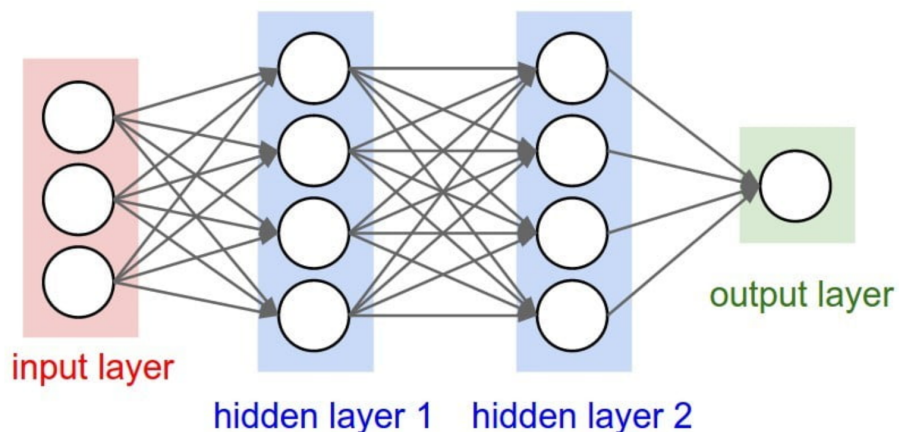


Deploying Yolo to VastAI

Background about Yolo and Deep Learning

Basic concepts about Deep Learning

Very simply put, a deep learning model is just a set of layers filled with neurons, connected to each other. It gets fancier and more complicated of course.



Each neuron takes as input the outputs of the previous layer of neurons multiplied by a weight. The weight is simply a coefficient (usually a float number).

Through training, we want to find the optimal set of weights that lead the network to make the right decision.

This is done by optimizing a loss function but I won't go into too much details, as it is not useful to know in order to use the model

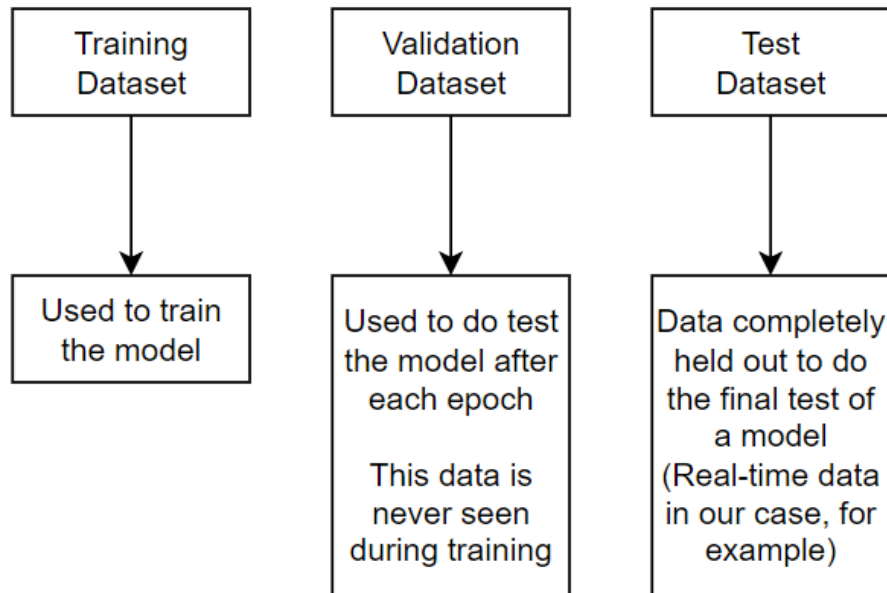
Training, Validation and Testing

The model is trained on a training dataset. In the case of Yolo, we feed it an image and the associated annotations.

In training, it is called an “epoch” when the model has been trained on the entire training dataset once. Models are generally trained over many epochs (10, 100+).

A validation dataset is used between each epoch to do a small estimation of the model’s performance. This data is never seen during training.

Finally, the real test of the model is done on unseen data. In our case, probably real-time data from the markets.



