PROJECT REPORT

Machine Learning Lab (AIL-302)

<u>COVID 19 LUNG INFECTION DETECTION</u>

BS(AI)-05

Name	Enrollment
HAFSA HAFEEZ SIDDIQUI	02-136212-026

Submitted to:

<u>Miss Zahida Naz</u>

BAHRIA UNIVERSITY KARACHI CAMPUS

Department of Computer Science

Contents

ABSTRACT 3
INTRODUCTION 4
PROJECT SCOPE 4
PROBLEM STATEMENT 4
METHODOLOGY 5
  Data Handling 5
  Creating the Model 5
  Training & Testing 5
CODE 6
  Data Handling 6
  Model Definition 10
  Configuration & Training 12
  Testing 15
OUTPUT 16
FUTURE DEVELOPMENT 17
CONCLUSION 17

ABSTRACT

The emergence of machine learning and deep learning techniques has transformed the discipline of medical image analysis, providing promising solutions to the issues faced by the COVID-19 epidemic. Using a Convolutional Neural Network (CNN) and transfer learning from a pre-trained VGG model, this experiment proposes a thorough workflow for segmenting COVID-19-related lung infections from CT scans.

The developed model demonstrates robust performance, achieving an accuracy of 92.20% on a test dataset. The integration of efficient data handling, model definition, and configuration parameters contributes to the creation of a reliable tool for aiding healthcare professionals in the diagnosis of COVID-19 through CT imaging.

INTRODUCTION

The COVID-19 pandemic has underscored the critical need for advanced diagnostic tools to facilitate prompt and accurate identification of lung infections. Computed Tomography (CT) scans have emerged as valuable resources in this context, providing detailed insights into pulmonary abnormalities associated with the virus. Leveraging machine learning, this study introduces a novel approach to automate the segmentation of COVID-19-related lung infections from CT images. The workflow includes robust data handling procedures, model definitions encompassing a CNN and a transfer learning-based VGG model, and meticulous configuration for effective training. The dataset at hand originates from real patients in hospitals located in Sao Paulo, Brazil, and is intended to serve as a catalyst for the advancement of artificial intelligence methodologies.

PROJECT SCOPE

The goal of this project was to create a segmentation system that can distinguish between Covid and Non-Covid regions in CT scan pictures. Initially, the project intended to gather labelled CT scan datasets from Kaggle or Zenodo; however, due to its thorough labelling of Covid and Non-Covid cases, the project shifted to the SARS-COV-2 Ct-Scan Dataset. The described process entails painstaking data preparation, which includes cleaning, splitting, and augmentation with the ImageDataGenerator class from the Keras package. Notably, a CNN architecture with a pretrained VGG model was adopted for the segmentation model. The inclusion of a pretrained model like VGG demonstrates the project's emphasis on exploiting existing characteristics learnt from huge datasets to improve the accuracy of the segmentation system. To support complete model evaluation, the dataset was carefully separated into training, testing, and validation sets.

PROBLEM STATEMENT

The primary goal is to facilitate the research and development of models capable of accurately identifying SARS-CoV-2 infection through the analysis of CT scans. Specifically, the focus lies on enhancing COVID-19 diagnosis by effectively segmenting infections within patient lung images derived from CT scans. The dataset is structured for binary classification, categorizing patients into two classes: those infected with Covid-19 and those not infected.

This project poses a critical challenge in the realm of medical image analysis and artificial intelligence, as the ability to accurately and efficiently identify COVID-19 infections through automated CT scan analysis can significantly aid healthcare professionals in timely and precise diagnosis. Success in this endeavor holds the potential to streamline diagnostic processes and contribute to the ongoing global efforts to combat the COVID-19 pandemic.

METHODOLOGY

This section explores the step-by-step methodology employed in the development and training of our machine learning models for the identification of COVID-19 infections through the analysis of CT scan images.

