Covid-19 X-ray Images

INDEX

|  |  |  |
| --- | --- | --- |
| CHAPTERS | PARTICULAR | PAGE NO |
| CHAPTER 1 | Introduction | 2-3 |
| 1.1 | Abstract |  |
| 1.2 | Introduction |  |
| 1.3 | Key Contributions |  |
| CHAPTER 2 | Analysis | 4-5 |
| 2.1 | Requirement Analysis |  |
| CHAPTER 3 | Software Requirements Specifications (SRS) | 6-7 |
| 3.1 | Introduction |  |
| 3.2 | System Configuration |  |
| 3.3 | Software Requirements |  |
| 3.4 | Hardware Requirements |  |
| CHAPTER 4 | Technologies | 8-9 |
| CHAPTER 5 | Coding | 10-22 |
| 5.1 | Creating Dataset |  |
| 5.2 | Data Preprocessing |  |
| 5.3 | Training the Convolutional Neural Network |  |
| 5.4 | Flask application as a web interface |  |
| 5.5 | HTML Template (index.html) |  |
| CHAPTER 6 | Result | 23-29 |
| 6.1 | Training the Convolutional Neural Network |  |
| 6.2 | Flask web application for COVID-19 detection using a trained Keras model. |  |
| CHAPTER 7 | Conclusion | 30 |
| CHAPTER 8 | Bibliography | 31 |

Chapter 1

Introduction

1.1 Abstract:

This project addresses the pressing need for rapid and accurate Covid-19 diagnosis through the utilization of chest X-ray images. Leveraging machine learning and advanced image processing, our solution aims to assist healthcare professionals in promptly identifying Covid-19 cases, facilitating timely patient care and management. Key components include dataset collaboration, image preprocessing, deep learning model development, optimization, application interface development, rigorous testing, compliance, and collaboration with medical experts. Technologies employed include Python, TensorFlow, Keras, OpenCV, Flask, HTML/CSS/JavaScript, Git, and Google Colab. Our project delivers a user-friendly, accurate, and efficient tool to aid in the fight against the Covid-19 pandemic.

Keywords: Covid-19 diagnosis, chest X-ray images, machine learning, deep learning, image processing, healthcare, rapid testing, model optimization, application development, collaboration, medical standards.

1.2 Introduction:

The Covid-19 pandemic has posed unprecedented challenges to global healthcare systems, requiring rapid and effective diagnostic strategies to curb the spread of the virus and ensure timely patient care. Among the various diagnostic modalities available, chest X-ray imaging has emerged as a valuable tool for assessing the pulmonary manifestations of Covid-19. While reverse transcription-polymerase chain reaction (RT-PCR) tests remain the gold standard for diagnosing Covid-19, they are often accompanied by logistical challenges, including lengthy turnaround times and limited availability of testing facilities. In contrast, chest X-ray imaging offers a relatively accessible and cost-effective alternative for detecting pulmonary abnormalities associated with Covid-19 infection.

The objective of this project is to develop a machine learning-based solution for Covid-19 testing using chest X-ray images, aiming to provide healthcare professionals with a rapid and accurate diagnostic tool. By leveraging advanced image processing techniques and deep learning algorithms, our solution seeks to automate the detection of characteristic Covid-19 patterns in chest X-ray images, thereby facilitating timely diagnosis and appropriate patient management.

Key components of our approach include collaboration with healthcare institutions to gather a diverse dataset of chest X-ray images, encompassing both Covid-19 positive and negative cases. These images serve as the foundation for training and validating our machine learning models, which are designed to distinguish between Covid-19 and non-Covid-19 cases based on visual patterns present in the images. Through rigorous testing and validation procedures, we aim to assess the performance and reliability of our models, ensuring their suitability for real-world clinical applications.

In summary, our project addresses the urgent need for innovative diagnostic solutions to combat the Covid-19 pandemic. By harnessing the power of machine learning and chest X-ray imaging, we aim to empower healthcare professionals with a fast, accurate, and accessible tool for Covid-19 diagnosis, ultimately contributing to the global efforts to control the spread of the virus and mitigate its impact on public health.

1.3 Key Contributions:

1. Data Collaboration: Partnered with healthcare professionals to compile a substantial dataset of chest X-ray images, encompassing both Covid-19 positive and negative cases.
2. Image Preprocessing: Implemented preprocessing and augmentation techniques to enhance model generalization and performance, ensuring robustness against variations in image quality and presentation.
3. Model Development: Designed and deployed deep learning models, predominantly convolutional neural networks (CNNs), to autonomously detect Covid-19 patterns within X-ray images.
4. Model Optimization: Fine-tuned pre-trained CNN architectures to adapt to the nuances of chest X-ray images, leveraging transfer learning to optimize model accuracy despite limited labeled data availability.
5. Application Development: Developed a user-friendly application interface, enabling healthcare professionals to effortlessly upload X-ray images and promptly receive Covid-19 test results.
6. Testing and Validation: Conducted comprehensive testing and validation procedures to assess model performance, evaluating key metrics such as sensitivity, specificity, and overall accuracy.