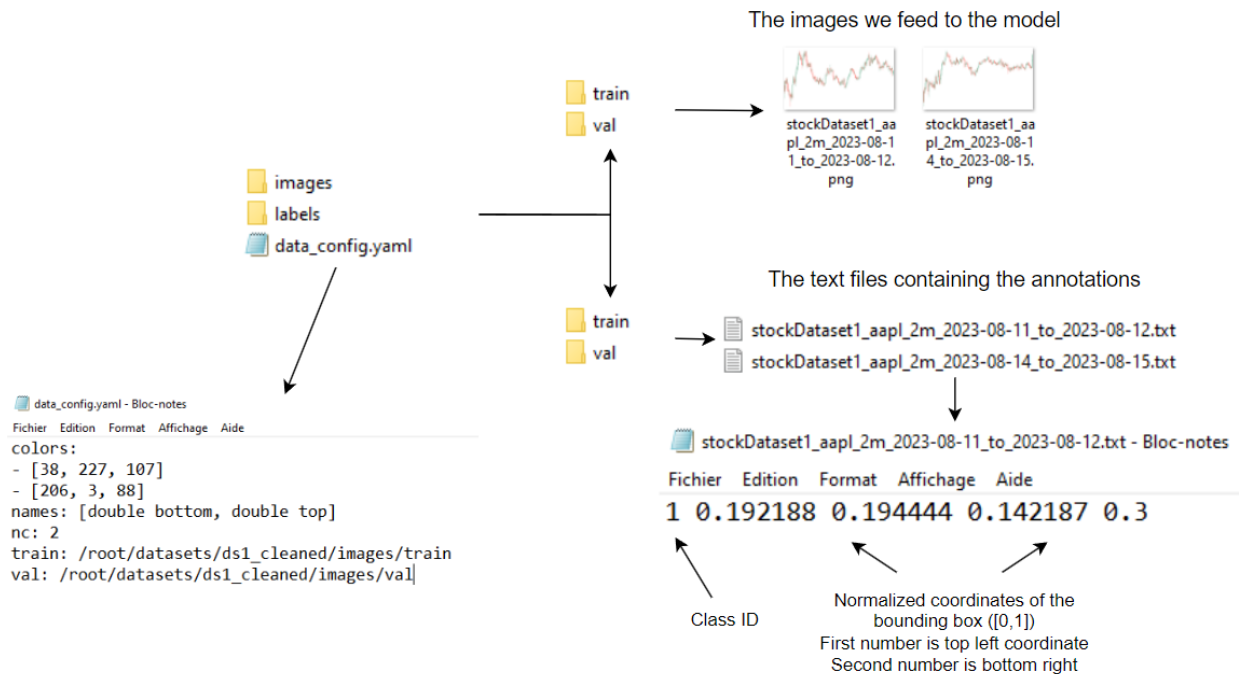
Between each training run, we want to look at the results we get, and decide what to change for the next training run.

We can also determine if the data is well balanced, or if we need more instances of a specific class depending on the results. There are more advanced optimizations and fine-tuning that can be done by digging deeper in the models.

Yolo Dataset Structure

It is important to understand how a Yolo dataset is organized, as there are important paths in the `data_config.yaml` file.

The image below shows the contents of the dataset.



At the root of the dataset, a data_config.yaml file can be found.

This is a configuration file that specifies:

- The color of the bounding boxes for each class
- The name of each class
- The number of classes
- The paths to the images and labels (here, an absolute path is used)

The “images” and “labels” directories both contain two subdirectories: “train” and “val”, for the training and validation datasets, respectively.

Images/train contains all of the training images. Labels/train contains the associated labels, which are named the same as the image, with the exception of having a .txt extension.

Inside the annotation file, each object is defined by its class ID and the points that indicate the coordinates of the bounding box (as relative values). The val subfolders contain the same data for the validation dataset.

The path in the yaml file matters, as this is where the Yolo script will look for the data when it runs. If the path is wrong, or if a relative path that is not pointing to the right folder is given in the yaml file, yolo will not run.

