```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
data_dir = "/content/drive/MyDrive/chest_xray 2"
```

```
!pip install tensorflow
```

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (2.0.2)
```

import required library

```
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout, BatchNormalization # Import MaxPooling2D from tensorflow.keras.layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import ReduceLROnPlateau
import cv2
import os
import numpy as np
import pandas as pd
```

```
import cv2
import os
import numpy as np

labels = ['PNEUMONIA', 'NORMAL']
img_size = 150

def get_training_data(data_dir):
    data = []
```

```
    for label in labels:  # Use the global 'labels' variable
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                # Ensure all images are resized to the same shape
                resized_arr = cv2.resize(img_arr, (img_size, img_size))
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)

    # Convert the list of images to a NumPy array with a consistent shape
    images = [item[0] for item in data]
    # Change the variable name for extracted labels to avoid conflict
    image_labels = [item[1] for item in data]

    # Reshape images to have a consistent channel dimension
    images = np.array(images)[:, :, :, np.newaxis]

    return images, np.array(image_labels)  # Return the new label variable
```

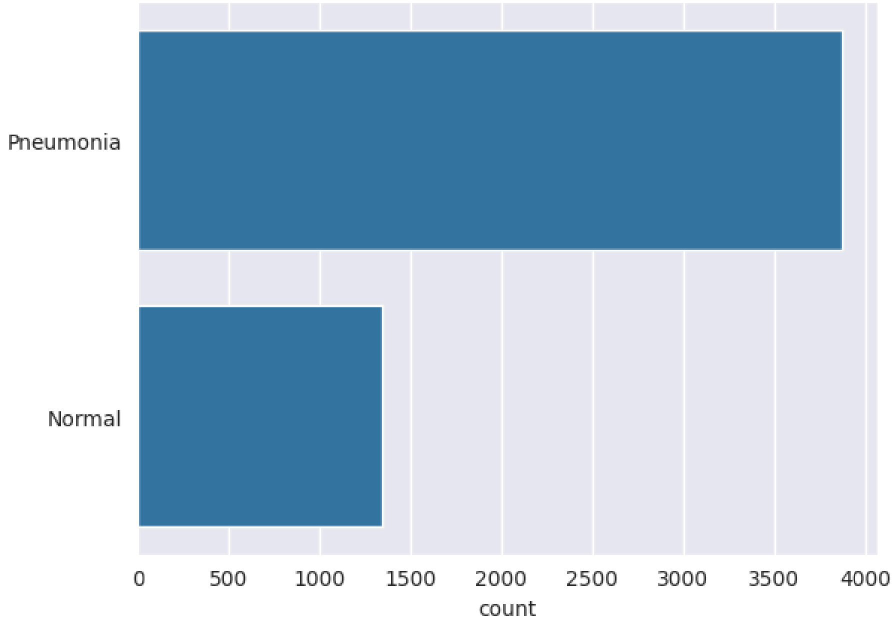
LOADING THE DATASET

```
train = get_training_data("/content/drive/MyDrive/chest_xray 2/train")
test = get_training_data("/content/drive/MyDrive/chest_xray 2/test")
val = get_training_data("/content/drive/MyDrive/chest_xray 2/val")
```

DATA VISUALIZATION & PREPROCESSING

```
l = []
for i in train[1]:  # Iterate through the labels in train[1]
    if i == 0:  # Check if the individual label is 0
        l.append("Pneumonia")
    else:
        l.append("Normal")
sns.set_style('darkgrid')
sns.countplot(l)
```

