ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: Программа обнаружения изменений на спутниковых данных

Выполнил:
Студент группы БПМИ204 — Подпись — А.И.Аржанцев (И.О.Фамилия)

19.05.2022
Дата

Принял:
Руководитель проекта — Родригес Залепинос Рамон Антонио
Имя, Отчество, Фамилия

доцент ФКН
Должность, ученое звание

ФКН НИУ ВШЭ
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки 19 мая 2022 — 8 — *Родригес*
Оценка (по 10-ти бальной шкале) — Подпись

Москва 2022

Содержание

Аннотация

Реализация и тестировка алгоритма для обнаружения изменений на спутников данных

1 Введение

Наша команда делала проект "Программа обнаружения изменений на спутниковых данных". Нарправление "change detection"крайне обширно и существует огромное количество различных подходов и идей решений этой определенной задачи. Мне надо было выбрать какой-то из алгоритмов, разобраться в нем и реализовать. Прочитав много статей и описаний методов в различных источниках я остановился на подходе, использующем нейронную сеть fuzzy ARTMAP, алгоритм показался мне довольно идейным, теоретически и практически подъемным для меня, в то же время совершенно не банальным.

Моей задачей было написать код данной нейросети, оценить ее эффективность, скорость и информативность результата, на основе нейросети написать алгоритм, решающий задачу нахождения изменений, показать результат его работы на примерах.

2 Описание алгоритма

Так как я уже подробно описал всю теоретическую часть алгоритма и все идеи, использующиеся в реализации, в работе для КТ1, здесь я вкраце расскажу основную идею алгоритма, проблемы, с которыми я столкнулся в процессе реализации, и то как я с ними справлялся.

Начнем с того, что задача обнаружения изменений является во многом вторичной при использовании этого алгоритма. Сама нейросеть обучается определять по изображению принадлежность соответствующей местности какому-то из классов (леса, поля, дороги, постройки и т.д). Умея решать эту задачу, задача сводится к сравнению результатов классификации двух изображений. Выбирая этот алгоритм для своего проекта, он понравился мне как раз тем, что информация об изменениях кажется наиболее информативной: мы мало что можем увидеть и проанализировать, сравнивая пиксели и другие графические параметры изображений, в то время как реультат действия моего алгоритма более наглядный и полезный, так как в итоге мы видим изменения классов, например, если на изображении был лес, а стало поле, то именно это и выдаст мой алгоритм.

2.1 Описание работы нейросети

Теперь вкраце о решении задачи классификации изображений. Fuzzy ARTMAP - нейросеть, состоящая из 3 уровней, где первый - входные данные, последний - итоговые классы, а между ними - размытые классы. Между вершиными первого и второго уровня у нас ребра какого-то веса, которые мы и будем обучать. Также есть какая-то инъекция из вершин второго уровня в итоговые классы, которые грубя говоря означают "этот размытый класс является подразрядом данного класса".
Когда мы обучаем нейросеть, мы используя функцию выбора для текущих весов ребер, выбираем из второго уровня наиболее подходящий к данному входу размытый класс. Далее в зависимости от того, соответствует ли данный размытый класс правильному ответу, либо пересчитываем веса, чтобы далее на близких к данному входах у нас функция выбора опять выбирала этот размытый, либо переходим следующей наиболее подходящей вершине. Если ни одна вершина в итоге не подошла, либо с какого-то момента они перестали проходить порок значения функции выбора, мы создаем новый размытый класс, предполагая, что входное изображение не похоже на другие и требует новый размытый класс.

Наконец, когда нейросеть обучилась на известных данных, мы можем использовать ее для классификации неизвестных изображений. Для этого мы должны также поместить входные данные в вершины первого уровня, той же функций выбора сделать найти подходящий размытый класс и ответом будет итоговый класс, соответствующий данному размытому. Если же ни один размытый класс не подошел под наш вход, мы вынуждены сказать, что классифицировать изображение невозможно(можно интерпретировать этот исход как принадлежность входу классу, не присутствующему в обучаемой выборке).

