

Bonus Break Classification (Deep Learning Project)

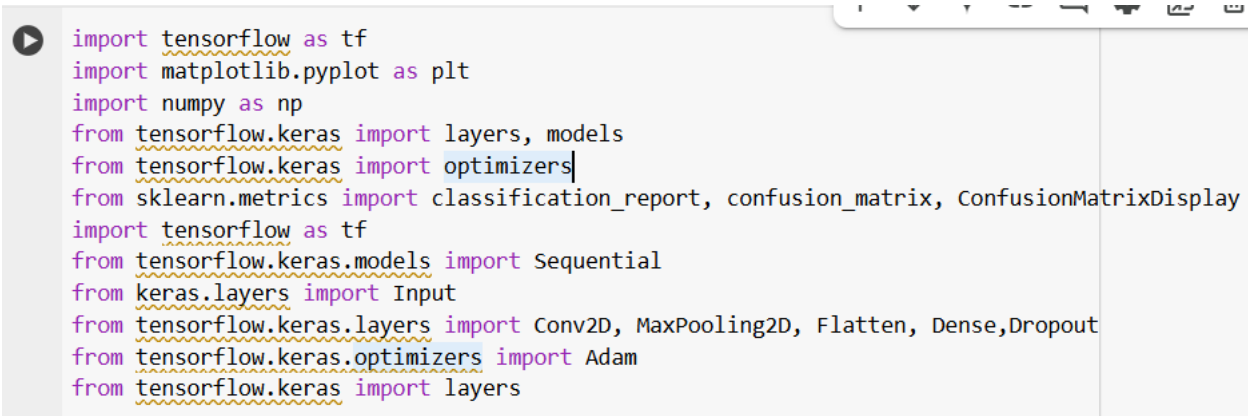
Objective :

The goal of this project is to classify X-ray or bone images into different categories indicating whether a bone is broken or not. This type of classification can assist in automated medical diagnosis.

Tools and Libraries :

The following Python libraries and frameworks were used:

- TensorFlow/Keras for model building and training
- Matplotlib & NumPy for visualization and numerical operations
- Scikit-learn for evaluation metrics such as classification report and confusion matrix



```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import layers, models
from tensorflow.keras import optimizers
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import tensorflow as tf
from tensorflow.keras.models import Sequential
from keras.layers import Input
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
```

Data Handling

The dataset was accessed from google drive and loaded using `image_dataset_from_directory()` from TensorFlow. The images were

- Converted to RGB format
- Resized to 256x256 pixels
- Divided into training and validation sets (90/10 split)
- Batched with a size of 64 and shuffled with a set seed for reproducibility

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

➡ Drive already mounted at /content/drive;

```
▶ #Training data
dir = '/content/drive/MyDrive/Bone Break Classification'
ds_train = tf.keras.preprocessing.image_dataset_from_directory(dir,
                                                                labels = 'inferred',
                                                                label_mode = 'int',
                                                                validation_split = 0.1,
                                                                subset = 'training',
                                                                shuffle = True,
                                                                color_mode = 'rgb',
                                                                image_size = (256, 256),
                                                                batch_size = 64,
                                                                seed = 50)
```

➡ Found 1129 files belonging to 10 classes.
Using 1017 files for training.

```
[ ] #validation data
ds_validation = tf.keras.preprocessing.image_dataset_from_directory(dir,
                                                                    labels = 'inferred',
                                                                    label_mode = 'int',
                                                                    validation_split = 0.1,
                                                                    subset = 'validation',
                                                                    shuffle = True,
                                                                    color_mode = 'rgb',
                                                                    image_size = (256, 256),
                                                                    batch_size = 64,
                                                                    seed = 50)
```

➡ Found 1129 files belonging to 10 classes.
Using 112 files for validation.

