

# LUNG X-RAY CLASSIFIER APP

Noah Vendrig

SDD MAJOR PROJECT 29/06/2022

## Table of Contents

Program.....	2
Project Log .....	0
Logbook.....	0
Gantt chart.....	2
Actual Progress .....	3
Annotated Source Code.....	0
IPO Chart.....	0
cli.py.....	0
app.py .....	1
predict.py.....	4
load_and_train.py.....	6
Testing Report.....	11
Selecting the Optimal Convolution Neural Network Model Configuration.....	11
Model Testing (training) .....	12
Graphs obtained from training results (Tensorboard):.....	12
Chosen Model: 6-conv-256-nodes-2-dense-3-dropout.....	13
Validation vs Training Loss.....	14
Validation vs Training Accuracy .....	15
Selected Model layer info: .....	15
Testing report of the software package.....	0
Quality Assessment.....	0
User Documentation.....	0
Technical Documentation .....	0
System Documentation.....	0
Algorithm description for custom logic module .....	1
Pseudocode:.....	1
Flowchart .....	1
Source code.....	2
cli.py.....	<b>Error! Bookmark not defined.</b>
app.py .....	<b>Error! Bookmark not defined.</b>
predict.py.....	<b>Error! Bookmark not defined.</b>
load_and_train.py.....	<b>Error! Bookmark not defined.</b>
Refined Data Dictionary.....	0
cli.py.....	0
app.py .....	0

Predict.py .....	2
Load_and_train.py .....	4
Structure Chart.....	0
Context Diagram .....	0
Level 1 Dataflow Diagram .....	0
Level 2 Dataflow Diagram .....	1

## Program

Submitted as zip file with lung-xray-classifier.exe inside. Refer to user documentation for usage.

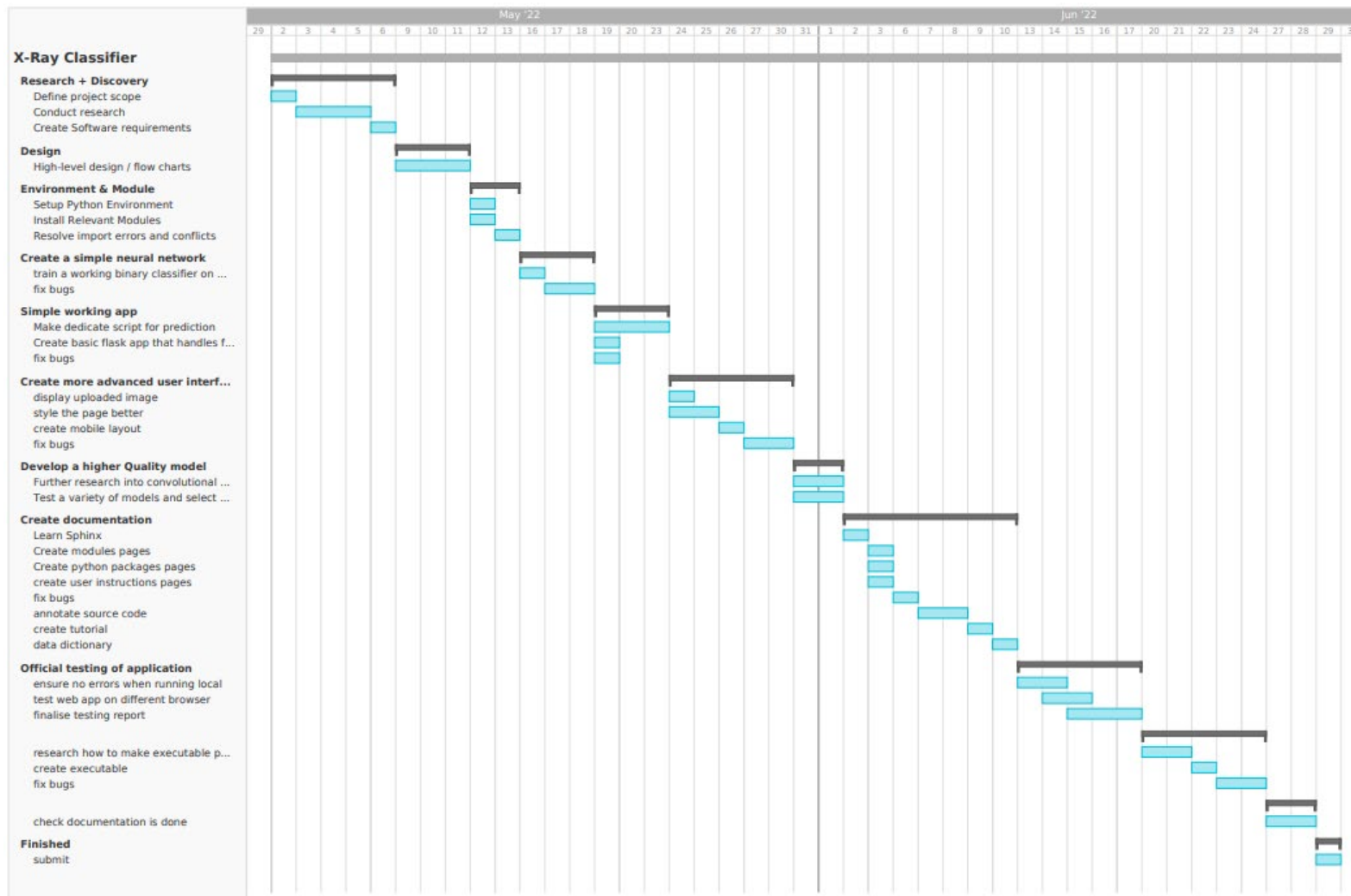
# Project Log

## Logbook

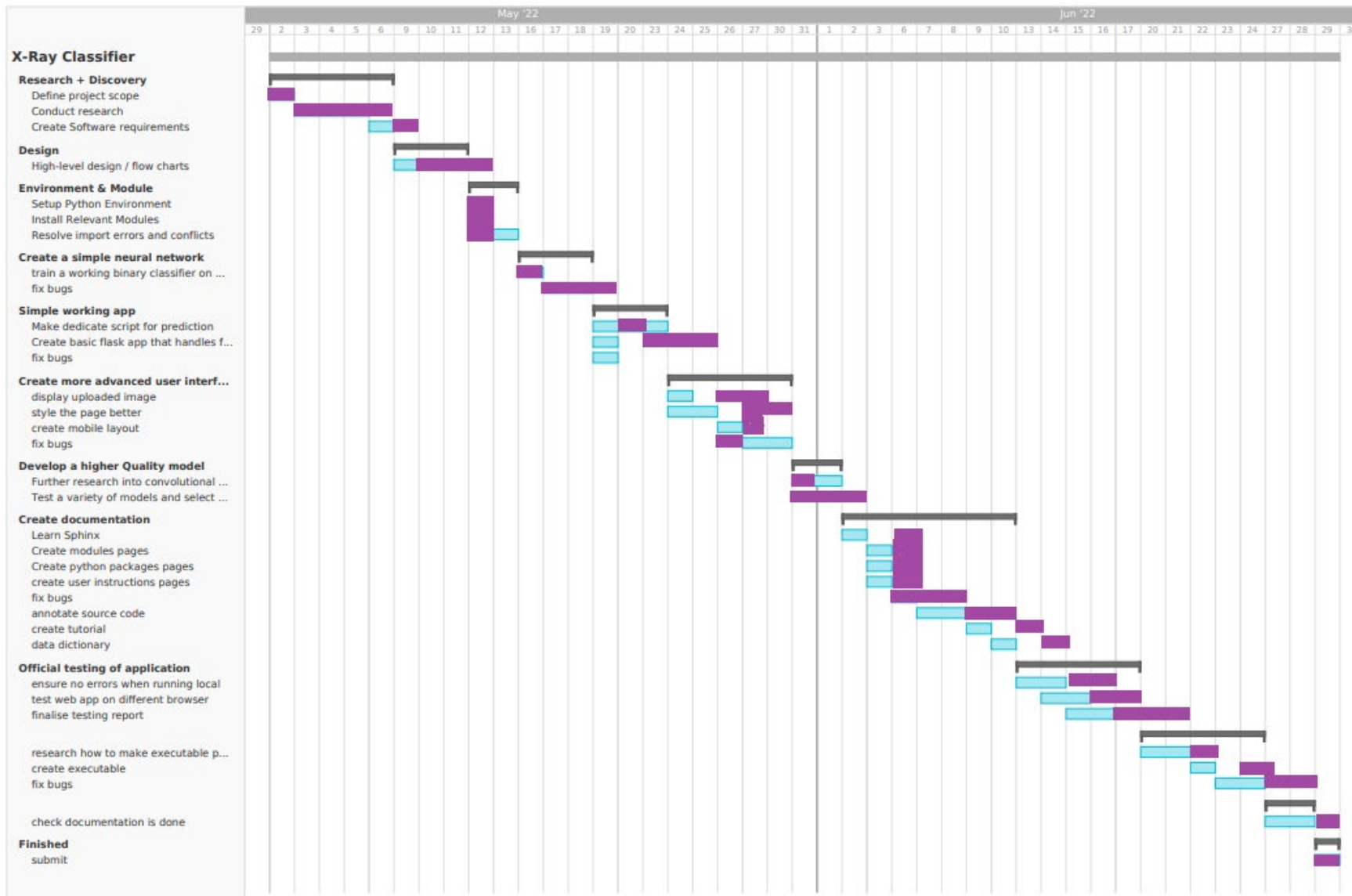
Date	Tasks Achieved	Issues/Solutions	Reflection on progress	Resources Used
2/05	Going to create an app that classifies an X-ray as either COVID or healthy	-need a binary classifier	Have a clear understanding of what needs to be done and how much needs needs to be done.	
3/05	I have decided to continue with rapid application development approach, as it is best suited to this project – solo developer, short time span. Continuing research	Use tensorflow instead of Pytorch since there are more tutorials available		Youtube tutorials + stack overflow
10/5	Well underway with the flowcharts, structure charts, DFD's and IPO chart.	No issues, may need to double check later to make sure that they're right		NESA Software Design and Development Course Specifications
12/5	Took a couple goes to figure out the installations. Currently using an Anaconda environment to install all the packages in.	Anaconda is the solution to broken installations.	A tiny bit behind but shouldn't cause too many problems	Stack overflow + googling
17/5	Have created a working binary classifier	Was having a bit of trouble with the numpy arrays created in the loop that iterates through the database. Fixed it, just needed to look at the docs to understand the usage properly	it doesn't have much accuracy and is pretty much useless but it serves as a baseline for the more extensive model to be created later.	Numpy docs
24/5	Working flask app nearly complete.	Had trouble figuring out the upload file button, found a good tutorial and adapted the code.	Was initially considering using tkinter instead but I'm more comfortable with integrating html + css webpage. I also think that a web page would make the software more accessible, since it can be run on a server for anyone to access.	Online tutorials
30/5	Finishing up with the fully functional GUI.	Had a stroke whilst fiddling with the CSS to make the uploaded image appear in a column next to	If time permits, the UI could be touched up a bit more so that it isn't just an upload button and a title.	My blood sweat and tears

