## Here you can find the necessary import

In [1]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, precision_recall_curve, average_precision_sc
from sklearn.preprocessing import normalize
import pandas as pd
import matplotlib.pyplot as plt
```

```
WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is d
eprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [2]:
```
# you need the current working directory NB: works both windows and linux
current_working_directory = os.getcwd()
current_working_directory = os.path.dirname(current_working_directory)

if not os.path.exists(f"{current_working_directory}/Datasets"):
    os.makedirs(f"{current_working_directory}/Datasets")

print(f"[DATASET] PUT THE DATASET here: {current_working_directory}/Datasets")
```

```
[DATASET] PUT THE DATASET here: C:\Users\Pikkis/Datasets
```

In [3]:
```
# get the directory where I want to download the dataset
path_of_dataset = os.path.join(*['..', current_working_directory, 'Datasets', 'pizza_not
print(f"[DIR] The directory of the current dataset is {path_of_dataset}")
```

```
[DIR] The directory of the current dataset is C:\Users\Pikkis\Datasets\pizza_not_pizza
```

## Data prep

In [4]:
```
# here let s do some functions that we can re-use also for other assignment
def load_the_data_and_the_labels(data_set_path: str, target_size: tuple or None = None):
    """
    This function help you to load the data dynamically
    :param data_set_path: (str) put the path created in the previous cell (is the datase
    :param target_size: (tuple) the desired size of the images
    :return:
        - array of images
        - array with labels
        - list of labels name (this is used for better visualization)
    """
    try:
        dataset, labels, name_of_the_labels = list(), list(), list()
        # let s loop here and we try to discover how many class we have
        for class_number, class_name in enumerate(os.listdir(data_set_path)):
            full_path_the_data = os.path.join(*[data_set_path, class_name])
            print(f"[WALK] I am walking into {full_path_the_data}")

            # add the list to nam _list
            name_of_the_labels.append(class_name)

            for single_image in os.listdir(f"{full_path_the_data}"):
                full_path_to_image = os.path.join(*[full_path_the_data, single_image])
```

```
                # add the class number
                labels.append(class_number)

                if target_size is None:
                    # let s load the image
                    image = tf.keras.utils.load_img(full_path_to_image)
                else:
                    image = tf.keras.utils.load_img(full_path_to_image, target_size=targ

                # transform PIL object in image
                image = tf.keras.utils.img_to_array(image)

                # add the image to the ds list
                dataset.append(image)

        return np.array(dataset, dtype='uint8'), np.array(labels, dtype='int'), name_of_
    except Exception as ex:
        print(f"[EXCEPTION] load the data and the labels throws exceptions {ex}")
```

Load the data

In [5]:
```
# load the data
data = load_the_data_and_the_labels(path_of_dataset, (224,224,3))
```

```
[WALK] I am walking into C:\Users\Pikkis\Datasets\pizza_not_pizza\not_pizza
[WALK] I am walking into C:\Users\Pikkis\Datasets\pizza_not_pizza\pizza
```

Normalize the data

In [6]:
```
# normalize the data
mean = np.mean(data[0])
std = np.std(data[0])
n_data = [(d-mean)/std for d in data[0]]

data_arr = np.asarray(n_data)
```

Split the data use the train_test_split function

In [7]:
```
# split the data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(data_arr, data[1], test_size=0.3, ra
```

## Create the CNN according the instruction:

```
        a. Input layer
        b. Data augmentation, with random flip (horizontal and vertical) and
    random rotation (0.2).
        c. Two hidden layers each composed with the following
    characteristics: 16 conv2d units, max pooling 2d and batch
    normalization, the second one should have 24 conv2d units max pooling 2d
    and batch normalization.
        d. After this, add a flatten layer and a dense layer with 8 units
        e. Add the final classifier (a  dense layer) with the correct number
    of output and activation
```

alt text

In [8]:

```
# create the cnn

#Creating the input layer
input_layer = tf.keras.Input(shape=(224,224,3))

# Augment layer with random flip and rotation
augment_layer = tf.keras.Sequential([tf.keras.layers.RandomFlip("horizontal_and_vertical

# Two hidden layers with Conv2D, MaxPooling2D and batch normalization
hidden_layer1 = tf.keras.layers.Conv2D(16, (3,3),activation='relu', input_shape=(224,224
max_pooling2d = tf.keras.layers.MaxPooling2D(2,2)(hidden_layer1)
batch_normalize = tf.keras.layers.BatchNormalization(axis=-1)(max_pooling2d)
hidden_layer2 = tf.keras.layers.Conv2D(24, (3,3), activation='relu', input_shape=(224,22
max_pooling2d2 = tf.keras.layers.MaxPooling2D(2,2)(hidden_layer2)
batch_normalize2 = tf.keras.layers.BatchNormalization(axis=-1)(max_pooling2d2)

# Flatten layer
flatten_layer = tf.keras.layers.Flatten()(batch_normalize2)

# Our final two dense layers
dense_layer = tf.keras.layers.Dense(units=8, activation='relu')(flatten_layer)
final_dense_layer = tf.keras.layers.Dense(units=1, activation='sigmoid')(dense_layer)

# Finishing our model
model = tf.keras.Model(input_layer, final_dense_layer)
model.summary()
```

```
WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is
deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site
-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is depr
ecated. Please use tf.nn.max_pool2d instead.

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 sequential (Sequential)     (None, 224, 224, 3)       0         
                                                                 
 conv2d (Conv2D)             (None, 222, 222, 16)      448       
                                                                 
 max_pooling2d (MaxPooling2  (None, 111, 111, 16)      0         
 D)                                                              
                                                                 
 batch_normalization (Batch  (None, 111, 111, 16)      64        
 Normalization)                                                  
                                                                 
 conv2d_1 (Conv2D)           (None, 109, 109, 24)      3480      
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 24)        0         
 g2D)                                                            
                                                                 
 batch_normalization_1 (Bat  (None, 54, 54, 24)        96        
 chNormalization)                                                
                                                                 
 flatten (Flatten)           (None, 69984)             0         
                                                                 
 dense (Dense)               (None, 8)                 559880    
                                                                 
 dense_1 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 563977 (2.15 MB)
Trainable params: 563897 (2.15 MB)
Non-trainable params: 80 (320.00 Byte)
_________________________________________________________________
```

## compile the model

Compile the model with Adam optimizer and binary cross entropy as loss function.

