МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №7

тема: «Нейрокомпьютерные сети основанные на соревновании.

Самоорганизующееся разбиение Кохонена.»

Выполнил: ст. группы МИВТ-221

Харитонов Сергей Дмитриевич

Белгород 2022 г.

**Лабораторная работа №7**

**Нейрокомпьютерные сети основанные на соревновании. Самоорганизующееся разбиение Кохонена.**

**Цель работы:** Приобретение и закрепление знаний, получение практических навыков работы с конкурирующим нейрокомпьютерными сетями, веса связей которых изменяются в ходе итерационного процесса, на примере самоорганизующегося разбиения Кохонена.

Реализация сети

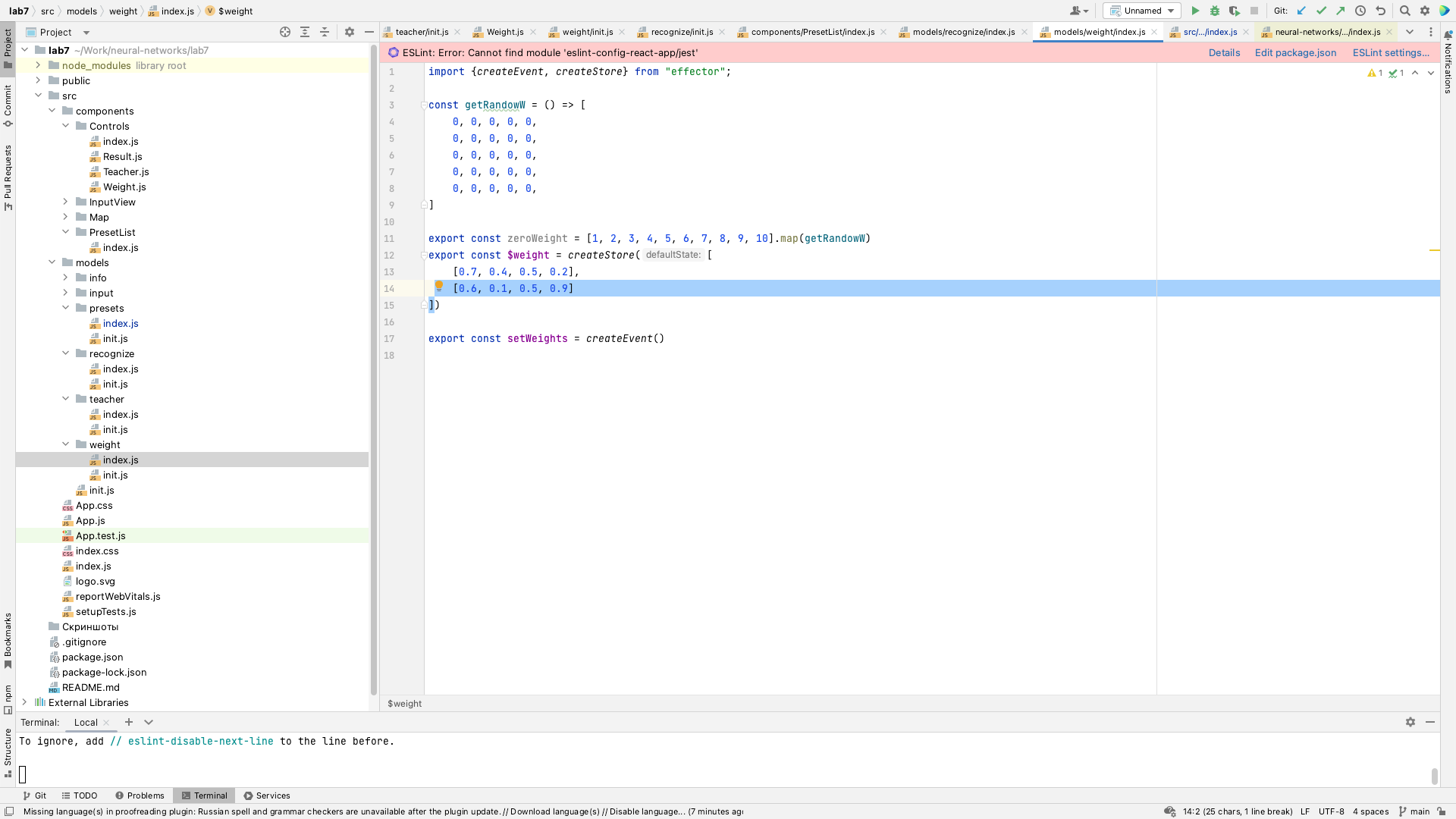
В рамках лабораторной работы была организована сеть Кохонена на основе кодовой базы предыдущих лабораторных.