2.2 Описание работы алгоритма

Несмотря на то, что задача обнаружения изменений уже после классификации изображений заключается по сути только в сравнении полученных классов, для более быстрой работы алгоритма, лучшей эффективности и большей наглядности результатов, здесь есть еще несколько тонкостей. Определять класс всего изображения мне показалось очень малоинформативным - просто сравнить классы двух изображений как-то мало. Поэтому входные изображения сначала делились на несколько маленьких изображений (даже больше, маленькие изображения делились на еще более маленькие, значения пикселей в которых усреднялось), каждое из которых классифицировалось и сравнивалось с соответствующим аналогом из другого времени. Этим действием я убил сразу несколько зайцев: результат стал состоять не из одного значения изменения класса, а из целой матрицы значений изменения, что удобней анализировать, а также этим я во много раз увеличил обучающую выборку и уменьшил размер входных данных, из-за которого сеть обучалась очень медленно и плохо (почти каждое изображение определялось в новый размытый класс, так как изображения были слишком большие, чтобы быть похожими).

Наконец, получив исходную матрицу изменений, для последующего анализа я преобразовавал ее в матрицу непосредственно изменений (изменился/не изменился класс) и в матрицу переходов из класса в класс.

3 Реализация алгоритма

Реализация всего алгоритма, как и многие выкладки из проекта, есть в гитхабе, который указан в литературе.

Самой объемной частью кода было описание класса Fuzzy ARTMAP со всеми необходимыми методами. Большая часть известных нейросетей уже присутствуют в различных библиотеках на питоне, но своя реализация fuzzy ARTMAP была одним из важных результатов проекта, и во многом помогла вникнуть в основные идеи, плюсы и минусы данного подхода.
Основными методами были собственно train (для обучения) и predict (для классификации).

```
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))
    I = self.make_input(I)

    for index, a_i in enumerate(I):
        #just to see how fast training goes
        #if index%1000==0:
        #    print(index)

        if index==0:
            self.W[0]=a_i
            self.C[0]=self.orig_result[0];
            continue
```

В качестве аргументов для train передается два массива np.array - массив разных входных данных и массив правильных ответов на них. Проходим по очереди по всем входным данным, для первого набора делаем размытый класс в точности соответстующий данному входу, далее уже каждый раз считаем массив T_{list} - значения функции выбора для существующих размытых классов, сортируем и по порядку перебираем, проверяем проходит ли элемент под текущую бдительность, и увеличиваем параметр бдительности, пока не найдем нужный размытый класс, для которого пересчитаем веса. Если нужного размытого класса не нашлось(то есть мы занулили все элементы T_{list}), создаем новый размытый класс. По итогу получаем обученные веса между первым и вторым уровнем нейронной сети.

В качестве аргументов для predict передается массив np.array - массив разных входных данных. Для каждого набора входных данных считаем T_{list}, сортируем, перебираем уже без проверки на соответствие с ответом, только проверяем соответствие бдительности, пока не найдем подходящий размытый класс, в массив ответов записываем соответствующий ему класс, иначе.

Дополнительные функции: make input - добавляет к входным данным эти же данные, вычтенные из 1(complement coding), min array - считает поэлементный минимум массивов, choice function - функция выбора для данного

```
    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
```

Рис. 1: Реализация нейросети - train

```
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
```

Рис. 2: Реализация нейросети - predict

```
#complete our input
def make_input(self, I):
    return np.concatenate((I,np.ones(I.shape) - I),axis=1)

#func to calculate a^b
def min_array(self,a,b):
    return np.array([min(a[i],b[i]) for i in range(len(a))])

#func with which we select the optimal F-level vertex for our input
def choice_function(self,x,w):
    T = sum(self.min_array(x,w))/(self.choice + sum(w))
    return T
```

Рис. 3: Реализация нейросети - доп.функции

входа и данного размытого класса.

Для того, чтобы проверять работу нейросети, корректировать входные параметры и находить ошибки в коде, я написал несколько простых задач для проверки, которые в то же время показывают работу нейросети, поэтому мне показалось целесообразным предъявить эти тесты с их результатами.

3.1 Проверка работы неройсети

Первой задачей нейросети было обучиться определять остаток от деления числа на 30 при входных остатках от числа на 2,3,5,7 и самого числа. Как можно понять, я проверял, что нейросеть обучится использовать только необходимые параметры входных данных и последние 2 параметра не будут учитываться с большим весом.

Как мы видим, все работает идеально, как и ожидалось на такой несложной задаче.

На этом тесте я во-первых, довел код до работающего, а также смог найти необходимые входные параметры (параметр бдительности - 0.75, параметр обучения - 0.5), необходимые для хорошей эффективности в этой задаче (как оказалось, в итоговой задаче эти параметры также работали лучше всего).

```
Ввод [8]: B = Fuzzy_Artmap()
testing_data = np.array([[(i%2)/2, (i%3)/3, (i%5)/5, (i%7)/7, i/10001] for i in range(10001)])
testing_answers = np.array([i % 30 for i in range(10001)])
B.train(testing_data, testing_answers)
test = np.array([random.randint(1,10000) for i in range(1000)])
test_predict = np.array([[(i%2)/2, (i%3)/3, (i%5)/5, (i%7)/7, i/10001] for i in test])
res, sum_reson = B.predict(test_predict)
good, bad, undef = results(res,test)
print("right answers:", good, "bad answers:", bad, "undefined answers:", undef)
print("average resonance:", sum_reson/1000)
print("amount of fuzzy classes:", B.W.shape[0])

right answers: 1000 bad answers: 0 undefined answers: 0
average resonance: 0.7692358160515792
amount of fuzzy classes: 129
```

Рис. 4: Результаты первого теста нейросети

Второй задачей было обучиться находить максимальный элемент в массиве из пяти чисел от 0 до 10. Эта задача уже более сложная и очевидно, не решается идеально этой нейросетью при адекватном размере обучающей выборки (входные данные могут быть очень разнообразны, а ответ на два "похожих"входа может не совпадать довольно часто). Поэтому я просто хотел добиться точности в 90 процентов.

Результаты показаны на изображении.

```
Ввод [14]: C = Fuzzy_Artmap()
testing_data = np.array([[random.randint(1, 10)/10 for i in range(5)] for i in range(3001)])
testing_answers = np.array([np.argmax(testing_data[i]) for i in range(3001)])
C.train(testing_data, testing_answers)
test_predict = np.array([[random.randint(1, 10)/10 for i in range(5)] for i in range(3000)])
res, sum_reson = C.predict(test_predict)
good, bad, undef = results(test_predict, res)
print("right answers:", good, "bad answers:", bad, "undefined answers:", undef)
print("amount of fuzzy classes:", C.W.shape[0])

right answers: 914 bad answers: 86 undefined answers: 0
amount of fuzzy classes: 335
```

Рис. 5: Результаты второго теста нейросети

На этом тесте я понял основную проблему, которая присутствует в этом алгоритме. Если входные данные имеют сложную структуру, то на очень многих входных данных будут появляться новые и новые размытые классы, что очень замедляет обучение и дальнейшую классификацию. В общем-то поэтому в дальнейшей реализации алгоритма и пришлось искать решения, как уменьшить размер входных данных и их структуру.

3.2 Работа с данными

Одной из главных проблем, с которой я столкнулся в этом проекте, был поиск подходящего датасета, на котором можно обучаться и тестироваться. Проблема в том, что от обучающей выборки я требую довольно много информации - помимо самого изображения, мне нужна была информация о классе, которому каждое изображение принадлежит. В одной из статей, которую я читал, использовался датасет, где значение класса было дано у каждого пикселя, но чего-то похожего мне нигде найти не удалось.
В итоге я нашел датасет на сайте, указанном в литературе. В нем было по 190 снимков со спутника района Ханьян, города Ухань от 2002 года и от 2009 года, где каждый пиксель представляется четырьмя параметрами - Blue, Green, Red и Near-Infrared. Каждое изображение также принадлежало какому-то одному из 8 классов, а именно:

1. Парковки
2. Вода
3. Редкая застройка
4. Плотная застройка
5. Жилые районы
6. Неактивный/заброшенные районы

