VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering
VIT Chennai
Fall semester 2022-2023

A Project Report
submitted as part of the course
Health Care Analytics (CSE4068)

Project Title: -

PNEUMONIA & COVID DETECTION

Submitted by

B. PHANINDRA SAI-19MIA1065
M. JAY KUMAR PATEL-19MIA1032
T. SIVA NIKHIL-19MIA1086

Submitted to

Raja Sree T

Slot: E1+TE1

Table of Content:

ABSTRACT:

Pneumonia is a life-threatening infectious disease affecting one or both lungs in humans commonly caused by bacteria called Streptococcus pneumoniae. One in three deaths in India is caused due to pneumonia as reported by World Health Organization (WHO). Chest X-Rays which are used to diagnose pneumonia need expert radiotherapists for evaluation. Thus, developing an automatic system for detecting pneumonia would be beneficial for treating the disease without any delay particularly in remote areas. Due to the success of deep learning algorithms in analysing medical images, Convolutional Neural Networks (CNNs) have gained much attention for disease classification. In addition, features learned by pre-trained CNN models on large-scale datasets are much useful in image classification tasks. In this work, we appraise the functionality of pre-trained CNN models utilized as feature-extractors followed by different classifiers for the classification of abnormal and normal chest X-Rays. We analytically determine the optimal CNN model for the purpose. Statistical results obtained demonstrates that pretrained CNN models employed along with supervised classifier algorithms can be very beneficial in analysing chest X-ray images, specifically to detect Pneumonia.

PROBLEM STATEMENT:

Pneumonia is a major health concern, resulting in millions of deaths worldwide each year. With advances in deep learning technology, there is an opportunity to detect and diagnose pneumonia more accurately and quickly than traditional diagnostic methods. The challenge is to develop a deep learning algorithm that can accurately identify instances of pneumonia from a given set of medical images or data. A successful deep learning algorithm should be able to accurately detect pneumonia cases with a high degree of accuracy, while also being robust enough to manage false positives or negatives with minimal effort. Furthermore, it should also be applicable to different types of imaging technologies such as X-ray, CT scan and ultrasound. In this project, we propose to develop a deep learning algorithm to detect pneumonia from medical images or data. Our solution should be able to accurately classify different cases of pneumonia, while integrating robustness to cope with certain variations in the data. Furthermore, it should be scalable for use in different healthcare settings. We will assess our solution's performance with respect to factors such as accuracy, false positives/negatives, scalability and robustness. This project aims to use deep learning technology to detect pneumonia accurately and quickly, while also contributing to the medical community. Our solution should be applicable to different imaging technologies, data formats, and healthcare environments to ensure wide applicability and impact. By successfully developing a deep learning algorithm and application for pneumonia detection, we can have an important impact in the health sciences field, helping to reduce the mortality rates associated with this disease.

Literature Survey

Serial no.	Name of the Journal	Author	Techniques/Observations	Results
1	Pneumonia Detection in chest X-ray images using Convolutional Neural Networks and Transfer Learning	Rachna Jain, Preeti Nagrath, Gaurav Kataria, V. Sirish Kaushik and D. Jude Hemanth	Convolutional Neural Networks, VGG Net, ResNet, and Inception-v3	VGG16-87.18%, VGG19-88.46%, ResNet50-77.56%, Inception-v3-70.99%
2	A Novel Transfer Learning Based Approach for Pneumonia Detection in Chest X-ray Images	Vikash Chouhan, Sanjay Kumar Singh, Aditya Khamparia	AlexNet, DenseNet121, InceptionV3, resNet18 and GoogLeNet neural networks, feature extraction and ensemble classification	AlexNet-92.86%, DenseNet121-92.62%, InceptionV3-92.01%, GoogLeNet-93.12%, Ensemble model-96.39%
3	A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2	Mohammad Rahimzadeh, Abolfazl Attar	Feature extractors, Resnet Xception, Classifier activation, softmax they trained the networks using the Categorical cross-entropy loss function and Nadam optimizer. The learning rate was set to 1e-4. They trained the network for 100 epochs in each training phase and, because of having 8 training phases, the models were trained for 800 epochs. For the Xception and ResNet50V2, they selected the batch size equal to 30. But as the concatenated network had more parameters than Xception and ResNet50V2, they set the batch size equal to 20. Data augmentation methods theyre also implemented to increase training efficiency and prevent the model from overfitting.	They validated our networks on 31 cases of COVID-19, 4420 cases of pneumonia, and 6851 normal cases. The reason our training data was less than the validation data is that they had a few cases of COVID-19 among many normal and pneumonia cases. Therefore, they could not use many images from the two other classes with COVID-19 fewer cases for training, because it would have made the network not learn COVID19 features. To solve this issue, they

				selected 3783 images for training in 8 different phases. They evaluated our network on the remainder of the data so that our trained network's ultimate performance would be clear. It must be noticed that exceptionally, in fold3, they had 30 cases of COVID19 for validation, and 150 other cases were allocated for training
4	A Deep Feature Learning Model for Pneumonia Detection Applying a Combination of mRMR Feature Selection and Machine Learning Models	M. Togaçar, B. Ergen, Z. Cömert	AlexNet, VGG-16, VGG-19	AlexNet-93.57%, VGG-16-94.73%, VGG-19-94.84%
5	Customized VGG19 Architecture for Pneumonia Detection in Chest X-Rays	Nilanjan Deya, Yu-Dong Zhang, V. Rajinikanth, R. Pugalenthi	AlexNet, VGG16, VGG19, ResNet50, VGG19-SVM-Linear, VGG19-SVM-RBF, VGG19-KNN, VGG19-RF, VGG19-DT	AlexNet 87.76%, VGG16 90.61%, VGG19 91.39%, ResNet50 91.52%, VGG19-SVM-Linear 94.91%, VGG19-SVM-RBF 97.88%, VGG19-KNN 97.52%, VGG19-RF 97.94%, VGG19-DT 97.45%
6	CovXNet: A multi-dilation convolutional	Tanvir Mahmud, Md Awsafur	DarkCovidNet, COVID-Net, VGG-19, ReNet-50/SVM, ResNet-50	DarkCovidNet 98.08%,

	neural network for automatic COVID-19 and other pneumonia detection from chest X-ray images with transferable multi-receptive feature optimization	Rahman, Shaikh Anowarul Fattah		COVID-Net 92.4%, VGG-19 93.48%, ReNet-50/SVM 95.38%, ResNet-50 98%
7	Viral Pneumonia Screening on Chest X-rays Using Confidence-Aware Anomaly Detection	Jianpeng Zhang, Yutong Xie, Guansong Pang, Zhibin Liao, Johan Verjans	Feature extractor Anomaly detection network Confidence prediction network EfficientNet, ConfiNet, AnoDet, CAAD COMPARING PERFORMANCE OF FOUR MODELS ON THE X-COVID AND OPENCOVID DATASET	The proposed CAAD model achieves an AUC of 83.61% on COVID-19 screening, which outperforms other AI-based methods [52]. Although achieving a sensitivity of only 71.70%, our CAAD model shows a screening ability that is comparable to that of radiologists, as a sensitivity of 69% was reported
8	PNEUMONIA DETECTION ON CHEST X-RAY USING RADIOMIC FEATURES AND CONTRASTIVE LEARNING	Yan Han, Chongyan Chen, Ahmed Tewfik	ResNet-18, ResNet-18FairRadi, ResNet-18Att, ResNet-18 AttFairRadi	Experimental Results Without Using Bounding Box Model Accuracy ResNet-18 76.3%, ResNet-18FairRadi 82.1%, ResNet-18Att 81.5%, ResNet-18AttFairRadi 85.4%
9	Diagnosis of Pneumonia from Chest X-Ray Images using Deep Learning	Enes AYAN, Halil Murat ÜNVER	Xception, VGG16	Xception 82%, VGG16 87%
10	Can AI help in screening Viral	Muhammad E. H. Chowdhury,	SqueezeNet, MobileNetv2, ResNet18, InceptionV3,	SqueezeNet 95.19%,

	and COVID-19 pneumonia?	Tawsifur Rahman, Amith Khandakar, Rashid Mazhar	ResNet101, CheXNet, DenseNet201, VGG19	MobileNetv2 95.9%, ResNet18 95.75%, InceptionV32 94.96%, ResNet101 95.36%, CheXNet 97.74%, DenseNet201 95.19%, VGG19 95.04%
11	Pneumonia Detection: An Efficient Approach Using Deep Learning	Ayush Pant, Akshat Jain, Kiran C Nayak, Daksh Gandhi	EfficientNet-B4 based U-Net, ResNet based U-Net, Ensemble of EfficientNetB4 based U Net and ResNet based U-Net	EfficientNet-B4 based U-Net 94%, ResNet based U-Net 82%, Ensemble of EfficientNetB4 based U Net and ResNet based U-Net 90%
12	An Efficient Network for Pneumonia Detection	Zhongliang Li, Juan Yu, Xuechen Li, Yingqi Li, Weicai Dai	PNet-5, PNet-6, AlexNet, VGG16	PNet-5-92.79%, PNet-6-91.96%, AlexNet-90.30%, VGG16-90.39%
13	Pneumonia Detection Using CNN based Feature Extraction	Dimpy Varshni, Kartik Thakral, Lucky Agarwal	Xception Vgg16 Vgg19 Densenet resnet Feature extractors and classifiers (svm,naïve bayes,random forest and etc) with optimal hyperparameter optimization	Model-DenseNet-169 classifier-SVM(rbf kernel) AUC-80.02%
14	Pneumonia Detection Using Deep Learning Approaches	Ashitosh Tilve, Shrameet Nayak, Saurabh Vernekar	Vgg16 CNN Image processing and segmentation	VGG16 as the algorithm to and achieved accuracy of 96.2% and 93.6% for detection and classification of pneumonia
15	Efficient Pneumonia Detection in Chest Xray Images Using Deep Transfer Learning	Mohammad Farukh Hashmi, Satyarth Katiyar, Avinash G Keskar	ResNet18, DenseNet121 ,InceptionV3, Xception ,MobileNetV2, Weighted Classifier	ResNet18 97.29%, DenseNet121 98.00%, InceptionV3 97.00%, Xception 96.57%, MobileNetV2 96.71%,

				Weighted Classifier 98.43%
16	PNEUMONIA DETECTION USING CNN THROUGH CHEST X-RAY	HARSHVARDHAN GM, MAHENDRA KUMAR GOURISARIA, SIDDHARTH SWARUP RAUTARAY	ANN CNN Batch normalization	sensitivity= 0.9007 and specificity= 0.9216. Both these metrics are above 90% which is a good indicator that the model performs excellently in both, having the ability to correctly identify most of the positive pneumonia cases, and also the ability of ruling out the negative cases. Moreover, the ROC curve illustrated in whose area under the curve (AUC) is 0.9582 which is very high. These metrics are comparable to most of the state-of-the-art architectures applied on CXRs for pneumonia detection
17	Pneumonia Detection through Adaptive Deep Learning Models of Convolutional Neural Networks	Sammy V. Militante, Nanette V. Dionisio,B randon G. Sibbaluca	Googlenet, lenet, vgg16, alexnet	GoogLeNet and LeNet are the top models based on performance accuracy of 98% while the ResNet-50 gained the last among the six models trained with an accuracy rate of 80%. VGG-16, AlexNet, and StridedNet achieved a very satisfactory performance with a mark of 96% to 97%.

18	Pneumonia Detection Using Convolutional Neural Networks	Puneet Gupta	Vgg16 Vgg19 CNN	For VGG16 they got validation accuracy of 92%and training accuracy of about 97% and for VGG19 they got validation accuracy of 89% and training accuracy of about 95% And for the model that they built they got validation accuracy of 93%and training accuracy of about 99%.

Observations from the above Research Papers:

1. most of the papers have used deep learning models and they have high accuracies due to the usage of fine-tuned pre-trained models.

2. Mostly they used convolutional neural network models as it is an image-based model and they have high accuracy in that area.

3. As mostly they used pre-trained models, they had not done any pine-tuning by themselves.

4. Most of the papers mostly used pre-trained models by which they did not create any new architecture by using differentiated extractors.

5. There is not even a single paper about diagnosing mild pneumonia or walking pneumonia because to find mild pneumonia a physical examination is needed.

6. All these papers are more about chest x-rays rather than the symptoms of pneumonia.

How can I tell if I have pneumonia versus the common cold or the flu?

It can be difficult to tell the difference between the symptoms of a cold, the flu and pneumonia, and only a healthcare provider can diagnose you. As pneumonia can be life-threatening, it’s important to seek medical attention for serious symptoms that could be signs of pneumonia, such as:

* Congestion or chest pain.

* Difficulty breathing.

* A fever of 102 degrees Fahrenheit (38.88 degrees Celsius) or higher.

* Coughing up yellow, green or bloody mucus or spit.

What is walking pneumonia?

Walking pneumonia is a mild form of pneumonia. Pneumonia is a lung infection that causes your airways to swell, the air sacs in your lungs to fill with mucus and other fluids, high fever and a cough with mucus. If you have walking pneumonia, you may feel well enough to walk around and carry out daily tasks without realizing you have pneumonia. “Walking pneumonia” is the common term for atypical pneumonia. Walking pneumonia can only be examined by doctors. By listing all their symptoms and if needed he will be medicated with antibiotics When a person got severe pneumonia then only his chest X-ray will be valid for an AI application diagnosis.

About the Dataset:

COVID-19 RADIOGRAPHY DATABASE (Winner of the COVID-19 Dataset Award by Kaggle Community)

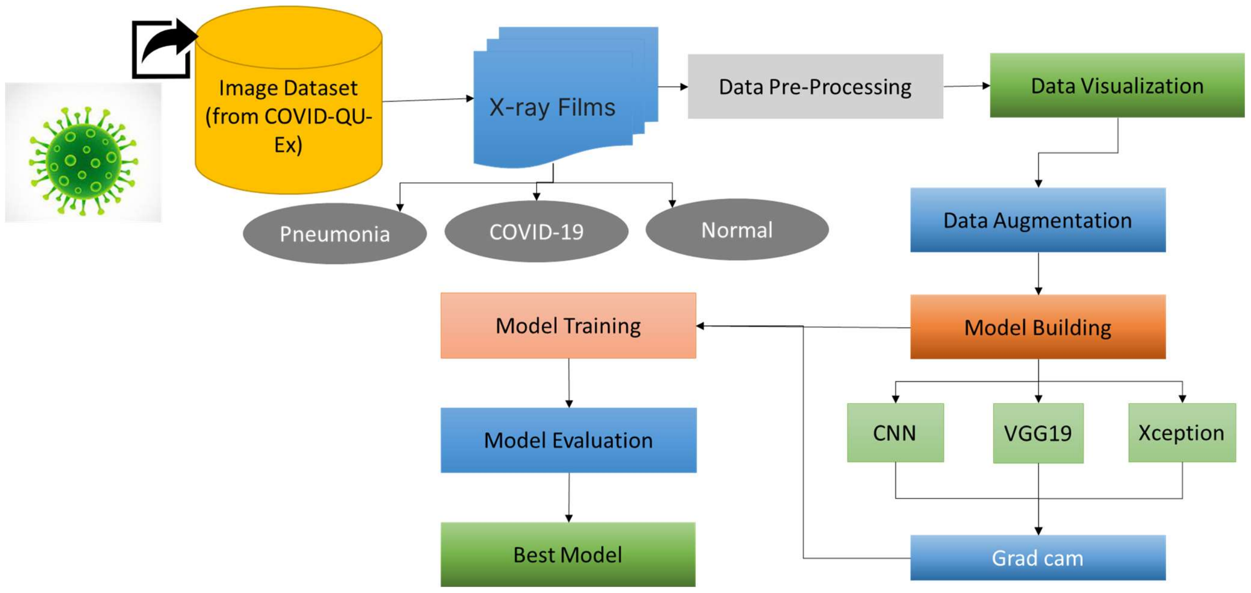
A team of researchers from Qatar University, Doha, Qatar, and the University of Dhaka, Bangladesh along with their collaborators from Pakistan and Malaysia in collaboration with medical doctors have created a database of chest X-ray images for COVID-19 positive cases along with Normal and Viral Pneumonia images. This COVID-19, normal, and other lung infection dataset is released in stages. In the first release, we have released 219 COVID-19, 1341 normal, and 1345 viral pneumonia chest X-ray (CXR) images. In the first update, we have increased the COVID-19 class to 1200 CXR images. In the 2nd update, we have increased the database to 3616 COVID-19 positive cases along with 10,192 Normal, 6012 Lung Opacity (Non-COVID lung infection), and 1345 Viral Pneumonia images and corresponding lung masks. We will continue to update this database as soon as we have new x-ray images for COVID-19 pneumonia patients.

The researchers of Qatar University have compiled the COVID-QU-Ex dataset, which consists of 33,920 chest X-ray (CXR) images including:

11,956 COVID-19, 11,263 Non-COVID infections (Viral or Bacterial Pneumonia), and 10,701 Normal

Ground-truth lung segmentation masks are provided for the entire dataset. This is the largest ever created lung mask dataset.

Flow chart:



Models:

Xception:

Xception is a deep neural network architecture that was introduced in 2016 by François Chollet, the creator of the Keras library. It is a variant of the Inception architecture that uses depthwise separable convolutions as building blocks.

