Name: Soumya Ranjan Nayak

PRN: 23070243063

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
paultimothymooney_chest_xray_pneumonia_path = kagglehub.dataset_download('paultimothymooney/chest-xray-pneumonia')
techsash_waste_classification_data_path = kagglehub.dataset_download('techsash/waste-classification-data')
ravirajsinh45_real_life_industrial_dataset_of_casting_product_path = kagglehub.dataset_download('ravirajsinh45/real-life-industrial-dataset-of-casting-product')
utkarshsaxenadn_car_vs_bike_classification_dataset_path = kagglehub.dataset_download('utkarshsaxenadn/car-vs-bike-classification-dataset')
unmoved_30k_cats_and_dogs_150x150_greyscale_path = kagglehub.dataset_download('unmoved/30k-cats-and-dogs-150x150-greyscale')
moazeldsokyx_dogs_vs_cats_path = kagglehub.dataset_download('moazeldsokyx/dogs-vs-cats')
devbatrax_fracture_detection_using_x_ray_images_path = kagglehub.dataset_download('devbatrax/fracture-detection-using-x-ray-images')
bmadushanirodrigo_fracture_multi_region_x_ray_data_path = kagglehub.dataset_download('bmadushanirodrigo/fracture-multi-region-x-ray-data')
darhan_fracture_multi_region_acc_1_0000_val_acc_0_9885_tensorflow2_fracture_xray_multiregion_model_1_path = kagglehub.model_download('darhan/fracture-multi-regior

