CS6005
Deep Learning
Mini Project – Transfer Learning
Covid-19 Detection from Chest Xray Images using Resnet

Date: 11/05/2021

Sayf Hussain Z
2018103059
P - Batch

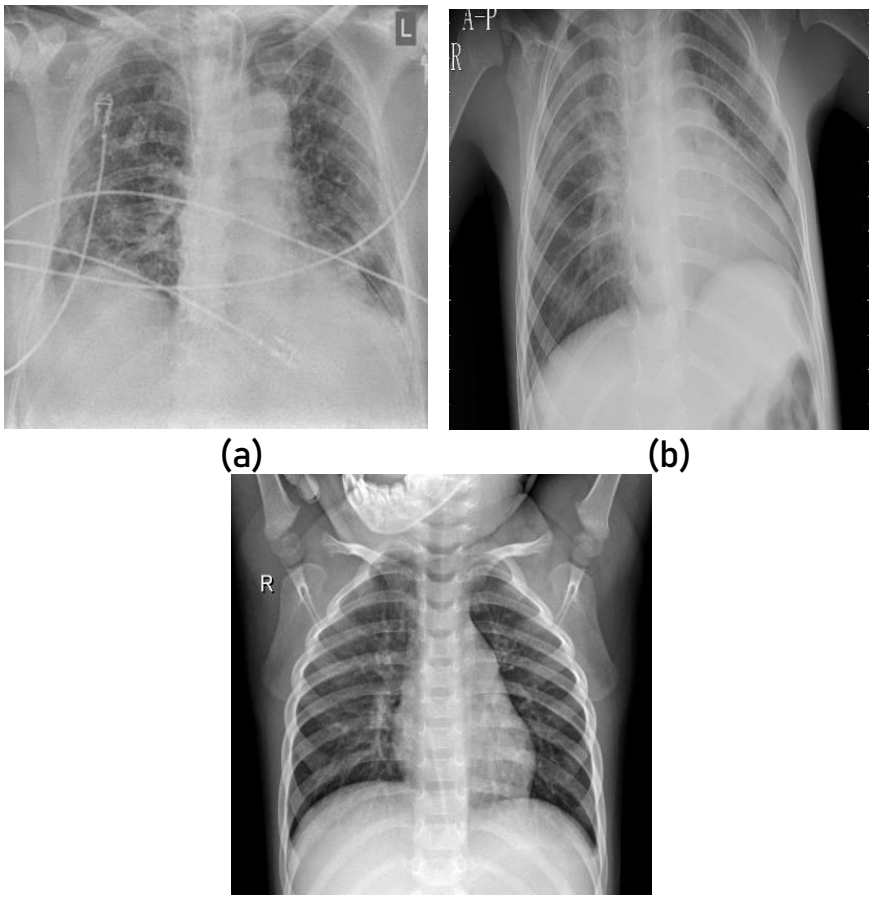
Problem statement:

The aim of this project is to use Transfer Learning to fine tune a model to classify chest X-ray images into that of a patient suffering from Covid-19, viral pneumonia or that of a healthy patient. For this purpose, a suitable dataset of chest X-ray images was identified. Resnet152 trained on the Imagenet dataset is used as the base model to be fine tuned to better adapt to the dataset at hand.

Dataset details:

The dataset used is "COVID-19 Radiography Database" curated by a team of re-searchers from Qatar University, Doha, Qatar, and the University of Dhaka, Bangladesh along with their collaborators from Pakistan and Malaysia [1]. The database contains a total of 3,886 CXR images for Covid-19 patients along with healthy and viral pneumonia patients. There are 1,200 Covid-19 CXR images, 1,341 CXR images for healthy patients and 1,345 CXR images for viral pneumonia patients.

Category	Number of Images
Covid – 19	1200
Viral pneumonia	1345
Healthy	1341



(a) (b)

(c)

(a) Chest X-ray of a Covid-19 patient (b) Chest X-ray of a viral pneumonia patient (c) Chest X-ray of a healthy patient

Model:

ResNet152

ResNet152 is a 152 layer Convolutional Neural Network model proposed by Kaiming He et. Al in the paper Deep Residual Learning for Image Recognition [2].

