```
import numpy as np
import pandas as pd
import tensorflow as tf
import os
import cv2
import matplotlib.pyplot as plt
from tqdm import tqdm

path="ImagePro"
files=os.listdir(path)
files.sort()
print(files)



image_array=[]
label_array=[]

for i in tqdm(range(len(files))):

        sub_file=os.listdir(path+"/"+files[i])

        #          print(len(sub_file))


        for j in range(len(sub_file)):



                file_path=path+"/"+files[i]+"/"+sub_file[j]
```

```
                image=cv2.imread(file_path)

                # resize image by 96x96
                image=cv2.resize(image,(96,96))
                # convert BGR image to RGB image
                image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

                # add this image at image_array
                image_array.append(image)

                # add label to label_array
                # i is number from 0 to len(files)-1
                # so we can use it as label
                label_array.append(i)

# save and run to see if it is working or not
# before that apply tqdm to for loop

# convert list to array

image_array=np.array(image_array)
label_array=np.array(label_array,dtype="float")

# split the dataset into test and train

from sklearn.model_selection import train_test_split
# output                                                    train image   label
spliting size
X_train,X_test,Y_train,Y_test=train_test_split(image_array,label_array,test_size=0.15)

del image_array,label_array
```

```
# to free memory
import gc
gc.collect()


# X_train will have 85% of images
# X_test will have 15% of images


# Create a model

from keras import layers,callbacks,utils,applications,optimizers
from keras.models import Sequential, Model, load_model

model=Sequential()
# add pretrained models to Sequential model
# I will use EfficientNetB0 pretrained model. You can try different model.
pretrained_model=tf.keras.applications.EfficientNetB0(input_shape=(96,96,3),include_top=False)
model.add(pretrained_model)

# add Pooling to model
model.add(layers.GlobalAveragePooling2D())

# add dropout to model
# We add dropout to increase accuracy by reduce overfitting
model.add(layers.Dropout(0.3))
# finally we will addd dense layer as an output
model.add(layers.Dense(1))
# For some tensorflow version we required to build model
model.build(input_shape=(None,96,96,3))
```

```
# to see model summary
model.summary()


model.compile(optimizer="adam",loss="mae",metrics=["mae"])

# create a checkpoint to save best accuracy model
ckp_path="trained_model/model"
model_checkpoint=tf.keras.callbacks.ModelCheckpoint(
                                                    filepath=ckp_path,
                                                    monitor="val_mae",
                                                    mode="auto",
                                                    save_best_only=True,
                                                    save_weights_only=True
                                                    )
# monitor: monitor validation mae loss to save model
# mode: Use to save model when val_mae is minimum or maximum
# It has 3 option: "min","max","auto".
# for us you can select either "min" or "auto"
# When val_mae reduce model will be saved
# save_best_only: False -> It will save all model
# save_weights_only: Save only weight.


# create learning rate reducer to reduce lr when accuracy does not improve
# Correct
reduce_lr=tf.keras.callbacks.ReduceLROnPlateau(
                                               factor=0.9,
                                               monitor="val_mae",
```

```
                                                    mode="auto",
                                                    cooldown=0,
                                                    patience=5,
                                                    verbose=1,
                                                    min_lr=1e-6)

# factor: when it is reduce next lr will be 0.9 times of current
# next lr= 0.9* current lr

# patience=X
# reduce lr after X epoch when accuracy does not improve
# verbose : show it after every epoch

# min_lr : minimum learning rate

# Start training model

Epochs=100
Batch_Size=32
# Select batch size according to your Graphic card
#
#X_train,X_test,Y_train,Y_test
history=model.fit(
                                    X_train,
                                    Y_train,
                                    validation_data=(X_test,Y_test),
                                    batch_size=Batch_Size,
                                    epochs=Epochs,
                                    callbacks=[model_checkpoint,reduce_lr]
                                    )
# Before training you can delete image_array and label_array to increase ram memory
```

```
# Save and run
# Everything is working

# after the training is done load best model

model.load_weights(ckp_path)

# convert model to tensorflow lite model

converter=tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model=converter.convert()

# save model
with open("model.tflite","wb") as f:
        f.write(tflite_model)

# if you want to see prediction result on test dataset
prediction_val=model.predict(X_test,batch_size=32)

# print first 10 values
print(prediction_val[:10])
# print first 10 values of Y_test
print(Y_test[:10])

# Save and run this python file
# loss: 0.4074 - mae: 0.4074 - val_loss: 0.3797 - val_mae: 0.3797
# we have mae and val_mae:
# mae: Is on X_train
# val_mae: X_test
# If val_mae is reducing that means your model is improving.
```