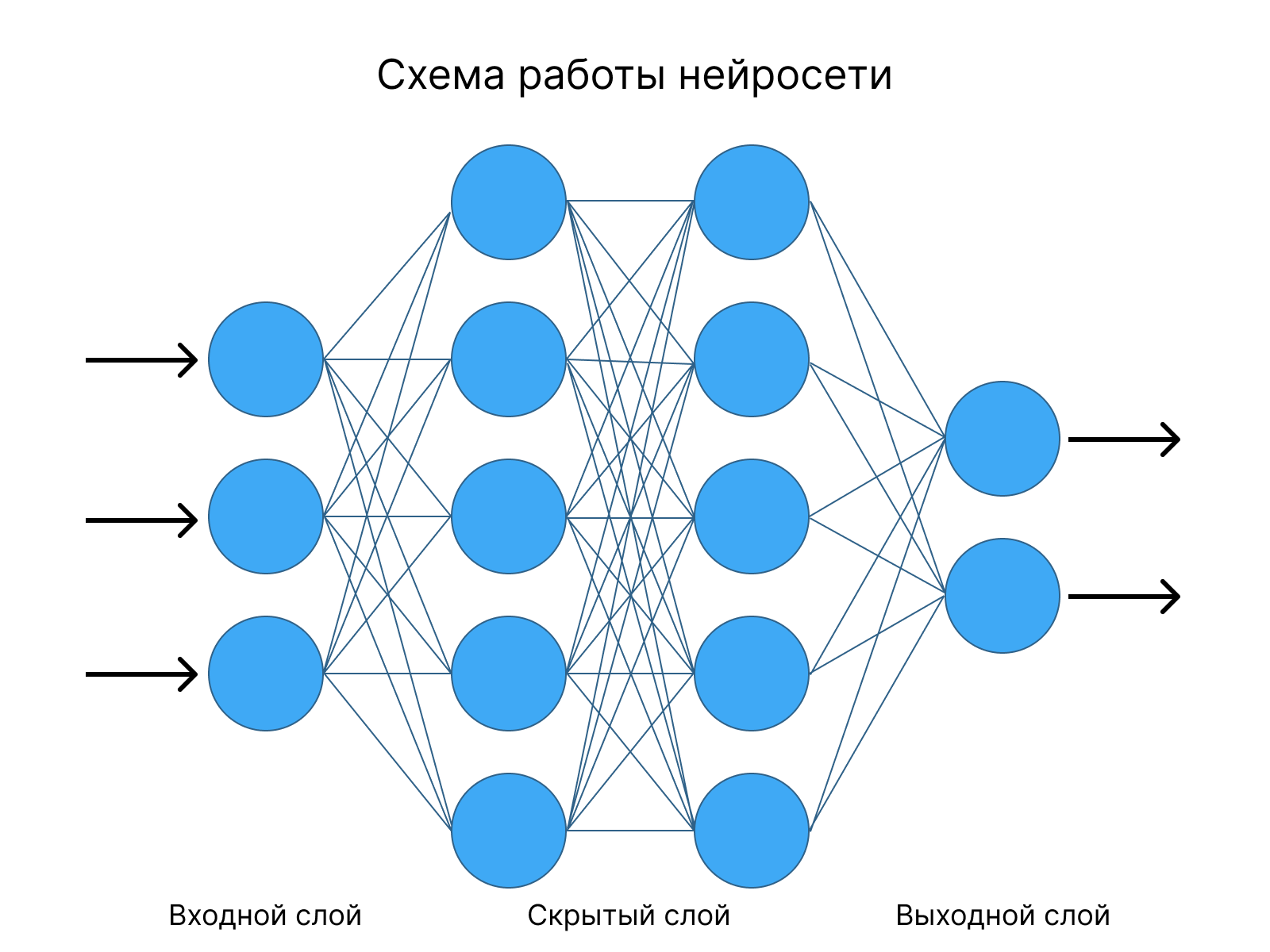
**Нейронные сети**

Будем работать еще с **num.py** модуль для обработки большого кол-ва данных, в массивах и тд.

И TensorFlow.

Крч чуть основ.



Крч входной слой обычно x, туда уходят все входные данные, потом они податься с помощью весов (стрелок) – алгоритмов в скрытые слои, которые составляют нейроны, после прогона, результат со скрытого слоя уходит в выходной.

Функция активации – это функция которая активирует нейроны, в начальном слое при подаче данных.

Интерация это счетчик который считает кол-во шагов.

Нейрон смещения -

Качаем **pip install opencv-python**

Num py докачиваеться автоматически.

import cv2

импортим opencv

img = cv2.imread("Photo\1.jpg")

объявляем переменную для чтения изображения методом imread и указываем путь к файлу.

