Ивонин Даниил Александрович, 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 и 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/Fast Food Classification V2/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/Fast Food Classification V2/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()}")

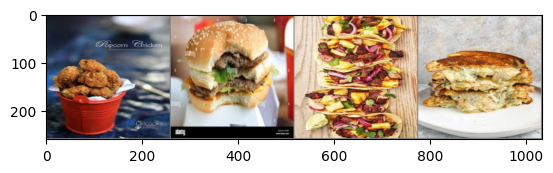
images = []

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



Crispy Chicken Burger Taco Sandwich

# 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, 30]

# 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 VGG19

from torchvision.models import vgg19

model = vgg19(pretrained=True)

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

model.to(device)

# Использование модели VGG19 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

# Поиск оптимальных параметров KNN

print("KNN accuracy comparison:")

best\_accuracy = 0

best\_n\_neighbors = 0

best\_metric = ""

best\_weights = ""

for metric in metrics\_array:

    for weights in weights\_array:

        for n in n\_neighbors:

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

            print(f"n\_neighbors = {n} \nmetric = {metric} \nweights = {weights} \naccuracy = {accuracy:.3f}")

            print('-'\*50)

            if accuracy > best\_accuracy:

                best\_accuracy = accuracy

                best\_n\_neighbors = n

                best\_metric = metric

                best\_weights = weights

print('-'\*50)

print("Best KNN accuracy:")

print(f"n\_neighbors = {best\_n\_neighbors} \nmetric = {best\_metric} \nweights = {best\_weights} \naccuracy = {best\_accuracy}")

KNN accuracy comparison:

n\_neighbors = 3

metric = minkowski

weights = uniform

accuracy = 0.266

--------------------------------------------------

n\_neighbors = 5

metric = minkowski

weights = uniform

accuracy = 0.267

--------------------------------------------------

n\_neighbors = 10

metric = minkowski

weights = uniform

accuracy = 0.269

--------------------------------------------------

...

--------------------------------------------------

Best KNN accuracy:

n\_neighbors = 20

metric = minkowski

weights = <function inverse\_squared\_distance at 0x00000190F1746340>

accuracy = 0.3514285714285714

# Точность модели

knn\_classifier = KNeighborsClassifier(n\_neighbors=best\_n\_neighbors, metric=best\_metric, weights=best\_weights)

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

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

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

precision recall f1-score support

Burger 0.58 0.47 0.52 200

Crispy Chicken 0.40 0.27 0.32 100

Donut 0.58 0.61 0.59 200

Fries 0.26 0.59 0.36 100

Hot Dog 0.45 0.14 0.22 200

Pizza 0.31 0.35 0.33 200

Sandwich 0.27 0.36 0.31 200

Taco 0.17 0.10 0.13 100

Taquito 0.07 0.08 0.08 100

accuracy 0.35 1400

macro avg 0.34 0.33 0.32 1400

weighted avg 0.38 0.35 0.34 1400

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.

# Поиск оптимальных параметров PCA

print("KNN + PCA accuracy comparison:")

best\_accuracy = 0

best\_n\_neighbors = 0

best\_metric = ""

best\_weights = ""

best\_n\_components = 0

for metric in metrics\_array:

    for weights in weights\_array:

        for n\_n in n\_neighbors:

            for n\_c in n\_components:

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

                accuracy = train\_and\_test\_PCA\_KNN(x\_train, y\_train, x\_test, y\_test, n\_c, knn\_classifier)

                print(f"n\_neighbors = {n\_n} \nmetric = {metric} \nweights = {weights}")

                print(f"n\_components = {n\_c} \naccuracy = {accuracy:.3f}")

                print('-'\*50)

                if accuracy > best\_accuracy:

                    best\_accuracy = accuracy

                    best\_n\_neighbors = n\_n

                    best\_metric = metric

                    best\_weights = weights

                    best\_n\_components = n\_c

print('-'\*50)

print("Best PCA+KNN accuracy:")

print(f"n\_neighbors = {best\_n\_neighbors} \nmetric = {best\_metric} \nweights = {best\_weights}")

print(f"n\_components = {best\_n\_components} \naccuracy = {best\_accuracy:.3f}")

KNN + PCA accuracy comparison:

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 10

accuracy = 0.286

--------------------------------------------------

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 50

accuracy = 0.313

--------------------------------------------------

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 100

accuracy = 0.314

--------------------------------------------------

...

--------------------------------------------------

Best PCA+KNN accuracy:

n\_neighbors = 30

metric = cosine

weights = <function inverse\_squared\_distance at 0x00000190F1746340>

n\_components = 50

accuracy = 0.389

# Точность модели

knn\_classifier = KNeighborsClassifier(n\_neighbors=best\_n\_neighbors, metric=best\_metric, weights=best\_weights)

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

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

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

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

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

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

precision recall f1-score support

Burger 0.38 0.49 0.43 200

Crispy Chicken 0.38 0.40 0.39 100

Donut 0.54 0.71 0.61 200

Fries 0.38 0.62 0.47 100

Hot Dog 0.38 0.21 0.27 200

Pizza 0.42 0.45 0.44 200

Sandwich 0.36 0.25 0.30 200

Taco 0.14 0.09 0.11 100

Taquito 0.14 0.13 0.14 100

accuracy 0.39 1400

macro avg 0.35 0.37 0.35 1400

weighted avg 0.37 0.39 0.37 1400

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

print("KNN + PCA + CNN accuracy comparison:")

best\_accuracy = 0

best\_n\_neighbors = 0

best\_metric = ""

best\_weights = ""

best\_n\_components = 0

for metric in metrics\_array:

    for weights in weights\_array:

        for n\_n in n\_neighbors:

            for n\_c in n\_components:

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

                accuracy = train\_and\_test\_PCA\_KNN(x\_train\_features, y\_train\_labels, x\_test\_features, y\_test\_labels, n\_c, knn\_classifier)

                print(f"n\_neighbors = {n\_n} \nmetric = {metric} \nweights = {weights}")

                print(f"n\_components = {n\_c} \naccuracy = {accuracy:.3f}")

                print('-'\*50)

                if accuracy > best\_accuracy:

                    best\_accuracy = accuracy

                    best\_n\_neighbors = n\_n

                    best\_metric = metric

                    best\_weights = weights

                    best\_n\_components = n\_c

print('-'\*50)

print("Best PCA+KNN+CNN accuracy:")

print(f"n\_neighbors = {best\_n\_neighbors} \nmetric = {best\_metric} \nweights = {best\_weights}")

print(f"n\_components = {best\_n\_components} \naccuracy = {best\_accuracy:.3f}")

KNN + PCA + CNN accuracy comparison:

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 10

accuracy = 0.646

--------------------------------------------------

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 50

accuracy = 0.701

--------------------------------------------------

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 100

accuracy = 0.724

--------------------------------------------------

...

--------------------------------------------------

Best PCA+KNN+CNN accuracy:

n\_neighbors = 10

metric = cosine

weights = <function inverse\_squared\_distance at 0x00000190F1746340>

n\_components = 200

accuracy = 0.783

# Точность модели

knn\_classifier = KNeighborsClassifier(n\_neighbors=best\_n\_neighbors, metric=best\_metric, weights=best\_weights)

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

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

Burger 0.82 0.79 0.80 200

Crispy Chicken 0.82 0.94 0.87 100

Donut 0.87 0.90 0.88 200

Fries 0.70 0.89 0.78 100

Hot Dog 0.83 0.72 0.77 200

Pizza 0.84 0.91 0.87 200

Sandwich 0.94 0.73 0.83 200

Taco 0.49 0.76 0.59 100

Taquito 0.36 0.20 0.26 100

accuracy 0.78 1400

macro avg 0.74 0.76 0.74 1400

weighted avg 0.78 0.78 0.77 1400

**Выводы**

В результате выполнения работы была решена задача классификации 3-мя различными способами:

* с помощью KNN
* с помощью KNN + PCA
* с помощью CNN + PCA + KNN

Были протестированы модели с различными параметрами.

Для KNN лучший результат показали следующие параметры:

*n\_neighbors = 20*

*metric = minkowski*

*weights = inverse\_squared\_distance*

Точность модели 35%

Для PCA + KNN лучший результат показали следующие параметры:

n\_neighbors = 30

metric = cosine

weights = inverse\_squared\_distance

n\_components = 50

Точность модели 39%

Для CNN + PCA + KNN лучший результат показали следующие параметры:

n\_neighbors = 10

metric = cosine

weights = inverse\_squared\_distance

n\_components = 200

Точность модели 78%