COVID-19 DETECTION

Introduction

Coronavirus Disease 2019 (COVID-19) has become a global pandemic with an exponential growth rate and an incompletely understood transmission process. The virus is harbored most commonly with little or no symptoms, but can also lead to rapidly progressive and often fatal pneumonia in 2–8% of those infected1,2,3. The exact mortality, prevalence, and transmission dynamics remain somewhat ill-defined in part due to the unique challenges presented by SARS-CoV-2 infection, such as peak infectiousness at or just preceding symptom onset and a poorly understood multi-organ pathophysiology with dominant features and lethality in the lungs4. The rapid rate of spread has strained healthcare systems worldwide due to shortages in key protective equipment and qualified providers5, partially driven by variable access to point-of-care testing methodologies, including reverse transcription-polymerase chain reaction (RT-PCR). As rapid RT-PCR testing becomes more available, challenges remain, including high false-negative rates, delays in processing, variabilities in test techniques, and sensitivity sometimes reported as low as 60–70%6,7.

AI in Covid-19

Due to the rapid increase in the number of new and suspected COVID-19 cases, there may be a role for artificial intelligence (AI) approaches for the detection or characterization of COVID-19 on imaging. CT provides a clear and expeditious window into this process, and deep learning of large multinational CT data could provide automated and reproducible biomarkers for the classification and quantification of COVID-19 disease. Prior single-center studies have demonstrated the feasibility of AI for the detection of COVID-19 infection or even differentiation from community-acquired pneumonia17,18. AI models are often severely limited in utility due to homogeneity of data sources, which in turn limits applicability to other populations, demographics, or geographies. This study aims to develop and evaluate an AI algorithm for the detection of COVID-19 on chest CT using data from a globally diverse, multi-institution dataset. Here we show robust models can achieve up to 90% accuracy in independent test populations, maintaining high specificity in non-COVID-19 related pneumonia, and demonstrating sufficient generalizability to unseen patient populations/centers.

In this blog, we have focused in distinguish the cough to determine if the person is infected with coronavirus or not.

We start by importing the packages and configuring some settings.

Libraries

```
# make session persistent
%%javascript
function ClickConnect(){
console.log("Working");
document.querySelector("colab-toolbar-button#connect").click()
}setInterval(ClickConnect,60000)
```

```
[ ] from sklearn.model_selection import train_test_split
    import pandas as pd
    import os
    import librosa
    import librosa.display
    import cv2
    import numpy as np
    import json
    import matplotlib.pyplot as plt
    import warnings
    warnings.simplefilter("ignore")
    import IPython.display as ipd
    %matplotlib inline
    import matplotlib.pyplot as plt
    from scipy.io import wavfile
    import shutil
```

Loading Data

We read the data set coughvid from the virufy repo.

```
[ ]  !git clone "https://github.com/virufy/virufy-cdf-coughvid.git"
     %cd virufy-cdf-coughvid
```

```
Cloning into 'virufy-cdf-coughvid'...
remote: Enumerating objects: 22051, done.
remote: Total 22051 (delta 0), reused 0 (delta 0), pack-reused 22051
Receiving objects: 100% (22051/22051), 1006.41 MiB | 30.66 MiB/s, done.
Resolving deltas: 100% (16/16), done.
Checking out files: 100% (22045/22045), done.
/content/virufy-cdf-coughvid
```

```
[ ]  coughvid = pd.read_csv("virufy-cdf-coughvid.csv")
     coughvid.head()
     x, sr = librosa.load('virufy-cdf-coughvid/0009eb28-d8be-4dc1-92bb-907e53bc5c7a.webm')
     #ipd.Audio('virufy-cdf-coughvid/0009eb28-d8be-4dc1-92bb-907e53bc5c7a.webm')
```

It contains a CSV file that contains the patient ids and some information about the patient

```
coughvid.head()
```

