In [1]:
```
!pip install numpy -q
!pip install pandas -q
!pip install matplotlib -q
!pip install tensorflow -q

!pip install opendatasets -q
```

In [2]:
```
# import necessary libraries
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import time

import opendatasets as od
```

Load Dataset

In [3]:
```
# download dataset

od.download("https://www.kaggle.com/datasets/vuppalaadithyasairam/bon
```

```
Skipping, found downloaded files in ".\bone-fracture-detection-using-xra
ys" (use force=True to force download)
```

In [4]:
```
BATCH_SIZE = 32
IMAGE_SIZE = (224,224)
```

In [5]:
```
train_data_dir = "C:/Users/reddy/Downloads/archive/archive (6)/train"
test_data_dir = "C:/Users/reddy/Downloads/archive/archive (6)/val"

train_data = tf.keras.utils.image_dataset_from_directory(train_data_d
                                                         batch_size=B
                                                         image_size=I
                                                         subset='trai
                                                         validation_s
                                                         seed=42)

validation_data = tf.keras.utils.image_dataset_from_directory(train_d
                                                         batch_size=B
                                                         image_size=I
                                                         subset='vali
                                                         validation_s
                                                         seed=42)

test_data = tf.keras.utils.image_dataset_from_directory(test_data_dir
                                                         batch_size=B
                                                         image_size=I
```

```
Found 8863 files belonging to 2 classes.
Using 7977 files for training.
Found 8863 files belonging to 2 classes.
Using 886 files for validation.
Found 600 files belonging to 2 classes.
```

In [6]:
```
class_names = train_data.class_names
class_names
```

Out[6]:
```
['fractured', 'not fractured']
```

In [7]:
```
for image_batch,label_batch in train_data.take(1):
    print(image_batch.shape)
    print(label_batch.shape)
```

```
(32, 224, 224, 3)
(32,)
```

In [8]:
```
# plot data sample
plt.figure(figsize=(10,4))
for image,label in train_data.take(1):
    for i in range(10):
        ax = plt.subplot(2,5,i+1)
        plt.imshow(image[i].numpy().astype('uint8'))
        plt.title(class_names[label[i]])
        plt.axis('off')
```

Scaling Images

In [9]:
```
for image,label in train_data.take(1):
    for i in range(1):
      print(image)
```

```
...
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 ...
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 ...
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]]], shape=(32, 224, 224, 3), dtype=float32)
```

In [10]:
```
train_data = train_data.map(lambda x,y:(x/255,y))
validation_data = validation_data.map(lambda x,y:(x/255,y))
test_data = test_data.map(lambda x,y:(x/255,y))
```

Data Augmentation

In [11]:
```
data_augmentation = tf.keras.Sequential(
  [
    tf.keras.layers.RandomFlip("horizontal",input_shape=(224,224,3)),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
  ]
)
```

Model Building

In [12]:
```
model = tf.keras.models.Sequential()

model.add(data_augmentation)

model.add(tf.keras.layers.Conv2D(32, kernel_size=3, activation='relu'
model.add(tf.keras.layers.MaxPooling2D())

model.add(tf.keras.layers.Conv2D(64, kernel_size=3, activation='relu'
model.add(tf.keras.layers.MaxPooling2D())

model.add(tf.keras.layers.Conv2D(128, kernel_size=3, activation='relu
model.add(tf.keras.layers.MaxPooling2D())

model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))

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

In [13]:
```
1 model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 dropout (Dropout)           (None, 26, 26, 128)       0

 batch_normalization (Batch  (None, 26, 26, 128)       512
 Normalization)

 flatten (Flatten)           (None, 86528)             0

 dense (Dense)               (None, 128)               11075712

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 32)                4128

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 11190145 (42.69 MB)
Trainable params: 11189889 (42.69 MB)
Non-trainable params: 256 (1.00 KB)
_________________________________________________________________
```

In [14]:
```
1 model.compile(optimizer=tf.keras.optimizers.Adam(),
2               loss=tf.keras.losses.BinaryCrossentropy(),
3               metrics=['accuracy'])
```

Model Training

In [15]:

```
start_time = time.time()

history = model.fit(train_data,
                    epochs=20,
                    validation_data=validation_data)

end_time = time.time()
```

```
Epoch 1/20
250/250 [==============================] - 780s 3s/step - loss: 0.6617 -
accuracy: 0.5917 - val_loss: 0.7136 - val_accuracy: 0.4707
Epoch 2/20
250/250 [==============================] - 868s 3s/step - loss: 0.6127 -
accuracy: 0.6296 - val_loss: 0.9537 - val_accuracy: 0.4289
Epoch 3/20
250/250 [==============================] - 454s 2s/step - loss: 0.5645 -
accuracy: 0.6847 - val_loss: 0.6509 - val_accuracy: 0.6456
Epoch 4/20
250/250 [==============================] - 648s 3s/step - loss: 0.5071 -
accuracy: 0.7419 - val_loss: 0.5667 - val_accuracy: 0.7167
Epoch 5/20
250/250 [==============================] - 499s 2s/step - loss: 0.4439 -
accuracy: 0.7833 - val_loss: 0.4721 - val_accuracy: 0.7889
Epoch 6/20
250/250 [==============================] - 517s 2s/step - loss: 0.3865 -
accuracy: 0.8237 - val_loss: 0.8982 - val_accuracy: 0.6806
Epoch 7/20
250/250 [==============================] - 512s 2s/step - loss: 0.3508 -
accuracy: 0.8437 - val_loss: 0.2782 - val_accuracy: 0.8736
Epoch 8/20
250/250 [==============================] - 525s 2s/step - loss: 0.3113 -
accuracy: 0.8676 - val_loss: 0.2339 - val_accuracy: 0.9029
Epoch 9/20
250/250 [==============================] - 508s 2s/step - loss: 0.2530 -
accuracy: 0.8965 - val_loss: 0.1442 - val_accuracy: 0.9515
Epoch 10/20
250/250 [==============================] - 507s 2s/step - loss: 0.2255 -
accuracy: 0.9101 - val_loss: 0.1450 - val_accuracy: 0.9424
Epoch 11/20
250/250 [==============================] - 513s 2s/step - loss: 0.1773 -
accuracy: 0.9328 - val_loss: 0.1417 - val_accuracy: 0.9379
Epoch 12/20
250/250 [==============================] - 505s 2s/step - loss: 0.1842 -
accuracy: 0.9269 - val_loss: 0.4337 - val_accuracy: 0.8476
Epoch 13/20
250/250 [==============================] - 501s 2s/step - loss: 0.1490 -
accuracy: 0.9416 - val_loss: 0.8240 - val_accuracy: 0.7506
Epoch 14/20
250/250 [==============================] - 505s 2s/step - loss: 0.1404 -
accuracy: 0.9475 - val_loss: 0.0517 - val_accuracy: 0.9797
Epoch 15/20
250/250 [==============================] - 508s 2s/step - loss: 0.1326 -
accuracy: 0.9524 - val_loss: 0.6310 - val_accuracy: 0.8002
Epoch 16/20
250/250 [==============================] - 501s 2s/step - loss: 0.1053 -
accuracy: 0.9608 - val_loss: 0.1139 - val_accuracy: 0.9605
Epoch 17/20
250/250 [==============================] - 511s 2s/step - loss: 0.1062 -
accuracy: 0.9615 - val_loss: 0.0161 - val_accuracy: 0.9966
Epoch 18/20
250/250 [==============================] - 503s 2s/step - loss: 0.0913 -
accuracy: 0.9668 - val_loss: 0.0508 - val_accuracy: 0.9842
Epoch 19/20
250/250 [==============================] - 505s 2s/step - loss: 0.0851 -
accuracy: 0.9682 - val_loss: 0.0512 - val_accuracy: 0.9808
```

