# COVID 19 SEGMENTATION API

**GUIDE**

Ms . Nivea Kesav

Assistant Professor

**BATCH NO: 2**
**TEAM NO: 14**

**Team Members**

Dheeraj N Unni

Antony Maliakkal

Alias Saju T

Jazzir Hussain

# INTRODUCTION

- **COVID-19** is an **infectious disease caused by the SARS-CoV-2**
- **Common method detecting** COVID-19 - **RT-PCR** testing,
- **CT scan imaging** - **fast and cost-effective** alternative
- **Deep learning techniques - U-Net models**
- This **promising research field** - significant step towards rapid and reliable COVID-19 detection

# MOTIVATION

- Detecting COVID-19 from CT-scan image using CNN models is a highly relevant and timely project, given the ongoing global pandemic.

- Creating a reliable CNN model to detect COVID-19 from CT-scans can assist healthcare professionals in early diagnosis and treatment, utilizing a common diagnostic tool.

- Developing a CNN model for CT-scan-based COVID-19 detection can aid in patient triage and improve outcomes in the fight against the pandemic. Ultimately, this project has the potential to contribute to the fight against COVID-19 and improve patient outcomes.

# PROBLEM STATEMENT

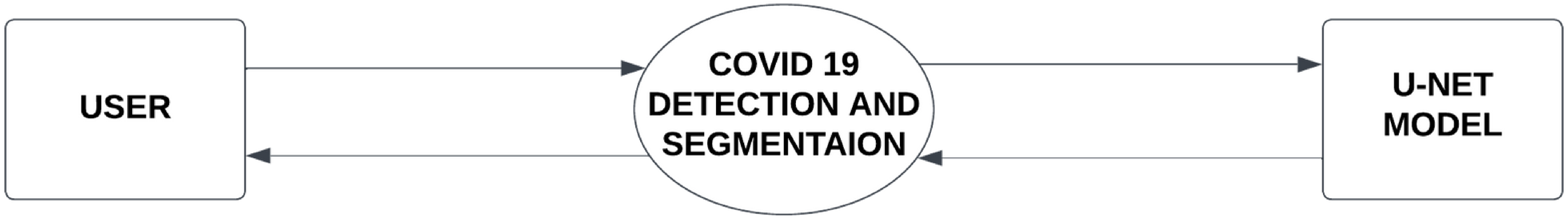
The accurate segmentation of COVID-19 infected regions in medical images is crucial for diagnosing and monitoring the disease. However, the manual segmentation process is time-consuming, subjective, and prone to human error. Therefore, there is a need to develop a reliable and efficient COVID-19 Segmentation API using the UNet deep learning model to automate the segmentation process and provide accurate and timely results.

# OBJECTIVES

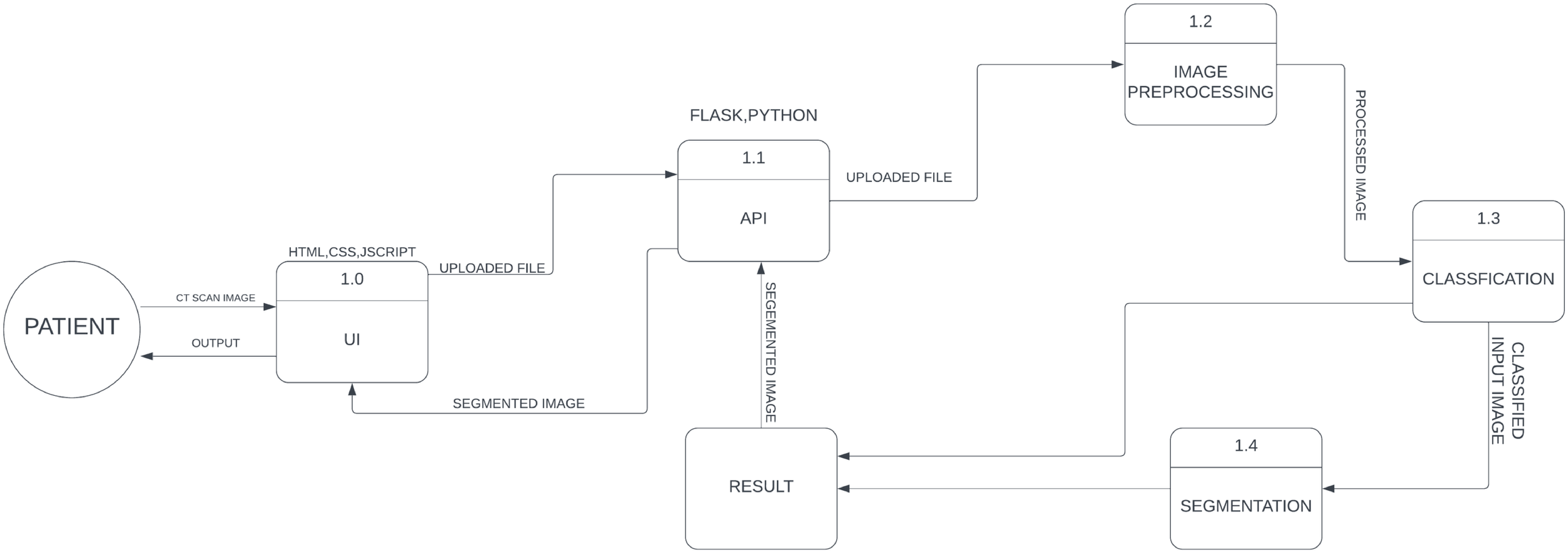
- Develop an API that allows users to perform automated segmentation of COVID-19 infected regions in medical images.
- Implement the UNet architecture, a deep learning model known for its effectiveness in image segmentation tasks, as the core algorithm for the API
- Optimize the model for efficient inference, ensuring that the API provides segmentation results in a timely manner
- Implement a user friendly interface.
- Provide comprehensive testing to validate the accuracy and reliability of the segmentation results across different types of COVID-19 lung images

