AIDS-2
EXPERIMENT NO. 9

Aim: Implementing Deep Learning Application using CNN.

Theory:

A Convolutional Neural Network (CNN) is a specialized deep learning model, primarily used for analyzing visual data. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input images, making them extremely powerful for tasks like image classification, object detection, and image segmentation.

1. Input Layer:

The CNN begins with an input layer where an image is provided in the form of pixel values. If the input is a grayscale image, the values represent light intensity, while for colored images, values are spread across different channels (e.g., red, green, blue).

2. Convolutional Layer:

The convolutional layer is the core building block of CNNs. It applies a set of filters or kernels to the input image, which scan over it in a process called convolution. This results in the extraction of important features like edges, corners, textures, and patterns from the image.

- Filters (Kernels): These are small matrices that are applied to the image to detect specific patterns. Different filters detect different features, such as horizontal edges or vertical edges.
- Stride: Stride defines how the filter moves across the image. A stride of 1 means the filter moves pixel by pixel, whereas a higher stride skips more pixels and produces smaller output.

3. Activation Function (ReLU):

After each convolution operation, an activation function is applied. The most common activation function in CNNs is ReLU (Rectified Linear Unit), which replaces all negative pixel values with zero, introducing non-linearity into the model.

4. Pooling Layer:

The pooling layer follows the convolutional layer and is responsible for downsampling the image. It reduces the spatial dimensions (width and height) of the image, helping reduce the computational complexity of the network.

- Max Pooling: It selects the maximum value from a specific window of the feature map, which reduces noise and retains important features.
- Average Pooling: It computes the average value of the elements in the window, focusing on smoothing and dimensionality reduction.

5. Flattening:

Once the convolution and pooling layers have reduced the image to an abstract representation, it is flattened into a single long vector. This is necessary for connecting the output of the CNN to the fully connected layers.

6. Fully Connected Layer (Dense Layer):

In the fully connected layer, every neuron is connected to every other neuron from the previous layer. This part of the CNN functions like a traditional neural network, using the learned features to classify the image.

7. Output Layer:

Finally, the CNN's output layer uses an activation function (like softmax for multi-class classification or sigmoid for binary classification) to produce a probability score for each class label.

8. Training the CNN:

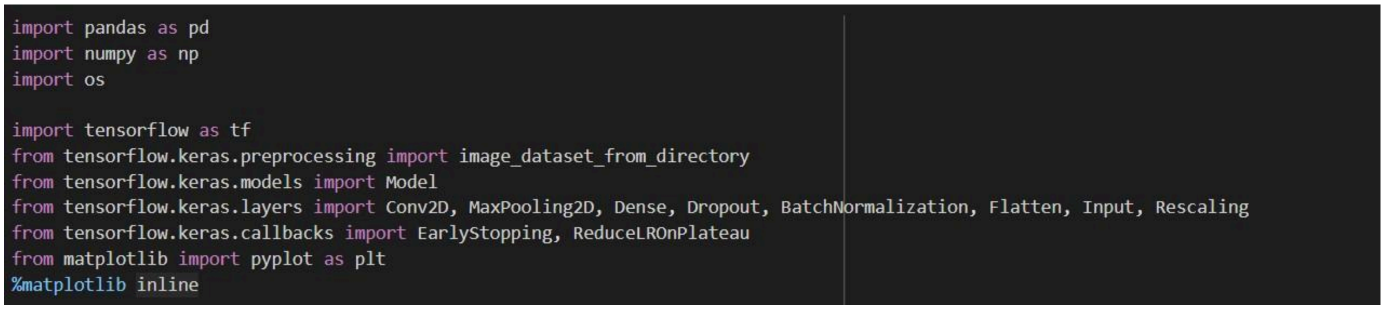
CNNs learn by adjusting the weights of the filters using backpropagation, an optimization technique that minimizes the difference between predicted and actual outputs. Through multiple epochs (iterations), the model improves its accuracy in classifying images.

Code and Output:

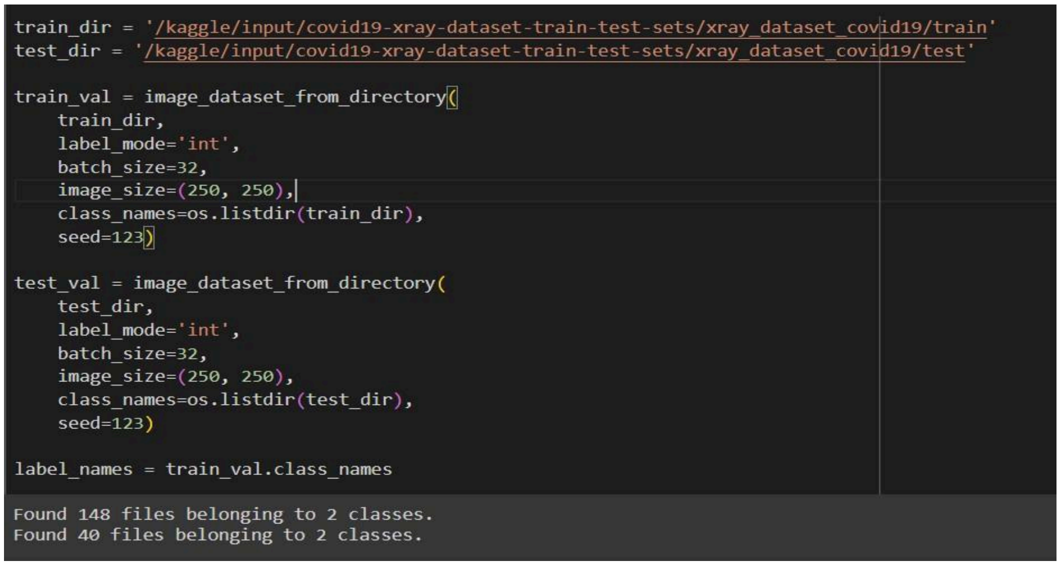
The 2019 novel coronavirus (COVID-19) presents several unique features. While the diagnosis is confirmed using polymerase chain reaction (PCR), infected patients with pneumonia may present on chest X-ray and computed tomography (CT) images with a pattern that is only moderately characteristic for the human eye Ng, 2020.

COVID-19’s rate of transmission depends on our capacity to reliably identify infected patients with a low rate of false negatives. In addition, a low rate of false positives is required to avoid further increasing the burden on the healthcare system by unnecessarily exposing patients to quarantine if that is not required.

Along with proper infection control, it is evident that timely detection of the disease would enable the implementation of all the supportive care required by patients affected by COVID-19.

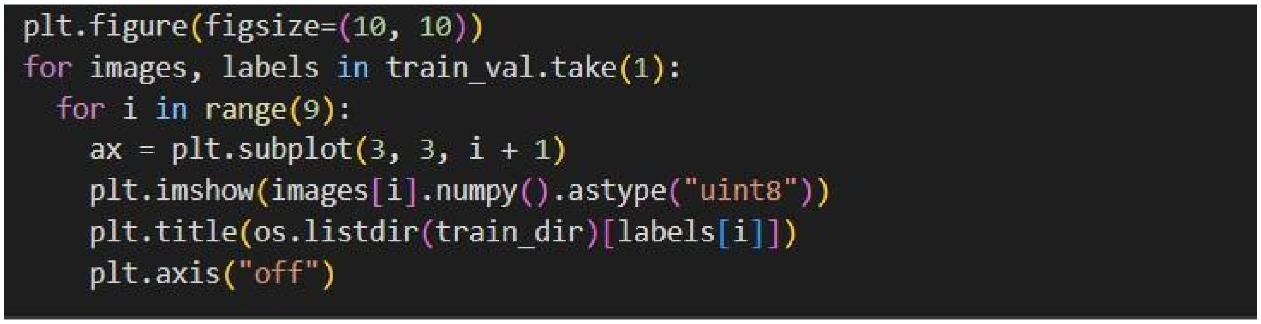


```
import pandas as pd
import numpy as np
import os

import tensorflow as tf
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, BatchNormalization, Flatten, Input, Rescaling
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from matplotlib import pyplot as plt
%matplotlib inline
```

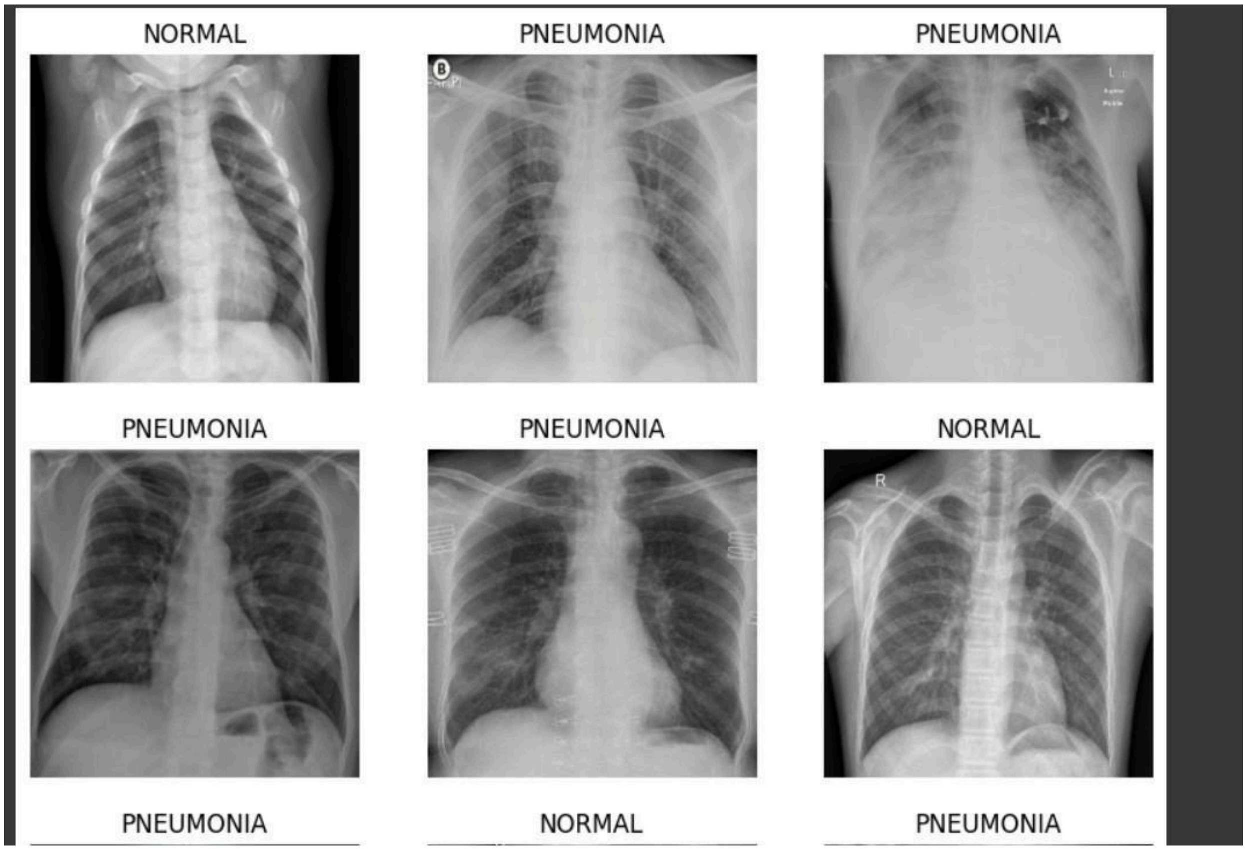


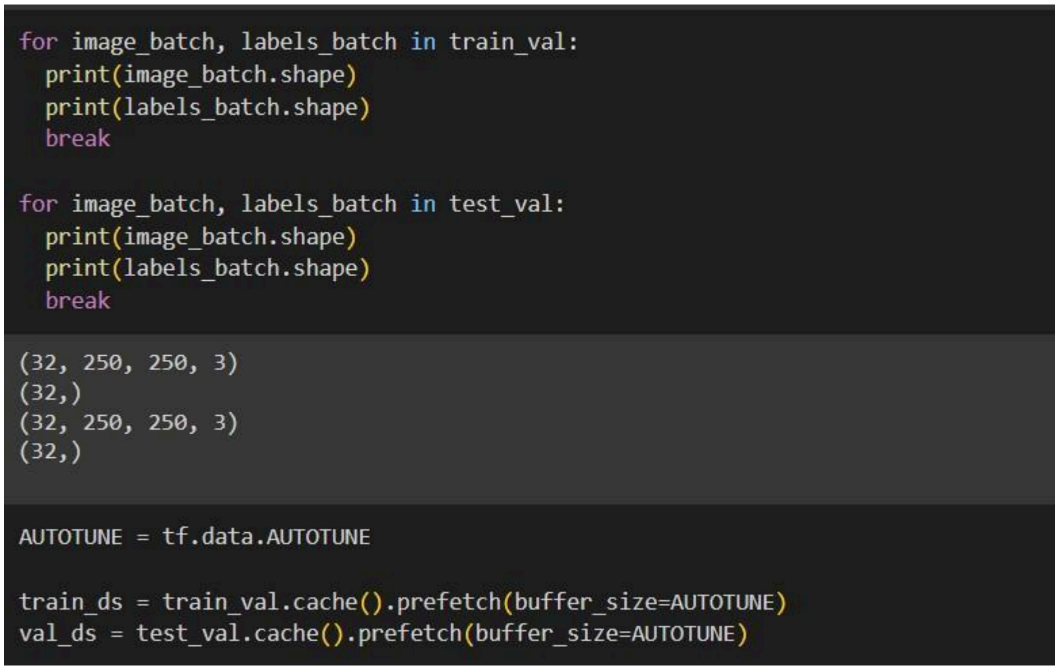
```
train_dir = '/kaggle/input/covid19-xray-dataset-train-test-sets/xray_dataset_covid19/train'
test_dir = '/kaggle/input/covid19-xray-dataset-train-test-sets/xray_dataset_covid19/test'

train_val = image_dataset_from_directory(
    train_dir,
    label_mode='int',
    batch_size=32,
    image_size=(250, 250),
    class_names=os.listdir(train_dir),
    seed=123)

test_val = image_dataset_from_directory(
    test_dir,
    label_mode='int',
    batch_size=32,
    image_size=(250, 250),
    class_names=os.listdir(test_dir),
    seed=123)

label_names = train_val.class_names
```

```
Found 148 files belonging to 2 classes.
Found 40 files belonging to 2 classes.
```



```
plt.figure(figsize=(10, 10))
for images, labels in train_val.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(os.listdir(train_dir)[labels[i]])
    plt.axis("off")
```

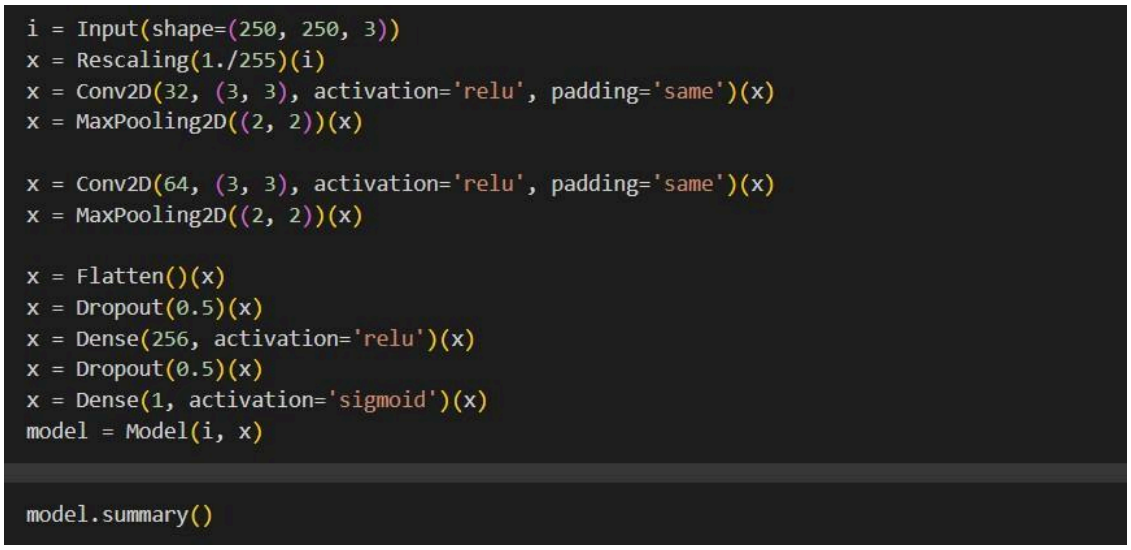




```python
for image_batch, labels_batch in train_val:
  print(image_batch.shape)
  print(labels_batch.shape)
  break

for image_batch, labels_batch in test_val:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

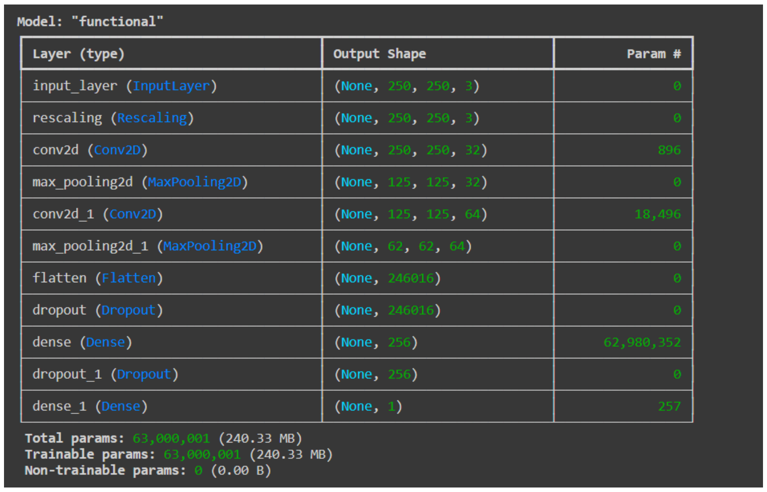
```
(32, 250, 250, 3)
(32,)
(32, 250, 250, 3)
(32,)
```

```python
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_val.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = test_val.cache().prefetch(buffer_size=AUTOTUNE)
```

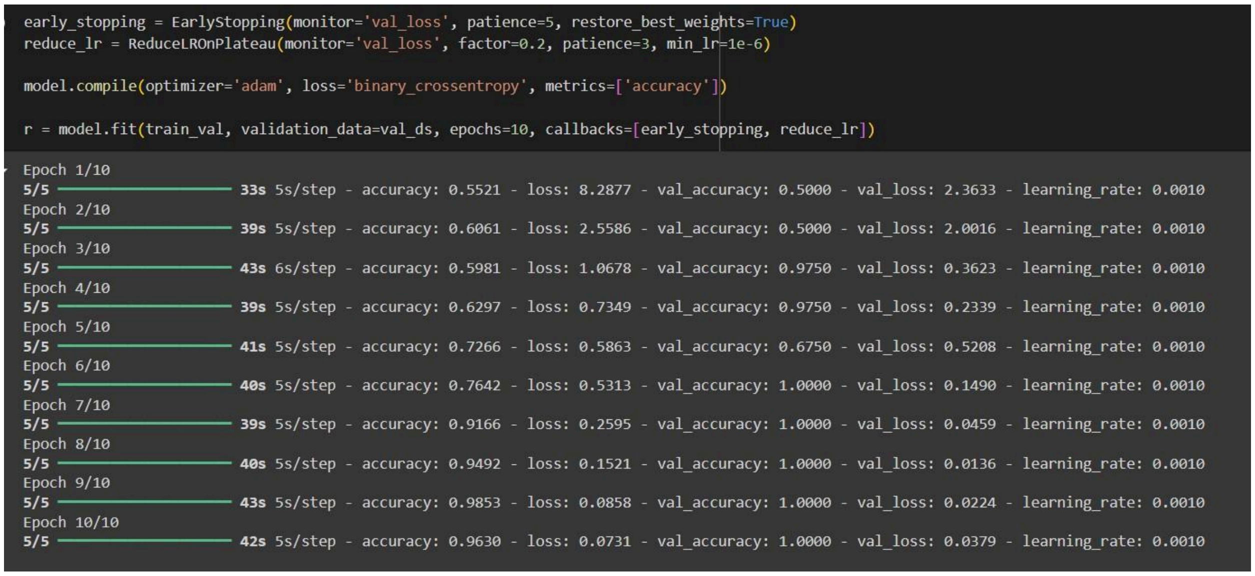


```python
i = Input(shape=(250, 250, 3))
x = Rescaling(1./255)(i)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(1, activation='sigmoid')(x)
model = Model(i, x)
```

```python
model.summary()
```

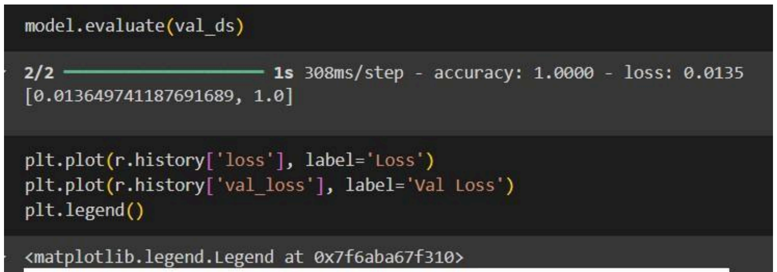


```
Model: "functional"
┌─────────────────────────────────┬────────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape               │       Param # │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ input_layer (InputLayer)        │ (None, 250, 250, 3)        │             0 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ rescaling (Rescaling)           │ (None, 250, 250, 3)        │             0 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ conv2d (Conv2D)                 │ (None, 250, 250, 32)       │           896 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 125, 125, 32)       │             0 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 125, 125, 64)       │        18,496 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 62, 62, 64)         │             0 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 246016)             │             0 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 246016)             │             0 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 256)                │    62,980,352 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 256)                │             0 │
├─────────────────────────────────┼────────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)                  │           257 │
└─────────────────────────────────┴────────────────────────────┴───────────────┘
 Total params: 63,000,001 (240.33 MB)
 Trainable params: 63,000,001 (240.33 MB)
 Non-trainable params: 0 (0.00 B)
```



```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6)

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

r = model.fit(train_val, validation_data=val_ds, epochs=10, callbacks=[early_stopping, reduce_lr])
```

```
Epoch 1/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 33s 5s/step - accuracy: 0.5521 - loss: 8.2877 - val_accuracy: 0.5000 - val_loss: 2.3633 - learning_rate: 0.0010
Epoch 2/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 39s 5s/step - accuracy: 0.6061 - loss: 2.5586 - val_accuracy: 0.5000 - val_loss: 2.0016 - learning_rate: 0.0010
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 43s 6s/step - accuracy: 0.5981 - loss: 1.0678 - val_accuracy: 0.9750 - val_loss: 0.3623 - learning_rate: 0.0010
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 39s 5s/step - accuracy: 0.6297 - loss: 0.7349 - val_accuracy: 0.9750 - val_loss: 0.2339 - learning_rate: 0.0010
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 41s 5s/step - accuracy: 0.7266 - loss: 0.5863 - val_accuracy: 0.6750 - val_loss: 0.5208 - learning_rate: 0.0010
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 40s 5s/step - accuracy: 0.7642 - loss: 0.5313 - val_accuracy: 1.0000 - val_loss: 0.1490 - learning_rate: 0.0010
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 39s 5s/step - accuracy: 0.9166 - loss: 0.2595 - val_accuracy: 1.0000 - val_loss: 0.0459 - learning_rate: 0.0010
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 40s 5s/step - accuracy: 0.9492 - loss: 0.1521 - val_accuracy: 1.0000 - val_loss: 0.0136 - learning_rate: 0.0010
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 43s 5s/step - accuracy: 0.9853 - loss: 0.0858 - val_accuracy: 1.0000 - val_loss: 0.0224 - learning_rate: 0.0010
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 42s 5s/step - accuracy: 0.9630 - loss: 0.0731 - val_accuracy: 1.0000 - val_loss: 0.0379 - learning_rate: 0.0010
```

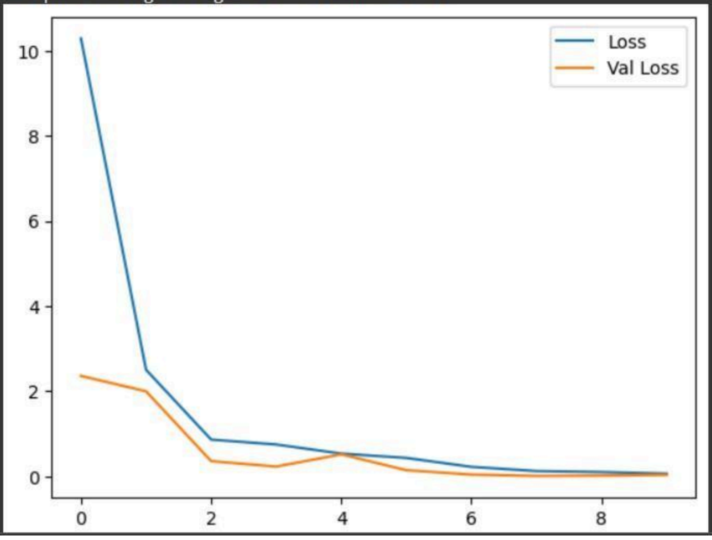


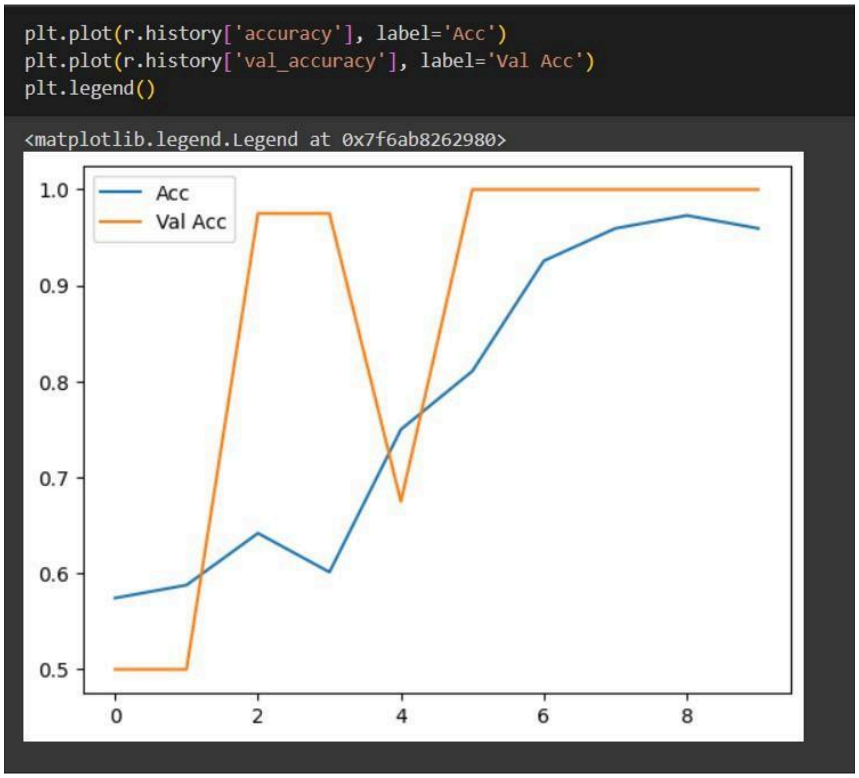
```
model.evaluate(val_ds)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 308ms/step - accuracy: 1.0000 - loss: 0.0135
[0.013649741187691689, 1.0]
```

```
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val Loss')
plt.legend()
```

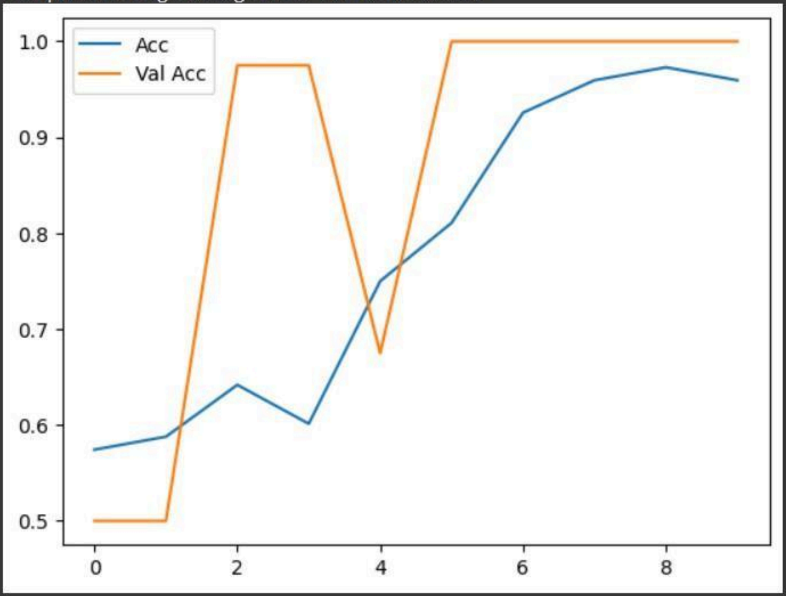
```
<matplotlib.legend.Legend at 0x7f6aba67f310>
```

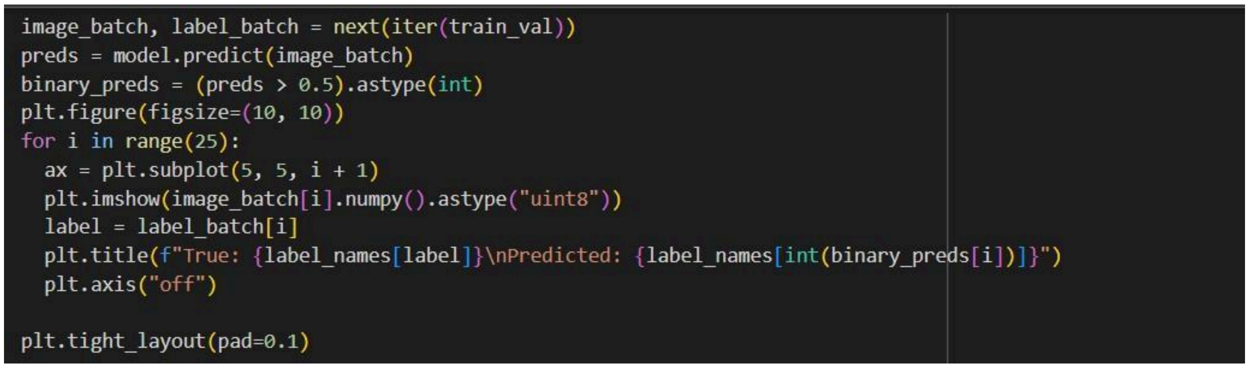




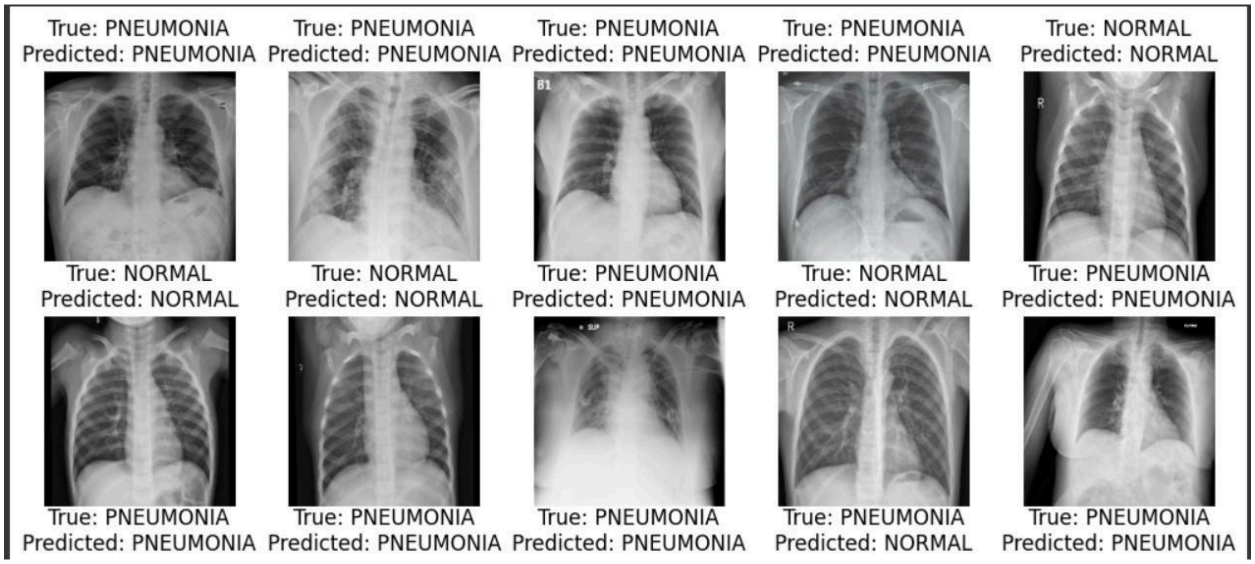
```
plt.plot(r.history['accuracy'], label='Acc')
plt.plot(r.history['val_accuracy'], label='Val Acc')
plt.legend()
```

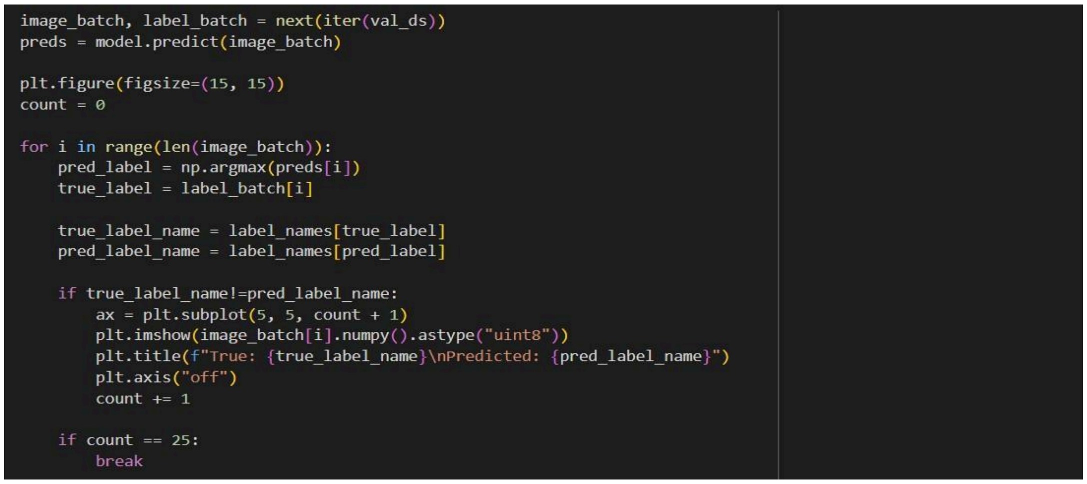
```
<matplotlib.legend.Legend at 0x7f6ab8262980>
```



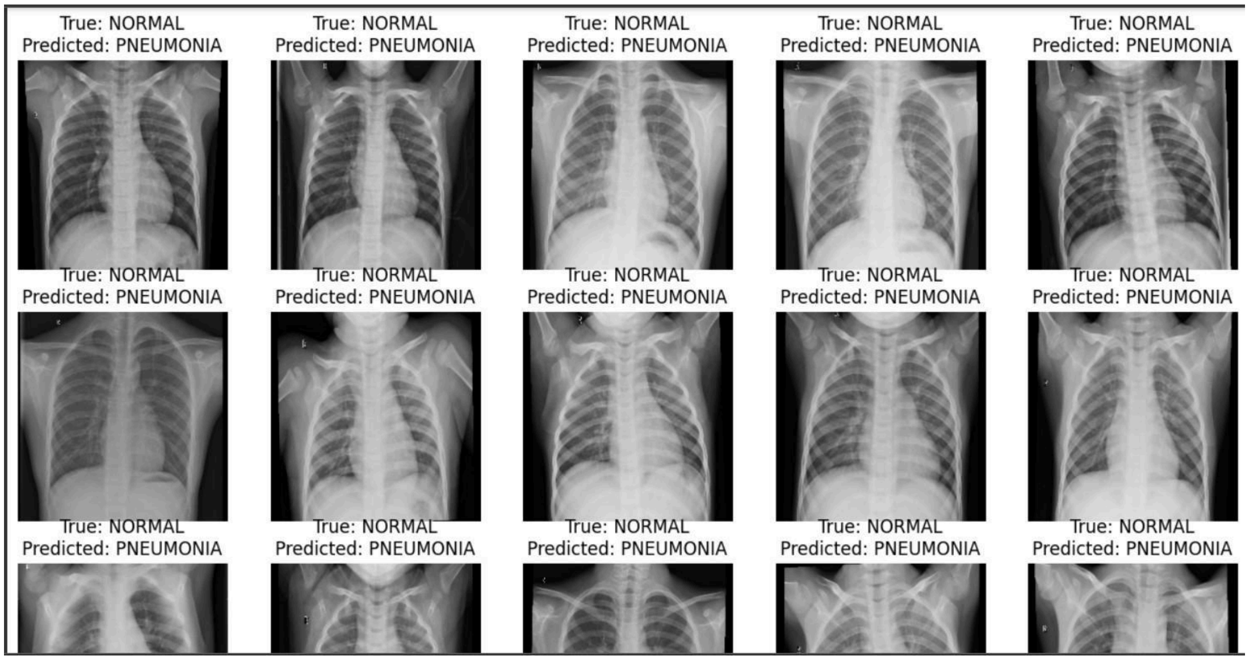


