Covid Detection using Deep Learning mechanism through CT images *

Jaspreet Kaur
Dept. of Computer Science and electronic engineering
University of Essex, Colchester United Kingdom
email:jk20873@essex.ac.uk

***Abstract*—This work uses the Deep learning mechanism(DLM) for classification of Covid CT images from Non-Covid CT images. The fine tuning is applied to optimise the performance of the model. LaTeX.The entire process of classification is partitioned into phases. in the first phase we perform examination and identification of image dataset.In the next phase, input pipeline is created. after that model is built. in the next phase, training and testing is performed. At the end, model is improved through fine tuning and entire process is repeated. The model classification accuracy of 100% is achieved along with confidence of 100% proving worth of study.**

***Index Terms*—Covid, Non-Covid, CT-Images, DLM**

I. Introduction

To perform the desired operation, first of all necessary files must be imported. We are going to use Tensor Flow and keras. The process of image classification within Deep learning is partitioned into phases. In the first phase, data understanding and exploration is performed. The covid and non-covid CT images are used as a dataset. Once dataset is loaded, model is defined and classifcation is performed[1]. The detection result already shows high classification accuracy, but still fine tune mechanism is applied to increase the result by 1%. At the end, the test image stored within Test_image folder is classified as Covid or Non Covid image depending upon the features extracted.

II. Necessary Files

The required libraries are imported as shown below LaTeX.

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

All of these files are required to perform the desired operation on the presented dataset.

III. Understanding Dataset

The dataset that is used contains JPG as well as PNG images. Some CT images are within COvid folder and some are present within the Non-Covid folder. The Dataset sample is given as under

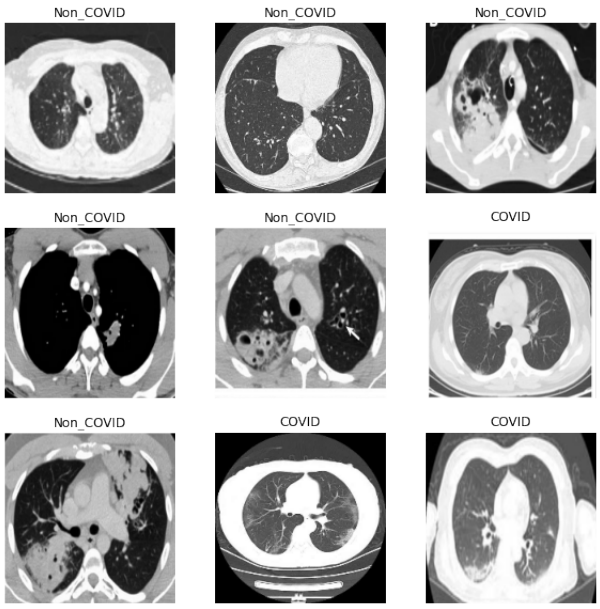

Fig. 1. Used Dataset

IV. Creating Input Pipeline

The task of pipeline will be to resize all the images to fit within the model. The labels must also be extracted from the image for classification purpose.This means proper dataset formatting is accomplished using input pipeline.

V. Building a model

A model is defined that consist of different layers. The model used in the proposed approach consist of 3 convolution blocks. Each convolution block contains a max pool layer. There exists a fully connected layer having 128 units activated by the relu activation function. This model is yet not tuned for

optimal performance[2][3].The standardization of this model is achieved with this phase that is optimised in upcoming sections. The structure of the model declared for achieving the optimization is given as under
num_classes = 2

model = Sequential([] layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
layers.Conv2D(16, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(32, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(64, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(num_classes)
])

VI. Model Compilation

Model is compiled to determine everything within the model is in place and working fine. Model compilation results in bug free model[4][5]. The command used for model compilation is given as under
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy
(from_logits=True),
metrics=['accuracy'])
The result of this compilation is presented using summary method
model.summary() The summary result in the following

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_6 (Rescaling)      (None, 200, 200, 3)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 200, 200, 16)      448
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 100, 100, 16)      0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 100, 100, 32)      4640
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 50, 50, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 25, 25, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 40000)             0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               5120128
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 258
=================================================================
Total params: 5,143,970
Trainable params: 5,143,970
Non-trainable params: 0
```

Fig. 2. Model Summary Result

VII. Fitting the model

This step is crucial as model accuracy and percentage loss is defined in terms of this step. The mechanism used for fitting the model is given as under
epochs=10
history = model.fit(
train_ds,
validation_data=val_ds,
epochs=epochs
)
Number of iterations used for defining this model is 10. The output corresponding to the fit is given in fig 3