7. Растительность
8. Промышленные районы

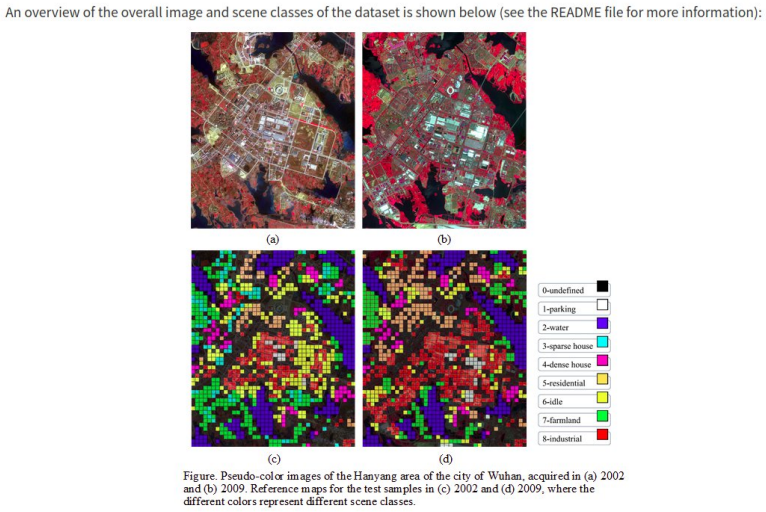

Рис. 6: Визуализация всего датасета(одно изображение - один маленький квадратик)

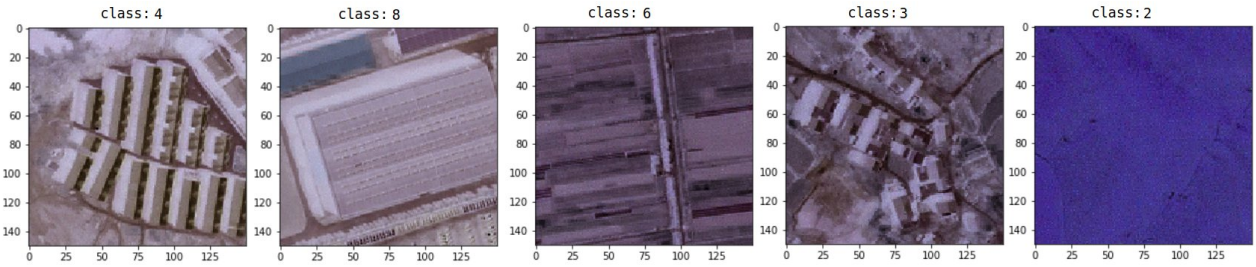

Рис. 7: Примеры изображений из датасета

С этими данными я проделал такие махинации: каждое изображение я поделил на 36 изображений 25×25, а в них в каждом квадрате 5×5 усреднил значение цвета. В итоге я получил 13680 входных данных в формате $np.array((5, 5, 4)$, из которых 10000 взял для обучения(в итоге понадобились не все) и остальные для тестирования. Одной из проблем было то, что класс, который раньше был у изображения 150×150 мне пришлось продублировать во все соответствущие ему входные данные. Это конечно, влияет на эффективность, так как отдельные маленькие изображения совсем не обязаны соответствовать классу изображения, откуда они взялись (например, в классе *water* частей изображения, где действительно вода, далеко не все), но как с этим бороться я не придумал, а датасет с более мелкими изображениями не нашел.

3.3 Финальная часть

Написав код и распарсив датасет, мне оставалось только протестировать его работу, проанализировать результаты. Для этого я сначала обучил нейросеть, после чего из оставшихся изображений выбрал 1000 и для них посчитал непосредственно точность классификации вместе с точностью классификации для каждой отдельной пары классов(результаты далее).

Наконец осталось проверить обнаружение изменений. Для этого надо было взять из того же датасета изображения одной и той же местности, но в разное время, прогнать их через нейросеть и сравнивать результаты классификации.

