DeepStack Documentation

Release 0.1

DeepQuestAI

Mar 04, 2019

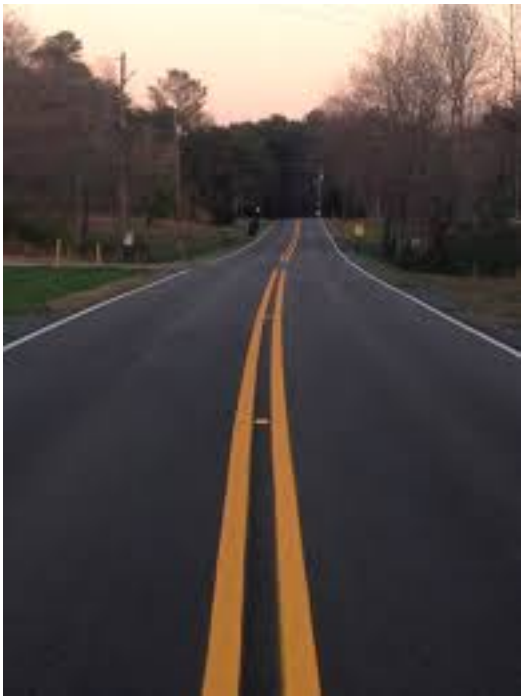
Tutorials:

DeepStack is an AI server that empowers every developer in the world to easily build state-of-the-art AI systems both on premise and in the cloud. The promises of Artificial Intelligence are huge but becoming a machine learning engineer is hard. DeepStack runs on the docker platform and can be used from any programming language.

You can learn more about Docker on Docker's Website Visit Docker Getting Started for instructions on setting up and using Docker for the first time.

CHAPTER 1

DeepStack is developed and maintained by DeepQuest AI



Below, using DeepStack we attempt to classify the scene of the above image

```
import requests

image_data = open("image.jpg","rb").read()

response = requests.post("http://localhost:80/v1/vision/scene",files={"image":image_
↪data}).json()

print(response)
```

Result

```
{'label': 'highway', 'success': True, 'confidence': 0.63377845}
```

You simply send in an image by POST and deepstack returns a JSON response detailing the label of the image as well as the confidence of the prediction on a scale of 0 - 1.

CHAPTER 2

Installing DeepStack - CPU Version

The code above demonstrates using DeepStack to predict the scene of an image, to run this, you can install DeepStack and start it with a single docker command.

2.1 Step 1: Install Docker

If you already have docker installed, you can skip this step.

On Linux

```
sudo apt-get update
sudo apt-get install curl
curl -fsSL get.docker.com -o get-docker.sh && sh get-docker.sh
```

