TOPIC - COVID-19 detection from Lung Ultrasound Images

Subject - Computer Vision EC-353

Submitted to: Proff. Rajiv Kapoor

Submitted by:

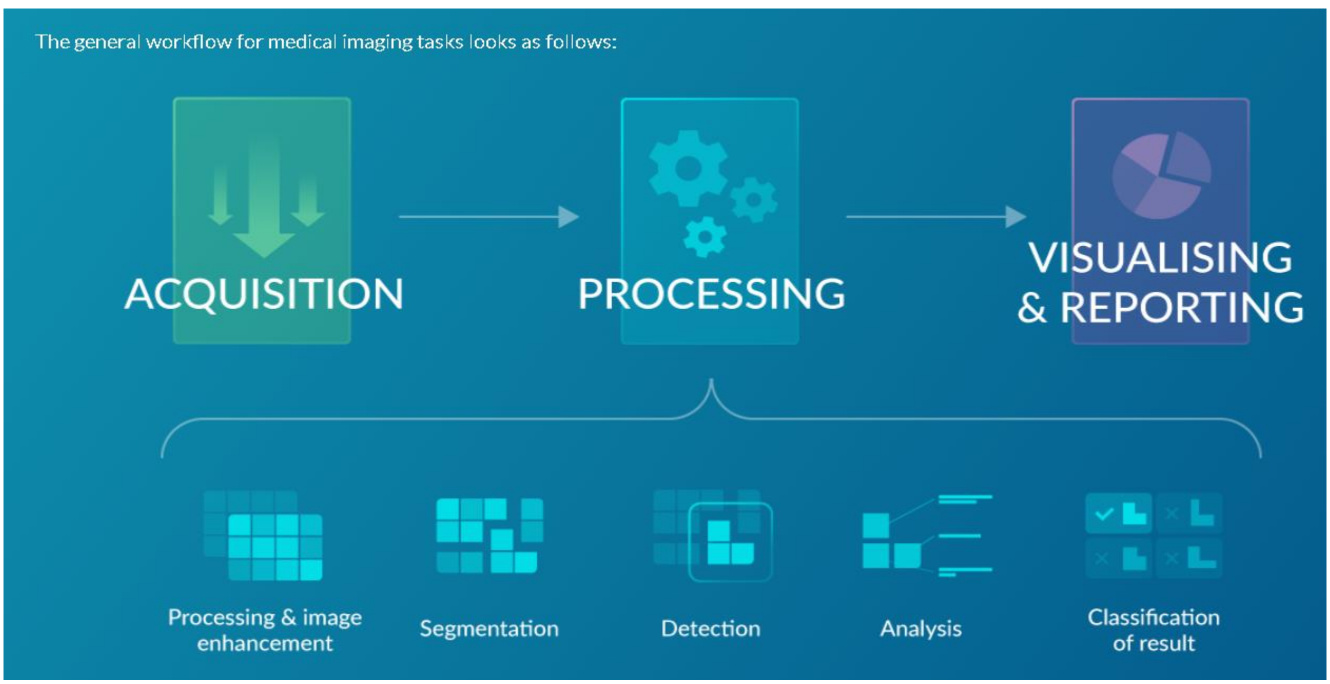
Arkya Bagchi (2K18/PS/014)

Yash Vardhan Gupta (2K18/AE/064)

Delhi Technological University

Delhi 110042

INDEX



The general workflow for medical imaging tasks looks as follows:
ACQUISITION
PROCESSING
VISUALISING & REPORTING
Processing & image enhancement
Segmentation
Detection
Analysis
Classification of result

INTRODUCTION

Abstract

With the rapid growth of COVID-19 into a global pandemic, there is an ever more urgent need for cheap, fast and reliable tools that can assist physicians in diagnosing COVID- 19. Medical imaging such as CT scans, X-ray Imaging and Ultrasound Scans using deep learning, can assist in complementing the traditional diagnostic tools from molecular biology. In this particular project, we discuss the application of POCUS (Point Of Care UltraSonography) in detection of COVID-19.

