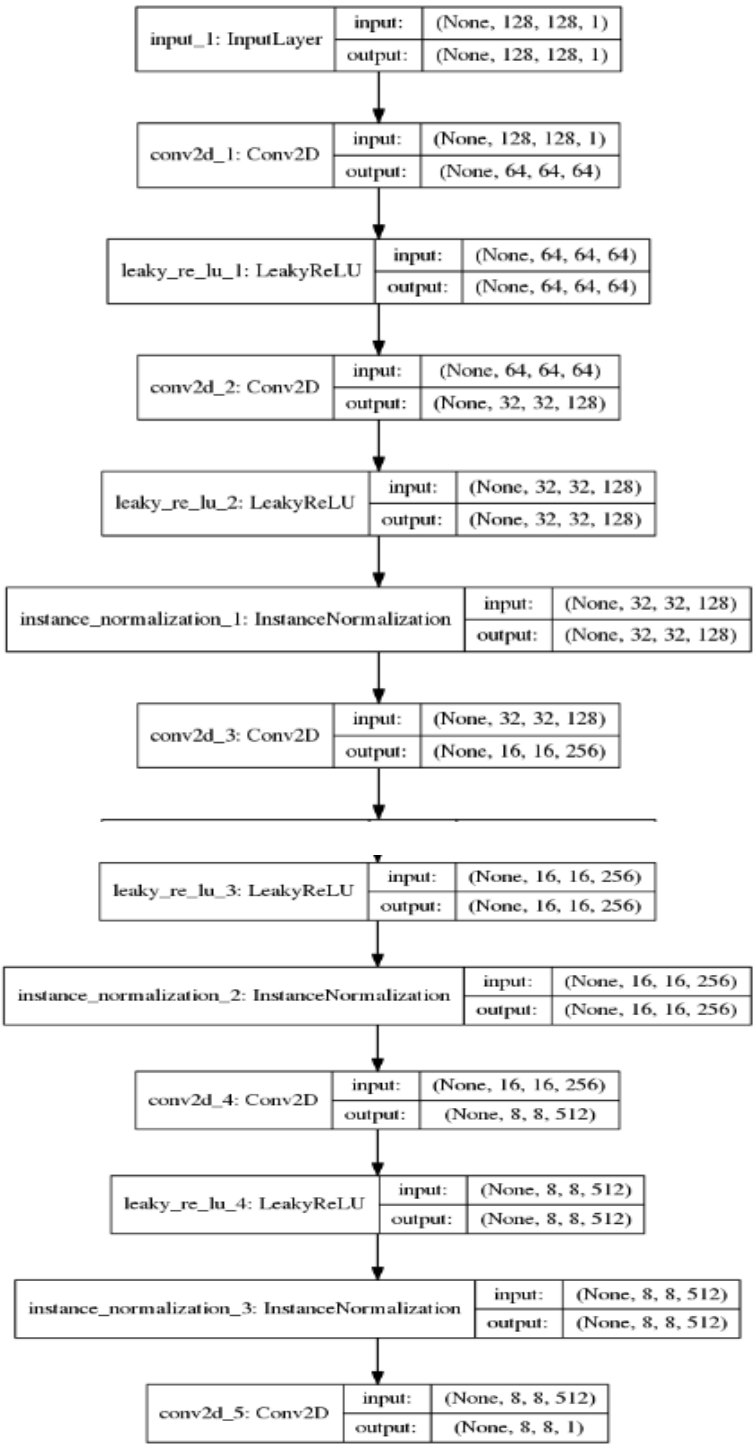


Fig 3.6 Discrminator Architecture

Chapter 4

OBSERVATION AND RESULTS

4.1 Training

- **Training CycleGAN model**

The model is trained for 30 epochs and for a batch size of 64

```
            BATCH_SIZE = 64
            EPOCHS = 30
            start_time = datetime.datetime.now()

# Adversarial loss ground truths
valid = np.ones((BATCH_SIZE,) + cg.disc_patch)
fake = np.zeros((BATCH_SIZE,) + cg.disc_patch)

for epoch in tqdm_notebook(range(EPOCHS), desc='Epochs'):
    for batch_i, (imgs_A, imgs_B) in tqdm_notebook(enumerate(loader_obj.load_batch(BATCH_SIZE)),
desc='Batch'):
        #  Train Discriminators
        if imgs_A is None:
            break
        # Translate images to opposite domain
        fake_B = cg.g_AB.predict(imgs_A)
        fake_A = cg.g_BA.predict(imgs_B)

        # Train the discriminators (original images = real / translated = Fake)
        dA_loss_real = cg.d_A.train_on_batch(imgs_A, valid)
        dA_loss_fake = cg.d_A.train_on_batch(fake_A, fake)
        dA_loss = 0.5 * np.add(dA_loss_real, dA_loss_fake)

        dB_loss_real = cg.d_B.train_on_batch(imgs_B, valid)
        dB_loss_fake = cg.d_B.train_on_batch(fake_B, fake)
        dB_loss = 0.5 * np.add(dB_loss_real, dB_loss_fake)

        # Total disciminator loss
        d_loss = 0.5 * np.add(dA_loss, dB_loss)

        #  Train Generators
        # Train the generators
        g_loss = cg.combined.train_on_batch([imgs_A, imgs_B],
                                            [valid, valid,
                                            imgs_A, imgs_B,
                                            imgs_A, imgs_B])

        elapsed_time = datetime.datetime.now() - start_time

    # Plot the progress at each epoch
    print ("[Epoch %d/%d] [Batch %d/%d] [D loss: %f, acc: %3d%%] [G loss: %05f, adv: %05f, recon: %05f,
id: %05f] time: %s " \
                                                           % ( epoch, EPOCHS,
                                                               batch_i, loader_obj.n_batches,
                                                               d_loss[0], 100*d_loss[1],
                                                               g_loss[0],
                                                               np.mean(g_loss[1:3]),
                                                               np.mean(g_loss[3:5]),
                                                               np.mean(g_loss[5:6]),
                                                               elapsed_time))

    clear_output()
    sample_images(cg, loader_obj, epoch, batch_i
```

4.2 Results & Snapshots

The model outputs loss and accuracy for each epoch and the images at the end of each epoch as shown below our model took around 22 hrs to train as Generative Adversarial Networks are computed slowly as each and every pixel is mapped and reconstructed again.