Была реализована функция обучения в соответствии с алгоритмом, остановка алгоритма осуществляется в соответствии с параметром ε тогда, когда модуль изменения каждого из весов при переходе между итерациями меньше ε.

const handleTeach = (teacher, {M, weight, recognize}) => {  
  
 let prevW = weight.map(w => [...w]);  
 let nextW = prevW;  
  
 let alfa = teacher.alfa;  
 let countOperations = 0;  
  
 const checkEpsilon = (prevW, nextW) => {  
 return prevW.reduce(  
 (result, prevWr, indexR) =>  
 result || prevWr.reduce(  
 (result, prevWre, indexE) => result || (Math.abs(prevWre - nextW[indexR][indexE]) > teacher.epsilon),  
 false  
 ),  
 false  
 )  
 }  
  
 do {  
 countOperations++  
 prevW = nextW.map(w => [...w])  
  
 M.forEach(({x}) => {  
 const {minI} = nextW.reduce(  
 ({minI, ds}, w, index) => {  
 const d = w.reduce(  
 (d, wi, i) => d + Math.pow(wi - x[i], 2),  
 0  
 )  
 if (index === 0)  
 return ({  
 minI: 0,  
 ds: [d]  
 })  
 else  
 return ({  
 minI: ds[minI] < d ? minI : index,  
 ds: [...ds, d]  
 })  
 },  
 {minI: 0, ds: []}  
 )  
  
 nextW[minI] = nextW[minI].map(  
 (wOld, i) => wOld + alfa \* (x[i] - wOld)  
 )  
  
 })  
  
 alfa \*= teacher.k  
 } while (checkEpsilon(prevW, nextW))  
  
 setCountOperations(countOperations)  
 setWeights(nextW)  
 setSs(  
 M  
 .map(  
 ({x}) => recognize.recognize(x, nextW)  
 )  
 )  
}

Запуск на примере из лекции

Для проверки корректности реализованного алгоритма был осуществлен запуск на примере из лекционного материала.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как стол

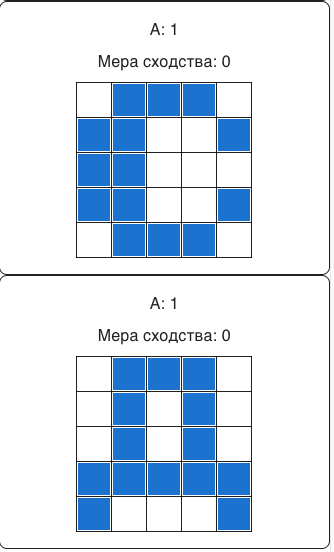
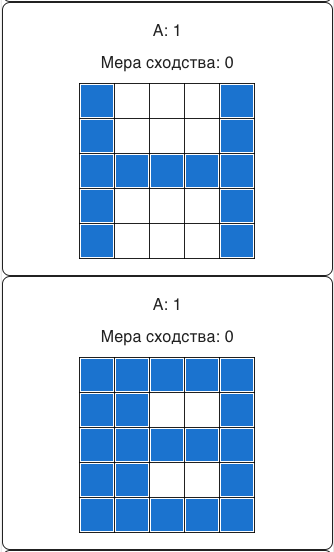
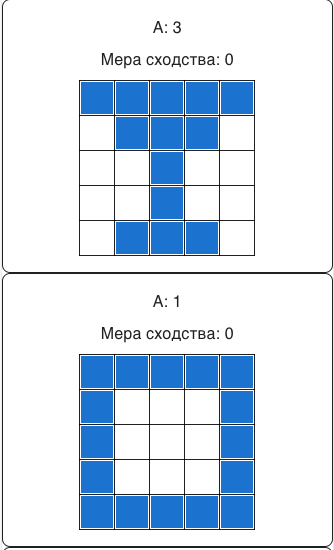
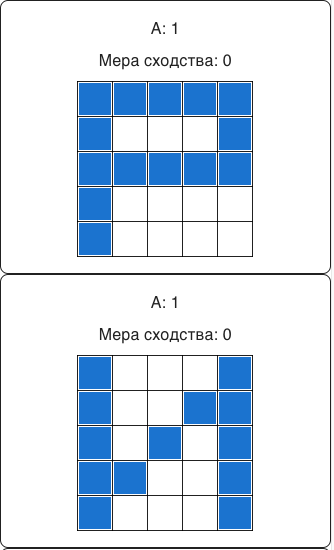
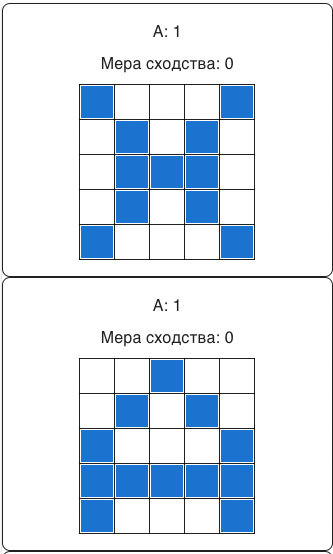
Автоматически созданное описаниеИзображение выглядит как стол

Автоматически созданное описание

При запуске на примере были получены точно такие же результаты, что позволяет заявлять о корректности реализации алгоритма.

Экранные копии изображений

Для проведения исследований были разработаны экранные формы букв: Х,А,Р,И,Т,О,Н,В,С,Д.



Меры сходства изображений

Для оценки качества кластеризации были рассчитаны меры сходства между изображениями букв.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Х | А | Р | И | Т | О | Н | В | С | Д |
| Х | 0 | 11 | 13 | 10 | 14 | 19 | 10 | 14 | 19 | 9 |
| А | 11 | 0 | 16 | 9 | 17 | 14 | 13 | 15 | 14 | 6 |
| Р | 13 | 16 | 0 | 9 | 15 | 8 | 5 | 7 | 12 | 14 |
| И | 10 | 9 | 9 | 0 | 18 | 9 | 4 | 10 | 15 | 15 |
| Т | 14 | 17 | 15 | 18 | 0 | 13 | 20 | 14 | 13 | 15 |
| О | 19 | 14 | 8 | 9 | 13 | 0 | 9 | 5 | 8 | 16 |
| Н | 10 | 13 | 5 | 4 | 20 | 9 | 0 | 8 | 15 | 15 |
| В | 14 | 15 | 7 | 10 | 14 | 5 | 8 | 0 | 7 | 13 |
| С | 19 | 14 | 12 | 15 | 13 | 8 | 15 | 7 | 0 | 12 |
| Д | 9 | 6 | 14 | 15 | 15 | 16 | 15 | 13 | 12 | 0 |

Так же была описана функция подсчитывающее количество кластеров на выходе нейронной сети и оценивающая такие параметры как:

* Размер кластера
* Максимальное различие между элементами в рамках одного кластера
* Минимальное различие между элементами в рамках одного кластера

Изображение выглядит как стол

Автоматически созданное описание

Исследования влияния параметров на разбиение

Далее было проведено исследование влияния α иk на итоговый результат кластеризации.

Исследование разбиения Кохонена при изменении изображения

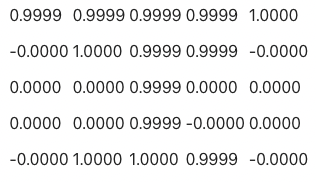
Листинг программы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **α** | к | Количество кластеров | Максимальное расстояние в рамках кластера | Минимальное расстояние в рамках кластера | Количество элементов в кластере | Количество итераций |
| 0,6 | 0,75 | 1 | 19 | 4 | 10 | 13 |
| 0,8 | 2 | 19;0 | 4;0 | 9;1 | 15 |
| 0,85 | 2 | 19;0 | 4;0 | 9;1 | 19 |
| 0,9 | 2 | 19;0 | 4;0 | 9;1 | 24 |
| 0,9 | 0,75 | 2 | 15;15 | 4;5 | 5;5 | 14 |
| 0,8 | 3 | 11;15;0 | 6;4;0 | 3;6;1 | 17 |
| 0,85 | 3 | 11;15;0 | 6;4;0 | 3;6;1 | 20 |
| 0,9 | 3 | 11;15;0 | 6;4;0 | 3;6;1 | 27 |
| 1,2 | 0,75 | 4 | 10;6;12;0 | 4;6;5;0 | 3;2;4;1 | 14 |
| 0,8 | 5 | 11;10;0;0;0 | 6;5;0;0;0 | 3;4;1;1;1 | 15 |
| 0,85 | 5 | 11;10;0;0;0 | 6;5;0;0;0 | 3;4;1;1;1 | 18 |
| 0,9 | 4 | 11;9;0;8 | 6;4;0;5 | 3;3;1;3 | 24 |
| 1,5 | 0,75 | 6 | 6;1 | 6;5 | 1;2;4;1;1;1 | 13 |
| 0,8 | 5 | 11;4;8 | 6;4;5 | 3;1;2;1;3 | 16 |
| 0,85 | 6 | 11;4;5 | 6;4;5 | 3;1;2;1;2;1 | 20 |
| 0,9 | 7 | 6;4;5 | 6;4;5 | 1;2;1;2;1;2;1 | 23 |

Исходя из результатов проведенного исследования можно сделать вывод, что при увеличении α иk, нейронная сеть делит изображения на все более мелкие кластеры, причем с увеличением количества кластеров уменьшается средняя разница между элементами, а в некоторых вырождается до нуля в связи с тем, что кластер состоит из одного элемента.

Исходя из этого можно сделать вывод о том, что чем меньше разница между элементами, тем более вероятно, что они будут определены в один и тот же кластер.

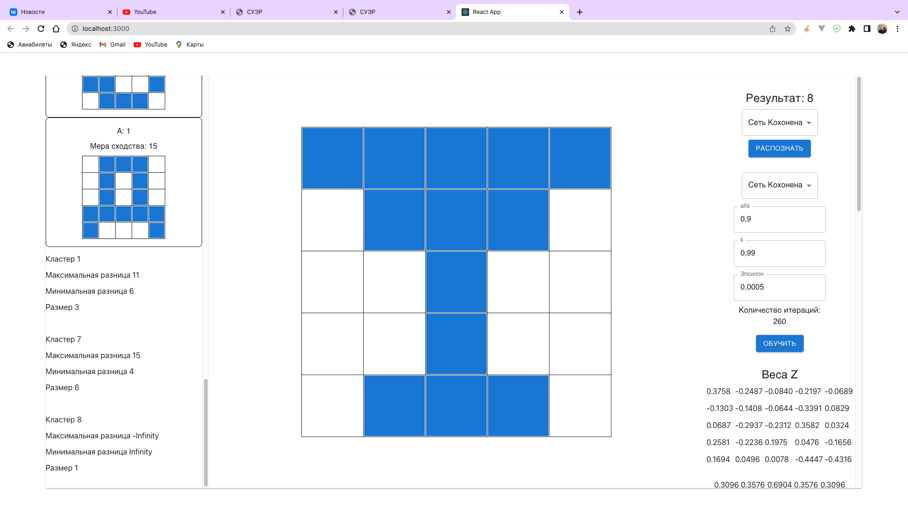
При дальнейших исследованиях, получилось разделить входные изображения максимум на 7 классов, при минимальном шаге линейной функции (k=0.99) и большим стартовым значением (α=2).



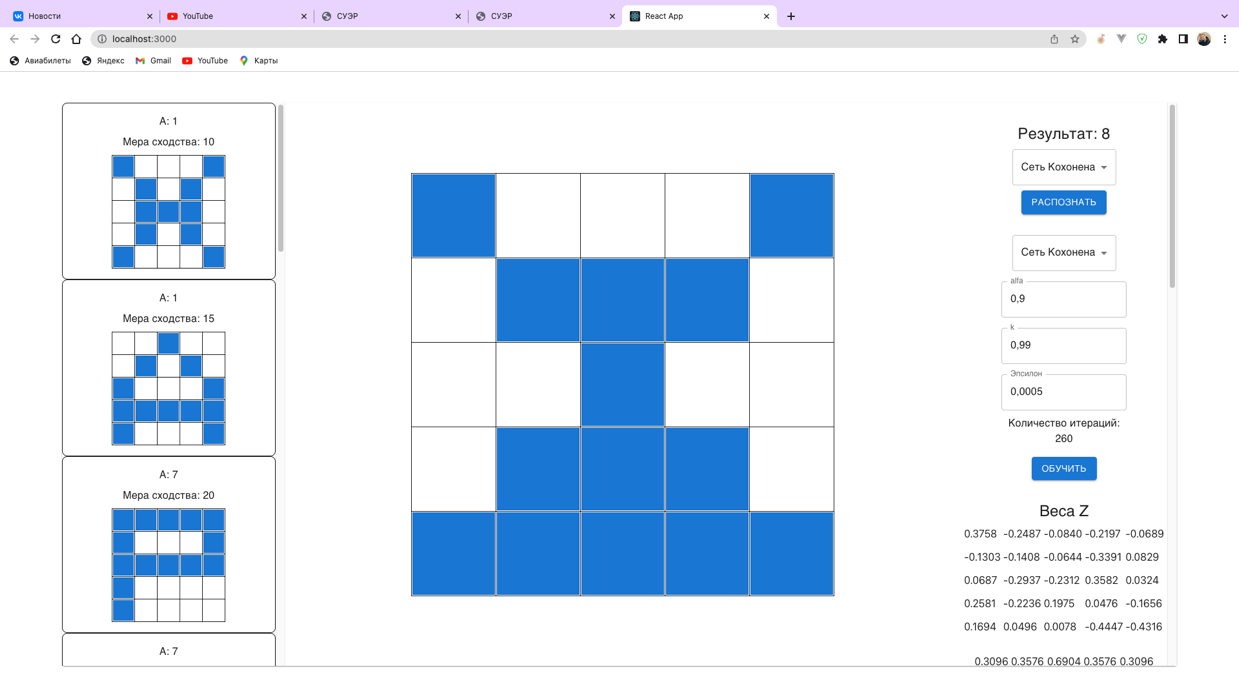
Так же в ходе обучения было замечено, что некоторые матрицы весов превращаются в матрицы входных изображений и классифицируют именно их.