# DATA FLOW DIAGRAM

## LEVEL 0

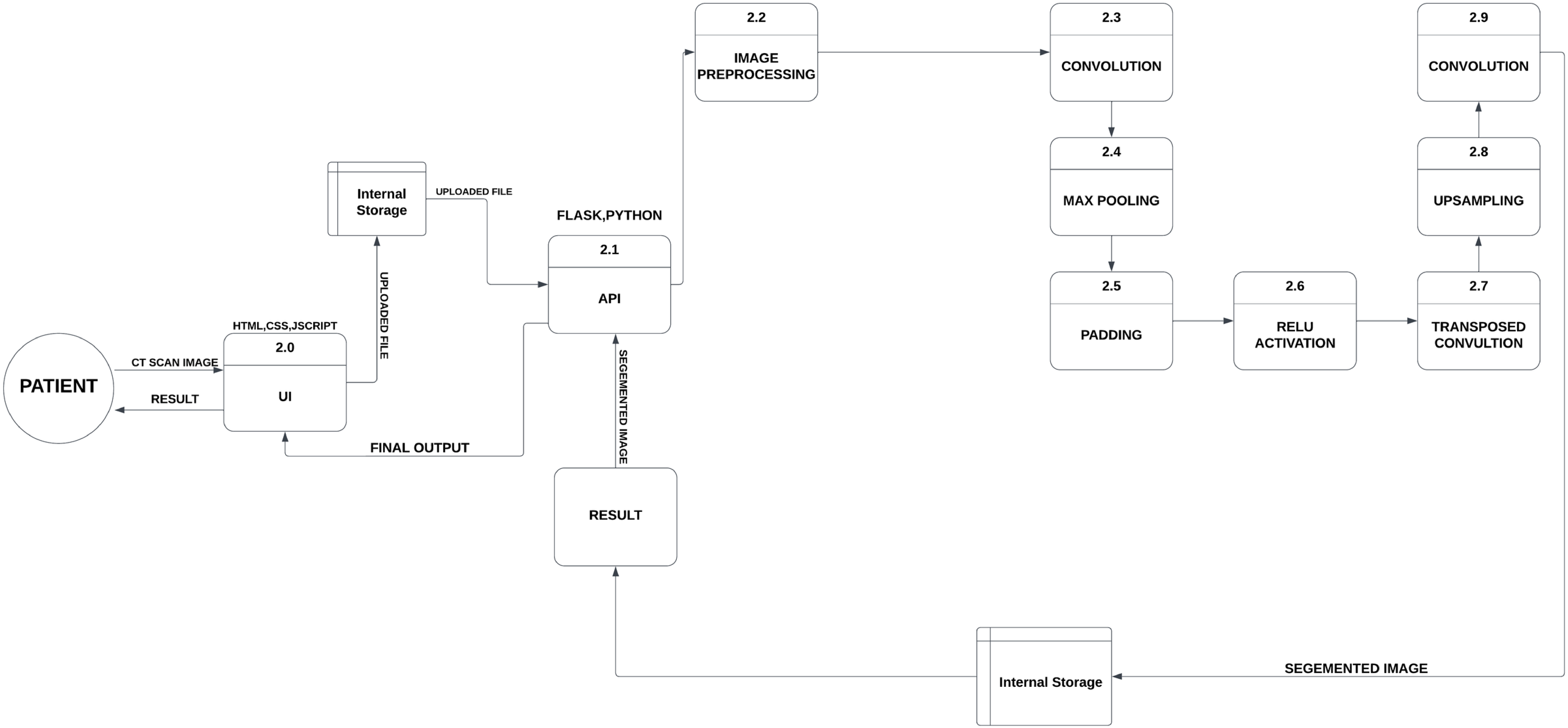


# LEVEL 1

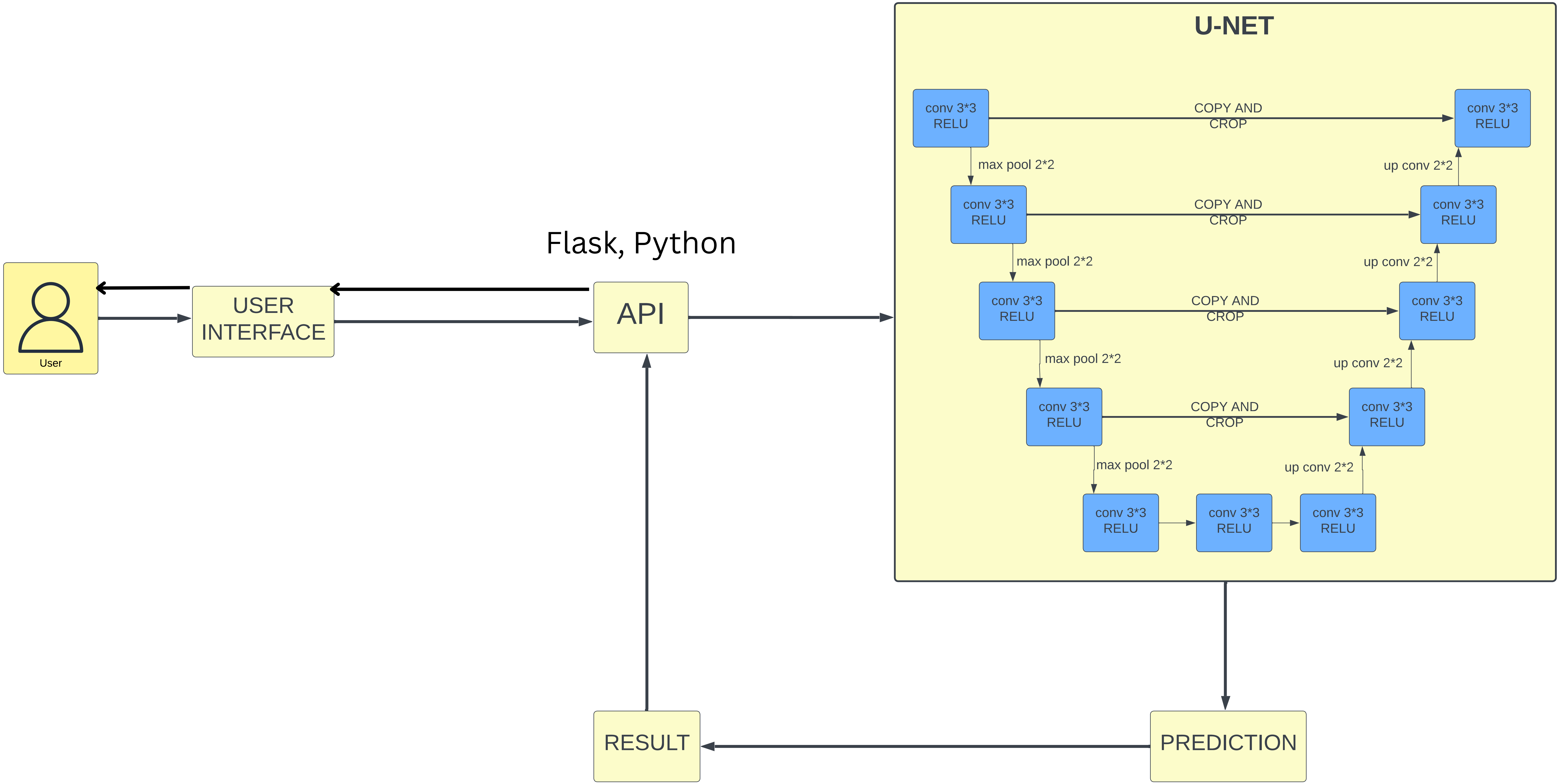


1.2
IMAGE PREPROCESSING
FLASK,PYTHON
1.1
API
UPLOADED FILE
PROCESSED IMAGE
1.3
CLASSFICATION
HTML,CSS,JSCRIPT
1.0
UI
UPLOADED FILE
CT SCAN IMAGE
PATIENT
OUTPUT
SEGEMENTED IMAGE
CLASSIFIED INPUT IMAGE
SEGMENTED IMAGE
1.4
SEGMENTATION
RESULT