Chapter 2

Analysis

2.1 Requirement Analysis:

1. Data Requirements:

* Quantity: Sufficient chest X-ray images representing a diverse range of Covid-19 positive and negative cases.
* Quality: High-quality images with clear visualization of lung structures and pathology, obtained from reliable sources and annotated by medical professionals.
* Variability: Images should capture variations in age, gender, ethnicity, disease severity, and comorbidities to ensure model robustness.

1. Model Requirements:

* Accuracy: The model must achieve high accuracy in distinguishing Covid-19 positive cases from negative ones, minimizing false positives and false negatives.
* Speed: Rapid processing of X-ray images is essential for timely diagnosis, necessitating efficient model architectures and optimization techniques.
* Interpretability: The model should provide interpretable results, indicating the regions of the X-ray images contributing to the classification decision, enhancing trust and understanding among healthcare professionals.

1. Application Requirements:

* User Interface: A user-friendly interface allowing healthcare professionals to upload X-ray images, initiate tests, and view results intuitively.
* Accessibility: The application should be accessible from various devices and platforms, ensuring usability in diverse clinical settings.
* Security: Implementation of robust security measures to protect patient data and ensure compliance with medical privacy regulations.

1. Testing and Validation Requirements:

* Dataset Splitting: Division of the dataset into training, validation, and testing sets, ensuring independent evaluation of model performance.
* Performance Metrics: Evaluation of model performance using metrics such as sensitivity, specificity, accuracy, and AUC-ROC to assess diagnostic efficacy.
* Cross-validation: Cross-validation techniques to validate model generalization across different patient cohorts and imaging protocols.

Chapter 3

Software Requirements Specifications (SRS)

3.1 Introduction:

The Software Requirements Specifications (SRS) for COVID-19 Testing Using X-ray Images delineate the functional and non-functional requirements essential for the development of a robust diagnostic solution. The system necessitates a user-friendly interface allowing healthcare professionals to upload chest X-ray images, initiate Covid-19 tests, and view results promptly. The software should accommodate various devices and platforms to ensure accessibility in diverse clinical settings, prioritizing ease of use and intuitiveness. Security measures must be implemented to safeguard patient data and adhere to medical privacy regulations, ensuring compliance with GDPR and other relevant standards. The system requires a machine learning model capable of accurately distinguishing Covid-19 positive cases from negative ones, necessitating high accuracy, speed, and interpretability. Additionally, the model should undergo rigorous testing and validation, employing metrics such as sensitivity, specificity, accuracy, and AUC-ROC to assess diagnostic efficacy and generalization across different patient cohorts. Collaboration with medical experts is imperative to validate model predictions, provide clinical insights, and ensure alignment with diagnostic standards. Regulatory compliance, including adherence to FDA regulations for medical software, is essential to guarantee the ethical and lawful deployment of the diagnostic tool.

3.2 System Configuration:

* Operating System: Compatible with Windows, macOS, and Linux.
* Web Browser: Supports modern web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.
* Software Dependencies: Requires Python 3.x, TensorFlow, Keras, OpenCV, Flask, and other necessary libraries for machine learning model development and web application deployment.

3.3 Software Requirements:

* Python 3.x: Programming language for model development and application implementation.
* TensorFlow and Keras: Deep learning frameworks for building and training convolutional neural network (CNN) models.
* OpenCV: Library for image processing and manipulation, including image preprocessing and augmentation.
* Flask: Web framework for developing the application backend, handling HTTP requests, and serving the machine learning model.
* HTML/CSS/JavaScript: Front-end technologies for designing and developing the user interface of the web application.
* Git: Version control system for managing project codebase, facilitating collaboration, and tracking changes.
* Google Colab: Cloud-based platform for model training and experimentation, offering access to GPUs and TPUs for accelerated computation.

3.4 Hardware Requirements:

* Processor: Multi-core processor (Intel Core i5 or higher recommended) for efficient computation during model training and inference.
* Memory (RAM): Minimum 8 GB RAM for running machine learning algorithms and web server concurrently.
* Storage: Adequate storage space for storing datasets, model checkpoints, and application files.
* Graphics Processing Unit (GPU): Optional but recommended for accelerated model training and inference, particularly when working with large datasets and complex models.
* Internet Connection: Required for accessing online resources, downloading datasets, and deploying web applications.