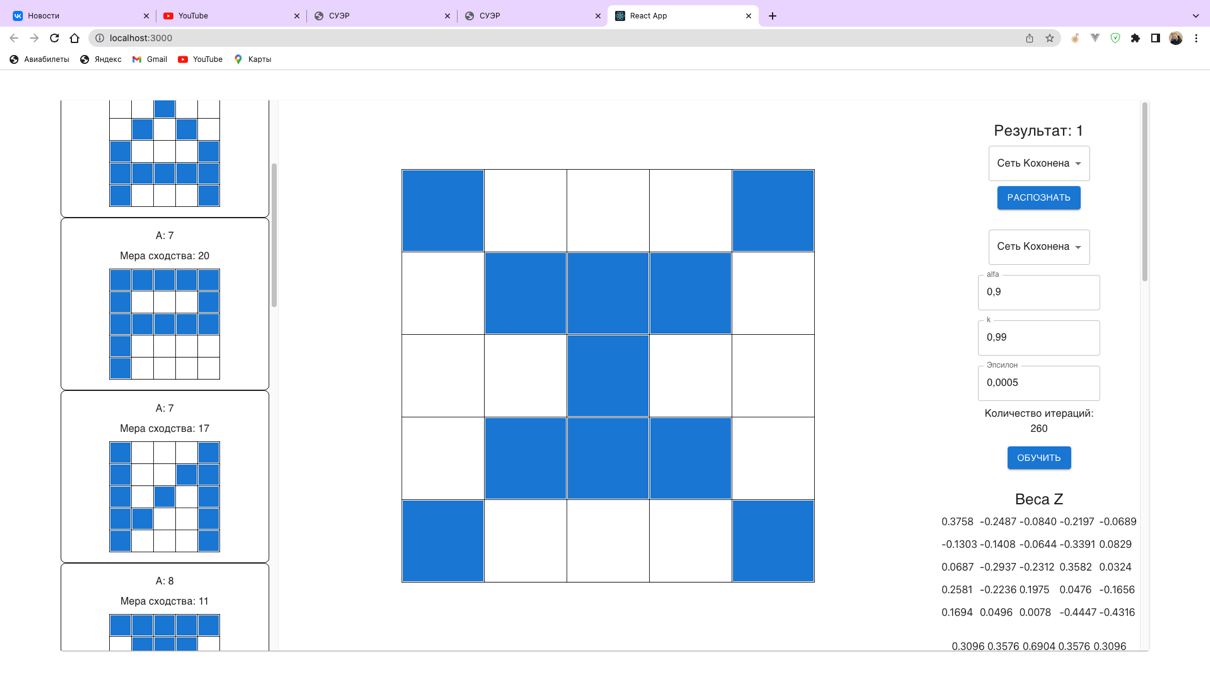
Исследование результатов определения в кластер при модификации изображения

При дальнейшем использовании удалось использовать разработанную программу для классификации образов отличающихся от набора исходных.



Полученные результаты классификации, так же объясняются мерой сходства между вводом нейронной сети и изображениями используемыми в качестве обучающей выборки, чем входное изображение ближе к изображениям определенного кластера, тем вероятнее оно попадёт в тот кластер к которому ближе.





Листинг программы

import {*createEvent*, *createStore*} from "effector";  
  
  
export const $M = *createStore*([  
 {  
 x: [  
 1, 0, 0, 0, 1,  
 0, 1, 0, 1, 0,  
 0, 1, 1, 1, 0,  
 0, 1, 0, 1, 0,  
 1, 0, 0, 0, 1,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 0, 0, 1, 0, 0,  
 0, 1, 0, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 1, 0, 0, 0, 0,  
 1, 0, 0, 0, 0,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 1, 0, 0, 0, 1,  
 1, 0, 0, 1, 1,  
 1, 0, 1, 0, 1,  
 1, 1, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 1, 1, 1, 1, 1,  
 0, 1, 1, 1, 0,  
 0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 0, 1, 1, 1, 0,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 1, 1, 1, 1, 1,  
 1, 1, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 1, 1, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 0, 1, 1, 1, 0,  
 1, 1, 0, 0, 1,  
 1, 1, 0, 0, 0,  
 1, 1, 0, 0, 1,  
 0, 1, 1, 1, 0,  
 ],  
 S: [],  
 diff: 0  
 },  
 {  
 x: [  
 0, 1, 1, 1, 0,  
 0, 1, 0, 1, 0,  
 0, 1, 0, 1, 0,  
 1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 ],  
 S: [],  
 diff: 0  
 },  
])  
  
  
export const setSs = *createEvent*()  
export const setDiffs = *createEvent*()

import {*getS*} from "../teacher";  
import {*createEvent*, *createStore*} from "effector";  
import {setCountOperations} from "../info";  
  
export function *average*(nums) {  
 return nums.reduce((a, b) => (a + b)) / nums.length;  
}  
  
export function *equalArray*(a, b){  
 return JSON.stringify(a) === JSON.stringify(b)  
}  
  
export const recognizeFunctions = [  
 {  
 recognize: (xs, ws) => ws.map(  
 weigh => *getS*(xs, weigh)  
 ),  
 id: 1,  
 name: 'Сеть Кохонена'  
 },  
  
]  
  
export const $recognize = *createStore*(recognizeFunctions[0])  
  
export const setRecognize = *createEvent*()

import {*attach*, *combine*, *createEffect*, *createEvent*, *createStore*} from "effector";  
import {$M} from '../presets'  
import {$weight} from "../weight";  
import {$recognize} from "../recognize";  
  
export const *getS* = (Xs, Ws) =>  
 Xs  
 .reduce(  
 (result, x, index) => result + x \* Ws[index],  
 0  
 )  
  
export const *getDiff* = (A, B) => A.reduce(  
 (res, a, indexB) => {  
 if (a !== B[indexB])  
 return res + 1;  
 return res;  
 },  
 0  
)  
  
export const kohTeacher = ({  
 id: 1,  
 name: 'Сеть Кохонена',  
 epsilon: 0.0005,  
 alfa: 0.6,  
 k: 0.75  
})  
  
export const teachers = [kohTeacher]  
  
export const $teacher = *createStore*(kohTeacher)  
  
export const changeTeacher = *createEvent*()  
  
export const teachEvent = *createEvent*()  
export const teachFx = *createEffect*(teachEvent)  
export const teach = *attach*({  
 effect: teachFx,  
 source: *combine*(  
 $M, $weight, $recognize,  
 (M, weight, recognize) => ({M, weight, recognize})  
 ),  
 mapParams: (\_, data) => (data)  
})

import {  
 $teacher,  
 changeTeacher, *getS*,  
 teachEvent,  
} from './index'  
import {setWeights} from "../weight";  
import {setSs} from "../presets";  
import {setCountOperations} from "../info";  
  
const handleChangeTeacher = (\_, teacher) => teacher  
const handleTeach = (teacher, {M, weight, recognize}) => {  
  
 let prevW = weight.map(w => [...w]);  
 let nextW = prevW;  
  
 let alfa = teacher.alfa;  
 let countOperations = 0;  
  
 const checkEpsilon = (prevW, nextW) => {  
 return prevW.reduce(  
 (result, prevWr, indexR) =>  
 result || prevWr.reduce(  
 (result, prevWre, indexE) => result || (Math.abs(prevWre - nextW[indexR][indexE]) > teacher.epsilon),  
 false  
 ),  
 false  
 )  
 }  
  
 do {  
 countOperations++  
 prevW = nextW.map(w => [...w])  
  
 M.forEach(({x}) => {  
 const {minI} = nextW.reduce(  
 ({minI, ds}, w, index) => {  
 const d = w.reduce(  
 (d, wi, i) => d + Math.pow(wi - x[i], 2),  
 0  
 )  
 if (index === 0)  
 return ({  
 minI: 0,  
 ds: [d]  
 })  
 else  
 return ({  
 minI: ds[minI] < d ? minI : index,  
 ds: [...ds, d]  
 })  
 },  
 {minI: 0, ds: []}  
 )  
  
 nextW[minI] = nextW[minI].map(  
 (wOld, i) => wOld + alfa \* (x[i] - wOld)  
 )  
  
 })  
  
 alfa \*= teacher.k  
 } while (checkEpsilon(prevW, nextW))  
  
 setCountOperations(countOperations)  
 setWeights(nextW)  
 setSs(  
 M  
 .map(  
 ({x}) => recognize.recognize(x, nextW)  
 )  
 )  
}  
  
$teacher  
 .on(changeTeacher, handleChangeTeacher)  
 .on(teachEvent, handleTeach)

import {*createEvent*, *createStore*} from "effector";  
  
const getRandomW = () => (Math.random() - 0.5)  
const getRandomWs = () => [  
 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0,  
].map(getRandomW)  
  
export const zeroWeight = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].map(getRandomWs)  
export const $weight = *createStore*(zeroWeight)  
  
export const setWeights = *createEvent*()