***

Minor 2 Project -Cse-Ai&Ml

Tittle: Implementation of image processing Algorithms for fracture detection on Different human body parts.

Team Members: 1. Priya Mittal , 2. Parth Madan

Project Mentor: Dr. Niharika Singh

***

Import Libraries (Requirement Analysis Part)

In [1]:
```
import os
import glob
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
warnings.filterwarnings("ignore")
import sklearn.metrics as metrics
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adamax
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import load_img,img_to_array
from sklearn.metrics import roc_auc_score, f1_score, classification_report, confusion_matrix
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
```

IMPORT FRACTURE IMAGES

In [2]:
```
images_fr = []
folder = r'D:\Fracture Detection System (Minor2)\A Dataset\All Fractured'
for filename in os.listdir(folder):
    try:
        img = mpimg.imread(os.path.join(folder, filename))
        if img is not None:
            images_fr.append(img)
    except:
        print('Cant import ' + filename)
images_fr = np.asarray(images_fr)
```

```
Cant import Comminuted-patellar-fracture-Preoperative-X-ray-a-postoperative-X-ray-b-final.png
Cant import Lateral-X-ray-of-the-knee-showing-an-AO-41-B31-fracture-type.png
Cant import X-ray-of-MT-V-stress-fracture-a-AP-view-b-Oblique-view-Stress-fracture-of-the-MT-V.png
```

Data Visualization part

Images in form of dataframe(Pixels)

In [3]:
```
images_fr
```

Out[3]:
```
array([array([[[ 10,  10,  10],
```

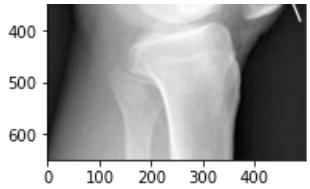
```
        [ 10,  10,  10],
        [ 10,  10,  10],
        ...,
        [  1,   1,   1],
        [  1,   1,   1],
        [  1,   1,   1]],

       [[ 10,  10,  10],
        [ 10,  10,  10],
        [ 10,  10,  10],
        ...,
        [  1,   1,   1],
        [  1,   1,   1],
        [  1,   1,   1]],

       [[ 10,  10,  10],
        [ 10,  10,  10],
        [ 10,  10,  10],
        ...,
        [  1,   1,   1],
        [  1,   1,   1],
        [  1,   1,   1]],

       ...,

       [[ 15,  15,  15],
        [  0,   0,   0],
        [ 35,  35,  35],
        ...,
        [  1,   1,   1],
        [  1,   1,   1],
        [  1,   1,   1]],

       [[186, 186, 186],
        [ 55,  55,  55],
        [  0,   0,   0],
        ...,
        [  1,   1,   1],
        [  1,   1,   1],
        [  1,   1,   1]],

       [[248, 248, 248],
        [215, 215, 215],
        [140, 140, 140],
        ...,
        [  1,   1,   1],
        [  1,   1,   1],
        [  1,   1,   1]]], dtype=uint8),
array([[[0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        ...,
        [0.56078434, 0.56078434, 0.56078434, 1.        ],
        [0.5647059 , 0.5647059 , 0.5647059 , 1.        ],
        [0.5647059 , 0.5647059 , 0.5647059 , 1.        ]],

       [[0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        ...,
        [0.56078434, 0.56078434, 0.56078434, 1.        ],
        [0.56078434, 0.56078434, 0.56078434, 1.        ],
        [0.56078434, 0.56078434, 0.56078434, 1.        ]],

       [[0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        ...,
        [0.5568628 , 0.5568628 , 0.5568628 , 1.        ],
        [0.5568628 , 0.5568628 , 0.5568628 , 1.        ],
        [0.5568628 , 0.5568628 , 0.5568628 , 1.        ]],

       ...,

       [[0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        [0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        [0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        ...,
        [0.6156863 , 0.6156863 , 0.6156863 , 1.        ],
        [0.62352943, 0.62352943, 0.62352943, 1.        ],
        [0.6392157 , 0.6392157 , 0.6392157 , 1.        ]],

       [[0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        [0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        [0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        ...,
        [0.63529414, 0.63529414, 0.63529414, 1.        ],
        [0.6392157 , 0.6392157 , 0.6392157 , 1.        ],
        [0.6431373 , 0.6431373 , 0.6431373 , 1.        ]],
```

```
       [[0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        [0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        [0.0627451 , 0.0627451 , 0.0627451 , 1.        ],
        ...,
        [0.64705884, 0.64705884, 0.64705884, 1.        ],
        [0.6392157 , 0.6392157 , 0.6392157 , 1.        ],
        [0.6392157 , 0.6392157 , 0.6392157 , 1.        ]]], dtype=float32),
array([[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]], dtype=uint8), ...,
array([[[  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],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8),
array([[[0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        ...,
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 0.41568628]],

       [[0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        ...,
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 0.41568628]],

       [[0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        ...,
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 0.41568628]],

       ...,
```

```
       [[0.08627451, 0.08627451, 0.08627451, 1.        ],
        [0.09803922, 0.09803922, 0.09803922, 1.        ],
        [0.03137255, 0.03137255, 0.03137255, 1.        ],
        ...,
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 0.41568628]],

       [[0.04313726, 0.04313726, 0.04313726, 1.        ],
        [0.03529412, 0.03529412, 0.03529412, 1.        ],
        [0.00392157, 0.00392157, 0.00392157, 1.        ],
        ...,
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 1.        ],
        [0.        , 0.        , 0.        , 0.41568628]],

       [[0.        , 0.        , 0.        , 0.09411765],
        [0.        , 0.        , 0.        , 0.09411765],
        [0.        , 0.        , 0.        , 0.09411765],
        ...,
        [0.        , 0.        , 0.        , 0.09411765],
        [0.        , 0.        , 0.        , 0.09411765],
        [0.        , 0.        , 0.        , 0.03921569]]], dtype=float32),
array([[[  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        ...,
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255]],

       [[  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        ...,
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255]],

       [[  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        ...,
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255]],

       ...,

       [[  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        ...,
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255]],

       [[  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        ...,
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255]],

       [[  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        ...,
        [  0,   0,   0, 255],
        [  0,   0,   0, 255],
        [  0,   0,   0, 255]]], dtype=uint8)], dtype=object)
```