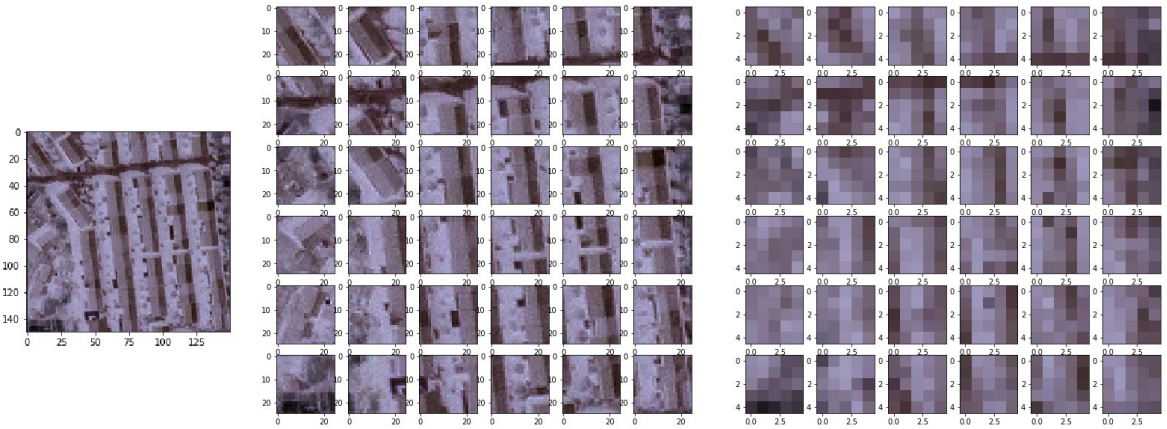

Рис. 8: Преобразование данных на примере одного снимка

4 Результаты

Проделав всю описанную работу, хочу привести результаты, к которым я пришел. Первая часть - про то, насколько хорошо работает нейросеть для моей задачи. Вторая часть - про то, как работает с помощью нее поиск изменений на тестовых заданиях.

4.1 Проверка эффективности нейросети

Посмотрим на результаты, которые дала наша нейросеть после обучения. Я пообучал нейросеть на выборках размера 1000 и 2000 и для каждого случая посчитал, 1) с какой точностью мы получаем ответ ,2) представил таблицу классов по тому, сколько раз класс i назывался классом j (рядом визуализация этой таблицы) 3) для наглядности пример, который иллюстрирует обнаружение изменений на изображении, которое вы видели выше.(в predicted image цвета квадратиков - номер класса изображения от 1 - белого, до 8 - черного, в change detection изменение - черный цвет)

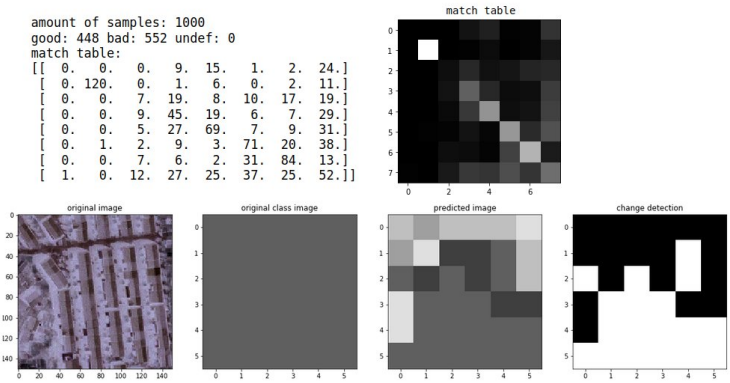

Рис. 9: Результаты при обучении на 1000 сэмплах

Мы видим, что нейросеть работает плохо. Точность не доходит и до 50 процентов, но при этом она явно ошибается сильно меньше среднего. Хорошо заметно светлую полосу на главной диагонали таблицы match table, а, например, класс water нейросеть отличает от другого почти идеально.

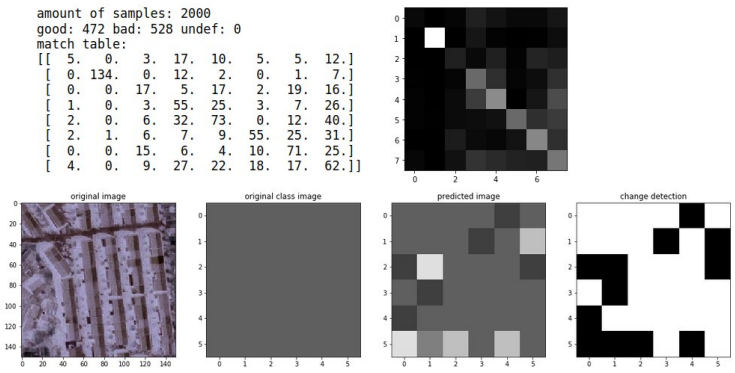

Рис. 10: Результаты при обучении на 2000 сэмплах

Также можно сказать, что чем больше выборка, тем эффективнее нейросеть обучается, что неудивительно. Проблема в том, что чем больше у нас обучающая выборка, тем больше новых размытых классов у нас появляется. Ждать обучения на большем количестве тестов слишком долго, как и ждать после нее классификацию большого количества изображений, что сказывается на работе алгоритма. Из-за этого я и обучался лишь на умеренно больших выборках.

