# import Libraries

In [1]:
```
import tensorflow as tf
import keras
from keras.layers import Conv2D,MaxPooling2D,Dense,Dropout,Flatten
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras.utils import to_categorical
import os
from tqdm import tqdm
import cv2
import random
import pickle as pk
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from keras.models import load_model
from keras.layers import BatchNormalization

from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.callbacks import EarlyStopping
from keras import regularizers

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import keras
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [2]:
```
#define constants
imges_num=5000
class_num=10
FAST_RUN = False
IMAGE_WIDTH=220
IMAGE_HEIGHT=220
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS=3
```

In [3]:
```
# training

trainingPath=r"C:\Users\kirti\OneDrive\Desktop\MLPA\chest xray\chest_xray\train"
categories=os.listdir(trainingPath)
categories
```

Out[3]:
```
['NORMAL', 'PNEUMONIA']
```

In [4]:
```
#save class names in dictionary

classnames={}
for i in range(2):
  classnames[i]=categories[i]
classnames
```

Out[4]: {0: 'NORMAL', 1: 'PNEUMONIA'}

In [5]:
```
#load the images and read it and save eacah image with folder name

training =[]
x=0
for category in categories:
    folder=os.path.join(trainingPath,category)
    # print(folder)
    x2=0
    for file in tqdm(os.listdir(folder)):
        # print(file)
        try:
            img=cv2.imread(os.path.join(folder,file))
            img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
            img=cv2.resize(img,IMAGE_SIZE)# size image for CNN input
            classNum=categories.index(category) #detrmine class
            training.append([img,classNum])
            # plt.imshow(img)
            # plt.show()
        except Exception as e:
            pass

        if x2>=imges_num:
            break
        x2+=1

    if x>=class_num-1:
        break
    x+=1
```

```
100%|██████████| 1341/1341 [00:45<00:00, 29.36it/s]
100%|██████████| 3875/3875 [00:35<00:00, 110.06it/s]
```

In [6]:
```
len(training)
```

Out[6]: 5216

In [7]:
```
#shuffle my data to avoid overfitting

training[0]
random.shuffle(training)
for trainSample in training[:10]:
    print(trainSample[1])
```

```
1
0
0
1
1
1
1
1
1
1
```

In [8]:
```
#split features and labels

x_train_data=[]
y_train_data=[]
```

```
for feature,label in training:
    x_train_data.append(feature)
    y_train_data.append(label)
```

In [9]:
```
#convert my data to nunpy array

x_train_data=np.array(x_train_data)
y_train_data=np.array(y_train_data)
```

In [10]:
```
#reshape data to RGB form #3 channels with 220*220

x_train_data=x_train_data.reshape(-1,IMAGE_HEIGHT,IMAGE_WIDTH,IMAGE_CHANNELS)
print(x_train_data.shape)
print(y_train_data.shape)
```

```
(5216, 220, 220, 3)
(5216,)
```

# test data

In [11]:
```
testPath=r"C:\Users\kirti\OneDrive\Desktop\MLPA\chest xray\chest_xray\test"
categories=os.listdir(trainingPath)
categories
```

Out[11]:
```
['NORMAL', 'PNEUMONIA']
```

In [12]:
```
#load the images and read it and save eacah image with folder name [class name]

test =[]
x=0

for category in categories:
    folder=os.path.join(testPath,category)
    # print(folder)
    x2=0
    for file in tqdm(os.listdir(folder)):
        # print(file)
        try:
            img=cv2.imread(os.path.join(folder,file))
            img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
            img=cv2.resize(img,IMAGE_SIZE)# size image for CNN input
            classNum=categories.index(category) #detrmine class
            test.append([img,classNum])
            # plt.imshow(img)
            # plt.show()
        except Exception as e:
            pass

        if x2>imges_num:
            break
        x2+=1

    if x>=class_num-1:
        break
    x+=1
    # print(x)
```

```
100%|██████████| 234/234 [00:05<00:00, 44.30it/s]
100%|██████████| 390/390 [00:03<00:00, 107.70it/s]
```

In [13]:
```
#shuffle my data to avoid overfitting

random.shuffle(test)
for trainSample in test[:10]:
    print(trainSample[1])
```

```
1
1
1
1
0
1
1
0
1
1
```

In [14]:
```
#split features and labels

x_test=[]
y_test=[]
for feature,label in test:
    x_test.append(feature)
    y_test.append(label)
```

In [15]:
```
#convert my data to nunpy array

x_test=np.array(x_test)
y_test=np.array(y_test)
```

In [16]:
```
#reshape data to RGB form 3 channels with 220*220

x_test=x_test.reshape(-1,IMAGE_HEIGHT,IMAGE_WIDTH,IMAGE_CHANNELS)

print(x_test.shape)
print(y_test.shape)
```

```
(624, 220, 220, 3)
(624,)
```

# split data

In [17]:
```
#split the train data into train and validation

# x_train=x_train_data
# y_train=y_train_data

x_train,x_validation_data,y_train,y_validation_data=train_test_split(x_train_dat
print(x_train.shape)
print(y_train.shape)

print(x_validation_data.shape)
print(y_validation_data.shape)
```

```
print(x_test.shape)
print(y_test.shape)
```

```
(4433, 220, 220, 3)
(4433,)
(783, 220, 220, 3)
(783,)
(624, 220, 220, 3)
(624,)
```

# shapes of data

In [18]:
```
print(f'x_train shape :         {x_train.shape}')
print(f'y_train shape :         {y_train.shape}\n')

print(f'x_validation shape :    {x_validation_data.shape}')
print(f'y_validation shape :    {y_validation_data.shape}\n')

print(f'x_test shape :          {x_test.shape}')
print(f'x_test shape :          {y_test.shape}')
```

```
x_train shape :         (4433, 220, 220, 3)
y_train shape :         (4433,)

x_validation shape :    (783, 220, 220, 3)
y_validation shape :    (783,)

x_test shape :          (624, 220, 220, 3)
x_test shape :          (624,)
```

# Build Model

In [19]:
```
model=keras.models.Sequential()

model.add(Conv2D(16,(3,3),input_shape=(IMAGE_WIDTH,IMAGE_HEIGHT,IMAGE_CHANNELS),
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(64,(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(1,activation='sigmoid'))
```

In [20]:
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

In [21]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 218, 218, 16)      448

 batch_normalization (BatchN  (None, 218, 218, 16)     64
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 109, 109, 16)     0
 )

 dropout (Dropout)           (None, 109, 109, 16)      0

 conv2d_1 (Conv2D)           (None, 107, 107, 64)      9280

 batch_normalization_1 (Batc  (None, 107, 107, 64)     256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 53, 53, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 53, 53, 64)        0

 flatten (Flatten)           (None, 179776)            0

 dense (Dense)               (None, 256)               46022912

 batch_normalization_2 (Batc  (None, 256)              1024
 hNormalization)

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 46,034,241
Trainable params: 46,033,569
Non-trainable params: 672
_________________________________________________________________
```

In [22]:
```
history=model.fit(x_train,y_train,epochs=10,validation_data=(x_validation_data,
```

```
Epoch 1/10
139/139 [==============================] - 95s 650ms/step - loss: 0.2524 - accura
cy: 0.9238 - val_loss: 0.3720 - val_accuracy: 0.8442
Epoch 2/10
139/139 [==============================] - 80s 574ms/step - loss: 0.0882 - accura
cy: 0.9707 - val_loss: 0.1871 - val_accuracy: 0.9361
Epoch 3/10
139/139 [==============================] - 77s 552ms/step - loss: 0.0660 - accura
cy: 0.9765 - val_loss: 0.1281 - val_accuracy: 0.9553
Epoch 4/10
139/139 [==============================] - 81s 582ms/step - loss: 0.0501 - accura
cy: 0.9817 - val_loss: 2.3034 - val_accuracy: 0.6833
Epoch 5/10
139/139 [==============================] - 87s 627ms/step - loss: 0.0401 - accura
cy: 0.9860 - val_loss: 11.1669 - val_accuracy: 0.7356
Epoch 6/10
139/139 [==============================] - 81s 585ms/step - loss: 0.0393 - accura
cy: 0.9880 - val_loss: 0.2373 - val_accuracy: 0.9208
Epoch 7/10
139/139 [==============================] - 83s 599ms/step - loss: 0.0187 - accura
cy: 0.9953 - val_loss: 0.3655 - val_accuracy: 0.8902
Epoch 8/10
139/139 [==============================] - 86s 617ms/step - loss: 0.0214 - accura
cy: 0.9950 - val_loss: 561.4347 - val_accuracy: 0.7331
Epoch 9/10
139/139 [==============================] - 88s 636ms/step - loss: 0.0218 - accura
cy: 0.9939 - val_loss: 0.3893 - val_accuracy: 0.8966
Epoch 10/10
139/139 [==============================] - 88s 634ms/step - loss: 0.0222 - accura
cy: 0.9937 - val_loss: 0.9791 - val_accuracy: 0.8046
```

In [23]:
```
# Saving the Model
model.save('chest_xray_model.h5')
```

In [24]:
```
model = load_model('chest_xray_model.h5')
```

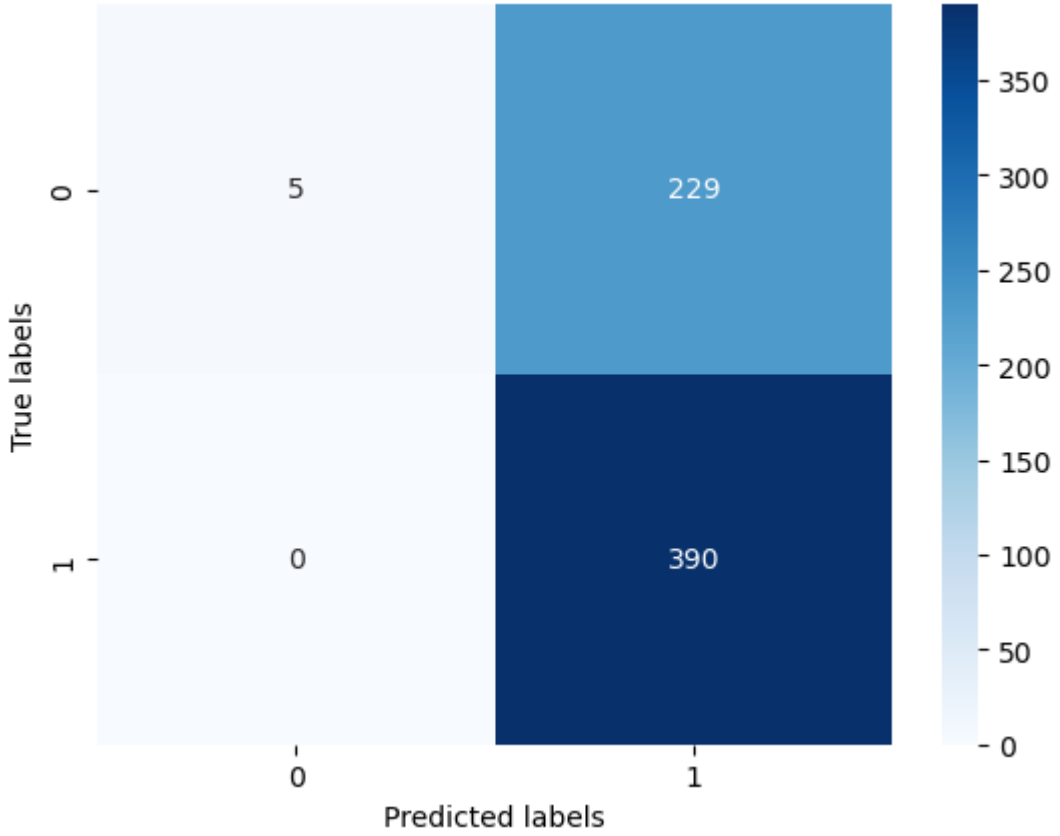
# Confusion Matrix

In [25]:
```
# Generate predictions on test data
y_pred = model.predict(x_test)
y_pred_binary = np.squeeze(np.round(y_pred))
# Generate confusion matrix
confusion_Matrix = confusion_matrix(y_test, y_pred_binary)
confusion_Matrix
```

```
20/20 [==============================] - 4s 154ms/step
```

Out[25]:
```
array([[  5, 229],
       [  0, 390]], dtype=int64)
```

In [26]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot confusion matrix as heatmap
sns.heatmap(confusion_Matrix, annot=True, cmap='Blues',fmt='.3g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

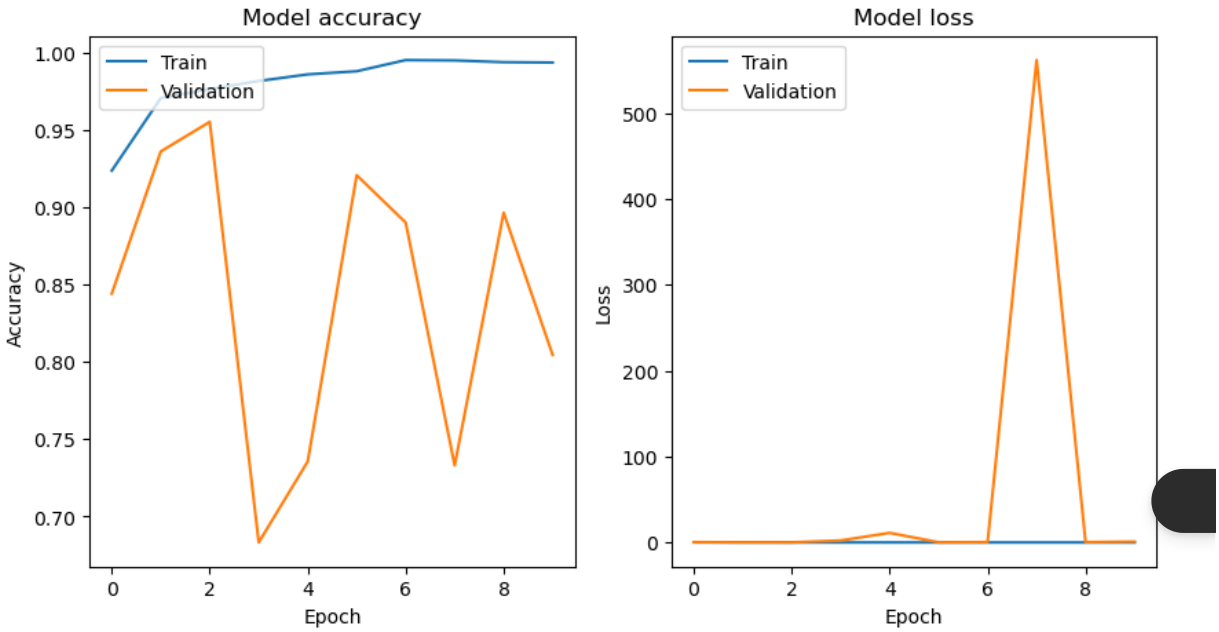



# graph between the training and validation accuracy and loss

In [27]:
```
history
```

Out[27]:
```
<keras.callbacks.History at 0x1c600262d90>
```

In [28]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
# Plot the training and validation accuracy
ax1.plot(history.history['accuracy'])
ax1.plot(history.history['val_accuracy'])
ax1.set_title('Model accuracy')
ax1.set_ylabel('Accuracy')
ax1.set_xlabel('Epoch')
ax1.legend(['Train', 'Validation'], loc='upper left')