```
Epoch 1/10
30/30 [==============================] - 10s 322ms/step - loss: 0.8675 - accuracy: 0.5494 - val_loss: 0.5881 - val_accuracy: 0.
6846
Epoch 2/10
30/30 [==============================] - 9s 315ms/step - loss: 0.5986 - accuracy: 0.6935 - val_loss: 0.4745 - val_accuracy: 0.7
651
Epoch 3/10
30/30 [==============================] - 10s 318ms/step - loss: 0.5097 - accuracy: 0.7454 - val_loss: 0.4737 - val_accuracy: 0.
7718
Epoch 4/10
30/30 [==============================] - 10s 319ms/step - loss: 0.4176 - accuracy: 0.8074 - val_loss: 0.4689 - val_accuracy: 0.
7785
Epoch 5/10
30/30 [==============================] - 10s 320ms/step - loss: 0.3482 - accuracy: 0.8275 - val_loss: 0.4612 - val_accuracy: 0.
7919
Epoch 6/10
30/30 [==============================] - 10s 321ms/step - loss: 0.2534 - accuracy: 0.8978 - val_loss: 0.4556 - val_accuracy: 0.
8054
Epoch 7/10
30/30 [==============================] - 10s 321ms/step - loss: 0.1619 - accuracy: 0.9430 - val_loss: 0.5101 - val_accuracy: 0.
8188
Epoch 8/10
30/30 [==============================] - 10s 325ms/step - loss: 0.1290 - accuracy: 0.9514 - val_loss: 0.5876 - val_accuracy: 0.
8121
Epoch 9/10
30/30 [==============================] - 10s 326ms/step - loss: 0.1076 - accuracy: 0.9698 - val_loss: 0.5951 - val_accuracy: 0.
7852
Epoch 10/10
30/30 [==============================] - 10s 325ms/step - loss: 0.0578 - accuracy: 0.9799 - val_loss: 0.8307 - val_accuracy: 0.
8054
```

Fig. 3. Model Fit Result

A. Plotting the Results

The obtained results after different iterations are plotted using matplotlib library. Validation and training accuracy as well as loss both are plotted. The result is given in fig 4

VIII. Fine Tuning for Optimization

Now the standard model we have created will be fine tuned. To accomplish this, first of all, model is made trainable. Also we will check the number of layers within the model. There will be total of 10 layers within the model. We will fine tune the model from 5th layer. The commands used for the same are given as under
model.trainable = True
print("Number of layers in the base model: ", len(model.layers))
fine_tune_at = 5
for layer in model.layers[:fine_tune_at]:
layer.trainable = False
Model is compiled again and result is obtained. The model summary is given in fig 5

A. Fine Tuning Model Fit

After fine tuning the model, it is compiled again along with fitting operation. The model shows optimization in terms of

Fig. 4. Model Validation and training accuracy and loss

Model: "sequential_4"

Layer (type)	Output Shape	Param #
rescaling_6 (Rescaling)	(None, 200, 200, 3)	0
conv2d_9 (Conv2D)	(None, 200, 200, 16)	448
max_pooling2d_9 (MaxPooling2	(None, 100, 100, 16)	0
conv2d_10 (Conv2D)	(None, 100, 100, 32)	4640
max_pooling2d_10 (MaxPooling	(None, 50, 50, 32)	0
conv2d_11 (Conv2D)	(None, 50, 50, 64)	18496
max_pooling2d_11 (MaxPooling	(None, 25, 25, 64)	0
flatten_3 (Flatten)	(None, 40000)	0
dense_6 (Dense)	(None, 128)	5120128
dense_7 (Dense)	(None, 2)	258

Total params: 5,143,970
Trainable params: 5,138,882
Non-trainable params: 5,088

Fig. 5. Fine tune model summary

result. Number of epochs or iterations for this model are 20. The code for the same is given as under

```
fine_tune_epochs = 10
total_epochs = epochs + fine_tune_epochs
history = model.fit(
train_ds,
validation_data=val_ds,
epochs=total_epochs
)
```

B. Fine tune Model Result

The result in terms of validation and training accuracy reaches 100%. The result is given within fig 6 since the size is too big so only first thirteen iterations are visible.

