MACHINE INTELLIGENCE

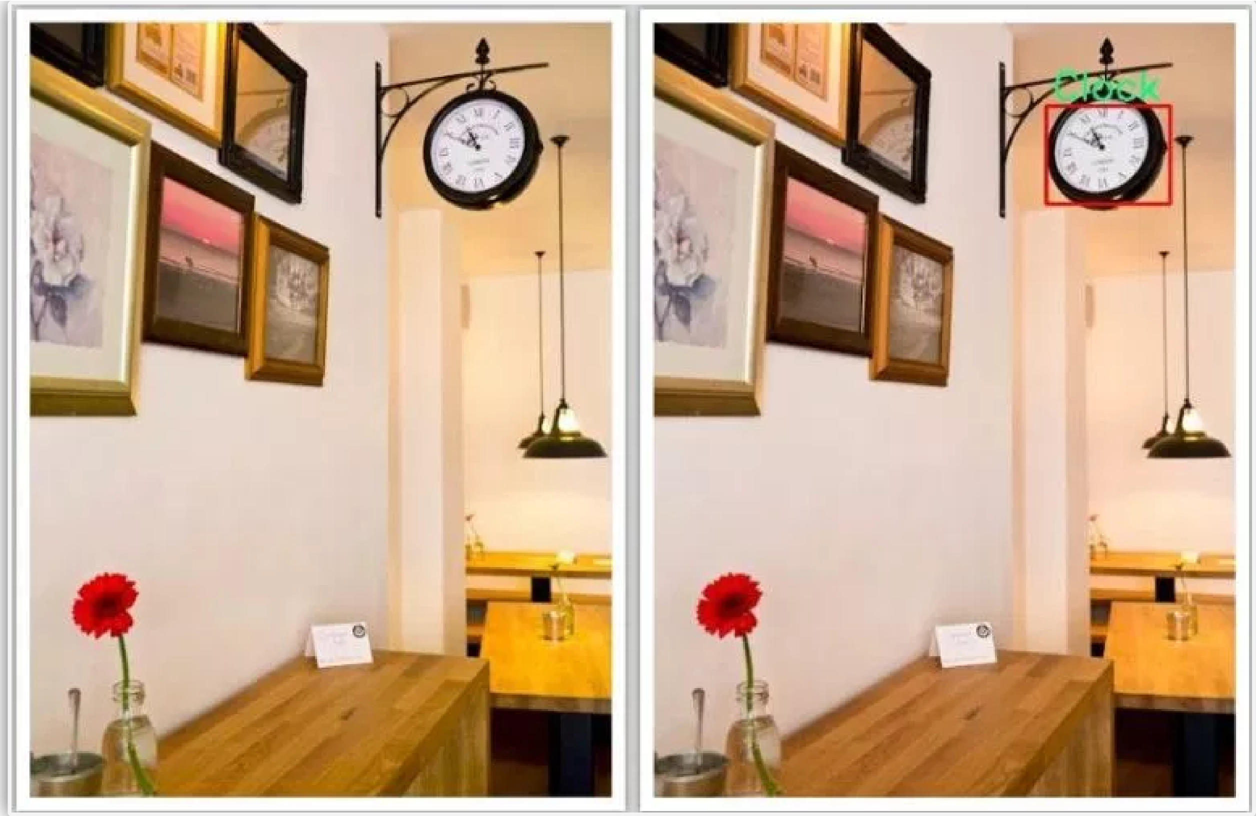
FOR HUMANS

MENU

CREATE YOUR OWN OBJECT DETECTOR

February 5, 2017

Creating a **custom object detector** was a challenge, but not now. There are many approaches for handling object detection. Of all, **Haarcascades** and **HOG+SVM** are very popular and best known for their performance. Though Haarcascades which were introduced by Viola and Jones are good in achieving decent accuracy, **HOG+SVM** proved to outperform the Haarcascades implementation. Here in this post we are going to build a object detector using **HOG+SVM** model. The output from our detector is something similar as shown below.



