ComputerVision

January 26, 2024

1 Computer Vision

1.0.1 "Data Science Applications in Physics" *Balkan School in Tirana 2024*

What is Computer Vision (CV)? Is interdisciplinary field of AI that: * Focuses on enabling computers to acquire, process, analyze, and understand digital images and videos. * Develops algorithms that gives computers high-level understanding from visual data (digital images & videos) to produce numerical or symbolic information. * Seeks to automate tasks that the human visual system can do.

What are the tasks of Computer Vision?

- Image classification,
- object detection,
- image segmentation,
- facial recognition,
- scene understanding, etc…

Computer Vision Applications

- Self-driving cars,
- surveillance systems,
- medical imaging,
- augmented reality,
- robotics
- quality control in manufacturing
- **PHYSICS Experiments**

1.1 Requirements!

Here we will use python API of two popular libraries: 1. **OpenCV** (https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html) 2. **YOLOv8** from ultralytics (https://docs.ultralytics.com/)

** Prerequisites: Python & numpy ** For quick introduction to Python see e.g. https://python.swaroopch.com/ or https://www.freecodecamp.org/news/the-python-guide-for-beginners/

For a quick start with **Numpy** see e.g. https://numpy.org/doc/stable/user/quickstart.html

1.1.1 Installation

Assuming you have a conda environment up and running you should create a new environment by running the following command in the terminal

```
conda create --name cv
```

then activate the new environment

```
conda activate cv
```

then install **OpenCV**

```
conda install -c conda-forge opencv
```

Install **Ultralytics** by excecuting the following command in the terminal:

```
pip install ultralytics
```

```
conda install pytorch torchvision ultralytics
```

1.2 Camera and its Sensor

A typical digital camera has an **optical system** an **image sensor** a **data aquisition circuit and a digital image processing unit** (+ a display).

This is how an image sensor (camera sensor) looks like

The pixels sensitive to different colors have different arrangements. The most common is the Bayer pattern:

Pixels are semiconductors (photo-sensitive p-n junctions)

Typical pixel quantum efficiency (sensitive spectrum)

1.3 Demonstration

Load an Image

```
import cv2
img = cv2.imread("DATA/buss.jpg") # loads the image in BGR color scheem
cv2.imshow("Display window", img) # shows the image in a dedicated canvas
```

An image is nothing but a matrix of numbers with values 0-255 (8-bit color depth). The dimensions of this matrix are the resolution of the image $N \times M \times 3$ for three colors Red (R), Green (G) and Blue (B).

[ ]:
```python
import cv2
img = cv2.imread("DATA/bus.jpg")
#cv2.imshow("Display window", img)
img.shape
print(f'Showing first 10x10 pixels of the image for each color\n\nBlue={img[:
↪10,:10,0]}\n\n\nGreen={img[:10,:10,1]}\n\n\nRed={img[:10,:10,2]}')
```

1.3.1 Load Videos or Cameras

We can show video from camera, loaded from disk or from internet e.g. from youtube. Check the following example.

```python
import cv2
cap = cv.VideoCapture(0) #camera 0 is the first, 1 is second etc... Changes this to select the
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    # Display the resulting frame
    cv.imshow('frame', frame)
    if cv.waitKey(1) == ord('q'):
        break
# When everything done, release the capture
cap.release()
cv.destroyAllWindows()
```

If you want to change the image into Gray:

```python
import cv2
cap = cv.VideoCapture(0) #camera 0 is the first, 1 is second etc... Changes this to select the
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    # Our operations on the frame come here
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # Display the resulting frame
    cv.imshow('frame', gray)
    if cv.waitKey(1) == ord('q'):
        break
# When everything done, release the capture
cap.release()
cv.destroyAllWindows()
```

[ ]:
```python
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Cannot open camera")
```

```python
    exit()
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    # Display the resulting frame
    # cv2.imshow('frame', frame) # show only the Blue part
    # cv2.imshow('frame', frame[:,:,0]) # show only the Blue part
    # cv2.imshow('frame', frame[:,:,1]) # show only the Green part
    cv2.imshow('frame', frame[:,:,2]) # show only the Red part
    if cv2.waitKey(1) == ord('q'):
        break
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

1.4 Controlling camera properties

We can controll camera settings via OpenCV. If you want to show camera settings in linux you can use e.g. `v4l2-ctl` (see `man v4l2-ctl` for help). Here we are interested in changing the frame rate, thus use:

```
**v4l2-ctl --list-formats-ext -d /dev/video0**
```

change 0 in `/dev/video0` in to whatever camera you are using, e.g. `/dev/video2` etc. The result of the command for the camera I am using here is:

1.4.1 [0]: 'MJPG' (Motion-JPEG, compressed)

```
**1920x1080 (30.000 fps)**
**1280x720  (60.000 fps)**
**1024x768  (30.000 fps)**
**640x480 (120.101 fps)**
**800x600  (60.000 fps)**
**1280x1024 (30.000 fps)**
**320x240 (120.101 fps)**
```

1.4.2 [1]: 'YUYV' (YUYV 4:2:2)

```
**1920x1080 (6.000 fps)**
**1280x720  (9.000 fps)**
**1024x768  (6.000 fps)**
** 640x480 (30.000 fps)**
** 800x600 (20.000 fps)**
```

```
**1280x1024 (6.000 fps)**
* *320x240 (30.000 fps)**
```

Therefore we can use one of these options, e.g. let us make a 6 fps

[ ]:
```
import cv2
cap = cv2.VideoCapture(0)

# cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('M','J','P','G'))
# cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
# cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
# cap.set(cv2.CAP_PROP_FPS, 60)

cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('Y','U','Y','V'))
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1024)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 768)
cap.set(cv2.CAP_PROP_FPS, 6)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) == ord('q'):
        break
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Load video from source Using some video file you can load it directly

[ ]:
```
import cv2
cap = cv2.VideoCapture("DATA/traffic.mp4")
if not cap.isOpened():
    print("Cannot open camera")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    cv2.imshow('frame', frame) # show only the Blue part
    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

1.5 Image processing

If you are interested in image processing (a very important topic) you have the following tutorials:
* **OpenCV**: https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html
* **scikit-image**: https://scikit-image.org/docs/dev/auto_examples/

Here you can every image manipilation such as: * Changing Colorspace * Geometry Transformations * Image Thresholding * Smoothing Images * Morphological Transformations * Image Gradients * Edge Detection * Contours & Histograms * Template Matching * Hough Line & Circle Transforms * Image Segmentation * etc …

1.6 Object Detection and Tracking with Neural Networks

A simple code how to use YOLOv8

```
from ultralytics import YOLO

# Load model
model = YOLO('yolov8n.pt')  # Load an official Detect model
#model = YOLO('yolov8n-seg.pt')  # Load an official Segment model
#model = YOLO('yolov8n-pose.pt')  # Load an official Pose model

src = "DATA/airplain.jpg"

results = model.track(source=src, show=True)  # Tracking with default tracker
#results = model.track(source=src, show=True, tracker="bytetrack.yaml")  # Tracking with ByteT
```

[ ]:
```
import cv2
from ultralytics import YOLO

# Load the YOLOv8 model
model = YOLO('yolov8n.pt')  # Load an official Detect model
#model = YOLO('yolov8n-seg.pt')  # Load an official Segment model
#model = YOLO('yolov8n-pose.pt')  # Load an official Pose model


#src = "https://www.youtube.com/watch?v=xRdQmO9qLHo"
#src = "DATA/dance.mp4"
src = "DATA/traffic.mp4"
#src = "DATA/peopleWalking.mp4"
#src = "DATA/airplain.jpg"

cap = cv2.VideoCapture(src)

# Loop through the video frames
while cap.isOpened():
    # Read a frame from the video
    success, frame = cap.read()
```

```
    if success:
        # Run YOLOv8 tracking on the frame, persisting tracks between frames
        #results = model.track(frame, persist=True, tracker="botsort.yaml",␣
↪conf=0.3, iou=0.5)
        results = model.track(frame, persist=True, tracker="bytetrack.yaml",␣
↪conf=0.6, iou=0.5)

        # Visualize the results on the frame
        annotated_frame = results[0].plot()

        # Display the annotated frame
        cv2.imshow("YOLOv8 Tracking", annotated_frame)
        #cv2.imshow("YOLOv8 Tracking", frame)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    else:
        # Break the loop if the end of the video is reached
        break

# Release the video capture object and close the display window
cap.release()
cv2.destroyAllWindows()
```

1.6.1 Use camera to detect objects from pretrained categories

Use pretrained model classes of objects to detect.

[ ]:
```
from ultralytics import YOLO
import cv2
import math
# start webcam
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

# model
model = YOLO("yolo-Weights/yolov8n.pt")

# object classes
classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus",␣
↪"train", "truck", "boat",
```

```
              "traffic light", "fire hydrant", "stop sign", "parking meter",␣
↪"bench", "bird", "cat",
              "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra",␣
↪"giraffe", "backpack", "umbrella",
              "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard",␣
↪"sports ball", "kite", "baseball bat",
              "baseball glove", "skateboard", "surfboard", "tennis racket",␣
↪"bottle", "wine glass", "cup",
              "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich",␣
↪"orange", "broccoli",
              "carrot", "hot dog", "pizza", "donut", "cake", "chair", "sofa",␣
↪"pottedplant", "bed",
              "diningtable", "toilet", "tvmonitor", "laptop", "mouse",␣
↪"remote", "keyboard", "cell phone",
              "microwave", "oven", "toaster", "sink", "refrigerator", "book",␣
↪"clock", "vase", "scissors",
              "teddy bear", "hair drier", "toothbrush"
              ]


while True:
    success, img = cap.read()
    results = model(img, stream=True)

    # coordinates
    for r in results:
        boxes = r.boxes

        for box in boxes:
            # bounding box
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2) # convert to␣
↪int values

            # put box in cam
            cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 255), 3)

            # confidence
            confidence = math.ceil((box.conf[0]*100))/100
            print("Confidence --->",confidence)

            # class name
            cls = int(box.cls[0])
            print("Class name -->", classNames[cls])

            # object details
```

```python
            org = [x1, y1]
            font = cv2.FONT_HERSHEY_SIMPLEX
            fontScale = 1
            color = (255, 100, 0)
            thickness = 2

            cv2.putText(img, classNames[cls], org, font, fontScale, color,␣
↪thickness)

    cv2.imshow('Webcam', img)
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

1.7 Train and Validate and Load Your Model

This is how you load a model, train it, evaluate its performance on a validation set, and even export it to ONNX format.

```python
from ultralytics import YOLO

# Create a new YOLO model from scratch
model = YOLO('yolov8n.yaml')

# Load a pretrained YOLO model (recommended for training)
model = YOLO('yolov8n.pt')

# Train the model using the 'coco128.yaml' dataset for 3 epochs
results = model.train(data='coco128.yaml', epochs=3)

# Evaluate the model's performance on the validation set
results = model.val()

# Perform object detection on an image using the model
results = model('https://ultralytics.com/images/bus.jpg')

# Export the model to ONNX format
success = model.export(format='onnx')
```

[ ]:

```python

```