Data Handling

The data handling process employed several key libraries, including sklearn.model_selection for train-test splitting, and pandas for efficient data manipulation. The **define_paths()** function was esstential in organizing and gathering file paths along with their corresponding labels from a directory structure. Through the utilization of the **split_data()** function, the lung infection images were categorized into training, testing, and validation sets. To facilitate data preparation, the Keras library, a powerful deep learning API built on TensorFlow, was used. Particularly, the ImageDataGenerator class from Keras played a pivotal role in the creation of data generators. These generators proved essential for handling large datasets that do not fit into memory, enabling on-the-fly data augmentation, normalization, and batching during the neural network training process.

Creating the Model

To construct the models, we employed convolutional neural network (CNN) architectures using the Keras library, specialized for image datasets. One notable component of our approach involved the utilization of a pretrained VGG model, a type of CNN architecture renowned for its effectiveness in image-related tasks. We chose an efficient approach by using a pretrained model due to the resource-intensive nature of training deep networks with a high number of parameters. Additionally, we employed a Classifier class to define a simpler CNN network using Keras' sequential API. This model consists of convolutional layers, which play a pivotal role in capturing intricate patterns in the image dataset. Activation functions, particularly Rectified Linear Unit (ReLU), are strategically applied to introduce non-linearity and enable the model to discern complex patterns.

Training & Testing

Using the Classifier class defined earlier when creating the model, will be now used to train the model and we inputted the configuration parameters for the model. The fit method is used to train the model. It takes the training and validation generators, the number of steps per epoch, the number of epochs, and the specified callbacks. The training progress and performance metrics are stored in the history variable. Two callback functions are defined. ModelCheckpoint saves the model with the lowest validation loss during training. EarlyStopping stops the training if the validation loss does not decrease for a specified number of epochs (patience). To test the model ImageDataGenerator is initialized and using the saved VGG model.

CODE

Data Handling

```
import os

from sklearn.model_selection import train_test_split

import pandas as pd


def define_paths(data_dir):

    filepaths = []

    labels = []


    folds = os.listdir(data_dir)

    for fold in folds:

        foldpath = os.path.join(data_dir, fold)

        filelist = os.listdir(foldpath)

        for file in filelist:

            fpath = os.path.join(foldpath, file)

            filepaths.append(fpath)

            labels.append(fold)


    return filepaths, labels



# Concatenate data paths with labels into one dataframe ( to later be fitted into the model )

def define_df(files, classes):

    Fseries = pd.Series(files, name= 'filepaths')

    Lseries  =  pd.Series(classes,  name='labels')

    return pd.concat([Fseries, Lseries], axis= 1)
```

```
# Split dataframe into train, valid, and test
def split_data(data_dir):
    # create train dataframe
    files, classes = define_paths(data_dir)
    df = define_df(files, classes)

    strat = df['labels']
    # split the whole dataset into train and non-train dataframes
    train_df, dummy_df = train_test_split(df, train_size=0.7, shuffle=True, random_state=101,
stratify=strat)

    # from the non-train dataset, create validation and test dataframes
    strat = dummy_df['labels']
    validation_df, test_df = train_test_split(dummy_df, train_size=0.5, shuffle=True, random_state=101,
stratify=strat)

    return train_df, validation_df, test_df

def data_generators(data_dir, img_size, batch_size, class_mode, color_mode):

    from keras.preprocessing.image import ImageDataGenerator

    train_df, val_df, test_df = split_data(data_dir)

    # model input is taken ONLY from imagedatagenerator
    # initializing the imagedatagenerator class
    training_data_generator = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    fill_mode='nearest'
    )

    testing_data_generator = ImageDataGenerator(
        rescale=1./255      # normalisation of images between 0 and 1 from 0 to 255 pixels
    )

    # initialize training, validation and testing generators
    train_generator = training_data_generator.flow_from_dataframe(
        train_df,
        x_col='filepaths',
        y_col='labels',
        target_size=img_size,
        batch_size=batch_size,
        class_mode=class_mode,
        color_mode=color_mode,
        shuffle=True,
    )

    val_generator = testing_data_generator.flow_from_dataframe(
        val_df,
        x_col='filepaths',
        y_col='labels',
        target_size=img_size,
        batch_size=batch_size,
        class_mode=class_mode,
        color_mode=color_mode,
        shuffle=True,
    )

    test_generator = testing_data_generator.flow_from_dataframe(
        test_df,
        x_col='filepaths',
        y_col='labels',
        target_size=img_size,
        batch_size=batch_size,
        class_mode=class_mode,
        color_mode=color_mode,
        shuffle=False,
    )

    return train_generator, val_generator, test_generator
```