Installing YOLOv8 locally

YOLOv8 can be installed very simply. All that is required is a Python ≥ 3.8 environment with PyTorch ≥ 1.8 . A Python virtual environment or an Anaconda environment can both be used.

The repository for YOLO, containing all the documentation can be found here:

<https://github.com/ultralytics/ultralytics>

The following command is used to install yolo:

- `pip install ultralytics`

This command should take care of installing all of the necessary dependencies.

It is also possible to pull the entire repo, and use “`pip install -r requirements.txt`” in the working directory, but this is a longer method.

Training YoloV8

The following script is used to train YoloV8

```
1 from ultralytics import YOLO
2
3 yaml_path = "/root/datasets/ds1_cleaned/data_config.yaml"
4
5 # Load a model
6 model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)
7
8 # Use the model
9 model.train(data=yaml_path, epochs=10, imgsz=(1280,720)) # train the model
10 metrics = model.val() # evaluate model performance on the validation set
11 path = model.export(format="onnx") # export the model to ONNX format
```

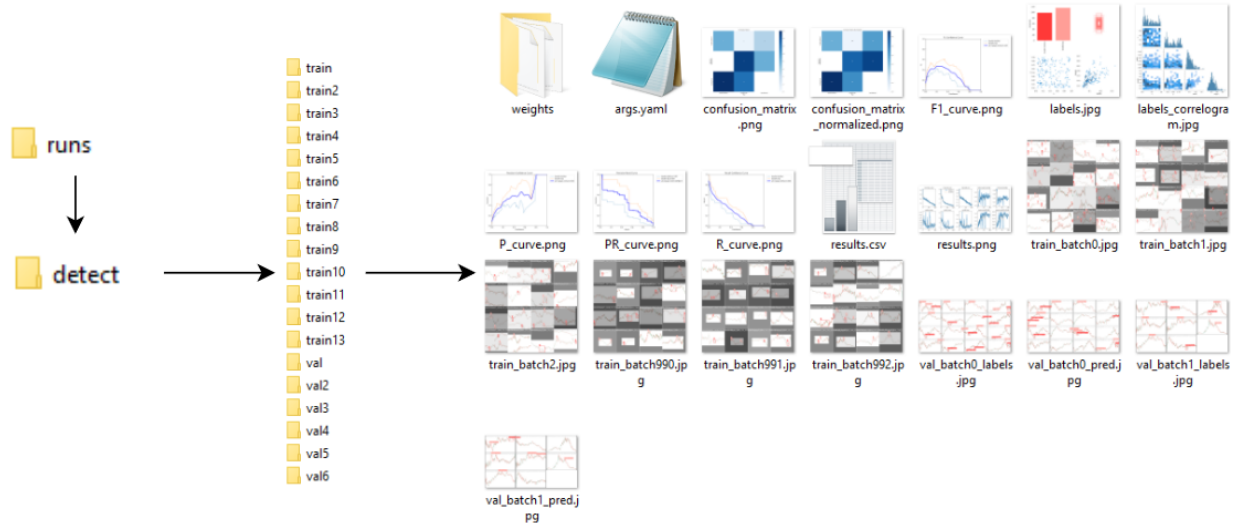
Very simply, this script:

- Imports YOLO,
- Specifies the path towards the .yaml file, which is the configuration file of the dataset.
- Loads a pre-trained YOLO model, in this case, the "nano" model, which is the smallest one (Can be replaced by yolov8s.pt, yolov8m.pt, yolov8l.pt, yolov8x.pt)
- Trains the model for 10 epochs, specifying the size of the images (When using larger models, you might want to add "batch=8" to only use 8 images per training batch, as limited VRAM might make it impossible to run with the default batch size)
- Evaluates the model on the validation dataset
- Exports the weights of the model and saves them to onnx format

By default, yolo should save the best and last weights of the model as “best.pt” and “last.pt”
These are the weights we will be interested in when we want to deploy a pre-trained Yolo in real-time.