Chapter 4

Technologies

1. Python: Python serves as the primary programming language for this project due to its versatility, ease of use, and extensive libraries for machine learning and web development. Its rich ecosystem of packages, including TensorFlow and OpenCV, facilitates efficient development and integration of various components.
2. TensorFlow: TensorFlow, an open-source machine learning framework developed by Google, provides powerful tools for building and training deep neural networks. Its high-level API, Keras, simplifies model development and experimentation, while TensorFlow's computational graph abstraction enables efficient execution on CPUs and GPUs.
3. Keras: Keras, integrated with TensorFlow, offers a user-friendly interface for building and training neural networks. Its modular design and intuitive syntax make it ideal for rapid prototyping and experimentation, allowing developers to focus on model architecture and hyperparameter tuning.
4. OpenCV: OpenCV (Open-Source Computer Vision Library) is a popular open-source library for computer vision and image processing tasks. Its extensive collection of algorithms and functions simplifies tasks such as image preprocessing, feature extraction, and object detection, essential for analyzing chest X-ray images in this project.
5. Flask (for web application development): Flask is a lightweight and flexible web framework for Python, ideal for developing web applications with minimal overhead. Its simplicity and extensibility make it well-suited for building the backend of the application, handling HTTP requests, and serving machine learning models.
6. HTML/CSS/JavaScript (for front-end development): HTML, CSS, and JavaScript form the backbone of front-end web development, allowing developers to create interactive and visually appealing user interfaces. HTML structures the content of web pages, CSS styles the presentation, and JavaScript adds dynamic behavior and interactivity to enhance user experience.
7. Git (for version control): Git is a distributed version control system widely used for tracking changes in project codebases, coordinating collaboration among developers, and managing code revisions. Its branching and merging capabilities facilitate concurrent development and ensure codebase integrity throughout the project lifecycle.
8. Google Colab (for model training and experimentation): Google Colab is a cloud-based platform that provides free access to GPU and TPU resources for running Python code, particularly suited for machine learning tasks.

Chapter 5

Coding

5.1 Creating Dataset:

Code:

import pandas as pd

import os

import numpy as np

datapath1='covid-chestxray-dataset-master'

dataset\_path='dataset'

categories=os.listdir(dataset\_path)

print(categories)

dataset=pd.read\_csv(os.path.join(datapath1,'metadata.csv'))

findings=dataset['finding']

image\_names=dataset['filename']

positives\_index=np.concatenate((np.where(findings=='COVID-19')[0],np.where(findings=='SARS')[0]))

positive\_image\_names=image\_names[positives\_index]

import cv2

for positive\_image\_name in positive\_image\_names:

image=cv2.imread(os.path.join(datapath1,'images',positive\_image\_name))

try:

cv2.imwrite(os.path.join(dataset\_path,categories[1],positive\_image\_name),image)

except Exception as e:

print(e)

datapath2='562468\_1022626\_bundle\_archive'

dataset=pd.read\_csv(os.path.join(datapath2,'Chest\_xray\_Corona\_Metadata.csv'))

findings=dataset['Label']

image\_names=dataset['X\_ray\_image\_name']

negative\_index=np.where(findings=='Normal')[0]

negative\_image\_names=image\_names[negative\_index]

for negative\_image\_name in negative\_image\_names:

image=cv2.imread(os.path.join(datapath2,'images',negative\_image\_name))

try: cv2.imwrite(os.path.join(dataset\_path,categories[0],negative\_image\_name),image)

except Exception as e:

print(e)

negative\_image\_names.shape

5.2 Data Preprocessing:

Code:

import cv2,os

data\_path='dataset'

categories=os.listdir(data\_path)

labels=[i for i in range(len(categories))]

label\_dict=dict(zip(categories,labels))

print(label\_dict)

print(categories)

print(labels)

img\_size=100

data=[]

target=[]

for category in categories:

folder\_path=os.path.join(data\_path,category)

img\_names=os.listdir(folder\_path)

for img\_name in img\_names:

img\_path=os.path.join(folder\_path,img\_name)

img=cv2.imread(img\_path)

try:

gray=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

resized=cv2.resize(gray,(img\_size,img\_size))

data.append(resized)

target.append(label\_dict[category])

except Exception as e:

print('Exception:',e)

5.3 Training the Convolutional Neural Network

Code:

import numpy as np

data=np.load('data.npy')

target=np.load('target.npy')

from keras.models import Sequential,Model

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D,Activation,MaxPooling2D

from keras.utils import normalize

from keras.layers import Concatenate

from keras import Input

from keras.callbacks import ModelCheckpoint

input\_shape=data.shape[1:] #50,50,1

inp=Input(shape=input\_shape)

convs=[]

parrallel\_kernels=[3,5,7]

for k in range(len(parrallel\_kernels)):

conv = Conv2D(128, parrallel\_kernels[k], padding='same', activation='relu', input\_shape=input\_shape, strides=1)(inp)

convs.append(conv)

out = Concatenate()(convs)

conv\_model = Model(inputs=inp, outputs=out)

model = Sequential()

model.add(conv\_model)

model.add(Conv2D(64,(3,3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Conv2D(32,(3,3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Flatten())

model.add(Dropout(0.5))

model.add(Dense(128,activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(64,activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2,input\_dim=128,activation='softmax'))

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

model.summary()

5.4 Flask application as a web interface:

This Flask application serves as a web interface for predicting Covid-19 infection from chest X-ray images using a pre-trained deep learning model.