# LEVEL 2

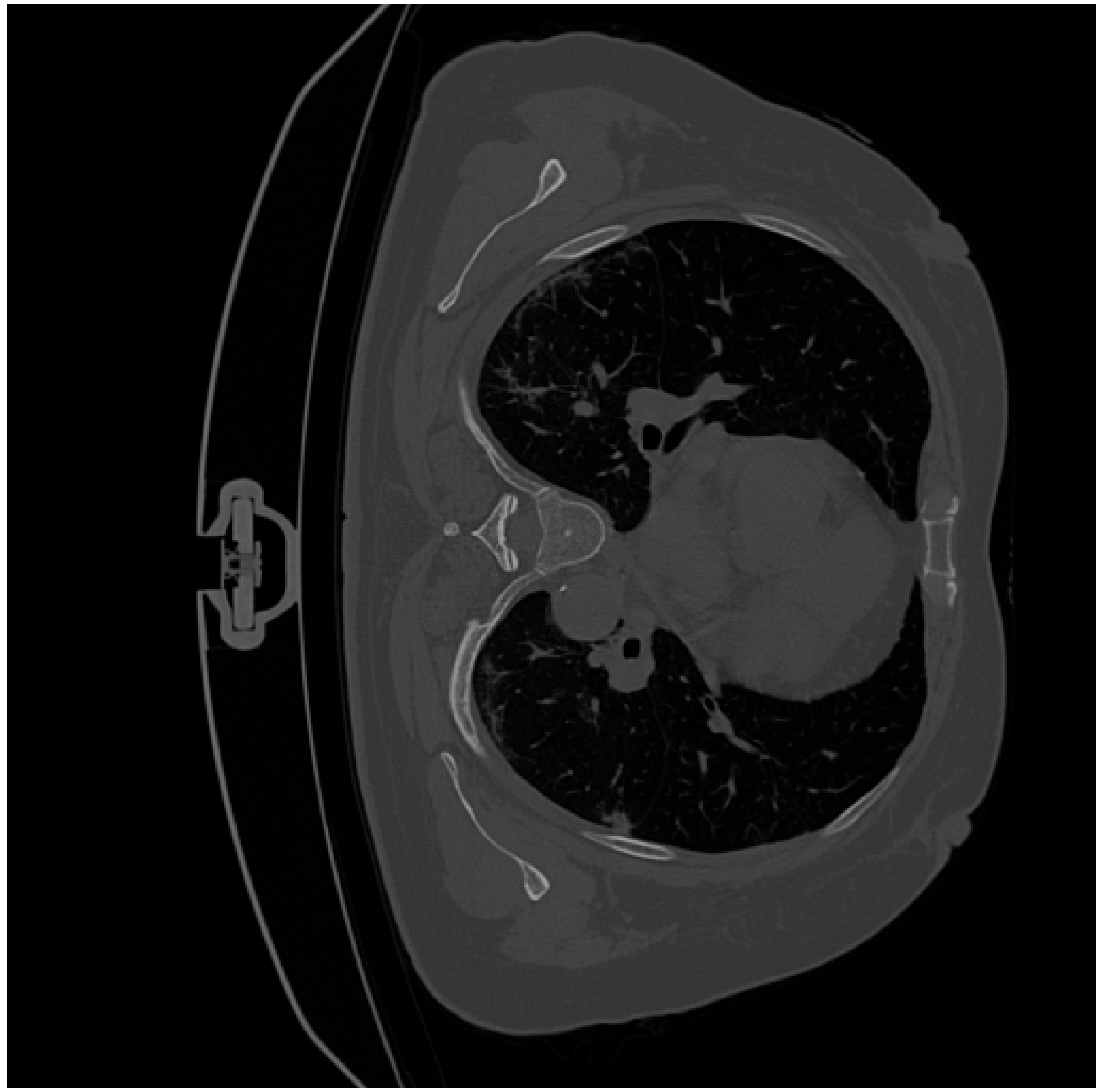


PATIENT
CT SCAN IMAGE
RESULT
HTML,CSS,JSCRIPT
2.0
UI
Internal Storage
UPLOADED FILE
UPLOADED FILE
FLASK,PYTHON
2.1
API
FINAL OUTPUT
SEGEMENTED IMAGE
RESULT
2.2
IMAGE PREPROCESSING
2.3
CONVOLUTION
2.4
MAX POOLING
2.5
PADDING
2.6
RELU ACTIVATION
2.7
TRANSPOSED CONVULTION
2.8
UPSAMPLING
2.9
CONVOLUTION
Internal Storage
SEGEMENTED IMAGE

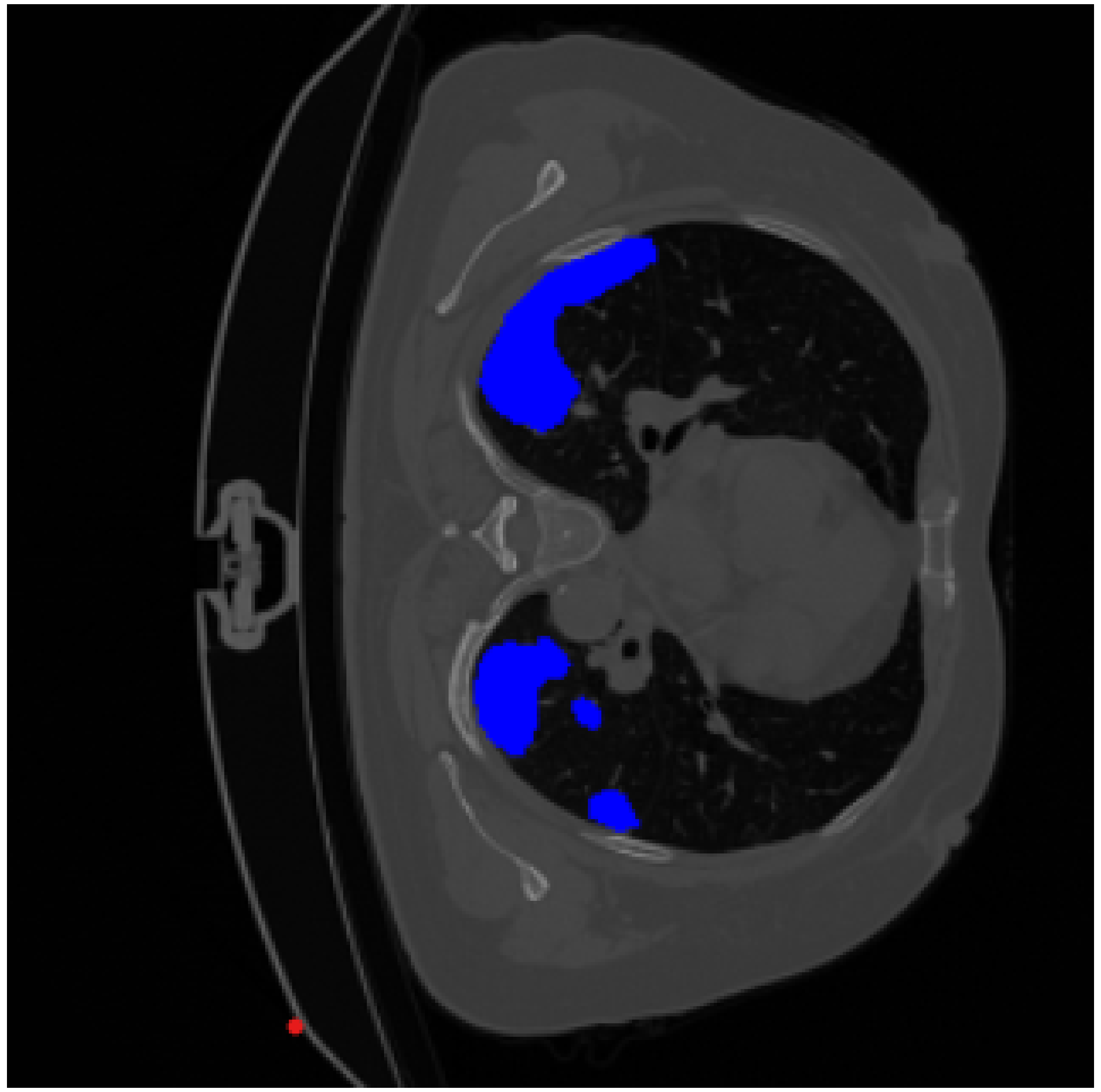


User
USER INTERFACE
Flask, Python
API
U-NET
conv 3*3 RELU
COPY AND CROP
max pool 2*2
up conv 2*2
PREDICTION
RESULT

## Input



## Output



# MODULE DETAILS

1. Design the user interface
2. Build API
3. Collection of the dataset
4. Implement UNET Model
5. Prediction

## DESIGNING USER INTERFACE

1. Design an intuitive and user-friendly interface.
2. It consists of a simple UI that is connected through an API to the model.
3. An option to add the CT Scan will be provided.
4. The result is then fetched from the API and is displayed to the user.

