Бозоров Руслан Миралиевич, группа 20-1

Лабораторная работа № 4

**Цель работы**

Выбрать и решить задачу классификации изображений. Выбрать необходимое количество классов для вашей задачи. Точность модели на тестовых данных не должна быть ниже 0.75.

**Задание**

1. Решите вашу задачу, используя обычный KNN. Подберите:
   1. Оптимальное количество k-соседей (параметр n\_neighbors).
   2. Оптимальную функцию вычисления расстояния между соседями (параметр metric).
   3. Оптимальную функцию для взвешивания расстояний между соседями (или может взвешивание вообще не нужно) (параметр weights).
2. Используйте «Метод главных компонент» (Principal Component Analysis или PCA) и подберите оптимальное число главных компонент, которые дадут в итоге наибольший прирост в точности вашей модели.
3. Решить вашу задачу, используя предварительно обученную свёрточную нейронную сеть совместно с KNN. При необходимости к выходным признакам свёрточной нейронной сети можете дополнительно применить метод PCA.

**Код программы / ответ на вопрос и т.п.**

# Подключение необходимых библиотек

import numpy as np

import torch

import torchvision

from torchvision import datasets

from torchvision import transforms

from tqdm import tqdm

import torch.nn as nn

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn.decomposition import PCA

# Переключение на GPU, если это возможно

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Создание и нормализация обучающей, тестовой выборки

batch\_size = 32

transform = transforms.Compose([transforms.ToTensor(),

                                transforms.Resize((256, 256)),

                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train\_path = "C:/Users/das-s/Downloads/Plants Classification/train"

train\_ds = datasets.ImageFolder(root=train\_path, transform=transform)

train\_dataloader = DataLoader(train\_ds, batch\_size=batch\_size, shuffle=True)

test\_path = "C:/Users/das-s/Downloads/Plants Classification/test"

test\_ds = datasets.ImageFolder(root=test\_path, transform=transform)

test\_dataloader = DataLoader(test\_ds, batch\_size=batch\_size, shuffle=True)

# Просматриваем изображения

def imshow(img):

    img = img / 2 + 0.5     # unnormalize

    npimg = img.numpy()

    plt.imshow(np.transpose(npimg, (1, 2, 0)))

    plt.show()

train\_features, train\_labels = next(iter(train\_dataloader))

print(f"Feature batch shape: {train\_features.size()}")

print(f"Labels batch shape: {train\_labels.size()}")

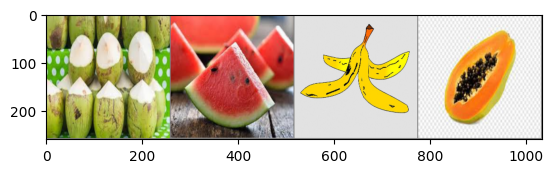
images = []

for i in range(4):

    images.append(train\_features[i])

imshow(torchvision.utils.make\_grid(images)) # show images

print(' '.join(f'{train\_ds.classes[train\_labels[j]]}\t' for j in range(4))) # print labels



coconut watermelon banana papaya

# Функция формирования набора данных (картинки и метки)

def extract\_dataset\_images(dataloader):

  labels = []

  images = []

  for batch, batch\_labels in tqdm(dataloader):

    images.append(batch.detach().numpy().reshape((batch.shape[0], -1))) # Делаем картинки в виде векторов

    labels.append(batch\_labels.detach().numpy())

  return np.concatenate(images), np.concatenate(labels)

x\_train, y\_train = extract\_dataset\_images(train\_dataloader)

x\_test, y\_test = extract\_dataset\_images(test\_dataloader)

# Функция обучения и тестирования модели KNN

def train\_and\_test\_KNN(x\_train, y\_train, x\_test, y\_test, n\_neighbors, metric, weights):

  # Обучение KNN на картинках

  knn\_classifier = KNeighborsClassifier(n\_neighbors=n\_neighbors, metric = metric, weights = weights)

  knn\_classifier.fit(x\_train, y\_train)

