COVID-19 I-NET: A Novel Deep Learning Based Model to Analyze COVID-19 Infection Severity using Chest X-Ray and CT Images

A PROJECT REPORT

Submitted by

KARTHIKEYAN V. [REG. NO: 211417104111]

PRASANNA VENKATESAN A. [REG. NO: 211417104193]

RAJESHKUMAR K. [REG. NO: 211417104213]

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.

ANNA UNIVERSITY, CHENNAI 600 025

AUGUST 2021

BONAFIDE CERTIFICATE

Certified that this project report **"COVID-19: A Novel Deep Learning based Model to analyze COVID-19 Infection Severity using Chest X-ray and CT Images"** is the bonafide work of "

KARTHIKEYAN V. (211417104111),

PRASANNA VENKATESAN A. (211417104193),

RAJESHKUMAR K. (211417104213).

" who carried out the project work under my supervision.

[illegible]

SIGNATURE	**SIGNATURE**
Dr. S. MURUGAVALLI, M.E., Ph.D., **HEAD OF THE DEPARTMENT**	**DR.M.SHANMUGANATHAN** **SUPERVISOR** **ASSOCIATE PROFESSOR**
DEPARTMENT OF CSE, PANIMALAR ENGINEERING COLLEGE, NAZARATHPETTAI, POONAMALLEE, CHENNAI-600 123.	DEPARTMENT OF CSE, PANIMALAR ENGINEERING COLLEGE, NAZARATHPETTAI, POONAMALLEE, CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voice Examination held on..........................

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to express our heartfelt and sincere thanks to our Directors **Tmt. C. VIJAYARAJESWARI**, **Dr. C. SAKTHIKUMAR, M.E., Ph.D.,** and **Tmt. SARANYASREE SAKTHIKUMAR B.E., M.B.A.,** for providing us with the necessary facilities for the completion of this project.

We also express our gratitude to our Principal **Dr. K. MANI, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of the CSE Department, **Dr. S.MURUGAVALLI, M.E., Ph.D.,** for the support extended throughout the project.

We would like to thank my **Project Guide Dr. M. SHANMUGANATHAN** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

NAME OF THE STUDENTS
KARTHIKEYAN V.
PRASANNA VENKATESAN A.
RAJESHKUMAR K.

ABSTRACT

COVID-19 pandemic is affecting numerous lives all around the world. Effective care and treatment planning for COVID-19 infected patients are mandatory. This infectious virus caused by severe acute respiratory syndrome corona virus 2 (SARS-CoV-2) requires primary investigation to access infection severity progression. The assessment metrics of COVID-19 infected patients involve lung involvement and opacity extent from CXR (Chest X-ray images) and CT scan. CXR (Chest x-ray images) are preferred over CT scans because of inferring a feasibly possible solution available to infected patients. We also analyzed CT scans of infected individuals to gain insights on progression. The proposed model (COVID-19 I-NET a deep convolution neural network) using U-NET and CHEST-X NET architecture resulted in image segmentation of infected areas from CXR and CT scan images respectively. A system-aided method is used for the assignment of scores to infection severity and progression on CXR. Deep learning neural architecture such as CHEST-X NET and U-NET, PSP NET are evaluated and tabulated upon CXR and CT scan images respectively. The segmentation results are promising and leading to be used as a tool for accessing image segmentation, heat map, lung area progression, and opacity of infected regions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	**DESCRIPTION**
CXR	Chest X-ray Images
CT	Computed Tomography
PSP	Pyramid Scene Parsing Network
SMM	Semantic Segmentation Model
CNN	Convolution Neural Network
ROI	Region Of Interest
DNN	Deep Neural Network

1. INTRODUCTION

1.1 OVERVIEW

COVID-19 is a fast-spreading disease that is causing thousands of deaths daily, all over the world. Early diagnosis of this disease proved to be one of the most effective methods for disease control. A large number of COVID-19 patients is rendering health care systems in many countries, overwhelmed. Hence, automation of identifying and analyzing the infected lung regions would be really important. The RTPCR (Real-Time Reverse Transcription-Polymerase Chain Reaction is being followed as the standard approach for COVID-19[1-2] screening. RT-PCR can detect the viral RNA in specimens obtained by nasopharyngeal swab, oropharyngeal swab, Broncho alveolar lavage, or tracheal aspirate. However, a variety of recent studies indicate that RTPCR testing suffers from low sensitivity, approximately around 71%, whereby repeated testing is needed for accurate diagnosis. Furthermore, RT-PCR screening is time-consuming and has increasing availability limitations due to a shortage of required material. An alternative solution to RT-PCR for COVID-19 screening in medical imaging like X-ray or computed tomography (CT). Medical imaging technology has made significant progress in recent years and is now a commonly used method for diagnosis, as well as for quantification assessment of numerous diseases. The chest CT screening is a routine diagnostic tool for pneumonia around the world. Since both COVID-19 and Pneumonia are known to infect the lungs mainly, chest CT imaging [3-5] has been recommended for COVID-19 diagnosis. In addition, CT imaging is playing an important role in COVID-19 severity assessment, as well as disease monitoring. COVID-19 infected areas can be identified on CT images by ground-glass opacity (GGO) in the early infection stage and by pulmonary consolidation in the late infection stage. In comparison to the RT-PCR test, several studies showed that a CT scan is more sensitive and effective for COVID-19 screening even without the occurrence of clinical symptoms. Even though increasing CT scan resolution and number of slices resulted in higher sensitivity and accuracy, these improvements also increased the workload. Also, annotations of medical images are often influenced by clinical experience. Automated medical image analysis could be a better solution for these challenges. The field of Image segmentation has been developing rapidly with the development of advanced deep-learning methods in Artificial Intelligence.

1.2 PROBLEM DEFINITION

The aim of medical image segmentation (MIS) is the automated identification and labelling of regions of interest (ROI). The quantification of COVID-19 infection in CT images using deep learning has not been investigated. Clinically there is no automatic tool to quantify the infection volume for COVID-19 patients.

In recent studies, medical image segmentation models based on neural networks proved powerful prediction capabilities and achieved similar results as radiologists regarding performance. It would be a helpful tool to implement such an automatic segmentation for COVID-19 infected regions as clinical decision support for physicians. By automatically highlighting abnormal features and ROIs, image segmentation can aid radiologists in diagnosis, disease course monitoring, and reduction of time-consuming inspection processes, and improvement of accuracy.

2. LITERATURE SURVEY

[1] Title: COVID-19 Chest X-Ray and CT scan Image Data Collection
Description: COVID-19 pandemic, is it crucial to streamline diagnosis. Data is the first step to developing any diagnostic tool or treatment. While there exist large public datasets of more typical chest X-rays, there is no collection of COVID-19 chest X-rays or CT scans designed to be used for computational analysis. In this paper[1], author Joseph Paul Cohen created a public database of pneumonia cases with chest X-ray or CT images, specifically COVID19 cases as well as MERS, SARS, and ARDS. Data is collected from public sources in order not to infringe patient confidentiality. This would provide essential data to train and test a Deep Learning based system, likely using some form of transfer learning. These tools could be developed to identify COVID-19 characteristics as compared to other types of pneumonia or in order to predict survival.

[2] Title: Chest X-Ray based Neural Network Analysis

Description: In this study, author Linda Wang [2] introduced COVID-Net, a deep convolutional neural network design tailored for the detection of COVID-19 cases from chest X-ray (CXR) images that is open source and available to the general public. To the best of the authors' knowledge, COVID-Net is one of the first open source network designs for COVID-19 detection from CXR images at the time of initial release. They also introduce COVIDx, an open access benchmark dataset that we generated comprising of 13,975 CXR images across 13,870 patient cases, with the largest number of publicly available COVID-19 positive cases to the best of the authors' knowledge. Furthermore, we investigate how COVID-Net makes predictions using an explain ability method in an attempt to not only gain deeper insights into critical factors associated with COVID cases, which can aid clinicians in improved screening, but also audit COVID-Net in a responsible and transparent manner to validate that it is making decisions based on relevant information from the CXR images.

[3] Title: CT based Neural Network Analysis

Description: In this study author Hayden Gunraj [3] introduced COVIDNet-CT, a deep convolutional neural network architecture that is tailored for detection of COVID-19 cases from chest CT images via a machine-driven design exploration approach. Additionally, we introduce COVIDx-CT, a benchmark CT image dataset derived from CT imaging data collected by the China National Center for Bioinformation comprising 104,009 images across 1,489 patient cases. Furthermore, in the interest of reliability and transparency, we leverage an explainability-driven performance validation strategy to investigate the decision-making behavior of COVIDNet-CT, and in doing so ensure that COVID Net-CT makes predictions based on relevant indicators in CT images. Both COVIDNet-CT and the COVIDx-CT dataset are available to the general public in an open-source and open access manner as part of the COVID-Net initiative.

[4] Title: Improving Performance of Neural Network Architecture and Evaluation

Description: In this work author, Simonyan, K [4] investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3 × 3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our

ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localization and classification tracks respectively.

[5] Title: Weakly Supervised Deep Learning Model for CT scan Images

Description: In this study, author Zheng, C [5] developed a deep learning-based model for automatic COVID-19 detection on chest CT is helpful to counter the outbreak of SARS-CoV-2. A weakly-supervised deep learning-based software system was developed using 3D CT volumes to detect COVID-19. For each patient, the lung region was segmented using a pre-trained U-Net; then the segmented 3D lung region was fed into a 3D deep neural network to predict the probability of COVID-19 infectious. 499 CT volumes collected from Dec. 13, 2019, to Jan. 23, 2020, were used for training and 131 CT volumes collected from Jan 24, 2020, to Feb 6, 2020, were used for testing. The deep learning algorithm obtained 0.959 ROC AUC and 0.976 PR AUC. There was an operating point with 0.907 sensitivity and 0.911 specificity in the ROC curve. When using a probability threshold of 0.5 to classify COVID-positive and COVID-negative, the algorithm obtained an accuracy of 0.901, a positive predictive value of 0.840 and a very high negative predictive value of 0.982. The algorithm took only 1.93 seconds to process a single patient's CT volume using a dedicated GPU. Our weakly-supervised deep learning model can accurately predict the COVID-19 infectious probability in chest CT volumes without the need for annotating the lesions for training. The easily-trained and high performance deep learning algorithm provides a fast way to identify COVID-19 patients, which is beneficial to control the outbreak of SARS-CoV-2.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Many research projects have been conducted for COVID-19 detection using deep learning techniques in image analysis of X-Ray and CT scans and they have given remarkable results. Yet, detailed segmentation of those images has been less appealing. A recent study designed a binary classifier (COVID-19, No information) and a multi classifier (COVID-19, No Information, Pneumonia) using a CNN with X-Ray images as an input, reaching an output of 0.98 for binary classes and 0.87 for a multi-class classifier for COVID19 severity detection. Another study used Xception and ResNet50V2 networks, resulting in an accuracy of 0.99 for the target class.

A detailed approach to localize abnormalities in COVID-19 Chest X-ray and CT scan using Neural Networks to aid physicians and radiologist, By automatically highlighting abnormal features and ROIs, image segmentation in diagnosis, disease course monitoring, and reduction of time-consuming inspection processes, and improvement of accuracy. Regionalizing, infected regions upon CT and CXR lung images and classification based upon severity to avail immediate response to highly prone COVID-19 affected individuals. This methodology is being researched, yet a possible solution to aid as a pre-eminent tool is a major drawback at the time of this study

3.2 PROPOSED SYSTEM

The primary goal of this study is to assess the feasibility of automated severity scoring of COVID-19 using deep learning techniques. . We develop and evaluate deep neural networks that can score Chest X-Rays and CT scan of patients with COVID-19.

In this work, we push towards creating an accurate and state-of-the-art MIS pipeline for COVID-19 lung infection segmentation, which is capable of being trained on small datasets consisting of CT volumes and x-ray. To avoid overfitting, we exploit extensive on-the-fly data augmentation, as well as diverse pre-processing methods. To further reduce the risk of overfitting, we implement the standard U-Net architecture, PSP and CHEST-X RAY NET. Moreover, we use 5-step cross validation for reliable performance evaluation. Implementing such a validation method would be a key component in a system that prioritizes patients by severity of infection. It would identify an infection and output its key spatial features such as location, distribution, and shape parameters.

To develop a deep learning (DL)-based system for automatic segmentation and quantification of infection regions as well as the entire lung from chest CT scans and CXR images.

3.3 REQUIREMENT ANALYSIS AND SPECIFICATION

3.3.1 INPUT REQUIREMENTS

The proposed system is expected to receive two different types of datasets such as CXR and CT scan Images.

CXR DATASET:

The train dataset comprises 6,334 chest scans in DICOM format, which are de-identified to protect patient privacy. All images were labelled by a panel of experienced radiologists for the presence of opacities as well as overall appearance. The hidden test dataset is of roughly the same scale as the training dataset.

- train_study_level.csv - the train study-level metadata, with one row for each study, including correct labels.

- train_image_level.csv - the train image-level metadata, with one row for each image, including both correct labels and any bounding boxes in a dictionary format. Some images in both test and train have multiple bounding boxes.

a. train_study_level.csv:

- id - unique study identifier
- Negative for Pneumonia - 1 if the study is negative for pneumonia, 0 otherwise
- Typical Appearance - 1 if the study has this appearance, 0 otherwise
- Indeterminate Appearance - 1 if the study has this appearance, 0 otherwise
- Atypical Appearance - 1 if the study has this appearance, 0 otherwise

b. train_image_level.csv:

- id - unique image identifier
- boxes - bounding boxes in easily-readable dictionary format
- label - the correct prediction label for the provided bounding boxes.

CT SCAN DATASET:

- Medseg part**:**

This is a dataset of 100 axial CT images from >40 patients with COVID-19 that were converted from openly accessible JPG images.
(i) *images_medseg.npy* - training images – 100 slices 512x512 size
(ii) *s_medseg.npy* - training masks – 100 masks with 4 channels: (0 - "ground glass", 1 - "consolidations", 2 - "lungs other", 3 - "background")
(iii) *test_images_medseg.npy* - test images – 10 slices 512x512 size

- Radiopedia part:

Segmented 9 axial volumetric CTs from Radiopaedia. This dataset includes whole volumes and includes, therefore, both positive and negative slices (373 out of the total of 829 slices have been evaluated by a radiologist as positive and segmented). These volumes are converted and normalized in a similar way as above.

(i) *images_radiopedia.npy* **-** training images – 829 slices 512x512 size
(ii) *masks_radiopedia.npy* - training masks – 829 masks with 4 channels: (0 - "ground glass", 1 - "consolidations", 2 - "lungs other", 3 - "background")

3.3.2 OUTPUT REQUIREMENTS

The proposed system generates localized segmented images of CT scan and CXR images automatically highlighting abnormal features and ROIs.

