```
# -*- coding: utf-8 -*-
"""2023_MCA_M210677CA_M210705CA_ObjectDetection.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1uN6gpmIbs-vBML4b28TIH9DKiClF5zGt
"""

'''
Title -->       10 Object Recognition using ResNet50

Definition -->  In our project we have used a image sample of 50,000 in which we used  40,000 as train data and 10,000
                as test data then we used basic keras sequential model to test predict out test data,
                but we did not get accuracy much. so used Resnet-50 a cnn model which first we trained with a huge
                database of image called imagenet then we used our dataset CIFR-10 to train that pre-trained model
                to detect image eventually we got much improved accuracy.

Members         Mukesh Patel      M210705CA         mukesh_m210705ca@nitc.ac.in
                Prakash Singh     M210677CA         prakash_m210677ca@nitc.ac.in

Submition Date->27 April 2023
  '''

# installing Kaggle to download data
!pip install kaggle

# configuring the path of Kaggle.json file
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Dataset API to load dataset from kaggle
!kaggle competitions download -c cifar-10

# to check dataset that downloaded
!ls

# Extracting the dataset from downloaded  zip dataset "cifar-10.zip"
from zipfile import ZipFile

# Loading dataset containing test and train dataset in zip
dataset = "/content/cifar-10.zip"
with ZipFile(dataset, 'r') as zip:

# Extracting data from zip file
    zip.extractall()
    print("Dataset is extracted")

# Here we can see that all extracted file from zip
!ls

# Downloading py7zr extracting libraries to extract py7zr type files.
!pip install py7zr

# importing py7zr to unzip train and test dataset from 7z type file.
import py7zr

# Loading train dataset in archive_train
archive = py7zr.SevenZipFile("/content/train.7z", mode = 'r')
archive.extractall()    # Extracting all file from training zip file.
archive.close()

# we can now have extracted train and test dataset file
!ls

# importing some libraries

#  os is used for creating and removing a directory (folder),
# fetching its contents, changing and identifying the current directory
import os

# numpy adds support for large, multi-dimensional arrays and matrices
import numpy as np

# pandas used for working with data sets
import pandas as pd

# PIL stands for Python Image Library used to deal with image
from PIL import Image

# Matplotlib for creating static, animated, and interactive visualizations in Python
import matplotlib.pyplot as plt
# mpimg deal with image
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
```

```
# Loading test dataset

# loading train data as list using os module
filenames = os.listdir("/content/train")

# To confirm it is in list form
print(type(filenames))

# Checking filenames length
print(len(filenames))

# To check  are not in order
# Printing top 5 rows of dataset
print(filenames[0:5])

# Printing last 5 rows of dataset
print(filenames[-5:])

"""**Label Processing**"""

#  labels_df cointain target class of training image dataset
labels_df = pd.read_csv("/content/trainLabels.csv")

# Checking the shape of labels_df
print(labels_df.shape)

# Printing first 5 data of labels_df to check data.
print(labels_df.head())

# just checking train image data and labels data are matching
print(labels_df[labels_df['id']==44493])

# labels_df contain data in order
# Printing first 10 rows of labels_df
print(labels_df.head(10))

# printing last 10 rows of labels_df
print(labels_df.tail(10))

# Printing the no. of intance of each object
print(labels_df['label'].value_counts())

labels_dictionary = {'airplane':0, 'automobile':1, 'bird':2, 'cat':3, 'deer':4, 'dog':5, 'frog':6, 'horse':7, 'ship':8, 'truck':9}

labels = [labels_dictionary[i] for i in labels_df['label']]

# Printing after encoding
print(labels[0:5])
print(labels[-5:])

# displaying sample image frog
import cv2
from google.colab.patches import cv2_imshow
# After given path of file train we need to put the name of image which we want to show
img = cv2.imread('/content/train/7796.png')
# Here we are showing the image of frog.
print(labels_df[labels_df['id']== 7796])
# A fn two show image
cv2_imshow(img)

# displaying sample image of dog
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('/content/train/45888.png')
print(labels_df[labels_df['id']== 45888])
cv2_imshow(img)

# creating a list contains id of each intances
id_list = list(labels_df['id'])
print("Lnegth of id_list_train",len(id_list))
print(id_list[0:5])
print(id_list[-5:])

"""**Image Processing**"""

# Image processing
# Loading the train dataset
data_folder = '/content/train/'

# to convert images to numpy arrays
data = []

for id in id_list:
    # eg. image = /content/train/001.png or /content/train/002.png, etc
```

```
    image = Image.open(data_folder + str(id) + '.png')

    #converting image to numpy array type
    image = np.array(image)
    # And then appending to list data
    data.append(image)

# print data type of variable data
print(type(data))
# Printing length of data
print("length of train_data list ", len(data))

# Printing type of element store in list data
print(type(data[0]))
# Printing shape of each element ie. image type
print("Size of each image store as ndarray in data",data[0].shape)
# i.e. (32, 32, 3) here 32 is width and length in pixel and 3 denotes color RGB

# Printing the image in pixel matrix form
print(data[0])

# convert image list and label list to numpy arrays

X = np.array(data) # converting list of image into numpy array
Y = np.array(labels) # convering our labels list into numpy array

"""**Train Test Split**"""

# Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)

# printing type and shape of X that contains our image
print("Type of x", type(X))
print("Shape of X ", X.shape)

# Printing type and shape of y
print("Type of Y", type(Y_train))
print("Shape of Y ", Y_train.shape)


print("Type of x_test", type(X_test))
print("Shape of X_test ", X_test.shape)

# Printing type and shape of y
print("Type of Y_test", type(Y_test))
print("Shape of Y_test ", Y_test.shape)

# scaling the data
X_train_scaled = X_train/255
X_test_scaled = X_test/255
# now the pixels will be in range 0-1 istead of 0-256

# printing X_train and X_test
print(X_train_scaled)

"""**Building Neural Network**"""

# importing libraries
import tensorflow as tf
# Keras is also used for distributed training of deep learning models
from tensorflow import keras

num_of_classes = 10

# setting up the layers of Neural Network
# A Sequential model is appropriate for a plain stack of
# layers where each layer has exactly one input tensor and one output tensor

# provides inbuilt modules for all neural network computations
model = keras.Sequential([

    # keras.flatten converts the multi-dimensional arrays into
    # flattened one-dimensional arrays or single-dimensional arrays
    keras.layers.Flatten(input_shape=(32,32,3)),  # Input Layer