print('Data source import complete.')
```

Fracture-multi-region-x-ray-data

```
import os, shutil, json, zipfile, random, math
from PIL import Image
import cv2
import numpy as np
import tensorflow as tf
from matplotlib import pyplot as plt
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, Input
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

```
2024-11-17 17:54:35.375858: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory
2024-11-17 17:54:35.375971: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory
2024-11-17 17:54:35.504582: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register facto
```

```
dataset_dir = '/kaggle/input/fracture-multi-region-x-ray-data'
project_name = os.path.basename(dataset_dir).capitalize()+'_'
print(dataset_dir)
print(project_name)
```

```
/kaggle/input/fracture-multi-region-x-ray-data
Fracture-multi-region-x-ray-data_
```

```
# Avoid OOM errors by setting GPU Memory Consumption Growth
def gpu_config():
    gpus = tf.config.experimental.list_physical_devices('GPU')
    for gpu in gpus:
        tf.config.experimental.set_memory_growth(gpu, True)
    print(tf.config.list_physical_devices('GPU'))
```

```
gpu_config()
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU'), PhysicalDevice(name='/physical_device:GPU:1', device_type='GPU')]
```

```
print(os.listdir(dataset_dir))
```

```
['README.dataset.txt', 'Bone_Fracture_Binary_Classification']
```

```
root_dir = '/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train'

class_1 = '/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/fractured'
class_2 = '/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/not fractured'


test_dir_1 = '/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/test/fractured'
test_dir_2 = '/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/test/not fractured'

val_dir_1 = '/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/val/fractured'
val_dir_2 = '/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/val/not fractured'
```

```
def create_directory(directories):
    for directory in directories:
        if not os.path.exists(directory):
            os.mkdir(directory)
            print('Created {}'.format(directory))
        else:
            print('{} exists'.format(directory))
```

```
logs_dir = '/kaggle/working/logs'
models_dir = '/kaggle/working/models'
hist_dir = '/kaggle/working/history'
figures_dir = '/kaggle/working/figures'
```

```
zipfiles_dir = '/kaggle/working/zipfiles'
train_dir = '/kaggle/working/train'

create_directory([logs_dir, models_dir, hist_dir, figures_dir, zipfiles_dir,train_dir])
```

```
Created /kaggle/working/logs
Created /kaggle/working/models
Created /kaggle/working/history
Created /kaggle/working/figures
Created /kaggle/working/zipfiles
Created /kaggle/working/train
```

```
def delete_files_and_folders(folder_path):
    for item in os.listdir(folder_path):
        item_path = os.path.join(folder_path, item)

        if os.path.isfile(item_path):
            os.unlink(item_path)
        elif os.path.isdir(item_path):
            shutil.rmtree(item_path)

    print("All files and folders inside", folder_path, "have been deleted.")
    print(os.listdir(folder_path))
```

```
# delete_files_and_folders(models_dir)
# delete_files_and_folders(hist_dir)
# delete_files_and_folders(logs_dir)
# delete_files_and_folders(figures_dir)
# delete_files_and_folders(zipfiles_dir)
# delete_files_and_folders(train_dir)
```

```
def data_info(directory):
    images = os.listdir(directory)
    print(f"Images in {os.path.basename(directory)} : {len(images)}")
    extensions = [os.path.splitext(file)[1] for file in images]
    unique_extensions = set(extensions)
    print("Images are present in these extensions:", unique_extensions)
```

```
def check_image_readability(folder_path):
    file_list = os.listdir(folder_path)

    errored_images = []
    correct_images = []

    for file_name in file_list:
        file_path = os.path.join(folder_path, file_name)
        try:
            with Image.open(file_path) as img:
                img.load()
            correct_images.append(file_path)
        except Exception as e:
            print(e)
            print(file_path)
            errored_images.append(file_path)
            continue

    if errored_images:
        print("There are errored images")
    else:
        print("All images are readable.")
    return errored_images, correct_images
```

```
data_info(class_1)
data_info(class_2)
```

```
Images in fractured : 4606
Images are present in these extensions: {'.jpg', '.jpeg', '.png'}
Images in not fractured : 4640
Images are present in these extensions: {'.jpg', '.png'}
```

```
errored_images_1, correct_images_1 = check_image_readability(class_1)
print(len(errored_images_1), len(correct_images_1))
```

```
All images are readable.
0 4606
```

```
errored_images_2, correct_images_2 = check_image_readability(class_2)
print(len(errored_images_2), len(correct_images_2))
```

```
image file is truncated (40 bytes not processed)
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/not fractured/IMG0004347.jpg
image file is truncated (14 bytes not processed)
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/not fractured/IMG0004148.jpg
image file is truncated (1 bytes not processed)
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/not fractured/IMG0004134.jpg
image file is truncated (33 bytes not processed)
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/not fractured/IMG0004149.jpg
image file is truncated (10 bytes not processed)
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/not fractured/IMG0004143.jpg
image file is truncated (40 bytes not processed)
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/train/not fractured/IMG0004308.jpg
There are errored images
6 4634
```

```
def move_images(images_list, to_dir):
    if not os.path.exists(to_dir):
        os.mkdir(to_dir)

    for image_path in images_list:
        image_name = os.path.basename(image_path)
        destination_path = os.path.join(to_dir, image_name)
        shutil.copy(image_path, destination_path)

    print("Images moved successfully to", to_dir)
```

```
move_images(correct_images_1, train_dir + '/' +os.path.basename(class_1))
move_images(correct_images_2, train_dir + '/' +os.path.basename(class_2))
```

```
Images moved successfully to /kaggle/working/train/fractured
Images moved successfully to /kaggle/working/train/not fractured
```

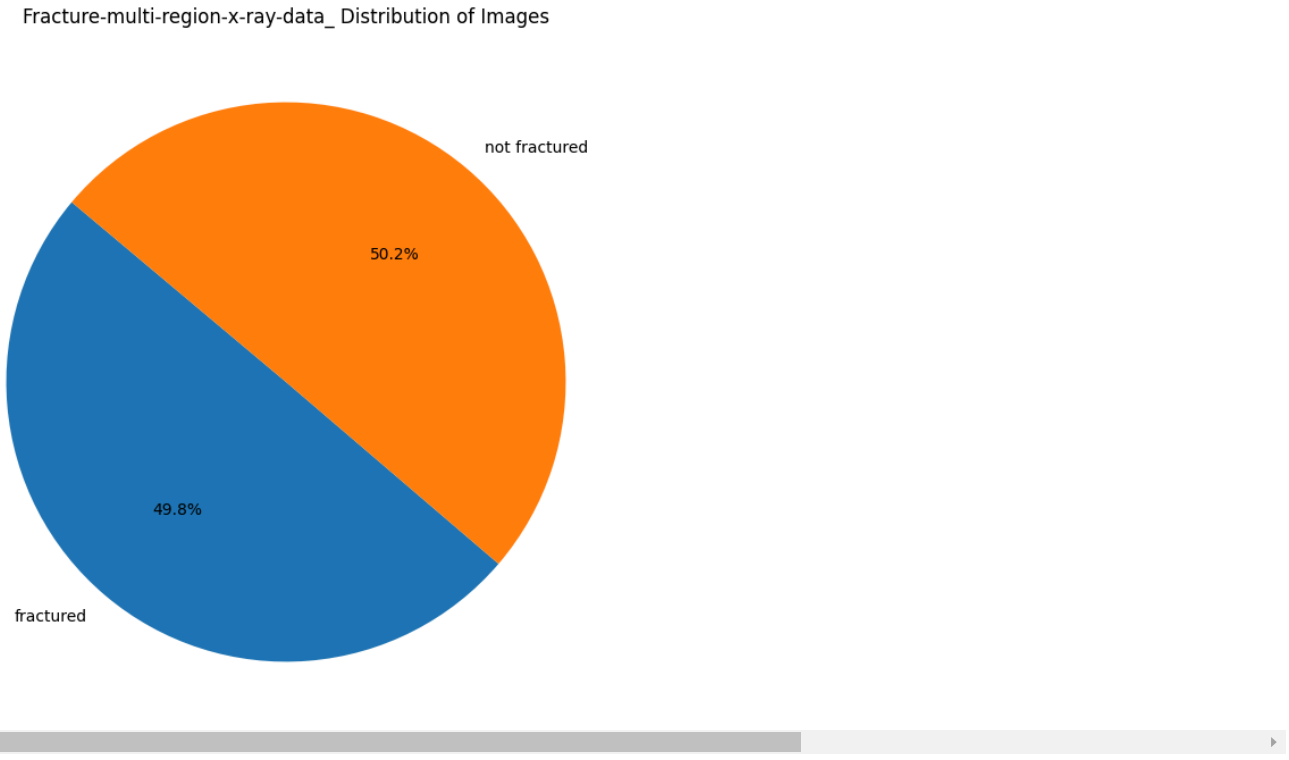
```
root_dir = '/kaggle/working/train'
class_1 = '/kaggle/working/train/fractured'
class_2 = '/kaggle/working/train/not fractured'
```

```
data_info(class_1)
data_info(class_2)
```

```
Images in fractured : 4606
Images are present in these extensions: {'.jpg', '.jpeg', '.png'}
Images in not fractured : 4634
Images are present in these extensions: {'.jpg', '.png'}
```

```
labels = [os.path.basename(class_1), os.path.basename(class_2)]
sizes = [len(os.listdir(class_1)), len(os.listdir(class_2))]
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
title = '{} Distribution of Images'.format(project_name)
plt.title(title)
plt.savefig(figures_dir+'/'+title+'.png')
print('{} saved to {}'.format(title, figures_dir))
plt.show()
```

```
Fracture-multi-region-x-ray-data_ Distribution of Images saved to /kaggle/working/figures
```




```
data_info(test_dir_1)
data_info(test_dir_2)
```

```
Images in fractured : 238
Images are present in these extensions: {'.jpg', '.png'}
Images in not fractured : 268
Images are present in these extensions: {'.jpg', '.png'}
```

```
data_info(val_dir_1)
data_info(val_dir_2)
```

```
Images in fractured : 337
Images are present in these extensions: {'.jpg', '.jpeg', '.png'}
Images in not fractured : 492
Images are present in these extensions: {'.jpg', '.png'}
```

```
data = tf.keras.utils.image_dataset_from_directory(root_dir)
```

```
Found 9240 files belonging to 2 classes.
```

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

```
print(type(data),'\n')
print(len(data),'\n')
print(data.element_spec)
```

```
<class 'tensorflow.python.data.ops.prefetch_op._PrefetchDataset'>

289

(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
```

```
class_labels = {}
print('Classes found {}'.format(data.class_names))
for idx, class_name in enumerate(data.class_names):
    print(f"Class Label: {idx}, Class Name: {class_name}")
    class_labels[class_name] = idx
print(class_labels)
```

```
Classes found ['fractured', 'not fractured']
Class Label: 0, Class Name: fractured
Class Label: 1, Class Name: not fractured
{'fractured': 0, 'not fractured': 1}
```

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

```
print(type(data_iterator))
```

```
<class 'tensorflow.python.data.ops.dataset_ops.NumpyIterator'>
```

