CLASSIFICATION CNN

STEPS -

- Create 3 folder in your desktop
- Training, Testing, Validation
- Inside training create 2 folder as h
- paste all the photo in testing part

In [1]:
```
#pip install tensorflow
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator #it will
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
img= image.load_img(r"C:\Users\hp\Downloads\Doctor-with-female-patient.jpg")
```

In [4]:
```
plt.imshow(img)
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x1cbc95c9b90>
```

In [5]:
```
img_arr=cv2.imread(r"C:\Users\hp\Downloads\Doctor-with-female-patient.jpg")
img_arr
# 3 dimension metrics are cre
# the value ranges from 0-255
```

Out[5]:
```
array([[[226, 226, 220],
        [226, 226, 220],
        [226, 226, 220],
        ...,
        [230, 227, 219],
        [231, 228, 220],
        [231, 228, 220]],

       [[226, 226, 220],
        [226, 226, 220],
        [226, 226, 220],
        ...,
        [231, 228, 220],
        [232, 229, 221],
        [232, 229, 221]],

       [[226, 226, 220],
        [226, 226, 220],
        [226, 226, 220],
        ...,
        [232, 229, 221],
        [233, 230, 222],
        [233, 230, 222]],

       ...,

       [[209, 198, 214],
        [209, 198, 214],
        [210, 199, 215],
        ...,
        [ 90,  88, 100],
        [ 90,  88, 100],
        [ 90,  88, 100]],

       [[209, 199, 212],
        [209, 199, 212],
        [210, 200, 213],
        ...,
        [ 90,  88, 100],
        [ 90,  88, 100],
        [ 90,  88, 100]],

       [[209, 199, 211],
        [209, 199, 211],
        [210, 200, 212],
        ...,
        [ 90,  88, 100],
        [ 90,  88, 100],
        [ 90,  88, 100]]], dtype=uint8)
```

In [6]:
```
img_arr.shape # height,width and RGB of image
```

Out[6]:
```
(1333, 2000, 3)
```

In [7]:
```
train=ImageDataGenerator(rescale= 1/255)
validation= ImageDataGenerator(rescale =1/255)
# to scale all the images i need to divide with 255
# we need to resize the image using 200, 200 pixel
```

In [9]:
```
train_dataset=train.flow_from_directory(r"C:\Users\hp\OneDrive\Documents\tra

validation_dataset=validation.flow_from_directory(r"C:\Users\hp\OneDrive\Doc
                                                  batch_size=3,class_mode='b
```

```
Found 15 images belonging to 2 classes.
Found 6 images belonging to 2 classes.
```

In [10]:
```
train_dataset.class_indices
```

Out[10]:
```
{'healthy kidney': 0, 'stone kidney': 1}
```

In [11]:
```
train_dataset.classes
```

Out[11]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

In [39]:
```
# apply convention layer  and maxpooling
model= tf.keras.models.Sequential([tf.keras.layers.Conv2D(16,(3,3),activatio
                                   tf.keras.layers.MaxPool2D(2,2),
                                ##
                                   tf.keras.layers.Conv2D(32,(3,3),activatio
                                   tf.keras.layers.MaxPool2D(2,2),
                                ##
                                   tf.keras.layers.Conv2D(64,(3,3),activatio
                                   tf.keras.layers.MaxPool2D(2,2),
                                ##
                                   tf.keras.layers.Conv2D(128,(3,3),activati
                                   tf.keras.layers.MaxPool2D(2,2),
                                ##
                                   tf.keras.layers.Flatten(),
                                ##
                                   tf.keras.layers.Dense(512,activation='rel
                                ##
                                   tf.keras.layers.Dense(1,activation='sigmo


                                    ])
```

In [40]:
```
model.compile(loss='binary_crossentropy',optimizer=tf.keras.optimizers.RMSpr
```

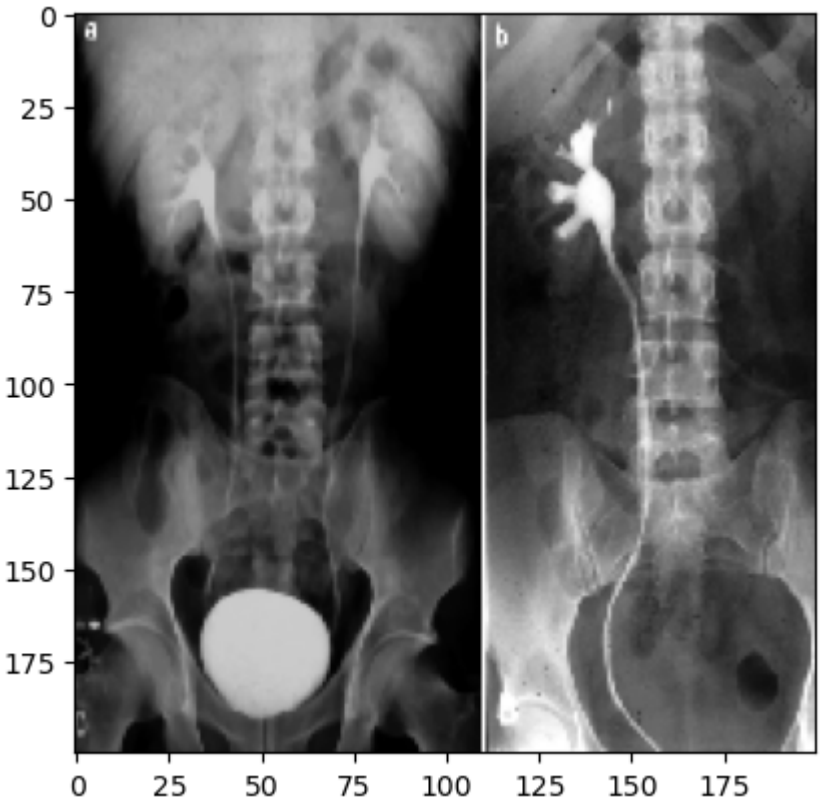
In [43]:
```
model_fit=model.fit(train_dataset,epochs=20,validation_data=validation_datas
```