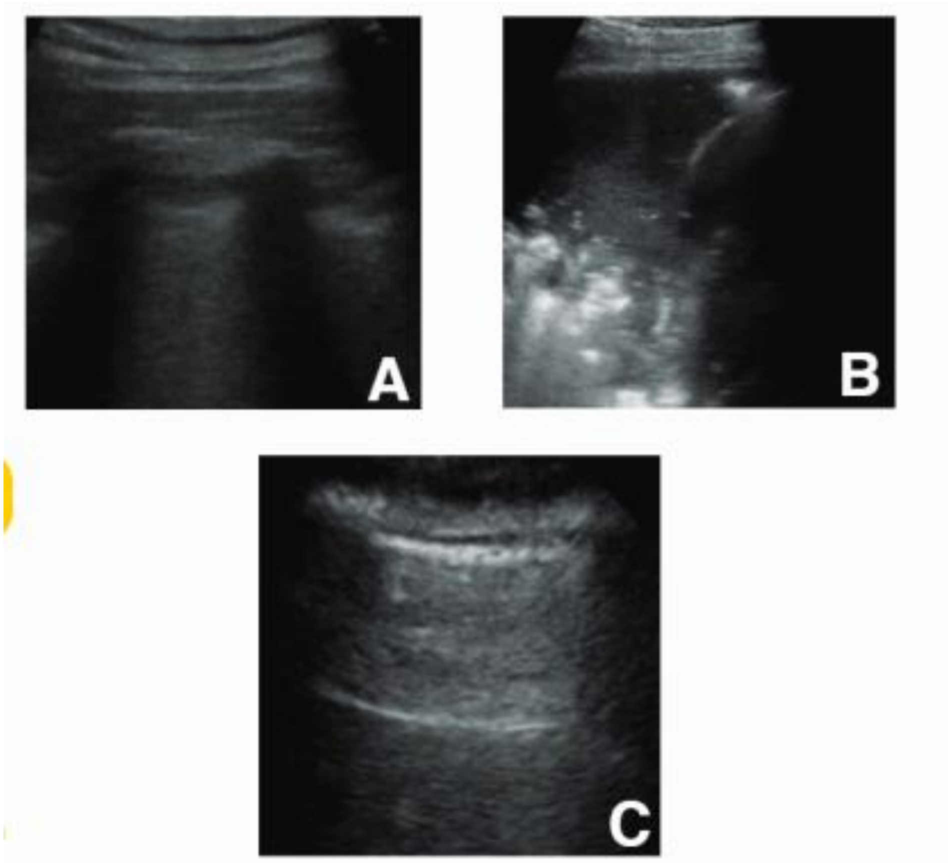
We carry out two major functions. First, we gather a lung ultrasound (POCUS) dataset consisting of about 710 images (amongst which the training set comprises 519 images (240 representing COVID positive and 279 representing healthy lungs) and the testing dataset comprises 191 images (84 representing COVID positive and 107 representing healthy lungs) from 25-30 sampled videos. Secondly, we train 3 deep convolutional neural networks namely VGG-16, MobileNet V2 and Nasnet, on this 2-class dataset and achieve an accuracy of 95.2%, 12.5% and 63.8% respectively.

Conventional Methods vs Biomedical Imaging

The **standard genetic test, reverse transcription polymerase chain reaction (RT-PCR)**, is characterized by high reliability (at least in most countries) but a relatively Jong processing time (more than an hour). Alternatively, fast **serology tests** are in early stages of development, and are based on antibodies that the immune system only produces in an advanced stage of the disease. In this context, **biomedical imaging techniques** have great potential to complement the conventional diagnostic techniques of COVID-19. It is important to note that **lung ultrasound** is already an established method for monitoring pneumonia and related lung diseases, **however they are only recently being used for COVID-19 detection.**

There are numerous strengths of POCUS such as:

- simplicity of execution, ease of repeatability, its noninvasiveness, its execution without relocation and its ease of disinfection at the bedside.
- The devices are small and portable and can be wrapped in single-use plastics to reduce the risk of contamination and promote sterilization procedures.
- Moreover, it is very cost-effective, with an estimated $140 for an examination compared to $370 for chest X-ray and 675 - $8600 for chest CT. The low price of the device itself, starting from 2000, facilitates the distribution to hospitals and primary care centers.
- The diagnostic routine can be accelerated by connecting the device to a cloud service and uploading the recordings automatically.