CXR IMAGES:

- For each test image, we predict a bounding box and class for all findings. If we predict that there are no findings, we should create a prediction of "none 1 0 0 1 1" ("none" is the class ID for no finding, and this provides a one-pixel bounding box with a confidence of 1.0). Further, for each test study, you should make a determination within the following labels:

'Negative for Pneumonia' 'Typical Appearance' 'Indeterminate Appearance' 'Atypical Appearance'

To make a prediction of one of the above labels, create a prediction string similar to the "none" class above: e.g. `atypical 1 0 0 1 1`. The images in DICOM format, which means they contain additional data that might be useful for visualizing and classifying.

CT SCAN:

These corresponding dataset are mentioned in input requirements, generated mapped image is of 829 masks with 4 channels: (0 - "ground glass", 1 - "consolidations", 2 - "lungs other", 3 - "background") for CT can. The test images are evaluated using for CT scan- test images – 10 slices 512x512 size.

3.3.3 FUNCTIONAL REQUIREMENTS

The system is expected to receive two different types of datasets as inputs namely, Chest X-ray NET and CT Scan Images. With the perception of datasets fed to an inference model, it is expected to derive an automated infected region analysis extent. For radiological scoring, CXR images are preferred over CT scan the assessment metrics is as follows,

The two assessment metrics used in the radiological scoring are geographic extent and opacity extent. i.e., for geographic extent, the extent of lung involvement by ground glass opacity or consolidation of each lung (with the right and left lung scored separately) is scored as 0 = no involvement; 1 = 75% involvement. The scores are then added together, and the total geographic extent score ranges from 0 to 8 (right + left lung). For opacity extent, the degree of opacity is: 0 = no opacity; 1 = ground-glass opacity; 2 = mix of consolidation and ground-glass opacity (less than 50% consolidation); 3 = mix of consolidation and ground-glass opacity (more than 50% consolidation); 4 = complete white-out. The scores are similarly added together, and the total opacity extent score ranges from 0 to 8 (right + left lung). The average scores are then calculated by the radiologists and it is used in the training of deep neural networks.

The inter-reader agreement assessed by intra-class correlation coefficient was 0.92 (95% CI: 0.91- 0.93) for the geographic extent scores, and 0.87 (95% CI: 0.85-0.89) for the opacity extent scores. After radiological scoring, all CXR image data used in this study, underwent data processing to facilitate faster and efficient training. To avoid the deep neural networks from learning irrelevant visual cues while making severity scoring predictions, the top 8% of the CXR data were cropped to remove boundary artefacts and embedded metadata, that contain patients information. Moreover, all CXR image data were resized to the same dimensions to enable the training of the deep neural networks in this study. In the end, the geographic extent scores (with a dynamic range of 0 to 8) and opacity extent scores (with a dynamic range of 0 to 8) were re-mapped to a unified dynamic range from 0 to 1.

3.4 TECHNOLOGY STACK

The major tools that have been used in our project are:

- DOCKER DESKTOP
- JUPYTER NOTEBOOK
- PYCHARM
- KAGGLE
- NUMPY
- TENSORFLOW
- KERAS

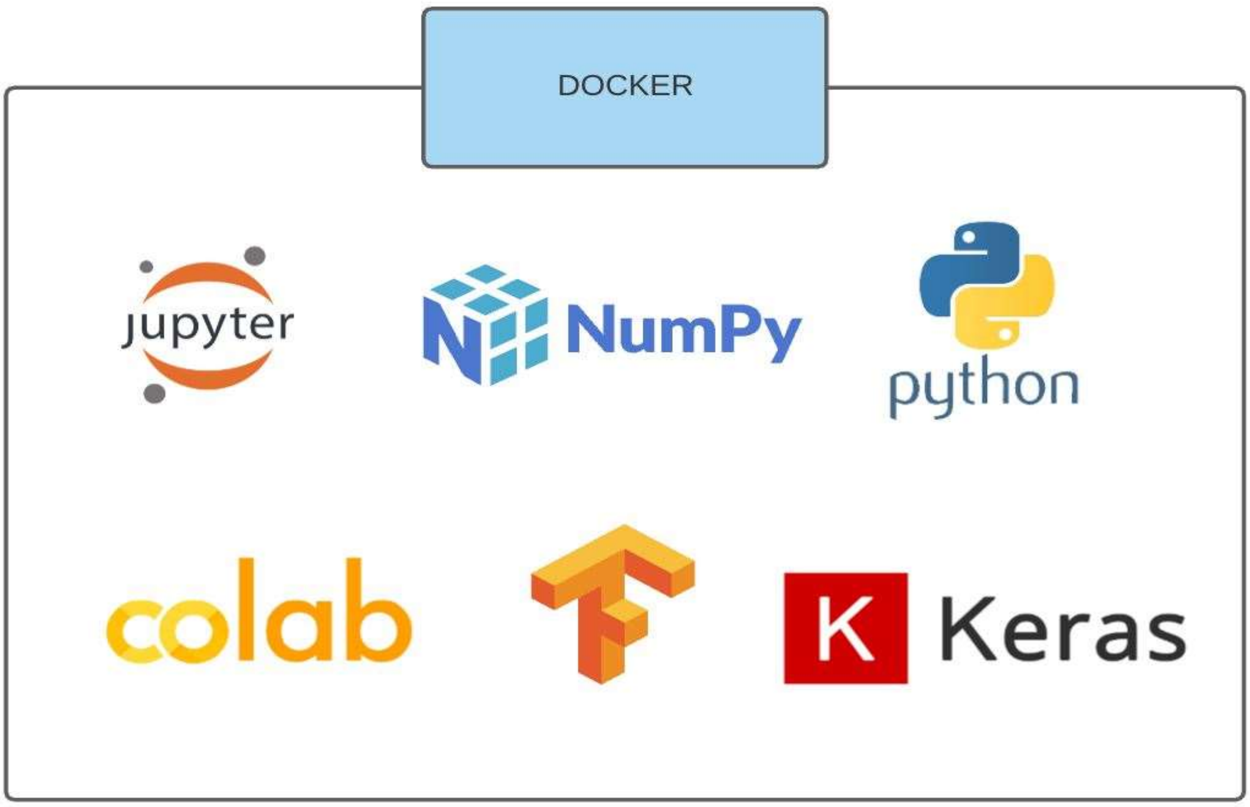


Figure 1 - TECHNOLOGY STACK

Docker Desktop is an easy to install application for your mac or windows environment that enables you to build and share containerized application and micro-services.

Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text and multimedia resources in a single document.

Kaggle allow user to find and publish data sets, explore and build models in a web-based data-science environment.

NumPy is the fundamental package for scientific computing ‘in python. it is python library that provides a multidimensional array object, various derived objects.

TensorFlow is an open source artificial intelligence library, using data flow graphs to build models.it allows developers to create large-scale neural networks with many layers.

Keras allows users to productive deep models on smartphone (IOS and Android) on the web, or on the Java Virtual Machine.

4. SYSTEM ARCHITECTURE

4.1 METHODOLOGICAL OVERVIEW

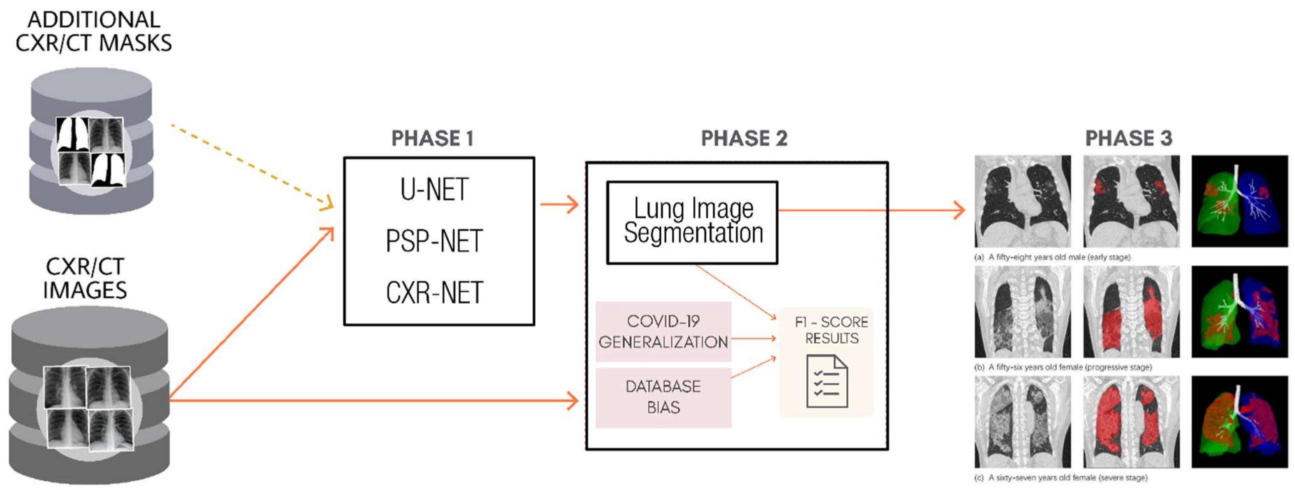


Figure 2 – METHODOLOGICAL OVERVIEW

4.2 ARCHITECTURAL OVERVIEW

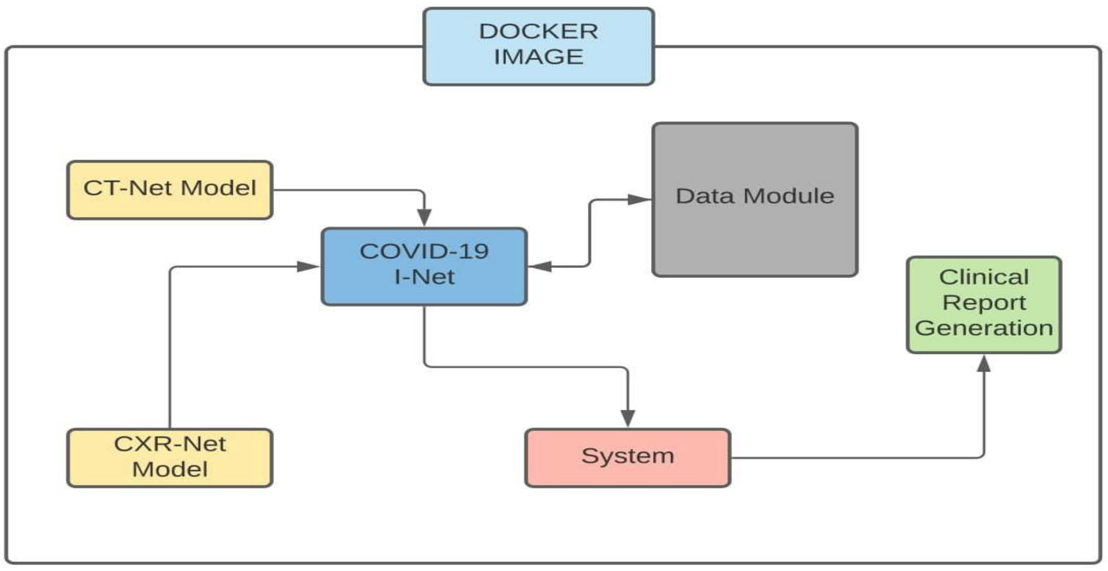


Figure 3 – ARCHITECTURE OVERVIEW

4.3 UML DIAGRAM

4.3.1 ACTIVITY DIAGRAM

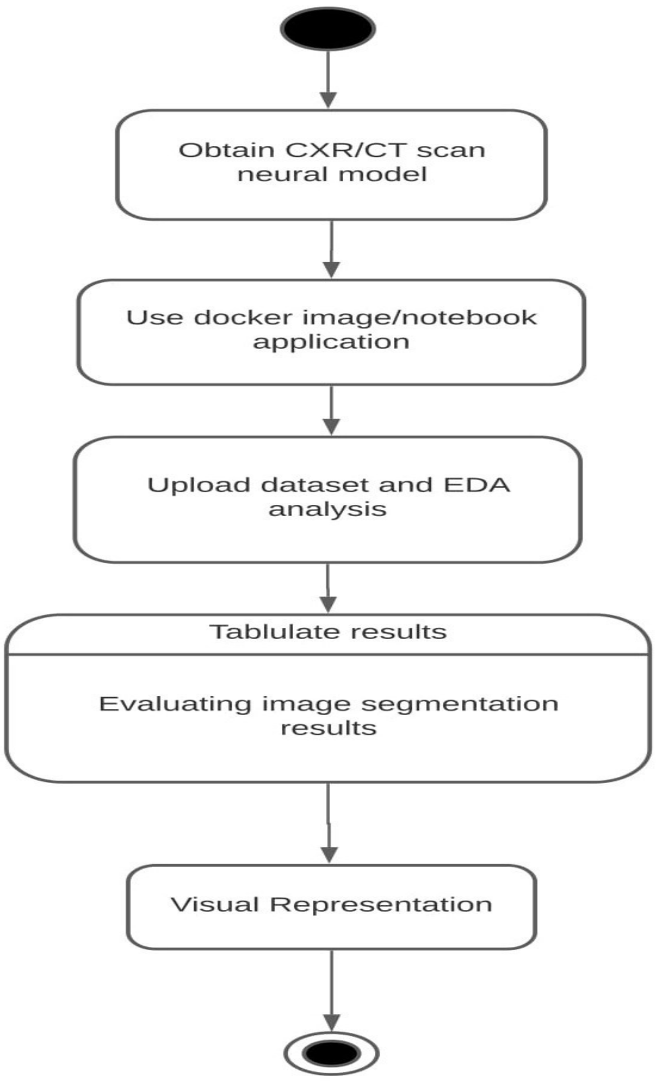

Obtain CXR/CT scan neural model
Use docker image/notebook application
Upload dataset and EDA analysis
Tablulate results
Evaluating image segmentation results
Visual Representation

