Readme

Добрый день тут я объясняю, что это такое и с чем его едят :-D

В качестве основной среды разработки использовалась MS VS 2017 + Resharper
В качестве основного подхода используется "инверсия зависимостей (IoC)" + библиотека для IoC от MS - Unity версии 5.8.4
В качестве основного фреймверка для тестирования используется NUnit версии 3.10.1

А теперь пройдемся более подробно по системе

В системе 2 решения
1) Aspose.sln - основное,
2) Kernel.Project.sln - вспомогательное

Все проекты в решениях модифицированны руками чтобы при переносе с компьютера на компьютер не терять подгружаемые библиотеки в каждом файле проекта часть пути "..\..\packages\" заменена на "$(SolutionDir)packages\"
также в файлы проектов были добавлены Target's типа "AfterBuild" для модифицирования config файлов согласно окружения

Начнем пожалуй с вспомогательного решения :)
Kernel.Project.sln - решение которое в себе аккамулирует мои наработки на протяжении некоторого времени, в этом решении находяться распостроненные классы которые я часто использую в работе и у меня нет желания постоянно их писать заново в новых проектах, по сути Kernel.Project.sln состоит из 2 проектов которые с помощью инструмента NuGet Package Explorer преобразуються в nuget packages и публикуются на мой локальный nuget сервер http://nuget.ageron.info
1) Project.Kernel - основной проект в решении, хранит базовую функциональность, в виде nuget package подгружает основные рабочие библиотеки в базовый проект
2) Project.Kernel.Dal - вспомогательный проект, хранит в себе классы связаные с Data Access Layer, в виде nuget package подгружает библиотеки связанные с ORM Entity Framewerk 6.0

Теперь рассмотрим основное решение :)
В нем есть 4 типа окружения
Debug, Release, NUnit, vasiliy.stan
Debug, Release - базовые
NUnit, vasiliy.stan - это персонализированные окружения для рабочих целей

Также в нем есть 2 проекта
1) Dal - слой доступа к данным, в основном хранит классы для работы с локальным контекстом
2) Math.Kernel - базовая библиотека которая хоанит в себе основную логику по тестовому заданию

Теперь более подробно как выполнялось тестовое :)

Как вы понимаете все знать не возможно и т. к. я не специализировался в своей практике на обработке изображений, то пришлось обращаться к великому Google с вопросом, а что такое "подобие изображений" вообщем и как определяет это подобие математика в часности :)
Т. к. перед мной стояла задача не использовать базовые библиотеки по обработке изображений вроде OpenCv и им подобным, а реализация и обучение с 0 сверточной нейронной сети штука не простая, пришлось искать сравнительно простой класс алгоритмов которые на входе получали картинку, а на выходе нечеткую нормализированную, в пределах 0..1, меру для сравнения. Таким классом алгоритмов оказались т. н. перцептивные хеши

идея базового алгоритма перцептивного хеша по сути проста как 3 копейки :)

1) переводим изображение в 256 оттенков серого
2) накладываем фильтры на изображение
3) сжимаем изображение в квадрат 8x8
4) на основании алгоритма бинаризации вычисляем 64-битный хеш картинки
5) сравниваем 2 хеша картинок на основании кода Хемминга
6) вычисляем меру сравнения по простой формуле: мера сравнения = 1 - код Хемминга/64.0
где 64.0 - количество бит в хеше

но как всегда дьявол кроется в деталях и по сути сжатие до размера 8x8 пикселей уничтожает много необходимой информации, а значит сильно огрубляет меру, по этому я немного модифицировал базовый алгоритм и получил 2 собственных алгоритма

1) на основе маски собеля для выделения границ
2) на основании совмещения двух фильтров, размытия и повышения четкости, базовый пиксель в данном алгоритме вычисляеться по формуле
result[x,y]=c*high[x,y]-(1-c)*low[x,y]
где

c - коэффициент в пределах от 3/5 до 5/6
result - результирующее изображение
high - результат обработки исходного изображения фильтром повышения четкости
low - результат обработки исходного изображения фильтром размытия

Также был модифицирован и сам базовый алгоритм для более корректного вычисления меры

1) сжимаем изображение до размера 128x128
2) переводим изображение в 256 оттенков серого
3) накладываем фильтры на исходное изображение для получения основного изображения для обработки
5) на основании алгоритма Отцу вычисляем порог для бинаризации изображения
4) разбиваем изображение на тайтлы 8x8
5) н основании порога бинаризации вычисляем 64-битный хеш каждого тайтла
6) сравниваем 2 списка хешей картинок на основании кода Хемминга
7) вычисляем меру сравнения по простому алгоритму:
7.1) проходим по спискам хешей 2 картинок
7.2) суммируем коды хемминга для каждого тайтла
7.3) вычисляем меру для сравнения по формуле: мера = 1 - сумма кодов Хемминга/(64.0*256)
где 64.0 - количество бит в хеше
а 256 - количество тайтлов 8x8 в картинке 128x128

Также было достигнуто 90% покрытие тестами решения при этом
1) Dal - косвенное покрытие тестами 54%
2) Math.Kernel - основное покрытие тестами 99%, реализовано 84 теста

Основную часть дополнительного времени было потрачено на дополнительный анализ архитектуры кода, модификацию тестов, и рефакторинг

P. S. если у вас будут вопросы по проекту или просто пообщаться, то со мной можно связаться по email: vasiliyStankevich@gmail.com