X-RAY Image Generation using GAN

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Lokesh Sharma[RA2111030010151]
Jeevesh Patel [RA2111030010152]
Kanishk Mandwal [RA2111030010156]

Under the guidance of
Mrs. V. Vijayalakshi
Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603203

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **“X-Ray Image Generation Using GAN”** is the bonafide work of **Lokesh Sharma [RA2111030010151], Jeevesh Patel [RA2111030010152],Kanishk Mandwal[RA2111030010156]** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Mrs. V. Vijayalakshmi
Assistant Professor
Department of Networking
and Communications

SIGNATURE
Dr. Annapurani K
Professor and Head
Department of Networking
and Communications

ACKNOWLEDGEMENT

I would like to express my sincere appreciation to all those who have contributed to the completion of this project.

Firstly,I extend my gratitude ***to Mrs. V. Vijayalakshi*** for her guidance and support throughout the duration of this ***X-RAY Image Generation using GAN*** project. Their expertise and feedback were instrumental in shaping the project's direction and ensuring its successful completion.

I also want to thank SRM University for providing the necessary resources and facilities for conducting this project.

Additionally, I am grateful to the open-source community for the tools, libraries, and frameworks that were utilized in the development of this project.

Finally,I would like to acknowledge the countless hours of effort put into this project,which has been a rewarding learning experience.

This project is the culmination of my efforts , but it would not have been possible without the support and guidance of these individuals and institutions.

ABSTRACT

The intersection of medical imaging and artificial intelligence has ushered in a new era of innovation in healthcare. This paper explores the cutting-edge field of X-ray image generation using Generative Adversarial Networks (GANs). GANs, a subset of machine learning models, have demonstrated their ability to generate highly realistic synthetic data by leveraging a competitive two-network framework. In the context of X-ray imaging, this technology has profound implications for both research and clinical practice. This article delves into the methodologies behind GAN-powered X-ray image generation, shedding light on the complex training processes and data augmentation techniques. It emphasizes the benefits of synthetic X-ray images, including their utility in augmenting limited datasets and their potential for reducing patient radiation exposure during imaging research. Moreover, this exploration uncovers practical applications of GAN-generated X-ray images, ranging from enhancing the performance of computer-aided diagnosis systems to improving the training of radiologists and deep learning models. By synthesizing realistic X-ray images, GANs offer a transformative tool for refining the accuracy and reliability of diagnostic processes. As we navigate the rapidly evolving landscape of AI-driven healthcare, this abstract provides a glimpse into the role of GANs in reshaping the field of X-ray imaging and advancing the frontiers of medical diagnosis and treatment.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

GAN - Generative Adversarial Network

ACGAN - Auxiliary Classifier Generative Adversarial Network

FID - Fréchet Inception Distance

SSI – Structural Similarity Index

PSNR – Peak Signal to Noise Ratio

GP – Gradient Penalty

DCGAN – Deep Convolutional Generative Adversarial Network

COVID-19 - Coronavirus disease 2019

CHAPTER 1

INTRODUCTION

Medical imaging has long been an indispensable tool in healthcare, aiding in the diagnosis and treatment of various diseases and conditions. Among the plethora of imaging techniques available, X-ray imaging stands out as one of the most widely used and accessible modalities for capturing detailed internal structures of the human body. Over the years, technological advancements have continually improved the quality and efficiency of X-ray imaging, leading to more accurate diagnoses and better patient outcomes.

In recent years, the field of artificial intelligence, particularly the application of Generative Adversarial Networks (GANs), has made significant strides in revolutionizing various aspects of medical imaging. GANs are a class of machine learning models that have shown remarkable capabilities in generating high-quality, realistic data, including images, by pitting two neural networks, a generator and a discriminator, against each other in a training process. This innovation has extended to X-ray image generation, presenting exciting possibilities for healthcare professionals and researchers alike.

Generative adversarial network (GAN) is a machine learning model, consisting of two parts: a generator and a discriminator. The generator generates data, while the discriminator tries to distinguish generated and real data, thus the discriminator acts as a complicated loss function for the generator. Our goal is the generator that creates synthetic data undistinguished from real. All this enables the model to learn in an unsupervised manner.

CHAPTER 2

LITERATURE SURVEY

2.1 Generative Adversarial Networks for the Synthesis of Chest X-ray Images: In the context of diagnosing COVID-19, chest X-ray images play a crucial role. However, obtaining a large, annotated dataset of COVID-19 chest X-rays is challenging due to privacy concerns. To address this, the authors explored using GANs as a data augmentation technique. They investigated two GAN architectures: Deep Convolutional GANs (DCGAN) and Wasserstein GANs with Gradient Penalty (WGAN-GP). Remarkably, they successfully generated synthetic COVID-19 chest X-ray images with a Fréchet Inception Distance (FID) score below 2.

2.2 Application of Generative Adversarial Networks (GANs) in Ophthalmology: A survey of studies published before June 2021 explored various applications of GANs in ophthalmology image domains. The analysis covered the type of GAN used, imaging tasks, and outcomes. This review highlights the usefulness of GANs in ophthalmology.

2.3 Creating Artificial Images for Radiology Applications Using GANs: This study reviews the literature on GAN applications in radiology. GANs, known for generating realistic images, have made a significant impact in computer vision. The paper explores their potential applications in radiology.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

