**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине: «Введение в нейронные сети»

на тему: **«**Задача классификации**»**

Выполнил: студент гр. ИТП-31

Расшивалов Н.И.

Принял: преподаватель-стажёр

Дашкевич Д.А.

Гомель 2021

**Цель работы**: разработать программное обеспечение для решения задачи распознавания образов и классификации.

**Задание:**

Исходными данными является изображение, содержащее распознаваемый

объект. Необходимо выполнить их распознавание (соотнесение с одним из

заданных классов).

В качестве параметров программа должна принимать:

– файл, содержащий;

– ссылки на файлы с обучающей выборкой;

– ссылки на файлы с тестируемыми образцами;

– ссылку на файл с результатами проверки образцов (принадлежит / не

принадлежит).

Если происходит запуск без параметров, то запускается программа с графическим интерфейсом, ссылки на обучающую выборку берутся из конфигурационного *XML*-файла аналогичной структуры, как и файл параметров командной строки.

Для классификации использовать персептрон с заданным количеством скрытых слоёв и количеством нейронов на выходном слое. Для обучения использовать метод обратного распространения ошибки с указанных методом минимизации.

Распознаваемые классы: мяч для настольного тенниса, мяч для большого тенниса, футбольный мяч;

Количество слоев/нейронов: 2/3;

Метод оптимизации: *RMSProp*;

Стратегия обучения: *Mini batch*.

**Ход работы**

На рисунке 1 представлен пример изображения мяча для малого тенниса.



Рисунок 1 – Пример мяча для малого тенниса

На рисунке 2 представлен пример изображения мяча для большого тенниса.



Рисунок 2 – Пример изображения мяча для большого теннис.

На рисунке 3 представлен пример изображения футбольного мяча.



Рисунок 3 – Пример изображения футбольного мяча

На рисунке 4 представлен пример конфигурационного файла.

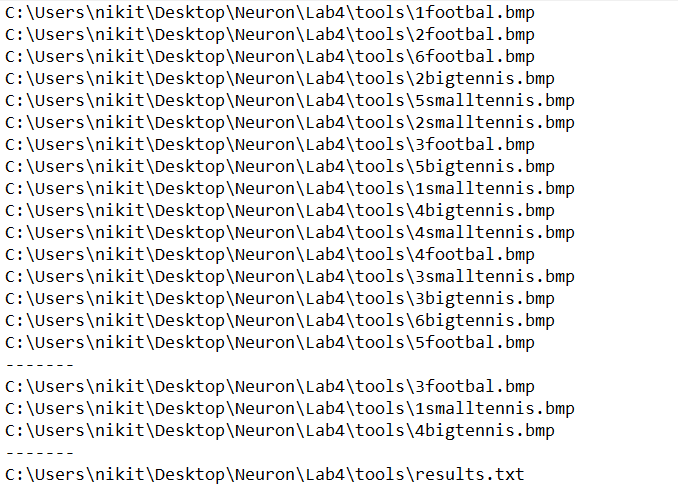


Рисунок 4 – Пример конфигурационного файла

На рисунке 5 представлен графический интерфейс программы.

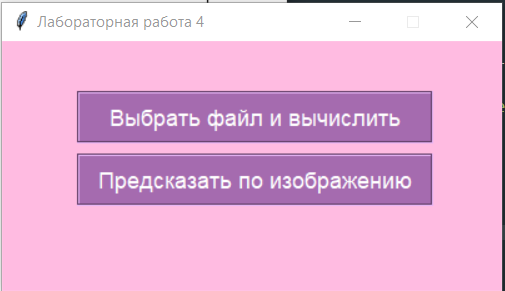


Рисунок 5 – Графический интерфейс программы

На рисунке 6 представлен график функции потерь.

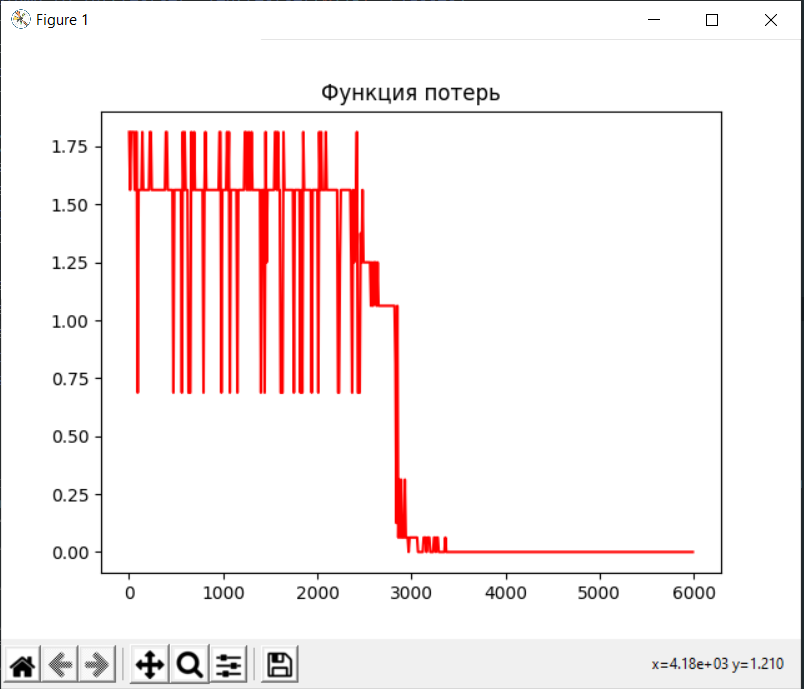


Рисунок 6 – График функции потерь

На рисунке 7 представлен результат предсказаний тестовой выборки.

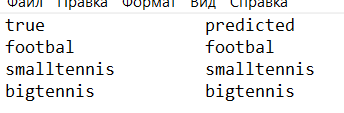


Рисунок 7 – Результат предсказаний тестовой выборки

На рисунке 8 представлен вывод предсказания класса заданного изображения.

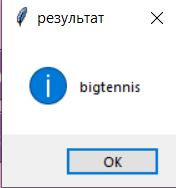


Рисунок 8 – Вывод предсказания класса заданного изображения

Наиболее простым в предсказании классом оказался мяч для настольного тенниса.

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы обучена нейронная сеть с помощью метода обратного распространения ошибки. В результате выполнения лабораторной работы получено приемлемое соотношение скорости и качества обучения нейронной сети.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

from random import seed

from random import randrange

from random import random

from random import randint

from csv import reader

from csv import writer

import csv

from types import new\_class

import numpy as np

import cv2

from math import exp, sqrt

import os

from tkinter import \*

from tkinter import messagebox

from tkinter import filedialog

import matplotlib.pyplot as plt

dataFilename = 'data.csv'

testFilename = 'test.csv'

l\_rate = 0.1

n\_epoch = 6000

n\_hidden = 3

classes = dict()

network = None

lossInfo = [[],[]]

def main():

    mainWindow = Tk()

    mainWindow.resizable(width=False, height=False)

    mainWindow.title("Лабораторная работа 4")

    screenWidth = mainWindow.winfo\_screenwidth()

    screenHeight = mainWindow.winfo\_screenheight()

    x = (screenWidth -400)/ 2

    y = (screenHeight -200)/ 2

    mainWindow.geometry('%dx%d+%d+%d' % (400, 200, x, y))

    mainWindow.configure(background="#ffbbe1")

    #Создание меню

    xCoordinate = 60

    buttonColor="#a56baf"

    buttonWidth="25"

    button1 = Button(text="Выбрать файл и вычислить", command=lambda:process(False),bg=buttonColor,fg="white",width=buttonWidth, height="1",font=("MS Sans Serif", 14),relief="groove",activebackground="#dc3f3f")

    button1.place(x=xCoordinate, y=40)

    button2 = Button(text="Предсказать по изображению", command=lambda:predictByImage(),bg=buttonColor,fg="white",width=buttonWidth, height="1",font=("MS Sans Serif", 14),relief="groove",activebackground="#dc3f3f")

    button2.place(x=xCoordinate, y=90)

    process(True)

    mainWindow.mainloop()