Example lung ultrasound images of the database

A: A typical COVID-19 infected lung, showing small subpleural consolidation and pleural irregularities. **B**: A pneumonia infected lung, with dynamic air bronchograms surrounded by alveolar consolidation. **C**: Healthy lung which is normally aerated with horizontal A-lines. All images were scraped from publicly available sources.

DATASET GENERATION

We collected the lung ultrasound videos from different internet sources. Since the videos were taken from various sources the format and illuminations differ significantly. In order to generate a diverse and still sufficiently large dataset images were individually processed from each and every video at a frame rate of 5 fps from an online source.

A total of about 710 images (amongst which the training set comprises 519 images (240 representing COVID positive and 279 representing healthy lungs) and the testing/validation dataset comprises 191 images (84 representing COVID positive and 107 representing healthy lungs) were selected from 25-30 sampled videos. We selected out the images of from convex probes since it allows for a better wider field of view. We reduced the problem of class imbalance of the resultant dataset by using 50% COVID and 50% non COVID images.

The image sources are as follows:

- grepmed.com (GrepMed is a community-sourced, searchable medical image repository for referencing clinically relevant medical images)
- thepocusatlas.com (The PocusAtlas is a Collaborative Ultrasound Education Platform)
- butterflynetwork.com (Butterfly is a healthtech company that launched portable US a device needing only single probe usable on the whole body that connects to a smartphone that can reproduce the work of various probes such as linear and convex)
- radiopaedia.org
- everydayultrasound.com
- nephropocus.com

Finalized Ultrasound Images Dataset
https://drive.google.com/drive/folders/1lazpRmQSBfzx8E2QGz7ZOHJzaGLr8QJ9?usp=sharing

METHODOLOGY

First, we use the convolutional part of VGG - 16, an established deep convolutional neural network that has been demonstrated to be successful on various image types. It is followed by one hidden layer of 64 neurons with ReLU activation, dropout of 0.5 and batch normalization; and further by the output layer with softmax activation. The model was pre-trained on Imagenet to extract image features such as shapes and textures.

During training, only the weights of the last three layers were fine-tuned, while the other ones were frozen to the values from pre-training. This results in a total of 262,401 trainable and 134,260,672 non-trainable parameters. The model is trained with a cross entropy loss function on the softmax outputs, and optimized with Adam optimizer.

```
Total params: 134,523,073
Trainable params: 262,401
Non-trainable params: 134,260,672
```

Furthermore, we use data augmentation techniques to diversify the dataset. In explanation, the Keras ImageDataGenerator is used, which applies a series of random transformations on each image when generating a batch. We used up various different augmentation techniques, we allow transformations of the following types: Rotations of up to 10 degrees, horizontal and vertical flips, and shifts of up to I0% of the image height or width respectively. As such transformations can naturally occur with diverse ultrasound devices and recording parameters, augmentation adds valuable and realistic diversity that helps to prevent overfitting.

Then we carried out similar training and testing in other convolutional neural networks such as MobileNetV2 and Nasnet to compare their accuracies.

```
# adding custom layers
vggconv_model.add(Dense(64,activation='relu'))
vggconv_model.add(Dropout(0.5))
vggconv_model.add(BatchNormalization())
vggconv_model.add(Dense(1,activation='sigmoid'))


vggconv_model.compile(loss= keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

RESULTS AND CONCLUSION

Model was trained to classify frames as COVID-19 or healthy. When training the dataset it was made assured that all techniques were followed accurately. We used the classification metrics as 'Accuracy' and used up the Adam optimizer which performs well in our case.

The accuracy is obtained as following: -

1. **VGG16 - 95.2%**
2. **MobileNet V2 - 12.5%**
3. **Nasnet - 63.8%**

It is thus concluded that VGG-16 performed much better than any other models with the highest accurate rate of 95.2%.

We believe that POCUS can provide a quick, easy and low cost method to assess the possibility of a SARS-CoV-2 infection. There are many possibilities that can be extended beyond the scope of this project such as:

- First, an evident improvement of the framework would be to perform inference directly on the videos (e.g. temporal CNNs) instead of the current frame based image analysis. Secondly, the benefit of pre-training the network on large image databases could be improved by training the model on (non lung) ultrasound samples instead of using ImageNet, a database of real life objects. This pre-training may help detecting ultrasound specific patterns such as B-Lines.
- We aim to extend the functionality to a website in the future, to further encourage the community effort of researchers, doctors and companies to build a dataset of POCUS images that can leverage the predictive power of automatic detection systems, and thereby also the value of ultrasound imaging for COVID-19 in general. If the approach turns out to be successful , we plan to build an app as suggested that can enable medical doctors to draw inference from their ultrasound images with unprecedented ease, convenience and speed.

Google Colab Code

https://colab.research.google.com/drive/1XMYu3FCjRbuG3dElGH5mDGPUgCabxpA5?usp=sharing

Code Snippet: VGG-16

Importing the required libraries for our work

```
import os
import numpy as np
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix
import seaborn as sns

import keras
from keras.layers import *
from keras.models import *
from keras.preprocessing import image

import tensorflow as tf
from tensorflow.keras.applications import (VGG16, MobileNetV2, NASNetMobile, ResNet50)
```

training and testing dataset paths

```
[ ] TRAIN_PATH = "/content/drive/MyDrive/computer vision/Ultra-sound dataset/Train"
    VAL_PATH = "/content/drive/MyDrive/computer vision/Ultra-sound dataset/Test"
```

Custom neural network with VGG 16 Model on top

Building the VGG-16 network on top of our custom model and using the convolutional part of the network

```
vgg16model = VGG16()
```

```
[ ] vgg16model.summary()
```

```
[ ] print(type(vgg16model))

    vggconv_model = Sequential()

    # converting the functional model into sequential model and removing the last layer of 1000 classifications
    for layer in vgg16model.layers[:-1]:
      vggconv_model.add(layer)

    print(type(vggconv_model))

    <class 'tensorflow.python.keras.engine.functional.Functional'>
    <class 'tensorflow.python.keras.engine.sequential.Sequential'>
```

```
[ ] # freeze the layers
    for layer in vggconv_model.layers:

      layer.trainable = False
```

```
[ ] # adding custom layers
    vggconv_model.add(Dense(64,activation='relu'))
    vggconv_model.add(Dropout(0.5))
    vggconv_model.add(BatchNormalization())
    vggconv_model.add(Dense(1,activation='sigmoid'))


    vggconv_model.compile(loss= keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

```
vggconv_model.summary()
```

Using the data augmentation techniques to reduce the over-fitting of the model

```
[ ] # train from scratch by creating image data generator

    train_datagen = image.ImageDataGenerator(
        rescale = 1./255,
        rotation_range = 10,
        horizontal_flip = True,
        vertical_flip = True
    )

    # for test dataset only rescaling the images
    test_datagen = image.ImageDataGenerator(rescale = 1./255)
```

```
train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary'
)
```

```
Found 519 images belonging to 2 classes.
```

```
[ ] validation_generator = test_datagen.flow_from_directory(
        VAL_PATH,
        target_size = (224,224),
        batch_size = 32,
        class_mode = 'binary'
    )
```

```
Found 191 images belonging to 2 classes.
```

```
[ ] print(train_generator.class_indices)
    print(validation_generator.class_indices)
```

```
{'Covid': 0, 'Normal': 1}
{'Covid': 0, 'Normal': 1}
```

Training our model with following conditions: steps_epoch as 6, epocs size 20 with 2 validations for each steps

```
hist = vggconv_model.fit(
    train_generator,
    steps_per_epoch=6,
    epochs = 20,
    validation_data = validation_generator,
    validation_steps=2
)
```

```
Epoch 1/20
6/6 [==============================] - 258s 43s/step - loss: 0.7508 - accuracy: 0.5052 - val_loss: 0.4508 - val_accuracy: 0.9062
Epoch 2/20
6/6 [==============================] - 122s 20s/step - loss: 0.5752 - accuracy: 0.7083 - val_loss: 0.3865 - val_accuracy: 0.8906
Epoch 3/20
6/6 [==============================] - 66s 11s/step - loss: 0.5642 - accuracy: 0.7344 - val_loss: 0.4908 - val_accuracy: 0.7344
Epoch 4/20
6/6 [==============================] - 46s 8s/step - loss: 0.5003 - accuracy: 0.7844 - val_loss: 0.3623 - val_accuracy: 0.9062
Epoch 5/20
6/6 [==============================] - 24s 4s/step - loss: 0.6042 - accuracy: 0.6287 - val_loss: 0.4259 - val_accuracy: 0.7656
Epoch 6/20
6/6 [==============================] - 15s 2s/step - loss: 0.5855 - accuracy: 0.7083 - val_loss: 0.4540 - val_accuracy: 0.7812
Epoch 7/20
6/6 [==============================] - 10s 2s/step - loss: 0.5159 - accuracy: 0.7552 - val_loss: 0.3253 - val_accuracy: 0.9844
Epoch 8/20
6/6 [==============================] - 6s 925ms/step - loss: 0.5161 - accuracy: 0.7665 - val_loss: 0.4366 - val_accuracy: 0.9219
Epoch 9/20
6/6 [==============================] - 6s 1s/step - loss: 0.4768 - accuracy: 0.8125 - val_loss: 0.4164 - val_accuracy: 0.8906
Epoch 10/20
6/6 [==============================] - 5s 810ms/step - loss: 0.4699 - accuracy: 0.7844 - val_loss: 0.3598 - val_accuracy: 0.9688
Epoch 11/20
6/6 [==============================] - 5s 764ms/step - loss: 0.5087 - accuracy: 0.8177 - val_loss: 0.4103 - val_accuracy: 0.9062
Epoch 12/20
6/6 [==============================] - 4s 653ms/step - loss: 0.3954 - accuracy: 0.8698 - val_loss: 0.3654 - val_accuracy: 0.9062
Epoch 13/20
6/6 [==============================] - 4s 663ms/step - loss: 0.4406 - accuracy: 0.8229 - val_loss: 0.4082 - val_accuracy: 0.9062
Epoch 14/20
6/6 [==============================] - 4s 616ms/step - loss: 0.4272 - accuracy: 0.8263 - val_loss: 0.3706 - val_accuracy: 0.9375
Epoch 15/20
6/6 [==============================] - 4s 677ms/step - loss: 0.3698 - accuracy: 0.8646 - val_loss: 0.3565 - val_accuracy: 0.9688
Epoch 16/20
6/6 [==============================] - 4s 597ms/step - loss: 0.3676 - accuracy: 0.8683 - val_loss: 0.3593 - val_accuracy: 0.9688
Epoch 17/20
6/6 [==============================] - 5s 856ms/step - loss: 0.4026 - accuracy: 0.8594 - val_loss: 0.3241 - val_accuracy: 0.9688
Epoch 18/20
6/6 [==============================] - 4s 587ms/step - loss: 0.3899 - accuracy: 0.8563 - val_loss: 0.3194 - val_accuracy: 0.9375
Epoch 19/20
6/6 [==============================] - 4s 655ms/step - loss: 0.3841 - accuracy: 0.8594 - val_loss: 0.4300 - val_accuracy: 0.9531
Epoch 20/20
6/6 [==============================] - 4s 680ms/step - loss: 0.4047 - accuracy: 0.8229 - val_loss: 0.4208 - val_accuracy: 0.9219
```

```
[ ] vggconv_model.evaluate(train_generator)

17/17 [==============================] - 9s 530ms/step - loss: 0.4439 - accuracy: 0.8979
[0.4438990652561188, 0.8978805541992188]
```

```
vggconv_model.evaluate(validation_generator)

#yhat_probs = model.predict(validation_generator, verbose=0)
#yhat_classes = model.predict_classes(validation_generator, verbose=0)

6/6 [==============================] - 4s 714ms/step - loss: 0.3964 - accuracy: 0.9529
[0.3964141011238098, 0.9528796076774597]
```

```
[ ] vggconv_model.save("vggconv_model_adv1.h5")
```

```
[ ] model = load_model("vggconv_model_adv1.h5")
```