Figure *4* - ACTIVITY DIAGRAM

4.3.2 USE CASE DIAGRAM

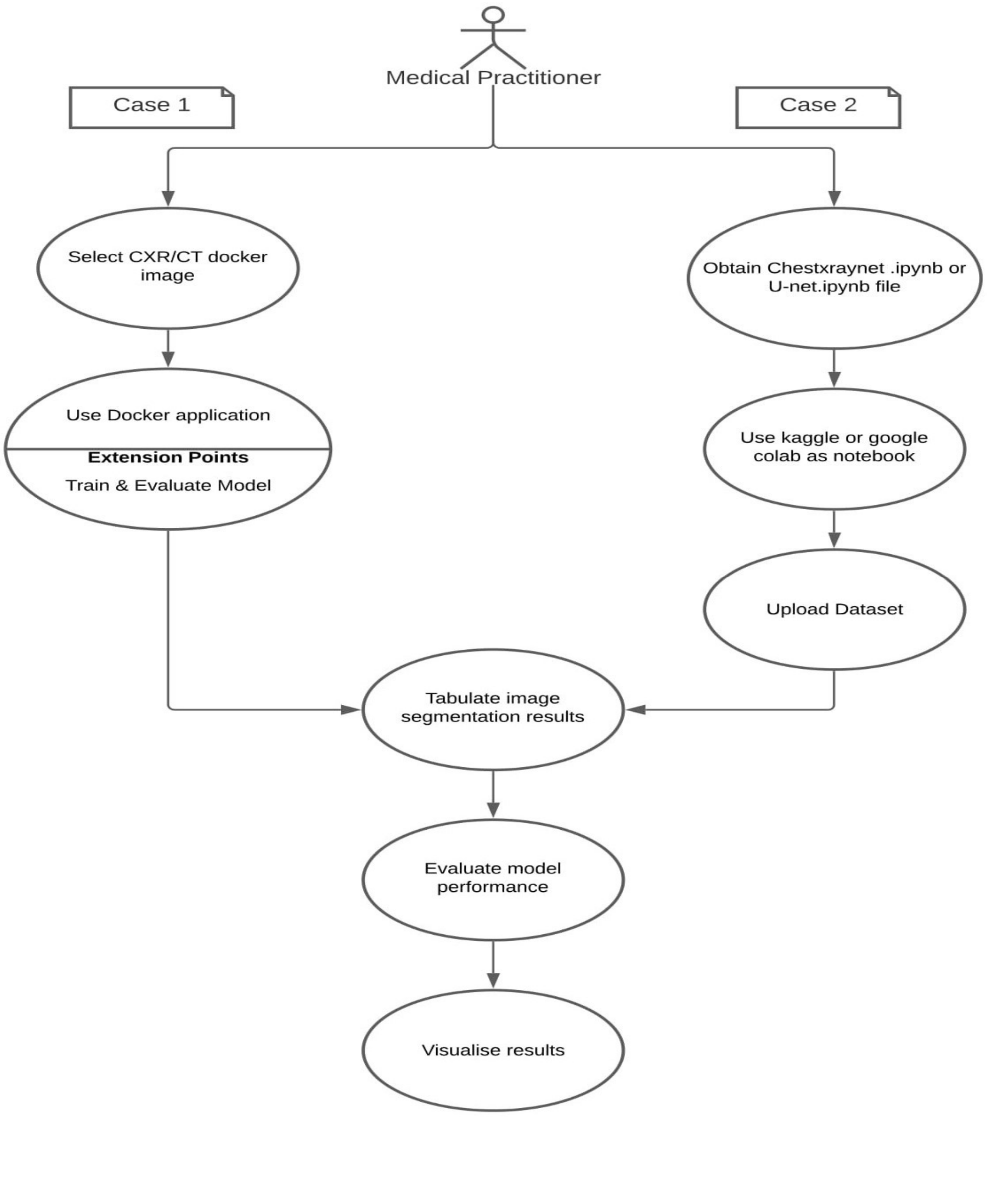

Medical Practitioner
Case 1
Case 2
Select CXR/CT docker image
Obtain Chestxraynet .ipynb or U-net.ipynb file
Use Docker application
Extension Points
Train & Evaluate Model
Use kaggle or google colab as notebook
Upload Dataset
Tabulate image segmentation results
Evaluate model performance
Visualise results

Figure *5* – USE CASE DIAGRAM

4.3.3 SEQUENCE DIAGRAM

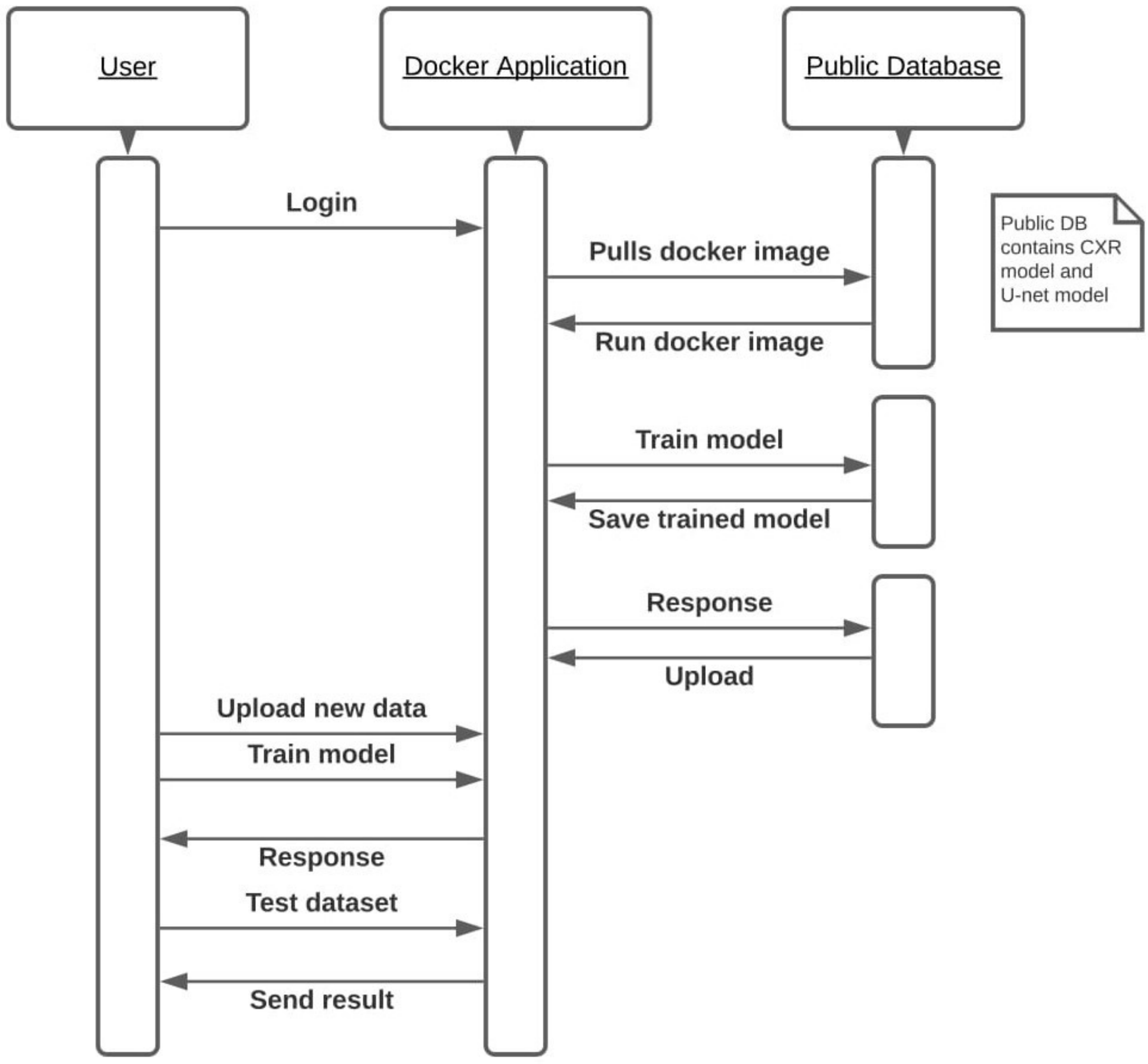


Figure 6 – SEQUENCE DIAGRAM

5. SYSTEM MODULE DESIGN SPECIFICATION

5.1 CHEST-XRAY NET MODEL

The Chest x-ray Net algorithm that can detect pneumonia from chest X-rays at a level exceeding practicing radiologists. Our algorithm, CheXNet, is a 121-layer convolutional neural network trained on ChestX-ray14, currently the largest publicly available chest Xray dataset, containing over 100,000 frontal view X-ray images with 14 diseases. Four practicing academic radiologists annotate a test set, on which we compare the performance of CheXNet to that of radiologists. We find that CheXNet exceeds average radiologist performance on the F1 metric. We extend CheXNet to detect all 14 diseases in ChestX-ray14 and achieve state of the art results on all 14 diseases.

5.2 U-NET BASED IMAGE SEGEMENTATION MODEL

The U-Net CNN architecture is a fully convolutional network (FCN) that has two main components: a contraction path, also called an encoder, which captures the image information; and the expansion path, also called decoder, which uses the encoded information to create the segmentation output. We used the U-Net CNN architecture with some small changes: we included dropout and batch normalization layers in each contracting and expanding block. These additions aim to improve training time and reduce overfitting. Figure 4 presents our adapted U-Net architecture. The two main parts of the network's architecture are contractive and expansive. The contracting path consists of several patches of convolutions with filters of size 3×3 and unity strides in both directions, followed by ReLU layers. The first path extracts the key features of the input and gives a feature vector of a specific length. The second path pulls information from the contractive path through copying and cropping mechanism, and from the feature vector using up convolutions, and by successive implementation, it generates an output segmentation map. The linking of the first and the second paths together allows the network to attain highly accurate information from the contractive path, thus generating the segmentation mask as close as possible to the intended output.

U-Net outperformed with higher efficiency which results are discussed as follows. Overall U-Net segmentation performance for the test set for each source we used to compose the lung segmentation database considering the Jaccard distance and the Dice coefficient metrics. As we expected, our manually created masks underperformed when compared to the other sources' results, this may have happened because our masks were not made by professional radiologists.

5.3 PYRAMID SCENE PARSING MODEL

In a deep neural network, the size of the receptive field roughly shows how much we use information context. Although the theoretical receptive field of ResNet is already larger than the input image, the empirical receptive field of CNN is much smaller than the latter, especially on higher level layers. Due to this, many networks do not sufficiently incorporate the momentous global scenery prior. To solve this issue, we propose an effective global level prior representation. Global level average pooling is an efficient baseline model as the global contextual prior, which is generally used in image classification tasks. It was successfully applied to semantic segmentation. But regarding the complex scene images in ADE20K, this strategy will not be enough to cover the necessary information. Pixels in a scene should be annotated on many things and objects. The direct fusion of them, to form a single vector may result in losing the spatial relation and cause ambiguity. Global context along with sub-level region context helps distinguish among various categories. Thus, a more powerful representation could be fused with information from different sub-regions with these receptive fields. A similar result was drawn in the classical work of scene/image classification.

5.4 SEGNET MODEL

SegNet is a Deep Neural Network designed for semantic segmentation. It consists of an encoder and a decoder network. The task requires the network to converge using highly imbalanced datasets since large areas of road images consist of classes such as road, sidewalk, and sky. We demonstrated numerically how the dataset used in this work exhibit disparity in class representation, from the dataset section. Due to this consequence, SegNet was our first choice.

To perform lung segmentation, we applied a CNN approach using the U-Net architecture. The U-Net input is the CXR image, and the output is a binary mask that indicates the region of interest (ROI). Thus, the training requires a previously set of binary masks. The COVID-19 dataset used does not have manually created binary masks for all images. Thus, we adopted a semiautomated approach to creating binary masks for all CXR images. First, we used three additional CXR datasets with binary masks to increase the training sample size and some binary masks provided by v7labs2. We then trained the U-Net model and used it to predict the binary masks for all images in our dataset. After that, we reviewed all predicted binary masks and manually created masks for those CXR images that the model was unable to generalize well. We repeated this process until we judged the result satisfactory and achieved a good intersection between target and obtained regions.