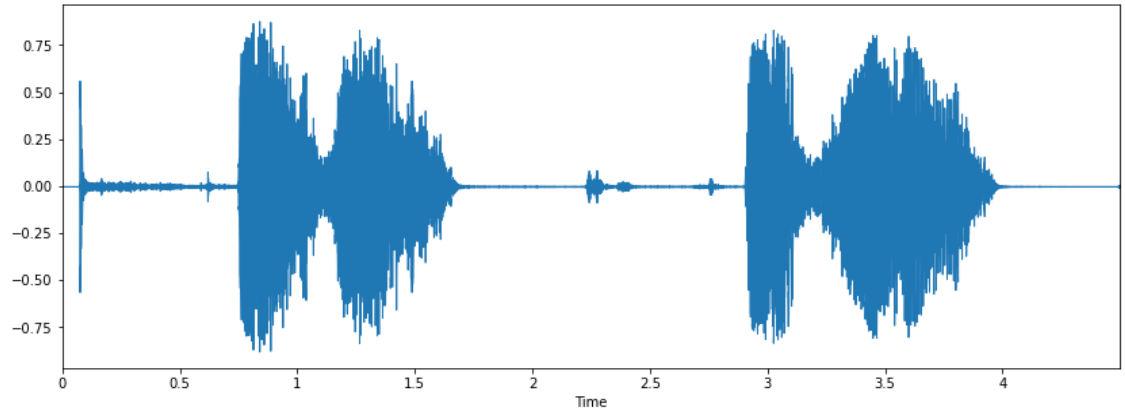
	Unnamed: 0	source	patient_id	cough_detected	cough_path	age	biological_sex	reported_gender	submission_date	pcr_test_date	pcr_result_date	respiratory_condition	fever
0	0	coughvid	ae029647-ddc3-47f5-904e-813358c9dda3	0.8929	virufy-cdf-coughvid/ae029647-ddc3-47f5-904e-81...	NaN	NaN	NaN	2020-10-20T11:05:04.501905+00:00	NaN	NaN	NaN	
1	1	coughvid	cac4a470-bb86-4db9-9453-d4e2706c3931	0.1431	virufy-cdf-coughvid/cac4a470-bb86-4db9-9453-d4...	NaN	NaN	NaN	2020-04-17T15:12:14.379396+00:00	NaN	NaN	NaN	
2	2	coughvid	2a3d201f-7c70-449c-adc5-50de728fccd0	0.0803	virufy-cdf-coughvid/2a3d201f-7c70-449c-adc5-50...	38.0	male	male	2020-04-09T20:56:01.725275+00:00	NaN	NaN	False	
3	3	coughvid	8b267d78-97f0-4530-bbfa-e938724376f4	0.1247	virufy-cdf-coughvid/8b267d78-97f0-4530-bbfa-e9...	NaN	NaN	NaN	2020-04-11T11:12:19.213465+00:00	NaN	NaN	NaN	
4	4	coughvid	68d5215b-afdb-4b7a-81d3-fbd5677de7b4	1.0000	virufy-cdf-coughvid/68d5215b-afdb-4b7a-81d3-fb...	46.0	male	male	2020-11-26T16:21:47.764016+00:00	NaN	NaN	False	

health.

Visualizing dataset

```
[ ]  plt.figure(figsize=(14, 5))
     librosa.display.waveplot(x, sr=sr)
```

```
<matplotlib.collections.PolyCollection at 0x7f21cca18b90>
```




```
[ ]  coughvid['pcr_test_result_inferred'].value_counts()#counts of unique values
```

```
negative    7178
untested    5399
positive     588
Name: pcr_test_result_inferred, dtype: int64
```

```
[ ]  msk = (coughvid.loc[:,'pcr_test_result_inferred']=='untested')#filter out untested results
     coughvid = coughvid.loc[~msk,:]
     coughvid['pcr_test_result_inferred'].value_counts()
```

```
negative    7178
positive     588
Name: pcr_test_result_inferred, dtype: int64
```

Audio Preprocessing

- Using the trim method from libsora library, we have implemented the trim_silence function to trim silence from the audio signal and its intervals.

```
def trim_silence(x, *args):
    try:
      """
      hop length: The number of samples between analysis frames
      frame_lengthint > 0 : The number of samples per analysis frame
      top_db number > 0 :The threshold (in decibels) below reference to consider as silence
      """
      pad,db_max,frame_length,hop_length = args[0],args[1],args[2],args[3]
    except:
        print('Please enter the following arguments: pad,db_max,frame_length,hop_length')
        return

        """ function : Trim leading and trailing silence from an audio signal
        return : trimmed signal and  ints refer to intervals
        """
    _, ints = librosa.effects.trim(x, top_db=db_max, frame_length=256, hop_length=64)
    start = int(max(ints[0]-pad, 0))
    end   = int(min(ints[1]+pad, len(x)))
    return x[start:end]
```

- Get mel spectrogram using libsora method and convert it to DB and save the result as image .png.

```
def get_melspec(sdir,audio,sr,name):
    #Mel Spectogram
    plt.ioff()
    fig = plt.figure()
    melspec  = librosa.feature.melspectrogram(y=audio,sr=sr) #compute melspectorgram
    s_db     = librosa.power_to_db(melspec, ref=np.max)#convert it to dicible
    librosa.display.specshow(s_db)
    fig.canvas.draw()
    img = np.fromstring(fig.canvas.tostring_rgb(), dtype=np.uint8, sep='')
    img = img.reshape(fig.canvas.get_width_height()[::-1] + (3,))
    plt.close(fig=fig)
    #img = img[80:250,80:300]

    savepath = os.path.join(sdir,name+'.png') # Currently saving melspectrogram images to the
    cv2.imwrite(savepath,img)
    return savepath
```

❖ Get raw MFCC feature and get the label of every patient.

```
def get_rawMFCCs(audio,sr,*args):
    try:
      hop_length,win_length,n_mfcc,n_mels,n_ftt = args[0],args[1],args[2],args[3],args[4]
    except:
        hop_length = np.floor(0.010*sr).astype(int) #10ms
        win_length = np.floor(0.020*sr).astype(int) #20ms
        n_mfcc,n_mels,n_ftt=13,13,2048

    rawMFCCs    = librosa.feature.mfcc(y=audio,sr=sr, n_mfcc=n_mfcc,n_mels=n_mels, n_fft=n_ftt, hop_length=hop_length)
    rawMFCCs    = np.mean(rawMFCCs.T,axis=0).tolist()
    return rawMFCCs

def getlabel(key, dataframe, chosen):
      return dataframe.loc[dataframe[chosen['id']]==key][chosen['pcr']].tolist()[0]
```

Then implement and filter data set to extract necessary features in JSON files.

```
def extract(df, chosen, savedir):
    if not os.path.isdir(savedir):
        os.mkdir(savedir)

    keys, dirs = df[chosen['id']].tolist(),df[chosen['path']].tolist()
    audio_objs = [process_cough_file(path,trim_silence) for path in dirs]
    false_indices = [i for i in range(len(audio_objs)) if isinstance(audio_objs[i],int) or isinstance(audio_objs[i],tuple)]

    audio_objs = [audio_objs[i] for i in range(len(audio_objs)) if i not in false_indices]
    audio_objs = np.array(audio_objs)
    audio,sr,hop_length,win_length = audio_objs[:,0],audio_objs[:,1],audio_objs[:,2],audio_objs[:,3]

    dirs = [dirs[i] for i in range(len(dirs)) if i not in false_indices]
    keys = [keys[i] for i in range(len(keys)) if i not in false_indices]
    data = {key:{'DIR':get_melspec(savedir,a_i,sr_i,key),
             'rawMFCC':get_rawMFCCs(a_i,sr_i),
             'label':getlabel(key, df, chosen)} for key,a_i,sr_i in list(zip(keys,audio,sr))}
    return data

def filter_DF(df):
    names = list(df.columns)
    chosen= {}
    for name in names:
        if 'inferred' in name.lower():
          chosen['pcr'] = name # Choosing the target (pcr_test_result_inferred)
        elif 'path' in name.lower():
          chosen['path'] = name
        elif 'patient' in name.lower() or 'id' == name.lower() :
          chosen['id'] = name
    return df[[chosen['id'],chosen['pcr'],chosen['path']]].dropna().reset_index(), chosen
```

The JSON file contains a dictionary of dictionary patient ID, mel spectrum image. MFCC coefficients, and labels.


```
{
    "feb9cd98-1e54-42b8-950c-f1e8dbd1e814": {
        "DIR": "train_melspecs/feb9cd98-1e54-42b8-950c-f1e8dbd1e814.png",
        "rawMFCC": [
            -145.77175059071723,
            35.965893268216135,
            -9.726178034876549,
            -1.6472076524257047,
            -4.357904630418849,
            -4.0559904935182844,
            0.4141639182641759,
            -1.3523823394396914,
            1.65659729244957,
            -0.6337369944446554,
            -2.3416402888512273,
            0.48101188702145237,
            -0.5061504562695437
        ],
        "label": "negative"
    },
    "511b8b47-b97a-45d8-9b23-db64c972182d": {
        "DIR": "train_melspecs/511b8b47-b97a-45d8-9b23-db64c972182d.png",
        "rawMFCC": [
            -143.46103859763,
            23.973843699037822,
            -11.290191710811246,
            5.0832665642683965,
            4.956657582663141,
            -0.3973688461984156,
            3.4628322592600354,
            1.3223940292625096,
            1.2742992194637148,
            -0.551835484138077,
            -1.4550746634243579,
            1.3349952973529091,
            -0.5865337806142575
        ],
        "label": "negative"
    },
```

Approach 1 using MFCC Coff and labels:

- Work on JSON files and create a CSV file contains only patient ID, rawMFCC, and labels.

MFCC_13	MFCC_12	MFCC_11	MFCC_10	MFCC_9	MFCC_8	MFCC_7	MFCC_6	MFCC_5	MFCC_4	MFCC_3	MFCC_2	MFCC_1	Labels	ID
-0.50615	0.481012	-2.34164	-0.63374	1.656597	-1.35238	0.414164	-4.05599	-4.3579	-1.64721	-9.72618	35.96589	-145.772	negative	train_mels
-0.58653	1.334995	-1.45507	-0.55184	1.274299	1.322394	3.462832	-0.39737	4.956658	5.083267	-11.2902	23.97384	-143.461	negative	train_mels
-1.273	1.070084	-0.80772	0.30262	0.663827	-1.85667	4.846473	2.702674	2.416739	4.868323	-8.58654	33.39232	-141.374	negative	train_mels
-0.58628	-0.33497	-2.75091	-1.55947	1.974609	-4.61013	3.549204	-0.04571	-1.80178	11.52577	-18.8113	43.57165	-105.429	negative	train_mels
-1.30366	0.682674	-0.13162	-3.38621	-2.97857	-3.0739	1.230547	1.962148	-1.0364	-1.19067	-1.54268	42.99237	-133.243	negative	train_mels
-1.04777	1.678747	-1.00813	-0.34103	-0.38306	-0.39145	5.941944	-2.08953	1.965106	6.212136	-16.0004	40.04529	-126.956	negative	train_mels
-0.75591	1.583934	-0.5182	0.091759	0.08448	-1.28234	1.633822	-0.88408	2.14053	1.132392	-7.88568	33.13721	-188.049	negative	train_mels
-0.58409	0.674095	0.399657	0.181901	-0.21971	1.399992	-0.30561	0.687173	-0.81547	1.88468	1.118786	11.96597	-157.393	negative	train_mels
0.848715	-0.45094	-0.8308	-1.95654	0.17084	1.324325	0.848749	0.219092	-3.725	2.528334	-2.02677	16.84826	-155.594	negative	train_mels
0.333299	1.196234	-2.15977	0.68025	1.891111	0.604035	5.473999	-3.42441	-1.61843	8.502107	-6.14999	29.61117	-132.529	negative	train_mels
-0.27504	0.630194	-0.45179	0.206742	0.659039	-0.46466	2.572753	-0.16584	0.415368	5.37437	-3.6334	16.71413	-160.956	negative	train_mels
0.187383	0.253146	-1.1204	0.259974	0.636792	0.79338	3.634433	1.112729	2.175834	3.530137	-0.78785	23.61217	-176.398	negative	train_mels
0.112219	-0.53289	0.408235	-0.63698	-0.62365	0.283031	-0.57626	2.078869	2.564069	1.654776	6.332078	30.89453	-123.482	negative	train_mels
-0.50785	0.605206	-0.9657	-0.38646	-0.68247	-0.86673	2.50947	-1.67555	0.755332	3.48738	-7.22888	21.52642	-179.054	negative	train_mels
-1.52817	0.785548	-1.7439	-0.47584	0.391164	-0.26538	3.672038	-1.97377	-0.1242	3.735607	-7.68532	25.34633	-155.545	negative	train_mels
0.539597	-2.51931	-0.38002	-2.32367	-1.85961	3.746521	5.720924	7.195659	-0.1135	-2.78436	-19.6623	35.37654	-97.9625	negative	train_mels
-0.47796	0.251616	-0.78579	-0.76642	-0.25044	0.468265	1.956559	-0.93088	-0.67881	0.027628	-3.0991	16.47246	-185.533	negative	train_mels
0.01759	0.949183	-0.62419	-0.99454	0.0348	-1.01248	2.793245	-0.76253	-2.56759	0.848079	-8.49862	28.07637	-138.12	negative	train_mels
-2.03551	1.724348	-2.41095	3.200737	3.088028	-2.44711	5.133361	-6.98908	-3.75057	10.62574	-17.5187	48.46482	-207.274	negative	train_mels
-1.42781	1.773637	-1.44803	-0.29112	-0.58959	-2.80091	3.237518	-3.17039	-0.49621	4.584354	-14.3567	28.90777	-128.538	negative	train_mels
-0.58713	1.493311	-0.82902	0.236284	0.549234	-2.35954	4.712415	1.278029	-0.19734	6.122111	-7.88058	25.39134	-143.003	negative	train_mels
-0.34264	1.098813	-1.01766	0.170051	0.078509	-1.48207	3.51603	-1.03701	-0.97237	4.337658	-6.302	20.8752	-249.724	negative	train_mels

Model Architecture:

- Implement logistic regression architecture:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

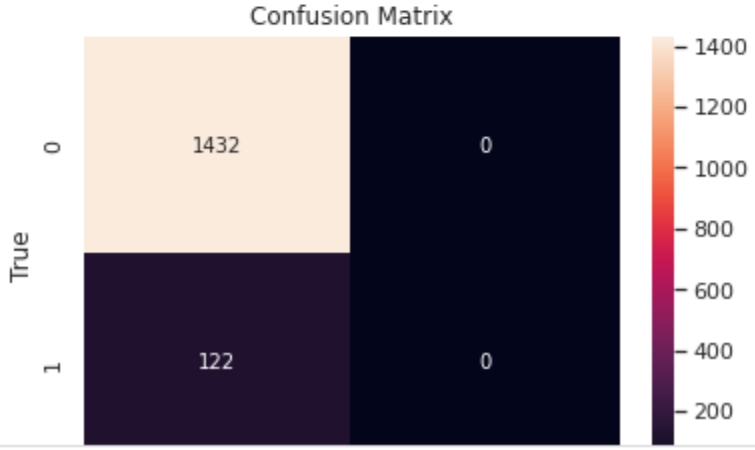
```
[ ] trained_data= pd.read_csv('train_features_final.csv')
    #label_encoder object knows how to understand word labels
    label_encoder = preprocessing.LabelEncoder()

    #Encode labels in column Sex and Embarked
    trained_data['Labels']= label_encoder.fit_transform(trained_data['Labels'])
    trained_data['ID']= label_encoder.fit_transform(trained_data['ID'])
```

```
[ ] trained_data.head()
    trained_data['ID'].value_counts()
```

❖ And the result is :

```
#Confusion matrix and classification report
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      1432
           1       0.00      0.00      0.00       122

    accuracy                           0.92      1554
   macro avg       0.46      0.50      0.48      1554
weighted avg       0.85      0.92      0.88      1554

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classificati
  _warn_prf(average, modifier, msg_start, len(result))
```



Approach 2:

In this approach, we have worked on images generated from mel scale and labels

- Preprocessing to extract images and their labels:

```
all_labels = []
for i in range(6212):
  label=all_data[i][49:]
  label=label[:-4]
  if data1[label]["label"]=="negative":
    shutil.copy(all_data[i],"/content/drive/MyDrive/Final_Mach/train_melspecs/neg")
  else:
    shutil.copy(all_data[i],"/content/drive/MyDrive/Final_Mach/train_melspecs/pos")

  all_labels.append(data1[label]["label"])
```

```
[ ] for i in range(1554):
      label=all_data[i][49:]
      label=label[:-4]
      if data1[label]["label"]=="negative":
        shutil.copy(all_data[i],"/content/drive/MyDrive/Final_Mach/teest_melspecs/neg")
      else:
        shutil.copy(all_data[i],"/content/drive/MyDrive/Final_Mach/teest_melspecs/pos")

      all_labels.append(data1[label]["label"])
```