MOBILE NET V2

```
input_size: tuple = (224, 224, 3)
hidden_size: int = 64
dropout: float = 0.5
num_classes: int = 3
trainable_layers: int = 0
log_softmax: bool = False

def fix_layers(model, num_flex_layers: int = 1):
    """
    Receives a model and freezes all layers but the last num_flex_layers ones.
    Arguments:
        model {tensorflow.python.keras.engine.training.Model} -- model
    Keyword Arguments:
        num_flex_layers {int} -- [Number of trainable layers] (default: {1})
    Returns:
        Model -- updated model
    """
    num_layers = len(model.layers)
    for ind, layer in enumerate(model.layers):
        if ind < num_layers - num_flex_layers:
            layer.trainable = False

    return model

act_fn = tf.nn.softmax if not log_softmax else tf.nn.log_softmax
```

```
[ ] mobilenetmodel = MobileNetV2(weights="imagenet",include_top=False,input_tensor=Input(shape=input_size))
```

```
mobilenetmodel.summary()
```

```
headModel = mobilenetmodel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(hidden_size)(headModel)
headModel = BatchNormalization()(headModel)
headModel = ReLU()(headModel)
headModel = Dropout(dropout)(headModel)
headModel = Dense(num_classes, activation=act_fn)(headModel)

model_mobilenet = Model(inputs=mobilenetmodel.input, outputs=headModel)
model_mobilenet = fix_layers(model_mobilenet, num_flex_layers=trainable_layers + 8)
```

```
model_mobilenet.summary()
```

```
[ ] model_mobilenet.compile(loss= keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

```
# train from scratch

train_datagen = image.ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 10,
    horizontal_flip = True,
    vertical_flip = True
)

test_datagen = image.ImageDataGenerator(rescale = 1./255)

train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary'
)
```

```
Found 519 images belonging to 2 classes.
```

```
[ ] validation_generator = test_datagen.flow_from_directory(
    VAL_PATH,
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary'
)
```

```
Found 191 images belonging to 2 classes.
```

```
print(train_generator.class_indices)
print(validation_generator.class_indices)
```

```
{'Covid': 0, 'Normal': 1}
{'Covid': 0, 'Normal': 1}
```

```
[ ] hist = model_mobilenet.fit(
    train_generator,
    steps_per_epoch=6,
    epochs = 18,
    validation_data = validation_generator,
    validation_steps=2
)
```