On Windows or MacOS

Follow instructions on Docker Getting Started

2.2 Step 2: Install DeepStack

```
docker pull deepquestai/deepstack
```

Once installed, you can run DeepStack with the command below

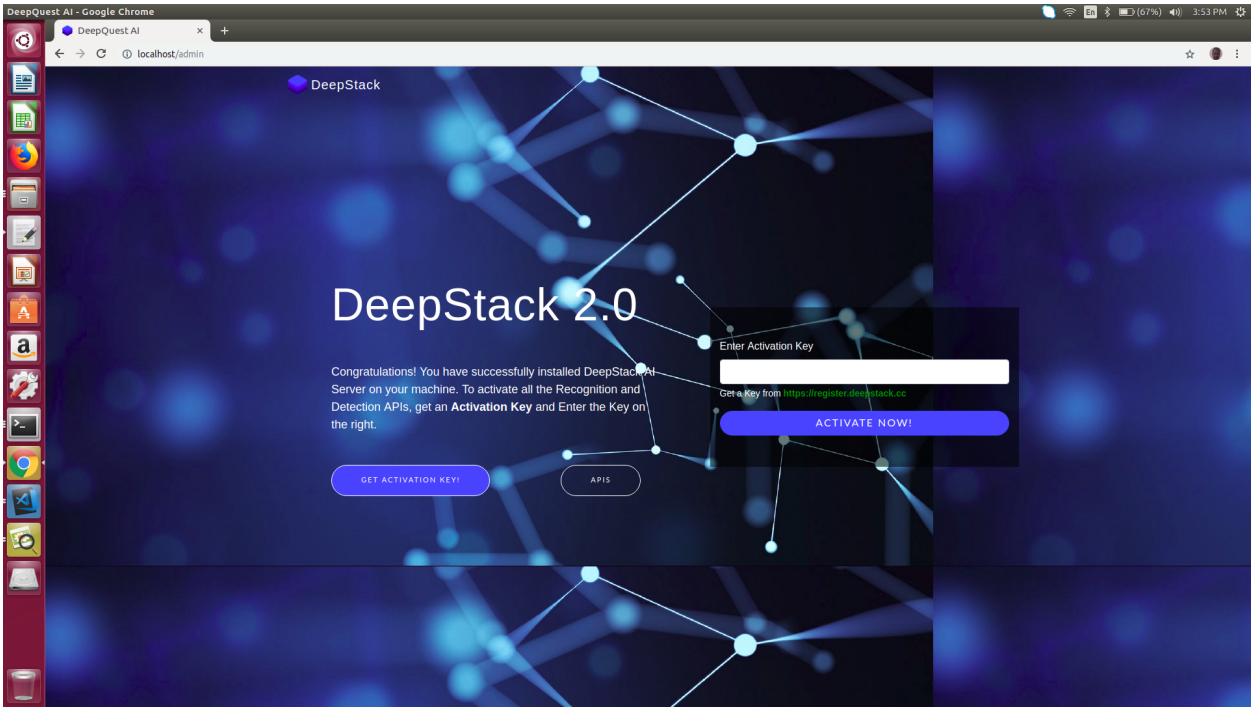
```
docker run -e VISION-SCENE=True -v localstorage:/datastore -p 80:5000 deepquestai/
↪deepstack
```

The command above runs deepstack with the scene recognition activated, once this is running, you can run the example above.

2.3 Step 3: Activate DeepStack

The first time you run deepstack, you need to activate it following the process below.

Once you initiate the run command above, visit localhost:80/admin in your browser. The interface below will appear.

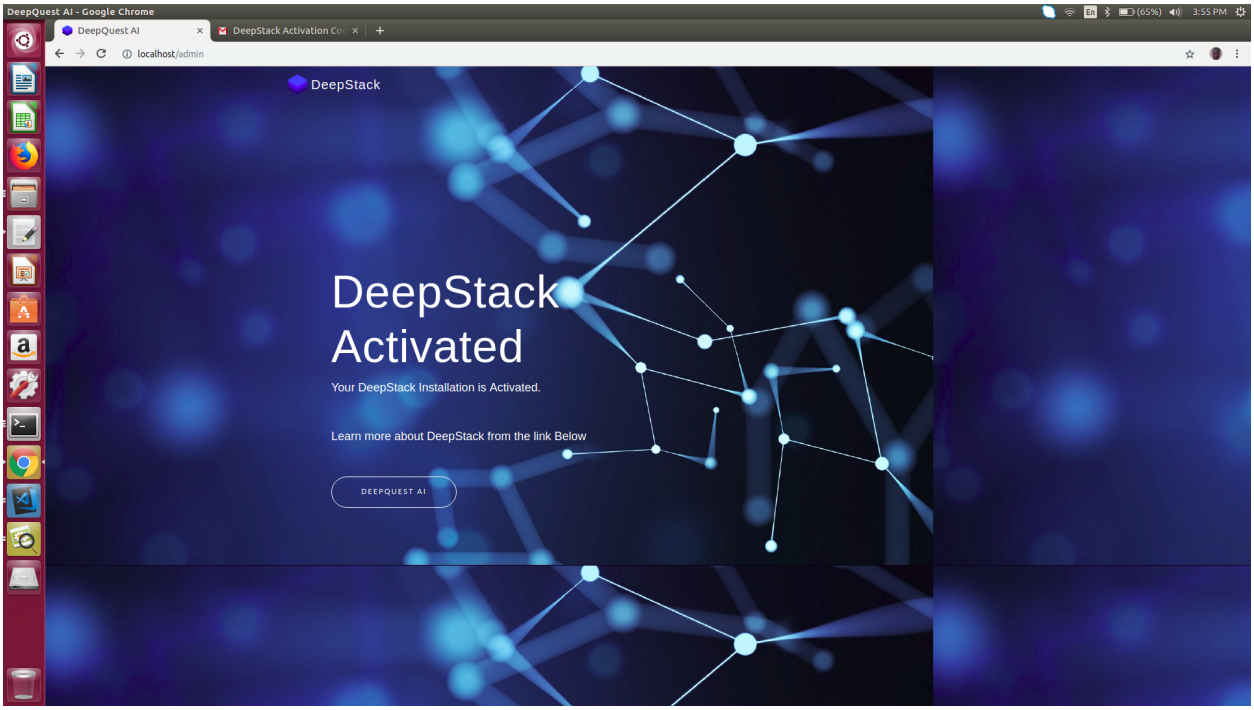


You can obtain a free activation key from https://register.deepstack.cc

Enter your key and click **Activate Now**

The interface below will appear.

