“Программа обнаружения изменений на спутниковых данных”

Автор: Аржанцев Андрей, группа 204
Руководитель: Родригес Залепинос Рамон Антонио
Тип проекта: программный, командный

Описание предметной области

- Разные подходы к решению такой задачи:
  На вход подается 2 спутниковых снимка одной местности в разное время, требуется описать изменения (не обычным сравнением, а более детально, информативно)

- Существует множество алгоритмов для решения, например: Decision Tree, SVM, алгоритмы, основанные на нейронных сетях, как CNN, DBN, RNN, и другие

Цели и задачи проекта

- Изучение алгоритмов решения задачи обнаружения изменений
- Реализация алгоритмов
- Тестирование работы алгоритмов, анализ результатов
- Создание удобного интерфейса для применения разных алгоритмов

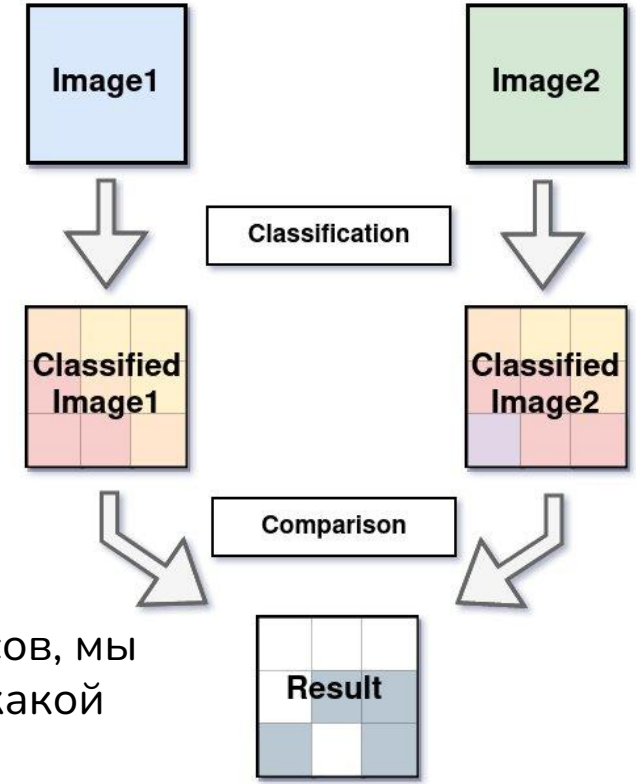
Моя часть проекта

- Моя часть проекта заключалась в реализации, тестировании и анализе результатов одного из алгоритмов.

- Я выбрал алгоритм, основанный на нейронной сети Fuzzy ARTMAP

Основная идея

1. На вход подаются два изображения одной местности в разное время
2. Каждое разделяется на более мелкие части
3. Каждую часть классифицирует наша нейросеть (говорит, на данном снимке лес, поле, дома или что-то еще)
4. Классы соответствующих частей сравниваются

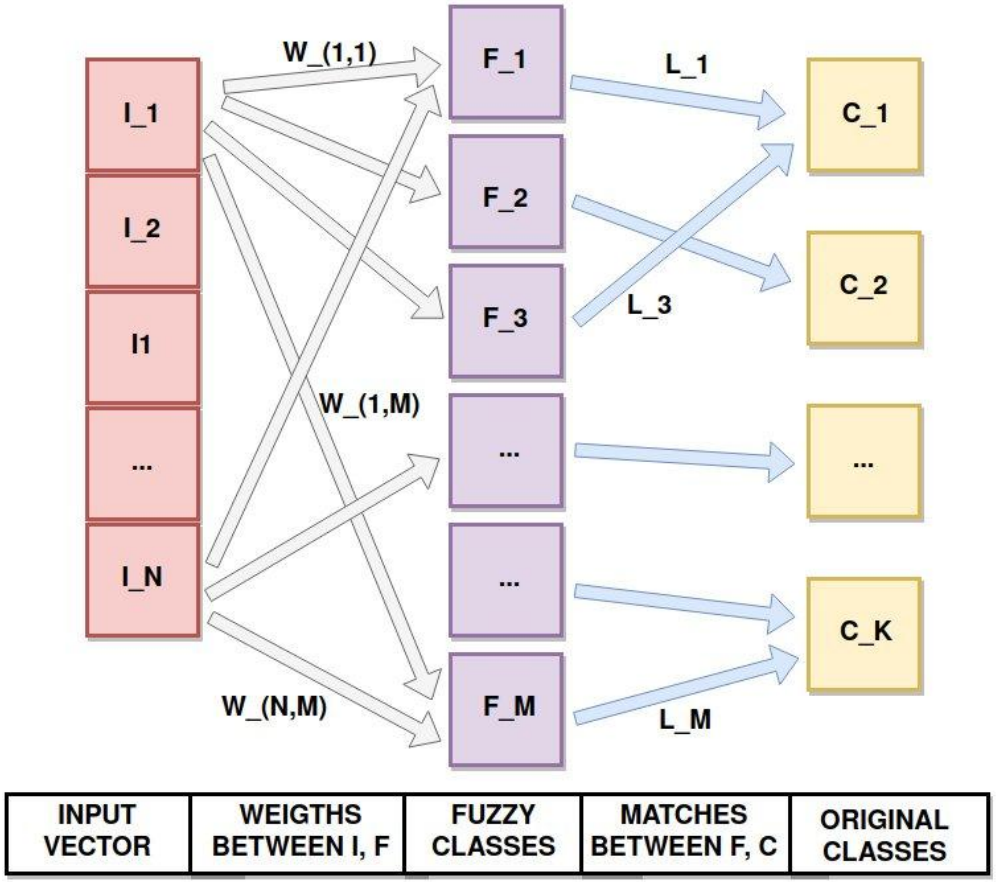


- Плюсы: Помимо результатов сравнения классов, мы получаем более ценную информацию о том, какой класс в какой перешел
- Минусы: Зависимость от наличия обучающей выборки, ее размера и информативности

Описание работы нейросети

Нейросеть FUZZY ARTMAP состоит из трех уровней:

- Вектор входных данных I (каждый из отрезка [0,1])
- Множество “размытых” классов F (некие подклассы, которые мы не знаем, но подразумеваем)
- Множество итоговых классов C



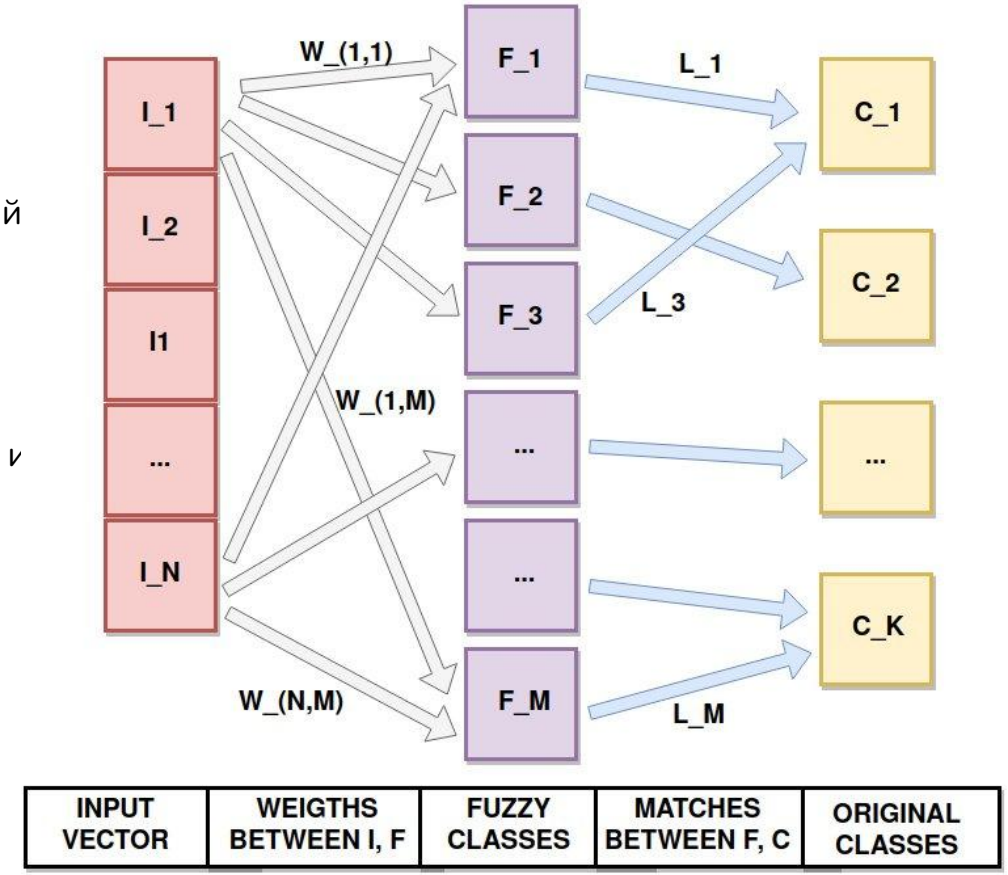
Описание работы нейросети

Между вершинами уровней есть веса (которые мы и обучаем):

- Веса между I и F - W_(i,j) (каждый из отрезка [0,1], отвечают за соответствие i-ого параметра j-ому размытому классу)
- Веса между F и C - L_i (задают соответствие между размытыми и настоящими классами)

Дополнительные параметры:

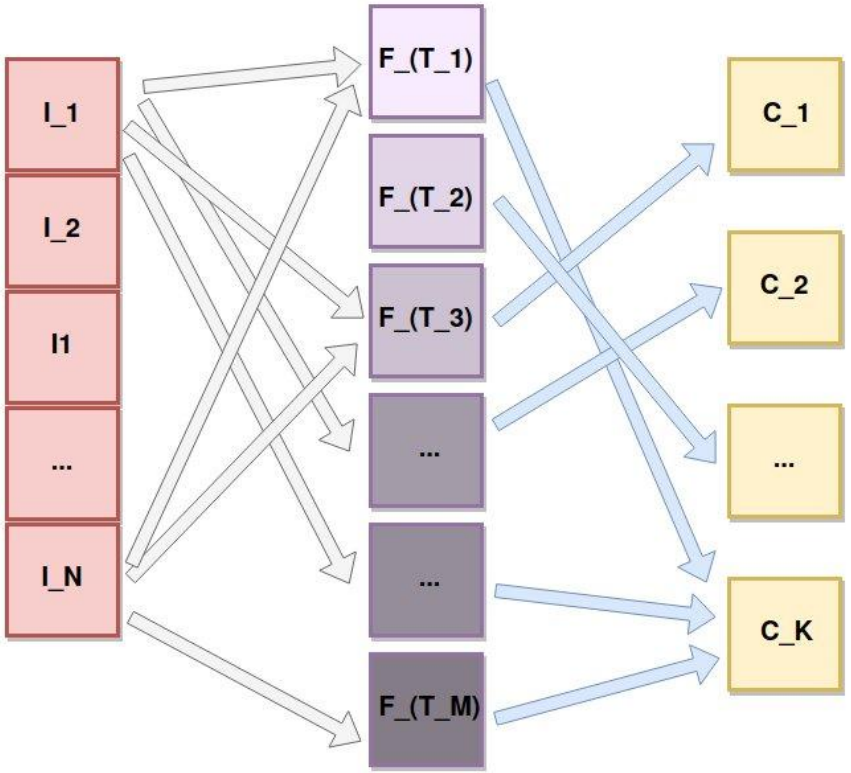
- параметр выбора (CH PARAM)
- параметр обучения (L)
- параметр бдительности (VIG)
- match параметр (EPS)



Описание работы нейросети

TRAIN:

- Делаем из входного вектора I вектор [I, np.ones - I]
- С помощью функции выбора сортируем все размытые классы по уменьшению “схожести” с входными данными



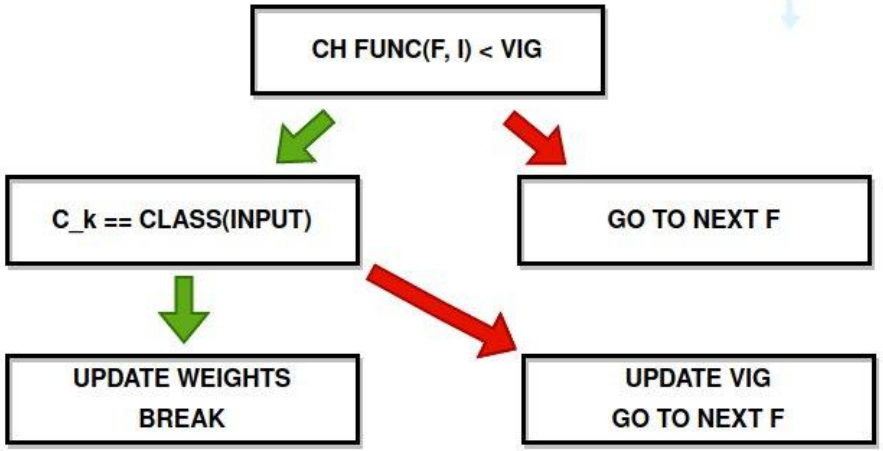
CHOICE FUNCTION (I, F_j) =
SUM (MIN (I , W_(., j)) /
/ (CHOICE PARAM + SUM(W_(., j)))

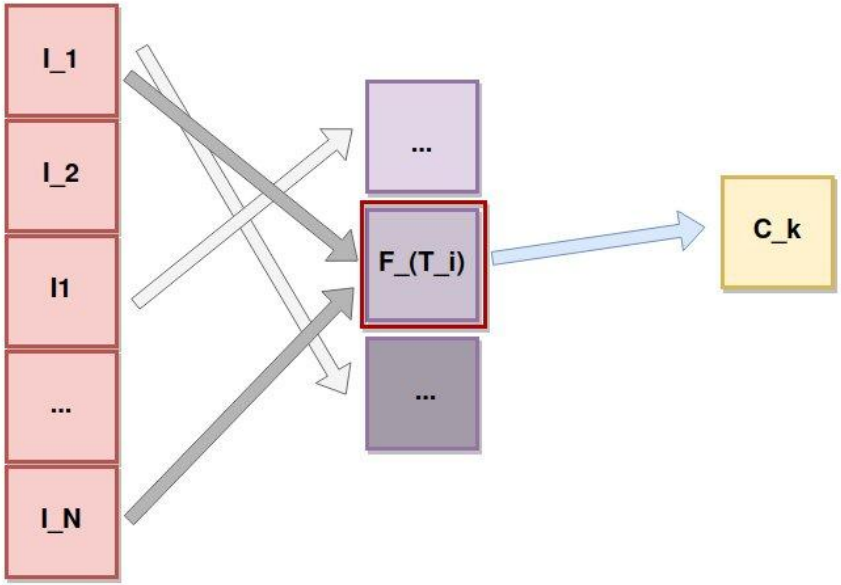
F_i < F_j <=> CH. FUNC. (I, F_i) < CH. FUNC. (I, F_j)

Описание работы нейросети

TRAIN:

- Перебираем вершины от лучшей к худшей и действуем по схеме





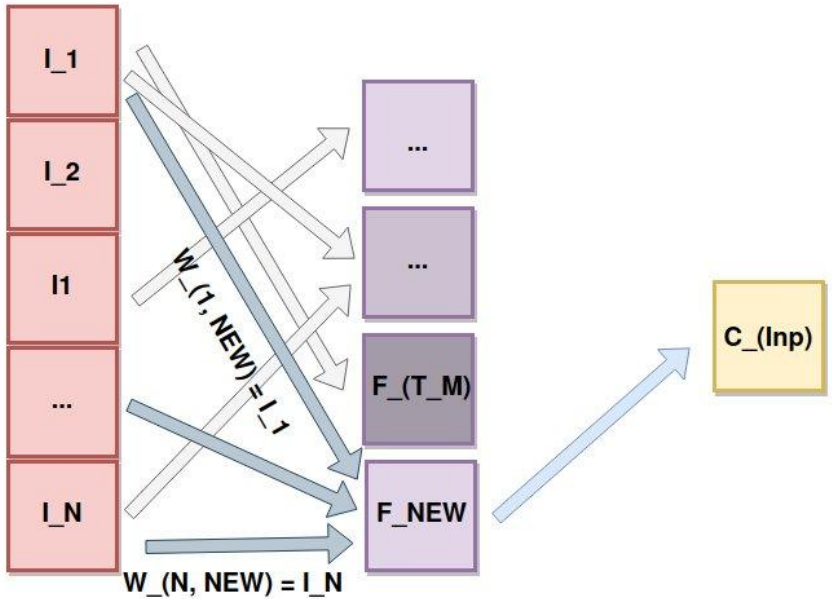
$$NEW\_W = L\,(\,MIN\,(I, W)\,) + (1 - L)\,W$$

$$NEW\_VIG = CH\ FUNC\,(F, I) + EPS$$

Описание работы нейросети

TRAIN:

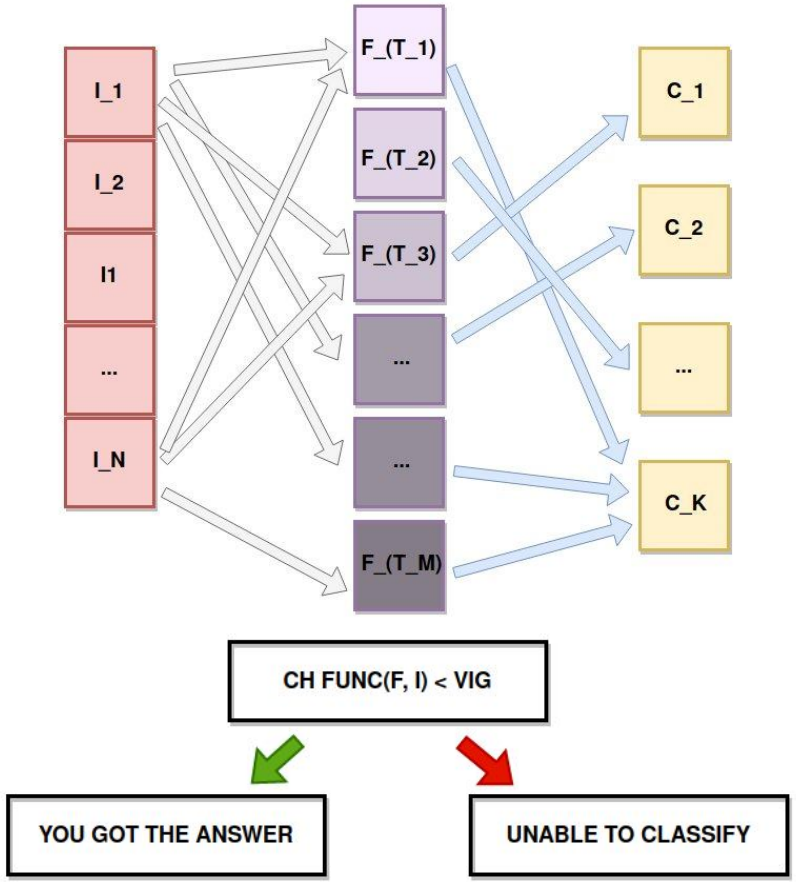
- Если мы прошли все вершины из F, но так и не нашли подходящую, создаем новую вершину, соответствующую в точности данному входу



Описание работы нейросети

PREDICT:

- Опять-таки сортируем размытые классу по “схожести” со входом
- Выбираем максимальный по “схожести” размытый класс. Ответ - соответствующий ему настоящий класс
- Если этот размытый не проходит тест на бдительность, вынуждены признать, что классифицировать эти входные данные точно мы не можем



Реализация нейросети

Основные моменты:

- Класс FuzzyARTMAP с объектами - массивы вер всех уровней, массивы весов, дополнительные параметры
- Методы класса - train и predict + доп.функции для них

```
class Fuzzy_Artmap():
    def __init__(self, M = 625, choice = 0.001,
                 learn = 0.5, vig = 0.75, st_vig = 0.75,eps = 0.001):
        #size of F-level
        self.M = M
        #choice parameter
        self.choice=choice
        #learning parameter
        self.learn=learn
        #changable vigilance parameter
        self.vig=vig
        #stable vigilance parameter
        self.st_vig=st_vig
        #matching parameter
        self.eps=eps
        #weights from F to C
        self.C=np.array([])
        #weights from I to F
        self.W=np.array([])
        #original data answers
        self.orig_result =np.array([])
        #size of W
        self.sz_W = np.array([])
```

```
    def train(self,I, I_res):
        self.M = len(I[0])
        self.orig_result=I_res
        self.C = [0]
        self.W = np.ones((1,self.M*2))
        self.sz_W = np.array([self.W.shape[0]])
        I = self.make_input(I)
        for index, a_i in enumerate(I):
            if index==0:
                self.W[0]=a_i
                self.C[0]=self.orig_result[0];
                continue
            T_list=np.array([self.choice_function(a_i, w_i) for w_i in self.W])
            T_max = np.argmax(T_list)
            while 1:
                #if no good F-vertex found, build a new one
                if sum(T_list)==0:
                    #if self.W.shape[0]<400:
                    self.C = np.concatenate((self.C,[self.orig_result[index]]))
                    self.W = np.concatenate((self.W,[a_i]),axis=0)
                    #else:
                    #    self.C[T_max] = self.orig_result[index]
                    break

                #best matching F-vertex
                J = np.argmax(T_list)
                res = sum(self.min_array(a_i,self.W[J]))/self.M
                if res >= self.vig:
                    if self.orig_result[index]==self.C[J]:
                        #making new weights
                        self.W[J] = self.learn*(self.min_array(a_i,self.W[J]))
                        + (1-self.learn)*(self.W[J])
                        break
                    else:
                        T_list[J] = 0
                        #making better vig, as we failed
                        self.vig = sum(self.min_array(a_i,self.W[J]))/self.M + self.eps
                else:
                    T_list[J] = 0
        self.vig = self.st_vig
        self.sz_W = np.concatenate((self.sz_W,[self.W.shape[0]]),axis=0)
```

```
def predict(self, input_i):
    I =self.make_input(input_i)
    ans = []
    for I_i in I:
        T_list=np.array([self.choice_function(I_i, W_i)
                         for w_i in self.W])
        while 1:
            if sum(T_list)==0:
                ans.append(-1)
                break;
            J = np.argmax(T_list)
            resonance = sum(self.min_array(I_i,self.W[J]))/self.M
            if resonance >= self.vig:
                ans.append(self.C[J])
                break
            else:
                T_list[J]=0
    return ans
```

Проверка работы нейросети

- Задача 1: научиться определять x%30 по [x%2, x%3, x%5, x%7, x]
- Результаты:

```
GOOD ANSWERS:1000
BAD ANSWERS: 0
"UNABLE": 0
AVG_VIG = 0.77
```

- Задача 2: научиться находить максимум в array(10), array[i]<10
- Результаты:

```
GOOD ANSWERS:914
BAD ANSWERS: 86
"UNABLE": 0
F_SIZE: 335
```

Описание используемого датасета

- 2 набора по 190 изображений района Ханьян города Ухань от 2002 и 2009 годов
- Изображения .tif, 150*150, у каждого пикселя 4 параметра (RGB+NIR)
- Каждое изображение принадлежит одному из 9 (на самом деле 8) классов

An overview of the overall image and scene classes of the dataset is shown below (see the README file for more information):

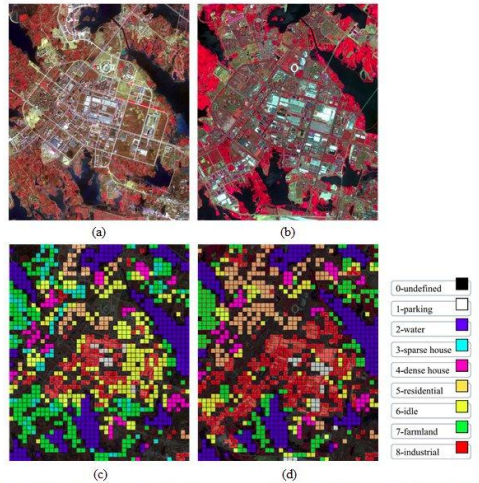
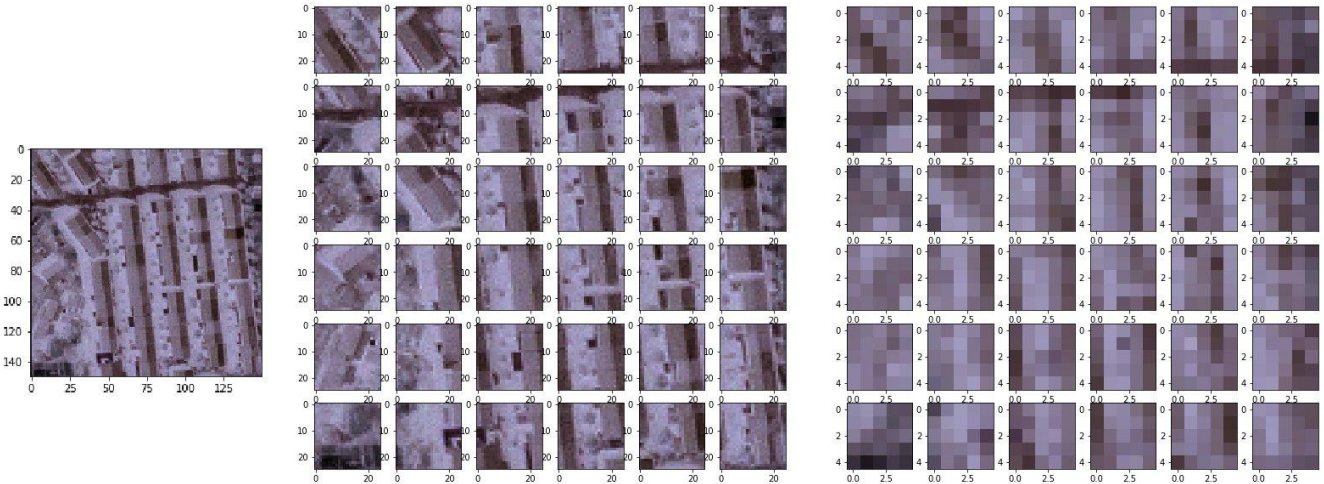


Figure. Pseudo-color images of the Hanyang area of the city of Wuhan, acquired in (a) 2002 and (b) 2009. Reference maps for the test samples in (c) 2002 and (d) 2009, where the different colors represent different scene classes.

0.UNDEF
1.ПАРКОВКИ
2.ВОДА
3.РЕДКАЯ ЗАСТРОЙКА
4.ПЛОТНАЯ ЗАСТРОЙКА
5.ЖИЛЫЕ РАЙОНЫ
6.ЗАБРОШЕННЫЕ РАЙОНЫ
7.РАСТИТЕЛЬНОСТЬ
8.ПРОМЫШЛЕННЫЕ РАЙОНЫ

Преобразование данных

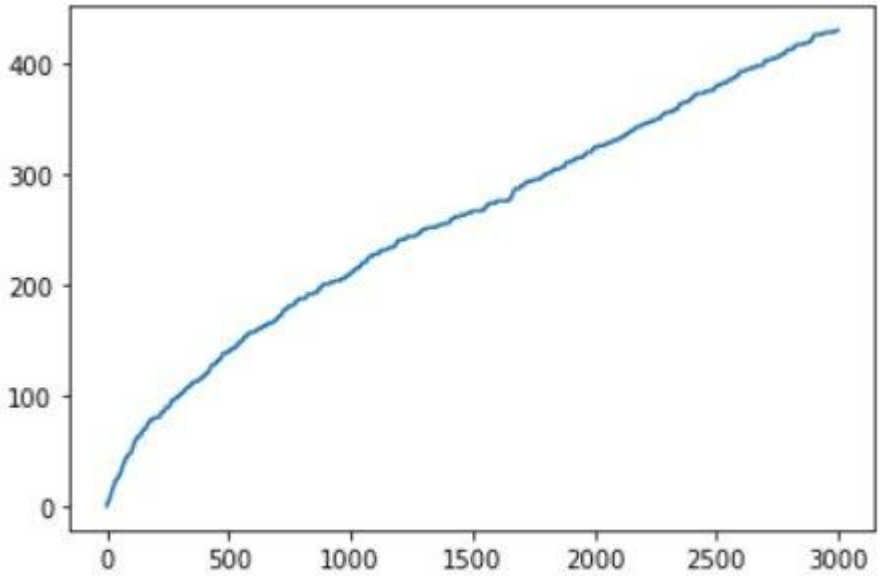
- Все изображение 150*150 я разбил на 36 изображений 25*25, каждое 25*25 сжал до 5*5, усреднив параметры в квадратах 5*5
- Таким образом, изначальный датасет превратился в 13680 изображений, которые я хранил, как np.array((5,5,4))



- Этими действиями я увеличил скорость и эффективность обучения, но потерял в информативности самого датасета

Преобразование данных

- График зависимости количества размытых классов от размера обучающей выборки:

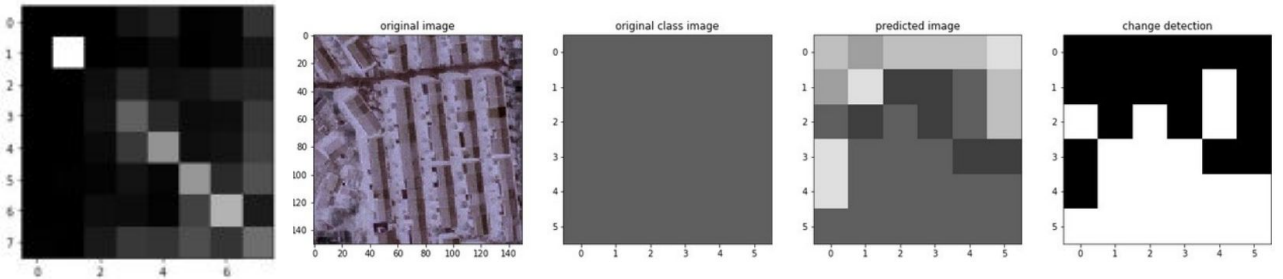


- Количество классов увеличивается очень быстро, а вместе с ним и скорость работы алгоритма

Результаты алгоритма классификации

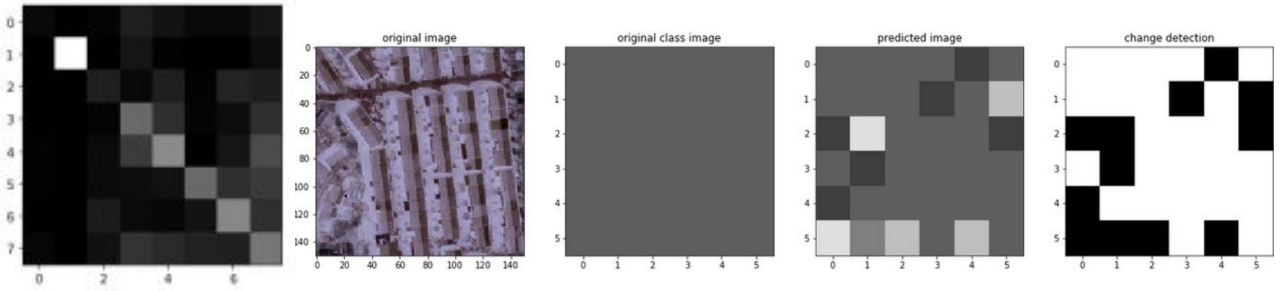
- Результат на выборке из 1000 сэмплов:

GOOD ANSWERS:448
BAD ANSWERS: 552
"UNABLE": 0



- Результат на выборке из 2000 сэмплов:

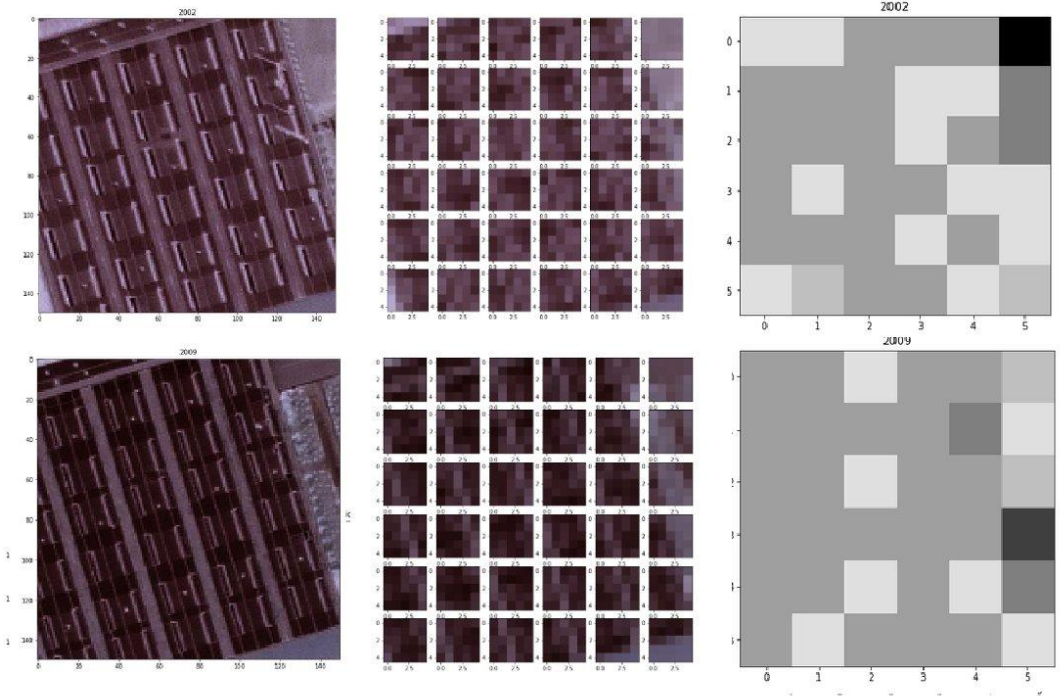
GOOD ANSWERS:472
BAD ANSWERS: 528
"UNABLE": 0

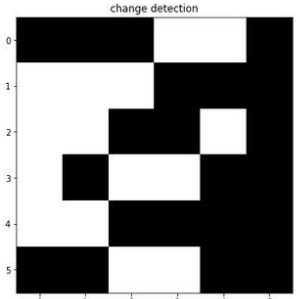


- Работает плохо, но ошибается сильно ниже среднего

Результаты алгоритма обнаружения изменений

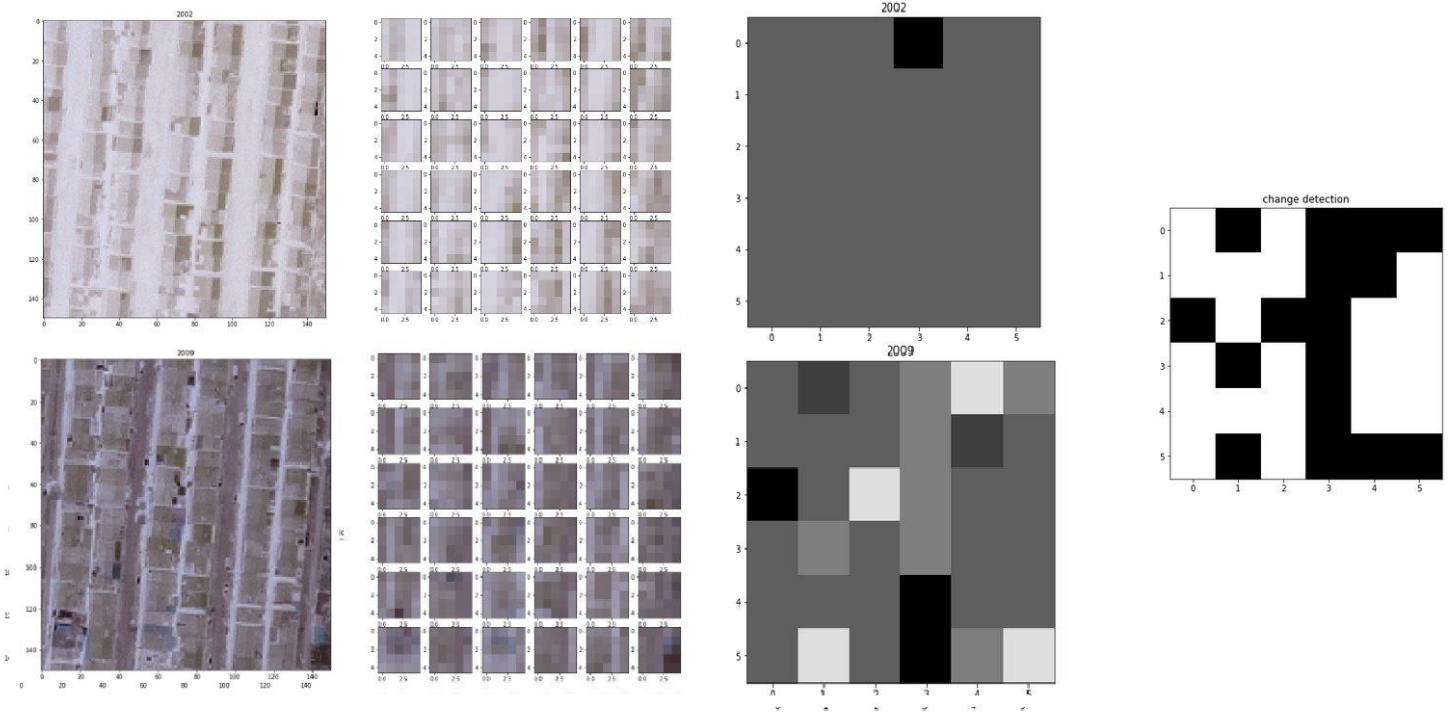
- Пример 1:





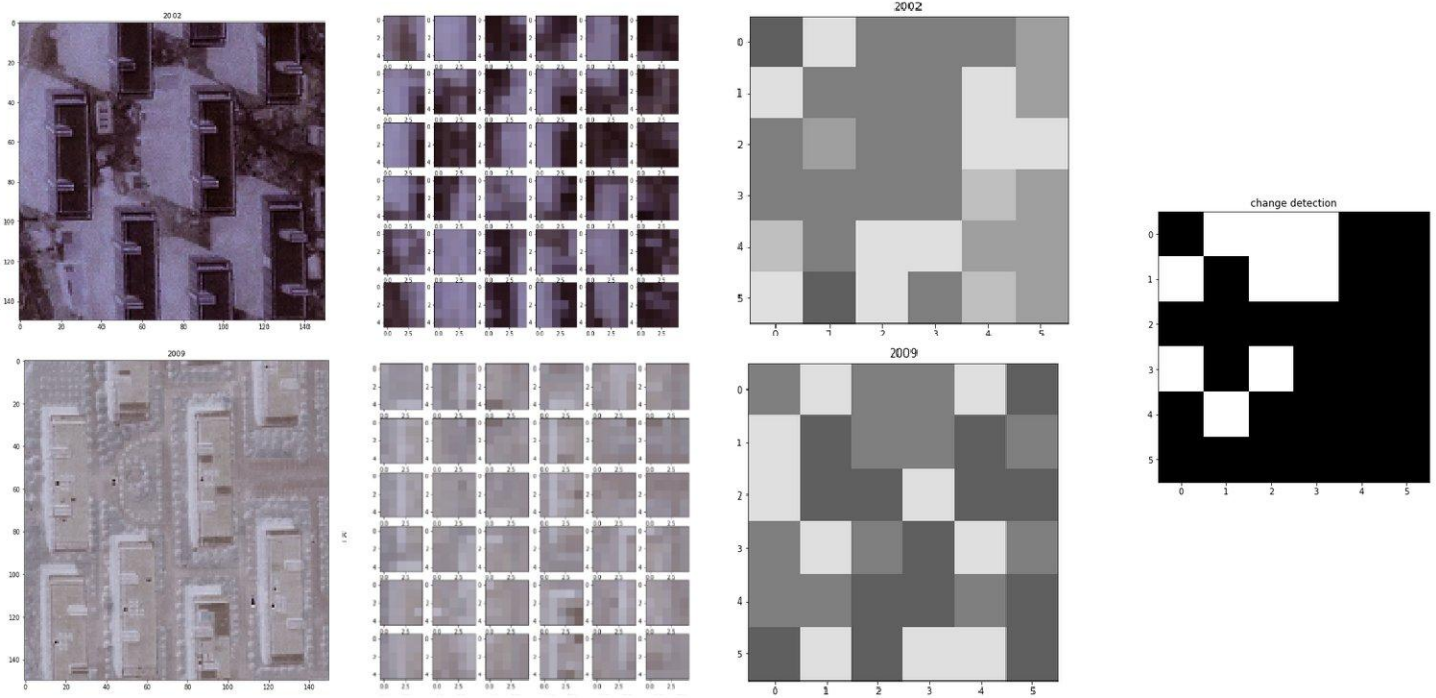
Результаты алгоритма обнаружения изменений

- Пример 2:



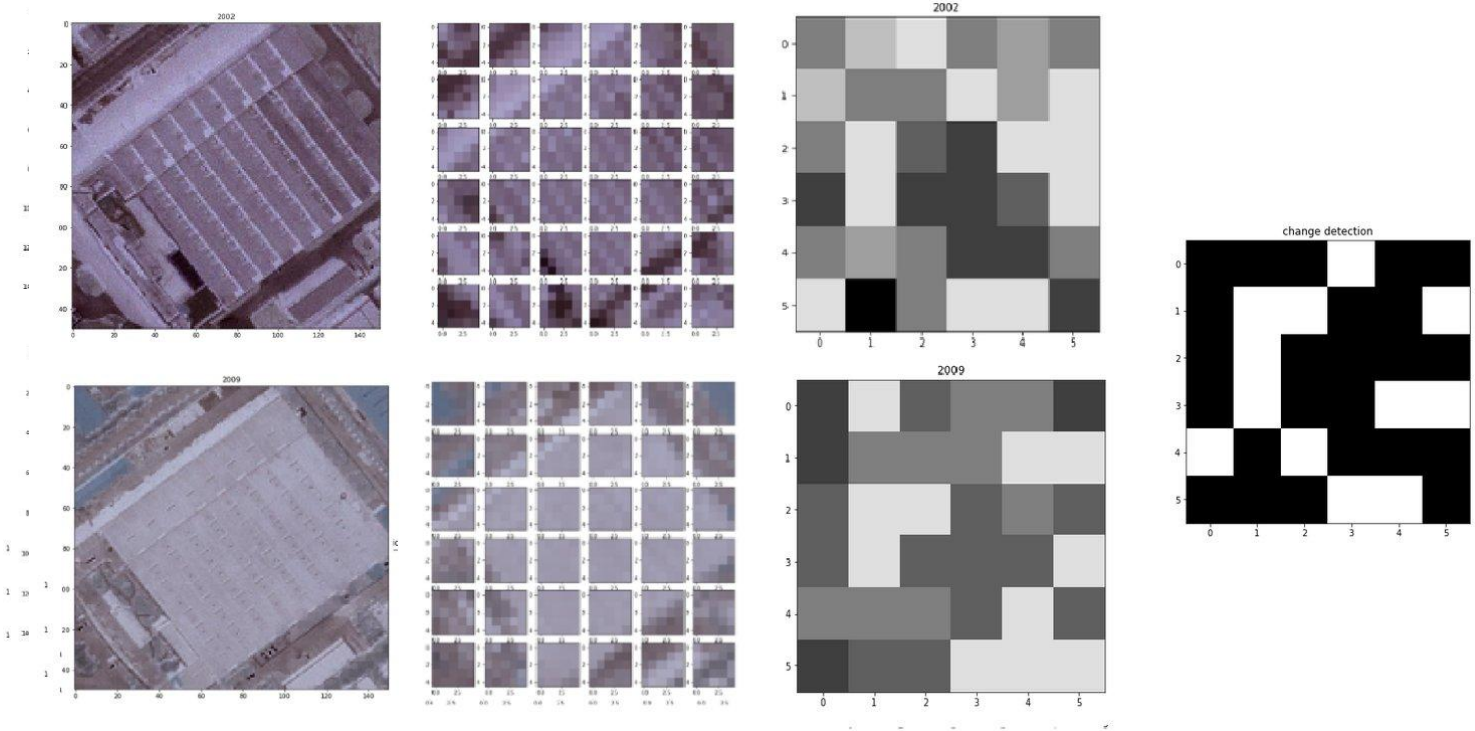
Результаты алгоритма обнаружения изменений

- Пример 3:



Результаты алгоритма обнаружения изменений

- Пример 4:



Итог

- Я изучил теорию и написал собственную реализацию нейросети Fuzzy ARTMAP для классификации изображений
- На основе данных из выбранного датасета я обучил и протестил нейросеть
- Результаты тестирования оказались довольно плохими (точность в районе 40-50 %), что объясняется многими факторами: ошибками в принадлежности к классам после дробления изображений, схожестью некоторых классов между собой, размером обучающей выборки
- Применяя алгоритм классификации изображений, я смог написать алгоритм для обнаружения изменений, который способен выдавать информацию не только об изменениях, но и о переходах частей изображений между классами

Литература

- Гитхаб с кодом проекта: https://github.com/reyarzhan/change_detection_project
- Статьи с описанием работы Fuzzy ARTMAP нейронной сети: https://clck.ru/bDamQ https://en.wikipedia.org/wiki/Adaptive_resonance_theory
- Статья о применении нейросети Fuzzy ARTMAP для решения задачи обнаружения изменений: https://www.researchgate.net/publication/248480978_Change_Detection_Using_Adaptive_Fuzzy_Neural_Networks
- Статья про применение нейронных сетей для задачи: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9136674
- Датасет: http://sigma.whu.edu.cn/newspage.php?q=2019_03_26_ENG

Спасибо за внимание