Covid-19 Chest X-Ray Image Recognition

Task :-

Build an image classification model which will Identify covid positive patient by looking into the chest Xray images.

Abstract:-

The outbreak of COVID-19 has become a global pandemic, leading to a significant increase in the demand for accurate and efficient diagnostic tools. Chest X-ray images have been widely used in the diagnosis of COVID-19, making it crucial to develop effective classification models. In this project, we explore the use of Convolutional Neural Networks (CNN) and two popular pre-trained models, ResNet50 and VGG16, for COVID-19 chest X-ray image classification. We evaluate the performance of each model based on accuracy, precision, recall, and F1-score, providing insights into their effectiveness in COVID-19 detection.

Introduction :-

The novel coronavirus disease first appeared in the city of Wuhan, China, in late 2019. While China's government took a number of precautionary measures, including a lockdown of the city, to limit the spread of COVID-19, it had, by then, been too late to contain the virus.
The main challenge is to detect COVID-19 with an accurate and low-cost detection methods. Convolutional Neural Networks (CNN) have been demonstrated to be very powerful in including extraction and learning, so they are generally adopted by researchers. The purpose of this work is to establish a fully automated system for the classification of COVID-19 and non-COVID-19 pneumonia. We have trained three popular convolutional networks on the elaborated dataset. These networks (CNN, VGG16 and ResNet50) have achieved compelling results in some tasks in recent years. We fine-tune them for the purpose of the detection of COVID-19. As of today, only a limited set of X-ray images relating to COVID-19 inquiries is available for public use. Thus, we could not train these models from scratch. In this work, we adopt two strategies to solve the COVID-19 image shortage problem:

- we apply data augmentation in order to build a converted version of the COVID-19 image (e.g., flip, small rotation, small distortion, etc.) to triple the set of samples.

- we fine-tuned the last layer of the model; thus, we can use less labelled samples per category for the training process.

Dataset:

- The dataset consisting of chest X-ray images COVID-19, pneumonia, and normal cases. The dataset extracted from https://www.kaggle.com/datasets/pranavraikokte/covid19-image-dataset. The dataset was divided into training, test sets, ensuring an equal distribution of images across the three classes. The constructed dataset for this work consists of 317 images.
- Covid: 90
- Viral pneumonia: 137
- Normal: 90
- In our experiment, we focus on reducing false positives and false negatives by using the transfer learning process with 3 Convolutional Neural Network (CNN) on an augmented dataset by implementing three augmentation strategies on our collected chest X-ray images.
- Image demonstration: We demonstrated the image of three classes Covid, Normal and Viral Pneumonia.

```
def image_plot(train_data,image_numbers):
    img_feature = train_data[0][0][:image_numbers]
    img_label = train_data[0][1][:image_numbers]

    plt.figure(figsize=(20, 15))
    for i in range(image_numbers):
        ax = plt.subplot(math.ceil(image_numbers/4),4, i + 1)
        plt.imshow(img_feature[i])
        plt.title("Normal" if img_label[i] == 0 else "Viral Pneumonia" if img_label[i] == 1 else "Covid")
        plt.axis("off")

image_plot(train_data,15)
```

Preprocessing:

- To ensure the compatibility of images with the CNN model, we resized them to a fixed dimension and normalized pixel values. The input images were in different original size, then they were all processed and they were made uniform by changing the dimensions to 224 × 224 pixels.

Model architecture:

Three different models are implemented for the classification task:
1. Custom CNN.
2. VGG16.
3. RESNET50.

CNN Model Implementation:

- Before we get into the coding details, let's first take a look at the general structure of the model we're proposing. Notice that the model has a similar structure to VGG-16 but has fewer layers and a much smaller input image size, and therefore far fewer trainable parameters. The model contains three convolutional blocks followed by a fully connected layer and an output layer. For reference, we've included the number of channels at key points in the architecture. We have also indicated the spatial size of the activation maps at the end of each convolutional block. This is a good visual to refer back to when studying the code below.
- For convenience, we're going to define the model in a function. Notice that the function has one optional argument: the input shape for the model. We first start by instantiating the model by calling the sequential() method. This allows us to build a model sequentially by adding one layer at a time. Notice that we define three convolutional blocks and that their structure is very similar.

