OPENCV Bootcamp : NOTES

COURSE INTRODUCTION

- DARPA 2005, Automomous Velucle Stanley → thanks to opencv

most extensive computer visiom library! comtaims >2000 algorithms
FOUNDAMENTAL for COMPUTER VISION and AI

- FIRST step to get started with opemCV!

- course im jupiter Notebook → PROGRAM

1. BASICS : images, video, read, write
2. IMAGE ENHANCEMENT
3. IMAGE FILTERING
4. IMAGE ALIGNMENT (PANORAMAS)
5. COMPUTATIONAL PHOTOGRAPHY
6. Deep Leamimy INference
   ↓
   FACE DETECTION / OBJECT TRACKING
7. DL FOR Object detectiom + Pose Estimatiom

INTERVIEW GUIDANCE (Satya Mallick)

understand TRADITIONAL + DEEP LEARNING approach to COMPUTER VISION

"AI": make machimes tnimk like humans → traim by given data
IS MACHINE LEARNING

while DEEP LEARNING solve ML by usimg
deep neural networks
↑ AI problems usimy NN

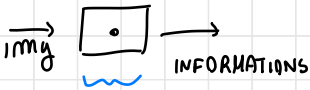
COMPUTER VISION ~ analysis of images and video

IMAGE PROCESSING

- enhance
- encode..

≠

COMPUTER VISION



broader than just AI

handle many things, NOT all related to AI algorithms

3D CV uses AI to enhance it!

AI has lots of other fields such as NATURAL LANGUAGE PROCESSING (deal with text data)
SPEECH RECOGNITION ...

CV has big potential to exploit richness of VISUAL information!

it is easy to have cameras accessible, in many fields as Automomous Driving, manufacturing, etc...

PREREQUISITES

- PROGRAMMING abilities → Python, C++ important for JOB

  - OpenCV, Pytorch (Deep Learning framework open source)
  - Tensor Flow, Keras

OpenCV library is available both in C++ and Python

Embedded Computer Vision: on NOT powerfull system → Algorithms in C++/C
because NOT GPU enabled!

GETTING STARTED WITH IMAGES

IMAGE HANDLING, COLOR CHANNELS, CONVERSIONS

In Jupyter Notebook

» %matplotlib inline — to display images in the notebook

» from IPython.display import Image — render and display image in notebook

↓

function to display image properly rendered with correct pixel size

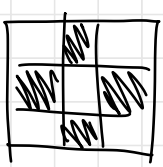
- OpenCV is used to store image as **numpy array** ← working with this to manipulate the image (mathematically!)

Reading Images

with cv2.imread(<filename_path>, <scale>)

↓

this return a numpy array 2d if in grayscale (0)



→

```
[[255   0  255],
 [  0 255   0 ],
 [255   0  255]]
```

each element i,j describe pixel intensity in grayscale

↑ in the range 0 ÷ 255, unsigned int 8 bit

↓

shape and dtype give the attributes

Display Image ⇒ IF using plt.imshow(·) from matplotlib

↓

math representation with x,y axis... looks different because matplotlib uses colormap to display image!

⇓

to display as grayscal, set: cmap = 'gray'

COLOR IMAGES

While function Image from IPython render it in the BROWSER...
with openCV cv2.imread we have a mathematical representation of the image

Now (<file>, 1) color image scale ≡ CV.IMREAD_COLOR

↓ in RGB representation

BUT....

↓

openCV works with BGR format

↓

we need to swap channels order — reverse R and B — img[:,:,::-1]

IMPORTANT! → channel order convention

SPLIT / MERGE COLOR CHANNELS

- b,g,r = cv2.split(image)

split in its components ⇒ each numpy array with each channel

↓

high intensity in the respective color channel if shown in grayscale

- img = cv2.merge((b,g,r))

CONVERT COLOR SCALE

cv2.cvtColor() → convert between color spaces

for example from BGR to RGB

for example also HSV representation ← Hue, Saturation, Value

- Hue = color of image (actual color)
- Saturation = intensity of color
- value = how light / dark

than it is possible to modify a single channel and merge again

for example modify just the Hue

SAVING IMAGES → with cv2.imwrite(<filename>, img)