#Open file

def openFile(isDefault):

    global isNew

    global trainData

    global train\_y\_trues

    if isDefault==True:

        path=os.path.dirname(os.path.abspath(\_\_file\_\_))+'\\config.txt'

    else:

        path = filedialog.askopenfilename(filetypes=(("Текстовый файл", "\*.txt"),("Текстовый файл", "\*.txt")))

    if path!='':

        images = []

        lines = open(path).readlines()

        files = []

        j = 0

        for i in range(3):

            files.append([])

            while  j < len(lines) and lines[j].\_\_contains\_\_("-----") == False:

                files[i].append(lines[j].replace("\n",""))

                j += 1

            j += 1

        train\_y\_trues = []

        for i in range(len(files[0])):

            if files[0][i].\_\_contains\_\_(".bmp"):

                images.append(binarize(cv2.imread(files[0][i], cv2.IMREAD\_COLOR)))

                train\_y\_trues.append([])

                if files[0][i].\_\_contains\_\_("bigtennis"):

                    images[-1].append("bigtennis")

                if files[0][i].\_\_contains\_\_("smalltennis"):

                    images[-1].append("smalltennis")

                if files[0][i].\_\_contains\_\_("footbal"):

                    images[-1].append("footbal")

        trainData = np.array(images)

        with open("data.csv", 'w', newline='') as myfile:

            wr = writer(myfile, quoting=csv.QUOTE\_ALL)

            for row in trainData:

                wr.writerow(row)

        images = []

        for i in range(len(files[1])):

            if files[1][i].\_\_contains\_\_(".bmp"):

                images.append(binarize(cv2.imread(files[1][i],cv2.IMREAD\_COLOR)))

                if files[1][i].\_\_contains\_\_("bigtennis"):

                    images[-1].append("bigtennis")

                if files[1][i].\_\_contains\_\_("smalltennis"):

                    images[-1].append("smalltennis")

                if files[1][i].\_\_contains\_\_("footbal"):

                    images[-1].append("footbal")

        testData = np.array(images)

        with open("test.csv", 'w', newline='') as myfile:

            wr = writer(myfile, quoting=csv.QUOTE\_ALL)

            for row in testData:

                wr.writerow(row)

        isNew = True

        return files[2][0]

# Binarize image

def binarize(img):

    vectorImg = []

    for i in range(len(img)):

        for j in range(len(img[0])):

            if img[i][j][0] > 245 and img[i][j][1] > 245 and img[i][j][2] > 245:

                vectorImg.append(0)

            else:

                vectorImg.append(1)

    dataRow = []

    rowSum = 0

    for i in range(len(vectorImg)):

        rowSum += vectorImg[i]

        if i != 0 and i % 23 == 0:

            dataRow.append(rowSum/24)

            rowSum = 0

    dataRow.append(sum(vectorImg)/len(vectorImg))

    return dataRow

# Load a CSV file

def load\_csv(filename):

    dataset = list()

    with open(filename, 'r') as file:

        csv\_reader = reader(file)

        for row in csv\_reader:

            if not row:

                continue

            dataset.append(row)

    return dataset

# Convert string column to float

def str\_column\_to\_float(dataset, column):

    for row in dataset:

        row[column] = float(row[column].strip())

# Convert string column to integer

def str\_column\_to\_int(dataset, column, lookup = None):

    if lookup == None:

        class\_values = [row[column] for row in dataset]

        unique = set(class\_values)

        lookup = dict()

        for i, value in enumerate(unique):

            lookup[value] = i

    for row in dataset:

        row[column] = lookup[row[column]]

    return lookup

# Split a dataset into k folds

def cross\_validation\_split(dataset, n\_folds):

    dataset\_split = list()

    dataset\_copy = list(dataset)

    fold\_size = int(len(dataset) / n\_folds)

    for i in range(n\_folds):

        fold = list()

        while len(fold) < fold\_size:

            index = randrange(len(dataset\_copy))

            fold.append(dataset\_copy.pop(index))

        dataset\_split.append(fold)

    return dataset\_split

# Calculate accuracy percentage

def accuracy\_metric(actual, predicted):

    correct = 0

    for i in range(len(actual)):

        if actual[i] == predicted[i]:

            correct += 1

    return correct / float(len(actual)) \* 100.0

# Calculate neuron activation for an input

def activate(weights, inputs):

    activation = weights[-1]

    for i in range(len(weights)-1):

        activation += weights[i] \* inputs[i]

    return activation

# Transfer neuron activation

def transfer(activation):

    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output

def forward\_propagate(network, row):

    inputs = row

    for layer in network:

        new\_inputs = []

        for neuron in layer:

            activation = activate(neuron['weights'], inputs)

            neuron['output'] = transfer(activation)

            new\_inputs.append(neuron['output'])

        inputs = new\_inputs

    return inputs

# Calculate the derivative of an neuron output

def transfer\_derivative(output):

    return output \* (1.0 - output)

# Backpropagate error and store in neurons

def backward\_propagate\_error(network, expected):

    for i in reversed(range(len(network))):

        layer = network[i]

        errors = list()

        if i != len(network)-1:

            for j in range(len(layer)):

                error = 0.0

                for neuron in network[i + 1]:

                    error += (neuron['weights'][j] \* neuron['delta'])

                errors.append(error)

        else:

            for j in range(len(layer)):

                neuron = layer[j]

                errors.append(neuron['output'] - expected[j])

        for j in range(len(layer)):

            neuron = layer[j]

            neuron['delta'] = errors[j] \* transfer\_derivative(neuron['output'])

# Update network weights with error

def update\_weights(network, row, l\_rate, beta = 0.9):

    for i in range(len(network)):

        inputs = row[:-1]

        if i != 0:

            inputs = [neuron['output'] for neuron in network[i - 1]]

            if i != 1:

                oldInputs = inputs

                inputs = []

                for j in range(len(oldInputs)):

                    current = oldInputs[j]

                    previousInputs = [neuron['output'] for neuron in network[i - 2]]

                    sumIn = 0

                    for item in previousInputs:

                        sumIn += item \* current

                    inputs.append(sumIn)

                if i != 2:

                    oldInputs = inputs

                    inputs = []

                    for j in range(len(oldInputs)):

                        current = oldInputs[j]

                        previousInputs = [neuron['output'] for neuron in network[i - 3]]

                        sumIn = 0

                        for item in previousInputs:

                            sumIn += item \* current

                        inputs.append(sumIn)

        for neuron in network[i]:

            for j in range(len(inputs)):

                neuron['squares'][j] = beta \* neuron['squares'][j] + (1 - beta) \* neuron['delta'] \*\* 2

                neuron['weights'][j] -= l\_rate \* neuron['delta'] \* inputs[j] / (sqrt(neuron['squares'][j]) + pow(1,-10))

            neuron['squares'][-1] = beta \* neuron['squares'][-1] + (1 - beta) \* neuron['delta'] \*\* 2

            neuron['weights'][-1] -= l\_rate \* neuron['delta'] / (sqrt(neuron['squares'][-1]) + pow(1,-10))

# Train a network for a fixed number of epochs

def train\_network(network, train, l\_rate, n\_epoch, n\_outputs):

    global lossInfo

    lossInfo = [[],[]]

    for epoch in range(n\_epoch):

        miniBatch = getMiniBatchItems(len(train),int(0.2\*len(train)))

        for rowIndex in miniBatch:

            row = train[rowIndex]

            outputs = forward\_propagate(network, row)

            expected = [0 for i in range(n\_outputs)]

            expected[row[-1]] = 1

            backward\_propagate\_error(network, expected)

            update\_weights(network, row, l\_rate)

        if epoch % 10 == True:

            pred = []

            true = []

            for item in train:

                pred.append(predict(network, item))

                true.append(item[-1])

            lossInfo[0].append(epoch)

            lossInfo[1].append(mseLoss(true, pred))

    fig, ax = plt.subplots()

    ax.plot(lossInfo[0],lossInfo[1],'r')

    plt.title('Функция потерь')

    plt.show()

def mseLoss(y\_true, y\_pred):

    sum = 0.0

    for i in range(len(y\_true)):

        sum += (y\_true[i] - y\_pred[i]) \*\* 2

    return sum/len(y\_true)

# Initialize a network

def initialize\_network(n\_inputs, n\_hidden, n\_outputs):

    global network

    network = list()

    hidden\_layer = [{'weights':[random() for i in range(n\_inputs + 1)], 'squares':[0 for i in range(n\_inputs + 1)]} for i in range(n\_hidden)]

    network.append(hidden\_layer)

    second\_hidden\_layer = [{'weights':[random() for i in range(len(hidden\_layer) + 1)], 'squares': [0 for i in range(len(hidden\_layer) + 1)]} for i in range(n\_hidden)]

    network.append(second\_hidden\_layer)

    third\_hidden\_layer = [{'weights':[random() for i in range(len(second\_hidden\_layer) + 1)], 'squares': [0 for i in range(len(second\_hidden\_layer) + 1)]} for i in range(n\_hidden)]

    network.append(third\_hidden\_layer)

    output\_layer = [{'weights':[random() for i in range(len(third\_hidden\_layer) + 1)], 'squares': [0 for i in range(len(third\_hidden\_layer) + 1)]} for i in range(n\_outputs)]

    network.append(output\_layer)

    return network

# Make a prediction with a network

def predict(network, row):

    outputs = forward\_propagate(network, row)

    return outputs.index(max(outputs))

# Backpropagation Algorithm With Stochastic Gradient Descent

def back\_propagation(train,l\_rate, n\_epoch, n\_hidden):

    n\_inputs = len(train[0]) - 1

    n\_outputs = len(set([row[-1] for row in train]))

    network = initialize\_network(n\_inputs, n\_hidden, n\_outputs)

    train\_network(network, train, l\_rate, n\_epoch, n\_outputs)

    return network

#get indexes for mini batch

def getMiniBatchItems(all, count):

    items = []

    while len(items) < count:

        item = randint(0,all-1)

        if items.\_\_contains\_\_(item) == False:

            items.append(item)

    return items

# Predict by selected image

def predictByImage():

    path = filedialog.askopenfilename(filetypes=(("Изображение", "\*.bmp"),("Изображение", "\*.bmp")))

    if path != '':

        image = binarize(cv2.imread(path, cv2.IMREAD\_COLOR))

        image.append(None)

        predicted = predict(network, image)

        messagebox.showinfo(title="результат", message=list(classes.keys())[list(classes.values()).index(predicted)])

# load and prepare data

def process(isDefault):

    global classes

    resultsPath = openFile(isDefault)

    dataset = load\_csv(dataFilename)

    for i in range(len(dataset[0])-1):

        str\_column\_to\_float(dataset, i)

    classes = str\_column\_to\_int(dataset, len(dataset[0])-1)

    testset = load\_csv(testFilename)

    for i in range(len(testset[0])-1):

        str\_column\_to\_float(testset, i)

    test\_trues = []

    for testRow in testset:

        test\_trues.append(testRow[-1])

    str\_column\_to\_int(testset, len(testset[0])-1, classes)

    network = back\_propagation(dataset, l\_rate, n\_epoch, n\_hidden)

    predictions = list()

    for row in testset:

        prediction = predict(network, row)

        predictions.append(prediction)

    print(predictions)

    if resultsPath == None:

        resultsPath = "results.txt"

    f = open(resultsPath , 'w')

    resultStr = "true    predicted\n"

    for i in range(len(test\_trues)):

        resultStr += '%-20s' % test\_trues[i]

        resultStr += '%-20s' % list(classes.keys())[list(classes.values()).index(predictions[i])]

        resultStr += "\n"

    f.write(resultStr)

    f.close()

main()