In [9]:
```python
# compile the CNN

# Compiling our cnn
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3), loss=tf.keras.loss
```

## Train the model with 128 epochs and 64 batch size

In [10]:
```python
# Training our model with 128 epochs and batch size 64
model.fit(X_train, y_train, epochs=128, batch_size=64)
```

```
Epoch 1/128
WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Pikkis\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

22/22 [==============================] - 6s 205ms/step - loss: 1.5158 - accuracy: 0.6206
Epoch 2/128
22/22 [==============================] - 5s 207ms/step - loss: 1.2996 - accuracy: 0.5778
Epoch 3/128
22/22 [==============================] - 5s 206ms/step - loss: 0.9886 - accuracy: 0.6235
Epoch 4/128
22/22 [==============================] - 5s 207ms/step - loss: 0.9598 - accuracy: 0.6795
Epoch 5/128
22/22 [==============================] - 5s 206ms/step - loss: 0.8972 - accuracy: 0.7020
Epoch 6/128
22/22 [==============================] - 5s 208ms/step - loss: 0.8130 - accuracy: 0.6962
Epoch 7/128
22/22 [==============================] - 5s 207ms/step - loss: 0.6393 - accuracy: 0.7311
Epoch 8/128
22/22 [==============================] - 4s 204ms/step - loss: 0.5812 - accuracy: 0.7456
Epoch 9/128
22/22 [==============================] - 4s 197ms/step - loss: 0.5343 - accuracy: 0.7558
Epoch 10/128
22/22 [==============================] - 4s 199ms/step - loss: 0.4969 - accuracy: 0.7558
Epoch 11/128
22/22 [==============================] - 4s 199ms/step - loss: 0.5212 - accuracy: 0.7515
Epoch 12/128
22/22 [==============================] - 4s 198ms/step - loss: 0.5376 - accuracy: 0.7631
Epoch 13/128
22/22 [==============================] - 4s 196ms/step - loss: 0.4811 - accuracy: 0.7616
Epoch 14/128
22/22 [==============================] - 4s 198ms/step - loss: 0.4899 - accuracy: 0.7631
Epoch 15/128
22/22 [==============================] - 4s 201ms/step - loss: 0.4855 - accuracy: 0.7754
Epoch 16/128
22/22 [==============================] - 4s 202ms/step - loss: 0.4482 - accuracy: 0.7769
Epoch 17/128
22/22 [==============================] - 4s 202ms/step - loss: 0.4364 - accuracy: 0.7849
Epoch 18/128
22/22 [==============================] - 4s 201ms/step - loss: 0.4647 - accuracy: 0.7842
Epoch 19/128
22/22 [==============================] - 4s 201ms/step - loss: 0.4471 - accuracy: 0.7885
Epoch 20/128
22/22 [==============================] - 4s 202ms/step - loss: 0.4419 - accuracy: 0.7791
Epoch 21/128
22/22 [==============================] - 4s 199ms/step - loss: 0.4341 - accuracy: 0.7922
Epoch 22/128
22/22 [==============================] - 4s 198ms/step - loss: 0.4521 - accuracy: 0.7733
Epoch 23/128
22/22 [==============================] - 4s 197ms/step - loss: 0.4154 - accuracy: 0.7834
Epoch 24/128
22/22 [==============================] - 4s 199ms/step - loss: 0.4381 - accuracy: 0.7878
Epoch 25/128
22/22 [==============================] - 4s 198ms/step - loss: 0.4132 - accuracy: 0.7987
Epoch 26/128
22/22 [==============================] - 4s 197ms/step - loss: 0.4148 - accuracy: 0.8089
```

```
22/22 [==============================] - 4s 197ms/step - loss: 0.4189 - accuracy: 0.7914
Epoch 28/128
22/22 [==============================] - 4s 197ms/step - loss: 0.4128 - accuracy: 0.8001
Epoch 29/128
22/22 [==============================] - 4s 198ms/step - loss: 0.4252 - accuracy: 0.7936
Epoch 30/128
22/22 [==============================] - 4s 204ms/step - loss: 0.4320 - accuracy: 0.7805
Epoch 31/128
22/22 [==============================] - 4s 198ms/step - loss: 0.4217 - accuracy: 0.7943
Epoch 32/128
22/22 [==============================] - 4s 197ms/step - loss: 0.4052 - accuracy: 0.8089
Epoch 33/128
22/22 [==============================] - 4s 197ms/step - loss: 0.3936 - accuracy: 0.7980
Epoch 34/128
22/22 [==============================] - 4s 199ms/step - loss: 0.3923 - accuracy: 0.8140
Epoch 35/128
22/22 [==============================] - 4s 204ms/step - loss: 0.3851 - accuracy: 0.8125
Epoch 36/128
22/22 [==============================] - 4s 199ms/step - loss: 0.3875 - accuracy: 0.8089
Epoch 37/128
22/22 [==============================] - 4s 199ms/step - loss: 0.3692 - accuracy: 0.8154
Epoch 38/128
22/22 [==============================] - 4s 196ms/step - loss: 0.3873 - accuracy: 0.8103
Epoch 39/128
22/22 [==============================] - 4s 199ms/step - loss: 0.3859 - accuracy: 0.8169
Epoch 40/128
22/22 [==============================] - 4s 199ms/step - loss: 0.3987 - accuracy: 0.8009
Epoch 41/128
22/22 [==============================] - 4s 196ms/step - loss: 0.3960 - accuracy: 0.8067
Epoch 42/128
22/22 [==============================] - 4s 197ms/step - loss: 0.3864 - accuracy: 0.8154
Epoch 43/128
22/22 [==============================] - 4s 196ms/step - loss: 0.3716 - accuracy: 0.8256
Epoch 44/128
22/22 [==============================] - 4s 199ms/step - loss: 0.3680 - accuracy: 0.8270
Epoch 45/128
22/22 [==============================] - 4s 197ms/step - loss: 0.3806 - accuracy: 0.8227
Epoch 46/128
22/22 [==============================] - 4s 197ms/step - loss: 0.3721 - accuracy: 0.8292
Epoch 47/128
22/22 [==============================] - 4s 198ms/step - loss: 0.3588 - accuracy: 0.8307
Epoch 48/128
22/22 [==============================] - 4s 198ms/step - loss: 0.3569 - accuracy: 0.8350
Epoch 49/128
22/22 [==============================] - 4s 197ms/step - loss: 0.3679 - accuracy: 0.8350
Epoch 50/128
22/22 [==============================] - 4s 200ms/step - loss: 0.3558 - accuracy: 0.8328
Epoch 51/128
22/22 [==============================] - 4s 196ms/step - loss: 0.3656 - accuracy: 0.8278
Epoch 52/128
22/22 [==============================] - 4s 199ms/step - loss: 0.3884 - accuracy: 0.8263
Epoch 53/128
22/22 [==============================] - 4s 197ms/step - loss: 0.3506 - accuracy: 0.8328
Epoch 54/128
22/22 [==============================] - 4s 198ms/step - loss: 0.3540 - accuracy: 0.8285
Epoch 55/128
22/22 [==============================] - 4s 197ms/step - loss: 0.3353 - accuracy: 0.8445
Epoch 56/128
22/22 [==============================] - 4s 196ms/step - loss: 0.3335 - accuracy: 0.8481
Epoch 57/128
22/22 [==============================] - 4s 198ms/step - loss: 0.3462 - accuracy: 0.8423
```

```
22/22 [==============================] - 5s 205ms/step - loss: 0.3505 - accuracy: 0.8343
Epoch 59/128
22/22 [==============================] - 4s 203ms/step - loss: 0.3572 - accuracy: 0.8379
Epoch 60/128
22/22 [==============================] - 5s 209ms/step - loss: 0.3379 - accuracy: 0.8481
Epoch 61/128