6. SYSTEM DESIGN

6.1 EDA ANALYSIS

```
import numpy as np
import pandas as pd
from tqdm import tqdm
import glob
import os
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import seaborn as sns
import pprint
import pydicom as dicom
from pydicom.pixel_data_handlers.util import apply_voi_lut
import albumentations as A
import cv2
import wandb

from PIL import Image
from colorama import Fore, Back, Style
# colored output
y_ = Fore.YELLOW
r_ = Fore.RED
g_ = Fore.GREEN
b_ = Fore.BLUE
m_ = Fore.MAGENTA

sns.set(font="Serif",style ="white")
from kaggle_secrets import UserSecretsClient
user_secrets = UserSecretsClient()
api_key = user_secrets.get_secret("api_key")

os.environ["WANDB_SILENT"] = "true"

CONFIG = {'competition': 'siim-fisabio-rsna', '_wandb_kernel': 'ruch'}

! wandb login $api_key
train_image_level = pd.read_csv("../input/siim-covid19-detection/train_image_leve
l.csv")
train_study_level = pd.read_csv("../input/siim-covid19-detection/train_study_leve
l.csv")
train_image_level.head()
train_study_level.head()
train_directory = "../input/siim-covid19-detection/train/"
test_directory = "../input/siim-covid19-detection/test/"

train_study_level['StudyInstanceUID'] = train_study_level['id'].apply(lambda x: x
.replace('_study', ''))
del train_study_level['id']
```

```
train_df = train_image_level.merge(train_study_level, on='StudyInstanceUID')
train_df.head()
training_paths = []

for sid in tqdm(train_df['StudyInstanceUID']):
    training_paths.append(glob.glob(os.path.join(train_directory, sid +"/*/*"))[0
])

train_df['path'] = training_paths
train_df.head()
params = {'legend.fontsize': 'x-large',
          'figure.figsize': (20, 32),
         'axes.labelsize': 'x-large',
         'axes.titlesize':'x-large',
         'xtick.labelsize':'x-large',
         'ytick.labelsize':'x-large'}
pylab.rcParams.update(params)

fig, ax = plt.subplots(4,2)
sns.kdeplot(train_df["Negative for Pneumonia"], shade=True,ax=ax[0,0],color="#ffb
4a2")
ax[0,0].set_title("Negative for Pneumonia Distribution",font="Serif", fontsize=20
,weight="bold")
sns.countplot(x = train_df["Negative for Pneumonia"], ax=ax[0,1],color="#ffb4a2")
ax[0,1].set_title("Negative for Pneumonia Distribution",font="Serif", fontsize=20
,weight="bold")

sns.kdeplot(train_df["Typical Appearance"], shade=True,ax=ax[1,0],color="#e5989b"
)
ax[1,0].set_title("Typical Appearance Distribution",font="Serif", fontsize=20,wei
ght="bold")
sns.countplot(x = train_df["Typical Appearance"], ax=ax[1,1],color="#e5989b")
ax[1,1].set_title("Typical Appearance Distribution",font="Serif", fontsize=20,wei
ght="bold")

sns.kdeplot(train_df["Indeterminate Appearance"], shade=True,ax=ax[2,0],color="#b
5838d")
ax[2,0].set_title("Indeterminate Appearance Distribution",font="Serif", fontsize=
20,weight="bold")
sns.countplot(x = train_df["Indeterminate Appearance"], ax=ax[2,1],color="#b5838d
")
ax[2,1].set_title("Indeterminate Appearance Distribution",font="Serif", fontsize=
20,weight="bold")

sns.kdeplot(train_df["Atypical Appearance"], shade=True,ax=ax[3,0],color="#6d6875
")
ax[3,0].set_title("Atypical Appearance Distribution",font="Serif", fontsize=20,we
ight="bold")
sns.countplot(x = train_df["Atypical Appearance"], ax=ax[3,1],color="#6d6875")
ax[3,1].set_title("Atypical Appearance Distribution",font="Serif", fontsize=20,we
ight="bold")
```

```
fig.subplots_adjust(wspace=0.2, hspace=0.4, top=0.93)
plt.show()
def plot_wb_bar(df,col1,col2):
    run = wandb.init(project='siim', job_type='image-visualization',name=col1,con
fig = CONFIG)

    dt = [[label, val] for (label, val) in zip(df[col1], df[col2])]
    table = wandb.Table(data=dt, columns = [col1,col2])
    wandb.log({col1 : wandb.plot.bar(table, col1,col2,title=col1)})

    run.finish()

#====== Function to create a dataframe of value counts ======
def count_values(df,col):
    df = pd.DataFrame(df[col].value_counts().reset_index().values,columns=[col, "
counts"])
    return df
plot_wb_bar(count_values(train_df,"Negative for Pneumonia"),"Negative for Pneumon
ia", 'counts')
plot_wb_bar(count_values(train_df,"Typical Appearance"),"Typical Appearance", 'co
unts')
plot_wb_bar(count_values(train_df,"Indeterminate Appearance"),"Indeterminate Appe
arance", 'counts')
plot_wb_bar(count_values(train_df,"Atypical Appearance"),"Atypical Appearance", '
counts')
voi_lut=True
fix_monochrome=True

def dicom_dataset_to_dict(filename,func):
    """Credit: https://github.com/pydicom/pydicom/issues/319
               https://www.kaggle.com/raddar/convert-dicom-to-np-array-the-correc
t-way
    """

    dicom_header = dicom.dcmread(filename)

    #====== DICOM FILE DATA ======
    dicom_dict = {}
    repr(dicom_header)
    for dicom_value in dicom_header.values():
        if dicom_value.tag == (0x7fe0, 0x0010):
            #discard pixel data
            continue
        if type(dicom_value.value) == dicom.dataset.Dataset:
            dicom_dict[dicom_value.name] = dicom_dataset_to_dict(dicom_value.valu
e)
        else:
            v = _convert_value(dicom_value.value)
            dicom_dict[dicom_value.name] = v
    del dicom_dict['Pixel Representation']

    if func!='metadata_df':
        #====== DICOM IMAGE DATA ======
        # VOI LUT (if available by DICOM device) is used to transform raw DICOM d
ata to "human-friendly" view
        if voi_lut:
            data = apply_voi_lut(dicom_header.pixel_array, dicom_header)
        else:
            data = dicom_header.pixel_array
        # depending on this value, X-ray may look inverted - fix that:
        if fix_monochrome and dicom_header.PhotometricInterpretation == "MONOCHRO
ME1":
            data = np.amax(data) - data
        data = data - np.min(data)
        data = data / np.max(data)
        modified_image_data = (data * 255).astype(np.uint8)

        return dicom_dict, modified_image_data

    else:
return dicom_dict

def _sanitise_unicode(s):
    return s.replace(u"\u0000", "").strip()

def _convert_value(v):
    t = type(v)
    if t in (list, int, float):
        cv = v
    elif t == str:
        cv = _sanitise_unicode(v)
    elif t == bytes:
        s = v.decode('ascii', 'replace')
        cv = _sanitise_unicode(s)
    elif t == dicom.valuerep.DSfloat:
        cv = float(v)
    elif t == dicom.valuerep.IS:
        cv = int(v)
    else:
        cv = repr(v)
    return cv
for filename in train_df.path[0:5]:
    df, img_array = dicom_dataset_to_dict(filename, 'fetch_both_values')

    fig, ax = plt.subplots(1, 2, figsize=[15, 8])
    ax[0].imshow(img_array, cmap=plt.cm.gray)
    ax[1].imshow(img_array, cmap=plt.cm.plasma)
    plt.show()

    pprint.pprint(df)
run = wandb.init(project='siim', config = CONFIG)
artifact = run.use_artifact('ruchi798/siim/dicom_metadata:v1', type='dataset')
artifact_dir = artifact.download()
run.finish()
```

```
path = os.path.join(artifact_dir,"dicom_metadata.csv")
metadata = pd.read_csv(path)
metadata = metadata.drop(columns=["Unnamed: 0"])
metadata.head()
m = metadata.copy()
m = m.rename(columns={"Patient's Sex": "Patient Sex"})
cols_to_plot = ["Modality","Photometric Interpretation","Body Part Examined","Private
Creator","De-identification Method","Patient Sex"]

for col in cols_to_plot:
    plot_wb_bar(count_values(m,col),col, 'counts')
In [17]:
linkcode
# initializing the run
run = wandb.init(project="siim",
                 job_type="upload",
                 config = CONFIG
                 )

# creating an artifact
artifact = wandb.Artifact(name="dicom_metadata_image", type="raw_data")

# setting up a WandB Table object to hold the dataset
columns = ['image',"Body Part Examined","Image Type","Modality"," Name","Patient ID"
,"Patient's Sex","Study Instance UID"]
table = wandb.Table(
    columns=columns
)

for filename in train_df.path[0:5]:
    data_di, img_array = dicom_dataset_to_dict(filename,'fetch_both_values')

    body_part_examined = data_di.get("Body Part Examined")
    img_type = data_di.get('Image Type')
    modality = data_di.get("Modality")
    p_name = data_di.get("Patient's Name")
    p_id = data_di.get("Patient ID")
    p_gender = data_di.get("Patient's Sex")
    study_inst_uid = data_di.get("Study Instance UID")

    img_object = Image.fromarray(img_array)
raw_img = wandb.Image(img_object)

    # adding a row to the table
    row = [raw_img,body_part_examined,img_type,modality,p_name,p_id,p_gender,stud
y_inst_uid]
    table.add_data(*row)

# adding the table to the artifact
artifact.add(table, "dicom_examples")

# logging the artifact
```

```
run.log_artifact(artifact)

run.finish()
run = wandb.init(project="siim",
                 job_type="upload",
                 config = CONFIG
                 )

# creating an artifact
artifact = wandb.Artifact(name="dicom_images", type="raw_data")

# setting up a WandB Table object to hold the dataset
columns=["dicom image", "class"]

table = wandb.Table(
    columns=columns
)
classes = ['Negative for Pneumonia','Typical Appearance', 'Indeterminate Appearan
ce', 'Atypical Appearance']
for siim_class in classes:
    print(siim_class)
    for _, row in train_df[train_df[siim_class]==1].iloc[:2].iterrows():
        filename = row['path']
        df, img_array = dicom_dataset_to_dict(filename,'fetch_both_values')

        fig, ax = plt.subplots(1, 2, figsize=[15, 8])
        ax[0].imshow(img_array, cmap=plt.cm.gray)
        ax[1].imshow(img_array, cmap=plt.cm.plasma)
        plt.show()

        img_object = Image.fromarray(img_array)

        # raw image
        raw_img = wandb.Image(img_object)

        # adding a row to the table
        row = [raw_img,siim_class]
        table.add_data(*row)
# adding the table to the artifact
artifact.add(table, "raw_examples")

# logging the artifact
run.log_artifact(artifact)

run.finish()
train_jpg_directory = '../input/siim-covid19-resized-to-256px-jpg/train'
test_jpg_directory = '../input/siim-covid19-resized-to-256px-jpg/test'

def getImagePaths(path):
    image_names = []
    for dirname, _, filenames in os.walk(path):
        for filename in filenames:
            fullpath = os.path.join(dirname, filename)
            image_names.append(fullpath)
    return image_names

train_images_path = getImagePaths(train_jpg_directory)
test_images_path = getImagePaths(test_jpg_directory)

print(f"{y_}Number of train images: {g_} {len(train_images_path)}\n")
print(f"{y_}Number of test images: {g_} {len(test_images_path)}\n")

def getShape(data, images_paths):
    shape = cv2.imread(images_paths[0]).shape
    for image_path in images_paths:
image_shape=cv2.imread(image_path).shape
        if (image_shape!=shape):
            return data +" - Different image shape"
        else:
            return data +" - Same image shape " + str(shape)
run = wandb.init(project='siim', name='count',config = CONFIG)

wandb.log({'Training samples': len(train_images_path) ,
           'Test samples': len(test_images_path)
          })

run.finish()
getShape('train',train_images_path)
getShape('test',test_images_path)
def plot_augmentations(images, titles, sup_title):
    fig, axes = plt.subplots(figsize=(20, 16), nrows=3, ncols=4, squeeze=False)

    for indx, (img, title) in enumerate(zip(images, titles)):
        axes[indx // 4][indx % 4].imshow(img)
        axes[indx // 4][indx % 4].set_title(title, fontsize=15)

    plt.tight_layout()
    fig.suptitle(sup_title, fontsize = 20)
    fig.subplots_adjust(wspace=0.2, hspace=0.2, top=0.93)
    plt.show()
def augment(paths, data):

    # list of albumentations
    albumentations = [A.RandomSunFlare(p=1), A.RandomFog(p=1), A.RandomBrightness
(p=1),
                              A.RandomCrop(p=1,height = 128, width = 128), A.Rota
te(p=1, limit=90),
                              A.RGBShift(p=1), A.RandomSnow(p=1),
                              A.HorizontalFlip(p=1), A.VerticalFlip(p=1), A.Rando
mContrast(limit = 0.5,p = 1),
                              A.HueSaturationValue(p=1,hue_shift_limit=20, sat_sh
ift_limit=30, val_shift_limit=50)]

    # image titles
    titles = ["RandomSunFlare","RandomFog","RandomBrightness",
                       "RandomCrop","Rotate", "RGBShift", "RandomSnow","Horizonta
lFlip", "VerticalFlip", "RandomContrast","HSV"]

    for i in paths:
        image_path = i

        # getting image name from path
        image_name = image_path.split("/")[4].split(".")[0]
# reading image
        image = cv2.imread(image_path)

        # list of images
        images = []

        # creating image augmentations
        for augmentation_type in albumentations:
            augmented_img = augmentation_type(image = image)['image']
            images.append(augmented_img)
# original image
        titles.insert(0, "Original")
        images.insert(0,image)

        sup_title = "Image Augmentation for " + data + " - " + image_name
        plot_augmentations(images, titles, sup_title)

        titles.remove("Original")
augment(train_images_path[0:2],'train')
augment(train_images_path[0:2],'test')
```