Here we build a Object detector that works for detecting any trained object, but for the explanation of the post let's stick to the example of detecting ***clocks*** in images. However it's just a matter of **annotating** the object in the images we want to detect, which we will see in a moment.

TASK

To build an custom end-to-end object detector. Since we need to detect objects of particular types (here clock), we will train our detector with the objects we want to detect. And for that we need to annotate the objects in images. So breaking down the steps to build an object detector at very high level,

- Collect training images.

- Annotate object locations in the training images.
- Train the Object Detector with the object regions.
- Save and test the trained detector.

PROJECT STRUCTURE

```
Object Detector
├── detector.py
├── gather_annotations.py
├── selectors/
├── train.py
└── test.py
```

- `selectors/` – It contains `BoxSelector` class which helps us to annotate (select) the object regions.
- `gather_annotations.py` – A script that allows to annotate each image using a selector.
- `detector.py` – It contains `ObjectDetector` class that is used for training and detecting objects.
- `train.py` – Used for training an object detector.
- `test.py` – The actual driver script to detect regions in an image.

COLLECT TRAINING IMAGES

Since we want to create a Object detector to detect any object we train it, It's just a matter of changing images and annotations to create any other object detector, as here we will be using clock images to train the detector as an example. I've collected some images containing clocks from internet. I would like to add that the **copyright** of the images belong to their **owners**. The training images are shown below.

ANNOTATE OBJECT LOCATIONS

Now that we have our training images ready. We need to ***annotate*** the coordinates of the clocks in those images. We will adopt `BoxSelector` class from the previous post. Let's build a script (`gather_annotations.py`) that helps us annotate the object regions using the `BoxSelector` class from `selectors` package and save the annotations to disk.

```
import numpy as np
import cv2
import argparse
from imutils.paths import list_images
from selectors import BoxSelector

#parse arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d","--dataset",required=True,help="path to images dataset...")
ap.add_argument("-a","--annotations",required=True,help="path to save annotations...")
ap.add_argument("-i","--images",required=True,help="path to save images")
args = vars(ap.parse_args())
```

We start off by importing necessary packages and parse the necessary arguments.

- `--dataset` – Path to training images dataset.
- `--annotations` – Path to save the annotations to disk.
- `--images` – Path to save the image paths to disk (to make consistent annotations).

```
#annotations and image paths
annotations = []
imPaths = []

#loop through each image and collect annotations
for imagePath in list_images(args["dataset"]):

    #load image and create a BoxSelector instance
    image = cv2.imread(imagePath)
    bs = BoxSelector(image,"Image")
    cv2.imshow("Image",image)
    cv2.waitKey(0)

    #order the points suitable for the Object detector
    pt1,pt2 = bs.roiPts
    (x,y,xb,yb) = [pt1[0],pt1[1],pt2[0],pt2[1]]
    annotations.append([int(x),int(y),int(xb),int(yb)])
    imPaths.append(imagePath)
```

We create two empty lists to hold the annotations and image paths. We need to save the image paths as the annotations for an image can be retrieved by ***index***. So there won't be any mistake in retrieving annotations i.e, retrieving incorrect annotations for an image. And then we loop over each image and create a `BoxSelector` instance to help us select the regions using mouse. We then collect the object location using the selection and append the annotation and image path to `annotations` and `imPaths` respectively.

```
#save annotations and image paths to disk
annotations = np.array(annotations)
imPaths = np.array(imPaths,dtype="unicode")
np.save(args["annotations"],annotations)
```

```
np.save(args["images"],imPaths)
```

Finally we convert the `annotations` and `imPaths` to `numpy` arrays and save them to disk.

CREATE AN OBJECT DETECTOR

If you do not know what exactly **HOG** (Histogram of Oriented Gradients), then I recommend you to go through this link and for **SVM** (Support Vector Machines) go through this link come back. Creating an **HOG+SVM** object detector from scratch is a bit difficult and a tedious process. Fortunately, we have `dlib` package which has an ***api*** for creating such object detectors. So here we create an abstraction to use the object detector from `dlib` with ease. The actual functioning of **HOG+SVM** can be broken down into the following steps.