  # Тестирование

  # Классификация тестовых изображений с использованием KNN

  y\_pred = knn\_classifier.predict(x\_test)

  accuracy = accuracy\_score(y\_test, y\_pred)

  return accuracy

def inverse\_squared\_distance(dist):

    arr = []

    for d in dist:

        arr.append(1 / (d\*\*2))

    return arr

# Параметры KNN

metrics\_array = ["minkowski", "euclidean", "cosine"]

weights\_array = ["uniform", "distance", inverse\_squared\_distance]

n\_neighbors = [3, 5, 10, 20]

# Функция обучения и тестирования модели KNN и PCA

def train\_and\_test\_PCA\_KNN(x\_train, y\_train, x\_test, y\_test, n\_components, knn\_classifier):

  pca = PCA(n\_components=n\_components)

  x\_train\_pca = pca.fit\_transform(x\_train)

  knn\_classifier.fit(x\_train\_pca, y\_train)

  # Тестирование

  # Применение созданного ранее PCA для тестовых данных

  x\_test\_pca = pca.transform(x\_test)

  # Классификация тестовых изображений с использованием KNN

  y\_pred = knn\_classifier.predict(x\_test\_pca)

  accuracy = accuracy\_score(y\_test, y\_pred)

  return accuracy

# Параметры PCA

n\_components = [10, 50, 100, 200, 300, 500, 700]

# Скачивание предобученной модели VGG16

from torchvision.models import vgg16

model = vgg16(pretrained=True)

model.classifier = nn.Flatten(start\_dim= 1)

model.to(device)

# Использование модели VGG16 для создания обучающей и тестовой выборки

def extract\_features(dataset, model):

    features = []

    labels = []

    for batch\_images, batch\_labels in tqdm(dataset):

        batch\_features = model(batch\_images.to(device))

        labels.append(batch\_labels.detach().numpy())

        features.append(batch\_features.cpu().detach().numpy())

    return np.concatenate(features), np.concatenate(labels)

x\_train\_features, y\_train\_labels = extract\_features(train\_dataloader, model)

x\_test\_features, y\_test\_labels = extract\_features(test\_dataloader, model)

**Результаты выполнения**

1. Решите вашу задачу, используя обычный KNN

#KNN accuracy

parameters = 0, "", ""

accuracy = 0

for metric, weights, n in itertools.product(metrics\_array, weights\_array, n\_neighbors):

    temp\_accuracy = train\_and\_test\_KNN(x\_train, y\_train, x\_test, y\_test, n, metric, weights)

    if accuracy < temp\_accuracy:

        parameters = n, metric, weights

        accuracy = temp\_accuracy

print("Best KNN accuracy:")

print(f"n\_neighbors = {parameters[0]} \nmetric = {parameters[1]} \nweights = {parameters[2]} \naccuracy = {accuracy}")

Best KNN accuracy:

n\_neighbors = 3

metric = cosine

weights = <function inverse\_squared\_distance at 0x000001E7750D34C0>

accuracy = 0.6105555555555555

1. Используйте «Метод главных компонент» (Principal Component Analysis или PCA) и подберите оптимальное число главных компонент, которые дадут в итоге наибольший прирост в точности вашей модели.

фвыа

#KNN + PCA accuracy

parameters = 0, "", "", 0

accuracy = 0

for metric, weights, n\_neigh, n\_comp in itertools.product(metrics\_array, weights\_array, n\_neighbors, n\_components):

    knn\_classifier = KNeighborsClassifier(n\_neighbors=n\_neigh, metric = metric, weights = weights)

    temp\_accuracy = train\_and\_test\_PCA\_KNN(x\_train, y\_train, x\_test, y\_test, n\_comp, knn\_classifier)

    if accuracy < temp\_accuracy:

        parameters = n\_neigh, metric, weights, n\_comp

        accuracy = temp\_accuracy

print("Best KNN accuracy:")

print(f"n\_neighbors = {parameters[0]} \nmetric = {parameters[1]} \nweights = {parameters[2]} \nn\_components = {parameters[3]} \naccuracy = {accuracy}")

Best KNN accuracy:

n\_neighbors = 5

metric = cosine

weights = <function inverse\_squared\_distance at 0x000001E7750D34C0>

n\_components = 100

accuracy = 0.645

1. Решить вашу задачу, используя предварительно обученную свёрточную нейронную сеть совместно с KNN. При необходимости к выходным признакам свёрточной нейронной сети можете дополнительно применить метод PCA.

# Подбор параметров модели

#KNN + PCA + CNN accuracy

parameters = 0, "", "", 0

accuracy = 0

for metric, weights, n\_neigh, n\_comp in itertools.product(metrics\_array, weights\_array, n\_neighbors, n\_components):

    knn\_classifier = KNeighborsClassifier(n\_neighbors=n\_neigh, metric = metric, weights = weights)

    temp\_accuracy = train\_and\_test\_PCA\_KNN(x\_train\_features, y\_train\_labels, x\_test\_features, y\_test\_labels, n\_comp, knn\_classifier)

    if accuracy < temp\_accuracy:

        parameters = n\_neigh, metric, weights, n\_comp

        accuracy = temp\_accuracy

print("Best KNN accuracy:")

print(f"n\_neighbors = {parameters[0]} \nmetric = {parameters[1]} \nweights = {parameters[2]} \nn\_components = {parameters[3]} \naccuracy = {accuracy}")

Best KNN accuracy:

n\_neighbors = 5

metric = cosine

weights = <function inverse\_squared\_distance at 0x000001E7750D34C0>

n\_components = 50

accuracy = 0.9277777777777778

# Точность модели

knn\_classifier = KNeighborsClassifier(n\_neighbors=parameters[0], metric=parameters[1], weights=parameters[2])

pca = PCA(n\_components=parameters[3])

x\_train\_pca = pca.fit\_transform(x\_train\_features)

knn\_classifier.fit(x\_train\_pca, y\_train\_labels)

x\_test\_pca = pca.transform(x\_test\_features)

y\_pred = knn\_classifier.predict(x\_test\_pca)

print(classification\_report(y\_test\_labels, y\_pred, target\_names=test\_ds.classes))

precision recall f1-score support

aloevera 0.97 0.89 0.93 200

banana 0.93 0.92 0.93 200

coconut 0.89 0.85 0.87 200

cucumber 0.94 0.97 0.95 200

orange 0.95 0.85 0.90 200

papaya 0.89 0.93 0.91 200

spinach 0.88 0.98 0.93 200

sweetpotatoes 0.96 0.99 0.97 200

watermelon 0.93 0.95 0.94 200

accuracy 0.93 1800

macro avg 0.93 0.93 0.93 1800

weighted avg 0.93 0.93 0.93 1800

**Выводы**

В ходе работы был изучен принцип работы модели машинного обучения KNN. Также удалось обучить и протестировать данную модель с различными параметрами, а также совместно с другими моделями – PCA и CNN.

В итоге мы получили следующие результаты:

1. Модель KNN на данном наборе данных показала лучшие результаты со следующими параметрами:  
   n\_neighbors = 3

metric = cosine

weights = inverse\_squared\_distance

Точность модели 61%

1. Модель KNN, вместе с PCA на данном наборе данных показала лучшие результаты со следующими параметрами:

n\_neighbors = 5

metric = cosine

weights = inverse\_squared\_distance

n\_components = 100

Точность модели 64,5%

1. Модель KNN, вместе с PCA и CNN vgg16 на данном наборе данных показала лучшие результаты со следующими параметрами:  
   n\_neighbors = 5

metric = cosine

weights = inverse\_squared\_distance

n\_components = 50

Точность модели 92,8%