How to train YOLOv3 on the custom dataset

April 28, 2020

How to train YOLOv3 on the custom dataset

This article is the step by step guide to train YOLOv3 on the custom dataset. I assume that you are already familiar with the YOLO architecture and its working, if not then check out my previous article YOLO: Real-Time Object Detection.

YOLO (You only look once) is the state of the art object detection system for the real-time scenario, it is amazingly fast and accurate. YOLOv3 is one of the widely used version of YOLO.

Search this site... Search

Recent Posts

How to train Mask R-CNN on the custom dataset.

Instance segmentation using Mask R-CNN

Region Proposal Network and Faster R-CNN

R-CNN and Fast R-CNN Object Detection.

Web Scraping in Python using BeautifulSoup

Categories

Artificial intelligence

Computer Vision

Deep Learning

Python

Dataset Preparation.

Every object detection system requires annotation data for training, this annotation data consists of the information about the boundary box (ground truth) coordinates, height, width, and the class of object.YOLO requires annotation data in a specific format.

Annotation format.

YOLOv3 requires the annotation information in the form of text file.

For training, we need to create a text file corresponding to each image. The name of the text file should be the same as the image file with a **.txt** extension. Suppose the image name is **sample.png** then it's text file name **sample.txt**.

Now each text file will contain the center coordinates, height and width of the box, and the class of the object. An image can have more than 1 object, so its text file will have multiple lines, one for each object.

Each line of the text file will have below format

```
<object-class> <x_center> <y_center> <width>
<height>
```

Object-Class: It is a number, which represents the class of the object. It ranges from **0 to (number of classes – 1)**. Suppose we are looking for the objects of the 2 classes, then this number could be 0 or 1.

x_center, **y_center, width, height**, are the x and y coordinates of the center, height, and width of the boundary box. It can ranges [0.0, 1.0].

Annotation is the tedious task and we have to do this, but don't worry, in the next section we'll talk about a python based GUI tool for annotation.

LabelImg annotation tool.

Before we start annotation, we need to create some directories. Inside your project folder, create one folder named **data** (you can name it anything). Inside the **data** folder create two sub-folders **images** and **labels** (name should not be changed) to store the images and their labels (annotation). Store all the images into the **data/images** folder.

Now let's talk about the tool, **LabelImg** is a graphical image annotation tool, which is based on QT and the Python. It provides the option to save the annotation in PASCAL VOC format (used in ImageNet) and YOLO format.

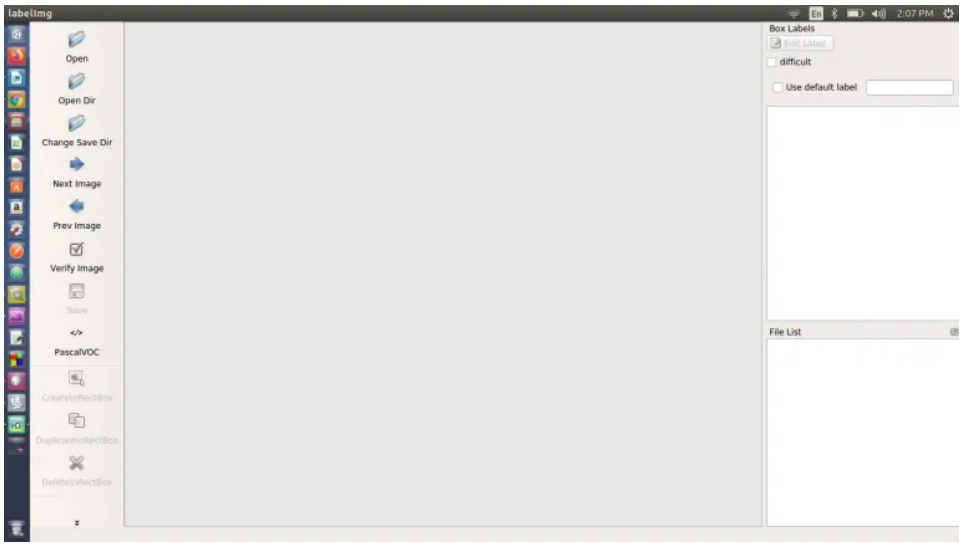
Installation:

You can install it directly using pip.

```
pip3 install labelImg
```

Usage:

