COVID-19 Detection using X-rays with Artificial Neural Networks

Siddhesh Desai

Student ID: 13008482

*MSc Data Science and Computational Intelligence, Coventry*

United Kingdom

Email: desais10@uni.coventry.ac.uk

*Abstract*—The implementation of Artificial Neural Networks for detecting COVID-19 using X-ray images is an innovative approach that has shown promising results. This technology has been able to detect the presence of the virus in individuals with a high degree of accuracy, thereby helping medical professionals to identify and isolate infected patients quickly. Moreover, it has been a significant step forward in the fight against the pandemic by reducing the number of false positives and ensuring that medical resources are utilized efficiently. With the continued development of this technology, we can expect to see a significant impact in the battle against COVID-19, making it easier to identify and treat those affected by the virus. While there is still a long way to overcome the virus completely, Artificial Neural Networks offer a glimpse of hope and a new way of approaching such healthcare issues in the future.

Keywords—COVID-19, Artificial Neural Network, Deep Learning, Image Classification

# Introduction

The SARS-CoV-2 virus is responsible for causing COVID-19, and as it undergoes mutations, new variants can emerge. Changes in the virus's genetic material can affect its transmissibility, severity, and response to vaccines and treatments. Several noteworthy variants, such as Alpha, Beta, Gamma, and Delta, have surfaced worldwide and are known to spread more quickly or cause more severe illness. Monitoring and researching these variants are vital to comprehend the virus's behaviour and developing effective measures to manage its spread. Immunization efforts and public health precautions like social distancing, mask-wearing, and hand hygiene remain essential in lessening the impact of COVID-19 and its mutations. Although, in 2023, the number of COVID-19 patients has decreased significantly but still the different variants possess a threat at times.

The 2019 novel coronavirus (COVID-19) has distinct characteristics. While polymerase chain reaction (PCR) confirms the diagnosis, chest X-ray and computed tomography (CT) images can reveal pneumonia with a moderately distinctive pattern for human observation. It is crucial to accurately identify infected patients with low false negative rates to control the virus's transmission. Moreover, it is equally essential to avoid false positives to prevent overwhelming the healthcare system by subjecting patients to unnecessary quarantine. Prompt detection of COVID-19 is critical for providing supportive care to patients and controlling its spread.

A Chinese research team published a paper that described the clinical and paraclinical characteristics of COVID-19. According to their findings, patients with COVID-19 exhibit abnormalities in chest CT images, with most patients having bilateral involvement. Chest CT images of ICU patients on admission reveal multiple lobular and subsegmental areas of consolidation, while non-ICU patients show bilateral ground-glass opacity and subsegmental areas of consolidation. Later chest CT images of these patients display bilateral ground-glass opacity with resolved consolidation. Radiological imaging could be a better diagnostic tool for COVID-19, according to Fang (2020) and Ai (2020).

As a part of this paper, we will be making use of different architectures and learning approaches available in neural network systems and constructing a model which will help in detecting if a particular patient is COVID-positive or not based on their X-ray image scan.

# Experimental Setup

* Code Repository – All the code of this experiment will be uploaded to a public Github repo. The link for the repo would be mentioned in the Appendix section.
* Google Colab – The code notebook was created and executed on the Google Colab platform.
* Dataset – Taken from Kaggle. The data input files will also be uploaded to Github so that it is easy to find everything in one place.

# Related Work

A few studies have been done and research papers have been published relating to the work of detecting COVID-19 using X-ray images and CT scan images.

One such paper published showcases how a deep convolutional neural network detects COVID-19 using X-rays (Bassi and Attux, 2021) [1]. This paper discusses the performance of different deep neural networks (DNNs) in COVID-19 detection using X-ray images. The authors used transfer learning and output neuron-keeping techniques to improve the accuracy of their models. They found that DNNs with output neuron keeping performed better than those without it. They also used Layer-wise Relevance Propagation (LRP) to understand the factors that influenced the classification of X-ray images by the DNNs. The authors found that words and letters on the X-ray images could introduce bias in the classification, but this had only a small effect on the accuracy of the DNNs. The study concluded that their proposed method and output neuron-keeping technique could improve the performance of DNNs in other classification problems. The authors also suggested that LRP could help radiologists in identifying the factors that influence the classification of X-ray images. Finally, the authors noted that their study was on par with the current state-of-the-art in COVID-19 detection with deep learning.

In the second paper published by Hussain et al., 2021 [2], the authors propose a new convolutional neural network (CNN) model called CoroDet, which can accurately detect COVID-19 using chest X-ray and CT images. The model is capable of providing accurate diagnostics for 2, 3, and 4 class classifications (COVID, Normal, Pneumonia, non-COVID viral pneumonia, and non-COVID bacterial pneumonia). The authors claim that the classification accuracy of the proposed model is the highest achieved accuracy to the best of their knowledge on the datasets used in the experiments. One of the significant contributions of this paper is the preparation of the largest dataset for the evaluation of classification algorithms, which is an essential step in developing a robust model. The authors also compare the performance of CoroDet with ten existing techniques and demonstrate its superiority in terms of accuracy. They provide empirical justification for their 22-layer model and show that the outcome of the model was accepted by a clinician. While the proposed model shows high accuracy in detecting COVID-19, the authors acknowledge that the hardware limitations prevented them from using larger image sets to train their model. Therefore, they plan to improve the model's performance by including more images in the training stage in the future.

In the paper published by Rahman et al., 2021 [3], the study aims to explore the use of CXR images for the detection of COVID-19, which is crucial for preventing the spread of the virus. The researchers compiled the largest CXR dataset, COVQU, which includes COVID-19, non-COVID lung opacity, and normal X-ray images, and proposed a novel variant of U-Net architecture for lung segmentation from X-ray images. They used deep Convolutional Neural Networks and evaluated the performance of seven different models for five different image enhancement techniques for the classification of COVID-19, non-COVID lung infection, and normal CXR images. The study's results show that a reliable COVID-19 diagnosis can be achieved with an accuracy, precision, and recall of 96.29%, 96.28%, and 96.28% without segmentation and 95.11%, 94.55%, and 94.56% with segmentation, respectively. The study also confirms the importance of accurate lung segmentation from CXR images, which can assist machine learning models in diagnostic decisions. This deep AI-based system can be useful as a fast screening tool that can save lives or prevent casualties, especially during the pandemic period when casualties can happen due to delay or miss-diagnosis.

# Dataset

The dataset chosen for this experiment is a public dataset which is hosted on Kaggle.

## A. Link to Dataset

<https://www.kaggle.com/datasets/khoongweihao/covid19-xray-dataset-train-test-sets>

## B. Description

This dataset is constructed as a part of the project (ieee8023, *IEEE8023/covid-chestxray-dataset: We are building an open database of COVID-19 cases with chest X-ray or CT images.*) [4] which is hosted on GitHub. The project's main emphasis was to gather as many X-ray or CT scan images as they could through indirect collection from hospitals and physicians and other public sources. Since the data contains medical data of actual patients so the number of images gathered has been limited. This dataset was extracted at a given point in time and was categorized into normal patients without any COVID or other infections and the other group of patients had PNEUMONIA being detected in their chest.

The dataset had X-ray images classified as train and test. Data in the train folder would be used to train the model and data in the test folder would be used to make predictions and calculate the accuracy of the model. Furthermore, each category is again divided into NORMAL and PNEUMONIA indicating the patient X-ray types and their images.

To train the algorithm we have 74 PNEUMONIA as well as NORMAL patient X-rays. For testing, we have 20 of each type again.

## C. Pre-processing data

Although, the data is classified and provided still there is a need for some pre-processing before using the data to train the model. The raw data looks like below.



Figure :X-ray images before pre-processing

After looking at the above images we can see that,

* Not all images have the right color format.
* They are not of the same size. We need to resize them and match the input size for the input layer of the deep learning model.
* Need to convert image pixels to float datatype.
* Convert the data into a NumPy array or a tensor object.

Once the above pre-processing is completed the image files are converted as below.

A picture containing text, x-ray film, dark

Description automatically generated

Figure : X-ray images after pre-processing

The last step before modelling the data is to convert the text labels into numeric codes for the classification task. In this case, all the NORMAL cases were converted to numeric code ‘0’ and PNEUMONIA to ‘1’.

# Ethical Considerations

Since the data that we are dealing with is sensitive and personal there are a few ethical considerations we need to make while gathering the data.

* Informed Consent: Prior to collecting any data, it is important to obtain informed consent from the patients whose X-ray images are being used in the dataset. Patients should be fully informed about the purpose of the study, the risks and benefits, and their right to refuse participation. Additionally, all patient information should be kept confidential and anonymous.
* Privacy: Protecting the privacy of patients is crucial when working with medical data. X-ray images should be de-identified to ensure that no personal information can be traced back to individual patients. Moreover, the dataset should be stored securely to prevent unauthorized access.
* Bias: Bias in the dataset can lead to skewed results and unfair outcomes. Therefore, it is important to ensure that the dataset is representative of the population it is meant to serve. To prevent bias, it is important to collect data from a diverse range of patients.
* Data Usage: Once the dataset is created, it is important to consider how it will be used. Researchers should ensure that their use of the dataset aligns with the original informed consent and that the data is used only for the purpose for which it was collected.
* Social Responsibility: Finally, researchers should consider the potential impact of their work on society. They should aim to use the dataset to improve patient outcomes and contribute to scientific knowledge. It is important to conduct research in a responsible and transparent manner that takes into account the broader social implications of the work.

# Methods

## Convolutional Neural Network (CNN)

A Convolutional Neural Network, also known as a ConvNet or CNN, is a Deep Learning technique that has the capability to identify different objects and aspects of an input image by assigning learnable weights and biases to them. Unlike other classification algorithms, the pre-processing required in a ConvNet is minimal. While traditional methods involve manually engineering filters, ConvNets can learn these filters through training.

The structure of a ConvNet is similar to the connectivity pattern of neurons in the human brain, and it is based on the organization of the visual cortex. In the visual cortex, neurons respond to stimuli only within a specific area called the receptive field. A group of receptive fields overlap to cover the entire visual area.

A Convolutional Neural Network consists of multiple layers and these layers are added or removed to get the appropriate precision from the model. The layers could be as below

* Convolutional Layer
* Pooling Layer
* Fully Connected Layer

Diagram

Description automatically generated

Figure : A CNN sequence (Saha, 2022) [5]

A Convolutional layer is a type of layer in neural networks that uses convolution, a mathematical operation that combines two functions to produce a third function. It applies a set of filters to the input data and produces a set of output values by sliding each filter over the input. The output values can then be used to extract features and make predictions by passing them through other layers in the neural network. Convolutional layers are ideal for image recognition tasks because they can process spatial data with local patterns and structures.

The Pooling layer, like the Convolutional Layer, reduces the size of the Convolved Feature. This helps reduce computational requirements by reducing dimensionality. It is also effective in extracting dominant features that are invariant to rotation and position, which facilitates effective model training. Two types of Pooling exist: Max Pooling and Average Pooling. Max Pooling selects the maximum value from the Kernel-covered portion of the image, while Average Pooling calculates the average of all the values in that region. Max Pooling acts as a noise suppressant by discarding noisy activations and reducing dimensionality, while also reducing noise. In contrast, Average Pooling only reduces dimensionality and serves as a noise-suppression technique. Therefore, Max Pooling is more effective than Average Pooling.

Incorporating a Fully-Connected layer is a cost-effective approach to learning non-linear combinations of the high-level features represented in the output of the convolutional layer. The Fully-Connected layer learns a potentially non-linear function within that space.

As a part of this experiment, we have created a custom CNN model. A sequential model is initialized in Keras, which is a linear stack of layers. We then start adding layers to it. A first convolutional layer is added to the model which uses a ReLU activation function. To add more filters a second layer of a convolutional layer is added with 64 filters. A pooling layer comes in as the third layer which is used in a down-sampling operation that reduces the spatial dimensions of the output from the convolutional layers. After these layers, a dropout regularization is added to avoid overfitting by randomly dropping out some of the outputs from the previous layers during training.

One more set of convolutional, max-pooling layers and dropout regularizations are added. All the output from the convolutional layers is flattened into a 1D tensor output before processing it further. A fully connected layer is added which uses ReLU activation and more dropout regularization is added to the fully connected layer. Finally, an output layer is added to the model which has a single unit with a sigmoid activation function.

Text

Description automatically generated

Figure : Summary of the custom CNN model

In this model, all the parameters are trainable and there are absolutely no non-trainable parameters.

## Xception Model

Xception, also known as Extreme Inception, is a convolutional neural network architecture designed by François Chollet, the creator of Keras, for image classification tasks. It was introduced in 2016 and is an advanced version of the Inception architecture, another deep neural network model.

The fundamental difference between Xception and Inception is that Xception uses depthwise separable convolutions instead of regular convolutions. Depthwise separable convolutions have two stages: the first stage performs a depthwise convolution by applying a single filter to each input channel individually, and the second stage applies a 1x1 convolution to combine the outputs from the previous stage. By using depthwise separable convolutions, Xception reduces the number of trainable parameters in the model while maintaining or even improving its accuracy. This makes Xception an efficient model, particularly for use in resource-limited settings such as mobile devices.

Xception has been demonstrated to outperform other models on multiple benchmark datasets for image classification, including the ImageNet Large Scale Visual Recognition Challenge.  
  
Diagram

Description automatically generated

Figure : Xception architecture (Nutan, Image classification with Xception Model 2021) [9]

As part of this experiment, just like CNN, we have also implemented an Xception model. The input data was first processed to fit the model perfectly. The image\_size was set to 224,224 for both the train and test data and the label\_mode was set to binary as we have only two classes to classify.

There are two pipeline operations that were created, one for data augmentation and another for the normalization of images. The first pipeline, data\_augmentation\_pipeline, is created using the keras.Sequential() function. It defines a sequence of data augmentation operations that can be applied to the images before they are fed into the Xception model. This has a bunch of operations that randomly flip the input image horizontally, rotate the image to a specific angle, applies random zoom to the image, and a random height and width shift.

The model was created using Keras and TensorFlow. Since all the images being fed to the model were resized to a particular format so the model was also defined using a tuple which set the input size of the images which could be used to train the model. A model checkpoint was set so that while training the model only the best model could be saved. An input layer is defined, and the data augmentation pipeline and the normalization pipeline are applied to the layer. The output of the pipelines is passed through the base Xception model, and a global average pooling layer is added to condense the output of the base model. A dense output layer with a sigmoid activation function is added to produce the final binary classification output. Finally, the keras.Model() function is used to create the final model by specifying the input layer and output layer. The resulting model is then used for training and evaluation. Then the model is compiled, and the summary of the model looks like below  
  
Text

Description automatically generated

Figure : Summary of the Xception model

In the context of building a deep learning model, the number of parameters refers to the number of learnable variables in the model. These variables are adjusted during the training process to enable the model to make accurate predictions. In this case, it appears that the total number of parameters to train in the Xception model was less than the custom CNN model that was previously created. This means that the Xception model required fewer learnable variables, making it more efficient in terms of computational resources and training time. Furthermore, the Xception model had some non-trainable parameters, unlike the custom CNN model where all the parameters were trainable. Non-trainable parameters are pre-set and cannot be updated during the training process, while trainable parameters are updated based on the input data during the training process. The fact that the Xception model had non-trainable parameters means that some aspects of the model were already optimized and did not require further optimization during the training process. As a result, the training time for the Xception model might have been reduced, and the potential for overfitting could have been lower. Overall, the Xception model's lower number of trainable parameters and non-trainable parameters could be advantageous in terms of training time, computational efficiency, and model performance.

# Experimental Results

## Custom CNN Model

Graphical user interface, application

Description automatically generated

Figure : Training and Validation matrix of the custom CNN model

A picture containing shape

Description automatically generated

Plot : Model Loss for the custom CNN model

Chart, line chart

Description automatically generated

Plot : Model Accuracy for the custom CNN model

Calendar

Description automatically generated

Figure : Accuracy score for the custom CNN model

## Xception Model

Graphical user interface, application

Description automatically generated

Figure :Training and Validation matrix of the Xception model

A picture containing graphical user interface

Description automatically generated

Plot : Model Loss for the Xception model

A picture containing graphical user interface

Description automatically generated

Plot : Model Accuracy for the Xception model



Figure : Accuracy score for the Xception model

# Discussion

An epoch is a term used in computing and machine learning to describe a full cycle of iteration through a dataset that occurs during the training phase of a machine learning model, such as a neural network. The model processes the entire dataset, usually in batches, and updates its weights and biases to improve its performance on the given task. After each epoch, the model's performance is assessed on a validation set, and the process is repeated for a certain number of epochs until the model's performance reaches an acceptable level or until it converges. Both models were trained for 20 epochs each.

During the start of epochs for the custom CNN model the training loss started with a much higher value and it gradually decreased as the number of epochs continued. Similarly, the training accuracy was less at the start and gradually it showed improvement and reached 1. For Validation loss, the numbers stayed more or less in the same range throughout the epochs. However, the validation accuracy showed a sudden drop at the start and later as well some minor drops.

The Xception model exhibited a high value of training loss at the beginning of the epochs, which became constant after the second epoch. However, the validation loss remained constant throughout all the epochs. The training accuracy, on the other hand, started at a relatively high value of 0.60 but experienced a drastic drop and remained constant at 0.50 thereafter. Meanwhile, the validation accuracy stayed the same throughout all the epochs.

This behaviour suggests that the Xception model may have to overfit the training data, as the training accuracy started high but dropped significantly, while the validation accuracy remained constant. This could indicate that the model learned the training data too well and failed to generalize to new data. To address this issue, techniques such as regularization or data augmentation could be employed during training to prevent overfitting and improve the model's generalization performance.

Overall, the custom CNN model showed a greater value of accuracy proving it to be a better fit for the dataset.

# Conclusion

When training a machine learning model, selecting the number of epochs can be a challenging task. Setting too few epochs can lead to an underfit model, which means it has not learned enough from the data and setting too many epochs can cause overfitting. Overfitting occurs when a model becomes too specialized in the training data, resulting in poor generalization of new data.

In addition to regularization and data augmentation techniques, another way to prevent overfitting is to monitor the model's performance during training by plotting the learning curves. Learning curves visualize how the model's training and validation loss and accuracy change over the epochs. By analyzing the learning curves, one can determine if the model is underfitting or overfitting and adjust the hyperparameters accordingly.

It is worth noting that the behaviour of the model during training can also depend on the complexity of the dataset and the architecture of the model. For instance, if the dataset is simple, a model with a large number of parameters may overfit the data. In contrast, if the dataset is complex, a simpler model may underfit the data. Therefore, selecting an appropriate model architecture for a given dataset is critical for achieving good performance.

In summary, the choice of the number of epochs during training is an essential hyperparameter that can significantly impact the performance of the machine learning model. Monitoring the learning curves and employing techniques such as regularization and data augmentation can help prevent overfitting, ensuring that the model generalizes well to new data. Finally, selecting an appropriate model architecture for the dataset can also play a crucial role in the performance of the model.

The custom CNN model exhibited a typical pattern of decreasing training loss and increasing training accuracy over the epochs, while the validation loss remained stable, and the validation accuracy showed some fluctuations. In contrast, the Xception model showed a high training loss at the beginning, which stabilized after the second epoch, while the validation loss remained constant throughout. However, the training accuracy dropped significantly, and the validation accuracy remained constant, suggesting overfitting of the training data.

Based on these observations, it can be concluded that the custom CNN model performed better than the Xception model, as it achieved higher accuracy on the dataset. To further improve the generalization performance of the models, techniques such as regularization or data augmentation could be employed to prevent overfitting and improve the models' ability to generalize to new data.

# Future Work

There are several potential areas for future work in research on COVID-19 detection using X-rays with artificial neural networks (ANNs).

* Expanding the dataset by collecting more X-ray and CT-scan images from diverse sources can increase the sample size and variation of the data, providing the model with more information to learn from. This could improve the accuracy and generalizability of the model's predictions and decrease the risk of overfitting to the limited set of images in the original dataset.
* Moreover, training a model to distinguish between CT-scan and X-ray images can enhance the understanding of a patient's health condition by enabling more specific diagnosis and treatment recommendations. CT-scan images provide a more detailed and comprehensive view of a patient's internal organs and tissues than X-ray images, which can be particularly useful in identifying and assessing complex medical conditions.
* By integrating a model that can differentiate between the two types of images, medical professionals can improve their diagnostic accuracy and customize treatment plans accordingly. This could potentially lead to more effective and efficient healthcare services and better outcomes for patients.
* To enhance the accuracy of COVID-19 detection using X-rays, there is a scope for improvement in ANN models. Researchers can explore various techniques such as new architectures, hyperparameters, and data augmentation methods to improve the model's accuracy.
* The lack of interpretability of ANNs has been a significant issue, as they are often perceived as black boxes. Developing an explainable model that can explain the decision-making process of the model can increase the transparency and trustworthiness of the model.
* To determine which ANN architecture is best suited for COVID-19 detection using X-rays, different architectures such as CNNs, RNNs, and transformer models can be compared.
* To boost the accuracy of COVID-19 detection using X-rays, researchers can build transfer learning models that use pre-trained models such as VGG16 or ResNet as a starting point for a new task.
* Researchers can explore the use of other imaging modalities such as CT scans, MRI scans, or ultrasound scans to develop a multi-modal model that can combine information from different imaging modalities to enhance the accuracy of COVID-19 detection.
* As the COVID-19 pandemic has affected people from various parts of the world with diverse genetic and environmental factors, developing a robust model that performs well on data from different populations and settings is crucial for the practical application of the model.

##### References

1. Bassi, P.R. and Attux, R., 2021. A deep convolutional neural network for COVID-19 detection using chest X-rays. Research on Biomedical Engineering, pp.1-10.
2. Hussain, E., Hasan, M., Rahman, M.A., Lee, I., Tamanna, T. and Parvez, M.Z., 2021. CoroDet: A deep learning based classification for COVID-19 detection using chest X-ray images. Chaos, Solitons & Fractals, 142, p.110495.
3. Rahman, T., Khandakar, A., Qiblawey, Y., Tahir, A., Kiranyaz, S., Kashem, S.B.A., Islam, M.T., Al Maadeed, S., Zughaier, S.M., Khan, M.S. and Chowdhury, M.E., 2021. Exploring the effect of image enhancement techniques on COVID-19 detection using chest X-ray images. Computers in biology and medicine, 132, p.104319.
4. ieee8023 (no date) IEEE8023/covid-chestxray-dataset: We are building an open database of COVID-19 cases with chest X-ray or CT images., GitHub. Available at: https://github.com/ieee8023/covid-chestxray-dataset (Accessed: March 7, 2023).
5. Saha, S. (2022) A comprehensive guide to Convolutional Neural Networks - the eli5 way, Medium. Towards Data Science. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accessed: March 2, 2023).
6. Hossamrizk (2022) Your first step in CNN, Kaggle. Kaggle. Available at: https://www.kaggle.com/code/hossamrizk/your-first-step-in-cnn#1.-What-is-the-CNN? (Accessed: March 3, 2023).
7. Alexturkmen (2022) Transfer learning - xception 100% val\_accuracy, Kaggle. Kaggle. Available at: https://www.kaggle.com/code/alexturkmen/transfer-learning-xception-100-val-accuracy (Accessed: March 3, 2023).
8. Gabrielmino (2021) CNN XRAY Covid Classification, Kaggle. Kaggle. Available at: https://www.kaggle.com/code/gabrielmino/cnn-xray-covid-classification (Accessed: March 3, 2023).
9. Nutan (2021) Image classification with Xception Model, Medium. Medium. Available at: https://medium.com/@nutanbhogendrasharma/image-classification-with-xception-model-e8094a9de4d2 (Accessed: March 5, 2023).

# Appendix

## Part 1 (Screenshots and Steps)

This experiment could be performed on any device using the below steps.

1. Clone repository - <https://github.com/siddesai80/covid_detection_ANN.git>

* This is a public repository.
* Folder [xray\_dataset\_covid19/](https://github.com/siddesai80/covid_detection_ANN/tree/main/xray_dataset_covid19) contains the dataset files which would be required to perform this experiment.
* [ANN\_Covid\_Detection.ipynb](https://github.com/siddesai80/covid_detection_ANN/blob/main/ANN_Covid_Detection.ipynb) contains the main code of the experiment.
* Folder [Plots/](https://github.com/siddesai80/covid_detection_ANN/tree/main/Plots) contain Training and Validation plots of losses and accuracy for both models.
* Folder [X-ray/](https://github.com/siddesai80/covid_detection_ANN/tree/main/X-ray%20Images) images contain samples of X-ray raw images from the dataset and the images which are pre-processed.

1. You can execute this file using a cloud platform like [Google Colab](https://colab.research.google.com/).
2. Import the file into Google Colab by clicking File -> Open Notebook and selecting the [ANN\_Covid\_Detection.ipynb](https://github.com/siddesai80/covid_detection_ANN/blob/main/ANN_Covid_Detection.ipynb) file from your local system.
3. The input files needed would also be cloned in the GIT repo so copy the path to those files and replace them in the [ANN\_Covid\_Detection.ipynb](https://github.com/siddesai80/covid_detection_ANN/blob/main/ANN_Covid_Detection.ipynb) file under Pre-processing data -> Setting train and test directories section.
4. Once all the changes are done Run the Notebook and observe the results.

A screenshot of a computer

Description automatically generated

Figure : Screenshot to show that the code was developed and executed locally on a personal laptop

Text

Description automatically generated

Figure : Epochs for the custom CNN model

Text

Description automatically generated

Figure : Epochs for the Xception model

## Part 2 (Code)

# -\*- coding: utf-8 -\*-

"""ANN\_Covid\_Detection.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1yi8bQ3dbyW2JIJFHHr0PgUJOTJNVTJa9

### Importing important libraries

"""

**import** **os**

**import** **cv2**

**import** **random**

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **seaborn** **as** **sns**

**import** **tensorflow** **as** **tf**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib.image** **as** **mpimg**

**from** **tensorflow** **import** keras

**from** **tensorflow.keras** **import** layers

**from** **tensorflow.keras.preprocessing** **import** image

**from** **tensorflow.keras.preprocessing.image** **import** ImageDataGenerator

**from** **tensorflow.keras.callbacks** **import** EarlyStopping,ReduceLROnPlateau,ModelCheckpoint

**from** **tensorflow.keras.models** **import** Sequential

**from** **tensorflow.keras.utils** **import** image\_dataset\_from\_directory

**from** **tensorflow.keras.layers** **import** Input,Dense,Flatten,MaxPooling2D,Conv2D,Dropout,Activation,BatchNormalization,SimpleRNN

**from** **tensorflow.keras.layers.experimental** **import** preprocessing

**from** **tensorflow.keras.layers.experimental.preprocessing** **import** RandomFlip, RandomRotation, RandomZoom, Rescaling

**from** **sklearn.metrics** **import** confusion\_matrix,classification\_report,accuracy\_score

# disabling warnings

**import** **logging**

logging.getLogger('tensorflow').disabled = True

**print**('All packages are Imported Successfully.')

"""### Pre-processing data"""

# Setting train and test directories

train\_dir='/content/drive/MyDrive/Datasets/ANN/xray\_dataset\_covid19/train'

test\_dir='/content/drive/MyDrive/Datasets/ANN/xray\_dataset\_covid19/test'

# Printing the sub-directories

**print**(os.listdir(train\_dir))

**print**(os.listdir(test\_dir))

# Displaying random images from the training folder

img\_folder='/content/drive/MyDrive/Datasets/ANN/xray\_dataset\_covid19/train'

plt.figure(figsize=(**20**,**20**))

**for** i **in** range(**6**):

class\_=random.choice(os.listdir(img\_folder))

class\_path=os.path.join(img\_folder, class\_)

file=random.choice(os.listdir(class\_path))

image\_path=os.path.join(class\_path,file)

**print**(image\_path)

img=mpimg.imread(image\_path)

ax=plt.subplot(**1**,**6**,(i+**1**))

plt.imshow(img)

ax.title.set\_text(class\_)

# Creating a function to standardise all the image files

**def** **create\_dataset**(img\_folders\_path, img\_width, img\_height):

images = []

labels = []

num\_images = **0**

**for** dirpath, \_, filenames **in** os.walk(img\_folders\_path):

**for** filename **in** filenames:

img\_path = os.path.join(dirpath, filename)

image = cv2.cvtColor(cv2.imread(img\_path), cv2.COLOR\_BGR2RGB)

image = cv2.resize(image, (img\_width, img\_height))

image = image.astype('float32') / **255**

images.append(image)

class\_name = os.path.basename(dirpath)

labels.append(class\_name)

num\_images += **1**

**return** np.array(images), np.array(labels), num\_images

# Converting the images by calling the function

IMG\_WIDTH=**224**

IMG\_HEIGHT=**224**

train\_img,train\_target,num\_img=create\_dataset(train\_dir,IMG\_WIDTH,IMG\_HEIGHT)

test\_img,test\_target,num\_test\_img=create\_dataset(test\_dir,IMG\_WIDTH,IMG\_HEIGHT)

# Displaying the new images after pre-processing

plt.figure(figsize=(**20**,**20**))

**for** i **in** range(**6**):

random\_num = random.randint(**0**,num\_img)

ax=plt.subplot(**1**,**6**,(i+**1**))

plt.imshow(train\_img[random\_num])

ax.title.set\_text(train\_target[random\_num])

# Convert text labels to numeric codes

target\_dict={k: v **for** v, k **in** enumerate(np.unique(train\_target))}

**print**(target\_dict)

train\_target= [target\_dict[train\_target[i]] **for** i **in** range(len(train\_target))]

train\_target=np.array(train\_target)

train\_img=np.array(train\_img)

test\_target= [target\_dict[test\_target[i]] **for** i **in** range(len(test\_target))]

test\_target=np.array(test\_target)

test\_img=np.array(test\_img)

"""# CNN Model

## Defining and Fitting Model

"""

# Early stopping for preventing overfitting

early\_stopping = EarlyStopping(monitor='loss', restore\_best\_weights=False, patience=**10**)

# Define a Sequential model

model = Sequential()

# Add the first convolutional layer

model.add(Conv2D(**32**, kernel\_size=(**3**,**3**), activation="relu",input\_shape=(**224**,**224**,**3**)))

# Add a second convolutional layer with 64 filters

model.add(Conv2D(**64**, kernel\_size=(**3**,**3**), activation="relu"))

# Add a max pooling layer

model.add(MaxPooling2D(pool\_size=(**2**,**2**)))

# Add dropout regularization

model.add(Dropout(**0.25**))

# Repeating the above steps to make a deeper network.

model.add(Conv2D(**128**, kernel\_size=(**3**,**3**), activation="relu"))

model.add(MaxPooling2D(pool\_size=(**2**,**2**)))

model.add(Dropout(**0.25**))

# Flatten the output from the convolutional layers

model.add(Flatten())

# Add a dense layer with 64 units and ReLU activation

model.add(Dense(**64**, activation = "relu"))

# Add more dropout regularization

model.add(Dropout(**0.5**))

# Create an output sigmoid function

model.add(Dense(**1**, activation="sigmoid"))

#Compile the model: binary\_crossentropy because this is a binary classification problem; adam as the optimizer; the metric that we want to monitor is accuracy.

model.compile(loss="binary\_crossentropy", optimizer="adam",metrics = ["accuracy"])

# Printe the model architecture to take a look at the number of parameters that the model will learn.

model.summary()

# Fit the model

model.fit(train\_img, train\_target,

validation\_split=**0.10**,

epochs=**20**,

batch\_size=**32**,

shuffle=True,

callbacks=[early\_stopping]

)

"""## Plotting loss function and accuracy"""

# Storing training and validation metrics

losses = pd.DataFrame(model.history.history)

losses.head()

# Plotting Training and Validation Losses

losses[['loss','val\_loss']].plot()

plt.title('Training and Validation Losses')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Training Loss', 'Validation Loss'])

plt.show()

# Plotting Training and Validation Accuracy

losses[['accuracy', 'val\_accuracy']].plot()

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Training Accuracy', 'Validation Accuracy'])

plt.show()

"""## Predicting the results"""

# Predictions on test set

pred = model.predict(test\_img,batch\_size=**32**)

label = [int(p>=**0.5**) **for** p **in** pred]

# Printing Accuracy Score

**print** ('Accuracy Score : ', accuracy\_score(label, test\_target), '**\n**')

# Printing Classification report

**print** ('Classification Report :**\n\n**' ,classification\_report(label, test\_target))

"""# XCeption Model

## Defining and Fitting Model

"""

# Creating train and test datasets

xception\_train\_data = image\_dataset\_from\_directory(directory = train\_dir,

image\_size = (**224**, **224**),

label\_mode = "binary",

batch\_size = **32**,

seed = **42**)

xception\_test\_data = image\_dataset\_from\_directory(directory = test\_dir,

image\_size = (**224**, **224**),

label\_mode = "binary",

batch\_size = **32**,

seed = **42**)

# Create a data augmentation pipeline with horizontal flipping, rotations, and zooms

data\_augmentation\_pipeline = keras.Sequential([

preprocessing.RandomFlip(mode="horizontal"),

preprocessing.RandomRotation(factor=**0.2**),

preprocessing.RandomZoom(height\_factor=**0.2**, width\_factor=**0.2**),

preprocessing.RandomHeight(factor=**0.2**),

preprocessing.RandomWidth(factor=**0.2**)

], name="data\_augmentation")

# Create a normalization pipeline

normalization\_pipeline = keras.Sequential([

preprocessing.Rescaling(scale=**1.**/**255.**)

], name="normalization")

# Define input shape

input\_shape = (**224**, **224**, **3**)

# Saving best model while monitoring accuracy

model\_checkpoint = ModelCheckpoint('best\_mod', save\_best\_only=True, verbose = **1**)

# Define Xception base model

xception\_base = tf.keras.applications.xception.Xception(include\_top=False)

# Create input layer

input\_layer = layers.Input(shape = input\_shape, name = "Input\_Layer")

# Add data augmentation as a layer

x = data\_augmentation\_pipeline(input\_layer)

# Add normalization layer

x = normalization\_pipeline(x)

# Pass inputs through the Xception base model

x = xception\_base(x, training = False)

# Add global average pooling layer to condense output of the base model

x = tf.keras.layers.GlobalAveragePooling2D(name = "Pooling\_Layer")(x)

# Add a dense output layer with sigmoid activation function

output\_layer = layers.Dense(**1**, activation = "sigmoid", name = "Output\_Layer")(x)

# Create the final model by specifying inputs and outputs

xception\_model = keras.Model(input\_layer, output\_layer)

# Compile the model

xception\_model.compile(

loss="binary\_crossentropy",

optimizer=tf.keras.optimizers.Adam(learning\_rate=**0.001**),

metrics=["accuracy"]

)

# Display the model summary

xception\_model.summary()

# Fit the model

xception\_model.fit(xception\_train\_data,

epochs=**20**,

steps\_per\_epoch=len(xception\_train\_data),

validation\_data=xception\_test\_data,

validation\_steps=len(xception\_test\_data),

callbacks=[early\_stopping,model\_checkpoint])

# Number of layers in our base model, and their trainablity status

**for** i, layer **in** enumerate(xception\_model.layers):

**print**(i, layer.name, layer.trainable)

"""## Plotting loss function and accuracy"""

# Storing training and validation metrics

losses = pd.DataFrame(xception\_model.history.history)

losses.head()

# Plotting Training and Validation Losses

losses[['loss','val\_loss']].plot()

plt.title('Training and Validation Losses')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Training Loss', 'Validation Loss'])

plt.show()

# Plotting Training and Validation Accuracy

losses[['accuracy', 'val\_accuracy']].plot()

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Training Accuracy', 'Validation Accuracy'])

plt.show()

"""## Evaluating the model"""

# Loading the model we saved during training using checkpoint and evaluating the test data with it

final\_model = tf.keras.models.load\_model("best\_mod")

final\_model.evaluate(xception\_test\_data)