Date	**15-10-2022**
Name	**P.komala**
Project name	**AI based Nutrient analysis and fitness ethusiastics**

Extract zip Folder

```
!unzip '/content/Flowers_Dataset.zip'
```

Data Augmentation

It is a technique used to increase the input images with slight changes. By doing this we can overcome overfitting problem.

Importing req. lib

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Initializing data augmentation to training variable

```
train_datagen = ImageDataGenerator(rescale= zoom_range=0.2,
horizontal_flip=True)

# Data augmentation on training data

xtrain = train_datagen.flow_from_directory('/content/dataset/Training',
                                           target_size=(64,64),
                                           class_mode='categorical',
                                           batch_size=100)

# Data augmentation on testing data

xtest = test_datagen.flow_from_directory('/content/dataset/Testing',
                                         target_size=(64,64),
                                         class_mode='categorical',
                                         batch_size=100)
```

CNN MODEL TRAINING

 # Import req. lib

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense
```

 # Building CNN Block

```
model = Sequential() # Initializing the model
model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3))) # Covolution layer
model.add(MaxPooling2D(pool_size=(2,2))) # Max pooling layer
model.add(Flatten()) # Flatten layer
model.add(Dense(300,activation='relu')) # Hidden layer 1
model.add(Dense(150,activation='relu')) # Hidden layer 2
model.add(Dense(4,activation='softmax')) # Output layer
```

 # Compiling the model

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

 # Training model

```
model.fit_generator(xtrain,
                    steps_per_epoch=len(xtrain),
                    epochs=10,
                    validation_data=xtest,
                    validation_steps=len(xtest))
```

 # Saving Model

```
model.save('Flowers.h5')
```

TESTING MODEL

```
import numpy as np
from tensorflow.keras.preprocessing import image


# Testing 1

img = image.load_img('/content/dataset/Testing/bears/k4 (88).jpeg',target_size=(64,64))
                                                    # Reading image
x = image.img_to_array(img) # Converting image to array
x = np.expand_dims(x,axis=0) # Expanding dimension
pred = np.argmax(model.predict(x)) # Predicting higher prob. index
print(pred, model.predict(x))
op = ['bears','crows','elephants','rats'] # Creating list of output categories
print(op[pred]) # Matching the index

# Testing 2

img =
image.load_img('/content/dataset/Testing/elephants/photo_1485579163316_2b0b19c43b79.jp
eg',target_size=(64,64)) # Reading image
x = image.img_to_array(img) # Converting image to array
x = np.expand_dims(x,axis=0) # Expanding dimension
pred = np.argmax(model.predict(x)) # Predicting higher prob. index
print(pred, model.predict(x))
op = ['bears','crows','elephants','rats'] # Creating list of output categories
print(op[pred]) # Matching the index

# Testing 3

img = image.load_img('/content/dataset/Testing/rats/images (93).jpeg',target_size=(64,64)) #
Reading image
x = image.img_to_array(img) # Converting image to array
x = np.expand_dims(x,axis=0) # Expanding dimension
pred = np.argmax(model.predict(x)) # Predicting higher prob. index
print(pred, model.predict(x))
op = ['bears','crows','elephants','rats'] # Creating list of output categories
print(op[pred]) # Matching the index
```

```
# Testing 4 (Google image)

img = image.load_img('/content/Corvus_corone_-near_Canford_Cliffs,_Poole,_England-
8.jpg',target_size=(64,64)) # Reading image
x = image.img_to_array(img) # Converting image to array
x = np.expand_dims(x,axis=0) # Expanding dimension
pred = np.argmax(model.predict(x)) # Predicting higher prob. index
print(pred, model.predict(x))
op = ['bears','crows','elephants','rats'] # Creating list of output categories
print(op[pred]) # Matching the index

# Testing 5 (Google image)

img = image.load_img('/content/dataset/Testing/bears/k4 (100).jpeg',target_size=(64,64)) #
Reading image
x = image.img_to_array(img) # Converting image to array
x = np.expand_dims(x,axis=0) # Expanding dimension
pred = np.argmax(model.predict(x)) # Predicting higher prob. index
print(pred, model.predict(x))
op = ['bears','crows','elephants','rats'] # Creating list of output categories
print(op[pred]) # Matching the index

xtrain.class_indices
```

TUNING MODEL

```
# Importing required lib.