MATPLOTLIB plt.imshow Vs OpenCV cv2.imshow

```
plt.imshow(img)
plt.show()
```

```
window1 = cv2.namedWindow("w1")        # create named window
cv2.imshow(window1, img)
cv2.waitKey(8000)                      # milliseconds that window is displayed. Otherwise No way to exit the window
cv2.destroyWindow(window1)
```

if waitKey(0) ↑ display until press Keyboard

OR exit only when specific key is pressed with while loop

↑ different ways to manage images in openCV visualization in imshow

BASIC IMAGE MANIPULATION

- changing values of individual pixels
- cropping, resizing, flipping

Access and modify individual pixels

↓

numpy array (0 based) [i, j] in gray scale
standard array indexing to access values!

to modify pixels 1) copy image with .copy() method

we can simply modify pixels by assigning new values
(also with slicing techniques)

Cropping it involves again ARRAY INDEXING

↓

to crop image we can just save in a new array the cropped image
accessed by **slicing**

↳ Just extract region of interest

Resizing resize (src, desiredsize [, fx, fy, interpolation])

↑

method from openCV, different techniques available

fx, fy are scaling factor fx=2, fy=2 correspond to double image

moreover it is possible to specify width, height

and properly maintain the scale

Flipping Images cv.flip(src, flip code)

↳ horiz. 1
↳ vertic. 0
↳ both. -1

allow for new image flipped.

IMAGE ANNOTATION

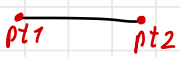
how to annotate images with { lines, circles, rectangls, +text } (also valid for video frame)

↓

even if we have grayscale image → we read it as colored, to use color annotation

DRAW LINE cv2.line(img, first pt, second pt, color, <optional>)

- BGR format is used to specify the color



lineType = cv2.LINE_AA Anti Alias := use semi transparent pixels (smooth line)

DRAW CIRCLE

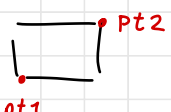
cv2.circle (img, center, radius, color, <optional>)

(same considerations as before)

DRAW RECTANGLE

cv2.rectangle (img, pt1, pt2 , color, <optional>)

(the rest is the same as before)



ADD TEXT

cv2.putText (img, text_string, text_origin, fontFace, fontScale, color...)

bottom left font style

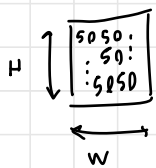
ex. FONT_HERSHEY_PLAIN

IMAGE ENHANCEMENT → Basic image processing techniques for enhancement and pre-process

↓ foundamental in many CV processing pipelines

BRIGHTNESS ADJUSTMENT

we initialize a matrix of 1 (np.ones) * 50



same HxW as original image (matrix of 50) → then cv.add ← brighter, cv.subtract ← darker

guarantee different brightness

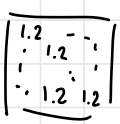
CONTRAST

this is the different of intensity values of pixels: requires multiplication

matrix 1

↓ low contrast

matrix 2



↓ high contrast

↳ we can get values higher than 255 causing issues!

than by matrix multiplication cv.multiply(..)

here img is converted to float64, than convert back to uint8

when convert back to UINT8, the values are corrupted to small numbers ~0

to properly convert in high contrast, avoiding bit corruption,
use np.clip(.)
↑ this clip values to 0÷255 before converting to uint8 for img

used to handle OVERFLOW

THRESHOLDING used often to create binary images to modify only a portion of image, maintaining the rest intact

cv.threshold(img, thresh, maxval, type [,dst])
- thresh: 0÷255
- maxval: for BINARY map
- type: type of thresholding, for ex: BINARY

according to the specified threshold, pixels in original image below threshold are 0 while the one above to 255
↑ BINARY map 0÷255 (maxval)

While
cv.adaptiveThreshold(img, maxValue, adaptiveMethod, thresholdType, blockSize, C)
- adaptiveMethod: method on how to perform adaptive thresholding
- blockSize: used by adaptive method; blockSize = pixel area considered

⇓
the Thresholded img
allow to select
parts of the image,
for example read sheet music
↳ try to
isolate information based on Thresholding

Try different thresholding, maybe with adaptive?

When Geobal threshold doesn't work well for our problem...
rely on adaptive Threshold

↓

can achieve much better performance in isolating parts of image

BITWISE OPERATION

cv2. { bitwise_and(img1, img2, [dst, mask])
bitwise_or()
bitwise_xor()
bitwise_not()

which portion of img2 to apply it
↓
mask = None
consider all

AND bitwise AND comparison..

255 (white) if both pixels in img1 & img2 are white
(only region where both pixels are white)

OR

white when img1 OR img2 is white

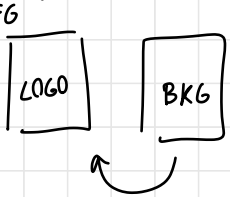
XOR exclusive OR

white (255) when either pixels are white BUT NOT both!

↓

by using BITWISE OPERATIONS + BINARY MAPS
it is possible to obtain good results

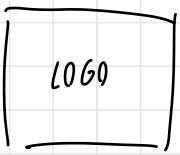
(foreground)
FG



display background (bkg) image behind white pixels on logo?

⇓

(1)



→ read as colored the foreground

→ read background
+ make sure it is same size as FG, using the resize() method

(2) CREATE masks in grayscale converted FG
create binary mask with cv.threshold (0 ÷ 255)

(3) INVERSE MASK → with bitwise_not() you can inverse

(4) you can do bitwise_AND using the mask= img_mask

using bitwise_and (bkg, bkg, mask)
bitwise_and on itself create white

(5) Isolate FG, bitwise_and on image_FG itself using the Inverse mask

(6) Finally merge images with add(·) (black sum to 0)

↓

foundamental in many img processing pipelines!

ACCESSING THE CAMERA

of your PC and stream the video to output window on your display

```
import cv2
import sys

s = 0
if len(sys.argv) > 1:
    s = sys.argv[1]

source = cv2.VideoCapture(s)

win_name = 'Camera Preview'
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)

while cv2.waitKey(1) != 27: # Escape
    has_frame, frame = source.read()
    if not has_frame:
        break
    cv2.imshow(win_name, frame)

source.release()
cv2.destroyWindow(win_name)
```

default camera device index

check if command line to overwrite specification

video capture object with device index=0 (access default one)

while loop to stream video until Escape key is pressed

read() method return a single frame from video stream + has_frame which check for errors

send to output window

↳ it is then possible to process camera video frame and post-process

WRITING A VIDEO USING OPENCV

how to save a video to disk ?

⇒ once specified a source video (for example an mp4)

than read it as VideoCapture object

check if isOpen() successfully

than read() retrieve FIRST FRAME

Finally we can load with HTML function for browser

VideoWriter function allow you to create video file on disk

(filename, fourcc, fps, frameSize)

fourcc: four character code of codec to compress video

frameSize: dimension of frame you wanna save to disk

.get(3), .get(4) retrieve W, H dimensions

AVI based on different fourcc as ('M','J','P','G')
MPG (* 'XVID')

you will need videoWriter object where saving all frames in loop

When finish you release resources

- CORRECT codec !
- CORRECT frame dimension !

IMAGE FILTERING (EDGE DETECTION)

Image processing techniques used in CV pipelines...
↳ using CAMERA sensor frames

↳ We do image processing on video Frame... Then send output to the output Display window

```
import cv2
import sys
import numpy

PREVIEW  = 0  # Preview Mode
BLUR     = 1  # Blurring Filter
FEATURES = 2  # Corner Feature Detector
CANNY    = 3  # Canny Edge Detector

feature_params = dict(maxCorners=500, qualityLevel=0.2, minDistance=15, blockSize=9)
s = 0
if len(sys.argv) > 1:
    s = sys.argv[1]

image_filter = PREVIEW
alive = True

win_name = "Camera Filters"
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)
result = None

source = cv2.VideoCapture(s)

while alive:
    has_frame, frame = source.read()
    if not has_frame:
        break

    frame = cv2.flip(frame, 1)

    if image_filter == PREVIEW:
        result = frame
    elif image_filter == CANNY:
        result = cv2.Canny(frame, 80, 150)
    elif image_filter == BLUR:
        result = cv2.blur(frame, (13, 13))
    elif image_filter == FEATURES:
        result = frame
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners = cv2.goodFeaturesToTrack(frame_gray, **feature_params)
        if corners is not None:
            for x, y in numpy.float32(corners).reshape(-1, 2):
                cv2.circle(result, (x, y), 10, (0, 255, 0), 1)
```

how close 2 corners features can be in the list of features ← min distance between adjacent feature corners (in px space)

different RUN mode for the SCRIPT

dictionary of parameter settings for CORNER feature detector

↑ max amount of CORNERS to return

↑ characterize min quality acceptable for CORNERS ⇒ best feature corners is multiplied, then used as threshold to filter features from algorithm output

size of pixels neighbors used to compute feature corners

read a frame from video stream

flip horizontally frame for convenience of view

just display

canny edge detection function... (lower, upper - thresh)