DataPreprocessing and Visualization

The dataset likely underwent normalization and exploratory visualization to understand class distribution and sample images. Visualizations were used to inspect dataset quality and structure before training.

```
[ ] #Normalize
def Normalize(image, label):
    return image/255, label

ds_train = ds_train.map(Normalize)
ds_validation = ds_validation.map(Normalize)
```

```
▶ # Test and Train
X_train = []
y_train = []

for images, labels in ds_train:
    X_train.append(images)
    y_train.append(labels)

X_train = tf.concat(X_train, axis=0)
y_train = tf.concat(y_train, axis=0)
```

```
[ ] x_val = []
y_val = []

for images, labels in ds_validation:
    x_val.append(images)
    y_val.append(labels)

x_val = tf.concat(x_val, axis=0)
y_val = tf.concat(y_val, axis=0)
```

```
[ ] #One hot encoding
class_names = 10
y_train = tf.keras.utils.to_categorical(y_train, class_names)
y_val = tf.keras.utils.to_categorical(y_val, class_names)
```

```
▶ class_label = ["Avulsion fracture",
                 "Comminuted fracture",
                 "Fracture Dislocation",
                 "Greenstick fracture",
                 "Hairline Fracture",
                 "Impacted fracture",
                 "Longitudinal fracture",
                 "Oblique fracture",
                 "Pathological fracture",
                 "Spiral Fracture"]

fig, axes = plt.subplots(2, 4, figsize=(15,5))

for i, ax in enumerate(axes.flat):
    images, labels = X_train[i], y_train[i]
    ax.imshow(images, cmap = 'gray')
    ax.set_title(f'{class_label[np.argmax(labels)]}')
    ax.axis('off')

plt.show()
```



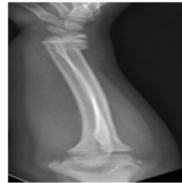
Avulsion fracture



Fracture Dislocation



Greenstick fracture



Oblique fracture



Model Architecture

A Convolutional Neural Network (CNN) was built using Keras. The architecture likely consisted of:

- **Input Layer**
- Multiple **Conv2D** layers with **MaxPooling**
- **Flatten** and **Dense** layers
- **Dropout** layers for regularization
- **Adam optimizer** and **cross-entropy loss function**

```
#CNN
model = Sequential()
model.add(Input(shape=(256, 256, 3)))

#First layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

#Second Layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

#Flatten and Fully connected layers
model.add(Flatten())
model.add(Dense(40, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 64)	1,792
max_pooling2d (MaxPooling2D)	(None, 127, 127, 64)	0
conv2d_1 (Conv2D)	(None, 125, 125, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 128)	0
flatten (Flatten)	(None, 492032)	0
dense (Dense)	(None, 40)	19,681,320
dense_1 (Dense)	(None, 10)	410

Total params: 19,757,378 (75.37 MB)

Trainable params: 19,757,378 (75.37 MB)

Non-trainable params: 0 (0.00 B)

Evaluation

After training the model was evaluated using

- **Accuracy and loss curves**
- **Confusion matrix**
- **Classification report** (Precision, Recall, F1-score)

```
[ ] model.compile(optimizer=Adam(learning_rate=0.0005),  
                  loss='categorical_crossentropy',  
                  metrics=['accuracy'])
```

```
[ ] #Train the model  
model.fit(X_train,y_train, epochs=5, batch_size=10, validation_data=(x_val, y_val))
```

```
Epoch 1/5  
102/102 ————— 178s 2s/step - accuracy: 0.1256 - loss: 2.7565 - val_accuracy: 0.1250 - val_loss: 2.2843  
Epoch 2/5  
102/102 ————— 179s 2s/step - accuracy: 0.2213 - loss: 2.2165 - val_accuracy: 0.1786 - val_loss: 2.1867  
Epoch 3/5  
102/102 ————— 196s 2s/step - accuracy: 0.4189 - loss: 1.7608 - val_accuracy: 0.1964 - val_loss: 2.1815  
Epoch 4/5  
102/102 ————— 197s 2s/step - accuracy: 0.6786 - loss: 1.0705 - val_accuracy: 0.2679 - val_loss: 2.8242  
Epoch 5/5  
102/102 ————— 170s 2s/step - accuracy: 0.8656 - loss: 0.5364 - val_accuracy: 0.3036 - val_loss: 3.3266  
<keras.src.callbacks.history.History at 0x7e84600c9fd0>
```

Results and Conclusion

Based on the model performance metrics in the notebook insights were drawn about the model's effectiveness in identifying bone breaks. This system can serve as a base prototype for clinical decision support systems.

Streamlit : An open-source python library that makes it easy to create and share custom web apps for machine learning and data science.

We will be creating a streamlit web interface that allows the user to upload an X-ray image, and this will display the predicted fracture type.

Below are the steps of implementing streamlit :

