**FRUITS CLASSIFICATION USING DEEP LEARNING TECHINIQUES**

**Description as a Document:**

In general lot of fruits are there in shops, they are similar in color to other fruits, if we ask kids to take red fruit they have many options to take like apple, cherry, pomegranate, tomato, etc. Here I mentioned a kid that is robot, if we use robot in shops to help to customer, the robot does not recognize the fruit by name, it will identify only by color, height, width, hue and saturation. This application used in various places to help people to identify the fruit species correctly and to know the price exactly. So we should train the robot how to identify the fruit. Here we proposed fruits detection based on Deep Learning concepts and Convolution Neural Networks (CNN) . This proposed model used Adadelta model. Used color images datasets for training and testing, training for evaluating and testing datasets for challenging to find the correct one. I collected the datasets from GitHub. 10 classes used, almost 5000 images used for training and 2000 images used for testing. In this process I got classification accuracy was 90% and 70% achieved.

**Novelty:**

Usually,

1) The classifier may not be robust because dissimilar fruit images may have similar or identical shape and color features.

2) Some recognition systems are not appropriate to classify all fruit species; they can simply classify the diversities of the same class. Other systems may only be used to detect infections or the level of maturity in the same fruit species.

3) The recognition system may require extra sensors, such as gas, invisible light, or weight sensors.

To overcome these shortcomings, this paper presents an efficient deep learning-based framework for visual fruit classification. More specifically, our framework is based on the convolutional neural networks (CNNs), which are multilayered feedforward neural networks that are able to learn task-specific invariant features in a hierarchical manner .

In the previous project they created Data Augmentation for only training process, so the system got will trained model to give good accuracy, but here I did Data Augmentation process for testing data, so this makes somewhat complexity for trained data to give better accuracy. During testing, for 10 classes I created 1000 images.

**Experiment Details:**

In this model used Adadelata model with 4 convolution layer, the convolution layer 1 size is (100,100,16), the convolution layer 2 size is(50,50,32), the convolution layer 3 size is (25,25,64) and the convolution layer 4 size is (12,12,128) and 4 pooling layer used, the first pooling layer using window size is (50,50,16), the second pooling layer using window size is (25,25,32), the third pooling layer using window size is (12,12,64), and the final pooling layer using window size is (6,6,128), and 2 dropout functions used the size of dropout layers are 1024 and 128, finally Dense Layer used with 10 output functions. I got validation accuracy is 90%, the testing accuracy is 70%.

**CODINGS**

**PREPROCESSING**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sn

import os

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

from keras.optimizers import Adadelta

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint

learning\_rate = 0.1

min\_learning\_rate = 0.00001  learning\_rate\_reduction\_factor = 0.5

patience = 3

verbose = 1  image\_size = (100, 100)   input\_shape = (100, 100, 3

use\_label\_file =  label\_file = 'labels.txt'

base\_dir = 'drive/My Drive/FruitsDataSet'  test\_dir = os.path.join(base\_dir, 'Test')

train\_dir = os.path.join(base\_dir, 'Training')

output\_dir = 'output\_files'

if not os.path.exists(output\_dir):

    os.makedirs(output\_dir)

if use\_label\_file:

    with open(label\_file, "r") as f:

        labels = [x.strip() for x in f.readlines()]

else:

    labels = os.listdir(train\_dir)

num\_classes = len(labels)

def plot\_model\_history(model\_history, out\_path=""):

    fig, axs = plt.subplots(1, 2, figsize=(15, 5))

    axs[0].plot(range(1, len(model\_history.history['acc']) + 1), model\_history.history['acc'])

    axs[0].plot(range(1, len(model\_history.history['val\_acc']) + 1), model\_history.history['val\_acc'])

    axs[0].set\_title('Model Accuracy')

    axs[0].set\_ylabel('Accuracy')

    axs[0].set\_xlabel('Epoch')

    axs[0].set\_xticks(np.arange(1, len(model\_history.history['acc']) + 1), len(model\_history.history['acc']))

    axs[0].legend(['train', 'val'], loc='best')

    axs[1].plot(range(1, len(model\_history.history['loss']) + 1), model\_history.history['loss'])

    axs[1].plot(range(1, len(model\_history.history['val\_loss']) + 1), model\_history.history['val\_loss'])

    axs[1].set\_title('Model Loss')

    axs[1].set\_ylabel('Loss')

    axs[1].set\_xlabel('Epoch')

    axs[1].set\_xticks(np.arange(1, len(model\_history.history['loss']) + 1), len(model\_history.history['loss']))

    axs[1].legend(['train', 'val'], loc='best')

    # save the graph in a file called "acc\_loss.png" to be available for later; the model\_name is provided when creating and training a model

    if out\_path:

        plt.savefig(out\_path + "/acc\_loss.png")

    plt.show()

# create a confusion matrix to visually represent incorrectly classified images

def plot\_confusion\_matrix(y\_true, y\_pred, classes, out\_path=""):

    cm = confusion\_matrix(y\_true, y\_pred)

    df\_cm = pd.DataFrame(cm, index=[i for i in classes], columns=[i for i in classes])

    plt.figure(figsize=(40, 40))

    ax = sn.heatmap(df\_cm, annot=True, square=True, fmt="d", linewidths=.2, cbar\_kws={"shrink": 0.8})

    if out\_path:

        plt.savefig(out\_path + "/confusion\_matrix.png")

    return ax

def build\_data\_generators(train\_folder, test\_folder, validation\_percent, labels=None, image\_size=(100, 100), batch\_size=50):

    train\_datagen = ImageDataGenerator(

        width\_shift\_range=0.0,

        height\_shift\_range=0.0,

        zoom\_range=0.0,

        horizontal\_flip=True,

        vertical\_flip=True,

        validation\_split=validation\_percent)

    test\_datagen = ImageDataGenerator()

    train\_gen = train\_datagen.flow\_from\_directory(train\_folder, target\_size=image\_size, class\_mode='sparse', batch\_size=batch\_size, shuffle=True, subset='training', classes=labels)

    validation\_gen = train\_datagen.flow\_from\_directory(train\_folder, target\_size=image\_size, class\_mode='sparse', batch\_size=batch\_size, shuffle=False, subset='validation', classes=labels)

    test\_gen = test\_datagen.flow\_from\_directory(test\_folder, target\_size=image\_size, class\_mode='sparse', batch\_size=batch\_size, shuffle=False, subset=None, classes=labels)

    return train\_gen, validation\_gen, test\_gen

def train\_and\_evaluate\_model(model, name="", epochs=10, batch\_size=100, verbose=verbose, useCkpt=False):

    print(model.summary())

    model\_out\_dir = os.path.join(output\_dir, name)

    if not os.path.exists(model\_out\_dir):

        os.makedirs(model\_out\_dir)

    if useCkpt:

        model.load\_weights(model\_out\_dir + "/model.h5")

    trainGen, validationGen, testGen = build\_data\_generators(train\_dir, test\_dir, validation\_percent=0.1, labels=labels, image\_size=image\_size, batch\_size=batch\_size)

    optimizer = Adadelta(lr=learning\_rate)

    model.compile(optimizer=optimizer, loss="sparse\_categorical\_crossentropy", metrics=["accuracy"])

    learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_loss', patience=patience, verbose=verbose, factor=learning\_rate\_reduction\_factor, min\_lr=min\_learning\_rate)

    save\_model = ModelCheckpoint(filepath=model\_out\_dir + "/model.h5", monitor='val\_acc', verbose=verbose,  save\_best\_only=True, save\_weights\_only=False, mode='max', period=1)

    history = model.fit\_generator(generator=trainGen,epochs=epochs,steps\_per\_epoch=(trainGen.n // batch\_size) + 1,validation\_data=validationGen, validationGen.n // batch\_size) + 1,

