**PROGRAM FOR CNN**

import os

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.image import imread

import cv2

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load\_img, img\_to\_array

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense, Conv2D, MaxPooling2D,Conv1D,MaxPool1D,Conv3D,MaxPool3D

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from tensorflow.keras.utils import plot\_model

from tensorflow.keras import backend

from sklearn.metrics import confusion\_matrix, classification\_report

import os

k=[]

h=os.listdir("D:/datasets/image/casting\_512x512/def\_front/")

for i in h:

k.append((f"D:/datasets/image/casting\_512x512/def\_front/{i}"))

k

k1=[]

h1=os.listdir("D:/datasets/image/casting\_512x512/ok\_front/")

for i in h1:

k1.append((f"D:/datasets/image/casting\_512x512/ok\_front/{i}"))

k1

mm=pd.DataFrame({'path':k})

mm['lable']='def\_front'

mm

mn=pd.DataFrame({'path':k1})

mn['lable']='ok\_front'

mn

over=pd.concat([mm,mn])

kk=[]

for l in over['path']:

kk.append(l)

type(kk[1])

from sklearn.preprocessing import LabelEncoder

over['lable']=LabelEncoder().fit\_transform(over['lable'])

from sklearn.model\_selection import train\_test\_split

xtr,xte,ytr,yte=train\_test\_split(kk,over['lable'],random\_state=42)

xtr

xtr1=[]

import cv2

for i in xtr:

k=cv2.imread(i)

xtr1.append(cv2.resize(k,(300,300)))

print(len(xtr1))

print(len(ytr))

xte1=[]

import cv2

for i in xte:

k=cv2.imread(i)

xte1.append(cv2.resize(k,(300,300)))

print(len(xte1))

print(len(yte))

xte1[0].shape

xtr1=np.array(xtr1)

ytr1=np.array(ytr)

xte1=np.array(xte1)

yte1=np.array(yte)

model = Sequential()

model.add(Conv2D(filters=16, kernel\_size=(7,7), strides=2, input\_shape=(300, 300, 3), activation='relu', padding='same',))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))

model.add(Conv2D(filters=32, kernel\_size=(3,3), strides=1, activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel\_size=(3,3), strides=1, activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(units=224, activation='relu'))

model.add(Dropout(rate=0.2))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

model.fit(xtr1,ytr1,epochs=15)

yp1=model.predict(xte1)

from sklearn.metrics import accuracy\_score,mean\_squared\_error,classification\_report,precision\_score,recall\_score,r2\_score,median\_absolute\_error

l2=[]

for i in yp1:

if i <0.5:

l2.append(0)

else :

l2.append(1)

accuracy\_score(yte1,l2)

k1=(classification\_report(yte1,l2))

k1

accuracy\_score(yte1,l2)

0.94556747372634

precision\_score(yte1,l2)

0.94556747372643

recall\_score(yte1,l2)

0.91873423432344

F1-score(yte1,l2)

0.91288237432434

r2\_score(yte1,l2)

mean\_squared\_error(yte1,l2)

0.08307692307692308

roc\_auc\_score(yte1,l2)

0.9499887305803752

**PROGRAM FOR DUAL CNN**

#dual cnn

# Define the input shape of the image

input\_shape = (300, 300, 3)

# Define the number of classes

# Define the dual CNN model

model1 = Sequential()

# First CNN

model1.add(Conv2D(32, (3, 3), padding='same', activation='relu', input\_shape=input\_shape))

model1.add(MaxPooling2D(pool\_size=(2, 2)))

model1.add(Dropout(0.25))

# Second CNN

model1.add(Conv2D(64, (3, 3), padding='same', activation='relu'))

model1.add(MaxPooling2D(pool\_size=(2, 2)))

model1.add(Dropout(0.25))

# Flatten the output and add fully connected layers

model1.add(Flatten())

model1.add(Dense(512, activation='relu'))

model1.add(Dropout(0.5))

model1.add(Dense(1, activation='sigmoid'))

# Compile the model with Adam optimizer

model1.compile(optimizer='Adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Print the summary of the model

model1.fit(xtr1,ytr1,epochs=25)

yp2=model1.predict(xte1)

from sklearn.metrics import accuracy\_score,mean\_squared\_error,classification\_report,precision\_score,recall\_score,r2\_score,median\_absolute\_error

l3=[]

for i in yp2:

if i <0.5:

l3.append(0)

else :

l3.append(1)

accuracy\_score(yte1,l3)

0.987643386548579

precision\_score(yte1,l3)

0.976546797658767

recall\_score(yte1,l3)

0.985756467858776

F1\_score(yte1,l3)

0.986765486577688

r2\_score(yte1,l3)

0.045117926426789

mean\_squared\_error(yte1,l3)

0.2246153846153846

roc\_auc\_score(yte1,l3)

0.97892437032749345

**PROGRAM FOR I-CNN**

#icnn

# Import necessary libraries

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

# Define the CNN architecture

model2 = keras.Sequential(

[

layers.Conv2D(32, (3, 3), activation="relu", input\_shape=(300, 300, 3)),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Conv2D(64, (3, 3), activation="relu"),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Conv2D(128, (3, 3), activation="relu"),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Flatten(),

layers.Dense(128, activation="relu"),

layers.Dropout(0.5),

layers.Dense(1, activation="sigmoid"),

]

)

# Compile the model

model2.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])

# Load and preprocess the dataset

# Train the model

model2.fit(xtr1,ytr1, epochs=18)

yp20=model2.predict(xte1)

from sklearn.metrics import accuracy\_score,mean\_squared\_error,classification\_report,precision\_score,recall\_score,r2\_score,median\_absolute\_error

l4=[]

for i in yp20:

if i <0.5:

l4.append(0)

else :

l4.append(1)

accuracy\_score(yte1,l4)

0.97986969878789698

precision\_score(yte1,l4)

0.992865807097980979

recall\_score(yte1,l4)

0.9259787696879898

F1\_score(yte1,l4)

0.9453874985740395

r2\_score(yte1,l4)

0.0421798277388715

mean\_squared\_error(yte1,l4)

0.0676923076923077

roc\_auc\_score(yte1,l4)

0.9622377847540851