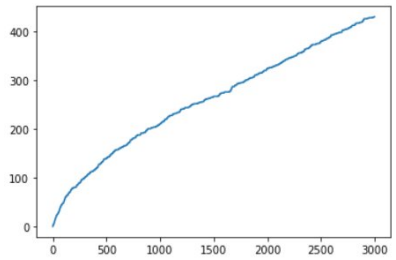

Рис. 11: График зависимости количества размытых классов от величины обучающей выборки

Если посмотреть на пример (2 и 4 строки на рисунке 10), то можно заметить, что он посчитался явно лучше среднего в обоих случаях. При этом в первом случае, вероятно, нейросеть смутила дорога, находящаяся наверху изображения и явно контрастирующая со всей остальной цветовой гаммой изображения.

Подытожив, можно сказать, что моя нейросеть на практике явно не будет пригодна для решения задачи классификации. Это во многом связано с датасетом, на котором проводилось обучение, где классы похожи между собой, с необходимостью дробить изображения на более мелкие и терять в точности ответов на тренировочном датасете. Но при этом мы видим и хорошие моменты - нейросеть хоть и плохо, но работает. С несложными задачами (например, отличить воду от не воды) она справляется, с более сложными точность падает до 40-50 процентов, что терпимо, учитывая вероятность случайно попасть в $\frac{1}{8}$.

4.2 Проверка работы алгоритма обнаружения изменений

Теперь о том, как я получаю непосредственно изменения на двух снимках из разного времени. Для примера из датасета я взял 2 изображения одного и того же места 2002 года и 2009 года. На изображении можно увидеть: их изначальный вид, вид после моего сжатия данных, классификацию каждого набора данных и матрицу изменений(сверху 2002, снизу 2009).

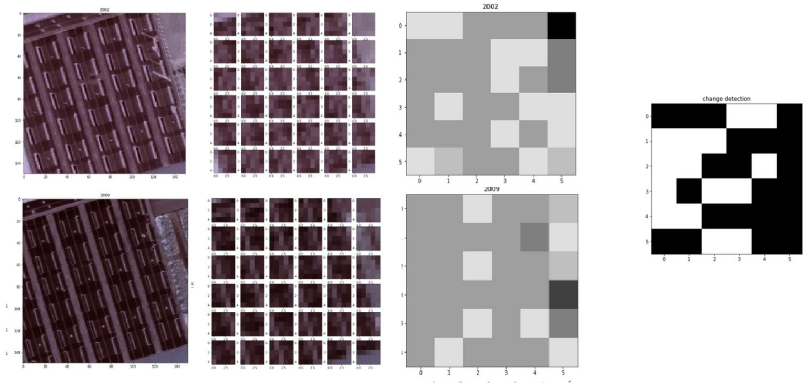

Рис. 12: Пример 1 работы алгоритма обнаружения изменений

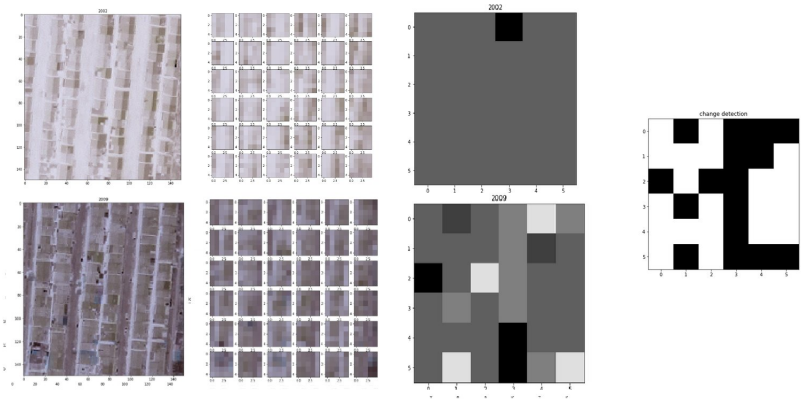

Рис. 13: Пример 2 работы алгоритма обнаружения изменений

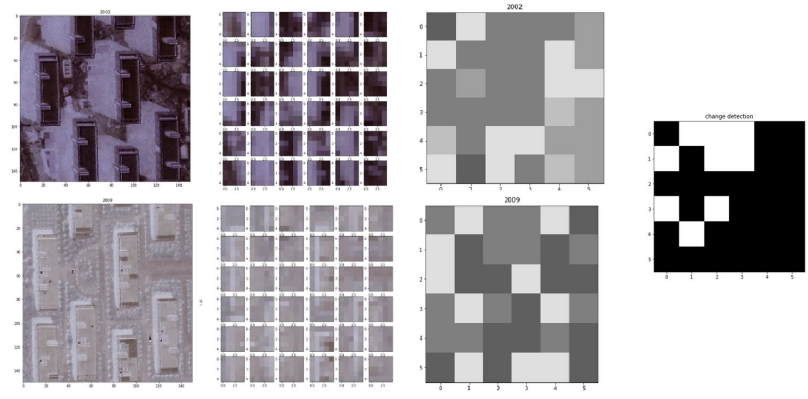

Рис. 14: Пример 3 работы алгоритма обнаружения изменений

Из примеров стоит отметить первый, где правая часть, которая как раз изменилась со временем, на рисунке изменений как раз отмечена черным, левая же часть пополам белая и черная, но по классификациям можно заметить, что в действительности классифицровалась она на обеих картинках преимущественно правильно, просто ошибки оказались в разных местах снимков.
Во втором примере мы сразу видим отличную классификацию первого изображения, но второе изображение классифицировалось не так хорошо, поэтому результат неоднозначный.
В третьем примере действительно визуально все очень изменилось, что и показывает наш алгоритм.

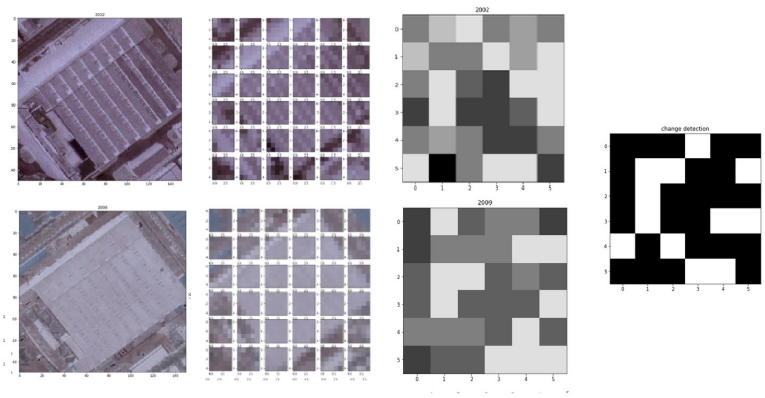

Рис. 15: Пример 4 работы алгоритма обнаружения изменений

В четвертом получились очень неоднозначные классифиакции, тем не менее алгоритм выдает нам достаточно большое количество совпадений, что в принципе логично, ведь наша нейросеть если и ошибается, то скорее всего ошибается на похожих входных данных одинаково, из-за анализз изменений не портится.

Говоря об итогах, мы можем сказать, что несмотря на не самую лучшую работу нашей нейронной сети для классификации, с помощью алгоритма многие пары изображений мы все-таки можем сравнить и проанализировать эти изменения. Также результат получился довольно наглядным, что дает возможность легче сверять конкретные данные с действительностью, которую мы видим на изображении.

5 Дополнительно

Так как проект у нас командный и мы должны сделать приложение, то дополнительной задачей было написать алгоритм так, чтобы его можно было использовать в приложении, которое сделал мой сокомандник. Все оказалось довольно просто, по сути единственной задачей было уметь в правильном виде передавать входные и выходные данные.
Для преобразования файла в numpy.array я использовал imread, что поддерживает все популярные форматы. От изображения требовалось только, чтобы размер соответствовал тому, какой он у меня был в датасете, то есть $25k \times 25n$. Для преобразования результата, который у меня в формате numpy.array, я использовал PIL.Image.fromarray.
Все что между вводом и выводом делается тем же кодом, которым я локально все тестировал без приложения, нейронную сеть, уже обученную, я сохранил в отдельный файл.

6 Литература

1. Гитхаб с кодом
https://github.com/reyarzhan/change_detection_project
2. Статья с описанием работы Fuzzy ARTMAP нейронной сети
https://clck.ru/bDamQ
3. Статья о применении Fuzzy ARTMAP нейронной сети
https://www.researchgate.net/publication/248480978_Change_Detection_Using_Adaptive_Fuzzy_Neural_Networks
4. Dataset
http://sigma.whu.edu.cn/newspage.php?q=2019_03_26_ENG
Другие использованные статьи и сайты:
6.https://clck.ru/bDU7p
7.https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9136674
8.https://en.wikipedia.org/wiki/Adaptive_resonance_theory

9.https://clck.ru/bDYL5