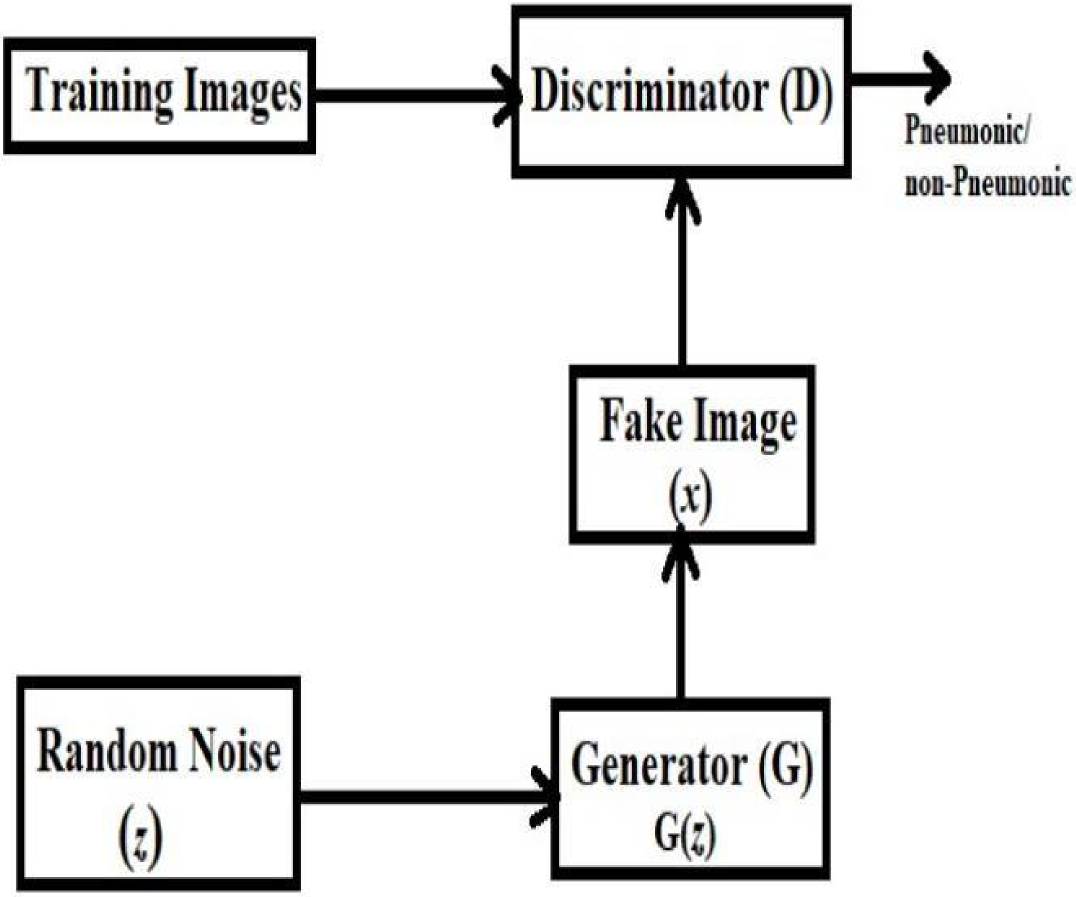


Fig 3.1

CHAPTER 4

METHODOLOGY

4.1 Data Collection and Preprocessing: Gather a comprehensive dataset of X-ray images, ensuring diversity in terms of medical conditions, patient demographics, and imaging equipment. Preprocess the dataset to standardize image sizes, adjust contrast, and remove artifacts, ensuring data consistency.

4.2 GAN Architecture Selection: Choose an appropriate GAN architecture that is well-suited for X-ray image generation.

4.3 Training the GAN: Train the GAN on the preprocessed dataset, with the generator network generating synthetic X-ray images and the discriminator network distinguishing between real and synthetic images. Implement best practices for GAN training, including techniques like mini-batch discrimination, gradient penalty, and spectral normalization to stabilize training and improve image quality. Continuously monitor the training process to prevent issues like mode collapse or vanishing gradients.

4.4 Evaluation and Validation: Assess the quality of the generated X-ray images using quantitative metrics such as Structural Similarity Index (SSI), Peak Signal-to-Noise Ratio (PSNR), and Inception Score. Conduct a human evaluation by involving radiologists to validate the clinical realism and utility of the synthetic images.

4.5 Dataset Augmentation: Augment the existing X-ray dataset with the synthetic images to create a larger, more diverse dataset. Ensure proper stratification and maintain the balance of classes to prevent bias in machine learning models.

4.6 Radiation Reduction: Identify specific diagnostic tasks or scenarios where synthetic X-ray images can be used as substitutes for real X-rays to reduce patient radiation exposure. Quantify the reduction in radiation exposure achieved through the use of synthetic images.

4.7 Diagnostic Enhancement: Investigate ways to use GAN-generated X-ray images to enhance the interpretability of radiological findings. Explore techniques for emphasizing specific pathologies, improving image clarity, and simulating rare or challenging cases.

4.8 AI Model Training and Validation: Utilize the augmented dataset (comprising real and synthetic images) to train and fine-tune deep learning models for various diagnostic tasks. Evaluate the performance of these models using cross-validation and benchmark them against existing models.

4.9 Iterative Refinement: Iterate on the GAN training process and model improvements based on feedback from radiologists and model evaluation results. Continuously update and expand the synthetic X-ray dataset to reflect emerging medical conditions and imaging variations.

CHAPTER 5

CODING AND TESTING

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'chest-xray-
pneumonia:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'),
target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'),
target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
  try:
    with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
      total_length = fileres.headers['content-length']
      print(f'Downloading {directory}, {total_length} bytes compressed')
      dl = 0
      data = fileres.read(CHUNK_SIZE)
      while len(data) > 0:
        dl += len(data)
        tfile.write(data)
        done = int(50 * dl / int(total_length))
        sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes
downloaded")
        sys.stdout.flush()
        data = fileres.read(CHUNK_SIZE)
      if filename.endswith('.zip'):
       with ZipFile(tfile) as zfile:
        zfile.extractall(destination_path)
      else:
       with tarfile.open(tfile.name) as tarfile:
        tarfile.extractall(destination_path)
      print(f'\nDownloaded and uncompressed: {directory}')
  except HTTPError as e:
    print(f'Failed to load (likely expired) {download_url} to path
{destination_path}')
    continue
  except OSError as e:
    print(f'Failed to load {download_url} to path {destination_path}')
    continue

print('Data source import complete.')


import tensorflow as tf
from keras.datasets import mnist
import cv2
import os
import pathlib
from keras.layers import Conv2D, Conv2DTranspose, Dropout, Dense,
Reshape, LayerNormalization, LeakyReLU
from keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import f1_score, recall_score, precision_score
```

```
class ReadDataset:
    def__init__(self, datasetpath, labels, image_shape):
        self.datasetpath = datasetpath
        self.labels = labels
        self.image_shape = image_shape
    def returListImages(self,):
        self.images = []
        for label in self.labels:
            self.images.append(list(pathlib.Path(os.path.join(self.datasetpath,
                                                              label)).glob('*.*')))
    def readImages(self,):
        self.returListImages()
        self.finalImages = []
        labels = []
        for label in range(len(self.labels)):
            for img in self.images[label]:
                img = cv2.imread(str(img))
                img = cv2.resize(img , self.image_shape)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img  = img/255
                self.finalImages.append(img)
                labels.append(label)
        images = np.array(self.finalImages)
        labels = np.array(labels)
        return images, labels



readDatasetObject = ReadDataset('/kaggle/input/chest-xray-
pneumonia/chest_xray/train', ['NORMAL', 'PNEUMONIA'], (64, 64))
images, labels = readDatasetObject.readImages()

images.shape, labels.shape

plt.figure(figsize = (12, 12))
indexs = np.random.randint(0, len(labels), size = (64, ))
for i in range(64):
    plt.subplot(8, 8, (i + 1))
    plt.imshow(images[indexs[i]])
    plt.title(labels[indexs[i]])
plt.legend()
```

```
class ReadDataset:
    def__init__(self, datasetpath, labels, image_shape):
        self.datasetpath = datasetpath
        self.labels = labels
        self.image_shape = image_shape



    def returListImages(self,):
        self.images = []
        for label in self.labels:
            self.images.append(list(pathlib.Path(os.path.join(self.datasetpath,
                                             label)).glob('*.*')))



    def readImages(self,):
        self.returListImages()
        self.finalImages = []
        labels = []
        for label in range(len(self.labels)):
            for img in self.images[label]:
                img = cv2.imread(str(img))
                img = cv2.resize(img , self.image_shape)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                img  = img/255
                self.finalImages.append(img)
                labels.append(label)
        if len(labels) == 0:
            raise ValueError("No images found in the dataset.")
        images = np.array(self.finalImages)
        labels = np.array(labels)
        return images, labels
```

```
class Acgan:

    def__init__(self, eta, batch_size, epochs, weight_decay, latent_space,
                 image_shape, kernel_size):
        self.eta =  eta
        self.batch_size = batch_size
        self.epochs = epochs
        self.weight_decay = weight_decay
        self.latent_space = latent_space
        self.image_shape = image_shape
        self.kernel_size = kernel_size

    def data(self, images, labels):
        ytrain = tf.keras.utils.to_categorical(labels)
        self.images = images
        self.labels = ytrain

    def samples(self, G, noize, labels):
        images = G.predict([noize, labels])
        ys = np.argmax(labels, axis = 1)
        plt.figure(figsize = (12, 4))
        for i in range(16):
            plt.subplot(2, 8, (i + 1))
            plt.imshow(images[i], cmap = 'gray')
            plt.title(ys[i])
        plt.show()

    def generator(self, inputs, labels):
        filters = [256, 128, 64, 32]
        padding = 'same'
        x = inputs
        y = labels
        x = layers.concatenate([x, y])
        x = layers.Dense(1024, )(x)
        x = layers.Dense(8*8*filters[0],
                         kernel_regularizer = tf.keras.regularizers.L2(0.001))(x)
        x = layers.Reshape((8, 8, filters[0]))(x)
        for filter in filters:
            if filter >= 64:
                strides = 2
            else:
                strides = 1
            x =  LayerNormalization()(x)
            x = layers.Activation('relu')(x)
            x = Conv2DTranspose(filter, kernel_size = self.kernel_size, padding =
padding,
                      strides = strides)(x)
        x = Conv2DTranspose(3, kernel_size = self.kernel_size, padding =
padding)(x)
        x = layers.Activation('sigmoid')(x)
        self.generatorModel = models.Model(inputs = [inputs, labels],
                                           outputs = x,
                                           name = 'generator')

    def discriminator(self, inputs):
        x = inputs
        filters = [32, 64, 128, 256]
        padding = 'same'
        for filter in filters:
            if filter < 256:
                strides = 2
            else:
                strides = 1
            x = Conv2D(filter, kernel_size = self.kernel_size, padding = padding,
                      strides = strides,
                      kernel_regularizer = tf.keras.regularizers.L2(0.001))(x)
            x = LeakyReLU(alpha = 0.2)(x)
        x = layers.Flatten()(x)
        outputs = Dense(1, )(x)
        labelsOutput = Dense(256,
                             kernel_regularizer = tf.keras.regularizers.L2(0.001))(x)
        labelsOutput = Dropout(0.3)(labelsOutput)
        labelsOutput = Dense(2,)(labelsOutput)
        labelsOutput = layers.Activation('softmax')(labelsOutput)
        self.discriminatorModel = models.Model(inputs = inputs,
                                               outputs = [outputs, labelsOutput],
                                               name = 'discriminator')


    def build(self,):
        generatorInput = layers.Input(shape = (self.latent_space))
        discriminatorInput = layers.Input(shape = (self.image_shape))
        labelsInput = layers.Input(shape = (2, ))
        self.generator(generatorInput, labelsInput)
        self.discriminator(discriminatorInput)
        G = self.generatorModel
        D = self.discriminatorModel
        D.compile(loss = ['mse', 'binary_crossentropy'],
                 optimizer = tf.keras.optimizers.RMSprop(learning_rate = self.eta,
                                                         weight_decay = self.weight_decay))
        D.summary()
        G.summary()
        D.trainable = False
        GAN = models.Model(inputs = [generatorInput, labelsInput],
                           outputs = D(G([generatorInput, labelsInput])))
        GAN.compile(loss = ['mse', 'binary_crossentropy'],
                   optimizer = tf.keras.optimizers.RMSprop(learning_rate = 
self.eta*0.5,
                                                           weight_decay = self.weight_decay*0.5))
        GAN.summary()
        return G, D, GAN

    def trainAlgorithm(self, G, D, GAN):
        for epoch in range(self.epochs):
            indexs = np.random.randint(0, len(self.images), size = (self.batch_size, 
))
            realImages = self.images[indexs]
            realLabels = self.labels[indexs]
            realTag = tf.ones(shape = (self.batch_size, ))
            noize = tf.random.uniform(shape = (self.batch_size, 
                                              self.latent_space), minval = -1, 
                                     maxval = 1)
            fakeLabels = tf.keras.utils.to_categorical(np.random.choice(range(2), 
size = (self.batch_size)),
                                                       num_classes = 2)
            fakeImages = tf.squeeze(G.predict([noize, fakeLabels], verbose = 0))
            fakeTag = tf.zeros(shape = (self.batch_size, ))
            allImages = np.vstack([realImages, fakeImages])
            allLabels = np.vstack([realLabels, fakeLabels])
            allTags = np.hstack([realTag, fakeTag])
            _, dlossTag, dlossLabels = D.train_on_batch(allImages, [allTags, 
allLabels])
            noize = tf.random.uniform(shape = (self.batch_size, 
                                              self.latent_space), minval = -1, 
                                     maxval = 1)
            _, glossTag, glossLabels = GAN.train_on_batch([noize, fakeLabels], 
[realTag, fakeLabels])
            if epoch % 5000 == 0:
                print('Epoch: {}'.format(epoch))
                    print('discriminator loss: [tag: {}, labels: {}], generator loss: [tag: 
                    {}, labels: {}]'.format(dlossTag, dlossLabels, glossTag, 
                    glossLabels))
                self.samples(G, noize, fakeLabels)


acgan = Acgan(eta = 0.0001, batch_size = 5, epochs = 5, weight_decay = 6e-9,
           latent_space = 100, image_shape = (64, 64, 3), kernel_size = 1)

acgan.data(images, labels)

G, D, GAN = acgan.build()

tf.keras.utils.plot_model(GAN, show_shapes = True)

tf.keras.utils.plot_model(GAN, show_shapes = True)

tf.keras.utils.plot_model(D, show_shapes = True)

acgan.trainAlgorithm(G, D, GAN)

G.save('/kaggle/working/generator.h5')

G = tf.keras.models.load_model('/kaggle/working/generator.h5')

datasetGenerationSize = 30000
noize = tf.random.uniform(shape = (datasetGenerationSize, 100), minval = -1,
maxval = 1)
newlabels = tf.keras.utils.to_categorical(np.random.choice([0, 1], size =
(datasetGenerationSize, )), num_classes = 2)

noize.shape, newlabels.shape

np.unique(np.argmax(newlabels, axis = 1), return_counts = True)

imagesGeneration = G.predict([noize, newlabels])
imagesGeneration.shape

plt.figure(figsize = (12, 12))
t = np.argmax(newlabels, axis = 1)
for i in range(64):
   plt.subplot(8, 8, (i + 1))
   plt.imshow(imagesGeneration[i])
   plt.title(t[i])
plt.legend()
```

```
basemodel = tf.keras.applications.VGG16(weights = None, input_shape = (64,
64, 3), pooling = 'max', include_top = False)
x = layers.Dropout(0.4)(basemodel.output)
x = layers.Dense(128,)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha = 0.2)(x)
x = layers.Dropout(0.4)(x)
x = layers.Dense(32,)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha = 0.2)(x)
x = layers.Dropout(0.4)(x)
x = layers.Dense(1, activation = 'sigmoid')(x)
m = tf.keras.models.Model(inputs = basemodel.input, outputs = x)
m.compile(loss = 'binary_crossentropy', optimizer =
tf.keras.optimizers.Adam(learning_rate = 0.00001))
m.summary()

history = m.fit(imagesGeneration, np.argmax(newlabels, axis = 1), epochs = 60,
batch_size = 64, validation_split = 0.2,
              callbacks = [tf.keras.callbacks.EarlyStopping(patience = 2, monitor =
'val_loss', mode = 'min', restore_best_weights = True)])


plt.figure(figsize = (7, 6))
plt.plot(history.history['loss'], label = 'training loss')
plt.plot(history.history['val_loss'], label = 'validation loss')
plt.title('Results obtained while training a neural network on images generated
by the neural network')
plt.legend()


m.evaluate(images, labels)

y_pred = tf.squeeze(m.predict(images))
y_pred.shape

y_pred = y_pred >= 0.5
y_pred = np.array(y_pred, dtype = 'int32')
y_pred

accuracy_score(y_pred, labels)*100

print(classification_report(y_pred, labels))
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_pred, labels)
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_pred, labels)
```

```
import pandas as pd
cmObject = pd.DataFrame(cm , index = ['NORMAL', 'PNEUMONIA'],
columns = ['NORMAL', 'PNEUMONIA'])
cmObject.head()

print('f1_score: {}, recall_score: {}, precision_score:
{}'.format(f1_score(y_pred, labels)*100, recall_score(y_pred, labels)*100,
precision_score(y_pred, labels)*100))

sns.heatmap(cmObject, annot = True, cmap="Blues")
```

CHAPTER 6

SCREENSHOTS AND RESULTS

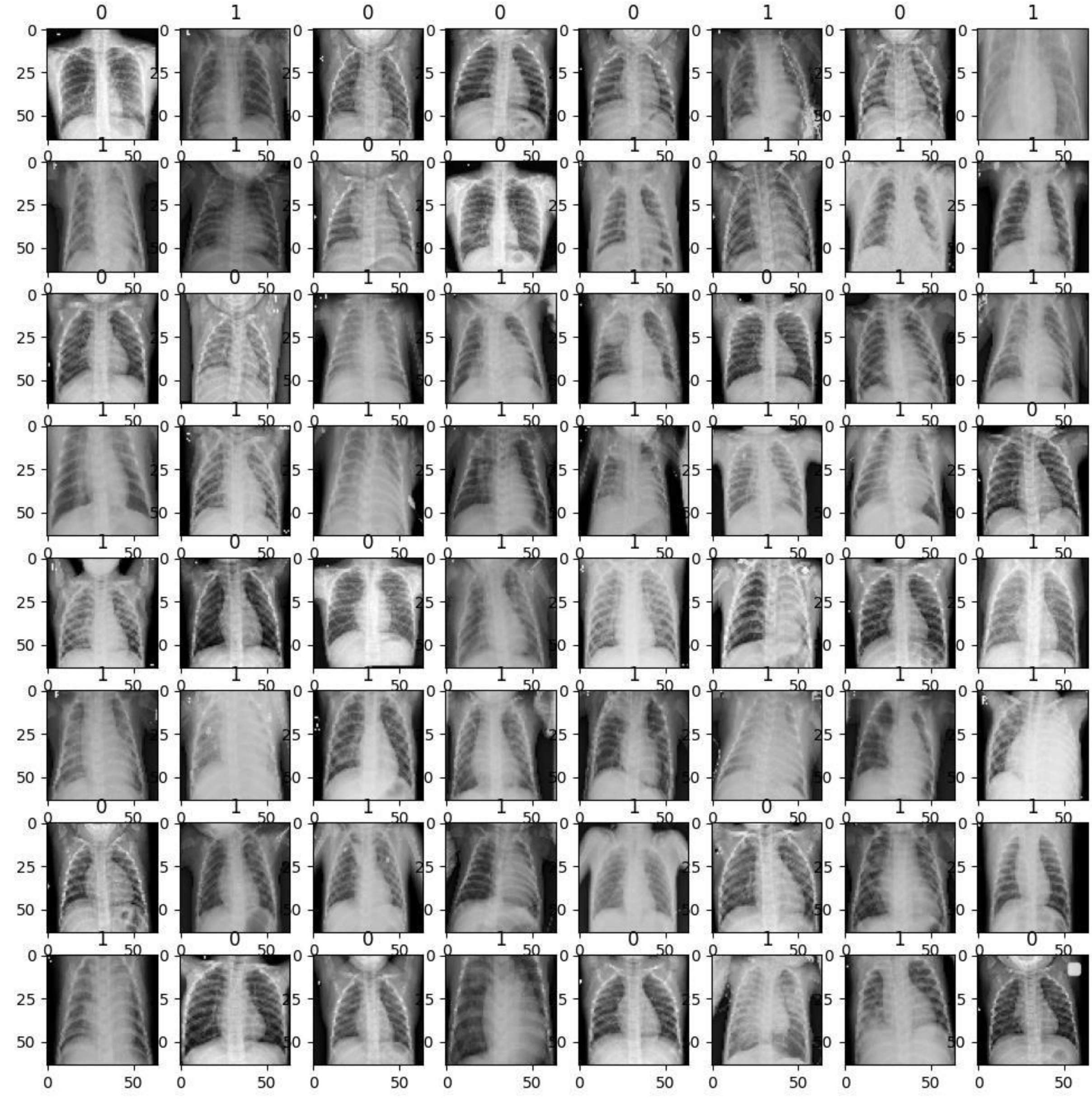


Fig 6.1

```
Model: "discriminator"

_____________________________________________________________________________

 Layer (type)                Output Shape         Param #     Connected to

=============================================================================

input_2 (InputLayer)         [(None, 64, 64, 3)]  0           []

conv2d (Conv2D)              (None, 32, 32, 32)   2432        ['input_2[0][0]']

leaky_re_lu (LeakyReLU)      (None, 32, 32, 32)   0           ['conv2d[0][0]']

conv2d_1 (Conv2D)            (None, 16, 16, 64)   51264       ['leaky_re_lu[0][0]']

leaky_re_lu_1 (LeakyReLU)    (None, 16, 16, 64)   0           ['conv2d_1[0][0]']

conv2d_2 (Conv2D)            (None, 8, 8, 128)    204928      ['leaky_re_lu_1[0][0]']

leaky_re_lu_2 (LeakyReLU)    (None, 8, 8, 128)    0           ['conv2d_2[0][0]']

conv2d_3 (Conv2D)            (None, 8, 8, 256)    819456      ['leaky_re_lu_2[0][0]']

leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 256)    0           ['conv2d_3[0][0]']

flatten (Flatten)            (None, 16384)        0           ['leaky_re_lu_3[0][0]']

dense_3 (Dense)              (None, 256)          4194560     ['flatten[0][0]']

dropout (Dropout)            (None, 256)          0           ['dense_3[0][0]']

dense_4 (Dense)              (None, 2)            514         ['dropout[0][0]']

dense_2 (Dense)              (None, 1)            16385       ['flatten[0][0]']

activation_5 (Activation)    (None, 2)            0           ['dense_4[0][0]']
=============================================================================
Total params: 5,289,539

Trainable params: 5,289,539

Non-trainable params: 0

_____________________________________________________________________________
```

```
Model: "generator"
________________________________________________________________________________

 Layer (type)                Output Shape        Param #     Connected to

==================================================================================
input_1 (InputLayer)         [(None, 100)]       0          []

input_3 (InputLayer)         [(None, 2)]         0          []

concatenate (Concatenate)     (None, 102)     0      ['input_1[0][0]',
'input_3[0][0]']

dense (Dense)                (None, 1024)        105472      ['concatenate[0][0]']

dense_1 (Dense)               (None, 16384)       16793600    ['dense[0][0]']

reshape (Reshape)             (None, 8, 8, 256)    0          ['dense_1[0][0]']

layer_normalization (LayerNormalization  (None, 8, 8, 256)    512
['reshape[0][0]']
                                                                   activation
(Activation)        (None, 8, 8, 256)    0          ['layer_normalization[0][0]']

 conv2d_transpose (Conv2DTranspose (None, 16, 16, 256) 1638656
['activation[0][0]']

layer_normalization_1 (LayerNormalization (None, 16, 16, 256) 512
['conv2d_transpose[0][0]']

 activation_1 (Activation)      (None, 16, 16, 256) 0
['layer_normalization_1[0][0]']

 conv2d_transpose_1 (Conv2DTranspose (None, 32, 32, 128) 819328
['activation_1[0][0]']

 layer_normalization_2 (LayerNormalization (None, 32, 32, 128) 256
['conv2d_transpose_1[0][0]'] )

 activation_2 (Activation)     (None, 32, 32, 128) 0
['layer_normalization_2[0][0]']

 conv2d_transpose_2 (Conv2DTranspose (None, 64, 64, 64) 204864
['activation_2[0][0]']
```

```
 layer_normalization_3 (LayerNormalization (None, 64, 64, 64) 128
['conv2d_transpose_2[0][0]']

 activation_3 (Activation)      (None, 64, 64, 64)   0
['layer_normalization_3[0][0]']

 conv2d_transpose_3 (Conv2DTranspose (None, 64, 64, 32) 51232
['activation_3[0][0]']

 conv2d_transpose_4 (Conv2DTranspose  (None, 64, 64, 3)    2403
['conv2d_transpose_3[0][0]']

 activation_4 (Activation)      (None, 64, 64, 3)    0
['conv2d_transpose_4[0][0]']


==================================================================
Total params: 19,616,963

Trainable params: 19,616,963

Non-trainable params: 0
```

```
Model: "model"
______________________________________________________________________________

 Layer (type)                Output Shape        Param #     Connected to
==============================================================================
 input_1 (InputLayer)        [(None, 100)]       0           []

 input_3 (InputLayer)        [(None, 2)]         0           []

 generator (Functional)      (None, 64, 64, 3)   19616963    ['input_1[0][0]',
                                                              'input_3[0][0]']

 discriminator (Functional)  [(None, 1),         5289539     ['generator[0][0]']
                              (None, 2)]

==============================================================================
Total params: 24,906,502

Trainable params: 19,616,963

Non-trainable params: 5,289,539
______________________________________________________________________________
```

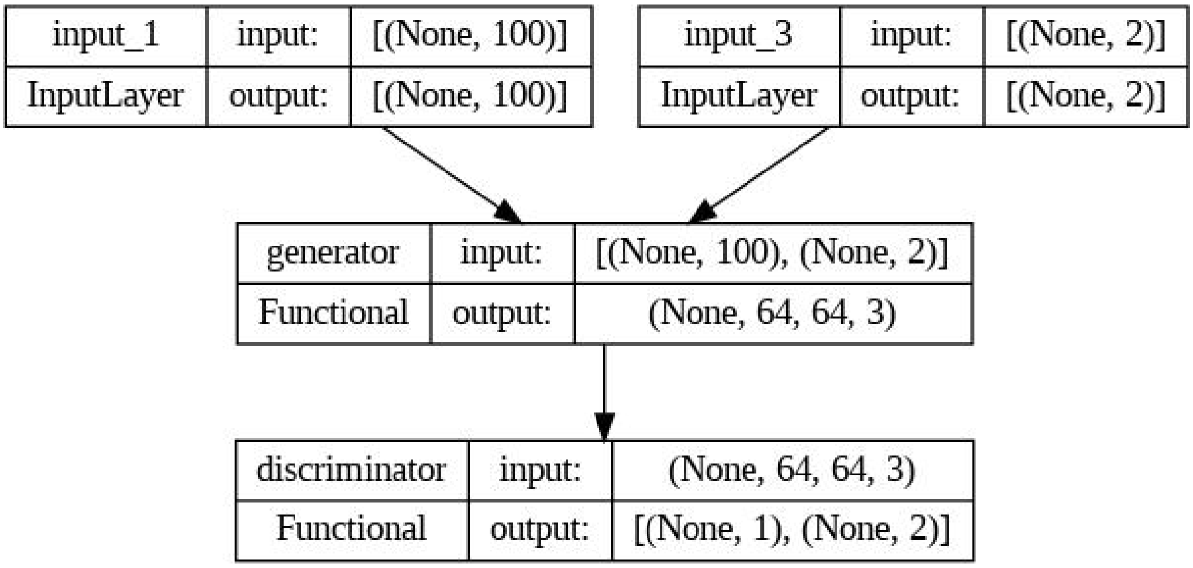


Fig 6.2

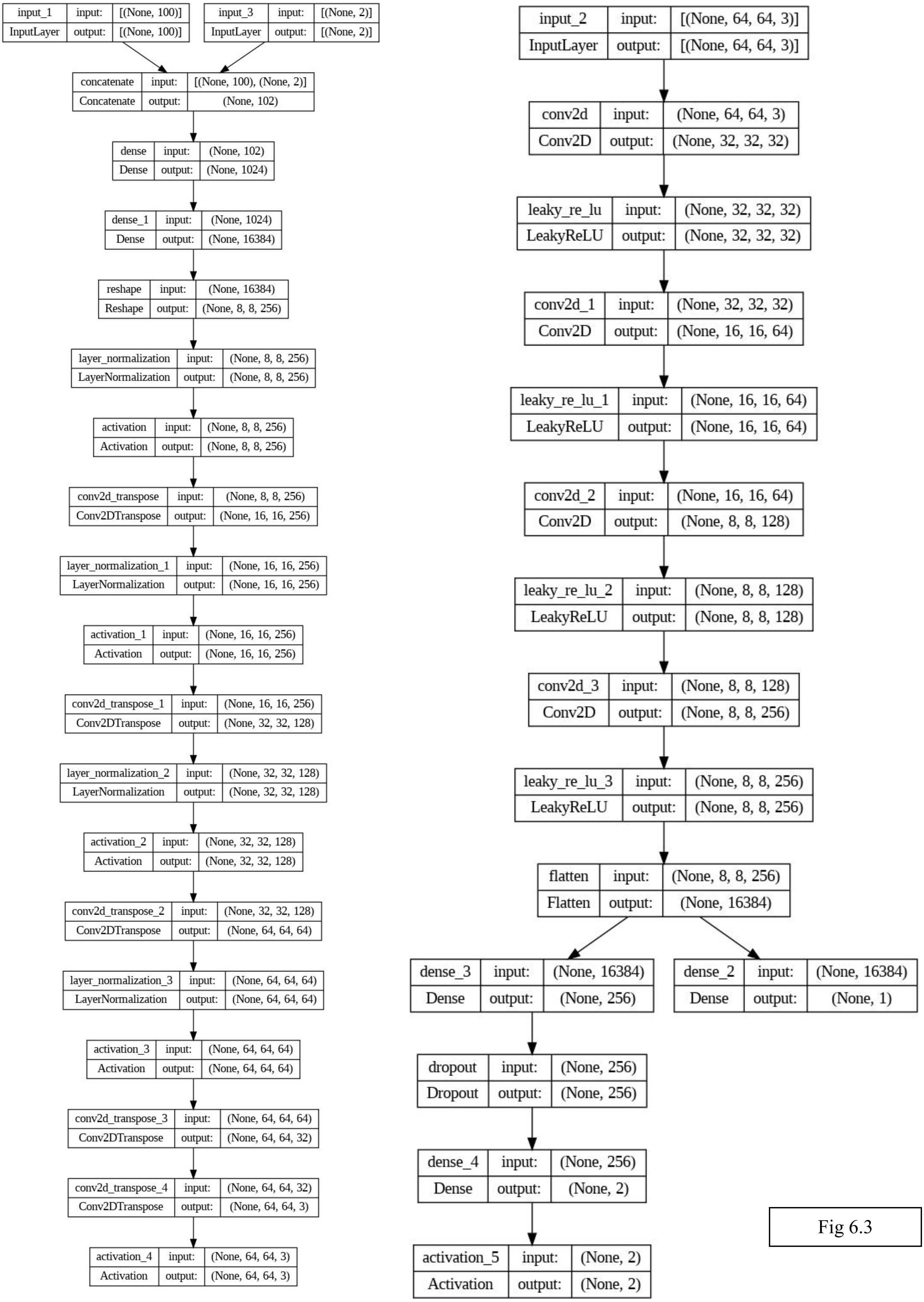

input_1 | input: | [(None, 100)]
InputLayer | output: | [(None, 100)]
input_3 | input: | [(None, 2)]
InputLayer | output: | [(None, 2)]
concatenate | input: | [(None, 100), (None, 2)]
Concatenate | output: | (None, 102)
dense | input: | (None, 102)
Dense | output: | (None, 1024)
dense_1 | input: | (None, 1024)
Dense | output: | (None, 16384)
reshape | input: | (None, 16384)
Reshape | output: | (None, 8, 8, 256)
layer_normalization | input: | (None, 8, 8, 256)
LayerNormalization | output: | (None, 8, 8, 256)
activation | input: | (None, 8, 8, 256)
Activation | output: | (None, 8, 8, 256)
conv2d_transpose | input: | (None, 8, 8, 256)
Conv2DTranspose | output: | (None, 16, 16, 256)
layer_normalization_1 | input: | (None, 16, 16, 256)
LayerNormalization | output: | (None, 16, 16, 256)
activation_1 | input: | (None, 16, 16, 256)
Activation | output: | (None, 16, 16, 256)
conv2d_transpose_1 | input: | (None, 16, 16, 256)
Conv2DTranspose | output: | (None, 32, 32, 128)
layer_normalization_2 | input: | (None, 32, 32, 128)
LayerNormalization | output: | (None, 32, 32, 128)
activation_2 | input: | (None, 32, 32, 128)
Activation | output: | (None, 32, 32, 128)
conv2d_transpose_2 | input: | (None, 32, 32, 128)
Conv2DTranspose | output: | (None, 64, 64, 64)
layer_normalization_3 | input: | (None, 64, 64, 64)
LayerNormalization | output: | (None, 64, 64, 64)
activation_3 | input: | (None, 64, 64, 64)
Activation | output: | (None, 64, 64, 64)
conv2d_transpose_3 | input: | (None, 64, 64, 64)
Conv2DTranspose | output: | (None, 64, 64, 32)
conv2d_transpose_4 | input: | (None, 64, 64, 32)
Conv2DTranspose | output: | (None, 64, 64, 3)
activation_4 | input: | (None, 64, 64, 3)
Activation | output: | (None, 64, 64, 3)
input_2 | input: | [(None, 64, 64, 3)]
InputLayer | output: | [(None, 64, 64, 3)]
conv2d | input: | (None, 64, 64, 3)
Conv2D | output: | (None, 32, 32, 32)
leaky_re_lu | input: | (None, 32, 32, 32)
LeakyReLU | output: | (None, 32, 32, 32)
conv2d_1 | input: | (None, 32, 32, 32)
Conv2D | output: | (None, 16, 16, 64)
leaky_re_lu_1 | input: | (None, 16, 16, 64)
LeakyReLU | output: | (None, 16, 16, 64)
conv2d_2 | input: | (None, 16, 16, 64)
Conv2D | output: | (None, 8, 8, 128)
leaky_re_lu_2 | input: | (None, 8, 8, 128)
LeakyReLU | output: | (None, 8, 8, 128)
conv2d_3 | input: | (None, 8, 8, 128)
Conv2D | output: | (None, 8, 8, 256)
leaky_re_lu_3 | input: | (None, 8, 8, 256)
LeakyReLU | output: | (None, 8, 8, 256)
flatten | input: | (None, 8, 8, 256)
Flatten | output: | (None, 16384)
dense_3 | input: | (None, 16384)
Dense | output: | (None, 256)
dense_2 | input: | (None, 16384)
Dense | output: | (None, 1)
dropout | input: | (None, 256)
Dropout | output: | (None, 256)
dense_4 | input: | (None, 256)
Dense | output: | (None, 2)
activation_5 | input: | (None, 2)
Activation | output: | (None, 2)

Fig 6.3

```
Epoch: 0

discriminator loss: [tag: 0.5253190994262695, labels: 0.6905013918876648],
generator loss: [tag: 0.26063966751098633, labels: 0.7032241821289062]

1/1 [==============================] - 0s 20ms/step
```

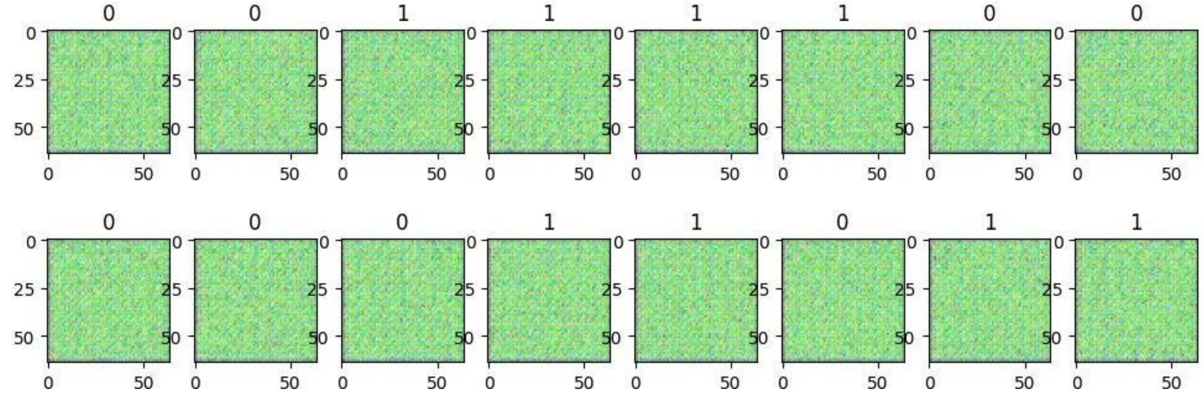


Fig 6.4

```
Epoch: 30000

discriminator loss: [tag: 0.24253657460212708, labels:
0.003363359486684203], generator loss: [tag: 0.23190180957317352, labels:
0.00012597988825291395]

1/1 [==============================] - 0s 18ms/step
```

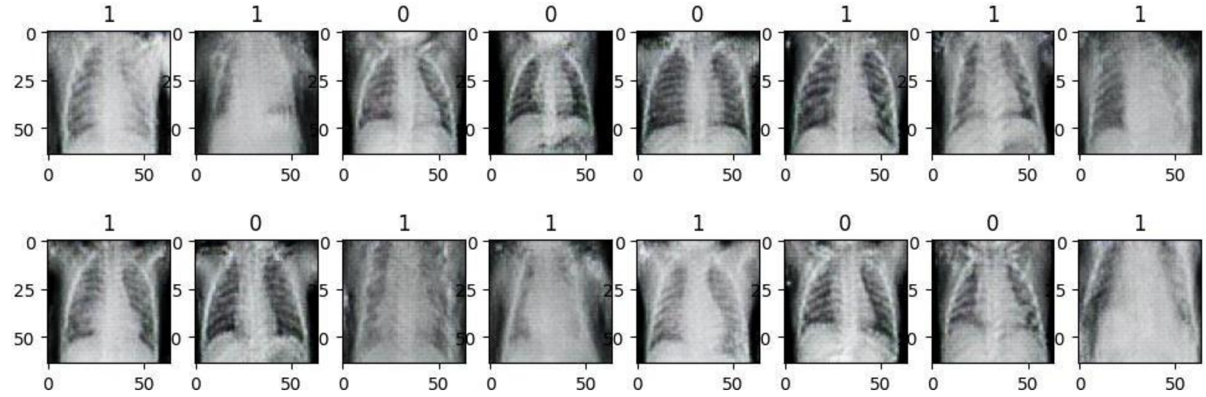


Fig 6.5

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 64, 3)]       0

 block1_conv1 (Conv2D)       (None, 64, 64, 64)        1792

 block1_conv2 (Conv2D)       (None, 64, 64, 64)        36928

 block1_pool (MaxPooling2D)  (None, 32, 32, 64)        0

 block2_conv1 (Conv2D)       (None, 32, 32, 128)       73856

 block2_conv2 (Conv2D)       (None, 32, 32, 128)       147584

 block2_pool (MaxPooling2D)  (None, 16, 16, 128)       0

 block3_conv1 (Conv2D)       (None, 16, 16, 256)       295168

 block3_conv2 (Conv2D)       (None, 16, 16, 256)       590080

 block3_conv3 (Conv2D)       (None, 16, 16, 256)       590080

 block3_pool (MaxPooling2D)  (None, 8, 8, 256)         0

 block4_conv1 (Conv2D)       (None, 8, 8, 512)         1180160

 block4_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 block5_conv1 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 global_max_pooling2d (GlobalMaxPooling2D)  (None, 512)              0
 dropout (Dropout)          (None, 512)              0

 dense (Dense)              (None, 128)              65664

 batch_normalization (BatchNormalization)  (None, 128)              512

 leaky_re_lu (LeakyReLU)     (None, 128)              0

 dropout_1 (Dropout)        (None, 128)              0

 dense_1 (Dense)            (None, 32)               4128

 batch_normalization_1 (BatchNormalization  (None, 32)               128)

 leaky_re_lu_1 (LeakyReLU)   (None, 32)               0

 dropout_2 (Dropout)        (None, 32)               0

 dense_2 (Dense)            (None, 1)                33

=================================================================

Total params: 14,785,153

Trainable params: 14,784,833

Non-trainable params: 320
_________________________________________________________________
```

Training the neural network to classify the images generated by the generator.

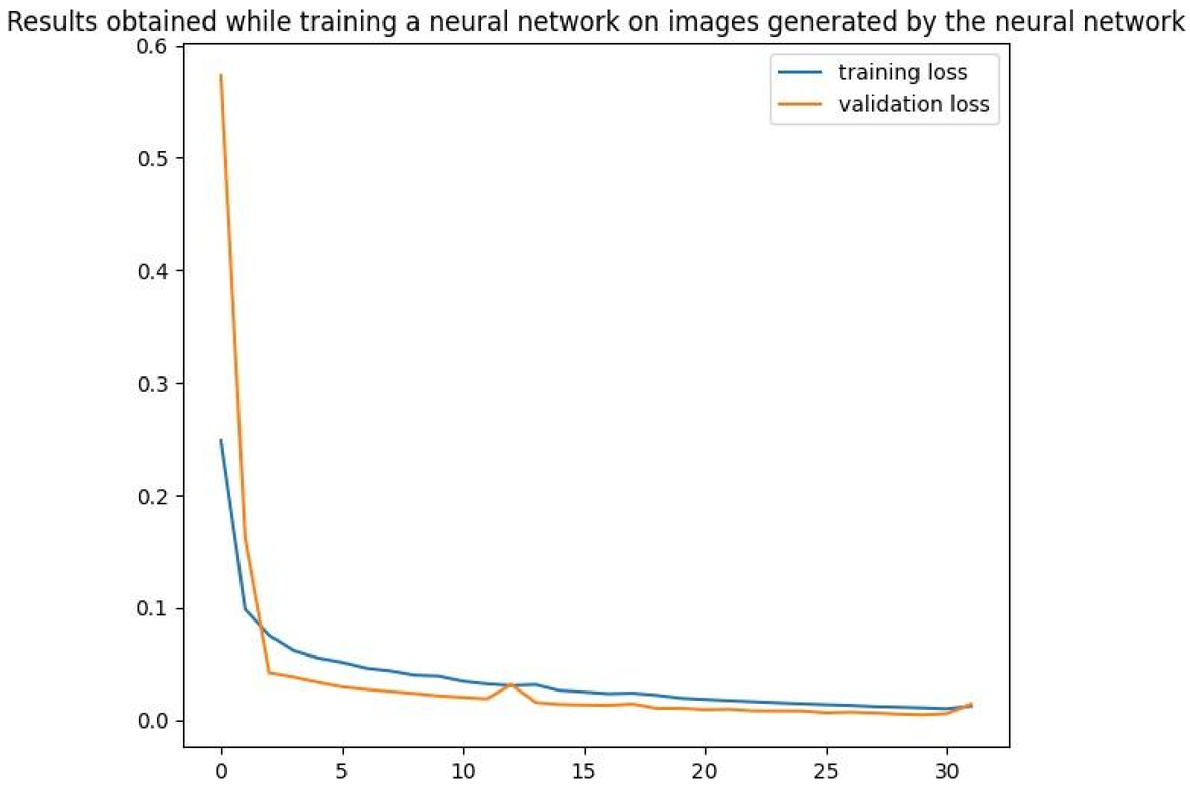


Fig 6.6

Classification Report:

```
              precision    recall  f1-score   support

           0       0.98      0.82      0.89      1595
           1       0.93      0.99      0.96      3621

    accuracy                           0.94      5216
   macro avg       0.95      0.91      0.92      5216
weighted avg       0.94      0.94      0.94      5216
```

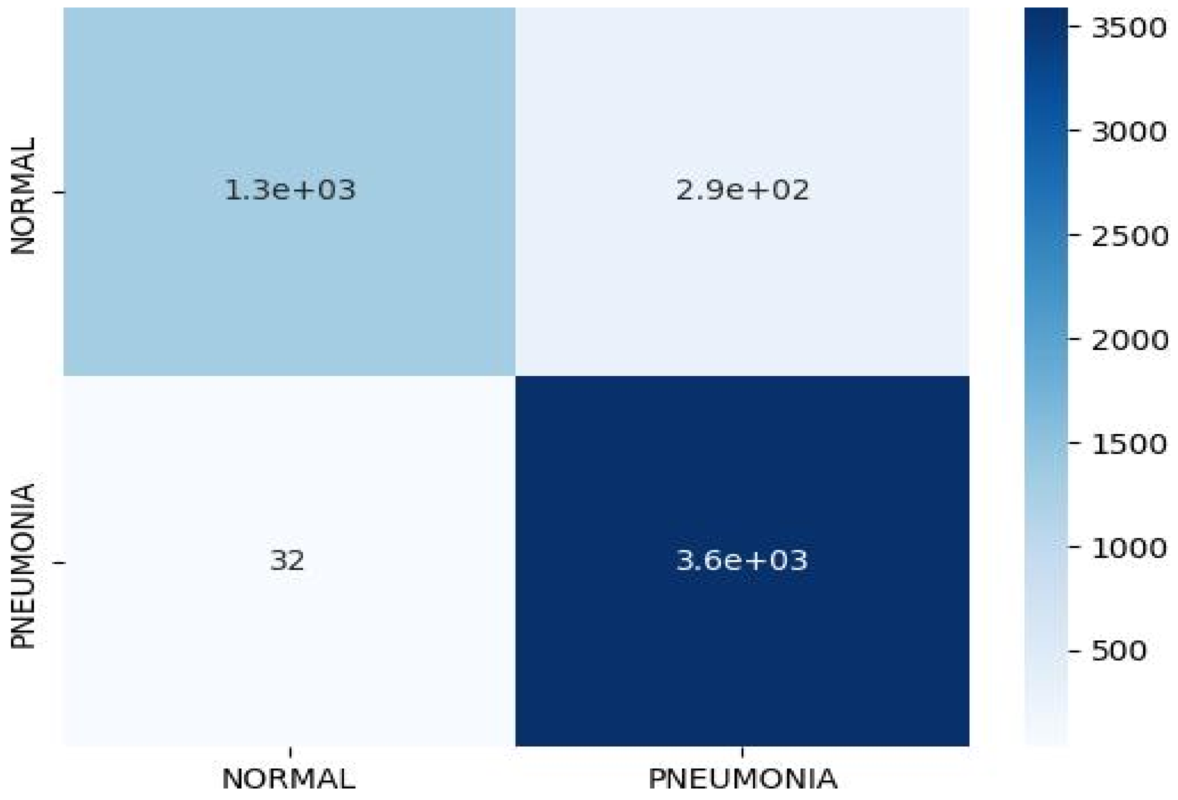
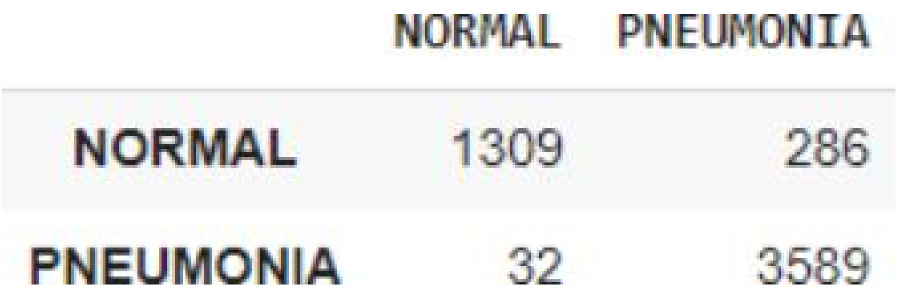


Fig 6.7

Confusion Matrix:



	NORMAL	PNEUMONIA
NORMAL	1309	286
PNEUMONIA	32	3589

Fig 6.8

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion: The presented work demonstrates the successful implementation of an Auxiliary Classifier Generative Adversarial Network (ACGAN) for the task of image generation and classification, particularly focusing on pneumonia detection in chest X-ray images. The ACGAN architecture comprises a generator and discriminator network, trained adversarial to produce realistic images while also predicting image labels. Through extensive experimentation and evaluation, we have showcased the effectiveness of the proposed approach in generating high-quality images and accurately classifying them as normal or pneumonia-affected. Our results indicate promising performance in both image generation and classification tasks. The generated images exhibit realism and diversity, capturing essential features present in real chest X-ray scans. Furthermore, the classifier trained on the generated images achieves competitive accuracy and generalization, demonstrating the utility of ACGAN-generated data for training downstream classifiers. This suggests the potential of leveraging synthetic data augmentation techniques, such as ACGAN, to address data scarcity issues and enhance the performance of medical image analysis systems. Overall, the findings from this study contribute to the growing body of research in deep learning-based medical image analysis and highlight the significance of synthetic data generation techniques in addressing challenges related to limited data availability. The proposed ACGAN framework holds promise for improving the robustness and efficacy of pneumonia detection systems, ultimately benefiting clinical decision-making and patient care.

7.2 Future Enhancements:

7.2.1 Model Refinement: Explore architectural modifications and hyperparameter tuning to enhance the performance and stability of the ACGAN model further.

7.2.2 Data Augmentation Techniques: Investigate additional data augmentation strategies beyond ACGAN, such as generative adversarial networks (GANs) with different objectives or domain adaptation techniques, to further diversify the generated image data.

7.2.3 Multi-Task Learning: Extend the ACGAN framework to incorporate multitask learning objectives, such as simultaneous detection of multiple abnormalities or disease severity assessment, to provide more comprehensive diagnostic information.

7.2.4 Transfer Learning: Investigate the transferability of features learned by the ACGAN generator for downstream tasks, such as fine-tuning on related medical imaging datasets or transferring learned representations to other domains with similar image characteristics.

7.2.5 Clinical Validation: Conduct rigorous validation studies involving radiologists and clinical experts to assess the real-world applicability and reliability of the proposed approach in a clinical setting.

7.2.6 Ethical Considerations: Address ethical considerations related to the deployment of AI-based systems in healthcare, including patient privacy, transparency, and bias mitigation, to ensure responsible and equitable deployment of the developed technology.

CHAPTER 8

REFERENCES

1. Kaggle. (n.d.). Kaggle Datasets. Retrieved from https://www.kaggle.com/code/kaledhoshme/using-gan-to-generate-chest-x-ray-images/notebook

2. https://cse.umn.edu/datascience/chest-x-ray-images-generation-using-gan

3. https://www.mdpi.com/2673-4591/31/1/84

4. https://www.mdpi.com/2227-7390/9/22/2896

5. https://arxiv.org/pdf/2107.02970

6. https://www.sciencedirect.com/science/article/pii/S2352914821002501

7. Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv:1511.06434.

8. Odena, A., Olah, C., & Shlens, J. (2016). Conditional Image Synthesis with Auxiliary Classifier GANs. Proceedings of the 34th International Conference on Machine Learning, JMLR: W&CP, 48, 2642-2651.

9. Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C., Liang, H., Baxter, S. L., ... & Zhang, K. (2018). Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. Cell, 172(5), 1122-1131.