TRAINING

- Create a ***HOG*** descriptor with certain `pixels_per_cell`, `cells_per_block` and `orientations`.
- Extract ***HOG*** features using the descriptor from each object region (annotated)
- Create and train a ***Linear SVM*** model on the extracted ***HOG*** features.

TESTING

- Estimate the average ***window size***.
- Scale down or up the images for several levels upto a certain termination and build an ***image pyramid***.
- ***Slide the window*** through each image in an image pyramid.
- ***Extract*** HOG features from each location.
- Estimate the ***probability*** of trained **SVM** model with the current **HOG** features. If it is more than certain threshold then it contains object otherwise not.

We won't implement the HOG+SVM model from scratch, instead we will use the `dlib` package as stated before. Let's open `detector.py` and start coding.

```
import dlib
import cv2

class ObjectDetector(object):
    def __init__(self,options=None,loadPath=None):
        #create detector options
        self.options = options
        if self.options is None:
            self.options = dlib.simple_object_detector_training_options()

        #load the trained detector (for testing)
        if loadPath is not None:
            self._detector = dlib.simple_object_detector(loadPath)
```

We import necessary packages and create an `ObjectDetector` class whose constructor takes two keyword arguments,

- `options` – object detector options for controlling ***HOG*** and ***SVM*** hyperparameters.
- `loadPath` – to load the trained detector from disk.

We create default options for training a simple object detector using `dlib.simple_object_detector_training_options()` if no options are provided explicitly. These options consists of several hyper parameters like *window_size*,*num_threads*,etc., which helps us create and tune the object detector. And we load the trained detector from disk in case of testing phase.

```
    def _prepare_annotations(self,annotations):
        annots = []
        for (x,y,xb,yb) in annotations:
            annots.append([dlib.rectangle(left=long(x),top=long(y),right=long(xb),bottom=long(yb))])
```

```
        return annots

    def _prepare_images(self,imagePaths):
        images = []
        for imPath in imagePaths:
            image = cv2.imread(imPath)
            image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
            images.append(image)
        return images
```

And then we define two methods namely `_prepare_annotations` and `_prepare_images` which helps preprocessing the given annotations to the form that are acceptable by the `dlib` detector. And also helps us loading the images from the `imagePaths` and converting them to RGB since `cv2` reads images as ***BGR*** and `dlib` expects the images of ***RGB*** format.

```
    def fit(self, imagePaths, annotations, visualize=False, savePath=None):
        annotations = self._prepare_annotations(annotations)
        images = self._prepare_images(imagePaths)
        self._detector = dlib.train_simple_object_detector(images, annotations, self.options)

        #visualize HOG
        if visualize:
            win = dlib.image_window()
            win.set_image(self._detector)
            dlib.hit_enter_to_continue()

        #save detector to disk
        if savePath is not None:
            self._detector.save(savePath)

        return self
```

We then create our `fit` method which takes in arguments as follows,

- `imagePaths` – a `numpy` array of type `unicode` containing paths to images.

- `annotations` – a `numpy` array consisting of annotations for corresponding images in the `imagePaths`.
- `visualize` – (default=`False`) a flag indicating whether or not to visualize the trained HOG features.
- `savePath` – (default=`None`) path to save the trained detector. If `None`, no detector will be saved.

