A Study of Detecting COVID-19 Severity using Lung Ultrasonography

- Abdul Wahed
- Amar Sayani
- Bhargav Nishanth Katikey

Problem Statement

- A critical need for precise and effective diagnostic techniques has arisen as a result of the COVID-19 pandemic.
- Although current diagnostic techniques like RT-PCR and chest CT scans are reliable, they can also be costly, time-consuming, and requires a lot of sources.
- Recent study research suggests that lung ultrasonography can be an effective method for identifying COVID-19 pneumonia. But further research and evaluation must be done to confirm the efficacy of this approach.
- Therefore, the purpose of this study is to find out the efficiency of using lung ultrasonography images with deep learning techniques to identify COVID-19 infection.

Best Approach

- Acquiring and creating a dataset with huge collection of COVID-19 positive and negative patient's lung ultrasound images. Images from multiple perspectives and at various stages of an infection should be included in the dataset.
- Using this image data, develop and train a deep learning algorithms to recognize and classify COVID-19 pneumonia in lung ultrasound pictures. The objective of this approach should be to work efficiently and correctly identify COVID-19 pneumonia.
- Validate the efficacy of the algorithm on the test data to make sure it can be applied to the new data.
- Develop a user-friendly software application that uses this algorithm and can be applied in clinical trials.
- Carry out a clinical trial to evaluate the accuracy of an algorithm devised for identifying COVID-19 pneumonia.
- This could help the early detection and treatment of the illness.

Challenges

- Limited availability of high-quality and diverse lung ultrasonography images of COVID-19 patients.
- It was quite challenging to gather a large and diverse set of lung ultrasound pictures.
- Currently there is no generally recognized and accepted procedures for lung ultrasonography in Covid affected patients , and this can result in discrepancies in picture quality and interpretation, which can influence the algorithm's accuracy.
- Another challenge is with developing a deep learning algorithm that can accurately recognize COVID-19 pneumonia in pictures of lung ultrasonography because there is limited amount of image data with discrepancies in image quality.

Introduction

Introduction:

- COVID-19 is a disease, caused by a virus (SARS-CoV-2).
- The World Health Organization declared a significant international public health emergency, and the condition was considered a health emergency
- Lung infections can range from just a common cold to a serious illness.
- One of the basic methods for diagnosing COVID-19 is the use of computed tomography (CT) and X-rays.
- Medical health standards recommend chest imaging as a rapid and effective procedure, and it has been cited in multiple studies as the initial tool in screening.
- However, diagnosis by visual interpretation can be challenging and time-consuming for doctors.

- The use of deep learning for the interpretation of radiological images has been the subject of multiple research.
- Using Deep learning approach, this study offers a technique for identifying COVID and non-COVID classes.
- For automatic COVID-19 categorization, different deep learning methods by extracting their features were compared
- The best method for identifying it among the most important deep learning algorithms is the CNN architecture. Deep learning algorithms, particularly CNN, have attracted a lot of attention for data processing.
- In this study , various deep learning architectures, including CNN, MobileNetV2, DenseNet121, VGG16, and NASNet, are employed to create a multiclass classification system for distinguishing COVID-19 infection and non-covid using LUS imagery.
- The method’s accuracy was tested using the COVID-19 dataset of chest X-ray and CT images.

Data Sources

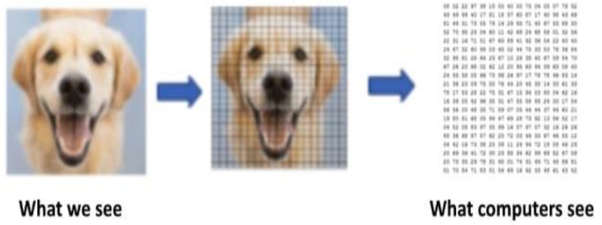
Data Sources

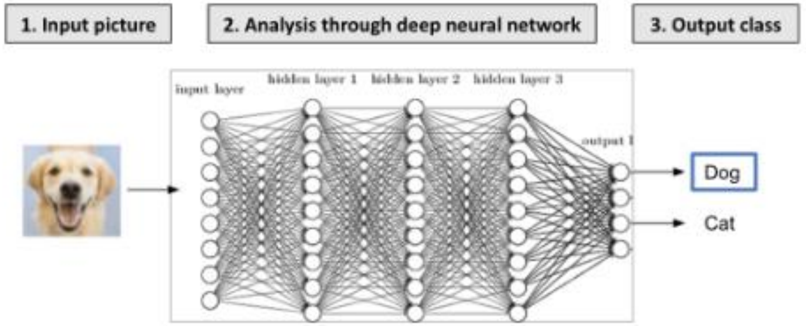
CT scans of the lungs are included in the dataset collection. A CT scan makes use of cutting-edge X-ray technology to precisely assess internal organs. Several publicly accessible sources, including the Italian Society of Radiology, NCBI, Eurorad, and Radiopedia, were used to gather the images for this dataset. and had 4335 images in it. The data has three classes: COVID positive and COVID negative and Normal.

Image Processing: How Do Image Classifiers Work?

Image Classification

- The first important thing to understand is what a computer sees when looking at an image: not actual visual content but an array of numeric pixel values.
- Image classifiers take an image as an input and gives output either a class or a probability that the image is a particular class.
- In simple terms, it gives Output of a class, for example "dog" or "cat“
- If you give a set of images each of which either depicts a cat or a dog. Instead of labeling the pictures all on your own, you want to use an algorithm to do the work for you: It "looks" at the whole picture and outputs probabilities for each of the classes it was trained on.





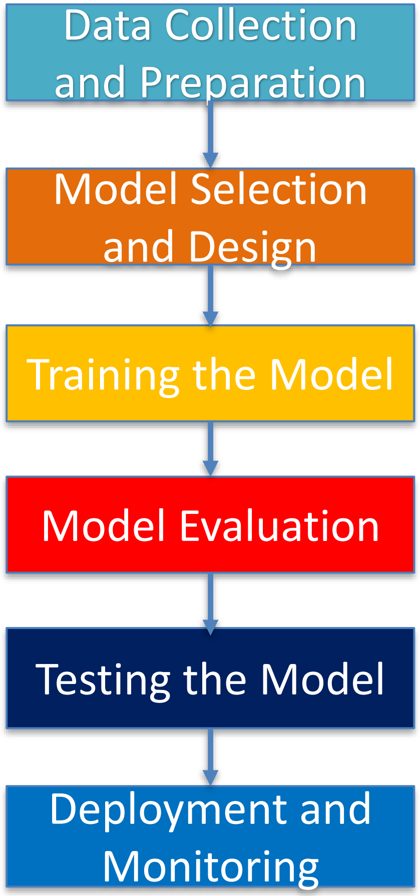
- Keras and TensorFlow are popular deep learning frameworks used for image classification tasks because they offer a wide range of powerful tools and pre-built models for developing and training neural networks.

- Keras provides a high-level API that allows developers to easily build and train deep learning models for image classification, with pre-built layers and models for common architectures like convolutional neural networks (CNNs).

- TensorFlow, on the other hand, is a lower-level framework that offers more flexibility for designing and customizing models.

- Together, Keras and TensorFlow provide a powerful combination of ease-of-use and flexibility for developing image classification models.

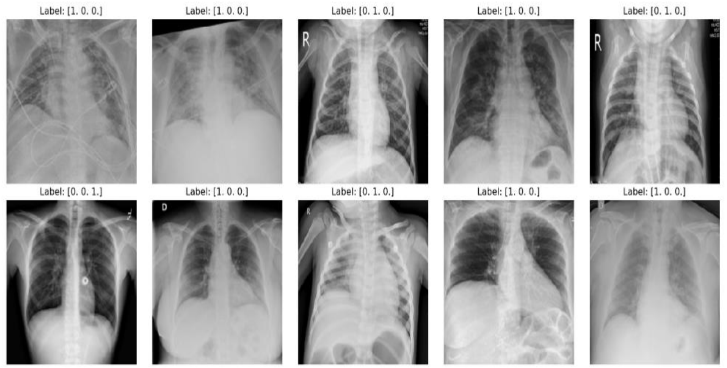
Methodology


Data Collection and Preparation
Model Selection and Design
Training the Model
Model Evaluation
Testing the Model
Deployment and Monitoring

Data Pre-Processing

Data Pre-Processing

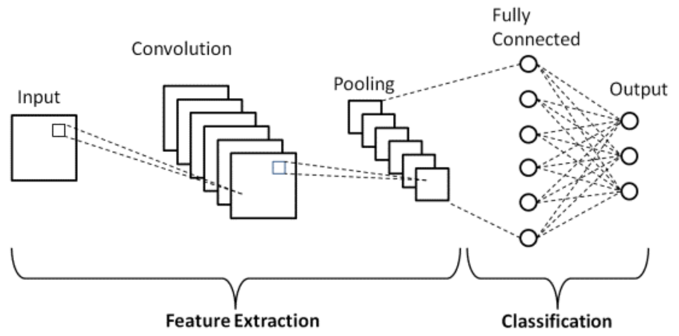
- The ImageDataGenerator function is used to create a generator for the training, and validation data, which rescales the pixel values of the images.
- The validation_split argument specifies that 20% of the data should be used for validation.
- The flow_from_directory method is then used to create generators for the training, and validation data. The method takes as input the path to the directory containing the images, the target size of the images (224x224 pixels in this case), the batch size (32 images at a time in this case), the class mode (categorical, indicating that the data has multiple classes), and the subset (training for the training data and validation for the validation data).
- Similarly, a generator is created for the test data using the test_datagen object created from ImageDataGenerator class. The flow_from_directory method is again used with the path to the directory containing the test images, the target size of the images, the batch size, and the class mode.



