**Субтитры ИИ\_24 «В поисках цвета: как находить объекты на изображениях»**

Интересно, а можно ли научить компьютер находить объекты на изображениях по цвету?

Так же, как люди быстро находят все желтые машины на улице или зеленые яблоки в магазине.

Ваня, это уже реализовано в различных приложениях и технологиях компьютерного зрения. И ты уже знаком с одним из таких решений.

А, да точно, я же начал изучать библиотеку OpenCV.

OpenCV часто используется для поиска объектов по цвету с помощью метода пороговой сегментации.

Пиксели выше порога считаются частью объекта, а пиксели ниже — частью фона.

Например, мы хотим найти красные объекты. Нам нужно указать оттенки красного и пороговая сегментация отметит все пиксели на изображении в заданном диапазоне.

Я хочу попробовать метод пороговой сегментации на практике!

Отлично!

Начнем с разработки плана действий для поиска объектов на изображениях. Обратимся к Гигачату.

Прежде чем приступить к преобразованию изображения в нужное цветовое пространство, необходимо выполнить следующие шаги. Создать новую папку «Сегментация».

В Visual Studio Code в левом верхнем углу нажать кнопку «Файл», выбрать «Открыть папку», затем выбрать на компьютере созданную папку «Сегментация» и нажать «Выбор папки».

Создать в этой же папке новый файл с расширением .py, а также добавить в эту папку все необходимые для урока изображения.

Убедиться, что установлены библиотеки OpenCV, MatPlotLib и Numpy.

Если matplotlib или numpy еще не установлены — воспользуйтесь командами для установки (или проверки наличия) данных библиотек.

Они находятся в материалах к занятию.

Я помню, что модель RGB наиболее популярна, а о других я даже не слышал.

Давай это исправлять!

Цветовое пространство HSV описывает цвет с помощью трех параметров:

Возьмем одну из подготовленных картинок под названием tiger1.png и преобразуем ее в цветовое пространство HSV.

Все картинки и дальнейший код доступны в материалах к занятию.

Вау, на картинке изображен бенгальский тигр!

У него яркий оранжевый окрас с черными полосами.  Мы будем фильтровать эти цвета на изображении?

Все верно! Нас интересуют оттенки оранжевого и белого, а темные участки пока учитывать не будем.

Для начала подключим библиотеки. OpenCV — для обработки нашего изображения, MatPlotLib — для построения графиков и NumPy — для вычислений.

OpenCV по умолчанию использует цветовое пространство BGR, оно аналогично RGB, но цвета расположены в другом порядке.

Нам нужно преобразовать изображение сначала из BGR в RGB, а затем в HSV, чтобы цвета отображались правильно.

Если сразу преобразовать из BGR в HSV, цвета могут измениться неправильно, и тигр может стать синим вместо оранжевого.

С помощью функции «imread» загрузим изображение в формате BGR.

А дальше выполним два преобразования: из BGR в RGB, из RGB в HSV.

Ой, это звучит как-то страшно и сложно! А для чего вообще нужен этот график?

Трехмерный график состоит из точек, каждая из которых соответствует цвету изображения.

Они расположены в пространстве по трем осям, которые отражают характеристики цвета: (H – hue) Оттенок, (S – saturation) Насыщенность, (V – value) Яркость.

Благодаря такому графику мы сможем найти пороговые значения, необходимые для создания маски объекта. Но это мы сделаем уже в следующем шаге. А сейчас давай займемся графиком.

Для этого напишем такую часть кода:

Действительно, несложно! А как мы увидим оттенки оранжевого и белого на этом графике?

Чтобы увидеть наши цвета на графике, мы заранее раскрасим точки в нужные оттенки, используя переменную “pixel\_colors”.

Когда мы построим график, точки будут в правильных цветах, и мы сможем легко различить оттенки оранжевого и белого.

Сколько строк!  Подожди, я хочу разобраться — напишу Гигачату.

Посмотри, какой график получился.

Классный! Видно, что оранжевый цвет скапливается в одном месте. А как выбрать нужный диапазон?

Для определения нужного диапазона оранжевого цвета нам нужно установить пороговые значения.

Давай оценим значения по осям на графике.

Чтобы поворачивать график, двигайте мышь с зажатой левой кнопкой. Это может происходить с небольшой задержкой — это нормально.

Посмотрим на график под разными углами. По оси оттенка нужные значения оранжевого лежат в диапазоне от 10 до 20.

По оси насыщенности подходят значения от 80 до 255.

По оси яркости подходят значения от 0 до 255.

Мы охватываем весь диапазон яркости, так как на тигре есть черные полосы и темные оттенки оранжевого.

Для более точных значений посмотрите в правый нижний угол графика, наведя мышкой на нужную область графика. Обращайте внимание только на целые числа.

Тогда нужно составить светлую и темную версию, прописав в каждом три целых числа!

Верно, тогда мы получаем значения нижнего и верхнего порога оранжевого цвета.

Примерно такие пороговые значения для белого цвета у вас должны получиться.

Теперь, когда у нас есть пороговые значения для оранжевого и белого цветов, перейдем к сегментации изображения по выбранным цветам.

Обновленный код доступен в материалах к занятию.

На основе этих диапазонов создадим маски, которые будут накладываться на наше изображение, оставляя только выбранные цвета.

"final\_mask" забавная переменная! Это объединение масок оранжевого и белого для того, чтобы оба этих цвета были видны в итоговом результате?

Именно так! И затем сформируем итоговое изображение, применив маски, и отобразим их на экране.

Далее запустим код и посмотрим результат.

На изображении видны не только части тигра, но и другие элементы из-за совпадения цветов объекта с фоном. Этот эффект можно частично устранить более точным выбором цветовых диапазонов, но если такие же цвета встречаются где-то еще, они все равно будут отображены.

Прикольно, я теперь знаю, как научить компьютер находить объекты на изображениях по цвету!

Вы хорошо потрудились! Увидимся на следующем занятии.