प्रगत संगणन विकास केंद्र, नोएडा
Centre for Development of Advanced Computing, Noida

# Project Report

## On

# MURA BONE ABNORMALITY DETECTION

CENTER FOR DEVELOPMENT OF ADVANCED COMPUTING

*Submitted*
*In partial fulfilment*
*For the award of the Degree of*

PG-Diploma in Artificial Intelligence

(C-DAC, Noida)
(2022-2023)

**Guided By:**

Mr. Shivam Pandey

**Submitted By:**

Tejaswini Singh Bhanwar (220920528053)

Sarita Kushwaha (220920528037)

Avinash Kumar (220920528007)

# Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, **Mr. Shivam Pandey**, C-DAC, Noida for his constant guidance and helpful suggestion to prepare this project **MURA- Bone Abnormality Detection**. We express our deepest gratitude towards him for his inspiring involvement and constructive criticism that provided us technical guidance during the course of this project.

It is our great pleasure in expressing sincere and deep gratitude towards **Mr. Ravi Payal** (Course Coordinator, PG-DAI) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to C-DAC,Noida for providing us the opportunity to implementthis prestigious project and enhance our learning skills in various technical fields.

We are also thankful to all other members and Staff of the department who were involved in the project either directly or indirectly for their valuable cooperation.

Last but not the least we would like to extend our thanks to our fellow students for their friendly cooperation.

Tejaswini Singh Bhanwar (210940128053)

Sarita Kushwaha (210940128005)

Avinash Kumar (210940128010)

# ABSTRACT

Mura is a large dataset of musculoskeletal radiographs containing 40,895 images from **14,982** studies and **12,251** patients, where each study is manually labelled by radiologists as either normal or abnormal. It involves **169-layer Dense Convolutional Neural Network** to detect the abnormalities. The musculoskeletal conditions affect over **1.7 billion** people worldwide causing pain and disability. It comes with train, test and valid folders containing radiographic images.

To evaluate models robustly and to get an estimate of radiologist performance, we collect additional labels from six board certified Stanford radiologists on the test set, consisting of **207** musculoskeletal studies. On this test set, the majority vote of a group of three radiologists serves as gold standard. We train a **169-layer DenseNet** baseline model to detect and localize abnormalities.

Model performance is comparable to the best radiologist performance in detecting abnormalities on finger and wrist studies. However, model performance is lower than best radiologist performance in detecting abnormalities on elbow, forearm, hand, humerus, and shoulder studies. We believe that the task is a good challenge for future research.

# Table of Contents

# Chapter 1
# Introduction
# (1.1)

## Introduction

Mura is a large dataset of musculoskeletal radiographs containing 40,895 images from 14,982 studies and **12,251** patients, where each study is manually labelled by radiologists as either normal or abnormal. It involves **169-layer Dense Convolutional Neural Network** to detect the abnormalities. The musculoskeletal conditions affect over **1.7 billion** people worldwide causing pain and disability. It comes with train, test and valid folders containing radiographic images.

Each radiographic image belongs to one of seven standard upper extremity radiographic study types and each study was labelled normal or abnormal by board-certified radiologists from Stanford University. It comes with train, test and valid folders containing radiographic images. Each image is labeled as **1** (abnormal) or **0** (zero, normal) based on whether it is corresponding study is negative or positive. The train set consists of: elbow, finger, forearm, hand, humerus, shoulder and wrist.

The study type contains folders named after patient ids, each view is RGB pixel ranging **[0,255]** and varies in dimensions. On each view Convolutional neural network predicts the probability of abnormality.

The MURA abnormality detection task is a binary classification task, where the input is an upper extremity radiograph study — with each study containing one or more views (images) — and the expected output is a binary label $\mathbf{y} \in \{0, 1\}$ indicating whether the study is normal or abnormal, respectively.

## 1.2-Objective and specification:

To determine whether the radiographic study is normal or abnormal is the critical radiographic task of this project.
Also, to evaluate models robustly and to get an estimate of the radiologist performance.

# Chapter 2
# Methodology and Techniques
# (2.1)

## Approach & Methodology/Techniques:

**The methods involved in the project are as follows:**

1. Write a code to be implemented in Python. The platform used to perform the proposed work is Google Collab or Jupyter notebook.

2. We import the required libraries
    - Pandas
    - NumPy
    - Matplotlib PyPlot
    - Keras
    - TensorFlow

3. HELPER FUNCTION: set up parameters for decoration of the plots

4. Getting the dataset from Stanford website. Then we form a directory to store the data of the zipped file and unzip it.

5. Getting the path to train and validation datasets. Then displaying the abnormal, normal bone train the set studies with labels, getting the heads, Counting the labels in the train set and validation set. Adding the labels to individual train set and validation set.

6. Abnormal images (positive) are labeled 1 and normal images (negative) are labeled 0. Counting individual numbers of the labels for train and validation dataset.

7. Applying data augmentation and visualize the augmented data

8. Trained different Pre-Trained model and taking best out of it.

9. Implementing transfer learning. Applying layers on top base model.

10. Then apply fine tuning on the model using different batch sizes and epoch values get different accuracy and loss values.

11. Plot the accuracy and loss curve

## 2.2-Dataset Contents

| STUDY | TRAIN NORMAL | TRAIN ABNORMAL | VALIDATION NORMAL | VALIDATION ABNORMAL | TOTAL |
|---|---|---|---|---|---|
| Elbow | 1094 | 660 | 92 | 66 | 1912 |
| Finger | 1280 | 665 | 92 | 83 | 2110 |
| Hand | 1497 | 521 | 101 | 66 | 2185 |
| Humerus | 321 | 271 | 68 | 67 | 727 |
| Forearm | 590 | 287 | 69 | 64 | 1010 |
| Shoulder | 1364 | 1457 | 99 | 95 | 3015 |
| Wrist | 2134 | 1326 | 140 | 97 | 3967 |

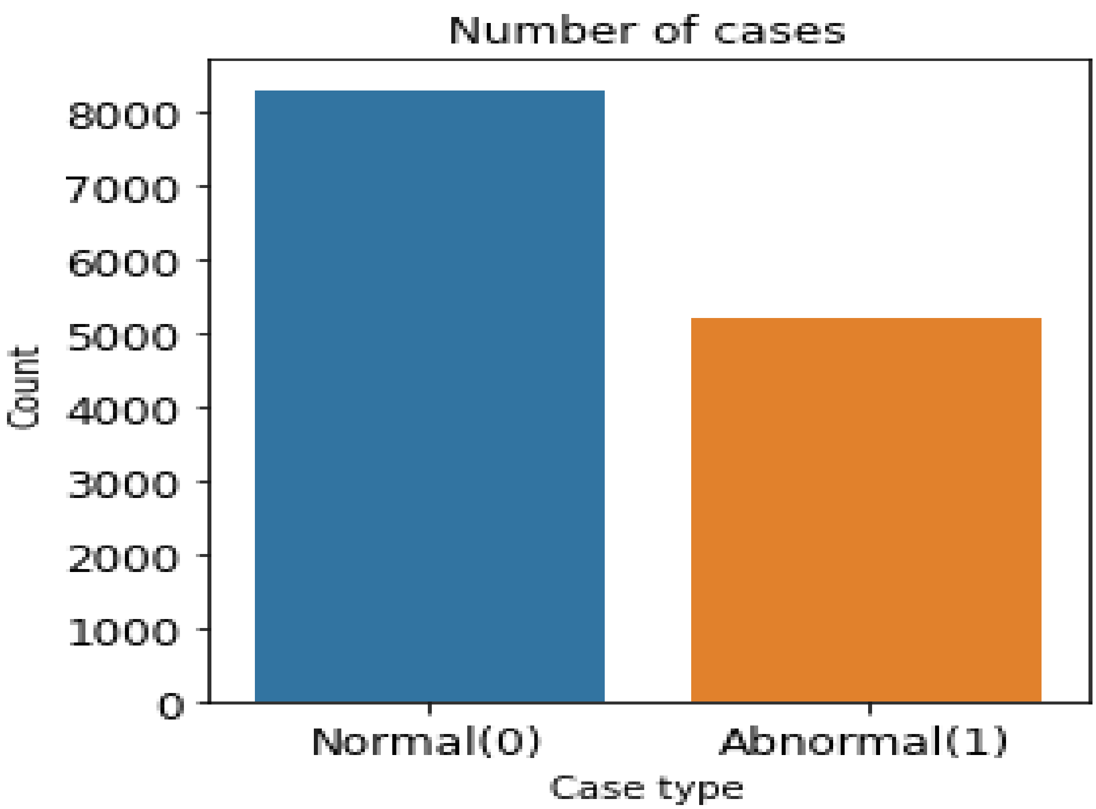
## 2.3-Data Visualization

### Count labels in train set

```
cases_count = df['Train_Label'].value_counts()
print(cases_count)

# Plot the results
plt.figure(figsize=(4,4))
sns.barplot(x=cases_count.index, y=cases_count.values)
plt.title('Number of cases', fontsize=12)
plt.xlabel('Case type', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.xticks(range(len(cases_count.index)), ['Normal(0)', 'Abnormal(1)'])
plt.show()
```

```
0    8280
1    5177
Name: Train_Label, dtype: int64
```

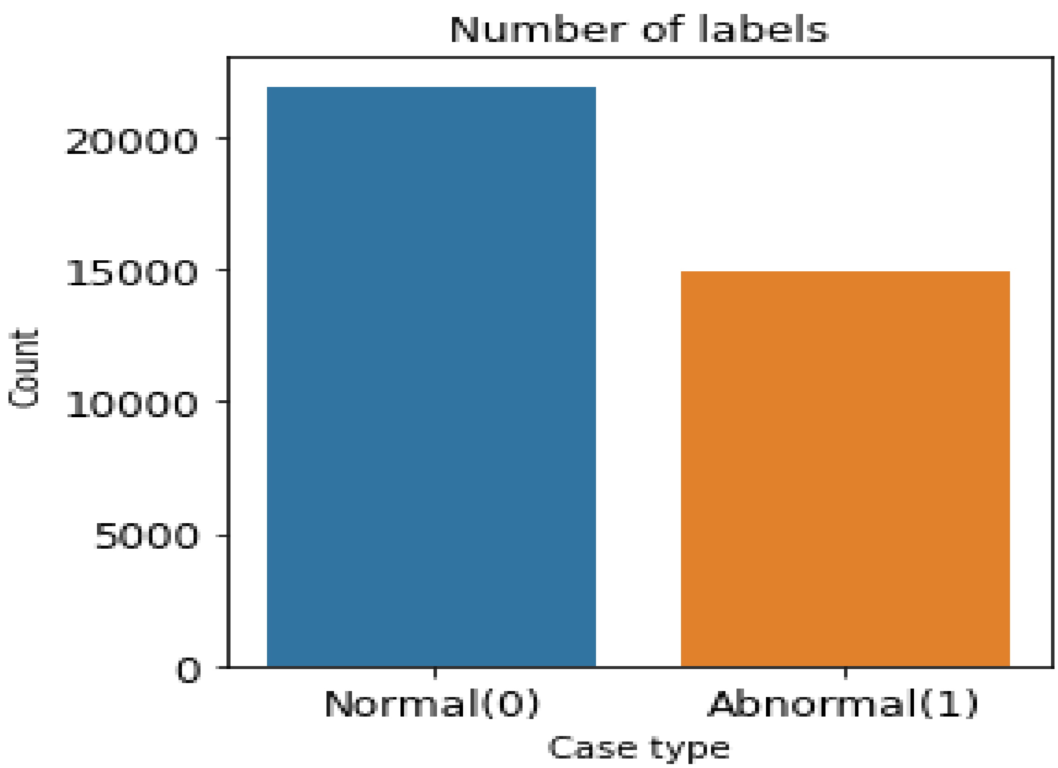


## Count individual number of labels of images in train set

```
cases_count = df['Train_Label'].value_counts()
print(cases_count)

# Plot the results
plt.figure(figsize=(4,4))
sns.barplot(x=cases_count.index, y=cases_count.values)
plt.title('Number of labels', fontsize=12)
plt.xlabel('Case type', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.xticks(range(len(cases_count.index)), ['Normal(0)', 'Abnormal(1)'])
plt.show()
```

```
0     21935
1     14873
Name: Train_Label, dtype: int64
```

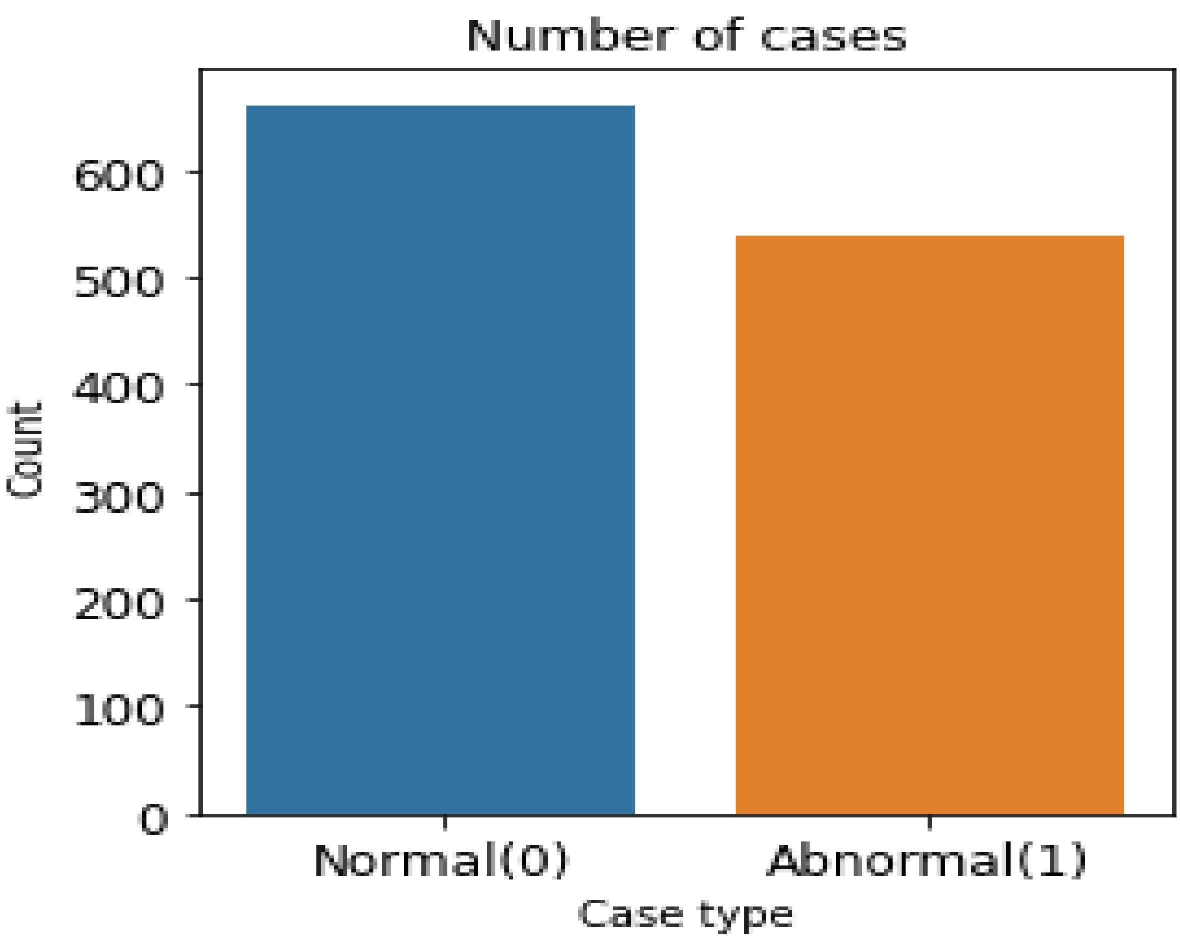


## Count labels in validation set

```
cases_count = df['Valid_Label'].value_counts()
print(cases_count)

# Plot the results
plt.figure(figsize=(4,4))
sns.barplot(x=cases_count.index, y=cases_count.values)
plt.title('Number of cases', fontsize=12)
plt.xlabel('Case type', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.xticks(range(len(cases_count.index)), ['Normal(0)', 'Abnormal(1)'])
plt.show()
```

```
0    661
1    538
Name: Valid_Label, dtype: int64
```

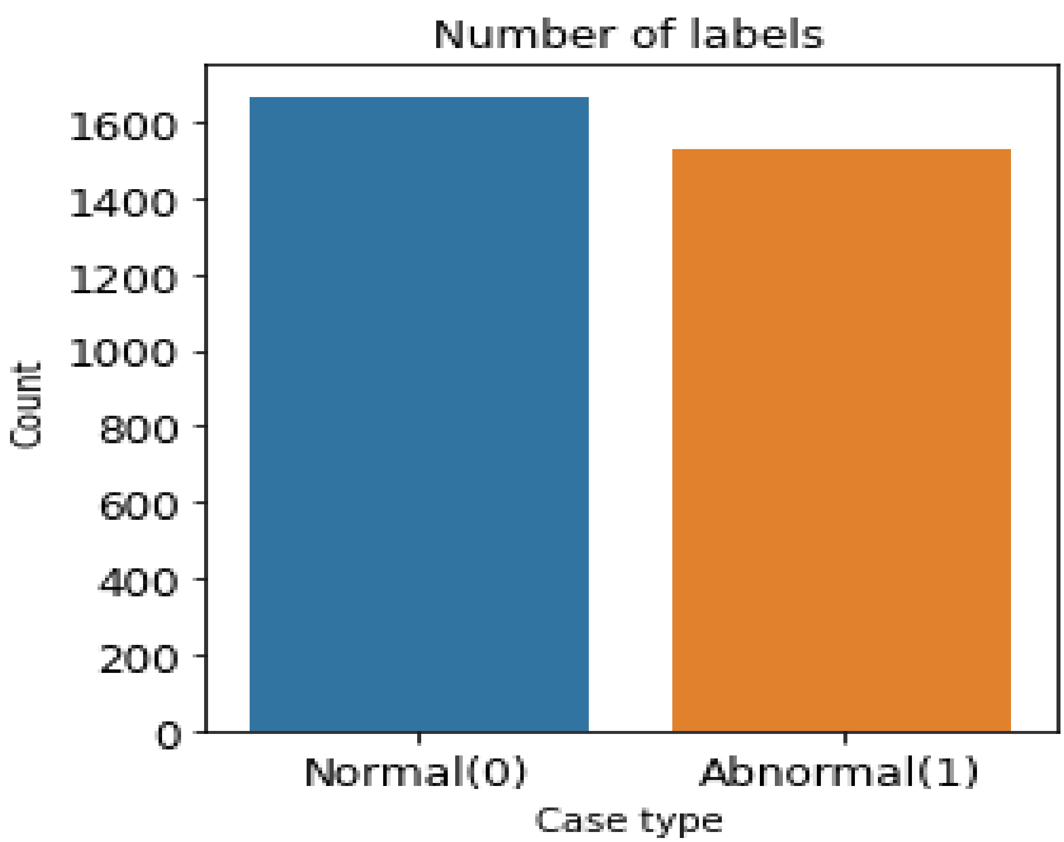


## Count individual number of labels of images in validation set

```python
cases_count = df['Valid_Label'].value_counts()
print(cases_count)

# Plot the results
plt.figure(figsize=(4,4))
sns.barplot(x=cases_count.index, y=cases_count.values)
plt.title('Number of labels', fontsize=12)
plt.xlabel('Case type', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.xticks(range(len(cases_count.index)), ['Normal(0)', 'Abnormal(1)'])
plt.show()
```

```
0    1667
1    1530
Name: Valid_Label, dtype: int64
```



## 2.4-DATA AUGMENTATION

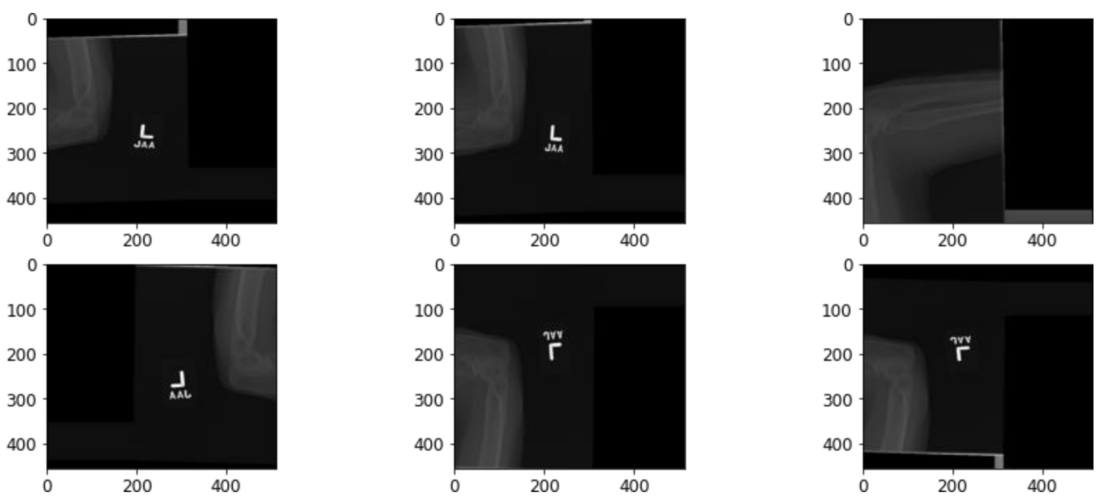
### Data Augmentation

```
[ ]  datagen = ImageDataGenerator(rescale=1./255, rotation_range=30,
                                  width_shift_range=0.2,
                                  height_shift_range=0.2,
                                  shear_range=0.2,
                                  zoom_range=0.2,
                                  horizontal_flip=True,
                                  fill_mode='nearest')
```

### Visualizing how data is being Augmented

**We Augmented one image to six different image**

- Random Shift (Width shift range, height shift range)
- Random Flip (Horizontal Flip, Vertical Flip)
- Random Zoom (Any value smaller than 1 will zoom in else zoom out)



## 2.5-Image Preprocessing

### Image Preprocessing

```
train_generator = datagen.flow_from_dataframe(dataframe=train_df, directory=None,
                                              x_col="Train_Image", y_col="Train_Label",
                                              target_size=(224,224), class_mode="binary",
                                              batch_size=64, shuffle=True)

valid_generator = datagen.flow_from_dataframe(dataframe=valid_df, directory=None,
                                              x_col="Valid_Image", y_col="Valid_Label",
                                              target_size=(224,224), class_mode="binary",
                                              batch_size=64, shuffle=True)
```

```
Found 36808 validated image filenames belonging to 2 classes.
Found 3197 validated image filenames belonging to 2 classes.
```

**Future Improvement :** Using Computer vision

- Implementing Edge Detection
- Highlighting Abnormal part of Bone in image

## 2.6-Trained different Pre-Trained Model for Transfer Learning taking best out off it.

- **ResNet-50**

```
from keras import regularizers
resnet_model=Sequential()
base_model = tf.keras.applications.resnet50.ResNet50(include_top=False,
                                                      input_shape=(224, 224, 3),
                                                      weights = 'imagenet')
for layer in base_model.layers:
    layer.trainable = False   #Freezing the weights of pre-trained model

resnet_model.add(base_model)
resnet_model.add(Flatten())

resnet_model.add(Dense(2048, activation="relu"))
resnet_model.add(Dense(1024,kernel_regularizer=regularizers.l2(0.01), activation="relu"))
resnet_model.add(Dense(512,kernel_regularizer=regularizers.l2(0.01), activation="relu"))
resnet_model.add(Dense(1, activation="sigmoid"))
resnet_model.compile(optimizer = Adam(learning_rate=0.001), loss="binary_crossentropy", metrics=["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 6s 0us/step
```

```
history = resnet_model.fit_generator(generator=train_generator,steps_per_epoch=train_steps,
                                     validation_data=valid_generator,validation_steps=valid_steps,
                                     epochs=10,
                                     callbacks=[checkpoint])
```

```
<ipython-input-29-9f18c90ad21f>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, whic
  history = resnet_model.fit_generator(generator=train_generator,steps_per_epoch=train_steps,
Epoch 1/10
575/575 [==============================] - ETA: 0s - loss: 3.7165 - accuracy: 0.5809
Epoch 1: val_accuracy improved from -inf to 0.52455, saving model to weights.hdf5
575/575 [==============================] - 688s 1s/step - loss: 3.7165 - accuracy: 0.5809 - val_loss: 1.7258 - val_accuracy: 0.5246
Epoch 2/10
575/575 [==============================] - ETA: 0s - loss: 1.2359 - accuracy: 0.5949
Epoch 2: val_accuracy did not improve from 0.52455
575/575 [==============================] - 685s 1s/step - loss: 1.2359 - accuracy: 0.5949 - val_loss: 0.9841 - val_accuracy: 0.5239
Epoch 3/10
575/575 [==============================] - ETA: 0s - loss: 0.8391 - accuracy: 0.5959
Epoch 3: val_accuracy did not improve from 0.52455
575/575 [==============================] - 626s 1s/step - loss: 0.8391 - accuracy: 0.5959 - val_loss: 0.8057 - val_accuracy: 0.5217
Epoch 4/10
575/575 [==============================] - ETA: 0s - loss: 0.7281 - accuracy: 0.5962
Epoch 4: val_accuracy did not improve from 0.52455
575/575 [==============================] - 599s 1s/step - loss: 0.7281 - accuracy: 0.5962 - val_loss: 0.7367 - val_accuracy: 0.5214
Epoch 5/10
575/575 [==============================] - ETA: 0s - loss: 0.6875 - accuracy: 0.5965
Epoch 5: val_accuracy did not improve from 0.52455
575/575 [==============================] - 637s 1s/step - loss: 0.6875 - accuracy: 0.5965 - val_loss: 0.7032 - val_accuracy: 0.5198
Epoch 6/10
575/575 [==============================] - ETA: 0s - loss: 0.6724 - accuracy: 0.5960
Epoch 6: val_accuracy improved from 0.52455 to 0.52679, saving model to weights.hdf5
575/575 [==============================] - 646s 1s/step - loss: 0.6724 - accuracy: 0.5960 - val_loss: 0.6930 - val_accuracy: 0.5268
Epoch 7/10
575/575 [==============================] - ETA: 0s - loss: 0.6687 - accuracy: 0.5966
Epoch 7: val_accuracy improved from 0.52679 to 0.52966, saving model to weights.hdf5
575/575 [==============================] - 679s 1s/step - loss: 0.6687 - accuracy: 0.5966 - val_loss: 0.7142 - val_accuracy: 0.5297
Epoch 8/10
575/575 [==============================] - ETA: 0s - loss: 0.6654 - accuracy: 0.5963
Epoch 8: val_accuracy did not improve from 0.52966
575/575 [==============================] - 664s 1s/step - loss: 0.6654 - accuracy: 0.5963 - val_loss: 0.6914 - val_accuracy: 0.5220
Epoch 9/10
575/575 [==============================] - ETA: 0s - loss: 0.6643 - accuracy: 0.5970
Epoch 9: val_accuracy did not improve from 0.52966
575/575 [==============================] - 655s 1s/step - loss: 0.6643 - accuracy: 0.5970 - val_loss: 0.7013 - val_accuracy: 0.5210
Epoch 10/10
575/575 [==============================] - ETA: 0s - loss: 0.6640 - accuracy: 0.5962
Epoch 10: val_accuracy did not improve from 0.52966
575/575 [==============================] - 608s 1s/step - loss: 0.6640 - accuracy: 0.5962 - val_loss: 0.6855 - val_accuracy: 0.5271
```

- **EfficientNetB0**

```
efficientnet_model=Sequential()
base_model = tf.keras.applications.efficientnet.EfficientNetB0(include_top=False,input_shape=(224, 224, 3),weights='imagenet')

for layer in base_model.layers:
    layer.trainable = False   #Freezing the weights of pre-trained model

efficientnet_model.add(base_model)
efficientnet_model.add(Flatten())
efficientnet_model.add(Dense(1280, activation="relu"))
efficientnet_model.add(Dropout(0.02))
efficientnet_model.add(Dense(640,kernel_regularizer=regularizers.l2(0.01), activation="relu"))
efficientnet_model.add(Dropout(0.03))
efficientnet_model.add(Dense(320,kernel_regularizer=regularizers.l2(0.01), activation="relu"))
efficientnet_model.add(Dense(1, activation="sigmoid"))
efficientnet_model.compile(optimizer = Adam(learning_rate=0.001), loss="binary_crossentropy", metrics=["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 0s 0us/step
```

```
history = efficientnet_model.fit(train_generator,steps_per_epoch=train_steps,
                                 epochs=10,validation_steps=valid_steps,
                                 validation_data=valid_generator)
```

```
Epoch 1/10
575/575 [==============================] - 627s 1s/step - loss: 2.0930 - accuracy: 0.5542 - val_loss: 1.1922 - val_accuracy: 0.5210
Epoch 2/10
575/575 [==============================] - 609s 1s/step - loss: 1.0126 - accuracy: 0.5949 - val_loss: 0.9284 - val_accuracy: 0.5207
Epoch 3/10
575/575 [==============================] - 607s 1s/step - loss: 0.8346 - accuracy: 0.5959 - val_loss: 0.8156 - val_accuracy: 0.5210
Epoch 4/10
575/575 [==============================] - 608s 1s/step - loss: 0.7594 - accuracy: 0.5960 - val_loss: 0.7666 - val_accuracy: 0.5236
Epoch 5/10
575/575 [==============================] - 606s 1s/step - loss: 0.7265 - accuracy: 0.5958 - val_loss: 0.7447 - val_accuracy: 0.5217
Epoch 6/10
575/575 [==============================] - 607s 1s/step - loss: 0.7091 - accuracy: 0.5960 - val_loss: 0.7288 - val_accuracy: 0.5210
Epoch 7/10
575/575 [==============================] - 613s 1s/step - loss: 0.6980 - accuracy: 0.5960 - val_loss: 0.7234 - val_accuracy: 0.5210
Epoch 8/10
575/575 [==============================] - 615s 1s/step - loss: 0.6902 - accuracy: 0.5959 - val_loss: 0.7141 - val_accuracy: 0.5201
Epoch 9/10
575/575 [==============================] - 608s 1s/step - loss: 0.6844 - accuracy: 0.5959 - val_loss: 0.7110 - val_accuracy: 0.5210
Epoch 10/10
575/575 [==============================] - 606s 1s/step - loss: 0.6805 - accuracy: 0.5958 - val_loss: 0.7050 - val_accuracy: 0.5207
```

- **VGG19**

```
from keras import regularizers
efficientnet_model=Sequential()
base_model = tf.keras.applications.vgg19.VGG19(include_top=False,input_shape=(224, 224, 3),weights='imagenet')

for layer in base_model.layers:
    layer.trainable = False   #Freezing the weights of pre-trained model

vgg19_model.add(base_model)
vgg19_model.add(Flatten())
vgg19_model.add(Dense(512, activation="relu"))
vgg19_model.add(Dropout(0.01))
vgg19_model.add(Dense(256,kernel_regularizer=regularizers.l2(0.01), activation="relu"))
vgg19_model.add(Dense(1, activation="sigmoid"))

vgg19_model.compile(optimizer = Adam(learning_rate=0.001), loss="binary_crossentropy", metrics=["accuracy"])
```

```
history = vgg19_model.fit(train_generator,steps_per_epoch=train_steps,
                          validation_data=valid_generator,validation_steps=valid_steps,
                          epochs=10,
                          callbacks=[checkpoint])
```

```
Epoch 1/10
575/575 [==============================] - ETA: 0s - loss: 1.1582 - accuracy: 0.6122
Epoch 1: val_accuracy improved from -inf to 0.60459, saving model to weights.hdf5
575/575 [==============================] - 684s 1s/step - loss: 1.1582 - accuracy: 0.6122 - val_loss: 0.7885 - val_accuracy: 0.6046
Epoch 2/10
575/575 [==============================] - ETA: 0s - loss: 0.7047 - accuracy: 0.6418
Epoch 2: val_accuracy improved from 0.60459 to 0.61161, saving model to weights.hdf5
575/575 [==============================] - 691s 1s/step - loss: 0.7047 - accuracy: 0.6418 - val_loss: 0.6988 - val_accuracy: 0.6116
Epoch 3/10
575/575 [==============================] - ETA: 0s - loss: 0.6511 - accuracy: 0.6520
Epoch 3: val_accuracy did not improve from 0.61161
575/575 [==============================] - 686s 1s/step - loss: 0.6511 - accuracy: 0.6520 - val_loss: 0.6926 - val_accuracy: 0.6052
Epoch 4/10
575/575 [==============================] - ETA: 0s - loss: 0.6297 - accuracy: 0.6579
Epoch 4: val_accuracy improved from 0.61161 to 0.61735, saving model to weights.hdf5
575/575 [==============================] - 668s 1s/step - loss: 0.6297 - accuracy: 0.6579 - val_loss: 0.6537 - val_accuracy: 0.6173
Epoch 5/10
575/575 [==============================] - ETA: 0s - loss: 0.6203 - accuracy: 0.6594
Epoch 5: val_accuracy improved from 0.61735 to 0.63106, saving model to weights.hdf5
575/575 [==============================] - 667s 1s/step - loss: 0.6203 - accuracy: 0.6594 - val_loss: 0.6333 - val_accuracy: 0.6311
Epoch 6/10
575/575 [==============================] - ETA: 0s - loss: 0.6096 - accuracy: 0.6647
Epoch 6: val_accuracy improved from 0.63106 to 0.64031, saving model to weights.hdf5
575/575 [==============================] - 666s 1s/step - loss: 0.6096 - accuracy: 0.6647 - val_loss: 0.6216 - val_accuracy: 0.6403
Epoch 7/10
575/575 [==============================] - ETA: 0s - loss: 0.6063 - accuracy: 0.6672
Epoch 7: val_accuracy improved from 0.64031 to 0.65370, saving model to weights.hdf5
575/575 [==============================] - 671s 1s/step - loss: 0.6063 - accuracy: 0.6672 - val_loss: 0.6158 - val_accuracy: 0.6537
Epoch 8/10
575/575 [==============================] - ETA: 0s - loss: 0.6019 - accuracy: 0.6694
Epoch 8: val_accuracy did not improve from 0.65370
575/575 [==============================] - 706s 1s/step - loss: 0.6019 - accuracy: 0.6694 - val_loss: 0.6510 - val_accuracy: 0.6033
Epoch 9/10
575/575 [==============================] - ETA: 0s - loss: 0.6005 - accuracy: 0.6726
Epoch 9: val_accuracy did not improve from 0.65370
575/575 [==============================] - 633s 1s/step - loss: 0.6005 - accuracy: 0.6726 - val_loss: 0.6321 - val_accuracy: 0.6231
Epoch 10/10
575/575 [==============================] - ETA: 0s - loss: 0.6001 - accuracy: 0.6709
Epoch 10: val_accuracy did not improve from 0.65370
575/575 [==============================] - 631s 1s/step - loss: 0.6001 - accuracy: 0.6709 - val_loss: 0.6146 - val_accuracy: 0.6460
```

- **DenseNet169**

```
from keras import regularizers
densenet_model=Sequential()
base_model = tf.keras.applications.densenet.DenseNet169(include_top=False,
                                                        input_shape=(224, 224, 3),
                                                        weights = 'imagenet')
for layer in base_model.layers:
    layer.trainable = False   #Freezing the weights of pre-trained model

densenet_model.add(base_model)
#densenet_model.add(Flatten())
densenet_model.add(GlobalAveragePooling2D())
densenet_model.add(Dense(1664, activation="relu"))
densenet_model.add(Dropout(0.02))
densenet_model.add(Dense(832, activation="relu"))

densenet_model.add(Dense(1, activation="sigmoid"))
densenet_model.compile(optimizer = Adam(learning_rate=0.001), loss="binary_crossentropy", metrics=["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet169_weights_tf_d
51877672/51877672 [==============================] - 2s 0us/step
```

```
history = densenet_model.fit(train_generator,steps_per_epoch=train_steps,
                              validation_data=valid_generator,validation_steps=valid_steps,
                              epochs=10,
                              callbacks=[checkpoint])
```

```
Epoch 1/10
575/575 [==============================] - ETA: 0s - loss: 0.6007 - accuracy: 0.6874
Epoch 1: val_accuracy improved from -inf to 0.66645, saving model to weights.hdf5
575/575 [==============================] - 688s 1s/step - loss: 0.6007 - accuracy: 0.6874 - val_loss: 0.5996 - val_accuracy: 0.6665
Epoch 2/10
575/575 [==============================] - ETA: 0s - loss: 0.5538 - accuracy: 0.7185
Epoch 2: val_accuracy improved from 0.66645 to 0.71429, saving model to weights.hdf5
575/575 [==============================] - 680s 1s/step - loss: 0.5538 - accuracy: 0.7185 - val_loss: 0.5465 - val_accuracy: 0.7143
Epoch 3/10
575/575 [==============================] - ETA: 0s - loss: 0.5434 - accuracy: 0.7273
Epoch 3: val_accuracy did not improve from 0.71429
575/575 [==============================] - 650s 1s/step - loss: 0.5434 - accuracy: 0.7273 - val_loss: 0.5630 - val_accuracy: 0.7057
Epoch 4/10
575/575 [==============================] - ETA: 0s - loss: 0.5374 - accuracy: 0.7330
Epoch 4: val_accuracy did not improve from 0.71429
575/575 [==============================] - 613s 1s/step - loss: 0.5374 - accuracy: 0.7330 - val_loss: 0.5576 - val_accuracy: 0.7089
Epoch 5/10
575/575 [==============================] - ETA: 0s - loss: 0.5324 - accuracy: 0.7352
Epoch 5: val_accuracy did not improve from 0.71429
575/575 [==============================] - 610s 1s/step - loss: 0.5324 - accuracy: 0.7352 - val_loss: 0.5472 - val_accuracy: 0.7133
Epoch 6/10
575/575 [==============================] - ETA: 0s - loss: 0.5256 - accuracy: 0.7404
Epoch 6: val_accuracy improved from 0.71429 to 0.72991, saving model to weights.hdf5
575/575 [==============================] - 609s 1s/step - loss: 0.5256 - accuracy: 0.7404 - val_loss: 0.5362 - val_accuracy: 0.7299
Epoch 7/10
575/575 [==============================] - ETA: 0s - loss: 0.5221 - accuracy: 0.7415
Epoch 7: val_accuracy did not improve from 0.72991
575/575 [==============================] - 643s 1s/step - loss: 0.5221 - accuracy: 0.7415 - val_loss: 0.5413 - val_accuracy: 0.7229
Epoch 8/10
575/575 [==============================] - ETA: 0s - loss: 0.5179 - accuracy: 0.7472
Epoch 8: val_accuracy did not improve from 0.72991
575/575 [==============================] - 643s 1s/step - loss: 0.5179 - accuracy: 0.7472 - val_loss: 0.5463 - val_accuracy: 0.7219
Epoch 9/10
575/575 [==============================] - ETA: 0s - loss: 0.5151 - accuracy: 0.7489
Epoch 9: val_accuracy improved from 0.72991 to 0.73342, saving model to weights.hdf5
575/575 [==============================] - 612s 1s/step - loss: 0.5151 - accuracy: 0.7489 - val_loss: 0.5281 - val_accuracy: 0.7334
Epoch 10/10
575/575 [==============================] - ETA: 0s - loss: 0.5135 - accuracy: 0.7471
Epoch 10: val_accuracy did not improve from 0.73342
575/575 [==============================] - 615s 1s/step - loss: 0.5135 - accuracy: 0.7471 - val_loss: 0.5331 - val_accuracy: 0.7328
```

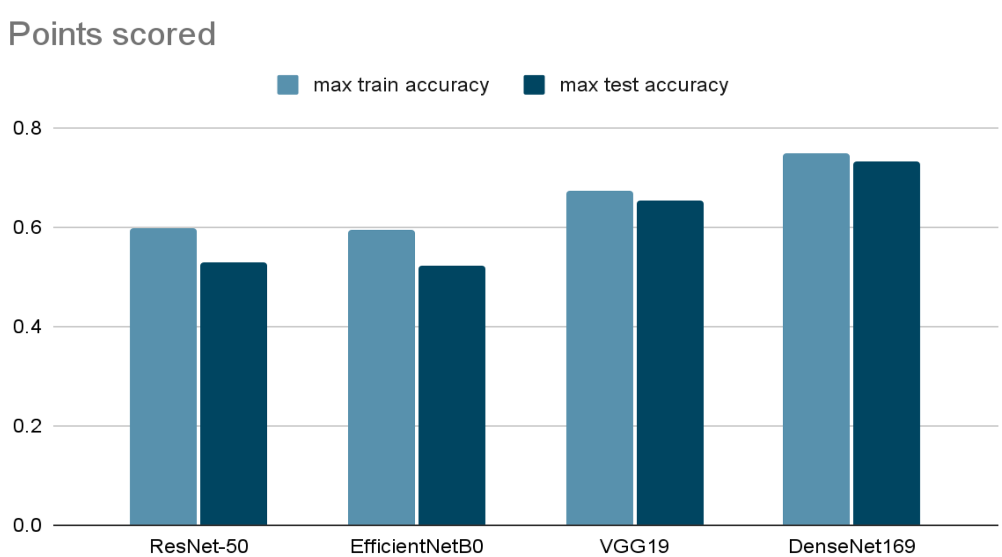
## 2.7-Benchmark on Training different Pre-Trained Model

| Epochs | ResNet-50 | EfficientNetB0 | VGG19 | DenseNet169 |
|---|---|---|---|---|
| 1 | 0.5809 | 0.5542 | 0.6122 | 0.687378 |
| 2 | 0.5949 | 0.5949 | 0.6418 | 0.718512 |
| 3 | 0.5959 | 0.5959 | 0.6520 | 0.727302 |
| 4 | 0.5962 | 0.5960 | 0.6579 | 0.733018 |
| 5 | 0.5965 | 0.5958 | 0.6594 | 0.735195 |
| 6 | 0.5960 | 0.5960 | 0.6647 | 0.740447 |
| 7 | 0.5966 | 0.5960 | 0.6672 | 0.741536 |
| 8 | 0.5963 | 0.5959 | 0.6694 | 0.747170 |
| 9 | 0.5970 | 0.5959 | 0.6726 | 0.748939 |
| 10 | 0.5962 | 0.5958 | 0.6709 | 0.747115 |
| Max% | 0.5970 | 0.5960 | 0.6726 | 0.748939 |

## 2.8- Benchmark on Testing different Pre-Trained Model

| Epochs | ResNet-50 | EfficientNetB0 | VGG19 | DenseNet169 |
|---|---|---|---|---|
| 1 | 0.5246 | 0.5210 | 0.6046 | 0.666454 |
| 2 | 0.5239 | 0.5207 | 0.6116 | 0.714286 |
| 3 | 0.5217 | 0.5210 | 0.6052 | 0.705676 |
| 4 | 0.5214 | 0.5236 | 0.6173 | 0.708865 |
| 5 | 0.5198 | 0.5217 | 0.6311 | 0.713329 |
| 6 | 0.5268 | 0.5210 | 0.6403 | 0.729911 |
| 7 | 0.5297 | 0.5210 | 0.6537 | 0.722895 |
| 8 | 0.5220 | 0.5201 | 0.6033 | 0.721939 |
| 9 | 0.5210 | 0.5210 | 0.6231 | 0.733418 |
| 10 | 0.5271 | 0.5207 | 0.6460 | 0.732781 |
| Max% | 0.5297 | 0.5236 | 0.6537 | 0.733418 |

## 2.9-Evaluting the highest performance per-trained model




| | | | | |
|---|---|---|---|---|
| Max Train Accuracy | 0.5970 | 0.5960 | 0.6726 | 0.748939 |
| Max Test Accuracy | 0.5297 | 0.5236 | 0.6537 | 0.733418 |

## 2.10-Implementing Transfer Learning

```
densenet_model=Sequential()
base_model = tf.keras.applications.densenet.DenseNet169(include_top=False,
                                                         input_shape=(224, 224, 3),
                                                         weights = 'imagenet')
for layer in base_model.layers:
    layer.trainable = False   #Freezing the weights of pre-trained model
```

## 2.11-Adding layers on top of Base model

```
densenet_model.add(base_model)
#densenet_model.add(Flatten())
densenet_model.add(GlobalAveragePooling2D())
densenet_model.add(Dense(1664, activation="relu"))
densenet_model.add(Dropout(0.02))
densenet_model.add(Dense(832, activation="relu"))

densenet_model.add(Dense(1, activation="sigmoid"))
densenet_model.compile(optimizer = Adam(learning_rate=0.001), loss="binary_crossentropy", metrics=["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet169_weights_t
51877672/51877672 [==============================] - 2s 0us/step
```

## 2.12-Fine Tuning

### Fine Tunning

```
[ ] base_model.trainable = True
```

```
[ ] # Let's take a look to see how many layers are in the base model
    print("Number of layers in the base model: ", len(base_model.layers))
```

```
Number of layers in the base model:  595
```

```
# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

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

## 2.13-Model Fine Tuning

```
[ ] fine_tune_epochs = 10
    total_epochs =  10 + fine_tune_epochs

    history_fine = densenet_model.fit(train_generator,
                                  epochs=total_epochs,
                                  initial_epoch=history.epoch[-1],
                                  validation_data=valid_generator)
```

```
Epoch 10/20
576/576 [==============================] - 794s 1s/step - loss: 0.5405 - accuracy: 0.7361 - val_loss: 0.5826 - val_accuracy: 0.7457
Epoch 11/20
576/576 [==============================] - 728s 1s/step - loss: 0.4970 - accuracy: 0.7702 - val_loss: 0.6363 - val_accuracy: 0.7075
Epoch 12/20
576/576 [==============================] - 692s 1s/step - loss: 0.4819 - accuracy: 0.7814 - val_loss: 0.4934 - val_accuracy: 0.7710
Epoch 13/20
576/576 [==============================] - 725s 1s/step - loss: 0.4714 - accuracy: 0.7866 - val_loss: 0.5103 - val_accuracy: 0.7682
Epoch 14/20
576/576 [==============================] - 685s 1s/step - loss: 0.4622 - accuracy: 0.7927 - val_loss: 0.5399 - val_accuracy: 0.7667
Epoch 15/20
576/576 [==============================] - 684s 1s/step - loss: 0.4570 - accuracy: 0.7955 - val_loss: 0.5365 - val_accuracy: 0.7285
Epoch 16/20
576/576 [==============================] - 716s 1s/step - loss: 0.4503 - accuracy: 0.8009 - val_loss: 0.4822 - val_accuracy: 0.7785
Epoch 17/20
576/576 [==============================] - 693s 1s/step - loss: 0.4456 - accuracy: 0.8024 - val_loss: 0.4699 - val_accuracy: 0.7860
Epoch 18/20
576/576 [==============================] - 697s 1s/step - loss: 0.4415 - accuracy: 0.8035 - val_loss: 0.4603 - val_accuracy: 0.7970
Epoch 19/20
576/576 [==============================] - 688s 1s/step - loss: 0.4341 - accuracy: 0.8099 - val_loss: 0.4806 - val_accuracy: 0.7810
Epoch 20/20
576/576 [==============================] - 704s 1s/step - loss: 0.4310 - accuracy: 0.8117 - val_loss: 0.4908 - val_accuracy: 0.7851
```

# Chapter 3
# Implementation
# (3.1)

## Implementation

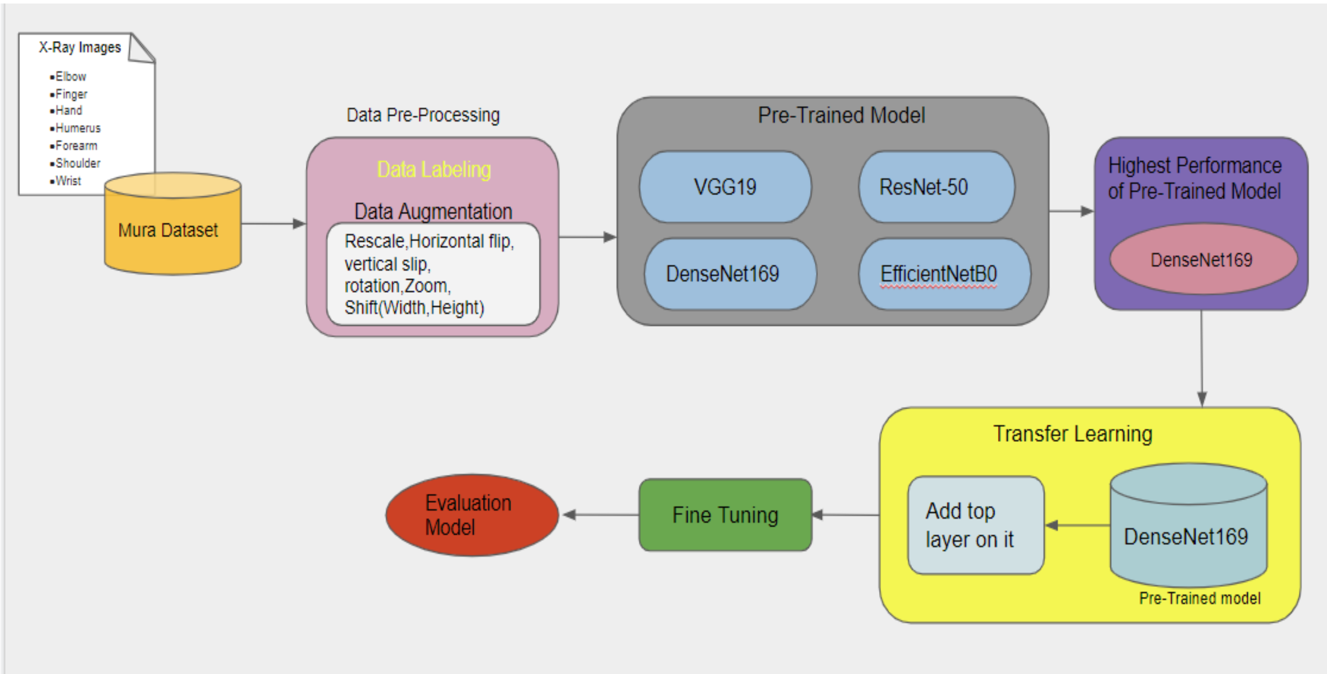
Use of Python Platform for writing the code with Keras, TensorFlow, OpenCV

**Hardware Configuration:**
CPU: 8 GB RAM, Quad core processor
GPU: 16GB RAM
Laptop

**Software Required:**
Jupyter Notebook: JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones.

**Google colab(PYTHON)**
Colab or colaboratory is a product from Google research . Colab allows anybody to write and execute arbitrary python code through the browser and is especially well suited to machine learning data analysis and education.

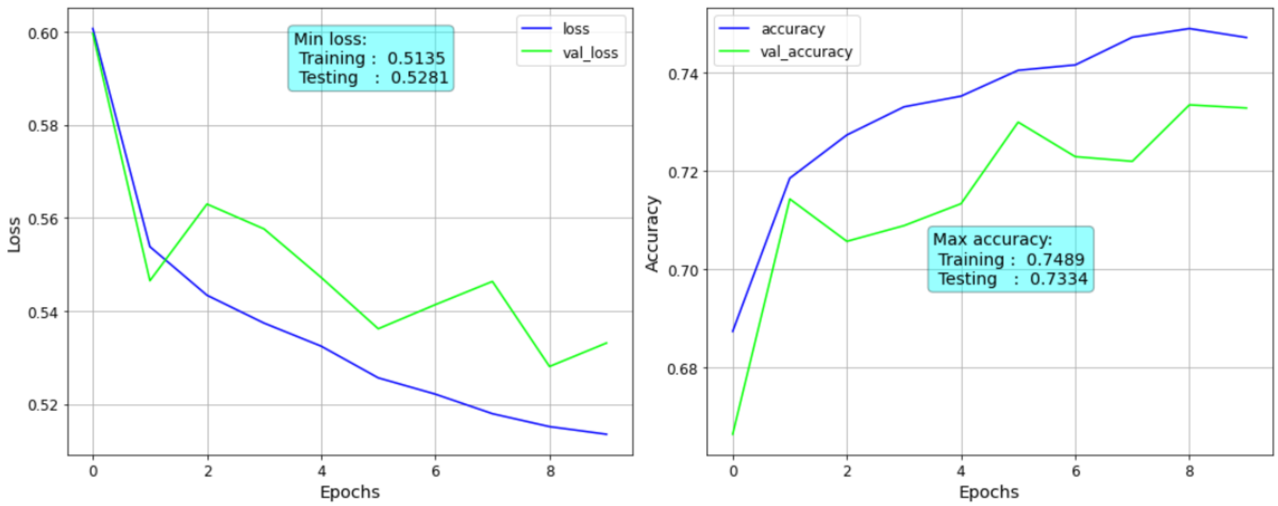
## 3.2-Flow Chart of Application



# Chapter 4
# Results
# (4.1)

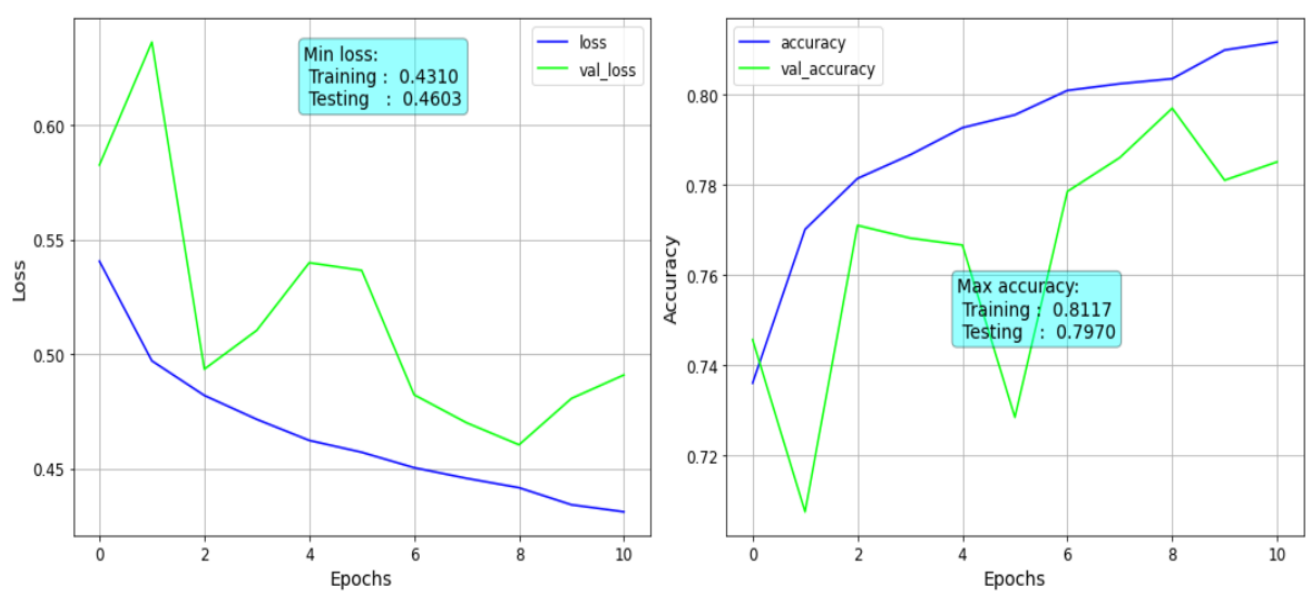
## Accuracy and loss graph after implementing Transfer Learning

```
[ ]  fn_plot_hist(res_df)
```

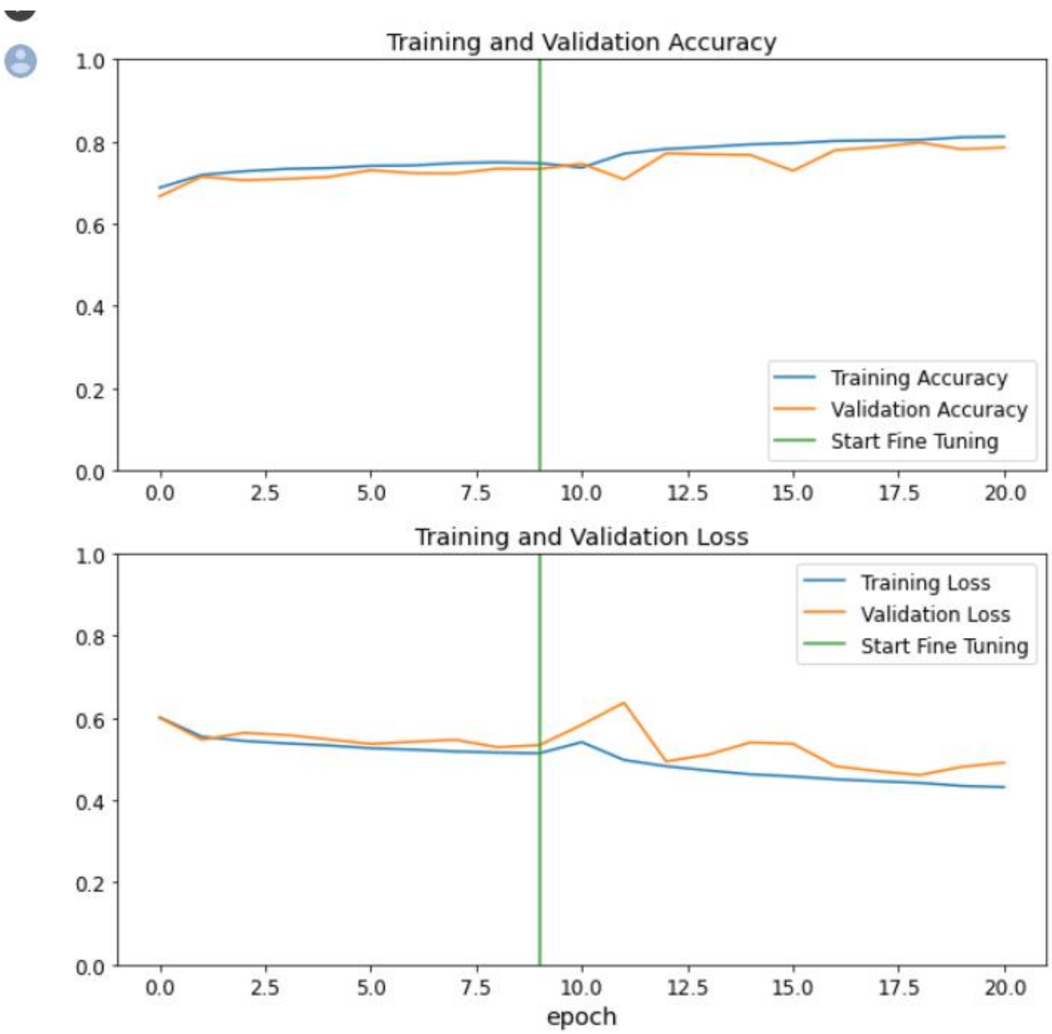



## Accuracy and loss graph after implementing Fine Tuning

```
fn_plot_hist(res_df1)
```



## Overall Performance of our Model



# Chapter5
# Conclusion
# (5.1)

## Conclusion

First, an abnormality detection model could be utilized for worklist prioritization. In this scenario, the studies detected as abnormal could be moved ahead in the image interpretation workflow, allowing the sickest patients to receive quicker diagnosis and treatment.

Furthermore, the examinations identified as normal could be automatically assigned a preliminary reading of ”normal”; this could mean (1) normal examinations can be properly triaged as lower priority on a worklist (2) more rapid results can be conveyed to the ordering provider (and patient) which could improve disposition in other areas of the healthcare system (i.e., discharged from the ED more quickly) (3) a radiology report template for the normal study could be served to the interpreting radiologist for more rapid review and approval.

# Chapter 6
# References
# (6.1)

## References:


- https://stanfordmlgroup.github.io/competitions/mura/
- https://arxiv.org/abs/1712.06957
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- https://www.tensorflow.org/tutorials/images/transfer_learning
- https://www.kaggle.com/code/kmader/mura-data-overview/data
- https://colab.research.google.com/drive/1i3v5gMoP_JEKXwx8JrTz78UjM_ZkFJLD?usp=sharing