Defining the convolutional block for the CNN:

```
model1 = Sequential()
model1.add(Conv2D(32, kernel_size = (3,3), activation = 'relu', input_shape = (224,224,3)))
model1.add(MaxPool2D())

model1.add(Conv2D(64, kernel_size = (3,3), activation = 'relu'))
model1.add(MaxPool2D())

model1.add(Conv2D(128, kernel_size = (3,3), activation = 'relu'))
model1.add(MaxPool2D())

model1.add(Flatten())
model1.add(Dense(128, activation = 'relu'))
model1.add(Dense(64, activation = 'relu'))
model1.add(Dense(no_labels, activation = 'softmax'))
```

- tf.keras.layers.Conv2D : 2D convolution layer function. It has various parameters like kernel_size, activation, padding, filters etc.
    1. tf.keras.layers.MaxPool2D : Max pooling operation for 2D spatial data.
    2. tf.keras.layers.Flatten : Flattens the input.
    3. tf.keras.layers.Dense : Just your regular densely-connected NN layer
    4. tf.keras.layers.Dropout : Applies Dropout to the input.

- Since the class labels are integer indices, we use sparse_categorical_cross entropy. If the class labels are one-hot vectors, we could use sparse_categorical_cross entropy.
- Now use this Conv2D along with the other layers(like keras.layers.MaxPooling2D, keras.layers.Dense, etc.) to build the classification model model. Here we set input_shape as [224,224,3] since the size of each image is 224x224x3.

Visualizing the model:

- We can now create an instance of the model by calling the function above and use the summary() method to display the model summary to the console.

```
1  model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 32)      896

max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
)

conv2d_1 (Conv2D)            (None, 109, 109, 64)      18496

max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0
2D)

conv2d_2 (Conv2D)            (None, 52, 52, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0
2D)

flatten (Flatten)            (None, 86528)             0

dense (Dense)                (None, 128)               11075712

dense_1 (Dense)              (None, 64)                8256

dense_2 (Dense)              (None, 3)                 195

=================================================================
Total params: 11,177,411
Trainable params: 11,177,411
Non-trainable params: 0
_________________________________________________________________
```

Compile the model:

- The next step is to compile the model, where we specify the optimizer type and loss function and any additional metrics we would like recorded during training. Here we specify Adam as the optimizer type for gradient descent, and we use a cross-entropy loss function which is the standard loss function for classification problems. We specifically use sparse categorical _cross entropy since our labels are one-hot encoded. Finally, we specify accuracy as an additional metric to record during training. The value of the loss function is always recorded by default, but we specified accuracy to get the accuracy.

```
1  model1.compile(optimizer="adam",
2                 loss='SparseCategoricalCrossentropy',
3                 metrics='accuracy')
```

Train the model:

```
history = model1.fit(train_data,steps_per_epoch = 7,
                     epochs = 12,
                     validation_data = test_data)
```

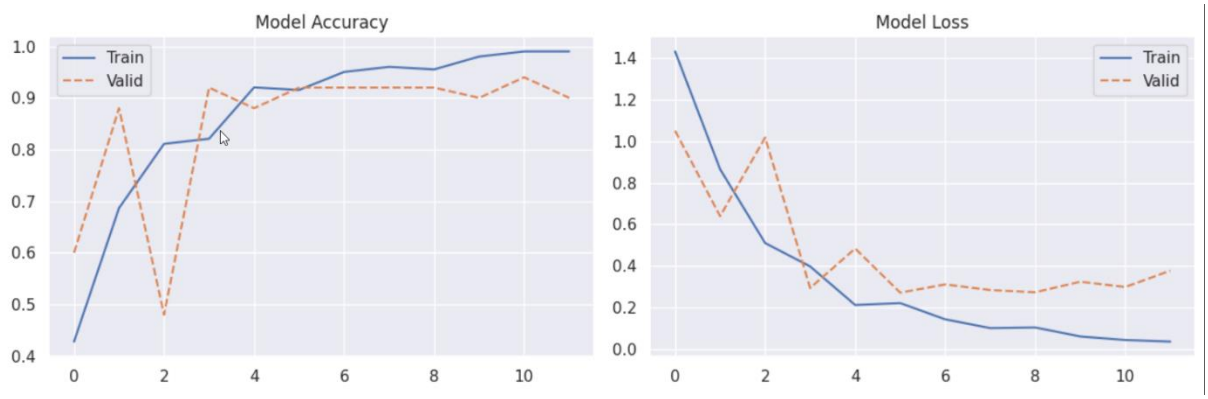
```
Epoch 1/12
7/7 [==============================] - 51s 7s/step - loss: 1.7694 - accuracy: 0.4279 - val_loss: 0.9765 - val_accuracy: 0.5303
Epoch 2/12
7/7 [==============================] - 22s 3s/step - loss: 0.6992 - accuracy: 0.7562 - val_loss: 0.4853 - val_accuracy: 0.7424
Epoch 3/12
7/7 [==============================] - 21s 3s/step - loss: 0.3601 - accuracy: 0.8607 - val_loss: 0.4844 - val_accuracy: 0.7273
Epoch 4/12
7/7 [==============================] - 21s 3s/step - loss: 0.4944 - accuracy: 0.7811 - val_loss: 0.6115 - val_accuracy: 0.7424
Epoch 5/12
7/7 [==============================] - 20s 3s/step - loss: 0.2984 - accuracy: 0.8955 - val_loss: 0.4495 - val_accuracy: 0.8182
Epoch 6/12
7/7 [==============================] - 22s 3s/step - loss: 0.2087 - accuracy: 0.9104 - val_loss: 0.3659 - val_accuracy: 0.8182
Epoch 7/12
7/7 [==============================] - 20s 3s/step - loss: 0.1634 - accuracy: 0.9353 - val_loss: 0.2251 - val_accuracy: 0.9091
Epoch 8/12
7/7 [==============================] - 22s 3s/step - loss: 0.1428 - accuracy: 0.9502 - val_loss: 0.2630 - val_accuracy: 0.9091
Epoch 9/12
7/7 [==============================] - 21s 3s/step - loss: 0.0761 - accuracy: 0.9701 - val_loss: 0.2316 - val_accuracy: 0.8939
Epoch 10/12
7/7 [==============================] - 22s 3s/step - loss: 0.0601 - accuracy: 0.9751 - val_loss: 0.2630 - val_accuracy: 0.8788
Epoch 11/12
7/7 [==============================] - 21s 3s/step - loss: 0.0622 - accuracy: 0.9652 - val_loss: 0.2995 - val_accuracy: 0.8939
Epoch 12/12
7/7 [==============================] - 22s 3s/step - loss: 0.0681 - accuracy: 0.9701 - val_loss: 0.4454 - val_accuracy: 0.8333
```

Plot the training result:

- The function below is a convenience function to plot training and validation losses and training and validation accuracies. It has a single required argument which is a list of metrics to plot.

```
#model.save("model.h5")

sns.set()
fig = plt.figure(0, (12, 4))

ax = plt.subplot(1, 2, 1)
sns.lineplot(x=history.epoch, y=history.history['accuracy'], label='Train')
sns.lineplot(x=history.epoch, y=history.history['val_accuracy'], label='Valid', linestyle = 'dashed')
plt.title('Model Accuracy')
plt.tight_layout()

ax = plt.subplot(1, 2, 2)
sns.lineplot(x=history.epoch, y=history.history['loss'], label='Train')
sns.lineplot(x=history.epoch, y=history.history['val_loss'], label='Valid', linestyle = 'dashed')
plt.title('Model Loss')
plt.tight_layout()
#plt.savefig('epoch_history_dcnn.png')
plt.show()
```

- The loss and accuracy metrics can be accessed from the history object returned from the fit method. We access the metrics using predefined dictionary keys, as shown below.



Model VGG16:

The VGG net architecture was proposed in 2014 by Simonyan et al. and referred to as Very Deep Convolutional Networks for Large-scale Image Recognition. The characteristics of VGG series networks are the 3 × 3 convolutional layers that are one on top of the other, and the depth is getting larger and larger. Reducing the volume size is treated by a maximum pooling.

Defining and Visualizing the convolutional block for the VGG16:

The VGG16 architecture is composed as following:

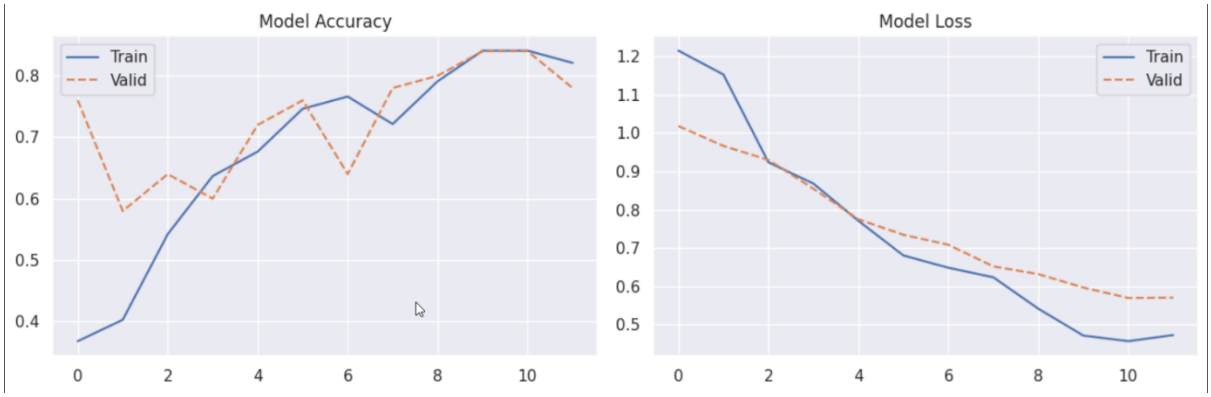
- Two Convolutional layers with 64 filters followed by Max pooling layer
- Two Convolutional layers with 128 filters followed by Max pooling layer
- Three Convolutional layers with 256 filters followed by Max pooling layer
- Two stack each with 3 convolutional layers with 512 filters and separated by a max pooling layer
- A final Max pooling layer

- Two fully connected layers with 4096 channels
- SoftMax output layer with 1000 classes

```
tf.keras.backend.clear_session()
input_shape = (224,224,3)
base_model = tf.keras.applications.vgg16.VGG16(
    weights='imagenet',
    include_top=False,
    input_shape=input_shape
)
base_model.trainable = False

model_vgg16 = tf.keras.Sequential()
model_vgg16.add(base_model)
model_vgg16.add(tf.keras.layers.GlobalAveragePooling2D())

model_vgg16.add(tf.keras.layers.Flatten())
model_vgg16.add(tf.keras.layers.Dense(256, activation='relu'))
model_vgg16.add(tf.keras.layers.Dropout(0.5))
model_vgg16.add(tf.keras.layers.Dense(256, activation='relu'))
model_vgg16.add(tf.keras.layers.Dropout(0.5))

model_vgg16.add(tf.keras.layers.Dense(3, activation='softmax'))

model_vgg16.compile(loss='SparseCategoricalCrossentropy',
                    optimizer=tf.keras.optimizers.Adam(0.001),
                    metrics=['acc'])
model_vgg16.summary()
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 3)                 771

=================================================================
Total params: 14,912,579
Trainable params: 197,891
Non-trainable params: 14,714,688
_________________________________________________________________
```

Plot the training result of VGG16:



Model Resnet50:

ResNet-50 is a CNN contains 50 layers; it is deeper than VGG16. Since a global average pool is used instead of a fully connected layer, the size of the model is actually much smaller, which reduces the model size of ResNet50 to 102 MB. The special part of Resnet is residual block learning. This means that each layer should feed into the next layer as well as directly into the layers about 2–3 hops away.

Defining and Visualizing the convolutional block for the Resnet50:

Its architecture is composed as follow:

- A convolutional layer with 64 filters and kernel size of 7 × 7. This is followed by a max pooling layer with a stride size of 2.

- Then, a convolutional layer with 64 filters and a kernel size of 1 * 1, followed by a second convolutional layer with 64 filters and a kernel size of 3 * 3. Then, we have another convolutional layer with 256 filters and a kernel size of 1 * 1. These 3 layers are replicated in total 3 time and 9 layers are obtained at this stage.

- Next, 3 convolutional layers, the first one is with 128 filters and a kernel size of 1 * 1, the second one is with 128 filters and kernel size of 3 * 3, and the third one is with 512 filters and a kernel size of 1 * 1. These layers are replicated 4 time to give us 12 layers at this stage.

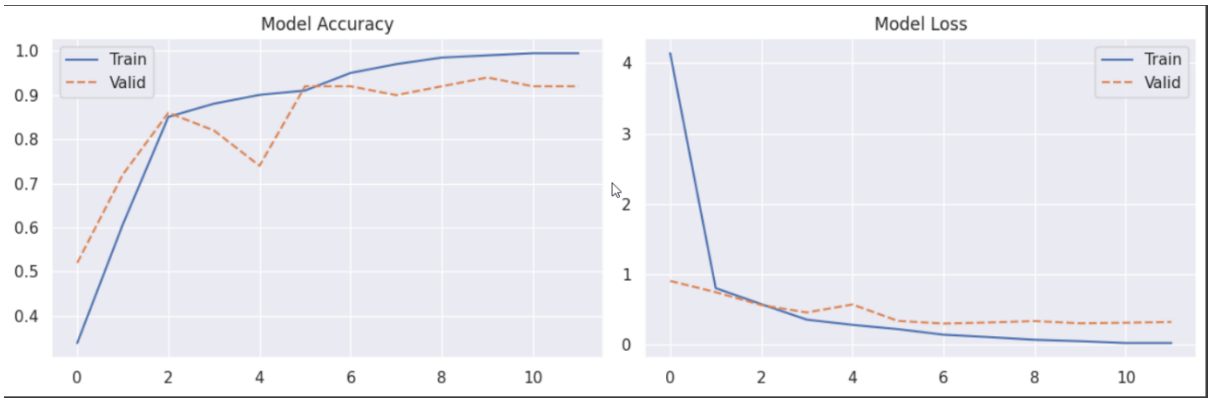
- Afterwards, we have convolutional layer with 256 filters and a kernel size of 1 * 1, and two others with 256, 1024 filters and a kernel size of 3 * 3, 1 * 1. This is replicated 6 time to give us totally 18 layers.

- Then, we have a convolutional layer with 512 filters and a kernel size of 1 * 1, with two others with 512, 2048 and a kernel size of 1 * 1, 3 * 3. This is replicated 3 times to give us totally 9 layers.

- Finally, we apply an average pooling and finish it with a fully connected layer (with 1000 nodes) and then a SoftMax function to give us 1 layer as a final stage.

```
# Modelling ResNet50
tf.keras.backend.clear_session()
model_resnet50 = Sequential([
    Conv2D(16, 3, padding="same", activation="relu", input_shape=(224,224, 3)),
    MaxPool2D(),
    Conv2D(32, 3, padding="same", activation="relu"),
    MaxPool2D(),
    Flatten(),
    Dense(128, activation="relu"),
    Dense(3, activation="softmax")
])
base_model = tf.keras.applications.resnet50.ResNet50(
    include_top=False,
    weights='imagenet',
    input_shape=(224,224,3)
)

model_resnet50.summary()
model_resnet50.compile(optimizer='adam', loss='SparseCategoricalCrossentropy', metrics='accuracy')
```

Plot the training result of model Resnet50:



Classification report comparison of three models CNN, VGG16 & RESNET50:

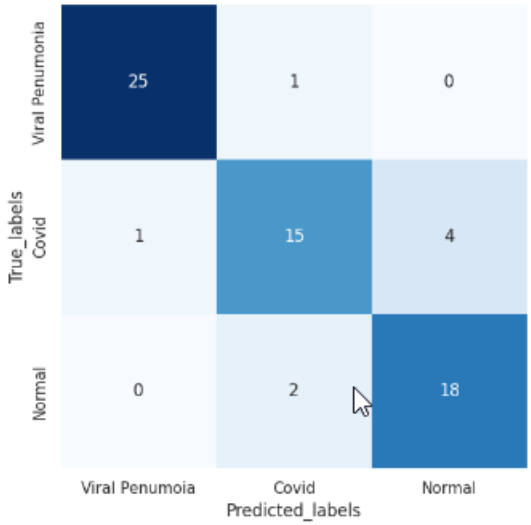
- The classification report provides precision, recall, F1-score, and support for each class. It gives insights into the model's performance for individual classes and overall accuracy. Precision measures the proportion of correctly predicted instances of a class out of all instances predicted as that class. Recall measures the proportion of correctly predicted instances of a class out of all instances that actually belong to that class. F1-score is the harmonic mean of precision and recall.