```
image_batch, label_batch = next(iter(train_val))
preds = model.predict(image_batch)
binary_preds = (preds > 0.5).astype(int)
plt.figure(figsize=(10, 10))
for i in range(25):
  ax = plt.subplot(5, 5, i + 1)
  plt.imshow(image_batch[i].numpy().astype("uint8"))
  label = label_batch[i]
  plt.title(f"True: {label_names[label]}\nPredicted: {label_names[int(binary_preds[i])]}")
  plt.axis("off")

plt.tight_layout(pad=0.1)
```





```
image_batch, label_batch = next(iter(val_ds))
preds = model.predict(image_batch)

plt.figure(figsize=(15, 15))
count = 0

for i in range(len(image_batch)):
    pred_label = np.argmax(preds[i])
    true_label = label_batch[i]

    true_label_name = label_names[true_label]
    pred_label_name = label_names[pred_label]

    if true_label_name!=pred_label_name:
        ax = plt.subplot(5, 5, count + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(f"True: {true_label_name}\nPredicted: {pred_label_name}")
        plt.axis("off")
        count += 1

    if count == 25:
        break
```



Conclusion:

Convolutional Neural Networks are one of the most effective methods for image processing tasks, learning intricate patterns through layers of filters, pooling, and dense connections. Their ability to generalize from image data has revolutionized fields like computer vision, medical imaging, and more.