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
early_stop = EarlyStopping(monitor='val_accuracy',
                           patience=5)

lr = ReduceLROnPlateau(monitor='val_accuracy',
                       factor=0.5,
                       patience=5,
                       min_lr=0.00001)
```

```
callbacks = [early_stop,lr]
# Training model

model.fit_generator(xtrain,
                    steps_per_epoch=len(xtrain),
                    epochs=100,
                    callbacks=callbacks,
                    validation_data=xtest,
                    validation_steps=len(xtest),)
# Testing 5

img = image.load_img('/content/dataset/Testing/bears/k4 (74).jpeg',target_size=(64,64)) #
Reading image
x = image.img_to_array(img) # Converting image to array
x = np.expand_dims(x,axis=0) # Expanding dimension
pred = np.argmax(model.predict(x)) # Predicting higher prob. index
print(pred, model.predict(x))
op = ['bears','crows','elephants','rats'] # Creating list of output categories
print(op[pred]) # Matching the index
```

ADDING ANN LAYERS

```
# Importing required libraries

import numpy as np
import pandas as pd
# Reading the dataset

df = pd.read_csv('/content/50_Startups.csv')
# Visualizing 1st 5 data

df.head()
# Checking for null values

df.isnull().sum()
# Checking for data types

df.info()
from sklearn.preprocessing import LabelEncoder
```

```
# Initializing encoder

le = LabelEncoder()
# Transforming string values to int

df['State'] = le.fit_transform(df['State'])
df.head()
# Split the data (independent and dependent variables)

x = df.iloc[:,0:4].values
y = df.iloc[:,4:].values
from sklearn.model_selection import train_test_split
# Split training and testing data

xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=0)
# Checking shape of data

xtrain.shape,xtest.shape
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Creating ANN skleton

reg = Sequential()
reg.add(Dense(4,activation='relu'))
reg.add(Dense(12,activation='relu'))
reg.add(Dense(8,activation='relu'))
reg.add(Dense(9,activation='relu'))
reg.add(Dense(1,activation='linear'))
# Computation

reg.compile(optimizer='adam',loss='mse',metrics=['mse'])
# Training the machine with training data

reg.fit(xtrain,ytrain,batch_size=10,epochs=300)
# Predicting test data

ypred = reg.predict(xtest)
from sklearn.metrics import r2_score
```

```
# Checking the accuracy of model

r2_score(ytest,ypred)*100
# Comparing actual value with predicted value

pd.DataFrame({'Actual values':ytest.flatten(),
        'Predicted values':ypred.flatten()}).head(10)
# Reading the dataset

data = pd.read_csv('/content/Breast Cancer Wisconsin (Diagnostic) Data Set.csv')
# Visualizing 1st 5 data

data.head()
# Checking for null values

data.isnull().sum()
# Drop unwanted columns

data.drop(['Unnamed: 32','id'],axis=1,inplace=True)
# Visualizing 1st 5 data

data.head()
# Checking for data types

data.info()
# Split the data (independent and dependent variables)

x = data.drop('diagnosis',axis=1)
y = data['diagnosis']
# Transforming string to int

le2 = LabelEncoder()

y=le2.fit_transform(y)
# Split training and testing data

xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.2,random_state=0)
xtrain.shape
# Creating ANN skleton
```

```
classification = Sequential()
classification.add(Dense(30,activation='relu'))
classification.add(Dense(128,activation='relu'))
classification.add(Dense(64,activation='relu'))
classification.add(Dense(32,activation='relu'))
classification.add(Dense(1,activation='sigmoid'))
classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
# Training the model

classification.fit(xtrain,ytrain,batch_size=10,epochs=300)
# Predicting test data

ypred = classification.predict(xtest)
# Compare actual data with predicted data

pd.DataFrame({'Actual value':ytest.flatten(),
        'Predicted value':np.round(ypred.flatten()).astype('int')}).head(20)
```