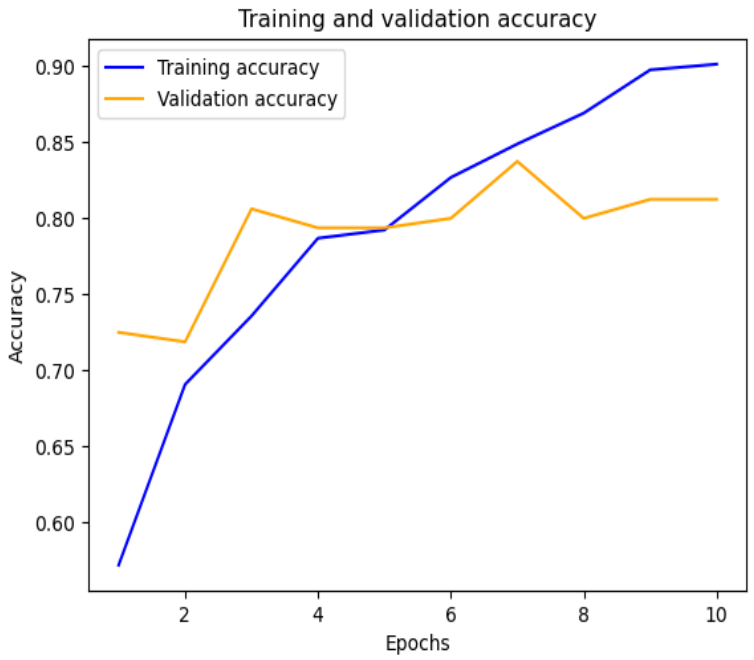
Model Selection and Design

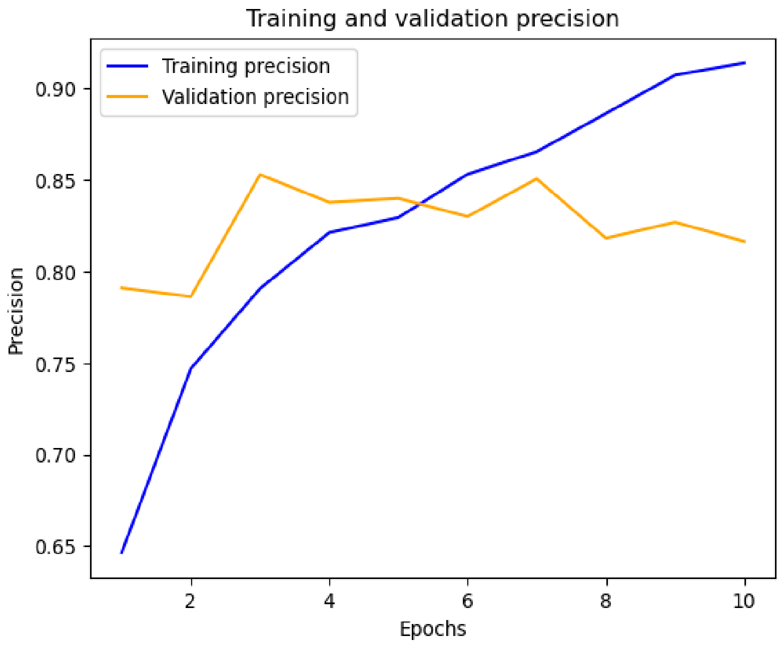
Basic CNN Architecture

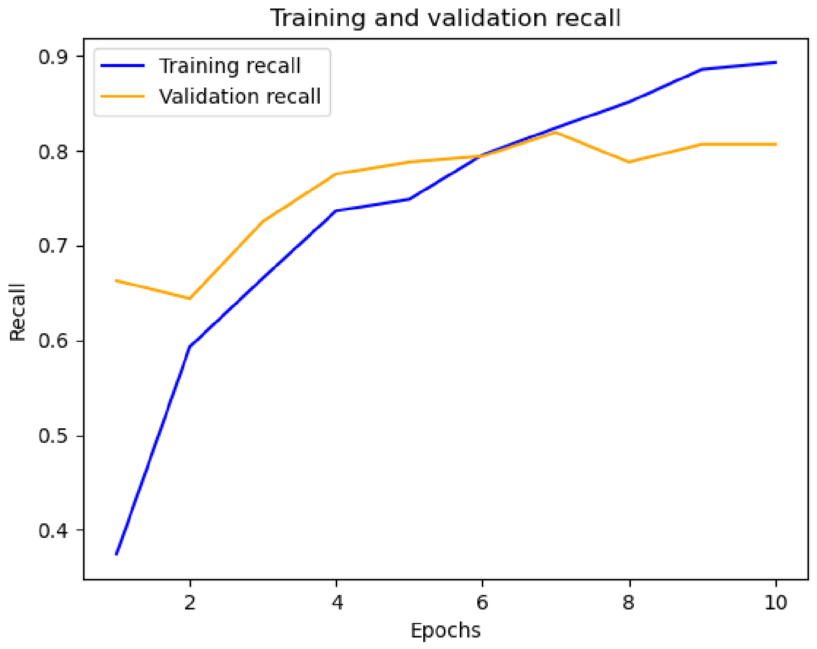
- CNNs are a class of Deep Neural Networks that can recognize and classify features from images and are widely used for analyzing visual images.
- Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.
- CNN has high accuracy, and because of the same, it is useful in image recognition. Image recognition has a wide range of uses in various industries such as medical image analysis.
- A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction.
- The network of feature extraction consists of many pairs of convolutional or pooling layers.
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.
- This CNN model of feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarizes the existing features contained in an original set of features. There are many CNN layers as shown in the CNN architecture diagram.



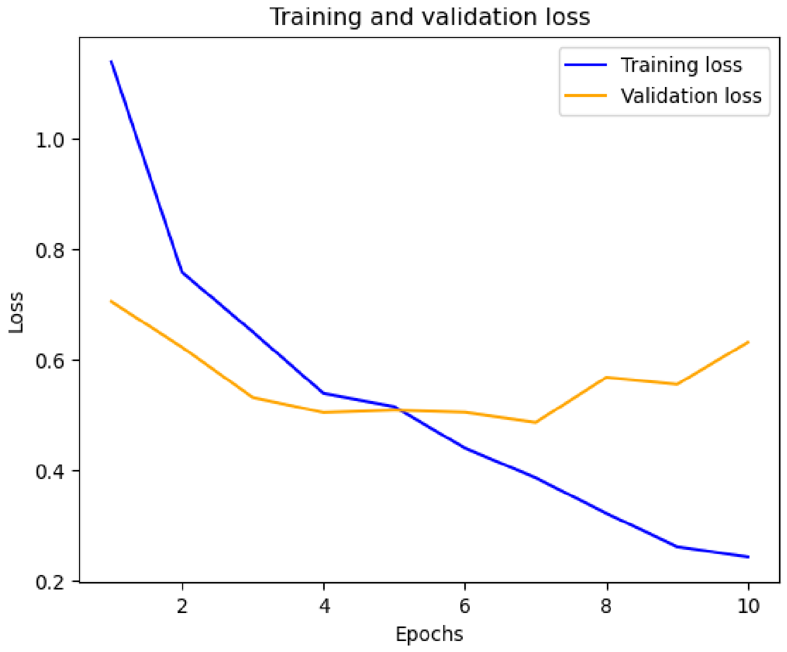
Evaluating the model performance



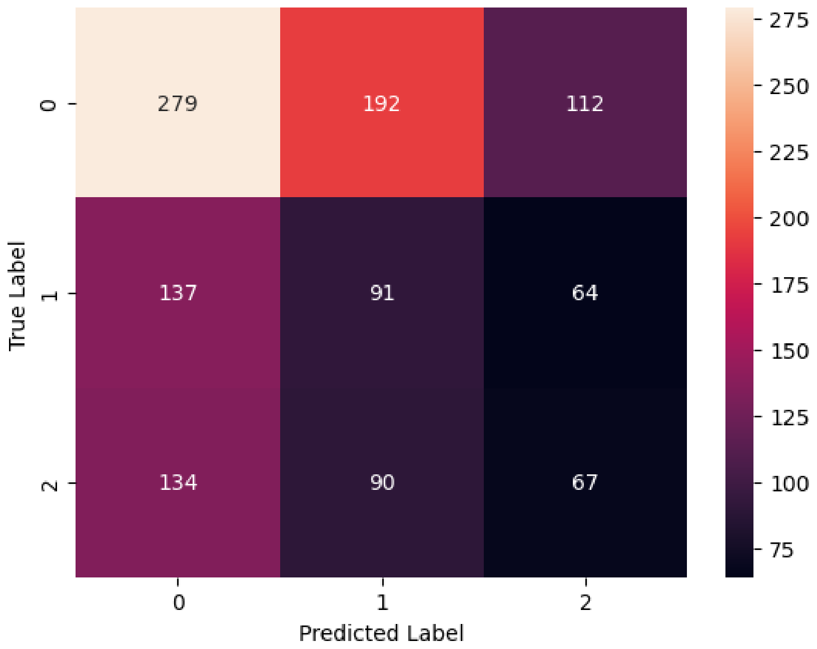




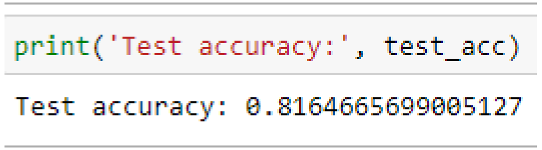
Training and validation recall
Training recall
Validation recall
Recall
Epochs
0.4
0.5
0.6
0.7
0.8
0.9
2
4
6
8
10



Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10

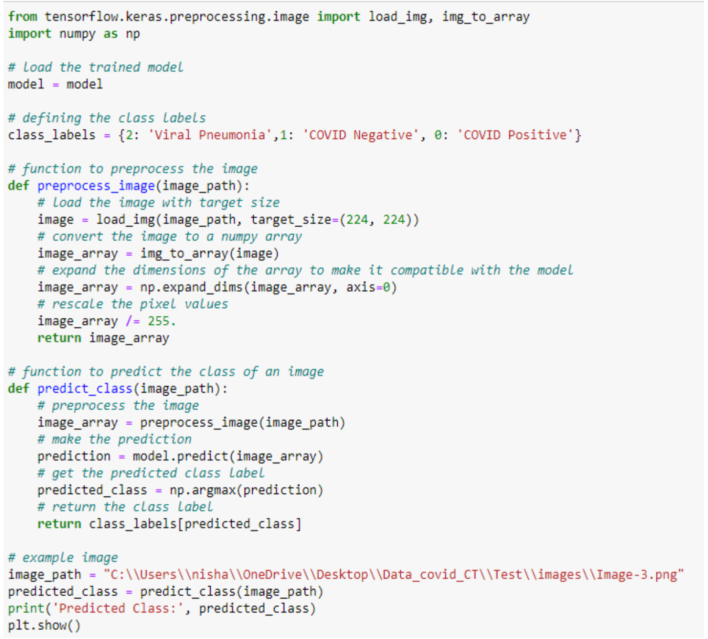


- **Accuracy of the CNN Model**


```
print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.8164665699005127
```

Deployment and Monitoring

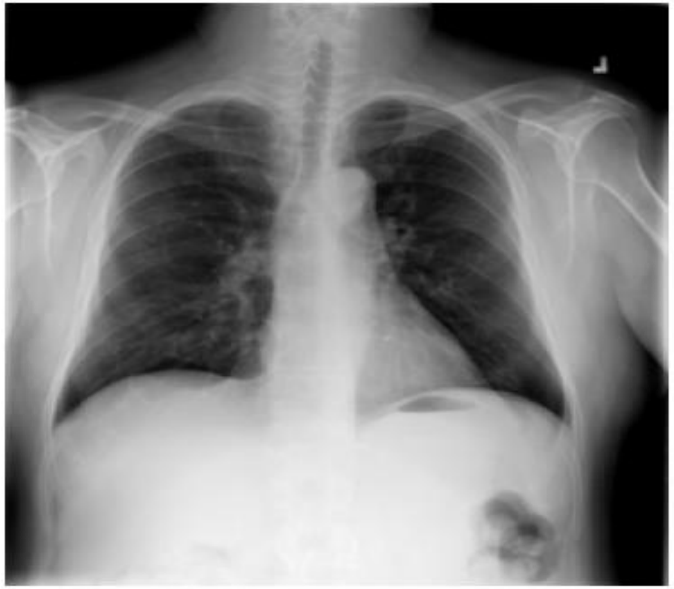


```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# load the trained model
model = model

# defining the class labels
class_labels = {2: 'Viral Pneumonia',1: 'COVID Negative', 0: 'COVID Positive'}

# function to preprocess the image
def preprocess_image(image_path):
    # load the image with target size
    image = load_img(image_path, target_size=(224, 224))
    # convert the image to a numpy array
    image_array = img_to_array(image)
    # expand the dimensions of the array to make it compatible with the model
    image_array = np.expand_dims(image_array, axis=0)
    # rescale the pixel values
    image_array /= 255.
    return image_array

# function to predict the class of an image
def predict_class(image_path):
    # preprocess the image
    image_array = preprocess_image(image_path)
    # make the prediction
    prediction = model.predict(image_array)
    # get the predicted class label
    predicted_class = np.argmax(prediction)
    # return the class label
    return class_labels[predicted_class]

# example image
image_path = "C:\\Users\\nisha\\OneDrive\\Desktop\\Data_covid_CT\\Test\\images\\Image-3.png"
predicted_class = predict_class(image_path)
print('Predicted Class:', predicted_class)
plt.show()
```

```
Predicted Class: Viral Pneumonia
```

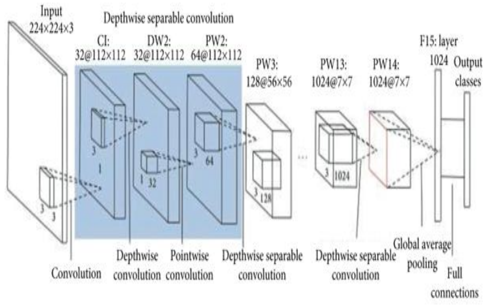
```
1/1 [==============================] - 0s 88ms/step
```

Viral Pneumonia

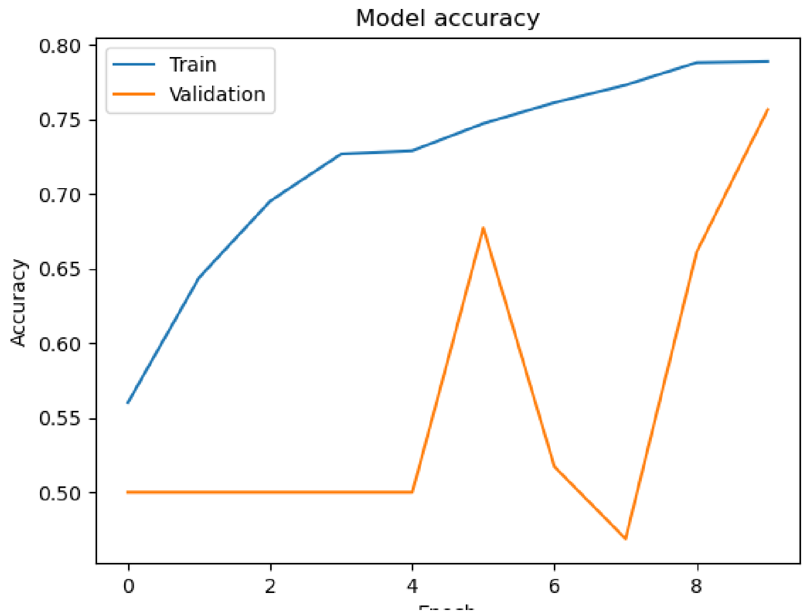


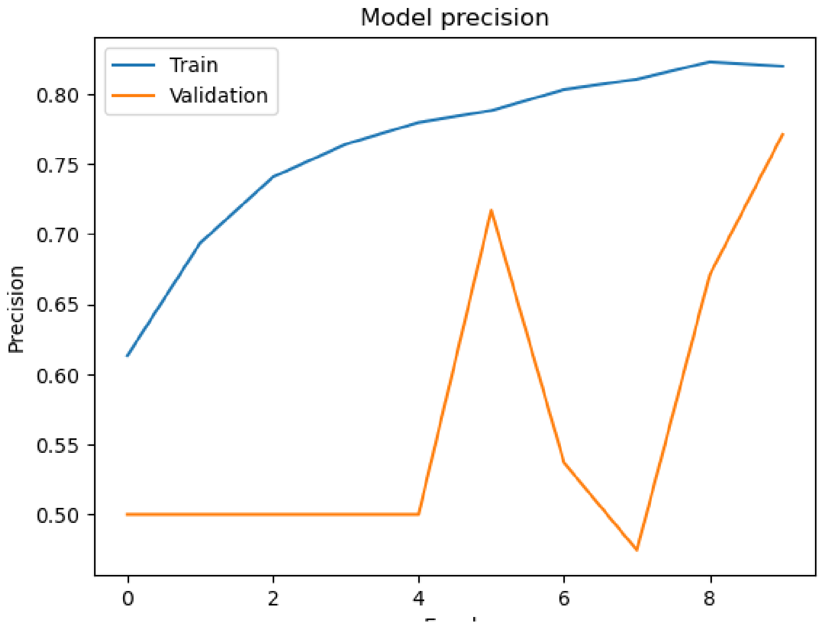
MobileNetV2

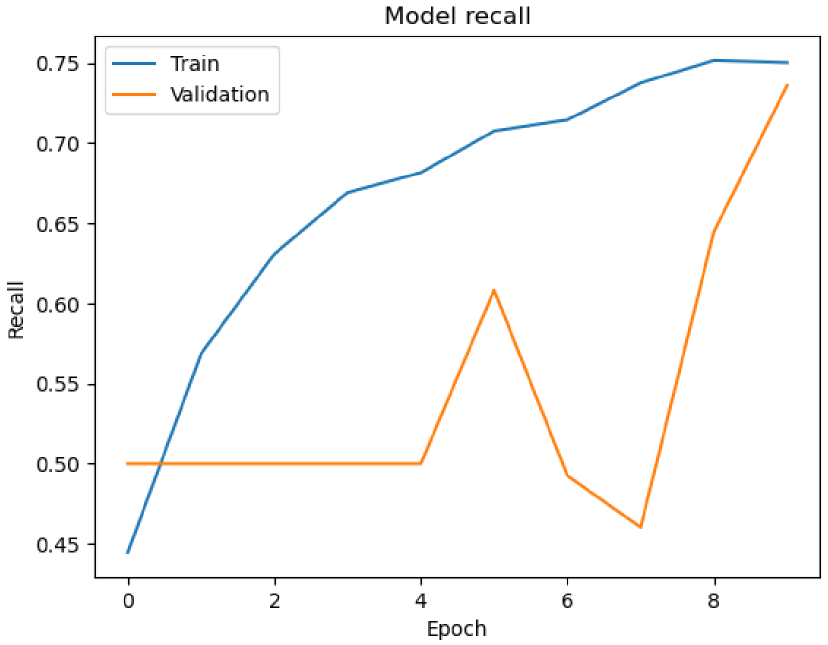
- MobileNet works by using depthwise separable convolutions, which are a type of convolutional layer that separates the spatial and channel-wise convolution operations.
- The architecture consists of 17 bottleneck layers with different output channels and strides.
- Each bottleneck layer has three parts: an expansion block, a depthwise block, and a projection block.
- The depthwise block conducts a depthwise convolution with a stride, the projection block reduces the number of channels to the required output channels, and the expansion block increases the number of input channels.

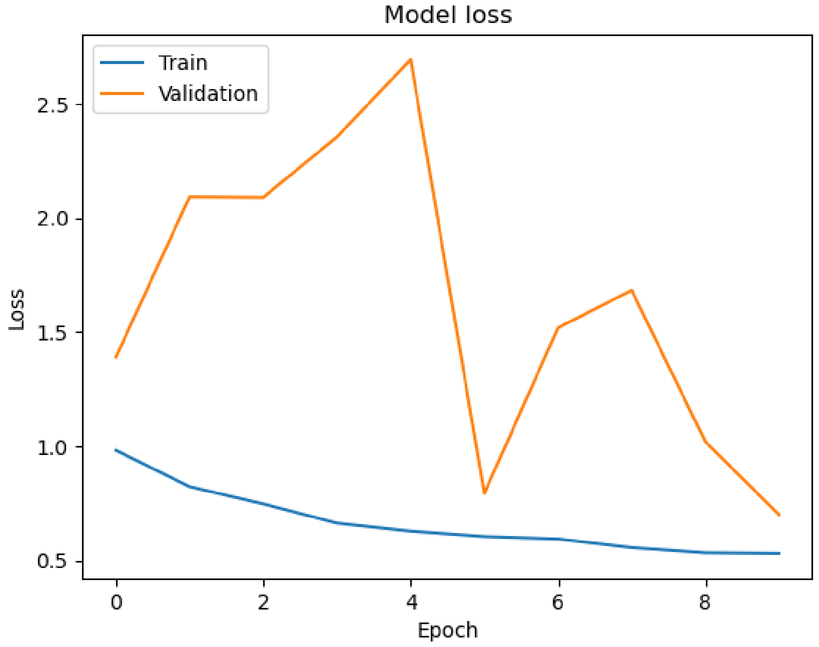


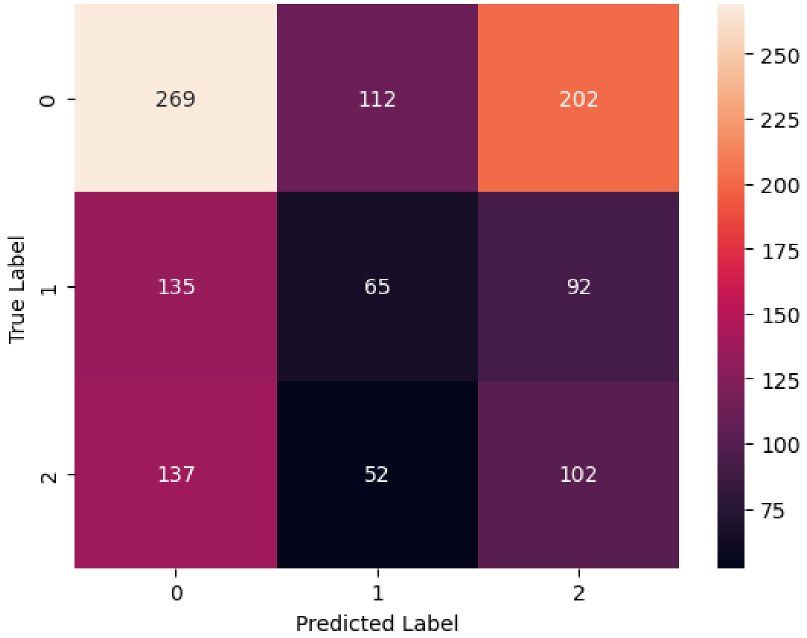
Evaluating the model performance



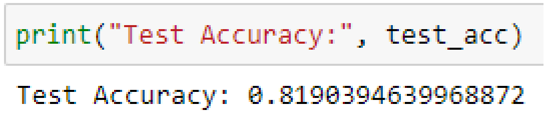



Model recall
Train
Validation
Recall
Epoch
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0
2
4
6
8


Model loss
Train
Validation
Loss
Epoch
0.5
1.0
1.5
2.0
2.5
0
2
4
6
8



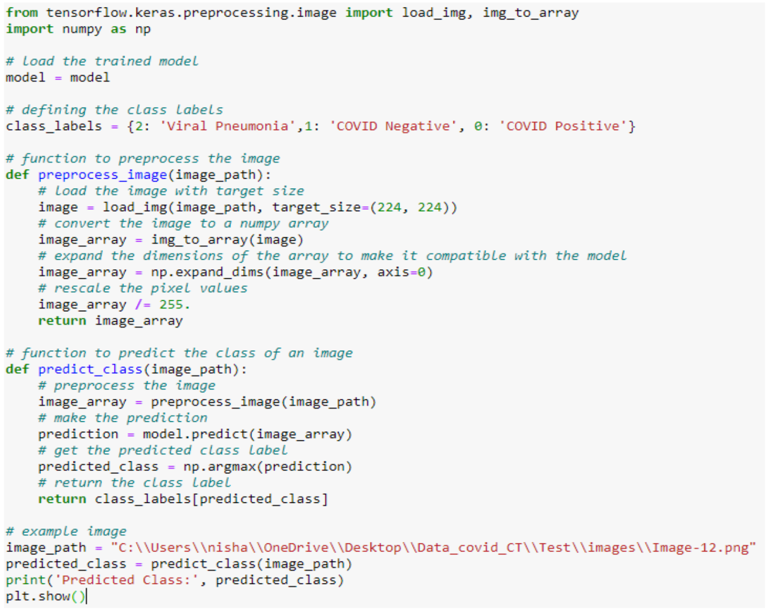
- **Printing the test accuracy**



```
print("Test Accuracy:", test_acc)
```

```
Test Accuracy: 0.8190394639968872
```

Deployment and Monitoring

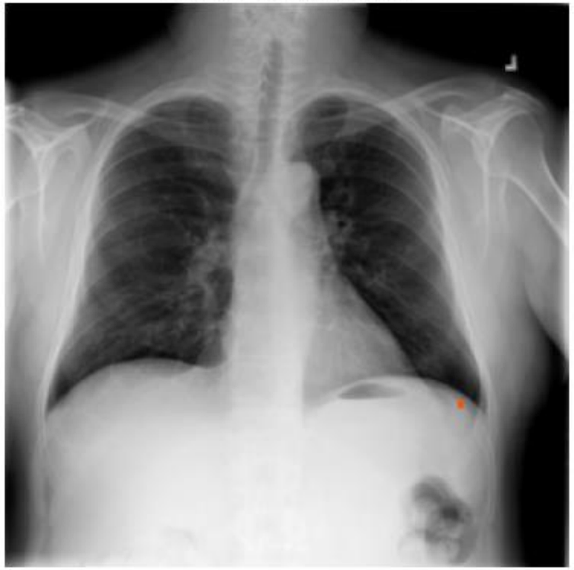


```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# load the trained model
model = model

# defining the class labels
class_labels = {2: 'Viral Pneumonia',1: 'COVID Negative', 0: 'COVID Positive'}

# function to preprocess the image
def preprocess_image(image_path):
    # load the image with target size
    image = load_img(image_path, target_size=(224, 224))
    # convert the image to a numpy array
    image_array = img_to_array(image)
    # expand the dimensions of the array to make it compatible with the model
    image_array = np.expand_dims(image_array, axis=0)
    # rescale the pixel values
    image_array /= 255.
    return image_array

# function to predict the class of an image
def predict_class(image_path):
    # preprocess the image
    image_array = preprocess_image(image_path)
    # make the prediction
    prediction = model.predict(image_array)
    # get the predicted class label
    predicted_class = np.argmax(prediction)
    # return the class label
    return class_labels[predicted_class]

# example image
image_path = "C:\\Users\\nisha\\OneDrive\\Desktop\\Data_covid_CT\\Test\\images\\Image-12.png"
predicted_class = predict_class(image_path)
print('Predicted Class:', predicted_class)
plt.show()
```

```
Predicted Class: COVID Negative
```

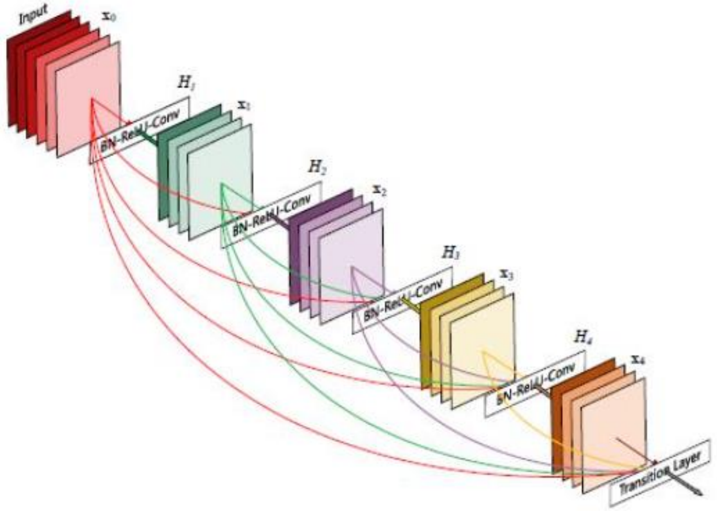
```
1/1 [==============================] - 1s 662ms/step
```

COVID-19 Negative

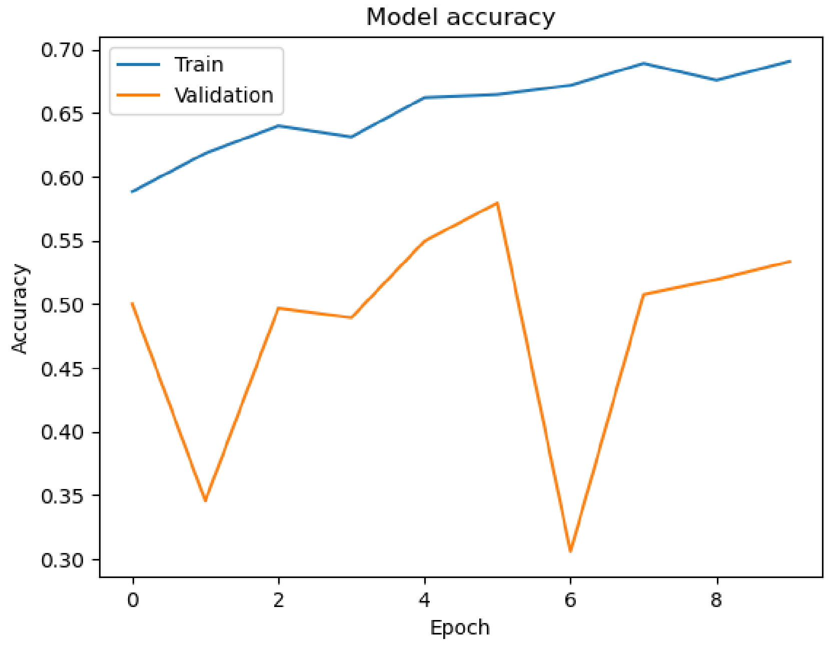


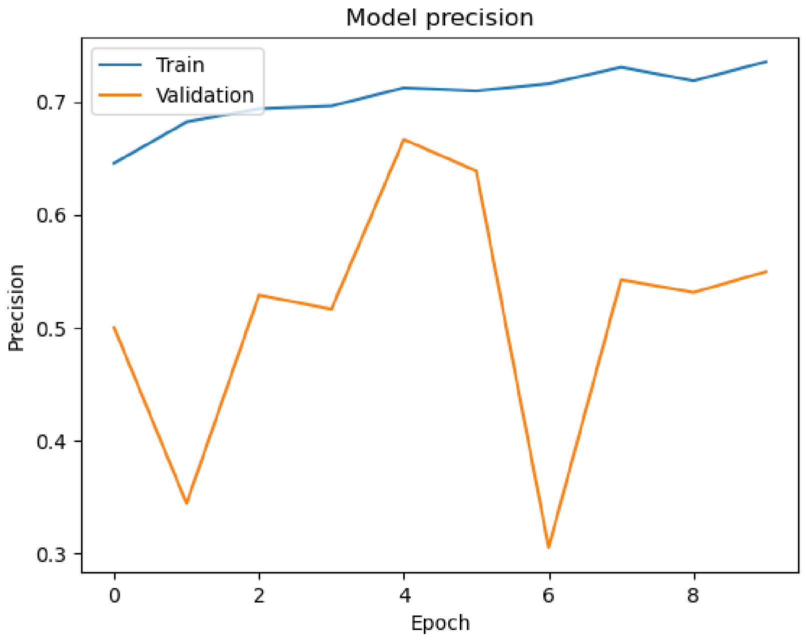
DenseNet121

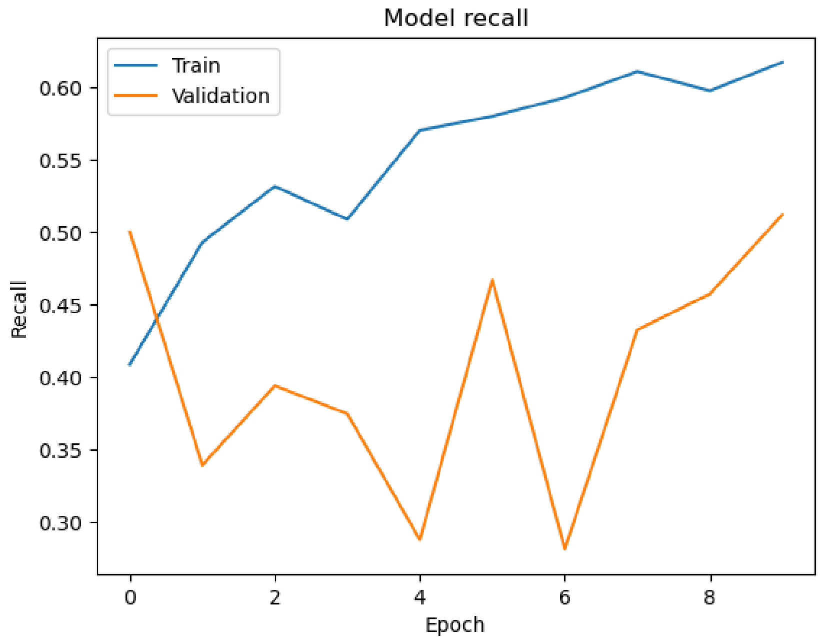
- DenseNet is a type of convolutional neural network (CNN) that is designed to improve the flow of information between layers and combat the vanishing gradient problem that can occur in deep neural networks.
- In traditional CNNs, each layer receives the output from the previous layer as its input.
- However, in DenseNet, each layer receives the output of all previous layers as its input. This creates a more direct path for information flow and allows the network to maintain strong gradients throughout training, leading to better convergence and performance.
- Additionally, DenseNet has a relatively small number of parameters compared to other state-of-the-art CNN architectures, making it efficient to train and deploy.

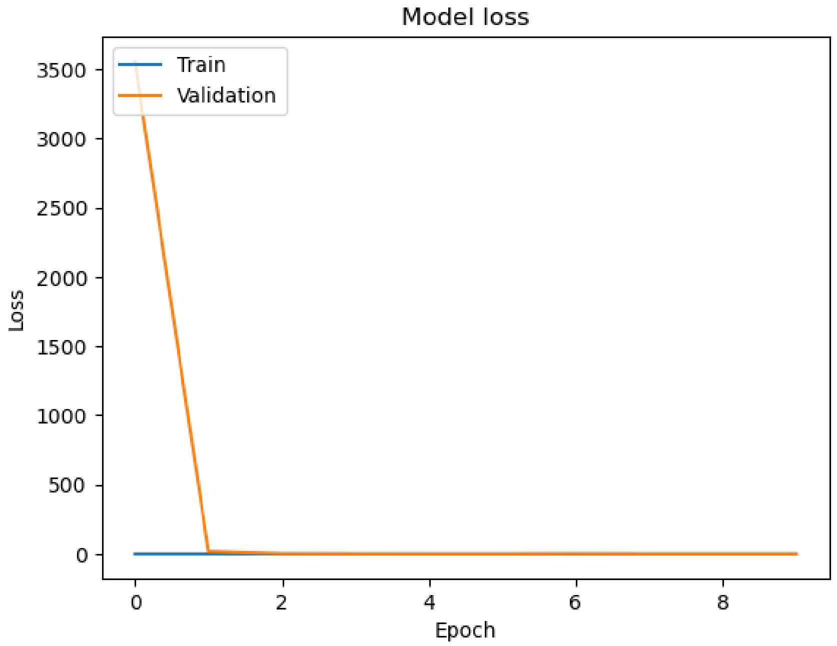


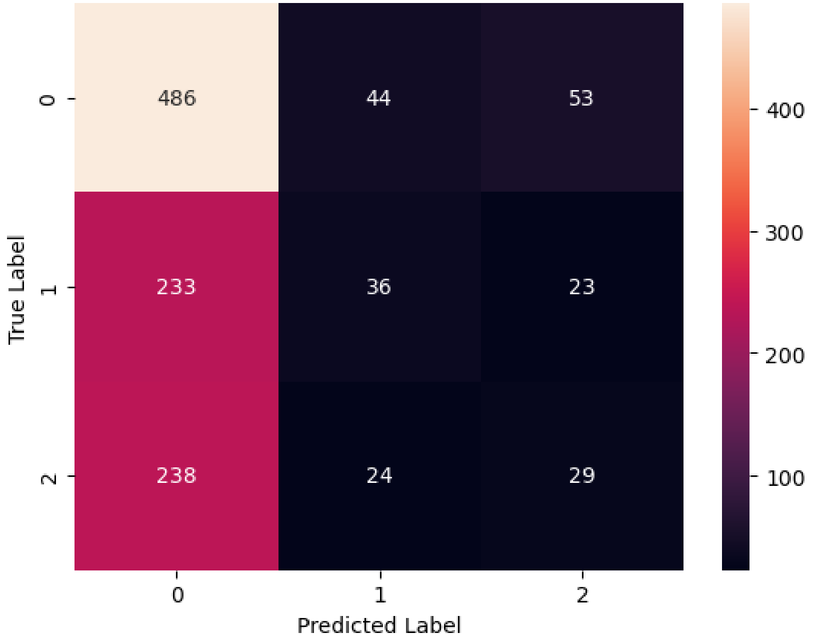
Evaluating the model performance

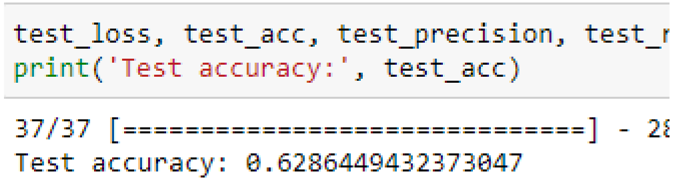





Model recall
Train
Validation
Recall
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
2
4
6
8
Epoch


Model loss
Train
Validation
Loss
3500
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
Epoch

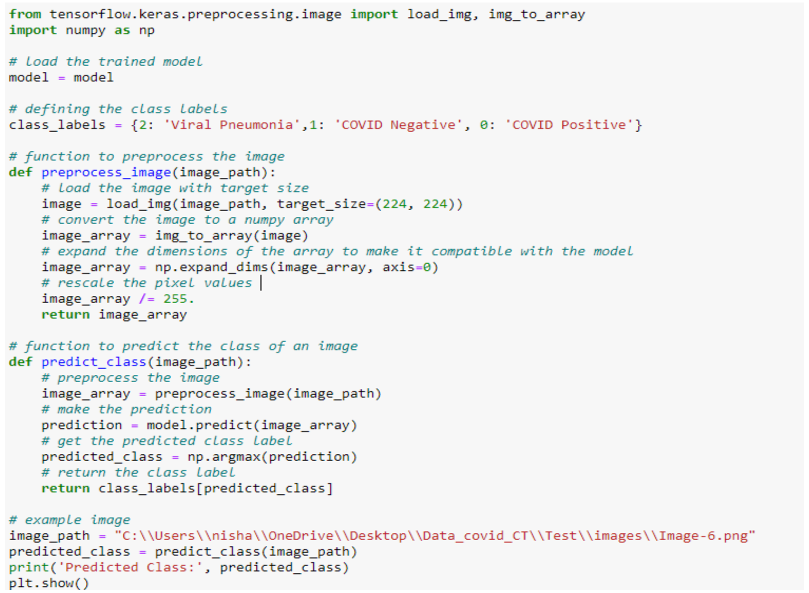




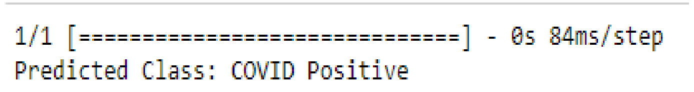
```
test_loss, test_acc, test_precision, test_r
print('Test accuracy:', test_acc)
```

```
37/37 [==============================] - 28
Test accuracy: 0.6286449432373047
```

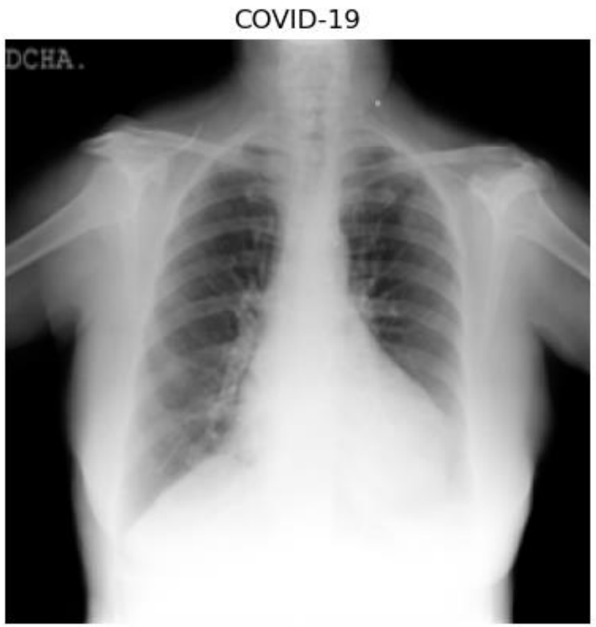
Deployment and Monitoring



```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# load the trained model
model = model

# defining the class labels
class_labels = {2: 'Viral Pneumonia',1: 'COVID Negative', 0: 'COVID Positive'}

# function to preprocess the image
def preprocess_image(image_path):
    # load the image with target size
    image = load_img(image_path, target_size=(224, 224))
    # convert the image to a numpy array
    image_array = img_to_array(image)
    # expand the dimensions of the array to make it compatible with the model
    image_array = np.expand_dims(image_array, axis=0)
    # rescale the pixel values |
    image_array /= 255.
    return image_array

# function to predict the class of an image
def predict_class(image_path):
    # preprocess the image
    image_array = preprocess_image(image_path)
    # make the prediction
    prediction = model.predict(image_array)
    # get the predicted class label
    predicted_class = np.argmax(prediction)
    # return the class label
    return class_labels[predicted_class]

# example image
image_path = "C:\\Users\\nisha\\OneDrive\\Desktop\\Data_covid_CT\\Test\\images\\Image-6.png"
predicted_class = predict_class(image_path)
print('Predicted Class:', predicted_class)
plt.show()
```

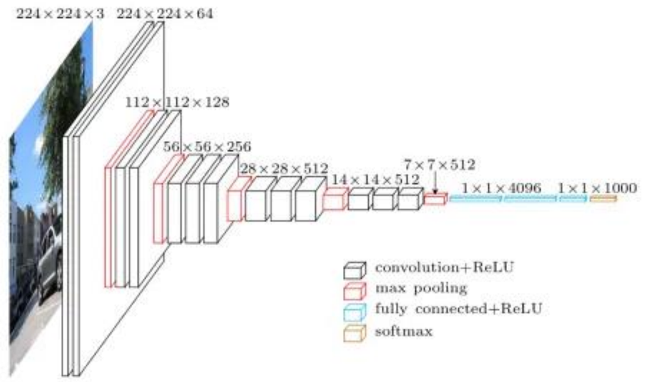


```
1/1 [==============================] - 0s 84ms/step
Predicted Class: COVID Positive
```



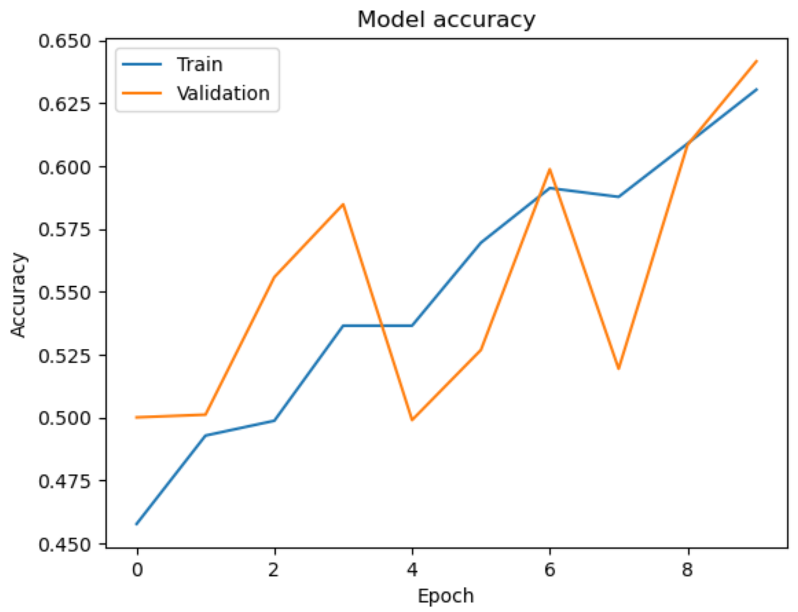
VGG16

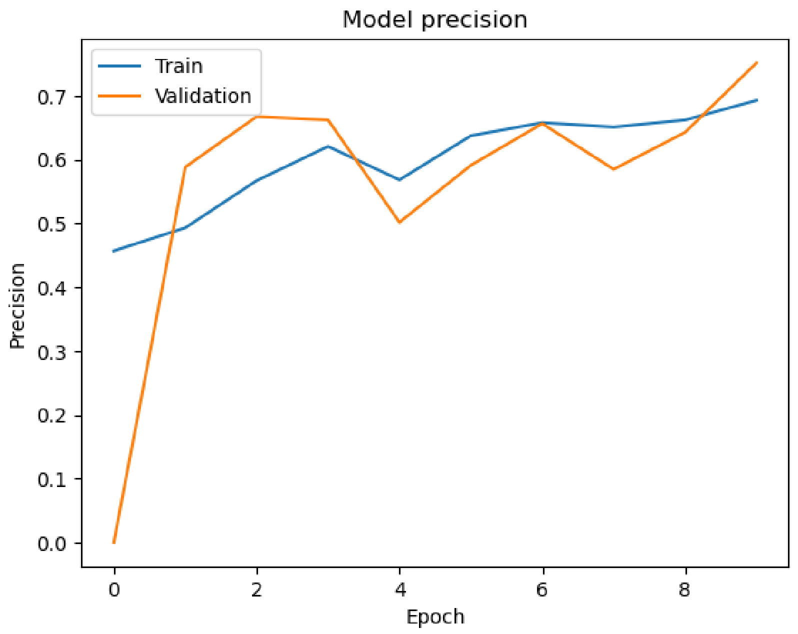
- VGG16 is a popular convolutional neural network architecture that is often used for image classification tasks.
- One of the main reasons why VGG16 is used is because of its simplicity and effectiveness.
- VGG16 uses small 3x3 convolutional filters throughout the network, which allows for the learning of more complex features in the images.
- VGG16 has also been used as a starting point for transfer learning, where the weights of the pre-trained network are used to improve performance on new datasets
- This new model takes an input image of size 224x224x3, processes it through the pre-trained VGG16 model, and passes the output through a dense layer with 256 neurons and ReLU activation function.

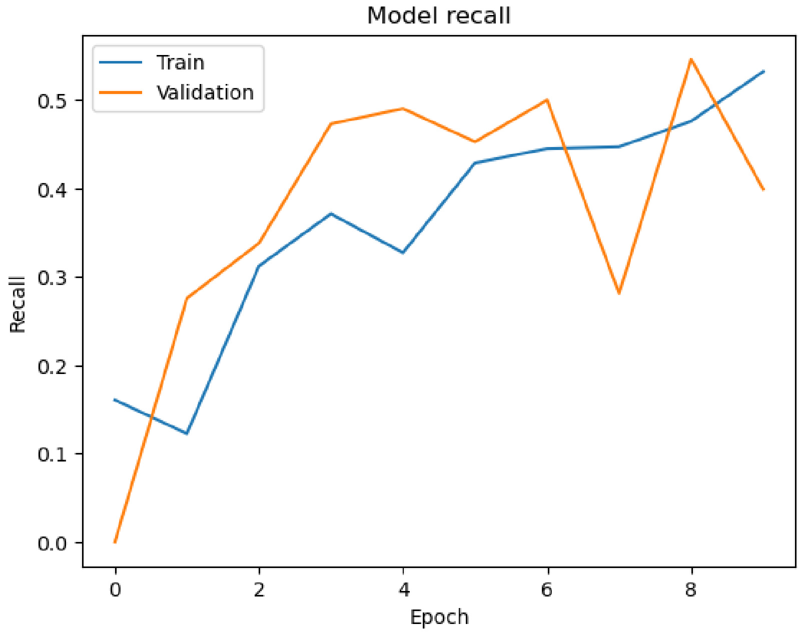


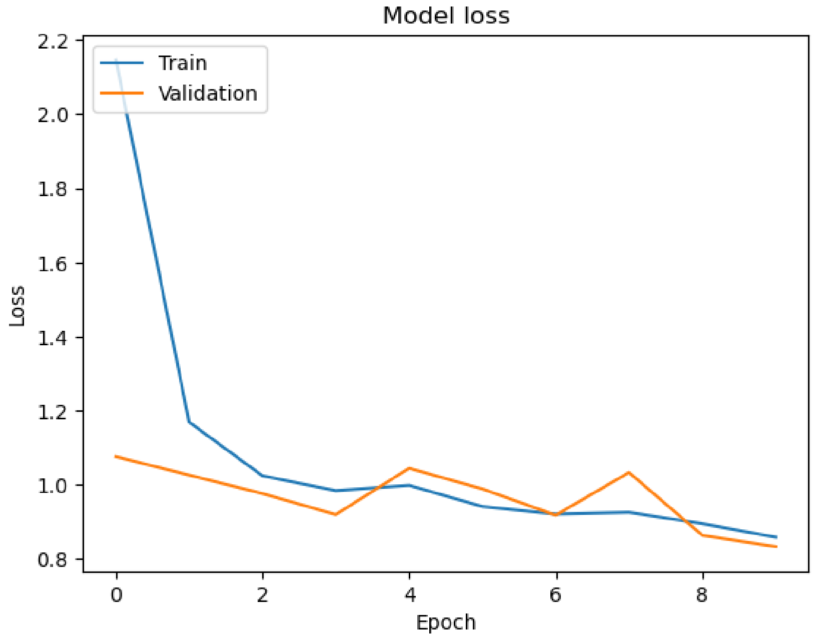
Architecture of VGG16

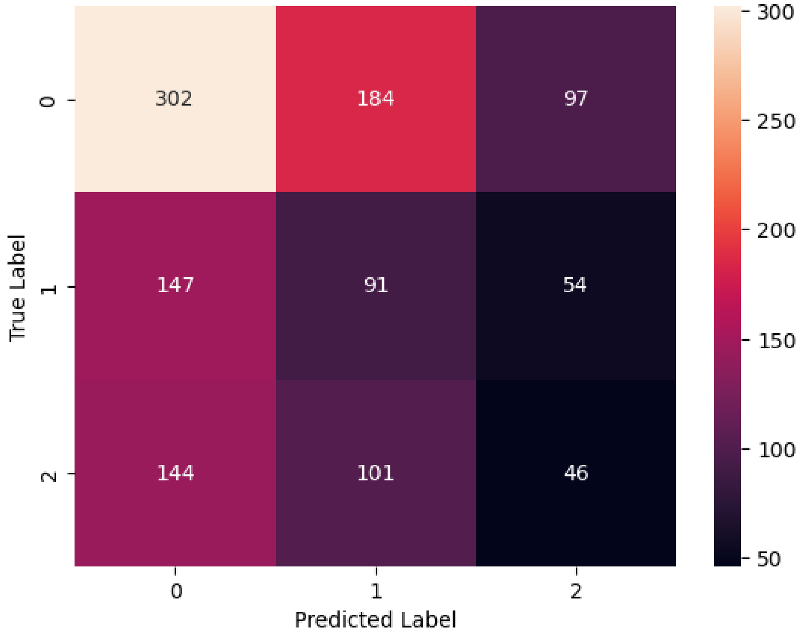
Evaluating the model performance

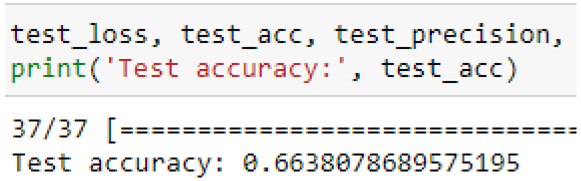





Model recall
Train
Validation
Recall
Epoch
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8


Model loss
Train
Validation
Loss
Epoch
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
0
2
4
6
8

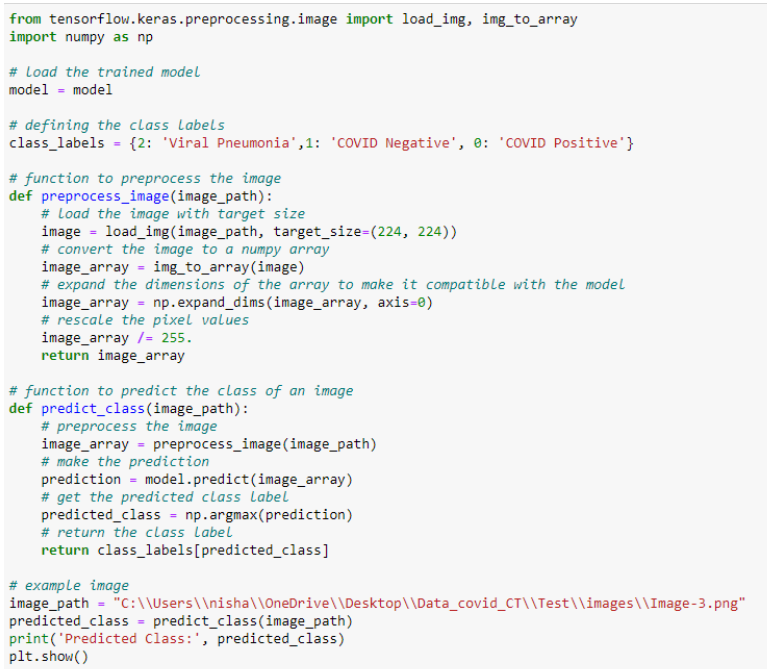




```
test_loss, test_acc, test_precision,
print('Test accuracy:', test_acc)
```

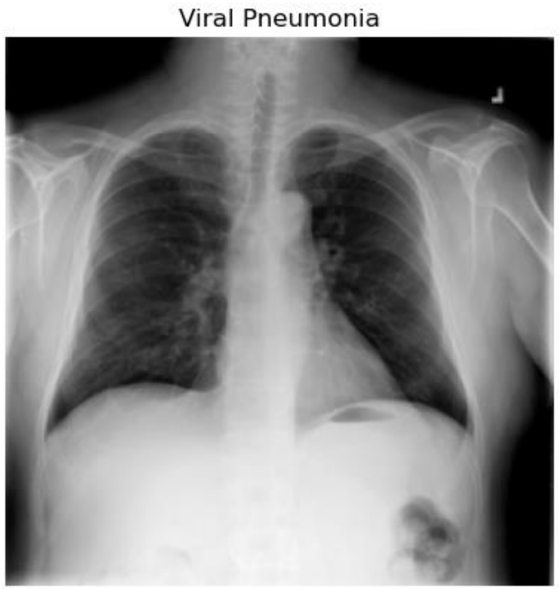
```
37/37 [================================
Test accuracy: 0.6638078689575195
```

Deployment and Monitoring



```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# load the trained model
model = model

# defining the class labels
class_labels = {2: 'Viral Pneumonia',1: 'COVID Negative', 0: 'COVID Positive'}

# function to preprocess the image
def preprocess_image(image_path):
    # load the image with target size
    image = load_img(image_path, target_size=(224, 224))
    # convert the image to a numpy array
    image_array = img_to_array(image)
    # expand the dimensions of the array to make it compatible with the model
    image_array = np.expand_dims(image_array, axis=0)
    # rescale the pixel values
    image_array /= 255.
    return image_array

# function to predict the class of an image
def predict_class(image_path):
    # preprocess the image
    image_array = preprocess_image(image_path)
    # make the prediction
    prediction = model.predict(image_array)
    # get the predicted class label
    predicted_class = np.argmax(prediction)
    # return the class label
    return class_labels[predicted_class]

# example image
image_path = "C:\\Users\\nisha\\OneDrive\\Desktop\\Data_covid_CT\\Test\\images\\Image-3.png"
predicted_class = predict_class(image_path)
print('Predicted Class:', predicted_class)
plt.show()
```

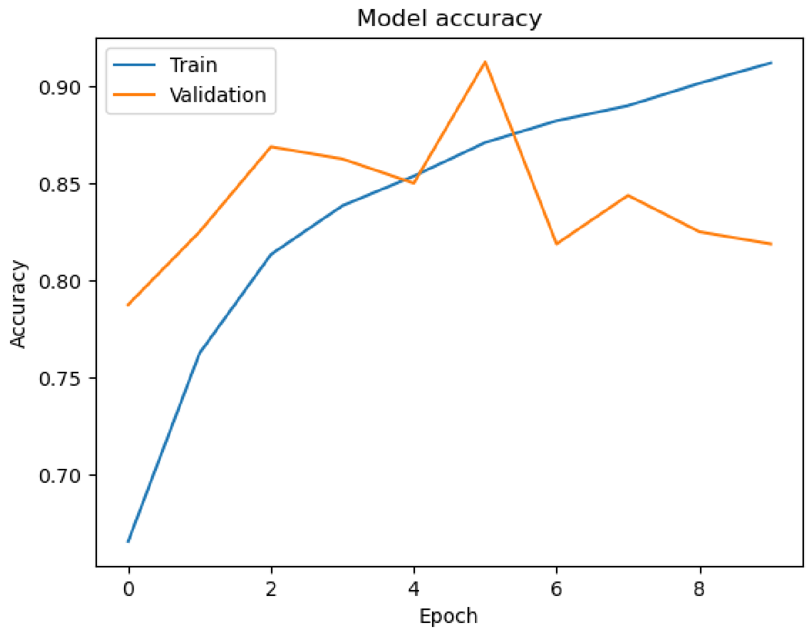
```
Predicted Class:
Viral Pneumonia
```

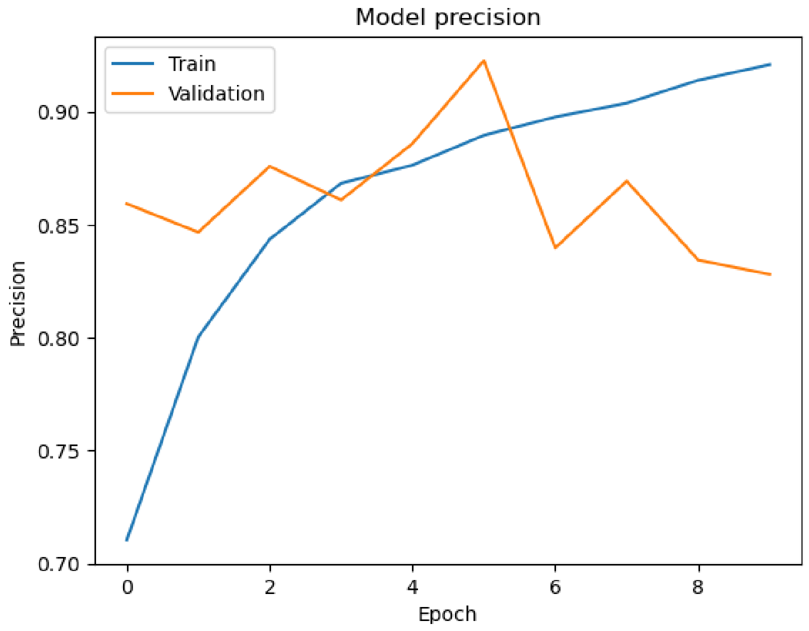


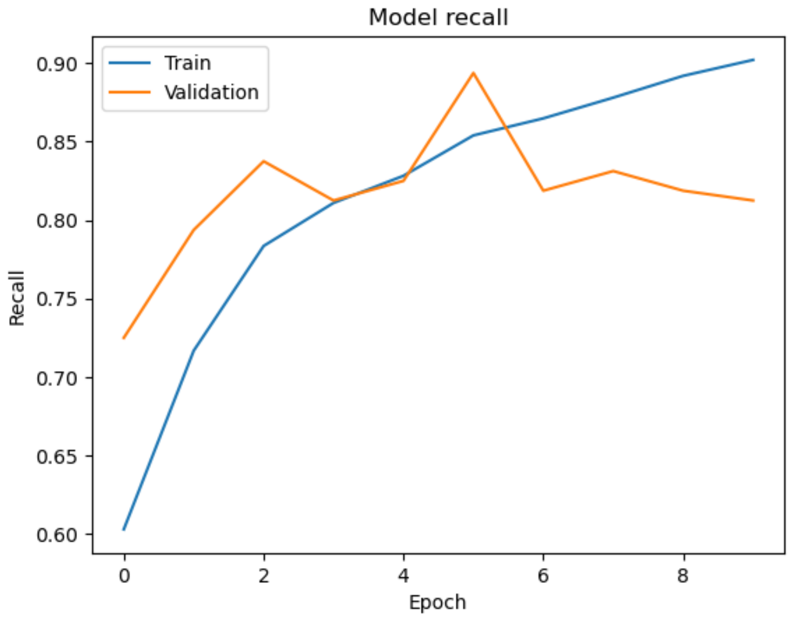
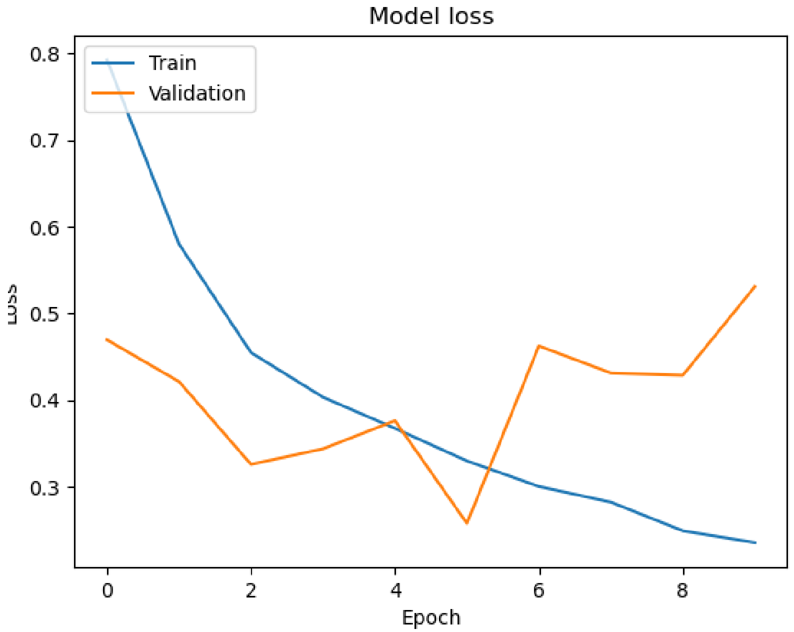
NASNet

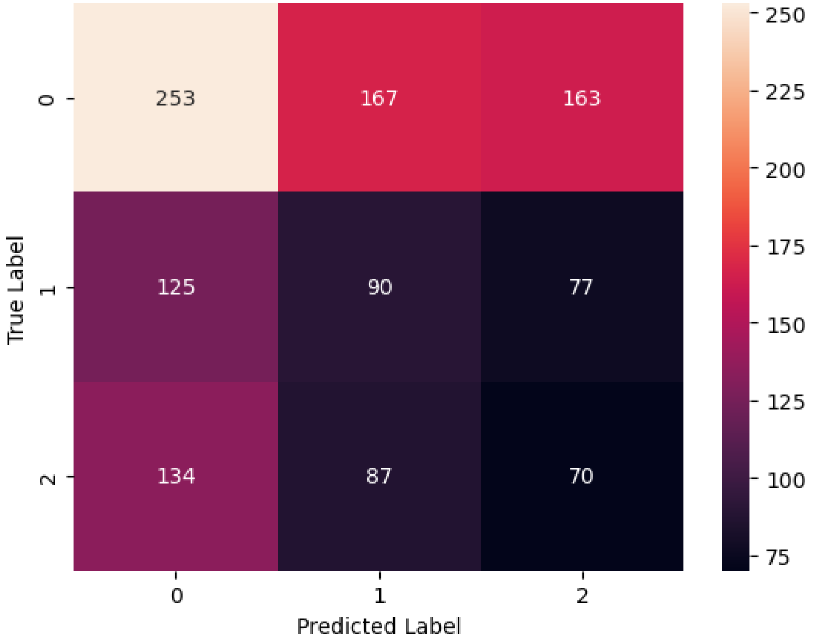
- NASNet (Neural Architecture Search Network) is a state-of-the-art deep learning architecture for image classification tasks.
- NASNet was developed by Google researchers and has achieved top performance on several benchmark datasets, including ImageNet.
- One of the main advantages of NASNet is its ability to adapt to different input sizes and resolutions, which makes it suitable for a wide range of applications, from mobile devices to high-performance computing systems.
- Overall, NASNet is a powerful architecture that can achieve state-of-the-art performance on various image classification tasks.
- The pre-trained layers of the model are then frozen to prevent their weights from being updated during training, which can help prevent overfitting and speed up the training process.

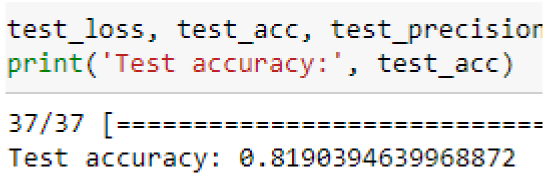
Evaluating the model performance






Model recall
Train
Validation
Recall
Epoch
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8

Model loss
Train
Validation
Loss
Epoch
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8

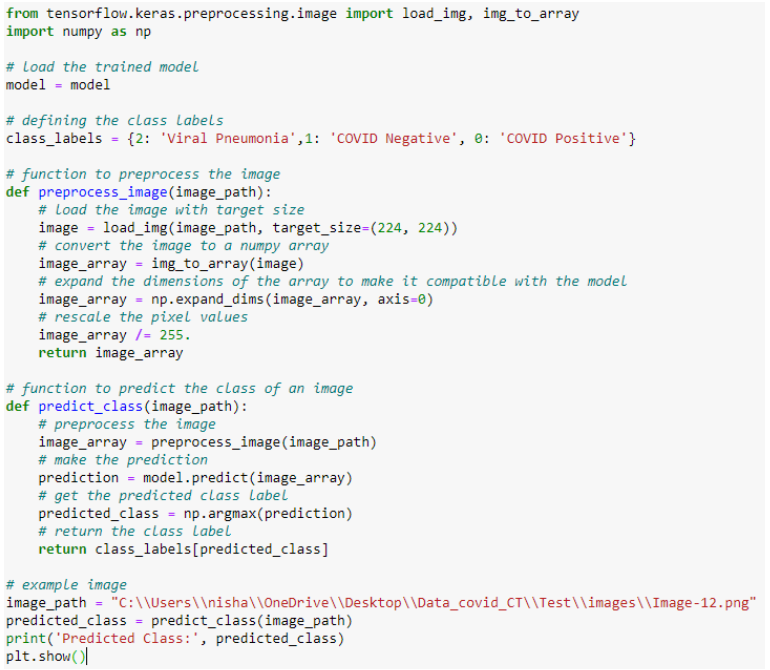




```
test_loss, test_acc, test_precision
print('Test accuracy:', test_acc)
```

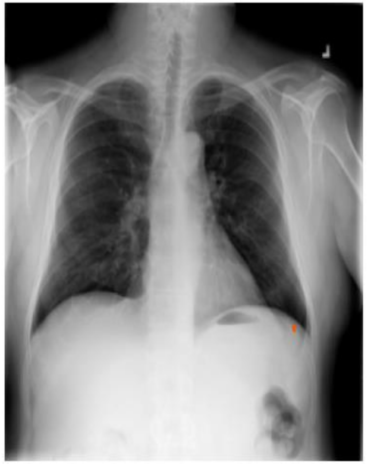
```
37/37 [=============================
Test accuracy: 0.8190394639968872
```

- Save the model using h5



```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# load the trained model
model = model

# defining the class labels
class_labels = {2: 'Viral Pneumonia',1: 'COVID Negative', 0: 'COVID Positive'}

# function to preprocess the image
def preprocess_image(image_path):
    # load the image with target size
    image = load_img(image_path, target_size=(224, 224))
    # convert the image to a numpy array
    image_array = img_to_array(image)
    # expand the dimensions of the array to make it compatible with the model
    image_array = np.expand_dims(image_array, axis=0)
    # rescale the pixel values
    image_array /= 255.
    return image_array

# function to predict the class of an image
def predict_class(image_path):
    # preprocess the image
    image_array = preprocess_image(image_path)
    # make the prediction
    prediction = model.predict(image_array)
    # get the predicted class label
    predicted_class = np.argmax(prediction)
    # return the class label
    return class_labels[predicted_class]

# example image
image_path = "C:\\Users\\nisha\\OneDrive\\Desktop\\Data_covid_CT\\Test\\images\\Image-12.png"
predicted_class = predict_class(image_path)
print('Predicted Class:', predicted_class)
plt.show()
```

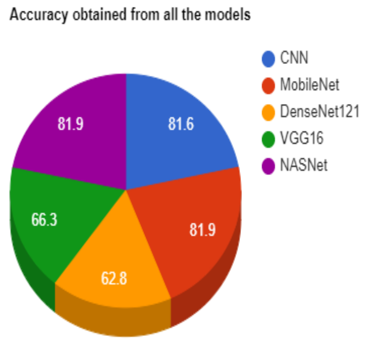
COVID-19 Negative



```
Predicted Class: COVID Negative
```

Conclusion

- Deep-learning architectures have shown promise in detecting COVID-19 in suspected patients by analyzing CT scan images.
- By evaluating patterns that identify the presence of COVID-19 in a person, deep learning methods have proven to be highly efficient.
- Some of the most high-performance deep learning architectures include CNN, VGG16, DenseNet, MobileNet, and NASNet.
- The models are designed to classify COVID, non-COVID images and viral pneumonia, and their accuracy is tested using a test dataset.
- Model performance is evaluated using accuracy, precision and recall.
- Among all the models, MobileNetV2 and NASNet achieved the highest accuracy of 81.9%.
- This makes the MobileNetV2 and NASNet model the best choice for classifying the provided CT scan images into COVID, non-COVID and Viral Pneumonia.



Future Scope

- There are several potential future directions to enhance the provided method's accuracy and usefulness.
- One promising approach is to expand the dataset for training the models by collecting more CT scan images and incorporating additional medical imaging data, such as X-rays or MRI scans. This can help improve the robustness and generalizability of the models and increase their accuracy in detecting COVID-19.
- In addition, it may be possible to develop more sophisticated models that classify COVID-19 from CT scan images and identify the affected lung regions. This can be accomplished through attention mechanisms or other advanced neural network architectures.
- Furthermore, the models can be integrated into a larger automated COVID-19 diagnosis system that can be used in clinical settings to help identify patients who require further testing or treatment. This can significantly reduce the burden on healthcare providers and improve the speed and accuracy of COVID-19 diagnosis.
- Overall, further exploration and development of these approaches can lead to significant advancements in COVID-19 diagnosis using CT scan images and improve patient outcomes.

References

- M. V. P. J. Sandeep Kumar Mathivanan, "Forecasting of the SARS-CoV-2 epidemic in India using SIR model, flatten curve and herd immunity," Journal of Ambient Intelligence and Humanized Computing, 2020.
- A. O. T. T. Sos Agaian, "Automatic COVID-19 lung infected region segmentation and measurement using CT-scans images," Elsevier - PMC COVID-19 Collection , 2020.
- S. K. M. Maheshwari V, "Social economic impact of COVID-19 outbreak in India," International Journal of Pervasive Computing and Communications, 2020.
- A. O. A. K. L. d. Sos Agaian b, "Automatic COVID-19 lung infected region segmentation and measurement using CT-scans images".
- D. A. T. M. A. Asaad, "COVID-19 Detection Using CNN Transfer Learning from x-Ray Images,," 2020.
- M. E. A. A. S. Saman Fouladi, "Efficient deep neural networks for classification of COVID-19 based on CT images: Virtualization via software defined radio," 2021.
- A. R. K. Ali Abbasian Ardakani, "Application of deep learning technique to manage COVID-19 in routine clinical practice using CT images: Results of 10 convolutional neural networks," 2020.
- Z. L. Gao Huang, "Densely Connected Convolutional Networks," 2018.