This step is only required the first time you run deepstack.


DeepQuest AI - Google Chrome
DeepQuest AI
DeepStack Activation Co
localhost/admin
DeepStack
DeepStack Activated
Your DeepStack Installation is Activated.
Learn more about DeepStack from the link Below
DEEPQUEST AI

CHAPTER 3

GPU Accelerated Version

DeepStack runs many times faster on machines with NVIDIA GPUS, to install and use the GPU Version, read *Using DeepStack with NVIDIA GPU*

HARDWARE AND SOFTWARE REQUIREMENTS

DeepStack runs on any platform with Docker installed. However, for best performance, the following minimum requirements are highly recommended.

- Intel Core i5 processor
- 8 GB RAM
- 10 GB Disk Space
- Linux or Windows 10 Pro

NOTE

DeepStack works best on linux Systems

3.1 Getting Started with DeepStack

DeepStack is distributed as a docker image. In this tutorial, we shall go through the complete process of using DeepStack to build a Face Recognition system.

3.1.1 Setting Up DeepStack

Follow instructions on read *DeepStack Beta - Nodejs Guide* to install the CPU Version of DeepStack If you have a system with Nvidia GPU, follow instruction on read *Using DeepStack with NVIDIA GPU* to install the GPU Version of DeepStack

To install the GPU Accelerated Version, follow *Using DeepStack with NVIDIA GPU*

Starting DeepStack

Below we shall run DeepStack with only the FACE features enabled

```
sudo docker run -e VISION-FACE=True -v localstorage:/datastore -p 80:5000 deepquestai/
↪deepstack
```

Basic Parameters

-e VISION-FACE=True This enables the face recognition APIs, all apis are disabled by default.

-v localstorage:/datastore This specifies the local volume where deepstack will store all data.

-p 80:5000 This makes deepstack accessible via port 80 of the machine.

NOTE FOR THE GPU VERSION

If you installed the GPU Version, remmember to add the args args **–rm –runtime=nvidia** The equivalent run command for the gpu version is

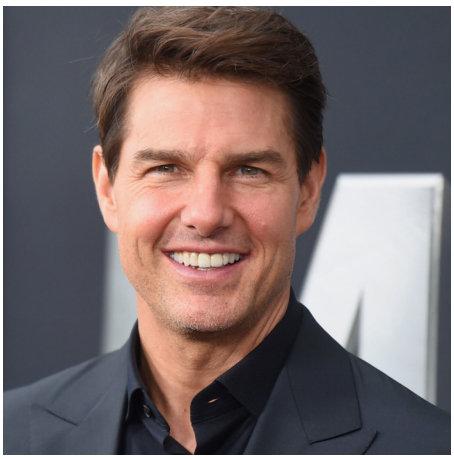
```
sudo docker run --rm --runtime=nvidia -e VISION-FACE=True -v localstorage:/
↪datastore \
-p 80:5000 deepquestai/deepstack:gpu
```

Face Recognition

Think of a software that can identity known people by their names. Face Recognition does exactly that. Register a picture of a number of people and the system will be able to recognize them again anytime. Face Recognition is a two step process: The first is to register a known face and second is to recognize unknown faces.