		the upload buttons. Really didn't want to cooperate but it works for now.		
2/6	Optimal model complete. I'm identifying it by: 8 conv layers, 2 dense, 3 dropout and 256 nodes.	Tried to make A LOT of models so that there would be a larger range to choose from but my GPU freaked out and froze the computer. Had to limit to 6 model configurations that I thought would work best.	I'm happy with the model, no need to go through more hassle to make a better model.	Stack overflow + googling
9/6	Sphinx documentation done	User documentation looks very mint, I'm glad I went through the hassle of learning sphinx	Happy with user docs, good progress. Still a tiny bit behind the overall schedule but it's not that far behind.	Stack overflow + googling
14/6	Documentation finished!	Had to use debug mode a lot to finish data dictionary but didn't have any problems.		
20/6	Testing report almost finished	Turns out there are some layout issues with the styling in Internet Explorer and Firefox. They aren't used as much anymore (Chrome is more dominant) so I will only fix that issue if time permits at the end.	Almost done with everything, still over a week to go. No concerns with running behind schedule	NESA Software Design and Development Course Specifications
28/6	Executable has been created	Its's a massive file but not much I can do about it since it has to have all the modules (tensorflow etc) + files. It works so I'm happy.	Basically done! Might double check documentation and make sure everything is ready for submission.	

# Gantt chart



Actual Progress (drawn over the original Gantt chart in purple)



## Annotated Source Code

### IPO Chart

User Input	Process	Output
Path of file to upload (str)	Check if the filetype is allowed (png, jpeg, jpg)  Save the file locally  Read the file into a numpy array and resize the image (so it can be displayed)  Use the image path to generate a prediction label.	Display success message if filetype is allowed, else display error message to user  Display the resized image  Display the prediction label

### cli.py

```
__author__ = 'Noah Vendrig'
__license__ = 'MIT' # copy of the license available @ https://prodicus.mit-
license.org/
__version__ = '4.8'
__email__ = 'noahvendrig@gmail.com'
__github__ = "github.com/noahvendrig" # @noahvendrig
__course__ = 'Software Design and Development'
__date__ = '21/06/2022'
__description__ = 'Flask App that allows users to classify lung X-Ray images
into categories of either COVID-19 or Normal.'
__info__ = "info available at: https://github.com/noahvendrig/covid-xray-
classifier/readme.md" # some info available here
__pyver__ = '3.8.10'

from app import application

"""Script that is run from command line to launch the app"""

if __name__ == "__main__":
    print("-----")
    print("-----")
    print(f"Developed by {__author__}, {__date__}")
    print(f"This project is a {__description__}")
    print("-----")
    print("-----\n")
    print()
    print("Starting application...")

    application() # Launch app
```



app.py

```
from predict import predict # v1.0.0 by Noah Vendrig (06/2022) - predict
function from './predict.py'
import cv2 # v4.5.5 by Intel Corporation, Willow Garage, Itseez (06/2000) -
Used for image processing: converting RGB images to numpy arrays and resizing
images so that they are compatible with the model.
from waitress import serve # v2.1.2 by Zope Foundation and Contributors
(30/12/2011) - Production WSGI server for the web app.
import os
import webbrowser

UPLOAD_FOLDER = 'static/files/' # folder to store uploaded images

import socket

#from pathlib import Path
# [f.unlink() for f in Path("./static/files").glob("*") if
f.is_file()] #remove all files in the folder not needed right now

import os
from flask import Flask, flash, request, redirect, url_for, render_template #
v2.1.2 by Armin Ronacher (01/04/2010) - Allows for the creation of the GUI as
a web application hosted on local IP
from werkzeug.utils import secure_filename # v2.1.2 by Armin Ronacher
(10/12/2007) - Create WSGI server for Flask

import logging

templatesDir = './templates' # directory where the templates are stored
staticDir = './static' # directory for static files
app = Flask(__name__, template_folder=templatesDir, static_folder=staticDir) #
create the Flask app

app.secret_key = "secret key" # needed for flask sessions
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER # set the upload folder
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024 # 16 MB

log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR) # only log errors in flask app, nothing else so
that console isn't cluttered

ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg']) # will only accept the
following image types

def resize(im):
    """Resizes image to a width of 500 so that it can be displayed on the
webpage
```

```

    Args:
        im (numpy arr): Input image as an array

    Returns:
        numpy arr: Resized image
    """
    h, w, channels = im.shape # get the height, width and channels of the
image
    max_w = 500 # maximum width
    ratio = h/w # ratio of height to width

    resized_h, resized_w = int(round(max_w*ratio)), max_w
    dims = (resized_w, resized_h) # get dimensions that retain image's aspect
ratio but have maximum width of 500

    resized_im = cv2.resize(im, dims) # resize image to specified dimensions
    return resized_im

def allowed_file(filename):
    """Check if file is allowed to be uploaded (filetypes is either png, jpg
or jpeg)

    Args:
        filename (str): name of the file to be uploaded (e.g. myimage.jpg)

    Returns:
        bool: Whether the file is allowed to be uploaded or not
    """
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

@app.route('/')
def upload_form():
    """Displays the main page with the file upload form
    """
    return render_template('index.html', )

@app.route('/', methods=['POST'])
def upload_image():
    """Upload the image selected by the user

    Returns:
        str: filename of the image uploaded
        str: predictions of the image uploaded
    """
    if 'file' not in request.files:
        flash('No file part') # display error message

```

```

        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        flash('No image selected for uploading') # display error message
        return redirect(request.url)

    if file and allowed_file(file.filename): # if the file is allowed and has
been uploaded
        filename = secure_filename(file.filename)
        path = os.path.join(app.config['UPLOAD_FOLDER'], filename) # save the
file to the upload folder
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        im = cv2.imread(path) # read image from disk into array
        os.remove(path) #delete the old file
        im = resize(im) # resize the image to be displayed
        cv2.imwrite(path, im) # save the image locally
        print(f"Image Saved To {path}")
        prediction = predict([path]) # predict the image
        print(f"Prediction: {prediction}")
        return render_template('index.html', filename=filename,
prediction=prediction) # display the image prediction and display the resized
image to the user.

    else:
        flash('Allowed image types are -> png, jpg, jpeg, gif') # display
error message if the filetype is not allowed
        return redirect(request.url)

@app.route('/display/<filename>')
def display_image(filename):

    """Display the uploaded image

    Args:
        filename (str): Filename of uploaded file
    """
    return redirect(url_for('static', filename='files/' + filename), code=301)

@app.route('/docs')
@app.route('/')
def docs():
    """Redirect the user to the documentation page (/docs)
    """
    return redirect(url_for('static', filename='build/html/index.html'),
code=302)

def application():
    """Host the App on local IP Address (port 5000 by default)

