|  |  |
| --- | --- |
|  | **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập - Tự do - Hạnh phúc** |
|  | *………ngày.....tháng.....năm......* |
|  |  |

**BÁO CÁO KẾT QUẢ THUẬT TOÁN VỚI DỮ LIỆU PHÂN LOẠI ẢNH**

**DỰA VÀO MÔ HÌNH NEURON NETWORK VỚI KNN**

Họ và tên: Nguyễn Phát

Nguyễn Thanh Quang

1. **Ý tưởng tiến hành.**

**- Đầu tiên là quyết định dữ liệu đầu vào và tìm cách chuẩn hóa hình ảnh**

**- Thu thập dữ liệu và tiến hành xây dựng mô hình**

**- Tìm cách lưu tiến trình chạy (vì trong quá trình chạy thì thời gian chuẩn hóa hình ảnh quá lâu)**

**-Tiếp tục thử và bổ sung thêm dữ liệu để xem xét và so sánh**

**-Viết báo cáo**

1. **Dữ liệu đầu vào**

* **Dữ liệu đầu vào gồm 577 ảnh quả chuối chín không hư và 530 ảnh chuối quá chín có bị hư, các hình ảnh nhiễu.**

1. **Code và kết quả chạy.**

#Khai báo các thư viện

import cv2 as cv

import numpy as np

import glob

from matplotlib import pyplot as plt

import pickle

label\_1 = glob.glob('/content/drive/MyDrive/save\_file/fresh\_banana/\*.png')

label\_2 = glob.glob('/content/drive/MyDrive/save\_file/rotten\_banana/\*.png')

#Chuẩn hóa ảnh

def labeling(name\_list,label, X, y):

    for i in name\_list:

        y.append(label)

        img=cv.imread(i,0)

        img=cv.resize(img, (50, 50))

        X.append(img.ravel())

X=[]

y=[]

labeling(label\_1,1,X,y)

labeling(label\_2,0,X,y)

X = np.vstack(X).astype(None)

y = np.array([y]).T

# Save biến vào file để lần sau chạy

with open('/content/drive/MyDrive/save\_file/config/label1.pkl', 'wb') as f:

    pickle.dump(X, f)

with open('/content/drive/MyDrive/save\_file/config/label2.pkl', 'wb') as f:

    pickle.dump(y, f)

# Load biến từ file để chạy chương trình

with open('/content/drive/MyDrive/save\_file/config/label1.pkl', 'rb') as f:

    X = pickle.load(f)

with open('/content/drive/MyDrive/save\_file/config/label2.pkl', 'rb') as f:

    y = pickle.load(f)

#Chia dữ liệu đầu vào

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_max = X\_train.max()

X\_min = X\_train.min()

X\_train\_scaled = np.array([(x - X\_min) / (X\_max - X\_min) for x in X\_train])

X\_test\_scaled = np.array([(x - X\_min) / (X\_max - X\_min) for x in X\_test])

#Xây dựng các lớp neuron và train model

class Sigmoidlayer:

  def \_\_init\_\_(self, n\_inp, n\_op, bias = True):

    if bias: n\_inp += 1

    w = np.random.randn(n\_inp, n\_op)

    self.w = w

    self.bias = bias

    self.is\_forward = False

  def sigmoid(self, z):

        return 1 / (1 + np.exp(-z))

  def forward(self, X):

    if self.bias:

      X = np.hstack((np.random.rand(X.shape[0], 1), X))

    z = np.dot(X, self.w)

    a = self.sigmoid(z)

    self.a = a

    self.x = X

    self.is\_forward = True

    return a

  def back\_probagation(self, delta):

    a = self.a

    dw = np.dot(self.x.T, delta \* a \* (1.0 - a))

    if self.bias:

      da = np.dot(delta \* a \* (1.0 - a), self.w.T)[:, 1:]

    else:

      da = np.dot(delta \* a \* (1.0 - a), self.w.T)

    return dw, da

def batch\_generator(X, y, batch\_size):

  idx = np.array([x for x in range(X.shape[0])])

  np.random.shuffle(idx)

  for i in range(int(X.shape[0] / batch\_size)):

    temp = idx[batch\_size\*i:min(batch\_size\*(i+1), len(idx))]

    yield X[temp, :], y[temp, :]

def predict(X, model):

  for l in model:

    X = l.forward(X)

  return X

def train\_step(X, y, model, lr, gamma):

  y\_pred = predict(X, model)

  delta = y\_pred - y

  for l in reversed(model):

    dw, delta = l.back\_probagation(delta)

    l.w -= lr \* dw + gamma \* l.w

def loss(y, y\_pred):

  l = y\*np.log(y\_pred) + (1-y)\*np.log(1-y\_pred)

  return -np.mean(l)

def train(X, y, model, lr, gamma, n\_epochs):

  for i in range(n\_epochs):

    for Xb, yb in batch\_generator(X, y, 32):

      train\_step(Xb, yb, model, lr, 0)

    print(f"epoch {i}")

    y\_pred\_train = predict(X\_train\_scaled, model)

    y\_pred\_test = predict(X\_test\_scaled, model)

    print(f"loss train: {loss(y\_train, y\_pred\_train)}, loss test: {loss(y\_test, y\_pred\_test)}")

model = [Sigmoidlayer(2500, 64, bias = True),

         Sigmoidlayer(64, 64),

         Sigmoidlayer(64, 32),

         Sigmoidlayer(32,1)]

train(X\_train\_scaled, y\_train, model, 0.1, 0.9, 100)

y\_pred = predict(X\_test\_scaled, model)

y\_pred[y\_pred >= 0.5] = 1

y\_pred[y\_pred < 0.5] = 0

#Đánh giá kết quả

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

**Đây là kết quả chạy của NN sử dụng thuật toán Sigmoid làm Layer**

precision recall f1-score support

0 0.95 0.91 0.93 97

1 0.93 0.96 0.94 125

accuracy 0.94 222

macro avg 0.94 0.93 0.94 222

weighted avg 0.94 0.94 0.94 222

**Cũng cùng dữ liệu trên và đây là của KNN:**

#KNN

def computeDistance(item, img):

  result = 0

  for i in range(len(item)):

    result += (item[i] - img[i])\*\*2

  return np.sqrt(result)

def computeKNN(imgs\_train, labels\_train, item, k):

  distance = []

  for img, label in zip(imgs\_train, labels\_train):

    distance.append(

        {

            "label": label,

            "value": computeDistance(item, img)

        }

    )

  distance.sort(key= lambda x:x["value"])

  label = [item['label'] for item in distance]

  return label[:k]

def calculate\_mode(arr: np.ndarray):

    (values, counts) = np.unique(arr, return\_counts=True)

    index = np.argmax(counts)

    return values[index]

def voteTheDistance(imgs\_train, labels\_train, item, k):

  knn = np.asarray(computeKNN(imgs\_train, labels\_train, item, k))

  label = calculate\_mode(knn)

  return label

y\_pred\_knn = []

for i in X\_test:

  y\_pred\_knn.append(voteTheDistance(X\_train, y\_train, i, 5))

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred\_knn))

**Kết quả của KNN**

precision recall f1-score support

0 1.00 0.72 0.84 97

1 0.82 1.00 0.90 125

accuracy 0.88 222

macro avg 0.91 0.86 0.87 222

weighted avg 0.90 0.88 0.87 222

1. **Nhận xét**

**Như vậy khi chạy trên bài toán phân loại ảnh thì NN lại thể hiện ưu thế phân loại tốt hơn so với KNN. F1\_score của NN cao hơn so với KNN cũng như thời gian train của model NN nhanh hơn KNN.**

**Người báo cáo**