upper := decide if pixels are edges... sure edge

lower := discarded if < if low < . < upper are declared as edges if nearby edge segments!

```
while alive:
    has_frame, frame = source.read()
    if not has_frame:
        break

    frame = cv2.flip(frame, 1)

    if image_filter == PREVIEW:
        result = frame
    elif image_filter == CANNY:
        result = cv2.Canny(frame, 80, 150)
    elif image_filter == BLUR:
        result = cv2.blur(frame, (13, 13))
    elif image_filter == FEATURES:
        result = frame
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners = cv2.goodFeaturesToTrack(frame_gray, **feature_params)
        if corners is not None:
            for x, y in numpy.float32(corners).reshape(-1, 2):
                cv2.circle(result, (x, y), 10, (0, 255, 0), 1)

    cv2.imshow(win_name, result)

    key = cv2.waitKey(1)
    if key == ord("Q") or key == ord("q") or key == 27:
        alive = False
    elif key == ord("C") or key == ord("c"):
        image_filter = CANNY
    elif key == ord("B") or key == ord("b"):
        image_filter = BLUR
    elif key == ord("F") or key == ord("f"):
        image_filter = FEATURES
    elif key == ord("P") or key == ord("p"):
        image_filter = PREVIEW

source.release()
cv2.destroyWindow(win_name)
```

Blurring the frame with box filter...
(img_frame, (box kernel dimension))
size of kernel define how much of blurry occurs

CORNER FEATURE detection
we use grayscale
this comput the CORNERS FEATURE
⇓
take gray-scale image + dictionary of feature parameters
↓
returns a list of detected corners

↑ location of features

monitor user INPUT Keyboard, so Runmode is interactively changed

- PREVIEW mode: used just to visualize video

↓

('b')

↓

- BLURING mode: usefull for noisy image ... by small blur it can be better OR as pre-process for Feature extraction

because feature extraction rely on Numerical gradient computation. Which can be noisy and not good on RAW PIXELS data

↓

image smoothing is more Robust when feature extraction

↓

('f')

- CORNER feature detector:

↓

generated based on min distance and quality level

you detect corners only if distant more than min distance! it depends on the pixels space distance...

the best corners define a threshold! when exposing better feature corners, the less evident one are filtered out! → higher score define the accuracy level

↓

('c')

- CANNY Edge detector:

allow to properly recognize edges of frame and adjusting upper, lower threshold it is possible to adapt the edge extractor

before I was (145, 150) NOT able to find weak edges

↓

(80, 150): you have better chance to find weaker edges connected with stronger...

Training and experiments allow to optimize this Algorithms

IMAGE ALIGNMENT ~ IMAGE REGISTRATION

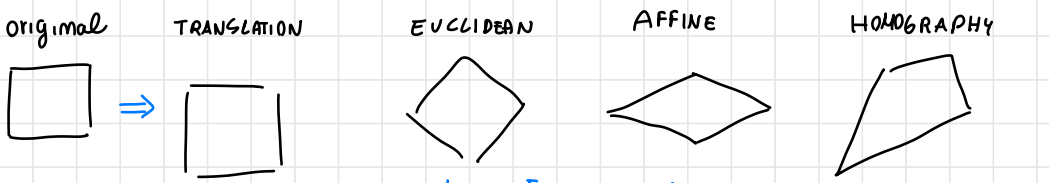
used in many applications such as building PANORAMIC images and constructing HDR images from different exposure images

+ Medical field to compare digital scans and compare small changes

DOCUMENT ALIGNMENT how to allign a document photo allowing then optical character recognition

Theory

TRANSFORMATIONS... Homography



simple shift of pixel coordinates

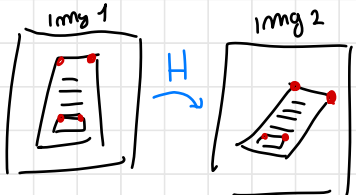
preserve shape and size of original

Rototranslation

include scale changes.. BUT parallel lines remain parallel!

transform into arbitrary... used when changing perspective

homography between two images....