The Xception architecture consists of a series of blocks, each of which contains a depthwise separable convolutional layer followed by a pointwise convolutional layer. The depthwise separable convolutional layer applies a spatial filter to each input channel separately, followed by a pointwise convolution that combines the filtered channels. This two-step process reduces the number of parameters and computational cost of the convolution operation, while still allowing the network to learn complex features.

The Xception architecture consists of 36 convolutional layers, including 14 separation convolutional layers and 22 pointwise convolutional layers. The first layer is a traditional convolutional layer that processes the input image, followed by several blocks of depthwise separable convolutional layers. The final layers of the network are a global average pooling layer and a fully connected layer with softmax activation, which output the classification probabilities for the input image.

The Xception architecture has been shown to achieve state-of-the-art performance on several image classification benchmarks, including ImageNet and CIFAR-10. It has also been used for other computer vision tasks, such as object detection and semantic segmentation.

VGG-19

VGG19 is a deep convolutional neural network model that was developed by researchers at the Visual Geometry Group (VGG) at the University of Oxford. It was designed to achieve high accuracy on the ImageNet classification task, which involves recognizing and categorizing objects in images.

The VGG19 model consists of 19 layers, including 16 convolutional layers and 3 fully connected layers. The convolutional layers are arranged in a series of 5 blocks, where each block contains multiple convolutional layers followed by max pooling layers. The fully connected layers at the end of the network serve as a classifier for the ImageNet task.

The VGG19 model has been widely used as a pre-trained model for various computer vision tasks, including image classification, object detection, and image segmentation. It has also been used as a starting point for developing more complex deep learning models.

Grad-CAM

There are several reasons why Grad-CAM visualization can be important:

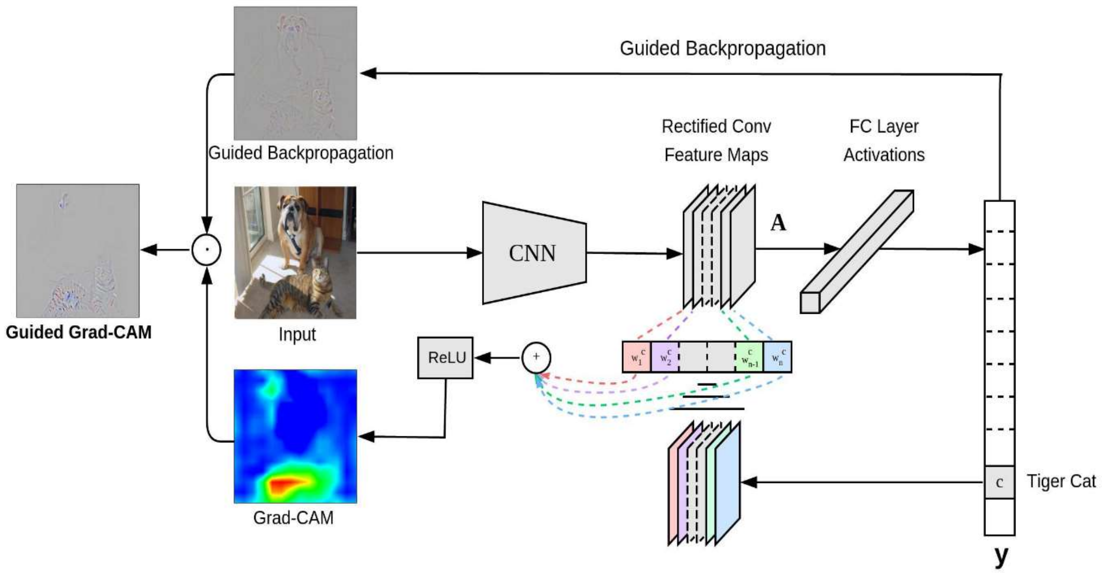
Model understanding: Grad-CAM visualization can be useful for understanding how a deep learning model is making its predictions. By highlighting the regions of the image that are most important for the model's predictions, Grad-CAM can help to provide insight into the decision-making process of the model.

Model debugging: Grad-CAM visualization can also be useful for debugging problems with a deep learning model. For example, if the model is making incorrect predictions, Grad-CAM can help to identify which parts of the image the model is focusing on and potentially identify any issues with the model's training or architecture.

Model explanation: Grad-CAM visualization can be useful for explaining the predictions of a deep learning model to non-technical stakeholders. By highlighting the regions of the image that are most important for the model's predictions, Grad-CAM can provide a more intuitive understanding of the model's decision-making process.

IMPORTANCE OF GRAD-CAM VISUALIZATION

Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualization technique that is used to understand which parts of an image are most important for a deep learning model's prediction. The technique works by generating a heatmap that highlights the regions of the image that the model is most sensitive to when making a prediction.



What is covid and pneumonia?

Covid

ABOUT COVID

COVID-19, also known as the coronavirus disease, is a highly infectious and potentially severe respiratory illness caused by the SARS-CoV-2 virus. It was first identified in Wuhan, China in 2019 and has since spread to affect people in many countries around the world.

Symptoms of COVID-19 can range from mild to severe and can include fever, cough, difficulty breathing, body aches, and loss of taste or smell. Some people with COVID-19 may be asymptomatic or have very mild symptoms, while others may develop severe illness, including pneumonia and respiratory failure, which can be fatal.

COVID-19 is primarily spread through respiratory droplets when an infected person speaks, coughs, or sneezes, but it can also be spread by touching a surface or object contaminated with the virus and then touching the face.

To prevent the spread of COVID-19, it is important to practice good hygiene, such as washing hands frequently, covering the mouth and nose when coughing or sneezing, and wearing a mask in public settings. Vaccines are also available to help protect against COVID-19.

IMPORTANCE OF COVID DETECTION

COVID-19 detection is important for a number of reasons. One of the main reasons is to help control the spread of the virus. By identifying and isolating people who are infected with COVID-19, we can limit the transmission of the virus to others and reduce the overall number of cases. This can help to prevent the healthcare system from being overwhelmed and allow for more effective treatment of those who are severely ill.

COVID-19 detection is also important for public health planning and resource allocation. By knowing the number of cases in a given area, public health officials can determine the level of risk and take appropriate measures to protect the population. For example, if there is a high number of cases in a particular region, officials may implement measures such as lockdowns or mass vaccination campaigns to help reduce the spread of the virus.

In addition, COVID-19 detection can help to protect individuals and communities. By getting tested and knowing their status, individuals can take steps to protect themselves and others, such as self-isolating if they are positive or getting vaccinated if they are negative. This can help to reduce the risk of severe illness and death from COVID-19.

Pneumonia

ABOUT PNEUMONIA

Pneumonia is an infection that affects the lungs, typically caused by bacteria, viruses, or other microorganisms. It is a serious condition that can range from mild to severe, and can be life-threatening in some cases.

Symptoms of pneumonia can include fever, cough, shortness of breath, chest pain, and fatigue. Pneumonia can also cause other symptoms such as sweating, shaking chills, and rapid breathing. In severe cases, pneumonia can cause confusion, blueness of the lips or skin, and low oxygen levels in the blood.

Pneumonia is often treated with antibiotics, but the specific treatment will depend on the cause of the infection and the severity of the illness. Other treatments may include oxygen therapy, cough medicine, and supportive care such as fluids and rest.

It is important to seek medical treatment if you think you may have pneumonia, as it can be a serious condition that requires prompt treatment. Risk factors for pneumonia include having a weakened immune system, being over 65 years old, or having underlying health conditions such as chronic obstructive pulmonary disease (COPD) or asthma.

IMPORTANCE OF PNEUMONIA DETECTION

Pneumonia detection is important for a number of reasons. One of the main reasons is to ensure that individuals with pneumonia receive appropriate treatment as soon as possible. Pneumonia can be a serious condition that requires medical attention, and early treatment can help to prevent the condition from worsening and can improve the chances of a full recovery.

Pneumonia detection is also important for public health purposes. By detecting cases of pneumonia, public health officials can track the prevalence of the condition and take steps to prevent its spread. For example, if there is an outbreak of pneumonia in a particular area, officials may implement measures such as vaccination campaigns or education programs to help reduce the risk of infection.

In addition, pneumonia detection can help to identify underlying health conditions or risk factors that may increase the risk of pneumonia. For example, individuals with chronic obstructive pulmonary disease (COPD) or asthma may be more prone to developing pneumonia, and detecting these conditions can help to prevent pneumonia from occurring or worsening.

Overall, pneumonia detection is an important part of ensuring the health and wellbeing of individuals and communities.

Installation of packages

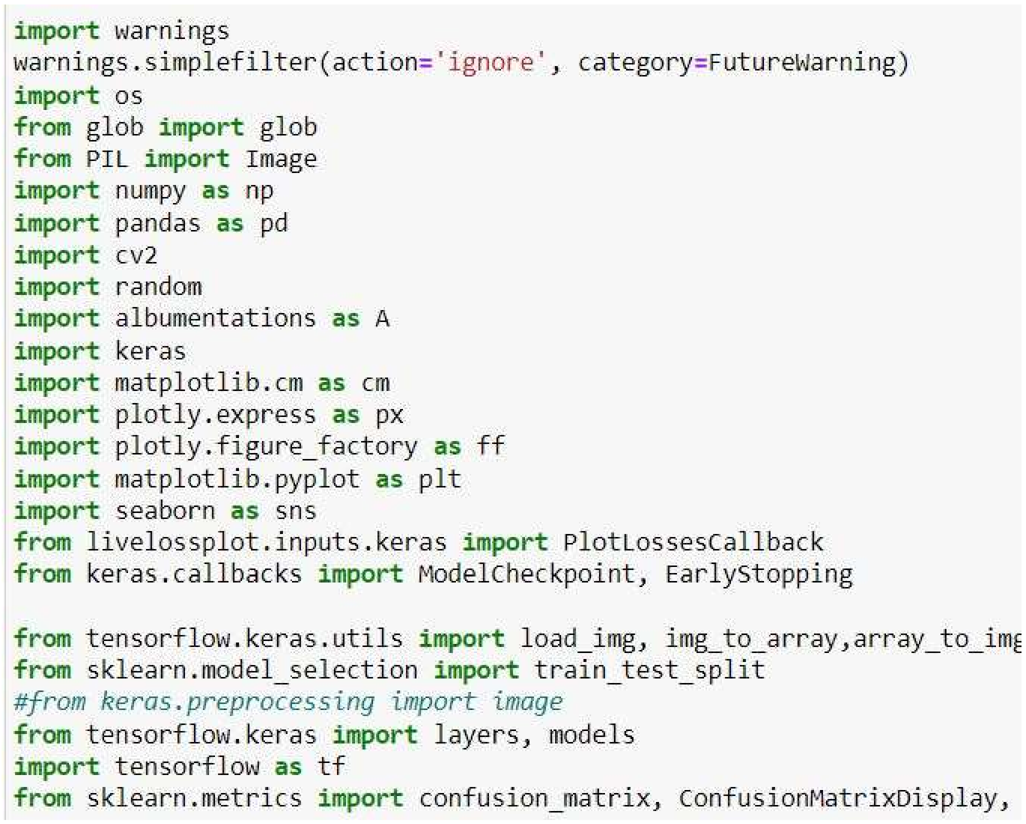
Livelossplot installation

WHAT IS LIVELOSSPLOT?

Livelossplot is a Python library that can be used to create live plots of loss and evaluation metrics during the training of machine learning models. It is often used in combination with deep learning frameworks such as Keras and PyTorch.

Livelossplot allows users to visualize the progress of their model training in real-time, which can be useful for identifying trends, debugging, and optimizing model performance. It is especially helpful when training deep learning models, which can be complex and time-consuming to train.

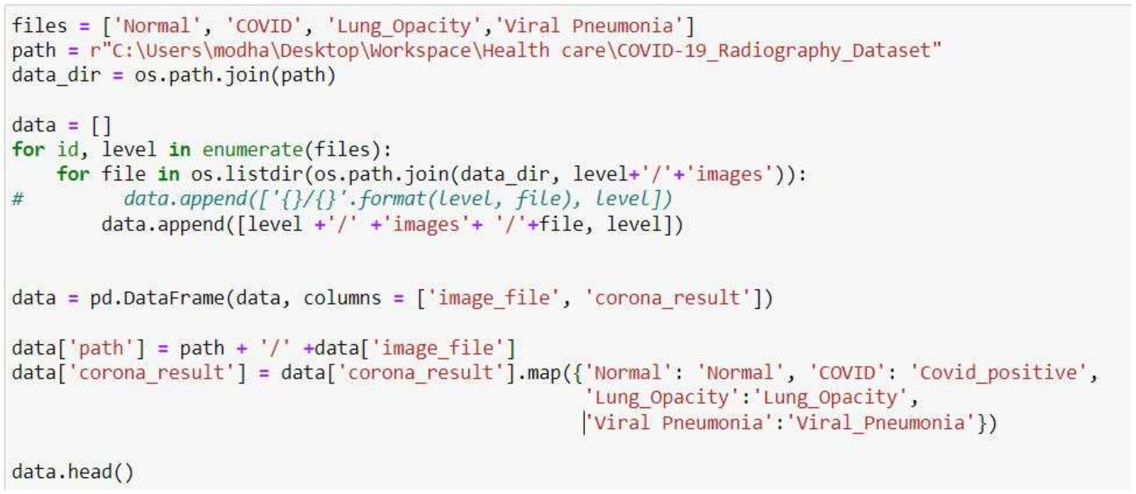
To use livelossplot, users simply need to specify the loss and evaluation metrics they want to track, and the library will create a dynamic plot that updates as the model is trained. Users can also customize the appearance of the plot, such as the colors, line styles, and legend labels.



```python
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import os
from glob import glob
from PIL import Image
import numpy as np
import pandas as pd
import cv2
import random
import albumentations as A
import keras
import matplotlib.cm as cm
import plotly.express as px
import plotly.figure_factory as ff
import matplotlib.pyplot as plt
import seaborn as sns
from livelossplot.inputs.keras import PlotLossesCallback
from keras.callbacks import ModelCheckpoint, EarlyStopping

from tensorflow.keras.utils import load_img, img_to_array,array_to_img
from sklearn.model_selection import train_test_split
#from keras.preprocessing import image
from tensorflow.keras import layers, models
import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
```

Image data collection

Here the images are categorised into classes and its path are collected & saved in a dataframe which can be used for manipulation later.



```
files = ['Normal', 'COVID', 'Lung_Opacity','Viral Pneumonia']
path = r"C:\Users\modha\Desktop\Workspace\Health care\COVID-19_Radiography_Dataset"
data_dir = os.path.join(path)

data = []
for id, level in enumerate(files):
    for file in os.listdir(os.path.join(data_dir, level+'/'+'images')):
#         data.append(['{}/{}'.format(level, file), level])
        data.append([level +'/' +'images'+ '/'+file, level])


data = pd.DataFrame(data, columns = ['image_file', 'corona_result'])

data['path'] = path + '/' +data['image_file']
data['corona_result'] = data['corona_result'].map({'Normal': 'Normal', 'COVID': 'Covid_positive',
                                                   'Lung_Opacity':'Lung_Opacity',
                                                  'Viral Pneumonia':'Viral_Pneumonia'})

data.head()
```

Exploratory Data Analysis

Data manipulation

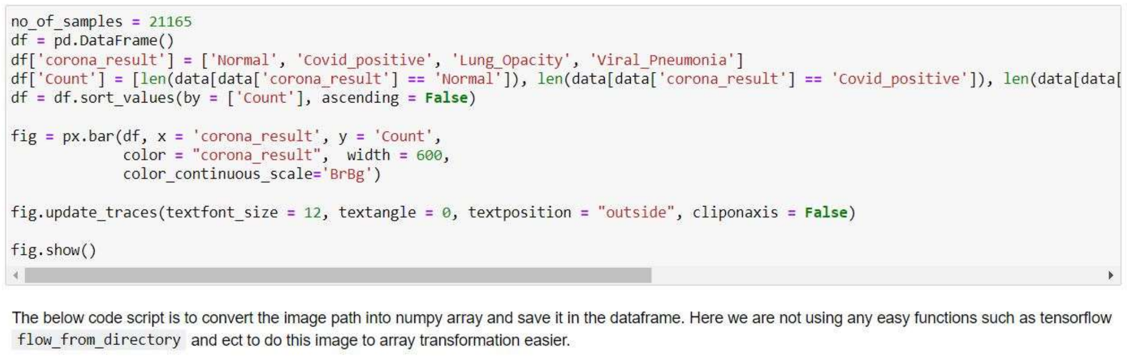
WHY DO WE USE PLOTLY FOR VISUALIZATION?

Plotly is a data visualization library that allows users to create a wide range of static, interactive, and animated visualizations in Python. It is particularly useful for creating interactive visualizations that can be displayed in web browsers.

Some of the benefits of using Plotly for visualization include:

1. Ease of use: Plotly has a user-friendly interface and simple syntax, making it easy to create a wide range of plots and charts.
2. Customization: Plotly allows users to customize their plots and charts in a variety of ways, including changing the colors, fonts, and layout.
3. Interactivity: Plotly's interactive visualizations allow users to explore and understand their data in a more intuitive way. For example, users can hover over data points to see the underlying values or zoom in on a specific region of a plot.
4. Compatibility: Plotly visualizations can be easily shared and displayed in a variety of formats, including as standalone HTML files, within Jupyter notebooks, and on websites using Plotly's online hosting service.