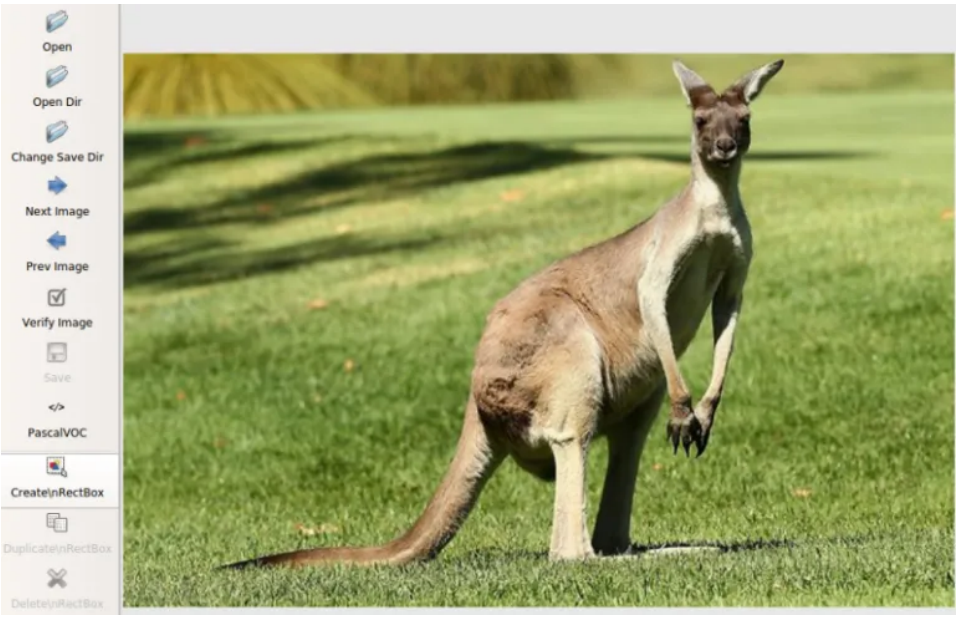
Once we installed it, open it by writing **labelImg** on terminal.



LabelImg Opening Screen

As you can see from the above image that LabelImg is started with empty workspace, and some option are at the left side of the image.

Use **Open Dir** options and choose the **images** folder, which has all the images. This will load the list of all the available images.



Crate Annotation Box

After opening the image, first change the annotation format, by clicking on **Pascal/VOC** option, it will change to **YOLO**.

Once the format is changed just click on the **Create\n Rect Box** option, after that you can create the box(es) on the images. If there are more than one object then create multiple boundary boxes. Try to create box(es) as accurately as possible, even though it is capturing other objects also.

When you are done with creating a boundary box, a small dialog box will pop up, in which you have to write a class of the object, for this example it will be a kangaroo. You have to write only the first time for every new class, from the next time it will show the list of all previously written classes, so choose accordingly.

After creating the boundary box(ex) and mentioning class click on the **Save** button and save the **.txt** file in the **labels** folder. This **labels** folder will contain a **classes.txt** which will have a list of all the classes of objects.

If everything works fine then annotate all the images, I know its a boring task but no other option available.

After annotation, we need to get the pretrained model, which we'll do in the next section.

Get the Darknet Model.

Clone the repository inside the project folder (at the same level as the data folder).

```
git clone https://github.com/AlexeyAB/darknet.git
cd darknet
```

If you have **NVIDIA GPU** based machine and installed **CUDA** then make **GPU=1** in the Makefile. You can also build it with OpenCV by making **OPENCV=1** in the Makefile, but make sure you have OpenCV installed (OpenCV for C/C++ not for Python).

After making changes just run **make**, this will create the executable **darknet.**

For the installation on other Operating Systems just check the here.

Download pretrained weights.

For training, we use convolution weights that are pretrained on the ImageNet dataset. Download the weight file inside the **data** folder using the command given below

```
wget https://pjreddie.com/media/files/darknet53.con
```

Configuration for custom dataset.

The pretrained model is been trained for the high number of classes and the size of the dataset was different, but to use it for our purpose we need to modify some config. So let's do it step by step.

First, create the **yolo-obj.cfg** file inside **darknet/cfg** folder, and then copy and paste the content from **yolov3.cfg** (it can be found in the same **darknet/cfg** folder).

Now we make changes in **yolo-obj-cfg** file.

Classes.

We have to change the number of classes according to our dataset. Search for the **classes** in the file, you'll find the multiple results, so change the number of classes in the **[yolo]** sections. For example, if you are using 3 classes then make **classes=3**.

Batch.