REGISTERING A FACE

Here we are building an application that can tell the names of a number of popular celebrities. First we collect pictures of a number of celebrities and we register them with deepstack



Below we will register the faces with their names

```
import requests

tom_cruise = open("cruise.jpg","rb").read()
adele = open("adele.jpg","rb").read()
elba = open("elba.jpg","rb").read()
perri = open("perri.jpg","rb").read()

requests.post("http://localhost:80/v1/vision/face/register",files={"image":tom_cruise}
↪, data={"userid":"Tom Cruise"})
```

(continues on next page)

(continued from previous page)

```
requests.post("http://localhost:80/v1/vision/face/register",files={"image":adele},␣
↪data={"userid":"Adele"})
requests.post("http://localhost:80/v1/vision/face/register",files={"image":elba},␣
↪data={"userid":"Idris Elba"})
requests.post("http://localhost:80/v1/vision/face/register",files={"image":perri},␣
↪data={"userid":"Christina Perri"})
```

Result

```
{ success: true, message: 'face added' }
{ success: true, message: 'face added' }
{ success: true, message: 'face added' }
```

{ success: true, message: 'face updated' }

RECOGNITION

Now we shall attempt to recognize any of these celebrities using DeepStack. Below we will send in a whole new picture of Adele and DeepStack will attempt to predict the name.

Prediction code

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("test

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v
↪recognize", formData:form},function(er

    response = JSON.parse(body)
    predictions = response["predictions"
    for(var i =0; i < predictions.length
```

(continues on next page)

(continued from previous page)

```
                                    console.log(predictions[i]["userid"]

                                    }

                                })
```

Result

```
                                Adele
```

We have just created a face recognition system. You can try with different people and test on different pictures of them.

The next tutorial is dedicated to the full power of the face recognition api as well as best practices to make the best out of it.

Performance

DeepStack offers three modes allowing you to tradeoff speed for peformance. During startup, you can specify performance mode to be , "High","Medium" and "Low"

The default mode is "Medium"

You can speciy a different mode as seen below

```
sudo docker run -e MODE=High -e VISION-FACE=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack
```

Note the **-e MODE=High** above

3.2 Face Recognition

In the Getting Started, we had an overview of the face recognition API. In this section, we shall explore all the functionalities of the API.

3.2.1 Face Registeration

The face registeration endpoint allows you to register pictures of person and associate it with a userid.

You can specify multiple pictures per person during registeration.

Example

```
const request = require("request")
const fs = require("fs")

run_prediction("image1.jpg","User Name")

function run_prediction(image_path,userid){

    image_stream = fs.createReadStream(image_path)

    var form = {"image":image_stream,"userid":userid}
```

(continues on next page)

(continued from previous page)

```
    request.post({url:"http://localhost:80/v1/vision/face/register", formData:form},
↪function(err,res,body){

        response = JSON.parse(body)
        console.log(response)

    })

}
```

Result

```
{ success: true, message: 'face added' }
```

The response above indicates the call was successful. You should always check for the "success" status. If their is an error in your request, you will receive a response like

```
{error: 'user id not specified', success: False}
```

This indicates that you ommited the userid in your request. If you ommited the image, the response will be

```
{error: 'No valid image file found', success: False}
```

3.2.2 Face Recognition

The face registeration endpoint detects all faces in an image and returns the USERID for each face. Note that the USERID was specified during the registeration phase. If a new face is encountered, the USERID will be unknown.

We shall test this on the image below.

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("test-image2.jpg")

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v1/vision/face/recognize", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]
    for(var i =0; i < predictions.length; i++){

        console.log(predictions[i]["userid"])

    }

    console.log(response)

})
```

Result


TORONTO
TELEFILM
CANADA

```
Idris Elba
unknown
{ success: true,
predictions:
    [ { confidence: 0.76965684,
        userid: 'Idris Elba',
        y_min: 154,
        x_min: 1615,
        y_max: 682,
        x_max: 1983 },
       { confidence: 0,
         userid: 'unknown',
        y_min: 237,
        x_min: 869,
        y_max: 732,
        x_max: 1214 } ] }
```

As you can see above, the first user is unknown since we did not previously register her, however, Idris Elba was detected as we registered a picture of his in the previous tutorial. Note also that the full response contains the coordinates of the faces.

3.2.3 Extracting Faces

The face coordinates allows you to easily extract the detected faces. Here we shall use the Easy Image library to extract the faces and save them

```
const request = require("request")
const fs = require("fs")
const easyimage = require("easyimage")

image_stream = fs.createReadStream("test-image2.jpg")

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v1/vision/face/recognize", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]
    for(var i =0; i < predictions.length; i++){
        pred = predictions[i]
        userid = pred["userid"]
        y_min = pred["y_min"]
        x_min = pred["x_min"]
        y_max = pred["y_max"]
        x_max = pred["x_max"]

        easyimage.crop(
            {
                src: "test-image2.jpg",
                dst: userid+".jpg",
                x: x_min,
                cropwidth: x_max - x_min,
                y: y_min,
                cropheight: y_max - y_min,
            }
```

(continues on next page)

(continued from previous page)

```
        )
    }
})
```

Result

Setting Minimum Confidence

DeepStack recognizes faces by computing the similarity between the embedding of a new face and the set of embeddings of previously registered faces. By default, the minimum confidence is 0.67. The confidence ranges between 0 and 1. If the similarity for a new face falls below the min_confidence, unknown will be returned.