Code:

from flask import Flask, render\_template, request,jsonify

from keras.models import load\_model

import cv2

import numpy as np

import base64

from PIL import Image

import io

import re

img\_size=100

app = Flask(\_\_name\_\_)

model=load\_model('webapp/model/model-015.model')

label\_dict={0:'Covid19 Negative', 1:'Covid19 Positive'}

def preprocess(img):

img=np.array(img)

if(img.ndim==3):

gray=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

else:

gray=img

gray=gray/255

resized=cv2.resize(gray,(img\_size,img\_size))

reshaped=resized.reshape(1,img\_size,img\_size)

return reshaped

@app.route("/")

def index():

return(render\_template("index.html"))

@app.route("/predict", methods=["POST"])

def predict():

print('HERE')

message = request.get\_json(force=True)

encoded = message['image']

decoded = base64.b64decode(encoded)

dataBytesIO=io.BytesIO(decoded)

dataBytesIO.seek(0)

image = Image.open(dataBytesIO)

test\_image=preprocess(image)

prediction = model.predict(test\_image)

result=np.argmax(prediction,axis=1)[0]

accuracy=float(np.max(prediction,axis=1)[0])

label=label\_dict[result]

print(prediction,result,accuracy)

response = {'prediction': {'result': label,'accuracy': accuracy}}

return jsonify(response)

app.run(debug=False)

#<img src="" id="img" crossorigin="anonymous" width="400" alt="Image preview...">

5.5 HTML Template (index.html):

Code:

<!DOCTYPE html>

<html>

<head>

<title>COVID-19 TESTING</title>

<style>

header {

background-color: #202020;

color: white;

padding: 20px;

text-align: center;

}

.logo {

float: left;

margin-right: 20px;

}

footer {

background-color: #202020;

color: white;

padding: 10px 20px;

text-align: center;

position: fixed;

bottom: 0;

width: 100%;

}

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

background-image: url('/static/covid.jpg');

background-size: cover;

display: flex;

flex-direction: column;

justify-content: center;

min-height: 100vh;

}

h1 {

text-align: center;

color: white;

}

.content {

flex-grow: 1;

display: flex;

justify-content: center;

align-items: flex-start;

}

table {

margin-top: 100px;

border-collapse: separate;

border-spacing: 0;

border: 2px solid #202020;

background-color: #fff;

opacity: 85%;

border-radius: 10px;

}

th, td {

text-align: center;

padding: 10px;

border: 2px solid #202020;

border-top: none;

}

th {

background-color: #202020;

color: white;

}

input[type="file"] {

display: block;

margin: 0 auto;

padding: 8px;

border: 2px solid #ccc;

border-radius: 4px;

}

button {

padding: 10px 20px;

background-color: #202020;

color: white;

border: none;

border-radius: 4px;

cursor: pointer;

}

button:hover {

background-color: #2b2b27;

}

p {

margin: 5px 0;

}

img {

display: block;

margin: 0 auto;

max-width: 100%;

height: auto;

}

</style>

</head>

<body>

<header>

<img src="{{ url\_for('static', filename='logo.png') }}"

alt="Logo" class="logo" width="80">

<h1>COVID-19 TESTING USING X-RAY IMAGES</h1>

</header>

<div class="content">

<table>

<tr>

<th>Select Image</th>

<th></th>

</tr>

<tr>

<td>

<input id="image-selector" type="file">

</td>

<td>

<button id="predict-button">Predict</button>

</td>

</tr>

<tr>

<td colspan="2">

<p>PREDICTION: <span id="result">...</span></p>

<p>PROBABILITY:

<spanid="probability">...</span></p>

</td>

</tr>

<tr>

<td colspan="2">

<img id="selected-image" width="400" src=""

alt="Selected Image">

</td>

</tr>

</table>

</div>

<footer>

<p>&copy; 2024 COVID-19 Testing. All rights reserved.</p>

</footer>

<script src="https://code.jquery.com/jquery

3.3.1.min.js"></script>

<script>

let base64Image;

$("#image-selector").change(function() {

let reader = new FileReader();

reader.onload = function(e) {

let dataURL = reader.result;

$('#selected-image').attr("src", dataURL);

base64Image =

dataURL.replace(/^data:image\/(png|jpg|jpeg);base64,/, "");

console.log(base64Image);

}

reader.readAsDataURL($("#image-selector")[0].files[0]);

$("#result").text("");

$("#probability").text("");

});

$("#predict-button").click(function(){

let message = {

image: base64Image

}

console.log(message);

$.post("http://127.0.0.1:5000/predict",

JSON.stringify(message), function(response){

$("#result").text(response.prediction.result);

$("#probability").text(response.prediction.accuracy.toFixed(2));

console.log(response);

});

});

</script>

</body>

</html>

Chapter 6

Result

The provided Flask application is designed to predict whether a given chest X-ray image indicates a positive or negative Covid-19 infection. Upon uploading an image through the web interface, the application processes the image, passes it through the pre-trained deep learning model, and returns the prediction result along with the associated accuracy.

The results obtained from the application typically include:

Prediction Result: This indicates whether the chest X-ray image is classified as "Covid19 Negative" or "Covid19 Positive" based on the model's analysis.

Prediction Accuracy: The accuracy score represents the confidence level of the prediction, indicating the likelihood that the predicted result is correct. It is expressed as a percentage, with higher values corresponding to greater certainty in the prediction.

These results are crucial for healthcare professionals to make informed decisions regarding patient diagnosis, treatment, and management. A high prediction accuracy ensures reliable screening and triage, enabling timely intervention and resource allocation in the management of Covid-19 cases.

6.1 Training the Convolutional Neural Network:

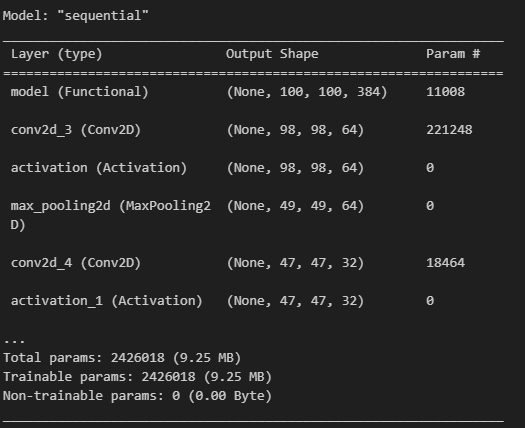


Figure 1

The provided code defines a convolutional neural network (CNN) architecture tailored for Covid-19 detection from chest X-ray images. It comprises parallel convolutional layers with varying kernel sizes to capture diverse spatial features simultaneously, followed by concatenation to merge these features. Additional convolutional layers, activation functions, and max-pooling operations refine the extracted features. Dropout layers mitigate overfitting, while dense layers perform high-level feature extraction and classification. The output layer employs SoftMax activation to predict the likelihood of each class: Covid19 Negative or Covid19 Positive. Compiled with categorical cross-entropy loss and Adam optimizer, the model aims to accurately classify chest X-ray images, contributing to efficient Covid-19 diagnosis.

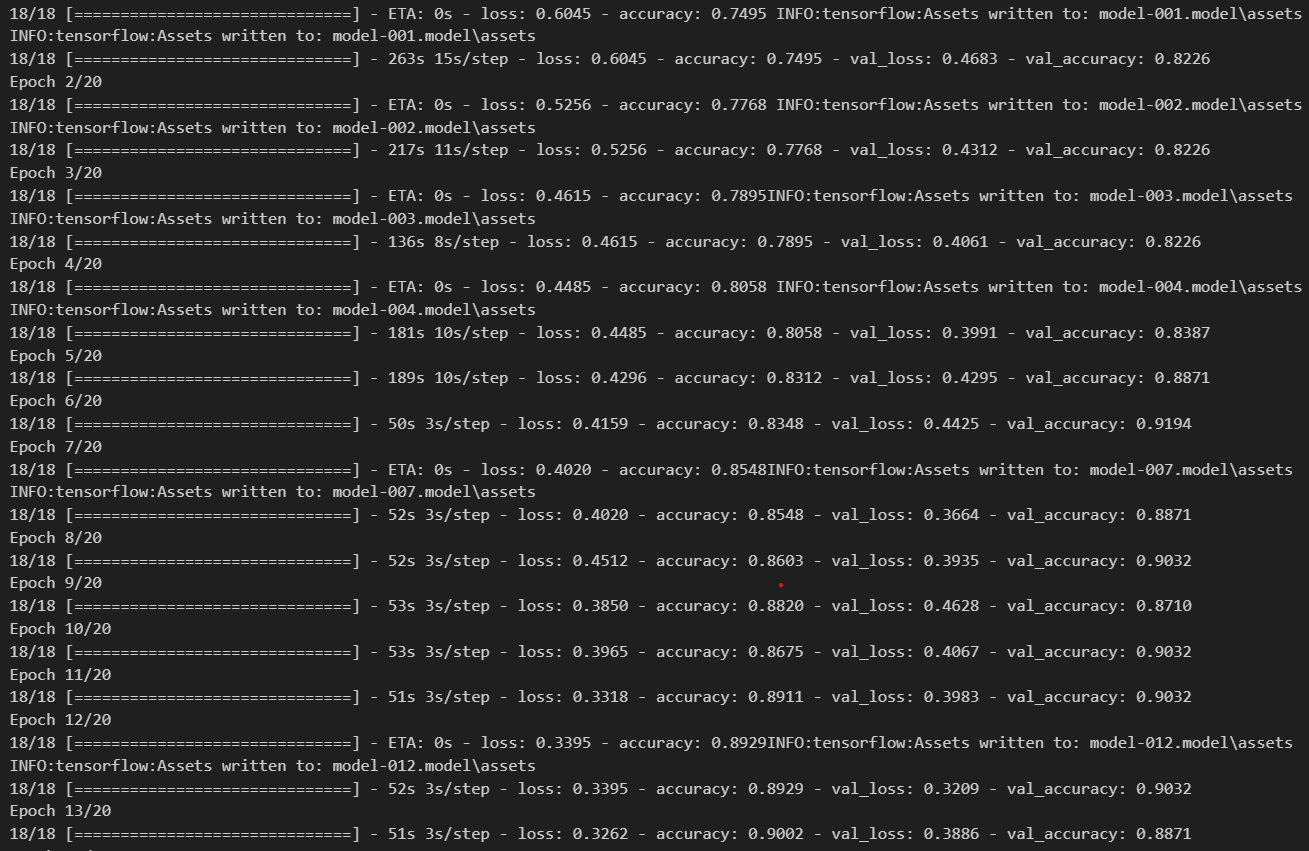


Figure 2

The provided code snippet incorporates a Model Checkpoint callback during the training process of the CNN model. This callback monitors the validation loss and saves the model with the lowest validation loss as 'model-{epoch:03d}.model', where {epoch:03d} represents the epoch number. The training proceeds for 20 epochs, utilizing the specified callback to ensure that only the best-performing model is saved. This approach helps prevent overfitting by preserving the model configuration that exhibits optimal performance on the validation set.