```
<Axes: xlabel='count'>
```

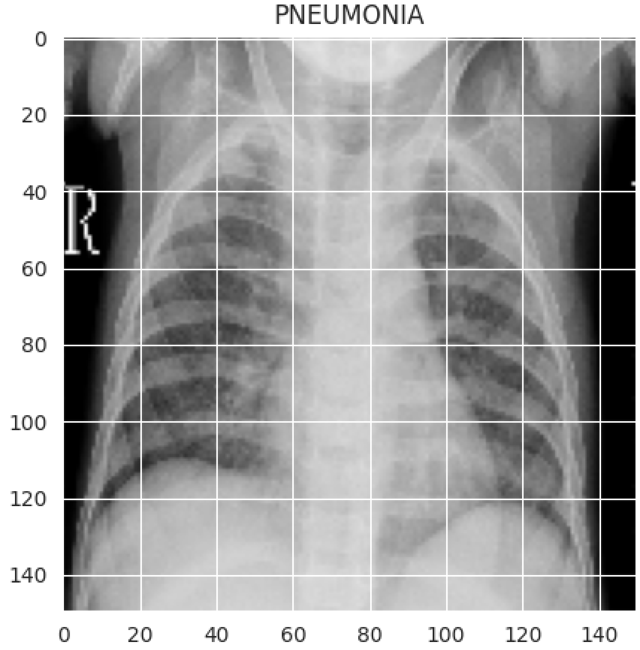


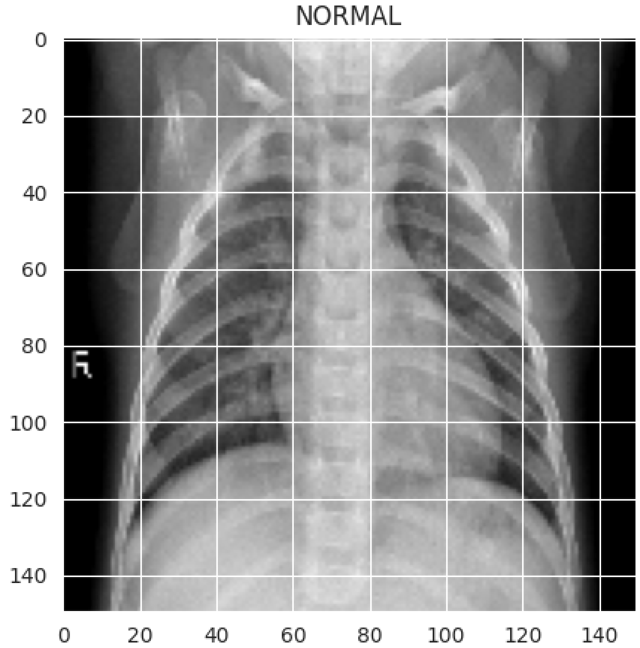
previewing the image

```
plt.figure(figsize = (5,5))
plt.imshow(train[0][0], cmap='gray')
plt.title(labels[train[1][0]]) # Access the label using train[1][0]

plt.figure(figsize = (5,5))
plt.imshow(train[0][-1], cmap='gray') # Access the image using train[0][-1]
plt.title(labels[train[1][-1]]) # Access the label using train[1][-1]
```

```
Text(0.5, 1.0, 'NORMAL')
```





```
x_train = train[0] # Access the images from train
y_train = train[1] # Access the labels from train

x_val = val[0] # Access the images from val
y_val = val[1] # Access the labels from val

x_test = test[0]  # Access the images from test
y_test = test[1]  # Access the labels from test


## normalize the data
x_train = np.array(x_train) / 255
x_val = np.array(x_val) / 255
x_test = np.array(x_test) / 255



## resize the data
x_train = x_train.reshape(-1, img_size, img_size, 1)
y_train = np.array(y_train)

x_val = x_val.reshape(-1, img_size, img_size, 1)
y_val = np.array(y_val)

x_test = x_test.reshape(-1, img_size, img_size, 1)
y_test = np.array(y_test)
```

data Augmentation

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip = True,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

TRAIN THE MODEL

```
model = Sequential()
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(256 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128 , activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1 , activation = 'sigmoid'))
model.compile(optimizer = "rmsprop" , loss = 'binary_crossentropy' , metrics = ['accuracy'])
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 32)	320
batch_normalization (BatchNormalization)	(None, 150, 150, 32)	128
max_pooling2d (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_1 (Conv2D)	(None, 75, 75, 64)	18,496
dropout (Dropout)	(None, 75, 75, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 75, 75, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 38, 38, 64)	0
conv2d_2 (Conv2D)	(None, 38, 38, 64)	36,928
batch_normalization_2 (BatchNormalization)	(None, 38, 38, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 19, 19, 64)	0
conv2d_3 (Conv2D)	(None, 19, 19, 128)	73,856
dropout_1 (Dropout)	(None, 19, 19, 128)	0
batch_normalization_3 (BatchNormalization)	(None, 19, 19, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 10, 10, 128)	0
conv2d_4 (Conv2D)	(None, 10, 10, 256)	295,168
dropout_2 (Dropout)	(None, 10, 10, 256)	0
batch_normalization_4 (BatchNormalization)	(None, 10, 10, 256)	1,024
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 256)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 128)	819,328
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 1,246,401 (4.75 MB)
Trainable params: 1,245,313 (4.75 MB)
Non-trainable params: 1,088 (4.25 KB)

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1,factor=0.3, min_lr=0.000001)
```

Start coding or generate with AI.

Start coding or generate with AI.