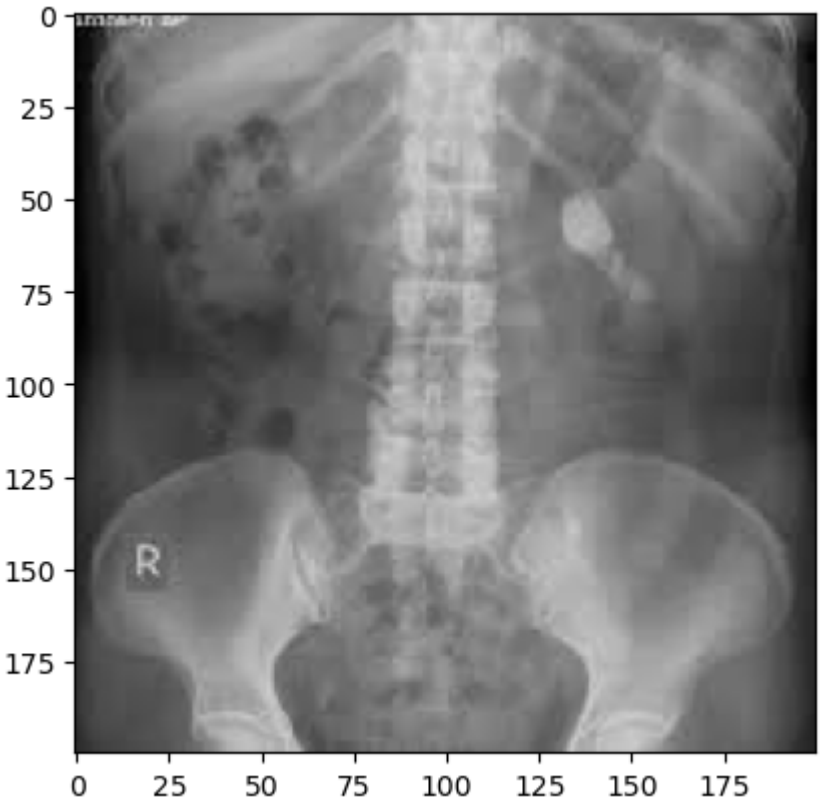
```
Epoch 1/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.8269 - loss: 0.5532 - v
al_accuracy: 0.5000 - val_loss: 0.5332
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.5565 - loss: 0.5356 - v
al_accuracy: 0.6667 - val_loss: 0.5628
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.7491 - loss: 0.3989 - v
al_accuracy: 0.6667 - val_loss: 0.6140
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.8954 - loss: 0.3721 - v
al_accuracy: 0.6667 - val_loss: 0.7153
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.7130 - loss: 0.4208 - v
al_accuracy: 0.5000 - val_loss: 0.5793
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.7796 - loss: 0.3925 - v
al_accuracy: 0.6667 - val_loss: 0.9833
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.7963 - loss: 0.4835 - v
al_accuracy: 0.8333 - val_loss: 0.6861
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.8546 - loss: 0.2779 - v
al_accuracy: 0.6667 - val_loss: 0.8709
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.8593 - loss: 0.2864 - v
al_accuracy: 0.6667 - val_loss: 1.0154
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - accuracy: 1.0000 - loss: 0.1335 - v
al_accuracy: 0.8333 - val_loss: 1.4524
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 1.0000 - loss: 0.0782 - v
al_accuracy: 0.6667 - val_loss: 4.4398
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.7722 - loss: 0.6116 - v
al_accuracy: 0.6667 - val_loss: 0.6163
Epoch 13/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.8398 - loss: 0.3572 - v
al_accuracy: 0.6667 - val_loss: 0.8430
Epoch 14/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9639 - loss: 0.2031 - v
al_accuracy: 0.6667 - val_loss: 1.2060
Epoch 15/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.9176 - loss: 0.1421 - v
al_accuracy: 0.6667 - val_loss: 1.6190
Epoch 16/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - accuracy: 1.0000 - loss: 0.0674 - v
al_accuracy: 0.8333 - val_loss: 2.1574
Epoch 17/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 1.0000 - loss: 0.0752 - v
al_accuracy: 0.8333 - val_loss: 2.7059
Epoch 18/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 1.0000 - loss: 0.0141 - v
al_accuracy: 0.8333 - val_loss: 3.2953
Epoch 19/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 1.0000 - loss: 0.0052 - v
al_accuracy: 0.8333 - val_loss: 3.3406
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 1.0000 - loss: 0.0028 - v
al_accuracy: 0.8333 - val_loss: 3.4127
```

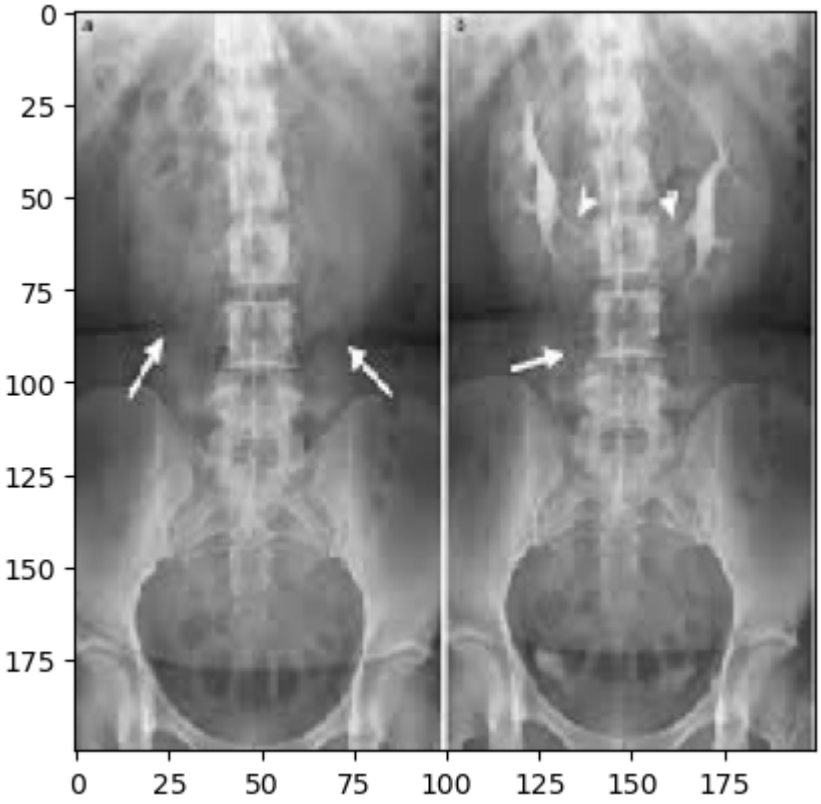
In [44]:
```
dir_path=r"C:\Users\hp\OneDrive\Documents\testing"
for i in os.listdir(dir_path):
    print(i)
```