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

```
print("Type of batch:", type(batch))
print("Length of batch:", len(batch))
print("Shape of batch[0]:", batch[0].shape)
print("Shape of batch[1]:", batch[1].shape)
print("Minimum value in batch[0]:", batch[0].min())
print("Maximum value in batch[0]:", batch[0].max())
```

```
Type of batch: <class 'tuple'>
Length of batch: 2
Shape of batch[0]: (32, 256, 256, 3)
Shape of batch[1]: (32,)
Minimum value in batch[0]: 0.0
Maximum value in batch[0]: 255.0
```

```
grid_shape = (3, 3)
total_images = 9
num_rows =3
num_cols = 3

plt.rcParams['font.size'] = 20
fig, ax = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, 20))
for idx, img in enumerate(batch[0][:total_images]):
    row = idx // grid_shape[1]
    col = idx % grid_shape[1]
    ax[row, col].imshow(img.astype(int))
    class_label = batch[1][idx]
    class_name = data.class_names[class_label]
    ax[row, col].title.set_text(f"Class: {class_label} - {class_name}")

plt.tight_layout()
plt.show()
```


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

```
print(type(scaled_data), len(scaled_data))
```

```
<class 'tensorflow.python.data.ops.map_op._MapDataset'> 289
```

```
scaled_batch = scaled_data.as_numpy_iterator().next()
```

```
print("Type of scaled_batch:", type(scaled_batch))
print("Length of scaled_batch:", len(scaled_batch))
print("Shape of scaled_batch[0]:", scaled_batch[0].shape)
print("Shape of scaled_batch[1]:", scaled_batch[1].shape)
print("Minimum value in scaled_batch[0]:", scaled_batch[0].min())
print("Maximum value in scaled_batch[0]:", scaled_batch[0].max())
```

```
Type of scaled_batch: <class 'tuple'>
Length of scaled_batch: 2
Shape of scaled_batch[0]: (32, 256, 256, 3)
Shape of scaled_batch[1]: (32,)
Minimum value in scaled_batch[0]: 0.0
```

```
Maximum value in scaled_batch[0]: 1.0
```

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

```
print("Type of train_size:", type(train_size))
print("Type of val_size:", type(val_size))
print("Type of test_size:", type(test_size), '\n')

print("train_size:", train_size)
print("val_size:", val_size)
print("test_size:", test_size)
```

```
Type of train_size: <class 'int'>
Type of val_size: <class 'int'>
Type of test_size: <class 'int'>

train_size: 202
val_size: 57
test_size: 28
```

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

```
print("Type of train:", type(train))
print("Type of val:", type(val))
print("Type of test:", type(test),'\n')

print("train:", train)
print("val:", val)
print("test:", test)
```

```
Type of train: <class 'tensorflow.python.data.ops.take_op._TakeDataset'>
Type of val: <class 'tensorflow.python.data.ops.take_op._TakeDataset'>
Type of test: <class 'tensorflow.python.data.ops.take_op._TakeDataset'>

train: <_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
val: <_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
test: <_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
```

```
input_shape = (256, 256, 3)
model_name = project_name+'model'
inputs = Input(shape=input_shape)

x = Conv2D(16, (3, 3), 1, activation='relu')(inputs)
x = MaxPooling2D()(x)
x = Conv2D(32, (3, 3), 1, activation='relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(16, (3, 3), 1, activation='relu')(x)
x = MaxPooling2D()(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs=inputs, outputs=outputs, name=model_name)
```

```
model.summary()
```

Model: "Fracture-multi-region-x-ray-data_model"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 254, 254, 16)	448
max_pooling2d (MaxPooling2D)	(None, 127, 127, 16)	0
conv2d_1 (Conv2D)	(None, 125, 125, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 32)	0
conv2d_2 (Conv2D)	(None, 60, 60, 16)	4,624
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 16)	0
flatten (Flatten)	(None, 14400)	0
dense (Dense)	(None, 256)	3,686,656
dense_1 (Dense)	(None, 1)	257

Total params: 3,696,625 (14.10 MB)
Trainable params: 3,696,625 (14.10 MB)

```
model_name = project_name
checkpoint_path = models_dir+"/"+model_name+"epoch_{epoch:02d} acc_{accuracy:.4f} loss_{loss:.4f} val_acc_{val_accuracy:.4f} val_loss_{val_loss:.4f}.keras"
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                    save_weights_only=False,
                                    save_freq='epoch')
print(checkpoint_path)
```

```
/kaggle/working/models/Fracture-multi-region-x-ray-data_epoch_{epoch:02d} acc_{accuracy:.4f} loss_{loss:.4f} val_acc_{val_accuracy:.4f} val_loss_{val_loss:.
```

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

callbacks = [checkpoint_callback]
```

```
print(callbacks)
```

```
[<keras.src.callbacks.model_checkpoint.ModelCheckpoint object at 0x7f042c5d7c10>]
```

```
for folder in [logs_dir, models_dir, hist_dir, figures_dir, zipfiles_dir]:
    print("Contents of folder:", folder)
    print(os.listdir(folder))
```

```
Contents of folder: /kaggle/working/logs
[]
Contents of folder: /kaggle/working/models
[]
Contents of folder: /kaggle/working/history
[]
Contents of folder: /kaggle/working/figures
['Fracture-multi-region-x-ray-data_ Distribution of Images.png']
Contents of folder: /kaggle/working/zipfiles
[]
```

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

```
print("Model Name:", model.name)
print("Loss Function:", model.loss)
print("Optimizer:", model.optimizer)
print("Input Shape:", model.input_shape)
print("Output Shape:", model.output_shape)
```

```
Model Name: Fracture-multi-region-x-ray-data_model
Loss Function: <keras.src.losses.losses.BinaryCrossentropy object at 0x7f0517a92800>
Optimizer: <keras.src.optimizers.adam.Adam object at 0x7f042c64b6a0>
Input Shape: (None, 256, 256, 3)
Output Shape: (None, 1)
```

```
epochs = 10
```

```
%%time
hist = model.fit(train, epochs=epochs, validation_data=val, callbacks=callbacks)
```

```
Epoch 1/10
  5/202 ━━━━━━━━━━━━━━━━━━━━ 5s 30ms/step - accuracy: 0.4561 - loss: 0.8284WARNING: All log messages before absl::InitializeLog() is called are written to S
I0000 00:00:1731866153.030411     113 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
W0000 00:00:1731866153.049211     113 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
202/202 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.7154 - loss: 0.5422W0000 00:00:1731866169.660695     115 graph_launch.cc:671] Fallback to op-by-op m
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 94ms/step - accuracy: 0.7160 - loss: 0.5414 - val_accuracy: 0.9435 - val_loss: 0.1666
Epoch 2/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9572 - loss: 0.1258 - val_accuracy: 0.9748 - val_loss: 0.0884
Epoch 3/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 19s 95ms/step - accuracy: 0.9866 - loss: 0.0451 - val_accuracy: 0.9846 - val_loss: 0.0762
Epoch 4/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9934 - loss: 0.0216 - val_accuracy: 0.9846 - val_loss: 0.0870
Epoch 5/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 19s 93ms/step - accuracy: 0.9978 - loss: 0.0061 - val_accuracy: 0.9781 - val_loss: 0.0815
Epoch 6/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 30s 145ms/step - accuracy: 0.9953 - loss: 0.0133 - val_accuracy: 0.9830 - val_loss: 0.0856
Epoch 7/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9956 - loss: 0.0138 - val_accuracy: 0.9852 - val_loss: 0.0793
Epoch 8/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 19s 94ms/step - accuracy: 0.9990 - loss: 0.0048 - val_accuracy: 0.9857 - val_loss: 0.0938
Epoch 9/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 20s 95ms/step - accuracy: 0.9988 - loss: 0.0045 - val_accuracy: 0.9874 - val_loss: 0.0979
Epoch 10/10
202/202 ━━━━━━━━━━━━━━━━━━━━ 19s 93ms/step - accuracy: 0.9974 - loss: 0.0080 - val_accuracy: 0.9841 - val_loss: 0.1041
CPU times: user 7min 13s, sys: 22.7 s, total: 7min 36s
Wall time: 3min 31s
```

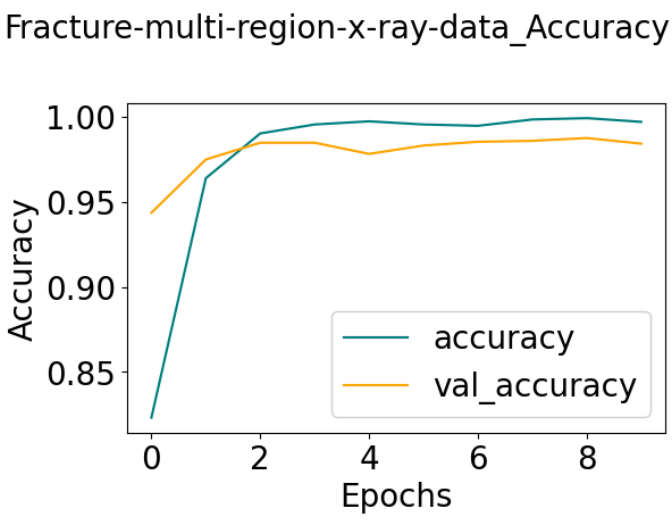
```
history = hist.history
hist_file = project_name + 'history.json'
hist_path = os.path.join(hist_dir, hist_file)
with open(hist_path, 'w') as f:
    json.dump(history, f)
    print('Training history saved to {}'.format(hist_path))
```

```
Training history saved to /kaggle/working/history/Fracture-multi-region-x-ray-data_history.json
```

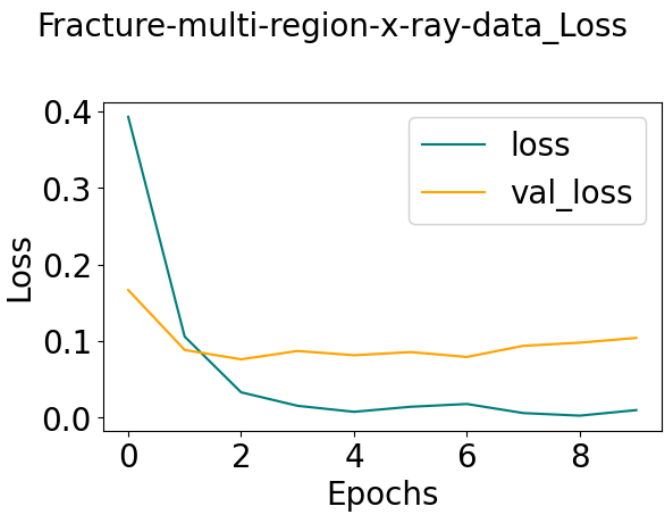
```
title = project_name

def plot_losses(hist):
    fig = plt.figure()
    plt.plot(hist['loss'], color='teal', label='loss')
    plt.plot(hist['val_loss'], color='orange', label='val_loss')
    fig.suptitle(title+'Loss', fontsize=20)
    plt.legend(loc="upper right")
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.tight_layout()
    plt.savefig(figures_dir+'/' + title+'losses.png')
    plt.show()
```

```
    print('Saved losses to {}'.format(figures_dir+'/' + title+'losses.png'))

def plot_accuracies(hist):
    fig = plt.figure()
    plt.plot(hist['accuracy'], color='teal', label='accuracy')
    plt.plot(hist['val_accuracy'], color='orange', label='val_accuracy')
    fig.suptitle(title+'Accuracy', fontsize=20)
    plt.legend(loc="lower right")
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.tight_layout()
    plt.savefig(figures_dir+'/' + title+'accuracies.png')
    plt.show()
    print('Saved accuracies to {}'.format(figures_dir+'/' + title+'accuracies.png'))
```

```
plot_accuracies(history)
plot_losses(history)
print(os.listdir(figures_dir))
```




```
Saved accuracies to /kaggle/working/figures/Fracture-multi-region-x-ray-data_accuracies.png
```




```
Saved losses to /kaggle/working/figures/Fracture-multi-region-x-ray-data_losses.png
['Fracture-multi-region-x-ray-data_losses.png', 'Fracture-multi-region-x-ray-data_accuracies.png', 'Fracture-multi-region-x-ray-data_ Distribution of Images
```

Evaluate Model

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

```
print(pre.result(), re.result(), acc.result())
```

```
tf.Tensor(0.9908257, shape=(), dtype=float32) tf.Tensor(1.0, shape=(), dtype=float32) tf.Tensor(0.99553573, shape=(), dtype=float32)
```

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

```
Precision : 0.9908257126808167
Recall : 1.0
Accuracy : 0.9955357313156128
```

Make Predictions