```
[ ] model.save('bone_fracture_break_model.h5')
```

⚡ WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `

```
[ ] from google.colab import files
files.download('bone_fracture_break_model.h5')
```

- We save the model as bone_fracture_break_model.h5 and we download it
- We implement a code for streamlit in python and save it as .py extension.

Python Code

```
import streamlit as st
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
from PIL import Image

model = load_model("bone_fracture_break_model.h5")

class_names = ['Avulsion fracture', 'Comminuted fracture', 'Fracture Dislocation',
               'Greenstick fracture', 'Hairline Fracture', 'Impacted fracture',
               'Longitudinal fracture', 'Oblique fracture', 'Pathological fracture',
               'Spiral Fracture']

# Streamlit UI
st.title("Bone Fracture Type Classification")
st.write("Upload an X-ray image and get the predicted fracture type.")

uploaded_file = st.file_uploader("Choose and upload the X-ray image", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    img = Image.open(uploaded_file).convert('RGB')
    st.image(img, caption='Uploaded X-ray', use_column_width=True)

    img = img.resize((256, 256))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    prediction = model.predict(img_array)
    predicted_class = class_names[np.argmax(prediction)]
    confidence = np.max(prediction) * 100

    st.success(f"**Predicted Class:** {predicted_class}")
    st.info(f"**Confidence Score:** {confidence:.2f}%")
```

❖ **Model and Library Setup**