Researchers have found that while training very deep neural networks, we run into problems of exploding or vanishing gradients. The problem of training very deep neural network layers can be alleviated by the ResNet architecture. ResNet introduced a shortcut or a skip connection that skips one or more layers. The architecture says that if the deeper layers are found to hurt the model's performance, then it is always easy to push the residuals to 0 (F(x)=0). If the deeper layers are found to be useful, then the weights of the layers would be non-zero values and the model performance could increase slightly.

Modules:

Reading the data:

The dataset is divided into 2 directories Train and Test directory. Within the 2 directories the images are stored class wise into the folders COVID, NORMAL and VIRAL PNEUMONIA. The images are read using Imagedata generator. Class 0, 1 and 2 represent classes Covid, Normal patients and Viral Pneumonia patients respectively.

Processing the data:

The images are of varied sizes. While reading the images using imagedatagenerator, the images are all resized to 224 x 224 across all 3 channels. Data Normalization techniques were then used to improve computational efficiency. This was achieved by scaling down each individual pixel by a factor of 255 and then setting the input mean to 0 over the entire dataset for different features. The input images are also randomly zoomed in by a factor of 0.07.

Importing the model:

The model used is Resnet152 and the imagenet pretrained weights were loaded onto the model. While loading the model, the include_top parameter was set to false so as to import only the feature extractor part of the model. On top of this, 3 additional fully connected layers with 512, 256 and 64 nodes were added. All the Fully connected layers had a dropout of 0.3 and were followed by a Batch normalization layer. The initial layers of the model were frozen and only the last few layers were retrained to better adapt to the dataset at hand.

Training Testing and validation split:

Category	Training	Validation	Testing
Covid	987	109	104
Viral pneumonia	1117	124	104
Healthy patients	1114	123	104

Training the model:

Parameter	Value
Optimizer	Adam
Learning rate	0.01
Epochs	10
Loss function	Sparse categorical cross entropy

Coding snapshots:

Importing libraries:

In [1]:
```
import numpy as np
import pandas as pd
from PIL import Image
import os
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras import Input
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from keras import optimizers, losses, activations, models
from keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D, BatchNormalization, GlobalAveragePooling2D,
```

Reading data:

In [2]:
```
TARGET_SIZE = 224

train_datagen = ImageDataGenerator(rescale=1./255.,
                                   featurewise_center=True,
                                     zoom_range = 0.07,
                                   validation_split = 0.1)

test_datagen = ImageDataGenerator(rescale= 1./255)

train_data = train_datagen.flow_from_directory(
    'data/COVID-19 Database/Train',
    class_mode="sparse",
    batch_size=32,
    target_size=(TARGET_SIZE,TARGET_SIZE),
    shuffle=True,
    subset="training",
)

validation_data = train_datagen.flow_from_directory(
    'data/COVID-19 Database/Train',
    class_mode="sparse",
    batch_size=32,
    target_size=(TARGET_SIZE, TARGET_SIZE),
    shuffle=True,
    subset="validation",
)

test_data = test_datagen.flow_from_directory(
    'data/COVID-19 Database/Test',
    class_mode="sparse",
    batch_size=32,
    target_size=(TARGET_SIZE, TARGET_SIZE),
    shuffle=False,
)
```

Importing model:

In [3]:
```
base_model = tf.keras.applications.ResNet152V2(weights="imagenet",include_top=False,input_shape=(TARGET_SIZE,TARGET_SIZE, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)

x = Dense(512, activation='relu')(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)

x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)

x = Dense(64, activation='relu')(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)

predictions = Dense(2, activation='softmax')(x)

for layer in base_model.layers[:300]:
    layer.trainable = False
for layer in base_model.layers[300:]:
    layer.trainable = True

model=Sequential()
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(loss='sparse_categorical_crossentropy', optimizer= tf.keras.optimizers.Adam(learning_rate=0.01),metrics=['accuracy'
model.summary()
model_save = ModelCheckpoint('weights.h5',
                             save_best_only = True,
                             save_weights_only = True,
                             monitor = 'val_loss',
                             mode = 'min', verbose = 1)
```

Fitting the model:

In [4]:
```
EPOCHS = 10
history = model.fit_generator(
     train_data,
     steps_per_epoch=train_data.samples//train_data.batch_size,
     epochs = EPOCHS,
     validation_data = validation_data,
     validation_steps=validation_data.samples//validation_data.batch_size,
     callbacks = [model_save])
```

Evaluating the model on the test set:

In [6]:
```
model.evaluate(test_data)
```

```
10/10 [==============================] - 4s 441ms/step - loss: 0.1371 - accuracy: 0.9519
```

Out[6]:
```
[0.13711410760879517, 0.9519230723381042]
```

Visualization:

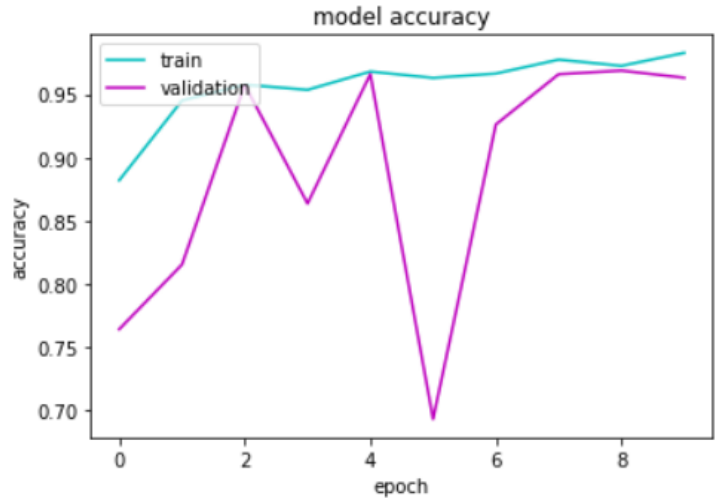
In [8]:
```
print(history.history.keys())
plt.plot(history.history['accuracy'],color='c')
plt.plot(history.history['val_accuracy'],color='m')
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.title('model training loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()

plt.plot(history.history['val_loss'])
plt.title('model validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```

In [9]:
```
from sklearn.metrics import classification_report, confusion_matrix
print('Confusion Matrix')
cm = confusion_matrix(test_data.classes, y_pred)
print(cm)
print(sns.heatmap(confusion_matrix(test_data.classes, y_pred),annot=True,fmt="d"))
print(classification_report(test_data.classes, y_pred, digits=5))
```

Results:

```
Epoch 1/10
100/100 [==============================] - 149s 1s/step - loss: 0.5360 - accuracy: 0.7852 - val_loss: 2.5501 - val_accuracy: 0.7642

Epoch 00001: val_loss improved from inf to 2.55005, saving model to weights.h5
Epoch 2/10
100/100 [==============================] - 87s 866ms/step - loss: 0.1585 - accuracy: 0.9426 - val_loss: 0.8922 - val_accuracy: 0.8153

Epoch 00002: val_loss improved from 2.55005 to 0.89224, saving model to weights.h5
Epoch 3/10
100/100 [==============================] - 87s 866ms/step - loss: 0.1316 - accuracy: 0.9542 - val_loss: 0.1455 - val_accuracy: 0.9574

Epoch 00003: val_loss improved from 0.89224 to 0.14553, saving model to weights.h5
Epoch 4/10
100/100 [==============================] - 87s 867ms/step - loss: 0.1328 - accuracy: 0.9530 - val_loss: 0.4058 - val_accuracy: 0.8636

Epoch 00004: val_loss did not improve from 0.14553
Epoch 5/10
100/100 [==============================] - 88s 884ms/step - loss: 0.0913 - accuracy: 0.9694 - val_loss: 0.0740 - val_accuracy: 0.9659

Epoch 00005: val_loss improved from 0.14553 to 0.07400, saving model to weights.h5
Epoch 6/10
100/100 [==============================] - 88s 882ms/step - loss: 0.0941 - accuracy: 0.9669 - val_loss: 2.5084 - val_accuracy: 0.6932

Epoch 00006: val_loss did not improve from 0.07400
Epoch 7/10
100/100 [==============================] - 88s 875ms/step - loss: 0.0890 - accuracy: 0.9684 - val_loss: 0.1899 - val_accuracy: 0.9261

Epoch 00007: val_loss did not improve from 0.07400
Epoch 8/10
100/100 [==============================] - 88s 878ms/step - loss: 0.0860 - accuracy: 0.9744 - val_loss: 0.0902 - val_accuracy: 0.9659

Epoch 00008: val_loss did not improve from 0.07400
Epoch 9/10
100/100 [==============================] - 88s 877ms/step - loss: 0.0854 - accuracy: 0.9698 - val_loss: 0.1058 - val_accuracy: 0.9688

