Covi, The Chatbot!

Problem Statement: ML01 - During this pandemic, the healthcare industry has developed rapidly in order to combat covid in the best way possible. One can have several questions regarding Covid-19 and one's health which should be addressed quickly. However, doctors and healthcare workers have been working on the frontline, due to which, they cannot answer these questions for everyone. In order to tackle the problem, you have to design a chatbot that will answer all people's questions related to Covid-19.

The problem statement involved designing a ML based chatbot which could answer basic Covid-19 related queries that people might have and include other functionalities if possible. Our chatbot's name is Covi. It can recognise 45 different types of queries and also has a lung x-ray report checker, capable of estimating the probability of a person being infected with Covid.

The GUI of the chatbot is designed using the tkinter module available in python, by creating a user defined chatbot class. Chatbot class in Deploy.py is as follows:

```
# ChatBot class
class ChatBot:

    def __init__(self):
        self.window = Tk()
        self._setup_main_window()


    # helper function to keep the window active
    def run(self):
        self.window.mainloop()


    # helper function to setup the main window
    def _setup_main_window(self):
        self.window.title("Covi hates COVID")
        self.window.resizable(width = False, height = False)
        self.window.configure(width = 500, height = 580, bg = BG_COLOR)
```

It is followed by basic structuring and designing of the chatbox to make it more user friendly. It also has events described which make the chatbot more dynamic and capable of executing the objective as instructed.

For binding the strings to the view only text box above, we used this function,

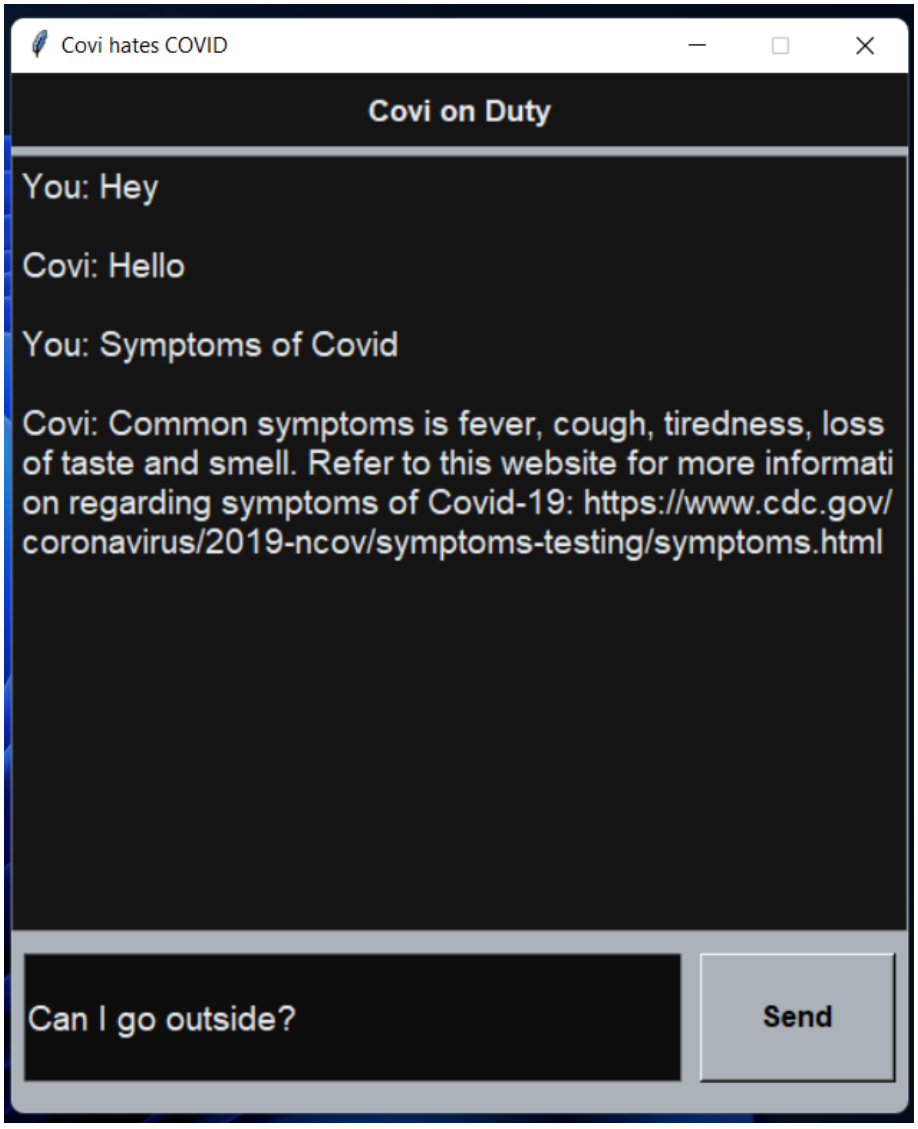
```
def _on_enter_pressed(self, event):
        msg = self.msg_entry.get()
        self._insert_message(msg, "You")


    def _insert_message(self, msg, sender):
        if not msg:
            return

        self.msg_entry.delete(0, END)
        msg1 = f"{sender}: {msg}\n\n"
        self.text_widget.configure(state = NORMAL)
        self.text_widget.insert(END, msg1)
        self.text_widget.configure(state = DISABLED)

        token, res = response(msg)

        if token == "lungs":
            img_path = fd.askopenfilename()
            res = predict_image(img_path)

        msg2 = f"Covi: {res}\n\n"
        self.text_widget.configure(state = NORMAL)
        self.text_widget.insert(END, msg2)
        self.text_widget.configure(state = DISABLED)

        self.text_widget.see(END)
```