The code loads a pre-trained deep learning model (bone_fracture_break_model.h5) using TensorFlow. Essential libraries like Streamlit (for the web UI), PIL (for image handling), and NumPy (for numerical operations) are imported.

❖ **User Interface with Streamlit**

A simple UI is created using Streamlit:

- Title and description are displayed.
- A file uploader allows users to upload an X-ray image (JPEG/PNG).

❖ **Image Preprocessing**

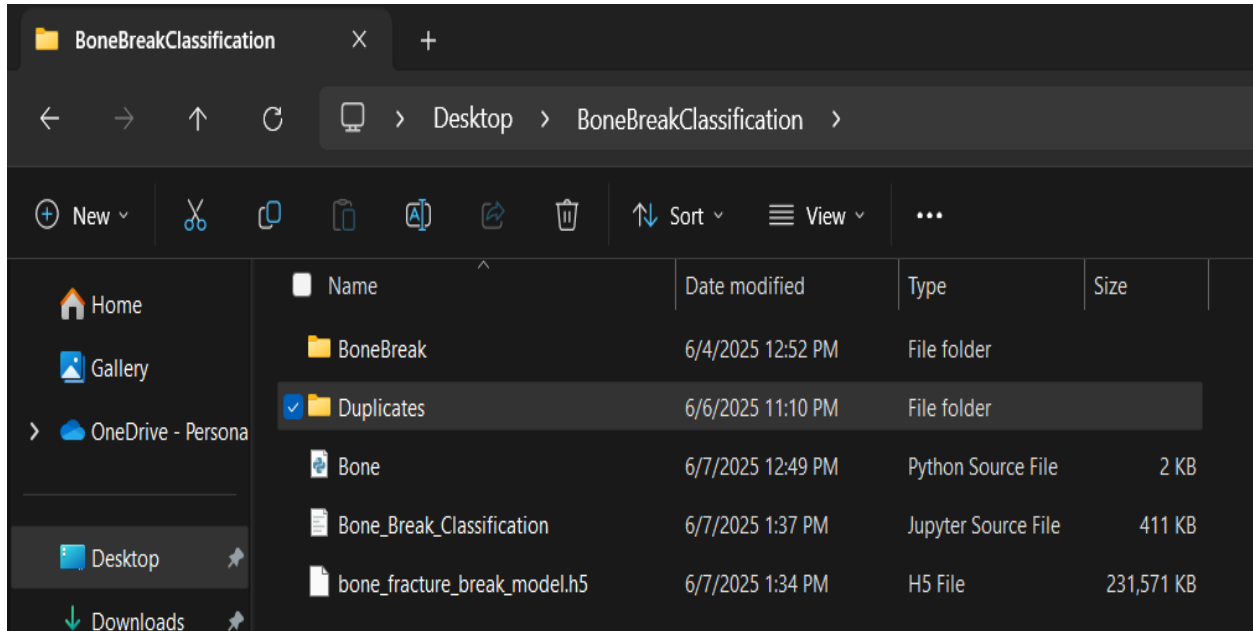
Once an image is uploaded:

- It's displayed on the interface.
- The image is resized to 256×256, converted to a NumPy array, normalized (divided by 255), and reshaped for model input.

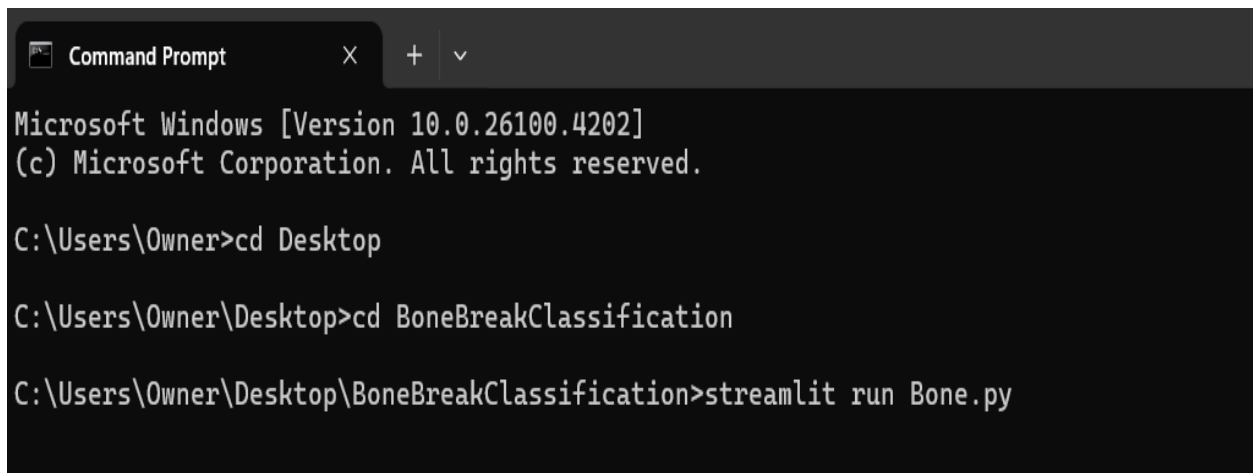
❖ **Prediction and Output**

The processed image is passed to the model for prediction. The class with the highest probability is selected, and both the predicted class and confidence score are shown to the user using Streamlit components.

Location of all the files presented.



In the command prompt we go to the file where .py extension file is presented and we use the command 'streamlit run Bone.py' which will direct us to web browser streamlit and we can upload the images and get the predicted fracture type.



Bone Fracture Type Classification

Upload an X-ray image and get the predicted fracture type.

Choose and upload the X-ray image



Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files



2_jpg.rf.d9b17b4172ebe50ee602011be7b272.jpg 16.1KB



Uploaded X-ray

Predicted Class: Comminuted fracture

Confidence Score: 96.68%