The min_confidence parameter allows you to increase or reduce the minimum confidence.

We lower the confidence allowed below.

Example

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("test-image2.jpg")

var form = {"image":image_stream,"min_confidence":0.30}

request.post({url:"http://localhost:80/v1/vision/face/recognize", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]
    for(var i =0; i < predictions.length; i++){
        pred = predictions[i]
        console.log(pred["userid"])
    }
})
```

Result

```
Adele
Idris Elba
```

By reducing the allowed confidence, the system detects the first face as Adele. The lower the confidence, the more likely for the system to make mistakes. When the confidence level is high, mistakes are extremely rare, however, the system may return unknown always if the confidence is too high.

For security related processes such as authentication, set the min_confidence at 0.7 or higher

3.2.4 Managing Registered Faces

The face recognition API allows you to retrieve and delete faces that has been previously registered with DeepStack.

Listing faces

```
const request = require("request")

request.post("http://localhost:80/v1/vision/face/list",function(err,res,body){
```

(continues on next page)

(continued from previous page)

```
    response = JSON.parse(body)
    console.log(response)

})
```

Result

```
{ success: true, faces: [ 'Adele', 'Christina Perri', 'Idris Elba', 'Tom Cruise' ] }
```

Deleting a face

```
const request = require("request")

var form = {"userid":"Idris Elba"}

request.post({url:"http://localhost:80/v1/vision/face/delete", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    console.log(response)
})
```

Result

```
{success: True}
```

Having deleted Idris Elba from our database, we shall now attempt to recognize him in our test image.

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("test-image2.jpg")

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v1/vision/face/recognize", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]
    for(var i =0; i < predictions.length; i++){
        pred = predictions[i]
        console.log(pred["userid"])
    }
})
```

Result

```
unknown
unknown
```

3.3 Face Detection

The face detection API detects faces and returns their coordinates. It functions similarly to the face recognition API except that it does not perform recognition.

Example

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("family.jpg")

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v1/vision/face", formData:form},function(err,
↪res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]

    console.log(response)
})
```

Result

```
{ success: true,
    predictions:
    [ { confidence: 0.99990666,
        y_min: 145,
        x_min: 626,
        y_max: 261,
        x_max: 712 },
```

(continues on next page)

(continued from previous page)

```
    { confidence: 0.99986553,
        y_min: 174,
        x_min: 543,
        y_max: 288,
        x_max: 620 },
    { confidence: 0.99986434,
        y_min: 163,
        x_min: 731,
        y_max: 242,
        x_max: 810 },
    { confidence: 0.99899536,
        y_min: 197,
        x_min: 477,
        y_max: 279,
        x_max: 542 } ] }
```

Using the face coordinates, we shall use the Easy Image library to extract the faces and save them

```
const request = require("request")
const fs = require("fs")
const easyimage = require("easyimage")

image_stream = fs.createReadStream("family.jpg")

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v1/vision/face", formData:form},function(err,
↪res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]
    for(var i =0; i < predictions.length; i++){

        pred = predictions[i]
        gender = pred["gender"]
        y_min = pred["y_min"]
        x_min = pred["x_min"]
        y_max = pred["y_max"]
        x_max = pred["x_max"]

        easyimage.crop(
            {
            src: "family.jpg",
            dst: i.toString() + "_.jpg",
            x: x_min,
            cropwidth: x_max - x_min,
            y: y_min,
            cropheight: y_max - y_min,
            }
        )

    }
})
```

Result

Performance

DeepStack offers three modes allowing you to tradeoff speed for peformance. During startup, you can specify performance mode to be , **"High" , "Medium" and "Low"**

The default mode is "Medium"

You can speciy a different mode as seen below

```
sudo docker run -e MODE=High -e VISION-FACE=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack
```

Note the **-e MODE=High** above

Setting Minimum Confidence

By default, the minimum confidence for detecting faces is 0.45. The confidence ranges between 0 and 1. If the confidence level for a face falls below the min_confidence, no face is detected.

The min_confidence parameter allows you to increase or reduce the minimum confidence.

We lower the confidence allowed below.

Example

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("family.jpg")

var form = {"image":image_stream, "min_confidence":0.30}

request.post({url:"http://localhost:80/v1/vision/face", formData:form},function(err,
↪res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]

    console.log(response)
})
```

3.4 Face Match

The face detection api compares faces in two different pictures and tells the similarity between them. A typical use of this is matching identity documents with pictures of a person.