```

```

"""
port = 5000 # my favourite port
hostname=socket.gethostname()
ip_addr=socket.gethostbyname(hostname) # get ip of the user's machine

webbrowser.open(f"http://{ip_addr}:{port}", new=1) # opens the app in a
new browser window
print(f"App Running at: {ip_addr}:{port}") # indicate where the app is
hosted
try:
    serve(app, host="0.0.0.0", port=5000) # for production
    # app.run(debug=True, host="0.0.0.0") # only for development
except Exception as e:
    print(e) # oh no there was an error!!

# application() # not needed since running from cli.py

```

predict.py

```

__version__ = '1.0.0'

# Required modules
import numpy as np # v1.23.0 by Travis Oliphant (2006) - Used to work with
arrays generated by OpenCV, Tensorflow also requires that data be passed in as
numpy arrays.
import cv2 # v4.5.5 by Intel Corporation, Willow Garage, Itseez (06/2000) -
Used for image processing: converting RGB images to numpy arrays and resizing
images so that they are compatible with the model.
import tensorflow as tf # v2.9.0 by Google Brain Team (09/11/2015) - Used by
Keras to compile, train and evaluate the CNN model.
import pickle
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # tensorflow INFO, WARNING, and ERROR
messages are not printed

class Classifier:
    """Classifier Class used to evaluate lung-xray images into either covid or
normal
    """
    def __init__(self, model_path, img_paths):
        """Initialise the Classifier class' attributes

        Args:
            model_path (str): Path to the CNN Model created in Tensorflow
            img_paths (list): list of paths to the image to be evaluated by
the model. Currently the list is only of one element
        """

```

```

        self.model = tf.keras.models.load_model(model_path) # Loads a compiled
Keras model instance
        self.labels = pickle.loads(open('labels.pickle', "rb").read()) #
deserialise the pickle object so that the labels can be used in script
        for img in img_paths: # iterate through list of img paths
            self.Classify(img) # call the classify function for every image
inputted (currently only one image)

def ImageToArray(self, file):
    """Converts input image into a numpy array

    Args:
        file (str): path to the image to be converted to numpy array

    Returns:
        img_arr (numpy arr): numpy array of the image
    """
    # reads an image in the BGR format
    try:
        img_arr = cv2.imread(file) # returns a 3d array
        img_arr = cv2.cvtColor(img_arr, cv2.COLOR_BGR2RGB) # convert BGR
-> RGB
        return img_arr
    except Exception as e:
        print("INCORRECT FILE PATH") # incase there are issues with the
file path
        os._exit(0) # exit the program but no error message to clog
terminal when debuggig

def Classify(self, img):
    #
    """Generates a prediction for the submitted image

    Args:
        img (str): Path to the image to be classified
    """
    # actual_label = img.split("/")[-1]
    img_arr = self.ImageToArray(img) # convert to numpy array
    new_img = cv2.resize(img_arr, (150,150)) # resize image so that it can
be used for training
    # new_img = (new_img[:, :, 2] / 255).astype('float32')
    new_img = (new_img / 255).astype('float32') # normalise values to range
from 0 to 1 so computation is easier. convert to float
    new_img = tf.expand_dims(new_img, 0) # model expects a dataset so we
expand_dims. shape goes from (150,150,3) --> TensorShape([1,150,150,3])

    try:

```

```

        self.prediction = self.model.predict(new_img) # returns list with
probabilities of being each label
        # print(self.prediction[0])
        self.prediction = np.argmax(self.prediction, axis=None, out=None)
# convert from categorical back to index

        self.prediction = self.labels[self.prediction] # get the string
from the index so it can be displayed

    except:
        return "Error in Model Prediction", os._exit(0) # display error
message

def predict(input):
    """This is the main function that drives the generation of the prediction

    Args:
        input (list): list of paths to the image to be evaluated by the model.
Currently the list is only of one element but can be further expanded to
evaluate multiple images

    Returns:
        str: 'prediction' attribute of the Classifier Instance, which is the
model prediction of either 'Normal' or 'Covid'
    """
    # c = Classifier("6-conv-128-nodes-2-dense-1654694547.model",
["./dataset/normal/images/Normal-10000.png" , "./dataset/covid/images/COVID-
3615.png"])

    # res = Classifier("6-conv-128-nodes-2-dense-1655171754.model", input)
    res = Classifier("6-conv-128-nodes-2-dense-1656432632.model", input) #
create instance of the Classifier class that uses the previously trained model

    return res.prediction # return the prediction made by the model to the UI

# print(predict(["./dataset/normal/images/Normal-10000.png"]))

```

## load\_and\_train.py

```

import tensorflow as tf # v2.9.0 by Google Brain Team (09/11/2015) - Used by
Keras to compile, train and evaluate the CNN model.
import numpy as np # v1.23.0 by Travis Oliphant (2006) - Used to work with
arrays generated by OpenCV, Tensorflow also requires that data be passed in as
numpy arrays.
import cv2 # v4.5.5 by Intel Corporation, Willow Garage, Itseez (06/2000) -
Used for image processing: converting RGB images to numpy arrays and resizing
images so that they are compatible with the model.

```

```

from keras.models import Sequential # v2.9.0 by Francois Chollet (27/03/2015)
- Keras allows for the construction and training of the convolutional neural
network model used by the software. It also allows for the model to be
evaluated and tested to prevent overfitting.
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from keras.utils import np_utils

from sklearn.model_selection import train_test_split # v1.1.1 by David
Cournapeau (06/2007) - Responsible for splitting the dataset into portions for
training and testing (Model currently uses a split of 80% train, 20% test)

import os
import time
import pickle

if tf.test.gpu_device_name():# check if gpu device is available to be used
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
    is_gpu = len(tf.config.list_physical_devices('GPU')) > 0
    print(is_gpu)
else:
    print("Please install GPU version of TF")

class ModelGenerator:
    def __init__(self, Labels, path):

        """_summary_

        Args:
            labels (_type_): _description_
            path (_type_): _description_
        """
        self.labels = Labels
        print(Labels)
        self.X = []
        self.y = []
        self.path = path

        self.X_train = []
        self.X_test = []
        self.y_train = []
        self.y_test = []
        self.model = Sequential()

        self.MODEL_NAME = ""

    def CheckGPU(self):
        """Check that there is a GPU device for Tensorflow to use
        """

```

```

if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
    print("Please install GPU version of TF")

if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
    is_gpu = len(tf.config.list_physical_devices('GPU')) > 0
    print(is_gpu)
else:
    print("Please install GPU version of TF")

def ImageToArray(self, file):
    """Converts an image from RGB to numpy array so that it can be
processed.

Args:
    file (str): Path to the file to be converted to an array

Returns:
    numpy arr: Numpy array of the image
    """
    img_arr = cv2.imread(file) # reads an image in the BGR format
    img_arr = cv2.cvtColor(img_arr, cv2.COLOR_BGR2RGB) # BGR -> RGB
    return img_arr

def ProcessImages(self):
    """ Generate the array of all the images in the dataset and t array
with the corresponding labels """

    for label in self.labels: # iterate through the dataset
        for filename in os.listdir(self.path+label+"/images/")[ :3615]: #
only 3615 files available in covid dataset, need to limit so that for loop
doesnt try and iterate further since normal dataset has 10,000 imgs
            # divide by 255 to normalise the data
            file = str(f"{self.path}{label}/images/{filename}")
            arr = self.ImageToArray(file)
            # having issues with appending np array (it clears the array
each time, so we conver to python list first and then make the whole thing a
np array later
            arr = arr[:, :2, :2].tolist() # Reduce size by factor of 2,
convert to list so we can append and also shrink resolution to 150x150
            label_index = self.labels.index(label) # get the index of the
Label

            self.X.append(arr) # add to array
            self.y = np.append(self.y, label_index)
            print(f"DONE: {label}")

```



```

def ProcessArrays(self):
    """ Process the training arrays to be compatible with the model """
    self.X = np.array(self.X) # convert to np array
    self.X = self.X/255 # normalise values to range from 0 to 1 so
computation is easier. convert to float
    self.X = self.X.astype('float32') # convert to float

    self.y = np.array(self.y, dtype='int8') # convert to np array of int
    # y = tf.one_hot(y, 3)
    # one hot encode outputs
    self.y = np_utils.to_categorical(self.y) # convert labels vector to
matrix of binary values

def SplitDataset(self):
    """Splits the dataset into training and testing sets.
    """
    self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(self.X, self.y, test_size=0.2) # split dataset into 80%
train, 20% test since we have small-ish dataset

def BuildModel(self):
    """Add layers to the model
    """
    # print(self.Labels)
    self.model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same', input_shape=(150, 150, 3))) #
shape = X.shape[1:] # Add convolution layer
    self.model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
    self.model.add(MaxPooling2D((2, 2))) # Add max pooling layer
    self.model.add(Dropout(0.2))

    self.model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same')) # Add convolution layers
    self.model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
    self.model.add(MaxPooling2D((2, 2))) # Add max pooling layer
    self.model.add(Dropout(0.2))

    self.model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same')) # Add convolution layers
    self.model.add(Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
    self.model.add(MaxPooling2D((2, 2))) # Add max pooling layer
    self.model.add(Dropout(0.2))

    self.model.add(Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same')) # Add convolution layers