This is the number of images chosen for each batch. You should choose this according to your size of system memory (GPU), but ideally, keep it 64.

```
batch=64
```

Subdivisions.

The batch is again divide into the blocks of images, keep it 16.

```
subdivisions=16
```

Width and Height.

By default **width=608** and **height = 608**, you can keep it as it is but if you want to change it then make sure that the number should be a multiple of 32.

Max Batches.

It is the maximum number of batches you want to run for training.

```
max_batches = (number of classes) * 2000
```

This can't be less than 4000 even you are using 1 class. if you are using 2 classes then it should be 4000 and for 3 classes it should be 6000. Use the result of the equation in cfg file, which would be number.

Steps.

Steps should be 80% and 90% of the max_batches. If the max_batches = 6000 then steps would be,

```
steps=4800,5400
```

Filters.

Check for **filters=255**, you will find 3 results in [convolutional] section just before [yolo] section. Change it with the result of the following calculation,

```
filters=(classes + 5)x3
```

So for 1 class, it will be **filters=18** and for 2 classes it will be **filters=21**. Don't write the equation in the **cfg** file, write the resultant value (a number).

Check out this thread for better understating about all the config parameters.

Training on custom data.

Now we have our Model and custom dataset ready, so finally its time for actual training.

We need to create some additional files to support the training.

Train/Test split.

For training and testing, we need to provide the lists of images in the text files.

Create two files inside the **data** folder, **train.txt,** and **test.txt**. These files will have the path of the images and path should relative to the **darknet** executable. If the relative path doesn't work then you can provide the full path also.

```
../data/images/1.jpg
../data/images/2.jpg
../data/images/3.jpg
../data/images/4.jpg
```

Since I am using Linux, so the path of the images (for both train.txt and test.txt) should be like above, for other OS you can change accordingly.

Put 80% of the total images into the **train.txt** and rest 20% into **test.txt**, but make sure there should not be any common image in these files.

*If the given image file has its annotation text file in the **labels** folder then only it should be in the **train.txt** or **text.txt**, otherwise, it will give an error during training.*

.names file

We need to create a file that will have the name of all the classes (like classes.txt), so just create **obj.names** file inside the **data** folder. The structure of the file should be as given below.

```
person
car
dog
```

.data file

This file contains the paths of other files and info about the number of classes. Create **obj.data** inside the **data** folder and save the contents given below.

```
classes= 2
train  = ../data/train.txt
valid  = ../data/test.txt
names  = ../data/obj.names
backup = backup/
```

First line is just number of classes of objects.

Second, third and fourth lines contain the paths of the other files, but again these paths should be relative to the **darknet** executable.

Backup is the path where the weights will be stored during training, you can find it inside the **darknet** repository. After every 100 iterations, the weights will be stored.

Start the training.

Reaching this point was really a long journey, but finally, the destination arrived.

This training is gonna be very computationally expensive, so if you have a GPU machine then well and good otherwise try **Google Colab** (GPU based cloud computing platform) for training, it's free. If you never used Google Colab before then check out Google Colab.

So if you followed every instruction then just run the command below and start the training.

```
./darknet detector train ../data/obj.data cfg/yolo-
```

Training is gonna take long time, so get yourself a drink and take rest.

This training keeps saving the weights after every 100 iterations in the **backup** folder, so after every 100 iterations, you can stop it and restart it from the same point.

So suppose you stopped the training after 500 iterations, then to restart from this point just run the following command.

```
./darknet detector train ../data/obj.data cfg/yolo-
```

Testing.

For testing multiple images run the following command.

```
./darknet detector test ../data/obj.data cfg/yolo-ol
```

In the above line, 1000 was the last checkpoint, you can change it accordingly. After running the above command you'll be prompted with "**Enter Image Path**", so pass the path of the test image and it will give you the class name with a confidence score.

If you want to get results for multiple images and save it to a file then run

```
./darknet detector test ../data/obj.data cfg/yolo-v
```

Above command will run on all the images from the **test.txt** and save the result in **result.txt** inside the **data** folder.

Checkout this for more options for testing.

So this is all about **How to train YOLOv3 on the custom dataset,** if you have any doubt or issue please let me know in comments.

Thanks for reading !

Tags: Computer Vision, Object Detection, YOLOv3

Related Posts

Video Classification in Keras using ConvLSTM

Region Proposal Network and Faster R-CNN

Tesseract : An OCR engine by Google

Back to Top ↑