Example

Here we shall compare two pictures of obama

```
const request = require("request")
const fs = require("fs")

image_stream1 = fs.createReadStream("test-image6.jpeg")
image_stream2 = fs.createReadStream("test-image7.jpg")

var form = {"image1":image_stream1,"image2":image_stream2}

request.post({url:"http://localhost:80/v1/vision/face/match", formData:form},
↪function(err,res,body){
```

(continues on next page)

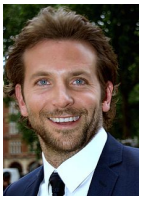
(continued from previous page)

```
    response = JSON.parse(body)
    console.log(response)
})
```

Result

```
{ success: true, similarity: 0.73975885 }
```

Here we shall compare a picture of Obama with that of Bradley Cooper



```
const request = require("request")
const fs = require("fs")

image_stream1 = fs.createReadStream("test-image6.jpeg")
image_stream2 = fs.createReadStream("test-image8.jpg")

var form = {"image1":image_stream1,"image2":image_stream2}

request.post({url:"http://localhost:80/v1/vision/face/match", formData:form},
↪function(err,res,body){
```

(continues on next page)

(continued from previous page)

```
    response = JSON.parse(body)
    console.log(response)
})
```

Result

```
{ success: true, similarity: 0.4456826 }
```

As seen above, the match for two different pictures of Obama was very high while the match for Obama and Bradley Cooper was very low.

Performance

DeepStack offers three modes allowing you to tradeoff speed for peformance. During startup, you can specify performance mode to be , **"High" , "Medium" and "Low"**

The default mode is "Medium"

You can speciy a different mode as seen below

```
sudo docker run -e MODE=High -e VISION-FACE=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack
```

Note the **-e MODE=High** above

3.5 Object Detection

The object detection API locates and classifies 80 different kinds of objects in a single image.

To use this API, you need to set **VISION-DETECTION=True** when starting DeepStack

```
sudo docker run -e VISION-DETECTION=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack
```

If using the GPU Version, run

```
sudo docker run --rm --runtime=nvidia -e VISION-DETECTION=True -v localstorage:/
↪datastore \
-p 80:5000 deepquestai/deepstack:gpu
```

Note also that you can have multiple endpoints activated, for example, both face and object detection are activated below

```
sudo docker run -e VISION-DETECTION=True  -e VISION-FACE=True -v localstorage:/
↪datastore \
-p 80:5000 deepquestai/deepstack
```

Example

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("test-image3.jpg")

var form = {"image":image_stream}
```

(continues on next page)

(continued from previous page)

```
request.post({url:"http://localhost:80/v1/vision/detection", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]
    for(var i =0; i < predictions.length; i++){

        console.log(predictions[i]["label"])

    }

    console.log(response)
})
```

Result

```
person
person
dog
{ success: true,
predictions:
[ { confidence: 99,
   label: 'person',
   y_min: 89,
   x_min: 297,
   y_max: 513,
   x_max: 444 },
 { confidence: 99,
   label: 'person',
   y_min: 114,
   x_min: 443,
   y_max: 516,
   x_max: 598 },
 { confidence: 99,
   label: 'dog',
   y_min: 354,
   x_min: 640,
   y_max: 544,
   x_max: 810 } ] }
```

Using the object coordinates, we shall use the Easy Image library to extract the faces and save them

```
const request = require("request")
const fs = require("fs")
const easyimage = require("easyimage")

image_stream = fs.createReadStream("test-image3.jpg")