```

```

        self.model.add(Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_uniform', padding='same'))
        self.model.add(MaxPooling2D((2, 2))) # Add max pooling layer
        self.model.add(Dropout(0.2))

        # example output part of the model
        self.model.add(Flatten()) # transform pooled feature map that is
generated in the pooling step into a 1D vector
        self.model.add(Dense(128, activation='relu',
kernel_initializer='he_uniform')) # dense layer
        self.model.add(Dense(len(self.labels), activation='softmax')) # final
layer dense 2 since we have 2 labels

        # compile model
        self.model.compile( # compile the model
            loss='binary_crossentropy', # USE SPARSE if WE ARE
USING THE ACTUAL LABEL NUMBERS E.G 1,2 BUT WE ALREADY CONVERTED TO CATEGORICAL
            metrics=['accuracy'],
            optimizer='adam'
        )
        print(self.model.summary()) # print summary of model

    def TrainModel(self):
        """ Trains the model using the dataset provided: epochs=20, batch
size=32, validation split=0.2
        """
        # print(self.X_train.shape)
        # print(self.y_train.shape)
        log_dir = f"logs/fit/6-conv-256-nodes-2-dense-3-dropout-
{int(time.time())}"
        tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1) # for logging on tensorboard (get metrics for every model)

        history = self.model.fit( # train the model
            self.X_train,
            self.y_train,
            epochs=8, # determined that 8 epochs was necessary for optimal
model
            batch_size=32,
            validation_data=(self.X_test, self.y_test),
            callbacks=[tensorboard_callback]
        )
        print(history)

    def EvaluateModel(self):
        """Get loss value & metrics values for the model in test mode.
        """

```

```

        evaluation = self.model.evaluate(self.X_test, self.y_test, verbose=0)
#evaluate model
        print(evaluation)

    def SaveModel(self):
        """Save the model so that it can be accessed later without having to
retrain each time
        """

        self.model.save(f"./6-conv-128-nodes-2-dense-
{int(time.time())}.model") # save model to file
        f = open('labels.pickle', "wb")
        f.write(pickle.dumps(self.labels)) # serialises the labels so that it
can be stored on disk and later deserialised for use.
        f.close()

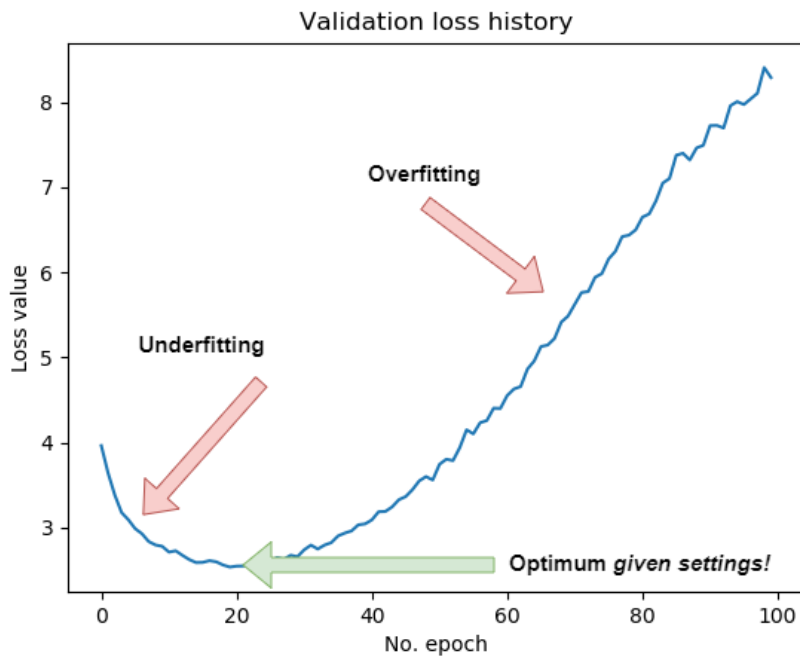
if __name__ == '__main__':
    gen = ModelGenerator(['normal', 'covid'], "./dataset/") # create
modelgenerator instance and begin the data processing + model creation
    gen.ProcessImages()
    gen.ProcessArrays()
    gen.SplitDataset()
    gen.BuildModel()
    gen.TrainModel()
    gen.EvaluateModel()
    gen.SaveModel()

```

## Testing Report

### Selecting the Optimal Convolution Neural Network Model Configuration

Goal	To create a model with the lowest validation loss and highest validation accuracy, whilst ensuring that the model hasn't over/under fit. Below is a graph which illustrates the indications of over/under fitting on a Validation Loss vs Epoch graph.
Met?	Yes, the optimal model configuration has been identified, and it has been trained to prevent overfitting/underfitting.



## Model Testing (training)

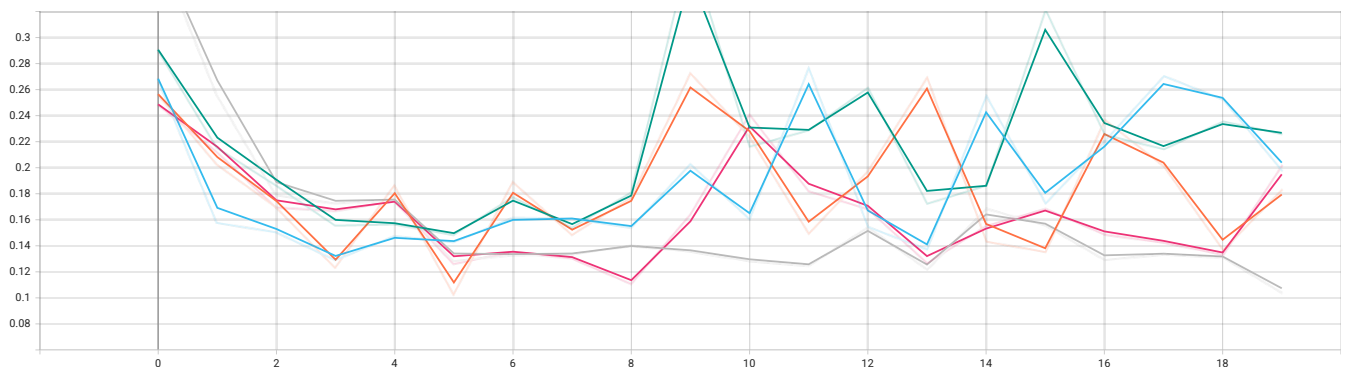
All models:

name	val loss (% after 20 epochs)	val acc (after 20 epochs)
6-conv-128-nodes-2-dense	0.1977	96.47
8-conv-128-nodes-2-dense	0.2259	95.09
6-conv-128-nodes-2-dense-1-dropout	0.1837	95.85
8-conv-256-nodes-2-dense	0.2024	96.06
8-conv-256-nodes-2-dense-3-dropout	0.1043	96.47

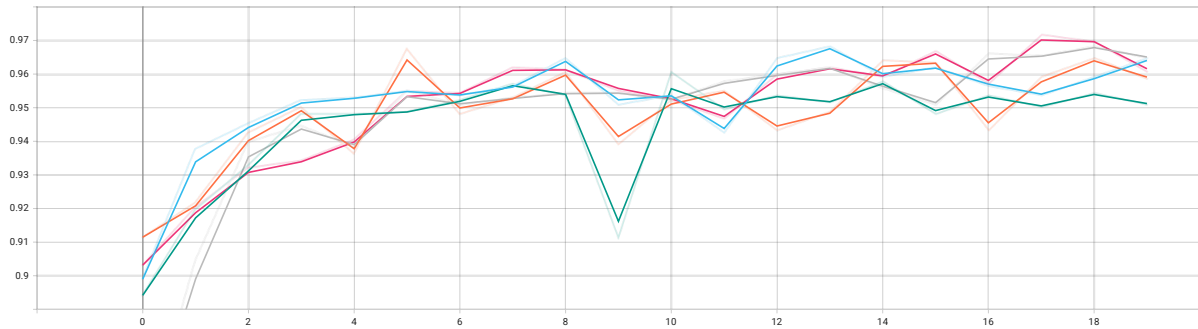
Graphs obtained from training results (Tensorboard):

As can be seen from the Validation loss line graph of all models (and is emphasised in the graph with extra smoothing), overfitting appears to be present after the 8<sup>th</sup> epoch, and underfitting is present before the 8<sup>th</sup> epoch. Therefore the model will be trained to 8 epochs to be optimal.