Results

After running, Yolo will create a “runs” folder and a “detect” subfolder. When performing a segmentation task, this subfolder will be called “segment”.



Inside, there will be a train folder for each training run, and identically for each time the process has gone all the way through to the validation (not manually stopped before).

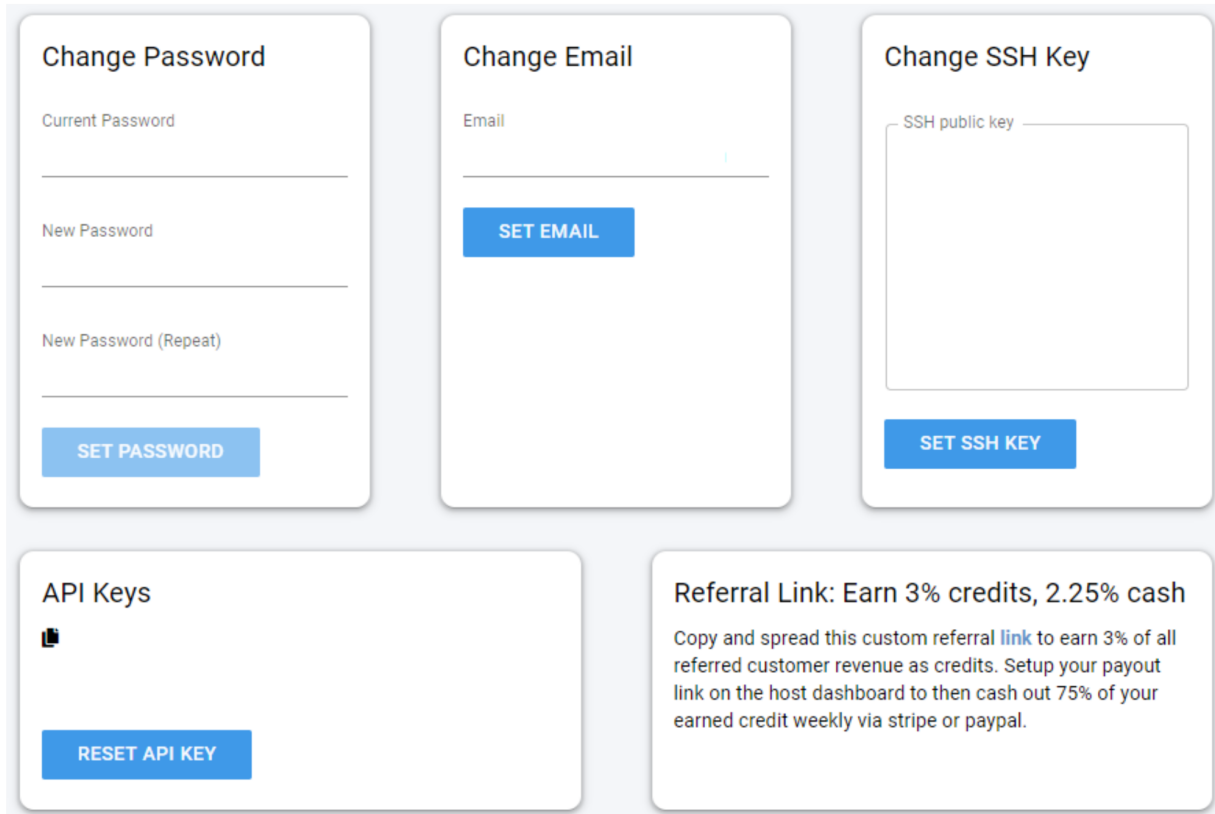
You can find results on the training data in the train_batch0/1.jpg images, and identically for the validation for the val_batch0/1_pred.jpg images.

As the training process finishes, you will also get detailed results of the Recall, Precision, mAP50 and mAP50-95 directly in the console.

```
Model summary (fused): 268 layers, 43608150 parameters, 0 gradients
val: Scanning /root/datasets/ds1_cleaned/labels/val.cache... 40 images, 0 backgrounds, 0 corrupt: 100%|_____| 40/40
      Class      Images  Instances   Box(P          R      mAP50  mAP50-95): 100%|_____| 5/5 [00:02<00
        all         40         52    0.439    0.443    0.358    0.182
    double bottom     40         24    0.549    0.458    0.312    0.165
        double top     40         28    0.33    0.429    0.403    0.199
Speed: 5.4ms preprocess, 18.7ms inference, 0.0ms loss, 3.8ms postprocess per image
Results saved to runs/detect/val5
Ultralytics YOLOv8.0.178 Python-3.10.11 torch-2.0.1 CPU (Intel Core(TM) i9-7900X 3.30GHz)
```

Using YOLOv8 on VastAI

After creating a VastAI account, you will be able to see your API key in the bottom left of that screen, in the “account” section



The screenshot displays the VastAI account settings interface. It features five main sections arranged in a grid:

- Change Password:** Includes input fields for 'Current Password', 'New Password', and 'New Password (Repeat)', followed by a 'SET PASSWORD' button.
- Change Email:** Includes an 'Email' input field and a 'SET EMAIL' button.
- Change SSH Key:** Includes a large text area for 'SSH public key' and a 'SET SSH KEY' button.
- API Keys:** Includes a small icon representing a key and a 'RESET API KEY' button.
- Referral Link:** Contains the text 'Referral Link: Earn 3% credits, 2.25% cash' and a paragraph explaining the referral program: 'Copy and spread this custom referral link to earn 3% of all referred customer revenue as credits. Setup your payout link on the host dashboard to then cash out 75% of your earned credit weekly via stripe or paypal.'

Setting up the environment

I have personally used PowerShell on Windows to do everything, but any console client will work, as long as you can connect using SSH, and as long as you can create a Python environment and install YOLO.

You can install Powershell for Windows here:

<https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.3>

You can install OpenSSH for Powershell here:

https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse?tabs=powershell

If you are comfortable with your environment, you can use whatever works for you, on any operating system too.

Installing and testing VastAI

This page shows a few interesting commands: <https://cloud.vast.ai/cli/>

The most useful commands are:

- `pip install --upgrade vastai`
- `vastai set api-key`
`3e784dc8199a10413f1eccf2883cc47c664bebb42d21fb768c491c8f0e794de0`
- `vastai search offers`

The first one installs vastAI in your Python environment.

The second one sets your API key, so you will have to use your own API key, found in your account section.

The last command shows you the GPUs available, but I recommend using the website to rent instances, as it provides a lot of useful commands and information.

Generating a SSH key

Before renting an instance, you will need to make sure that you can connect to it using SSH. For this, you will need to generate a SSH key on your computer, unless you already have one.

This can be done using the command:

- `ssh-keygen`

You can use a passphrase or leave it empty.

You can change the file path, or use the default one.

If you leave the default path, you should be able to find your SSH key at this path:

- `C:\Users\<YourUsername>\.ssh\id_rsa`

Once your SSH key is generated, you should go to the file with your PUBLIC ssh key:

- `C:\Users\<YourUsername>\.ssh\id_rsa.pub`

And copy/paste the key in the SSH section in your account tab, on the VastAI website.

Renting an instance

You can see a list of all the available GPUs at this link: <https://cloud.vast.ai/>

You can explore the different filters to see what GPUs are available. To train Yolo, I have personally used a single RTX 3090, and the model was trained in a matter of minutes.

I recommend using “On-Demand” GPUs. They are more expensive, but with the “Interruptible” GPUs, anytime someone outbids you, you instantly lose access to the instance without warning, and you lose everything.

16GB of space should be plenty, after a few training run, I only used about 2GB with all the stored results. Space is orders of magnitude cheaper than the GPU anyway.

Instance Configuration

no image selected

Launch Type: ssh
On-start script: Not set

EDIT IMAGE & CONFIG...

Disk Space To Allocate
16.00 GB

#GPUs: ANY 0X 1X 2X 4X 8X 8X+ On-Demand Any GPU Planet Earth Auto Sort

m:11742	host:73118	Spain, ES	ROMED8 PCIe 4.0,16x 12.6 GB/s	↑1261 Mbps ↓2170 Mbps	100 ports	verified Max Duration 3 days	\$0.376/hr
V8RT.ai	1x RTX 4090	81.4 TFLOPS	24 GB	AMD EPYC 7532 ...	nvme	74.1 DLPerf	RENT
Type #0254251	Max CUDA: 12.0	3537.3 GB/s	16.0/64 cpu	64/258 GB	4838 MB/s	150.8 GB	196.9 DLP/\$/hr

m:9218	host:9656	Spain, ES	ROMED8 PCIe 4.0,16x 12.7 GB/s	↑801 Mbps ↓683 Mbps	200 ports	verified Max Duration 3 days	\$0.741/hr
V8RT.ai	2x RTX 4090	162.8 TFLOPS	24 GB	AMD EPYC 7443...	nvme	144.3 DLPerf	RENT
Type #0702346	Max CUDA: 12.0	3407.1 GB/s	24.0/48 cpu	258/516 GB	3140 MB/s	348.0 GB	194.7 DLP/\$/hr

m:9570	host:9656	Spain, ES	H12SSL PCIe 4.0,8x 12.7 GB/s	↑794 Mbps ↓692 Mbps	100 ports	verified Max Duration 3 days	\$0.376/hr
V8RT.ai	1x RTX 4090	81.4 TFLOPS	24 GB	AMD EPYC 7543 ...	nvme	73.4 DLPerf	RENT
Type #0702472	Max CUDA: 12.0	3465.0 GB/s	16.0/64 cpu	129/516 GB	3612 MB/s	178.0 GB	195.0 DLP/\$/hr

m:9218	host:9656	Spain, ES	ROMED8 PCIe 4.0,16x 12.7 GB/s	↑801 Mbps ↓683 Mbps	100 ports	verified Max Duration 3 days	\$0.376/hr
V8RT.ai	1x RTX 4090	81.4 TFLOPS	24 GB	AMD EPYC 7443...	nvme	73.3 DLPerf	RENT
Type #0702346	Max CUDA: 12.0	3407.1 GB/s	12.0/48 cpu	129/516 GB	3140 MB/s	174.0 GB	195.0 DLP/\$/hr

m:8683	host:9656	Spain, ES	ROMED8 PCIe 4.0,16x 25.0 GB/s	↑714 Mbps ↓590 Mbps	100 ports	verified Max Duration 3 days	\$0.376/hr
V8RT.ai	1x RTX 4090	81.4 TFLOPS	24 GB	AMD EPYC 7443...	nvme	73.3 DLPerf	RENT
Type #0340222	Max CUDA: 12.0	2711.1 GB/s	12.0/48 cpu	129/516 GB	3255 MB/s	190.8 GB	195.0 DLP/\$/hr

Filter Options

Availability

Host Reliability
90.00%

Max Instance Duration
3 days

Before renting an instance, you have to pick the docker image that will run on it.

Instance Configuration

Image: [Pytorch 2.0.1 cuda11.7 devel](#)

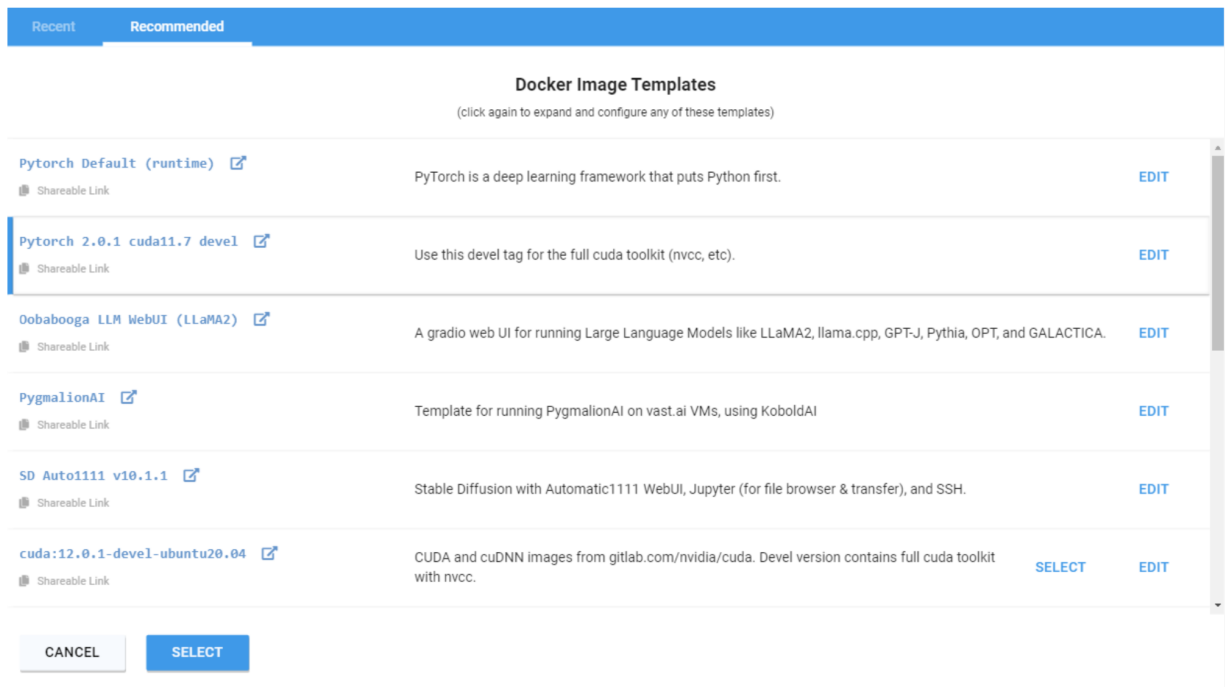
Image CUDA version: 11.7
Incompatible images hidden

Launch Type: jupyter
On-start script: Set

EDIT IMAGE & CONFIG...

Disk Space To Allocate
16.00 GB

After clicking on “Edit image & config” on the panel on the left, you will see a window with two tabs: “Recent” and “Recommended”. In the Recommended tab, you will find a “Pytorch 2.0.1 cuda11.7 devel” image, which is the one we are going to use.



In the future, I will configure a custom Docker image, because the one recommended lacks a couple of things that require a few commands to install. It would be better to have a “ready to deploy” docker instance. It also saves rental time spent configuring the machine over and over.

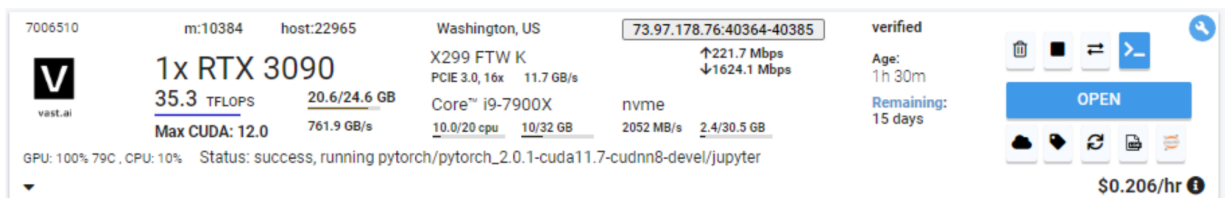
Connecting and setting up the environment

Once the instance is rented, you can manage it and get useful commands at:

<https://cloud.vast.ai/instances/>

You can start, stop and destroy your instance here.

Clicking on the console/arrow blue button on the right gives you the entire ssh command necessary to connect to the instance.



If your environment is correctly configured (SSH working, VastAI API key set, valid SSH key), copying and pasting the command should land you in a linux environment. The command should look like:

- `ssh -p 40340 root@162.156.171.190 -L 8080:localhost:8080`

With different parameters.

