

ПРАВИТЕЛЬСТВО САНКТ – ПЕТЕРБУРГА КОМИТЕТ ПО ОБРАЗОВАНИЮ
Государственное бюджетное общеобразовательное учреждение ГБОУ лицей №329
Невского административного района г. Санкт – Петербурга
Адрес: г. Санкт-Петербург, пр. Елизарова. 7б
Телефон/факс 417-27-18, e-mail: school329spb@yandex.ru

ПРОЕКТНО – ИССЛЕДОВАТЕЛЬСКАЯ ДЕЯТЕЛЬНОСТЬ

ТЕМА ПРОЕКТА

**Создание классификатора изображений на основе алгоритмов
свёрточных нейронных сетей**

Руководитель проекта
Зилинских Анна Васильевна,
учитель информатики лицея
№329
Работу выполнил
Шамов Илья Андреевич,
10А класс

г. Санкт – Петербург
2017

Оглавление

Тезисы.....	3
Машинное обучение.....	4
Исторические корни.....	4
Виды.....	4
Свёрточные нейронные сети.....	4
Виды слоёв СНС.....	5
Тренировка НС.....	6
Пример.....	7
Задача.....	7
Решение задачи.....	7
Разбор работы СНС.....	8
Архитектура СНС и результаты работы.....	9
Приложение.....	12
ImageWorker.py.....	12
activations.py.....	13
Convolution.py.....	13
Список литературы.....	17

Тезисы

Тема работы: «Создание классификатора изображений на основе алгоритмов свёрточных нейронных сетей»

Целью работы является разработка программы, способной обучаться классификации изображений на основе обучающей выборки. В работе сделан акцент на разработку без использования библиотек машинного обучения.

В работе были использованы такие методы, как:

- Теоретический анализ литературы и материалов, представленных в виде интернет-ресурсов
- Анализ исходного текста программ машинного обучения, реализованных на языках программирования Python, C++, Matlab и JavaScript
- При выполнении исследовательской части работы — исследовательский метод

Данная работа актуальна как введение в алгоритмы машинного обучения в связи с построением и разбором принципов и алгоритмов СНС с нуля. Также программа может применяться в сфере релевантного подбора графической информации для пользователей социальных сетей, развлекательных сервисов, распознавания лиц в системах безопасности, распознавания рукописного текста в школах и т.д.

В результате работы была реализована программа на языке программирования Python3, обучающаяся классификации графических изображений на основе опыта, полученного в ходе «обучения» на заранее подготовленной выборке.

Машинное обучение

Исторические корни

Рассмотренные в работе алгоритмы являются частью математической модели искусственных нейронных сетей (ИНС), построенных по принципу функционирования и организации нервных клеток организма человека.

Впервые модель нейрона ввели Уолтер Питтс и Уоррен Мак-Каллок в 1943 году [1]. В течение середины-конца XX века тема машинного обучения активно разрабатывалась, но не могла быть полностью проверена на практике из-за недостатка вычислительных ресурсов.

В XXI веке идея ИНС переросла в нейросетевую парадигму программирования. Нейросети активно используются в сфере анализа Больших Данных, сервисах подбора информации, классификации информации и т. д.

Виды

В основном, все ИНС можно разделить на категории:

- Рекуррентные НС
- Свёрточные НС
- НС прямого распространения
- Шифровальщики
- Машины Больцмана

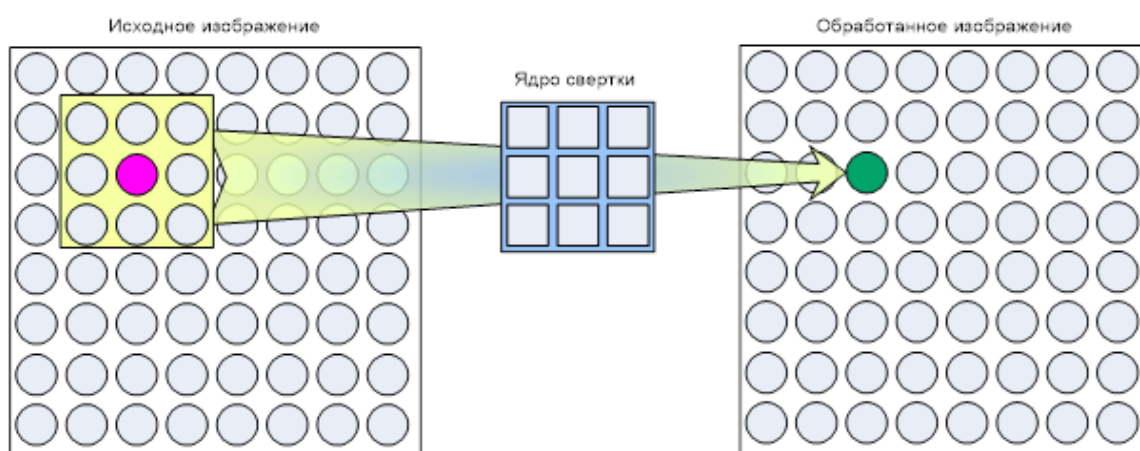
В работе акцент делается на свёрточные нейронные сети, так как результат их работы проще всего оценить, а сфера специализации данного вида НС близка повседневным человеческим задачам.

Свёрточные нейронные сети

Свёрточные нейронные сети являются видом ИНС, но использующие в своей основе операцию свёртки. Математическое определение свёртки выглядит так:

$$(f * g)[m, n] = \sum_{k, l} f[m - k, n - l] * g[k, l]$$

Программно же свёртку можно описать, как применение фильтра (ядра свёртки, матрицы свёртки) ко всем участкам графического изображения.



Нейросетевым значением свёртки является привязка нейронов сети к определённым участкам изображения. То есть, если нейрон, предназначенный на определение признака, видит на своём участке подаваемого изображения искомый признак, он «активируется», то есть подаёт сигнал внутрь СНС.

Свёрточная нейросеть состоит из «слоёв», выполняющих математические операции над поступающими сигналами (сигналами являются как пиксели входного изображения, так и сигналы нейронов верхних уровней сети). Структура слоёв СНС определяется архитектором сети и подбирается, чаще всего, эмпирически учитывая специфику задачи.

Виды слоёв СНС

Почти во всех СНС имеются такие слои, как:

- Слой свёртки (упоминалось выше)
Следует отметить, что к изображению применяется множество фильтров, направленных на определение разнообразных особенностей изображения.
- Слой логистической функции сглаживания значений
У каждого вида функций множество подвидов, все они имеют свои преимущества и недостатки. Применение каждой конкретной функции напрямую зависит от специфики задачи, что делает задачу проектировки СН сложной и наукоёмкой.

Виды:

- Сигмоида
$$f(x) = \frac{1}{1+e^{-x}}$$
, в основном, используется её производная:
$$s(x) = x(1-x)$$
- Гиперболический тангенс
$$f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$
, его производная:
$$s(x) = \frac{4}{(e^x+e^{-x})^2}$$

Редко используется на первых слоях из-за сложности вычисления значений по типу e^{523}
- «Выпрямитель» (ReLU (Rectifier linear units))
$$f(x) = \max(0, x)$$
, наиболее проста для вычислений, чаще всего используется на верхних слоях
- Объединение (Pooling)
Схоже с операцией свёртки. Обычно применяется max-pooling, то есть нейрон, следящий за поступающими сигналами в области, ограниченной квадратом $n \times n$, выдаёт наибольший из видимых им значений. Операция используется для уменьшения изображения.
- Полносвязный слой
Этот слой обычно располагается в конце сети и анализирует дискретные сигналы, полученные от верхних слоёв.

Тренировка НС

Необходимо заметить, что для обучения НС нужна обучающая выборка (существуют способы обучения сетей без выборок, но это не входит в рамки данной исследовательской работы), являющаяся последовательностью «данные на вход НС – данные на выходе НС»

Процесс тренировки нейросети можно ассоциировать с развитием ребёнка. В момент запуска нейросети (первого запуска программы) сеть ничего не знает о своей задаче, её «мозгом» и «глазами» является некоторая группа многомерных матриц (тензоров), которые генерируются случайным образом в момент запуска.

Само обучение является многократно повторяющимся процессом подачи нейросети обучающей выборки, то есть на вход подаются данные, результат обработки которых нам и НС заранее известны. В случае ошибочного предсказания нейросети мы ей «указываем» на её ошибку, нейросеть реагирует на наше замечание и корректирует свои «органы чувств» в соответствии с выбранной логистической функцией. Рассмотрим пример использования СНС

Пример

Задача

В школе устроили конкурс на лучшее сочинение по теме «Важность каллиграфии в жизни человека». В конкурсе приняли участие ученики 7-11 классов, из каждого класса по 17 человек. Каждое сочинение имеет объём не менее 4 листов, причём сочинения написаны очень непонятно, на разбор даже 1 листа требуется много времени.

Решение задачи

Чтобы сохранить зрение и нервы проверяющих, можно использовать свёрточную НС, задача которой будет заключаться в распознавании многостраничных сочинений школьников.