22/22 [==============================] - 5s 207ms/step - loss: 0.3586 - accuracy: 0.8372
Epoch 62/128
22/22 [==============================] - 5s 213ms/step - loss: 0.3634 - accuracy: 0.8234
Epoch 63/128
22/22 [==============================] - 5s 211ms/step - loss: 0.3395 - accuracy: 0.8416
Epoch 64/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3264 - accuracy: 0.8612
Epoch 65/128
22/22 [==============================] - 5s 210ms/step - loss: 0.3183 - accuracy: 0.8496
Epoch 66/128
22/22 [==============================] - 4s 204ms/step - loss: 0.2903 - accuracy: 0.8677
Epoch 67/128
22/22 [==============================] - 5s 206ms/step - loss: 0.3110 - accuracy: 0.8619
Epoch 68/128
22/22 [==============================] - 5s 207ms/step - loss: 0.3191 - accuracy: 0.8496
Epoch 69/128
22/22 [==============================] - 5s 208ms/step - loss: 0.3057 - accuracy: 0.8597
Epoch 70/128
22/22 [==============================] - 5s 208ms/step - loss: 0.3242 - accuracy: 0.8423
Epoch 71/128
22/22 [==============================] - 5s 208ms/step - loss: 0.3199 - accuracy: 0.8583
Epoch 72/128
22/22 [==============================] - 5s 207ms/step - loss: 0.3294 - accuracy: 0.8423
Epoch 73/128
22/22 [==============================] - 5s 206ms/step - loss: 0.3205 - accuracy: 0.8517
Epoch 74/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2890 - accuracy: 0.8721
Epoch 75/128
22/22 [==============================] - 5s 208ms/step - loss: 0.3001 - accuracy: 0.8750
Epoch 76/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2950 - accuracy: 0.8568
Epoch 77/128
22/22 [==============================] - 5s 210ms/step - loss: 0.2936 - accuracy: 0.8677
Epoch 78/128
22/22 [==============================] - 5s 208ms/step - loss: 0.3129 - accuracy: 0.8648
Epoch 79/128
22/22 [==============================] - 5s 208ms/step - loss: 0.3006 - accuracy: 0.8626
Epoch 80/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2914 - accuracy: 0.8721
Epoch 81/128
22/22 [==============================] - 5s 209ms/step - loss: 0.3079 - accuracy: 0.8590
Epoch 82/128
22/22 [==============================] - 5s 210ms/step - loss: 0.3029 - accuracy: 0.8590
Epoch 83/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2822 - accuracy: 0.8699
Epoch 84/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2882 - accuracy: 0.8663
Epoch 85/128
22/22 [==============================] - 5s 207ms/step - loss: 0.2894 - accuracy: 0.8699
Epoch 86/128
22/22 [==============================] - 5s 206ms/step - loss: 0.2773 - accuracy: 0.8786
Epoch 87/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2623 - accuracy: 0.8852
Epoch 88/128
22/22 [==============================] - 5s 210ms/step - loss: 0.2699 - accuracy: 0.8786
```

```
22/22 [==============================] - 5s 211ms/step - loss: 0.2733 - accuracy: 0.8844
Epoch 90/128
22/22 [==============================] - 5s 216ms/step - loss: 0.2626 - accuracy: 0.8794
Epoch 91/128
22/22 [==============================] - 5s 215ms/step - loss: 0.2669 - accuracy: 0.8823
Epoch 92/128
22/22 [==============================] - 5s 217ms/step - loss: 0.2569 - accuracy: 0.8823
Epoch 93/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2741 - accuracy: 0.8815
Epoch 94/128
22/22 [==============================] - 5s 239ms/step - loss: 0.2756 - accuracy: 0.8757
Epoch 95/128
22/22 [==============================] - 5s 230ms/step - loss: 0.2528 - accuracy: 0.8823
Epoch 96/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2720 - accuracy: 0.8779
Epoch 97/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2767 - accuracy: 0.8772
Epoch 98/128
22/22 [==============================] - 5s 216ms/step - loss: 0.2567 - accuracy: 0.8830
Epoch 99/128
22/22 [==============================] - 5s 215ms/step - loss: 0.2472 - accuracy: 0.8823
Epoch 100/128
22/22 [==============================] - 5s 212ms/step - loss: 0.2532 - accuracy: 0.8903
Epoch 101/128
22/22 [==============================] - 5s 215ms/step - loss: 0.2679 - accuracy: 0.8866
Epoch 102/128
22/22 [==============================] - 5s 214ms/step - loss: 0.2831 - accuracy: 0.8794
Epoch 103/128
22/22 [==============================] - 5s 216ms/step - loss: 0.2593 - accuracy: 0.8874
Epoch 104/128
22/22 [==============================] - 5s 213ms/step - loss: 0.2633 - accuracy: 0.8794
Epoch 105/128
22/22 [==============================] - 5s 213ms/step - loss: 0.2467 - accuracy: 0.8743
Epoch 106/128
22/22 [==============================] - 5s 216ms/step - loss: 0.2560 - accuracy: 0.8881
Epoch 107/128
22/22 [==============================] - 5s 215ms/step - loss: 0.2505 - accuracy: 0.8866
Epoch 108/128
22/22 [==============================] - 5s 211ms/step - loss: 0.2738 - accuracy: 0.8735
Epoch 109/128
22/22 [==============================] - 5s 211ms/step - loss: 0.2527 - accuracy: 0.8765
Epoch 110/128
22/22 [==============================] - 5s 210ms/step - loss: 0.2543 - accuracy: 0.8939
Epoch 111/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2393 - accuracy: 0.8939
Epoch 112/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2430 - accuracy: 0.8924
Epoch 113/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2406 - accuracy: 0.8750
Epoch 114/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2571 - accuracy: 0.8808
Epoch 115/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2581 - accuracy: 0.8961
Epoch 116/128
22/22 [==============================] - 5s 210ms/step - loss: 0.2509 - accuracy: 0.8881
Epoch 117/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2384 - accuracy: 0.8924
Epoch 118/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2161 - accuracy: 0.8910
Epoch 119/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2282 - accuracy: 0.8953
```

```
22/22 [==============================] - 5s 209ms/step - loss: 0.2519 - accuracy: 0.8830
Epoch 121/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2201 - accuracy: 0.9026
Epoch 122/128
22/22 [==============================] - 5s 217ms/step - loss: 0.2362 - accuracy: 0.8997
Epoch 123/128
22/22 [==============================] - 5s 218ms/step - loss: 0.2387 - accuracy: 0.8910
Epoch 124/128
22/22 [==============================] - 5s 221ms/step - loss: 0.2257 - accuracy: 0.8932
Epoch 125/128
22/22 [==============================] - 5s 212ms/step - loss: 0.2140 - accuracy: 0.9128
Epoch 126/128
22/22 [==============================] - 5s 208ms/step - loss: 0.2269 - accuracy: 0.8983
Epoch 127/128
22/22 [==============================] - 5s 209ms/step - loss: 0.2135 - accuracy: 0.9055
Epoch 128/128
22/22 [==============================] - 5s 205ms/step - loss: 0.2140 - accuracy: 0.9092
```