Model Definition

```
from keras.layers import *

from keras.models import *

from keras.optimizers import Adam



# defining Convolutional Neural Network (CNN)

# after initializing the imagedatagenerator


class Classifier:

  def get_model(input_shape):

    model = Sequential() # runs the layers in sequence

    model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=input_shape))

    model.add(Conv2D(64, (3,3), activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Dropout(0.25)) # used to reduce overfitting


    model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Dropout(0.25))


    model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Dropout(0.25))


    model.add(Flatten())

    model.add(Dense(64, activation='relu'))

    model.add(Dropout(0.5))

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

```
        model.compile(optimizer=Adam(),  loss='categorical_crossentropy',  metrics=['accuracy'])

        return model

    def VGG_model(input_shape):         # VERY LARGE CNN MODEL ON WHICH WE DID TRANSFER
LEARNING
                                         # pre-trained model
        from keras.applications import VGG16

        num_classes=2

        # VGG_model HAS BEEN TRAINED ON imagenet DATASET
        vgg = VGG16(input_shape=input_shape, weights = 'imagenet', include_top = False)

        for layer in vgg.layers:   # only using feature extractor, every layer will remain the same
            layer.trainable = False

        x = Flatten()(vgg.output)   # using our classification layer
        x = Dense(128, activation = 'relu')(x)
        x = Dense(64, activation = 'relu')(x)
        x = Dense(num_classes, activation = 'softmax')(x)

        model = Model(inputs = vgg.input, outputs = x)

        model.compile(optimizer=Adam(),  loss='categorical_crossentropy',  metrics=['accuracy'])

        return model
```

Configuration & Training

```
from keras.callbacks import ModelCheckpoint, EarlyStopping


# setting only relative path to get dataset, not actually getting the dataset

data_dir = '/content/drive/MyDrive/dataset'


# setting only relative path to save / retrieve models

# (because model will not be trained everytime you want to use it)

model_dir = '/content/drive/MyDrive/models'



# configuration parameters for the classifier we created:

# INPUT_SHAPE = (128,128,1)                 # input shape of model

# model = Classifier.get_model(INPUT_SHAPE)

# COLOR_MODE = 'greyscale'                  # read the images as greyscale - 1 channel

# CLASS_MODE = 'categorical'                # softmax layer at the output of the model

# model_path = '/model.h5'                     # this path will be used to save the model

# MODEL_NAME = 'CLASSY'



# configuration parameters for the VGG model (used for transfer learning):

INPUT_SHAPE = (224,224,3)                   # input shape of model

model = Classifier.VGG_model(INPUT_SHAPE)

COLOR_MODE = 'rgb'                          # read the images as colored - 3 channels because VGG model is
trained on colored images even though our dataset images are greyscale

CLASS_MODE = 'categorical'                  # softmax layer at the output of the model

model_path = '/vgg_model.h5'                  # this path will be used to save the model

MODEL_NAME = 'VGG'
```

```
# input_shape of model (entry layer) and image size should be the same

# INPUT_SHAPE[0] = 224 , INPUT_SHAPE[1] = 224

img_size = (INPUT_SHAPE[0],INPUT_SHAPE[1])