Epoch 00009: val_loss did not improve from 0.07400
Epoch 10/10
100/100 [==============================] - 88s 879ms/step - loss: 0.0541 - accuracy: 0.9833 - val_loss: 0.0824 - val_accuracy: 0.9631

Epoch 00010: val_loss did not improve from 0.07400
```

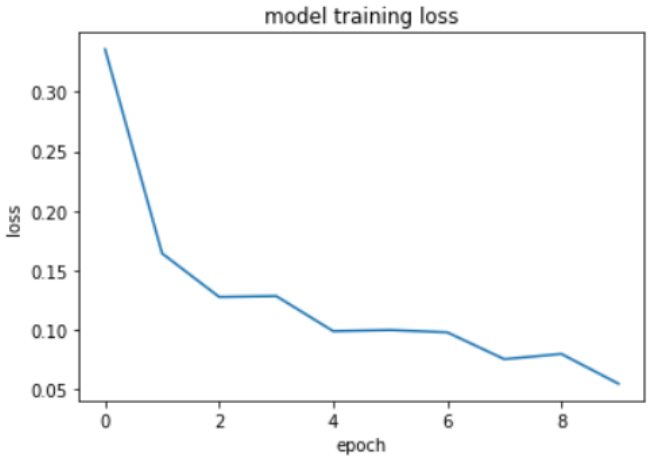
Training and validation accuracy vs Epochs:



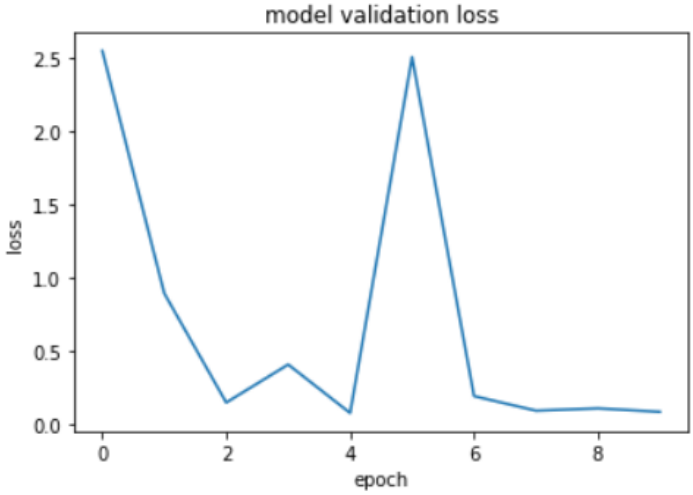
Evaluating model on test set:

```
In [6]: model.evaluate(test_data)
        10/10 [==============================] - 4s 441ms/step - loss: 0.1371 - accuracy: 0.9519
Out[6]: [0.13711410760879517, 0.9519230723381042]
```

Training Loss vs Epochs:



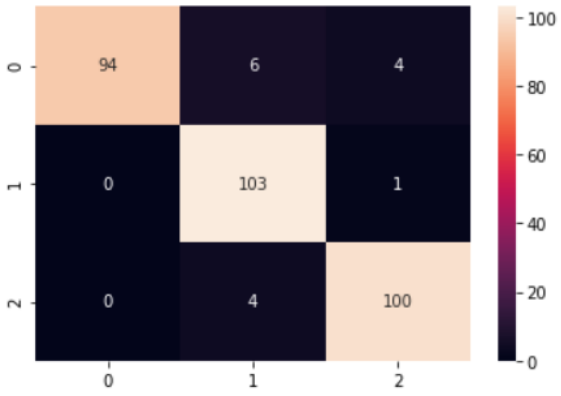
Model validation loss:



Performance metrics:

	precision	recall	f1-score	support
0	1.00000	0.90385	0.94949	104
1	0.91150	0.99038	0.94931	104
2	0.95238	0.96154	0.95694	104
accuracy			0.95192	312
macro avg	0.95463	0.95192	0.95191	312
weighted avg	0.95463	0.95192	0.95191	312

Confusion Matrix:



Conclusion:

The model has achieved a training and validation set accuracy of 98.33% and 96.31% respectively after training for 10 epochs. On evaluation on the test set the model achieves an accuracy of 95.19%.

As a result, the basics of transfer learning and its implementation methodology has been understood through this experimental project.

References:

[1] https://www.kaggle.com/tawsifurrahman/covid19-radiography-database

[2] https://arxiv.org/abs/1512.03385