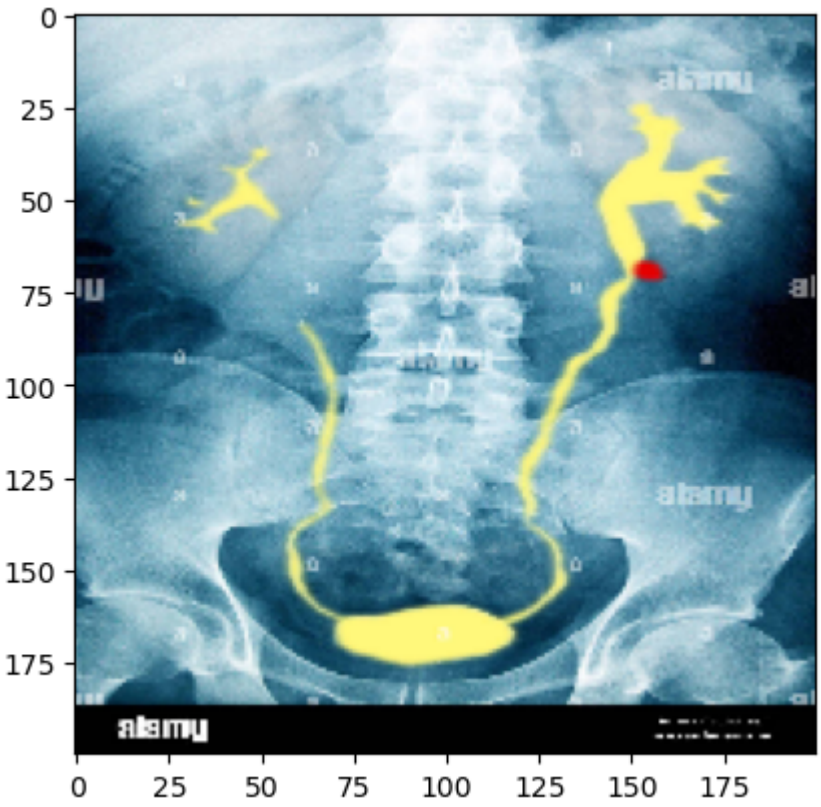
```
978-3-540-87597-0_2_Fig39_HTML.png
download (1).jpeg
download.jpeg
kidney-stone-x-ray-DWT8DT.jpg
Normal-Kidney-Ureter-and-Bladder-X-ray-with-no-radiopaque-foreign-body-see
n.png
shutterstock_632967740.webp
Staghorn_calc.gif
stock-photo-a-kub-xray-film-of-a-patient-with-a-large-urinary-bladder-stone
-and-multiple-left-kidney-stones-1924145921.jpg
x-ray-of-spinal-cord-and-normal-kidneys-HRF8GX.jpg
```

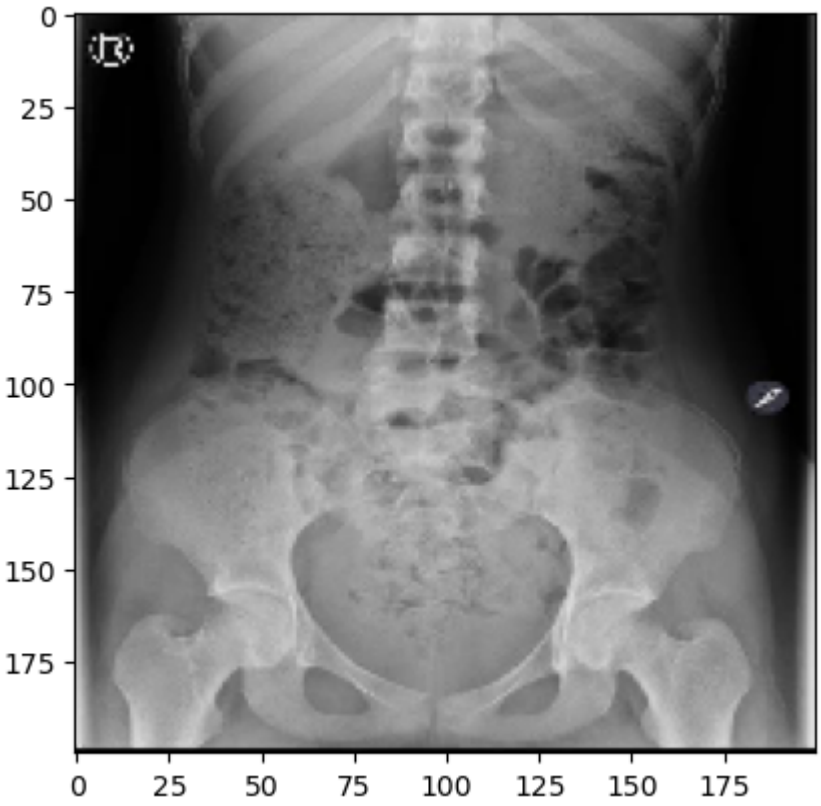
In [46]:
```
dir_path=r"C:\Users\hp\OneDrive\Documents\testing"
for i in os.listdir(dir_path):
    img=image.load_img(dir_path+ '//'+i,target_size=(200,200))
    plt.imshow(img)
    plt.show()
```