# BUILDING API

- The back-end API will receive the CT-scan image from the front-end and process it using UNET model.

- To build an API a web framework such as Flask (Python), Express (Node.js), or Ruby on Rails (Ruby) is required.

- When the API receives the image, it will need to preprocess the image to prepare it for input to the UNET model.

# COLLECTION & TRAINING

- Public dataset from **Ma et al.** which consists of 700 annotated COVID-19 chest CT volumes.
- 3D volume set with annotated COVID-19 infection segmentation.
- It is used by the model to provide accurate and helpful responses to users.

# IMPLEMENTING U-NET MODEL

- Segmentation :It provides fine-grained information about the image as well as the shapes and boundaries of the objects.

- Encoding layers : Extracting  features from the input image using convolution,padding,maxpooling operations.

- Decoding Layers : precise localization using transposed convolutions

## SEGMENTATION & RESULT

Receive a CT scanned image of a lung through an API to be Segmented by a Convolutional neural network.

# SYSTEM REQUIREMENTS

## Software Requirement

- Language : Python
- Supporting libraries like KERAS
- API framework : FLASK
- IDE : VS Code

## Hardware Requirement

- Processor : Intel(R) Xeon w-2255 CPU(3.7Ghz)
- RAM : 64 GB
- GPU : RTX 3080 10 GB

# COMPARISON WITH EXISTING SYSTEM

- Current Existing systems are
  - Reverse-Transcription Polymerase Chain Reaction (RT-PCR)
  - Rapid Antigen Test
  - Chest X-Ray/ CT Scan

- Compared to these systems the model will have:
  - Lower False Positives / False Negatives
  - Easily Available
  - Lesser Turnaround Time
  - Reduced Cost

# WORKPLAN

**Week 1-3: Planning and research**

- Define project requirements and objectives
- Research on CNN U-NET
- Determine feasibility of project within the given timeline and resources

**Week 4-5:Development Process**

- Collect required training datas
- Training the U-NET model with the datas and tuning hyperparameters
- Developing API for the system
- Making user interface

**Week 6-7: Testing and refinement**

- Test the model with sample data and check for accuracy
- Gather feedback from users and identify areas for improvement.

# IMPLEMENTATION

## IMPORTS

```
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import numpy as np
import cv2
from glob import glob
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Recall, Precision
from model import build_unet
from metrics import dice_loss, dice_coef, iou
```

All the necessary libraries & packages required to run the program are installed.

# CONVULTION BLOCK

```python
def conv_block(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
```

- The convolution operation, when performed on an image, extracts features from it.
- The number of filters in the Conv2D block is defined with num_filters
- Normalization ensures that all the values are normalized between 0 and 1.
- The activation function chosen is the Relu function.
- A single convolution block contains two Conv2D layers, 2 Normalization layers followed by an activation function

# ENCODER BLOCK & DECODER BLOCK

```
def encoder_block(input, num_filters):
    x = conv_block(input, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p

def decoder_block(input, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(input)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x
```

The above block contains the code for U-Net Architecture.It has 2 parts:

- Encoder Block
    - Responsible for extracting patterns in the image, such as infected regions.
    - Uses convolutions and max-pooling layers.
    - Convolutions make patterns more obscure and reduce image size.
    - Max-pooling ensures important features are retained.
- Decoder Block
    - Opposite of Encoder Block
    - Localizes the extracted features

# U-NET MODEL

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

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    b1 = conv_block(p4, 1024)

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

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

    model = Model(inputs, outputs, name="U-Net")
    return model
```

The U-Net model consists of:

- 4 Encoder blocks
- Bottle neck layer
- 4 Decoder Blocks
- Activation Function : Sigmoid Function

# MODEL SUMMARY

The summary of the model that we have constructed to perform segmentation of covid-19 infected CT scans.

```
Model: "U-Net"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 512, 512, 3  0           []
                                )]

 conv2d_16 (Conv2D)             (None, 512, 512, 64  1792        ['input_3[0][0]']
                                )

 batch_normalization_14 (BatchN  (None, 512, 512, 64  256        ['conv2d_16[0][0]']
 ormalization)                  )

 activation_14 (Activation)     (None, 512, 512, 64  0           ['batch_normalization_14[0][0]']
                                )

 max_pooling2d_6 (MaxPooling2D)  (None, 256, 256, 64  0          ['activation_14[0][0]']
                                )

 conv2d_17 (Conv2D)             (None, 256, 256, 12  73856       ['max_pooling2d_6[0][0]']
                                8)

 batch_normalization_15 (BatchN  (None, 256, 256, 12  512        ['conv2d_17[0][0]']
 ormalization)                  8)

 activation_15 (Activation)     (None, 256, 256, 12  0           ['batch_normalization_15[0][0]']
                                8)

 max_pooling2d_7 (MaxPooling2D)  (None, 128, 128, 12  0          ['activation_15[0][0]']
                                8)
```

```
 activation_18 (Activation)     (None, 128, 128, 25  0           ['batch_normalization_18[0][0]']
                                6)

 conv2d_transpose_7 (Conv2DTran  (None, 256, 256, 12  131200     ['activation_18[0][0]']
 spose)                         8)

 concatenate_7 (Concatenate)    (None, 256, 256, 25  0           ['conv2d_transpose_7[0][0]',
                                6)                                'activation_15[0][0]']

 conv2d_21 (Conv2D)             (None, 256, 256, 12  295040      ['concatenate_7[0][0]']
                                8)

 batch_normalization_19 (BatchN  (None, 256, 256, 12  512        ['conv2d_21[0][0]']
 ormalization)                  8)

 activation_19 (Activation)     (None, 256, 256, 12  0           ['batch_normalization_19[0][0]']
                                8)

 conv2d_transpose_8 (Conv2DTran  (None, 512, 512, 64  32832      ['activation_19[0][0]']
 spose)                         )

 concatenate_8 (Concatenate)    (None, 512, 512, 12  0           ['conv2d_transpose_8[0][0]',
                                8)                                'activation_14[0][0]']

 conv2d_22 (Conv2D)             (None, 512, 512, 64  73792       ['concatenate_8[0][0]']
                                )

 batch_normalization_20 (BatchN  (None, 512, 512, 64  256        ['conv2d_22[0][0]']
 ormalization)                  )

 activation_20 (Activation)     (None, 512, 512, 64  0           ['batch_normalization_20[0][0]']
                                )

 conv2d_23 (Conv2D)             (None, 512, 512, 1)  65          ['activation_20[0][0]']

==================================================================================================
Total params: 3,793,985
Trainable params: 3,791,169
Non-trainable params: 2,816
```

The input shape of the image is (512,512)
The output shape of the image is (512,512)

# TRAINING & TESTING