                                  verbose=verbose,

                                  callbacks=[learning\_rate\_reduction, save\_model])

    model.load\_weights(model\_out\_dir + "/model.h5")

    validationGen.reset()

    loss\_v, accuracy\_v = model.evaluate\_generator(validationGen, steps=(validationGen.n // batch\_size) + 1, verbose=verbose)

    loss, accuracy = model.evaluate\_generator(testGen, steps=(testGen.n // batch\_size) + 1, verbose=verbose)

    print("Validation: accuracy = %f  ;  loss\_v = %f" % (accuracy\_v, loss\_v))

    print("Test: accuracy = %f  ;  loss\_v = %f" % (accuracy, loss))

    plot\_model\_history(history, out\_path=model\_out\_dir)

    testGen.reset()

    y\_pred = model.predict\_generator(testGen, steps=(testGen.n // batch\_size) + 1, verbose=verbose)

    y\_true = testGen.classes[testGen.index\_array]

    plot\_confusion\_matrix(y\_true, y\_pred.argmax(axis=-1), labels, out\_path=model\_out\_dir)

    class\_report = classification\_report(y\_true, y\_pred.argmax(axis=-1), target\_names=labels)

    with open(model\_out\_dir + "/classification\_report.txt", "w") as text\_file:

        text\_file.write("%s" % class\_report)

    # print(class\_report)

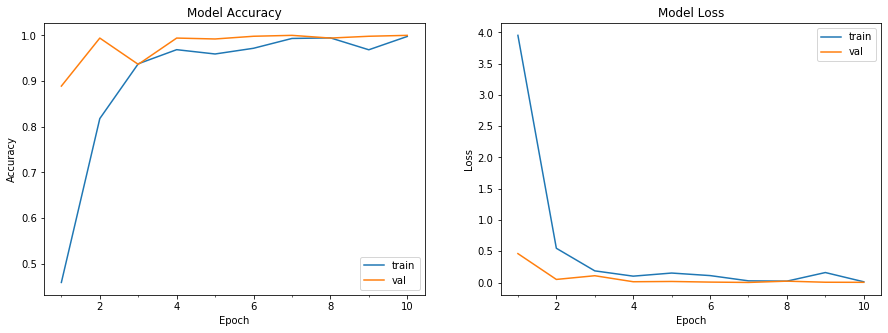
print(labels)

print(num\_classes)

TO RUN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| from keras.models import Model  from keras.layers import Input, Dense, Conv2D, MaxPooling2D, Flatten, Activation, Dropout, Lambda  # Create a custom layer that converts the original image from  # RGB to HSV and grayscale and concatenates the results  # forming in input of size 100 x 100 x 4  def image\_process(x):      import tensorflow as tf      hsv = tf.image.rgb\_to\_hsv(x)      gray = tf.image.rgb\_to\_grayscale(x)      rez = tf.concat([hsv, gray], axis=-1)      return rez  def network(input\_shape, num\_classes):      img\_input = Input(shape=input\_shape, name='data')      x = Lambda(image\_process)(img\_input)      x = Conv2D(16, (5, 5), strides=(1, 1), padding='same', name='conv1')(x)      x = Activation('relu', name='conv1\_relu')(x)      x = MaxPooling2D((2, 2), strides=(2, 2), padding='valid', name='pool1')(x)      x = Conv2D(32, (5, 5), strides=(1, 1), padding='same', name='conv2')(x)      x = Activation('relu', name='conv2\_relu')(x)      x = MaxPooling2D((2, 2), strides=(2, 2), padding='valid', name='pool2')(x)      x = Conv2D(64, (5, 5), strides=(1, 1), padding='same', name='conv3')(x)      x = Activation('relu', name='conv3\_relu')(x)      x = MaxPooling2D((2, 2), strides=(2, 2), padding='valid', name='pool3')(x)      x = Conv2D(128, (5, 5), strides=(1, 1), padding='same', name='conv4')(x)      x = Activation('relu', name='conv4\_relu')(x)      x = MaxPooling2D((2, 2), strides=(2, 2), padding='valid', name='pool4')(x)      x = Flatten()(x)      x = Dense(1024, activation='relu', name='fcl1')(x)      x = Dropout(0.2)(x)      x = Dense(128, activation='relu', name='fcl2')(x)      x = Dropout(0.2)(x)      out = Dense(num\_classes, activation='softmax', name='predictions')(x)      rez = Model(inputs=img\_input, outputs=out)      return rez  model = network(input\_shape=input\_shape, num\_classes=num\_classes)  train\_and\_evaluate\_model(model, name="fruit-360 model")   |  |  | | --- | --- | |  |  | |  |  |

**SCREEN SHOTS**



**CONFUSION MATRIX**

