МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)
Чистопольский филиал «Восток»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

РАСПОЗНАВАНИЕ ОБРАЗОВ

Индекс по учебному плану: **Б1.В.ДВ.08.01**
Направление подготовки: **09.03.01 Информатика и вычислительная техника**
Квалификация: **Бакалавр**
Профиль подготовки: **Вычислительные машины, комплексы, системы и сети**
Типы задач профессиональной деятельности: **проектный, производственно-технологический**

Рекомендовано УМК ЧФ КНИТУ-КАИ

Чистополь
2023 г.

Лабораторная работа №1

Фильтрация изображений

Теоретические сведения

Фильтр «Яркость/контрастность» Для изменения яркости на N процентов используется следующая формула:

I = I + N • 128 / 100 (1)

где I — соответственно R, G, B каналы каждой точки изображения.

Уменьшение контрастности на N процентов:

I = (I • (100 — N) + 128 • N) / 100 (2)

Увеличение контрастности на N процентов:

I = (I • 100 — 128 • N) / (100 — N) (3)

Если новое I не попадает в диапазон 0..255 — то его следует урезать.

Фильтр «Цветовой баланс»

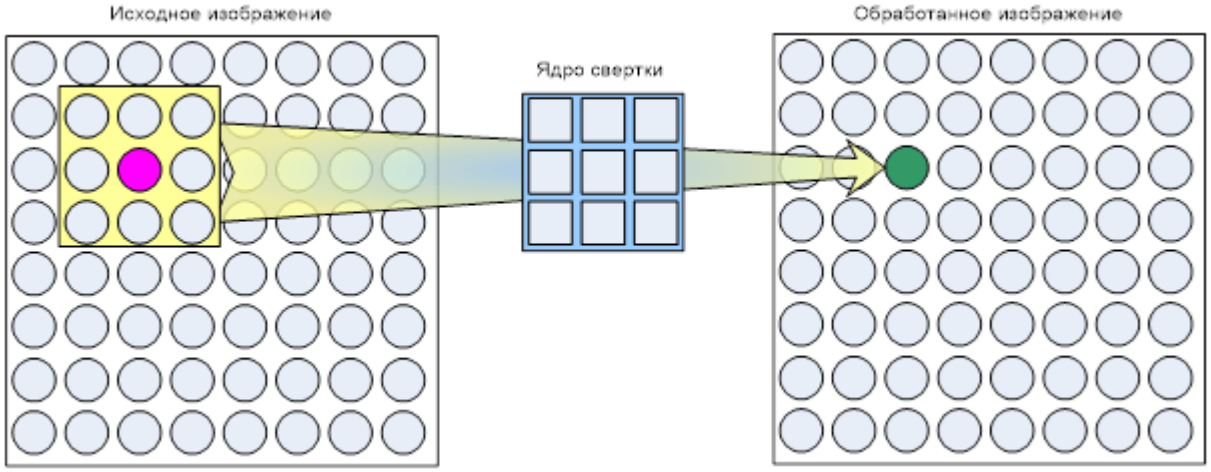
Для изменения цветового баланса по одному из каналов R, G, B на N процентов следует вычислить новое значение цветового канала по формуле:

I = I + N • 128 / 100, где I – это R, G или B каждой точки изображения. **(4)**

Если новое I не попадает в диапазон 0..255 — то его следует урезать.

Фильтры «Повысить резкость» и «Размыть»

Эти фильтры реализуются на основе ядра свертки. Элемент изображения получает новое значение на основе группы элементов, примыкающих к данному. Область примыкания есть квадратная матрица, размерность которой совпадает с размером выбранного ядра свертки, и центром в обрабатываемом элементе.



Ядро свертки представляет собой матрицу размером 3x3, 5x5, 7x7 и т.д., на которой определена некоторая функция. Ядро свертки называется окном, а заданная на нем функция – весовой или функцией окна. Каждому элементу окна соответствует число, называемое весовым множителем. Совокупность всех весовых множителей и составляет весовую функцию. Нечетные размеры окна необходимы для однозначного определения центрального элемента.

По сути, ядро свертки является фильтром, который позволяет усилить или ослабить компоненты изображения. Фильтрация осуществляется перемещением окна фильтра по изображению. Весовая функция в процессе перемещения остается неизменной. В каждом положении окна происходит операция свертки – линейная комбинация значений элементов изображения:

$$\begin{vmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{vmatrix} \begin{vmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{vmatrix} \sum_{i=1}^{9} p_i K_i = p_5'$$

где p_i - элементы области примыкания,

k_i - весовые множители,

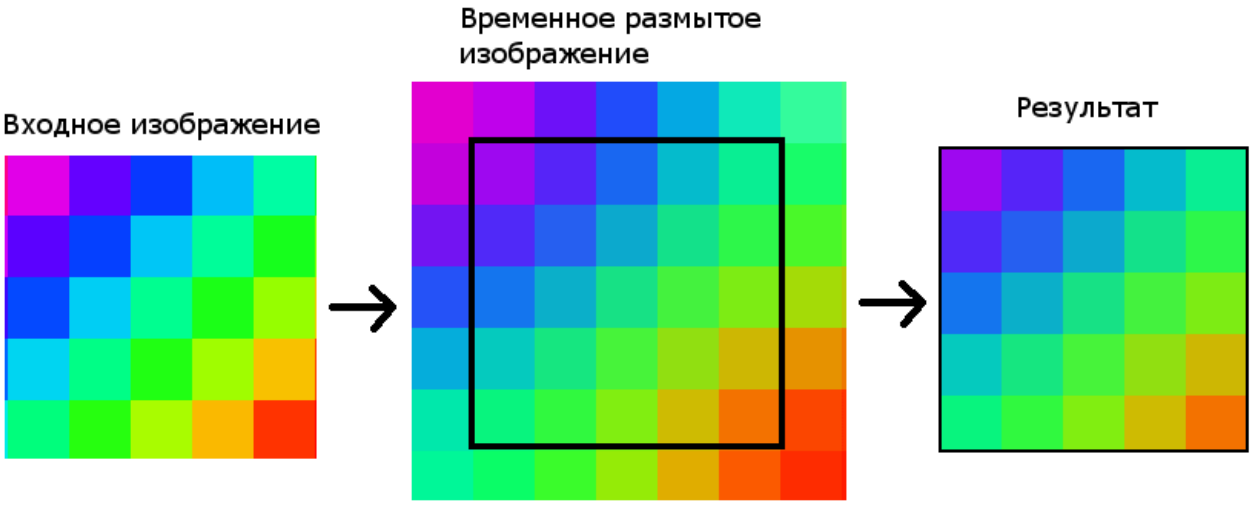
p_5' - новое значение пикселя.

При каждом положении окна весовая функция поэлементно умножается на значение соответствующих пикселей исходного изображения и произведения суммируются. Полученная сумма называется откликом фильтра и присваивается тому пикселю нового изображения, который соответствует положению центра окна.

Результат обработки пикселя записывается в соответствующую ячейку временной матрицы такого же размера, как и исходное изображение. Запись в отдельную временную матрицу необходима для того, чтобы исключить влияние уже обработанных пикселей на еще не обработанные.

Обязательно следует упомянуть о граничных условиях. Например, у верхнего левого пикселя не существует «соседей» слева и сверху, следовательно, нам не на что умножать коэффициенты матрицы.

Для решения этой проблемы требуется создание промежуточного изображения. Идея в том, чтобы создавать временное изображение с размерами (width + 2 • gap / 2, height + 2 • gap / 2, где width и height – ширина и высота фильтруемого изображения, а gap – размерность матрицы свертки). В центр изображения копируется входная картинка, а края заполняются крайними пикселями изображения. Размытие применяется к промежуточному буферу, а потом из него извлекается результат.



При использовании алгоритма увеличения резкости подчеркиваются различия между цветами смежных пикселей и выделяются незаметные детали. В ядре резкости центральный коэффициент больше 1, а окружен он отрицательными числами, сумма которых на единицу меньше центрального коэффициента. Таким образом, увеличивается любой существующий контраст

между цветом пикселя и цветами его соседей. При обработке каждого пикселя в изображении используется ядро резкости размерами 3×3:

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

При использовании алгоритма размытия в изображении перераспределяются цвета и смягчаются резкие границы. Ядро сглаживания состоит из совокупности коэффициентов, каждый из которых меньше 1, а их сумма составляет 1. Это означает, что после фильтрации каждый пиксель поглотит что-то из цветов соседей, но полная яркость изображения останется неизменной. При обработке изображения используется следующее ядро свертки:

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Описание алгоритмов

Изменение яркости

В функцию изменения яркости передается значение цвета текущего пикселя, текущее положение ползунка (poz) изменения яркости и максимальное значение, которое он может принимать (length).

1) Вычисляется количество процентов N = (100 / lenght) * poz.

2) По формуле (1) для каждого цветового канала рассчитывается новое значение.

3) Контролируем переполнение переменных:

ЕСЛИ I < 0, **ТО** I = 0,

ЕСЛИ I > 255, **ТО** I = 255,

где I — соответственно R, G, B пикселя.

4) return значения цвета пикселя.

Изменение контрастности

В функцию изменения контрастности передается значение цвета текущего пикселя, текущее положение ползунка (poz) изменения контрастности и максимальное значение, которое он может принимать (length).

1) Вычисляется количество процентов N = (100 / lenght) * poz.

2) **ЕСЛИ** N < 0, то новое значение цвета каждого канала рассчитывается (N берется по модулю) по формуле (2)

ИНАЧЕ по формуле (3).

3) Контролируем переполнение переменных:

ЕСЛИ I < 0, **ТО** I = 0,

ЕСЛИ I > 255, **ТО** I = 255,

где I — соответственно R, G, B пикселя.

4) return значения цвета пикселя.

Изменение цветового баланса

В функцию изменения цветового баланса по соответствующему каналу передается значение цвета текущего пикселя, текущее положение ползунка (poz) изменения цветового баланса и максимальное значение, которое он может принимать (length).

1) Вычисляется количество процентов N = (100 / lenght) * poz.

2) По формуле (4) для соответствующего цветового канала рассчитывается новое значение.

3) Контролируем переполнение переменных:

ЕСЛИ I < 0, **ТО** I = 0,

ЕСЛИ I > 255, **ТО** I = 255,

где I — соответствующий (R, G или B) канал цвета пикселя.

4) return значения цвета пикселя.

Повышение резкости и размытие

В функцию передаются значения ширины и высоты редактируемого изображения, матрица пикселей этого изображения, размерность матрицы свертки и сама матрица.

1) Создание временной матрицы расширенного изображения и ее заполнение.

2) Применение ядра свертки для каждого пикселя, при этом контролируем переполнение переменных (0…255 для каждого канала). Новые значения цветов пикселей заносятся в отдельную матрицу (newpixel)

3) Возвращение функцией матрицы newpixel.

Описание структуры ПО

Для построения пользовательского интерфейса применяется графиче-ская подсистема Windows Forms. Во время выполнения программы используются классы Form1, Form2, Form3, BrightnessContrast, ColorBalance, Filter, которые включают следующие методы:

Brightness – метод, изменяющий яркость текущей точки.

Contrast – метод, изменяющий контрастность текущей точки.

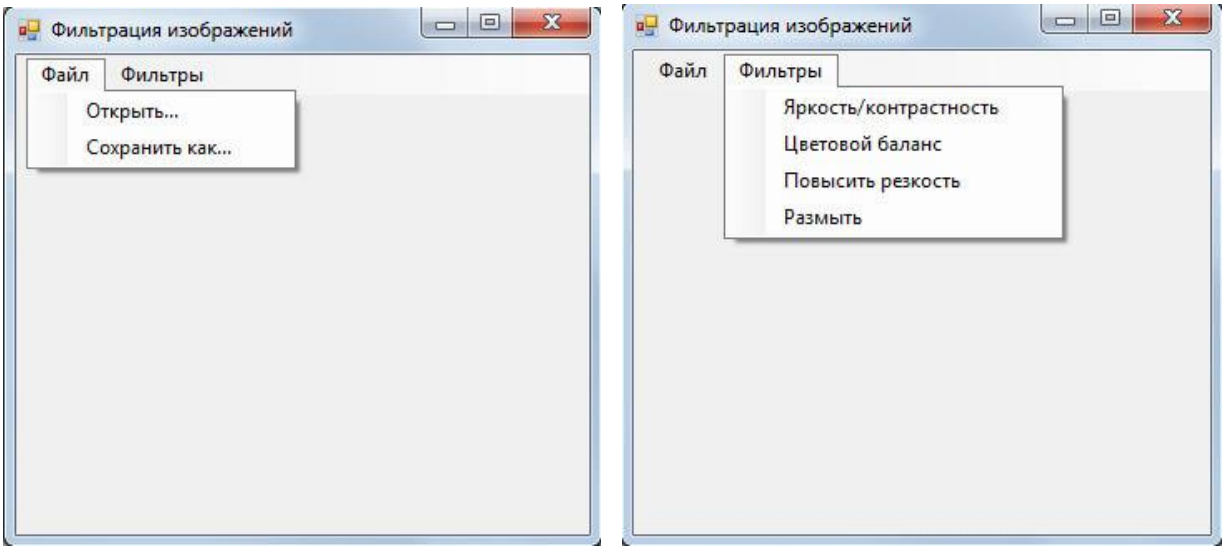
ColorBalance_R – метод, изменяющий цветовой баланс точки по каналу R.

ColorBalance_G – метод, изменяющий цветовой баланс точки по каналу G.

ColorBalance_B – метод, изменяющий цветовой баланс точки по каналу B.

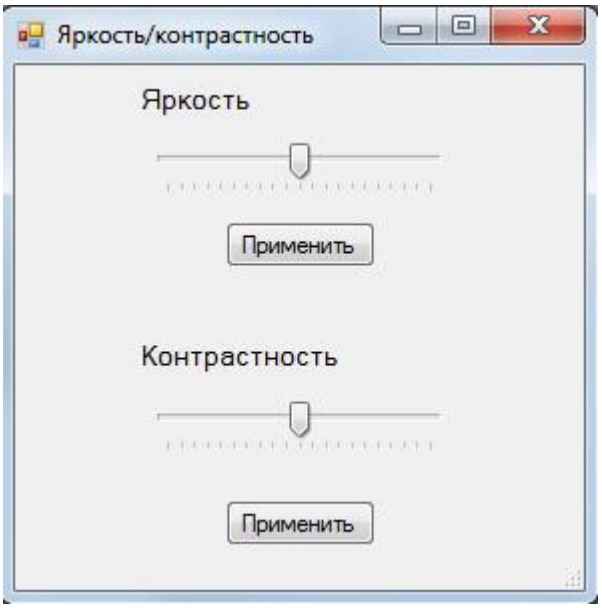
matrix_filtration – метод, выполняющий фильтрацию изображения на основе ядра свертки.

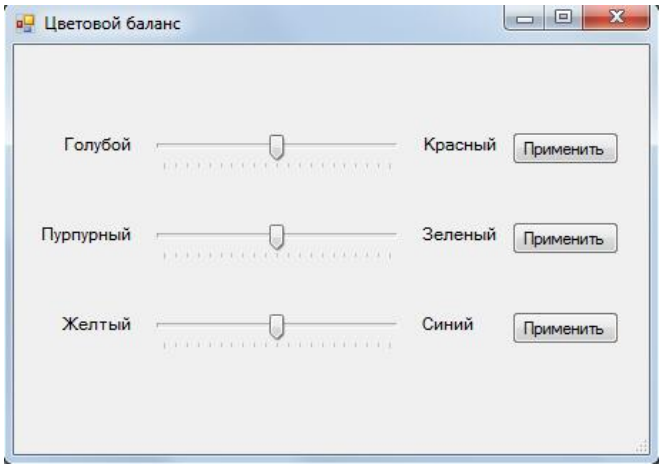
Описание структуры пользовательского интерфейса



На рисунках выше представлено главное меню программы. Во вкладке «Файл» находятся кнопки для открытия и сохранения изображения. Во вкладке «Фильтры» представлены инструменты для фильтрации загруженного

изображения: «Яркость/контрастность», «Цветовой баланс», «Повысить резкость», «Размыть». При вызове первых двух открываются новые окна, в которых находятся слайдеры для изменения соответствующих компонентов фильтруемого изображения и кнопки «Применить», для сохранения изменений:

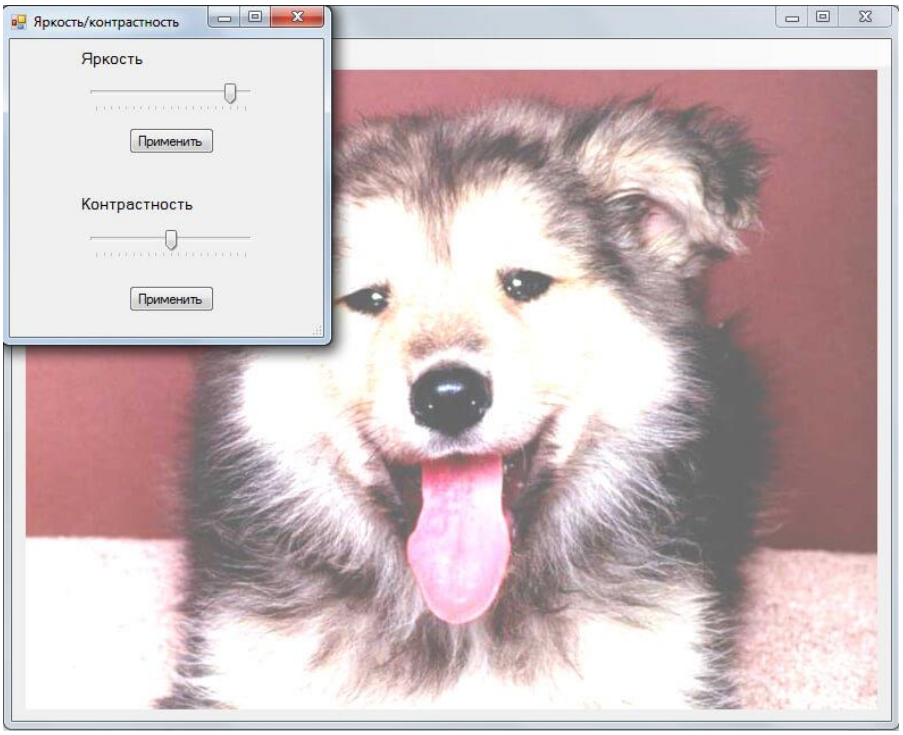




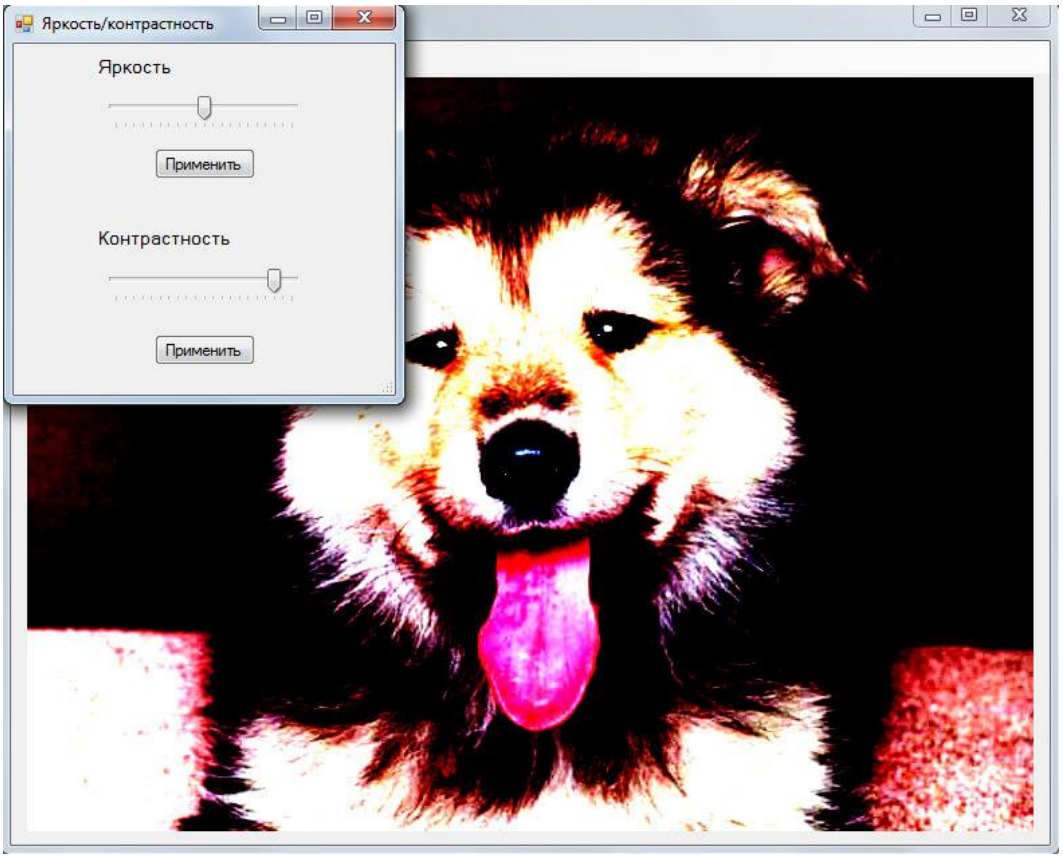
Демонстрация работы программы

Исходное изображение:

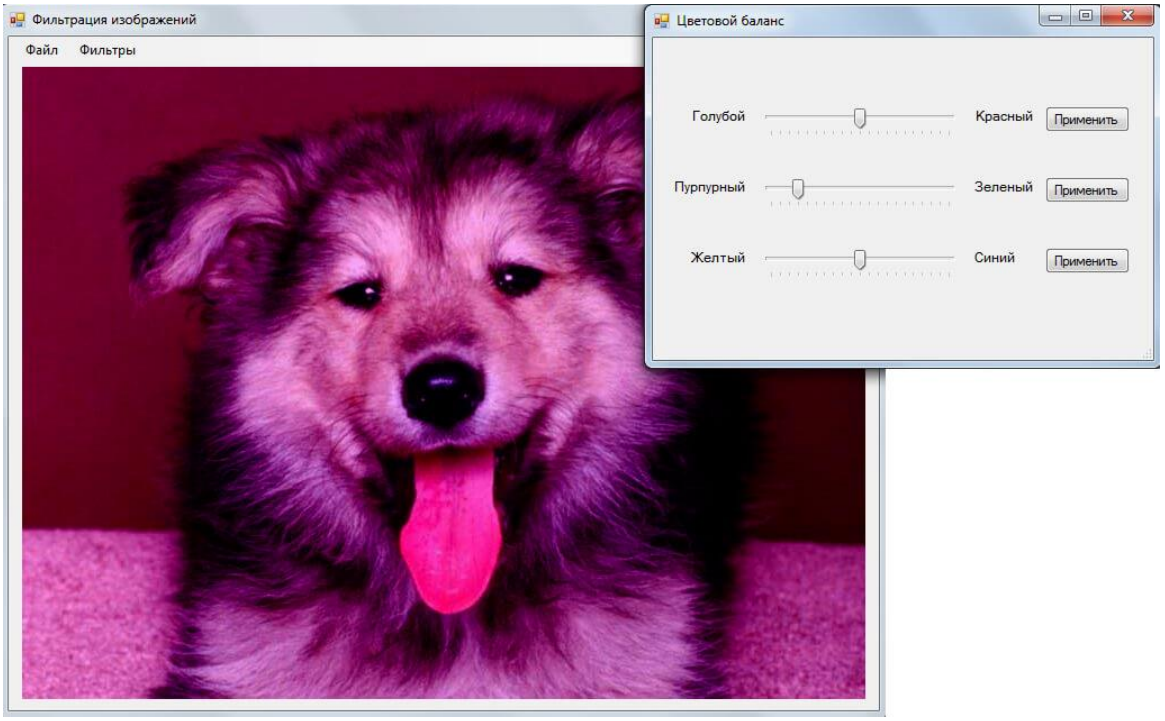
Изменение яркости:



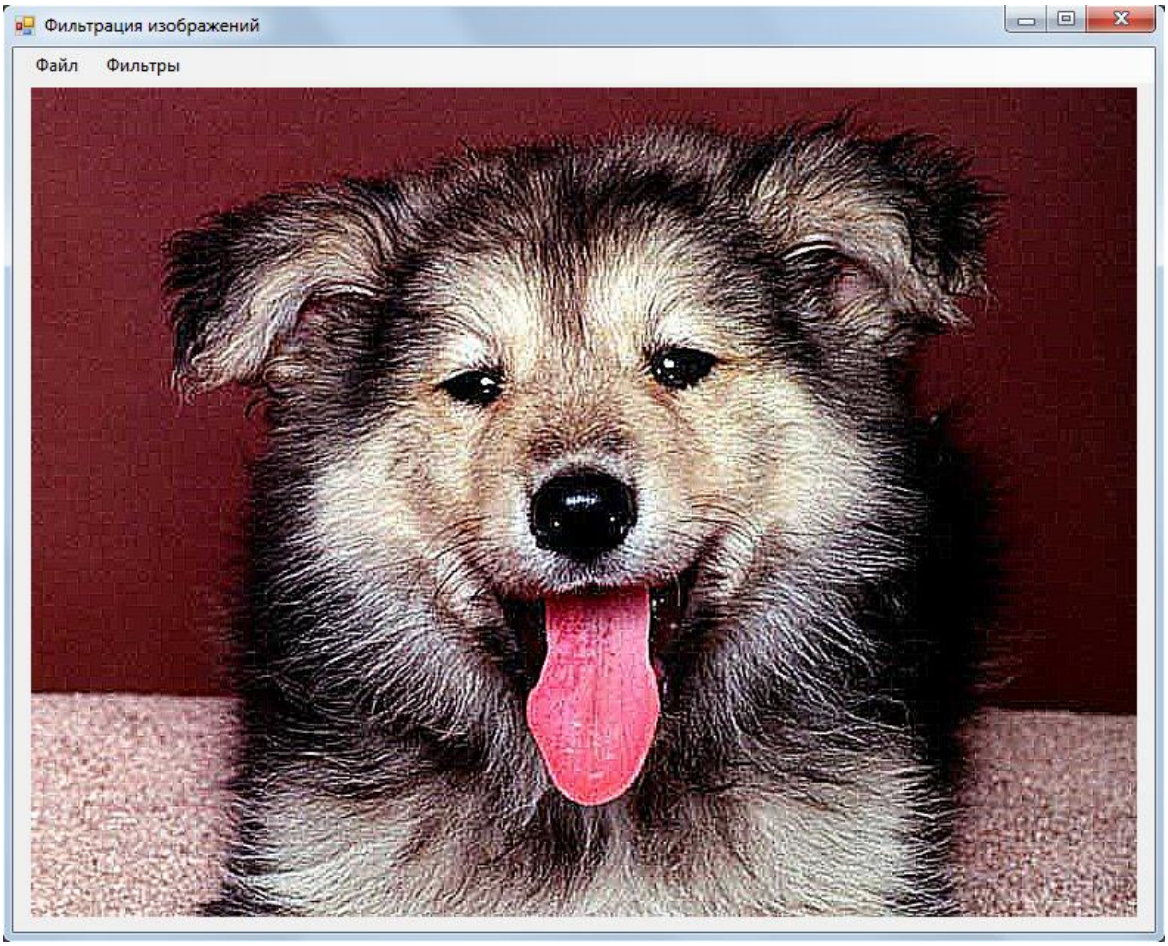
Изменение контрастности:



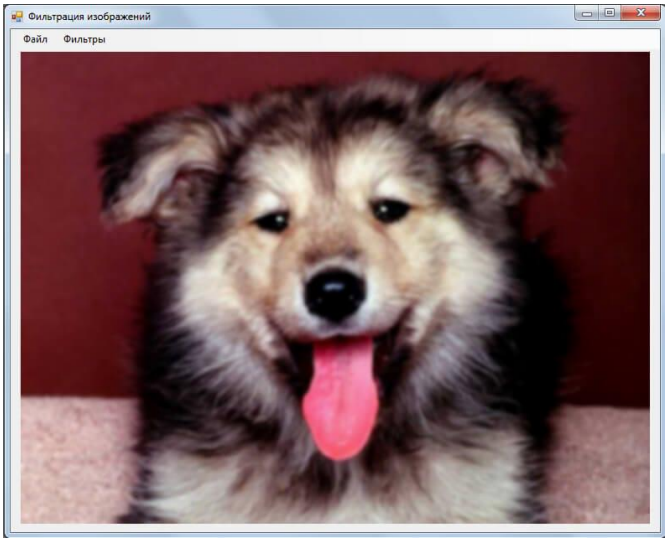
Изменение цветового баланса:



Повышение резкости:



Размытие изображения:



Лабораторная работа №2
Фрактальное сжатие

Цель задания

Необходимо написать программу, которая умеет сжимать изображение и распаковывать его, используя классический алгоритм фрактального сжатия, описанный на лекциях.

Задача - достичь как можно большего сжатия при использовании сетки фиксированного размера (4x4, 8x8 и 16x16 по параметру) и как можно лучшего соотношения степени сжатия к качеству для варианта с квадродеревом.

Мерой потерь будет служить PSNR - метрика обратно логарифмически пропорциональная среднеквадратичному отклонению пикселов.

Замер будет проводиться в двух номинациях:

- Сжатие с использованием сетки 4x4, 8x8 и 16x16 (допустим ТОЛЬКО классический алгоритм, который был рассмотрен на лекции).
- Сжатие с использованием квадродерева и указанием качества.

Для повышения соотношения степени сжатия и качества рекомендуется использовать следующие методы:

1. Использование перевода в YUV и дискретизацию в 4 раза цветовых компонент U & V.
2. Использование квадродерева - 16x16 - 8x8 - 4x4 с дроблением очередного блока, если его приближение (например), хуже среднего значения для всех остальных блоков. Другой вариант - просто по порогу, пропорционально пересчитываемому для каждого уровня квадродерева.
3. Использование арифметического сжатия для дожатия афинных коэффициентов (в первую очередь сдвига и контрастности).

Первая часть задания

Программа должна уметь получать на вход BMP файл и по опции -c - сжимать его, по опции -d распаковывать его.

Чтение BMP из файла проще возложить на систему - посмотрите в MSDN "LoadImage":HBITMAP image = (HBITMAP) LoadImage (0,"image.bmp",IMAGE_BITMAP,0,0,LR_LOADFROMFILE);

Пример простой программы, которая работает с BMP также входит в комплект VS.

По умолчанию должен использовать фрактальный алгоритм разбиением изображения на сетки 4x4, 8x8 и 16x16.

Программа должна быть консольным приложением и запускаться так:

```
compress -c image.bmp out_file.fr -s 8
compress -d out_file.fr out_image.bmp
```

Т.е. тестовый файл test.bat вида:

```
compress -c image1.bmp out_file1.fr -s 8
compress -d out_file1.fr out_image1.bmp
compress -c image2.bmp out_file2.fr -s 8
compress -d out_file2.fr out_image2.bmp
compress -c image3.bmp out_file3.fr -s 8
compress -d out_file3.fr out_image3.bmp
compress -c image4.bmp out_file4.fr -s 8
compress -d out_file4.fr out_image4.bmp
compress -c image5.bmp out_file5.fr -s 8
compress -d out_file5.fr out_image5.bmp
```

(обратите внимание - минус перед параметром означает, что это опция)

должен сжимать 5 тестовых изображений (image1.bmp, ... image5.bmp). Для оценки степени сжатия используются размеры каждого сжатого изображения, а для оценки качества - разница между исходным и распакованным изображениями.

Вторая часть задания

Для повышения соотношения качества к размеру (и набора дополнительных баллов) нужно реализовать работу с квадродеревом и управление качеством.

С ним ваша программа должна уметь выполнять тестовый файл test_qual.bat вида:

```
compress -c image1.bmp out_file1.fr -q 10
compress -d out_file1.fr out_image1.bmp
compress -c image2.bmp out_file2.fr -q 10
compress -d out_file2.fr out_image2.bmp
compress -c image3.bmp out_file3.fr -q 10
compress -d out_file3.fr out_image3.bmp
compress -c image4.bmp out_file4.fr -q 10
compress -d out_file4.fr out_image4.bmp
compress -c image5.bmp out_file5.fr -q 10
compress -d out_file5.fr out_image5.bmp
```

При этом q - параметр качества должен изменяться от 1 до 100 (включительно).

Для тестов будут использованы 4-6 значений, условно 10, 30, 50, 70, 85, 100.

Лабораторная работа №3
Моделирование выбранной функциональной зависимости

Для обучения нейронной сети прогнозированию используется выборка известных членов ряда. После обучения сеть должна прогнозировать временной ряд на упреждающий промежуток времени.

Задание 1. Моделирование заданной функциональной зависимости

Пусть дана математическая функция $y = -0{,}1 * \sin(3x) + 0.5$. Для получения дискретного ряда с целью обучения нейронной сети прогнозированию значений данной функции необходимо протабулировать ее с некоторым шагом.

1. Протабулировать функцию $y = -0{,}1 * \sin(3x) + 0.5$ с шагом 0,1 на отрезке [-20;20]. По результатам этого задать функцию $y = -0{,}1 * \sin(3x) + 0.5$ на отрезке [-20;20] в дискретной (табличной) форме.
2. По дискретной форме, созданной на шаге (1), создать обучающую выборку для нейронной сети, включающей порядка 400 примеров вычисления функции $y = -0{,}1 * \sin(3x) + 0.5$. Обучающую выборку создавать для условий прогнозирования функции методом скользящего окна с шириной p=4.
3. В пакете Neural Network Wizard создать и обучить нейронную сеть с одним скрытым слоем из 11 нейронов для прогнозирования методом скользящего окна функции $y = -0{,}1 * \sin(3x) + 0.5$.
4. С помощью обученной нейронной сети спрогнозировать значения функции $y = -0{,}1 * \sin(3x) + 0.5$ на отрезке [20,23] с шагом 0.1. Выяснить абсолютную погрешность прогнозирования для каждой точки отрезка [20,23]. Результаты исследования представить в отчете в следующей форме.

Точка отрезка	Спрогнозированное значение функции	Реальное значение	Абсолютная погрешность

	у	функции у	прогнозирования
20.1			
20.2			
20.3			
…..			
23			

Лабораторная работа №4

Распознавание образов на основе нейронных сетей

Теоретический материал

Пусть десятичные цифры 0-9 отображаются на бинарной матрице размерностью 8x8 в виде, представленном на рис. 1.

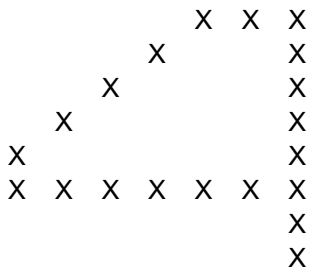


Рис 1.

Задача нейронной сети заключается в распознавании таких цифр в условиях помех. Помехами в этом случае могут являться случайные появления или пропадания отдельных пикселов в бинарной матрице, изменения формы.

Для распознавания цифр можно обучить нейронную сеть, состоящую из 64 входных нейронов и 10 выходных.

Входами нейронной сети будут являться элементы 0 или 1, в зависимости от того, черная или белая клетка используется в соответствующей битовой матрице.

Каждый выходной нейрон будет ответственен за какую либо цифру от 0 до 9. Например, первый – за 1, второй – за 2, … десятый – за 0.

Значениями выходов нейронной сети также будут являться числа 0 или 1. Активность i-ого выходного нейрона означает похожесть входного образа на i-ую цифру.

Для создания и обучения нейронной сети предлагается использовать пакет Neural Network Wizard.

Работа с пакетом Neural Network Wizard

Neural Network Wizard является пакетом для создания моделей искусственных многослойных нейронных сетей прямого распространения.

Для создания модели нейронной сети необходимо выполнить следующие шаги:

1. Создать файл с обучающей выборкой, в котором должны быть представлены представительные примеры решения задачи (вход-выход).

2. Выбрать файл с обучающей выборкой в Neural Network Wizard (рис. 2.)

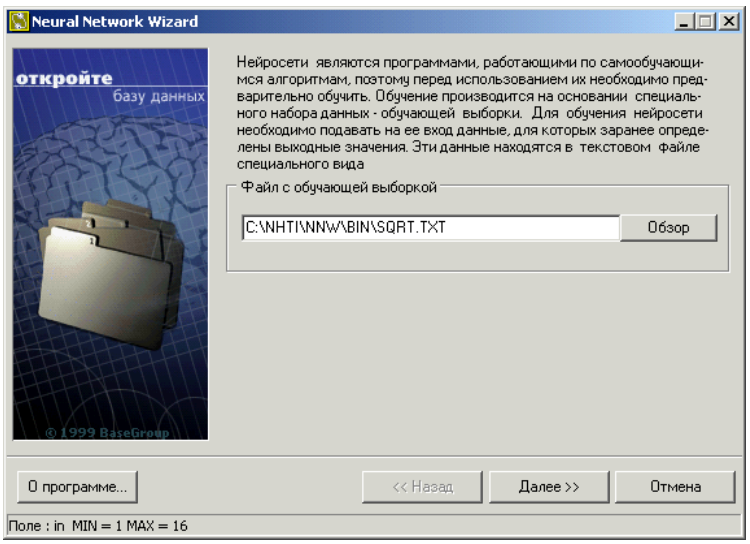


Рис. 2. Выбор файла с обучающей выборкой

3. Задать входные и выходные элементы нейронной сети (рис. 3.)

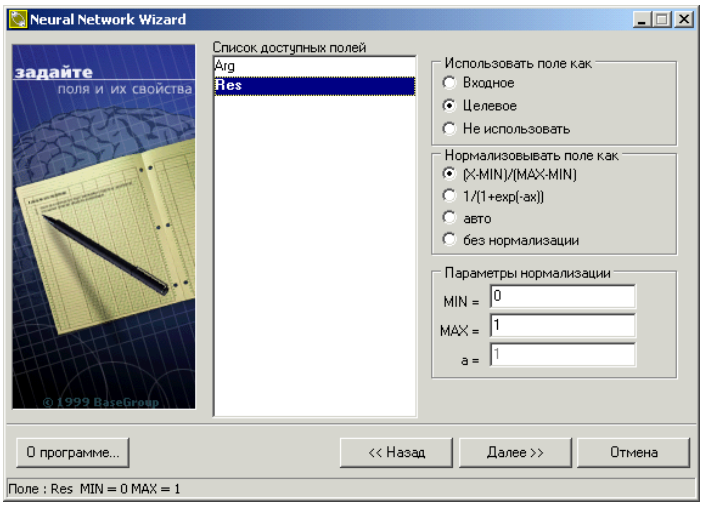


Рис. 3. Определение входов и выходов нейронной сети.

4. Указать количество слоев, количество нейронов в скрытых слоях и вид функции активации (рис. 4).

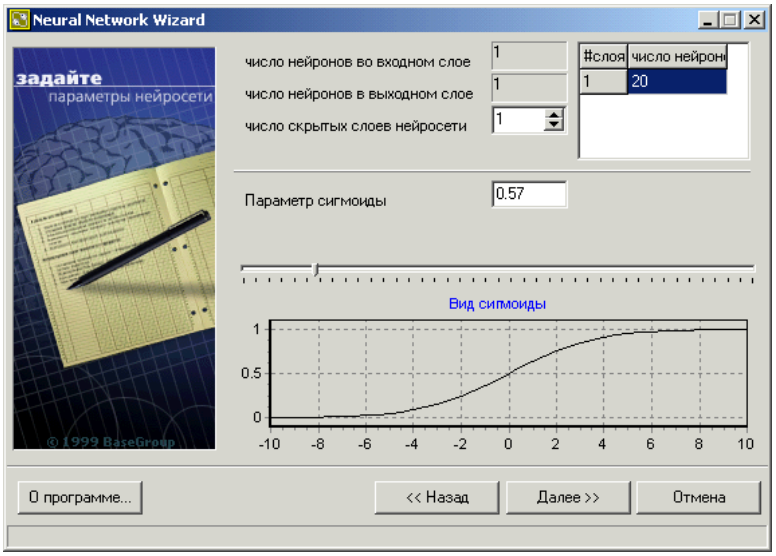


Рис. 4. Выбор количества скрытых слоев, количества нейронов в скрытых слоях и вид функции активации

5. Задать параметры обучения (рис. 5).

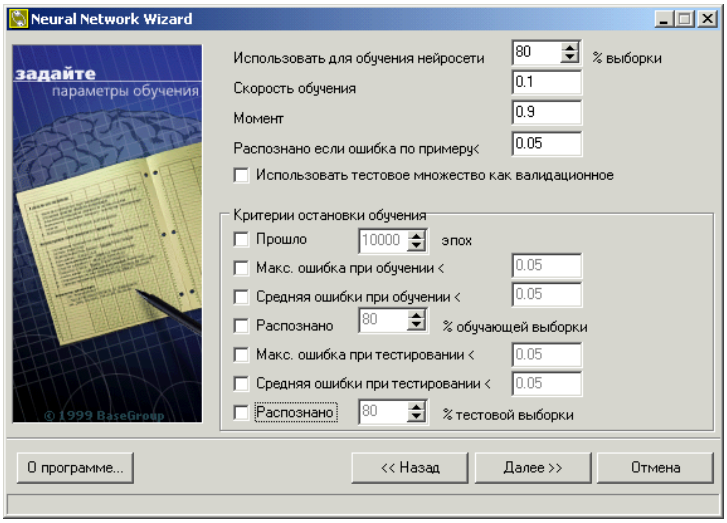


Рис. 5. Параметры обучения нейронной сети.

6. Обучить нейронную сеть.

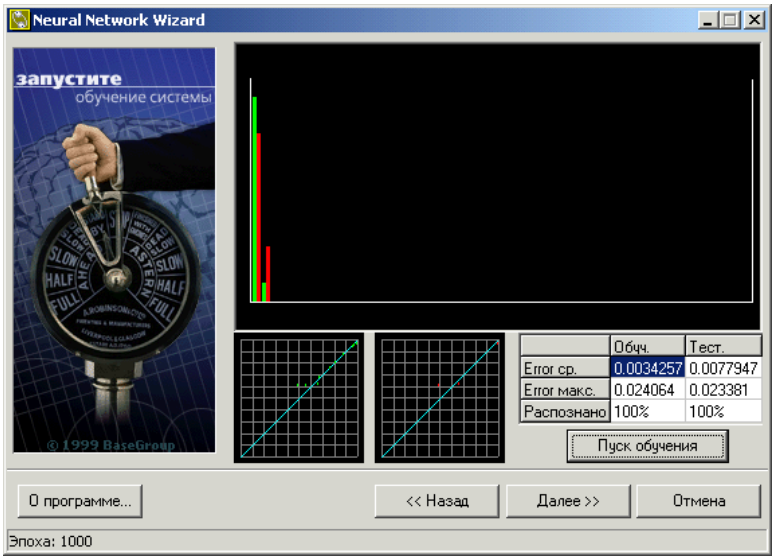


Рис. 6. Обучение нейронной сети

7. Подача на вход нейронной сети необходимых значений и нахождение результата (рис. 7).

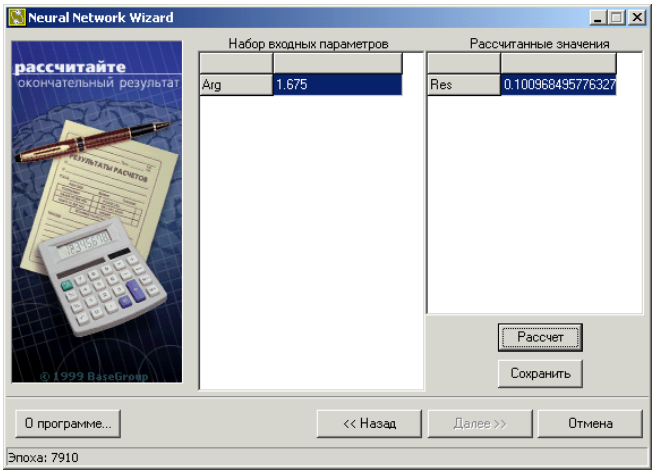


Рис. 7. Работа с нейронной сетью.

Задание на лабораторную работу

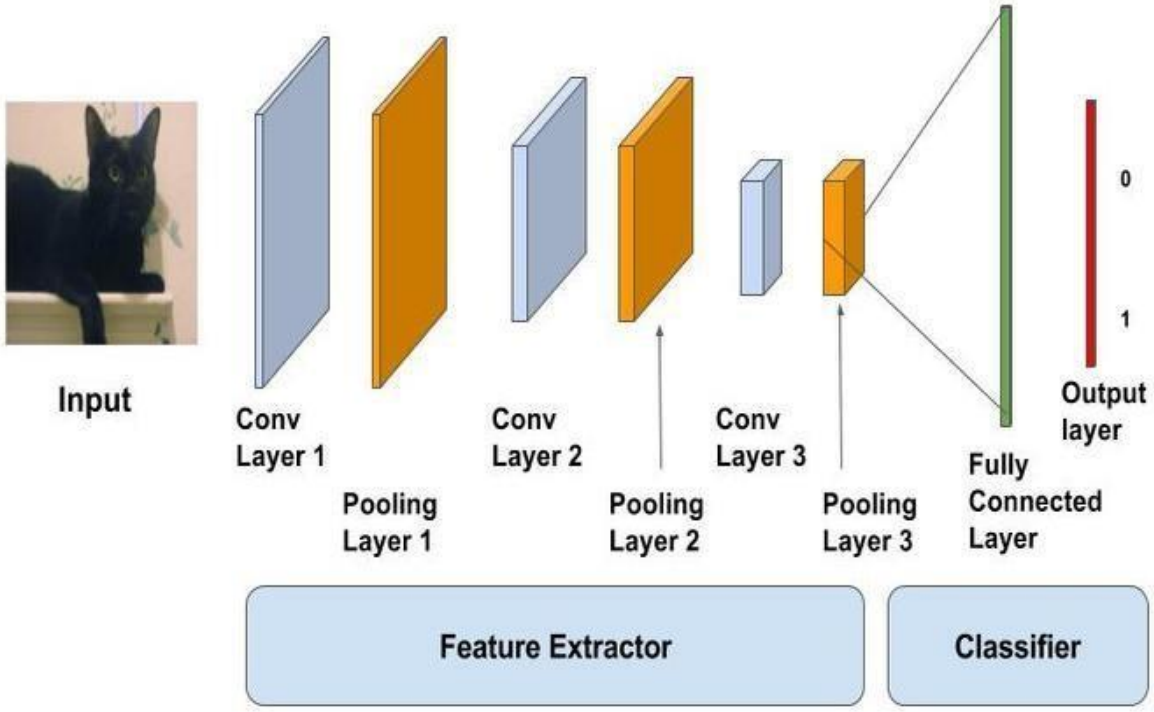
1. Подготовить 20 битовых масок, демонстрирующих примеры написания цифр – по 2 маски для каждой цифры. Данные маски внести в отчет по лабораторной работе.
2. Подготовить обучающую выборку для обучения нейронной сети, распознающей цифры. Обучающая выборка должна включать 64 входа и 10 выходов.
3. Обучить в пакете Neural Network Wizard нейронную сеть для распознавания цифр.

4. Подготовить по 2 тестовых примера для каждой цифры. Тестовые примеры должны включать шумовые элементы (появление или пропадание пикселей в битовой маске).
5. Исследовать работу нейронной сети с зашумленными образами. Насколько корректно она распознает зашумленные цифры? Тестовые битовые маски и точность их распознавания внести в отчет по лабораторной работе.

Лабораторная работа №4

КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ НЕЙРОННЫХ СЕТЕЙ

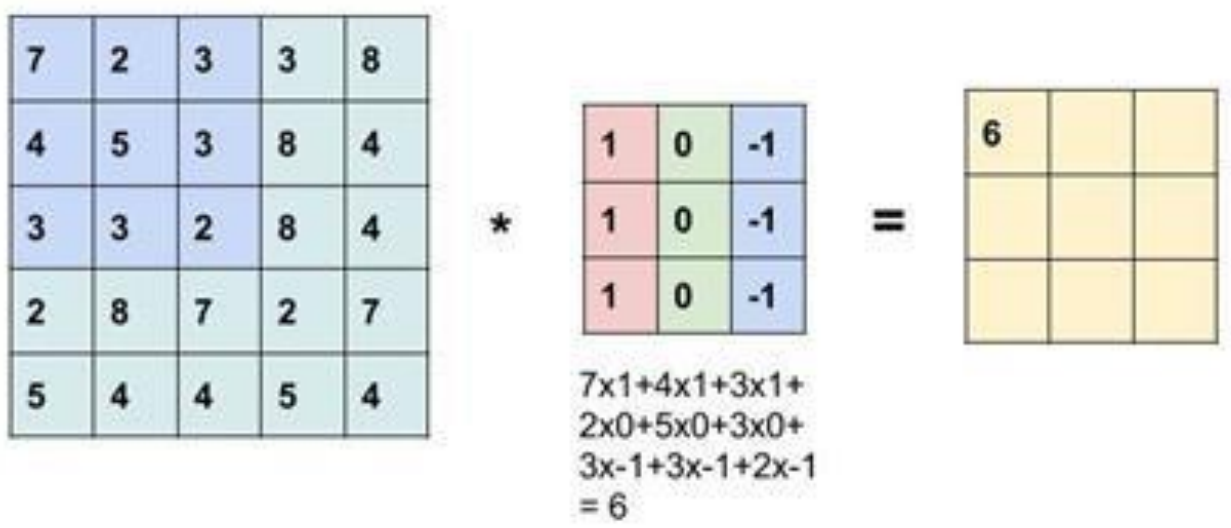
Сверточные нейронные сети являются одной из форм многослойных нейронных сетей. Здесь приведена схема типичного CNN. Первая часть состоит из слоев свертки и максимального пула, которые выступают в качестве экстрактора признаков. Вторая часть состоит из полносвязного слоя, который выполняет нелинейные преобразования извлеченных признаков и действует как классификатор.



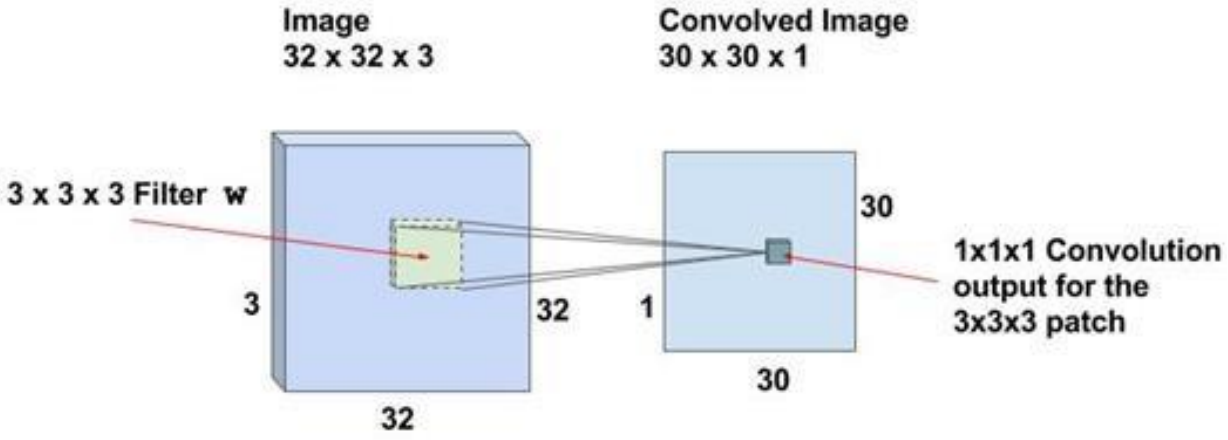
На приведенной выше диаграмме вход подается в сеть последовательных слоев **Conv**, **Pool** и **Dense**. Выходной сигнал может быть слоем **softmax**, указывающим, есть ли кошка или что-то еще. Также, а качестве выходного может быть использован сигмоидный слой, на выходе которого будет вероятность того, что изображение будет кошкой. Рассмотрим слои более подробно.

Сверточный слой можно рассматривать как глаза сверточной нейронной сети. Нейроны в этом слое ищут определенные особенности. Свертку можно рассматривать как взвешенную сумму между двумя сигналами или функциями.

Пример операции свертки на матрице размером 5 × 5 с ядром размером 3 × 3 показан ниже. Ядро свертки скользит по всей матрице для получения карты активации.



Предположим, что входное изображение имеет размер 32x32x3, т.е. это трехмерный массив глубины 3. Любой фильтр свертки, который определяем на этом слое, должен иметь глубину, равную глубине ввода. Поэтому можем выбрать фильтры свертки глубины 3 (например, 3x3x3 или 5x5x3 или 7x7x3 и т. Д.). Выберем фильтр свертки размера 3x3x3, т.е сверточное ядро будет кубом вместо квадрата.



Если сможем выполнить операцию свертки, сдвинув фильтр 3x3x3 на все изображение размером 32x32x3, то получим изображение с разрешением 30x30x1. Это связано с тем, что операция свертки невозможна для полосы шириной 2 пикселя вокруг изображения. Фильтр всегда находится внутри изображения и

поэтому 1 пиксель удаляется от левой, правой, верхней и нижней части изображения.

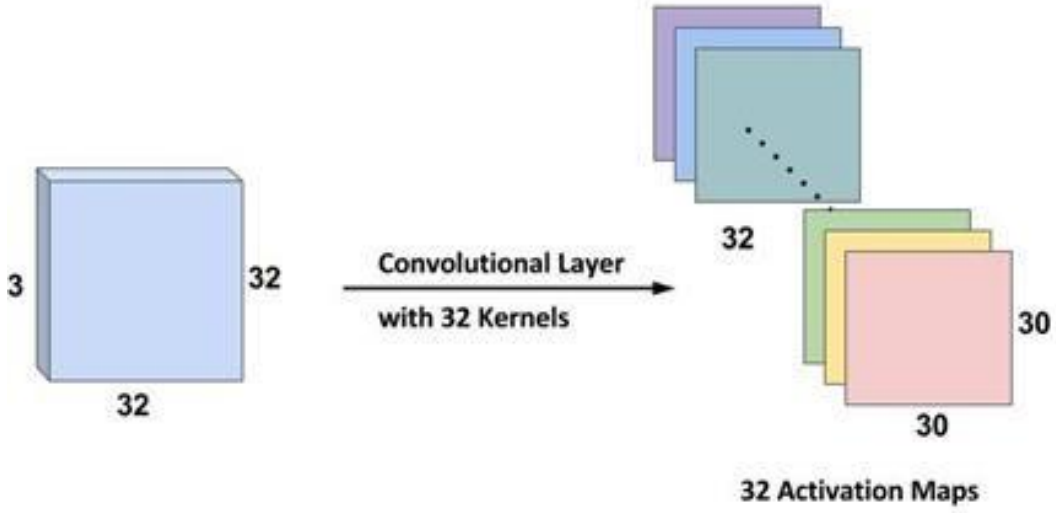
Для входного изображения 32x32x3 и размера фильтра 3x3x3 у нас есть 30x30x1 местоположения, и для каждого местоположения существует нейрон. Тогда выходы 30x30x1 или активации всех нейронов называются картами активации. Карта активации одного уровня служит входом для следующего слоя.

В нашем примере есть 30 × 30 = 900 нейронов, потому что есть много мест, где может применяться фильтр 3x3x3. В отличие от традиционных нейронных сетей, где веса и смещения нейронов независимы друг от друга, в случае сверточных нейронных сетей, нейроны, соответствующие одному фильтру в слое, имеют одинаковые веса и смещения. В приведенном выше случае сдвигаем окно на 1 пиксель за раз. Требуется также можем сдвинуть окно более чем на 1 пиксель. Это число называется шагом.

Как правило, используют более одного фильтра в одном слое свертки. Если используем 32 фильтра, у нас будет карта активации размером 30x30x32.

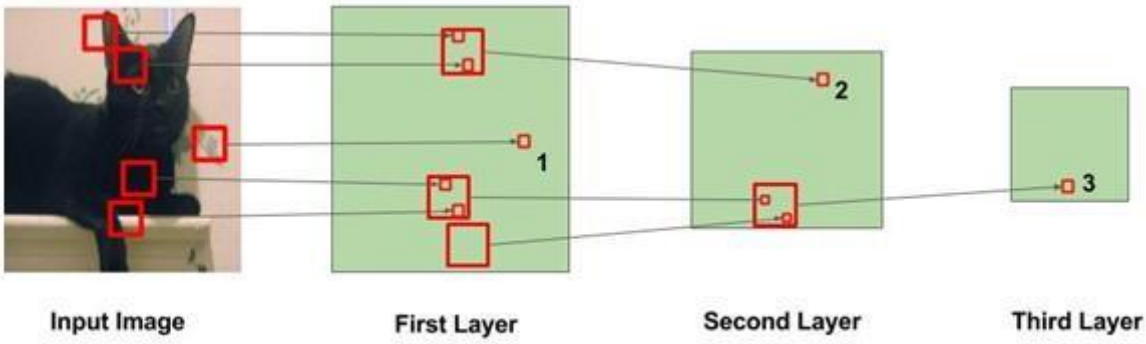
Обратите внимание, что все нейроны, связанные с одним и тем же фильтром, имеют одинаковые веса и смещения. Таким образом, количество весов при использовании 32 фильтров - это 3x3x3x32 = 288, а число смещений - 32.

На картинке показаны 32 карты активации, полученные от применения сверточных ядер.



Как можно видеть, после каждой свертки результат уменьшается по размеру (так как в этом случае переходим от 32 × 32 до 30 × 30). Для удобства стандартная практика заключается в том, чтобы накладывать нули на границу входного слоя таким образом, чтобы выход был такого же размера, как и входной. Итак, в этом примере, если добавим дополнение размером 1 по обе стороны от входного слоя, размер выходного уровня будет 32x32x32, что упростит реализацию.

Рассмотрим, как сверточные нейронные сети анализируют изображения.



На приведенном выше рисунке большие квадраты указывают область, в которой выполняется операция свертки, а малые квадраты указывают выход операции, которая является просто числом. Следует отметить следующие замечания:

1. В первом слое квадрат, обозначенный 1, получается из области изображения, на которой окрашены листья.

2. Во втором слое квадрат с меткой 2 получается из большего квадрата в

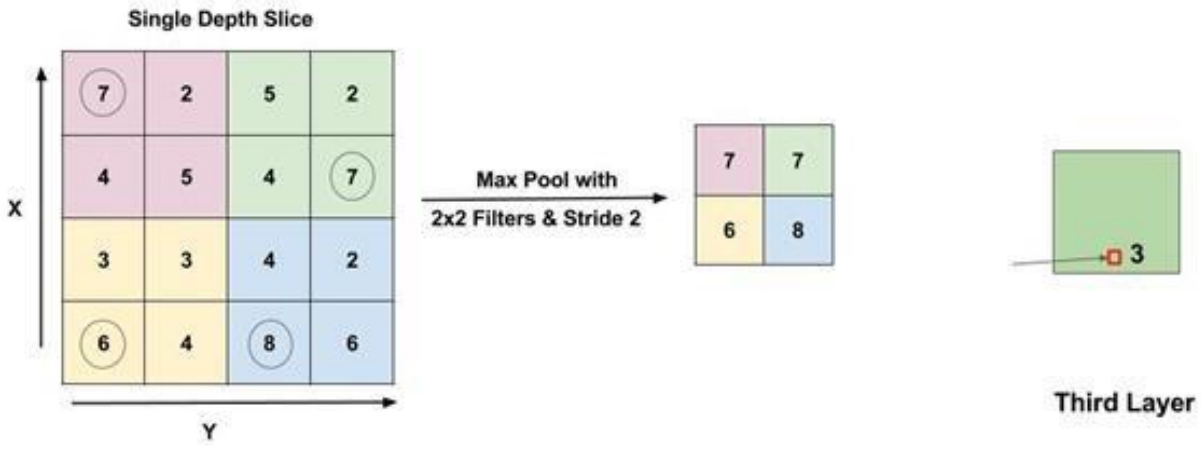
первом слое. Числа в этом квадрате получены из нескольких областей из входного изображения. В частности, вся площадь вокруг левого уха кошки отвечает за значение на квадрате, отмеченном 2.

3. Аналогично, в третьем слое этот каскадный эффект приводит к тому, что квадрат, обозначенный 3, получается из большой области вокруг области ноги.

Из сказанного выше можно сказать, что начальные слои анализируют более мелкие области изображения и, следовательно, могут обнаруживать только простые признаки, такие как края / углы и т. д. По мере того как идем глубже в сеть, нейроны получают информацию из более крупных частей изображения и от различных других нейронов. Таким образом, нейроны на более поздних слоях могут изучить более сложные функции, такие как глаза, ноги, и т.д

Слой пулинга в основном используется сразу после сверточного слоя для уменьшения пространственного размера (только по ширине и высоте, а не по глубине). Это уменьшает количество параметров, поэтому вычисление уменьшается. Использование меньшего количества параметров позволяет избежать переобучения. **Переобучение** - это условие, когда обученная модель отлично работает с данными обучения, но не очень хорошо работает в тестовых данных.

Наиболее распространенной формой пулинга является максимальный пулинг, в котором берем фильтр размера и применяем максимальную операцию **max** с определенной частью изображения.



На рисунке показан максимальный пул с размером фильтра 2 × 2 и шагом

2. Выход представляет собой максимальное значение в области 2 × 2, показанной с использованием окруженных цифр. Наиболее распространенная операция пулинга выполняется с фильтром размером 2 × 2 с шагом 2. Это существенно уменьшает размер ввода на половину.

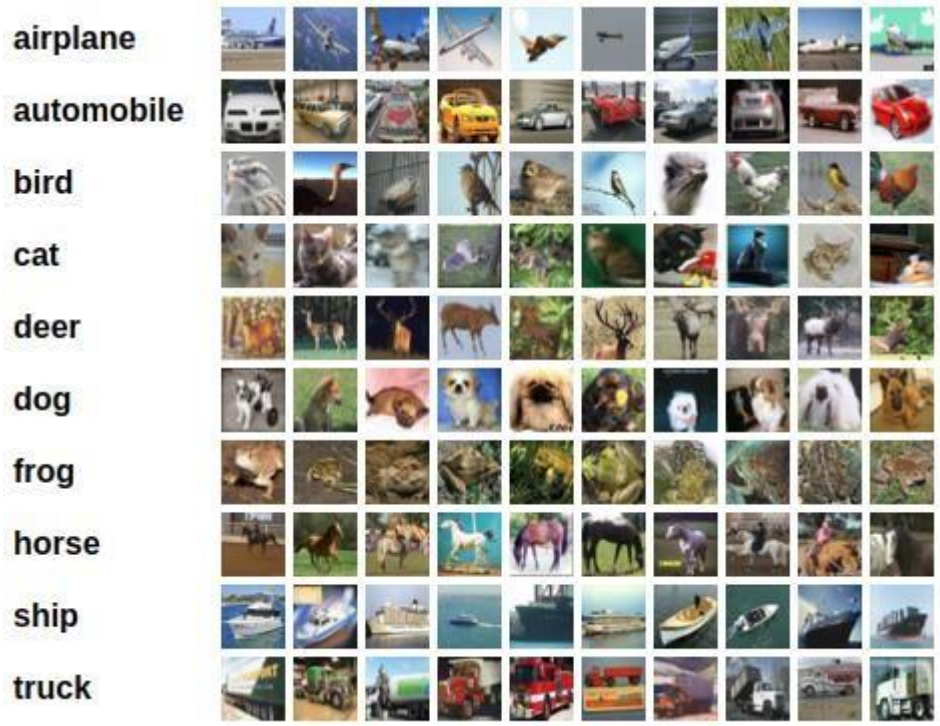
Набор данных - CIFAR10

Набор данных CIFAR10 поставляется вместе с Keras. Он имеет 50 000 учебных образов и 10000 тестовых изображений 10 классов, таких как самолеты, автомобили, птицы, кошки, олени, собаки, лягушки, лошади, корабли и грузовики. С помощью следующего програмного кода можно осуществить загрузку и подготовку данных CIFAR10 для дальнейшей обработки с помощью нейронных сетей:

```
from_future_import print_function import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten from keras.layers import Conv2D, MaxPooling2D
import os batch_size = 32
num_classes = 10
epochs = 100
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models') model_name = 'keras_cifar10_trained_model.h5'
# Разделяем данные на обучающий и тестовый наборы: (x_train, y_train), (x_test, y_test) = cifar10.load_data() print('x_train shape:', x_train.shape) print(x_train.shape[0], 'train samples') print(x_test.shape[0], 'test samples')
# Преобразование векторов классов в двоичные матрицы y_train = keras.utils.to_categorical(y_train, num_classes) y_test =
```

keras.utils.to_categorical(y_test, num_classes)

Изображения имеют размер 32 × 32. На рисунке приведены несколько примеров.



Сверточная нейронная сеть будет состоять из сверточных слоев, и слоев **MaxPooling**. Также включим **Dropout** слой для избежания переобучения. На выходе сети добавим полносвязный слой (**Dense**), за которым следует слой **softmax**. Здесь приведен программный код создания структуры модели.

В приведенном выше коде используем 6 сверточных слоев и 1 полносвязный слой. Сначала в модель добавляем сверточные слои с 32 фильтрами с размером окна 3 × 3. Далее добавляем сверточный слой с 64 фильтрами. За каждым слоем добавлен слой максимального пуллинга с размером окна 2 × 2. Также добавлены слои **Dropout** с коэффициентами 0,25 и 0.5 для того чтобы не произошло переобучение сети. В заключительных строках добавляем плотный слой **Dense**, который выполняет классификацию среди 10 классов с использованием функции активации **softmax.**

model = Sequential()

model.add(Conv2D(32,(3,3),padding='same', input_shape=x_train.shape[1:]))
model.add(Activation('relu'))

model.add(Conv2D(32, (3, 3))) model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.25)) model.add(Conv2D(64, (3, 3), padding='same')) model.add(Activation('relu')) model.add(Conv2D(64, (3, 3))) model.add(Activation('relu')) model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.25))

model.add(Flatten()) model.add(Dense(512)) model.add(Activation('relu')) model.add(Dropout(0.5)) model.add(Dense(num_classes)) model.add(Activation('softmax')) model.summary()

Если выведем информацию о структуре модели, увидим следующую таблицу с подробным описанием каждого слоя.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
activation_1 (Activation)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
activation_2 (Activation)	(None, 30, 30, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 15, 15, 32)	0

dropout_1 (Dropout)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 15, 15, 64)	18496
activation_3 (Activation)	(None, 15, 15, 64)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	36928
activation_4 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 64)	0
dropout_2 (Dropout)	(None, 6, 6, 64)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 512)	1180160
activation_5 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130
activation_6 (Activation)	(None, 10)	0

Total params: 1,250,858
Trainable params: 1,250,858

Обучение сети

Поскольку это проблема классификации по 10 классам, будем использовать категорическую потерю энтропии и использовать оптимизатор **RMSProp** для обучения сети.

 # initiate RMSprop optimizer

opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

 # Let's train the model using RMSprop model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy']) x_train = x_train.astype('float32')

x_test = x_test.astype('float32') x_train /= 255

x_test /= 255

//Запустим его на количество эпох **epochs**.

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test), shuffle=True)

Сохраняем модель и веса

if not os.path.isdir(save_dir): os.makedirs(save_dir)

model_path = os.path.join(save_dir, model_name) model.save(model_path)

print('Saved trained model at %s ' % model_path)

Проверяем точность работы модели.

scores = model.evaluate(x_test, y_test, verbose=1) print('Test loss:', scores[0])

print('Test accuracy:', scores[1])

Лабораторная работа №5
Распознавание рукописных цифр с использованием нейронных сетей

MNIST - это набор данных, разработанный Янном ЛеКуном, Коринной Кортес и Кристофер Бургес для оценки моделей машинного обучения по проблеме классификации рукописных цифр. Набор данных был построен из ряда отсканированных наборов документов, доступных в Национальном институте стандартов и технологий (NIST). Изображения цифр были взяты из множества отсканированных документов, нормированных по размеру и по центру. Это делает его отличным набором данных для оценки моделей, позволяя разработчику сосредоточиться на механизме обучения с очень небольшой очисткой данных или необходимой подготовкой.

Каждое изображение представляет собой квадрат размером 28 на 28 пикселей (всего 784 пикселя). В этом наборе 60 000 изображений используются для обучения модели, и для ее тестирования используется отдельный набор из 000 изображений. Это задача распознавания 10 цифр (от 0 до 9) или классификация на 10 классов.

Библиотека глубокого обучения Keras предоставляет удобный метод **mnist.load_data()** для загрузки набора данных MNIST. Набор данных загружается автоматически при первом вызове этой функции и сохраняется в вашем домашнем каталоге в ~ / .keras /datasets /mnist.pkl.gz в виде файла 15 МБ. Это очень удобно для разработки и тестирования моделей глубокого обучения.

Чтобы продемонстрировать, насколько легко загружать набор данных MNIST, мы сначала напишем небольшой скрипт для загрузки и визуализации первых четырех изображений в наборе учебных материалов.

from keras.datasets import mnist import matplotlib.pyplot as plt

 # load (downloaded if needed) the MNIST dataset (X_train, y_train), (X_test, y_test) = mnist.load_data()

 # plot 4 images as gray scale plt.subplot(221)

plt.imshow(X_train[0], cmap=plt.get_cmap('gray')) plt.subplot(222)

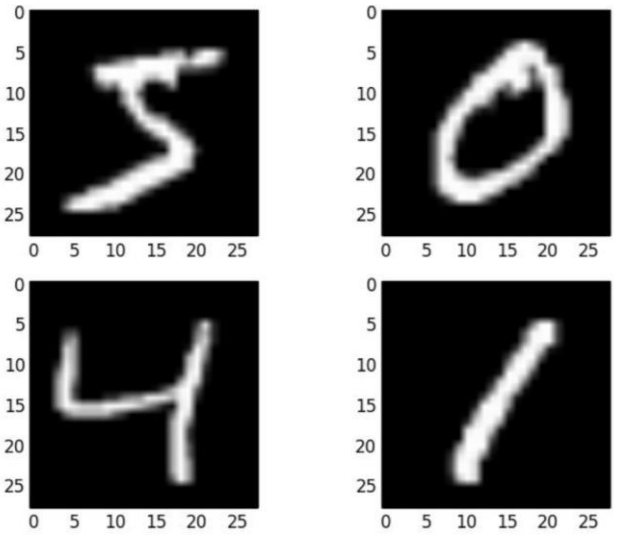
plt.imshow(X_train[1], cmap=plt.get_cmap('gray')) plt.subplot(223)

plt.imshow(X_train[2], cmap=plt.get_cmap('gray')) plt.subplot(224)

plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))

 # show the plot plt.show()

Запустив приведенный выше пример, вы должны увидеть изображение ниже.



Базовая модель с многослойным перцептроном

Чтобы понять действительно ли нам нужна сложная модель, такая как сверточная нейронная сеть сначала попробуем использовать очень простую модель нейронной сети с одним скрытым слоем. Мы будем использовать эту

сеть как основу для сравнения более сложных сверточных моделей нейронных сетей.

Начнем с импорта классов и функций, которые нам понадобятся.

import numpy

from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.utils import np_utils

Учебный набор данных структурирован как трехмерный массив. Чтобы подготовить данные, сперва мы представим изображения в виде одномерных массивов (так как считаем каждый пиксель отдельным входным признаком). В этом случае изображения размером 28 × 28 будут преобразованы в массивы, содержащие 784 элементов.

Мы можем сделать это преобразование, используя функцию **reshape()** библиотеки **NumPy**. Для уменьшения потребления оперативной памяти преобразуем точность значений пикселей в 32.

*(X_train, y_train), (X_test, y_test) = mnist.load_data() num_pixels = X_train.shape[1] * X_train.shape[2]*

X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32') X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')

Значения пикселей заданы в оттенках серого со значениями от 0 до 255. Для эффективного обучения нейронных сетей практически всегда рекомендуется выполнять некоторое масштабирование входных значений. Мы можем нормализовать значения пикселей в диапазоне 0 и 1, разделив каждое значение на максимальные значения 255.

X_train = X_train / 255 X_test = X_test / 255

Выходная переменная представляет собой целое число от 0 до 9, т.к. это задача классификации с несколькими классами. Хорошей практикой является

использование кодирования значений класса преобразованием вектора целых чисел класса в двоичную матрицу.

Мы можем легко сделать это, используя встроенную вспомогательную функцию **np_utils.to_categorical ()** в Keras.

y_train = np_utils.to_categorical(y_train) y_test = np_utils.to_categorical(y_test) num_classes = y_test.shape[1]

Теперь создадим простую модель однослойной нейронной сети и определим ее в функции.

def baseline_model():

 # create model model = Sequential()

model.add(Dense(num_pixels,input_dim=num_pixels, kernel_initializer='normal', activation='relu'))

model.add(Dense(num_classes,kernel_initializer='normal', activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy'])

return model

Модель представляет собой простую нейронную сеть с одним скрытым слоем с таким же количеством нейронов, что и количество входов (784). В скрытом слое используем полулинейную функцию активации **relu**.

На выходном слое используется функция активации **softmax** для преобразования выходов в вероятностные значения и позволяет выбрать один класс из 10 в качестве выходного значения модели. Теперь нам осталось только определить функцию потерь, алгоритм оптимизации и метрики, которые мы будет собирать. В задачах с вероятностной классификацией, в качестве функции потерь лучше всего использовать не квадратичную ошибку, а перекрестную энтропию. Потери будут меньше для вероятностных задач (например, с логистической/**softmax** функцией для выходного слоя), в основном из-за того, что данная функция предназначена для максимизации уверенности модели в правильном определении класса, и ее не заботит распределение вероятностей

попадания образца в другие классы. Используемый алгоритм оптимизации будет напоминать какую-то форму алгоритма градиентного спуска, отличие будет лишь в том, как выбирается скорость обучения. В нашем случае мы будем использовать оптимизатор Адама, который обычно показывает хорошую производительность. Так как наши классы сбалансированы (количество рукописных цифр, принадлежащих каждому классу, одинаково), подходящей метрикой будет точность (**accuracy**) — доля входных данных, отнесенных к правильному классу.

Теперь мы можем обучить и оценить качество обученности модели.

model = baseline_model()

model.fit(X_train, y_train, validation_data=(X_test,y_test), epochs=10, batch_size=200, verbose=2)

*scores = model.evaluate(X_test, y_test, verbose=0) print("Baseline Error: %.2f%%" % (100-scores[1]*100))*

Модель подходит 10 эпох обучения, при каждом обновлении весов используется 200 изображений. Тестовые данные которые используются в качестве набора данных валидации, позволяют видеть качество распознавания модели по мере ее обучения. Значение **verbose** =2 используется для уменьшения вывода на одну строку для каждой учебной эпохи. Наконец, тестовый набор данных используется для оценки модели и печатается ошибка классификации.

Train on 60000 samples, validate on 10000 samples

Epoch 1/10 - 21s - loss: 0.2781 - acc: 0.9213 - val_loss: 0.1443 - val_acc: 0.9585
Epoch 2/10 - 21s - loss: 0.1100 - acc: 0.9686 - val_loss: 0.0943 - val_acc: 0.9709
Epoch 3/10 - 18s - loss: 0.0709 - acc: 0.9798 - val_loss: 0.0809 - val_acc: 0.9739
Epoch 4/10 - 18s - loss: 0.0511 - acc: 0.9855 - val_loss: 0.0679 - val_acc: 0.9781
Epoch 5/10 - 18s - loss: 0.0361 - acc: 0.9898 - val_loss: 0.0650 - val_acc: 0.9801

Epoch 6/10 - 18s - loss: 0.0265 - acc: 0.9936 - val_loss: 0.0640 - val_acc: 0.9790
Epoch 7/10 - 18s - loss: 0.0191 - acc: 0.9953 - val_loss: 0.0624 - val_acc: 0.9810
Epoch 8/10 - 18s - loss: 0.0145 - acc: 0.9965 - val_loss: 0.0592 - val_acc: 0.9822
Epoch 9/10 - 18s - loss: 0.0109 - acc: 0.9977 - val_loss: 0.0554 - val_acc: 0.9827

Epoch 10/10 - 18s - loss: 0.0079 - acc: 0.9986 - val_loss: 0.0596 - val_acc: 0.9814
Baseline Error: 1.86%

Как видно, модель достигает точности приблизительно 98.14% и ошибки 1.86% на тестовом наборе данных, это вполне достойно для такой простой модели.

Простая сверточная нейронная сеть для MNIST

Создадим простую CNN для MNIST, которая продемонстрирует, как использовать все аспекты современной реализации CNN.

Первый шаг - импортировать необходимые классы и функции.

import numpy

from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D from keras.utils import np_utils

from keras import backend as K K.set_image_dim_ordering('th')

Далее инициализируем генератор случайных чисел на постоянное начальное значение для воспроизводимости результатов.

seed = 7 numpy.random.seed(seed)

Затем нам нужно загрузить набор данных MNIST и изменить его, чтобы он был подходящим для обучения CNN.

(X_train, y_train), (X_test, y_test) = mnist.load_data()

 # reshape to be [samples][pixels][width][height]

X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')

X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')

Как и прежде нормализуем значения пикселей в диапазоне 0 и 1.

X_train = X_train / 255

X_test = X_test / 255

y_train = np_utils.to_categorical(y_train) y_test = np_utils.to_categorical(y_test) num_classes = y_test.shape[1]

Затем определяем модель нейронной сети. Сверточные нейронные сети более сложны, чем стандартные многослойные персептроны, поэтому начнем с использования простой структуры

Ниже представлена архитектура сети.

1. Первый скрытый слой - это сверточный слой, Convolution2D. Этот слой имеет 32 карты функций, размер которых равен 5 × 5 и функции активации **relu**.

2. Затем мы определяем слой пулинга **maxPooling2D** с размером пула 2 × 2, который дает максимальные значения.

3. Следующий уровень - это уровень регуляризации **Dropout**. Он настроен на случайное исключение 20% нейронов в слое, чтобы уменьшить переобучение.

4. Далее - слой, который преобразует данные двумерной матрицы в вектор, называемый **Flatten**. Он позволяет обрабатывать выходные данные стандартными полносвязными слоями.

5. Затем полносвязный слой с 128 нейронами и функцией активации **relu**.

6. Наконец, выходной слой имеет 10 нейронов для 10 классов и функцию активации **softmax** для вывода вероятностных результатов распознавания для каждого класса.

def baseline_model():

 # create model model = Sequential()

model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.2)) model.add(Flatten())

model.add(Dense(128, activation='relu')) model.add(Dense(num_classes, activation='softmax'))

 # Compile model

model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['ac

curacy']) return model

Как и в примере с моногослойным персептроном эта модель 10 эпох обучения, при каждом обновлении весов используется 200 изображений.

model = baseline_model()

model.fit(X_train, y_train, validation_data=(X_test,y_test), epochs=10, batch_size=200, verbose=2)

*scores = model.evaluate(X_test, y_test, verbose=0) print("CNN Error: %.2f%%" % (100-scores[1]*100))*

Точность классификации модели печатается в каждую эпоху обучения и в конце отпечатается ошибка ошибки классификации.

Train on 60000 samples, validate on 10000 samples

Epoch 1/10 - 276s - loss: 0.2224 - acc: 0.9366 - val_loss: 0.0783 - val_acc: 0.9754 Epoch 2/10 - 279s - loss: 0.0710 - acc: 0.9789 - val_loss: 0.0454 - val_acc: 0.9846 Epoch 3/10 - 449s - loss: 0.0510 - acc: 0.9841 - val_loss: 0.0444 - val_acc: 0.9854 Epoch 4/10 - 267s - loss: 0.0389 - acc: 0.9881 - val_loss: 0.0403 - val_acc: 0.9876

Epoch 5/10 - 269s - loss: 0.0325 - acc: 0.9898 - val_loss: 0.0349 - val_acc: 0.9883 Epoch 6/10 - 313s - loss: 0.0267 - acc: 0.9919 - val_loss: 0.0321 - val_acc: 0.9896 Epoch 7/10 - 255s - loss: 0.0220 - acc: 0.9930 - val_loss: 0.0339 - val_acc: 0.9888 Epoch 8/10 - 271s - loss: 0.0192 - acc: 0.9939 - val_loss: 0.0329 - val_acc: 0.9896 Epoch 9/10 - 266s - loss: 0.0157 - acc: 0.9951 - val_loss: 0.0323 - val_acc: 0.9891 Epoch 10/10 - 279s - loss: 0.0145 - acc: 0.9956 - val_loss: 0.0333 - val_acc: 0.9889 CNN Error: 1.11%

Обучения сверточной нейронной сети занимает больше времени чем обучение простого персептрона, рассмотренного выше. Однако ошибка достигает 1.11%, что значительно меньше по сравнению с персепроном.

Большая сверточная нейронная сеть для MNIST

Создадим модель, которая может быть близка к новейшим научным

результатам.

В начале программы импортируем классы и функции, затем загружаем и готовим данные так же, как в предыдущем примере CNN.

import numpy

from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D from keras.utils import np_utils

from keras import backend as K K.set_image_dim_ordering('th') seed = 7

numpy.random.seed(seed)

 # load data

(X_train, y_train), (X_test, y_test) = mnist.load_data()

 # reshape to be [samples][pixels][width][height]

X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')

X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')

 # normalize inputs from 0-255 to 0-1 X_train = X_train / 255

X_test = X_test / 255

y_train = np_utils.to_categorical(y_train) y_test = np_utils.to_categorical(y_test) num_classes = y_test.shape[1]

Создадим большую архитектуру сверточной нейронной сети с дополнительными сверточными слоями, слоями пуллинга и полностью связанными слоями. Сетевую топологию можно резюмировать следующим образом.

1. Сверточный слой **Conv2D** с 30 функциональными картами размером 5 × 5.
2. Слой максимального пулинга **MaxPooling2D** размером 2 * 2.
3. Сверточный слой **Conv2D** с 15 картинными картами размером 3 × 3.
4. Слой максимального пулинга **MaxPooling2D** размером 2 * 2.
5. Слой исключения **Dropout** с вероятностью 20%.
6. Слой **Flatten**.
7. Полносвязный слой **Dense** с 128 нейронами и функцией активации **relu**.
8. Полносвязный слой **Dense** с 50 нейронами и функцией активации **relu**.
9. Выходной полносвязный слой Dense с функцией активации **softmax**.

def larger_model():

 # create model model = Sequential()

model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(15, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.2)) model.add(Flatten())

model.add(Dense(128, activation='relu')) model.add(Dense(50, activation='relu')) model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['ac curacy'])

return model

model = larger_model()

model.fit(X_train, y_train, validation_data=(X_test,y_test), epochs=10, batch_size=200)

*scores = model.evaluate(X_test, y_test, verbose=0) print("Large CNN Error: %.2f%%" % (100-scores[1]*100))*

Эта модель уже достигает уровня ошибки классификации 0,89%.

Основная литература

1. Волкова, М.А. Методы обработки и распознавания изображений. Учебно-методическое пособие по лабораторному практикуму. [Электронный ресурс] / М.А. Волкова, В.Р. Луцив. — Электрон. дан. — СПб. : НИУ ИТМО, 2016. — 40 с. — Режим доступа: https://e.lanbook.com/reader/book/91416/#1— Загл. с экрана.

2. Ростовцев, В. С. Искусственные нейронные сети : учебник / В. С. Ростовцев. — Санкт-Петербург : Лань, 2019. — 216 с. — ISBN 978-5-8114-3768-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/122180 — Режим доступа: для авториз. пользователей.

3. Вакуленко, С. А. Практический курс по нейронным сетям : учебное пособие / С. А. Вакуленко, А. А. Жихарева. — Санкт-Петербург : НИУ ИТМО, 2018. — 71 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/136500 — Режим доступа: для авториз. пользователей.

Дополнительная литература

1. Фисенко, В.Т. Компьютерная обработка и распознавание изображений. Методические указания к лабораторным работам. [Электронный ресурс] / В.Т. Фисенко, Т.Ю. Фисенко. — Электрон. дан. — СПб. : НИУ ИТМО, 2008. — 42 с. — Режим доступа: http://e.lanbook.com/book/40794 — Загл. с экрана.

2. Ежова, К.В. Моделирование и обработка изображений. [Электронный ресурс] — Электрон. дан. — СПб. : НИУ ИТМО, 2011. — 93 с. — Режим доступа: https://e.lanbook.com/reader/book/40820/#1— Загл. с экрана.

3. Гонсалес, Р. Цифровая обработка изображений. [Электронный ресурс] / Р. Гонсалес, Р. Вудс. — Электрон. дан. — Москва : Техносфера,

2012. — 1104 с. — Режим доступа: http://e.lanbook.com/book/73514 — Загл. с экрана.