```
Epoch 20/20
250/250 [==============================] - 452s 2s/step - loss: 0.0757 -
accuracy: 0.9737 - val_loss: 0.0229 - val_accuracy: 0.9932
```

In [16]:

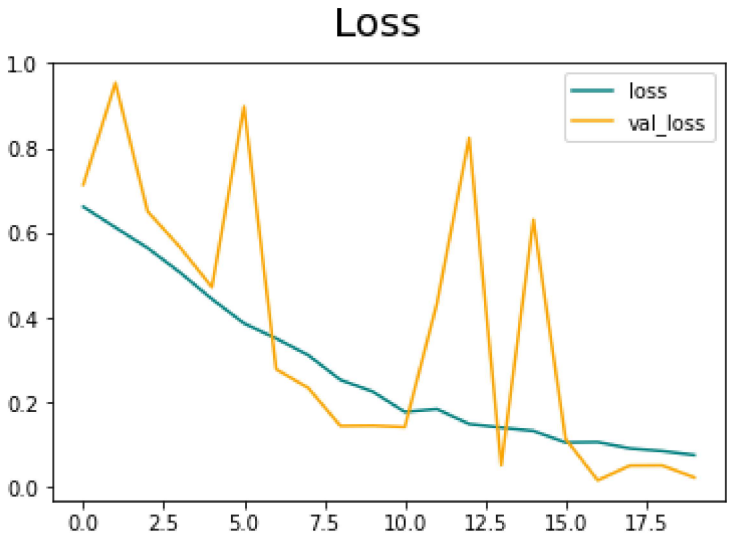
```
1 print(f'Total time for training {(end_time-start_time):.3f} seconds')
```

```
Total time for training 10822.695 seconds
```

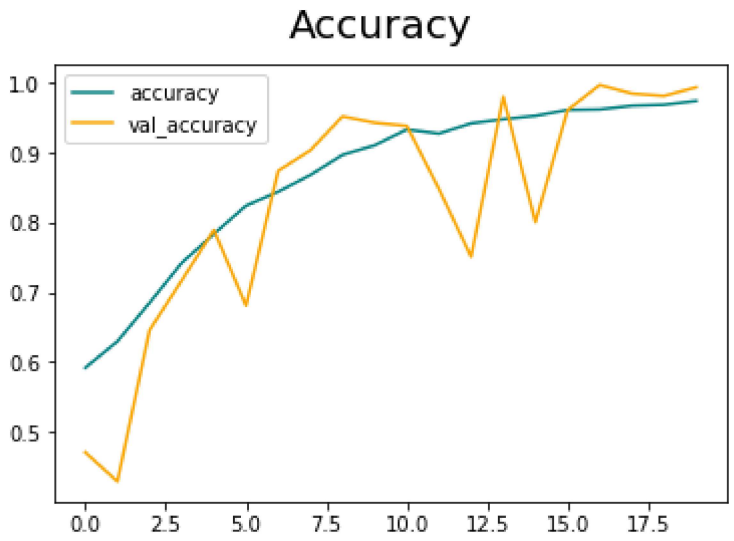
Performance Analysis

In [17]:

```
1 fig = plt.figure()
2 plt.plot(history.history['loss'], color='teal', label='loss')
3 plt.plot(history.history['val_loss'], color='orange', label='val_loss
4 fig.suptitle('Loss', fontsize=20)
5 plt.legend()
6 plt.show()
```



In [18]:
```
fig = plt.figure()
plt.plot(history.history['accuracy'], color='teal', label='accuracy')
plt.plot(history.history['val_accuracy'], color='orange', label='val_
fig.suptitle('Accuracy', fontsize=20)
plt.legend()
plt.show()
```



Model Evaluation

In [19]:
```
precision = tf.keras.metrics.Precision()
recall = tf.keras.metrics.Recall()
accuracy = tf.keras.metrics.BinaryAccuracy()
```

In [20]:
```
for batch in test_data.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)
    precision.update_state(y, yhat)
    recall.update_state(y, yhat)
    accuracy.update_state(y, yhat)
```

```
1/1 [==============================] - 1s 565ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 248ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 259ms/step
1/1 [==============================] - 0s 252ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 258ms/step
1/1 [==============================] - 0s 266ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 252ms/step
1/1 [==============================] - 0s 277ms/step
1/1 [==============================] - 0s 243ms/step
1/1 [==============================] - 0s 271ms/step
1/1 [==============================] - 0s 243ms/step
1/1 [==============================] - 0s 260ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 269ms/step
1/1 [==============================] - 0s 306ms/step
```

In [21]:
```
precision.result()
```

Out[21]: `<tf.Tensor: shape=(), dtype=float32, numpy=0.6052632>`

In [22]:
```
recall.result()
```

Out[22]: `<tf.Tensor: shape=(), dtype=float32, numpy=0.9583333>`

In [23]:
```
accuracy.result()
```

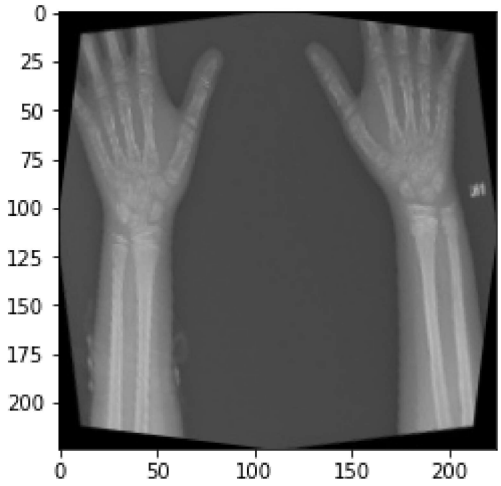
Out[23]: `<tf.Tensor: shape=(), dtype=float32, numpy=0.73333335>`

Test

In [24]:
```
!pip install opencv-python -q
```

In [25]:
```
import cv2
```

In [26]:
```
1 img = cv2.imread('C:/Users/reddy/Downloads/archive/archive (6)/train/
2 plt.imshow(img)
3 plt.show()
```



In [27]:
```
1 resized_image = tf.image.resize(img, IMAGE_SIZE)
2 scaled_image = resized_image/255
```

In [28]:
```
1 scaled_image.shape
```

Out[28]:
```
TensorShape([224, 224, 3])
```

In [29]:
```
1 np.expand_dims(scaled_image, 0).shape
```

Out[29]:
```
(1, 224, 224, 3)
```

In [30]:
```
1 yhat = model.predict(np.expand_dims(scaled_image, 0))
```

```
1/1 [==============================] - 0s 49ms/step
```

In [31]:
```
1 yhat
```

Out[31]:
```
array([[1.2999705e-13]], dtype=float32)
```

In [32]:
```
1 class_names
```

Out[32]:
```
['fractured', 'not fractured']
```

In [33]:
```
1 if yhat > 0.5:
2     print(f'{class_names[1]}')
3 else:
4     print(f'{class_names[0]}')
```

```
fractured
```

In [34]:
```
!pip install opencv-python-headless --user -q

```

In [35]:
```
pip install opencv-python-headless

```

```
Requirement already satisfied: opencv-python-headless in c:\users\reddy
\appdata\roaming\python\python39\site-packages (4.9.0.80)
Requirement already satisfied: numpy>=1.17.3 in c:\users\reddy\anaconda3
\lib\site-packages (from opencv-python-headless) (1.22.4)
Note: you may need to restart the kernel to use updated packages.
```

In [36]:
```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import cv2
import numpy as np

```

In [ ]:

```
class FractureDetectorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Fracture Detector")

        self.canvas = tk.Canvas(root, width=800, height=400)
        self.canvas.pack()

        self.load_button = tk.Button(root, text="Load Image", command
        self.load_button.pack()

        self.result_label = tk.Label(root, text="")
        self.result_label.pack()

    def load_image(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            self.process_image(file_path)

    def process_image(self, file_path):
        img = cv2.imread(file_path)
        resized_img = cv2.resize(img, (224, 224))
        scaled_img = resized_img / 255.0  # Normalize image

        yhat = model.predict(np.expand_dims(scaled_img, axis=0))[0][0
        predicted_class = class_names[int(round(yhat))]

        self.display_image(file_path)
        self.result_label.config(text=f"Predicted: {predicted_class}"

    def display_image(self, file_path):
        image = Image.open(file_path)
        image = image.resize((300, 300), Image.ANTIALIAS)
        self.img_tk = ImageTk.PhotoImage(image)
        self.canvas.create_image(0, 0, anchor=tk.NW, image=self.img_t

if __name__ == "__main__":
    root = tk.Tk()
    app = FractureDetectorApp(root)
    root.mainloop()

```

```
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 49ms/step
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```