6.2 CHEST X-RAY NET MODEL

```
import numpy as np
import pandas as pd
from tensorflow.keras.applications import DenseNet121
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import models
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint,
 EarlyStopping
```

```
import cv2
import os
from skimage import exposure
import matplotlib
matplotlib.rcParams.update({'font.size': 16})
import matplotlib.pyplot as plt
import warnings

import numpy as np
import pandas as pd
from tensorflow.keras.applications import DenseNet121
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import models
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint,
 EarlyStopping
import cv2
import os
from skimage import exposure
import matplotlib
matplotlib.rcParams.update({'font.size': 16})
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import tensorflow.keras.backend as K
import tensorflow as tf
from tensorflow.math import confusion_matrix
from sklearn.metrics import accuracy_score
from seaborn import heatmap
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from ast import literal_eval
from matplotlib.patches import Rectangle

df_image = pd.read_csv('../input/siim-covid19-
detection/train_image_level.csv')
df_study = pd.read_csv('../input/siim-covid19-
detection/train_study_level.csv')
df_study['id'] = df_study['id'].str.replace('_study',"")
df_study.rename({'id': 'StudyInstanceUID'},axis=1, inplace=True)
df_train = df_image.merge(df_study, on='StudyInstanceUID')
```

```
df_train.loc[df_train['Negative for Pneumonia']==1, 'study_label'] = 'nega
tive'
df_train.loc[df_train['Typical Appearance']==1, 'study_label'] = 'typical'
df_train.loc[df_train['Indeterminate Appearance']==1, 'study_label'] = 'in
determinate'
df_train.loc[df_train['Atypical Appearance']==1, 'study_label'] = 'atypica
l'
df_train.drop(['Negative for Pneumonia','Typical Appearance', 'Indetermina
te Appearance', 'Atypical Appearance'], axis=1, inplace=True)
df_train['id'] = df_train['id'].str.replace('_image', '.jpg')
df_train['image_label'] = df_train['label'].str.split().apply(lambda x : x
[0])
df_size = pd.read_csv('../input/covid-jpg-512/size.csv')
df_train = df_train.merge(df_size, on='id')
df_train.head(3)


train_dir = '../input/covid-jpg-512/train'

def preprocess_image(img):
    equ_img = exposure.equalize_adapthist(img/255, clip_limit=0.05, kernel
_size=24)
    return equ_img

df_opa = df_train[df_train['image_label']=='opacity'].reset_index()
fig, axs = plt.subplots(5, 2, figsize=(10,20))
fig.subplots_adjust(hspace=.2, wspace=.2)
n=5
for i in range(n):
    img = cv2.imread(os.path.join(train_dir, df_opa['id'][i]))
    img_proc = preprocess_image(img)
    axs[i, 0].imshow(img)
    axs[i, 1].imshow(img_proc)
    axs[i, 0].axis('off')
    axs[i, 1].axis('off')
    boxes = literal_eval(df_opa['boxes'][i])
    for box in boxes:
        axs[i, 0].add_patch(Rectangle((box['x']*(512/df_opa['dim1'][i]), b
ox['y']*(512/df_opa['dim0'][i])), box['width']*(512/df_opa['dim1'][i]), bo
x['height']*(512/df_opa['dim0'][i]), fill=0, color='y', linewidth=3))
        axs[i, 0].set_title(df_opa['study_label'][i])
        axs[i, 1].add_patch(Rectangle((box['x']*(512/df_opa['dim1'][i]), b
ox['y']*(512/df_opa['dim0'][i])), box['width']*(512/df_opa['dim1'][i]), bo
x['height']*(512/df_opa['dim0'][i]), fill=0, color='r', linewidth=3))
        axs[i, 1].set_title('After CLAHE')
```

```
plt.show()

img_size = 224
batch_size = 16

image_generator = ImageDataGenerator(
        validation_split=0.2,
        horizontal_flip = True,
        zoom_range = 0.15,
        brightness_range = [0.8, 1.2],
        fill_mode='nearest',
        preprocessing_function=preprocess_image
)

image_generator_valid = ImageDataGenerator(validation_split=0.2,preprocess
ing_function=preprocess_image)

train_generator = image_generator.flow_from_dataframe(
        dataframe = df_train,
        directory='../input/covid-jpg-512/train',
        x_col = 'id',
        y_col =  'image_label',
        target_size=(img_size, img_size),
        batch_size=batch_size,
        subset='training', seed = 23)

valid_generator=image_generator_valid.flow_from_dataframe(
    dataframe = df_train,
    directory='../input/covid-jpg-512/train',
    x_col = 'id',
    y_col = 'image_label',
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='validation', shuffle=False, seed=23)

for j in range(4):
    aug_images = [train_generator[0][0][j] for i in range(5)]
    fig, axes = plt.subplots(1, 5, figsize=(24,24))
    axes = axes.flatten()
    for img, ax in zip(aug_images, axes):
        ax.imshow(img)
        ax.axis('off')
plt.tight_layout()
plt.show()
```

```
chex_weights_path = '../input/chexnet-
weights/brucechou1983_CheXNet_Keras_0.3.0_weights.h5'

pre_model = DenseNet121(weights=None,
                                include_top=False,
                                input_shape=(img_size,img_size,3)
                               )
out = Dense(14, activation='sigmoid')(pre_model.output)
pre_model = Model(inputs=pre_model.input, outputs=out)
pre_model.load_weights(chex_weights_path)
pre_model.trainable = False
x = pre_model.layers[-2].output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.1)(x)
output = Dense(2, activation='softmax')(x)
model = Model(pre_model.input, output)


model.compile(Adam(lr=1e-3),loss='binary_crossentropy',metrics='accuracy')


rlr = ReduceLROnPlateau(monitor = 'val_acc', factor = 0.2, patience = 2, v
erbose = 1,
                                min_delta = 1e-4, min_lr = 1e-
4, mode = 'max')
es = EarlyStopping(monitor = 'val_acc', min_delta = 1e-
4, patience = 5, mode = 'max',
                    restore_best_weights = True, verbose = 1)

ckp = ModelCheckpoint('model.h5',monitor = 'val_acc',
                      verbose = 0, save_best_only = True, mode = 'max')

history = model.fit(
      train_generator,
      epochs=20,
      validation_data=valid_generator,
      callbacks=[es,rlr, ckp],
      verbose=1)


pre_model.trainable = True

model.compile(Adam(lr=1e-5),loss='binary_crossentropy',metrics='accuracy')
```

```
rlr2 = ReduceLROnPlateau(monitor = 'val_acc', factor = 0.1, patience = 3, 
verbose = 1,
                                min_delta = 1e-4, min_lr = 1e-
7, mode = 'max')

es2 = EarlyStopping(monitor = 'val_acc', min_delta = 1e-
4, patience = 7, mode = 'max', 
                    restore_best_weights = True, verbose = 1)

history2 = model.fit(
      train_generator,
      epochs=30,
      validation_data=valid_generator,
      callbacks=[es2,rlr2, ckp],
      verbose=1)

K.clear_session()


actual =  valid_generator.labels
preds = np.argmax(model.predict(valid_generator), axis=1)
cfmx = confusion_matrix(actual, preds)
acc = accuracy_score(actual, preds)


print ('Test Accuracy:', acc )
heatmap(cfmx, annot=True, cmap='plasma',
        xticklabels=['Normal','Opacity'],fmt='.0f', yticklabels=['Normal',
 'Opacity'])
plt.show()




hist = pd.DataFrame(history.history)
fig, (ax1, ax2) = plt.subplots(figsize=(12,12),nrows=2, ncols=1)
hist['loss'].plot(ax=ax1,c='k',label='training loss')
hist['val_loss'].plot(ax=ax1,c='r',linestyle='--
', label='validation loss')
ax1.legend()
hist['accuracy'].plot(ax=ax2,c='k',label='training accuracy')
```

```
hist['val_accuracy'].plot(ax=ax2,c='r',linestyle='--
',label='validation accuracy')
ax2.legend()
plt.show()


hist = pd.DataFrame(history2.history)
fig, (ax1, ax2) = plt.subplots(figsize=(12,12),nrows=2, ncols=1)
hist['loss'].plot(ax=ax1,c='k',label='training loss')
hist['val_loss'].plot(ax=ax1,c='r',linestyle='--
', label='validation loss')
ax1.legend()
hist['accuracy'].plot(ax=ax2,c='k',label='training accuracy')
hist['val_accuracy'].plot(ax=ax2,c='r',linestyle='--
',label='validation accuracy')
ax2.legend()
plt.show()


def grad_cam(input_image, model, layer_name):

    desired_layer = model.get_layer(layer_name)
    grad_model = Model(model.inputs, [desired_layer.output, model.output])

    with tf.GradientTape() as tape:
        layer_output, preds = grad_model(input_image)
        ix = (np.argsort(preds, axis=1)[:, -1]).item()
        output_idx = preds[:, ix]

    gradient = tape.gradient(output_idx, layer_output)
    alpha_kc = np.mean(gradient, axis=(0,1,2))
    L_gradCam = tf.nn.relu(np.dot(layer_output, alpha_kc)[0])
    L_gradCam = (L_gradCam - np.min(L_gradCam)) / (np.max(L_gradCam) - np.
min(L_gradCam))
    return L_gradCam.numpy()




def blend(img_path, gradCam_img, alpha, colormap = cv2.COLORMAP_JET):
    origin_img = img_to_array(load_img(img_path))
    gradCam_resized = cv2.resize(gradCam_img, (origin_img.shape[1], origin
_img.shape[0]), interpolation = cv2.INTER_LINEAR)
```

```
    heatmap  = cv2.applyColorMap(np.uint8(gradCam_resized*255), colormap)
    superimposed_image = cv2.cvtColor(origin_img.astype('uint8'), cv2.COLO
R_RGB2BGR) + heatmap * alpha
    return heatmap, superimposed_image


def plot_results(model, gen, label=0):
    n = 50
    fig, axs = plt.subplots(10, 5, figsize=(20,60))
    fig.subplots_adjust(hspace=.5, wspace=.1)
    axs = axs.ravel()
    gen.next()
    classes = list(gen.class_indices.keys())
    if label==0:
        idx = np.array(np.where(np.array(gen.labels) ==0)).ravel()
    else:
        idx = np.array(np.where(np.array(gen.labels) ==1)).ravel()

    layer_name = 'relu'
    for i in range(n):
        sample_img_path = os.path.join(train_dir, df_train['id'][idx[i]])
        img = load_process(sample_img_path, img_size)
        pred = model.predict(img)
        grad_cam_img = grad_cam(img, model, layer_name)
        heatmap_img, result_img = blend(sample_img_path, grad_cam_img, 0.5
)
        axs[i].imshow(result_img[:,:,::-1]/255)
        axs[i].set_xticklabels([])
        axs[i].set_yticklabels([])
        if type(df_train['boxes'][idx[i]])==str:
            boxes = literal_eval(df_train['boxes'][idx[i]])
            for box in boxes:
                axs[i].add_patch(Rectangle((box['x']*(512/df_train['dim1']
[idx[i]]), box['y']*(512/df_train['dim0'][idx[i]])), box['width']*(512/df_
train['dim1'][idx[i]]), box['height']*(512/df_train['dim0'][idx[i]]), fill
=0, color='y', linewidth=2))
                axs[i].set_title(f"{df_train['study_label'][idx[i]]}, {df_
train['image_label'][idx[i]]}")
        else:
            axs[i].set_title(df_train['study_label'][idx[i]])

        axs[i].set_xlabel(f"{classes[np.argmax(pred)]}, {round(pred[0][np.
argmax(pred)]*100, 2)}%")
plot_results(model, valid_generator,label=0)
plot_results(model, valid_generator,label=1)
```

6.3 U-NET MODEL

```
import numpy as np
import pandas as pd

DIR_PATH = "/kaggle/input/covid-segmentation/"
TRAIN_X_FILE = "images_medseg.npy"
TRAIN_Y_FILE = "masks_medseg.npy"

imgs_medseg = np.load(DIR_PATH+TRAIN_X_FILE).astype(np.float32)
msks_medseg = np.load(DIR_PATH+TRAIN_Y_FILE).astype(np.float32)

import matplotlib.pyplot as plt
import numpy as np

plt.style.use("dark_background")

def visualize(image_batch, mask_batch=None, pred_batch=None, num_samples=8
):
    num_classes = mask_batch.shape[-1] if mask_batch is not None else 0
    fix, ax = plt.subplots(num_classes + 1, num_samples, figsize=(num_samp
les * 2, (num_classes + 1) * 2))
    for i in range(num_samples):
        ax_image = ax[0, i] if num_classes > 0 else ax[i]
        ax_image.imshow(image_batch[i,:,:,0], cmap="Greys")
        ax_image.set_xticks([])
        ax_image.set_yticks([])

        if mask_batch is not None:
            for j in range(num_classes):
                if pred_batch is None:
                    mask_to_show = mask_batch[i,:,:,j]
                else:
                    mask_to_show = np.zeros(shape=(*mask_batch.shape[1:-
1], 3))
                    mask_to_show[..., 0] = pred_batch[i,:,:,j] > 0.5
                    mask_to_show[..., 1] = mask_batch[i,:,:,j]
                ax[j + 1, i].imshow(mask_to_show, vmin=0, vmax=1)
                ax[j + 1, i].set_xticks([])
                ax[j + 1, i].set_yticks([])

    plt.tight_layout()
    plt.show()

visualize(imgs_medseg, msks_medseg)
def plot_hists(images):
    plt.hist(images.ravel(), bins=100, density=True, color='b', alpha=1)
plot_hists(imgs_medseg)

def normalize_img(img):
    img = img.astype("float32")
    img[img > 500] = 500
    img[img < -1500] = -1500
    min_perc, max_perc = np.percentile(img, 5), np.percentile(img, 95)
    img_valid = img[(img > min_perc) & (img < max_perc)]
    mean, std = img_valid.mean(), img_valid.std()
    img = (img-mean)/std
    return img
imgs_medseg = normalize_img(imgs_medseg)
plot_hists(imgs_medseg)

from sklearn.model_selection import train_test_split
train_x, val_x, train_y, val_y = train_test_split(imgs_medseg, msks_medseg
, test_size=0.1, random_state=42)

import tensorflow as tf
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation
, MaxPool2D, Conv2DTranspose, Concatenate, Input
from tensorflow.keras.models import Model

def convolution(input, num_filters):
  x = Conv2D(num_filters, 3, padding="same")(input)
  x = BatchNormalization()(x)
  x = Activation("relu")(x)

  x = Conv2D(num_filters, 3, padding="same")(x)
  x = BatchNormalization()(x)
  x = Activation("relu")(x)
  return x

def downsample(input, num_filters):
  x = convolution(input, num_filters)
  p = MaxPool2D((2,2))(x)
  return x, p

def upsample(input, skip_connections, num_filters):
  x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(inpu
t)
  x = Concatenate()([x, skip_connections])
  x = convolution(x, num_filters)
  return x

def build_unet(input_shape):
  inputs = Input(input_shape)

  sc1, p1 = downsample(inputs, 64)
  sc2, p2 = downsample(p1, 128)
  sc3, p3 = downsample(p2, 256)
  sc4, p4 = downsample(p3, 512)

  b1 = convolution(p4, 1024)

  d1 = upsample(b1, sc4, 512)
  d2 = upsample(d1, sc3, 256)
  d3 = upsample(d2, sc2, 128)
  d4 = upsample(d3, sc1, 64)

  outputs = Conv2D(4,(1,1),padding="same",activation="softmax")(d4)

  model = Model(inputs, outputs, name="U-Net")
  return model
def iou(true_y, pred_y):
    def f(true_y, pred_y):
        intersection = (true_y * pred_y).sum()
        union = true_y.sum() + pred_y.sum() - intersection
        x = (intersection)/(union)
        x = x.astype(np.float32)
        return x
    return tf.numpy_function(f, [true_y, pred_y], tf.float32)

unet = build_unet(imgs_medseg.shape[1:])
unet.compile(loss="binary_crossentropy", optimizer="adam", metrics=["acc",
 iou])
results = unet.fit(train_x, train_y, validation_data=(val_x, val_y), epoch
s=100, batch_size=1, verbose=1)

plt.figure(0)
plt.plot(results.history["acc"])
plt.plot(results.history["val_acc"])
plt.title("Training vs Validation Accuracy", color="white")
plt.xlabel("epoch", color="white")
plt.ylabel("accuracy", color="white")
plt.legend(["train", "val"])
plt.show()

plt.figure(1)
plt.plot(results.history["iou"])
plt.plot(results.history["val_iou"])
plt.title("Training vs Validation IoU", color="white")

plt.xlabel("epoch", color="white")
plt.ylabel("IoU", color="white")
plt.legend(["train", "val"])
plt.show()


pred_y = unet.predict(val_x)
plot_hists(pred_y)

def filter_pixels(img):
    for i in range(img.shape[0]):
        for j in range(img.shape[3]):
            for k in range(img.shape[1]):
                for l in range(img.shape[2]):
                    img[i,k,l,j] = 1 if (img[i,k,l,j] > 0.5) else 0;
    return img

pred_y = filter_pixels(pred_y)
plot_hists(pred_y)

print(pred_y.shape)
for i in range(0, 10):
    plt.figure(i)
    fig, ax = plt.subplots(1,2, figsize=(10,10))
    ax[0].imshow(pred_y[i,:,:,0])
    ax[0].title.set_text("Predicted Ground Class")
    ax[1].imshow(val_y[i,:,:,0])
    ax[1].title.set_text("Actual Ground Class")
```