Model	Classification Report				
Custom_CNN	precision	recall	f1-score	support	
	Covid	0.96	0.96	0.96	26
	Viral Pneumonia	0.83	0.75	0.79	20
	Normal	0.82	0.90	0.86	20
	accuracy			0.88	66
	macro avg	0.87	0.87	0.87	66
	weighted avg	0.88	0.88	0.88	66
VGG16	precision	recall	f1-score	support	
	Covid	1.00	1.00	1.00	26
	Viral Pneumonia	0.93	0.65	0.76	20
	Normal	0.73	0.95	0.83	20
	accuracy			0.88	66
	macro avg	0.89	0.87	0.86	66
	weighted avg	0.90	0.88	0.88	66
Resnet50	precision	recall	f1-score	support	
	Covid	0.93	1.00	0.96	26
	Viral Pneumonia	1.00	0.85	0.92	20
	Normal	0.95	1.00	0.98	20
	accuracy			0.95	66
	macro avg	0.96	0.95	0.95	66
	weighted avg	0.96	0.95	0.95	66

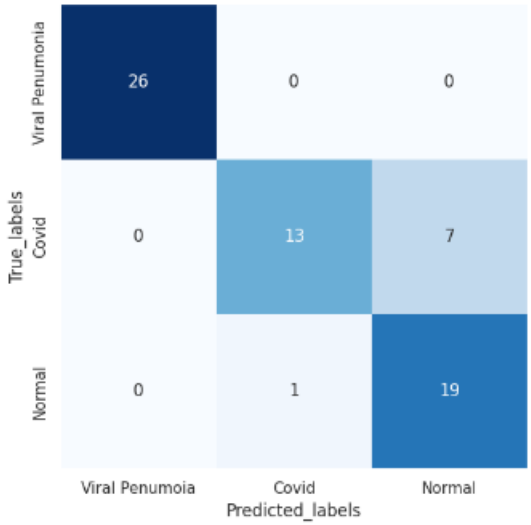
Confusion Matrix:

The confusion matrix is a visual representation of the model's performance, displaying the number of true positives, true negatives, false positives, and false negatives for each class. It helps identify misclassifications and evaluate the model's ability to differentiate between classes.

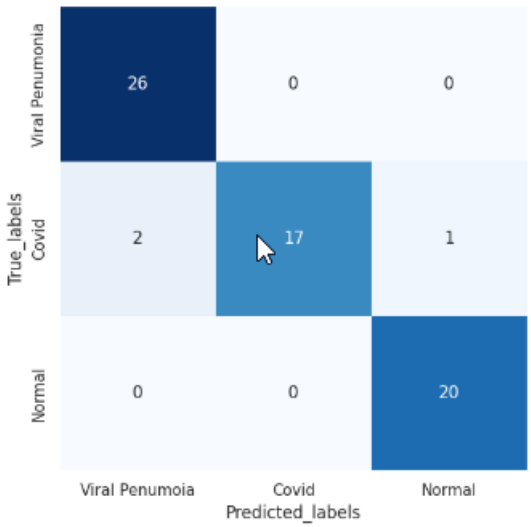
1. Custom CNN.



2. VGG_16



3. Resnet50



Further Improvements:

Several areas can be explored to further enhance the performance of the models and their applicability in real-world scenarios:

- Exploring other pre-trained models and transfer learning techniques to leverage the knowledge learned from large-scale datasets.
- Fine-tuning the model architectures to improve performance by adjusting hyperparameters or adding more layers.
- Investigating different data augmentation techniques, such as rotation, zooming, or random cropping, to improve the model's ability to generalize to unseen data.
- Collecting and incorporating more diverse and larger datasets to improve the model's robustness and generalization.
- Deploying the best-performing model as a web or mobile application for real-time COVID-19 detection, providing a user-friendly interface for healthcare professionals.

By conducting further research and incorporating these suggestions, the accuracy and reliability of COVID-19 detection using chest X-ray images can be significantly improved, thus contributing to the field of medical imaging and disease diagnosis.

Conclusion:

In this project, we developed and evaluated deep learning models for COVID-19 chest X-ray classification. The models were trained and evaluated using a balanced dataset containing images of COVID-19, Viral Pneumonia, and Normal cases. The performance of three models, including a custom CNN model, VGG16, and ResNet50, was compared using various evaluation metrics. The ResNet50 model demonstrated the best performance, achieving an accuracy of 95% and outperforming the other models in precision, recall, and F1-score. These results suggest that deep learning models have the potential to assist in the detection and diagnosis of COVID-19 cases based on chest X-ray images.