In [4]:
```
plt.imshow(images_fr[100])
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x13e8052e9a0>
```



In [5]:

```
images_nonfr = []
folder = r'D:\Fracture Detection System (Minor2)\A Dataset\All Non Fractured'
for filename in os.listdir(folder):
    try:
        img = mpimg.imread(os.path.join(folder, filename))
        if img is not None:
            images_nonfr.append(img)
    except:
        print('Cant import ' + filename)
images_nonfr = np.asarray(images_nonfr)
```

```
Cant import A-plain-X-ray-right-hand-AP-B-magnified-view-for-proximal-phalanges-showing-loss-of.png
```

In [6]:

```
images_nonfr
```

Out[6]:

```
array([array([[ 1,  0,  1, ...,  0,  0,  0],
              [ 0,  0,  0, ...,  0,  0,  0],
              [ 0,  1,  0, ...,  0,  0,  0],
              ...,
              [29, 28, 28, ..., 44, 48, 10],
              [ 0,  0,  0, ..., 49, 41, 16],
              [ 1,  1,  1, ..., 14, 15,  2]], dtype=uint8),
       array([[[250, 250, 250],
               [252, 252, 252],
               [252, 252, 252],
               ...,
               [252, 252, 252],
               [252, 252, 252],
               [255, 255, 255]],

              [[117, 117, 117],
               [120, 120, 120],
               [121, 121, 121],
               ...,
               [176, 176, 176],
               [250, 250, 250],
               [255, 255, 255]],

              [[  0,   0,   0],
               [  1,   1,   1],
               [  1,   1,   1],
               ...,
               [109, 109, 109],
               [248, 248, 248],
               [254, 254, 254]],

              ...,

              [[ 45,  45,  45],
               [ 43,  43,  43],
               [ 41,  41,  41],
               ...,
               [112, 112, 112],
               [244, 244, 244],
               [255, 255, 255]],

              [[ 41,  41,  41],
               [ 42,  42,  42],
               [ 42,  42,  42],
               ...,
               [109, 109, 109],
               [244, 244, 244],
               [255, 255, 255]],

              [[ 36,  36,  36],
               [ 37,  37,  37],
               [ 39,  39,  39],
               ...,
               [108, 108, 108],
               [244, 244, 244],
               [255, 255, 255]]], dtype=uint8),
       array([[[219, 219, 191],
               [217, 217, 191],
```

```
        [211, 211, 185],
        ...,
        [167, 168, 126],
        [182, 183, 139],
        [195, 197, 150]],

       [[222, 222, 196],
        [212, 212, 186],
        [196, 196, 172],
        ...,
        [131, 131,  93],
        [155, 156, 116],
        [175, 176, 134]],

       [[219, 222, 195],
        [200, 202, 178],
        [174, 176, 154],
        ...,
        [ 87,  86,  56],
        [122, 121,  90],
        [153, 153, 119]],

       ...,

       [[187, 185, 164],
        [166, 164, 143],
        [135, 132, 113],
        ...,
        [ 48,  49,  31],
        [ 84,  86,  65],
        [131, 133, 112]],

       [[205, 203, 180],
        [185, 183, 160],
        [156, 154, 133],
        ...,
        [ 76,  79,  52],
        [110, 113,  84],
        [156, 159, 130]],

       [[213, 212, 184],
        [201, 199, 174],
        [183, 181, 158],
        ...,
        [130, 134,  99],
        [154, 158, 121],
        [175, 179, 142]]], dtype=uint8), ...,
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [ 69,  67,  81],
        [ 67,  66,  80],
        [ 65,  64,  78]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [ 69,  67,  81],
        [ 66,  65,  79],
        [ 63,  62,  76]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [ 69,  67,  81],
        [ 64,  63,  77],
        [ 61,  60,  74]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],
```

```
       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8),
array([[[  7,  10,  17],
        [  7,  10,  17],
        [  8,  11,  18],
        ...,
        [ 39,  67,  71],
        [ 38,  66,  70],
        [ 36,  64,  68]],

       [[  7,  10,  17],
        [  7,  10,  17],
        [  8,  11,  18],
        ...,
        [ 51,  74,  80],
        [ 50,  73,  79],
        [ 49,  72,  78]],

       [[  7,  10,  17],
        [  7,  10,  17],
        [  8,  11,  18],
        ...,
        [ 42,  56,  65],
        [ 42,  56,  65],
        [ 41,  55,  64]],

       ...,

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]],

       [[255, 255, 255],
        [255, 255, 255],
        [255, 255, 255],
        ...,
        [255, 255, 255],
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8),
array([[[ 3,  2,  8],
        [ 5,  4,  9],
        [ 3,  3,  3],
        ...,
        [ 1,  1,  3],
        [ 3,  3,  5],
        [ 4,  2,  5]],

       [[ 2,  1,  6],
        [ 3,  3,  5],
        [ 2,  2,  2],
        ...,
        [ 2,  2,  2],
        [ 3,  3,  5],
        [ 3,  1,  4]],

       [[ 4,  4,  6],
        [ 3,  3,  5],
        [ 2,  2,  2],
        ...,
        [ 0,  2,  1],
        [ 2,  2,  4],
        [ 3,  1,  4]],

       ...,

       [[18, 28, 30],
        [22, 36, 36],
        [19, 41, 38],
        ...,
        [ 2,  4,  3],
        [ 4,  5,  7],
        [ 1,  2,  4]],
```

```
       [[22, 27, 33],
        [25, 36, 38],
        [24, 44, 43],
        ...,
        [ 2,  4,  3],
        [ 3,  5,  4],
        [ 3,  5,  4]],

       [[16, 34, 36],
        [25, 43, 45],
        [27, 47, 48],
        ...,
        [ 4,  4,  6],
        [ 5,  5,  3],
        [ 6,  5,  0]]], dtype=uint8)], dtype=object)
```