We first prepare annotations and images using the above defined methods `_prepare_annotations` and `_prepare_images`. Then we create an instance of `dlib.train_simple_object_detector` using the images, annotations and options obtained above. We then handle the visualization of HOG features and saving the trained detector to disk.

```
    def predict(self,image):
        boxes = self._detector(image)
        preds = []
        for box in boxes:
            (x,y,xb,yb) = [box.left(),box.top(),box.right(),box.bottom()]
            preds.append((x,y,xb,yb))
        return preds

    def detect(self,image,annotate=None):
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        preds = self.predict(image)
        for (x,y,xb,yb) in preds:
            image = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

            #draw and annotate on image
            cv2.rectangle(image,(x,y),(xb,yb),(0,0,255),2)
            if annotate is not None and type(annotate)==str:
                cv2.putText(image,annotate,(x+5,y-5),cv2.FONT_HERSHEY_SIMPLEX,1.0,(128,255,0),2)
        cv2.imshow("Detected",image)
        cv2.waitKey(0)
```

Now that we have our `fit` method defined and we proceed to defined `predict` method which takes in an image and outputs the list of bounding boxes for the detected objects in the image. And finally we define `detect` method which takes in an image, converts to RGB, predicts the bounding boxes and draw the rectangle and annotate the text above the detected location using the keyword argument `annotate`.

We are all ready to train our detector. We create a file named `train.py` and fill the following code in it. The code itself is self explainatory.

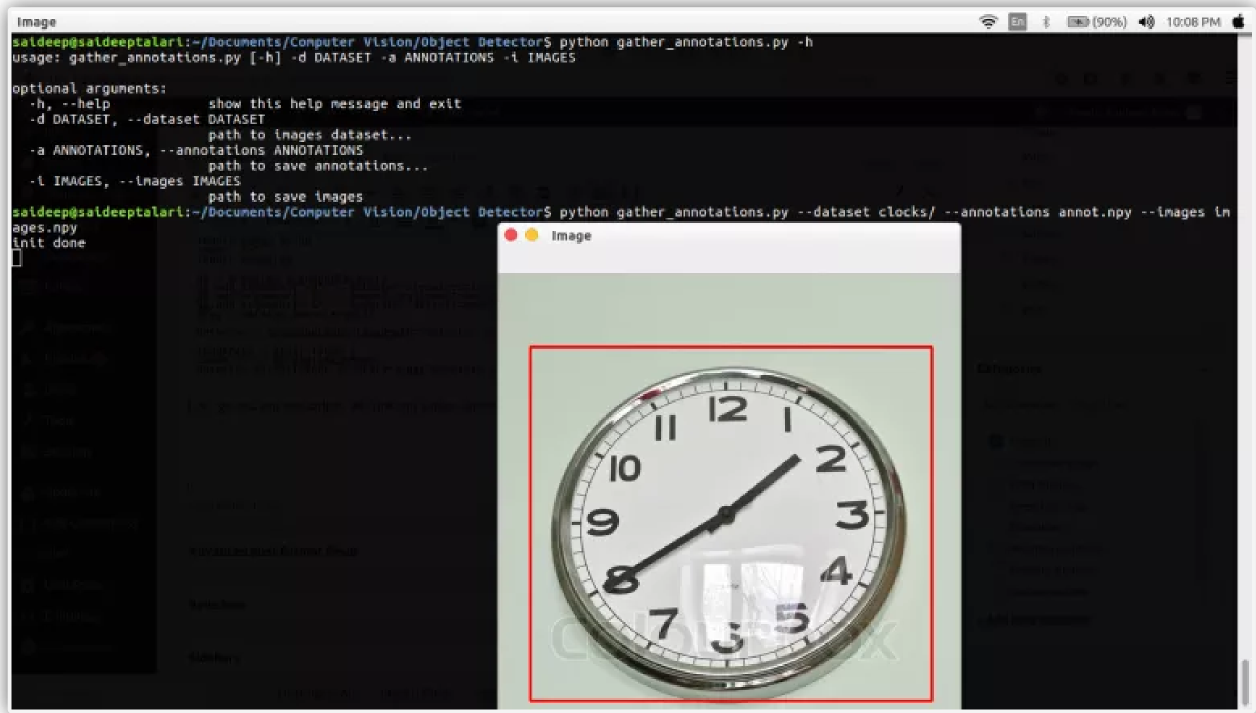
```
from detector import ObjectDetector
import numpy as np
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("-a","--annotations",required=True,help="path to saved annotations...")
ap.add_argument("-i","--images",required=True,help="path to saved image paths...")
ap.add_argument("-d","--detector",default=None,help="path to save the trained detector...")
args = vars(ap.parse_args())

print "[INFO] loading annotations and images"
annots = np.load(args["annotations"])
imagePaths = np.load(args["images"])

detector = ObjectDetector()
print "[INFO] creating & saving object detector"

detector.fit(imagePaths,annots,visualize=True,savePath=args["detector"])
```

