```
import cv2
import imghdr
import os
import tensorflow as tf
```

REMOVING DODGY IMAGES

```
data_dir = '/kaggle/input/xray-fractured-data/FRACTURE-XRAY'
```

```
image_exts = ['jpeg' , 'jpg' , 'bmp' , 'png']
```

```
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in list: {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image: {}'.format(image_path))
            os.remove(image_path)
```

LOAD DATA

```
tf.data.Dataset??
```

Show hidden output

```
import numpy as np
from matplotlib import pyplot as plt
```

```
 tf.keras.utils.image_dataset_from_directory('/kaggle/input/xray-fractured-data/FRACTURE-XRAY')
```

```
Found 9463 files belonging to 2 classes.
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
```

```
tf.keras.utils.image_dataset_from_directory??
```

Show hidden output

```
data = tf.keras.utils.image_dataset_from_directory('/kaggle/input/xray-fractured-data/FRACTURE-XRAY')
```

```
Found 9463 files belonging to 2 classes.
```

```
data_iterator = data.as_numpy_iterator()
```

```
batch = data_iterator.next()
```

```
batch
```

Show hidden output

```
len(batch)
```

```
2
```

```
#images represented as numpy arrays
#class 1 = fractured
#class 0 =  non-fractured
batch[0].shape
```

```
(32, 256, 256, 3)
```

```
fig , ax= plt.subplots(ncols=4 , figsize=(20, 20))
for idx , img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```

Show hidden output

```
scaled = batch[0] / 255
```

```
scaled.max()
```

Show hidden output

PREPROCESS DATA

```
data =  data.map(lambda x, y: (x/255 , y))
```

```
scaled_iterator = data.as_numpy_iterator()
```

```
batch = scaled_iterator.next()
```

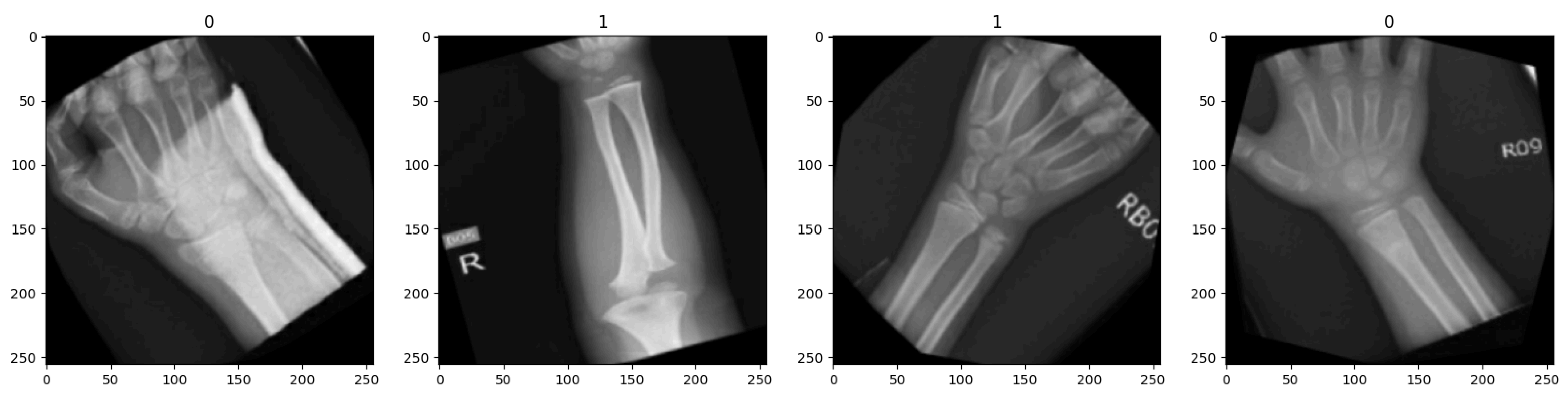
```
batch[0].min()
```

```
0.0
```

```
#Now our data is scaled and our images is between 0 and 1
batch[0].max()
```

```
1.0
```

```
fig , ax= plt.subplots(ncols=4 , figsize=(20, 20))
for idx , img in enumerate(batch[0][:4]):
    ax[idx].imshow(img)
    ax[idx].title.set_text(batch[1][idx])
```



SPLITTING DATA

```
len(data)
```

Show hidden output

```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)+1
test_size = int(len(data)*.1)+1
```

```
train_size+val_size+test_size
```

Show hidden output

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

DEEP MODEL

BUILD DEEP LEARNING MODEL

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D , MaxPooling2D , Dense , Flatten , Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

```
MaxPooling2D??
```

Show hidden output

```
model = Sequential()
```

```
#16 filters that scans over the image and extract the relevant information inside of this image and each filter will be 3*3 pixels in size ,and stride =1 this means that it wil one pixel each time
model.add(Conv2D(16, (3,3) , 1 , activation='relu' , input_shape=(256,256,3)))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3,3) , 1 , activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(16, (3,3) , 1 , activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile('adam' , loss=tf.losses.BinaryCrossentropy() , metrics=['accuracy'])
```

```
model.summary()
```

Show hidden output

TRAIN

```
logdir='logs'
```

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

```
hist = model.fit(train , epochs=8 , validation_data=val , callbacks=[tensorboard_callback])
```

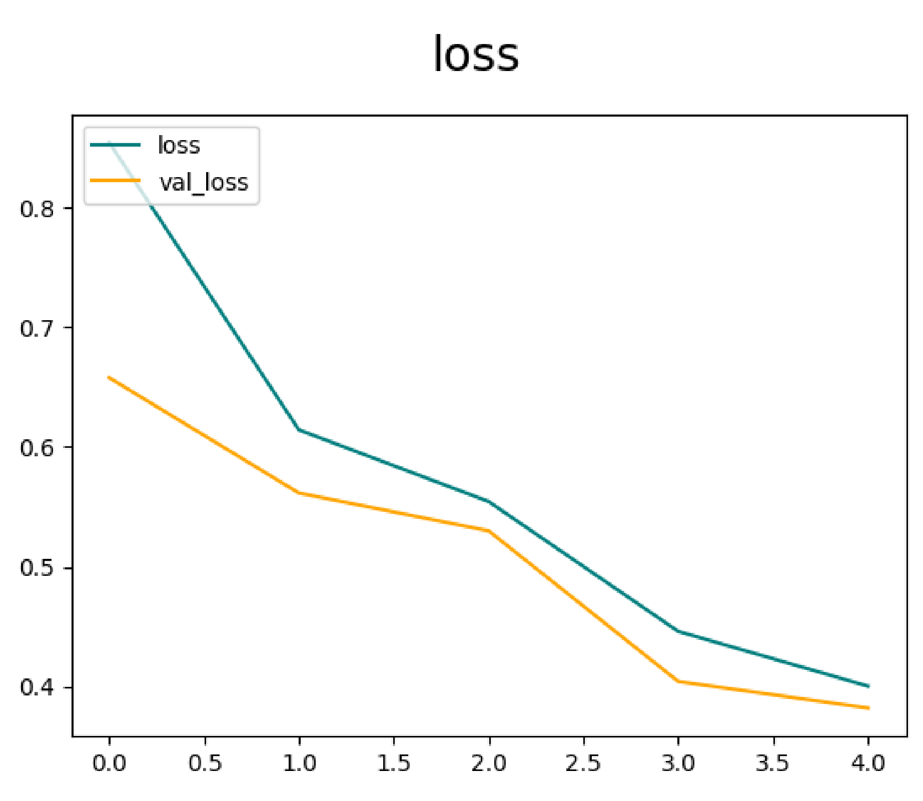
Show hidden output

```
hist.history
```

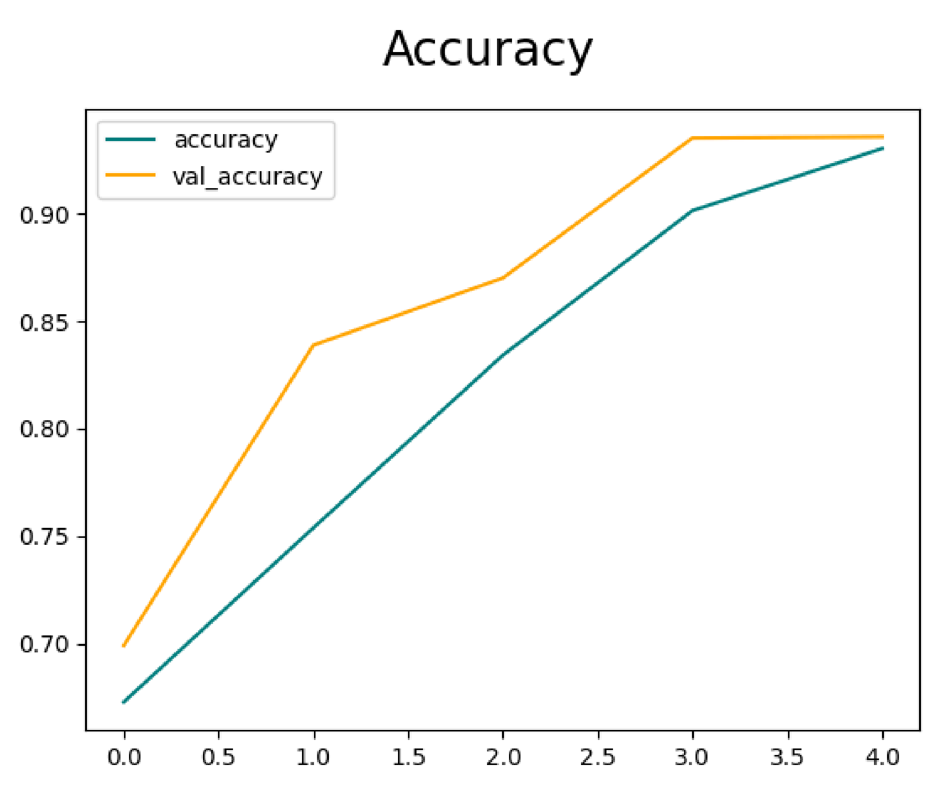
Show hidden output

PLOTTING PERFORMANCE

```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal' , label='loss')
plt.plot(hist.history['val_loss'], color='orange' , label='val_loss')
fig.suptitle('loss' , fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal' , label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange' , label='val_accuracy')
fig.suptitle('Accuracy' , fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



EVALUATE PERFORMANCE

```
from tensorflow.keras.metrics import Precision , Recall , BinaryAccuracy
```

```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

```
for batch in test.as_numpy_iterator():
    X,y = batch
    yhat = model.predict(X)
    pre.update_state(y , yhat)
    re.update_state(y , yhat)
    acc.update_state(y, yhat)
```

Show hidden output

+ Code + Text

```
print(f'Precision:{pre.result().numpy()} ,Recall:{re.result().numpy()} ,Accuracy: {acc.result().numpy()}')
```

```
Precision:0.9239130616188049 ,Recall:0.9100642204284668 ,Accuracy: 0.9162132740020752
```

SAVE THE MODEL