```
print(class_labels)
```

```
{'fractured': 0, 'not fractured': 1}
```

```
val_0_dir = val_dir_1
val_1_dir = val_dir_2

print(val_0_dir,'\n')
print(val_1_dir)
```

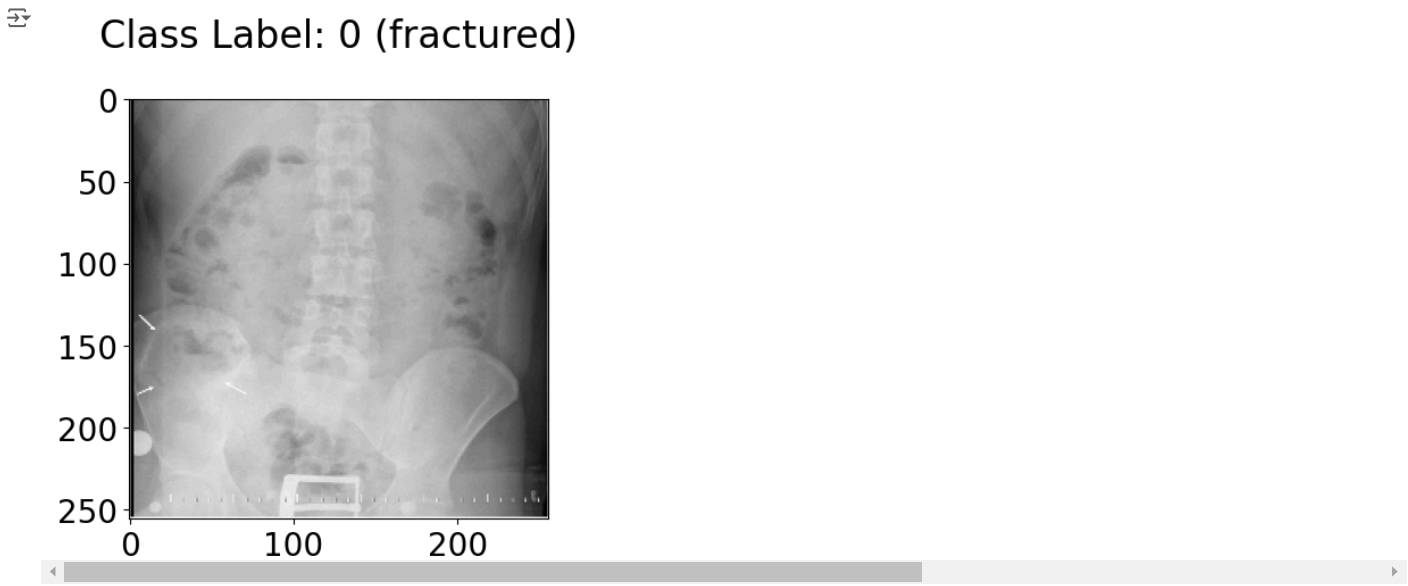
```
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/val/fractured

/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/val/not fractured
```

```
file_paths_val_0 = [os.path.join(val_0_dir, filename) for filename in os.listdir(val_0_dir)]
file_paths_val_1 = [os.path.join(val_1_dir, filename) for filename in os.listdir(val_1_dir)]

class_0_test = random.choice(file_paths_val_0)
class_1_test = random.choice(file_paths_val_1)

# Get the class label from the file path
class_label_0 = class_0_test.split('/')[-2]
class_label_1 = class_1_test.split('/')[-2]

# Get the class label value (0 or 1) from the class_labels dictionary
class_value_0 = class_labels[class_label_0]
class_value_1 = class_labels[class_label_1]

print(class_0_test)
print(class_1_test)
```

```
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/val/fractured/gr1_lrg2.jpg
/kaggle/input/fracture-multi-region-x-ray-data/Bone_Fracture_Binary_Classification/Bone_Fracture_Binary_Classification/val/not fractured/40-rotated1-rotated
```

Class 0

```
resize = tf.image.resize(img, (256,256))
plt.title(f'Class Label: {class_value_0} ({class_label_0})',y=1.1)
plt.imshow(resize.numpy().astype(int))
plt.show()
```




```
prediction = loaded_model.predict(np.expand_dims(resize/255, 0))
predicted_class = np.round(prediction).astype(int)
print(f"Predicted class is {predicted_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 769ms/step
Predicted class is [[0]]
```

Class 1

```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.title(f'Class Label: {class_value_1} ({class_label_1})', y=1.1)
plt.show()
```

Class Label: 1 (not fractured)