6.4 PSP NET MODEL

```
!pip uninstall keras -y
!pip install git+https://github.com/qubvel/segmentation_models
!git clone https://github.com/SlinkoIgor/ImageDataAugmentor.git
```

```
#add your input file path here
images_radiopedia = np.load('/content/rdrive/MyDrive/covid-
segmentation/images_radiopedia.npy').astype(np.float32)
masks_radiopedia = np.load('/content/rdrive/MyDrive/covid-
segmentation/masks_radiopedia.npy').astype(np.int8)
images_medseg = np.load('/content/rdrive/MyDrive/covid-
segmentation/images_medseg.npy').astype(np.float32)
masks_medseg = np.load('/content/rdrive/MyDrive/covid-
segmentation/masks_medseg.npy').astype(np.int8)

test_images_medseg = np.load('/content/rdrive/MyDrive/covid-
segmentation/test_images_medseg.npy').astype(np.float32)


import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import numpy as np


def visualize(image_batch, mask_batch=None, pred_batch=None, num_samples=8
):
    num_classes = mask_batch.shape[-1] if mask_batch is not None else 0
    fix, ax = plt.subplots(num_classes + 1, num_samples, figsize=(num_samp
les * 2, (num_classes + 1) * 2))

    for i in range(num_samples):
        ax_image = ax[0, i] if num_classes > 0 else ax[i]
        ax_image.imshow(image_batch[i,:,:,0], cmap='Greys')
        ax_image.set_xticks([])
        ax_image.set_yticks([])

    if mask_batch is not None:
            for j in range(num_classes):
                if pred_batch is None:
                    mask_to_show = mask_batch[i,:,:,j]
                else:
                    mask_to_show = np.zeros(shape=(*mask_batch.shape[1:-
1], 3))
                    mask_to_show[..., 0] = pred_batch[i,:,:,j] > 0.5
                    mask_to_show[..., 1] = mask_batch[i,:,:,j]
                ax[j + 1, i].imshow(mask_to_show, vmin=0, vmax=1)
                ax[j + 1, i].set_xticks([])
                ax[j + 1, i].set_yticks([])
```

```
plt.tight_layout()
plt.show()

visualize(images_medseg, masks_medseg)

visualize(test_images_medseg)


def plot_hists(images1, images2=None):
    plt.hist(images1.ravel(), bins=100, density=True, color='b', alpha=1 i
f images2 is None else 0.5)
    if images2 is not None:
        plt.hist(images2.ravel(), bins=100, density=True, alpha=0.5, color
='orange')
    plt.show();



plot_hists(images_radiopedia, images_medseg)

plot_hists(test_images_medseg, images_medseg)

def preprocess_images(images_arr, mean_std=None):
    images_arr[images_arr > 500] = 500
    images_arr[images_arr < -1500] = -1500
    min_perc, max_perc = np.percentile(images_arr, 5), np.percentile(image
s_arr, 95)
    images_arr_valid = images_arr[(images_arr > min_perc) & (images_arr <
max_perc)]
    mean, std = (images_arr_valid.mean(), images_arr_valid.std()) if mean_
std is None else mean_std
    images_arr = (images_arr - mean) / std
    print(f'mean {mean}, std {std}')
    return images_arr, (mean, std)

images_radiopedia, mean_std = preprocess_images(images_radiopedia)
images_medseg, _ = preprocess_images(images_medseg, mean_std)
test_images_medseg, _ = preprocess_images(test_images_medseg, mean_std)

plot_hists(images_radiopedia, images_medseg)

plot_hists(test_images_medseg, images_medseg)
```

```
val_indexes, train_indexes = list(range(24)), list(range(24, 100))

train_images = np.concatenate((images_medseg[train_indexes], images_radiop
edia))
train_masks = np.concatenate((masks_medseg[train_indexes], masks_radiopedi
a))
val_images = images_medseg[val_indexes]
val_masks = masks_medseg[val_indexes]

batch_size = len(val_masks)

del images_radiopedia
del masks_radiopedia
del images_medseg
del masks_medseg

import tensorflow

import albumentations
import cv2

#for PSPNet
SOURCE_SIZE = 510
TARGET_SIZE = 384

train_augs = albumentations.Compose([
    albumentations.Rotate(limit=360, p=0.9, border_mode=cv2.BORDER_REPLICA
TE),
    albumentations.RandomSizedCrop((int(SOURCE_SIZE * 0.75), SOURCE_SIZE),

                                   TARGET_SIZE,
                                   TARGET_SIZE,
                                   interpolation=cv2.INTER_NEAREST),
    albumentations.HorizontalFlip(p=0.5),

])

val_augs = albumentations.Compose([
    albumentations.Resize(TARGET_SIZE, TARGET_SIZE, interpolation=cv2.INTE
R_NEAREST)
])


class Dataset:
    def __init__(
            self,
            images,
            masks,
            augmentations=None
    ):
        self.images = images
        self.masks = masks
        self.augmentations = augmentations

    def __getitem__(self, i):
        image = self.images[i]
        mask = self.masks[i]

        if self.augmentations:
            sample = self.augmentations(image=image, mask=mask)
            image, mask = sample['image'], sample['mask']
        return image, mask

    def __len__(self):
        return len(self.images)


class Dataloder(tensorflow.keras.utils.Sequence):
    """Load data from dataset and form batches

    Args:
        dataset: instance of Dataset class for image loading and preproces
sing.
        batch_size: Integet number of images in batch.
        shuffle: Boolean, if `True` shuffle image indexes each epoch.
    """

    def __init__(self, dataset, batch_size=1, shuffle=False):
        self.dataset = dataset
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.indexes = np.arange(len(dataset))
        self.on_epoch_end()

    def __getitem__(self, i):

        # collect batch data
        start = i * self.batch_size
        stop = (i + 1) * self.batch_size
        images = []
        masks = []
        for j in range(start, stop):
            image, mask = self.dataset[self.indexes[j]]
            images.append(image)
            masks.append(mask)

        images = np.stack(images, axis=0)
        masks = np.stack(masks, axis=0).astype(np.float32)

        return (images, masks)


def __len__(self):
        """Denotes the number of batches per epoch"""
        return len(self.indexes) // self.batch_size

def on_epoch_end(self):
        """Callback function to shuffle indexes each epoch"""
        if self.shuffle:
            self.indexes = np.random.permutation(self.indexes)

train_dataset = Dataset(train_images, train_masks, train_augs)
val_dataset = Dataset(val_images, val_masks, val_augs)

train_dataloader = Dataloder(train_dataset, batch_size=batch_size, shuffle
=True)
val_dataloader = Dataloder(val_dataset, batch_size=batch_size, shuffle=Fal
se)


assert train_dataloader[0][0].shape == (batch_size, TARGET_SIZE, TARGET_SI
ZE, 1)
assert train_dataloader[0][1].shape == (batch_size, TARGET_SIZE, TARGET_SI
ZE, 4)


visualize(*next(iter(train_dataloader)))
visualize(*next(iter(val_dataloader)))


def fscore_glass(y_true, y_pred):
    return sm.metrics.f1_score(y_true[..., 0:1],
                               y_pred[..., 0:1])

def fscore_consolidation(y_true, y_pred):
    return sm.metrics.f1_score(y_true[..., 1:2],
                               y_pred[..., 1:2])

def fscore_lungs_other(y_true, y_pred):
    return sm.metrics.f1_score(y_true[..., 2:3],
                               y_pred[..., 2:3])

def fscore_glass_and_consolidation(y_true, y_pred):
    return sm.metrics.f1_score(y_true[..., :2],
                               y_pred[..., :2])



from segmentation_models import PSPNet
import segmentation_models as sm

from tensorflow.keras.layers import Input, Conv2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
np.random.seed(0)

base_model = PSPNet(backbone_name='efficientnetb0',
                  encoder_weights='imagenet',
                  classes=4,
                  activation='softmax')

model = Sequential([Input(shape=(TARGET_SIZE, TARGET_SIZE, 1)),
                    Conv2D(3, (1, 1)),  # map N channels data to 3 channel
                    base_model])

model.compile(Adam(learning_rate=0.001, amsgrad=True),
              loss=sm.losses.categorical_crossentropy,
              metrics=[fscore_glass, fscore_consolidation, fscore_lungs_ot
her, fscore_glass_and_consolidation])


del train_images
del train_masks


model = tensorflow.keras.models.load_model('best_model/',
                                           compile=False,
                                           custom_objects={
                                                'categorical_crossentropy'
: sm.losses.categorical_crossentropy,
                                                'fscore_consolidation': fs
core_consolidation,
                                                'fscore_glass': fscore_gla
ss,
                                                'fscore_lungs_other': fsco
re_lungs_other,
                                                'fscore_glass_and_consolid
ation': fscore_glass_and_consolidation})

model.compile(Adam(learning_rate=0.001, amsgrad=True),
              loss=sm.losses.jaccard_loss)



input = val_dataloader[0]
image_batch, mask_batch = input

preds = model.predict_on_batch(image_batch)
visualize(image_batch, mask_batch, pred_batch=preds)

# yellow is TP, red is FP, green is FN.
axis=0
test_preds = model.predict_on_batch(image_batch)
test_masks_prediction = test_preds > 0.5
visualize(image_batch, test_masks_prediction, num_samples=len(test_images_
medseg)
import scipy
test_masks_prediction_original_size = scipy.ndimage.zoom(test_masks_predic
tion[..., :-2], (1, 2, 2, 1), order=0)
test_masks_prediction_original_size.shape
```

7. SOFTWARE TESTING

7.1 TESTING STRATEGIES

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development are:

7.1.1 Unit Testing:

Unit Testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements. Each module can be tested using the following two Strategies:

- Black Box Testing:

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been uses to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structure or external database access
- Performance errors
- Initialization and termination errors.

In this testing only the output is checked for correctness. The logical flow of the data is not checked.

- White Box testing:

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been uses to generate the test cases in the following cases:

- Guarantee that all independent paths have been executed.
- Execute all logical decisions on their true and false Sides.
- Execute all loops at their boundaries and within their operational bounds
- Execute internal data structures to ensure their validity

7.1.2 Integration Testing:

Integration testing ensures that software and subsystems work together a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together.

7.1.3 System Testing:

Involves in-house testing of the entire system before delivery to the user. Its aim is to satisfy the user the system meets all requirements of the client's specifications.

7.1.4 Acceptance Testing:

It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors.

7.2 Validation and Verification:

The system has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirements specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed. In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfil its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle.

Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between Verification: Are we building the product right?

- *Validation*: Are we building the right product? According to the Capability Maturity Model (CMMI-SW v1.1)
- *Software Verification*: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase [IEEE-STD-610].
- *Software Validation*: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements [IEEE-STD-610].

In other words, software verification is ensuring that the product has been built according to the requirements and design specifications, while software validation ensures that the product actually meets the user's needs, and that the specifications were correct in the first place. Software verification ensures that "you built it right" Software validation ensures that "you built the right thing". Software validation confirms that the product, as provided, will fulfil its intended use. From testing perspective:

- *Fault* – wrong or missing function in the code.
- *Failure* – the manifestation of a fault during execution.
- *Malfunction* – according to its specification the system does not meet its specified functionality.

8. CONCLUSION

8.1 Conclusion and Future Enhancements

In this study, we hypothesized that computer-aided deep learning algorithms can accurately predict infection severity on CXRs and CT scan associated with COVID-19 against expert chest radiologist ground level check-up, and the results of the study support this hypothesis. Results from the stratified Monte Carlo cross-validation experiments showed that the learned Covid I-Net deep neural networks could achieve mean R2 between predicted scores and radiologist scores for geographic extent and opacity extent greater than 0.5 when evaluated for 100 different subsets of CXR and CT data. Severity scoring for COVID-19 has gained recent attention due to the rise and continued spread of the COVID19 across the globe, and the need to find the severity of an infected patient is crucial for determining the best course of action regarding treatment and care. The researchers introduced a scoring scheme for severity analysis of COVID19 by adapting and simplifying the Radiographic Assessment of lungs, where each lung was divided into three zones (a total of six zones for a human) and each zone was assigned a binary score based on opacity, with the final severity score being the collection of the scores from the different zones.

In this paper, the performance of various deep learning networks (U-NET, and PSP Net) was compared in their ability to detect diseased areas in medical images of the lungs of COVID-19 patients. The results demonstrated the ability of the CXR-Net network to distinguish between infected and healthy tissues in these images. A comparison of these two networks was also performed in multiple infected areas in lung images. The results showed the U-net network's ability to distinguish between these areas outperforms PSP Net. The results obtained in this paper represent promising prospects for the possibility of using deep learning to assist in an objective diagnosis of COVID-19 disease through CT and CXR images of the lung.