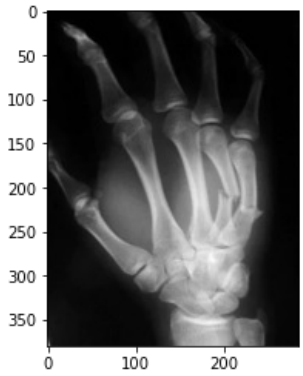
In [7]:
```
plt.imshow(images_nonfr[110])
```

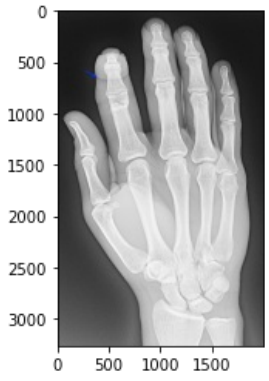
Out[7]:
```
<matplotlib.image.AxesImage at 0x13e805bc5e0>
```

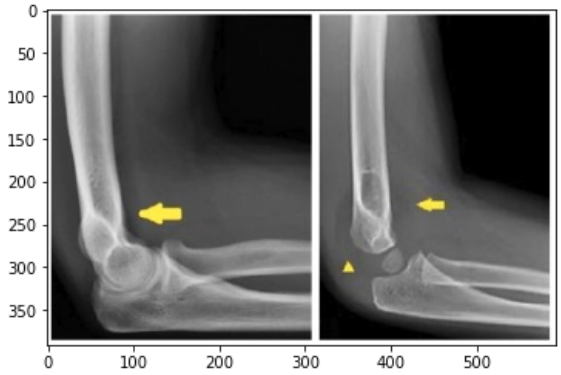
Plot Some Fractured and Non Fractured Images

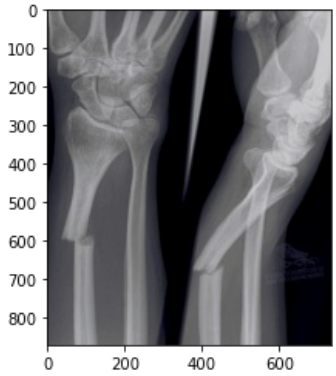
In [8]:
```
print("Some Fractured Images")
for i in range(60,65):
    plt.figure()
    plt.imshow(images_fr[i])
```

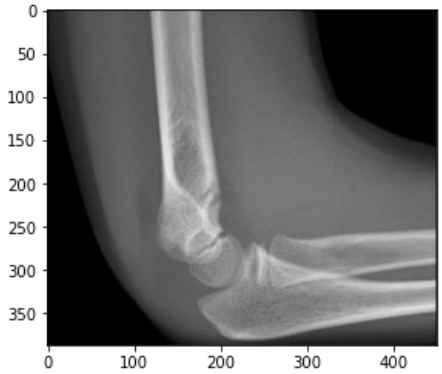
```
Some Fractured Images
```







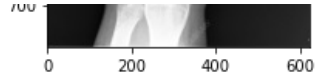




In [9]:

```
print("Some Non Fractured Images:")
for j in range(110,115):
    plt.figure()
    plt.imshow(images_nonfr[j])
```

Some Non Fractured Images:



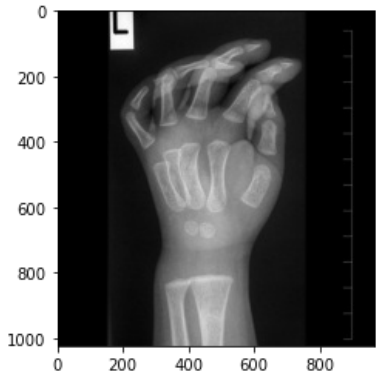


Image Augmentation.

In [10]:
```
train_datagen = ImageDataGenerator(preprocessing_function=tf.keras.applications.xception.preprocess_input,zoom_ra
                                   width_shift_range=0.1,height_shift_range=0.1,validation_split=0.1)
test_datagen=ImageDataGenerator(preprocessing_function=tf.keras.applications.xception.preprocess_input)
```

Train Validation and test Split of dataset

(Training-70% ,Validation-20% ,Testing-10%)

In [ ]:
```
#donot run it again(one run only)
import splitfolders
input_folder = 'E:\Fracture Detection System (Minor2)\A Dataset/'
```

In [ ]:
```
#Donot run it again(one run only)
splitfolders.ratio(input_folder, output="D:\Fracture Detection System (Minor2)\Split_dataset",
                   seed=42, ratio=(.7, .2, .1),
                   group_prefix=None) # default values
```

In [12]:
```
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

img = load_img(r'D:\Fracture Detection System (Minor2)\Split_dataset\train\All Fractured/1b5b2c658bcf8e16baf84ce5
x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
x = x.reshape((1,) + x.shape)  # this is a Numpy array with shape (1, 3, 150, 150)
```

Model Building Part

1.Convolutional Neural Network(CNN)-Using 2 Class

In [13]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

In [14]:
```
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

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

In [15]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 activation (Activation)     (None, 148, 148, 32)      0

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        9248

 activation_1 (Activation)   (None, 72, 72, 32)        0

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 activation_2 (Activation)   (None, 34, 34, 64)        0

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 64)       0
 2D)

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 64)                1183808

 activation_3 (Activation)   (None, 64)                0

 dropout (Dropout)           (None, 64)                0
```

```
 dense_1 (Dense)             (None, 1)                 65

 activation_4 (Activation)   (None, 1)                 0

=================================================================
Total params: 1,212,513
Trainable params: 1,212,513
Non-trainable params: 0
_________________________________________________________________
```