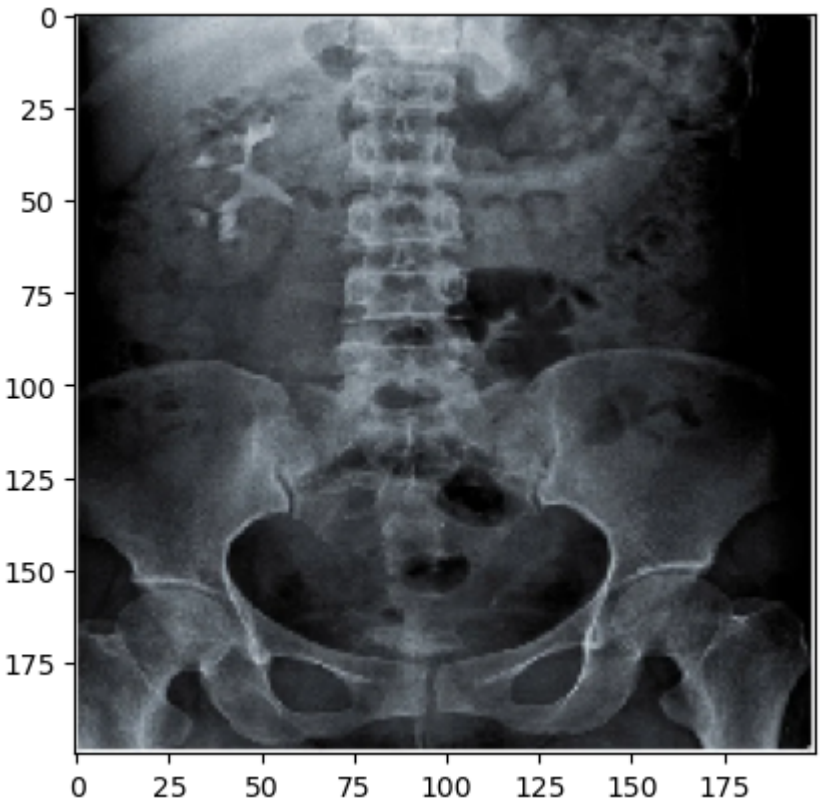


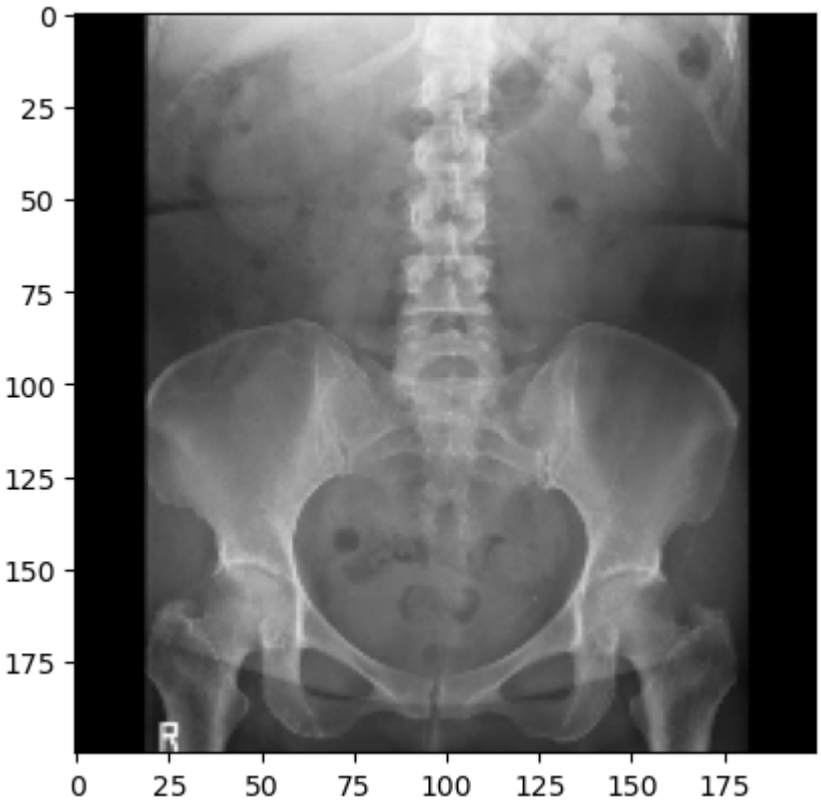


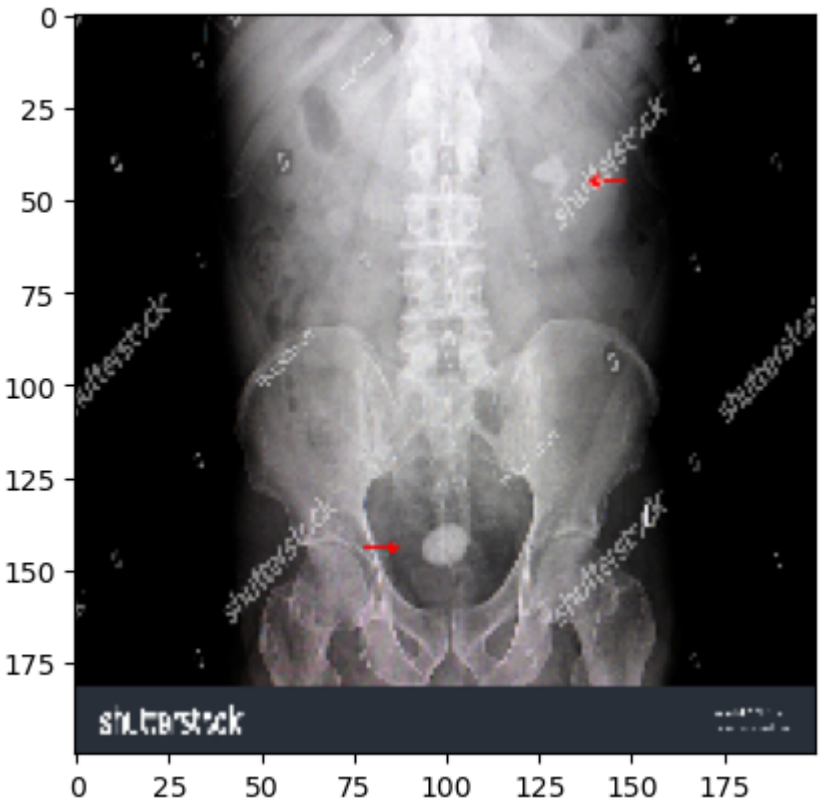

0
25
50
75
100
125
150
175
0
25
50
75
100
125
150
175

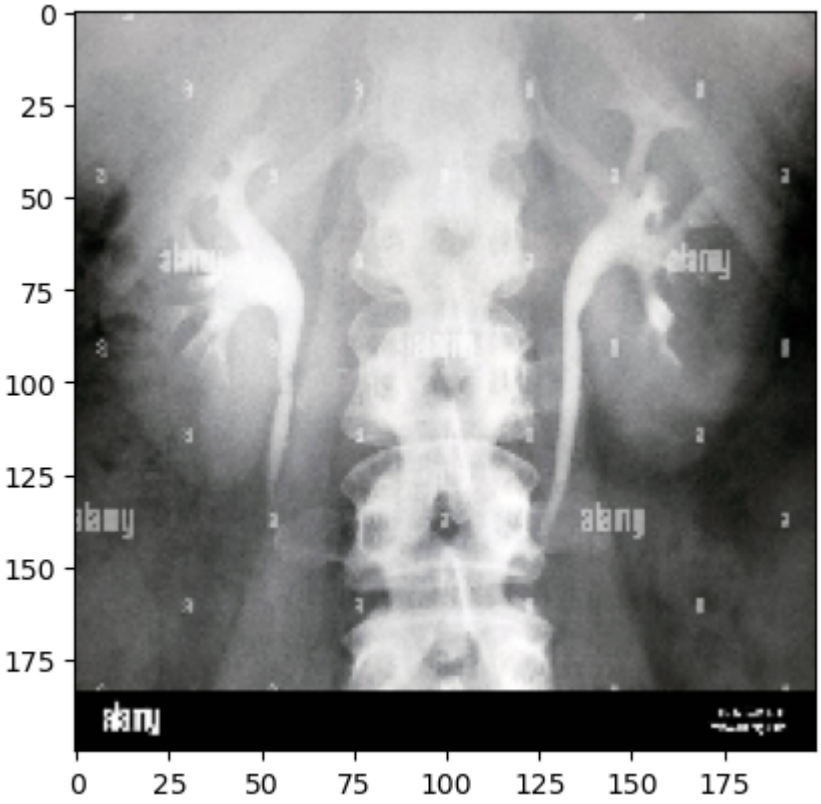

0
25
50
75
100
125
150
175
0
25
50
75
100
125
150
175


0
25
50
75
100
125
150
175
0
25
50
75
100
125
150
175

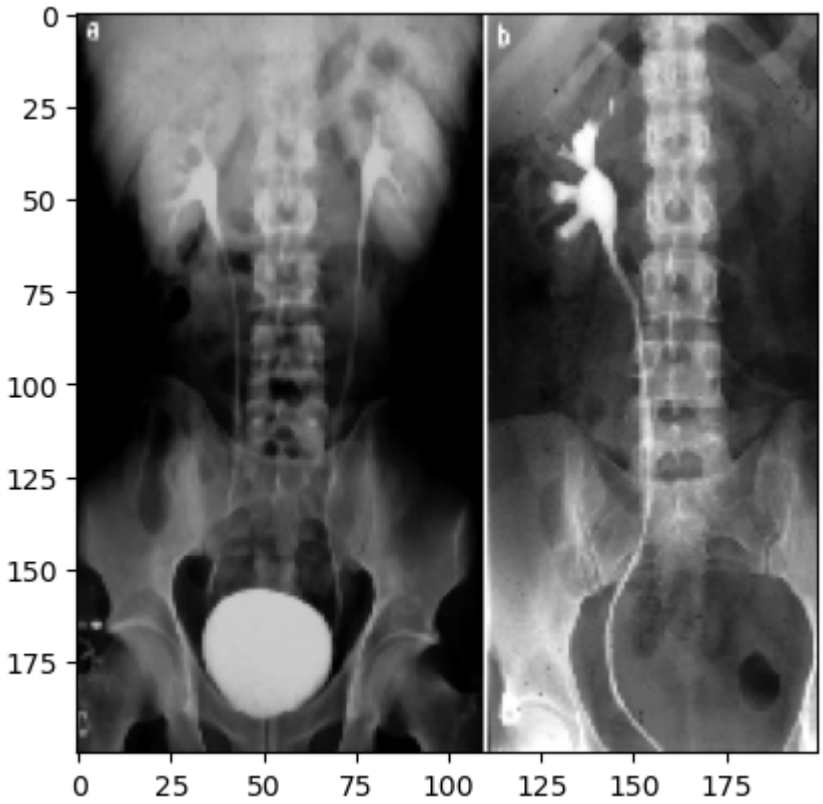