Чтобы СНС могла «читать» сочинения, нужно подавать ей фотографии текста, который будет разбиваться алгоритмом на буквы. Разбиение на буквы — промежуточная задача, решаемая классическим алгоритмом нахождения «центра масс» изображения. Проблема же

заключается в уникальности почерков школьников. Таким образом, создать обучающую выборку (выборкой для такой задачи будут изображения рукописного текста и оцифрованный вариант этого текста), по которой СНС сможет эффективно распознавать текст невозможно. В таком случае, нам потребуется шаблон данных. Пусть обучающая выборка формируется перед решением задачи. Тогда проверяющему потребуется «расшифровать» лишь половину странички сочинения и передать сети изображение этой странички и оцифрованный вариант текста. СНС быстро обучится на небольшой выборке и будет готова к анализу 4-страничного сочинения, чем сэкономит много человеческого времени и ресурсов.

Разбор работы СНС

На вход сети подаются изображения, по алгоритму нахождения центра масс изображения картрика разбивается на маленькие квадраты — буквы. Нейросеть последовательно сворачивает изображение каждой буквы, пропускает через слои сжатия и фильтрации и на выходе получает некоторый вектор признаков (может быть последовательностью цифр/строкой) и сравнивает с соответствующим символом в оцифрованном варианте. Если сеть видит, что вектор признаков совпадает с оцифрованным объектом (буквой), то сеть увеличивает коэффициенты фильтров с помощью, к примеру, производной сигмоиды (нейросетевой смысл этой операции заключается в «подкреплении» нейронов, активировавшихся на нужную картинку), если же сеть видит, что ошиблась, то та же производная помогает «ослабить» нейроны, давшие ложный сигнал. Такая группа операций называется тренировкой

Таким образом, после ~60000 таких тренировок (тренировка распознавания текста происходит быстро из-за маленького размера изображений букв — порядка 20x20 пикселей), занимающих около 1-2 минут, сеть может выдать оцифрованную версию сочинения школьника.

Ещё одно преимущество в том, что сети безразличны орфографические ошибки и проверять грамотность в сочинениях также становится проще.

Архитектура СНС и результаты работы

Мною была спроектирована минимально возможная для выполнения задачи архитектура СНС. Она включает в себя:

Вид слоя	Размерность входной матрицы значений	Размерность выходной матрицы значений
Входное изображение	-	600x600x3
Слой свёртки	600x600x3	200x200x8
Слой ReLU	200x200x8	200x200x8
Слой объединения	200x200x8	90x90x8
Слой свёртки	90x90x8	90x90x32
Слой ReLU	90x90x32	90x90x32
Полносвязный слой	90x90x32	32
Полносвязный слой	32	1

На вход НС подаётся последовательность изображений цветового формата RGB, предварительно составленная архитектором НС. Такую последовательность называют выборкой. От выборки напрямую зависит качество, время и вычислительные ресурсы на тренировку НС. В случае свёрточной НС затраты весьма велики из-за вычислительной сложности процесса свёртки каждого изображения.

Реализация производительного алгоритма свёртки, так как именно он занимает 67% времени обучения сети (результат получен многократными замерами).

Python3:

Консольный интерфейс программы:

10

Вводимое в примере изображение:



Приложение

Полный исходный код СНС распределён по 3 файлам и приведён ниже в 3 текстовых полях соответственно.

ImageWorker.py

```
import numpy as np
from PIL import ImageDraw, Image

def MatrixToPicture(matrix, filename, TRAIN_PATH):
    newPic = Image.new('L', (np.shape(matrix)[0], np.shape(matrix)[1]))
    draw = ImageDraw.Draw(newPic)
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] > 255:
                while matrix[i][j] > 255:
                    matrix[i][j] -= 255
                draw.point((i, j), matrix[i][j])
    newPic.save(TRAIN_PATH+filename, 'JPEG')
    del newPic
    return 1

def PictureToMatrix(picName, TRAIN_PATH):
    trainingArray = []
    data = Image.open(TRAIN_PATH + picName)
    pix = data.load()
    for i in range(data.size[0]):
        trainingArray.append([])
        for j in range(data.size[1]):
            trainingArray[-1].append(pix[i, j])
    return np.array(trainingArray)

def extractLayerFromDim3Matrix(dim3Matrix, layerNumber):
    layer = np.zeros((np.shape(dim3Matrix)[0], np.shape(dim3Matrix)[1]))
    for i in range(np.shape(dim3Matrix)[0]):
        for j in range(np.shape(dim3Matrix)[1]):
            layer[i][j] = dim3Matrix[i][j][layerNumber]
    return layer
```