The function uses two imported functions, response() from Response_Prediction.py and predict_iamge() from Lungs_Prediction.py which send the queries obtained from the user to the respective models.

response() takes in the user query and gets the prediction from the chatbot model we trained. It returns two objects, the response and query_tag.

If the query_tag is ‘lungs’, that is, the user is asking to check their lung report, a prompt window opens asking the user to upload the x-ray image from their system. It then predicts the probability of that person having Covid-19.

To deploy the chatbox, we must run the Deploy.py file, all the modules necessary have been listed in requirements.txt. After running the file, a chatbot window comes up which looks and we can enter queries in the text window by pressing enter or clicking the submit button as shown below:

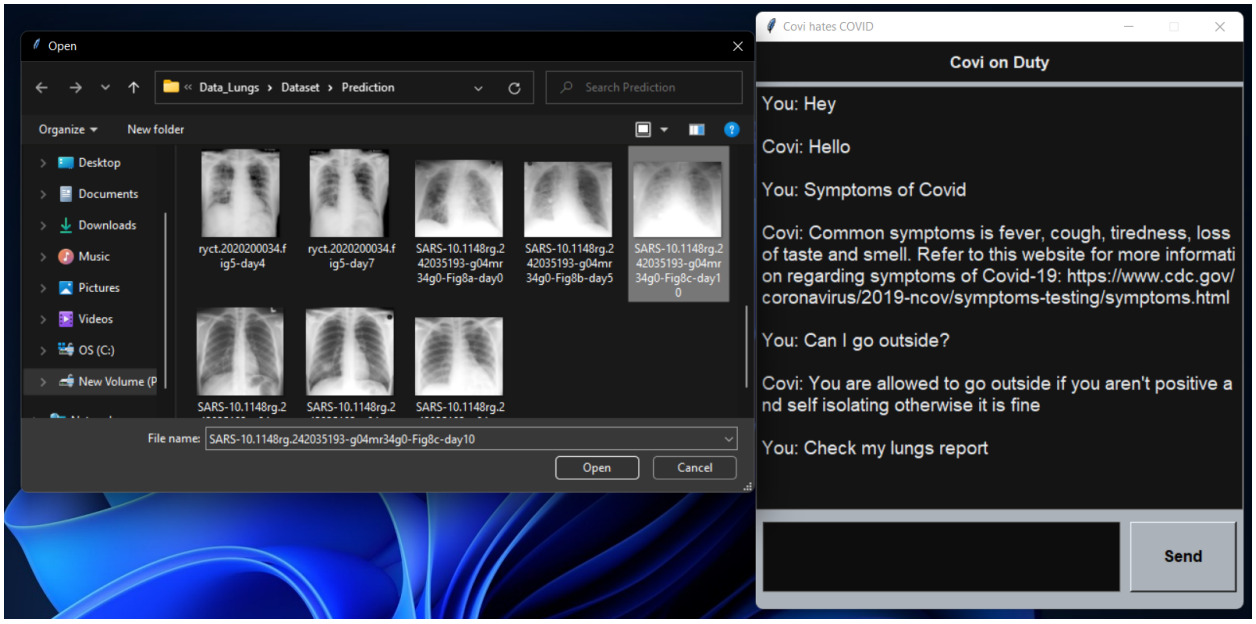


The chatbot replies accordingly to all the queries. Some example queries are as follows:

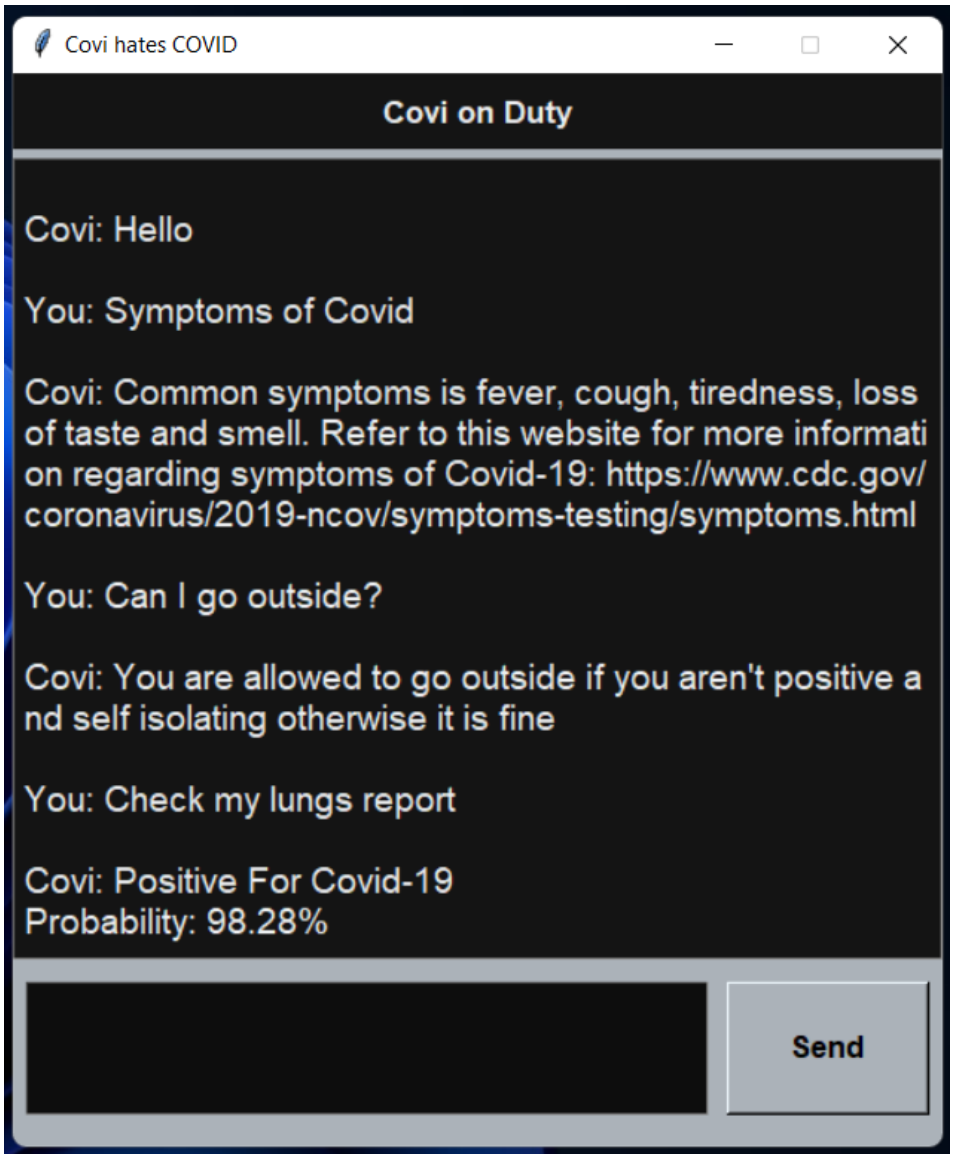
- Where can I get masks?
- Can I travel?
- Is it safe to go outside?
- Who are you?
- What does pandemic mean?
- Can I get tested?
- Help me find a doctor.
- How long should I stay at home?
- Can I volunteer?
- Should I test myself?
- Check my report.

and so on.

On querying anything related to reports, a prompt window opens asking to upload the x-ray image as shown below:



After opening the image, the query_tag is checked and sent to the lung report cnn model we trained. It then displays the probability of person having covid based on the x-ray report as shown below:



Please refer to the other documentation for details about the model hyperparameters and training. Thank you.