CORRESPONDING POINTS

homography... relate two images on plane

IF we have 4 points in both images (same location)

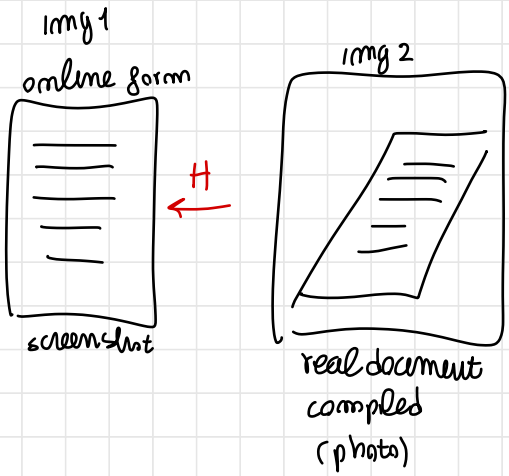
↓

we can compute homography between two images

same physical points with different pixels coordinates

when we have 4 correspondence between 2 images..
openCV function extract homography to then apply it to changes perspective

minimum number required..
openCV functions allow to find N correspondence



how to apply homography to img 2 to allign it as in img 1 (to be readable)

1) FIRST step is finding keypoints in both images
↓
- we need to use GRAYSCALE to perform feature extraction
- orb object needs to be configured cv2.ORB_create

ORB are part of image feature and feature extraction algorithms
↓
to extract features from images → objective of extracting meaningfull informations contextually related to the image itself

We look for edges / corners / texture...
many ways to represent this information compactly
one way is by ORB FEATURES
↳ available in openCV

ORB object is used to detect and compute keypoints / descriptor for each image

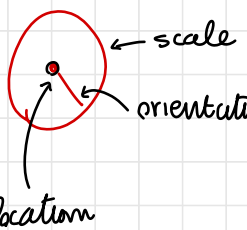
orb.detectAndCompute ⇒ return keypoints and associated descriptors

keypoints: interesting features associated with sharp edges / corners ↓ described by pixel coordinates (location) + size of keypoints + orientation

descriptors: list of descriptors, each descriptor is a vector of information that describe region around keypoint effective signature of keypoint (pixel info)

If we look for same keypoint in both imgs we can try to use descriptors to match them

it is possible to draw keypoints by draw keypoints ↓



⇒ associated with each keypoints, there is a VECTOR representation of that image patch at that keypoint

descriptor is what is used to match keypoints

img1, img2 have different and some overlapping keypoints that we want to find for deriving homography

2) Matching key points... done through a cv.DescriptorMatcher.create

↓

matcher object base on some configuration

↓

matching algorithm + metric

cv.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING

BRUTEFORCE: algorithm

HAMMING: metric to compute distance between descriptors

hamming metrics.. required because ORB descriptor are binary string

then match function can be called to get best matches..

⇓

list of matches than are listed and sorted based on descriptor distance

↓

then only take best 10% only

finally with drawMatches(.) it is possible to draw identified matches

several key points are correctly matched!

BUT some outliers can occurs due to descriptor coincidence...

FALSE positive can occur!

3) Compute Homography simply with cv.findHomography(H) using the two points (filtered) + cv.RANSAC

two points: pre-process and type casting of points

cv.RANSAC: algorithm used to compute H

4) once H 3×3 matrix is extracted, we can apply it to the img2 by warpPerspective method to get register (alligned) image

↓

alligning images... can be very useful for different applications...

PANORAMAS in OpenCV

↳ panoramas from multiple images using OpenCV

very similar pipeline of previous lesson on Image Alignment...

⇓

again is required ⇒ Key points/descriptor + correspondence through feature matching

⇓

+ estimate homography for Image ORBING ⇒ finally additional step to stitch and blend images together

Am high-level convenient function in OpenCV allow to create panoramas from set of images...

↓

created with single function call ⇒ using photos at same daytime to avoid different lighting

↑ with stitcher class

collect images in a set of images (append)

- using cv.stitcher.create() → stitcher object... then use it to call stitch with list of images... which return the final panorama

BUT it create black regions because of required orbing!

↓

(you can use thresholding + bitmaps + contour for this task)

(HDR) HIGH DYNAMIC RANGE IMAGING

exposure set up occurs when taking a photo..
because actual intensity in the scene exceed camera capability
because cameras uses 8 bit per channel representation...
⇒ there is NOT enough bit to capture full dynamic range

↓

While in HDR mode both the foreground/background are properly exposed!

↙

it is done taking three photos at different exposure (quickly! so there is no movement between shots)
takes low dynamic range photo and merge them to obtain HDR photo

↳ From multiple photos at different exposure
(each with different details taken properly)

↑

collectively a sequence of images contains information to merge and obtain a satisfactory photo

Images + exposure time
(it can be extracted from image metadata)

↓

1) collect images, times in a list

2) Allign images at pixel level! to avoid possible ghost artifact that ruin the photo

↓

important to avoid artifacts

↳ because different exposure... look different... standard allignment techniques doesn't work

⇓

special class uses BITMAPS to allign!

createAlignMTB().process(Images, Images)

↑ alignment

3) compute Camera Response function..

Because lots of cameras are NOT LINEAR → problem to merge different exposure

IF it was linear intensity could be scaled by exposure time...
then on same radiants to compute average intensity then

↙

we need to estimate Camera Response Function and linearize it before combining

↓

it can be estimated by OpenCV functions (in classes) using algorithms that optimizes the identification step

createCalibrateDebevec ⇒ we get the response function

↑ people who invented it

LOT NON Linear @ high INTENSITY + different channels function due to different sensitivity

↓

this function $f(x)$ can be used to linearize the INPUT images mapping measured pixel intensity to the calibrated intensity and merge appropriately

4) Merge exposure into HDR

↳ merging process ignore values close to 0, 255 because NOT very informative

→ multiple image → for every pixel we expect good values...

BUT intensity now is no more in 0÷255

HDR can require 0÷inf in 32 bit brightness ⇐

but we need 8 bit to display

↓

bring back to 0÷255 TONEMAPPING

process of mapping HDR to 8 bit using opencv functionality

- used to achieve pleasent images... different function and configurations

ex: Drago's method obtain tone map which is processed → even background etc.

ex. OR Reinhard's method very realistic
OR Mantiuk's method which is a sort of combination of the previous

OBJECT TRACKING

estimating the location of an object and predicting its location in future times

↓

Im CV context: detect interested object in video frame and predicting its location in following frames...

Accomplished by: MOTION MODEL + Appearance model

MOTION MODEL → to estimate pos/vel of obj in pixel coordinates and use it to predict location in the next frame ⇓ approx. where object may be located in the future

Appearance model → encode how to object looks like and box the region around it as predicted location to fine tune location ⇓ fine tune motion estimate

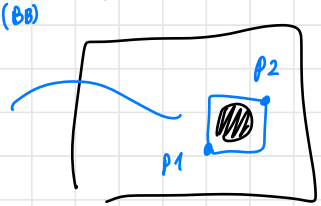
→ we do it by opencv tracker class API

↓

first, the tracking Algorithm is initialized by specifying INITIAL LOCATION of object by a bounding box

then we want to track object in subsequent video frames by producing BB in each new frame

← bounding box (BB) specified by p1, p2



↓

there are 8 different tracking Algorithms each one more suited to specific application

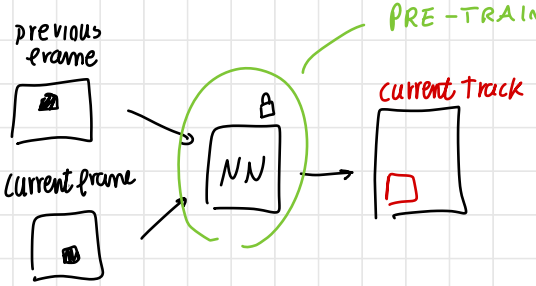
- Robustness
- speed
- Accuracy

with GOTURN which is Deep Learning Based

the appearance / lighting change during motion → challenges for tracking Algorithms

+ useful support functions are defined to draw bounding boxes

GOTURN is Deep learning (DL) based, which uses NN to predict where BB will be in next video frame



PRE-TRAINED NN model (INFERENCE model)

which take as input two images { previous current

↓

and use previous BB to crop both images and locate current

inference model properly locate new BB

1) create tracker instance of a specific type

↓

different available in tracker API from OpenCV

2) set up I/O video streams
creating a video writer to annotate output video

3) Initialize the TRACKER defining initial bbox (manually) by P1, P2 (bottom left, upper right)
usually can be done
by DETECTION ALGORITHM OR by USER INTERFACE

↓

initialize with init(frame, bbox)

initial frame and BB

4) process all remaining video frames
with the update (frame)
+ Annotate video frames

↓

Different tracker behaves differently

KCF has some issues when appearance change a lot

CSRT also when big changes occurs drift error with bad location

GOTURN (based on NN) maintain a "satisfactory" tracky with
fairly accurate estimate

↑

in more challenging situation GOTURN outperforms the others!

FACE DETECTION

Using a PRE-TRAINED NN with opencv Framework to read a pretrain model and perform inference with that model

```
import os
import cv2
import sys
from zipfile import ZipFile
from urllib.request import urlretrieve


# =======================-Downloading Assets-=======================
def download_and_unzip(url, save_path):
    print(f"Downloading and extracting assests....", end="")

    # Downloading zip file using urllib package.
    urlretrieve(url, save_path)

    try:
        # Extracting zip file using the zipfile package.
        with ZipFile(save_path) as z:
            # Extract ZIP file contents in the same directory.
            z.extractall(os.path.split(save_path)[0])

        print("Done")

    except Exception as e:
        print("\nInvalid file.", e)


URL = r"https://www.dropbox.com/s/efitgt363ada95a/opencv_bootcamp_assets_12.zip?dl=1"

asset_zip_path = os.path.join(os.getcwd(), f"opencv_bootcamp_assets_12.zip")

# Download if assest ZIP does not exists.
if not os.path.exists(asset_zip_path):
    download_and_unzip(URL, asset_zip_path)
# ====================================================================


s = 0
if len(sys.argv) > 1:
    s = sys.argv[1]

source = cv2.VideoCapture(s)

win_name = "Camera Preview"
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)

net = cv2.dnn.readNetFromCaffe("deploy.prototxt", "res10_300x300_ssd_iter_140000_fp16.caffemodel")
# Model parameters
in_width = 300
in_height = 300
mean = [104, 117, 123]
conf_threshold = 0.7
```

set up camera

PROTO text file → has Network architecture information

↑ pre-trained NN model red by Caffe (or pytorch, tensor-flow...)

↓

you CANNOT use opencv to train a NN but you can use it to perform INFERENCE based on a pre-trained NN → usefull in Video

function defined to read in a Caffe model

Caffe model file → containing the weights of the DNN model we are using

```python
# =======================-Downloading Assets-=======================
def download_and_unzip(url, save_path):
    print(f"Downloading and extracting assests....", end="")

    # Downloading zip file using urllib package.
    urlretrieve(url, save_path)

    try:
        # Extracting zip file using the zipfile package.
        with ZipFile(save_path) as z:
            # Extract ZIP file contents in the same directory.
            z.extractall(os.path.split(save_path)[0])

        print("Done")

    except Exception as e:
        print("\nInvalid file.", e)


URL = r"https://www.dropbox.com/s/efitgt363ada95a/opencv_bootcamp_assets_12.zip?dl=1"

asset_zip_path = os.path.join(os.getcwd(), f"opencv_bootcamp_assets_12.zip")

# Download if assest ZIP does not exists.
if not os.path.exists(asset_zip_path):
    download_and_unzip(URL, asset_zip_path)
# ============================================================


s = 0
if len(sys.argv) > 1:
    s = sys.argv[1]

source = cv2.VideoCapture(s)

win_name = "Camera Preview"
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)

net = cv2.dnn.readNetFromCaffe("deploy.prototxt", "res10_300x300_ssd_iter_140000_fp16.caffemodel")
# Model parameters
in_width = 300
in_height = 300
mean = [104, 117, 123]
conf_threshold = 0.7
```

It is taken from /opencv/opencv GIT repo at the download_models.py used to download model referenced in a yaml file → with proper parameters related on How model has been trained

how model was trained

Instance of the NETWORK used for inference on video stream

all images MUST be processed as the trained image has been

size of input img 300×300

mean values for RGB across used imgs

conf_threshold define the sensitivity of detection

blob representation of INPUT imag frame

input preprocessing so that it is in the proper format to perform inference

```
while cv2.waitKey(1) != 27:
    has_frame, frame = source.read()
    if not has_frame:
        break
    frame = cv2.flip(frame, 1)
    frame_height = frame.shape[0]
    frame_width = frame.shape[1]

    # Create a 4D blob from a frame.
    blob = cv2.dnn.blobFromImage(frame, 1.0, (in_width, in_height), mean, swapRB=False, crop=False)
    # Run a model
    net.setInput(blob)
    detections = net.forward()
```

← flip horizzontaly just for convenience

size of video frame

imag size

subtracted mean

img

scale factor as im game... depends on DNN train

swap Red Blue, NO because OpenCV and Coffe uses same convention for RGB

← pre-process on image frame

NO cropping image im input... here RESIZE

prepare for inference

inference on the input given...