activations.py

```
import numpy as np
from math import exp

def sigmoid(x, deriv=False):
    if(deriv==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))

def ReLU(tensor, brightness): # rectified linear unit
    #print('ReLU process...')
    for i in range(len(tensor)):
        for j in range(len(tensor[i])):
            for layer in range(len(tensor[i][j])):
                if abs(tensor[i][j][layer]) <= brightness: # General
                    tensor[i][j][layer] = 0
                else:
                    tensor[i][j][layer] = 1
    return tensor

def hiperbolicTang(self, x, deriv=False): # Don't work with 1000+ values
    returnedX = np.zeros((np.shape(x)[0], np.shape(x)[1]))
    if deriv: # берём производную
        for i in range(len(x)):
            for j in range(len(x[i])):
                returnedX[i][j] = (1 + x[i][j]) * (1 - x[i][j])
    print('HipTang returned shape: ', np.shape(returnedX))
    return returnedX

    for i in range(len(x)):
        for j in range(len(x[i])):
            p0 = exp(x[i][j])
            p1 = - exp(-x[i][j])
            p2 = (exp(x[i][j]) + exp(-x[i][j]))
            returnedX[i][j] = (p0 - p1) / p2
    return returnedX
```

Convolution.py

```

import numpy as np
import activations
import ImageWorker as iw
debug = True
def debugPrint(*args):
    global debug
    if debug:
        print(' '.join([str(x) for x in args]))
    else:
        return

class Kernels():

    def __init__(self):
        self.NDM = np.array([[ -1,0,1], [ -2,0,2], [ -1,0,1]])
        self.NWDM = np.array([[ 0,1,2],[ -1,0,1],[ -2,-1,0]])
        self.WDM = np.array([[ 1,2,1],[ 0,0,0],[ -1,-2,-1]])
        self.filters_3_3_Pack = np.array([self.NDM, self.NWDM, self.WDM, ])

    def all(self):
        return self.filters_3_3_Pack

    def filtersNumber(self):
        return np.shape(self.filters_3_3_Pack)[0]

    def kernelsShape(self):
        return np.shape(self.filters_3_3_Pack)

class ConvNeuroNet():
    """
    Логика работы СНС
    """
    kernels = Kernels()
    FILTR_WIDTH = 3
    FILTR_HEIGHT = 3
    #FILTR_DEPTH = 3
    #STEP = 2
    NUMBER_OF_FILTERS = kernels.filtersNumber()
    TRAIN_PATH = 'ConvNeuroNet/neuroTyan/'
    numberOfIteratios = 5

    def __init__(self):
        l2g = []
        l2b = []
        l2_error = []
        l1_error = []
        np.random.seed(3)
        syn4 = np.ones((24))
        syn5 = np.random.random((50,300))
        syn6 = np.random.random((300,2))
        answers = {'g' : 100, 'b' : -100}
        debugPrint('Kernel(s) dims - %s' % (str(self.kernels.kernelsShape())))

```

```

for train in range(self.numberOffiteratios):
    for m in (range(1)):
        if m % 2 == 0:
            prefix = 'g'
        else:
            prefix = 'b'

        rawPicture = iw.PictureToMatrix('tyan'+str(m)+'_'+prefix+'.jpg',
self.TRAIN_PATH)

        debugPrint('ITER'+str(train)+'--IMAGE '+str(m)+' -----')

        syn0 = self.Convolution(rawPicture, m, step=4, layers=3)
        debugPrint('  syn0 shape: ', np.shape(syn0))

        syn1 = activations.sigmoida(self.maxPool2x(syn0, m, 3))
        debugPrint('  syn1 shape: ', np.shape(syn1))

        layer_0 = np.dot(syn1, syn4) # (50,50)
        debugPrint('  layer_0 shape: ', np.shape(layer_0))

        layer_1 = np.dot(layer_0, syn5) # (1,50)
        debugPrint('  layer_1 shape: ', np.shape(layer_1))

        layer_2 = np.dot(layer_1, syn6) # (1)
        debugPrint('  layer_2 shape: ', np.shape(layer_2))
        debugPrint('  layer_2: ', layer_2)

        # как сильно мы ошиблись относительно нужной величины?
        layer_2_error = answers[prefix] - layer_2
        l2_error.append(layer_2_error)

        # Усиливаем вероятное и ослабляем невероятное
        layer_2_delta = layer_2_error * activations.sigmoida(layer_2, deriv=True)

        # как сильно значения layer_1 влияют на ошибки в layer_2?
        layer_1_error = layer_2_delta.dot(syn6.T)

        # в каком направлении нужно двигаться, чтобы прийти к layer_1? Если мы
        # были уверены в предсказании, то сильно менять его не надо
        layer_1_delta = layer_1_error * activations.sigmoida(layer_1, deriv=True)
        #print(layer_2_delta)
        # калибровка
        #syn6 = syn6.dot(layer_2_delta)
        #debugPrint('  Calibrate:\n  syn6 at %s' % syn6.dot(layer_2_delta))