```
Epoch 1/18
6/6 [==============================] - 5s 751ms/step - loss: 1.1354 - accuracy: 0.3750 - val_loss: 0.9732 - val_accuracy: 0.1250
Epoch 2/18
6/6 [==============================] - 3s 557ms/step - loss: 1.0521 - accuracy: 0.3333 - val_loss: 1.0010 - val_accuracy: 0.2812
Epoch 3/18
6/6 [==============================] - 3s 535ms/step - loss: 1.0710 - accuracy: 0.3174 - val_loss: 1.0289 - val_accuracy: 0.4531
Epoch 4/18
6/6 [==============================] - 4s 601ms/step - loss: 0.9700 - accuracy: 0.3073 - val_loss: 1.0532 - val_accuracy: 0.5625
Epoch 5/18
6/6 [==============================] - 4s 609ms/step - loss: 0.9606 - accuracy: 0.3054 - val_loss: 0.8695 - val_accuracy: 0.3750
Epoch 6/18
6/6 [==============================] - 4s 602ms/step - loss: 0.9479 - accuracy: 0.2969 - val_loss: 0.8379 - val_accuracy: 0.5312
Epoch 7/18
6/6 [==============================] - 3s 579ms/step - loss: 0.9379 - accuracy: 0.3281 - val_loss: 0.8459 - val_accuracy: 0.4531
Epoch 8/18
6/6 [==============================] - 3s 530ms/step - loss: 0.9556 - accuracy: 0.3533 - val_loss: 0.9017 - val_accuracy: 0.4375
Epoch 9/18
6/6 [==============================] - 3s 516ms/step - loss: 0.9350 - accuracy: 0.3054 - val_loss: 0.8939 - val_accuracy: 0.4375
Epoch 10/18
6/6 [==============================] - 4s 589ms/step - loss: 0.9330 - accuracy: 0.2812 - val_loss: 0.9298 - val_accuracy: 0.6875
Epoch 11/18
6/6 [==============================] - 4s 600ms/step - loss: 0.9289 - accuracy: 0.3281 - val_loss: 0.9114 - val_accuracy: 0.7031
Epoch 12/18
6/6 [==============================] - 3s 529ms/step - loss: 0.9277 - accuracy: 0.3713 - val_loss: 0.8778 - val_accuracy: 0.7969
Epoch 13/18
6/6 [==============================] - 3s 532ms/step - loss: 0.8451 - accuracy: 0.3533 - val_loss: 0.8823 - val_accuracy: 0.7344
Epoch 14/18
6/6 [==============================] - 3s 535ms/step - loss: 0.9447 - accuracy: 0.3114 - val_loss: 0.9124 - val_accuracy: 0.6875
Epoch 15/18
6/6 [==============================] - 3s 537ms/step - loss: 0.8559 - accuracy: 0.2994 - val_loss: 0.8533 - val_accuracy: 0.5938
Epoch 16/18
6/6 [==============================] - 3s 532ms/step - loss: 0.9029 - accuracy: 0.3054 - val_loss: 0.8979 - val_accuracy: 0.2812
Epoch 17/18
6/6 [==============================] - 3s 526ms/step - loss: 0.8721 - accuracy: 0.2814 - val_loss: 0.7997 - val_accuracy: 0.2031
Epoch 18/18
6/6 [==============================] - 3s 577ms/step - loss: 0.8544 - accuracy: 0.3646 - val_loss: 0.8419 - val_accuracy: 0.1406
```

```
print(model_mobilenet.evaluate(train_generator))
print(model_mobilenet.evaluate(validation_generator))
```

```
17/17 [==============================] - 8s 483ms/step - loss: 0.8184 - accuracy: 0.1869
[0.8184353113174438, 0.1868978738784791]
6/6 [==============================] - 1s 216ms/step - loss: 0.8403 - accuracy: 0.1257
[0.8403012752532959, 0.1256544440984726]
```

```
model_mobilenet.save("model_mobilenet_adv1.h5")
```

NASNET

Custom neural network with NasNet mobile Model on top

```
nasnetModel = NASNetMobile(
        weights="imagenet",
        include_top=False,
        input_tensor=Input(shape=input_size)
    )
    # construct the head of the model that will be placed on top of the
    # the base model
headModel = nasnetModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(hidden_size)(headModel)
headModel = ReLU()(headModel)
headModel = Dropout(dropout)(headModel)
headModel = BatchNormalization()(headModel)
headModel = Dense(num_classes, activation=act_fn)(headModel)

# place the head FC model on top of the base model
model_nasnet = Model(inputs = nasnetModel.input, outputs=headModel)

model_nasnet = fix_layers(model_nasnet, num_flex_layers=trainable_layers + 8)
```

```
model_nasnet.compile(loss= keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

```
# train from scratch

train_datagen = image.ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 10,
    horizontal_flip = True,
    vertical_flip = True
)

test_datagen = image.ImageDataGenerator(rescale = 1./255)

train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary'
)
```

```
Found 519 images belonging to 2 classes.
```

```
validation_generator = test_datagen.flow_from_directory(
    VAL_PATH,
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary'
)
```

```
Found 191 images belonging to 2 classes.
```

```
[ ] print(train_generator.class_indices)
    print(validation_generator.class_indices)

    {'Covid': 0, 'Normal': 1}
    {'Covid': 0, 'Normal': 1}
```

```
[ ] hist = model_nasnet.fit(
        train_generator,
        steps_per_epoch=6,
        epochs = 18,
        validation_data = validation_generator,
        validation_steps=2
    )
```

```
Epoch 1/18
6/6 [==============================] - 53s 9s/step - loss: 1.0555 - accuracy: 0.3438 - val_loss: 0.9519 - val_accuracy: 0.3281
Epoch 2/18
6/6 [==============================] - 52s 9s/step - loss: 0.9610 - accuracy: 0.3653 - val_loss: 0.9386 - val_accuracy: 0.2656
Epoch 3/18
6/6 [==============================] - 28s 5s/step - loss: 0.9308 - accuracy: 0.3293 - val_loss: 0.9283 - val_accuracy: 0.3438
Epoch 4/18
6/6 [==============================] - 19s 3s/step - loss: 0.9515 - accuracy: 0.4115 - val_loss: 0.9330 - val_accuracy: 0.5781
Epoch 5/18
6/6 [==============================] - 15s 2s/step - loss: 0.9278 - accuracy: 0.3952 - val_loss: 0.8932 - val_accuracy: 0.2188
Epoch 6/18
6/6 [==============================] - 7s 1s/step - loss: 0.9259 - accuracy: 0.3125 - val_loss: 0.8486 - val_accuracy: 0.2500
Epoch 7/18
6/6 [==============================] - 4s 681ms/step - loss: 0.8751 - accuracy: 0.3353 - val_loss: 0.8712 - val_accuracy: 0.1250
Epoch 8/18
6/6 [==============================] - 3s 531ms/step - loss: 0.8820 - accuracy: 0.2395 - val_loss: 0.7980 - val_accuracy: 0.2188
Epoch 9/18
6/6 [==============================] - 4s 733ms/step - loss: 0.8682 - accuracy: 0.3713 - val_loss: 0.8539 - val_accuracy: 0.2031
Epoch 10/18
6/6 [==============================] - 4s 693ms/step - loss: 0.9081 - accuracy: 0.3906 - val_loss: 0.7774 - val_accuracy: 0.3281
Epoch 11/18
6/6 [==============================] - 3s 578ms/step - loss: 0.8926 - accuracy: 0.3698 - val_loss: 0.8137 - val_accuracy: 0.3438
Epoch 12/18
6/6 [==============================] - 3s 512ms/step - loss: 0.8604 - accuracy: 0.3653 - val_loss: 0.7779 - val_accuracy: 0.3438
Epoch 13/18
6/6 [==============================] - 3s 510ms/step - loss: 0.8997 - accuracy: 0.3353 - val_loss: 0.7888 - val_accuracy: 0.4375
Epoch 14/18
6/6 [==============================] - 3s 574ms/step - loss: 0.8744 - accuracy: 0.3542 - val_loss: 0.8213 - val_accuracy: 0.5000
Epoch 15/18
6/6 [==============================] - 3s 570ms/step - loss: 0.8741 - accuracy: 0.3177 - val_loss: 0.8557 - val_accuracy: 0.5625
Epoch 16/18
6/6 [==============================] - 4s 585ms/step - loss: 0.8152 - accuracy: 0.3490 - val_loss: 0.8077 - val_accuracy: 0.6719
Epoch 17/18
6/6 [==============================] - 4s 587ms/step - loss: 0.7852 - accuracy: 0.2865 - val_loss: 0.8694 - val_accuracy: 0.7500
Epoch 18/18
6/6 [==============================] - 4s 591ms/step - loss: 0.8138 - accuracy: 0.4427 - val_loss: 0.8026 - val_accuracy: 0.5781
```

```
print(model_nasnet.evaluate(train_generator))
print(model_nasnet.evaluate(validation_generator))
```

```
17/17 [==============================] - 8s 468ms/step - loss: 0.7977 - accuracy: 0.5376
[0.7977374196052551, 0.5375722646713257]
6/6 [==============================] - 1s 205ms/step - loss: 0.8281 - accuracy: 0.6387
[0.8280627727508545, 0.6387434601783752]
```