Overall, Plotly is a powerful and flexible tool for creating high-quality data visualizations that can be easily shared and interacted with by others.
In our case, the below visulaization shows the count of data used over each classes.



```
no_of_samples = 21165
df = pd.DataFrame()
df['corona_result'] = ['Normal', 'Covid_positive', 'Lung_Opacity', 'Viral_Pneumonia']
df['Count'] = [len(data[data['corona_result'] == 'Normal']), len(data[data['corona_result'] == 'Covid_positive']), len(data[data[
df = df.sort_values(by = ['Count'], ascending = False)

fig = px.bar(df, x = 'corona_result', y = 'Count',
             color = "corona_result",  width = 600,
             color_continuous_scale='BrBg')

fig.update_traces(textfont_size = 12, textangle = 0, textposition = "outside", cliponaxis = False)

fig.show()
```

The below code script is to convert the image path into numpy array and save it in the dataframe. Here we are not using any easy functions such as tensorflow `flow_from_directory` and ect to do this image to array transformation easier.

IMPORTANCE OF PLOTTING MAX VALUE DISTRIBUTION

One possible use of plotting the maximum value distribution on an image dataset is to visualize the range of values present in the dataset. This can be helpful for understanding the characteristics of the dataset and identifying any potential issues or trends that may be present. For example, if the maximum value distribution is heavily skewed towards one end of the range, it may indicate that there are a large number of outliers present in the dataset. On the other hand, if the maximum value distribution is more evenly distributed, it may indicate that the dataset is more balanced and potentially easier to work with.

In addition to understanding the characteristics of the dataset, plotting the maximum value distribution can also be useful for identifying the appropriate scaling or normalization techniques to apply to the data. For example, if the maximum value distribution is heavily skewed, it may be necessary to apply a transformation such as log scaling to the data in order to better distribute the values.

IMPORTANCE OF PLOTTING MIN VALUE DISTRIBUTION

Like the maximum value distribution, plotting the minimum value distribution on an image dataset can be useful for understanding the characteristics of the dataset and identifying any potential issues or trends that may be present. For example, if the minimum value distribution is heavily skewed towards one end of the range, it may indicate that there are a large number of outliers present in the dataset. On the other hand, if the minimum value distribution is more evenly distributed, it may indicate that the dataset is more balanced and potentially easier to work with.

In addition to understanding the characteristics of the dataset, plotting the minimum value distribution can also be useful for identifying the appropriate scaling or normalization techniques to apply to the data. For example, if the minimum value distribution is heavily skewed, it may be necessary to apply a transformation such as log scaling to the data in order to better distribute the values.

IMPORTANCE OF PLOTTING MEAN VALUE DISTRIBUTION

Plotting the mean value distribution on an image dataset can be useful for understanding the overall characteristics of the dataset and identifying any potential issues or trends that may be present. For example, if the mean value distribution is heavily skewed towards one end of the range, it may indicate that there are a large number of outliers present in the dataset. On the other hand, if the mean value distribution is more evenly distributed, it may indicate that the dataset is more balanced and potentially easier to work with.

In addition to understanding the characteristics of the dataset, plotting the mean value distribution can also be useful for identifying the appropriate scaling or normalization techniques to apply to the data. For example, if the mean value distribution is heavily skewed, it may be necessary to apply a transformation such as log scaling to the data in order to better distribute the values.

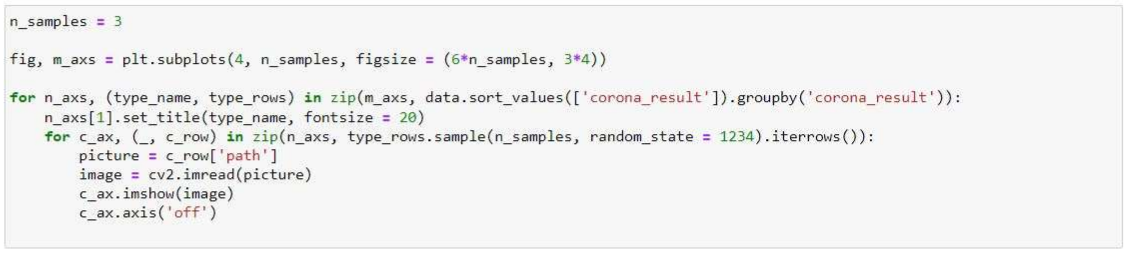
IMPORTANCE OF PLOTTING MEAN vs STANDARD DEVIATION OF IMAGES

Scatter plotting the mean and standard deviation of image samples can be useful for understanding the characteristics of the dataset and identifying any potential issues or trends that may be present. For example, if the scatter plot shows a large amount of variance in the mean values of the samples, it may indicate that the dataset is highly heterogeneous and may require additional preprocessing or normalization. On the other hand, if the scatter plot shows little variance in the mean values of the samples, it may indicate that the dataset is more homogeneous and potentially easier to work with.

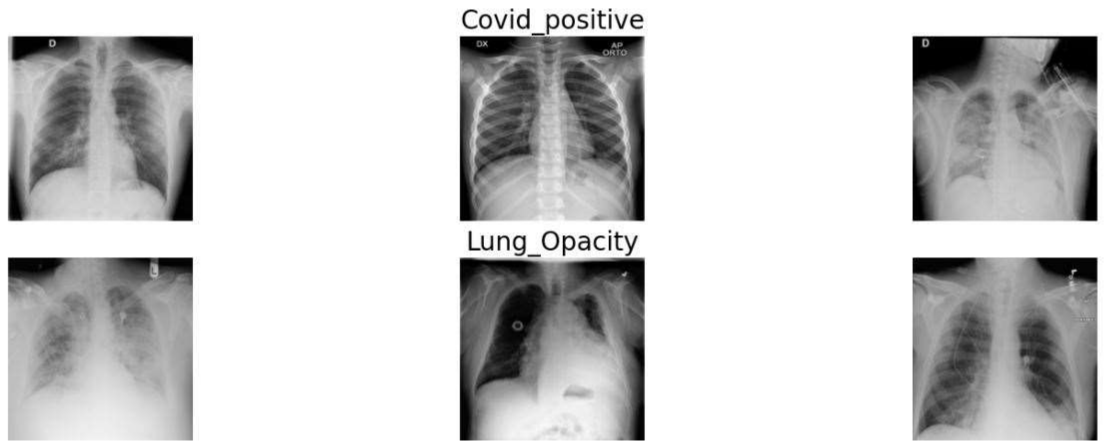
In addition to understanding the characteristics of the dataset, scatter plotting the mean and standard deviation of image samples can also be useful for identifying patterns or correlations between the two values. For example, if there is a positive correlation between the mean and standard deviation, it may indicate that the samples with higher mean values also tend to have higher standard deviations, and vice versa. This information can be useful for identifying potential trends or issues within the dataset.

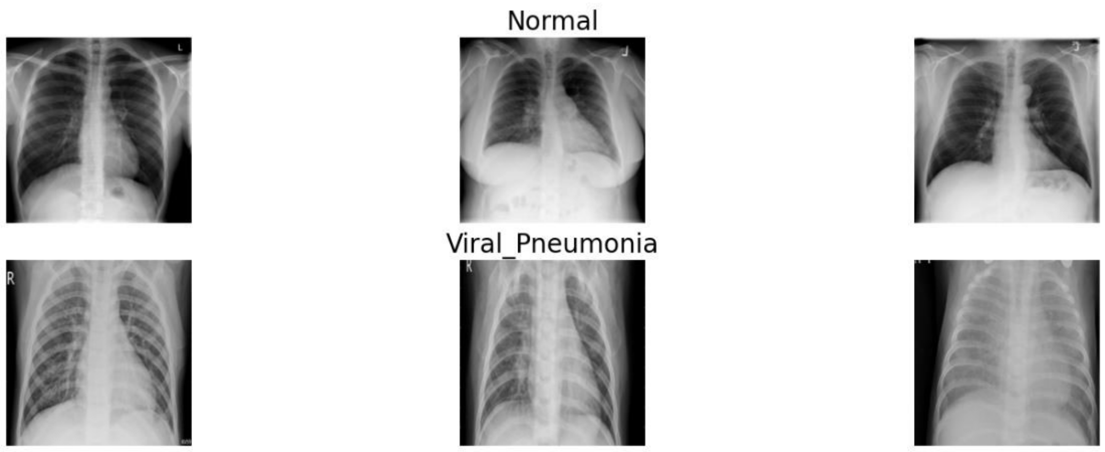
PRINTING RANDOM IMAGES FROM ALL CLASSES

To get started, let us visualize some images from all the four classes.



```
n_samples = 3

fig, m_axs = plt.subplots(4, n_samples, figsize = (6*n_samples, 3*4))

for n_axs, (type_name, type_rows) in zip(m_axs, data.sort_values(['corona_result']).groupby('corona_result')):
    n_axs[1].set_title(type_name, fontsize = 20)
    for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(n_samples, random_state = 1234).iterrows()):
        picture = c_row['path']
        image = cv2.imread(picture)
        c_ax.imshow(image)
        c_ax.axis('off')
```





Types of image transformation applied and it's uses

B CHANNEL IMAGES

In image processing, the "b" channel typically refers to the blue channel of an image, which represents the intensity of blue color in each pixel. There are several reasons why it might be useful to convert an image to the b channel for exploratory data analysis (EDA):

1. **Simplicity**: Converting an image to a single channel can make it easier to focus on specific aspects of the image, such as the distribution of blue color, without being distracted by other colors.
2. **Visualization**: Converting an image to a single channel can make it easier to visualize and understand the distribution of pixel values within the image. For example, a grayscale image can be created by converting an image to the "l" channel (lightness), which can be useful for identifying patterns or features within the image.
3. **Preprocessing**: Converting an image to a single channel can be a useful preprocessing step for certain types of image analysis or machine learning tasks. For example, if a model is only sensitive to the blue channel of an image, it may be more efficient to convert the image to the b channel rather than processing all three channels.

Overall, converting an image to a single channel can be a useful technique for understanding and working with image data, depending on the specific goals and requirements of the analysis.

BEN GRAHAM'S METHOD

Ben Graham's method is a statistical technique that is often used for exploratory data analysis (EDA) in image datasets. The method involves calculating the mean and standard deviation of pixel values within each image, and then plotting the mean values against the standard deviation values. This can provide a useful overview of the characteristics of the dataset and identify any potential issues or trends that may be present.

Some specific reasons why Ben Graham's method might be useful for EDA in image datasets include:

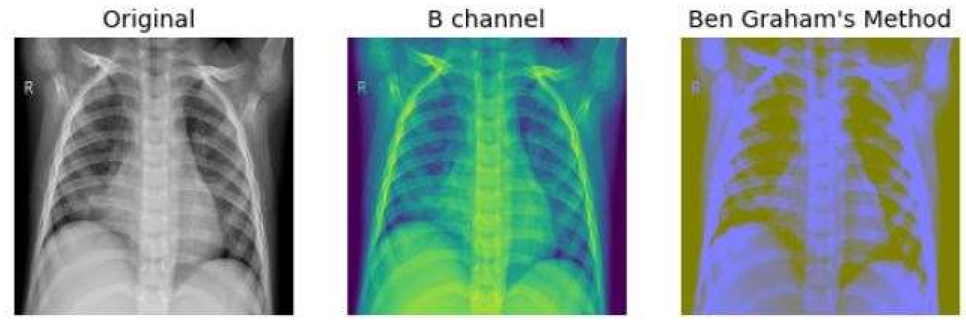
1. Understanding the distribution of pixel values: By plotting the mean and standard deviation values of each image, Ben Graham's method can help to visualize the

distribution of pixel values within the dataset and identify any potential outliers or anomalies.

2. Identifying patterns or correlations: Ben Graham's method can also be useful for identifying patterns or correlations between the mean and standard deviation values of the images. For example, a positive correlation between the two values might indicate that images with higher mean values tend to have higher standard deviations, and vice versa.
3. Preprocessing and normalization: Ben Graham's method can also be useful for identifying the appropriate preprocessing or normalization techniques to apply to the data. For example, if the scatter plot shows a large amount of variance in the mean values, it may be necessary to apply a transformation such as log scaling to the data in order to better distribute the values.

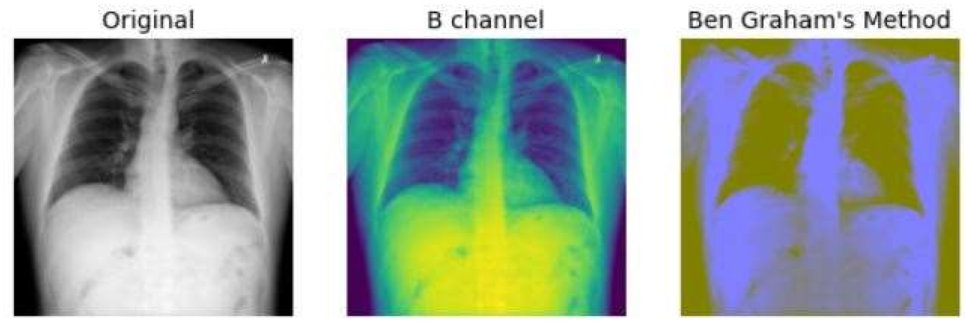
```
random_analysis(list_images_sample[0])
```

```
-----------------------------IMAGE DETAILS ( VIRAL PNEUMONIA )-------------------

Image Shape: (299, 299, 3)
Image Height: 299
Image Width: 299
Image Dimension: 3
Image Size: 261kb
Image Data Type: uint8
Maximum RGB value of the image: 243
Minimum RGB value of the image: 0
```



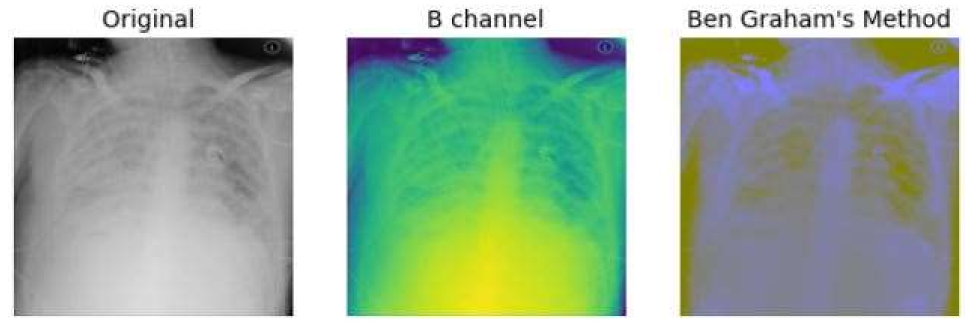
```
random_analysis(list_images_sample[1])
```

```
-----------------------------IMAGE DETAILS ( NORMAL )--------------------------

Image Shape: (299, 299, 3)
Image Height: 299
Image Width: 299
Image Dimension: 3
Image Size: 261kb
Image Data Type: uint8
Maximum RGB value of the image: 250
Minimum RGB value of the image: 0
```



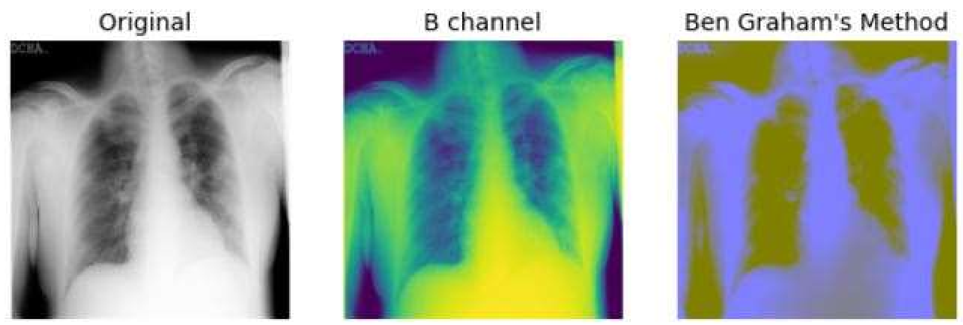
```
random_analysis(list_images_sample[2])
```

```
-----------------------------IMAGE DETAILS ( LUNG_OPACITY )------------------

Image Shape: (299, 299, 3)
Image Height: 299
Image Width: 299
Image Dimension: 3
Image Size: 261kb
Image Data Type: uint8
Maximum RGB value of the image: 224
Minimum RGB value of the image: 0
```



```
random_analysis(list_images_sample[3])
```

```
----------------------------IMAGE DETAILS ( COVID )------------------------

Image Shape: (299, 299, 3)
Image Height: 299
Image Width: 299
Image Dimension: 3
Image Size: 261kb
Image Data Type: uint8
Maximum RGB value of the image: 255
Minimum RGB value of the image: 0
```



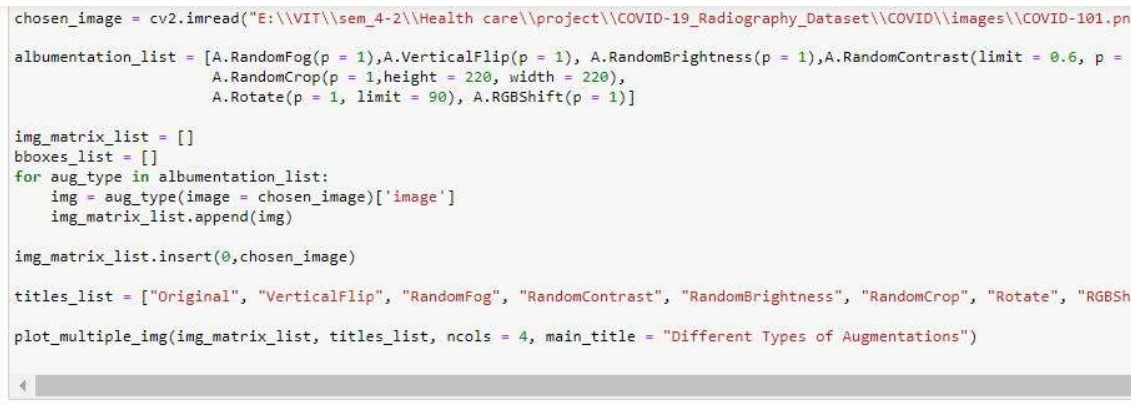
Albumentation

USE OF ALBUMENTATION

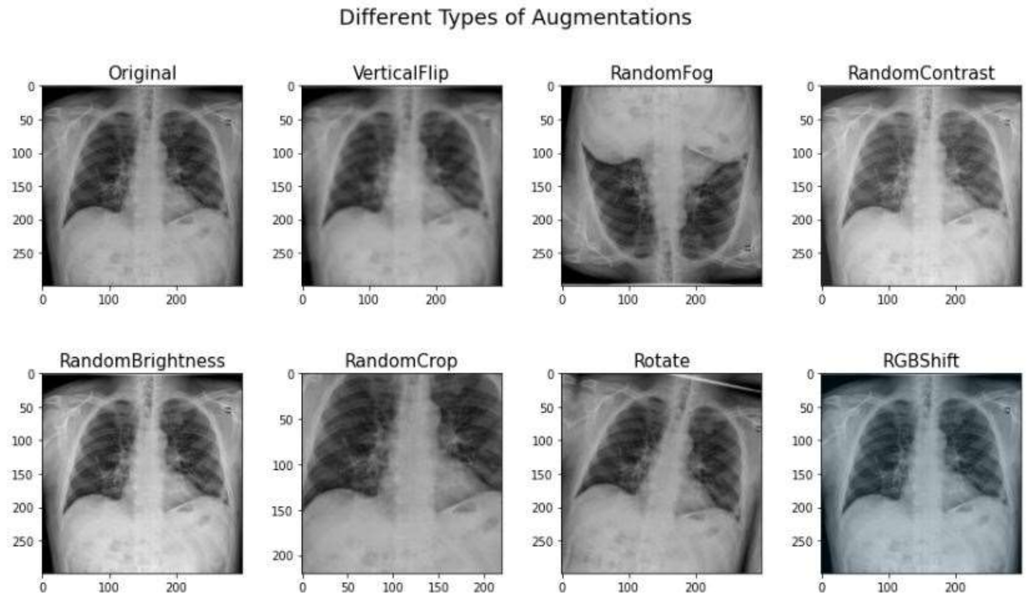
Albumentation is a Python library for image augmentation. Image augmentation refers to the process of applying various modifications to an image, such as resizing, rotating, cropping, or adding noise, in order to augment the training data for machine learning models. The goal of image augmentation is to improve the generalizability and robustness of machine learning models by exposing them to a wider range of data during training.

Some specific uses of Albumentation include:

1. **Data augmentation**: Albumentation can be used to create additional training data by applying various transformations to existing images. This can be especially useful when working with small or imbalanced datasets, as it can help to improve the performance of machine learning models by increasing the amount of training data available.
2. **Preprocessing**: Albumentation can also be used to preprocess image data by applying transformations such as scaling, normalization, or resizing to the data. This can help to ensure that the data is in a suitable format for machine learning models and can improve the performance of the models.
3. **Visualization**: Albumentation can be used to visualize the transformations applied to an image, which can be helpful for understanding the effects of the transformations and debugging any issues that may arise.



```
chosen_image = cv2.imread("E:\\VIT\\sem_4-2\\Health care\\project\\COVID-19_Radiography_Dataset\\COVID\\images\\COVID-101.pn

albumentation_list = [A.RandomFog(p = 1),A.VerticalFlip(p = 1), A.RandomBrightness(p = 1),A.RandomContrast(limit = 0.6, p =
                      A.RandomCrop(p = 1,height = 220, width = 220),
                      A.Rotate(p = 1, limit = 90), A.RGBShift(p = 1)]

img_matrix_list = []
bboxes_list = []
for aug_type in albumentation_list:
    img = aug_type(image = chosen_image)['image']
    img_matrix_list.append(img)

img_matrix_list.insert(0,chosen_image)

titles_list = ["Original", "VerticalFlip", "RandomFog", "RandomContrast", "RandomBrightness", "RandomCrop", "Rotate", "RGBSh

plot_multiple_img(img_matrix_list, titles_list, ncols = 4, main_title = "Different Types of Augmentations")
```

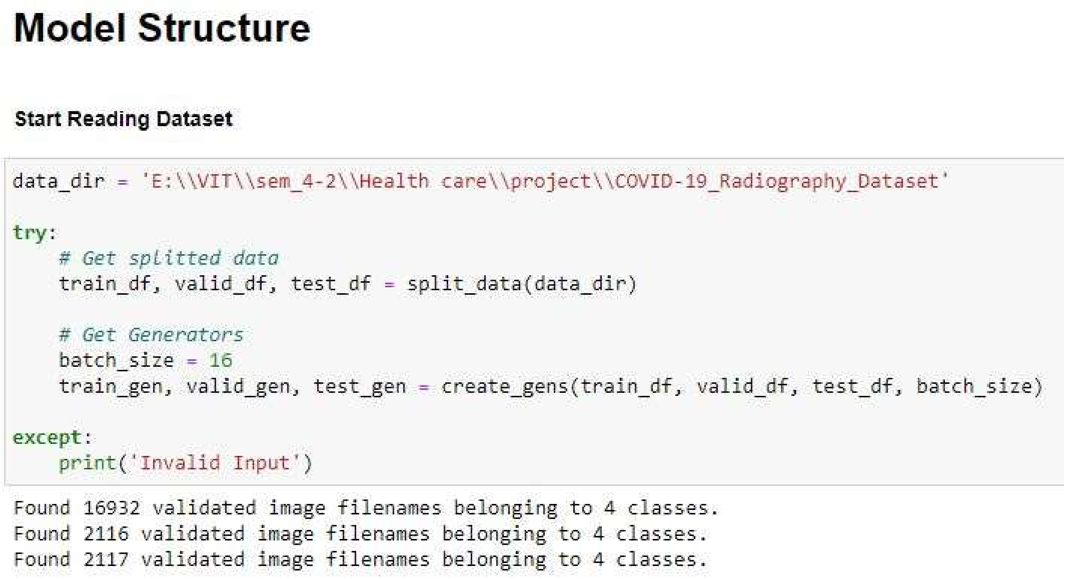


VGG-19

VGG19 is a deep convolutional neural network model that was developed by researchers at the Visual Geometry Group (VGG) at the University of Oxford. It was designed to achieve high accuracy on the ImageNet classification task, which involves recognizing and categorizing objects in images.

The VGG19 model consists of 19 layers, including 16 convolutional layers and 3 fully connected layers. The convolutional layers are arranged in a series of 5 blocks, where each block contains multiple convolutional layers followed by max pooling layers. The fully connected layers at the end of the network serve as a classifier for the ImageNet task.

The VGG19 model has been widely used as a pre-trained model for various computer vision tasks, including image classification, object detection, and image segmentation. It has also been used as a starting point for developing more complex deep learning models.



Model Structure

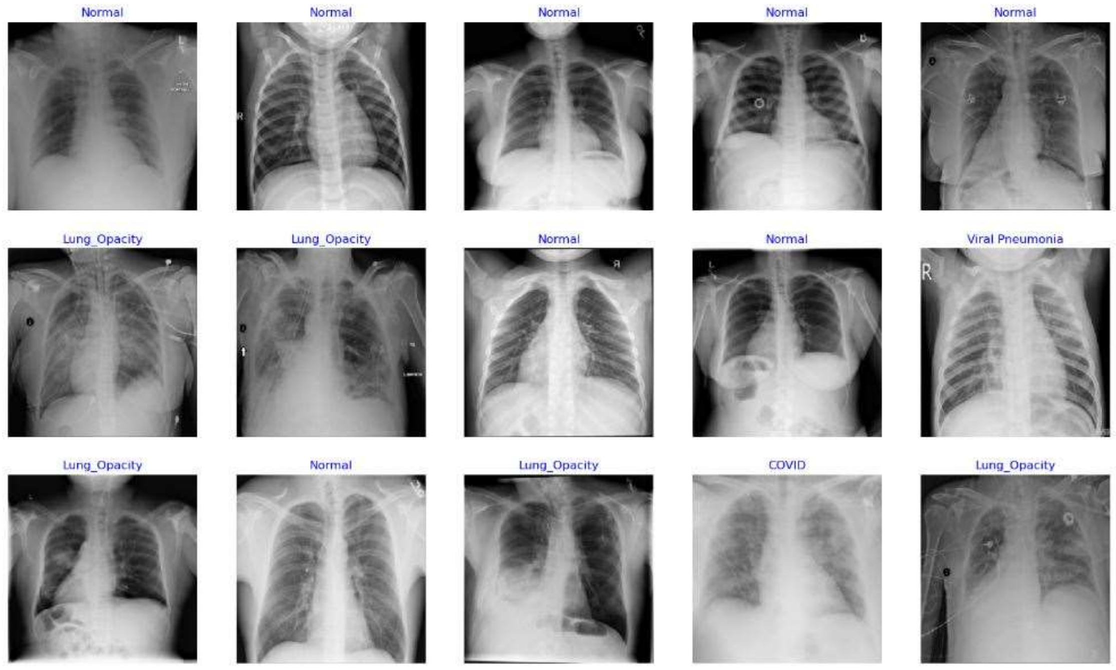
Start Reading Dataset

```
data_dir = 'E:\\VIT\\sem_4-2\\Health care\\project\\COVID-19_Radiography_Dataset'

try:
    # Get splitted data
    train_df, valid_df, test_df = split_data(data_dir)

    # Get Generators
    batch_size = 16
    train_gen, valid_gen, test_gen = create_gens(train_df, valid_df, test_df, batch_size)

except:
    print('Invalid Input')
```

```
Found 16932 validated image filenames belonging to 4 classes.
Found 2116 validated image filenames belonging to 4 classes.
Found 2117 validated image filenames belonging to 4 classes.
```

Display Image Sample

```
show_images(train_gen)
```



Generic Model Creation

```
# Create Model Structure
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)
class_count = len(list(train_gen.class_indices.keys())) # to define number of classes in dense layer

# create pre-trained model (you can built on pretrained model such as :  efficientnet, VGG , Resnet )
# we will use efficientnetb3 from EfficientNet family.
base_model = tf.keras.applications.vgg19.VGG19(include_top= False, weights= "imagenet", input_shape= img_shape, pooling= 'max')

model = Sequential([
    base_model,
    Dense(class_count, activation= 'softmax')
])

model.compile(Adamax(learning_rate= 0.001), loss= 'categorical_crossentropy', metrics= ['accuracy'])

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kerne
ls_notop.h5
80134624/80134624 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 512)               20024384

 dense (Dense)               (None, 4)                 2052

=================================================================
Total params: 20,026,436
Trainable params: 20,026,436
Non-trainable params: 0
```

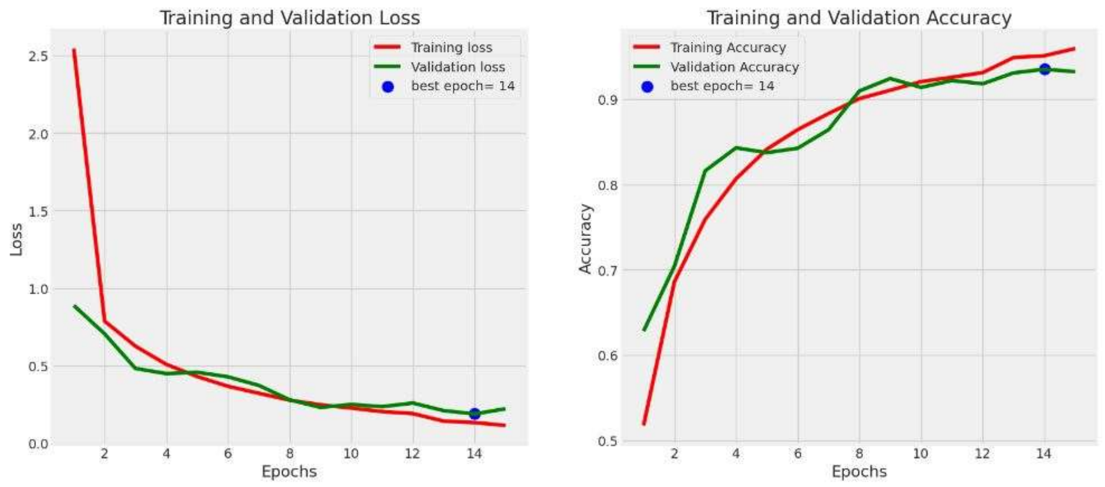
Train model

```
history = model.fit(x= train_gen, epochs= epochs, verbose= 0, callbacks= callbacks,
                    validation_data= valid_gen, validation_steps= None, shuffle= False)
```

```
Do you want model asks you to halt the training [y/n] ?

 y

 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 1 /100   2.546   51.742   0.88863  62.807  0.00100  0.00100  accuracy     0.00    212.16
 2 /100   0.787   68.592   0.70539  70.416  0.00100  0.00100  accuracy    32.56    143.83
 3 /100   0.626   75.927   0.48203  81.616  0.00100  0.00100  accuracy    10.69    143.27
 4 /100   0.509   80.682   0.44906  84.310  0.00100  0.00100  accuracy     6.26    143.48
 5 /100   0.430   84.148   0.45745  83.743  0.00100  0.00100  accuracy     4.30    143.41
enter H to halt training or an integer for number of epochs to run then ask again

 5

 training will continue until epoch 10
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 6 /100   0.367   86.446   0.42826  84.263  0.00100  0.00100  accuracy     2.73    143.41
 7 /100   0.322   88.330   0.37296  86.437  0.00100  0.00100  accuracy     2.18    143.05
 8 /100   0.277   90.084   0.27997  90.974  0.00100  0.00100  val_loss    24.93    143.79
 9 /100   0.248   91.064   0.23112  92.439  0.00100  0.00100  val_loss    17.45    143.59
10 /100   0.227   92.074   0.24985  91.399  0.00100  0.00100  val_loss    -8.10    143.76
enter H to halt training or an integer for number of epochs to run then ask again

 5

 training will continue until epoch 15
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
11 /100   0.204   92.588   0.23580  92.202  0.00100  0.00100  val_loss    -2.02    143.95
12 /100   0.191   93.131   0.25911  91.824  0.00100  0.00050  val_loss   -12.11    143.32
13 /100   0.142   94.897   0.21003  93.100  0.00050  0.00050  val_loss     9.12    144.00
14 /100   0.133   95.110   0.18881  93.526  0.00050  0.00050  val_loss    10.10    144.21
15 /100   0.115   95.960   0.22215  93.242  0.00050  0.00050  val_loss   -17.65    144.46
enter H to halt training or an integer for number of epochs to run then ask again

 h

training has been halted at epoch 15 due to user input
training elapsed time was 0.0 hours, 40.0 minutes, 23.40 seconds)
```

Display model performance

```
plot_training(history)
```



Evaluate model

```
ts_length = len(test_df)
test_batch_size = test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0
test_steps = ts_length // test_batch_size

train_score = model.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score = model.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score = model.evaluate(test_gen, steps= test_steps, verbose= 1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
```

```
29/29 [==============================] - 2s 71ms/step - loss: 0.0973 - accuracy: 0.9591
29/29 [==============================] - 2s 65ms/step - loss: 0.1956 - accuracy: 0.9440
29/29 [==============================] - 28s 781ms/step - loss: 0.1585 - accuracy: 0.9485
Train Loss:  0.09732537716627121
Train Accuracy:  0.9590517282485962
--------------------
Validation Loss:  0.19558091461658478
Validation Accuracy:  0.943965494632721
--------------------
Test Loss:  0.1585310846567154
Test Accuracy:  0.9485120177268982
```