```
history = model.fit(datagen.flow(x_train,y_train, batch_size = 32) ,epochs =25, validation_data = datagen.flow(x_val, y_val) ,callbacks = [learning_rate_reduction]
```

```
Epoch 1/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 475s 3s/step - accuracy: 0.9694 - loss: 0.0948 - val_accuracy: 0.5625 - val_loss: 1.5085 - learning_rate: 9.0000e-05
Epoch 2/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 462s 3s/step - accuracy: 0.9668 - loss: 0.1000 - val_accuracy: 0.5625 - val_loss: 0.9743 - learning_rate: 9.0000e-05
Epoch 3/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 467s 3s/step - accuracy: 0.9691 - loss: 0.0953 - val_accuracy: 1.0000 - val_loss: 0.1724 - learning_rate: 9.0000e-05
Epoch 4/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 461s 3s/step - accuracy: 0.9637 - loss: 0.0968 - val_accuracy: 0.6875 - val_loss: 0.7948 - learning_rate: 9.0000e-05
Epoch 5/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9666 - loss: 0.0921
Epoch 5: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
163/163 ━━━━━━━━━━━━━━━━━━━━ 467s 3s/step - accuracy: 0.9666 - loss: 0.0921 - val_accuracy: 0.5625 - val_loss: 2.0688 - learning_rate: 9.0000e-05
Epoch 6/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 468s 3s/step - accuracy: 0.9747 - loss: 0.0799 - val_accuracy: 0.5625 - val_loss: 1.8878 - learning_rate: 2.7000e-05
Epoch 7/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9737 - loss: 0.0754
Epoch 7: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
163/163 ━━━━━━━━━━━━━━━━━━━━ 510s 3s/step - accuracy: 0.9737 - loss: 0.0755 - val_accuracy: 0.6875 - val_loss: 0.8649 - learning_rate: 2.7000e-05
Epoch 8/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 469s 3s/step - accuracy: 0.9673 - loss: 0.0949 - val_accuracy: 0.5625 - val_loss: 1.0631 - learning_rate: 8.1000e-06
Epoch 9/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9691 - loss: 0.0885
Epoch 9: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.
163/163 ━━━━━━━━━━━━━━━━━━━━ 470s 3s/step - accuracy: 0.9691 - loss: 0.0884 - val_accuracy: 0.6250 - val_loss: 1.3179 - learning_rate: 8.1000e-06
Epoch 10/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 470s 3s/step - accuracy: 0.9666 - loss: 0.0914 - val_accuracy: 0.6250 - val_loss: 1.4770 - learning_rate: 2.4300e-06
Epoch 11/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9686 - loss: 0.0819
Epoch 11: ReduceLROnPlateau reducing learning rate to 1e-06.
163/163 ━━━━━━━━━━━━━━━━━━━━ 468s 3s/step - accuracy: 0.9686 - loss: 0.0819 - val_accuracy: 0.6250 - val_loss: 1.4073 - learning_rate: 2.4300e-06
Epoch 12/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 463s 3s/step - accuracy: 0.9657 - loss: 0.0976 - val_accuracy: 0.6250 - val_loss: 1.1731 - learning_rate: 1.0000e-06
Epoch 13/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 504s 3s/step - accuracy: 0.9660 - loss: 0.0921 - val_accuracy: 0.6875 - val_loss: 0.9856 - learning_rate: 1.0000e-06
Epoch 14/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 457s 3s/step - accuracy: 0.9726 - loss: 0.0894 - val_accuracy: 0.7500 - val_loss: 0.7758 - learning_rate: 1.0000e-06
Epoch 15/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 452s 3s/step - accuracy: 0.9671 - loss: 0.0921 - val_accuracy: 0.5625 - val_loss: 1.2353 - learning_rate: 1.0000e-06
Epoch 16/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 454s 3s/step - accuracy: 0.9730 - loss: 0.0796 - val_accuracy: 0.6875 - val_loss: 0.9578 - learning_rate: 1.0000e-06
```

```
Epoch 17/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 465s 3s/step - accuracy: 0.9731 - loss: 0.0794 - val_accuracy: 0.5625 - val_loss: 1.2103 - learning_rate: 1.0000e-06
Epoch 18/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 497s 3s/step - accuracy: 0.9716 - loss: 0.0859 - val_accuracy: 0.6875 - val_loss: 0.6033 - learning_rate: 1.0000e-06
Epoch 19/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 511s 3s/step - accuracy: 0.9727 - loss: 0.0782 - val_accuracy: 0.6250 - val_loss: 1.1148 - learning_rate: 1.0000e-06
Epoch 20/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 469s 3s/step - accuracy: 0.9762 - loss: 0.0695 - val_accuracy: 0.6250 - val_loss: 0.9790 - learning_rate: 1.0000e-06
Epoch 21/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 467s 3s/step - accuracy: 0.9698 - loss: 0.0772 - val_accuracy: 0.5625 - val_loss: 1.0537 - learning_rate: 1.0000e-06
Epoch 22/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 502s 3s/step - accuracy: 0.9661 - loss: 0.0989 - val_accuracy: 0.6875 - val_loss: 1.0839 - learning_rate: 1.0000e-06
Epoch 23/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 468s 3s/step - accuracy: 0.9701 - loss: 0.0858 - val_accuracy: 0.6250 - val_loss: 0.7873 - learning_rate: 1.0000e-06
Epoch 24/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 466s 3s/step - accuracy: 0.9684 - loss: 0.0865 - val_accuracy: 0.6875 - val_loss: 1.0272 - learning_rate: 1.0000e-06
Epoch 25/25
163/163 ━━━━━━━━━━━━━━━━━━━━ 471s 3s/step - accuracy: 0.9709 - loss: 0.0825 - val_accuracy: 0.8125 - val_loss: 0.7226 - learning_rate: 1.0000e-06
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define a CNN Model with Dropout
# Changed input_shape to match the actual input data shape (150, 150, 1)
model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 1)),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),  # Dropout after the first pooling layer

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),  # Dropout after the second pooling layer

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Dropout before the final dense layer
    layers.Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

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

# Now, when you call model.fit(), make sure 'train' and 'val' data
# have the correct shape (num_samples, 150, 150, 1).

history = model.fit(train[0], train[1], validation_data=(val[0], val[1]), epochs=20) # Pass train[0] for images and train[1] for labels
```

```
Epoch 1/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - accuracy: 0.7057 - loss: 105.0736 - val_accuracy: 0.6875 - val_loss: 0.8636
Epoch 2/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - accuracy: 0.8727 - loss: 0.3102 - val_accuracy: 0.7500 - val_loss: 0.9732
Epoch 3/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.8861 - loss: 0.2771 - val_accuracy: 0.7500 - val_loss: 1.2621
Epoch 4/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - accuracy: 0.8995 - loss: 0.2554 - val_accuracy: 0.7500 - val_loss: 1.1577
Epoch 5/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - accuracy: 0.9218 - loss: 0.2086 - val_accuracy: 0.7500 - val_loss: 1.2889
Epoch 6/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9159 - loss: 0.2133 - val_accuracy: 0.7500 - val_loss: 1.2214
Epoch 7/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9302 - loss: 0.1846 - val_accuracy: 0.6875 - val_loss: 1.2817
Epoch 8/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9402 - loss: 0.1537 - val_accuracy: 0.7500 - val_loss: 1.1582
Epoch 9/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9506 - loss: 0.1255 - val_accuracy: 0.7500 - val_loss: 1.7910
Epoch 10/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - accuracy: 0.9542 - loss: 0.1339 - val_accuracy: 0.8125 - val_loss: 1.6373
Epoch 11/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - accuracy: 0.9507 - loss: 0.1278 - val_accuracy: 0.8125 - val_loss: 2.1183
Epoch 12/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - accuracy: 0.9727 - loss: 0.0802 - val_accuracy: 0.7500 - val_loss: 1.6384
Epoch 13/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9691 - loss: 0.0781 - val_accuracy: 0.8125 - val_loss: 1.8179
Epoch 14/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9748 - loss: 0.0714 - val_accuracy: 0.7500 - val_loss: 1.8844
Epoch 15/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - accuracy: 0.9788 - loss: 0.0624 - val_accuracy: 0.8125 - val_loss: 1.9619
Epoch 16/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 18ms/step - accuracy: 0.9790 - loss: 0.0632 - val_accuracy: 0.7500 - val_loss: 1.9582
Epoch 17/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9772 - loss: 0.0600 - val_accuracy: 0.8125 - val_loss: 2.5408
Epoch 18/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9845 - loss: 0.0436 - val_accuracy: 0.7500 - val_loss: 2.8651
Epoch 19/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - accuracy: 0.9801 - loss: 0.0531 - val_accuracy: 0.7500 - val_loss: 3.4313
Epoch 20/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9823 - loss: 0.0500 - val_accuracy: 0.8125 - val_loss: 2.9960
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