APPENDICES

A.1 SAMPLE SCREENS

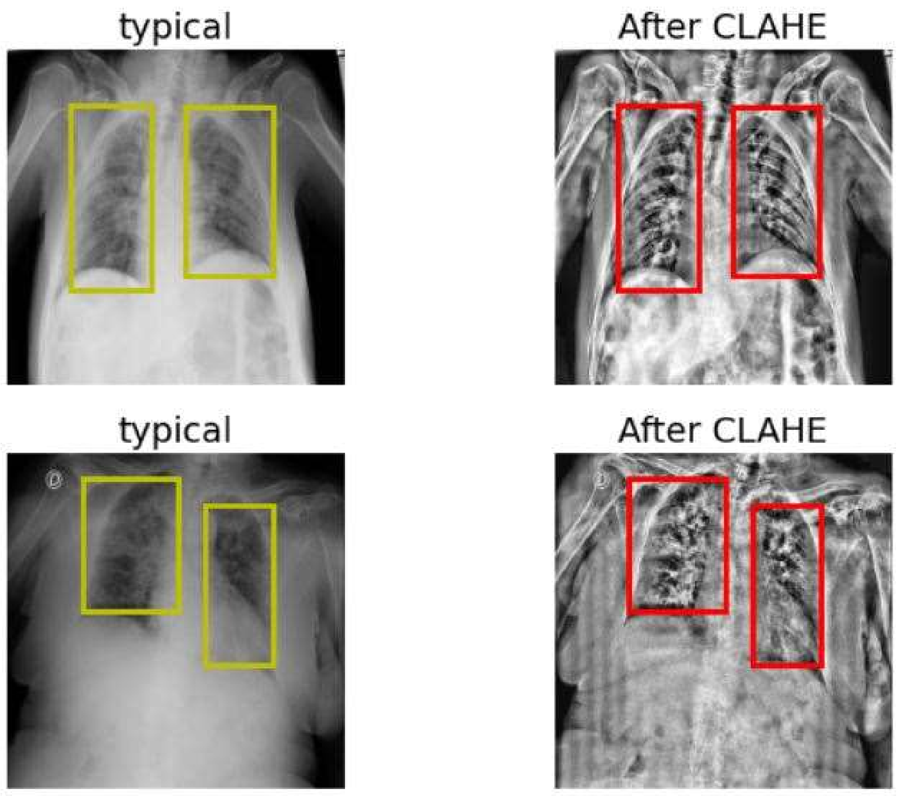


Figure 7 – CXR NET IMAGE PREPROCESSING

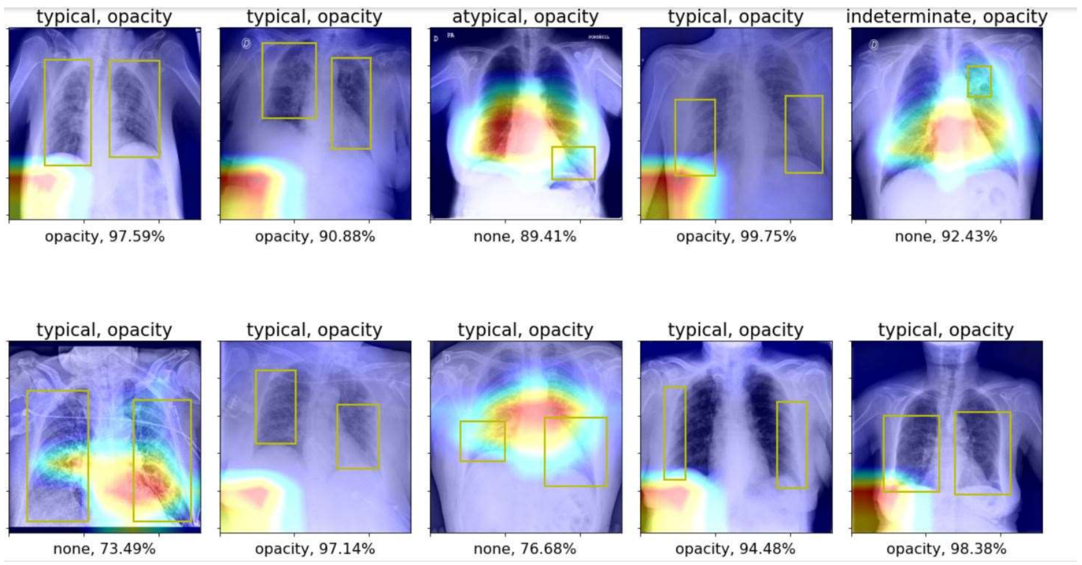


Figure 8 – CHEST X-RAY NET RESULTS

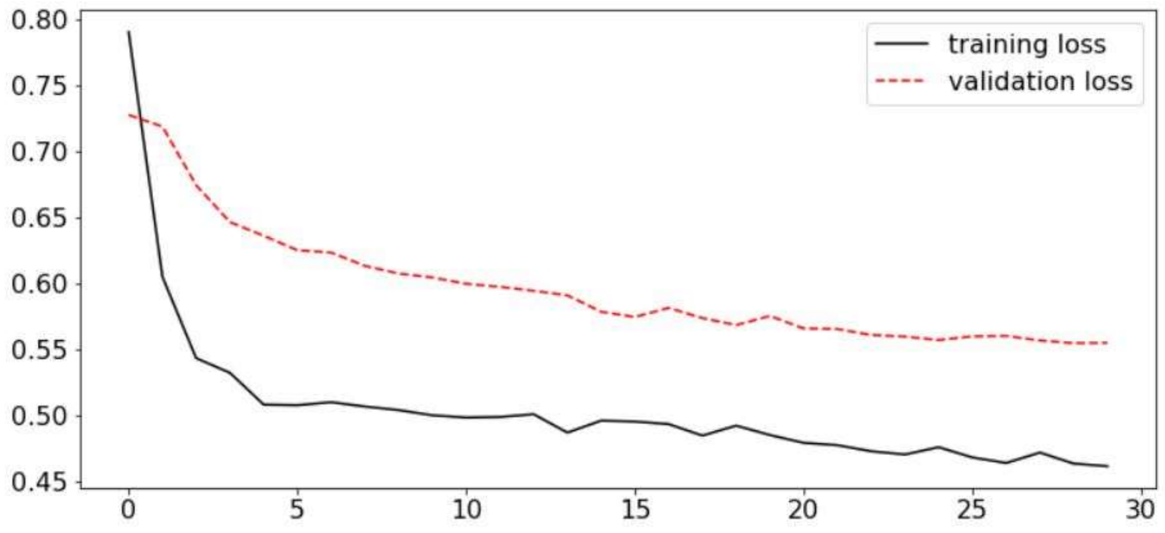


Figure 9 –CXR TRAINING ANALYTICS

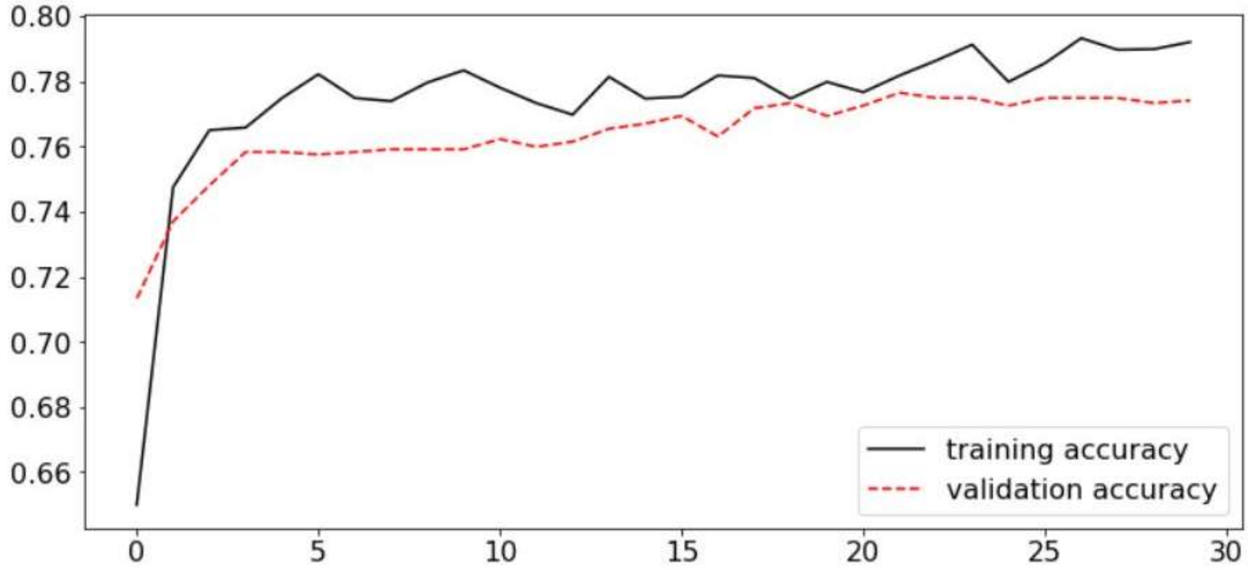


Figure 10 – CXR ACCURACY ANALYTICS

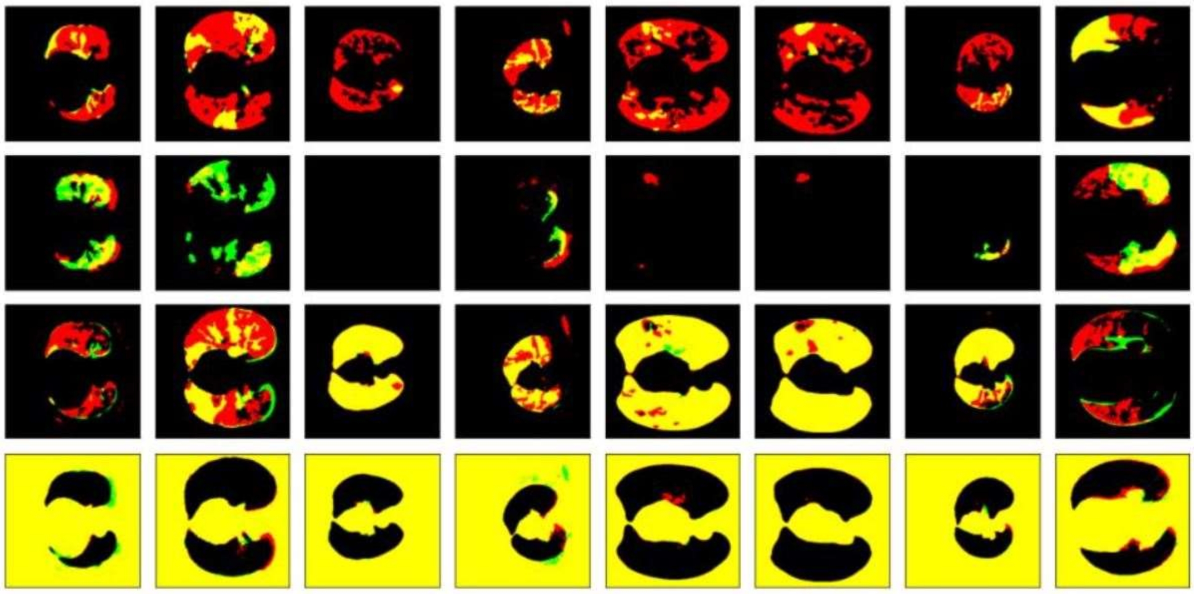


Figure *11* – U NET SEGMENTATION RESULTS

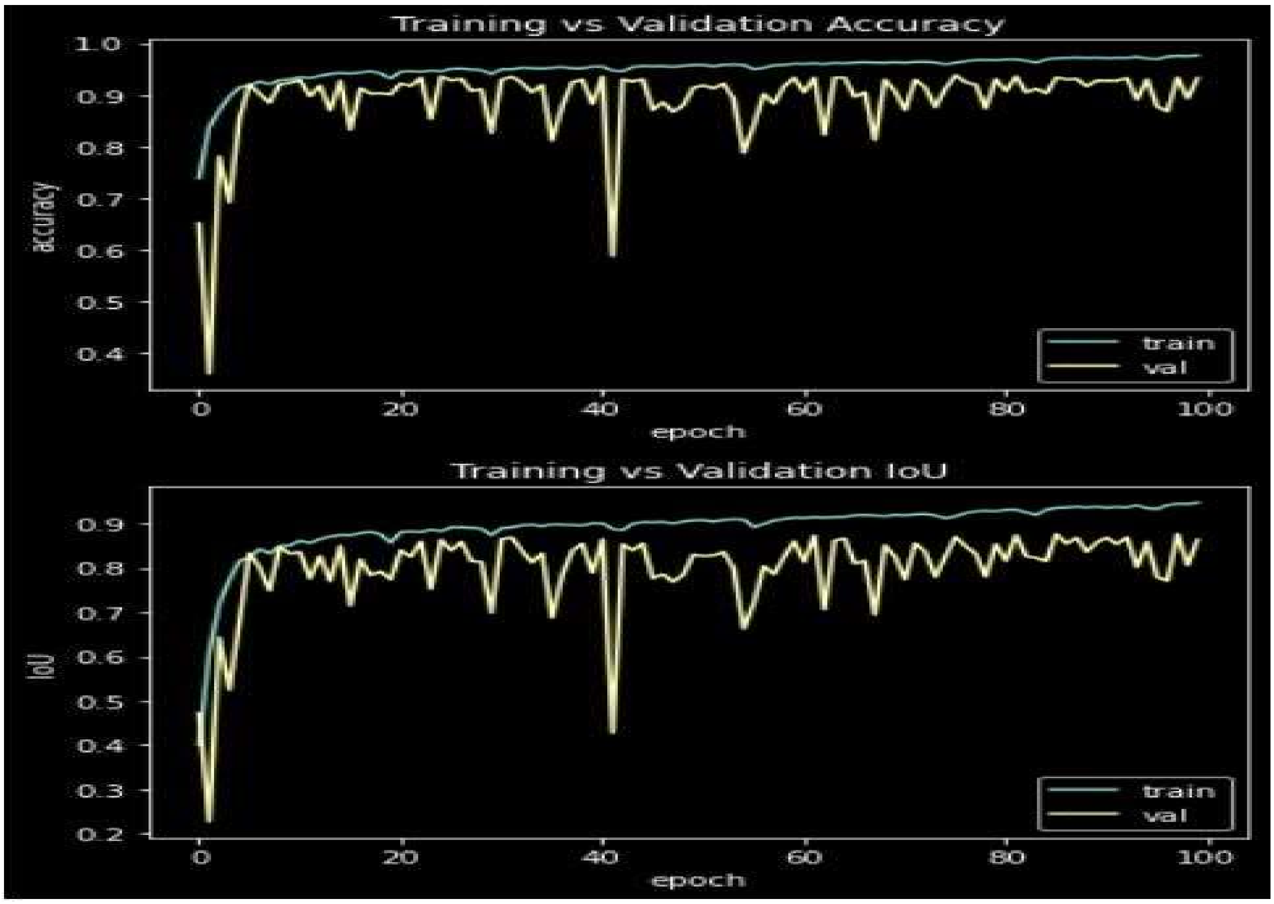


Figure *12* - PERFORMANCE ANALYSIS

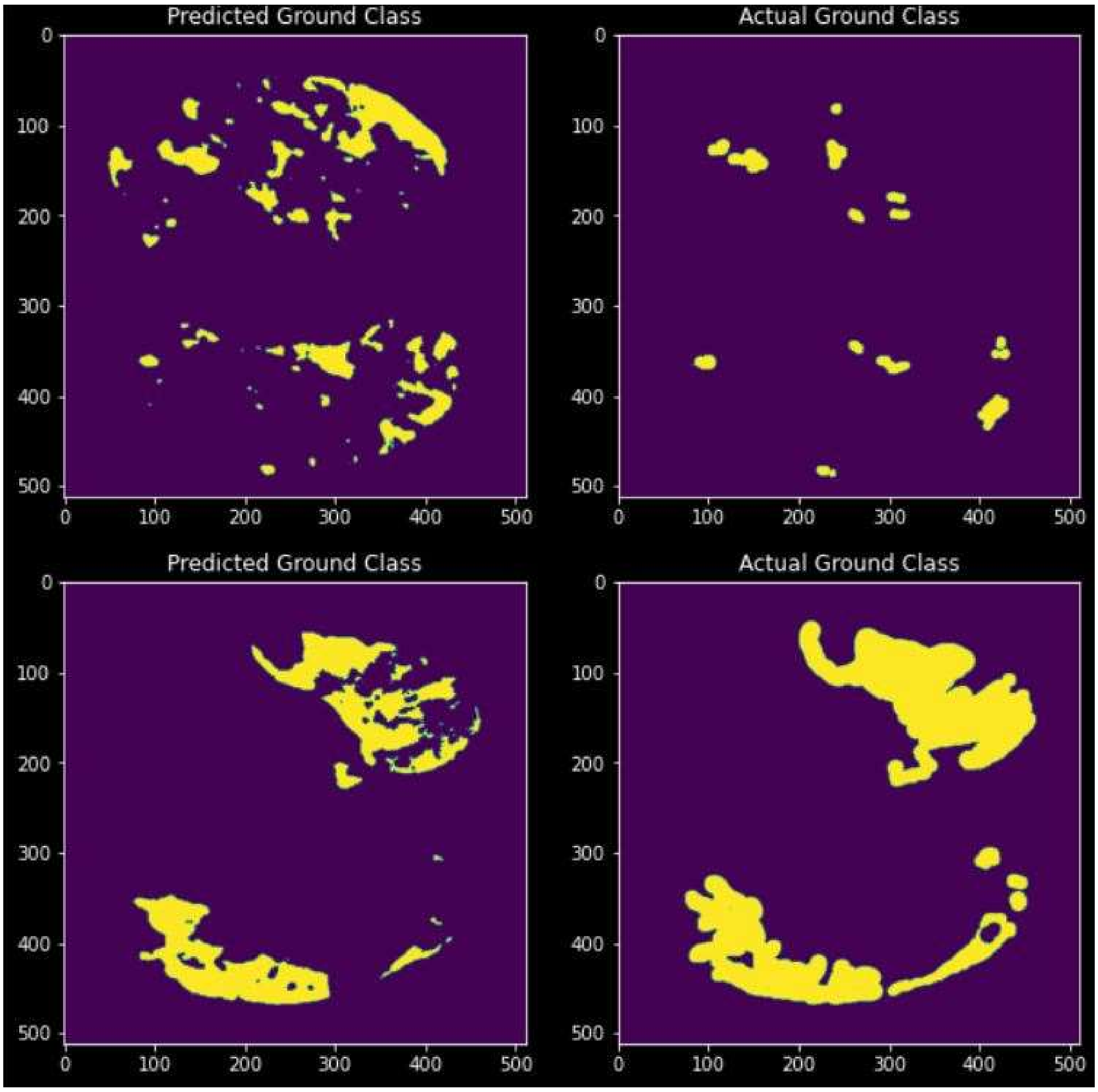


Figure *13* – UNET CT EVALUATION ANALYSIS

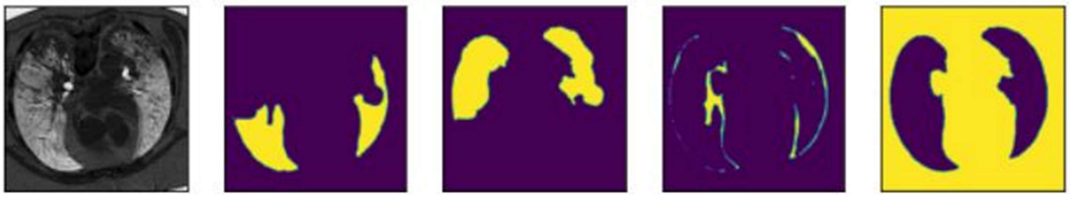
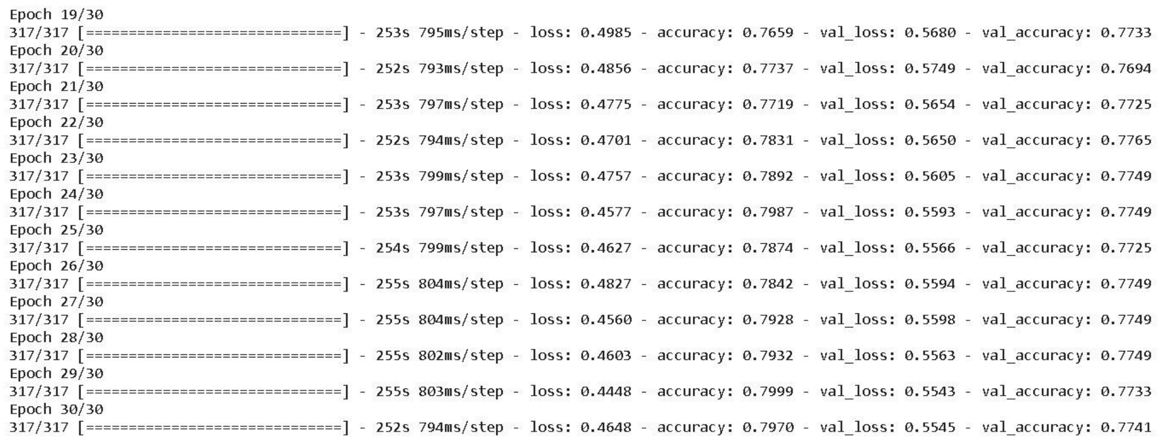
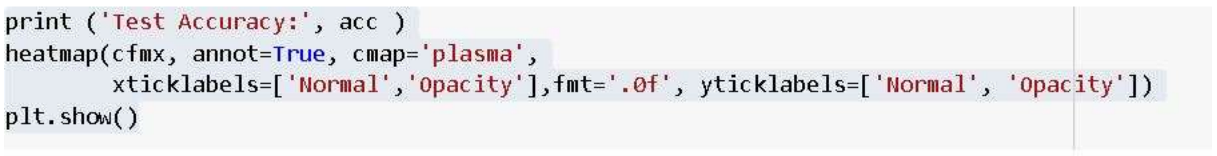


Figure *14* -PSP NET IMAGE SEGMENTATION RESULTS



```
Epoch 19/30
317/317 [==============================] - 253s 795ms/step - loss: 0.4985 - accuracy: 0.7659 - val_loss: 0.5680 - val_accuracy: 0.7733
Epoch 20/30
317/317 [==============================] - 252s 793ms/step - loss: 0.4856 - accuracy: 0.7737 - val_loss: 0.5749 - val_accuracy: 0.7694
Epoch 21/30
317/317 [==============================] - 253s 797ms/step - loss: 0.4775 - accuracy: 0.7719 - val_loss: 0.5654 - val_accuracy: 0.7725
Epoch 22/30
317/317 [==============================] - 252s 794ms/step - loss: 0.4701 - accuracy: 0.7831 - val_loss: 0.5650 - val_accuracy: 0.7765
Epoch 23/30
317/317 [==============================] - 253s 799ms/step - loss: 0.4757 - accuracy: 0.7892 - val_loss: 0.5605 - val_accuracy: 0.7749
Epoch 24/30
317/317 [==============================] - 253s 797ms/step - loss: 0.4577 - accuracy: 0.7987 - val_loss: 0.5593 - val_accuracy: 0.7749
Epoch 25/30
317/317 [==============================] - 254s 799ms/step - loss: 0.4627 - accuracy: 0.7874 - val_loss: 0.5566 - val_accuracy: 0.7725
Epoch 26/30
317/317 [==============================] - 255s 804ms/step - loss: 0.4827 - accuracy: 0.7842 - val_loss: 0.5594 - val_accuracy: 0.7749
Epoch 27/30
317/317 [==============================] - 255s 804ms/step - loss: 0.4560 - accuracy: 0.7928 - val_loss: 0.5598 - val_accuracy: 0.7749
Epoch 28/30
317/317 [==============================] - 255s 802ms/step - loss: 0.4603 - accuracy: 0.7932 - val_loss: 0.5563 - val_accuracy: 0.7749
Epoch 29/30
317/317 [==============================] - 255s 803ms/step - loss: 0.4448 - accuracy: 0.7999 - val_loss: 0.5543 - val_accuracy: 0.7733
Epoch 30/30
317/317 [==============================] - 252s 794ms/step - loss: 0.4648 - accuracy: 0.7970 - val_loss: 0.5545 - val_accuracy: 0.7741
```

Figure 15 – CXR NET MODEL TRAINING



```
print ('Test Accuracy:', acc )
heatmap(cfmx, annot=True, cmap='plasma',
        xticklabels=['Normal','Opacity'],fmt='.0f', yticklabels=['Normal', 'Opacity'])
plt.show()
```

```
Test Accuracy: 0.7740916271721959
```

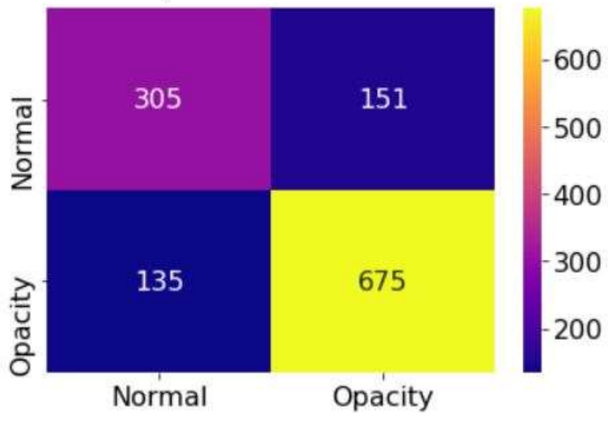
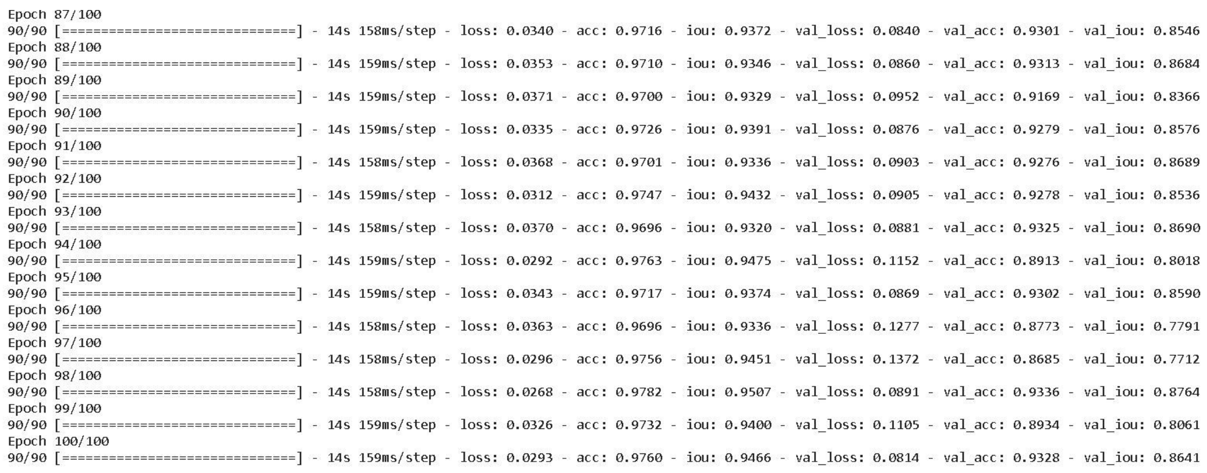


Figure *16 – CXR CONFUSION MATRIX ANALYSIS*



```
Epoch 87/100
90/90 [==============================] - 14s 158ms/step - loss: 0.0340 - acc: 0.9716 - iou: 0.9372 - val_loss: 0.0840 - val_acc: 0.9301 - val_iou: 0.8546
Epoch 88/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0353 - acc: 0.9710 - iou: 0.9346 - val_loss: 0.0860 - val_acc: 0.9313 - val_iou: 0.8684
Epoch 89/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0371 - acc: 0.9700 - iou: 0.9329 - val_loss: 0.0952 - val_acc: 0.9169 - val_iou: 0.8366
Epoch 90/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0335 - acc: 0.9726 - iou: 0.9391 - val_loss: 0.0876 - val_acc: 0.9279 - val_iou: 0.8576
Epoch 91/100
90/90 [==============================] - 14s 158ms/step - loss: 0.0368 - acc: 0.9701 - iou: 0.9336 - val_loss: 0.0903 - val_acc: 0.9276 - val_iou: 0.8689
Epoch 92/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0312 - acc: 0.9747 - iou: 0.9432 - val_loss: 0.0905 - val_acc: 0.9278 - val_iou: 0.8536
Epoch 93/100
90/90 [==============================] - 14s 158ms/step - loss: 0.0370 - acc: 0.9696 - iou: 0.9320 - val_loss: 0.0881 - val_acc: 0.9325 - val_iou: 0.8690
Epoch 94/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0292 - acc: 0.9763 - iou: 0.9475 - val_loss: 0.1152 - val_acc: 0.8913 - val_iou: 0.8018
Epoch 95/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0343 - acc: 0.9717 - iou: 0.9374 - val_loss: 0.0869 - val_acc: 0.9302 - val_iou: 0.8590
Epoch 96/100