Chatbot Machine Learning Model

Dataset:

The dataset used in the chatbot is stored in a JSON file.
Part of the dataset is obtained from https://developer.ibm.com/exchanges/data/all/cqa/.

Each data is grouped, and the grouped data is divided into three parts, tag, pattern, and responses.

Example :

```
    {
      "tag": "quarantine_living_alone",
      "patterns": [
        "Am I allowed to go to my garden in quarantine?",
        "what to do while alone",
        "quarintine living alone",
        "Can I go shopping when I'm in quarantine?",
        "Can I order food while in quarantine?",
        "If I am in quarantine, can I go for walks?",
        "What if I live alone and can't go out to buy food?",
        "What if I live by myself and I'm in quarantine?",
        "What should I do if I live by myself and I'm in quarantine?"
      ],
      "responses": [
        "If you’re in self-isolation, you should really not be outside except
for your own property, call a friend or family member to get your groceries"
      ]
    }
```

Tags are used to identify the group of patterns

The pattern tags include example questions that help the machine learning model find patterns and give an appropriate answer when asked.

The program chooses a random input from responses if greater than one response.

There are 757 Different Patterns and 45 Different Tags.

Importing Data

We load the JSON file and input all the results in a dictionary and convert it into a pandas dataframe

```
tags=[]
patternss=[]
responses={}
for intent in data['intents']:
    responses[intent['tag']]=intent['responses']
```

```
    for patterns in intent['patterns']:
        patternss.append(patterns)
        tags.append(intent['tag'])
data = pd.DataFrame({'patterns':patternss, 'tag':tags})
```

Preprocessing

We remove the punctuation and convert all patterns to lowercase

```
data['patterns'] = data['patterns'].apply(lambda x:
x.translate(str.maketrans('', '', string.punctuation)))
data['patterns'] = data['patterns'].apply(lambda x: x.lower())
```

We then tokenize the data, we do this in order to get our computer to understand any text, we need to break that word down in a way that our machine can understand.

```
tokenizer = Tokenizer(num_words=2000)
tokenizer.fit_on_texts(data['patterns'])
train = tokenizer.texts_to_sequences(data['patterns'])
x_train = tokenizer.texts_to_sequences(data['patterns'])
```

We add padding to the text

```
x_train = pad_sequences(train)
```

We encode the data

```
encoder = LabelEncoder()
encoder.fit(data['tag'])
y_train = encoder.fit_transform(data['tag'])
```

Neural Network

We use keras to create the Machine learning model

```
i = Input(shape=(input_shape1,))
```

Embedding is Turns positive integers (indexes) into dense vectors of fixed size. It is used as the first layer in a model

```
x = Embedding(vocab_size+1, 10)(i)
```

We then use Long Short Term Memory (LSTM) layer as they are a type of neural network that is especially helpful in sequence prediction problems.

```
x = LSTM(10,return_sequences=True)(x)
```

Finally we create a densely connected NN layer with softmax activation function. Softmax is an activation function that scales numbers into probabilities.

```
x = Flatten()(x)
x = Dense(output_size, activation='softmax')(x)
```

We compile the model with loss function as sparse_categorical_crossentropy, which produces a category index of the most likely matching category and with optimizer as adam as it had the best results.

```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

We then train the model with 100epochs

```
train = model.fit(x_train, y_train, epochs=200)
```

Getting Output from Model

We use the model to predict the output. We find the matching tag by reversing the decoder

```
output = model.predict(prediction_input)
output = output.argmax()