```python
""" Global parameters """
H = 512
W = 512

def create_dir(path):
    """ Create a directory. """
    if not os.path.exists(path):
        os.makedirs(path)

def load_data(path, split=0.1):
    images = sorted(glob(os.path.join(path, "images", "*.png")))
    masks = sorted(glob(os.path.join(path, "masks", "*.png")))


    split_size = int(len(images) * split)

    train_x, valid_x = train_test_split(images, test_size=split_size, random_state=42)
    train_y, valid_y = train_test_split(masks, test_size=split_size, random_state=42)

    train_x, test_x = train_test_split(train_x, test_size=split_size, random_state=42)
    train_y, test_y = train_test_split(train_y, test_size=split_size, random_state=42)

    return (train_x, train_y), (valid_x, valid_y), (test_x, test_y)
```

Total dataset : 700 images

Training Data : 400 images
Validation : 200 images
Testing : 100 images

# MODEL DEFINITION

```
model = build_unet((H, W, 3))
metrics = [dice_coef, iou, Recall(), Precision()]
model.compile(loss=dice_loss, optimizer=Adam(lr), metrics=metrics)

callbacks = [
    ModelCheckpoint(model_path, verbose=1, save_best_only=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, min_lr=1e-7, verbose=1),
    CSVLogger(csv_path)
]

model.fit(
    train_dataset,
    epochs=num_epochs,
    validation_data=valid_dataset,
    callbacks=callbacks
)
```

The model is build by :

- Defining it's structure
- Setting metrics
- Implementing callbacks when desired performance is achieved
- model.fit() is used to train the model with pre-defined epochs & datasets

# METRICS

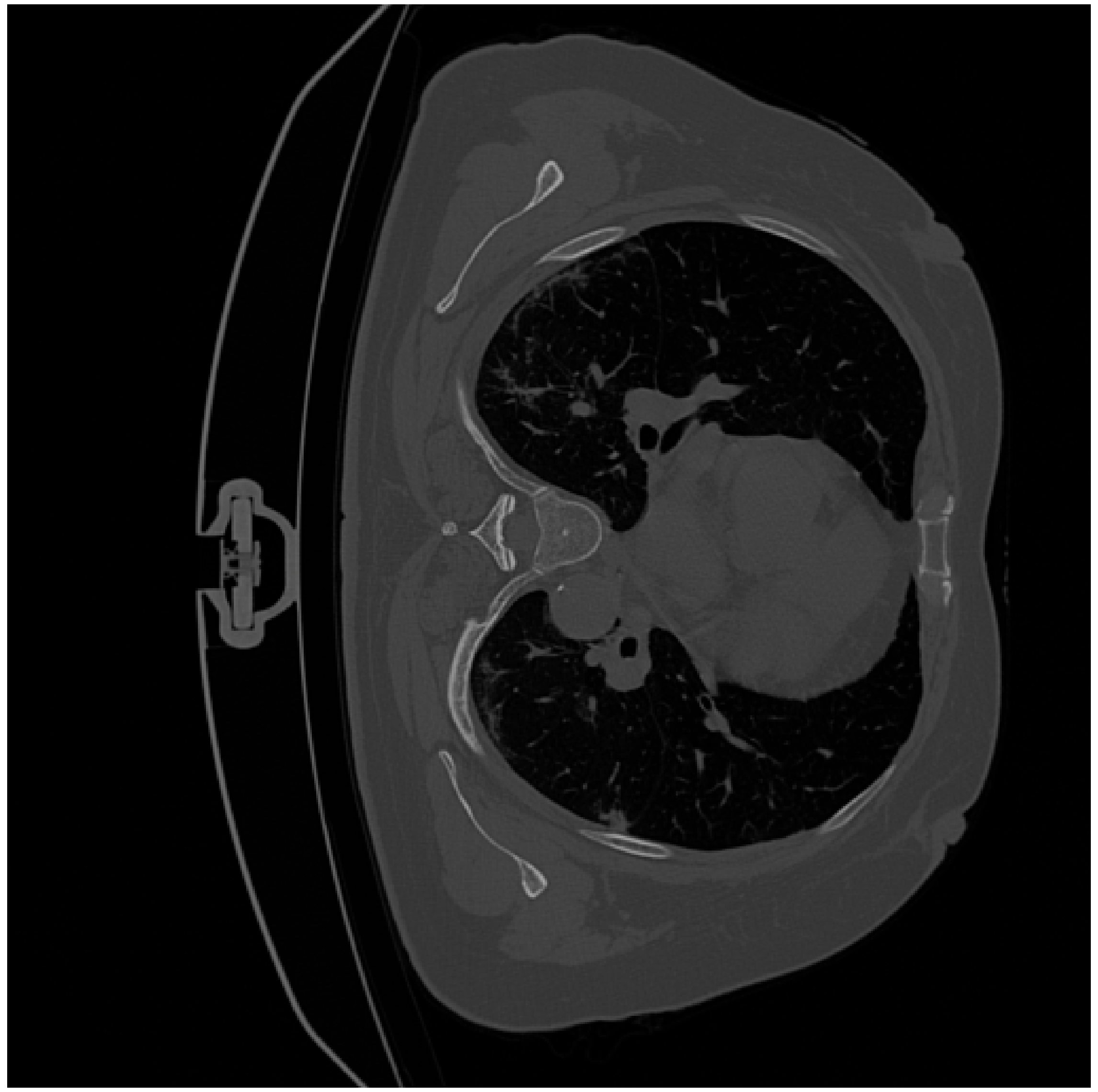
```python
def iou(y_true, y_pred):
    def f(y_true, y_pred):
        intersection = (y_true * y_pred).sum()
        union = y_true.sum() + y_pred.sum() - intersection
        x = (intersection + 1e-15) / (union + 1e-15)
        x = x.astype(np.float32)
        return x
    return tf.numpy_function(f, [y_true, y_pred], tf.float32)

smooth = 1e-15
def dice_coef(y_true, y_pred):
    y_true = tf.keras.layers.Flatten()(y_true)
    y_pred = tf.keras.layers.Flatten()(y_pred)
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