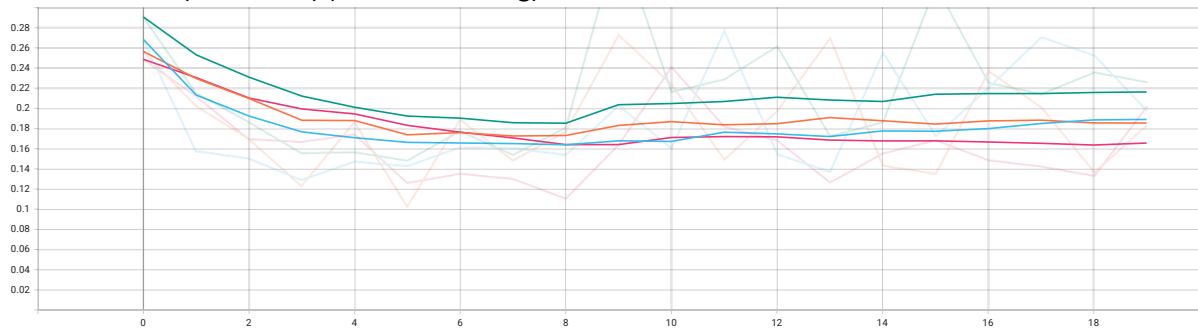
Validation Loss (all models)



### Validation Accuracy (all models)



### Validation Loss (all models) (extra smoothing)



All models:

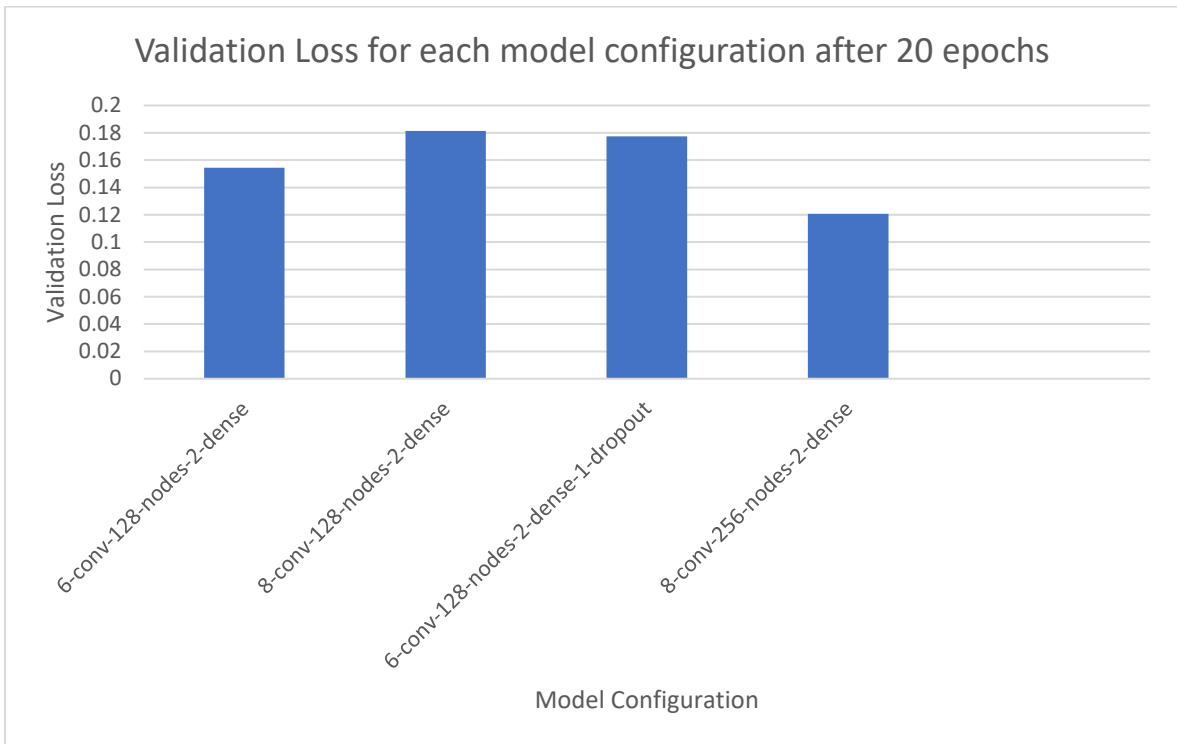
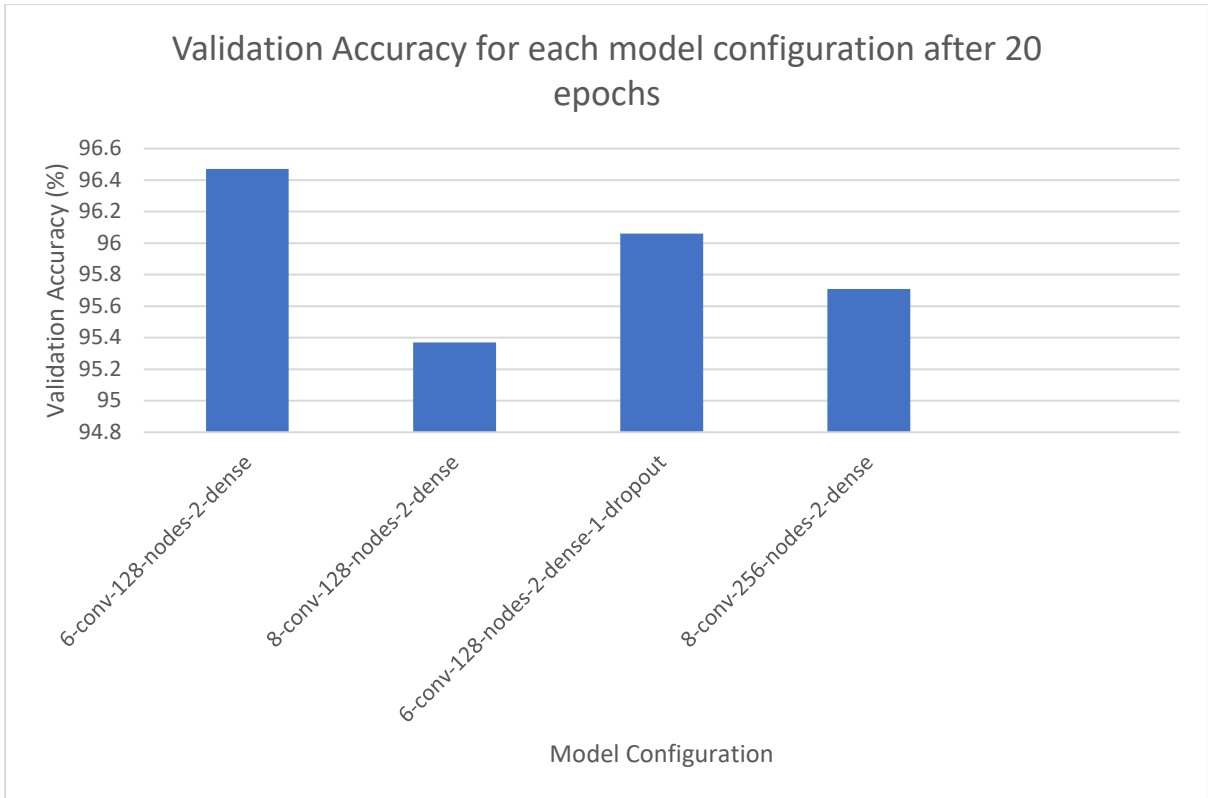
name	val loss (% after 8 epochs)	val acc (after 8 epochs)
6-conv-128-nodes-2-dense	0.1977	96.47
8-conv-128-nodes-2-dense	0.2259	95.09
6-conv-128-nodes-2-dense-1-dropout	0.1837	95.85
8-conv-256-nodes-2-dense	0.2024	96.06
6-conv-256-nodes-2-dense-3-dropout	0.1043	96.47

Chosen Model: 6-conv-256-nodes-2-dense-3-dropout

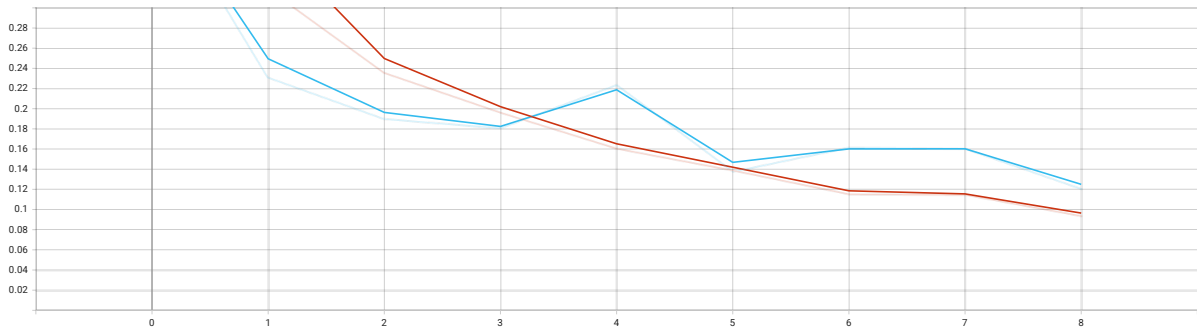
This model has been selected as it has the highest validation accuracy and lowest validation loss over 20 epochs without overfitting.