The first thing you want to do is to create two folders: scripts and datasets:

- `mkdir scripts`
- `mkdir datasets`

Then you want to install yolo using:

- `pip install ultralytics`

After everything is done downloading and installing, you should preemptively fix a missing package that would make running yolo impossible. It seems like this docker image is lacking an OpenGL package. The commands to install it are:

- `apt-get update`
- `apt-get install -y libgl1-mesa-glx`

Transferring the script and the dataset

After that, you will have to transfer both the yolo script and the dataset. This can be done using the SCP command.

Through another console client you should go to where the yolo script is stored on your local computer.

After that, the command you want to type looks like this:

- `scp -P 40340 yolo_test.py root@162.156.171.190:~/scripts/`

Assuming you are located where your yolo script is, and using the right port and IP (found in the Instance section on the VastAI website), you should be able to transfer the ready to run script to the remote instance.

Then you can transfer the entire dataset with this command.

It uses the '-r' parameter to transfer the entire directory, and the first './dataset' should be replaced with the name of the dataset directory:

- `scp -P 40340 -r ./dataset root@162.156.171.190:~/datasets/`

Once this is done, you can get to the "scripts" directory and use:

- `python yolo_test.py`

to execute the script.

If you want to modify the script, you can either install vim on the instance using:

- `apt update && apt install -y vim`

And then using:

- vim yolo_test.py

You can modify the script (number of epochs, batch size, other parameters..)

Otherwise, you can:

- modify the script locally on your computer
- remove the old script on the remote instance (rm yolo_test.py)
- and transfer the script again using scp

Some IDE allow the direct edition of a script on a distant machine, I tried with Visual Studio Code, but I kept getting a connection timeout. This is something I will try to figure out. Using vim, or modifying the script locally works well, as it only requires two commands, but it is not optimal.

Yolo Results

The first yolo run should create a "runs" folder in your working directory.

Under this, a "detect" subfolder will be created as well as "train" and "val" subfolders containing the results of each run, as explained in the Yolo section.

If you want to keep these results, you should transfer them back to your local computer.

This can be done using the scp command, but reversing the addresses.

- scp -P 40340 -r root@162.156.171.190:~/scripts/runs /runs

This command would copy the entire "runs" directory, located at ~/scripts/ to a new "runs" directory created on your local machine, in your current directory.

These folders also contain the yolo weights, so the transfer may last a while after a few runs, if you decide to SCP everything. It will be necessary to have the weights for the real-time model, but if you are only interested in the results, you could manually go to the last val subfolder in the runs/detect folders, and SCP the files you are interested in.

Stopping and shutting down the instance

On VastAI, as long as your instance is running, you pay for it, even if no computations are running. I don't know if a stopped instance costs anything, so you might be able to stop the instance, and start it again when you need it. However, all instances have a time limit, after which the instance is automatically deleted. I think the default is 3 days, but it varies a lot from case to case.

By going into your instances, you can stop (stop button) and then destroy (trashcan button) the instance once you are done with it.

7006510 m:10384 host:22965 Washington, US 73.97.178.76:40364-40385 verified

1x RTX 3090
 35.3 TFLOPS 20.6/24.6 GB
 Max CUDA: 12.0 761.9 GB/s

X299 FTW K
 PCIe 3.0, 16x 11.7 GB/s
 Core™ i9-7900X nvme
 10.0/20 cpu 10/32 GB 2052 MB/s 2.4/30.5 GB

GPU: 100% 79C, CPU: 10% Status: success, running pytorch/pytorch_2.0.1-cuda11.7-cudnn8-devel/jupyter

Age: 1h 30m
 Remaining: 15 days
 \$0.206/hr

In that case, my instance was 1h30 old, and I had a maximum of 15 days before losing it.

This document should give you the basics, from understanding how Yolo works, how to train it, deploy it to vastAI, and how to understand the results.

Here is a summary of the useful links:

Yolo:

<https://github.com/ultralytics/ultralytics>

<https://github.com/ultralytics/ultralytics/blob/main/requirements.txt>

VastAI:

<https://cloud.vast.ai/account/>

<https://cloud.vast.ai/>

<https://cloud.vast.ai/cli/>

Powershell:

<https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-windows?view=powershell-7.3>

OpenSSH:

https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse?tabs=powershell