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v1/vision/detection", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]
    for(var i =0; i < predictions.length; i++){

        pred = predictions[i]
        label = pred["label"]
        y_min = pred["y_min"]
        x_min = pred["x_min"]
        y_max = pred["y_max"]
        x_max = pred["x_max"]

        easyimage.crop(
            {
            src: "test-image3.jpg",
            dst: i.toString() + "_" + label+"_.jpg",
            x: x_min,
            cropwidth: x_max - x_min,
            y: y_min,
            cropheight: y_max - y_min,
        }
    )

    }

})
```

Result

Performance

DeepStack offers three modes allowing you to tradeoff speed for peformance. During startup, you can specify performance mode to be , **"High" , "Medium" and "Low"**

The default mode is "Medium"

You can speciy a different mode as seen below

```
sudo docker run -e MODE=High -e VISION-FACE=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack
```

Note the **-e MODE=High** above

Setting Minimum Confidence

By default, the minimum confidence for detecting objects is 0.45. The confidence ranges between 0 and 1. If the confidence level for an object falls below the min_confidence, no object is detected.

The min_confidence parameter allows you to increase or reduce the minimum confidence.

We lower the confidence allowed below.

Example

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("test-image3.jpg")

var form = {"image":image_stream, "min_confidence":0.30}

request.post({url:"http://localhost:80/v1/vision/detection", formData:form},
↪function(err,res,body){

    response = JSON.parse(body)
    predictions = response["predictions"]

    console.log(response)
})
```

CLASSES

The following are the classes of objects DeepStack can detect in images

```
person,   bicycle,   car,   motorcycle,   airplane,
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,   donot,   cake,   chair,   couch,   potted plant,   bed, dining␣
↪table,
toilet,   tv,   laptop,   mouse,   remote,   keyboard,   cell phone,   microwave,
oven,   toaster,   sink,   refrigerator,   book,   clock,   vase,   scissors,   teddy␣
↪bear,
hair dryer, toothbrush.
```

3.6 Scene Recognition

The traffic recognition api classifies an image into one of 365 scenes

To use this API, you need to set **VISION-SCENE=True** when starting DeepStack

```
sudo docker run -e VISION-SCENE=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack
```

If using the GPU Version, run

```
sudo docker run --rm --runtime=nvidia -e VISION-SCENE=True -v localstorage:/datastore␣
↪\
-p 80:5000 deepquestai/deepstack:gpu
```

Note also that you can have multiple endpoints activated, for example, both traffic and scene recognition are activated below

```
sudo docker run -e VISION-SCENE=True  -e VISION-TRAFFIC=True -v localstorage:/
↪datastore \
-p 80:5000 deepquestai/deepstack
```

Example

```
const request = require("request")
const fs = require("fs")

image_stream = fs.createReadStream("test-image5.jpg")

var form = {"image":image_stream}

request.post({url:"http://localhost:80/v1/vision/scene", formData:form},function(err,
↪res,body){

    response = JSON.parse(body)
    console.log(response)
})
```

Result

```
{ success: true, label: 'conference_room', confidence: 73.739815 }
```

3.7 Using DeepStack with NVIDIA GPU

DeepStack GPU Version serves requests 5 - 20 times faster than the CPU version if you have an NVIDIA GPU.

NOTE: THE GPU VERSION IS ONLY SUPPORTED ON LINUX

Before you install the GPU Version, you need to follow the steps below.

Step 1: Setup NVIDIA Drivers and CUDA

Install the NVIDIA Driver

GUIDE: Nvidia Driver Install

Install the CUDA Toolkit

GUIDE: Install CUDA Toolkit

Step 2: Install NVIDIA Docker

The native docker engine does not support GPU access from containers, however **nvidia-docker2** modifies your docker install to support GPU access.

Run the commands below to modify the docker engine

```
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \
sudo apt-key add -

distribution=$(. /etc/os-release;echo $ID$VERSION_ID)

curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | \
sudo tee /etc/apt/sources.list.d/nvidia-docker.list

sudo apt-get update

sudo apt-get install -y nvidia-docker2

sudo pkill -SIGHUP dockerd
```

If you run into issues, you can refer to this GUIDE

Step 3: Install DeepStack GPU Version

```
sudo docker pull deepquestai/deepstack:gpu
```

RUN DeepStack with GPU Access

Once the above steps are complete, when you run deepstack, add the args **–rm –runtime=nvidia**

```
sudo docker run --rm --runtime=nvidia -e VISION-FACE=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack:gpu
```

In the above example, activated only the FACE Api. You can activate multiple endpoints simultaneously as seen below

```
sudo docker run --rm --runtime=nvidia -e VISION-FACE=True -e VISION-DETECTION=True \
-v localstorage:/datastore -p 80:5000 deepquestai/deepstack:gpu
```

Here we activated two endpoints at the same time, note that the more endpoints you activate, the more the memory usage of DeepStack. GPUs have limited memory, hence, you should only activate the features you need. Your system can hang if the memory is overloaded

3.8 DeepStack Beta 2.0 - Release Notes

DeepStack Beta 2.0 features a new face detection engine, significantly improving the face detection and recognition APIS.

3.8.1 Improvements

- **New Face Detection Engine**

  The face detection APIs have been improved to detect faces even when occluded.

- **Improved Face Recognition**

  The face recognition APIs have been improved significantly. Recognized faces now report confidence over 70%. The default min_confidence is now set to 0.67

- **New Face Match API**

  The new *Face Match* API allows you to compute the similarity score on two images containing two faces.

- **Speed Modes**

  Speed modes have been introduced to allow you easily tradeoff performance for accuracy.

  There are three speed modes, **"High" , "Medium" and "Low"**

  You can specify your speed mode has exemplified below.

```
sudo docker run -e MODE=High -e VISION-DETECTION=True -v localstorage:/
↪datastore \
-p 80:5000 deepquestai/deepstack

Note the -**e MODE=High** above
```

- **Minimum Confidence**

  The "min_confidence" parameter allows you to control the level of confidence of results for object-detection , facedetection and *Face Recognition*

3.8.2 Breaking Changes

- **TRAFFIC API REMOVED**

  The traffic api has been removed, a more improved version maybe re-introduced in future versions of DeepStack.

- **GENDER API REMOVED**

  The face detection api no longer return gender information, only bounding boxes are now returned. Gender prediction maybe re-introduced in future versions of DeepStack..

3.9 DeepStack Beta - Release Notes

DeepStack Beta is more optimized for storage, memory, compute and is more accurate.

3.9.1 Improvements

- **New Fast GPU Version**

  We have released the GPU Version of DeepStack, accelerating responses by orders of magnitude if you have an Nvidia GPU. See *Using DeepStack with NVIDIA GPU* for setup instructions

- **Improved Face Recognition Engine**

  The face recognition API is now powered by state-of-the-art face recognition engine.

- **Better error handling and crash recovery**

  DeepStack now gracefully handles invalid images and reports errors better.

- **50% Reduction in Install Size**

  The total install size DeepStack is now half the original size.

3.9.2 Breaking Changes

- **ENDPOINT ACTIVATION**

  To avoid unneccesary memory usage. Only endpoints you activate are loaded.

  For example, to use any face API, you should run deepstack as below

```
sudo docker run -e VISION-FACE=True -v localstorage:/datastore \
-p 80:5000 deepquestai/deepstack
```

  In this example, you can query all face related APIs, however, you cannot query the object detection API and other endpoints.

  You can activate multiple endpoints at once as below

```
sudo docker run -e VISION-FACE=True -e VISION-SCENE=True -e VISION-
↪TRAFFIC=True \
-v localstorage:/datastore -p 80:5000 deepquestai/deepstack
```

- **FACE RECOGNITION API-MINIMUM CONFIDENCE**

  The *min_distance* parameter has been replaced by *min_confidence* See *Face Recognition* for usage instructions

- **FACE RECOGNITION API - DATA INCOMPATIBILITY**

  Any faces registered with the Alpha Version is incompatible with this Version. You need to re-register previously registered faces. The new face recognition engine is stable and future releases will remain compatible with this Version.

- **FACE RECOGNITION API - RESPONSES**

  Face registeration response has been changed from

```
{'predictions': {'message': 'face added'}, 'success': True}
```

  To

```
{'message': 'face added', 'success': True}
```

  Face delete response has been changed from

```
{'success': True, 'message': 'user deleted successfully'}
```

To

```
{'success': True}
```

- **SCENE AND TRAFFIC API - RESPONSES**

  Responses has been changed from

```
{'success': True, 'predictions': [{'label': 'conference_room', 'confidence
↪': 73.73981475830078}]}
```

  To

```
{'success': True, 'confidence': 73.73981, 'label': 'conference_room'}
```

- **OBJECT DECTION API - SPEED MODES**

  Speed Modes have been deprecated.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search