def dice_loss(y_true, y_pred):
    return 1.0 - dice_coef(y_true, y_pred)
```

The metrics that we have chosen to evaluate our model are:

- Dice coefficient : (2 * intersection) / (sum of sizes).
- Intersection over Union (IoU) : IoU = intersection / union
- Precision : Precision = True Positives / (True Positives + False Positives)
- Recall : Recall = True Positives / (True Positives + False Negatives)

# SAMPLE TRAINING DATA

CT SCAN



MASK

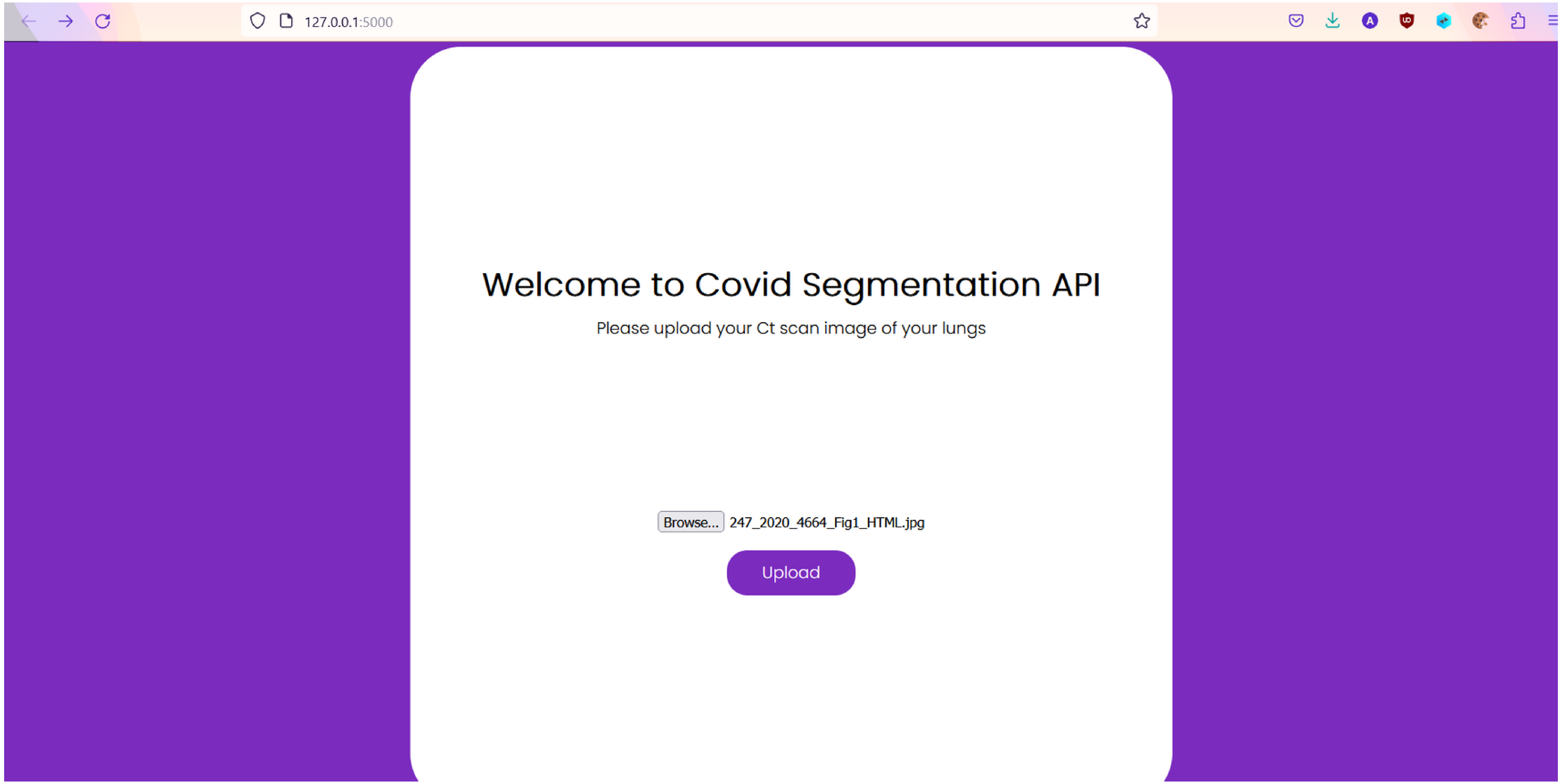
# SEGMENTATION

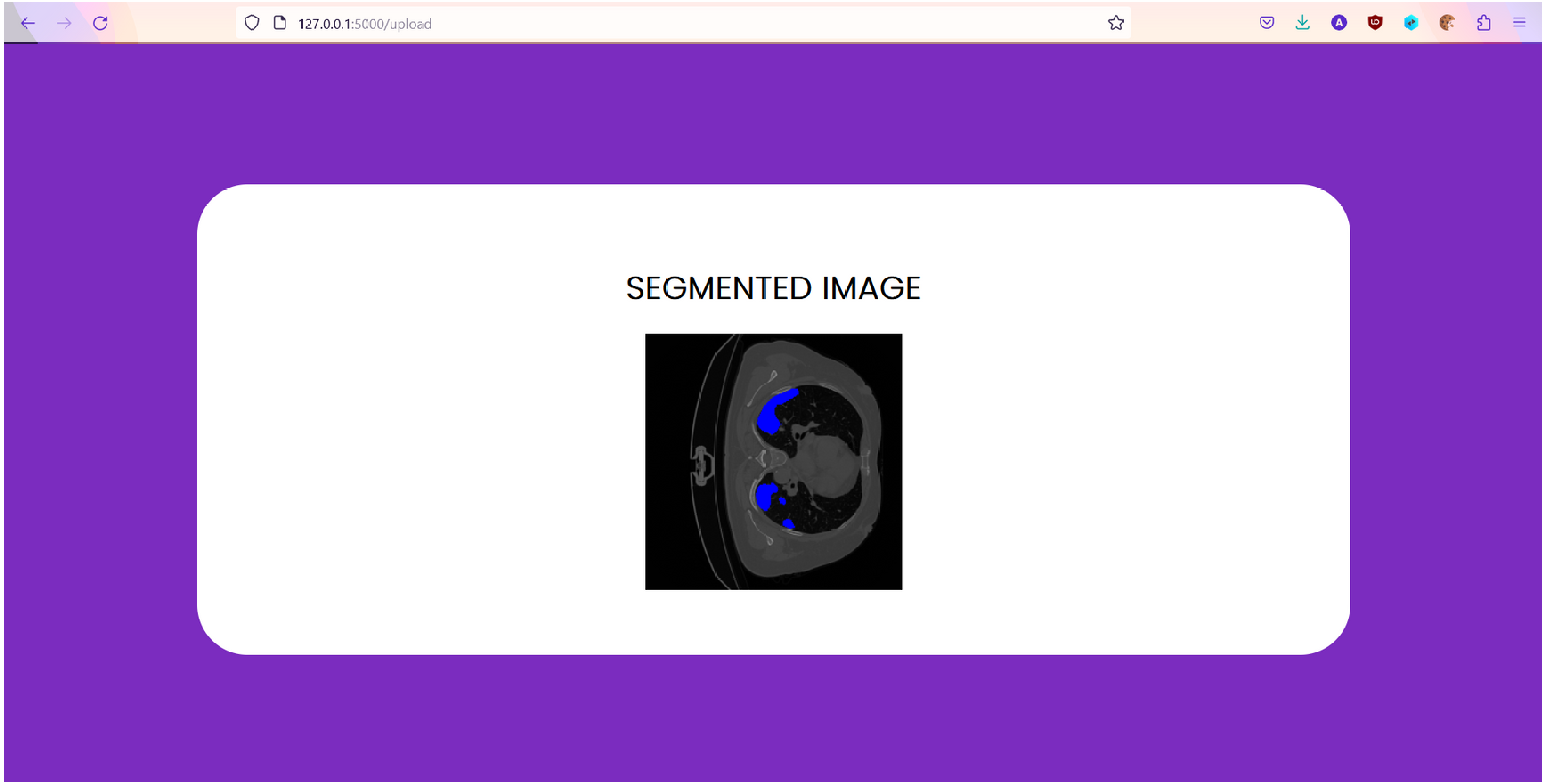
```python
""" Predicting the mask """
for x, y in tqdm(zip(test_x, test_y), total=len(test_x)):
    """ Extracing the image name. """
    image_name = x.split("/")[-1]

    """ Reading the image """
    ori_x = cv2.imread(x, cv2.IMREAD_COLOR)
    ori_x = cv2.resize(ori_x, (W, H))
    x = ori_x/255.0
    x = x.astype(np.float32)
    x = np.expand_dims(x, axis=0)

    """ Reading the mask """
    ori_y = cv2.imread(y1, cv2.IMREAD_GRAYSCALE)
    # ori_y2 = cv2.imread(y2, cv2.IMREAD_GRAYSCALE)
    # ori_y = ori_y1 + ori_y2
    ori_y = cv2.resize(ori_y, (W, H))
    ori_y = np.expand_dims(ori_y, axis=-1)  ## (512, 512, 1)
    ori_y = np.concatenate([ori_y, ori_y, ori_y], axis=-1)  ## (512, 512, 3)

    """ Predicting the mask. """
    y_pred = model.predict(x)[0] > 0.5
    y_pred = y_pred.astype(np.int32)

    """ Saving the predicted mask along with the image and GT """
    save_image_path = f"results/{image_name}"
    y_pred = np.concatenate([y_pred, y_pred, y_pred], axis=-1)

    sep_line = np.ones((H, 10, 3)) * 255

    cat_image = np.concatenate([ori_x, sep_line, ori_y, sep_line, y_pred*255], axis=1)
    cv2.imwrite(save_image_path, cat_image)
```

# API

```python
@app.route('/')


def index():
    return render_template('h.html')

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        uploaded_file = request.files['file']
        temp_file_path='D:\\downloads\\flsk\\static\\input.png'
        uploaded_file.save(temp_file_path)
        output= process_image(temp_file_path)

        return render_template('result.html',output='output.png')


app.run()
```

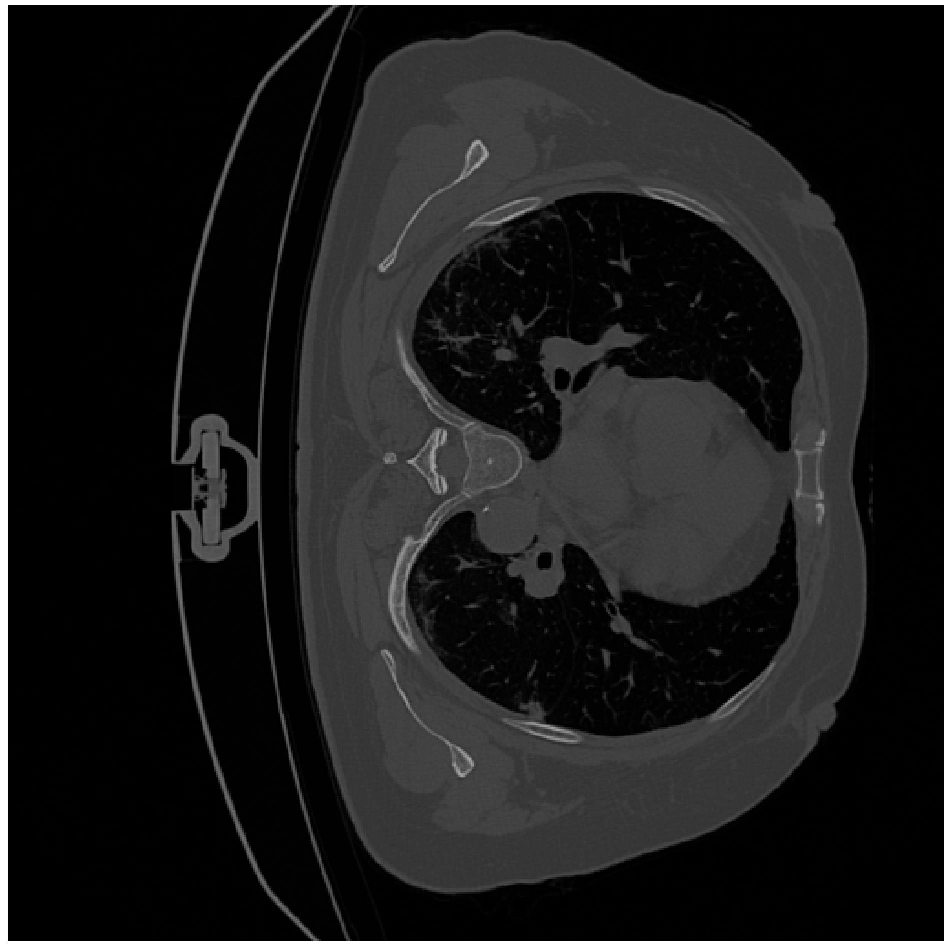
# INTERFACE


127.0.0.1:5000
Welcome to Covid Segmentation API
Please upload your Ct scan image of your lungs
Browse... 247_2020_4664_Fig1_HTML.jpg
Upload


127.0.0.1:5000/upload
SEGMENTED IMAGE

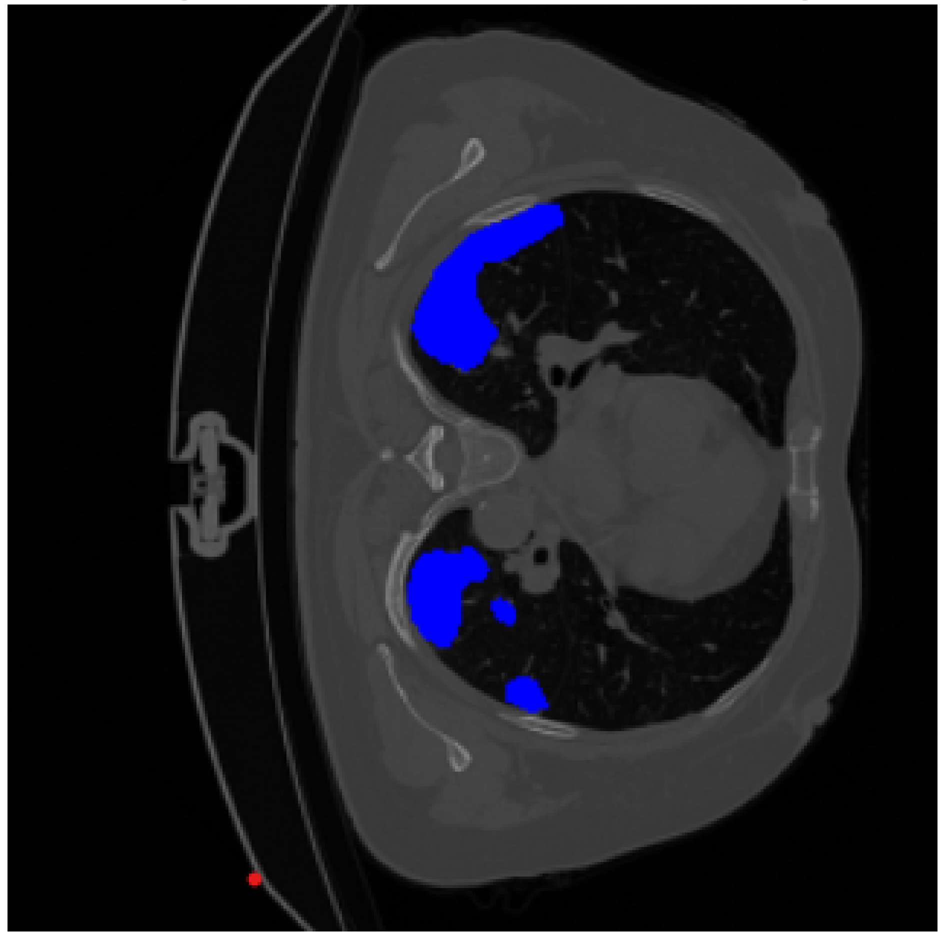
# Sample Test Data

CT-Scan

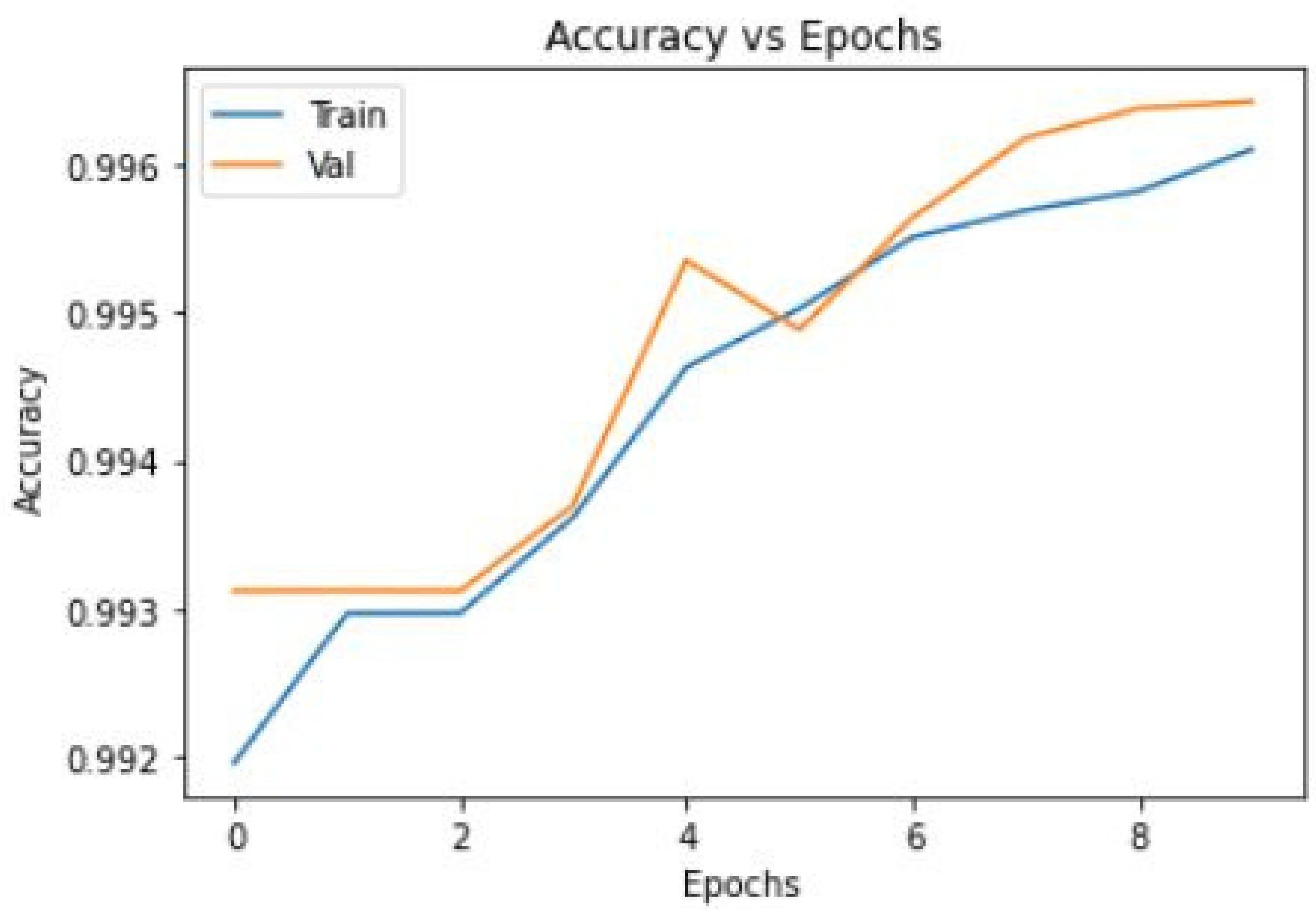


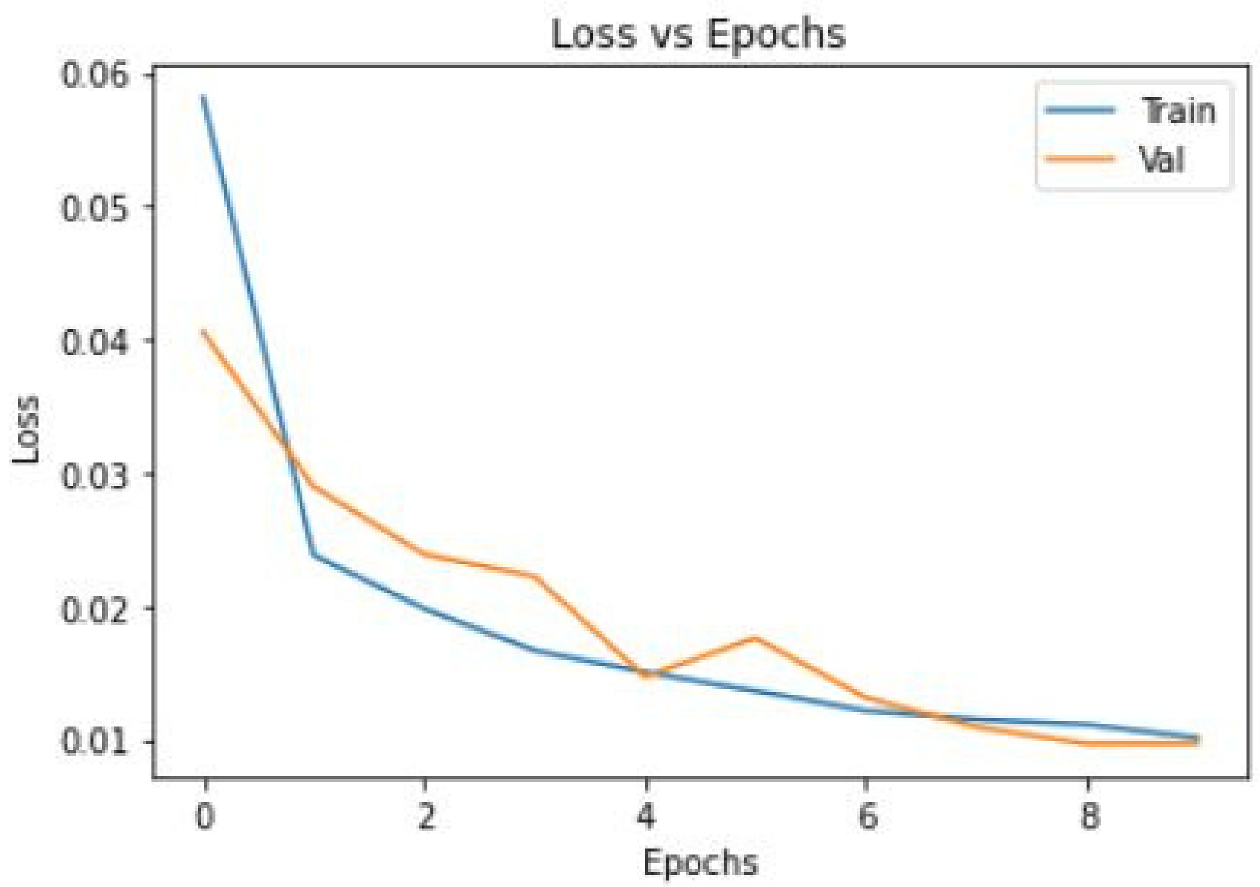
Mask

Segmented image



# PERFORMANCE




- No overfitting is observed as the graphs align with each other and rise together.
- The training accuracy curve converges at 97% at 10 epochs.
- The validation accuracy reaches 95% at 10 epochs.

# CONCLUSION

- The **'Covid-19 Segmentation API'** utilizes the U-net model to accurately segment COVID-19 infections.

- It provides a user-friendly interface for researchers and medical professionals to input COVID-19 images and obtain segmented results efficiently.

- It has the potential to be extended for segmenting other types of infections, making it valuable for medical research and diagnosis