We finally create another script named `test.py` used for testing our trained object detector over an image.

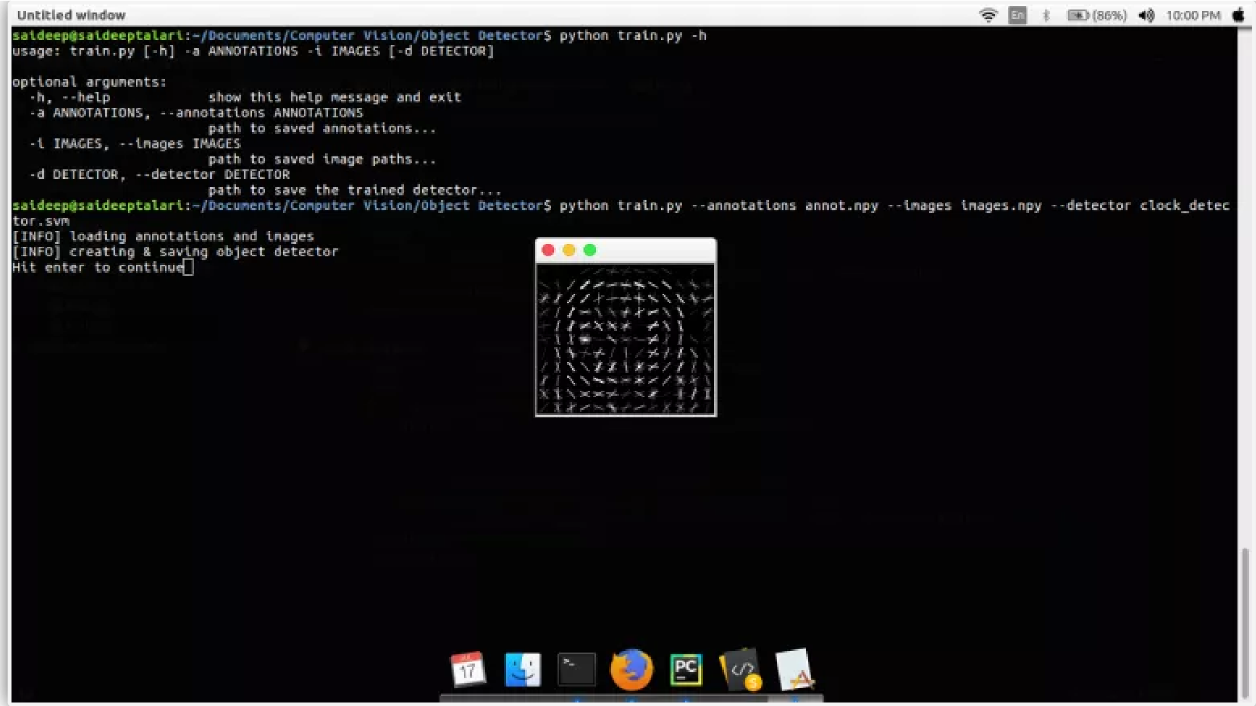
```
from detector import ObjectDetector
import numpy as np
import cv2
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("-d","--detector",required=True,help="path to trained detector to load...")
```

```
ap.add_argument("-i","--image",required=True,help="path to an image for object detection...")
ap.add_argument("-a","--annotate",default=None,help="text to annotate...")
args = vars(ap.parse_args())

detector = ObjectDetector(loadPath=args["detector"])

imagePath = args["image"]
image = cv2.imread(imagePath)
detector.detect(image,annotate=args["annotate"])
```

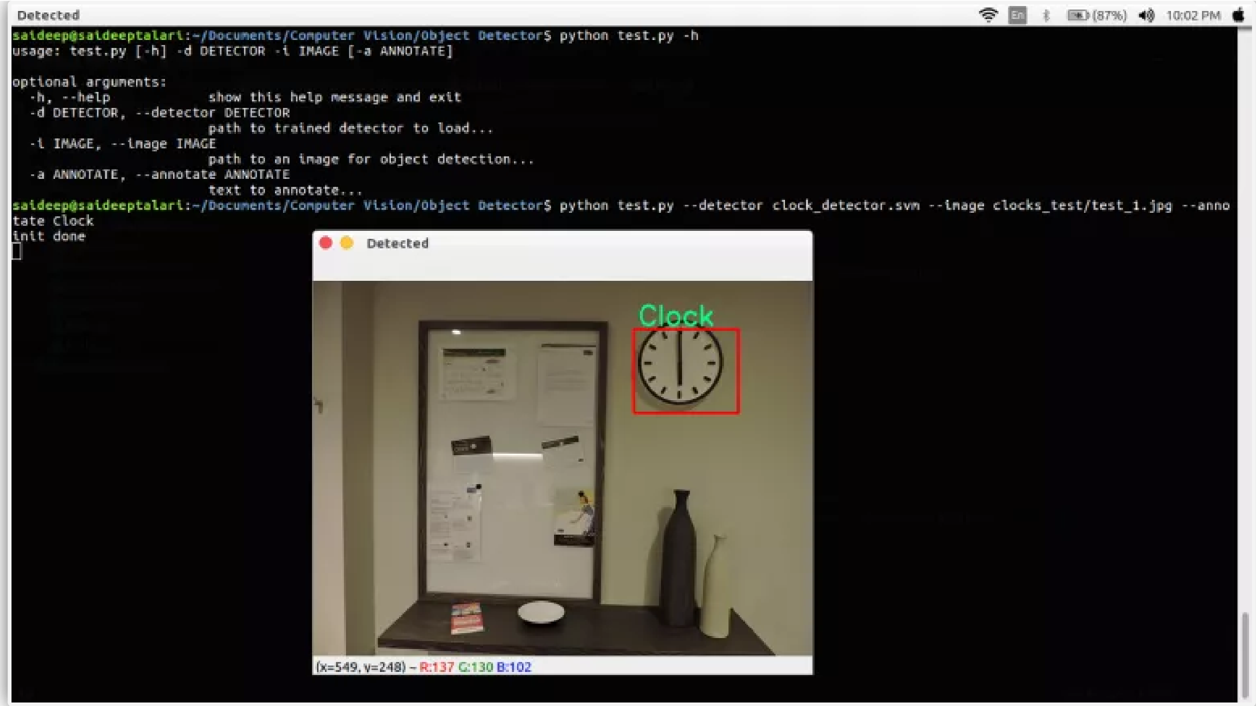
Let's go and run our scripts. We first run `gather_annotations.py` and select the regions of object for each image.



Now that we have annotations and image arrays. We are good to go for training our object detector using the script `train.py`.



We have trained our detector and we can see the trained **HOG** features visualized. This **HOG** is pretty enough to carry out our detection phase. We then run our `test.py` script giving it an input image and let it to detect objects in the image.



Finally we have created our own object detector which is capable of detecting any trained object. The code for this post can be downloaded from my github.

Thank you, Have a nice day…

SHARE THIS:

RELATED

Face Recognition with Deep Learning
September 25, 2017

Object Tracking
January 28, 2017

Optimize Neural nets
December 27, 2016

In "Computer Vision" In "Computer Vision" In "Deep Learning"

Posted in: Computer Vision, General, Machine Learning | *Tagged: dlib, OpenCV, Python*

← *Object Tracking* *Implementing a simple RNN* →

18 Comments hackevolve Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Anirudh Katti • a month ago

where is the clock_detector.svm?