#finding the right tag and predicting
response_tag = encoder.inverse_transform([output])[0]
final_output = random.choice(responses[response_tag])
print("Covid 19- Chat bot : ",random.choice(responses[response_tag]))
```

COVID PREDICTION THROUGH CHEST X-RAY IMAGES

Dataset :
https://github.com/RadhikaChhabra17/Tri-nit-hackathon/tree/main/Notebooks/data/Dataset
It contains two directory of covid and normal images.

TECH STACK :

- Tensorflow
- Matplotlib
- OpenCV

Hyperparameters :

- Activation function **:** sigmoid
- Batch size : 6
- Epochs : 11
- Optimizer **:** rmsprop

Model:

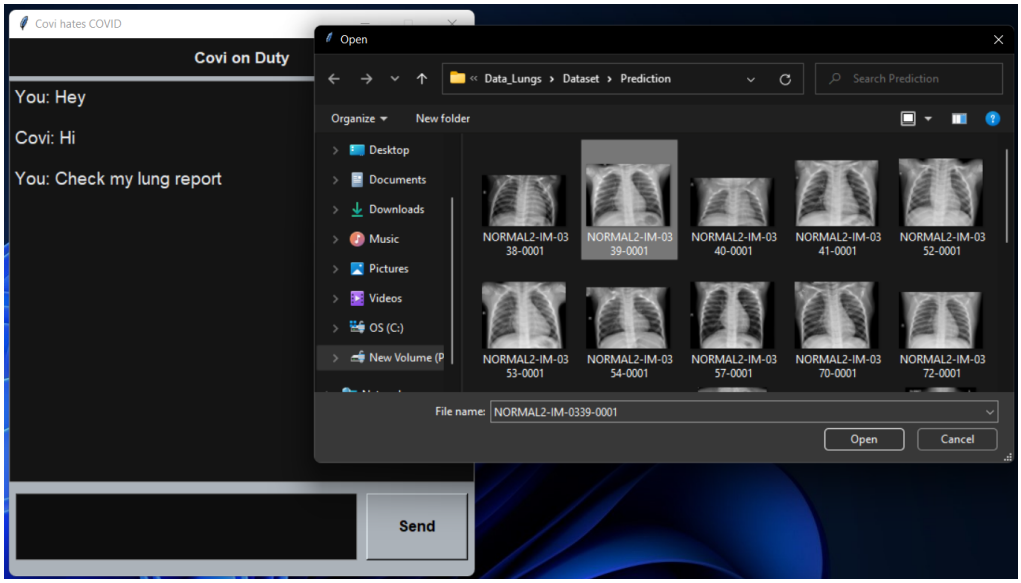
An input image of size (150,150) is passed to CNN Model. A CNN Model with a series of convolution, activation and pooling layers.Filter size for convolution layer is (3,3). Relu activation function is used for these layers. Pooling layer has a pool size (2,2). The output from these layers is flattened before passing it to the dense layer. Dropout layer is used to drop neurons randomly. Finally, the sigmoid activation function is used to predict the probability as an output.

Evaluation metrics : accuracy

Model architecture :

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 activation (Activation)     (None, 148, 148, 32)      0

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        9248

 activation_1 (Activation)   (None, 72, 72, 32)        0

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 activation_2 (Activation)   (None, 34, 34, 64)        0

 conv2d_3 (Conv2D)           (None, 32, 32, 250)       144250

 activation_3 (Activation)   (None, 32, 32, 250)       0

 conv2d_4 (Conv2D)           (None, 30, 30, 128)       288128

 activation_4 (Activation)   (None, 30, 30, 128)       0

 average_pooling2d (AverageP  (None, 15, 15, 128)      0
 ooling2D)

 conv2d_5 (Conv2D)           (None, 13, 13, 64)        73792

 activation_5 (Activation)   (None, 13, 13, 64)        0



 average_pooling2d_1 (Averag  (None, 6, 6, 64)         0
 ePooling2D)

 conv2d_6 (Conv2D)           (None, 5, 5, 256)         65792

 activation_6 (Activation)   (None, 5, 5, 256)         0

 max_pooling2d_2 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 32)                32800

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

 activation_7 (Activation)   (None, 1)                 0

=================================================================
Total params: 633,435
Trainable params: 633,435
Non-trainable params: 0
_________________________________________________________________
```

RESULT :


Covi hates COVID
Covi on Duty
You: Hey
Covi: Hi
You: Check my lung report
Send
Open
Data_Lungs › Dataset › Prediction
Search Prediction
Organize
New folder
Desktop
Documents
Downloads
Music
Pictures
Videos
OS (C:)
New Volume (P
NORMAL2-IM-0338-0001
NORMAL2-IM-0339-0001
NORMAL2-IM-0340-0001
NORMAL2-IM-0341-0001
NORMAL2-IM-0352-0001
NORMAL2-IM-0353-0001
NORMAL2-IM-0354-0001
NORMAL2-IM-0357-0001
NORMAL2-IM-0370-0001
NORMAL2-IM-0372-0001
File name: NORMAL2-IM-0339-0001
Open
Cancel


Covi hates COVID
Covi on Duty
You: Hey
Covi: Hi
You: Check my lung report
Covi: Negative for Covid-19
Probability: 97.12%
Send