Get Predictions

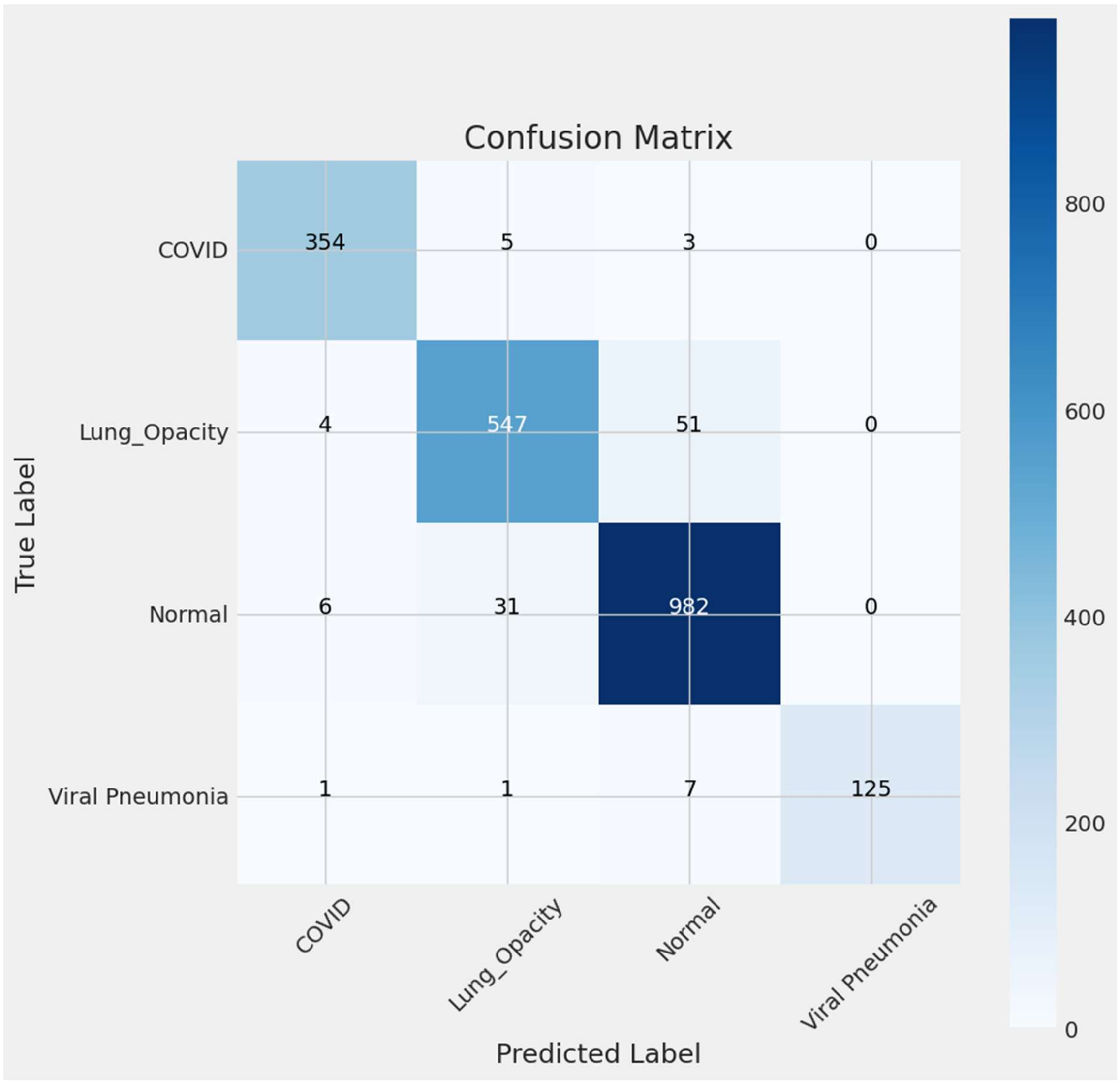
```
preds = model.predict_generator(test_gen)
y_pred = np.argmax(preds, axis=1)
print(y_pred)
```

```
[1 1 1 ... 2 2 2]
```

Confusion Matrics and Classification Report

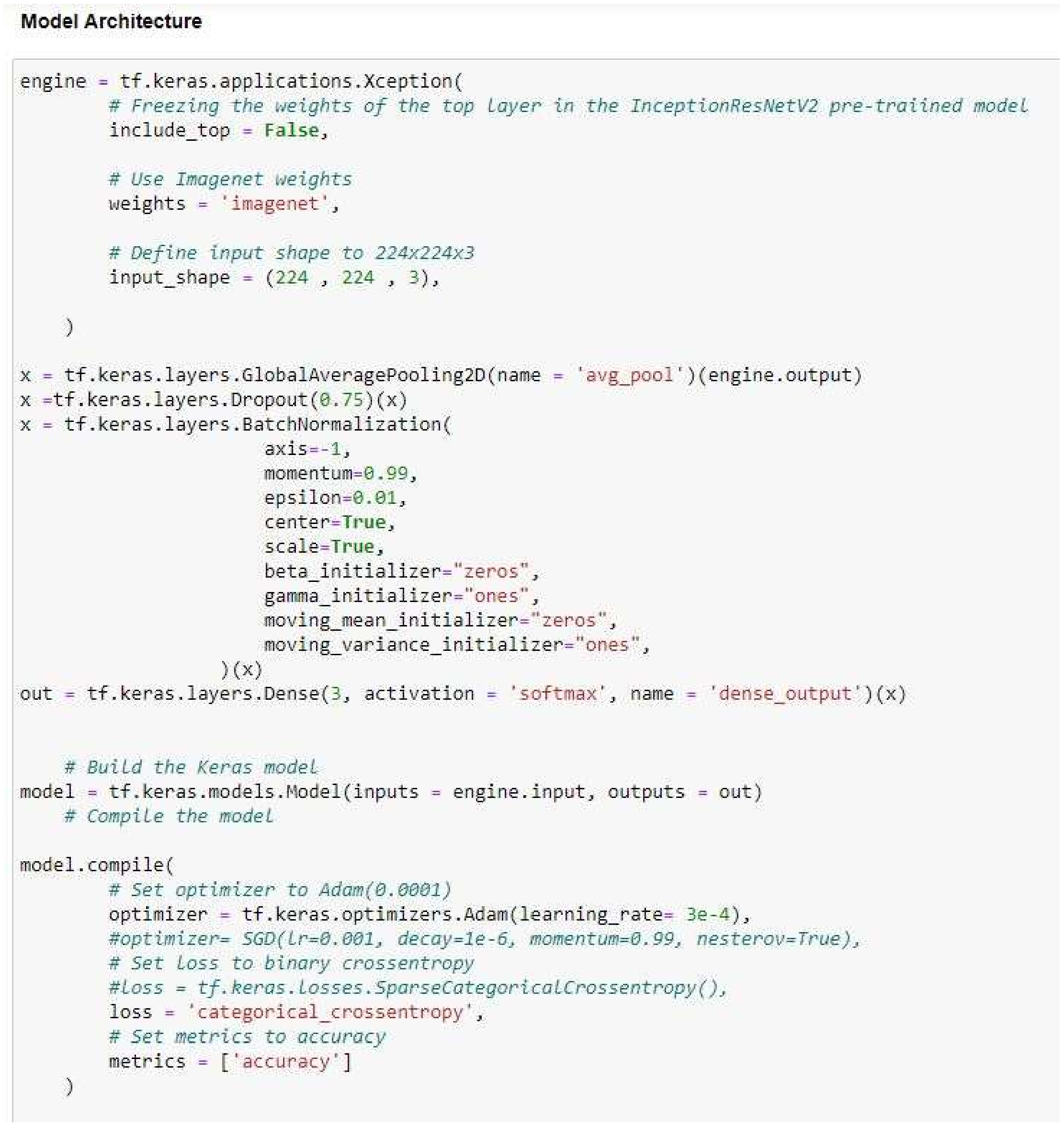
```
g_dict = test_gen.class_indices
classes = list(g_dict.keys())

# Confusion matrix
cm = confusion_matrix(test_gen.classes, y_pred)
plot_confusion_matrix(cm= cm, classes= classes, title = 'Confusion Matrix')

# Classification report
print(classification_report(test_gen.classes, y_pred, target_names= classes))
```

```
Confusion Matrix, Without Normalization
[[354   5   3   0]
 [  4 547  51   0]
 [  6  31 982   0]
 [  1   1   7 125]]
                 precision    recall  f1-score   support

          COVID       0.97      0.98      0.97       362
   Lung_Opacity       0.94      0.91      0.92       602
         Normal       0.94      0.96      0.95      1019
Viral Pneumonia       1.00      0.93      0.97       134

       accuracy                           0.95      2117
      macro avg       0.96      0.95      0.95      2117
   weighted avg       0.95      0.95      0.95      2117
```



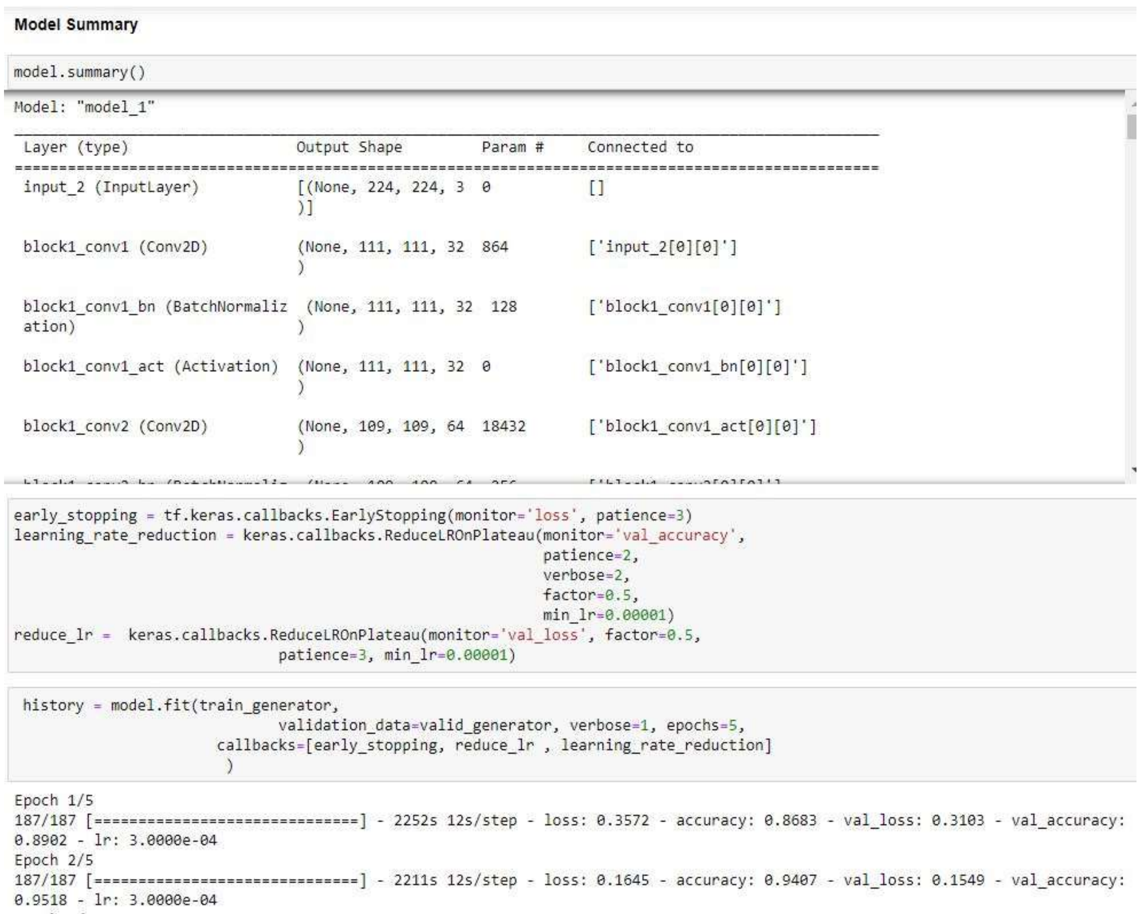
Confusion Matrix
True Label
Predicted Label
COVID
Lung_Opacity
Normal
Viral Pneumonia
354
5
3
0
4
547
51
0
6
31
982
0
1
1
7
125
800
600
400
200
0

Model Training - Xception



Model Architecture

```
engine = tf.keras.applications.Xception(
        # Freezing the weights of the top layer in the InceptionResNetV2 pre-traiined model
        include_top = False,

        # Use Imagenet weights
        weights = 'imagenet',

        # Define input shape to 224x224x3
        input_shape = (224 , 224 , 3),

    )

x = tf.keras.layers.GlobalAveragePooling2D(name = 'avg_pool')(engine.output)
x =tf.keras.layers.Dropout(0.75)(x)
x = tf.keras.layers.BatchNormalization(
                      axis=-1,
                      momentum=0.99,
                      epsilon=0.01,
                      center=True,
                      scale=True,
                      beta_initializer="zeros",
                      gamma_initializer="ones",
                      moving_mean_initializer="zeros",
                      moving_variance_initializer="ones",
                  )(x)
out = tf.keras.layers.Dense(3, activation = 'softmax', name = 'dense_output')(x)


    # Build the Keras model
model = tf.keras.models.Model(inputs = engine.input, outputs = out)
    # Compile the model

model.compile(
        # Set optimizer to Adam(0.0001)
        optimizer = tf.keras.optimizers.Adam(learning_rate= 3e-4),
        #optimizer= SGD(lr=0.001, decay=1e-6, momentum=0.99, nesterov=True),
        # Set loss to binary crossentropy
        #loss = tf.keras.losses.SparseCategoricalCrossentropy(),
        loss = 'categorical_crossentropy',
        # Set metrics to accuracy
        metrics = ['accuracy']
    )
```



Model Summary

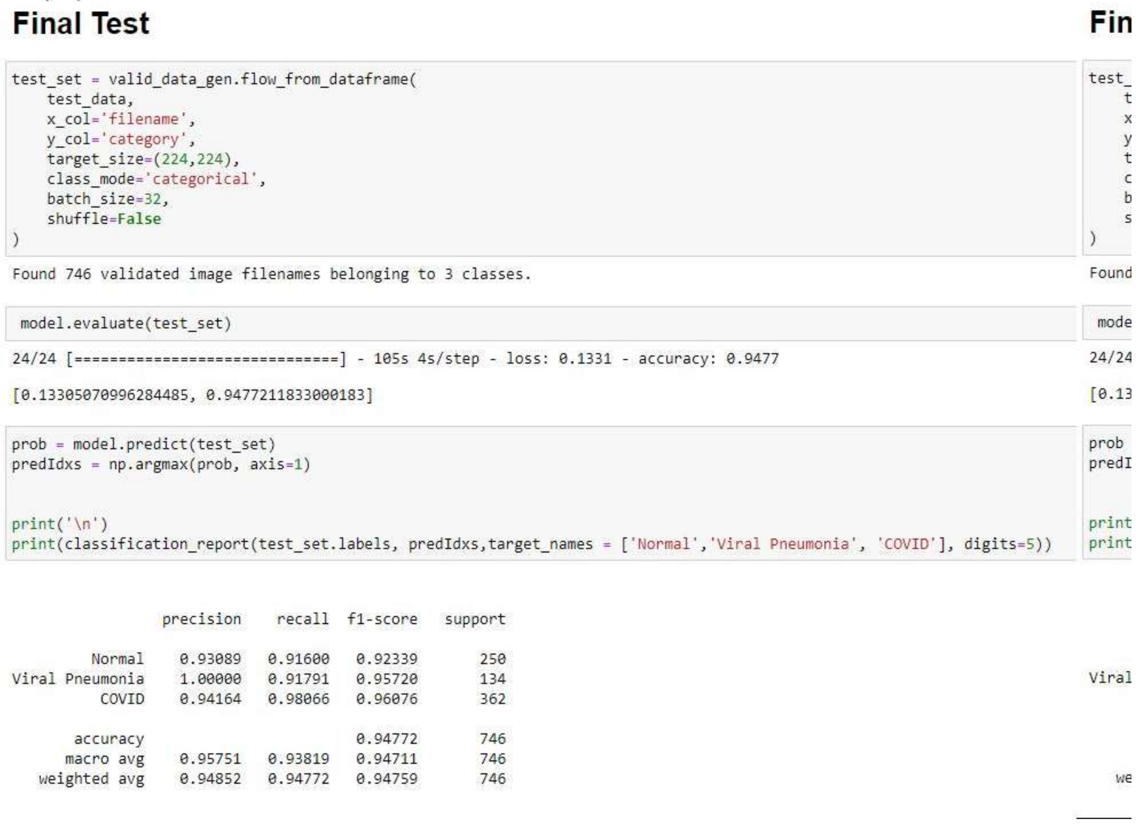
```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 block1_conv1 (Conv2D)          (None, 111, 111, 32  864         ['input_2[0][0]']
                                )

 block1_conv1_bn (BatchNormaliz  (None, 111, 111, 32  128        ['block1_conv1[0][0]']
 ation)                         )

 block1_conv1_act (Activation)  (None, 111, 111, 32  0           ['block1_conv1_bn[0][0]']
                                )

 block1_conv2 (Conv2D)          (None, 109, 109, 64  18432       ['block1_conv1_act[0][0]']
                                )
```

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
learning_rate_reduction = keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
                                                            patience=2,
                                                            verbose=2,
                                                            factor=0.5,
                                                            min_lr=0.00001)
reduce_lr =  keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                              patience=3, min_lr=0.00001)
```

```
 history = model.fit(train_generator,
                              validation_data=valid_generator, verbose=1, epochs=5,
                       callbacks=[early_stopping, reduce_lr , learning_rate_reduction]
                        )
```

```
Epoch 1/5
187/187 [==============================] - 2252s 12s/step - loss: 0.3572 - accuracy: 0.8683 - val_loss: 0.3103 - val_accuracy:
0.8902 - lr: 3.0000e-04
Epoch 2/5
187/187 [==============================] - 2211s 12s/step - loss: 0.1645 - accuracy: 0.9407 - val_loss: 0.1549 - val_accuracy:
0.9518 - lr: 3.0000e-04
```



Final Test

```
test_set = valid_data_gen.flow_from_dataframe(
    test_data,
    x_col='filename',
    y_col='category',
    target_size=(224,224),
    class_mode='categorical',
    batch_size=32,
    shuffle=False
)
```

```
Found 746 validated image filenames belonging to 3 classes.
```

```
 model.evaluate(test_set)
```

```
24/24 [==============================] - 105s 4s/step - loss: 0.1331 - accuracy: 0.9477

[0.13305070996284485, 0.9477211833000183]
```

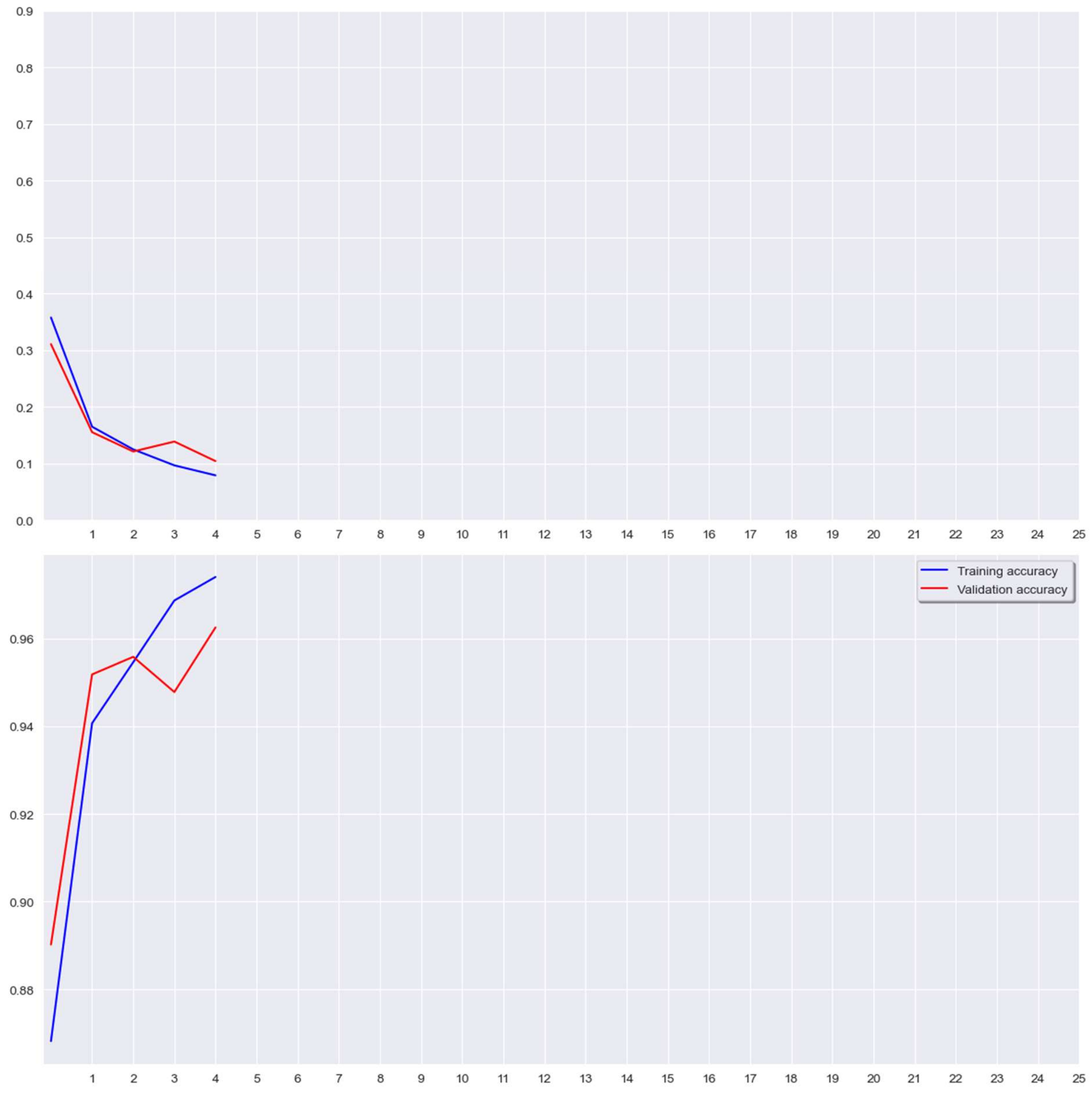
```
prob = model.predict(test_set)
predIdxs = np.argmax(prob, axis=1)


print('\n')
print(classification_report(test_set.labels, predIdxs,target_names = ['Normal','Viral Pneumonia', 'COVID'], digits=5))
```

```
                 precision    recall  f1-score   support

         Normal    0.93089   0.91600   0.92339       250
Viral Pneumonia    1.00000   0.91791   0.95720       134
          COVID    0.94164   0.98066   0.96076       362

       accuracy                        0.94772       746
      macro avg    0.95751   0.93819   0.94711       746
   weighted avg    0.94852   0.94772   0.94759       746
```

Plotting

- Training & Validation Loss
- Training & Validation Loss

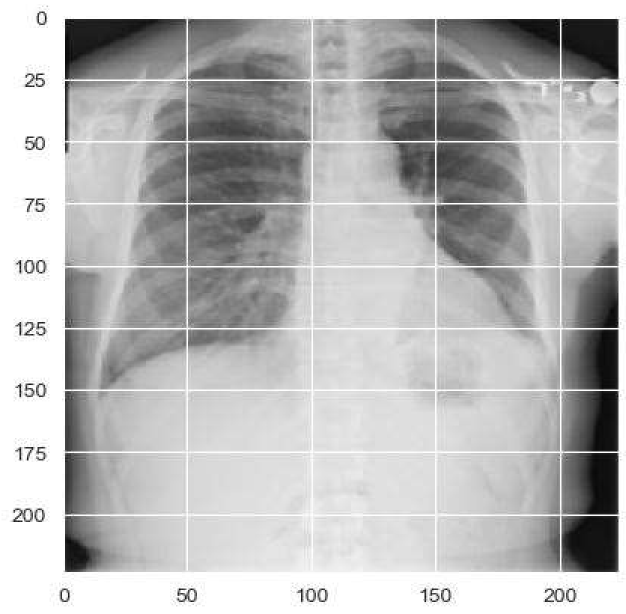
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
ax1.plot(history.history['loss'], color='b', label="Training loss")
ax1.plot(history.history['val_loss'], color='r', label="validation loss")
ax1.set_xticks(np.arange(1, 26, 1))
ax1.set_yticks(np.arange(0, 1, 0.1))

ax2.plot(history.history['accuracy'], color='b', label="Training accuracy")
ax2.plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
ax2.set_xticks(np.arange(1, 26, 1))

legend = plt.legend(loc='best', shadow=True)
plt.tight_layout()
plt.show()
```



Model Testing

```
sample = random.choice(test_data['filename'])


category = sample.split('/')[-1].split('-')[0].upper()
true = ''
if category == 'COVID':
    true = 'COVID'
elif category == 'VIRAL PNEUMONIA':
    true = 'Viral Pneumonia'
else:
    true = 'Normal'

print(f'True value is : {true}')

image = load_img(sample, target_size=(224, 224))
img = img_to_array(image)/255
img = img.reshape((1, 224, 224, 3))

result = model.predict(img)
result = np.argmax(result, axis=-1)
print('Prediction is:')
if result == 0:
    print("Normal")
elif result == 1:
    print("Viral Pneumonia")
else:
    print("COVID ")

plt.imshow(image)
```

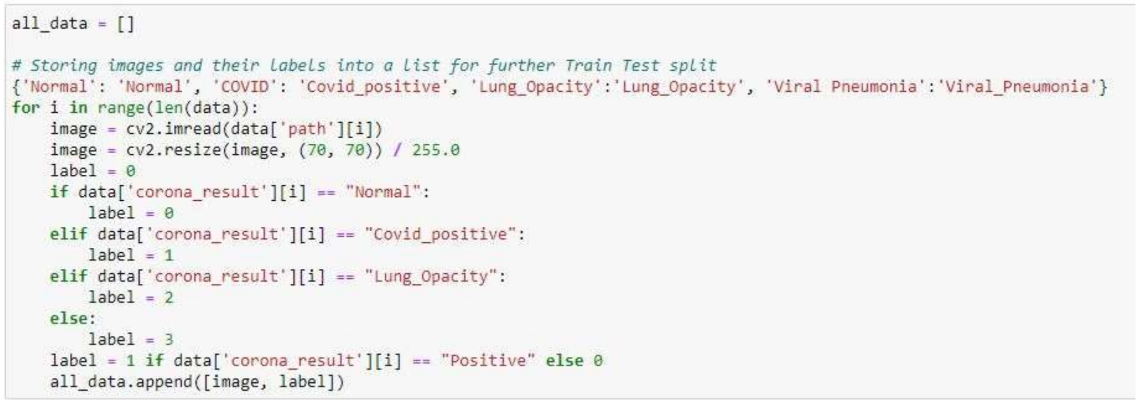
```
True value is : Normal
Prediction is:
Normal

<matplotlib.image.AxesImage at 0x14bc29dcec8>
```



GRADCAM

Grad-CAM (Gradient-weighted Class Activation Mapping) is a popular technique in the field of computer vision that helps to visualize the regions of an image that are important for a particular neural network's classification decision. It is a class activation mapping technique that uses the gradients of a target class flowing into the final convolutional layer of a convolutional neural network (CNN) to produce a coarse localization map highlighting the important regions in the image that contribute to the final classification output. The Grad-CAM technique is useful for understanding the decision-making process of a neural network and can also be used for model interpretability and debugging.

Modelling



```
all_data = []

# Storing images and their labels into a list for further Train Test split
{'Normal': 'Normal', 'COVID': 'Covid_positive', 'Lung_Opacity':'Lung_Opacity', 'Viral Pneumonia':'Viral_Pneumonia'}
for i in range(len(data)):
    image = cv2.imread(data['path'][i])
    image = cv2.resize(image, (70, 70)) / 255.0
    label = 0
    if data['corona_result'][i] == "Normal":
        label = 0
    elif data['corona_result'][i] == "Covid_positive":
        label = 1
    elif data['corona_result'][i] == "Lung_Opacity":
        label = 2
    else:
        label = 3
    label = 1 if data['corona_result'][i] == "Positive" else 0
    all_data.append([image, label])
```

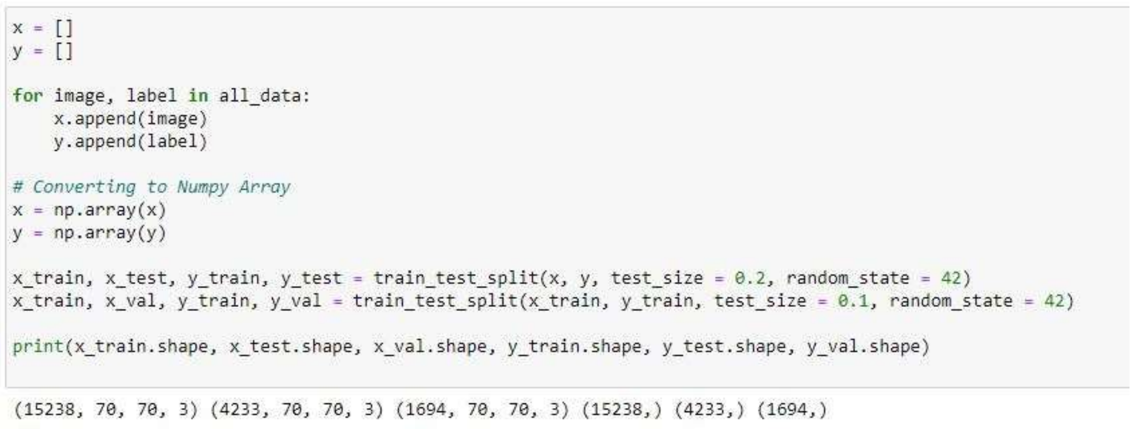
Data split

Train | Test | Validation - split

Splitting a dataset into train, test, and validation sets is a common practice in machine learning. The purpose of this split is to provide separate datasets for training, evaluating, and fine-tuning machine learning models.

Here are some specific reasons why we might split an image dataset into train, test, and validation sets:

1. **Model training**: The train set is used to fit or train the machine learning model. The model is trained on the training data, and the model parameters are adjusted to minimize the error between the predicted output and the true output.
2. **Model evaluation**: The test set is used to evaluate the performance of the trained model on unseen data. This allows us to gauge the generalization performance of the model and ensure that it is not overfitting to the training data.
3. **Model fine-tuning**: The validation set is used to fine-tune the model by adjusting the model parameters to optimize the model's performance on the validation data. This can help to ensure that the model is not overfitting to the training data, while still being able to perform well on unseen data.



```
x = []
y = []

for image, label in all_data:
    x.append(image)
    y.append(label)

# Converting to Numpy Array
x = np.array(x)
y = np.array(y)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = 0.1, random_state = 42)

print(x_train.shape, x_test.shape, x_val.shape, y_train.shape, y_test.shape, y_val.shape)
```

```
(15238, 70, 70, 3) (4233, 70, 70, 3) (1694, 70, 70, 3) (15238,) (4233,) (1694,)
```

We will be using 15238 images for training, 4233 images for testing, 1694 images for validation ~

Cnn model definition

MODEL ARCHITECTURE DEFINITION

We will be using CNN model with 3 convolutional layes,

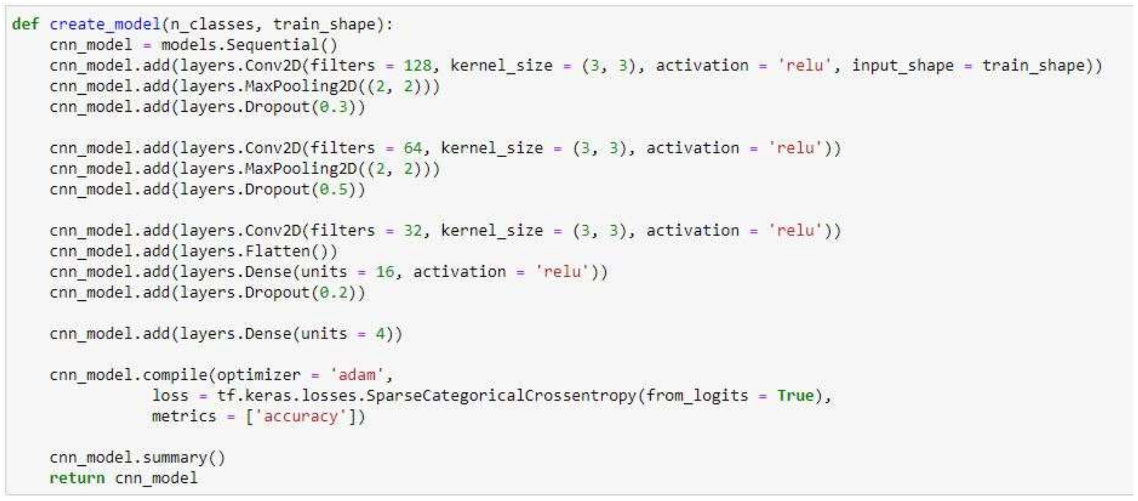
Layer 1 - filter = 128, activation = rectified linear unit

Layer 2 - filter = 64, activation = rectified linear unit

Layer 3 - filter = 32, activation = rectified linear unit

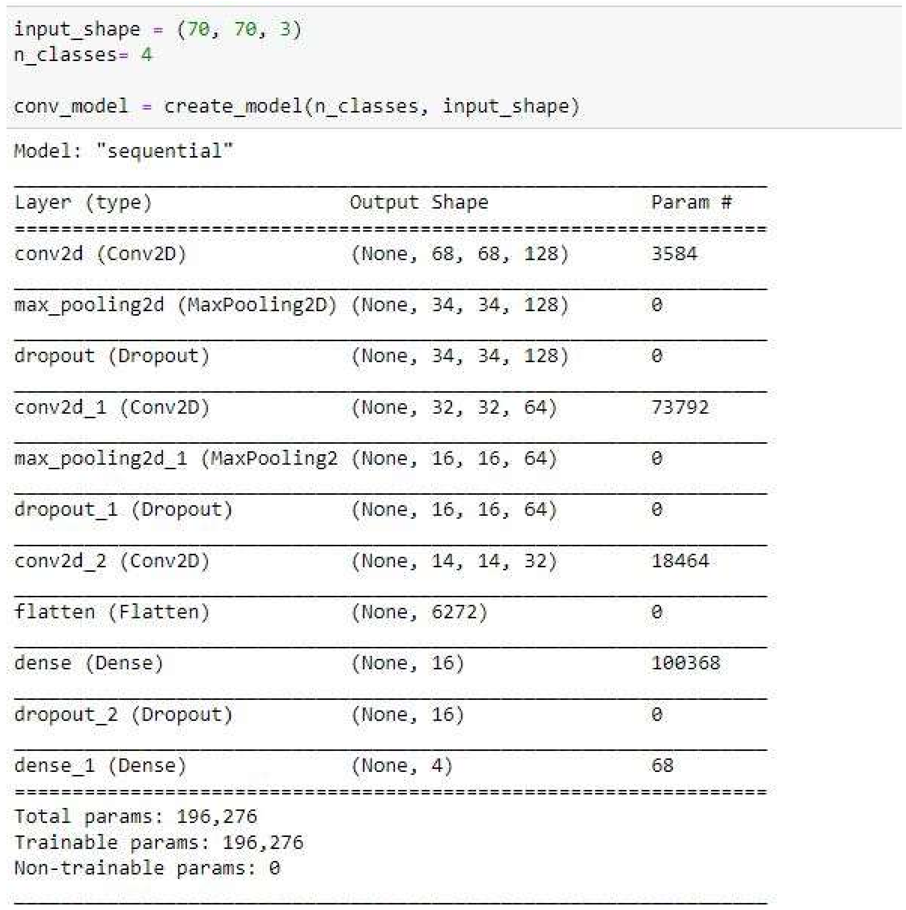
Optimizer used is ADAM

Loss => Sparse Categorical Crossentropy Metrics => Accuracy



```
def create_model(n_classes, train_shape):
    cnn_model = models.Sequential()
    cnn_model.add(layers.Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu', input_shape = train_shape))
    cnn_model.add(layers.MaxPooling2D((2, 2)))
    cnn_model.add(layers.Dropout(0.3))

    cnn_model.add(layers.Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
    cnn_model.add(layers.MaxPooling2D((2, 2)))
    cnn_model.add(layers.Dropout(0.5))

    cnn_model.add(layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu'))
    cnn_model.add(layers.Flatten())
    cnn_model.add(layers.Dense(units = 16, activation = 'relu'))
    cnn_model.add(layers.Dropout(0.2))

    cnn_model.add(layers.Dense(units = 4))

    cnn_model.compile(optimizer = 'adam',
               loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
               metrics = ['accuracy'])

    cnn_model.summary()
    return cnn_model
```

We have four classes such as [Normal, COVID, Lung_Opacity, Pneumonia]



```
input_shape = (70, 70, 3)
n_classes= 4

conv_model = create_model(n_classes, input_shape)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 68, 68, 128)       3584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 34, 34, 128)       0
_________________________________________________________________
dropout (Dropout)            (None, 34, 34, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        73792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        18464
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 16)                100368
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 68
=================================================================
Total params: 196,276
Trainable params: 196,276
Non-trainable params: 0
_________________________________________________________________
```

Callback definition

USE OF **PlotLossesCallback()**

PlotLossesCallback() is a function in the livelossplot library that allows users to plot the training and validation losses during the training of a deep learning model. The function works by creating a plot of the losses and updating the plot in real-time as the model is trained. This can be useful for understanding how the model is performing during training, identifying any trends or patterns in the losses, and debugging any issues that may arise.