loop over all detections im the inference... determining if enough confidence

```
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > conf_threshold:
            x_top_left = int(detections[0, 0, i, 3] * frame_width)
            y_top_left = int(detections[0, 0, i, 4] * frame_height)
            x_bottom_right  = int(detections[0, 0, i, 5] * frame_width)
            y_bottom_right  = int(detections[0, 0, i, 6] * frame_height)

            cv2.rectangle(frame, (x_top_left, y_top_left), (x_bottom_right, y_bottom_right), (0, 255, 0))
            label = "Confidence: %.4f" % confidence
            label_size, base_line = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)

            cv2.rectangle(
                frame,
                (x_top_left, y_top_left - label_size[1]),
                (x_top_left + label_size[0], y_top_left + base_line),
                (255, 255, 255),
                cv2.FILLED,
            )
            cv2.putText(frame, label, (x_top_left, y_top_left), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))

    t, _ = net.getPerfProfile()
    label = "Inference time: %.2f ms" % (t * 1000.0 / cv2.getTickFrequency())
    cv2.putText(frame, label, (0, 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0))
    cv2.imshow(win_name, frame)

source.release()
cv2.destroyWindow(win_name)
```

bounding box creation

return the time required to perform inference ← converted in millisecond

↑ display annotated frame!

TENSOR FLOW (TF) OBJECT DETECTION

↓

DL based objed detection (OD), using NN called single shot Multi-Box (SSD) Detection

↓

trained using tensor Flow

OpenCV is used to read the model and perform inference on sample test images ←

"SSD"

└ SINGLE SHOT := we make single forward path through net for inference

(Multi-box) DETECTOR := Detecting Multiple object with in an image

↑

It can be trained with different architectural backbones...

Here a MOBILENET architecture is used (small model)

Many object detection models are available (open source)

↓

From the archive you need only the "frozen_inference_graph.pb"

↑ this contains the weights for model

+ Additional file to run the NET

- configuration file: .pbtxt
- class labels: .txt (for Dataset)

↓ here "coco" dataset

a single SCRIPT can be used to generate the files...

as tf_text_graph_ssd.py

↳ In the class labels: here Detector with all classes

≠

from Detector in Computer Vision ≠ Deep learning Object detector

↓ one for each class, separate model

↓ Simple model detects MANY objects

3 steps: 1) Load model / input image
2) Detect object by forward pass through Net
3) Display objects with BBox and Labels

1) read Net From Tensorflow → return a Net instance

2) supplementary functions...
first BLOB the image to pre-process as desired by TRAINING set
we used different training images convention by OpenCV... so swapRB ... while crop NO, then Resized
↓
pre-process step... finally you can forward() the Net

+ supplementary functions to display text and boxes

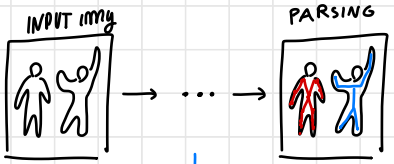
3) We can use the net and image to display the image with objects detected..

↑
this model NN is ROBUST with ~ 80 classes available!

FALSE POSITIVES can occurs ...
↳ you can use it to perform hard negative mining to reduce # False positives...

POSE ESTIMATION USING Open Pose

↳ Perform 2D Human-Pose estimation using pre-trained model Open Pose



- identify human body keypoints (JOINTS)
- connect logically keypoints

↓

it outputs { Part confidence Maps ↙ Probability Maps...
Part Affinity Maps

Very HARD Problem... • Joints NOT always visible (occlusion is common)
+ Associate with correct people when multiple

↓

Improvements using Deep Learning to this models...

here we use Open Pose Caffe models trained on a multi-purpose image dataset

↑

it can be used on
Video Streams OR single Images...

1) Load Caffe model for the DNN
+ specify parameters for inference → readNetFromCaffe to load net

nPoints := # model points
POSE_PAIRS := set of indices in the human linkages...
0 head, 1 neck... mapping used to process output

2) Read Image (OR Video Frame)

3) Pre-process the image by the blobFromImage function
to convert as TRAINING data
↓
them passed as input to the net.setInput

4) forward the net!
↓
use the model to inference pose ⇒ return { confidence maps / affinity field

for each point we have a PROBability maps (heatmaps)
↓
Likelihood for each keypoints of human joints

then rescale it to overline the keypoints to the final image

cv.minMaxLoc → return location with MAX PROBABILITY
then scaled up on original SIZE ONLY IF PROPER SIZE!

5) finally we can show the image
using supplementary functions to plot the data points
↑
properly parsing the output and render information on original image

↑
leverage pre-trained model to inference