❖ Using ImageGenerator to overcome unbalanced data:

```
from keras.preprocessing.image import ImageDataGenerator

BATCH_SIZE = 64
HEIGHT = 224
WIDTH = 224
VAL_SPLIT = 0.1


# 1. Construct an instance of the `ImageDataGenerator` class
train_datagen =  ImageDataGenerator(
      rescale = 1.0/255,
      featurewise_std_normalization=True,
      samplewise_std_normalization=True,
      validation_split = VAL_SPLIT
    )

# 2. Retrieve the iterator
train_generator = train_datagen.flow_from_directory(TRAIN_DIR,
                                                    shuffle = True,
                                                    seed = 7,
                                                    target_size=(HEIGHT, WIDTH),
                                                    batch_size=BATCH_SIZE,
                                                    color_mode='rgb',
                                                    class_mode='categorical',
                                                    subset='training')
```

❖ Then, Implement a CNN model

```
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from keras.models import Sequential

def model_builder():
  # Specify model architecture
  model = Sequential()
  # Block 1
  model.add(Conv2D(32, (3, 3),
                   activation='relu',
                   kernel_initializer='he_uniform',
                   padding='same',
                   input_shape=(HEIGHT, WIDTH, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))
  # Block 2
  model.add(Conv2D(64,(3,3),activation='relu'))
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(0.25))
  # FC part
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(2, activation='softmax'))

  # Print summary
  model.summary()

  # Compile model
  model.compile(optimizer="rmsprop", loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

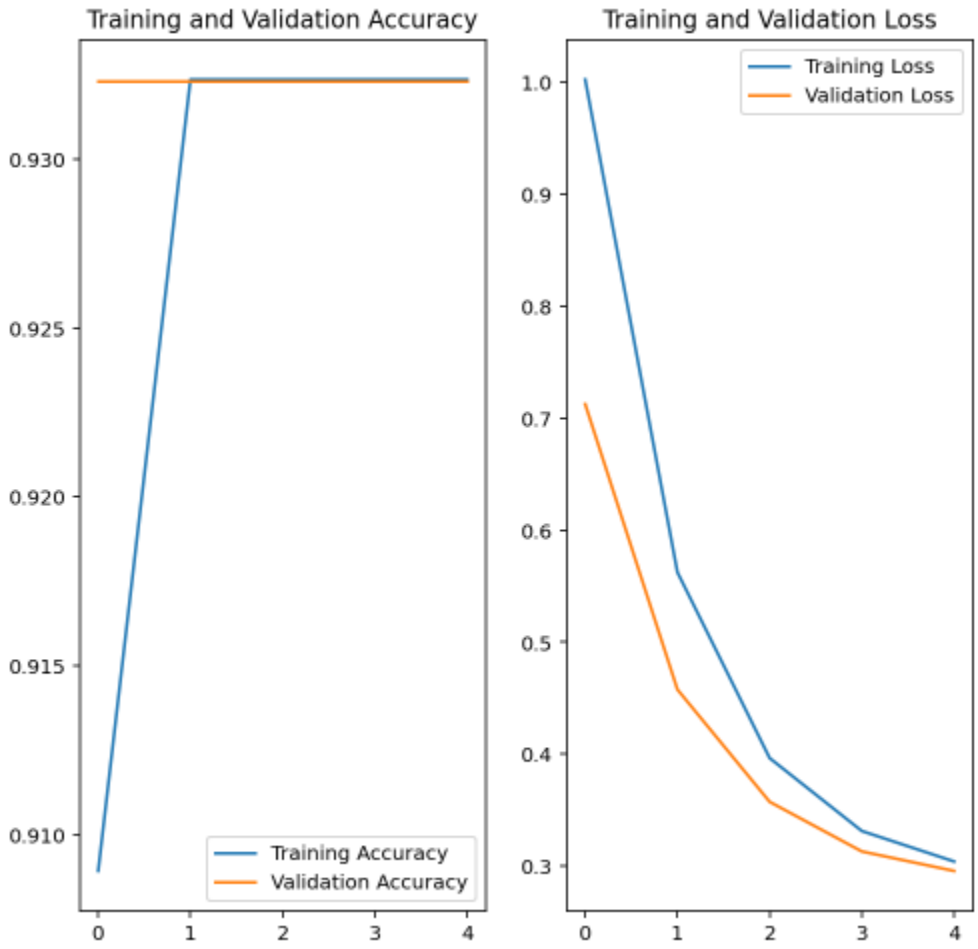
❖ Using Mobilenet architecture and adam optimization