# approximated by dividing the number of images in the training set for a single iteration

# when all the batches are completed running that is equal to total number of images which is equal to 1
epoch

batch_size = 32


# get ImageDataGenerators.flow_from_dataframe - get training, validation, and testing data generators

train_generator, validation_generator, test_generator = \

    data_generators(data_dir=data_dir, img_size=img_size, batch_size=batch_size,
class_mode=CLASS_MODE, color_mode=COLOR_MODE)


# defining model


# Callback functions for monitoring, training, and saving purposes of the fitting of the model:

# checkpoint and early_stopping functions w.r.t val_loss by default


# saves the model if the validation loss is lowest

# verbose: Mode 0 is silent, and mode 1 displays messages when the callback takes an action

checkpoint = ModelCheckpoint(model_dir+model_path, verbose=1, save_best_only=True)


# stops the training if the validation loss does not decrease constantly

# patience: Number of epochs with no improvement after which training will be stopped.

early_stopping = EarlyStopping( patience=8)
```

```
###################################### TRAIN THE MODEL
######################################

# train the model using fit_generator

# after defining the CNN model

history = model.fit(

    train_generator,

    steps_per_epoch=len(train_generator),

    epochs=25,

    validation_data=validation_generator,

    validation_steps=len(validation_generator),

    callbacks=[checkpoint, early_stopping]

)
```

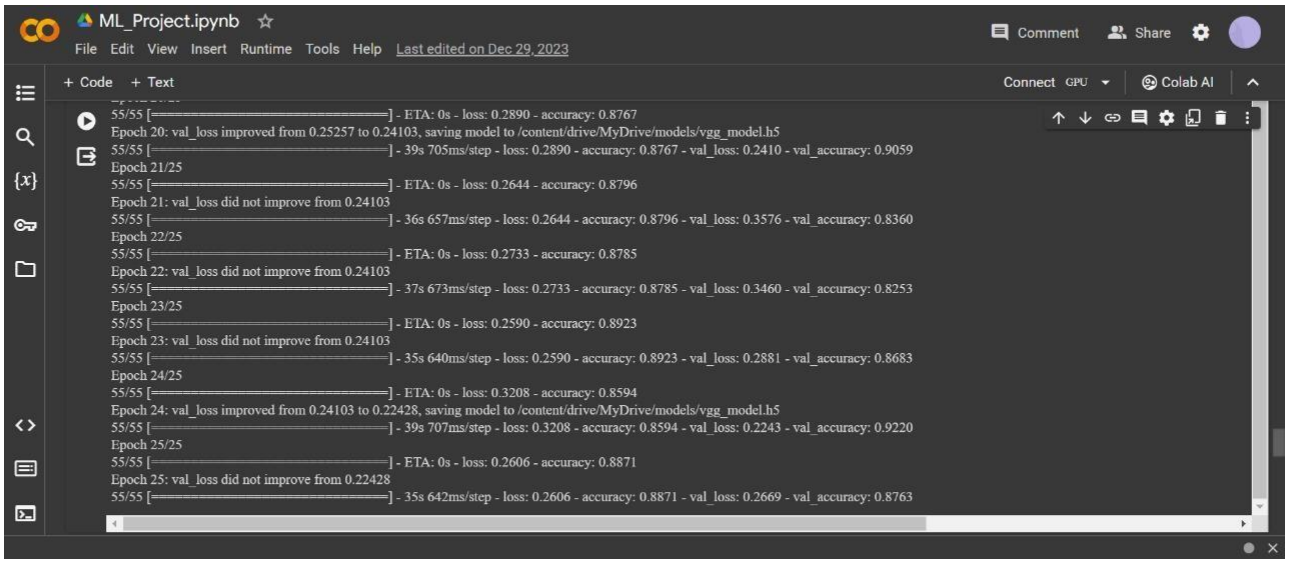
Testing

```
import keras

from keras.preprocessing.image import ImageDataGenerator


data_dir = '/content/drive/MyDrive/dataset'

model_dir = '/content/drive/MyDrive/models/vgg_model.h5'