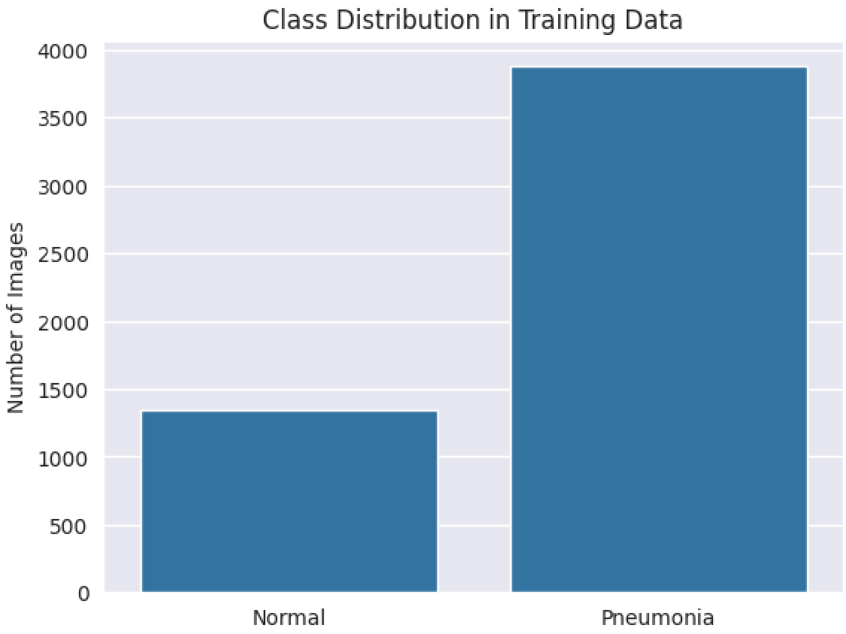
```
import os
train_normal = len(os.listdir('/content/drive/MyDrive/chest_xray 2/train/NORMAL'))
train_pneumonia = len(os.listdir('/content/drive/MyDrive/chest_xray 2/train/PNEUMONIA'))
print(f"Normal: {train_normal}, Pneumonia: {train_pneumonia}")
```

```
Normal: 1341, Pneumonia: 3875
```

```
## import required library
import os
import matplotlib.pyplot as plt
import seaborn as sns
print(f"Normal: {train_normal}, Pneumonia: {train_pneumonia}")

# Plot class distribution
sns.barplot(x=['Normal', 'Pneumonia'], y=[train_normal, train_pneumonia])
plt.title("Class Distribution in Training Data")
plt.ylabel("Number of Images")
plt.show()
```

```
Normal: 1341, Pneumonia: 3875
```




```
## we need to balanced the data by using data augumentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalize pixel values
    rotation_range=20,  # Randomly rotate images
    width_shift_range=0.2,  # Horizontal shift
    height_shift_range=0.2,  # Vertical shift
    shear_range=0.2,  # Shear transformation
    zoom_range=0.2,  # Random zoom
    horizontal_flip=True,  # Flip images
    fill_mode='nearest',  # Fill missing pixels
    validation_split=0.2  # Split for validation
)

test_datagen = ImageDataGenerator(rescale=1./255)  # No augmentation for test set

# Load Train Dataset
train_generator = train_datagen.flow_from_directory(
    os.path.join(data_dir, 'train'),
    target_size=(150, 150),  # Resize images
    batch_size=32,
    class_mode='binary',  # Binary classification
```

```
    subset='training'  # Training subset
)

# Load Validation Dataset
validation_generator = train_datagen.flow_from_directory(
    os.path.join(data_dir, 'train'),
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation'  # Validation subset
)

# Load Test Dataset
test_generator = test_datagen.flow_from_directory(
    os.path.join(data_dir, 'test'),
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 4173 images belonging to 2 classes.
Found 1043 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Define class labels
class_labels = ['NORMAL', 'PNEUMONIA']
class_counts = [train_normal,train_pneumonia]

# Compute class weights
weights = compute_class_weight(class_weight="balanced", classes=np.array([0, 1]), y=[0]*train_normal + [1]*train_pneumonia)
class_weights = {0: weights[0], 1: weights[1]}

print(f"Class Weights: {class_weights}")
```

```
Class Weights: {0: np.float64(1.9448173005219984), 1: np.float64(0.6730322580645162)}
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# CNN Model Architecture
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2,2),
```

```
        Conv2D(64, (3,3), activation='relu'),
        MaxPooling2D(2,2),

        Conv2D(128, (3,3), activation='relu'),
        MaxPooling2D(2,2),

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')  # Binary Classification (Normal/Pneumonia)
    ])

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

    # Train Model with Class Weights
    history = model.fit(train_generator,
                        validation_data=validation_generator,
                        epochs=10,
                        class_weight=class_weights)  # Apply class weights
```

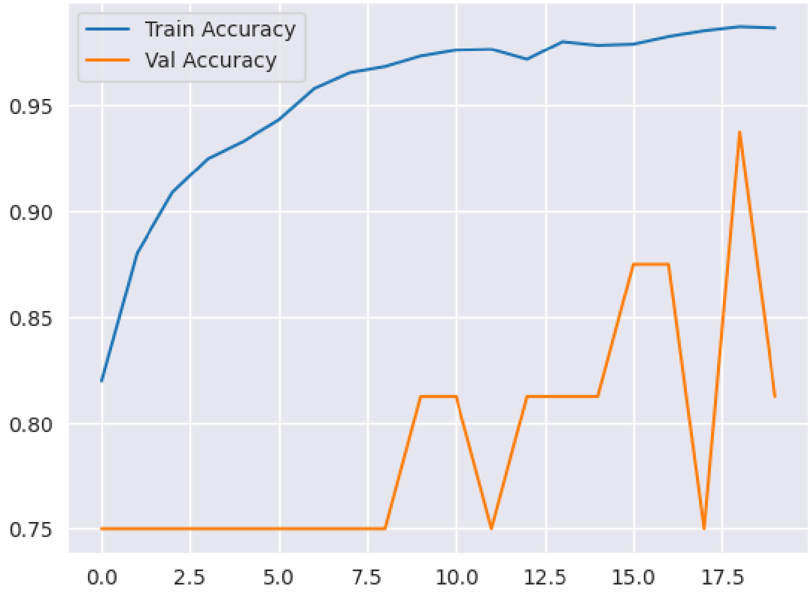
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__
  self._warn_if_super_not_called()
Epoch 1/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 95s 692ms/step - accuracy: 0.4555 - loss: 0.7328 - val_accuracy: 0.7239 - val_loss: 0.5847
Epoch 2/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 83s 638ms/step - accuracy: 0.8123 - loss: 0.4124 - val_accuracy: 0.8102 - val_loss: 0.3906
Epoch 3/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 88s 675ms/step - accuracy: 0.8460 - loss: 0.3513 - val_accuracy: 0.8159 - val_loss: 0.3866
Epoch 4/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 84s 642ms/step - accuracy: 0.8395 - loss: 0.3697 - val_accuracy: 0.8293 - val_loss: 0.3933
Epoch 5/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 86s 661ms/step - accuracy: 0.8664 - loss: 0.3099 - val_accuracy: 0.8207 - val_loss: 0.4286
Epoch 6/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 85s 645ms/step - accuracy: 0.8817 - loss: 0.3001 - val_accuracy: 0.8428 - val_loss: 0.3584
Epoch 7/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 84s 646ms/step - accuracy: 0.8845 - loss: 0.2771 - val_accuracy: 0.8821 - val_loss: 0.2613
Epoch 8/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 88s 676ms/step - accuracy: 0.8838 - loss: 0.2696 - val_accuracy: 0.8830 - val_loss: 0.2784
Epoch 9/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 84s 641ms/step - accuracy: 0.8876 - loss: 0.2575 - val_accuracy: 0.8974 - val_loss: 0.2486
Epoch 10/10
131/131 ━━━━━━━━━━━━━━━━━━━━ 85s 652ms/step - accuracy: 0.8904 - loss: 0.2852 - val_accuracy: 0.8869 - val_loss: 0.2621
```

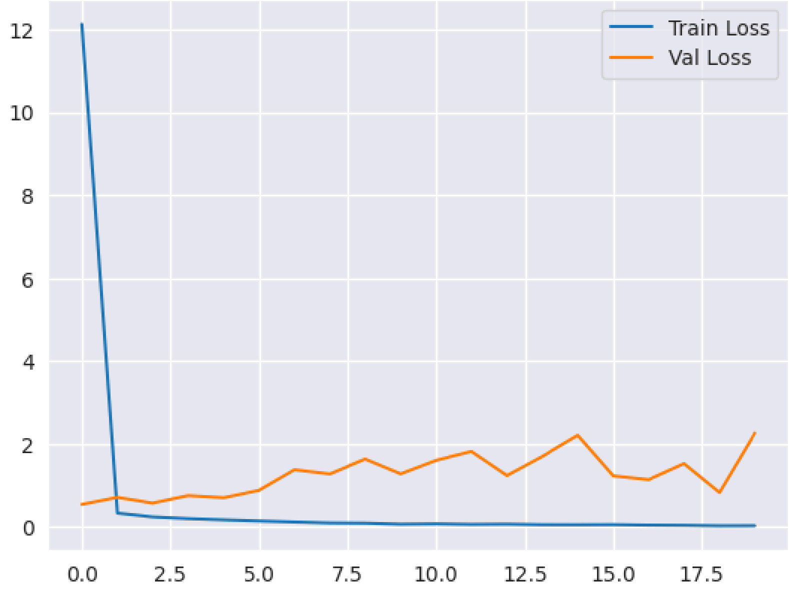
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.legend()
plt.show()

# Plot training & validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend()
plt.show()
```






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

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
```

```
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(train_generator,
                    validation_data=validation_generator,
                    epochs=20,
                    class_weight=class_weights,
                    callbacks=[early_stop])
```

```
Epoch 1/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 87s 663ms/step - accuracy: 0.9047 - loss: 0.2143 - val_accuracy: 0.8658 - val_loss: 0.3143
Epoch 2/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 84s 644ms/step - accuracy: 0.8928 - loss: 0.2502 - val_accuracy: 0.8878 - val_loss: 0.2675
Epoch 3/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 88s 672ms/step - accuracy: 0.8992 - loss: 0.2296 - val_accuracy: 0.9070 - val_loss: 0.2154
Epoch 4/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 84s 641ms/step - accuracy: 0.8953 - loss: 0.2447 - val_accuracy: 0.8629 - val_loss: 0.3196
Epoch 5/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 84s 646ms/step - accuracy: 0.8981 - loss: 0.2331 - val_accuracy: 0.8840 - val_loss: 0.2547
Epoch 6/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 84s 641ms/step - accuracy: 0.9095 - loss: 0.2167 - val_accuracy: 0.9137 - val_loss: 0.2224
```

```
from google.colab import drive
drive.mount('/content/drive')

# Save model to Google Drive
model.save('/content/drive/My Drive/pneumonia_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recomm
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
class_weights = {0: 1.0, 1: 3.0}  # Increase PNEUMONIA weight
```

```
from tensorflow.keras.models import load_model

# Load the trained model
```

```
model = load_model('/content/drive/MyDrive/pneumonia_model.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the mc
```

check the model on new data

```
from google.colab import files

# Upload an image
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving IM-0025-0001.jpeg to IM-0025-0001 (1).jpeg
```

```
class_weights = {0: 1.0, 1: 3.0}  # Increase PNEUMONIA weight
```

```
import numpy as np
import cv2
from tensorflow.keras.preprocessing import image

# Load and preprocess the uploaded image
img_path = list(uploaded.keys())[0]  # Get uploaded filename
# Resize to match the input shape your model was trained on (e.g., 150x150)
img = image.load_img(img_path, target_size=(150, 150))

# Convert image to array and normalize
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension


# Make prediction
prediction = model.predict(img_array)

# Get class labels (assuming you used 0 for 'NORMAL' and 1 for 'PNEUMONIA')
class_labels = ['NORMAL', 'PNEUMONIA']
predicted_class = class_labels[np.argmax(prediction)]

# Print the result
print(f"The model predicts: {predicted_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
The model predicts: NORMAL
```

```
    img = image.load_img(img_path, target_size=(224, 224))  # Match model input size
    img_array = image.img_to_array(img) / 255.0  # Normalize (important!)
```