```
[ ]  URL = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"

     feature_extractor = hub.KerasLayer(URL, input_shape=(224, 224,3))

     feature_extractor.trainable = False

     model = tf.keras.Sequential(feature_extractor)

     model.add(Dense(2, kernel_regularizer='l1_l2',activation = 'softmax'))

     model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_1 (KerasLayer)   (None, 1280)              2257984
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 2562
=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

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

❖ The result is :



Then using Resnet 50 and use 20 epochs to train :

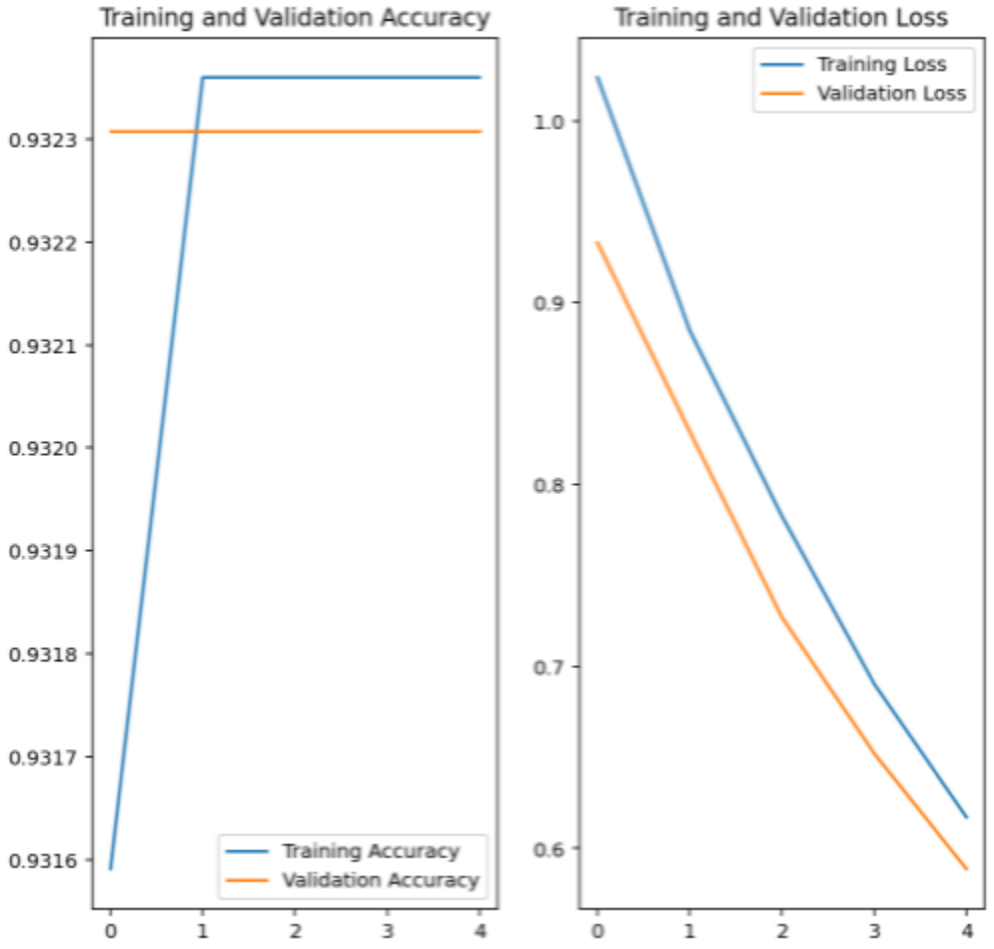
```
[ ]  URL="https://tfhub.dev/google/bit/m-r50x1/ilsvrc2012_classification/1"

     resnet_50 = hub.KerasLayer(URL, input_shape=(224, 224,3))
     resnet_50.trainable = False
```

```
[ ]  model_3 = tf.keras.Sequential(resnet_50)
     model_3.add(Dense(2, kernel_regularizer='l1_l2',activation = 'sigmoid'))
```

```
[ ]  model_3.compile(optimizer = "adamax",
                     loss='binary_crossentropy',
                     metrics=[tf.keras.metrics.Precision()])
```

```
     early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=4)
     history = model_3.fit(train_generator,
                           epochs = 20,
                           validation_data=val_generator,
                           callbacks=[early_stopping])
```

❖ **Getting the result :**




```
25/25 [==============================] - 22s 849ms/step - loss: 0.6077 - accuracy: 0.9350

Loss on the TEST Set: 0.248
Accuracy on the TEST Set: 93.501%
```