_, test_df, _ = split_data(data_dir)



testing_data_generator = ImageDataGenerator(

        rescale=1./255,

    )


test_generator = testing_data_generator.flow_from_dataframe(

    test_df,

    x_col='filepaths',

    y_col='labels',

    target_size=(224, 224),

    batch_size=32,

    class_mode='categorical',

    color_mode='rgb',

    shuffle=False,

)

model = keras.models.load_model(model_dir)

loss, acc = model.evaluate(test_generator)

print('\n\n')

print('Saved VGG Model, accuracy: {:5.2f}%'.format(100*acc))

print('\n')
```

OUTPUT

The accuracy of the model was assessed through the validation set. The model underwent 25 epoch cycles during training without early stopping.

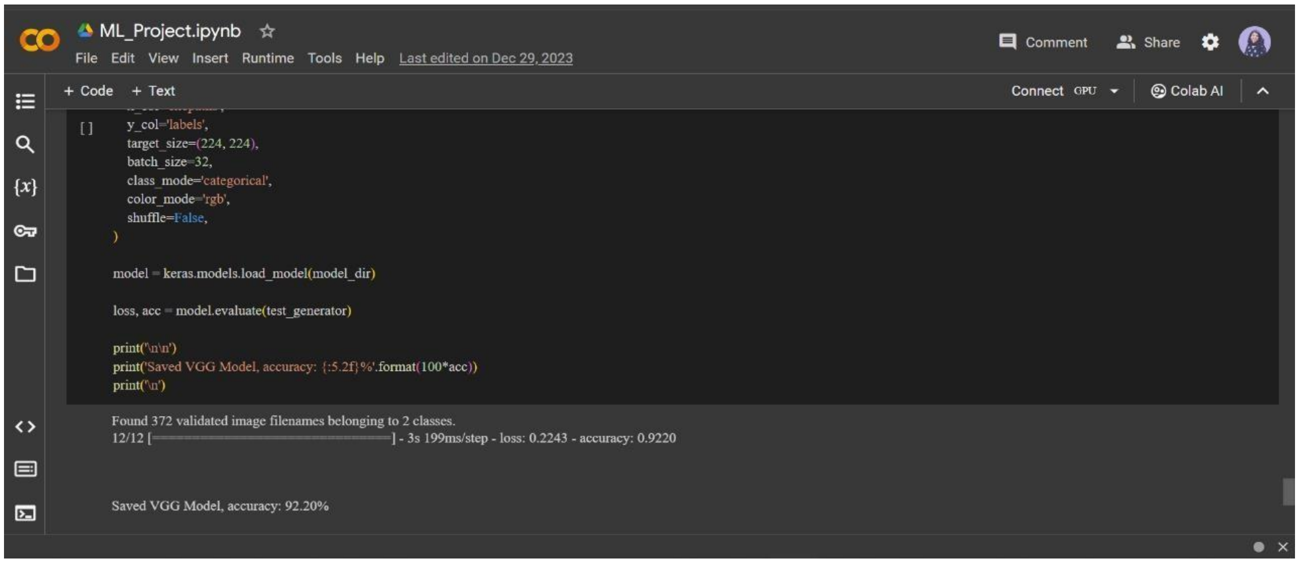
The training accuracy is given as 87.63%.


ML_Project.ipynb
File Edit View Insert Runtime Tools Help Last edited on Dec 29, 2023
Comment Share
+ Code + Text
Connect GPU
Colab AI

```
55/55 [==============================] - ETA: 0s - loss: 0.2890 - accuracy: 0.8767
Epoch 20: val_loss improved from 0.25257 to 0.24103, saving model to /content/drive/MyDrive/models/vgg_model.h5
55/55 [==============================] - 39s 705ms/step - loss: 0.2890 - accuracy: 0.8767 - val_loss: 0.2410 - val_accuracy: 0.9059
Epoch 21/25
55/55 [==============================] - ETA: 0s - loss: 0.2644 - accuracy: 0.8796
Epoch 21: val_loss did not improve from 0.24103
55/55 [==============================] - 36s 657ms/step - loss: 0.2644 - accuracy: 0.8796 - val_loss: 0.3576 - val_accuracy: 0.8360
Epoch 22/25
55/55 [==============================] - ETA: 0s - loss: 0.2733 - accuracy: 0.8785
Epoch 22: val_loss did not improve from 0.24103
55/55 [==============================] - 37s 673ms/step - loss: 0.2733 - accuracy: 0.8785 - val_loss: 0.3460 - val_accuracy: 0.8253
Epoch 23/25
55/55 [==============================] - ETA: 0s - loss: 0.2590 - accuracy: 0.8923
Epoch 23: val_loss did not improve from 0.24103
55/55 [==============================] - 35s 640ms/step - loss: 0.2590 - accuracy: 0.8923 - val_loss: 0.2881 - val_accuracy: 0.8683
Epoch 24/25
55/55 [==============================] - ETA: 0s - loss: 0.3208 - accuracy: 0.8594
Epoch 24: val_loss improved from 0.24103 to 0.22428, saving model to /content/drive/MyDrive/models/vgg_model.h5
55/55 [==============================] - 39s 707ms/step - loss: 0.3208 - accuracy: 0.8594 - val_loss: 0.2243 - val_accuracy: 0.9220
Epoch 25/25
55/55 [==============================] - ETA: 0s - loss: 0.2606 - accuracy: 0.8871
Epoch 25: val_loss did not improve from 0.22428
55/55 [==============================] - 35s 642ms/step - loss: 0.2606 - accuracy: 0.8871 - val_loss: 0.2669 - val_accuracy: 0.8763
```

Figure i - Training Accuracy

Final Accuracy of accuracy of the VGG Model is 92.20%.


ML_Project.ipynb
File Edit View Insert Runtime Tools Help Last edited on Dec 29, 2023
Comment Share
+ Code + Text
Connect GPU
Colab AI

```
    y_col='labels',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=False,
)

model = keras.models.load_model(model_dir)

loss, acc = model.evaluate(test_generator)

print('\n\n')
print('Saved VGG Model, accuracy: {:5.2f}%'.format(100*acc))
print('\n')
```

```
Found 372 validated image filenames belonging to 2 classes.
12/12 [==============================] - 3s 199ms/step - loss: 0.2243 - accuracy: 0.9220



Saved VGG Model, accuracy: 92.20%
```

Figure ii - Final Model Accuracy

FUTURE DEVELOPMENT

There are various opportunities for further improvement and refinement of the current paradigm as we move forward. For starters, using larger and more diverse datasets, such as differences in imaging methods and patient demographics, can improve the model's resilience and generalizability. Exploring other pre-processing techniques and modifying the CNN's architecture may increase the model's accuracy even further.

The model's real-time deployment in clinical situations is an important future concern. Integration with healthcare information systems and collaboration with medical professionals can help to streamline the diagnostic workflow, resulting in more prompt and informed decisions. Furthermore, ongoing updates and adaptation of the model based on new scientific knowledge and virus variants can assure its relevance in dynamic healthcare contexts.

Investigating the merging of multimodal data, such as clinical information and various imaging modalities, could improve the model's overall diagnostic accuracy. Additionally, as these technologies become more integrated into healthcare procedures, adherence to ethical principles and transparency in model interpretation are critical. Future improvements should priorities the development of tools that are not only accurate, but also ethical and easily interpretable by healthcare practitioners.

CONCLUSION

The developed model, particularly the VGG-based transfer learning approach, exhibits commendable accuracy in the segmentation of COVID-19-induced lung infections. The systematic data handling and model definition contribute to the model's ability to generalize well across diverse datasets. The training process, monitored through checkpoints and early stopping, results in a model achieving 92.20% accuracy on a test dataset. This demonstrates the potential of machine learning in enhancing the diagnostic capabilities of healthcare professionals dealing with COVID-19 cases. The presented framework provides a solid foundation for further advancements in automated CT image analysis for infectious diseases.