Below is information on the layers applied to the model. Below are the training/validation loss/accuracy graphs for the chosen model. As can be seen, no overfitting is present since it has been trained to only 8 epochs.

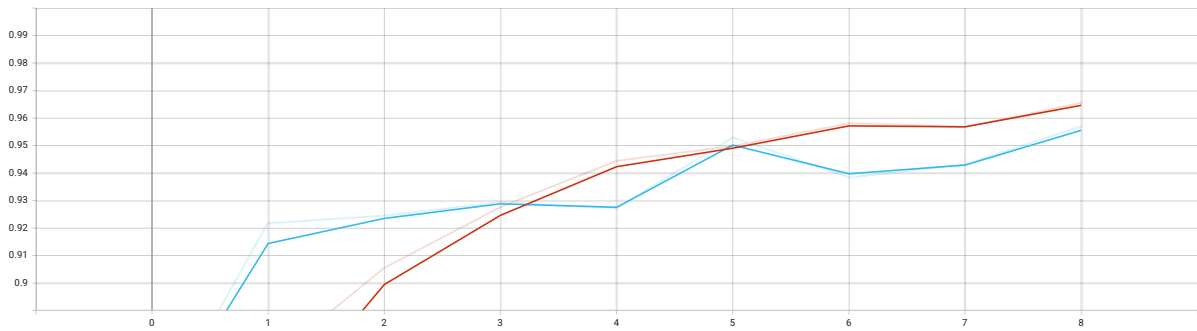
After retraining to 8 epochs, there is a slight dip in validation accuracy (it is now ranked closely 3/4) however this is not of large concern since the validation loss is significantly lower than the others, meaning that this model configuration has much higher confidence.



Validation vs Training Loss  
 (blue=train, red=validation)



Validation vs Training Accuracy  
(blue=train, red=validation)



Selected Model layer info:

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 32)	896
conv2d_1 (Conv2D)	(None, 150, 150, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 75, 75, 32)	0
dropout (Dropout)	(None, 75, 75, 32)	0
conv2d_2 (Conv2D)	(None, 75, 75, 64)	18496
conv2d_3 (Conv2D)	(None, 75, 75, 64)	36928
max_pooling2d_1 (MaxPooling (2D))	(None, 37, 37, 64)	0
dropout_1 (Dropout)	(None, 37, 37, 64)	0
conv2d_4 (Conv2D)	(None, 37, 37, 128)	73856
conv2d_5 (Conv2D)	(None, 37, 37, 128)	147584

max_pooling2d_2 (MaxPooling 2D)	(None, 18, 18, 128)	0
dropout_2 (Dropout)	(None, 18, 18, 128)	0
conv2d_6 (Conv2D)	(None, 18, 18, 256)	295168
conv2d_7 (Conv2D)	(None, 18, 18, 256)	590080
max_pooling2d_3 (MaxPooling 2D)	(None, 9, 9, 256)	0
dropout_3 (Dropout)	(None, 9, 9, 256)	0
flatten (Flatten)	(None, 20736)	0
dense (Dense)	(None, 128)	2654336
dense_1 (Dense)	(None, 2)	258
=====	=====	=====
Total params: 3,826,850		
Trainable params: 3,826,850		
Non-trainable params: 0		



## Testing report of the software package

### Test Report

**Item Being Tested:** Lung X-Ray Classification App

**Date:** 17/6/22

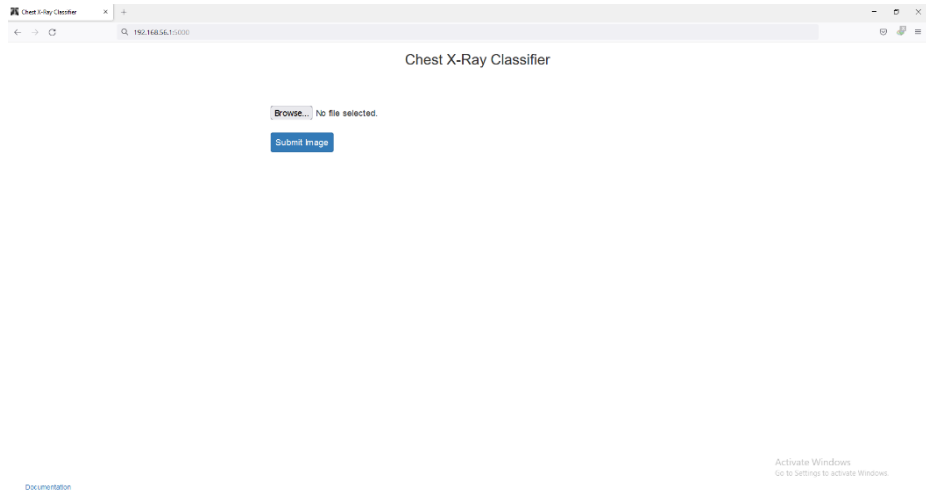
**Tester:** Noah Vendrig

**Type of test:** Black box test

**Appendix 1:** Test actions and expected results

### Interpretation of Test


Though all the tests passed, there are some issues with the software that I'd like to note:

Observed Issue	Possible Solution	Evidence (if any)
<p>In Internet Explorer and Firefox, the uploaded image is not displayed in the corner, and it looks out of place from a UI perspective. Chrome and Edge display well.</p>	<p>Fix styling of the webpage</p>	<p>Firefox:</p> 

Chest X-Ray Classifier

No file selected.

Prediction: covid



[Documentation](#)

Activate Windows  
Go to Settings to activate Windows.

### Internet Explorer:

Chest X-Ray Classifier

[Documentation](#)

Activate Windows  
Go to Settings to activate Windows.

### Chest X-Ray Classifier

Browse...

Submit Image

Prediction: covid



[Documentation](#)

Activate Windows  
Go to Settings to activate Windows.

### Chrome:



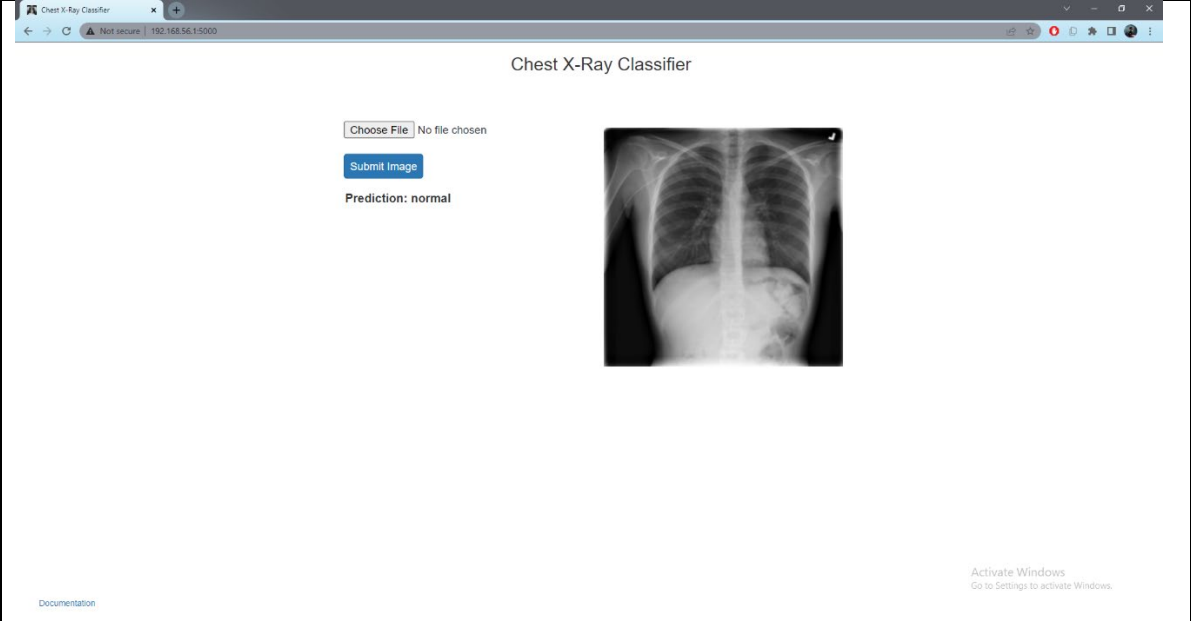
### Chest X-Ray Classifier

Choose File No file chosen

Submit Image

[Documentation](#)

Activate Windows  
Go to Settings to activate Windows.

		 <p>Edge:</p>
--	--	---

Chest X-Ray Classifier

Choose File No file chosen

Submit Image

Documentation


Activate Windows  
Go to Settings to activate Windows.