# Keras Dense layer is the layer that contains all the neurons that are deeply connected within themselves
# Relu remove every negative value from the filtered image and replace it with zero.
    keras.layers.Dense(128, activation='relu'),   # Hidden Layer

# The softmax function is used as the activation function in the output layer of
# neural network models that predict a multinomial probability distribution.
    keras.layers.Dense(num_of_classes, activation='softmax')    # Output Layer
])

# compile the neural network
```

```
# compilation is a step that transforms the simple sequence of layers
# that we previously defined into a highly efficient series of matrix transformations

# Optimizer it helps in reducing the overall loss and improving accuracy.

# Adam optimization is a stochastic gradient descent method that is based on
# adaptive estimation of first-order and second-order moments.

# sparse_categarical_cross. when each sample belongs exactly to one
# categorical crossentropy when one sample can have multiple classes or labels are

# soft probabilities (like [0.5, 0.3, 0.2]).
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])

# training the neural network

# validation_split is Fraction of the training data to be used as validation data
# Epoch is the total number of iterations of the training data in one cycle
model.fit(X_train_scaled, Y_train, validation_split=0.1, epochs=10)

"""**ResNet-50**"""

# ResNet-50
# importing libraries to use one of cnn model ResNet-50 to predict image

from tensorflow.keras import Sequential, models, layers
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50 # the ccn model we will use
from tensorflow.keras import optimizers

# parameter -1
# The ImageNet project is a large visual database designed for use in visual object recognition software research.
# More than 14 million images have been hand-annotated by the project to indicate what
# objects are pictured and in at least one million of the images, bounding boxes are also provided.

# parameter -2
# include_top = false means in imagenet we have thousand of class but need only 10, so we use own output layer
# in imagenet out class of image is also included.

#parameter -3
# input_shape the pixel of input image imagenet have image in that dimension on e.g (256,256,3)
# but our image dimension is (32,32,3).
convolutional_base = ResNet50(weights='imagenet', include_top=False, input_shape=(256,256,3))
convolutional_base.summary() # this show the different layer the model have.

# Building our model
num_of_classes = 10 # No of classes of images

model = models.Sequential() # we will stack all our layer in model

# Since our image is of length and width(32,32) our resnet-50 could only take of (256,256)
# We need to upscale out image dimension, so we need to upscale three times each time it doubles it dimension

# first upscaling it will increase the dimension of image to (64,64)
model.add(layers.UpSampling2D((2,2)))

# Second upscaling it will increase the dimension of image to (128,128)
model.add(layers.UpSampling2D((2,2)))

# Third upscaling it will increase the dimension of image to (256,256)
model.add(layers.UpSampling2D((2,2)))

# Then we need to add out upscaled image to the nodel
model.add(convolutional_base)

# Again we do flatten , which we have seen ealier
model.add(layers.Flatten())

# we normalize the values on same scale in the end of each layer
model.add(layers.BatchNormalization())

# 128 Hidden layer with activation relu
model.add(layers.Dense(128, activation='relu'))

# To handle overfittng we use dropout, it applied to the hidden layers.
# For instance, if the hidden layers have 1000 neurons (nodes) and a dropout is
# applied with drop probability = 0.5, then 500 neurons would be randomly dropped in every iteration (batch)
model.add(layers.Dropout(0.5))

# In the end of layer we normalize
model.add(layers.BatchNormalization())
```

```
# Again we use hidden layer with 64 neurons
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
# for output layer we use activation fn softmax
model.add(layers.Dense(num_of_classes, activation='softmax'))

# RMSprop optimizer is better than adam
# The gist of RMSprop is to: Maintain a moving (discounted) average of
# the square of gradients. Divide the gradient by the root of this average.

# lr is learning rate it decides how much change in weight be there in one updation.
# all other parameter we have seen ealier
model.compile(optimizer=optimizers.RMSprop(lr=2e-5), loss='sparse_categorical_crossentropy', metrics=['acc'])

history = model.fit(X_train_scaled, Y_train, validation_split=0.1, epochs=10)

# evaluating loss and accuracy on test data
loss, accuracy = model.evaluate(X_test_scaled, Y_test)
print('Test Accuracy =', accuracy)

h = history

# plot the loss value
plt.plot(h.history['loss'], label='train loss')
plt.plot(h.history['val_loss'], label='validation loss')
plt.legend()
plt.show()

# plot the accuracy value
plt.plot(h.history['acc'], label='train accuracy')
plt.plot(h.history['val_acc'], label='validation accuracy')
plt.legend()
plt.show()
```