# Plot the training and validation loss
ax2.plot(history.history['loss'])
ax2.plot(history.history['val_loss'])
ax2.set_title('Model loss')
ax2.set_ylabel('Loss')
ax2.set_xlabel('Epoch')
ax2.legend(['Train', 'Validation'], loc='upper left')

# Display the plots
plt.show()
```




# Classification Report

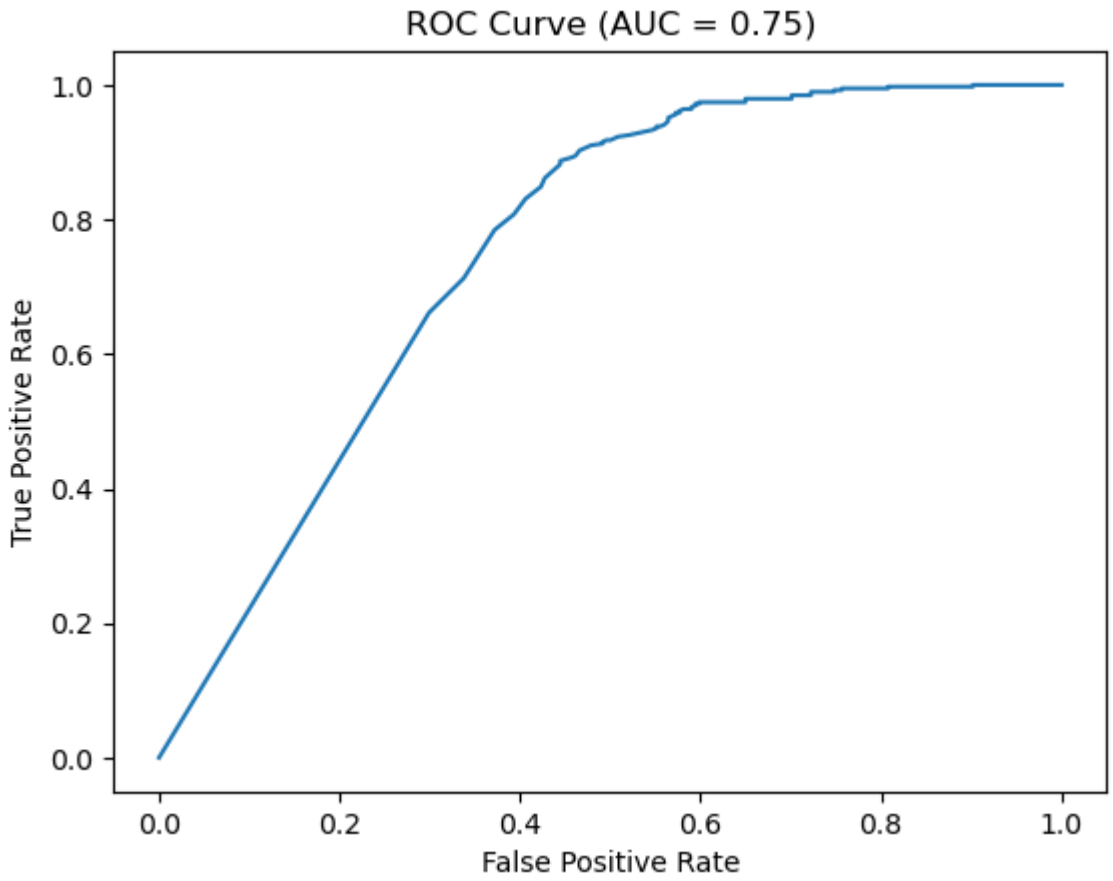
In [29]:
```
y_pred_binary = np.squeeze(np.round(y_pred))
print(classification_report(y_test, y_pred_binary))
```

```
              precision    recall  f1-score   support

           0       1.00      0.02      0.04       234
           1       0.63      1.00      0.77       390

    accuracy                           0.63       624
   macro avg       0.82      0.51      0.41       624
weighted avg       0.77      0.63      0.50       624
```

# AUC curve for binary classification

In [30]:
```
from sklearn.metrics import roc_curve, roc_auc_score

# Get the predicted probabilities for the positive class
y_prob = model.predict(x_test)

# Compute the FPR, TPR, and thresholds for various classification thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Compute the AUC score
auc_score = roc_auc_score(y_test, y_prob)

# Plot the ROC curve
import matplotlib.pyplot as plt
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve (AUC = {auc_score:.2f})')
plt.show()
```

```
20/20 [==============================] - 3s 132ms/step
```




# Save the model

In [31]:
```
model.save('chest_xray_pneumonia_model.h5')
```

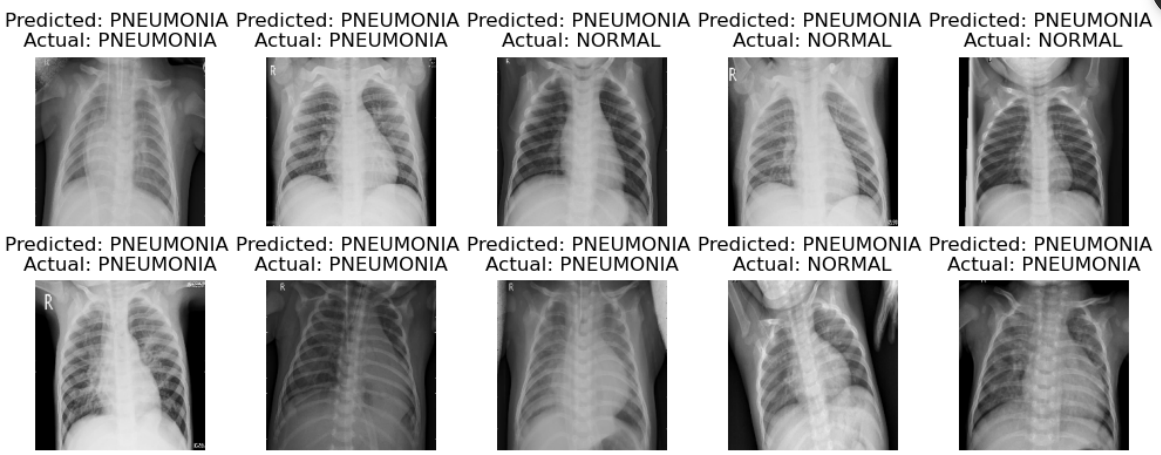
# Load Model and Test

In [32]:
```
modelLoaded = load_model('chest_xray_pneumonia_model.h5')
```

In [33]:
```
def predictImg():
    fig, axs = plt.subplots(2, 5, figsize=(10, 4))
    axs = axs.flatten()

    for i in range(10):
        testimgnum = i+30
        predicted = modelLoaded.predict(np.array([x_test[testimgnum]]))
        predicted = int(np.round(predicted))
        predictedClass = int(predicted >= 0.5)
        actual = y_test[testimgnum]

        axs[i].imshow(x_test[testimgnum], cmap='gray')
        axs[i].set_title(f'Predicted: {classnames[predictedClass]} \nActual: {cl
        axs[i].axis('off')

    plt.tight_layout()
    plt.show()
```

In [34]:
```
predictImg()
```

```
1/1 [==============================] - 0s 203ms/step
1/1 [==============================] - 0s 47ms/step
C:\Users\kirti\AppData\Local\Temp\ipykernel_6668\2954564863.py:8: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  predicted = int(np.round(predicted))
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
```




In [ ]: