Анализ изображений. Задание. Осень 2022.

1. Гамма коррекция *[1-е задание, срок сдачи: 29.09.2022]*

Требуется написать алгоритм гамма коррекции изображения:

$I' = a * I^b$

где:

- I - входное изображение, значения которого должны лежать в отрезке [0,1]
- a, b - параметры алгоритма
- I' - результат

2. Автоконтрастирование *[1-е задание, срок сдачи: 29.09.2022]*

Требуется написать алгоритм автоконтрастирования изображения:

```
def autocontrast(img: np.ndarray, white_percent: float, black_percent: float) -> np.ndarray:
```

Результат работы алгоритма должен соответствовать следующим требованиям (в порядке приоритета):

- black_percent - доля наиболее темных пикселей, которые нужно сделать черным (0) (например, 0.2)
- white_percent - доля наиболее светлых пикселей, которые нужно сделать белым (255) (например, 0.1)
- значения всех остальных пикселей нужно линейно растянуть на интервал (0, 255)

Если следующее требование выполнить невозможно (например, все пиксели после второго этапа уже черные или белые), алгоритм должен прекращать работу

3. Фильтр среднего *[2-е задание, срок сдачи: 13.10.2022]*

Требуется реализовать линейный фильтр изображения, усредняющий значения в заданной прямоугольной окрестности.

```
def box_filter(img: np.ndarray, w: int, h: int) -> np.ndarray
```

Параметры: (w, h) - размеры окна фильтра

Сложность алгоритма - не более, чем O(N), где N - количество пикселей входного изображения (не должна зависеть от размеров окна фильтра).

Для сравнения результата работы можно использовать cv2.blur. Граничные условия произвольные.

4. Бинаризация Отсу *[2-е задание, срок сдачи: 13.10.2022]*

Требуется реализовать алгоритм поиска порога бинаризации Отсу и провести бинаризацию входного изображения по этому порогу

```
def otsu(img: np.ndarray) -> np.ndarray:
```

Алгоритм не имеет параметров.

Для сравнения можно воспользоваться функцией cv2.threshold с параметром cv2.THRESH_OTSU

5. Преобразование Хафа *[3-е задание, срок сдачи: 27.10.2022]*

На вход подается модуль градиента серого изображения (уже реализовано в шаблоне). Требуется реализовать 2 функции: преобразование Хафа (1) и поиск прямых линий (2) с его помощью.

5.1. Преобразование Хафа

```
def hough_transform(
        img: np.ndarray, theta: float, rho: float
) -> (np.ndarray, list, list)
```

Параметры:

- **img** - входное изображение (границы, полученные как модуль градиента)
- **theta** - шаг по оси углов (расстояние между двумя ближайшими углами в пространстве Хафа), в радианах

- **rho** - шаг по оси расстояния (аналогично **theta**, но в пикселях)
- **ht_map** [out] - построенное пространство Хафа; ht_map.shape = len(rhos), len(thetas)
- **thetas** [out] - ось углов
- **rhos** [out] - ось расстояния

5.2. Поиск прямых

На вход функции подается посчитанное пространство Хафа (**ht_map**) и полученные значения **rhos** и **thetas**.

По нему требуется найти **n_lines** наиболее выраженных прямых, перевести в вид *y=kx+b* и вернуть список параметров *(k_i, b_i)*.

Также требуется, чтобы в результате не возникало прямых, близких друг к другу. Для этого вводятся дополнительные параметры:

- **min_delta_rho** - минимальное расстояние между двумя ближайшими прямыми (в пикселях, как и **rho**)
- **min_delta_theta** - минимальный угол между двумя ближайшими прямыми (в радианах, как и **theta**)

```
def get_lines(
        ht_map: np.ndarray, n_lines: int,
        thetas: list, rhos: list,
        min_delta_rho: float, min_delta_theta: float
) -> list
```

6. RANSAC *[4-е задание, срок сдачи: 10.11.2022]*

Необходимо реализовать генерацию зашумленных данных и поиск на них прямой с использованием алгоритма RANSAC

Для выполнения нужно реализовать 4 функции:

1. **generate_data** – генерация зашумленных данных
2. **compute_ransac_threshold** – вычисление порогового значения для оценки соответствия отдельной точки посчитанной модели
3. **compute_ransac_iter_count** – вычисление необходимого для сходимости количества итераций
4. **compute_line_ransac** – подбор параметров модели с использованием алгоритма RANSAC

6.1 generate_data

```
def generate_data(
        img_size: tuple, line_params: tuple,
        n_points: int, sigma: float, inlier_ratio: float
) -> np.ndarray
```

Требуется:

Построить набор из **n_points** точек в области **img_size**, таких, что:

- **inlier_ratio** из них соответствовали бы модели **line_params**, но были бы зашумлены нормальным шумом с дисперсией **sigma** и нулевым матожиданием
- (1 - **inlier_ratio**) были бы равномерно распределены на области **img_size** (не соответствовали бы модели)

Параметры:

- **img_size** (**W**x**H**) – размер прямоугольной области, в которой расположены все данные
- **line_params** (**a**, **b**, **c**) – параметры прямой **a**x+**b**y+**c**=0, которой должны соответстовать сгенерированные данные
- **n_points** – требуемое количество точек
- **sigma** – дисперсия нормального распределения шума для точек, соответствующих модели
- **inlier_ratio** – доля точек (из всех), соответствующих модели
- **data** [out] – зашумленный набор точек, соответствующих модели прямой

6.2 compute_ransac_threshold

```
def compute_ransac_threshold(
        alpha: float, sigma: float
) -> float
```

Требуется: Определить пороговое значение расстояния, по которому определяется, соответствует ли точка некоторой модели или нет

Параметры:

- **alpha** – требуемая вероятность рассмотрения точки как соответствующей модели, если она действительно соответствует модели
- **sigma** – дисперсия нормального распределения шума для точек, соответствующих модели
- **threshold** [out] – пороговое значение для определения соответствия точки и модели

6.3 compute_ransac_iter_count

```
def compute_ransac_iter_count(
        conv_prob: float, inlier_ratio: float
) -> int
```

Требуется: Определить количество итераций, необходимых RANSAC, чтобы сойтись с вероятностью **conv_prob**

Параметры:

- **conv_prob** – вероятность, с которой алгоритм сойдется к модели, по которой построены данные
- **inlier_ratio** – доля точек (из всех), соответствующих модели
- **iter_count** [out] – необходимое количество итераций

6.4 compute_line_ransac

```
def compute_line_ransac(
        data: np.ndarray, threshold: float, iter_count: int
) -> tuple
```

Требуется: Реализовать алгоритм RANSAC для поиска модели прямой на полученных зашумленных данных

Параметры:

- **data** – зашумленный набор точек, соответствующих модели прямой
- **threshold** – пороговое значение для определения соответствия точки и модели
- **iter_count** – необходимое количество итераций
- **line** [out] – оценка параметров модели прямой

7. Metric learning (TBD) *[5-е задание, срок сдачи: 24.11.2022]*