Федеральное государственное автономное образовательное учреждение
высшего образования

**«Национальный исследовательский университет
«Высшая школа экономики»**

Факультет компьютерных наук

ОП «Прикладная математика и информатика»

Отчёт о прохождении практики

Студент: Рубачёв Иван Викторович

Группа: БПМИ165

Вид практики: Учебная

Руководитель: К.ф.-м.н., доцент,
Конушин Антон Сергеевич

Москва, 2017

Содержание

Введение

Цель данной практики: получение базовых знаний и ознакомление с компьютерным зрением и машинным обучением. Во время прохождения практики было необходимо просмотреть и изучить 5 видеолекций курса «Введение в компьютерное зрение и глубинное обучение» на следующие темы:

- История и предмет компьютерного зрения
- Основы обработки изображений
- Особые точки и сопоставление изображений
- Введение в машинное обучение
- Классификация изображений

Также нужно было выполнить 3 практических задания для отработки изученных методов и закрепления знаний, полученных при просмотре лекций. Первое задание заключалось в реконструкция изображений Прокудина-Горского, во втором задании необходимо было реализовать алгоритм изменения размеров изображения с сохранением пропорций. В последнем задании необходимо было использовать алгоритм классификации SVM для распознавания дорожных знаков. Каждую из задач необходимо было сдать в тестирующую систему, которая оценивала выполнение в соответствии с критериями указанными в заданиях.

В основной части отчёта содержится более подробная информация о проделанной работе. Заключение содержит информацию о полученных знаниях и навыках, их актуальности.

Основная часть

В данном разделе отчёта более подробно описана проделанная работа (в частности выполнение практических заданий). Смысл поставленных передо мной задач заключался по-большому счету в получении новых знаний и практических навыков в области компьютерного зрения. Поскольку практика подразумевает введение в область, знания приобретенные во время её прохождения необходимы и актуальны при изучении других более продвинутых разделов компьютерного зрения.

При выполнении данной практики я старался выполнять практические задания после изучения лекций, содержавших материалы имеющие к ним отношение.

Совмещение каналов изображения

К выполнению первого практического задания я приступил после просмотра вводной лекции (об истории и предмете компьютерного зрения), так как для успешного выполнения этого задания было достаточно знать python и то как представляются цветные изображения.

В первом задании необходимо было реализовать функцию, которая принимает изображение, полученное сканированием фотопластинки содержащей три изображения в градациях серого, и возвращает цветное изображение. Три изображения на входе соответствуют синему, зеленому и красному цветовым каналам, также на вход подается координата точки зеленого канала. Помимо совмещенного цветного изображения, функция должна вернуть соответствующие координаты синего и красного цветовых каналов.

Данную задачу было предложено выполнять последовательно: сначала реализовать базовое совмещение слоев, а затем дополнить его и реализовать более эффективный алгоритм совмещения «с помощью пирамиды».

Базовое совмещение

Базовый алгоритм совмещения прост: входное изображение делим на 3 равные части по горизонтали — получаем синий зеленый и красный слои. Далее каждый слой обрезаем, чтобы избавиться от рамок, затем находим сдвиги синего и красного слоев относительно зеленого, для этого используя метрику (среднеквадратичное отклонение или нормализованную кросс-корелляцию) находим перебором среди всех возможных пар сдвигов от -15 до 15 пикселов ту, для которой значение метрики оптимально.

Так как изображения на вход подаются в градациях серого, то разделить изображение на три канала, а затем обрезать полученные три изображения — простые операции над двумерными массивами. Затем используя одну из двух метрик находим сдвиг (перебирая все возможные сдвиги), затем сдвигая одно изображение относительно другого, для этого можно использовать функцию `np.roll()`. На очередной итерации сдвигаем изображение, находим значение метрики перезаписываем оптимальный сдвиг если значение меньше (больше для кросс-корелляции). В итоге получаем оптимальный сдвиг одного изображения относительно другого. Стоит отметить, что после проверок было решено использовать среднюю квадратичную ошибку, так как нормализованная кросс-корелляция не давала существенного увеличения точности. При подсчете среднеквадратичного отклонения на целых изображениях, а не в пределах перекрывающихся областей, изменений в точности замечено не было. Оставалось просто найти описанным выше способом сдвиг синего и красного каналов относительно зеленого и совместить эти три слоя предварительно сдвинув с помощью `np.roll()`. В результате получаем совмещенное цветное изображение. Зная сдвиги не сложно посчитать координаты соответствующих точек красного и синего каналов по заданной точке зеленого.

Совмещение с помощью пирамиды

При совмещении больших изображений базовый подход с перебором всех возможных сдвигов работает очень долго. Совмещение с помощью пирамиды — метод, который позволяет ускорить совмещение больших изображений. Для реализации данного типа совмещения была использована рекурсивная функция, работающая следующим образом: если высота переданного изображения < 300 то используем базовый алгоритм совмещения и находим сдвиг в промежутке $[-15, 15]$, иначе уменьшаем изображение в два раза и вызываем функцию рекурсивно, затем сохраняем сдвиги уменьшенных изображений, полученных рекурсивно. Умножаем сдвиги на два и возвращаем результат базового совмещения для изображений исходного размера (не уменьшенных на данном шаге рекурсии) со сдвигом в промежутке $[dx - 1; dx + 1]$ и $[dy - 1; dy + 1]$, где dx, dy — сдвиги найденные функцией рекурсивно.

Тестирование показало, что нижняя граница в 300 пикселей и уточнение сдвига на ± 1 пиксель позволяют достичь желаемой точности нахождения сдвига, при этом работая быстро (С данными параметрами на большом изображении 2,61 секунды на совмещение).

В итоге был реализован базовый алгоритм совмещения каналов изображения, который впоследствии был доработан для увеличения эффективности.

Контекстно-зависимое масштабирование изображений

Перед выполнением второго практического задания были просмотрены вторая и третья лекции. Из второй лекции я узнал о некоторых базовых определениях используемых в обработке изображений (примеры), а также о наиболее часто встречающихся дефектах (слабая контрастность, неправильные цвета, шумы) и способах их устранения (линейная тональная коррекция, робастная линейная коррекция, гамма коррекция, модель «серого мира» и более сложные модели для цветокоррекции). Также был рассмотрен способ выделения краев изображения — данная информация была полезна при выполнении практического задания. Из 3 лекции я узнал о сопоставлении изображений и нахождении особенностей изображения.

Во втором задании было необходимо ознакомится с алгоритмом контекстно-зависимого изменения размеров изображения (seam carving) и реализовать функцию уменьшающую изображение на 1 пиксель в ширину или высоту. Данный алгоритм позволяет изменять размер изображения сохраняя при этом размеры наиболее важных объектов — это одно из преимуществ алгоритма по отношению к стандартному подходу к масштабированию (при котором объекты на изображении изменяют свои размеры вместе с изображением). Второе преимущество заключается в том, что этот алгоритм позволяет с помощью маски выделять объекты, которые необходимо удалить из изображения (или оставить на изображении). Более продвинутые версии данного алгоритма используются в программах для редактирования изображений, также существуют версии алгоритма для масштабирования видеозаписей. Контекстно-зависимое изменение размеров изображения используемый на практике и полезный инструмент.

Сжатие изображения

Алгоритм сжатия заключается в нахождении швов, удаление которых будет наименее заметно. Швом называется связанная кривая, соединяющая первую и последнюю строчки или столбцы изображения. На каждом шаге необходимо удалять шов с минимальной энергией, где энергия каждого отдельного пикселя — модуль градиента в этой точке, энергия шва — сумма энергий точек в него входящих. Таким образом, тот шов который проходит через наименьшее количество перепадов яркости будет иметь меньшую энергию и будет удален раньше, тем самым удается сохранить объекты, которые имеют достаточно сложную структуру.

Так как на входе мы получаем цветное изображение, а для нахождения шва используем яркость, необходимо перевести изображение из цветовой модели RGB в YUV и взять компоненту Y — яркость изображения. По началу было решено использовать библиотечную функцию для пе-

ревода `skimage.color.rgb2yuv()`, в дальнейшем выяснилось, что при таком подходе находится не тот шов и было решено использовать формулу, после чего швы совпадали с сохраненными для проверки:

$$\mathrm{Y} = 0{,}299 \cdot \mathrm{R} + 0{,}587 \cdot \mathrm{G} + 0{,}114 \cdot \mathrm{B}$$

После того, как была получена яркость необходимо было посчитать градиент изображения, для этого нужно было пройтись по изображению, в каждой точке найти частную производную по x и по y, воспользовавшись формулами:

$$I'_x = \mathrm{Y}(x+1, y) - \mathrm{Y}(x-1, y)$$
$$I'_y = \mathrm{Y}(x, y+1) - \mathrm{Y}(x, y-1)$$

На границе изображения производные находятся с помощью аппроксимации первого порядка. Затем находим норму градиента: `img_energy` $= \sqrt{(I'_x)^2 + (I'_y)^2}$.

Для нахождения шва с минимальной энергией создаем копию матрицы с энергиями точек (можно создать пустую и скопировать первую строку). Далее заполняем с помощью динамического программирования матрицу и получаем в последней строке значения с энергиями некоторых швов. Среди них выбираем самую левую точку с наименьшим значением энергии (просто проходясь в цикле по последней строке). Затем восстанавливаем ответ, находя координаты каждой точки шва с нижней по верхнюю. В конце удаляем шов из изображения. Таким образом реализовано сжатие изображения по горизонтали. Чтобы сжать его по вертикали достаточно повернуть его на 90°. На удивление с поворотом изображения было больше всего проблем. Сначала было решено использовать функцию `scipy.ndimage.interpolation.rotate()`, которая вообще говоря наиболее универсальна (поворачивает на любой угол), поэтому работает с погрешностями при повороте изображения на 90°, что приводит к тому, что полученные швы не сходятся с проверочными. После обнаружения этой проблемы было решено использовать функцию `np.rot90()`, которая как оказалось тоже работает не точно и при проверке было обнаружено, что найденные швы отличаются от сохраненных для проверки в 3 – 4 точках (такие швы к сожалению считались не правильными, несмотря на то, что абсолютное большинство координат точек совпадало). В итоге решением этой проблемы было использование функции `np.swapaxis()`. Ниже представлены два изображения: до и после изменения размера с использованием реализованного алгоритма.

Рис. 1: Изображения до и после сжатия

Работа с маской и расширение изображения

С помощью маски можно контролировать выбор швов, искусственно уменьшая или увеличивая значения энергии точек на изображении (чтобы они попали в швы или наоборот не попали). Для удобства к алгоритму написанному ранее можно добавить маску и в случае, когда маску на входе не передают использовать маску с нулевыми значениями (которая просто ничего не будет изменять). Также чтобы добавить поддержку маски нужно при нахождении энергии изображения нужно учесть значения маски следующим образом: `img_energy += mask * image.shape[0] * image.shape[1] * 256`.

Рис. 2: Результат работы алгоритма с ошибкой

Для расширения изображения нужно снова найти шов с минимальной энергией и справа от него добавить новый, который является усреднением минимального и следующего за ним. Затем мы изменяем маску, добавляя в нее минимальный шов, чтобы избежать ситуации в которой минимальный шов при нескольких итерациях все время один и тот же. Если этого не сделать при увеличении изображения будем получать размытую область как на рисунке 2.

В итоге был реализован алгоритм контекстно-зависимого масштабирования изображений, прошедший тестирование. Стоит отметить, что данная реализация далеко не самая производительная так на удаление 10 швов уходит 20,2 секунд.

Распознавание автодорожных знаков

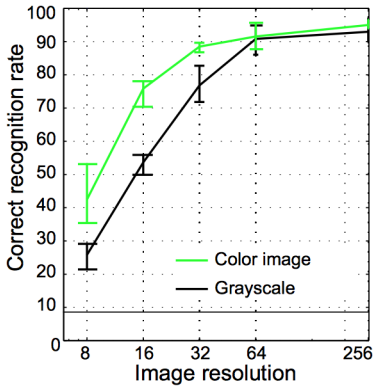

Рис. 3: График из лекции

Далее были просмотрены лекции о машинном обучении и классификации изображений. В 3 задании было необходимо реализовать классификатор автодорожных знаков на основе признаков HOG с помощью SVM.

Первый этап — предобработка изображений. Для начала отметим, что с погрешностью в 1 – 2 пикселя соотношения сторон во всех изображениях: 1 к 1. Так как все изображения различного размера, необходимо было обрезать и изменить их размер. Было решено использовать для классификации изображения в градациях серого, размером 64 на 64 пикселя. Такой выбор был осуществлен в связи с полученной из 5 лекции информацией о точности классификации изображений человеком в зависимости от размера и с учетом или без учета цвета. Понятно, что все изображения нужно привести к одному и тому же размеру, чтобы получить векторы одинаковой длины для дальнейшего обучения классификатора.

Функция извлечения признаков HOG

Опишем алгоритм извлечения признаков из изображения. Алгоритм извлечения признаков позволяет упростить изображение, извлекая полезную информацию и выкидывая излишнюю. На входе такие алгоритмы обычно получают изображения, а возвращают вектор признаков. В данной задаче нужно было написать дескриптор HOG (Histogram of Oriented Gradients). Основная информация, которую использует алгоритм — модули и направления градиентов изображения. После предобработки изображения, посчитаем частные производные `dx, dy` (для этого, чтобы ускорить работу алгоритма будем использовать функцию `np.gradient`), далее находим модули градиентов по формуле $\texttt{magnitude} = \sqrt{\texttt{dx}^2 + \texttt{dy}^2}$. Затем находим углы по формуле $\texttt{ang} = \arctan\left(\frac{\texttt{dy}}{\texttt{dx}}\right)$. Дополнительно можем зеркалировать направления векторов градиента. Теперь в каждой из ячеек изображения некоторой фиксированной ширины cellCols и высоты cellRows построим гистограммы направлений по следующему принципу: создаем вектор длиной в количество корзин, в каждом элементе вектора будем накапливать значения модулей векторов с соответствующими углами. Иными словами, если угол вектора равен углу одной из корзин, увеличиваем соответствующий этой корзине элемент вектора на значение его модуля, если же угол не совпадает ни с одной из корзин, делим значение модуля между двумя соседними корзинами в соотношении отклонений

от них. Наглядно алгоритм построения гистограммы выглядит следующим образом:

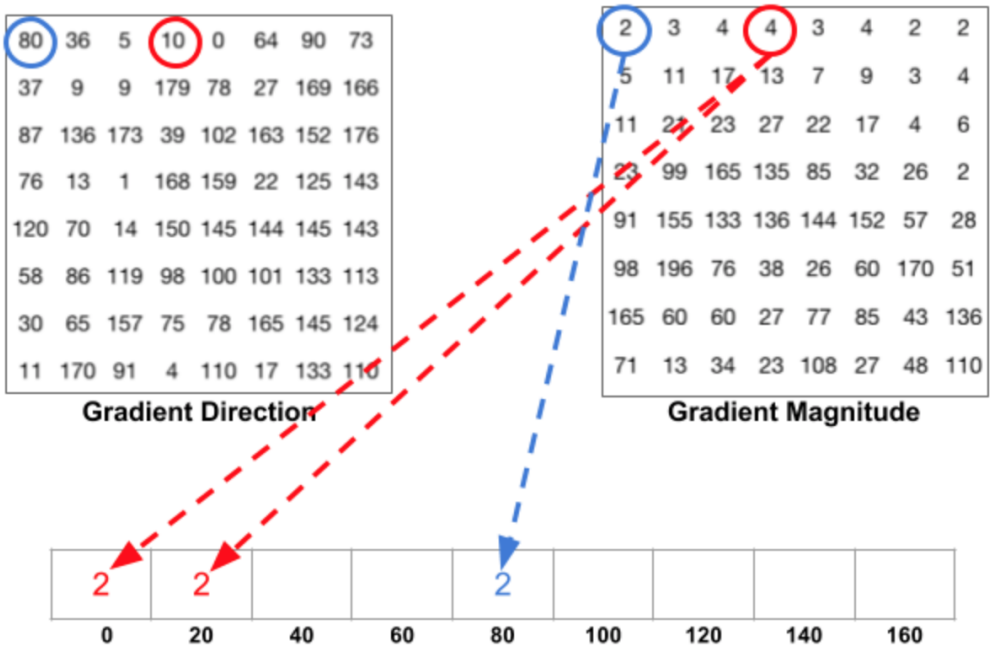

Рис. 4: Два шага алгоритма построения гистограммы. [3]

После этого полученные гистограммы необходимо нормировать, так как мы составляем их на основе модулей градиентов и в случае, если измениться яркость изображения, то и значения векторов (гистограмм) будут различны, что не желательно, так как с точки зрения классификации изображения должны быть одинаковы. Для этого объединяя ячейки в блоки по blockRows $\times$ blockColls ячеек и нормируем:

$$v = \frac{v}{\sqrt{|v|^2 + \varepsilon}}$$

Стоит отметить, что блоки мы формируем с наложением, то есть все ячейки кроме крайних участвуют в нескольких блоках.

Классификатор. Поиск оптимальных параметров

Для классификации изображений было предложено использовать метод опорных векторов. Для обучения классификатора и его дальнейшего тестирования была предоставлена выборка из 39209 изображений содержащих 43 различных класса дорожных знаков. Для менее точного, но значительно более быстрого тестирования и первоначального подбора параметров, выборка была случайно разбита на две части: тренировочную выборку и контрольную (обычно в отношении 0.7 к 0.3). Первоначальные параметры для дескриптора HOG были подобраны по аналогии с

теми, что используются в статье [2], как показала практика, с этими параметрами можно достичь высокой точности и при распознавании дорожных знаков.

Первоначально был опробован линейный SVM, со стандартным параметром C, точность классификации была порядка .97. В случае с линейным классификатором было замечено, что изменение параметра C не изменяет результат в лучшую сторону, это происходило по той причине, что данные были линейно разделимы (при описанных выше параметрах HOG). Несмотря на высокую точность при простой проверке, кросс-валидация показала, что с такими параметрами точность не достаточно высока (точнее была ниже .93 на 1%) При использовании ядра rbf с параметром $C = 150$ (изменения C в пределах ± 100 давали результат отличающийся по точности классификации лишь в тысячных) была получена достаточно высокая точность (на кросс-валидации $.933\cdots$). Как оказалось и при тестировании на скрытой выборке была получена высокая точность в .9312.

Заключение

В результате прохождения этой практики я приобрел базовые знания из области компьютерного зрения: получил информацию о развитии этой области компьютерных наук, узнал основные методы обработки изображений, а также получил более расширенное представление о машинном обучении и его приложениях в компьютерном зрении. Отдельно стоит отметить то, что выполнение практических заданий было хорошей практикой в использовании python и numpy, также в результате выполнения этих заданий я ближе познакомился с популярными библиотеками для обработки изображений и машинного обучения. Последнее, но не менее важное — практика в самостоятельном изучении ранее не знакомой дисциплины.

Все эти знания и практические навыки будут полезны в будущем при обучении на факультете и при самообразовании. Прохождение этой практики может в дальнейшем немного упростить освоение курсов по компьютерному зрению и машинному обучению, ведь зная некоторые основы области изучать её проще, также полезен будет и опыт использования python для выполнения практических заданий. Скорее всего большая часть, приобретенных в результате прохождения практики знаний, также пригодится и в дальнейшей работе, так как в современном мире машинное обучение и компьютерное зрение применяется практически повсеместно.

Так как курс, на базе которого построена практика предполагает введение в компьютерное зрение, понятно что полученные знания — лишь вершина айсберга. В дальнейшем мне хотелось бы более подробно и глубоко изучить эту область компьютерных наук.

Список используемых источников

[1] Library of congress. *Чудеса фотографии: восстановление фотографического наследия С. М. Прокудина-Горского*. URL: http://www.loc.gov/exhibits/empire/empire-ru.html.

[2] Navneet Dalal и Bill Triggs. “Histograms of Oriented Gradients for Human Detection”. В: *CVPR*. 2005, с. 886—893.

[3] Satya Mallick. *Histogram of Oriented Gradients*. URL: http://www.learnopencv.com/histogram-of-oriented-gradients/.

[4] scikit-image. *skimage source code*. URL: https://github.com/scikit-image/scikit-image.