Хаустов Игорь Сергеевич, группа 20-1

Лабораторная работа № 4

**Цель работы**

Выбрать и решить задачу классификации изображений арифметических операций и цифр. 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 = 128

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

                                transforms.Resize((75, 75)),

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

train\_path = "final\_symbols\_split\_ttv/train"

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

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

test\_path = "final\_symbols\_split\_ttv/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))

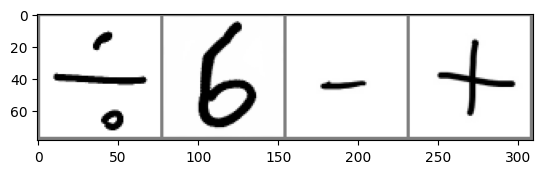
images = []

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



divide six subtract add

# 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

# Функция просчёта весов

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, 500, 700]

# Скачивание предобученной модели alexnet

from torchvision.models import alexnet

model = alexnet(pretrained=True)

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

model.to(device)

AlexNet(

(features): Sequential(

(0): Conv2d(3, 64, kernel\_size=(11, 11), stride=(4, 4), padding=(2, 2))

(1): ReLU(inplace=True)

(2): MaxPool2d(kernel\_size=3, stride=2, padding=0, dilation=1, ceil\_mode=False)

(3): Conv2d(64, 192, kernel\_size=(5, 5), stride=(1, 1), padding=(2, 2))

(4): ReLU(inplace=True)

(5): MaxPool2d(kernel\_size=3, stride=2, padding=0, dilation=1, ceil\_mode=False)

(6): Conv2d(192, 384, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(7): ReLU(inplace=True)

(8): Conv2d(384, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(9): ReLU(inplace=True)

(10): Conv2d(256, 256, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

(11): ReLU(inplace=True)

(12): MaxPool2d(kernel\_size=3, stride=2, padding=0, dilation=1, ceil\_mode=False)

)

(avgpool): AdaptiveAvgPool2d(output\_size=(6, 6))

(classifier): Flatten(start\_dim=1, end\_dim=-1)

)

# 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

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 = 0.781

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

n\_neighbors = 5

metric = minkowski

weights = uniform

accuracy = 0.765

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

n\_neighbors = 10

metric = minkowski

weights = uniform

accuracy = 0.733

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

...

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

Best KNN accuracy:

n\_neighbors = 3

metric = minkowski

weights = inverse\_squared\_distance

accuracy = 0.792755787901419

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 = 0.834

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

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 50

accuracy = 0.912

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

...

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

Best PCA+KNN accuracy:

n\_neighbors = 3

metric = cosine

weights = inverse\_squared\_distance

n\_components = 50

accuracy = 0.928

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 = 0.817

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

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 50

accuracy = 0.931

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

n\_neighbors = 3

metric = minkowski

weights = uniform

n\_components = 100

accuracy = 0.945

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

...

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

Best PCA+KNN+CNN accuracy:

n\_neighbors = 5

metric = cosine

weights = <function inverse\_squared\_distance at 0x0000015B08DBFF40>

n\_components = 200

accuracy = 0.953

# 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

add 0.97 0.98 0.98 355

divide 0.98 0.99 0.99 330

eight 0.91 0.94 0.93 348

five 0.90 0.94 0.92 354

four 0.95 0.92 0.94 399

multiply 0.97 0.99 0.98 395

nine 0.94 0.94 0.94 403

one 0.97 0.98 0.98 455

seven 0.96 0.91 0.93 407

six 0.96 0.90 0.93 373

subtract 0.99 1.00 1.00 432

three 0.93 0.93 0.93 324

two 0.95 0.94 0.95 480

zero 0.94 0.97 0.95 301

accuracy 0.95 5356

macro avg 0.95 0.95 0.95 5356

weighted avg 0.95 0.95 0.95 5356

**Выводы**

Была решена задача классификации игральных карт с помощью метода KNN. Так же было изучено влияние на точность модели использования моделей PCA и CNN совместно с KNN. В итоге модель KNN показала наилучший результат со следующими параметрами:

n\_neighbors = 3

metric = minkowski

weights = inverse\_squared\_distance

Точность модели – 79%.

Модель KNN совместно с PCA показала наилучшие результаты со следующими параметрами:

n\_neighbors = 3

metric = cosine

weights = inverse\_squared\_distance

n\_components = 50

Точность модели – 92,8%.

Используя предобученную нейронную сеть alexnet, KNN + CNN + PCA лучший результат показала со следующими параметрами:

n\_neighbors = 5

metric = cosine

weights = inverse\_squared\_distance

n\_components = 200

Точность модели – 95,3%.