In [16]:

```
batch_size = 80
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1./255)

# this is a generator that will read pictures found in
# subfolers of 'data/train', and indefinitely generate
# batches of augmented image data
train_generator = train_datagen.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset\train',  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size,
        class_mode='binary')  # since we use binary_crossentropy loss, we need binary labels

# this is a similar generator, for validation data
validation_generator = test_datagen.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset\val',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
```

```
Found 3468 images belonging to 2 classes.
Found 991 images belonging to 2 classes.
```

In [17]:

```
#Model Training
model.fit_generator(
        train_generator,
        steps_per_epoch=2000 // batch_size,
        epochs=20,
        validation_data=validation_generator,
        verbose=1,
        validation_steps=800 // batch_size)
```

```
Epoch 1/20
25/25 [==============================] - 24s 923ms/step - loss: 0.6949 - accuracy: 0.5488 - val_loss: 0.6901 - va
l_accuracy: 0.5775
Epoch 2/20
25/25 [==============================] - 20s 805ms/step - loss: 0.6868 - accuracy: 0.5672 - val_loss: 0.6836 - va
l_accuracy: 0.5863
Epoch 3/20
25/25 [==============================] - 20s 809ms/step - loss: 0.6802 - accuracy: 0.5815 - val_loss: 0.6783 - va
l_accuracy: 0.5775
Epoch 4/20
25/25 [==============================] - 20s 797ms/step - loss: 0.6805 - accuracy: 0.5760 - val_loss: 0.6775 - va
l_accuracy: 0.5850
Epoch 5/20
25/25 [==============================] - 20s 805ms/step - loss: 0.6780 - accuracy: 0.5910 - val_loss: 0.6895 - va
l_accuracy: 0.5625
Epoch 6/20
25/25 [==============================] - 20s 781ms/step - loss: 0.6801 - accuracy: 0.5821 - val_loss: 0.6805 - va
l_accuracy: 0.5750
Epoch 7/20
25/25 [==============================] - 20s 787ms/step - loss: 0.6750 - accuracy: 0.5893 - val_loss: 0.6763 - va
l_accuracy: 0.5913
Epoch 8/20
25/25 [==============================] - 20s 789ms/step - loss: 0.6734 - accuracy: 0.5811 - val_loss: 0.6850 - va
l_accuracy: 0.5725
Epoch 9/20
25/25 [==============================] - 20s 788ms/step - loss: 0.6721 - accuracy: 0.5924 - val_loss: 0.6834 - va
l_accuracy: 0.5738
Epoch 10/20
25/25 [==============================] - 20s 807ms/step - loss: 0.6731 - accuracy: 0.5785 - val_loss: 0.6694 - va
l_accuracy: 0.5975
Epoch 11/20
```

```
25/25 [==============================] - 21s 837ms/step - loss: 0.6714 - accuracy: 0.5845 - val_loss: 0.6747 - va
l_accuracy: 0.5875
Epoch 12/20
25/25 [==============================] - 21s 828ms/step - loss: 0.6721 - accuracy: 0.5780 - val_loss: 0.6797 - va
l_accuracy: 0.5863
Epoch 13/20
25/25 [==============================] - 21s 849ms/step - loss: 0.6747 - accuracy: 0.5760 - val_loss: 0.6778 - va
l_accuracy: 0.5938
Epoch 14/20
25/25 [==============================] - 21s 824ms/step - loss: 0.6730 - accuracy: 0.5796 - val_loss: 0.6743 - va
l_accuracy: 0.5800
Epoch 15/20
25/25 [==============================] - 21s 840ms/step - loss: 0.6712 - accuracy: 0.5800 - val_loss: 0.6802 - va
l_accuracy: 0.5850
Epoch 16/20
25/25 [==============================] - 20s 780ms/step - loss: 0.6759 - accuracy: 0.5708 - val_loss: 0.6802 - va
l_accuracy: 0.5800
Epoch 17/20
25/25 [==============================] - 19s 767ms/step - loss: 0.6707 - accuracy: 0.5770 - val_loss: 0.6884 - va
l_accuracy: 0.5825
Epoch 18/20
25/25 [==============================] - 19s 775ms/step - loss: 0.6699 - accuracy: 0.5930 - val_loss: 0.6829 - va
l_accuracy: 0.5850
Epoch 19/20
25/25 [==============================] - 19s 761ms/step - loss: 0.6665 - accuracy: 0.5970 - val_loss: 0.6829 - va
l_accuracy: 0.5738
Epoch 20/20
25/25 [==============================] - 19s 744ms/step - loss: 0.6639 - accuracy: 0.5903 - val_loss: 0.6751 - va
l_accuracy: 0.5838
```

Out[17]:
```
<keras.callbacks.History at 0x13ea2047910>
```

In [18]:
```
model.save_weights('Cnnmodel_2class_model.h5')
```

Print Accuracy and Error Report Of CNN(2-Class) Model

In [19]:
```
print("****REPORT GENERATION(CNN-2 Class)****")
print("Training Accuracy of the CNN(2-Class) Model: 59.99%")
print("Validation Accuracy of the CNN(2-Class) Model: 56.38%")
print("Training loss of the CNN(2-Class) Model: 0.6596")
print("Validation loss of the CNN(2-Class) Model: 0.6897")
```

```
****REPORT GENERATION(CNN-2 Class)****
Training Accuracy of the CNN(2-Class) Model: 59.99%
Validation Accuracy of the CNN(2-Class) Model: 56.38%
Training loss of the CNN(2-Class) Model: 0.6596
Validation loss of the CNN(2-Class) Model: 0.6897
```

2.Convolutional Neural Network(CNN)-Using Batch wise Learning

In [20]:
```
#split the different classes folder into train,test and valid
input_folder01 = 'D:\Fracture Detection System (Minor2)\WholeDataset/'
```