Some specific uses of PlotLossesCallback() include:

1. **Visualizing the training process**: PlotLossesCallback() allows users to visualize the training process in real-time, which can be helpful for understanding how the model is learning and identifying any potential issues or trends.
2. **Debugging**: PlotLossesCallback() can be useful for debugging problems with the model, such as overfitting or underfitting. For example, if the training loss is much lower than the validation loss, it may indicate that the model is overfitting to the training data.
3. **Optimizing model performance**: PlotLossesCallback() can also be useful for optimizing the model performance by identifying when the model is starting to overfit or underfit and adjusting the model parameters accordingly.

USE OF ModelCheckpoint()

ModelCheckpoint() is a function in the Keras library that allows users to save the weights of a deep learning model during the training process. The function works by periodically saving the weights of the model to a specified file, allowing users to interrupt and resume training at a later time if necessary.

There are several reasons why using ModelCheckpoint() can be useful:

1. **Interrupting and resuming training**: ModelCheckpoint() allows users to interrupt training at any time and resume training later on by loading the saved weights. This can be useful if training takes a long time or if the training process is interrupted due to a power outage or other unexpected event.
2. **Saving the best model**: ModelCheckpoint() also allows users to specify a metric to monitor (e.g. validation loss) and save only the weights that result in the best performance on that metric. This can be useful for selecting the best model from a set of trained models.
3. **Saving intermediate models**: ModelCheckpoint() can also be used to save intermediate models during the training process, which can be useful for debugging or tracking the progress of the model over time.

USE OF EarlyStopping()

EarlyStopping() is a callback function in the Keras library that allows users to specify when to stop training a deep learning model based on the performance of the model on a validation dataset. The function works by monitoring a specified metric (e.g., accuracy or loss) and stopping training when the metric stops improving or starts to degrade. This can be useful for preventing overfitting and ensuring that the model is able to generalize to new data.

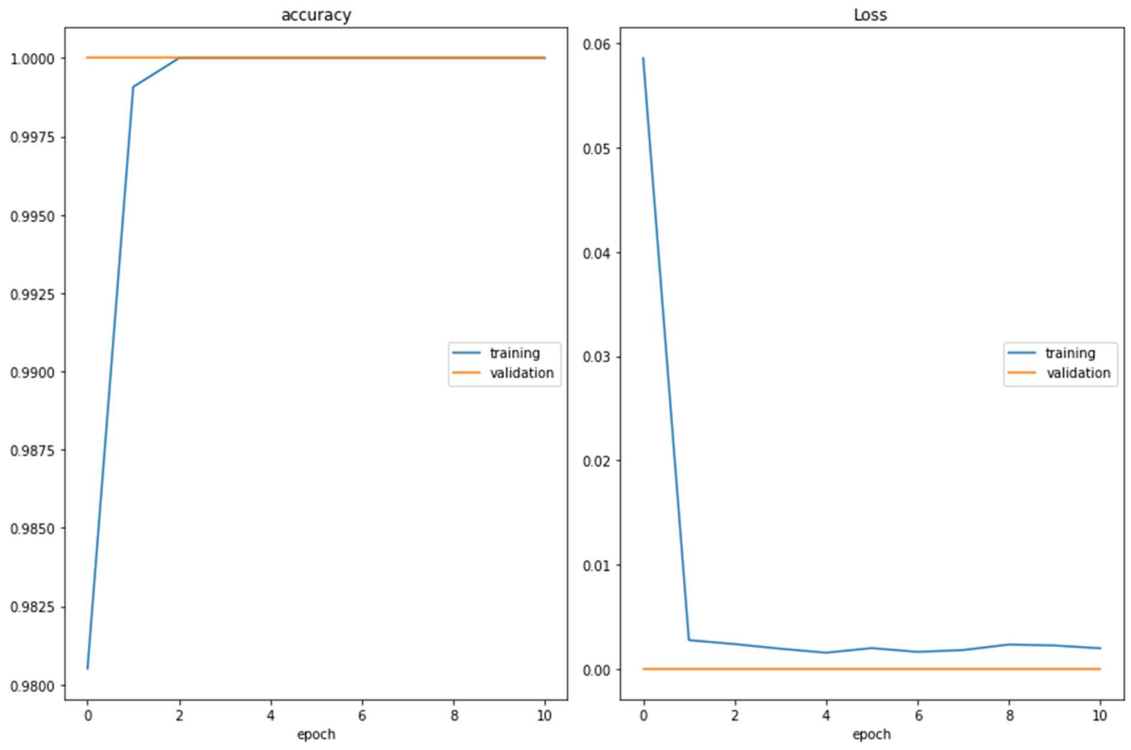
Some specific reasons why we might use EarlyStopping() include:

1. **Preventing overfitting**: One of the main benefits of using EarlyStopping() is to prevent overfitting, which is when a model performs well on the training data but poorly on unseen data. By monitoring the performance of the model on a validation dataset, EarlyStopping() can help to ensure that the model is not overfitting to the training data and is able to generalize to new data.
2. **Saving time**: Training deep learning models can be time-consuming, especially when the model is not improving. By using EarlyStopping(), users can save time by stopping the training process when the model's performance stops improving or starts to degrade.
3. **Improving model performance**: EarlyStopping() can also be useful for improving the performance of a model by identifying the optimal number of training epochs. This can help to ensure that the model is trained to the best possible performance without overfitting or underfitting.

```
plot_loss_1 = PlotLossesCallback()

tl_checkpoint_1 = ModelCheckpoint(filepath='tl_model_v1.weights.best.hdf5',
                                  save_best_only=True,
                                  verbose=1)

early_stop = EarlyStopping(monitor='val_loss',
                           patience=10,
                           restore_best_weights=True,
                           mode='min')
```

Let us train our model with training data.

```
%%time

plot_loss_2 = PlotLossesCallback()

conv_history = conv_model.fit(x_train, y_train,
                        epochs = 50, batch_size = 256,
                        validation_data = (x_val, y_val),
                        callbacks = [tl_checkpoint_1, early_stop, plot_loss_2],
                       verbose=1)
```



```
accuracy
        training                 (min:    0.981, max:    1.000, cur:    1.000)
        validation               (min:    1.000, max:    1.000, cur:    1.000)
Loss
        training                 (min:    0.002, max:    0.059, cur:    0.002)
        validation               (min:    0.000, max:    0.000, cur:    0.000)
CPU times: user 37.8 s, sys: 6.08 s, total: 43.9 s
Wall time: 48.9 s
```

Model validation

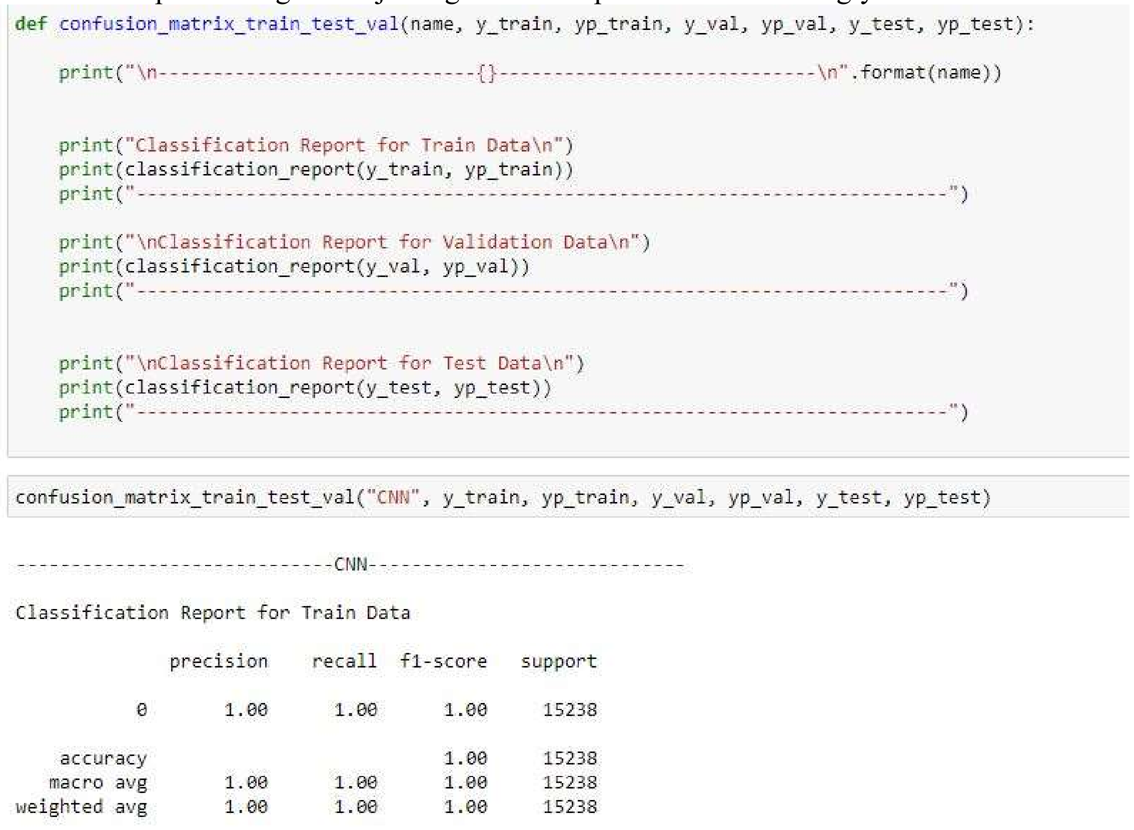
USE OF **classification_report()**

classification_report() is a function in the sklearn library that generates a report summarizing the performance of a classification model. The function takes as input the true labels and predicted labels for a set of data, and it produces a report that includes various evaluation metrics such as precision, recall, and f1-score.

The classification_report() function can be particularly useful as a callback during the training of a classification model because it allows users to monitor the performance of the model in real-time. This can be helpful for understanding how the model is performing and identifying any issues or trends that may be present.

Some specific uses of classification_report() as a callback include:

1. **Model evaluation**: Classification_report() can be used to evaluate the performance of a classification model on a particular dataset. This can be helpful for understanding how the model is performing and identifying any areas where it may be underperforming.
2. **Model comparison**: Classification_report() can also be used to compare the performance of different classification models on the same dataset. This can be useful for selecting the best model for a particular task.
3. **Model optimization**: Classification_report() can be useful for optimizing the performance of a classification model by identifying areas where the model is underperforming and adjusting the model parameters accordingly.



```
def confusion_matrix_train_test_val(name, y_train, yp_train, y_val, yp_val, y_test, yp_test):

    print("\n----------------------------{}----------------------------\n".format(name))


    print("Classification Report for Train Data\n")
    print(classification_report(y_train, yp_train))
    print("--------------------------------------------------------------------------")

    print("\nClassification Report for Validation Data\n")
    print(classification_report(y_val, yp_val))
    print("--------------------------------------------------------------------------")


    print("\nClassification Report for Test Data\n")
    print(classification_report(y_test, yp_test))
    print("--------------------------------------------------------------------------")
```

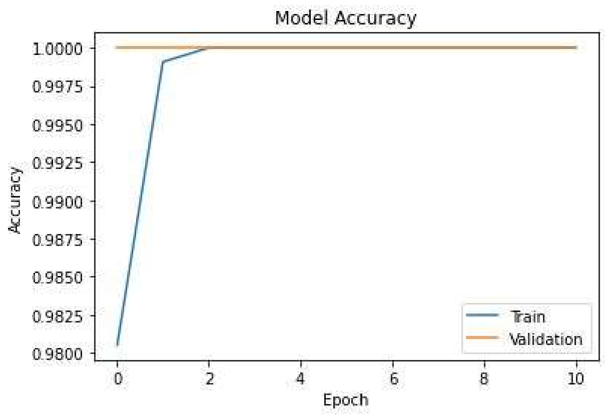
```
confusion_matrix_train_test_val("CNN", y_train, yp_train, y_val, yp_val, y_test, yp_test)
```

```
----------------------------CNN----------------------------

Classification Report for Train Data

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     15238

    accuracy                           1.00     15238
   macro avg       1.00      1.00      1.00     15238
weighted avg       1.00      1.00      1.00     15238
```

```
Classification Report for Validation Data

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1694

    accuracy                           1.00      1694
   macro avg       1.00      1.00      1.00      1694
weighted avg       1.00      1.00      1.00      1694

-------------------------------------------------------------------------

Classification Report for Test Data

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4233

    accuracy                           1.00      4233
   macro avg       1.00      1.00      1.00      4233
weighted avg       1.00      1.00      1.00      4233

-------------------------------------------------------------------------
```

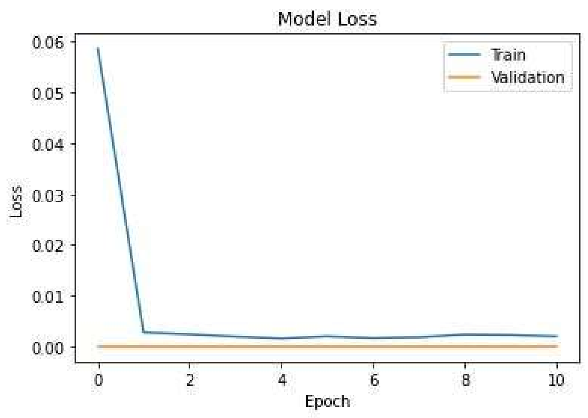
Model performance over accuracy

```
# Summarize History for Accuracy

plt.plot(conv_history.history['accuracy'])
plt.plot(conv_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'lower right')
plt.show()
```



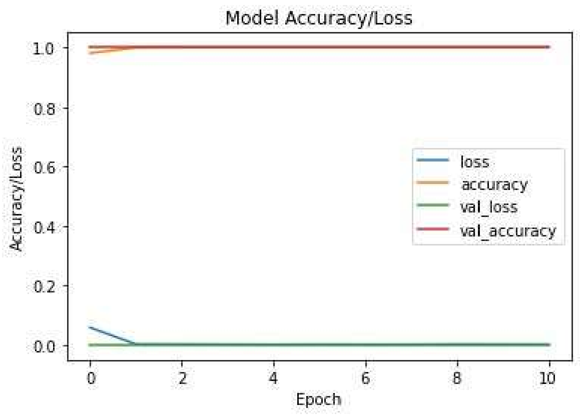
Model performance over loss

```
# Summarize History for Loss

plt.plot(conv_history.history['loss'])
plt.plot(conv_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper right')
plt.show()
```



Model performance over accuracy / loss

```
# Accuracy Loss Graph

pd.DataFrame(conv_history.history).plot()
plt.title('Model Accuracy/Loss')
plt.ylabel('Accuracy/Loss')
plt.xlabel('Epoch')
plt.show()
```



```
conv_model.save('covid_model.h5')
```

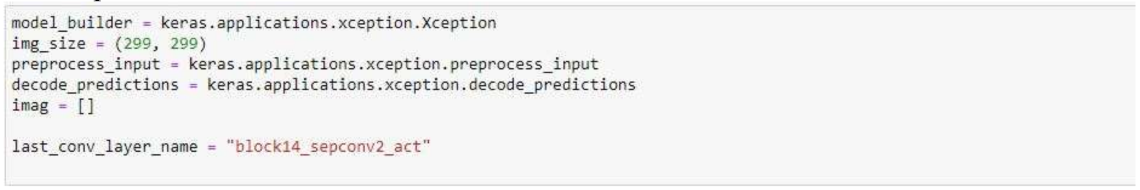
Grad cam visualization

IMPORTANCE OF GRAD-CAM VISUALIZATION

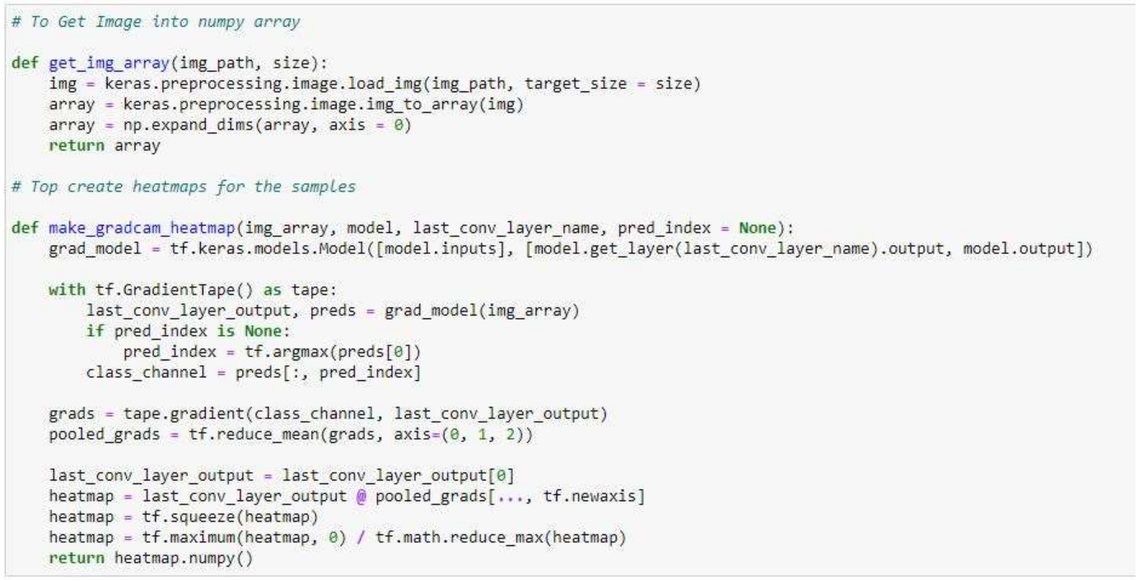
Grad-CAM (Gradient-weighted Class Activation Mapping) is a visualization technique that is used to understand which parts of an image are most important for a deep learning model's predictions. The technique works by generating a heatmap that highlights the regions of the image that the model is most sensitive to when making a prediction.

There are several reasons why Grad-CAM visualization can be important:

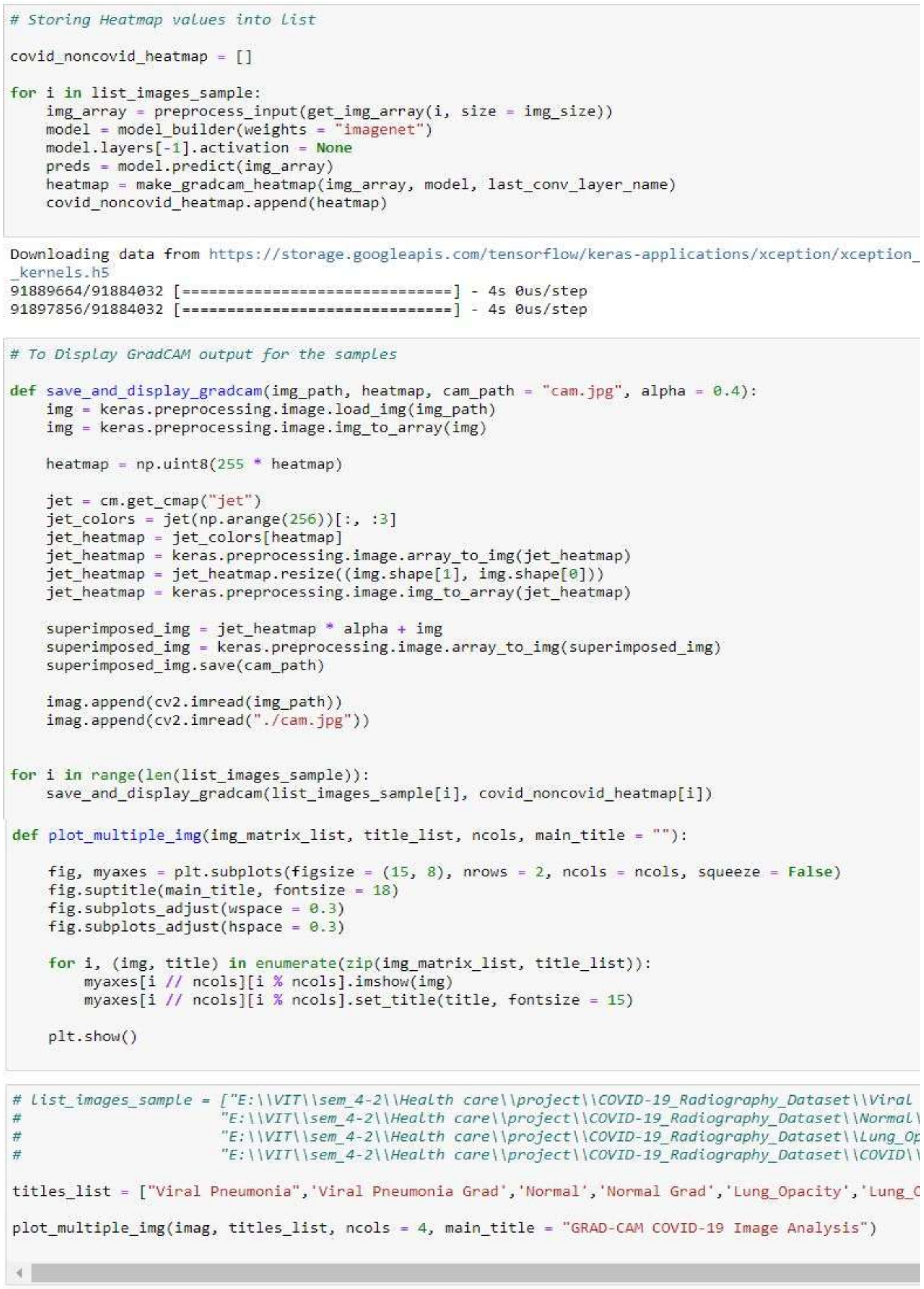
1. **Model understanding**: Grad-CAM visualization can be useful for understanding how a deep learning model is making its predictions. By highlighting the regions of the image that are most important for the model's predictions, Grad-CAM can help to provide insight into the decision-making process of the model.
2. **Model debugging**: Grad-CAM visualization can also be useful for debugging problems with a deep learning model. For example, if the model is making incorrect predictions, Grad-CAM can help to identify which parts of the image the model is focusing on and potentially identify any issues with the model's training or architecture.
3. **Model explanation**: Grad-CAM visualization can be useful for explaining the predictions of a deep learning model to non-technical stakeholders. By highlighting the regions of the image that are most important for the model's predictions, Grad-CAM can provide a more intuitive understanding of the model's decision-making process.



```python
model_builder = keras.applications.xception.Xception
img_size = (299, 299)
preprocess_input = keras.applications.xception.preprocess_input
decode_predictions = keras.applications.xception.decode_predictions
imag = []

last_conv_layer_name = "block14_sepconv2_act"
```



```python
# To Get Image into numpy array

def get_img_array(img_path, size):
    img = keras.preprocessing.image.load_img(img_path, target_size = size)
    array = keras.preprocessing.image.img_to_array(img)
    array = np.expand_dims(array, axis = 0)
    return array

# Top create heatmaps for the samples

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index = None):
    grad_model = tf.keras.models.Model([model.inputs], [model.get_layer(last_conv_layer_name).output, model.output])

    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    grads = tape.gradient(class_channel, last_conv_layer_output)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```



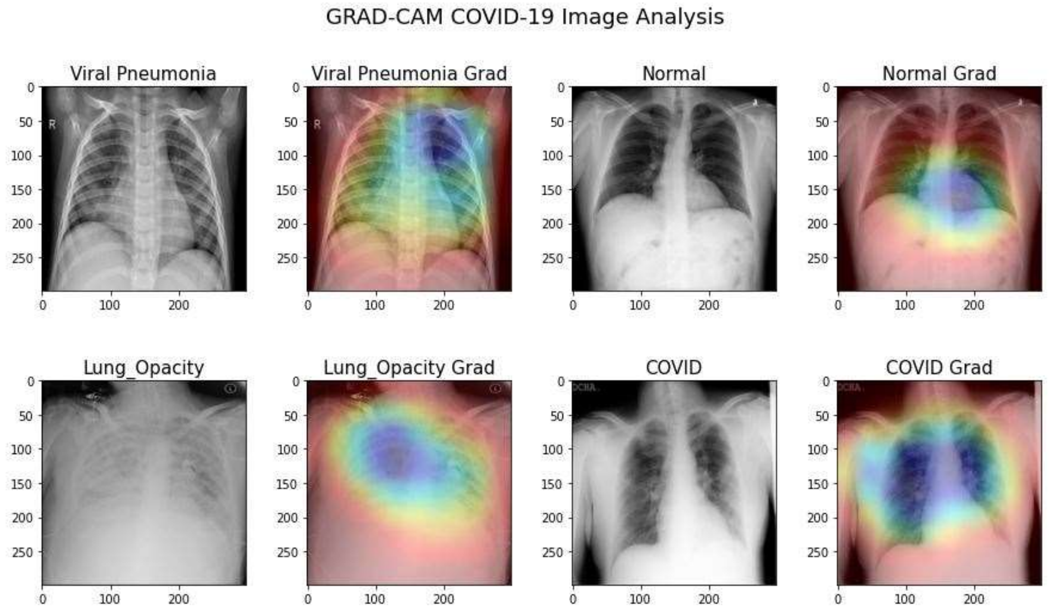
```
# Storing Heatmap values into List

covid_noncovid_heatmap = []

for i in list_images_sample:
    img_array = preprocess_input(get_img_array(i, size = img_size))
    model = model_builder(weights = "imagenet")
    model.layers[-1].activation = None
    preds = model.predict(img_array)
    heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
    covid_noncovid_heatmap.append(heatmap)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_
_kernels.h5
91889664/91884032 [==============================] - 4s 0us/step
91897856/91884032 [==============================] - 4s 0us/step
```

```
# To Display GradCAM output for the samples

def save_and_display_gradcam(img_path, heatmap, cam_path = "cam.jpg", alpha = 0.4):
    img = keras.preprocessing.image.load_img(img_path)
    img = keras.preprocessing.image.img_to_array(img)

    heatmap = np.uint8(255 * heatmap)

    jet = cm.get_cmap("jet")
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]
    jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)

    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)
    superimposed_img.save(cam_path)

    imag.append(cv2.imread(img_path))
    imag.append(cv2.imread("./cam.jpg"))


for i in range(len(list_images_sample)):
    save_and_display_gradcam(list_images_sample[i], covid_noncovid_heatmap[i])
```

```
def plot_multiple_img(img_matrix_list, title_list, ncols, main_title = ""):

    fig, myaxes = plt.subplots(figsize = (15, 8), nrows = 2, ncols = ncols, squeeze = False)
    fig.suptitle(main_title, fontsize = 18)
    fig.subplots_adjust(wspace = 0.3)
    fig.subplots_adjust(hspace = 0.3)

    for i, (img, title) in enumerate(zip(img_matrix_list, title_list)):
        myaxes[i // ncols][i % ncols].imshow(img)
        myaxes[i // ncols][i % ncols].set_title(title, fontsize = 15)

    plt.show()
```

```
# list_images_sample = ["E:\\VIT\\sem_4-2\\Health care\\project\\COVID-19_Radiography_Dataset\\Viral
#                       "E:\\VIT\\sem_4-2\\Health care\\project\\COVID-19_Radiography_Dataset\\Normal\
#                       "E:\\VIT\\sem_4-2\\Health care\\project\\COVID-19_Radiography_Dataset\\Lung_Op
#                       "E:\\VIT\\sem_4-2\\Health care\\project\\COVID-19_Radiography_Dataset\\COVID\\

titles_list = ["Viral Pneumonia",'Viral Pneumonia Grad','Normal','Normal Grad','Lung_Opacity','Lung_C

plot_multiple_img(imag, titles_list, ncols = 4, main_title = "GRAD-CAM COVID-19 Image Analysis")
```



Conclusion:

By using deep learning algorithms for the classification of COVID-19 using X-ray images can be a powerful tool for identifying and diagnosing the disease. By training a model on a large dataset of X-ray images labelled with COVID-19 diagnosis, it is possible to achieve high levels of accuracy and robustness in the model's predictions. Through the use of techniques such as image augmentation and model optimization, it is possible to improve the performance of the model and increase its ability to generalize to new data. Overall, deep learning algorithms can be a valuable tool for the classification of COVID-19 using X-ray images, and their use in this application has the potential to significantly impact the diagnosis and treatment of the disease.

The best model we trained is Xception model which estimated an accuracy of 97%. we also analyzed the images using grad-cam that which regions are varied with covid19 and pneumonia with VIBGYOR colors

References:

Jain, R., Nagrath, P., Kataria, G., Kaushik, V. S., & Hemanth, D. J. (2020). Pneumonia detection in chest X-ray images using convolutional neural networks and transfer learning. *Measurement*, *165*, 108046.

Chouhan, V., Singh, S. K., Khamparia, A., Gupta, D., Tiwari, P., Moreira, C., ... & De Albuquerque, V. H. C. (2020). A novel transfer learning based approach for pneumonia detection in chest X-ray images. *Applied Sciences*, *10*(2), 559.

Rahimzadeh, M., & Attar, A. (2020). A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2. *Informatics in medicine unlocked*, *19*, 100360.

Toğaçar, M., Ergen, B., Cömert, Z., & Özyurt, F. (2020). A deep feature learning model for pneumonia detection applying a combination of mRMR feature selection and machine learning models. *Irbm*, *41*(4), 212-222.

Mahmud, T., Rahman, M. A., & Fattah, S. A. (2020). CovXNet: A multi-dilation convolutional neural network for automatic COVID-19 and other pneumonia detection from chest X-ray images with transferable multi-receptive feature optimization. *Computers in biology and medicine*, *122*, 103869.

Zhang, J., Xie, Y., Pang, G., Liao, Z., Verjans, J., Li, W., ... & Xia, Y. (2020). Viral pneumonia screening on chest X-rays using confidence-aware anomaly detection. *IEEE transactions on medical imaging*, *40*(3), 879-890.

Han, Y., Chen, C., Tewfik, A., Ding, Y., & Peng, Y. (2021, April). Pneumonia detection on chest x-ray using radiomic features and contrastive learning. In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)* (pp. 247-251). IEEE.

E. Ayan and H. M. Ünver, "Diagnosis of Pneumonia from Chest X-Ray Images Using Deep Learning," *2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT)*, Istanbul, Turkey, 2019, pp. 1-5, doi: 10.1109/EBBT.2019.8741582.

Chowdhury, M. E., Rahman, T., Khandakar, A., Mazhar, R., Kadir, M. A., Mahbub, Z. B., ... & Islam, M. T. (2020). Can AI help in screening viral and COVID-19 pneumonia?. *Ieee Access*, *8*, 132665-132676.

A. Pant, A. Jain, K. C. Nayak, D. Gandhi and B. G. Prasad, "Pneumonia Detection: An Efficient Approach Using Deep Learning," *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2020, pp. 1-6, doi: 10.1109/ICCCNT49239.2020.9225543.

Z. Li *et al*., "PNet: An Efficient Network for Pneumonia Detection," *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, Suzhou, China, 2019, pp. 1-5, doi: 10.1109/CISP-BMEI48845.2019.8965660.

D. Varshni, K. Thakral, L. Agarwal, R. Nijhawan and A. Mittal, "Pneumonia Detection Using CNN based Feature Extraction," *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore, India, 2019, pp. 1-7, doi: 10.1109/ICECCT.2019.8869364.

A. Tilve, S. Nayak, S. Vernekar, D. Turi, P. R. Shetgaonkar and S. Aswale, "Pneumonia Detection Using Deep Learning Approaches," *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, Vellore, India, 2020, pp. 1-8, doi: 10.1109/ic-ETITE47903.2020.152.

Hashmi MF, Katiyar S, Keskar AG, Bokde ND, Geem ZW. Efficient Pneumonia Detection in Chest Xray Images Using Deep Transfer Learning. Diagnostics (Basel). 2020 Jun 19;10(6):417. doi: 10.3390/diagnostics10060417. PMID: 32575475; PMCID: PMC7345724.

GM, H., Gourisaria, M. K., Rautaray, S. S., & Pandey, M. A. N. J. U. S. H. A. (2021). Pneumonia detection using CNN through chest X-ray. *Journal of Engineering Science and Technology (JESTEC)*, *16*(1), 861-876.

Gupta, P. (2021). Pneumonia detection using convolutional neural networks. *Science and Technology*, *7*(01), 77-80.