Далее мы должны открыть это фото

cv2.imshow("Result", img)

указываем 2 аргументы, первый название, любое, второе, само фото.

Далее мы должны указать время сколько фото будет открыто вот такой командой

cv2.waitKey(0)

внутри аргумент 0 указывает что она будет отображаться бесконечно, или можно поставить время в милли секундах 10000 – 10 сек.

Далее откроем видео.

cap = cv2.VideoCapture(r"Video\1.mp4")

для отображения видео нужно создать цикл ведь это набор картинок идущие друг за другом.

cap = cv2.VideoCapture(0)

если прописан 0, то будет считываться запись с вебки, если 1 то ко второй и тд…

succses, img = cap.read()

cv2.imshow("res", img)

    if cv2.waitKey(1) & 0xFF == ord("q"):

        break

далее объявляем две переменные, в первую автоматом уходит значение true или false, которое говорить смогли ли мы прочитать изображение.

В img уходит все наши фотки с видео

Далее мы читаем изображение и показываем его, но работать ни хрена не будет пока не укажем тайминг появления фото в вайт кей, и укажем кнопку для выхода из цикла, чтобы мы смогли закрыть наше видео.

cap.set(3, 1280)

cap.set(4, 720)

далее можем задать разрешения видео.

Первый параметр id метода 3 – ширина, 4 – высота.

print(img.shape)

чтобы получить данные о картинки, кортеж вернеться (720, 1280, 3)

3 параметр это кол-во слоев цвето, 3 – это rgb

Чтобы изменить размер картинки

new\_img = cv2.resize(img, (1280, 720))

cv2.imshow('Result', new\_img)

создаем новую переменную.

И метод resize.

Можно это и в одной переменной делать.

cv2.imshow('Result', new\_img[0:100, 0:150])

обрезание картинки.

img = cv2.GaussianBlur(img, (15, 15), 0)

этот метод размывает изображение в кортеже указываем степень размытия по гаусу, важно что туда не идут все четные числа, так как внутреняя формула их делит на цело.

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

чтобы убрать каналы изображения, и сделать ее чб.

img = cv2.Canny(img, 20, 20)

перевод картинки в бинарный формат, обводит контур. Лучше конечно же перекидывать ее в чб и потом переводить в бинарный формат, 2 параметра это пороги, лучше с этим поиграться.

Мы можем увеличить обводку.

Сначала импортим numpy чтобы создать матрицу для обводки, потом записываем ее в karnel, потом методом dilate прописываем ширину обводки, принимает 3 параметра, наше фото, матрицу и кол-во повторений оводки, чем число больше, тем логично и обводка больше.

karnel = np.ones((5, 5), np.uint8)

img = cv2.dilate(img, karnel, iterations=1)

И после обводки как не странно мы должны ее уменьшить, зачем, потому что с обводкой все углы фото сошлись, а сейчас мы ее просто уменьшим для качества.

img = cv2.erode(img, karnel, iterations=1)

Чтобы создать свое изображение нужно создать матрицу, логично юзаем numpy.

my\_photo = np.zeros((300, 300, 3), dtype = "uint8")

zeros позволяет создать матрицу где все элементы = 0, первые два параметра в кортеже это кол-во элементов в матрице, т.е 300 эл и в них еще 300, по факту это разрешение матрицы в пикселях, где один элемент один пиксель, 0 черный 1 белый, далее 3 это кол-во слоев, как помним 3 это rgb, dtype чтобы все ок обрабатывалось.

cv2.imshow("MY\_PHOTO", my\_photo)

cv2.waitKey(0)

выводим фото

чтобы окрасить наше фото в другой цвет мы должны обраться по срезу к его пикселям, эл матрицы.

my\_photo[:] = 255, 0, 0

к элементам обращаться можно разным, [:] – это выбрать все, далее идет кортеж из каналов матрицы где первое значение это Blue второе Green и третье Red, они перевернули rgb в bgr.

my\_photo[100:150, 200:250] = 105, 120, 20

вот так можно окрашивать что-то на картинке.

Первые две точки это высота, вторые две ширина.

И тут интересный момент, чтобы нам самим не париться с созданием фигур на холсте, есть внутряковые методы которые нам помогут это сделать.

cv2.rectangle(my\_photo, (0, 0), (100, 100), (105, 120, 20), thickness=1)

разберем

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Получили квадрат без закрашивания,

Первый параметр, обращение к объекту где фото, второй, это точка, от кеда начинаем, третий до кеда. Потом в кортеже указываем цвет обводки, и thickness = это плотность обводки.

 thickness=cv2.FILLED

если указать так, то будет полный закрас.

cv2.line(my\_photo, (0, 50), (500, 0), (105, 120, 20), thickness=3)

Это метод для создания линии, первые два кортежа точки a и b.

cv2.circle(my\_photo, (250, 250), 50, (105, 120, 20), thickness=3)

Для круга

Можно брать my\_photo.shape[0] или 1 для того чтобы получить размер фото, 0 вроде горизонт, 1 фронт.

cv2.putText(my\_photo, "Butterflue", (100, 50), cv2.FONT\_HERSHEY\_TRIPLEX, 2, (0, 255, 0), thickness=3)

чтобы вставить текст,

указывает текст, после него координаты в одном кортеже по ширине и высоте, далее шрифт, потом длину текста, потом цвет и обводку.

img = cv2.flip(img, -1)

метод для переворота или отзеркаливания фото, принимает 1 0 -1.

def rotate(img, angle: int = 180):

    height, widht = img.shape[:2]

    point = (widht // 2, height // 2)

    mat = cv2.getRotationMatrix2D(point, angle, 1)

    return cv2.warpAffine(img, mat, (1280, 720))

img = rotate(img, angle=80)

вращение изображения, создали функцию можно и без нее, первый параметр наше фото, второй угл.

Создали две переменные в которые положили высоту и ширину созображения шейпом, потом создали точку вращения, ровно по центру, далее создали матрицу интересным методом, в которую передали точку вращения, угл и на сколько увеличим фото при вращении.

Далее мы возвращаем перевернутое фото с методом новым, где передаем матрицу, фото и размеры фото.

def transform(img\_param, x, y):

    mat = np.float32([[1, 0, x], [0, 1, y]])

    return cv2.warpAffine(img, mat, (1280, 720))

img = transform(img, 30, 200)

Вот функция для смещения фото, думаю все понятно, потом не бомби мне влом писать, разберешься. Главное запомнить что выполнение функции присваивать к переменной с фото надо.

Теперь мы будем получать контуры изображения, так же как там только + мы получим позицию их в цифровом формате.

img = cv2.imread(r"C:\Users\Nike\Desktop\Scripts\Python\Neural Network\Photo\1.jpg")

img = cv2.resize(img, (1280, 720))

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

img = cv2.GaussianBlur(img, (5, 5), 0)

img = cv2.Canny(img, 100, 150)

con, hir = cv2.findContours(img, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_NONE)

print(con)

итак процесс тот же самый что и раньше только добавляем новый метод, который присваиваем у двум переменным, 1-я con это точки, 2-я hir объекты, далее в сам метод идет изображение, потом указываем что ищем все координаты, и последний метод что ищем все внутренние точки с обводкой.

Теперь мы когда получили все кординаты фото, мы сможем нарисовать новое по ним

new\_img = np.zeros(img.shape, dtype="uint8")

cv2.drawContours(new\_img, con, -1, (255, 0, 0), thickness=1)

метод для перевода фото из GBR в HSV это другой режим цветов.

img = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

или в LAB

img = cv2.cvtColor(img, cv2.COLOR\_BGR2LAB)

В будущем может возникнуть проблема, в чем, в том что если мы попробуем сконвертировать другими библиотеками типо pillow наше фото, в rgb то будет все плохо, потому-что оно в BGR и они будут неправильно сконвертированы.

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

и мы юзаем метод выше который нам в этом поможет.

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

r, g, b = cv2.split(img)

cv2.imshow("RES", g)

мы можем разбить картинку на слои и вывести их, методом сплит.

img = cv2.merge([r, g, b])

этот метод позволяет заново собрать в переменную все слои фото, или часть из них.

img = cv2.merge([b, r, b])

ток переменных должно быть три, картинка не может состоять из двух слоев.

ПОБИТОВЫЕ ОПЕРАЦИИ

Объединение картинок.

img = np.zeros((500, 500), dtype="uint8")

создадим фото.

circle = cv2.circle(img.copy(), (250, 250), 50, 255, -1)

мы создали круг, но не этом фото, а на его копии, чтобы круг был как бы отдельно, потом указали центр, и точку радиуса, 255 это цвет, конечно мы могли указать 255,255,255 но мы взяли только один карсный канал, так тоже можно и указали что обводка идет в -1

cv2.imshow("RES", circle)

вот так их выводим

img = cv2.bitwise\_and(circle, rec)

теперь мы объединили круг и квадрат, этот метод находит одинаковые части фото, и показывает их пересечение.

img = cv2.bitwise\_or(circle, rec)

вот or показывает все фото, с объединением.

img = cv2.bitwise\_xor(circle, rec)

а этот метод, объединил два объекта, но общую их часть удалил.

img = cv2.bitwise\_not(circle, rec)

а этот метод not делает инверсию объектов фото.

img = cv2.bitwise\_and(photo, photo, mask=rec)

вот так вот мы создали маску.