90/90 [==============================] - 14s 158ms/step - loss: 0.0363 - acc: 0.9696 - iou: 0.9336 - val_loss: 0.1277 - val_acc: 0.8773 - val_iou: 0.7791
Epoch 97/100
90/90 [==============================] - 14s 158ms/step - loss: 0.0296 - acc: 0.9756 - iou: 0.9451 - val_loss: 0.1372 - val_acc: 0.8685 - val_iou: 0.7712
Epoch 98/100
90/90 [==============================] - 14s 158ms/step - loss: 0.0268 - acc: 0.9782 - iou: 0.9507 - val_loss: 0.0891 - val_acc: 0.9336 - val_iou: 0.8764
Epoch 99/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0326 - acc: 0.9732 - iou: 0.9400 - val_loss: 0.1105 - val_acc: 0.8934 - val_iou: 0.8061
Epoch 100/100
90/90 [==============================] - 14s 159ms/step - loss: 0.0293 - acc: 0.9760 - iou: 0.9466 - val_loss: 0.0814 - val_acc: 0.9328 - val_iou: 0.8641
```

Figure 17 – UNET SEGMENTATION MODEL PERFORMANCE

A.2 PUBLICATIONS

REFERENCES

[1] Cohen, J. P., Morrison, P. & Dao, L. COVID-19 image data collection. arXiv 2003.11597 (2020). 7/8.

[2]Chung, A.Figure1 COVID-19 chest x-ray data initiative. https://github.com/agchung/Figure1-COVID-chestxraydataset (2020).

[3] Gunraj, H., Wang, L. & Wong, A. COVIDNet-CT: A tailored deep convolutional neural network design for detection of COVID-19 cases from chest CT images. Front. Medicine DOI: 10.3389/fmed.2020.608525 (2020).

[4] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)Google Scholar.

[5] Song, Y., Zheng, S., Li, L., Zhang, X., Zhang, X., Huang, Z., Chen, J., Zhao, H., Jie, Y., Wang, R., et al: Deep learning enables accurate diagnosis of novel coronavirus (covid-19) with ct images. medRxiv (2020)Google Scholar.

[6] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.,Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.:Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015) Google Scholar.

[7] Wang, L., Wong, A.: Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. arXiv preprint arXiv:2003.09871 (2020)Google Scholar.

[8] Wang, S., Kang, B., Ma, J., Zeng, X., Xiao, M., Guo, J.,Cai, M., Yang, J., Li, Y., Meng, X., et al: A deep learning algorithm using ct images to screen for corona virus disease (covid-19). MedRxiv (2020) Google Scholar.

[9] Narin, A., Kaya, C., Pamuk, Z.: Automatic detection of coronavirus disease (covid-19) using x-ray images and deep convolutional neural networks. arXiv preprint arXiv:2003.10849 (2020)Google Scholar

[10] Zhang, B., Zhang, L., Zhang, L., Karray, F.: Retinal vessel extraction by matched filter with first-order derivative of gaussian. Computers in biology and medicine 40(4), 438–445 (2010) PubMed Google Scholar.

[11] Zhao, J., Zhang, Y., He, X., Xie, P.: Covid-ct-dataset: a ct scan dataset about covid-19. arXiv preprint arXiv:2003.13865 (2020)Google Scholar.

[12] Zheng, C., Deng, X., Fu, Q., Zhou, Q., Feng, J., Ma, H.,Liu, W., Wang, X.: Deep learning-based detection for covid-19 from chest ct using weak label.

medRxiv (2020)Google Scholar.

[13]. Ng M-Y, Lee EY, Yang J, et al. Imaging profile of the COVID-19 infection: radiologic findings and literature review. Radiology: Cardiothoracic Imaging 2020; 2(1): e200034.

[14].Milletari F, Navab N, Ahmadi S-A. V-net: Fully convolutional neural networks for volumetric medical image segmentation. 2016 Fourth International Conference on 3D Vision (3DV); 2016: IEEE; 2016. p. 565-71.

[15] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition; 2016; 2016. p. 770-8.

[16] Han M, Zhang Y, Zhou Q, et al. Large-scale evaluation of V-Net for organ segmentation in image guided radiation therapy. Medical Imaging 2019: Image-Guided Procedures, Robotic Interventions, and Modeling; 2019: International Society for Optics and Photonics; 2019. p. 109510O.

[17] Pang T, Guo S, Zhang X, Zhao L. Automatic Lung Segmentation Based on Texture and Deep Features of HRCT Images with Interstitial Lung Disease. BioMed Research International 2019; 2019.

[18] Bernheim A, Mei X, Huang M, et al. Chest CT Findings in Coronavirus Disease-19 (COVID19): Relationship to Duration of Infection. Radiology 2020: 200463.

[19] He, K., Fan, H., Wu, Y., Xie, S. & Girshick, R. Momentum contrast for unsupervised visual representation learning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9729–9738 (2020)

[20] Sharma, S. Drawing insights from COVID-19-infected patients using CT scan images and machine learning techniques: A study on 200 patients. *Environ. Sci. Pollut. Res.* **27**, 37155–37163 (2020).