0
25
50
75
100
125
150
175
0
25
50
75
100
125
150
175


0
25
50
75
100
125
150
175
R
0
25
50
75
100
125
150
175

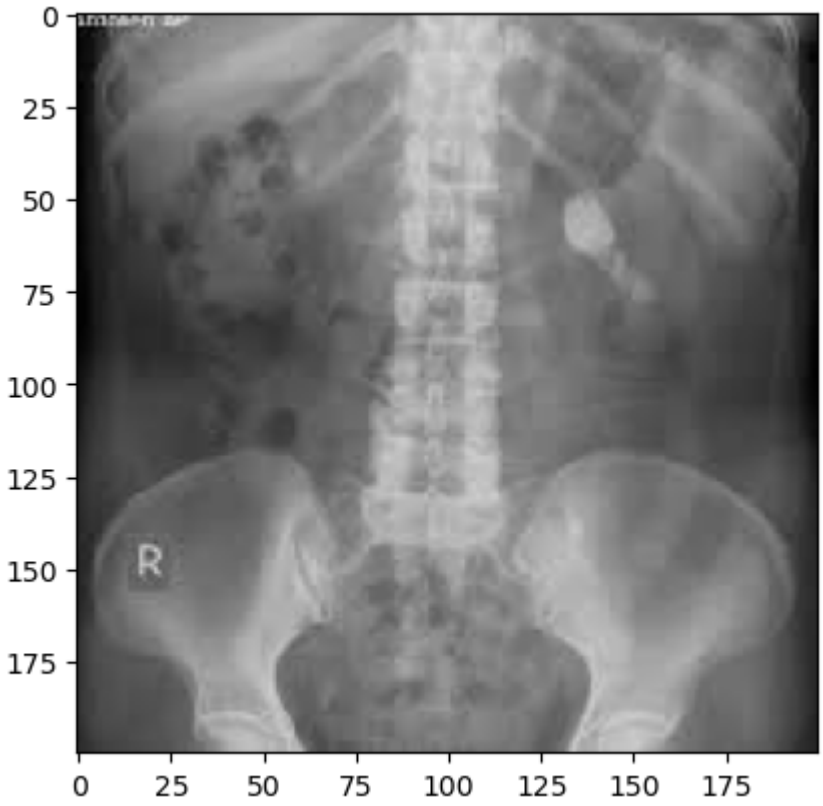

0
25
50
75
100
125
150
175
shutterstock
0
25
50
75
100
125
150
175


0
25
50
75
100
125
150
175
0
25
50
75
100
125
150
175

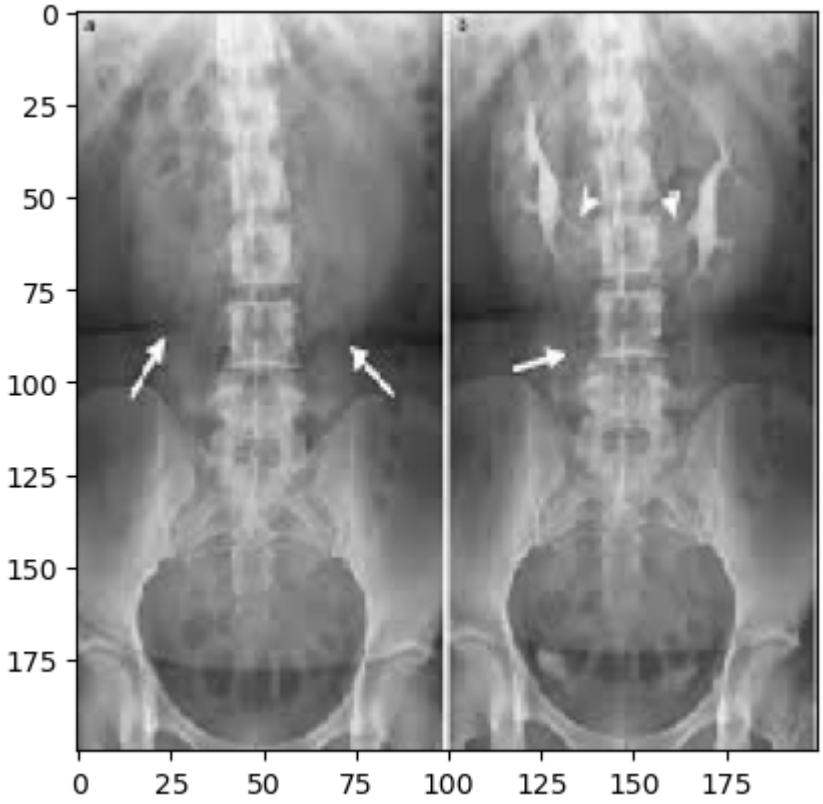
In [47]:
```
dir_path=r"C:\Users\hp\OneDrive\Documents\testing"
for i in os.listdir(dir_path ):
    img = image.load_img(dir_path+ '//'+i, target_size = (200,200))
    plt.imshow(img)
    plt.show()

    x= image.img_to_array(img)
    x=np.expand_dims(x,axis = 0)
    images = np.vstack([x])

    val = model.predict(images)
    if val == 0:
        print( 'HEALTHY KIDNEY')
    else:
        print('STONE KIDNEY')
```



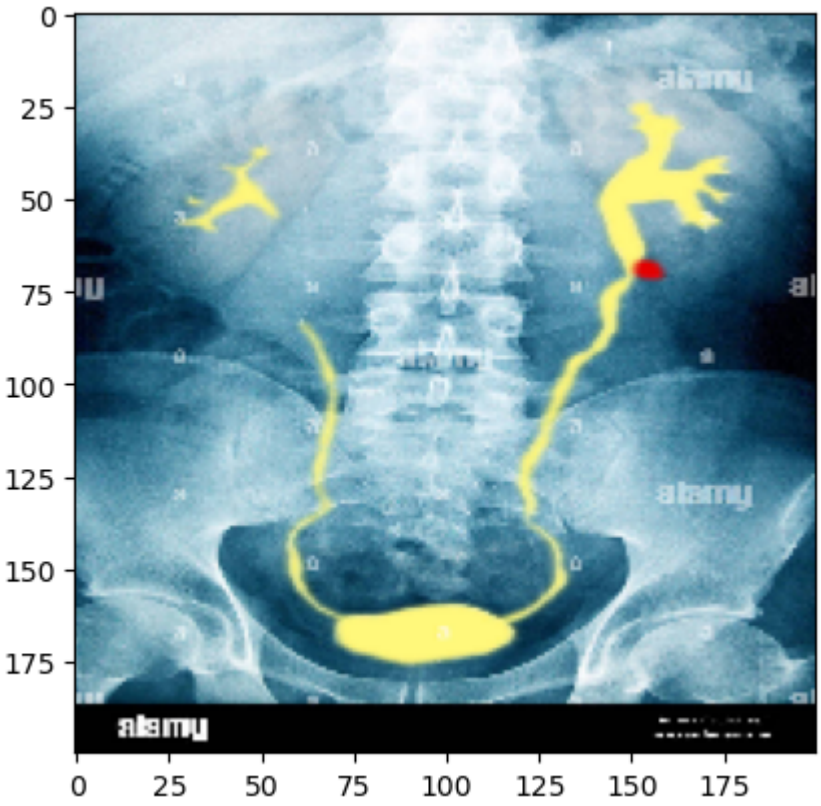

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step
HEALTHY KIDNEY
```



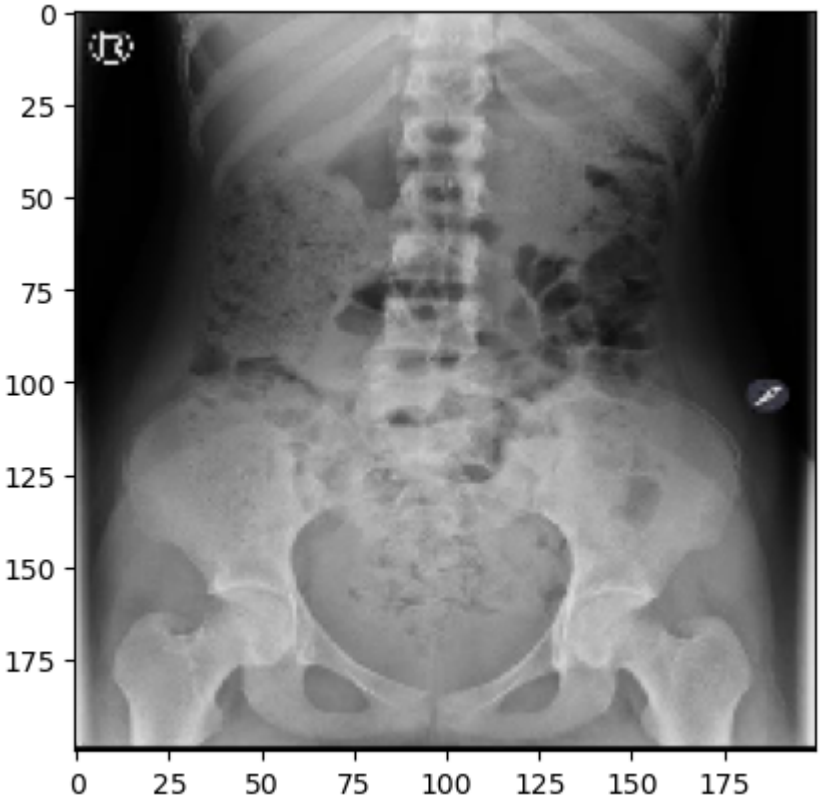

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
HEALTHY KIDNEY
```



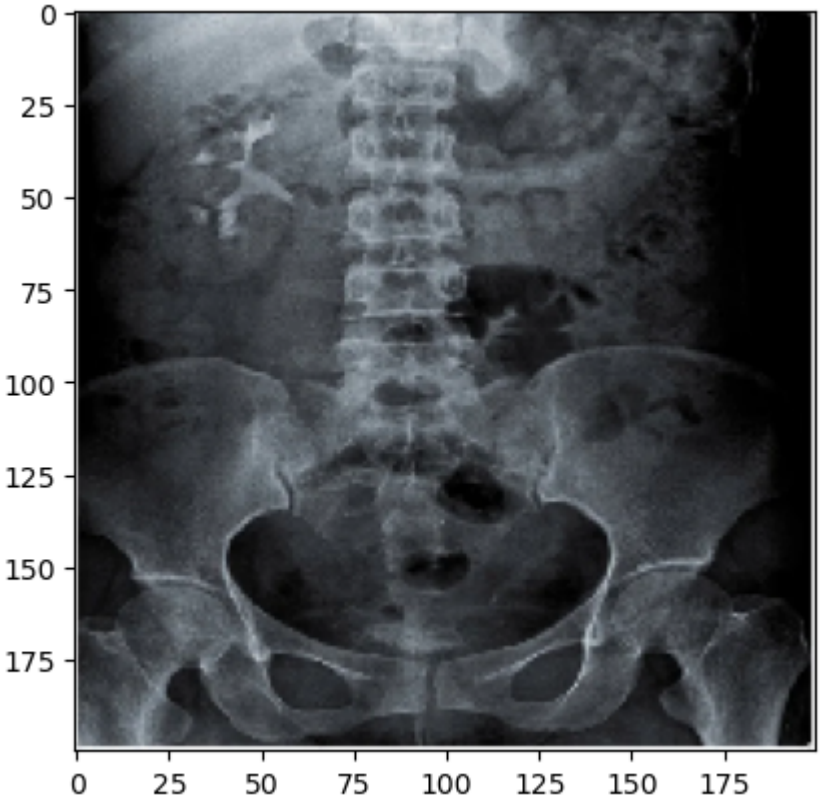

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
STONE KIDNEY
```



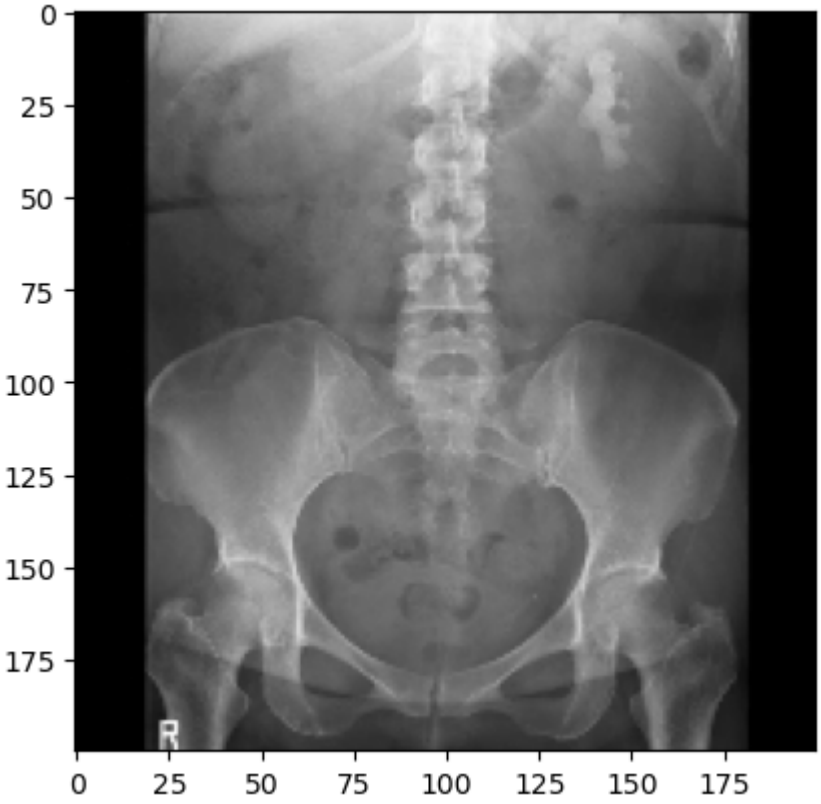

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
STONE KIDNEY
```



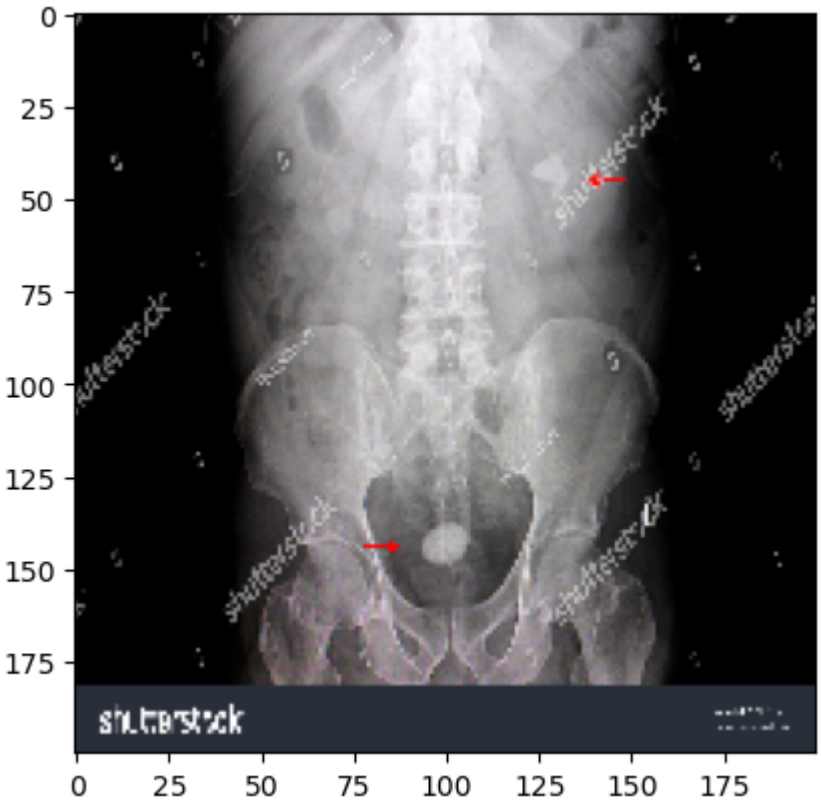

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
HEALTHY KIDNEY
```



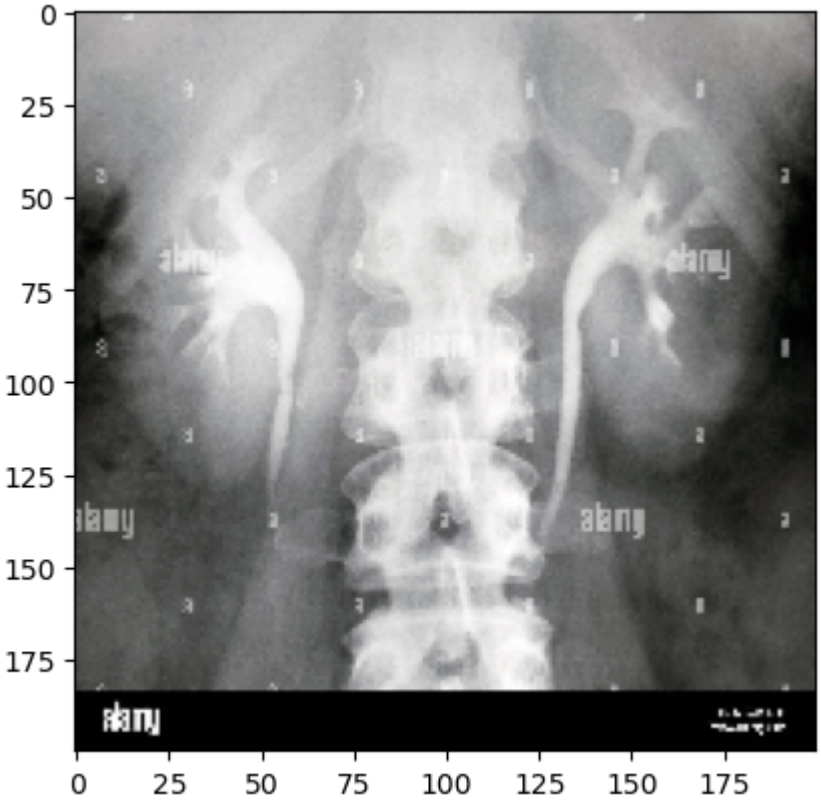

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
STONE KIDNEY
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
STONE KIDNEY
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
STONE KIDNEY
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
HEALTHY KIDNEY
```

training and testing of healthy kidney and stone kidneys xrays is done by binary classification of cnn, aquired accuracy 99%

In [ ]: