РЕФЕРАТ

Курсовая работа 41 с., 6ч., 5 рис., 1табл., 5 источников.

ТЕСТ ИШИХАРА, НЕЙРОННЫЕ СЕТИ, МАТЕМАТИЧЕСКАЯ МОРФОЛОГИЯ, ФИЛЬТРАЦИЯ ИЗОБРАЖЕНИЙ, ЦВЕТОВЫЕ ПРОСТРАНСТВА, ПОРОГОВАЯ ОБРАБОТКА, СГЛАЖИВАНИЕ ИЗОБРАЖЕНИЙ, ФИЛЬТРАЦИЯ ИЗОБРАЖЕНИЙ, ПЕРЦЕПТРОН, БИНАРИЗАЦИЯ

Объектом исследования являются методы обработки и распознавания цветных изображений.

Цель курсовой работы – изучение методов хранения, обработки и распознавания объектов в условиях цветовой близости. Итог проделанной работы – программная реализация генератора карточек теста Ишихара и системы распознавания цифр с них.

СОДЕРЖАНИЕ

Введение.....4
1 Определения, связанные с теорией распознавания.....5
2 Цветовые пространства.....7
2.1 Человеческое зрение.....7
2.2 Представление цвета в машинной графике 8
2.3 Цветовое пространство RGB.....9
2.4 Цветовое пространство LAB.....10
3 Методы обработки изображения.....11
3.1 Пороговая обработка (Thresholding) 11
3.2 Фильтрация изображений.....12
3.2.1 Линейные фильтры.....12
3.2.2 Нелинейный фильтры.....13
3.3 Математическая морфология.....14
4 Метод классификации изображении: персептрон.....18
5 Программная реализация.....21
5.1 Постановка задачи.....21
5.2 Используемые технологии.....22
5.3 Генерация тестового набора.....23
5.4 Фильтрация и векторизация изображений.....24
5.5 Реализация персептрона.....26
5.6 Тестирование.....27
Заключение.....28
Список используемых источников…….....29
Приложение а.....30

ВВЕДЕНИЕ

Теория распознавания образов является одним из основных разделов кибернетики, как в теоретическом, так и в прикладном плане. Так, автоматизация некоторых процессов предполагает создание устройств, способных реагировать на изменяющиеся характеристики внешней среды некоторым количеством положительных реакций.

Базой для решения задач такого уровня являются результаты классической теории статистических решений, в рамках которой разработаны алгоритмы определения класса, к которому может быть отнесен распознаваемый объект. На базе статистики за счет применения разделов прикладной математики, теории информации, методов алгебры логики, математического программирования и системотехники основана теория распознавания.

Кроме того, в основе теории распознавания графических образов лежат концепции теории обработки изображений. В процессе распознавания цветных объектов применяется фильтрация изображений, операции математической морфологии, изменение яркости и контраста изображений, квантование цвета и преобразование графических изображений в другие цветовые пространства.

На данный момент распознавание образов – одно из ведущих направлений кибернетики: оно упрощает взаимодействие человека с компьютером и создает предпосылки для применения различных систем искусственного интеллекта.

В данной работе рассматриваются методы деления изображений на цветовые составляющие, выделения, фильтрации и последующего распознавания образов из данного изображения.

1 Определения, связанные с теорией распознавания

Распознавание образов (объектов, сигналов, ситуаций, явлений или процессов) - задача идентификации объекта или определения каких-либо его свойств по его изображению (оптическое распознавание) или аудиозаписи (акустическое распознавание) или другим характеристикам.

Образ - это описание объекта или процесса, позволяющее выделять его из окружающей среды и группировать с другими объектами или процессами для принятия необходимых решений [1]. Образы обладают характерными объективными свойствами в том смысле, что разные люди, обучающиеся на различном материале наблюдений, большей частью одинаково и независимо друг от друга классифицируют одни и те же объекты.

Изображение объекта - отображение какого-либо воспринимающего органа распознающей системы, независимо от его положения относительно этих органов. Множество изображений, объединенное какими-либо общими свойствами, называется образом. Методику отнесения элемента к какому-либо образу называют решающим правилом. Гистограммой называют графическое представление распределения яркостей изображения.

Признаком изображения называется его простейшая отличительная характеристика или свойство [2]. Некоторые признаки являются естественными в том смысле, что они устанавливаются визуальным анализом изображения, тогда как другие, так называемые искусственные признаки, получаются в результате его специальной обработки или измерений. К естественным признакам относятся светлота (яркость) и текстура различных областей изображения, форма контуров объектов. Гистограммы распределения яркости и спектры пространственных частот дают примеры искусственных признаков.

Метрика - способ определения расстояния между элементами некоторого множества. Чем меньше это расстояние, тем более похожими являются объекты (числа, функции, символы, звуки и другое). Обычно

элементы задаются в виде набора чисел, а метрика - в виде функции. От выбора представления образов и реализации метрики зависит эффективность программы, один алгоритм распознавания с разными метриками будет ошибаться с разной частотой.

Обучение - процесс выработки в некоторой системе той или иной реакции на группы внешних идентичных сигналов путем многократного воздействия на систему внешней корректировки. Такую внешнюю корректировку в обучении принято называть "поощрениями" и "наказаниями". Механизм генерации этой корректировки практически полностью определяет алгоритм обучения. Самообучение отличается от обучения тем, что дополнительная информация о верности реакции системе не сообщается.

Адаптация - процесс изменения параметров и структуры системы, а возможно - и управляющих воздействий, на основе текущей информации с целью достижения определенного состояния системы при начальной неопределенности и изменяющихся условиях работы.

2 Цветовые пространства

2.1 Человеческое зрение

Для человека цвет - психологическое ощущение, вызванное отражением света от объекта. Это ощущение возникает в мозге при возбуждении и торможении цветочувсительных клеток - рецепторов глазной сетчатки. В глазу человека содержатся два типа светочувствительных клеток, называемые фоторецепторами: высокочувствительные палочки и менее чувствительные колбочки.

Палочки функционируют в условиях относительно низкой освещенности и отвечают за действие механизма ночного зрения, однако при этом обеспечивают только нейтральное в цветовом отношении восприятие действительности. Колбочки работают при более высоких уровнях освещенности или яркости, чем палочки и отвечают за механизм дневного зрения, отличительной особенностью которого является способность обеспечения цветового зрения. Колбочки соответствуют красной, зелёной и синей частям спектра и часто называются длинными (L), средними (M) и короткими (S) согласно длинам волн, к которым они наиболее чувствительны. Каждое цветовое ощущение человека может быть представлено в виде суммы ощущений трех основных цветов: красного, зеленого и синего. Субъективное восприятие цвета зависит от многих параметров: яркости, скорости изменения яркости, цветовой температуры, цвета соседних объектов и физиологических отклонений.

2.2 Представление цвета в машинной графике

Понятие цвета возникает при описании восприятия глазами человека электромагнитных волн в определенном диапазоне частот . Человек воспринимает волны длинной λот 400 нм - фиолетовый цвет, до 700 нм красный цвет [3]. Таким образом, самым общим описанием светового потока может служить его спектральная функция.

Пики на кривых чувствительности отвечают красному, зеленому и синему цветам. При этом следует заметить, что восприимчивость к синему цвету значительно ниже, чем к двум другим. Также важным свойством восприятия света человеком является его линейность: при освещении двумя источниками света со спектральными функциями$I_1(\lambda)$ и $I_2(\lambda)$, человек воспринимает их как один со спектральной функцией, равной сумме $I(\lambda) = I1(\lambda)+I2(\lambda)$. Этот факт называется законом Грассмана [3]. Так как области восприятия для разных типов колбочек перекрываются, то возникают метамеры - потоки волн с разными спектральными характеристиками, но воспринимаемые как имеющие один и тот же цвет.

В машинной графике цветовым пространством называют математическую модель представления цвета, в который каждый цвет представляет собой координату в некотором пространстве базисных цветов. Для большинства цветовых пространств отображение координат пространства в цвета является биективным, хотя в общем случае такое отображение сюръективно.

Цветоделением называют технологический этап воспроизведения цветного изображения, при котором свет сложного спектрального состава разделяется на несколько монохромный составляющих, каждая из которых содержит информацию только об одном цвете или другом параметре цветового пространства. Полученные в результате цветоделения изображения называются цветовыми каналами.

2.3 Цветовое пространство RGB

Из рассмотренной выше модели человеческого зрения вытекает, что достаточно обоснованной является трехмерная цветовая модель RGB, в которой базовыми цветами являются красный, зеленый и синий соответственно. Каждая координата цвета целочисленная и лежит в отрезке $[0,255]$.Таким образом, модель RGB содержит $256^3 = 1677721$ цветов. Эта модель характеризуется свойством аддитивности в том смысле, что сложение двух цветов (r_1, g_1, b_1) и (r_2, g_2, b_2) будет составлять новый цвет, вычисленный по формуле (1).

$$\frac{sign(255 - x) + 1}{2}(x - 255) + 255, x = (r_1 + r_2, g_1 + g_2, b_1 + b_2) \quad (1)$$

Векторам $(x, x, x), x \in [0,255]$ в системе RGB соответствуют цвета градации серого, причем нулевому вектору $(0, 0, 0)$ соответствует черный цвет, а вектору $(255, 255, 255)$ - белый. Будем называть Когда одна из компонент достаточно велика то есть лежит в отрезке $[200,255]$, а две другие в $[0,50]$ то есть малы, то получаемый оттенок близок к доминирующему цвету. Иначе, если величина двух каких-либо цветов велика, а оставшегося мала, то получаемые оттенки называют вторичными цветами. Их названия можно видеть в таблице 1.

Таблица 1 – Доминирующие и соответвующие им вторичные цвета

Доминирующие цвета	Соответствующие им вторичные цвета
R,G	Yellow(Желтый)
R,B	Magenta(Пурпурный)
G,B	Cyan(Голубой)

Цветовая модель RGB нашла широкое распространение в технике, используется в мониторах и проекторах.

2.4 Цветовое пространство CIE LAB

CIE LAB (также обозначаемое как L*a*b) - цветовое пространство, введенное международной комиссией по освещению (фр. Commission internationale de l'éclairage) в 1976 году. Пространство представляет из себя трехмерное пространство R^3, в котором можно представить бесконечное число цветов, включая те, что невидимы человеческому глазу. Для цифрового представления, CIE LAB отображается в ограниченное трехмерное целочисленное пространство. Зачастую L находится в пределах $[0,100]$, a и b в пределах $[-128,127]$ или $[0,255]$

Величина L отвечает за яркость точки: L = 0 представляет черный цвет, L = 100 - белый. Значения a=0 и b=0 соответствуют нейтрально серым оттенкам. Ось a представляет красно-зеленую компоненту цвета, зеленые цвета находятся на отрицательных значениях абсцисс, красные - на положительных. Аналогично b представляет желто-голубые цвета, голубые на отрицательной части прямой, желтые - на положительной. В случае если $a, b \in [0,255]$, нулем отсчета считается точка 128.

В модели, между элементами L,a,b заданы нелинейные отношения, предназначенные для имитации нелинейного отклика глаза. Кроме того, равномерные изменения компонентов в цветовом пространстве L*a*b стремятся соответствовать равномерным изменениям воспринимаемого человеком цвета, поэтому относительные расстояния или различия в восприятии между любыми двумя цветами в пространстве L*a*b можно аппроксимировать, рассматривая каждый цвет как точку в трехмерном пространстве. Тогда расстояние между цветами будет определяться через Евклидового расстояния между соответствующими точками [5].

3 Методы обработки изображения

3.1 Пороговая обработка (Thresholding)

Пороговая обработка - простейший метод сегментации ч/б изображений. Заменяет каждый пиксель изображения на белый (255), если значение цвета больше, чем некоторый порог. Если значение пикселя меньше порога – цвет пикселя заменяется на черный (0). Метод также называют бинаризацией. Если интересующий нас объект имеет белый цвет и расположен на черном фоне или наоборот, то определение точек объекта представляет собой тривиальную задачу установления порога по средней яркости. Порог средней яркости - значение, с которым сравнивается яркость каждого пикселя.

Пример: алгоритм автоматического подбора порога.

1) задать начальное приближение – порог (например, использовать половину яркости изображения);

2) разделить изображение на две части: область, в которой яркость пикселей меньше или равна пороговой; область, в которой яркость пикселей больше пороговой;

3) вычислить среднюю яркость на каждой области;

4) вычислить новый порог как среднее средних яркостей на вычисленных областях;

5) если новый порог отличается от предыдущего не больше, чем на заданную малую величину – то порог вычислен и равен новому порогу. Иначе заменить значение порога новым и повторить второй шаг.

Кроме эвристических методов поиска порога, широко применяются и статистические методы, такие как метод Оцу. Метод Оцу предполагает наличия двух выделяющихся пиков на гистограмме. Метод находит такое значение порога при которых он находится на равном расстоянии от пиков. Пример работы метода Оцу можно увидеть на рис1.

Когда изображение освещено неравномерно, используется адаптивная пороговая обработка. Суть её в том, что порог задается не глобально для всего изображения, а вычисляется локально для некоторой области. Это позволяет выделить контуры изображения там, где бинаризация с фиксированным для всего изображения порогом справиться не может.

Рисунок 1 – Пример бинаризации методом Оцу

3.2 Фильтрация изображений

3.2.1 Линейные фильтры

Сглаживающие фильтры делают изображения нерезкими или размытыми. В частности для сглаживания контуров фигур используются фильтр, называемый сверткой. Его суть заключается в том, что каждый пиксель изображения заменяется на некоторое среднее значение окружающих его пикселей. Каждый фильтр имеет свое ядро – матрицу коэффициентов, на которую умножаются соседствующие пиксели целевого изображения. Это ядро может иметь разную размерность, в зависимости от нее увеличивается или уменьшается интенсивность фильтра.

Простейшим фильтром свертки является фильтр усреднения [2]. У него квадратное ядро, каждый элемент равен $\frac{1}{n*m}$ где $n * m$ – размерность ядра.

Ядро нормировано, чтобы процедура подавления шума не вызывала смещения средней яркости обработанного изображения.

Другой пример – размытие по Гауссу. Его ядро для пикселя (m, n) размерности (u, v) радиуса r вычисляется по формуле (2)

$$y(m,n) = \frac{1}{2\pi r^2} \sum_{u,v} e^{\frac{-(u^2+v^2)}{2r^2}} x(m+u, n+v) \quad (2)$$

3.2.2 Нелинейные фильтры

Изображение может повреждаться шумами и помехами различного происхождения, например шумом видеодатчика, шумом зернистости фотоматериалов и ошибками в канале передачи. Их влияние можно минимизировать, пользуясь классическими методами статистической фильтрации. Другой возможный подход основан на использовании эвристических методов пространственной обработки. Шумы видеодатчиков или ошибки в канале передачи обычно проявляются на изображении как разрозненные изменения изолированных элементов, не обладающие пространственной корреляцией. Искаженные элементы часто весьма заметно отличаются от соседних элементов. Это наблюдение послужило основой для многих алгоритмов, обеспечивающих подавления шума.

Медианная фильтрация — метод нелинейной обработки сигналов, разработанный Тьюких. Он особо эффективен для фильтрации белого шума. Одномерный медианный фильтр представляет собой скользящее окно, охватывающее нечетное число элементов, изображения. Центральный элемент заменяется медианой всех элементов изображения в окне. Медианой дискретной последовательности $a_1..a_n$ для нечетного n является тот ее элемент, для которого существуют $\frac{n-1}{2}$ элементов, меньших или равных ему по величине, и $\frac{n-1}{2}$ элементов, больших или равных ему по величине [2].

Окно перемещается вдоль фильтруемого сигнала и вычисления повторяются. В отличие от фильтра усреднения центральный пиксель изображения не вычисляется, а заменяется некоторым пикселем из окна, что увеличивает качества фильтрации. Пример использования медианного фильтра можно увидеть на рис2. Медианный фильтр является нелинейным т.к. медиана суммы двух произвольных последовательностей не равна сумме их медиан, что в ряде случаев может усложнять математический анализ сигналов.

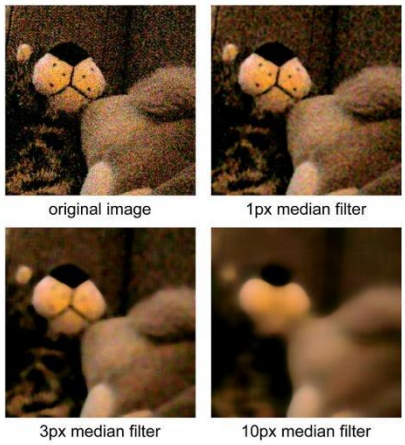
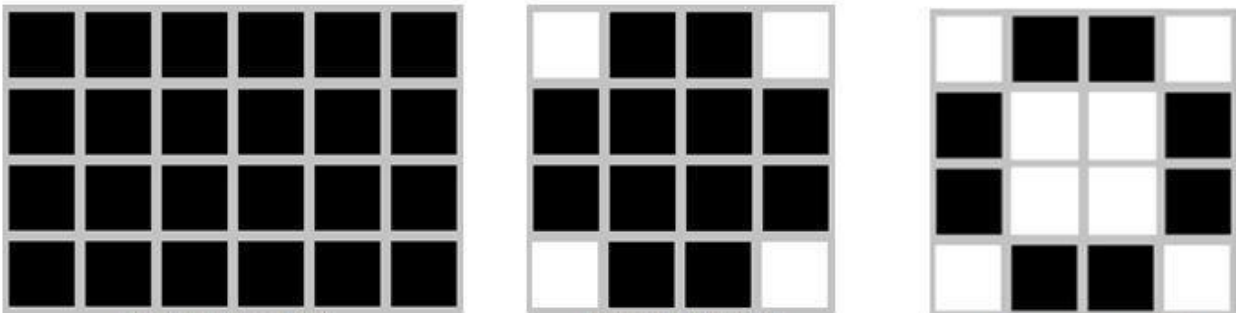


Рисунок 2 – Пример использования медианного фильтра к зашумленному изображению с тремя различными значениями радиуса окна фильтрации

3.3 Математическая морфология

Теория и техника морфологического анализа и обработки изображений основана на теории множеств. Рассмотрим бинарную морфологию: изображение представляется в виде прямоугольных бинарных матриц, где единица — означает белый цвет, а нуль – черный. Для каждой морфологической операции, так же, как и для фильтров, необходимо ядро, которое в данном случае называется структурным элементом [4].Основные структурные элементы можно увидеть на рис3.



1 2 3

Рисунок 3 – Основные структурные элементы математической морфологии:

1) BOX[H,W] - прямоугольник заданного размера;
2) DISK[R] - диск заданного размера;
3) RING[R] - кольцо заданного размера.

Пусть $P(x, y) \in N^2$– координаты пикселей изображения.

Базовые операции:

– Перенос (3)

$$\boldsymbol{P_t} = \{\boldsymbol{x} + \boldsymbol{t} | \boldsymbol{x} \in \boldsymbol{P}\} \quad (3)$$

Операция сдвигает каждый пиксель изображения на вектор t

– Наращивание(4)

$$\boldsymbol{A} \oplus \boldsymbol{B} = \cup_{\boldsymbol{b} \in \boldsymbol{B}} \boldsymbol{A_b} \quad (4)$$

Структурный элемент B применяется ко всем пикселям бинарного изображения. Каждый раз, когда начало координат(центр) структурного элемента совмещается с единичным бинарным пикселем, ко всему структурному элементу применяется перенос и последующее логическое сложение (логическое ИЛИ) с соответствующими пикселями бинарного изображения. Результаты логического сложения записываются в выходное

бинарное изображение, которое изначально инициализируется нулевыми значениями.

– Эрозия(5)

$$\boldsymbol{A} \ominus \boldsymbol{B} = \{\boldsymbol{B}_{\boldsymbol{z}} \subseteq \boldsymbol{A}\} \tag{5}$$

Если в некоторой позиции каждый единичный пиксель структурного элемента совпадает с единичным пикселем бинарного изображения, то выполняется логическое сложение центрального пикселя структурного элемента с соответствующим пикселем выходного изображения. В результате применения операции эрозии все объекты, меньшие, чем структурный элемент, стираются, объекты, соединенные тонкими линиями, становятся разъединёнными и размеры всех объектов уменьшаются.

– Замыкание (6)

$$\boldsymbol{A} \bullet \boldsymbol{B} = (\boldsymbol{A} \oplus \boldsymbol{B}) \ominus \boldsymbol{B} \tag{6}$$

Операция замыкания «закрывает» небольшие внутренние «дырки» в изображении, и убирает углубления по краям области. Если к изображению применить сначала операцию наращивания, то мы сможем избавиться от малых дыр и щелей, но при этом произойдет увеличение контура объекта. Избежать этого увеличения позволяет операция эрозия, выполненная сразу после наращивания с тем же структурным элементом.

– Размыкание (7)

$$\boldsymbol{A} \circ \boldsymbol{B} = (\boldsymbol{A} \ominus \boldsymbol{B}) \oplus \boldsymbol{B} \tag{7}$$

Операция эрозии полезна для удаления малых объектов и различных шумов, но у этой операции есть недостаток — все остающиеся объекты уменьшаются в размере. Этого эффекта можно избежать, если после операции эрозии применить операцию наращивания с тем же структурным элементом. Размыкание отсеивает все объекты, меньшие чем структурный элемент, но при этом помогает избежать сильного уменьшения размера объектов. Также размыкание идеально подходит для удаления линий, толщина которых меньше, чем диаметр структурного элемента. Также важно помнить, что после этой операции контуры объектов становятся более гладкими.

4 Метод классификации изображении: персептрон

Для решения задачи классификации полученных отфильтрованных изображений очень эффективен метод классификации, основанный на персептроне. Персептрон, в свою очередь, основан на искусственных нейронах - модель клетки мозга.

Искусственный нейрон - взвешенный сумматор, единственный выход которого определяется через его входы и матрицу весов по формуле (8).

$$y = f(u), u = \sum_{i=1}^{n} x_i w_i + x_0 w_0 A \circ B = (A \ominus B) \oplus B \tag{8}$$

Здесь функция u называется индуцированным локальным полем, $f(u)$ – функцией активации или пороговой функцией, а x_i и w_i — соответственно есть сигналы на входах нейрона и веса входов. Функция активации определяет зависимость сигнала на выходе нейрона от взвешенной суммы сигналов на его входах. В большинстве случаев она является монотонно возрастающей и имеет область значений $[0,1]$ или $[-1,1]$, однако существуют исключения.

Персептрон состоит из трех основных структурных элементов: Входного, скрытого и выходного слоя. Скрытых слоев может быть несколько. Каждый слой в персептроне связан со следующим, причем связь полная то есть каждый нейрон первого слоя связан с каждым нейроном следующего. Ребра, соединяющие слои нейронов называют синапсами.

– Входной слой, традиционно, передает входной сигнал в каждый нейрон первого скрытого слоя;

– Скрытые слои состоят из нейронов, обычно в качестве функции активации используется (9)

$$f(s) = \frac{1}{1 + e^{-2\alpha s}} \quad (9)$$

– Выходной слой также состоит из нейронов, выходы выходного слоя интерпретируются как выходное значение персептрона.

Чтобы предсказать некоторое событие или, например, определить цифру, изображенную на изображении, на входы персептрона подается значения, обычно в пределах [0,1], представляющее входные данные. Данные из входного слоя посредством синапсов подаются в первый скрытый слой нейронов. Пройдя все скрытые слои, сигнал попадает в выходной слой, значения на выходах, которых называют выходом нейронной сети или значением нейронной сети.

Обучение персептрона - вычисление таких коэффициентов w_i для всех нейронов кроме выходных, при которых значение нейронной сети будет близкой к желаемому. Для обучения любой нейросети необходим некоторый опыт, который для персептрона представляется набором размеченных тестовых данных. Размеченными они называются потому, что каждому входному набору ставится в соответствие правильный ответ, который нейронная сеть должна выдать в качестве ответа.

Для обучения персептрона эффективен метод обратного распространения ошибки. Для реализации этого процесса необходимо понятие метрики. Метрика - функция, определяющее расстояние между точками некоторого пространства, в нашем случае пространства ответов R n [0,1] где н - кол-во выходных нейронов.

Обучение происходит следующим образом: на вход персептрона подается элемент обучающей выборки, сеть вычисляет свое выходное значение, вычисляется ошибка выходного слоя - расстояние между правильным ответом и ответом нейронной сети. Методом обратного распространения ошибки последовательно вычисляются значения ошибок

для всех скрытых слоев персептрона, начиная с последнего, а затем веса искусственных нейронов, основываясь на величине ошибки, корректируются.

Преимущество этого метода заключается в том, что он может обучить все слои нейронной сети, и его легко просчитать локально для каждого слоя. Однако этот метод является очень долгим, к тому же, для его применения активационная функция нейронов должна быть дифференцируемой.

5 Программная реализация

5.1 Постановка задачи

Написать программу, которая бы распознавала числа из теста Ишихара – рис4.

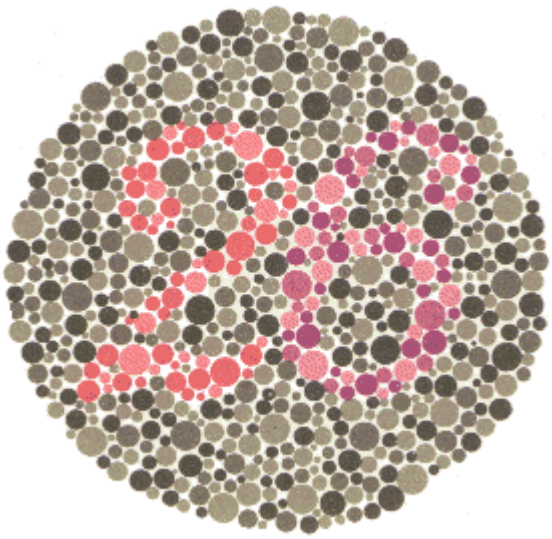


Рисунок 4 – Оригинальная карточка теста Ишихара

Тест Ишихара - первый успешный тест восприятия цвета, состоящий из так называемых псевдо-изохроматических карточек. Назван в честь Японского ученого Shinobu Ishihara, который опубликовал этот тест 1917 году.

Тест содержит карточки, на каждой из которых изображена окружность, включающая в себя точки (окружности) разных цветов и размеров. Однако точки окрашены так, что некоторые из них выделяют фон, а остальные – цифру. Люди с нормальным цветовым восприятием могут отделить цифру от фона. Люди с дихроматическим зрением могут сделать ошибочные выводы, так как видят только два из трех доступных цветовых канала. Люди с монохроматическим зрением не смогут различить цифру.

Проведения теста Ишихара:

Тестируемому дается три секунды на то, чтобы определить цифру с карточки. Тестируемому не разрешается советоваться или трогать карточки. Некоторые из карточек «Легкие»: распознать цифру может даже человеку с монохромным зрением. Во избежание заучивания карточек они перемешиваются, так что «легкие» карточки подаются вперемешку с обычными.

Тест Ишихара используется повсеместно благодаря высокой точности и простоте тестирования.

5.2 Используемые технологии

OpenCV – библиотека алгоритмов компьютерного зрения с открытым исходным кодом. До первой версии разрабатывалась в Центре разработки программного обеспечения Intel (российской командой в Нижнем Новгороде). OpenCV написана на языке высокого уровня (C/C++, Python) и содержит алгоритмы для интерпретации изображений, калибровки камеры по эталону, устранения оптических искажений, определения сходства, анализ перемещения объекта, определения формы объекта и слежения за объектом, 3D-реконструкции, сегментации объекта, распознавания жестов и т.д.

Получила широкое распространения благодаря открытости и возможности бесплатного использования как в учебных, так и коммерческих целях.

Keras — это библиотека с открытым исходным кодом, позволяющая легко создавать нейронные сети. Библиотека совместима с TensorFlow, Theano и другими библиотеками машинного обучения. Tensorflow и Theano являются наиболее часто используемыми платформами для разработки алгоритмов глубокого обучения, но они довольно сложны в использовании.

Keras напротив предоставляет простой и удобный способ создания моделей глубокого обучения. Ее создатель, François Chollet, разработал ее

для того, чтобы максимально ускорить и упростить процесс создания нейронных сетей. Он сосредоточил свое внимание на расширяемости, модульности, минимализме и поддержке Python. Библиотека keras внесла большой вклад в коммерциализацию глубокого обучения и искусственного интеллекта, поскольку содержит современные алгоритмы глубокого обучения, которые ранее были не только недоступными, но и непригодными для использования.

Для тренировки персептрона и выявления закономерностей каждой отдельной цифры необходимо сгенерировать входной набор данных. Для создания тестового набора была использована несколько модифицированная бесплатная программа ishihara_generator. Также использовались шрифты google fonts.

5.3 Генерация тестового набора

Первый этап: генерация карточек Ишихара.

Посредством программы convert-im6 было сгенерированы изображения цифр разных шрифтов (30 случайно выбранных из 2000 имеющихся шрифтов) Таким образом, цифры на карточках в тестовой выборке имеют разную форму, что не даст персептрону переобучиться на единственном шрифте. Карточки генерируются в форме квадрата.

Для генерации карточек было выбрано 4 цветовые темы, изображенные на рис5.

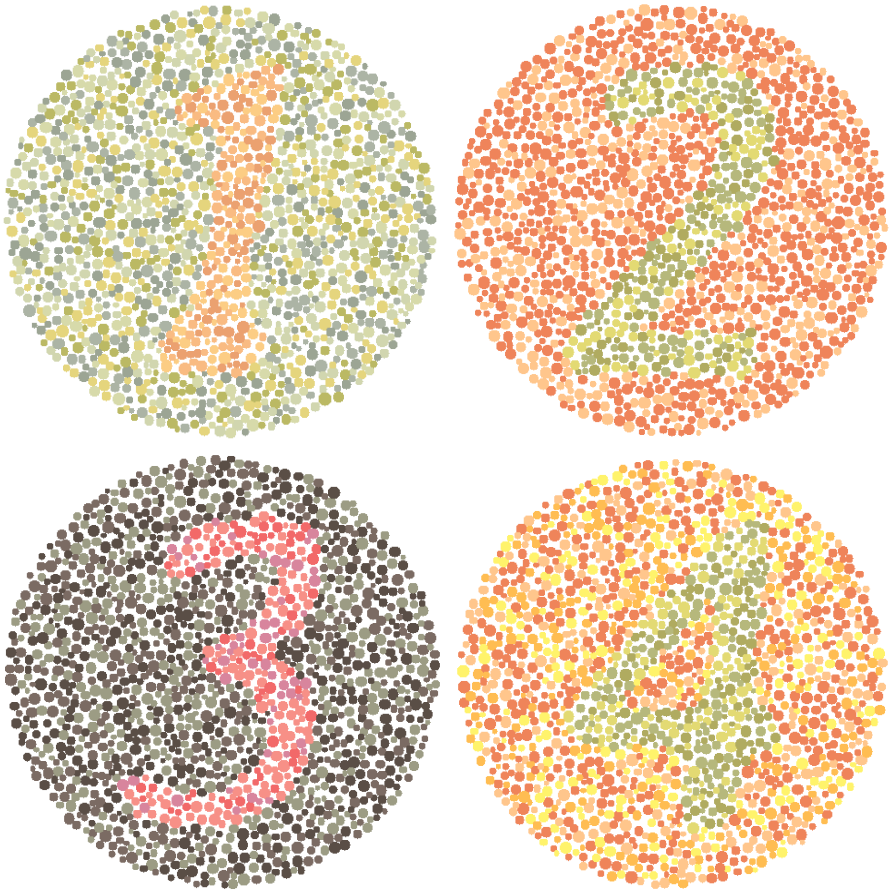


Рисунок 5 – Примеры сгенерированных карточек

После генерации изображение зашумлялось: к яркости каждого пикселя каждого канала добавлялось -50 тире 50 единиц.
В итоге было сгенерировано 1400 карточек Ишихара, что вполне достаточно для обучения.

5.4 Фильтрация и векторизация изображений

Для фильтрации изображений была использована библиотека OpenCV, а именно функции, реализующие бинаризацию, фильтрацию и методы математической морфологии. На первом этапе производилась бинаризация изображений. Все изображения делились на три категории:

– Канал A в среднем темнее, чем середина диапазона. Это значит, что на изображении преобладает зеленый или синий цвет. Следовательно, исходя

из имеющихся цветовых тем, мы имеем красновато-бежевую цифру на зеленом фоне. В канале А красные цвета очень яркие, так что для отделения цифры достаточно применить к этому каналу бинаризацию с порогом чуть выше середины диапазона.

Если же в среднем канал А светлый, рассматривается среднее значение яркости канала В

– При условии светлого каналов А и В получаем средний цвет в области красного. Из цветовых тем видно, что цифра скорее всего будет зеленой. Красный канал будет светлым, но зеленая цифра будет достаточно темной на ней. Поэтому к красному каналу применяется фильтр с высоким порогом, а после изображение инвертируется.

– В оставшемся случае получим средний цвет в районе серого и фиолетово-розового цвета. В этом случае цифра может быть красно-розовой на сером фоне. Таким образом, в канале А наблюдается сильный цветовой контраст, и метод бинаризации Оцу хорошо отделяет цифру от фона.

В итоге карточки преобразуются в черно-белые изображения, где на черном фоне белыми точками выделялась форма цифр. Следующий этап фильтрации объединяет эти точки в одну непрерывную фигуру. Это действие выполняется с помощью математической морфологии поэтапно:

Сначала изображение очищается от мелких шумов путем замыкания с небольшим ядром (Единичная матрица $2 * 2$). После этого кружки объединяются в непрерывную фигуру путем замыкания с единичным ядром размера $12 * 12$

К полученному изображению, для сглаживания ребер цифр, применяется медианный фильтр.

Полученные изображения методами openCV сжимаются до размера 28*28. Так как все тестовые карточки имеют квадратную форму, то сжатые изображения не имеют геометрических искажений. Размер сжатого изображения оптимален: на нем сохраняются все необходимые для распознавания признаки той или иной цифры. Увеличение размера входного

изображения влечет за собой не линейный рост сложности обучения и не гарантирует увеличения точности распознавания чисел.

5.5 Реализация персептрона

Для реализации персептрона использовалось готовое решение – Фреймворк keras. Персептрон, с помощью которого распознавались отфильтрованные тесты, имел следующую архитектуру:

– Входной слой 28*28 входов

– Скрытый слой из 512 элементов, функция активации – relu (10),

$$relu(x) = max(\varepsilon, x), \varepsilon \geq 0 \tag{10}$$

– Выходной слой из 10 элементов, функция активации – softmax (11),

$$softmax(z)_i = \frac{z_i}{\sum_{k=0}^{n} z_k}, z = \{z_i\}, softmax(z) = \{softmax(z)_i\}, \\ i = \overline{0, n}, relu(x) = max(\varepsilon, x), \varepsilon \geq 0 \tag{11}$$

– В качестве функции ошибки используется перекрестная энтропия (12),

$$H(t, o) = -\sum_{k} t_i log(o_i) \tag{12}$$

Где o – вывод нейронной сети, t – верный ответ

В качестве алгоритма обучения используется RMSProp – несколько усовершенствованный метод градиентного спуска, включающий в себя метод бегущего среднего и адаптивность.

5.6 Тестирование

Тестирование было проведено на сгенерированном зашумленном наборе карт Ишихара. Этап выбора цветовых каналов, отделения цифр от фона и восстановления их формы прошел удовлетворительно. Вся выборка была разделена на тестовую и обучающую в соотношении 600/800. В результате обучения персептрона на обучающей выборке, точность распознавания на тестовых данных достигла 99%.

ЗАКЛЮЧЕНИЕ

В курсовой работе были рассмотрены методы представления, распознавания и обработки изображений, рассмотрены операции математической морфологии, реализовано приложение, решающая тест Ишихара.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Чабан. Л.Н. Теория и алгоритмы распознавания образов. Учебное пособие. М.: МИИГАиК. 2004. – 70с.

2 Прэтт У. Цифровая обработка изображений. М.: Мир, 1982. Т.2.

3 Алгоритмические основы построения растровой графики. URL https://www.intuit.ru/studies/courses/993/163/lecture/4491 (дата обращения: 10.12.2018).

4 Форсайт Д., Понс Ж.. Компьютерное зрение. Современный подход. изд. — М.: Вильямс, 2004. — 928 с.

5 Jain, Anil K. (1989). Fundamentals of Digital Image Processing. New Jersey, United States of America: Prentice Hall. ISBN 0-13-336165-9.

ПРИЛОЖЕНИЕ А

ttf2png.py:

```
import sys
import os
import subprocess
import random
from fontTools.ttLib import TTFont

RESULT_PATH = "fonts"
FONT_COUNT = 30
BAD_FONTS =
(b'cst',b'rsfs',b'lklug',b'esint',b'ani',b'cmex',b'msam',b'EagleLake')

def gen_pics(TTF_PATH,FONT_SIZE = "500"):
	'''
	Generate png picture of digits with fonts from TTF_PATH.
	'''
	TTF_NAME = os.path.splitext(os.path.basename(TTF_PATH))[0]

	for i in range(10):
	    name = str(i)
	    output_png = os.path.join( RESULT_PATH ,name + "_" +
TTF_NAME + ".png")
	    subprocess.call(["convert-im6.q16", "-font", TTF_PATH, "-
pointsize", FONT_SIZE, "-background", "#FFFFFF", "label:" + str(i)  ,
output_png])

def main():
	# Read all fonts
	process = subprocess.Popen(['convert', '-list','font' ],
stdout=subprocess.PIPE)
	out = process.communicate()[0]
	# Choose good fonts
	out = [x.decode("utf-8") for x in out.split() \
	        if x.find(b'ttf') != -1 and all( map( lambda type:
x.find(type) == -1 , BAD_FONTS ) ) ]
	# Choise random fonts
	ttfs = random.sample(out,FONT_COUNT)
	print("Generate:")
	for x in ttfs:
		print(x)
	'''
	ttfs = ["/home/dupeljan/.local/share/fonts/GamjaFlower-
Regular.ttf",\
	        "/usr/share/fonts/truetype/tlwg/Kinnari.ttf",\
	        "/usr/share/fonts/truetype/tlwg/Laksaman-
Italic.ttf",\
	        "/home/dupeljan/.local/share/fonts/Lora-
Bold.ttf",\
	        "/usr/share/fonts/truetype/malayalam/Uroob.ttf"]
	'''
	if not os.path.isdir(RESULT_PATH):
		os.mkdir(RESULT_PATH)
```

```
        for ttf in ttfs:
                gen_pics(ttf)
        print("Generate digits successfuly")


if __name__ == '__main__':
        main()
```

norm.py:

```
import cv2
import numpy as np
import os

DIR_NAME = "fonts"
BORDER = 1.5
def gen_norm(name):
    '''
    Search contour, cut out the shape
    and insert into the center of the new picture.

    Border is the ratio of the length of square edge
    of obtained image to the larger side of the cut-out digit.
    '''


    img = cv2.imread(os.path.join(DIR_NAME,name+'.png'))
    hsv = cv2.cvtColor( img, cv2.COLOR_BGR2HSV )
    thresh = cv2.inRange( hsv, 0, 255,0 )
    contours, hierarchy = cv2.findContours(thresh,
cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)[1:]


    try:
        x0 = ( min(contours[0][i][0][0] for i in
range(contours[0].shape[0])), min(contours[0][i][0][1] for i in
range(contours[0].shape[0])) )
        x1 = ( max(contours[0][i][0][0] for i in
range(contours[0].shape[0])), max(contours[0][i][0][1] for i in
range(contours[0].shape[0])) )
    except IndexError:
        print("Error to norm ",name)
        return


    # gen new picture
```

```
    weight = height = int ( max ( x1[0] - x0[0] , x1[1] - x0[1] )  *
BORDER )

    new_img = np.zeros((weight,height,3), np.uint8)
    cv2.rectangle(new_img,(0,0),(weight,height),(255,255,255),-1)

    dx = int ( weight / 2 - (x1[1] - x0[1]) / 2 )
    dy = int ( height / 2 - (x1[0] - x0[0]) / 2 )
    new_img[dx:x1[1]-x0[1] + dx, dy :x1[0] - x0[0]  + dy ] =
img[x0[1]:x1[1], x0[0]:x1[0]]



    cv2.imwrite(os.path.join(DIR_NAME,name+'.png'),new_img)

def main():
    files = os.listdir(DIR_NAME)
    for filename in files:
        name,ext = os.path.splitext(filename)
        if ext == ".png":
            gen_norm(name)

    print("Normalize all pic in current dir")
if __name__ == '__main__':
    main()
ishihara.py
import math
import random
import os
import sys

from PIL import Image, ImageDraw

SOURCE_PATCH = "fonts"
RESULT_PATH = "test5"
GEN_COUNT = 1

try:
    from scipy.spatial import cKDTree as KDTree
    import numpy as np
    IMPORTED_SCIPY = True
```

```
except ImportError:
    IMPORTED_SCIPY = False

BACKGROUND = (255, 255, 255)
TOTAL_CIRCLES = 1500

color = lambda c: ((c >> 16) & 255, (c >> 8) & 255, c & 255)
if sys.argv[1]:
    COLOR_THEAM = int(sys.argv[1] ) % 5
else:
    COLOR_THEAM = 1

TYPE = COLOR_THEAM if COLOR_THEAM <= 3 else 3

#   type 1
if COLOR_THEAM == 1:
    COLORS_ON = [
        color(0xF9BB82), color(0xEBA170), color(0xFCCD84)
    ]
    COLORS_OFF = [
        color(0x9CA594), color(0xACB4A5), color(0xBBB964),
        color(0xD7DAAA), color(0xE5D57D), color(0xD1D6AF)
    ]

# type 2
elif COLOR_THEAM == 2:
    COLORS_ON = [
        color(0xb6b87c), color(0xe3da73), color(0xb0ab60)
    ]
    COLORS_OFF = [
        color(0xef845a), color(0xffc68c), color(0xef845a),
    ]

# type 3
elif COLOR_THEAM == 3:
    COLORS_ON = [
        color(0xf79087), color(0xf26969), color(0xd8859d),\
        color(0xf79087)
    ]
```

```
    COLORS_OFF = [
        color(0x5a4e46), color(0x7b6b63), color(0x9c9c84),
    ]

# type 3 too
elif COLOR_THEAM == 4:
    COLORS_ON = [
        color(0xb6b87c), color(0xe3da73), color(0xb0ab60)
    ]
    COLORS_OFF = [
        color(0xef845a), color(0xffc68c), color(0xef845a),\
        color(0xfff36b), color(0xffbd52)
    ]



def generate_circle(image_width, image_height, min_diameter,
max_diameter):
    radius = random.triangular(min_diameter, max_diameter,
                               max_diameter * 0.8 + min_diameter *
0.2) / 2

    angle = random.uniform(0, math.pi * 2)
    distance_from_center = random.uniform(0, image_width * 0.48 -
radius)
    x = image_width  * 0.5 + math.cos(angle) * distance_from_center
    y = image_height * 0.5 + math.sin(angle) * distance_from_center

    return x, y, radius



def overlaps_motive(image, par):
    (x, y, r) = par
    points_x = [x, x, x, x-r, x+r, x-r*0.93, x-r*0.93, x+r*0.93,
x+r*0.93]
    points_y = [y, y-r, y+r, y, y, y+r*0.93, y-r*0.93, y+r*0.93, y-
r*0.93]

    for xy in zip(points_x, points_y):
        try:
            if image.getpixel(xy)[:3] != BACKGROUND:
                return True
```

```
        except IndexError:
            print ("Exept gen")
            return False
        except TypeError:
            print ("second exept gen")
            return False

    return False


def circle_intersection(par1, par2):
    (x1, y1, r1) = par1
    (x2, y2, r2) = par2
    return (x2 - x1)**2 + (y2 - y1)**2 < (r2 + r1)**2


def circle_draw(draw_image, image, par):
    (x, y, r) = par
    fill_colors = COLORS_ON if overlaps_motive(image, (x, y, r)) else
COLORS_OFF
    fill_color = random.choice(fill_colors)

    draw_image.ellipse((x - r, y - r, x + r, y + r),
                       fill=fill_color,
                       outline=fill_color)


def gen_test(name,gen_n = 0):
    '''
    Generate Ishihara test card
    from file name.png to name+gen_n+.png
    '''
    image = Image.open(os.path.join(SOURCE_PATCH,name + ".png"))
    image2 = Image.new('RGB', image.size, BACKGROUND)
    draw_image = ImageDraw.Draw(image2)

    width, height = image.size

    min_diameter = (width + height) / 200
```

```
    max_diameter = (width + height) / 75

    circle = generate_circle(width, height, min_diameter,
max_diameter)
    circles = [circle]

    circle_draw(draw_image, image, circle)

    try:
        for i in range(TOTAL_CIRCLES):
            tries = 0
            if IMPORTED_SCIPY:
                kdtree = KDTree([(x, y) for (x, y, _) in circles])
                while True:
                    circle = generate_circle(width, height,
min_diameter, max_diameter)
                    elements, indexes = kdtree.query([(circle[0],
circle[1])], k=12)
                    for element, index in zip(elements[0],
indexes[0]):
                        if not np.isinf(element) and
circle_intersection(circle, circles[index]):
                            break
                    else:
                        break
                    tries += 1
            else:
                while any(circle_intersection(circle, circle2) for
circle2 in circles):
                    tries += 1
                    circle = generate_circle(width, height,
min_diameter, max_diameter)

            #print ('{}/{} {}'.format(i, TOTAL_CIRCLES, tries) )

            circles.append(circle)
            circle_draw(draw_image, image, circle)
    except (KeyboardInterrupt, SystemExit):
        pass

    name =  name + "theme_" + str(COLOR_THEAM) + " type_" + str(TYPE)
+ ".png"
```

```
    image2.save(os.path.join(RESULT_PATH , name ) , "PNG")
    print ("Generate " + name +" succesfully" )

def main():
    files = os.listdir(SOURCE_PATCH)
    if not os.path.isdir(RESULT_PATH):
        os.mkdir(RESULT_PATH)
    count = 0
    for filename in files:
        name,ext = os.path.splitext(filename)
        if ext == ".png":
            for i in range(GEN_COUNT):
                gen_test(name,i)
            count += 1
            print("not more than " +str( len(files) - count ) + "
files left" )
    print("Generate test success")

if __name__ == '__main__':
    main()
```

noise.py:

```
import cv2
import numpy as np
from random import randint
import os

SOURCE_PATCH = "test5"
RESULT_PATH = "noise test5"

def add_noise(name,k=10):
    '''
    Add noise in each pixel of name.png.
    256/k - ratio of dispersion.
    '''
    range_ = int(512/k)
    img = cv2.imread(os.path.join(SOURCE_PATCH ,  name + ".png"))
```

```
    for i in range(len(img)):
        for j , x in enumerate(img[i]):
            rand = [ int (randint(0,range_) - range_ / 2)  for i in
range(3)]
            for k in range(3):
                sum_ = x[k] + rand[k]
                if sum_ <= 0:
                    rand[k] = 0
                elif 0 < sum_ < 255:
                    rand[k] = sum_
                else:
                    rand[k] = 255
            img[i][j] =  rand
    cv2.imwrite(os.path.join(RESULT_PATH , name+".png"),img)
    print ( name+".png" + " generated")

def main():
    if not os.path.isdir(RESULT_PATH):
        os.mkdir(RESULT_PATH)
    files = os.listdir(SOURCE_PATCH)
    count = 0
    for filename in files:
        name,ext = os.path.splitext(filename)
        if ext == ".png":
            add_noise(name)
            count += 1
            print("not more than " + str(len(files) - count) + " files
left" )
    print("Generate test success")

if __name__ == '__main__':
    main()
```

filter.py:

```
import cv2
import numpy as np
import os

SOURCE_PATCH = "noise test5"
RESULT_PATH = "filtred_noise 5"

def filter(name):
```

```
    '''
    Select number from name.png test card.
    Create b/w picture of it.
    '''
    rgb = cv2.imread(os.path.join(SOURCE_PATCH,name+'.png'))
    lab = cv2.cvtColor(rgb, cv2.COLOR_BGR2Lab)

    average_a = np.array( list(x[1]  for i in range(len(lab)) for x in
lab[i] )) .mean()
    average_b = 0

    a = cv2.split(lab)[1]
    if average_a < 130: # MANY GREEN OR BLUE, IN A CONTRAST
        rgb = cv2.threshold(a,132,255,cv2.THRESH_BINARY)[1]
        state = 1
    else:
        average_b = np.array( list(x[2]  for i in range(len(lab)) for x
in lab[i] )) .mean()
        if  average_b < 135: # LITTLE RED => RED CHANNEL DARK, IN A
CONTRAST
            state = 2
            rgb =
cv2.threshold(a,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]#
        else:
            state = 3 # МНОГО КРАСНОГО => MANY RED => RED CHANNEL LIGHT,
IN RED CONTRAST
            rgb =
cv2.threshold(cv2.split(rgb)[2],227,255,cv2.THRESH_BINARY)[1]
            rgb = cv2.bitwise_not(rgb)

    kernel_opening = np.ones((2,2),np.uint8)
    kernel_closing = np.ones((12, 12),np.uint8)

    opening = cv2.morphologyEx(rgb, cv2.MORPH_OPEN, kernel_opening)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE,
kernel_closing)

    dst = cv2.medianBlur(closing,5)

    cv2.imwrite( os.path.join(RESULT_PATH ,  name +".png"),dst )
    print(name+".png filtered")


def main():
    if not os.path.isdir(RESULT_PATH):
        os.mkdir(RESULT_PATH)
    files = os.listdir(SOURCE_PATCH)

    for filename in files:
        name,ext = os.path.splitext(filename)
        if ext == ".png":
            filter(name)

if __name__ == '__main__':
    main()
network.py
from keras.utils import to_categorical
```

```
from keras import models
from keras import layers
import numpy as np
import os
import cv2

TEST_PATH = "filtred_noise_5"
DATASET_PATH = TEST_PATH +"_dataset" + ".npz"
TRAIN_COUNT = 800

def create_dataset():
    '''
    Generate BORDER in file DATASET_PATH.npz element "dataset"
    '''
    images = list()
    labels = list()
    files = os.listdir(TEST_PATH)
    for filename in files:
        name,ext = os.path.splitext(filename)
        if ext == ".png":
            img = cv2.imread(os.path.join(TEST_PATH,name+'.png'))
            img = np.compress([True],cv2.resize(img, (28,
28)).reshape(784,3), axis=1 )
            images.append( img )
            labels.append( int(name[0]) )
    np.savez(DATASET_PATH,dataset= np.array( [images, labels] ) )

def get_dataset(path= DATASET_PATH):
    '''
    Generate BORDER from file DATASET_PATH.npz
    '''
    data = np.load(path)['dataset']
    return ( np.stack(data[0],axis=0) , np.stack(data[1],axis=0) )


def create_network():
    images,labels = get_dataset()
    train_images, test_images = images[:TRAIN_COUNT],
images[TRAIN_COUNT:]
    train_labels, test_labels = labels[:TRAIN_COUNT],
labels[TRAIN_COUNT:]

    network = models.Sequential()
    network.add(layers.Dense(512, activation='relu', input_shape=(28 *
28,)))
    network.add(layers.Dense(10, activation='softmax'))

    network.compile(optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy'])


    train_images = train_images.reshape((TRAIN_COUNT, 28 * 28))
    train_images = train_images.astype('float32') / 255
    test_images = test_images.reshape((len(images) - TRAIN_COUNT, 28 *
28))
    test_images = test_images.astype('float32') / 255
```

```
    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)

    network.fit(train_images, train_labels, epochs=5, batch_size=128)

    test_acc = network.evaluate(test_images, test_labels)[1]
    print("Accuracy: ", test_acc)

def main():
    create_dataset()
    create_network()

if __name__ == '__main__':
    main()
```