```

```

def Convolution(self, Input, iter_, step, layers):    # Свёртка
debugPrint('Convolution process...')
_width, _height, _depth = np.shape(Input)[0], np.shape(Input)[1], np.shape(Input)[2]
# Свёрнутая пикча занимает (picW - (filtrW - 1))x(picH - (filtrH - 1))
convMatrixes = np.zeros((( _width - self.FILTR_WIDTH) // step + 1, (_height - self.FILTR_HEIGHT)
// step + 1, self.NUMBER_OF_FILTERS * layers))
debugPrint('  Compressing pic %s to size %s' % (( _width, _height, _depth), np.shape(convMatrixes)))
filters = self.kernels.all()
lastLayerCounter = 0
# Проверим валидность пикчи для фильтра
if (_width >= self.FILTR_WIDTH + 1) and (_height >= self.FILTR_HEIGHT + 1):
    #np.set_printoptions(threshold=np.nan)
    # Для каждого фильтра
    for iii in (range(self.NUMBER_OF_FILTERS)):    #tqdm
        # Идём по пикче
        for layer in range(layers):    # Для каждого канала
            for w in (range(self.FILTR_WIDTH, _width, step)):
                for h in range(self.FILTR_HEIGHT, _height, step):
                    for wf in range(self.FILTR_WIDTH):
                        for hf in range(self.FILTR_HEIGHT):
                            Coord1 = (h-self.FILTR_HEIGHT)//step
                            Coord2 = (w-self.FILTR_WIDTH)//step
                            InputPixVal = Input[w-wf][h-hf][layer]
                            KernelPixVal = filters[iii][wf][hf]
                            convMatrixes[Coord2][Coord1]

[lastLayerCounter] += InputPixVal * KernelPixVal
                            lastLayerCounter += 1
            convMatrixes = activations.ReLU(convMatrixes, 210)
        #for iii in range(self.NUMBER_OF_FILTERS * layers):
            #currentMatrix = iw.extractLayerFromDim3Matrix(convMatrixes, iii)
        return convMatrixes
# Очень зависит от настроек свёртки
def maxPool2x(self, tensor, iter_, step):
_width, _height, _maps = np.shape(tensor)[0], np.shape(tensor)[1], np.shape(tensor)[2]
pooledMatrixes = np.zeros((( _width - self.FILTR_WIDTH) // step + 1, (_height -
self.FILTR_HEIGHT) // step + 1, _maps))
debugPrint('  Pooling tensor %s to size %s' % (( _width, _height, _maps), np.shape(pooledMatrixes)))
if (_width >= self.FILTR_WIDTH + 1) and (_height >= self.FILTR_HEIGHT + 1):
    for featureMap in (range(_maps)):    #tqdm
        for w in (range(self.FILTR_WIDTH, _width, step)):
            for h in range(self.FILTR_HEIGHT, _height, step):
                localMaximum = 0
                for wf in range(self.FILTR_WIDTH):
                    for hf in range(self.FILTR_HEIGHT):
                        Coord1 = (h-self.FILTR_HEIGHT)//step
                        Coord2 = (w-self.FILTR_WIDTH)//step
                        InputPixVal = tensor[w-wf][h-hf][featureMap]
                        if InputPixVal > localMaximum:
                            localMaximum = InputPixVal
                pooledMatrixes[Coord2][Coord1][featureMap] = localMaximum
    return pooledMatrixes

if __name__ == '__main__':
    ConvNeuroNet()

```


Список литературы

[1] У. Мак-Каллок и У. Питтс. Логическое исчисление идей, относящихся к нервной активности, 1943г

1. habrahabr.ru/post/309508/
2. christianperone.com/2015/08/convolutional-neural-networks-and-feature-extraction-with-python/
3. tutorialspoint.com/dip/robinson_compass_mask.htm
4. machinelearning.ru/wiki/images/1/1b/DL16_lecture_3.pdf
5. <http://mechanoid.kiev.ua/ml-lenet.html>