In [ ]:
```
#Donot run it again
import splitfolders
splitfolders.ratio(input_folder01, output="E:\Fracture Detection System (Minor2)\Split_dataset(10 class)",
                   seed=42, ratio=(.7, .2, .1),  #70%train,20%test,10%validatiom
                   group_prefix=None)
```

define Classes

In [21]:
```
classes = ["Arm Fracture","Elbow Fracture","Foot Fracture","Hand Fracture","Knee Fracture","Normal Arm X Rays",
           "Normal Elbow X Rays","Normal Foot X Rays","Normal Hand X Rays","Normal knee X Rays"]
print("Total Number Of classes: 10 ")
```

```
Total Number Of classes: 10
```

Implementing Image Augmentation

In [22]:
```
batch_size01 = 32
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1./255)

# this is a generator that will read pictures found in
# subfolers of 'data/train', and indefinitely generate
# batches of augmented image data
train_generator01 = train_datagen.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset(10 class)\train',  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size01,
        class_mode='binary')  # since we use binary_crossentropy loss, we need binary labels

# this is a similar generator, for validation data
validation_generator01 = test_datagen.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset(10 class)\val',
        target_size=(150, 150),
        batch_size=batch_size01,
        class_mode='binary')
```

```
Found 3100 images belonging to 10 classes.
Found 883 images belonging to 10 classes.
```

Build CNN Model

In [23]:
```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(150,150, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(20, activation='softmax')
])
```

In [24]:
```
cnn.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

In [25]:
```
cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 74, 74, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_4 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 82944)             0

 dense_2 (Dense)             (None, 64)                5308480

 dense_3 (Dense)             (None, 20)                1300

=================================================================
Total params: 5,329,172
Trainable params: 5,329,172
Non-trainable params: 0
```

Model Training

In [27]:

```
cnn.fit_generator(
        train_generator01,
        steps_per_epoch=3000 // batch_size01,
        epochs=50,
        validation_data=validation_generator01,
        verbose=1,
        validation_steps=1000 // batch_size01)
```

```
Epoch 1/50
93/93 [==============================] - 26s 280ms/step - loss: 1.9236 - accuracy: 0.3075
Epoch 2/50
93/93 [==============================] - 26s 283ms/step - loss: 1.8503 - accuracy: 0.3220
Epoch 3/50
93/93 [==============================] - 26s 280ms/step - loss: 1.8135 - accuracy: 0.3462
Epoch 4/50
93/93 [==============================] - 27s 284ms/step - loss: 1.7318 - accuracy: 0.3688
Epoch 5/50
93/93 [==============================] - 26s 281ms/step - loss: 1.6980 - accuracy: 0.3964
Epoch 6/50
93/93 [==============================] - 26s 276ms/step - loss: 1.6408 - accuracy: 0.4014
Epoch 7/50
93/93 [==============================] - 26s 277ms/step - loss: 1.5789 - accuracy: 0.4458
Epoch 8/50
93/93 [==============================] - 26s 281ms/step - loss: 1.5465 - accuracy: 0.4374
Epoch 9/50
93/93 [==============================] - 26s 282ms/step - loss: 1.5081 - accuracy: 0.4536
Epoch 10/50
93/93 [==============================] - 26s 279ms/step - loss: 1.4605 - accuracy: 0.4727
Epoch 11/50
93/93 [==============================] - 26s 279ms/step - loss: 1.4271 - accuracy: 0.4929
Epoch 12/50
93/93 [==============================] - 26s 284ms/step - loss: 1.3987 - accuracy: 0.4966
Epoch 13/50
93/93 [==============================] - 27s 284ms/step - loss: 1.3524 - accuracy: 0.5178
Epoch 14/50
93/93 [==============================] - 26s 276ms/step - loss: 1.3149 - accuracy: 0.5302
Epoch 15/50
93/93 [==============================] - 26s 276ms/step - loss: 1.2818 - accuracy: 0.5427
Epoch 16/50
93/93 [==============================] - 26s 280ms/step - loss: 1.2562 - accuracy: 0.5505
Epoch 17/50
93/93 [==============================] - 26s 282ms/step - loss: 1.2126 - accuracy: 0.5619
Epoch 18/50
93/93 [==============================] - 26s 277ms/step - loss: 1.1654 - accuracy: 0.5764
Epoch 19/50
93/93 [==============================] - 26s 276ms/step - loss: 1.1620 - accuracy: 0.5828
Epoch 20/50
93/93 [==============================] - 26s 277ms/step - loss: 1.1102 - accuracy: 0.5935
Epoch 21/50
93/93 [==============================] - 26s 277ms/step - loss: 1.0854 - accuracy: 0.6083
Epoch 22/50
93/93 [==============================] - 26s 279ms/step - loss: 1.0843 - accuracy: 0.6211
Epoch 23/50
93/93 [==============================] - 26s 277ms/step - loss: 1.0580 - accuracy: 0.6228
Epoch 24/50
93/93 [==============================] - 26s 281ms/step - loss: 1.0085 - accuracy: 0.6460
Epoch 25/50
93/93 [==============================] - 26s 279ms/step - loss: 1.0099 - accuracy: 0.6447
Epoch 26/50
93/93 [==============================] - 27s 287ms/step - loss: 0.9686 - accuracy: 0.6484
Epoch 27/50
93/93 [==============================] - 26s 283ms/step - loss: 0.9987 - accuracy: 0.6491
Epoch 28/50
93/93 [==============================] - 26s 281ms/step - loss: 0.9087 - accuracy: 0.6699
Epoch 29/50
93/93 [==============================] - 26s 280ms/step - loss: 0.9228 - accuracy: 0.6682
Epoch 30/50
93/93 [==============================] - 26s 278ms/step - loss: 0.9130 - accuracy: 0.6746
Epoch 31/50
93/93 [==============================] - 26s 279ms/step - loss: 0.8645 - accuracy: 0.6904
Epoch 32/50
93/93 [==============================] - 27s 284ms/step - loss: 0.9070 - accuracy: 0.6797
Epoch 33/50
93/93 [==============================] - 26s 284ms/step - loss: 0.8514 - accuracy: 0.6925
Epoch 34/50
93/93 [==============================] - 27s 288ms/step - loss: 0.8564 - accuracy: 0.7059
Epoch 35/50
93/93 [==============================] - 26s 283ms/step - loss: 0.8196 - accuracy: 0.7110
Epoch 36/50
```

```
93/93 [==============================] - 27s 286ms/step - loss: 0.8123 - accuracy: 0.7059
Epoch 37/50
93/93 [==============================] - 26s 279ms/step - loss: 0.8072 - accuracy: 0.7022
Epoch 38/50
93/93 [==============================] - 26s 279ms/step - loss: 0.7775 - accuracy: 0.7298
Epoch 39/50
93/93 [==============================] - 26s 280ms/step - loss: 0.7630 - accuracy: 0.7338
Epoch 40/50
93/93 [==============================] - 27s 284ms/step - loss: 0.7512 - accuracy: 0.7301
Epoch 41/50
93/93 [==============================] - 26s 283ms/step - loss: 0.7780 - accuracy: 0.7281
Epoch 42/50
93/93 [==============================] - 26s 281ms/step - loss: 0.7440 - accuracy: 0.7328
Epoch 43/50
93/93 [==============================] - 27s 284ms/step - loss: 0.7183 - accuracy: 0.7476
Epoch 44/50
93/93 [==============================] - 27s 286ms/step - loss: 0.7353 - accuracy: 0.7312
Epoch 45/50
93/93 [==============================] - 27s 287ms/step - loss: 0.6946 - accuracy: 0.7561
Epoch 46/50
93/93 [==============================] - 27s 286ms/step - loss: 0.7065 - accuracy: 0.7490
Epoch 47/50
93/93 [==============================] - 27s 287ms/step - loss: 0.7021 - accuracy: 0.7510
Epoch 48/50
93/93 [==============================] - 27s 291ms/step - loss: 0.6906 - accuracy: 0.7587
Epoch 49/50
93/93 [==============================] - 27s 287ms/step - loss: 0.6672 - accuracy: 0.7732
Epoch 50/50
93/93 [==============================] - 26s 283ms/step - loss: 0.6489 - accuracy: 0.7715
```

Out[27]:
```
<keras.callbacks.History at 0x13eb977d490>
```

In [28]:
```
cnn.save_weights('Cnnmodel_10class_model.h5')
```

Print Accuracy and Error Report Of CNN(Batch wise Learning) Model

In [29]:
```
print("****REPORT GENERATION(CNN-Batch Wise Learning)****")
print("Training Accuracy of the CNN(Batch Wise) Model:77.52 %")
print("Validation Accuracy of the CNN(Batch Wise) Model:33.47 %")
print("Training loss of the CNN(Batch Wise) Model:0.7104 ")
print("Validation loss of the CNN(Batch Wise) Model:3.2176 ")
```

```
****REPORT GENERATION(CNN-Batch Wise Learning)****
Training Accuracy of the CNN(Batch Wise) Model:77.52 %
Validation Accuracy of the CNN(Batch Wise) Model:33.47 %
Training loss of the CNN(Batch Wise) Model:0.7104
Validation loss of the CNN(Batch Wise) Model:3.2176
```

Implementing Transfer Learning Models

3.Implement VGG16 Model

In [30]:
```
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
```

In [31]:
```
IMAGE_SIZE = [150, 150]   #Kept Image size (150,150)
```

In [32]:
```
model_vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 18s 0us/step
58900480/58889256 [==============================] - 18s 0us/step
```

In [33]:
```
model_vgg.input
```

Out[33]:
```
<KerasTensor: shape=(None, 150, 150, 3) dtype=float32 (created by layer 'input_1')>
```

In [34]:
```
for layer in model_vgg.layers:
    layers.trainable = False
```

In [35]:
```
folders = r'D:\Fracture Detection System (Minor2)\Split_dataset\train'
print(len(folders))
```

```
57
```

In [70]:
```
x = Flatten()(model_vgg.output)
prediction = Dense(len(folders)-56, activation='sigmoid')(x)
vgg_model = Model(inputs=model_vgg.input, outputs=prediction)
vgg_model.summary()
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 flatten_7 (Flatten)         (None, 8192)              0

 dense_9 (Dense)             (None, 1)                 8193

=================================================================
Total params: 14,722,881
Trainable params: 14,722,881
Non-trainable params: 0
_________________________________________________________________
```

In [71]:
```
vgg_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Image Augumentation

In [72]:
```
train_datagen01 = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

In [73]:
```
test_datagen01 = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

In [74]:
```
train_generator02 = train_datagen01.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset\train',  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size01,
        class_mode='binary')  # since we use binary_crossentropy loss, we need binary labels

# this is a similar generator, for validation data
validation_generator02 = test_datagen01.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset\val',
        target_size=(150, 150),
        batch_size=batch_size01,
        class_mode='binary')
```

```
Found 3468 images belonging to 2 classes.
Found 991 images belonging to 2 classes.
```

In [75]:
```
from datetime import datetime
from keras.callbacks import ModelCheckpoint
```

In [76]:
```
checkpoint = ModelCheckpoint(filepath='vgg16model.h5',
                             verbose=1, save_best_only=True)

callbacks = [checkpoint]
```

Train the Vgg16 Model

In [78]:
```
vgg_model_history=vgg_model.fit_generator(
  train_generator02 ,
  validation_data=validation_generator02 ,
  epochs=20,
  steps_per_epoch=15,
  validation_steps=32,
  callbacks=callbacks,
  verbose=1)
```

```
Epoch 1/20
15/15 [==============================] - ETA: 0s - loss: 115.1899 - accuracy: 0.5229WARNING:tensorflow:Your input
ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per
_epoch * epochs` batches (in this case, 32 batches). You may need to use the repeat() function when building your
dataset.

Epoch 1: val_loss improved from inf to 0.68355, saving model to vgg16model.h5
15/15 [==============================] - 79s 5s/step - loss: 115.1899 - accuracy: 0.5229 - val_loss: 0.6836 - val
_accuracy: 0.5772
Epoch 2/20
15/15 [==============================] - ETA: 0s - loss: 0.7086 - accuracy: 0.5271WARNING:tensorflow:Can save bes
t model only with val_loss available, skipping.
15/15 [==============================] - 58s 4s/step - loss: 0.7086 - accuracy: 0.5271
Epoch 3/20
15/15 [==============================] - ETA: 0s - loss: 0.6908 - accuracy: 0.5854WARNING:tensorflow:Can save bes
t model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6908 - accuracy: 0.5854
Epoch 4/20
15/15 [==============================] - ETA: 0s - loss: 0.6839 - accuracy: 0.5500WARNING:tensorflow:Can save bes
t model only with val_loss available, skipping.
```

```
15/15 [==============================] - 57s 4s/step - loss: 0.6839 - accuracy: 0.5500
Epoch 5/20
15/15 [==============================] - ETA: 0s - loss: 0.6832 - accuracy: 0.5792WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6832 - accuracy: 0.5792
Epoch 6/20
15/15 [==============================] - ETA: 0s - loss: 0.6922 - accuracy: 0.5688WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6922 - accuracy: 0.5688
Epoch 7/20
15/15 [==============================] - ETA: 0s - loss: 0.6819 - accuracy: 0.5813WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6819 - accuracy: 0.5813
Epoch 8/20
15/15 [==============================] - ETA: 0s - loss: 0.6903 - accuracy: 0.5646WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6903 - accuracy: 0.5646
Epoch 9/20
15/15 [==============================] - ETA: 0s - loss: 0.6917 - accuracy: 0.5562WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6917 - accuracy: 0.5562
Epoch 10/20
15/15 [==============================] - ETA: 0s - loss: 0.6712 - accuracy: 0.6042WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 55s 4s/step - loss: 0.6712 - accuracy: 0.6042
Epoch 11/20
15/15 [==============================] - ETA: 0s - loss: 0.6881 - accuracy: 0.5833WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6881 - accuracy: 0.5833
Epoch 12/20
15/15 [==============================] - ETA: 0s - loss: 0.6886 - accuracy: 0.5729WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 57s 4s/step - loss: 0.6886 - accuracy: 0.5729
Epoch 13/20
15/15 [==============================] - ETA: 0s - loss: 0.6993 - accuracy: 0.5854WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 55s 4s/step - loss: 0.6993 - accuracy: 0.5854
Epoch 14/20
15/15 [==============================] - ETA: 0s - loss: 0.6839 - accuracy: 0.5771WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 58s 4s/step - loss: 0.6839 - accuracy: 0.5771
Epoch 15/20
15/15 [==============================] - ETA: 0s - loss: 0.6906 - accuracy: 0.5478WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 54s 4s/step - loss: 0.6906 - accuracy: 0.5478
Epoch 16/20
15/15 [==============================] - ETA: 0s - loss: 0.7114 - accuracy: 0.5083WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 57s 4s/step - loss: 0.7114 - accuracy: 0.5083
Epoch 17/20
15/15 [==============================] - ETA: 0s - loss: 0.6725 - accuracy: 0.6000WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 61s 4s/step - loss: 0.6725 - accuracy: 0.6000
Epoch 18/20
15/15 [==============================] - ETA: 0s - loss: 0.6781 - accuracy: 0.6125WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 55s 4s/step - loss: 0.6781 - accuracy: 0.6125
Epoch 19/20
15/15 [==============================] - ETA: 0s - loss: 0.6930 - accuracy: 0.5562WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 56s 4s/step - loss: 0.6930 - accuracy: 0.5562
Epoch 20/20
15/15 [==============================] - ETA: 0s - loss: 0.6714 - accuracy: 0.6062WARNING:tensorflow:Can save best model only with val_loss available, skipping.
15/15 [==============================] - 55s 4s/step - loss: 0.6714 - accuracy: 0.6062
```

Print Accuracy and Error Report Of VGG16 Model

In [79]:

```
print("****REPORT GENERATION(VGG 16 Model)****")
print("Training Accuracy of the CNN(Batch Wise) Model: 60.62%")
print("Validation Accuracy of the CNN(Batch Wise) Model:61.47 %")
print("Training loss of the CNN(Batch Wise) Model:0.6714 ")
print("Validation loss of the CNN(Batch Wise) Model:0.7785 ")
```

```
****REPORT GENERATION(VGG 16 Model)****
Training Accuracy of the CNN(Batch Wise) Model: 60.62%
Validation Accuracy of the CNN(Batch Wise) Model:61.47 %
Training loss of the CNN(Batch Wise) Model:0.6714
Validation loss of the CNN(Batch Wise) Model:0.7785
```

4.Implement RESNET50 Model

In [156…]
```
batch_size02=128
train_generator03 = train_datagen01.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset\train',  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size02,
        class_mode='binary')  # since we use binary_crossentropy loss, we need binary labels

# this is a similar generator, for validation data
validation_generator03 = test_datagen01.flow_from_directory(
        r'D:\Fracture Detection System (Minor2)\Split_dataset\val',
        target_size=(150, 150),
        batch_size=batch_size02,
        class_mode='binary')
```

```
Found 3468 images belonging to 2 classes.
Found 991 images belonging to 2 classes.
```

Import resnet50 Model

In [157…]
```
from tensorflow.keras.optimizers import Adam
resnet_model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(150,150,3),
                   pooling='avg',classes=5,
                   weights='imagenet')   # Pre trained weight are taken from imagenet dataset and model.
for layer in pretrained_model.layers:
        layer.trainable=False

resnet_model.add(pretrained_model)
```