Out[10]:
```
<keras.src.callbacks.History at 0x1c1590d7850>
```

### Evaluate the model and report the accuracy

In [11]:
```python
#Evaluating our model and reporting the accuracy
scores = model.evaluate(X_test, y_test)
```

```
19/19 [==============================] - 1s 26ms/step - loss: 0.6329 - accuracy: 0.8051
```

### Make prediction with the test set and use a threshold of 0.5 as boundaries decision between the classes.

In [12]:
```python
# do it here

#Making some predictions
predictions = model.predict(X_test)

#Setting a 0.5 threshold
threshold_predictions = (predictions > 0.5).astype(int)

#Modifying our array to better fit the prediction function below
test = threshold_predictions.tolist()
real_predictions = []

for xs in test:
    for x in xs:
        real_predictions.append(x)
```

```
19/19 [==============================] - 1s 26ms/step
```

### show predictions

In [13]:
```python
def show_some_prediction(number_of_subplot, test_set, predictions, name_of_the_labels):
    for i in range(number_of_subplot):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(test_set[i])
        plt.title(f'{name_of_the_labels[predictions[i]]}')
        plt.axis("off")
    plt.show()
```

In [14]:

```python
# do it here

#Using our premade function to display some predicted images
show_some_prediction(9, X_test, real_predictions, data[2])
```