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:7: TqdmDeprecationWarning: This function will be removed in tqdm==5.
0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  import sys


Epochs:   0%|          | 0/30 [00:00<?, ?it/s]

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8: TqdmDeprecationWarning: This function will be removed in tqdm==5.
0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`


Batch: 0it [00:00, ?it/s]

Found 1600 validated image filenames.
Found 1408 validated image filenames.
```

Fig 4.1 Model Under Training

epochs shows the progress of the current epoch and also processing time per epoch batch shows the current status of the batch under consideration and processing time per batch

```
In [24]:
sample_images(cg, loader_obj, EPOCHS, 2)
```

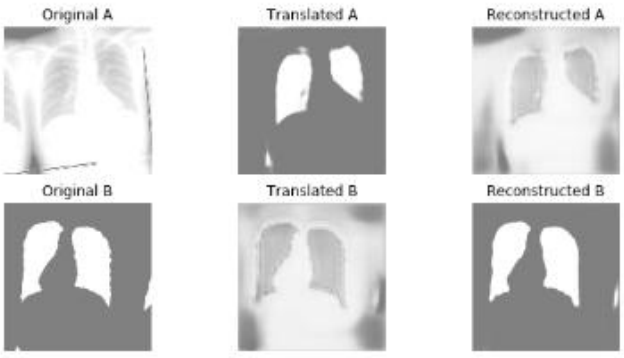


Fig 4.2 epoch 2 of model training

From fig 4.2 it is clearly seen that reconstructed image has significant differences compared to the original image which implies that model is not yet trained fully and

need more epochs to get properly and accurately trained

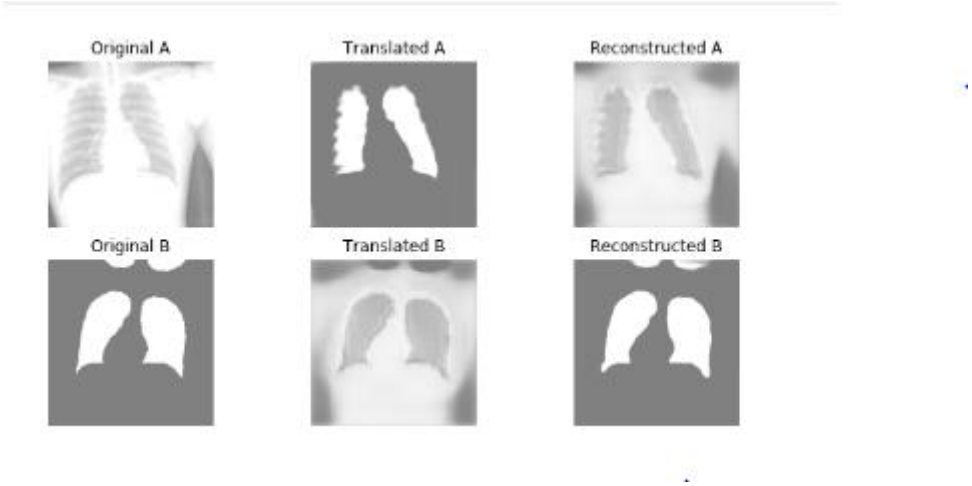


Fig 4.3 Final Output of Training

The final Output after 30 epochs is shown by fig 4.3 the model has successfully and accurately reconstructed the original image without any overfitting or underfitting. It is observed that reconstructed image is near perfect to the original image which was given as input.

Chapter 5

CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

This study is aimed at evaluating the effectiveness of the state-of-the-art pretrained Generative Adversarial Networks (GANs) on the automatic diagnosis of Pneumonia from chest X-rays (CXRs). it could be applied to lots of issues where paired training data are not available. It also produces more 'realistic' segmentations since the discriminator is actively trying to determine if the image output is discernible real data. The model was successfully and accurately trained for 30 epochs and accurate predictions are contained in the output file.

5.2 Future Enhancement

The Generative Adversarial Networks are currently still under research so new varities of GAN maybe founded which can be used and which might be more accurate and the main drawback being the training time as Generative Adversarial Networks analyzes each and every pixel of the input image it takes a lot of time for computation and mapping with gpu it took 22 hrs for the dataset we have used without gpu it can take 48+ hrs to process depending on the size of the dataset.There is no solution currently for this even though tpu can be used but it is ideal for videos and highly pixilated images rather than x-rays.

Chapter 6

REFERENCE

1. **Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks by** Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros 2017.

2. Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash and A. Torralba, *Cross-modal scene networks*, 2016,

3. K. Bousmalis, N. Silberman, D. Dohan, D. Erhan and D. Krishnan, *Unsupervised pixel-level domain adaptation with generative adversarial networks*, 2016,

4. E.L. Denton, S. Chintala, R. Fergus et al., "Deep generative image models using a laplacian pyramid of adversarial networks", *NIPS*, pp. 1486-1494, 2015.

5. Quora questions and answers

6. Stack overflow for resolving errors