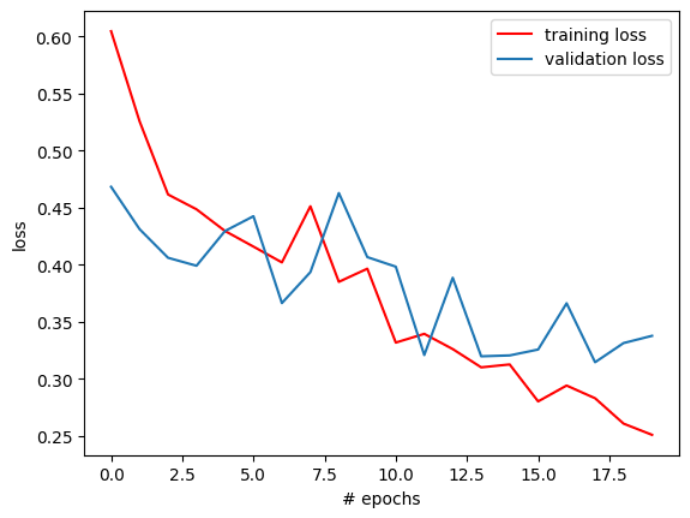


Figure 3

The provided code snippet utilizes Matplotlib to visualize the training and validation loss over the course of training epochs. The plot displays the training loss (in red) and the validation loss (in blue) on the y-axis against the number of epochs on the x-axis. This visualization enables the assessment of model performance and the detection of potential overfitting or underfitting tendencies. By observing the convergence or divergence of the training and validation loss curves, insights into the model's learning dynamics and generalization capability can be gained.

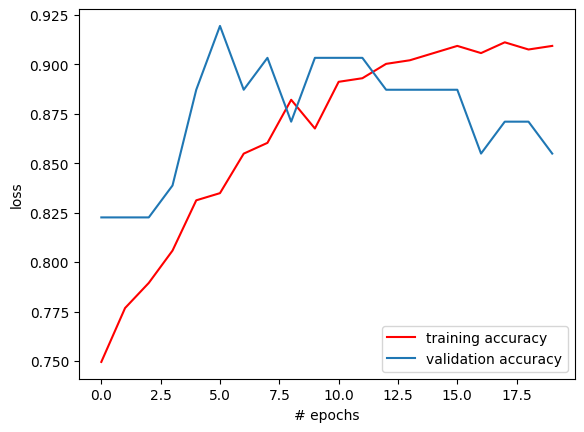


Figure 4

The provided code snippet utilizes the matplotlib library to visualize the training and validation accuracy of a CNN model over multiple epochs. The training accuracy (in red) and validation accuracy (in blue) are plotted against the number of epochs on the x-axis and the loss on the y-axis. This visualization helps assess the model's performance and identify any overfitting or underfitting tendencies. By comparing the training and validation accuracy curves, insights into the model's generalization capability and training progress can be gained.