Chest X-Ray Classifier

Choose File No file chosen

Submit Image

Prediction: covid



Documentation

Activate Windows  
Go to Settings to activate Windows.

There should be a loading bar or something similar to indicate to the user that their prediction is generating. If they are impatient, they may just assume that the program is not working and close it.	Shorten loading times or add loading bar	

Appendix 1. Test actions and expected results

Action	Expected Result	Actual Result	Pass/Fail	Comment
IP is entered into web page	Web app is displayed	Web app is displayed	Pass	
Access docs through IP/docs	Routed to docs site	Routed to docs site	Pass	
Image is uploaded through upload form	File name is displayed	File name is displayed	Pass	
Image is submitted	Image + prediction is displayed	Image + prediction is displayed	Pass	
Upload another image	New filename is displayed	New File name is displayed	Pass	

## Quality Assessment

Standard	Does it satisfy the requirement?
Changeability	Yes, the app can be used as exe, the scripts can be adapted to run on a server, and there is a mobile view supported. This means that the app can be expanded very easily.
Usability	The app is very usable, as it has a simple UI and the user docs are within reach of the user once opening the app. This means that even if they are unfamiliar with the software they can learn how to use it from the docs
Compatibility	Partly, the app works on different browsers however the styling is not preserved between all. For OS compatibility, it has only been tested on Windows

Seeing as the software has satisfied all three quality standards, it can be determined that the software is of high quality, and will satisfy the user/client.

## User Documentation

User documentation for the application is available either through the app or at <http://noahvendrig.com/docs/html>

## Technical Documentation

### System Documentation

More documentation available here: <http://noahvendrig.com/docs/html>

Instructional Video attached with this document.

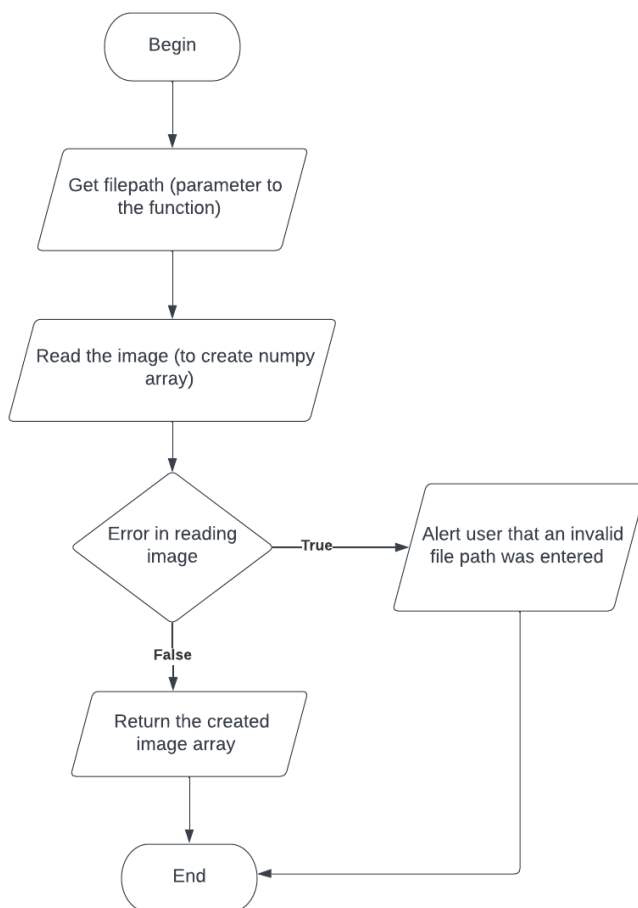
## Algorithm description for custom logic module

The 'ImageToArray(file) module is a function used in both predict.py and load\_and\_train.py as it is used for providing an image as a numpy array, which is required to pass through the model (for training and generating predictions). Below is the algorithm description for this module.

Pseudocode:

```
BEGIN ImageToArray (parameter=filepath)
  try:
    array = read(filepath) # read RGB image using opencv
    return array
  except error:
    display "Incorrect file path, submitted file could not be found"
    exit
END ImageToArray
```

Flowchart





Source code

Click [here](#) for the source code

## Refined Data Dictionary

cli.py

(no vars)

app.py

Variable Name	Datatype	Example	Description
app	flask.app.Flask	<Flask 'app'>	Flask app
log	logging.Logger	<Logger werkzeug (ERROR)>	Used to limit the amount of content logged by flask (only show 'important' logs)
ALLOWED_EXTENSIONS	set	{'jpeg', 'png', 'jpg'}	Set of permitted file extensions for input images
h	integere	300	Height of the image that is passed into the image resizing function
w	intger	300	Width of the image that is passed into the image resizing function
channels	integer	3	
max_w	integer	500	Maximum width of an image to be displayed on the screen (at the discretion of the developer, I chose 500px)
ratio	float	1.0	Ratio of width to height so that the image can be scaled whilst retaining its aspect ratio
resized_h	integer	500	Height after resizing
resized_w	Integer	500	Height after resizing
dims	tuple	(500, 500)	Tuple of (resized_w, resized_h) representing the dimensions of the image
resized_im	Numpy.ndarray of unsigned 8 bit integers [0, 255]	[[[ 45, 45, 45], [ 43, 43, 43], [ 39, 39, 39], ..., [ 60, 60, 60], [ 54, 54, 54],	Resized image as a numpy array

		[ 50, 50, 50]]]	
file	werkzeug.datastructures.FileStorage	<FileStorage: 'Viral Pneumonia-13.png' ('image/png')>	File uploaded through the upload form
filename	string	'Viral_Pneumonia-13.png'	Filename of uploaded file
im	List of Numpy.ndarrays of unsigned 8 bit integers [0, 255]	[[[ [10, 10, 10], [16, 16, 16], [26, 26, 26], ..., [46, 46, 46], [46, 46, 46], [46, 46, 46] ]]]	Uploaded image that has been read into opencv as numpy array
prediction	string	'normal'	Model prediction of uploaded image. Is equal to the prediction attribute of Classifier (from predict.py)
path	string	'static/files/Viral_Pneumonia-1330.png'	Path to the location that the uploaded image was saved in
port	integer	5000	Port number used for the application
hostname	string	'DESKTOP-ABCD2EF'	Port for the web app to occupy
ip_addr	string	'123.456.78.9'	Ip address of the local machine in which the web app will be hosted on

Predict.py

Variable Name	Datatype	Example	Description
model	Keras Sequential object	<keras.engine.sequential.Sequential object at 0x000001A7A648D9D0>	A compiled Keras model instance
labels	List of strings	['normal', 'covid']	Labels that the dataset is trained on
img	string	'./dataset/normal/images/Normal-10000.png'	Path to the image to be evaluated
img_arr	Numpy.ndarray of unsigned 8 bit integers Shape: (299, 299, 3)	[[[ 45, 45, 45], [ 43, 43, 43], [ 39, 39, 39], ..., [ 60, 60, 60], [ 54, 54, 54], [ 50, 50, 50]]]	Numpy array of the image so that it can be processed and fed to the model
new_img	Tensor of 32 bit floats (single precision) Tensor Shape: ([1, 150, 150, 3])	[[[[[0.6039216 , 0.6039216 , 0.6039216 ], [0.57254905, 0.57254905, 0.57254905], [0.54509807, 0.54509807, 0.54509807], ..., [0.01176471, 0.01176471, 0.01176471], [0.01960784, 0.01960784, 0.01960784], [0.01960784, 0.01960784, 0.01960784]]]]]	Resized version of the img_arr, but normalised and expanded to be compatible with the model
prediction	string	'covid'	Attribute of Classifier instance
res	__main__.Classifier Object	<__main__.Classifier object at 0x000001C36944A130>	Instance of Classifier class
input	List of strings	["/images/Normal-10000.png"]	List containing the path to the image to be evaluated

Variable Name	Datatype	Example	Description
model	Keras Sequential object	<keras.engine.sequential.Sequential object at 0x000001A7A648D9D0>	A compiled Keras model instance
labels	List of strings	['normal', 'covid']	Labels that the dataset is trained on
img	string	'./dataset/normal/images/Normal-10000.png'	Path to the image to be evaluated
img_arr	Numpy.ndarray of unsigned 8 bit integers Shape: (299, 299, 3)	[[[ 45, 45, 45], [ 43, 43, 43], [ 39, 39, 39], ..., [ 60, 60, 60], [ 54, 54, 54], [ 50, 50, 50]]]	Numpy array of the image so that it can be processed and fed to the model
new_img	Tensor of 32 bit floats (single precision) Tensor Shape: ([1, 150, 150, 3])	[[[[[0.6039216 , 0.6039216 , 0.6039216 ], [0.57254905, 0.57254905, 0.57254905], [0.54509807, 0.54509807, 0.54509807], ..., [0.01176471, 0.01176471, 0.01176471], [0.01960784, 0.01960784, 0.01960784], [0.01960784, 0.01960784, 0.01960784]]]]]	Resized version of the img_arr, but normalised and expanded to be compatible with the model
prediction	string	'covid'	Attribute of Classifier instance
res	__main__.Classifier Object	__main__.Classifier object at 0x000001C36944A130	Instance of Classifier class
input	List of strings	["/images/Normal-10000.png"]	List containing the path to the image to be evaluated

Load\_and\_train.py

Variable Name	Datatype	Example	Description
is_gpu	bool	TRUE	Used to check whether there is a GPU available for training.
gen	class '__main__.ModelGenerator'	__main__.ModelGenerator object at 0x00000237DE9995E0	Instance of the ModelGenerator class
labels	List of strings	['normal','covid']	Attribute of class ModelGenerator which has the labels that the model is trained to predict on.
X	Numpy ndarray of	array([[[[ 0.42352942, 0.42352942], [[0.24705882, 0.24705882, 0.24705882], [0.47843137, 0.47843137, 0.47843137], [0.47058824, 0.47058824, 0.47058824], ..., [0.43529412, 0.43529412, 0.43529412], [0.39215687, 0.39215687, 0.39215687], [0.3529412 , 0.3529412 , 0.3529412 ]], [0.37254903, 0.37254903, 0.37254903], [0.34509805, 0.34509805, 0.34509805], [0.3137255 , 0.3137255 , 0.3137255 ]]]], dtype=float32)	All of the datasetimages in array form
Y	Numpy ndarray of 32 bit floats.	array([[0., 1.], [0., 1.]	All the labels matching those images.

		<pre>[0., 1.], [1., 0.], [0., 1.], [0., 1.], [0., 1.], [1., 0.], [1., 0.], [1., 0.], [1., 0.], [1., 0.], [0., 1.], [1., 0.], [0., 1.], [1., 0.]], dtype=float32)</pre>	
path	string	./dataset	Path to dataset
X_train	Numpy ndarray of 32 bit floats	<pre>array([[[[ 0.42352942, 0.42352942],       [[0.24705882, 0.24705882, 0.24705882],       [0.47843137, 0.47843137, 0.47843137],       [0.47058824, 0.47058824, 0.47058824],       ...,       [0.43529412, 0.43529412, 0.43529412],       [0.39215687, 0.39215687, 0.39215687],       [0.3529412 , 0.3529412 , 0.3529412 ]],       [0.37254903, 0.37254903, 0.37254903],</pre>	80% of X has been allocated to this variable for model training purposes

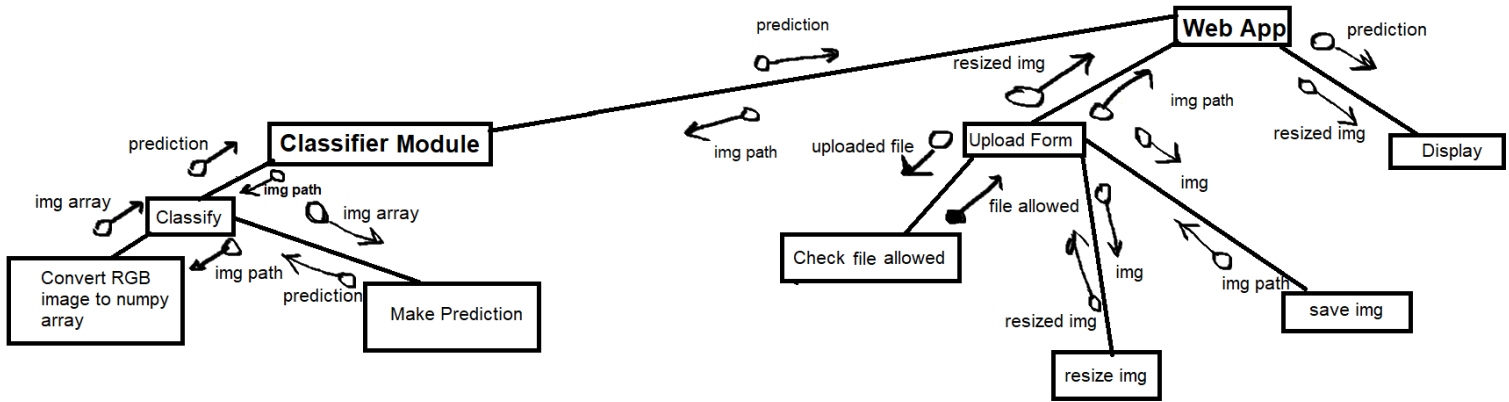
		[0.34509805, 0.34509805, 0.34509805], [0.3137255 , 0.3137255 , 0.3137255 ]]], dtype=float32)	
X_test	Numpy ndarray of 32 bit floatats	array([[[[ 0.42352942, 0.42352942], [[0.24705882, 0.24705882, 0.24705882], [0.47843137, 0.47843137, 0.47843137], [0.47058824, 0.47058824, 0.47058824],  ..., [0.43529412, 0.43529412, 0.43529412], [0.39215687, 0.39215687, 0.39215687], [0.3529412 , 0.3529412 , 0.3529412 ]], [0.37254903, 0.37254903, 0.37254903], [0.34509805, 0.34509805, 0.34509805], [0.3137255 , 0.3137255 , 0.3137255 ]]], dtype=float32)	20% of X has been allocated to this variable for model testing purposes
y_train	Numpy ndarray of 32 bit float	array([[0., 1.], [0., 1.], [0., 1.], [1., 0.], [0., 1.], [0., 1.], [0., 1.], [1., 0.]	80% of y has been allocated to this variable for model training purposes



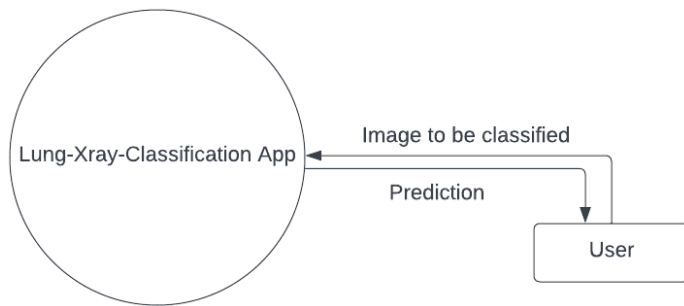
		[1., 0.], [1., 0.], [1., 0.], [1., 0.], [0., 1.], [1., 0.], [0., 1.], [1., 0.]], dtype=float32)	
y_test	Numpy array of 32 bit float	array([[1., 0.], [0., 1.], [1., 0.], [0., 1.]], dtype=float32)	20% of y has been allocated to this variable for model testing purposes
model	keras.engine.sequential.Sequential object	keras.engine.sequential.Sequential object at 0x00000267F0BE6670	
img_arr	Numpy.ndarray of unsigned 8 bit integers Shape: (299, 299, 3)	[[[ 45, 45, 45], [ 43, 43, 43], [ 39, 39, 39], ... [ 60, 60, 60], [ 54, 54, 54], [ 50, 50, 50]]]	Numpy array of the image so that the model can be trained (it obviously cant just look at the picture, it has to analyse the numbers to learn features).
evaluation	List of floats	[0.2148616462945938, 0.9612724781036377]	Validation loss and Accuracy
history	keras.callbacks.History object	<keras.callbacks.History object at 0x00000182BE190610>	Model history
file	string	"/images/photo.png"	Path to image to be converted to array
arr	Numpy ndarray	[[[0 0 0] [0 0 0] [0 0 0] ... [0 0 0] [0 0 0] [0 0 0]]]	List of the image with reduced dimensions to 150x150

label_index	Int	0	Get the index of the current label in LABELS (e.g. label 'covid' may correspond to index 0)
-------------	-----	---	---

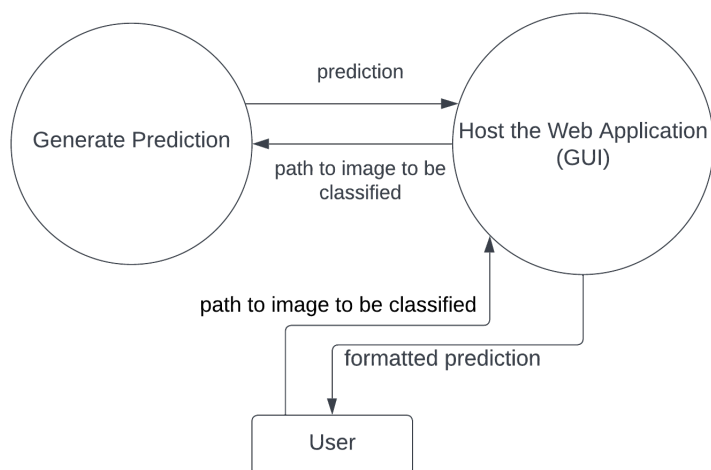
## Structure Chart



## Context Diagram



## Level 1 Dataflow Diagram



## Level 2 Dataflow Diagram