```
Epoch 1/20
30/30 [==============================] - 17s 224ms/step - loss: 0.1418 - accuracy: 0.9531 - val_loss: 0.7959 - val_accuracy: 0.
7785
Epoch 2/20
30/30 [==============================] - 5s 151ms/step - loss: 0.0271 - accuracy: 1.0000 - val_loss: 0.8318 - val_accuracy: 0.7
785
Epoch 3/20
30/30 [==============================] - 5s 161ms/step - loss: 0.0115 - accuracy: 0.9983 - val_loss: 0.8778 - val_accuracy: 0.7
987
Epoch 4/20
30/30 [==============================] - 4s 150ms/step - loss: 0.0072 - accuracy: 0.9983 - val_loss: 0.9362 - val_accuracy: 0.8
054
Epoch 5/20
30/30 [==============================] - 4s 148ms/step - loss: 0.0086 - accuracy: 0.9966 - val_loss: 1.0120 - val_accuracy: 0.7
852
Epoch 6/20
30/30 [==============================] - 4s 148ms/step - loss: 0.0125 - accuracy: 0.9966 - val_loss: 1.0011 - val_accuracy: 0.7
852
Epoch 7/20
30/30 [==============================] - 5s 157ms/step - loss: 0.0058 - accuracy: 1.0000 - val_loss: 1.0942 - val_accuracy: 0.7
919
Epoch 8/20
30/30 [==============================] - 5s 152ms/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 1.0873 - val_accuracy: 0.8
121
Epoch 9/20
30/30 [==============================] - 4s 147ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 1.0980 - val_accuracy: 0.7
852
Epoch 10/20
30/30 [==============================] - 4s 149ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 1.1478 - val_accuracy: 0.7
987
Epoch 11/20
30/30 [==============================] - 5s 156ms/step - loss: 8.9015e-04 - accuracy: 1.0000 - val_loss: 1.1311 - val_accuracy:
0.8121
```

Fig. 6. Fine tune model result

IX. Testing the Model

After the model is compiled, model is tested by presenting the individual image from the dataset.The model is tested using the fig 7 The result is predicted with 100% confidence

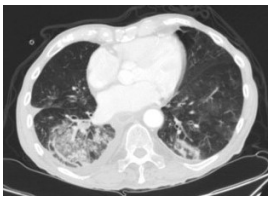

Fig. 7. Fine tune model result

as predicted. The result is as under This image most likely belongs to Non_COVID with a 100.00 percent confidence.

X. Conclusion

Covid 19 is pandemic that affected almost every country. Its early detection and prediction could be crucial for decreasing the devastating affect. In this work deep learning mechanism with fine tuning is used for predicting the Covid cases with 100% accuracy. The validation accuracy and testing accuracy after fine tuning is increased and hence worth of study is proved. in addition, validation and training loss is also decreased. The dataset is accessed from local drive. The Validation and test folders does not exists within the dataset and has to be defined manually. This could be an issue that must be rectified in future.

References

[1] TensorFlow, "Load and preprocess images — TensorFlow Core," Website, 2021. https://www.tensorflow.org/tutorials/load_data/images (accessed Sep. 12, 2021).

[2] udacity, "How to Process Images With TensorFlow — Udacity," Website, 2020. https://www.udacity.com/blog/2018/04/how-to-process-images-with-tensorflow.html (accessed Sep. 12, 2021).

[3] StackAbuse, "Image Recognition in Python with TensorFlow and Keras," Website, 2021. https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/ (accessed Sep. 12, 2021).

[4] Prabhu, "A Simple Tutorial to Classify Images Using TensorFlow — Step by Step Guide — by Prabhu — Medium," Website, 2021. https://medium.com/@RaghavPrabhu/a-simple-tutorial-to-classify-images-using-tensorflow-step-by-step-guide-7e0fad26c22 (accessed Sep. 12, 2021).

[5] TensorFlow, “Image classification — TensorFlow Core,” Website, 2021. https://www.tensorflow.org/tutorials/images/classification (accessed Sep. 12, 2021).