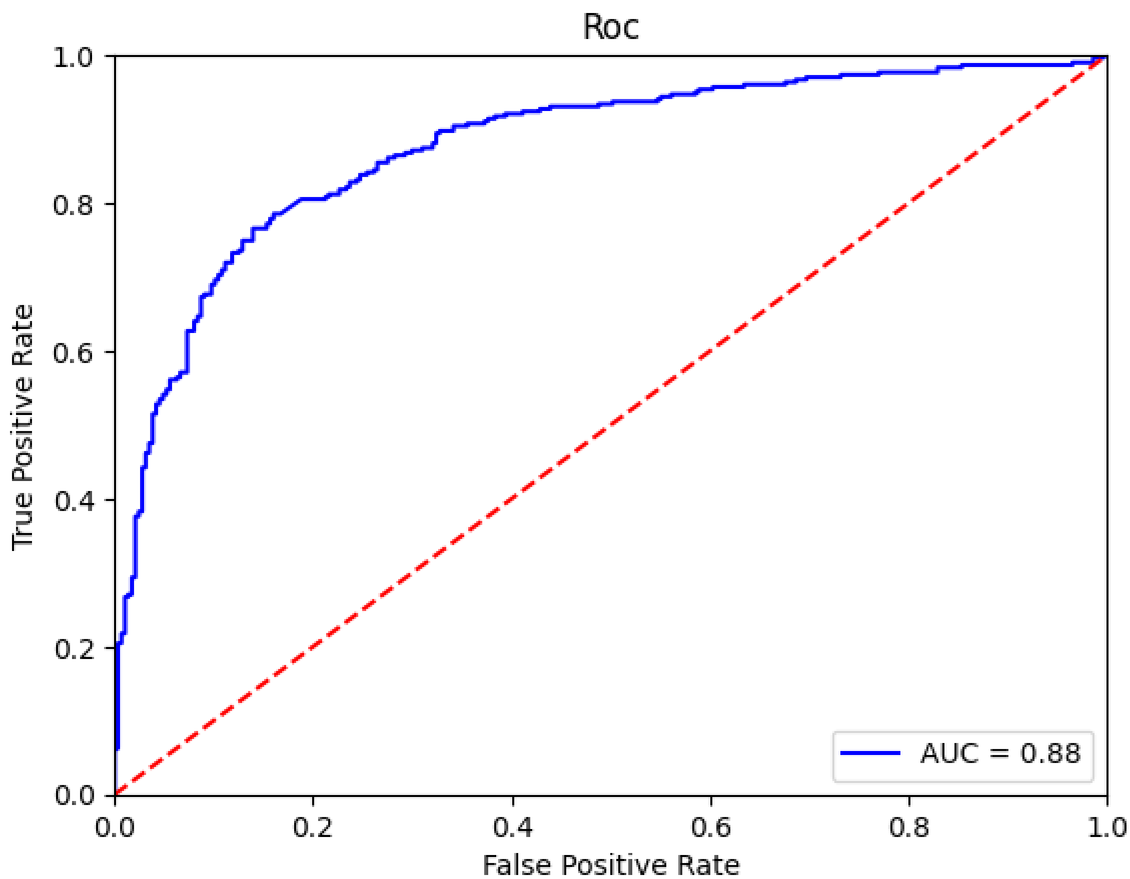
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


show metrics like confusion matrix or ROC curve or both (sklearn has already implemented all these stuff)

In [15]:

```python
#Calculating our roc
fpr, tpr, threshold = roc_curve(y_test, predictions)
roc_auc = auc(fpr, tpr)

#Plotting and displaying our roc curve
plt.title('Roc')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
1], [0, 1],'r--')
```

```
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




## Build another base CNN, but at point c add an extra hidden layer with 32 units of conv2d. Repeat all the other steps. What happened to the accuracy of the model? Why?

alt text

In [16]:
```
#Creating second cnn

#Input layer
xinput_layer = tf.keras.Input(shape=(224,224,3))

#Data augmentation layer with random flip and rotation
xaugment_layer = tf.keras.Sequential([tf.keras.layers.RandomFlip("horizontal_and_vertica

#Three hidden layers with Conv2D, MaxPooling2D and Batch normalization
xhidden_layer1 = tf.keras.layers.Conv2D(16, (3,3),activation='relu', input_shape=(224,22
xmax_pooling2d = tf.keras.layers.MaxPooling2D(2,2)(xhidden_layer1)
xbatch_normalize = tf.keras.layers.BatchNormalization(axis=-1)(xmax_pooling2d)

xhidden_layer2 = tf.keras.layers.Conv2D(24, (3,3), activation='relu', input_shape=(224,2
xmax_pooling2d2 = tf.keras.layers.MaxPooling2D(2,2)(xhidden_layer2)
xbatch_normalize2 = tf.keras.layers.BatchNormalization(axis=-1)(xmax_pooling2d2)

xhidden_layer3 = tf.keras.layers.Conv2D(24, (3,3), activation='relu', input_shape=(224,2
xmax_pooling2d3 = tf.keras.layers.MaxPooling2D(2,2)(xhidden_layer3)
xbatch_normalize3 = tf.keras.layers.BatchNormalization(axis=-1)(xmax_pooling2d3)
```

```
#Flatten layer
xflatten_layer = tf.keras.layers.Flatten()(xbatch_normalize3)

#Two dense layers
xdense_layer = tf.keras.layers.Dense(units=8, activation='relu')(xflatten_layer)
xfinal_dense_layer = tf.keras.layers.Dense(units=1, activation='sigmoid')(xdense_layer)

#Final model
model2 = tf.keras.Model(xinput_layer, xfinal_dense_layer)
model2.summary()
```

```
Model: "model_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential_1 (Sequential)   (None, 224, 224, 3)       0

 conv2d_2 (Conv2D)           (None, 222, 222, 16)      448

 max_pooling2d_2 (MaxPoolin  (None, 111, 111, 16)      0
 g2D)

 batch_normalization_2 (Bat  (None, 111, 111, 16)      64
 chNormalization)

 conv2d_3 (Conv2D)           (None, 109, 109, 24)      3480

 max_pooling2d_3 (MaxPoolin  (None, 54, 54, 24)        0
 g2D)

 batch_normalization_3 (Bat  (None, 54, 54, 24)        96
 chNormalization)

 conv2d_4 (Conv2D)           (None, 52, 52, 24)        5208

 max_pooling2d_4 (MaxPoolin  (None, 26, 26, 24)        0
 g2D)

 batch_normalization_4 (Bat  (None, 26, 26, 24)        96
 chNormalization)

 flatten_1 (Flatten)         (None, 16224)             0

 dense_2 (Dense)             (None, 8)                 129800

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 139201 (543.75 KB)
Trainable params: 139073 (543.25 KB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
```

In [17]:

```
#Compiling our model
model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3), loss=tf.keras.los
```

In [18]:

```
#Training our model with batch size 64 and 128 epochs.
model2.fit(X_train, y_train, epochs=128, batch_size=64)
```

```
Epoch 1/128
22/22 [==============================] - 6s 221ms/step - loss: 0.7058 - accuracy: 0.6788
Epoch 2/128
22/22 [==============================] - 5s 219ms/step - loss: 0.5981 - accuracy: 0.7173
Epoch 3/128
22/22 [==============================] - 5s 220ms/step - loss: 0.6198 - accuracy: 0.7442
Epoch 4/128
22/22 [==============================] - 5s 220ms/step - loss: 0.5413 - accuracy: 0.7435
Epoch 5/128
22/22 [==============================] - 5s 220ms/step - loss: 0.5265 - accuracy: 0.7645
Epoch 6/128
22/22 [==============================] - 5s 220ms/step - loss: 0.5294 - accuracy: 0.7660
Epoch 7/128
22/22 [==============================] - 5s 219ms/step - loss: 0.5015 - accuracy: 0.7624
Epoch 8/128
22/22 [==============================] - 5s 220ms/step - loss: 0.4691 - accuracy: 0.7856
Epoch 9/128
22/22 [==============================] - 5s 219ms/step - loss: 0.4419 - accuracy: 0.7922
Epoch 10/128
22/22 [==============================] - 5s 219ms/step - loss: 0.4585 - accuracy: 0.7907
Epoch 11/128
22/22 [==============================] - 5s 220ms/step - loss: 0.4894 - accuracy: 0.7892
Epoch 12/128
22/22 [==============================] - 5s 222ms/step - loss: 0.4407 - accuracy: 0.8067
Epoch 13/128
22/22 [==============================] - 5s 220ms/step - loss: 0.4354 - accuracy: 0.8140
Epoch 14/128
22/22 [==============================] - 5s 220ms/step - loss: 0.4296 - accuracy: 0.8089
Epoch 15/128
22/22 [==============================] - 5s 218ms/step - loss: 0.4490 - accuracy: 0.8009
Epoch 16/128
22/22 [==============================] - 5s 220ms/step - loss: 0.4501 - accuracy: 0.7987
Epoch 17/128
22/22 [==============================] - 5s 219ms/step - loss: 0.4319 - accuracy: 0.7958
Epoch 18/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3976 - accuracy: 0.8118
Epoch 19/128
22/22 [==============================] - 5s 220ms/step - loss: 0.3911 - accuracy: 0.8292
Epoch 20/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3673 - accuracy: 0.8416
Epoch 21/128
22/22 [==============================] - 5s 222ms/step - loss: 0.3590 - accuracy: 0.8416
Epoch 22/128
22/22 [==============================] - 5s 217ms/step - loss: 0.3751 - accuracy: 0.8336
Epoch 23/128
22/22 [==============================] - 5s 220ms/step - loss: 0.3692 - accuracy: 0.8452
Epoch 24/128
22/22 [==============================] - 5s 218ms/step - loss: 0.3557 - accuracy: 0.8372
Epoch 25/128
22/22 [==============================] - 5s 223ms/step - loss: 0.3743 - accuracy: 0.8328
Epoch 26/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3579 - accuracy: 0.8394
Epoch 27/128
22/22 [==============================] - 5s 220ms/step - loss: 0.3415 - accuracy: 0.8445
Epoch 28/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3299 - accuracy: 0.8597
Epoch 29/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3175 - accuracy: 0.8641
Epoch 30/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3227 - accuracy: 0.8634
Epoch 31/128
=======================] - 5s 219ms/step - loss: 0.3318 - accuracy: 0.8503
```

```
Epoch 32/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3317 - accuracy: 0.8597
Epoch 33/128
22/22 [==============================] - 5s 218ms/step - loss: 0.3171 - accuracy: 0.8656
Epoch 34/128
22/22 [==============================] - 5s 220ms/step - loss: 0.3127 - accuracy: 0.8706
Epoch 35/128
22/22 [==============================] - 5s 221ms/step - loss: 0.2917 - accuracy: 0.8794
Epoch 36/128
22/22 [==============================] - 5s 218ms/step - loss: 0.3029 - accuracy: 0.8757
Epoch 37/128
22/22 [==============================] - 5s 220ms/step - loss: 0.3123 - accuracy: 0.8641
Epoch 38/128
22/22 [==============================] - 5s 221ms/step - loss: 0.3146 - accuracy: 0.8532
Epoch 39/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2915 - accuracy: 0.8706
Epoch 40/128
22/22 [==============================] - 5s 222ms/step - loss: 0.2869 - accuracy: 0.8794
Epoch 41/128
22/22 [==============================] - 5s 223ms/step - loss: 0.3005 - accuracy: 0.8706
Epoch 42/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2724 - accuracy: 0.8859
Epoch 43/128
22/22 [==============================] - 5s 218ms/step - loss: 0.2947 - accuracy: 0.8794
Epoch 44/128
22/22 [==============================] - 5s 219ms/step - loss: 0.3068 - accuracy: 0.8735
Epoch 45/128
22/22 [==============================] - 5s 217ms/step - loss: 0.2995 - accuracy: 0.8706
Epoch 46/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2830 - accuracy: 0.8765
Epoch 47/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2768 - accuracy: 0.8881
Epoch 48/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2476 - accuracy: 0.8895
Epoch 49/128
22/22 [==============================] - 5s 218ms/step - loss: 0.2733 - accuracy: 0.8830
Epoch 50/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2610 - accuracy: 0.8910
Epoch 51/128
22/22 [==============================] - 5s 218ms/step - loss: 0.2626 - accuracy: 0.8910
Epoch 52/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2695 - accuracy: 0.8706
Epoch 53/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2518 - accuracy: 0.8903
Epoch 54/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2269 - accuracy: 0.9026
Epoch 55/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2537 - accuracy: 0.8910
Epoch 56/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2277 - accuracy: 0.8997
Epoch 57/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2411 - accuracy: 0.8932
Epoch 58/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2159 - accuracy: 0.9004
Epoch 59/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2283 - accuracy: 0.9033
Epoch 60/128
22/22 [==============================] - 5s 218ms/step - loss: 0.2353 - accuracy: 0.8983
Epoch 61/128
22/22 [==============================] - 5s 218ms/step - loss: 0.2327 - accuracy: 0.8953
Epoch 62/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2143 - accuracy: 0.9142
```

```
Epoch 63/128
22/22 [==============================] - 5s 221ms/step - loss: 0.2282 - accuracy: 0.9055
Epoch 64/128
22/22 [==============================] - 5s 221ms/step - loss: 0.2209 - accuracy: 0.9004
Epoch 65/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2334 - accuracy: 0.8910
Epoch 66/128
22/22 [==============================] - 5s 220ms/step - loss: 0.2379 - accuracy: 0.8968
Epoch 67/128
22/22 [==============================] - 5s 218ms/step - loss: 0.2161 - accuracy: 0.9142
Epoch 68/128
22/22 [==============================] - 5s 217ms/step - loss: 0.2330 - accuracy: 0.8990
Epoch 69/128
22/22 [==============================] - 5s 219ms/step - loss: 0.2197 - accuracy: 0.9070
Epoch 70/128
22/22 [==============================] - 5s 230ms/step - loss: 0.2226 - accuracy: 0.9099
Epoch 71/128
22/22 [==============================] - 5s 230ms/step - loss: 0.1979 - accuracy: 0.9179
Epoch 72/128
22/22 [==============================] - 5s 222ms/step - loss: 0.2171 - accuracy: 0.9099
Epoch 73/128
22/22 [==============================] - 5s 235ms/step - loss: 0.1964 - accuracy: 0.9157
Epoch 74/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1935 - accuracy: 0.9135
Epoch 75/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1965 - accuracy: 0.9142
Epoch 76/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1865 - accuracy: 0.9186
Epoch 77/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1857 - accuracy: 0.9208
Epoch 78/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1971 - accuracy: 0.9157
Epoch 79/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1934 - accuracy: 0.9186
Epoch 80/128
22/22 [==============================] - 5s 223ms/step - loss: 0.1864 - accuracy: 0.9266
Epoch 81/128
22/22 [==============================] - 5s 220ms/step - loss: 0.1959 - accuracy: 0.9172
Epoch 82/128
22/22 [==============================] - 5s 222ms/step - loss: 0.2067 - accuracy: 0.9157
Epoch 83/128
22/22 [==============================] - 5s 219ms/step - loss: 0.1924 - accuracy: 0.9179
Epoch 84/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1840 - accuracy: 0.9208
Epoch 85/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1923 - accuracy: 0.9150
Epoch 86/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1596 - accuracy: 0.9331
Epoch 87/128
22/22 [==============================] - 5s 223ms/step - loss: 0.1769 - accuracy: 0.9324
Epoch 88/128
22/22 [==============================] - 5s 223ms/step - loss: 0.1822 - accuracy: 0.9273
Epoch 89/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1862 - accuracy: 0.9259
Epoch 90/128
22/22 [==============================] - 5s 223ms/step - loss: 0.1788 - accuracy: 0.9222
Epoch 91/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1877 - accuracy: 0.9237
Epoch 92/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1662 - accuracy: 0.9273
Epoch 93/128
==========================] - 5s 225ms/step - loss: 0.1533 - accuracy: 0.9382
```

```
Epoch 94/128
22/22 [==============================] - 5s 237ms/step - loss: 0.1787 - accuracy: 0.9273
Epoch 95/128
22/22 [==============================] - 5s 236ms/step - loss: 0.1403 - accuracy: 0.9360
Epoch 96/128
22/22 [==============================] - 5s 224ms/step - loss: 0.1837 - accuracy: 0.9266
Epoch 97/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1787 - accuracy: 0.9273
Epoch 98/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1616 - accuracy: 0.9302
Epoch 99/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1484 - accuracy: 0.9382
Epoch 100/128
22/22 [==============================] - 5s 223ms/step - loss: 0.1475 - accuracy: 0.9419
Epoch 101/128
22/22 [==============================] - 5s 220ms/step - loss: 0.1616 - accuracy: 0.9419
Epoch 102/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1727 - accuracy: 0.9324
Epoch 103/128
22/22 [==============================] - 5s 220ms/step - loss: 0.1732 - accuracy: 0.9310
Epoch 104/128
22/22 [==============================] - 5s 219ms/step - loss: 0.1560 - accuracy: 0.9382
Epoch 105/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1538 - accuracy: 0.9440
Epoch 106/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1459 - accuracy: 0.9390
Epoch 107/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1671 - accuracy: 0.9331
Epoch 108/128
22/22 [==============================] - 5s 220ms/step - loss: 0.1466 - accuracy: 0.9469
Epoch 109/128
22/22 [==============================] - 5s 219ms/step - loss: 0.1610 - accuracy: 0.9310
Epoch 110/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1418 - accuracy: 0.9397
Epoch 111/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1347 - accuracy: 0.9448
Epoch 112/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1534 - accuracy: 0.9375
Epoch 113/128
22/22 [==============================] - 5s 224ms/step - loss: 0.1352 - accuracy: 0.9484
Epoch 114/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1372 - accuracy: 0.9411
Epoch 115/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1359 - accuracy: 0.9419
Epoch 116/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1212 - accuracy: 0.9571
Epoch 117/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1373 - accuracy: 0.9440
Epoch 118/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1385 - accuracy: 0.9411
Epoch 119/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1548 - accuracy: 0.9382
Epoch 120/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1592 - accuracy: 0.9310
Epoch 121/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1419 - accuracy: 0.9397
Epoch 122/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1228 - accuracy: 0.9535
Epoch 123/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1514 - accuracy: 0.9411
Epoch 124/128
========================] - 5s 222ms/step - loss: 0.1070 - accuracy: 0.9608
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 125/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1450 - accuracy: 0.9433
Epoch 126/128
22/22 [==============================] - 5s 222ms/step - loss: 0.1174 - accuracy: 0.9557
Epoch 127/128
22/22 [==============================] - 5s 221ms/step - loss: 0.1165 - accuracy: 0.9622
Epoch 128/128
22/22 [==============================] - 5s 219ms/step - loss: 0.1312 - accuracy: 0.9477
```

Out[18]:

```
<keras.src.callbacks.History at 0x1c1a1f0ef80>
```

In [19]:

```
#Evaluating our second model and reporting the accuracy
scores2 = model2.evaluate(X_test, y_test)
```

```
19/19 [==============================] - 1s 28ms/step - loss: 0.5912 - accuracy: 0.8169
```

In [20]:

```
# do it here

#Making some predictions
predictions2 = model2.predict(X_test)

#Setting a 0.5 threshold
threshold_predictions2 = (predictions2 > 0.5).astype(int)

#Modifying our array to better fit the prediction function below
test2 = threshold_predictions2.tolist()
real_predictions2 = []

for xs in test2:
    for x in xs:
        real_predictions2.append(x)
```

```
19/19 [==============================] - 1s 28ms/step
```

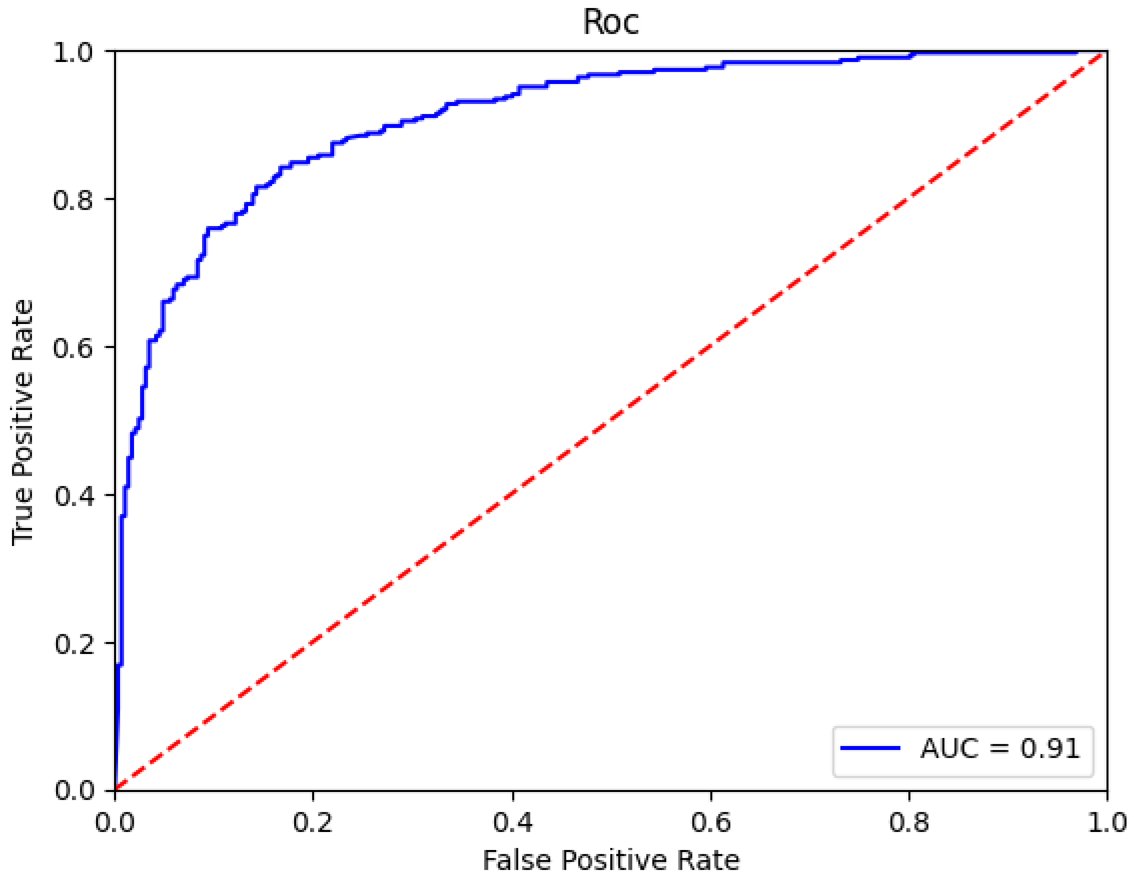
In [21]:

```
# do it here

#Using our premade function to display some predicted images
show_some_prediction(9, X_test, real_predictions, data[2])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```


In [22]:

```python
#Calculating our roc
fpr, tpr, threshold = roc_curve(y_test, predictions2)
roc_auc = auc(fpr, tpr)

#Plotting and displaying our roc curve
plt.title('Roc')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




According do the epochs, our model seemed more accurate but after checking the score, it was about the same as our first model. I'd trust the epochs more as more convolutions should produce a better estimate of our images.

```
In [ ]:
```