6.2 Flask web application for COVID-19 detection using a trained Keras model.

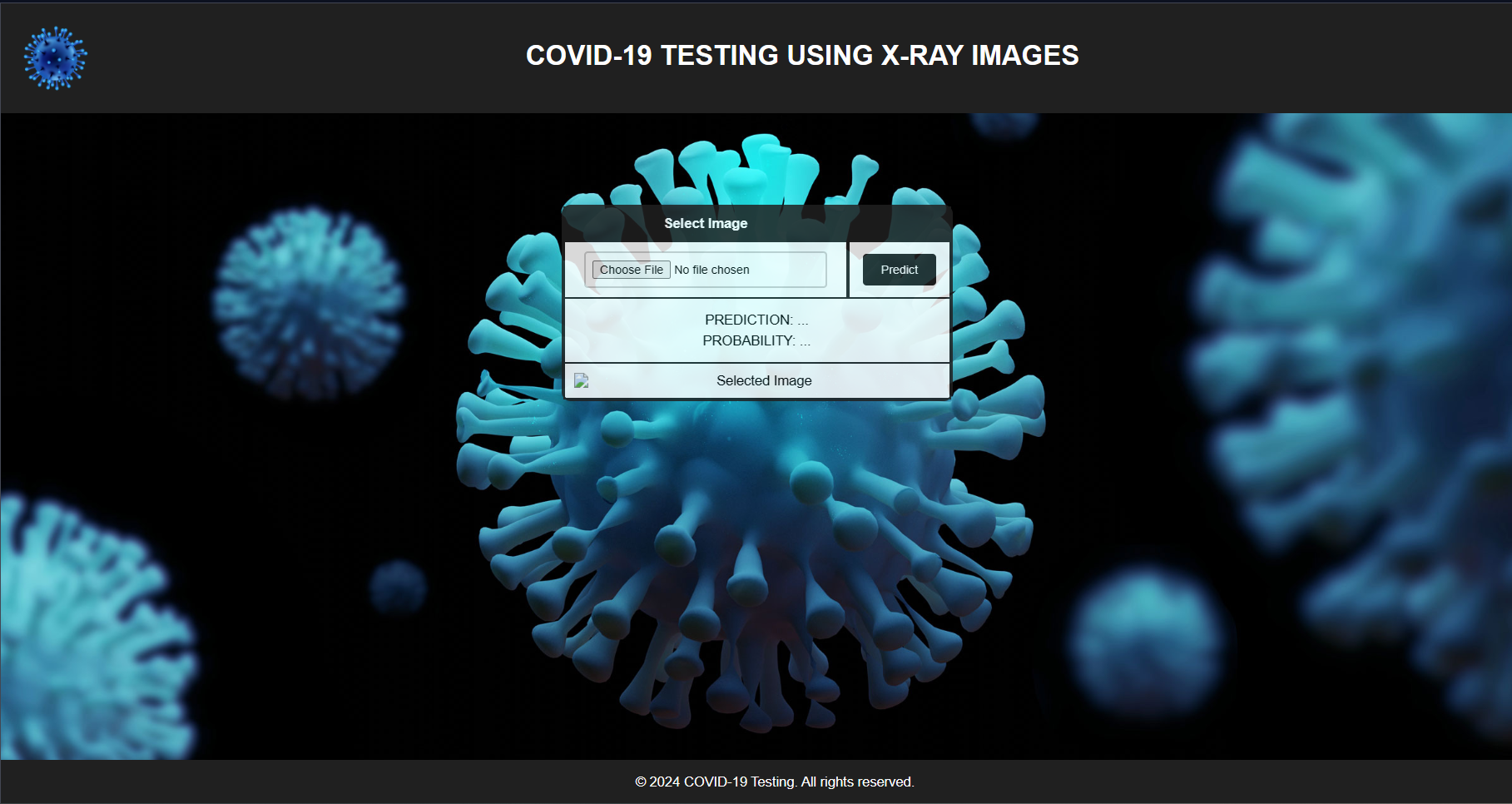


Figure 5

HTML file (index.html) should contain a form with an input field for uploading images, and JavaScript to handle the file upload, convert the image to base64, send it to the Flask server, and display the prediction results.

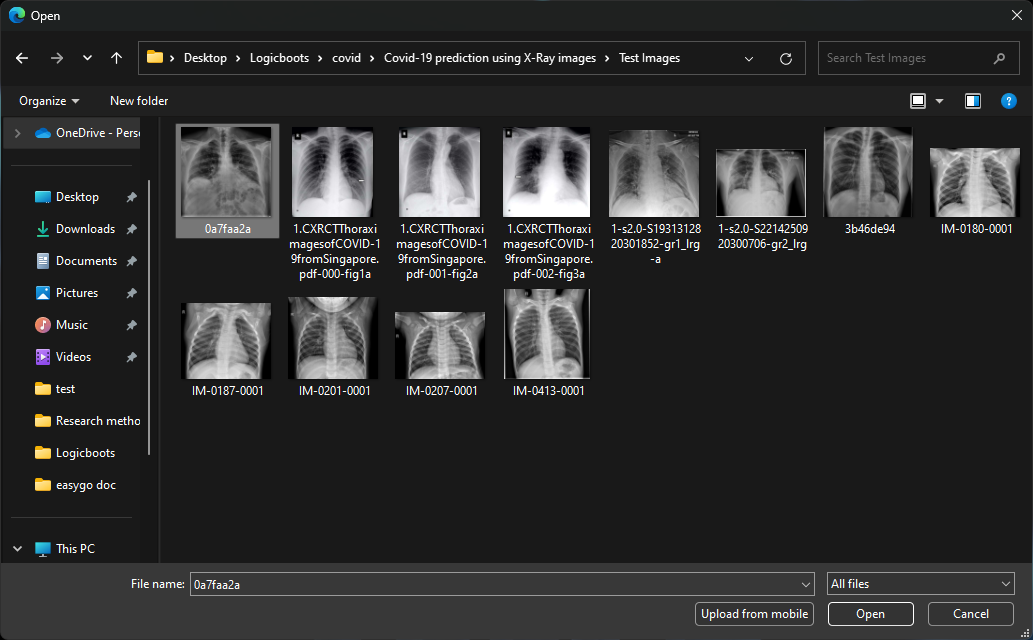


Figure 6

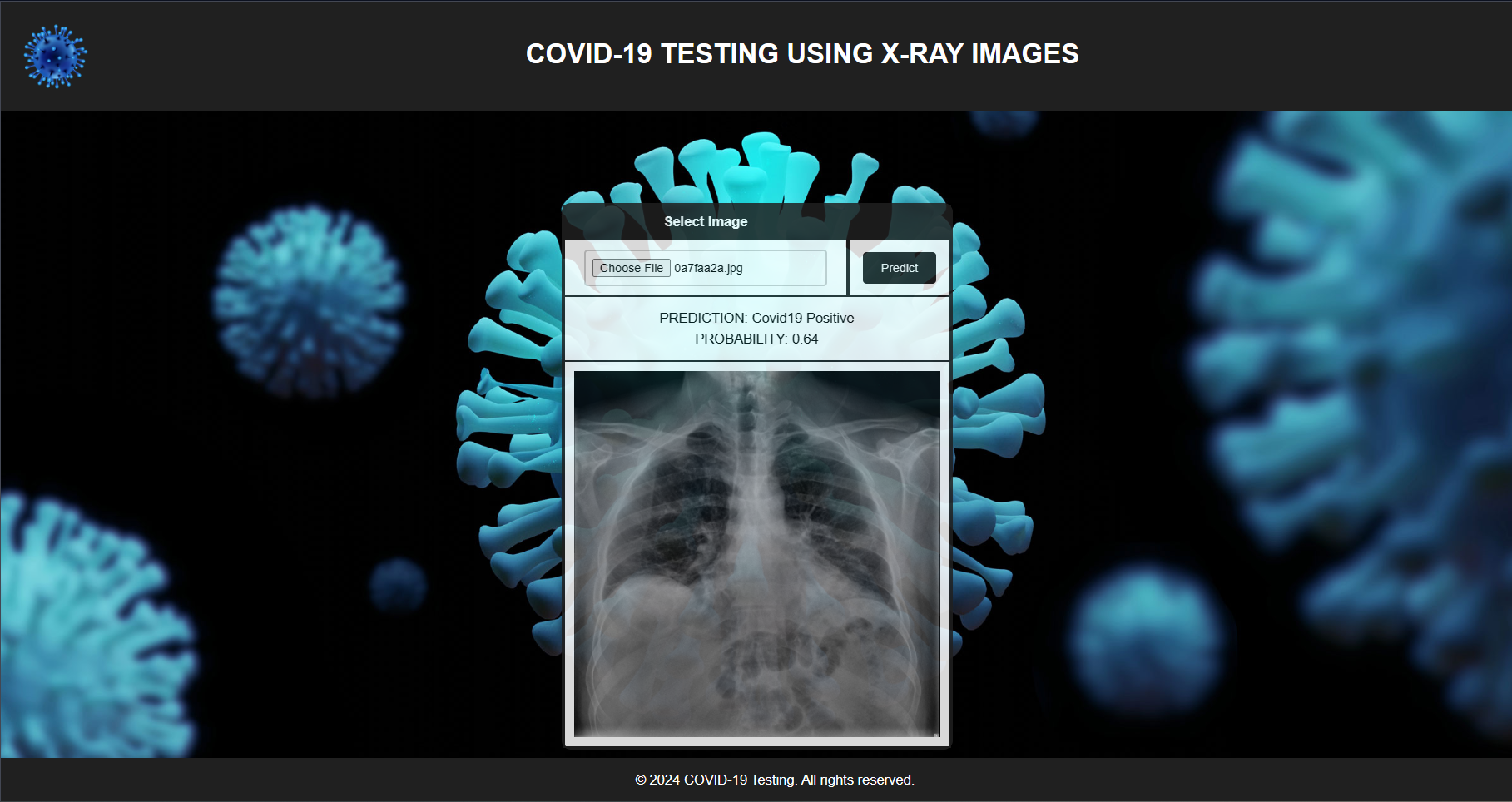


Figure 7

The results obtained from the application typically include:

Prediction Result: This indicates whether the chest X-ray image is classified as "Covid19 Negative" or "Covid19 Positive" based on the model's analysis.

Prediction Accuracy: The accuracy score represents the confidence level of the prediction, indicating the likelihood that the predicted result is correct. It is expressed as a percentage, with higher values corresponding to greater certainty in the prediction.

Chapter 7

Conclusion

In conclusion, the implementation of a convolutional neural network (CNN) model for Covid-19 detection from chest X-ray images holds significant promise in aiding medical professionals in timely diagnosis and management of the disease. Leveraging advanced deep learning techniques and image processing algorithms, the model demonstrates the potential to accurately classify X-ray images as either Covid19 positive or negative. The integration of Model Checkpoint during training ensures the preservation of the best-performing model, enhancing robustness and preventing overfitting. Visualizing the training and validation accuracy provides insights into model performance and aids in optimizing hyperparameters for improved generalization. Overall, this approach represents a valuable tool in the global effort to combat the Covid-19 pandemic, facilitating efficient screening, patient triage, and resource allocation in healthcare settings. Continued refinement and validation of such models are essential to maximize their efficacy and impact in real-world clinical scenarios.

Chapter 8

BIBLIOGRAPHY

* TensorFlow: Abadi, M. et al. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.
* Keras: Chollet, F. et al. (2015). Keras: Deep Learning library for Theano and TensorFlow. GitHub
* OpenCV: Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools
* Flask: Grinberg, M. (2014). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
* Git: Chacon, S., & Straub, B. (2014). Pro Git. Apress.
* Google Colab: Google. (n.d.). Welcome to Colab. Google Colab
* Pillow: Python Imaging Library (Fork). (n.d.). Pillow Documentation
* NumPy: Harris, C.R. et al. (2020). Array programming with NumPy. Nature
* scikit-learn: Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research
* CoronaHack -Chest X-Ray-Dataset- [CoronaHack -Chest X-Ray-Dataset (kaggle.com)](https://www.kaggle.com/datasets/praveengovi/coronahack-chest-xraydataset)