4 • Reply • Share ›

clawdis • a month ago

is there a way to train for 2 or 3 objects such as: clock , bottle or glass ?

• Reply • Share ›

Padma Kannan • 4 months ago

Hi...for training images, can't we have an image with two clocks? Is it like only one clock per image and drawing a bounding box around it?
Is there any way to draw more than one box on an image while training?

• Reply • Share ›

Ahmed Sadman • 8 months ago

How do I use this to annotate multiple objects in a Single image? Suppose that I need to detect some other object in the image alongside Clock

• Reply • Share ›

Fish Bear • a year ago

Thank for you post, can you share you datasets for learning?

• Reply • Share ›

Saideep Mod ➦ Fish Bear • a year ago

You can build your own dataset by collecting images from Google, Bing, etc. Below links will help you with creating datasets.

https://www.pyimagesearch.c...

https://www.pyimagesearch.c...

1 ^ | ˅ • Reply • Share ›

Laura • a year ago

Is there a way to detect more than one object in an image? such as two clocks.

^ | ˅ • Reply • Share ›

Saideep Mod ➦ Laura • a year ago

It detects the trained objects in an image. There can be more than one target object in the image. All you need to make sure is your training set is good enough to make the model generalised. Also, I would recommend you to see a post at this link: https://www.pyimagesearch.c... for more accurate and reliable way of doing object detection using deep learning.

^ | ˅ • Reply • Share ›

Haaris • a year ago

Im getting this error while implementing this

annotations.append([int(x),int(y),int(xb),int(yb)])
AttributeError: 'numpy.ndarray' object has no attribute 'append'

it terminates after the 2nd image

^ | ˅ • Reply • Share ›

Saideep Mod ➦ Haaris • a year ago

Actually, it happens when you convert the annotations <list> to <numpy.ndarray>. I've converted after iterating through all the images. Please note that line 34 is not in the for loop.

^ | ˅ • Reply • Share ›

Norman Siboro • 2 years ago

good tutorial.

I have error in gather_annotations
from box_selector import BoxSelector
ImportError: No module named 'box_selector'
did I miss something?
I already copied the file from your github repository

^ | v • Reply • Share ›

Saideep Mod ↪ Norman Siboro • 2 years ago

Make sure you follow the same package structures.
|--selectors/
|----box_selector.py

If you are using python3 make sure of relative imports! See the link below for more details.
https://stackoverflow.com/q...

^ | v • Reply • Share ›

Norman Siboro ↪ Saideep • 2 years ago

thanks for the help. I have solved that issue
but I have another error

self.orig = image.copy()
AttributeError: 'NoneType' object has no attribute 'copy'

^ | v • Reply • Share ›

Show more replies

Rahul Vijay Soans • 2 years ago

Great work. I have error in gather_annotations
from selectors import BoxSelector
ImportError: No module named selectors

Which module i am missing

^ | v • Reply • Share ›

Saideep Mod ↪ Rahul Vijay Soans • 2 years ago

Hello Rahul,

Have you downloaded the github repository I mentioned at the end of the post. Otherwise please download it from https://github.com/saideept...

Implementing a Box selector in opencv is not much related to this post. So I did not discuss about it much here. If you are interested in how I developed that kindly look into this blog post http://hackevolve.com/objec...

Reply • Share ›

Rahul Vijay Soans → Saideep • 2 years ago

Thank you Saideep. The problem is now fixed. But how to install dlib in python 2.7? It requires cmake also. i tried installing but there is a problem

Reply • Share ›

Show more replies

Cristhian Malakan • 2 years ago

Great tutorial, do you have the images? Did you only use the ones depicted in the post? (10 clocks)

Reply • Share ›

Saideep Mod → Cristhian Malakan • 2 years ago

Yeah, I have! By the way I just downloaded them from the google.

Reply • Share ›

question regarding that. What is a difference … Saideep.