In [162…]
```
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='sigmoid'))
resnet_model.add(Dense(1, activation='softmax'))
```

In [163…]
```
resnet_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten_12 (Flatten)        (None, 2048)              0

 dense_16 (Dense)            (None, 512)               1049088

 dense_17 (Dense)            (None, 1)                 513

 flatten_13 (Flatten)        (None, 1)                 0

 dense_18 (Dense)            (None, 512)               1024

 dense_19 (Dense)            (None, 1)                 513

=================================================================
Total params: 24,638,850
Trainable params: 1,051,138
Non-trainable params: 23,587,712
_________________________________________________________________
```

In [164…]
```
resnet_model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
```

Model Training

In [165…]
```
history = resnet_model.fit(train_generator03 , validation_data=validation_generator03, epochs=20,verbose=1)
```

```
Epoch 1/20
```

```
28/28 [==============================] - 70s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 2/20
28/28 [==============================] - 67s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 3/20
28/28 [==============================] - 69s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 4/20
28/28 [==============================] - 68s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 5/20
28/28 [==============================] - 72s 3s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 6/20
28/28 [==============================] - 65s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 7/20
28/28 [==============================] - 66s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 8/20
28/28 [==============================] - 66s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 9/20
28/28 [==============================] - 65s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 10/20
28/28 [==============================] - 65s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 11/20
28/28 [==============================] - 64s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 12/20
28/28 [==============================] - 68s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 13/20
28/28 [==============================] - 65s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 14/20
28/28 [==============================] - 66s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 15/20
28/28 [==============================] - 66s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 16/20
28/28 [==============================] - 66s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 17/20
28/28 [==============================] - 65s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 18/20
28/28 [==============================] - 68s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 19/20
28/28 [==============================] - 68s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
Epoch 20/20
28/28 [==============================] - 68s 2s/step - loss: 0.0000e+00 - accuracy: 0.4207 - val_loss: 0.0000e+00
- val_accuracy: 0.4208
```

In [166…

```
resnet_model.save_weights('Cnnmodel_10class_model.h5')
```

Print Accuracy and Error Report Of Resnet50 Model.

In [167…

```
print("****REPORT GENERATION(Resnet50 Model)****")
print("Training Accuracy of the Resnet50 Model: 42.07%")
print("Validation Accuracy of the Resnet50 Model:42.08 %")
print("Result: Model Saturate at 42.07% Accuracy")
```

```
****REPORT GENERATION(Resnet50 Model)****
Training Accuracy of the Resnet50 Model: 42.07%
Validation Accuracy of the Resnet50 Model:42.08 %
Result: Model Saturate at 42.07% Accuracy
```

Visualization Techniques (For Comparitive Study of Different Algorithms used in this Project)

In [168…

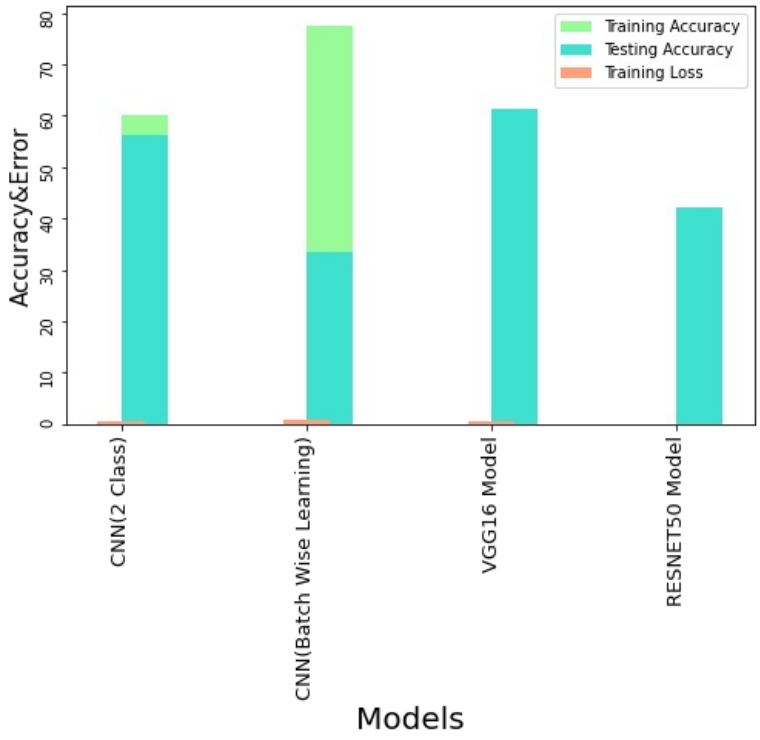
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [171…

```
models=['CNN(2 Class)', 'CNN(Batch Wise Learning)','VGG16 Model','RESNET50 Model']
ValuesTr=[59.99,77.52,60.62,42.07]
Valueste=[56.38,33.47,61.47,42.07]
Valuesms=[0.6596,0.7104,0.6714,0.000]
ypos = np.arange(len(models))
#size of graph
plt.figure(figsize=(8, 5))
#angle of x ticks
plt.xticks(rotation=90, fontsize=13)
plt.yticks(rotation=90, fontsize=10)
# Plotting Training Accuracy
plt.bar(models, ValuesTr, width=0.25, align='edge', color='palegreen', label = "Training Accuracy")
# Plotting Trainng Accuracy
plt.bar(models, Valueste, width=0.25, align='edge', color='turquoise', label = "Testing Accuracy")
# Plotting Mean Square Error
plt.bar(models, Valuesms, width=0.25, color='lightsalmon', label="Training Loss")
plt.xlabel("Models", fontsize = 20)
plt.ylabel("Accuracy&Error", fontsize= 15, rotation= 90)
plt.legend()
```

Out[171…

```
<matplotlib.legend.Legend at 0x13f48361b50>
```



END

In [ ]: