Links to MSDN Blog publications:

- https://blogs.msdn.microsoft.com/esmsdn/2018/04/02/post-invitado-analysis-and-object-detection-of-artworks-with-tensorflowgpu-on-windows-10/

- https://blogs.msdn.microsoft.com/esmsdn/2018/04/09/post-invitado-part2-step-by-step-how-to-training-your-own-detector-classifier/

- https://blogs.msdn.microsoft.com/esmsdn/2018/04/24/post-invitado-part-3-step-by-step-how-to-train-an-objects-classifier-understanding-computer-vision-techniques-with-python-and-opencv/

Analysis and Object Detection of Artworks with Tensorflow(GPU) on Windows 10

The following post was written by *Microsoft Student Partner Alexander Gonzalez* as part of final year Project in Computer Science. Thanks to Microsoft Azure, I had the ability to use sophisticated tools which I have never used before and I could not have finished my experiment without azure virtual machine hardware. This article is dedicated to all Spain Technical Community.

Nowadays, all major technology companies are committed to innovative projects using technologies such as Machine Learning, Deep Learning. Both will be the most important technologies for 2018, making a difference on the currently way of living and working. Machine learning and Deep learning are constantly making new business models, jobs, as well as, innovative and efficient industries.

Machine learning is around us more than we think. It can be found in mobile phones, cars, homes and in our own job, helping us to make better decisions, access to higher quality information and faster. According to several surveys and analytical companies, these technologies will be present in all the new software products in 2020.

From the business point of view, artificial intelligence (AI) will be used as a stepping stone to improve and increase the effectiveness and efficiency of the services of any company as well as quality for the user. The trends indicate that most sectors will see radical changes in the way they use all their products, the only risk of such change is to completely ignore it. In the future, products that we currently know will change thanks to the AI . Furthermore, and it is calculated that in the next three years, around 1.2 trillion dollars will change hands thanks to this new field in computing. Consequently, this means that every year the AI is taking a lot of strength and support; therefore, it will leave a mark setting differences between companies in the up-coming years.

The objective of these post series is to show the possibilities that we currently have to perform machine learning and computer vision projects which they will be published in 4 parts:

1. The first part is dedicated to the installation and explanation of the software that we will need to take any project in this case, with TensorFlow, CUDA and CuDNN.

2. The second part will cover step by step the necessary processes to make our dataset, in this case artworks images, followed by training. Finally, the evaluation and obtaining of relevant graphics for the preparation of documentation will be explained. Link to post

Image of Fast-RCNN on surface webcam with python program

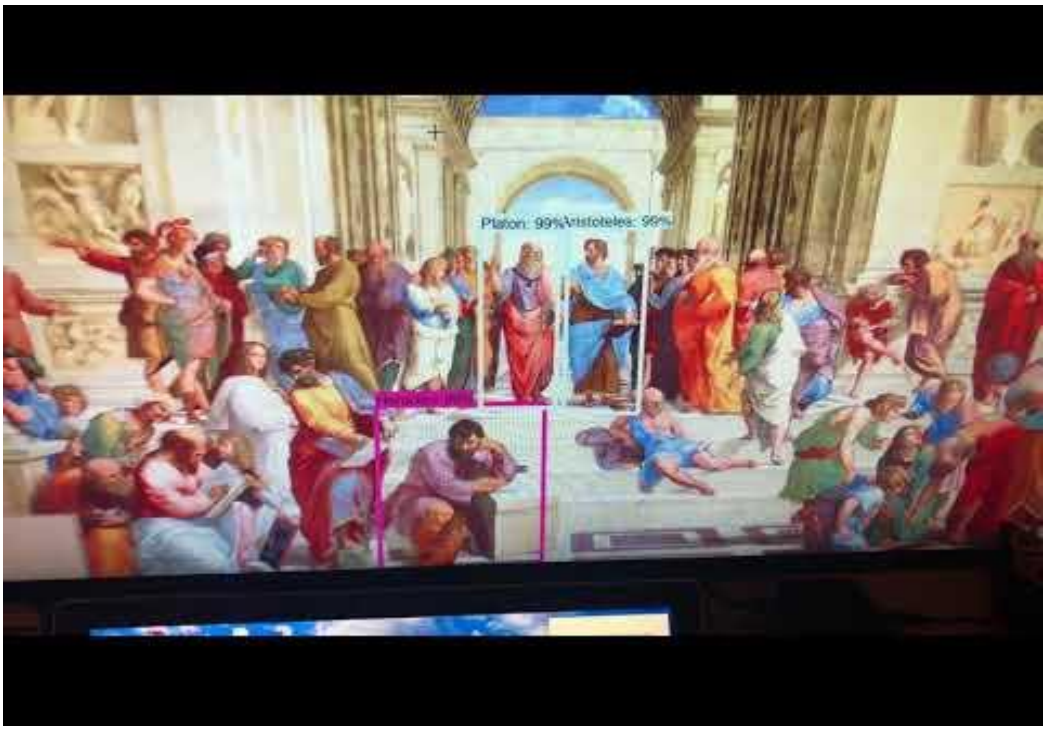


Image of Fast-RCNN on MS Surface webcam with python program

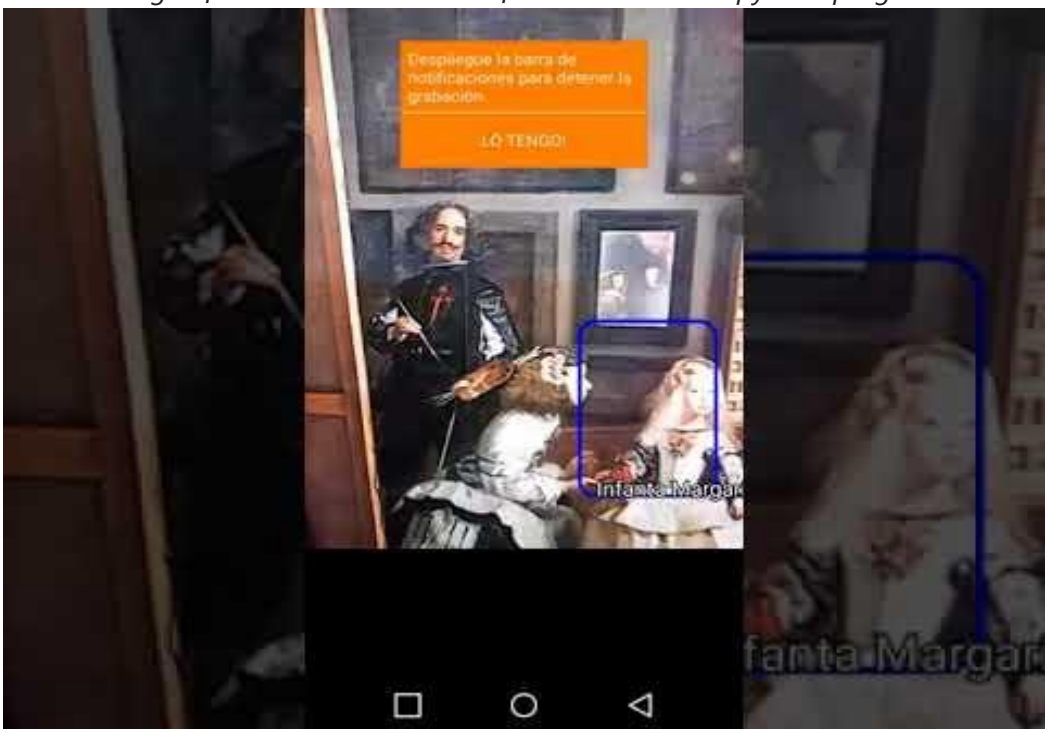


Image of SSD-Mobilenet on LG mobile

3. The third post will explain another way of recognizing and classifying images (20 artworks) using scikit learn and python without having to use models of TensorFlow, CNTK or other technologies which offer models of convolved neural networks. Moreover, we will explain how to set up your own web app with python. For this part a simple API which will collect information about the captured image of our mobile application in Xamarin will be needed, so it will inference with our model made in python, and it

will return the corresponding prediction. With that methodology we can get easy classification without heavy hardware like TensorFlow or CNTK. Link to

post

4. Finally, the last post is dedicated to a comparison between the use of TensorFlow and CNTK, the results obtained, the management and usability, points of interest and final conclusions about the research that has been carried out for the development of all the posts. Link to post

1. How to install Tensorflow 1.5 on Windows 10 with CUDA and CudaDNN

Prerequisites

- Nvidia GPU (GTX 650 or newer. The GTX1050 is a good entry level choice)
- Anaconda with Python 3.6(or 3.5)
- CUDA ToolKit(versión 9)
- CuDNN(7.1.1)

if we want to get results quickly the first thing to think about is with what hardware we will face our computer vision project, since the demands are high in terms of GPU and processing. My advice is to use Data Science Virtual Machine that Azure supports. They are complete virtual machines preconfigured for the modelling, development and implementation of science data. Below, I highlight several documents provided by Azure team that will help us understand the provisioning of these machines and prices:

- https://azure.microsoft.com/es-es/services/virtual-machines/data-science-virtual-machines/
- https://docs.microsoft.com/es-es/azure/machine-learning/data-science-virtual-machine/provision-vm
- https://docs.microsoft.com/es-es/azure/machine-learning/data-science-virtual-machine/overview

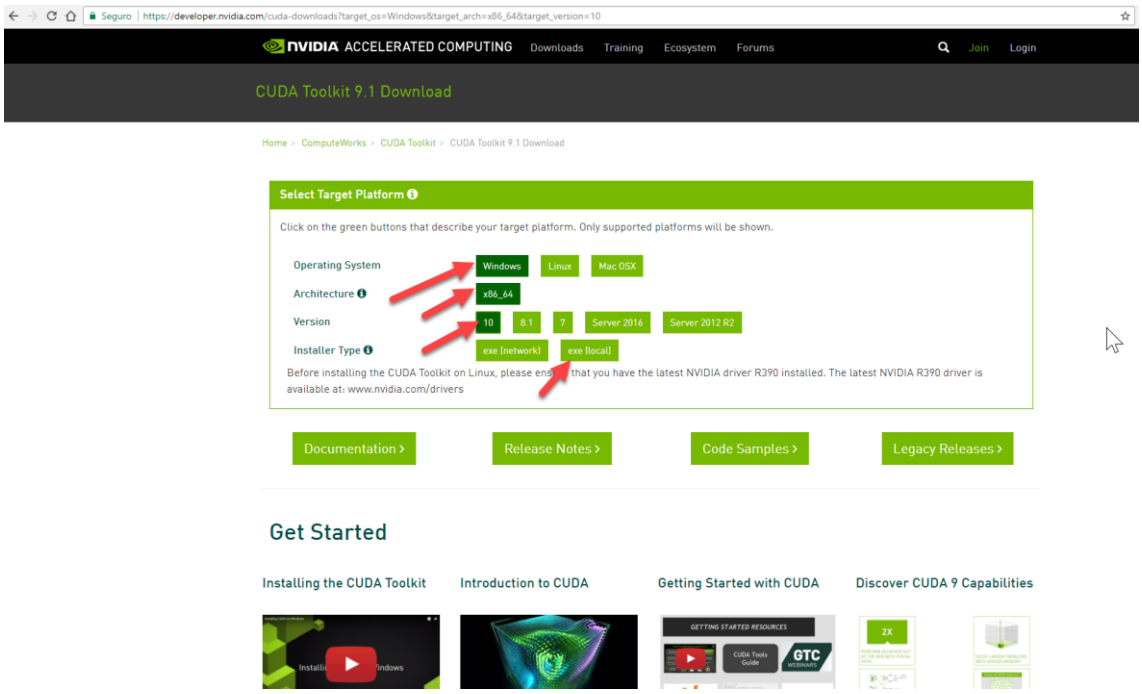
**Note: We must install a Data Science virtual machine of NC6 or NV6 model with GPU.*

Once we have our machine created, we can start to install

Download CUDA 9.0

First of all, we will have to install CUDA Tool Kit:

- Download version 9.0 here: https://developer.nvidia.com/cuda-downloads
- Currently version 9.0 is supported by Tensorflow 1.5



Installer type **exe(network)** is the lighter way one. More complete one is **exe(local)**

Set your environment Variables:

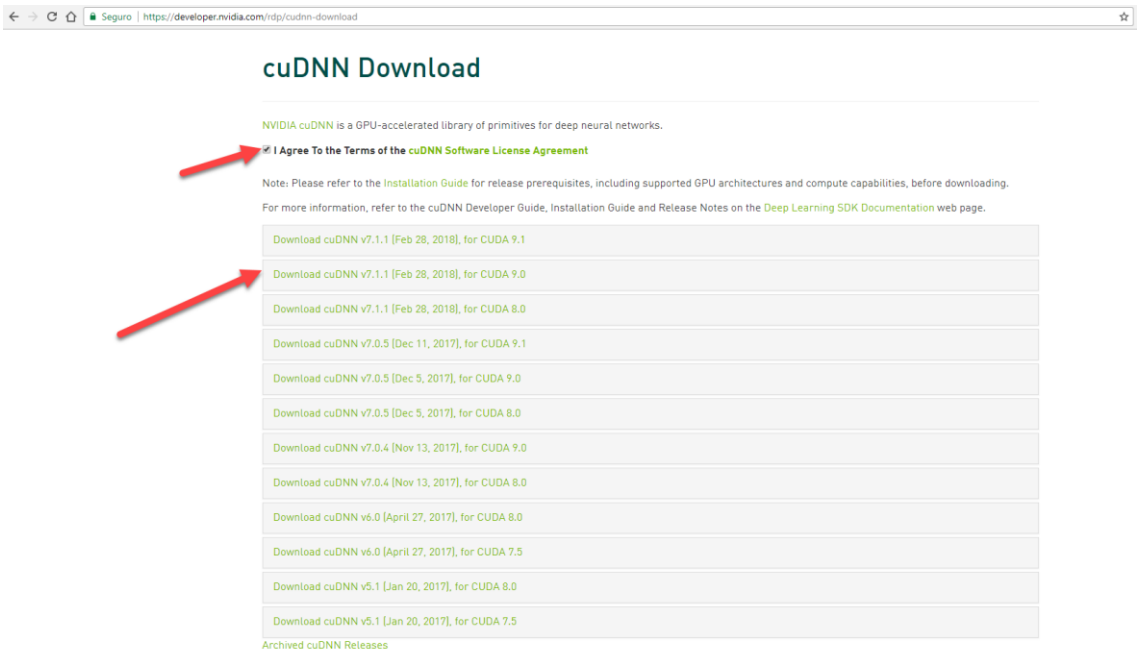
- *Go to start and search "environment variables"*
- *Click th environmen variables button*
- *Add the following paths:*
    - *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\extras\CUPTI\libx64*
    - *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\libnvvp*
    - *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\bin*

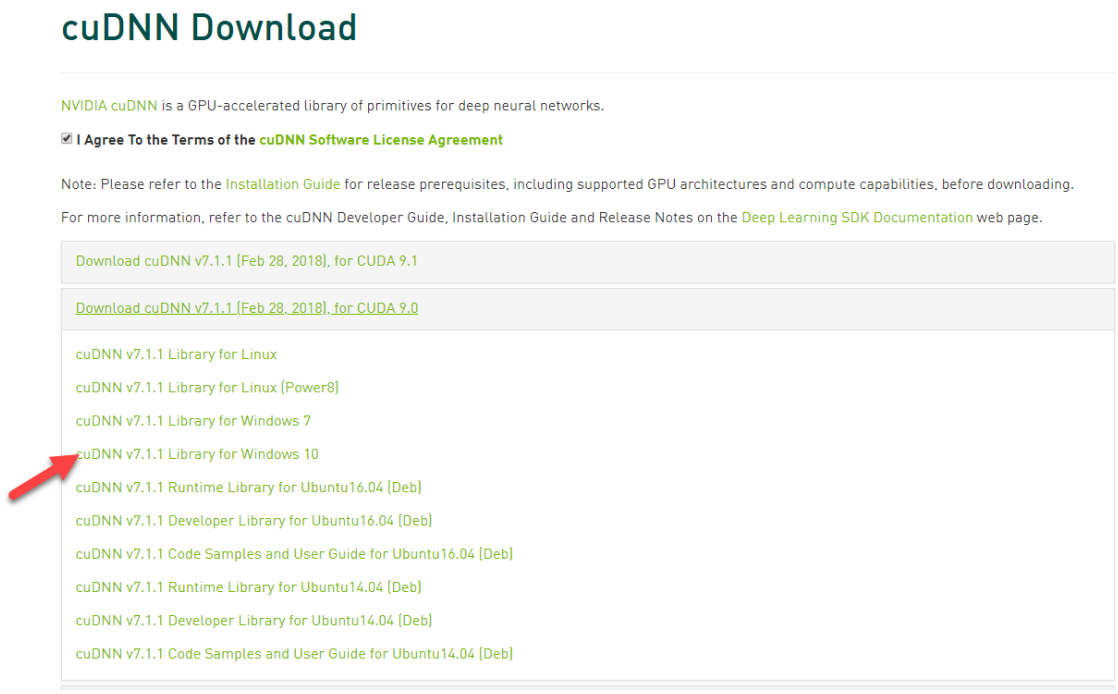
In data Science machine we have CUDA install and we can find this path already install

- **Variable name:** CUDA_PATH_V9_0
- **Variable value:** C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0

Download CUDNN 8.0

- Go to https://developer.nvidia.com/rdp/cudnn-download
- Create a user profile if needed
- Select CUDNN 7.1.1 for CUDA tool kit 9.0

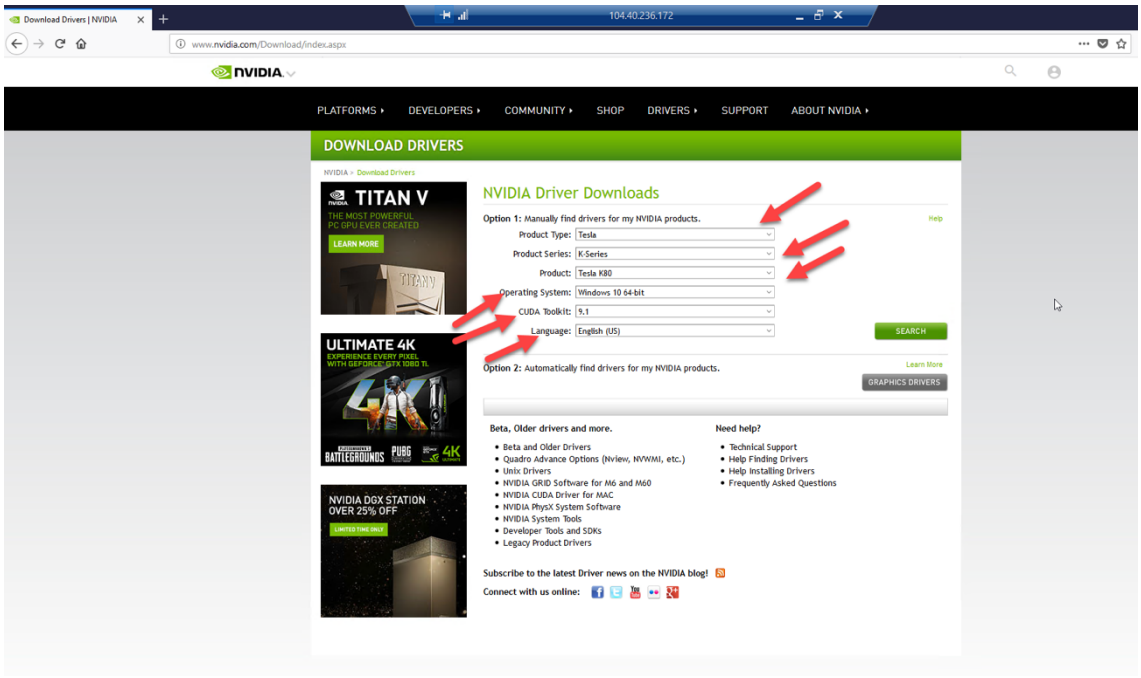




- Extract the zip file and place them somewhere (**C:// directory for example**)
- Add a in environment variable path to the bin folder: For example: **C:/cuda/bin**

Update GPU Driver

- Go to http://www.nvidia.com/Download/index.aspx
- Select your version to download
- Install the drive



Install Anaconda 3.6

In these VM Anaconda environment has been installed. Howver, the installation steps of Anaconda were not explained in this post.

Install Tensorflow GPU on Windows 10

Open your cmd window and type the following command:

```
pip install –ignore-installed –upgrade tensorflow-gpu
```

Notes Tensorflow documentation:*

https://www.tensorflow.org/install/

Test your tensorflow install:

1. Open anaconda prompt
2. Write "**python –-version"**
3. Write "**python**"
4. Once the interpreter opens type the following:
   - `import tensorflow as tf`
   - `hello= tf.constant('Hello, Tensorflow!')`
   - `Sess = tf.session()`
   - `print(sess.run(hello))`

In the next post, we will cover step by step the necessary processes to make our dataset. In this case, artwork images, followed by training and finally the evaluation and obtaining relevant graphics for document preparation. Index of the next post:

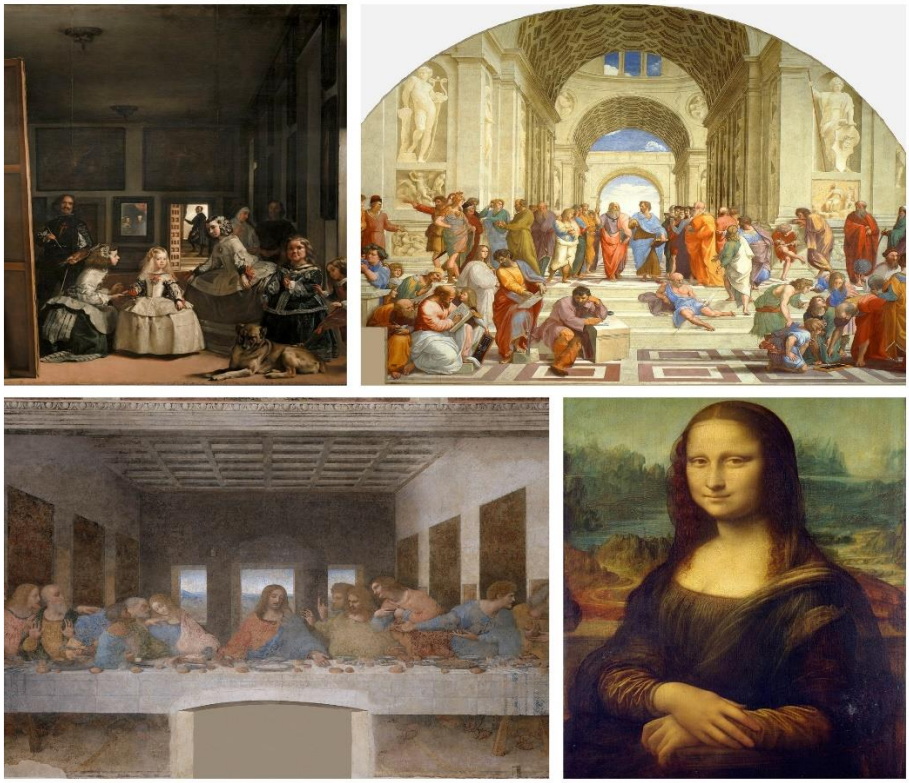
1. *Labelling pictures*
2. *Generating training data*

2. How to train an object detector classifier for multiple objects using tensorflow(GPU) on Windows 10

Once we have installed Tensorflow, Cuda and CuDNN, we can pass to the next level! The purposes of this post is to explain how to train your own convolutional neural network object detection classifier for multiple objects. In this post we do not develop a nueronal network of zero, we will only take this github:

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

It provides a collection of detection models pre-trained datasets of differents kind of objects. They are also useful for initializing your models when training on novel datasets like our dataset of artworks. At the end of this post, you will have the idea to develope your own model that can identify and program that can draw boxes around specific ítem in that case 12 items objects of different artworks, like *Gioconda, Athens School, Las Meninas, The Last Supper*.



There are several good tutorials available for how to use TensorFlow's Object Detection API to train a classifier for a single object, like this one:

https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0

The post is written for Windows 10. But without forgetting Linux, only emphasize that the development process is very similar and can also be used for Linux operating systems, but the file paths and package installation commands should be modified accordingly. Only tell you before start that TensorFlow-GPU allows your PC or VM to use the video card to provide extra processing power while training, so it will be used for this post, Regular Tensorflow not. In my experience doing this experiment, using TensorFlow-GPU instead of regular TensorFlow reduces training time by a factor of about 8hr using Fast-RCNN model, 3 hours to train instead of 8 hours with SSD-Mobilenet model for 3 objects, 21 hours for 12 objects. You can search differents models to tensorflow here:

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Regular TensorFlow can also be used for this post, I have tried it in my PC, but it will take longer. If you use regular TensorFlow, you do not need to install CUDA and cuDNN like in teh first post **[Link to post].**

1. Installing TensorFlow-GPU

Install Tensorflow-GPU following the instructions of the first post. Dowload CUDA 9.0 and CuDNN 7.1.1, but you will likely need to continue updating the CUDA and CuDNN versions to the latest supported version.

2. Setting up the Object Detection directory structure and Anaconda Virtual Environment

The TensorFlow Object Detection API requires using the specific directory structure provided in its GitHub repository. It also requires several additional Python packages, specific additions to the PATH and PYTHONPATH variables, and a few extra setup commands to get everything set up to run or train an object detection model. Stay tuned at each step because the processes are very meticulous, and the minimum failure can give us error.

Download TensorFlow Object Detection API repository from GitHub

Create a folder directly in C: and name it "tensorflow" or whatever you want. This working directory will contain the full TensorFlow object detection framework, as well as your training images, training data, trained classifier, configuration files, and everything else needed for the object detection classifier.

Download the full TensorFlow object detection repository located at https://github.com/tensorflow/models by clicking the "Clone or Download" button and downloading the zip file. Open the downloaded zip file and extract the "models-master" folder directly into your C:\DIRECTORY you just created. Rename "models-master" to just "models".

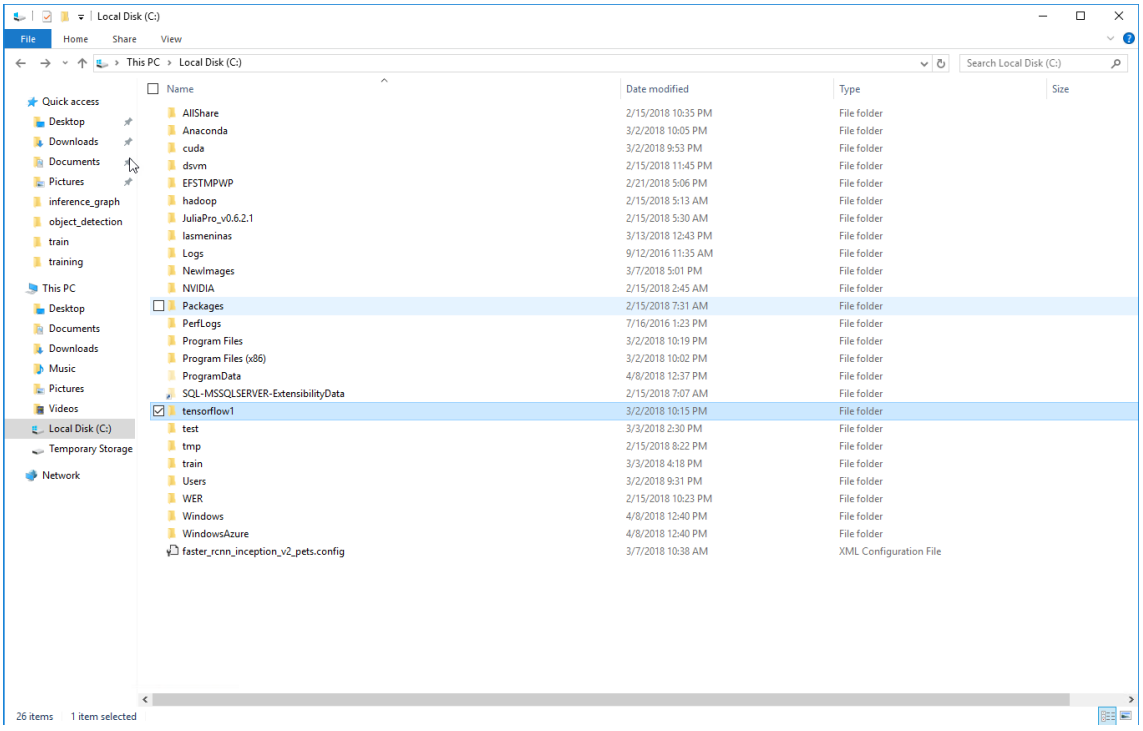


Image of the principal directory

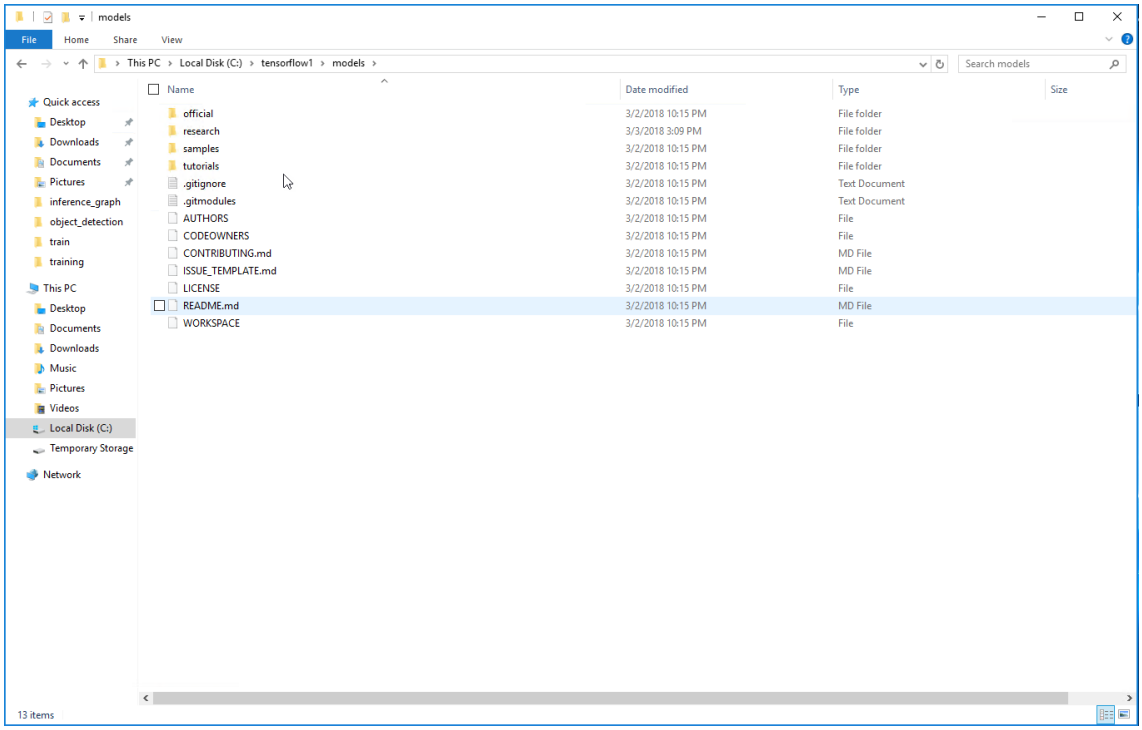


Image of Tensorflow Object Detection API directory

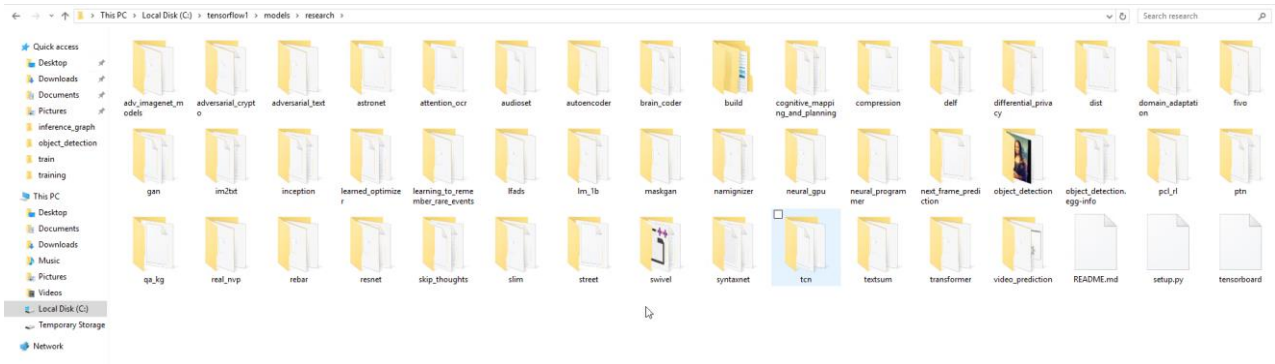


Image of Tensorflow Object Detection API, Research directory

Download the Faster-RCNN and SSD-Mobilenet models

TensorFlow provides several object detection models (pre-trained classifiers with specific neural network architectures) in its model zoo. If we look at the README of this github we can see that they tell us the importance within each model of three files.

You can un-tar each tar.gz file via, e.g.,:

```
tar -xzvf ssd_mobilenet_v1_coco.tar.gz
```

Inside the un-tar'ed directory, you will find:

- a graph proto (`graph.pbtxt`)
- a checkpoint (`model.ckpt.data-00000-of-00001` , `model.ckpt.index` , `model.ckpt.meta`)
- a frozen graph proto with weights baked into the graph as constants (`frozen_inference_graph.pb`) to be used for out of the box inference (try this out in the Jupyter notebook!)
- a config file (`pipeline.config`) which was used to generate the graph. These directly correspond to a config file in the samples/configs) directory but often with a modified score threshold. In the case of the heavier Faster R-CNN models, we also provide a version of the model that uses a highly reduced number of proposals for speed.

In the following sections we will explain what we should do with them, in the case of this project we have used *ssd_mobilenet_v1_coco* and *faster_rcnn_inception_v2_coco*. But before I would like to explain the importance of understanding the following table of models proposed by tensorflow. Especially understand well the speed, and the mAP column.

COCO-trained models {#coco-models}

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

**Note: What is column mAP? https://stackoverflow.com/questions/46094282/why-we-use-map-score-for-evaluate-object-detectors-in-deep-learning*

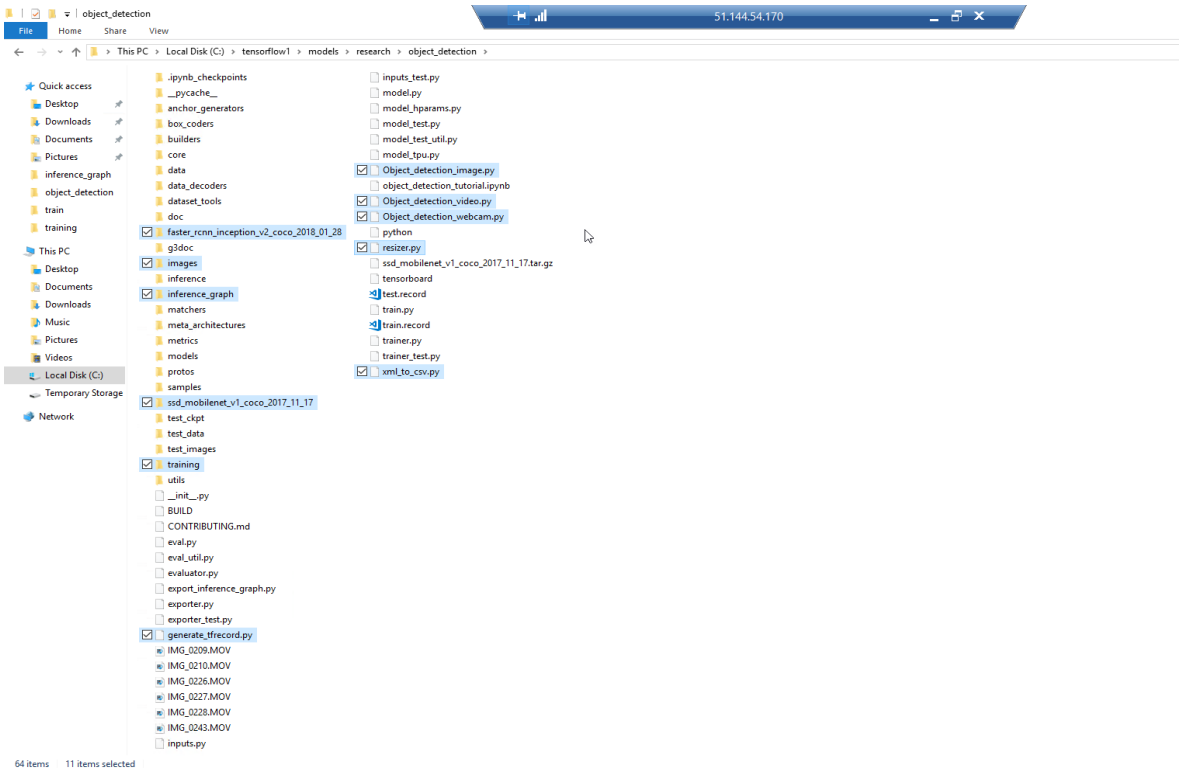
As we can see in the table SSD-Mobilenet has much more speed than the other models and smaller mAP, we could conclude that it is the best but once my project has been realized I have noticed that for example in this case with art paintings, it has given good results in terms of detection but in terms of accuracy it give me low level of success. On the other hand, faster rcnn has surprised me because according to the table it has a very slow detection speed with respect to SSD-Mobilenet but it has gone very well on my MS Surface webcam. At the end to have been analyzing both models decided to train with both and extract their inference graph to use it both on tablets, laptops and mobile or Iot devices. You can choose which model to train your objection detection classifier on. If you are planning on using the object detector on a device with low computational like mobile, use the SDD-MobileNet model. If you will be running your object detector on a laptop or desktop PC, use one of the RCNN models.

In this post will use the Faster-RCNN-Inception-V2 model and ssd_mobilenet_v1_coco. Download the models and open the downloaded faster_rcnn_inception_v2_coco_2018_01_28.tar.gz and ssd_mobilenet_v1_coco_2017_11_17.tar.gz file with a file archiver such as WinZip or 7-Zip and extract the faster_rcnn_inception_v2_coco_2018_01_28 and ssd_mobilenet_v1_coco_2017_11_17 folder to the *C:\tensorflow1\models\research\object_detection* folder.

**Note: The models date and versions will likely change in the future, but it should still work with this tutorial.*

Download this tutorial's repository from GitHub

Download the full repository located on this page, scroll to the top and click Clone or Download and extract all the contents directly into the C:\tensorflow1\models\research\object_detection directory. This establishes a specific directory structure that will be used for the rest of the post. At this point, your \object_detection folder should look like:



Final directory with Object Detection API with 11 extra-files(selected)

This repository contains the images, annotation data, .csv files, and TFRecords needed to train a "Artworks" objects detector. It also contains Python scripts that are used to generate the training data. It has scripts to test out the object detection classifier on images, videos, or a webcam feed.

If you want to practice training your own "Artwork" Detector, you can leave all the files as they are. You can follow along with this tutorial to see how each of the files were generated, and then run the training. You will still need to generate the TFRecord files train.record and test.record as described in nexts steps.

You can also use the frozen inference graph for my trained Artwork model detector from the I and extract the contents to \object_detection\inference_graph. This inference graph will work "out of the box". You can test it after all the setup instructions have been completed by running the Object_detection_image.py or video or webcam script.

If you want to train your own object detector, delete the following files (do not delete the folders):

- All files in \object_detection\images\train and \object_detection\images\test
- The “test_labels.csv” and “train_labels.csv” files in \object_detection\images
- All files in \object_detection\training
- All files in \object_detection\inference_graph

Now, you are ready to start from scratch in training your own object detector. This post will assume that all the files listed above were deleted and will go on to explain how to generate the files for your own training dataset.

Set up new Anaconda virtual environment

Next, we'll work on setting up a virtual environment in Anaconda for tensorflow-gpu. From the Start menu in Windows, search for the Anaconda Prompt utility, right click on it, and click “Run as Administrator”.

Activate the environment of tensorflow by issuing:

```
C:\> activate tensorflow
```

Install the other necessary packages by issuing the following commands:

```
(tensorflow) C:\> conda install -c anaconda protobuf
(tensorflow) C:\> pip install pillow
(tensorflow) C:\> pip install lxml
(tensorflow) C:\> pip install jupyter
(tensorflow) C:\> pip install matplotlib
(tensorflow) C:\> pip install pandas
(tensorflow) C:\> pip install opencv-python
```

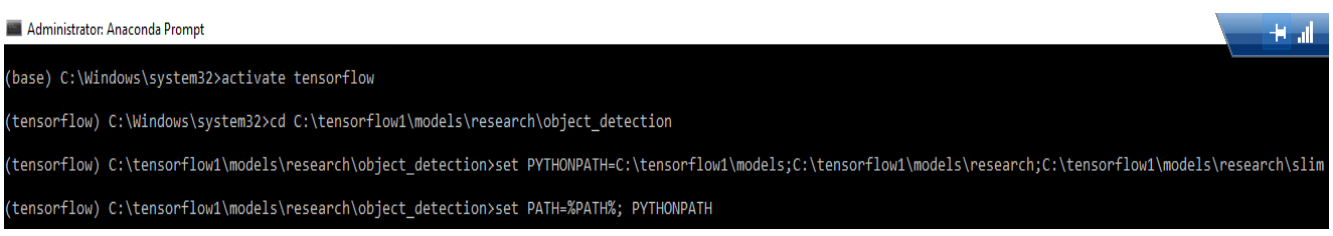
**Note: The ‘pandas’ and ‘opencv-python’ packages are not needed by TensorFlow, but they are used in the Python scripts to generate TFRecords and to work with images, videos, and webcam feeds.*

Configure PATH and PYTHONPATH environment variables

The PATH variable must be configured to add the \models, \models\research, and \models\research\slim directories. bor some reason, you need to create a PYTHONPATH variable with these directories, and then add PYTHONPATH to the PATH variable. Otherwise, it will not work. Do this by issuing the following commands (from any directory):

```
(tensorflow) C:\> set PYTHONPATH=C:\tensorflow1\models;
C:\tensorflow1\models\research;C:\tensorflow1\models\research\slim
(tensorflow1) C:\> set PATH=%PATH%; PYTHONPATH
```

**Note: Every time the "tensorflow1" virtual environment is exited, the PATH and PYTHONPATH variables are reset and need to be set up again.*



Compile Protobufs and run setup.py

Now we need to compile the Protobuf files, which are used by TensorFlow to configure model and training parameters. Unfortunately, the short protoc compilation command posted on TensorFlow's Object Detection API installation page does not work on Windows. Every .proto file in the \object_detection\protos directory must be called out individually by the command.

To understand what the protobuf files are (*Protocol Buffer, Mechanism for serializing structured data by Google*):

- https://www.tensorflow.org/extend/tool_developers/
- https://developers.google.com/protocol-buffers/?hl=en

In the Anaconda Command Prompt, change directories to the \models\research directory and copy and paste the following command into the command line and press Enter:

```
protoc --python_out=. .\object_detection\protos\anchor_generator.proto
.\object_detection\protos\argmax_matcher.proto
.\object_detection\protos\bipartite_matcher.proto .\object_detection\protos\box_coder.proto
.\object_detection\protos\box_predictor.proto .\object_detection\protos\eval.proto
.\object_detection\protos\faster_rcnn.proto
.\object_detection\protos\faster_rcnn_box_coder.proto
.\object_detection\protos\grid_anchor_generator.proto
.\object_detection\protos\hyperparams.proto .\object_detection\protos\image_resizer.proto
.\object_detection\protos\input_reader.proto .\object_detection\protos\losses.proto
.\object_detection\protos\matcher.proto
.\object_detection\protos\mean_stddev_box_coder.proto .\object_detection\protos\model.proto
.\object_detection\protos\optimizer.proto .\object_detection\protos\pipeline.proto
.\object_detection\protos\post_processing.proto .\object_detection\protos\preprocessor.proto
.\object_detection\protos\region_similarity_calculator.proto
.\object_detection\protos\square_box_coder.proto .\object_detection\protos\ssd.proto
.\object_detection\protos\ssd_anchor_generator.proto
.\object_detection\protos\string_int_label_map.proto .\object_detection\protos\train.proto
.\object_detection\protos\keypoint_box_coder.proto
.\object_detection\protos\multiscale_anchor_generator.proto
```

This creates a name_pb2.py file from every name.proto file in the \object_detection\protos folder.

**Note: TensorFlow occassionally adds new .proto files to the \protos folder. You may need to update the protoc command to include the new .proto files.*

Finally, run the following commands from the C:\tensorflow1\models\research directory:

```
(tensorflow) C:\tensorflow1\models\research> python setup.py build
(tensorflow) C:\tensorflow1\models\research> python setup.py install
```

3. Labelling pictures

Now that the TensorFlow Object Detection API is all set up and ready to go, we need to provide the images it will use to train a new detection classifier.

Gather Pictures

TensorFlow needs hundreds of images of an object to train a good detection classifier. To train a robust classifier, the training images should have random objects in the image along with the desired objects and should have a variety of backgrounds and lighting conditions. There should be some images where the desired object is partially obscured, overlapped with something else, or only halfway in the picture.

For my Artwork Detection classifier, I have twelve different objects I want to detect (Diego Velazquez, Maria Agustina, Infanta Margarita, Isaben de Velasco and Mari barbola part of “Las Meninas”, Platon, aristoteles and heraclito of “School of Athens”, Gioconda of “Mona Lisa” and finally Jesus, Judas and Mateo of “The Last Supper”).

I used Google Image Search to find about 80 pictures of each artwork. You can use your phone or download images of the objects from Google Image Search. I recommend having at least 200 pictures overall. I used 310 aprox, pictures to train my artworks detector. In addition to a better training has been cut the complete image of the art box focusing on the object we want to detect.

Images of the different artworks cropped

In the case of art paintings, we are lucky that they are static images that will always be presented to our model in the same way, they are not objects with movement or relevant changes of appearance such as an animal, cars, etc. With the images of the art pictures I played when choosing the images for the dataset with the main properties of computer vision, luminosity, size cause. They always appear in the same way, we can only extract real photos with people in front of the picture as another training case.

You can use the resizer.py script in this repository to reduce the size of the images because they should be less than 200KB each, and their resolution shouldn't be more than 720x1280. The larger the images are, the longer it will take to train the classifier.

After you have all the pictures you need, move 20% of them to the \object_detection\images\test directory, and 80% of them to the \object_detection\images\train directory. Make sure there are a variety of pictures in both the \test and \train directories.

Label Pictures

With all the pictures gathered, it's time to label the desired objects in every picture. Labellimg is a great and new tool for me for labeling images, and its GitHub page has very clear instructions on how to install and use it.

[LabelImg GitHub link]

[LabelImg download link]

Download and install LabelImg, point it to your \images\train directory, and then draw a box around each object in each image. Repeat the process for all the images in the \images\test directory. This will take a while and a lot of patience…

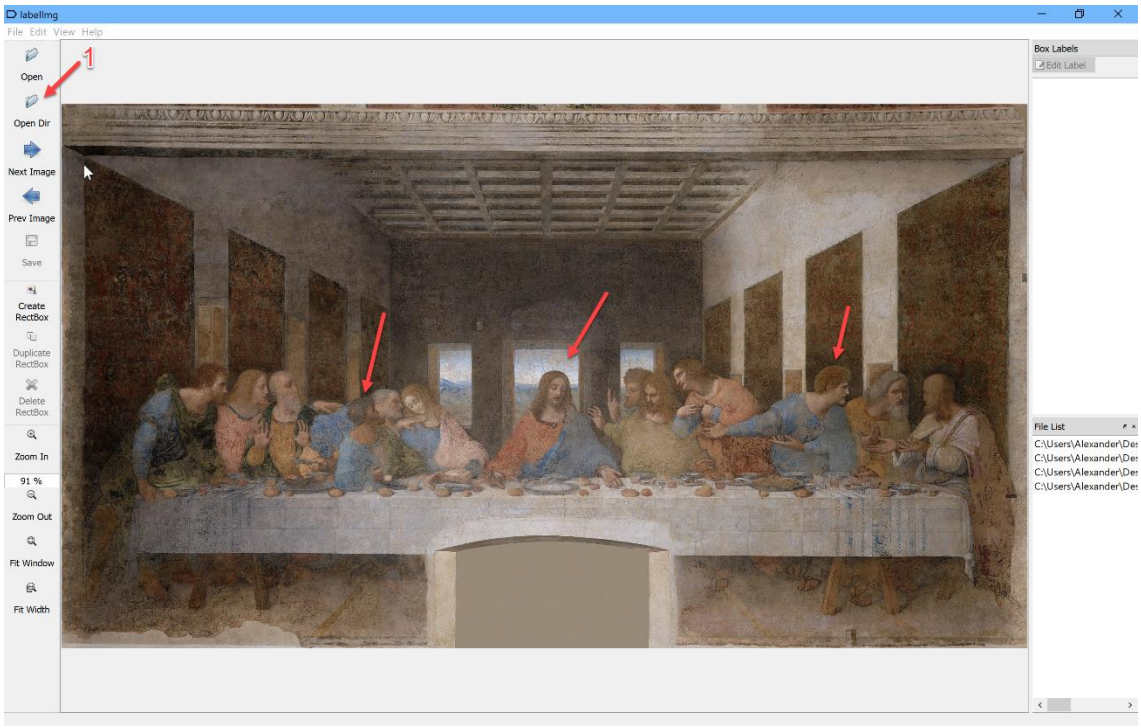


Image open file directory and observe what objects we want

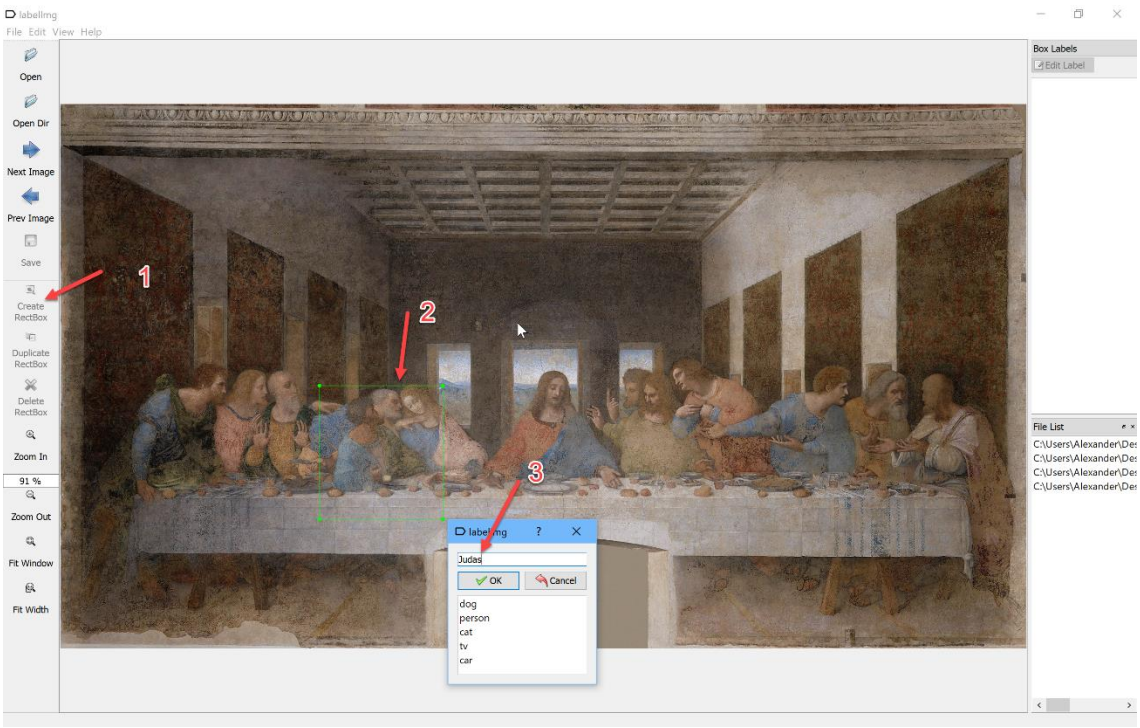


Image about create RectBox around objects

4. Generating training data

LabelImg saves a .xml file containing the label data for each image. These .xml files will be used to generate TFRecords, which are one of the inputs to the TensorFlow trainer. Once you have labeled and saved each image, there will be one .xml file for each image in the \test and \train directories.

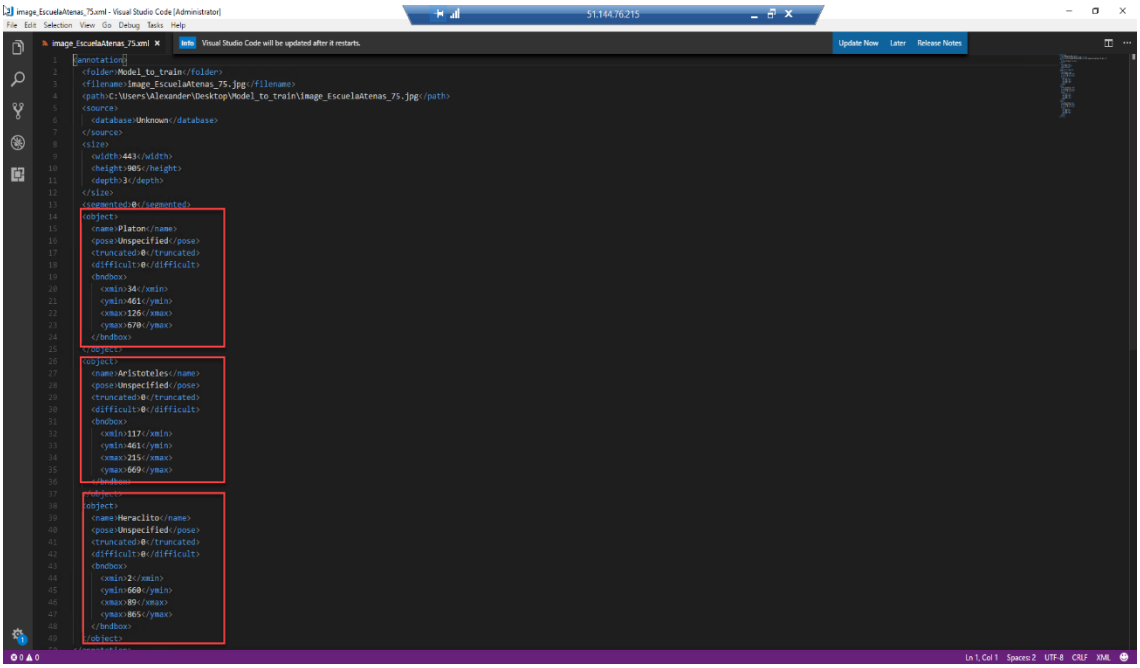


Image of XML file generate by LabelImg with the objects with their respectives coordinates.

With the images labeled, it's time to generate the TFRecords that serve as input data to the TensorFlow training model. This tutorial uses the xml_to_csv.py and generate_tfrecord.py scripts from [Dat Tran's Raccoon Detector dataset], with some slight modifications to work with our directory structure.

First, the image .xml data will be used to create .csv files containing all the data for the train and test images. From the \object_detection folder, issue the following command in the Anaconda command prompt:

```
(tensorflow) C:\tensorflow1\models\research\object_detection> python xml_to_csv.py
```

This creates a train_labels.csv and test_labels.csv file in the \object_detection\images folder.

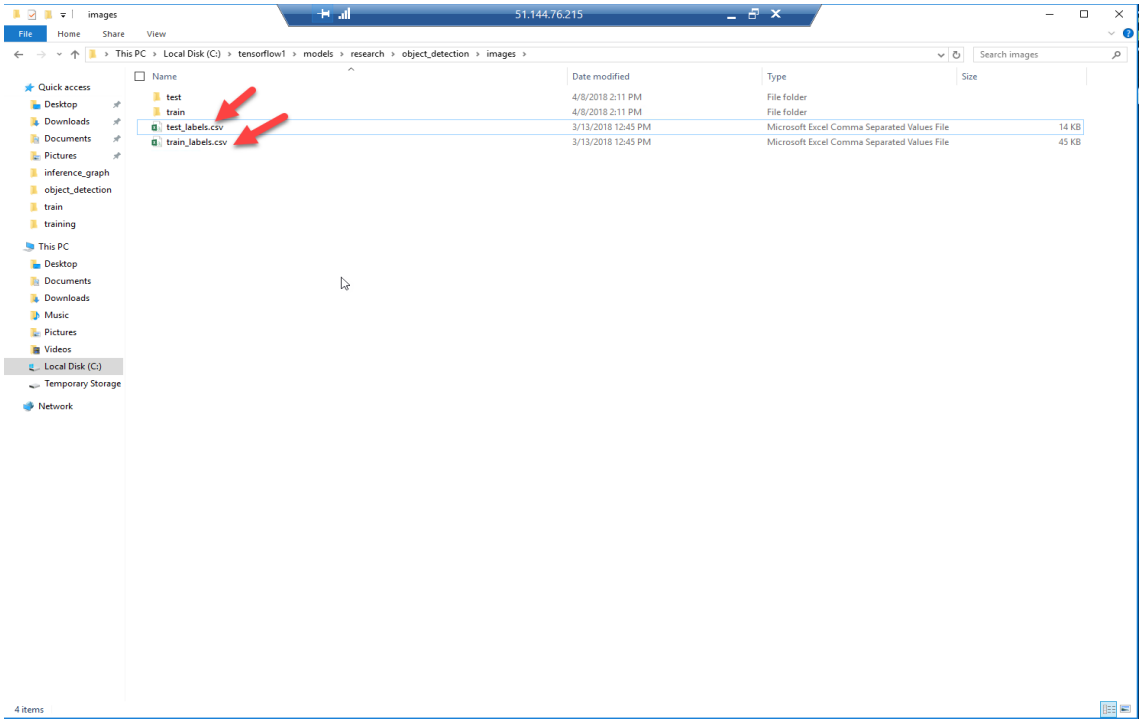
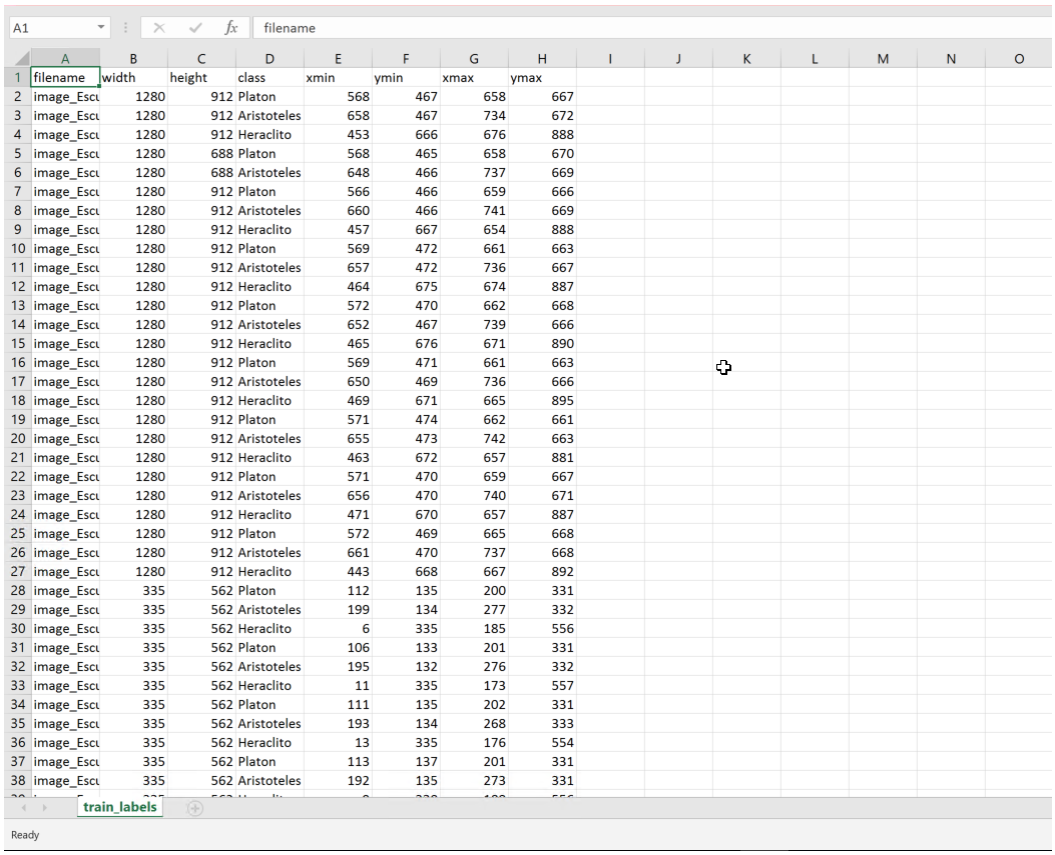


Image of train and test csv files from xml file of LabelImg



	A	B	C	D	E	F	G	H
1	filename	width	height	class	xmin	ymin	xmax	ymax
2	image_Escu	1280	912	Platon	568	467	658	667
3	image_Escu	1280	912	Aristoteles	658	467	734	672
4	image_Escu	1280	912	Heraclito	453	666	676	888
5	image_Escu	1280	688	Platon	568	465	658	670
6	image_Escu	1280	688	Aristoteles	648	466	737	669
7	image_Escu	1280	912	Platon	566	466	659	666
8	image_Escu	1280	912	Aristoteles	660	466	741	669
9	image_Escu	1280	912	Heraclito	457	667	654	888
10	image_Escu	1280	912	Platon	569	472	661	663
11	image_Escu	1280	912	Aristoteles	657	472	736	667
12	image_Escu	1280	912	Heraclito	464	675	674	887
13	image_Escu	1280	912	Platon	572	470	662	668
14	image_Escu	1280	912	Aristoteles	652	467	739	666
15	image_Escu	1280	912	Heraclito	465	676	671	890
16	image_Escu	1280	912	Platon	569	471	661	663
17	image_Escu	1280	912	Aristoteles	650	469	736	666
18	image_Escu	1280	912	Heraclito	469	671	665	895
19	image_Escu	1280	912	Platon	571	474	662	661
20	image_Escu	1280	912	Aristoteles	655	473	742	663
21	image_Escu	1280	912	Heraclito	463	672	657	881
22	image_Escu	1280	912	Platon	571	470	659	667
23	image_Escu	1280	912	Aristoteles	656	470	740	671
24	image_Escu	1280	912	Heraclito	471	670	657	887
25	image_Escu	1280	912	Platon	572	469	665	668
26	image_Escu	1280	912	Aristoteles	661	470	737	668
27	image_Escu	1280	912	Heraclito	443	668	667	892
28	image_Escu	335	562	Platon	112	135	200	331
29	image_Escu	335	562	Aristoteles	199	134	277	332
30	image_Escu	335	562	Heraclito	6	335	185	556
31	image_Escu	335	562	Platon	106	133	201	331
32	image_Escu	335	562	Aristoteles	195	132	276	332
33	image_Escu	335	562	Heraclito	11	335	173	557
34	image_Escu	335	562	Platon	111	135	202	331
35	image_Escu	335	562	Aristoteles	193	134	268	333
36	image_Escu	335	562	Heraclito	13	335	176	554
37	image_Escu	335	562	Platon	113	137	201	331
38	image_Escu	335	562	Aristoteles	192	135	273	331

Image inside train and test .csv file

Next, open the generate_tfrecord.py file in a text editor. Replace the label map starting at line 31 with your own label map, where each object is assigned an ID number. This same number assignment will be used when configuring the labelmap.pbtxt file.

You will replace the following code in generate_record.py:

```
# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'Diego Velazquez':
        return 1
    elif row_label == 'Maria Agustina':
        return 2
    elif row_label == 'Infanta Margarita':
        return 3
    elif row_label == 'Isabel de Velasco':
        return 4
    elif row_label == 'Mari Barbola':
        return 5
    elif row_label == 'Platon':
        return 6
    elif row_label == 'Aristoteles':
        return 7
    elif row_label == 'Heraclito':
        return 8
    elif row_label == 'Gioconda':
```

```
        return 9
    elif row_label == 'Jesucristo':
        return 10
    elif row_label == 'Judas':
        return 11
    elif row_label == 'Mateo':
        return 12
    else:
        None
```

Then, generate the TFRecord files by issuing these commands from the \object_detection folder:

```
python generate_tfrecord.py --csv_input=images\train_labels.csv --image_dir=images\train --output_path=train.record
python generate_tfrecord.py --csv_input=images\test_labels.csv --image_dir=images\test --output_path=test.record
```

These generate a train.record and a test.record file in \object_detection. These will be used to train the new object detection classifier.

5. Creating a label map and configuring training

The last thing to do before training is to create a label map and edit the training configuration file.

Label map

The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers. Use a text editor to create a new file and save it as labelmap.pbtxt in the C:\tensorflow1\models\research\object_detection\training folder. *Don't forget that the file type is .pbtxt, not .txt*. In the text editor, copy or type in the label map in the format below. The example below is the label map for my Artworks Detector:

```
item {
  id: 1
  name: 'Diego Velazquez'
}

item {
  id: 2
  name: 'Maria Agustina'
}

item {
  id: 3
  name: 'Infanta Margarita'
}

item {
```

```
  id: 4
  name: 'Isabel de Velasco'
}

item {
  id: 5
  name: 'Mari Barbola'
}

item {
  id: 6
  name: 'Platon'
}

item {
  id: 7
  name: 'Aristoteles'
}

item {
  id: 8
  name: 'Heraclito'
}

item {
  id: 9
  name: 'Gioconda'
}


item {
  id: 10
  name: 'Jesucristo'
}

item {
  id: 11
  name: 'Judas'
}

item {
  id: 12
  name: 'Mateo'
}
```

Configure training

Finally, the object detection training pipeline must be configured. It defines which model and what parameters will be used for training. This is the last step before running training 😊

In this step we can do the same with SSD-Mobilenet and Faster-RCNN. First, we will do the training with Faster-RCNN and then I will explain how it would be done with the Mobilenet model.

Navigate to C:\tensorflow1\models\research\object_detection\samples\configs and copy the faster_rcnn_inception_v2_pets.config file into the \object_detection\training directory. Then, open the file with a text editor. There are several changes to make to the .config file, mainly changing the number of classes and examples, and adding the file paths to the training data.

Make the following changes to the faster_rcnn_inception_v2_pets.config file. Note: The paths must be entered with single forward slashes (NOT backslashes), or TensorFlow will give a file path error when trying to train the model. Also, the paths must be in double quotation marks (“), not single quotation marks (‘).

- Line 9. Change num_classes to the number of different objects you want the classifier to detect. For the above basketball, shirt, and shoe detector, it would be num_classes : 3 .

- Line 110. Change fine_tune_checkpoint to:

    - fine_tune_checkpoint : "C:/tensorflow1/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"

- Lines 126 and 128. In the train_input_reader section, change input_path and label_map_path to:

    - input_path : "C:/tensorflow1/models/research/object_detection/train.record"
    - label_map_path: "C:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"

- Line 132. Change num_examples to the number of images you have in the \images\test directory.

- Lines 140 and 142. In the eval_input_reader section, change input_path and label_map_path to:

    - input_path : "C:/tensorflow1/models/research/object_detection/test.record"
    - label_map_path: "C:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"

Save the file after the changes have been made. Great! Now we can start the training job!

For Mobilenet of other model we must do the same process, get the config file of each model and change the parameters that I explained before. The model of SSD-Mobilenet you can find it here like Faster-RCNN:

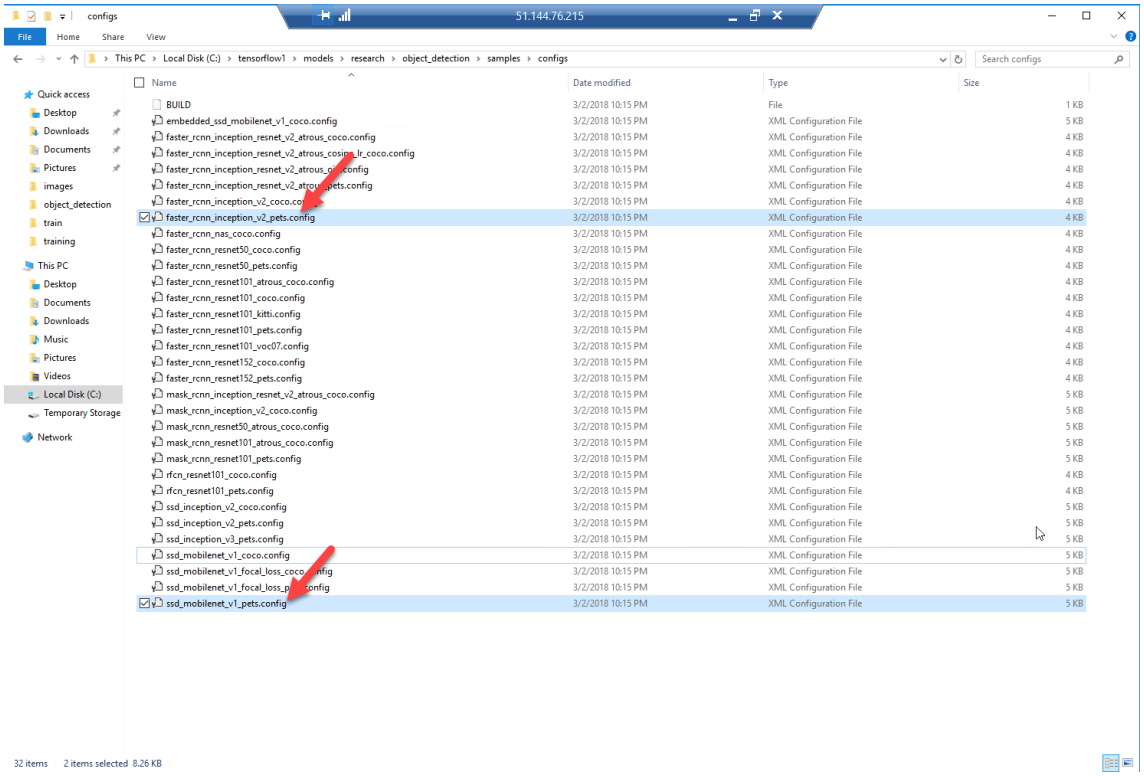


Image directory of Tensorflow pre-trained models (Coco or Pets datasets)

6. Training

From the \object_detection directory, issue the following command to begin training:

```
python train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/faster_rcnn_inception_v2_pets.config
```

If we want to use SSD-Mobilenet to train your model, you only need to change the config_path:

```
python train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/ssd_mobilenet_v1_pets.config
```

If everything has been set up correctly, TensorFlow will initialize the training. When training begins, it will look like this:

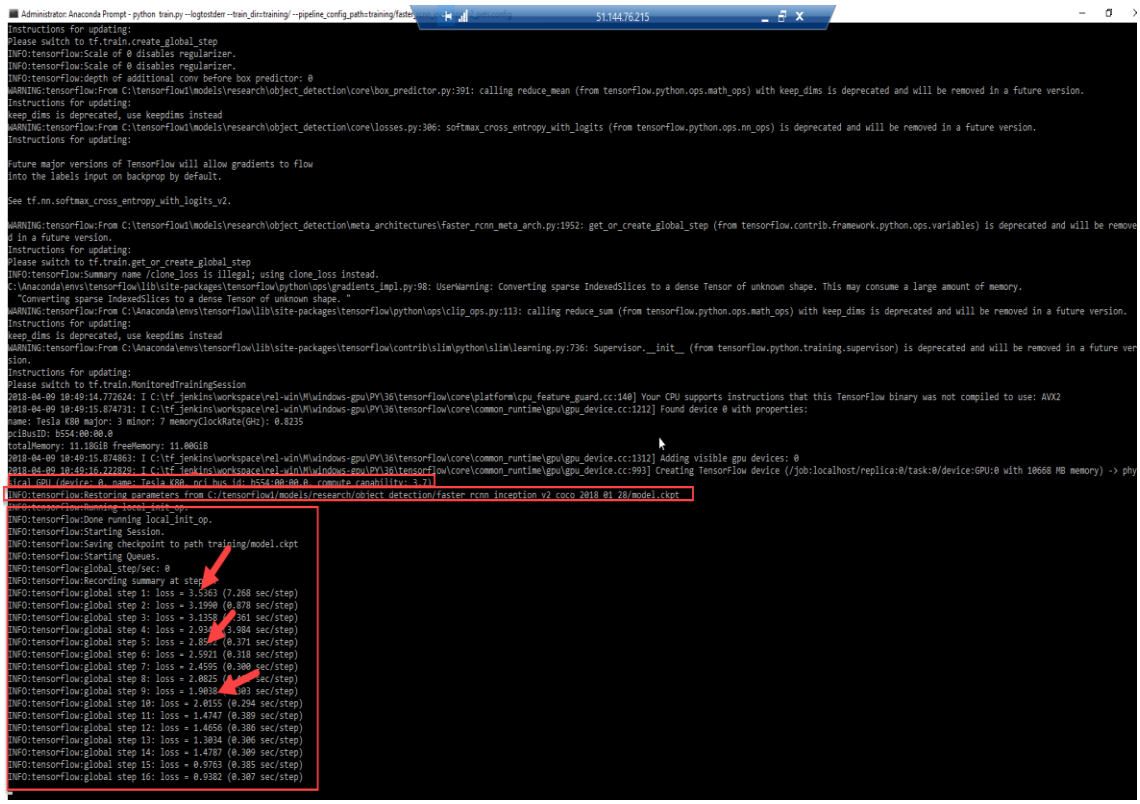


Image training with Faster RCNN.

Each step of training reports the loss. It will start set up our graphic card (Tesla K80) then get the parameters of our model.config. This step starts with high los and get lower and lower as training progresses. For my training on the Faster-RCNN-Inception-V2 model, it started at about 3.0 and quickly dropped below 0.8. I recommend allowing your model to train until the loss consistently drops below 0.05, which will take about 40,000 steps, or about 2-3 hours depending on how powerful your CPU and GPU are.

**Note: The loss numbers will be different if a different model is used. MobileNet-SSD starts with a loss of about 40 and should be trained until the loss is consistently under 2.*

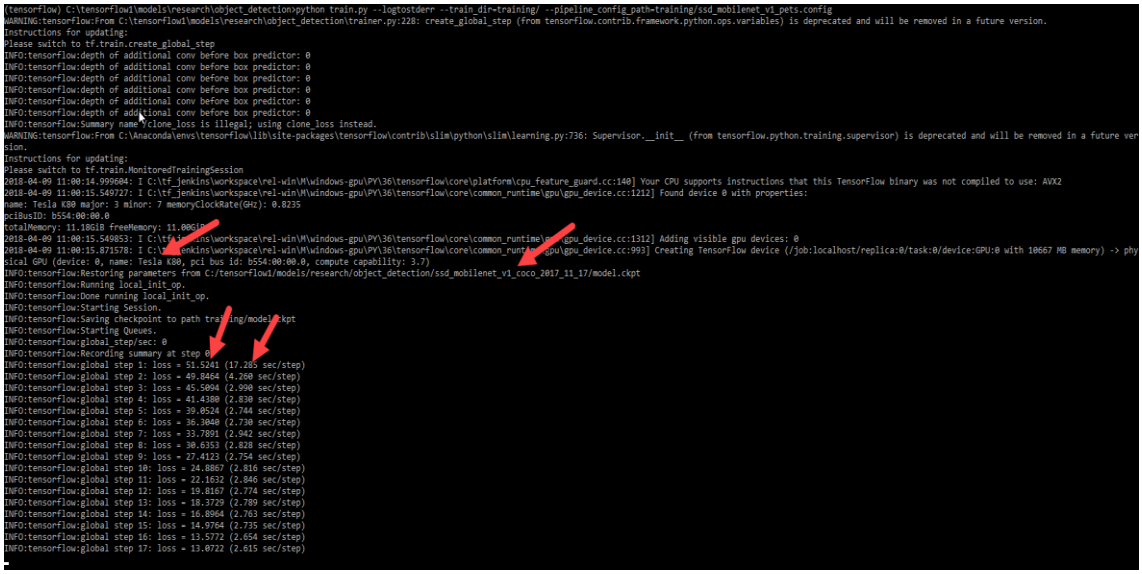


Image Training SSD-Mobilenet

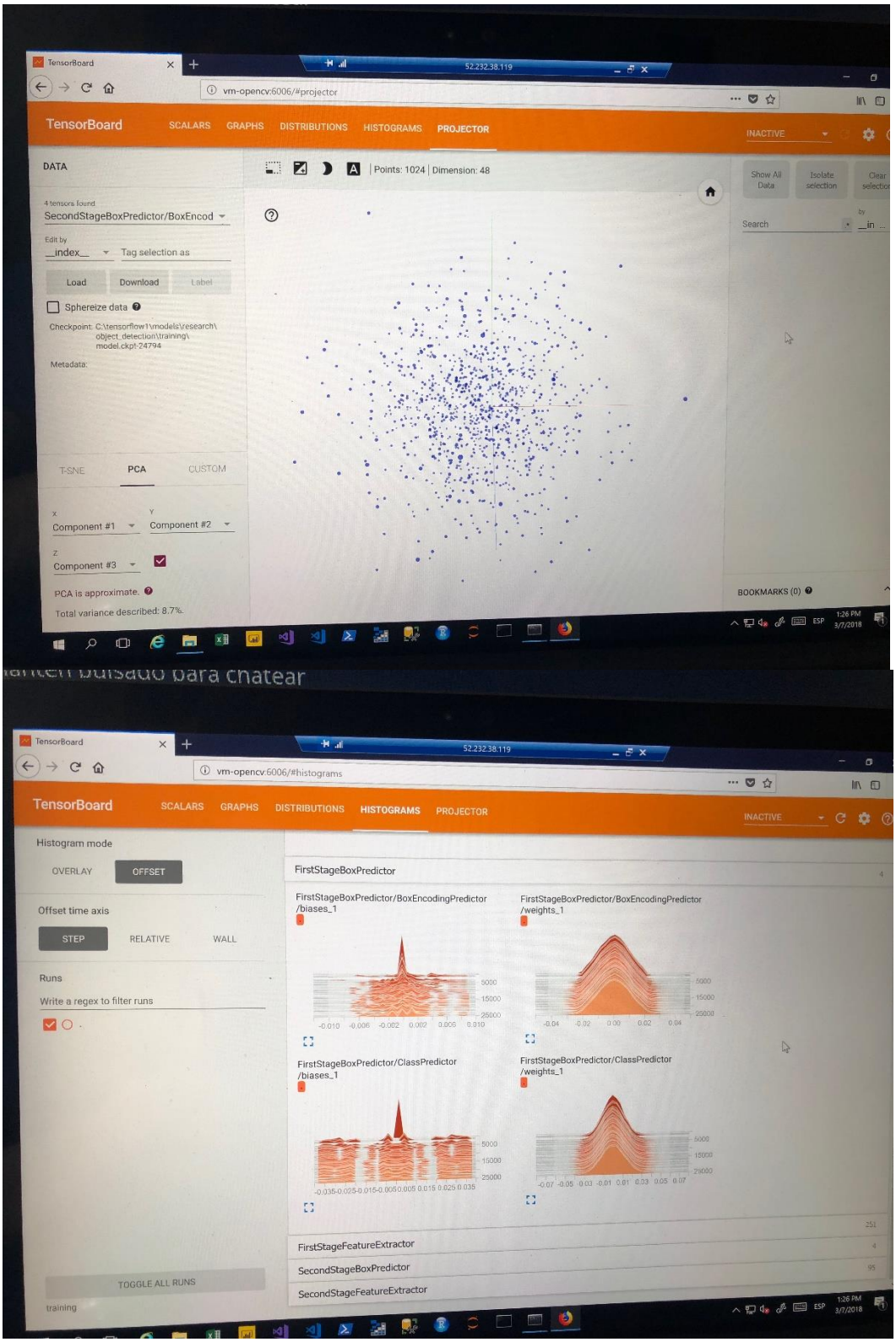
You can view that FasterRCNN training los is faster than SSD-Mobilenet. Also, you can view progress of the training job by using TensorBoard. Thanks to tensorboard I have been able to explain in my final project through graphs like training has been from beginning to end, demonstrating also differences between both models.

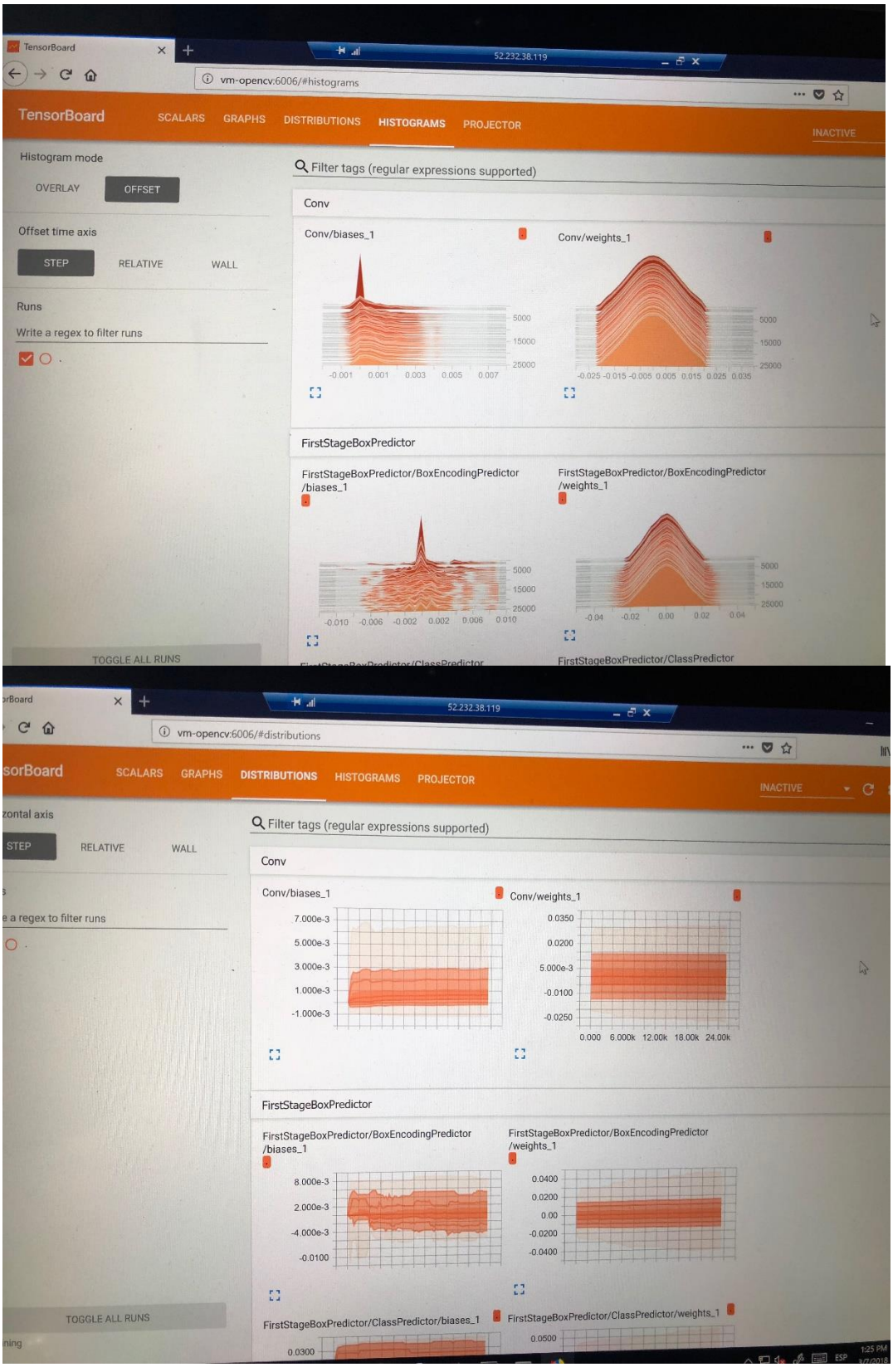
https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

To do this, open a new instance of Anaconda Prompt, activate the tensorflow virtual environment, change to the C:\tensorflow1\models\research\object_detection directory, and issue the following command:

```
(tensorflow) C:\tensorflow1\models\research\object_detection>tensorboard --logdir=training
```

This will create a webpage on your local machine at PCNAME:6006, which can be viewed through a web browser. The TensorBoard page provides information and graphs that show how the training is progressing. One important graph is the Loss graph, which shows the overall loss of the classifier over time.


TensorBoard
vm-opencv:6006/#projector
SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS PROJECTOR
INACTIVE
DATA
Points: 1024 | Dimension: 48
SecondStageBoxPredictor/BoxEncod
Load Download Label
Sphereize data
Checkpoint: C:\tensorflow1\models\research\object_detection\training\model.ckpt-24794
Metadata:
T-SNE PCA CUSTOM
Component #1
Component #2
Component #3
PCA is approximate.
Total variance described: 8.7%.
Search
BOOKMARKS (0)
vm-opencv:6006/#histograms
Histogram mode
OVERLAY OFFSET
Offset time axis
STEP RELATIVE WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
training
FirstStageBoxPredictor
FirstStageBoxPredictor/BoxEncodingPredictor/biases_1
FirstStageBoxPredictor/BoxEncodingPredictor/weights_1
FirstStageBoxPredictor/ClassPredictor/biases_1
FirstStageBoxPredictor/ClassPredictor/weights_1
FirstStageFeatureExtractor
SecondStageBoxPredictor
SecondStageFeatureExtractor



The training routine periodically saves checkpoints about every five minutes. You can terminate the training by pressing Ctrl+C while in the command prompt window. I typically wait until just after a checkpoint has been saved to terminate the training. You can terminate training and start it later, and it will restart from the last saved

checkpoint. The checkpoint at the highest number of steps will be used to generate the frozen inference graph.

7. Create and exporting the frozen inference graph

Now that training is complete, the last step is to generate the frozen inference graph (.pb file). From the \object_detection folder, issue the following command, where "XXXX" in "model.ckpt-XXXX" should be replaced with the highest-numbered .ckpt file in the training folder:

```
python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/faster_rcnn_inception_v2_pets.config --trained_checkpoint_prefix training/model.ckpt-XXXX --output_directory inference_graph
```

This creates a frozen_inference_graph.pb file in the \object_detection\inference_graph folder. The. pb file contains the object detection classifier.

8. Testing and using your newly trained object detection classifier

Now that we have our classifier ready we can put it to the test. I enclosed them in the github repository Python scripts developed by Evan Juras to test it on an image, video or source of the webcam.

Before executing the Python scripts, we will have to change several code variables, then those changes are shown in the image:

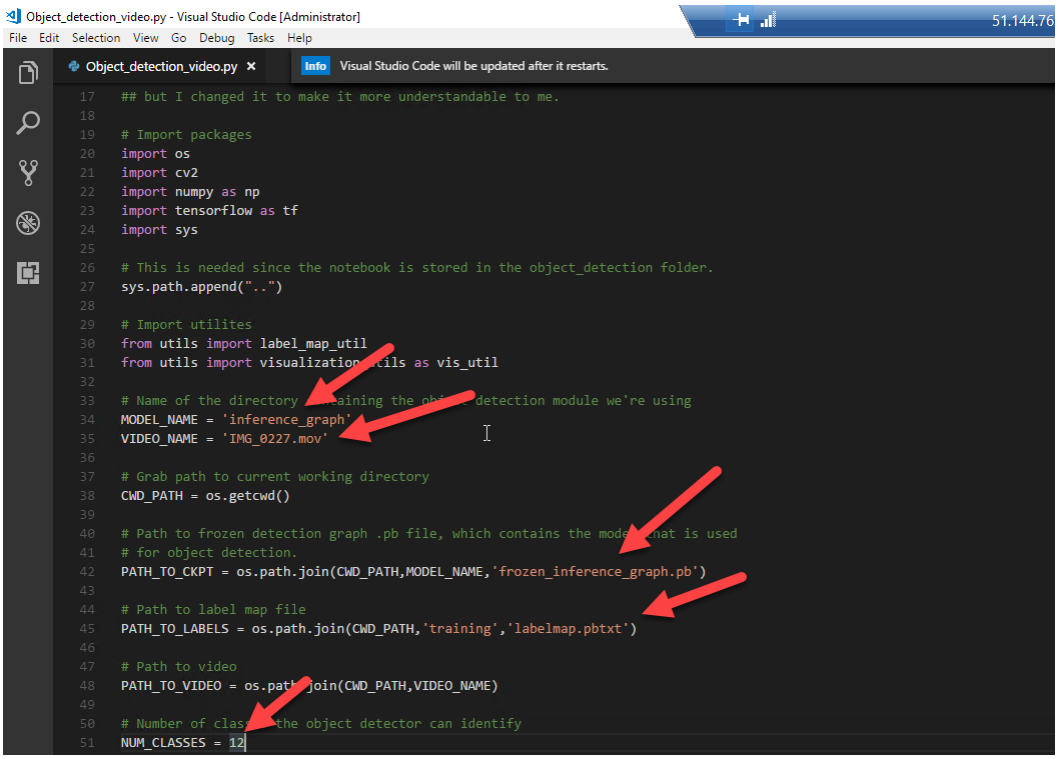


Image Example of changes in Object_Detection_video.py

9. Common errors throughout the Project

1. *Google may add more .proto files to the object_detection/protos folder, so it may be necessary to add more files to the "protoc" command at 13:13. You can do this by adding ".\object_detection\protos\FILENAME.proto" to the end of the long command string for each new file.*

2. *When running the "python train.py" command, if you get an error that says "TypeError: __init__() got an unexpected keyword argument 'dct_method'.", then remove the "dct_method=dct_method" argument from line 110 of the object_detection/data_decoders/tf_example_decoder.py file.*

3. *When running "python train.py", if you get an error saying "google.protobuf.text_format.ParseError: 110:25 : Expected string but found: '"' ", try re-typing the quotation marks around each of the filepaths. If you copied the filepaths over from my GitHub tutorial, the quotation marks sometimes copy over as a distinctive character type, and TensorFlow doesn't like that.*

4. *For train.py, if you get an error saying "TypeError: Expected int32, got rang e(0, 3) of type 'range' instead.", it is likely an issue with the learning_schedules.py file. In the \object_detection\utils\learning_schedules.py file, change line 153*

5. *from "tf.constant(range(num_boundaries), dtype=tf.int32)," to "tf.constant(list(range(num_boundaries)), dtype=tf.int32),".*

10. Next post!

The third post will explain another way of recognizing and classifying images (20 artworks) using scikit learn and python without having to use models of TensorFlow, CNTK or other technologies which offer models of convolved neural networks. Moreover, we will explain how to set up your own web app with python. For this part a fairly simple API which will collect information about the captured image of our mobile application in Xamarin will be needed, so it will inference with our model made in python, and it will return the corresponding prediction. With that methodology we can get easy classification without heavy hardware like TensorFlow or CNTK.

**Note: Also explain how I can prove my frozen inference model in Android!*

3. Step by step - How to train an objects classifier understanding Computer Vision techniques with Python and OpenCV

In the previous post I explained how to create your own image detector with tensorflow. It should be noted that you must differentiate between a classifier and an image detector. So, what is the difference between Object Detection and Object Recognition! Well, recognition simply implies establishing whether an image contains a specific object or not. while detection also requires the position of the object within the image. For example, there is an entry image that contains car, traffic lights, people, dogs, etc. The task is to be able to recognize which of the objects are contained in the image.

In this new post we will explain step by step how to create your own image classifier, not detection image as we did in the previous post. This time without using third-party technologies. If you have a computer with enough computer level and you also have a GPU like Nvidia GTX 650 or newer, you will not have any problems but also in the case that you do not have a very powerful computer you can follow this post and make a classifier without problems.

1. How to set up your virtual machine on Linux with Python and OpenCV

The classifier can be developed as we want, we can do it:

1. Data Science Virtual Machine for Linux(Ubuntu) on Azure

In the first place using virtual machines in Azure, already preconfigured. If we have deployed our machine in Azure as in the first post:

https://blogs.msdn.microsoft.com/esmsdn/2018/04/02/post-invited-analysis-and-object-detection-of- artworks-with-tensorflowgpu-on-windows-10/

This time we can do the same but this time we will change OS instead of Windows, we will develop everything with Linux. Why? My intention is to show you that in machines with Windows 10 and Linux Ubuntu we can carry out our projects of machine learning with python without problems. Thanks to Anaconda we can create without problems our environment variables that will help us to carry out the same processes in both systems. You can see more information here:

https://azuremarketplace.microsoft.com/en-us/marketplace/apps/microsoft-ads.linux-data-science-vm-ubuntu

https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/dsvm-tools-overview

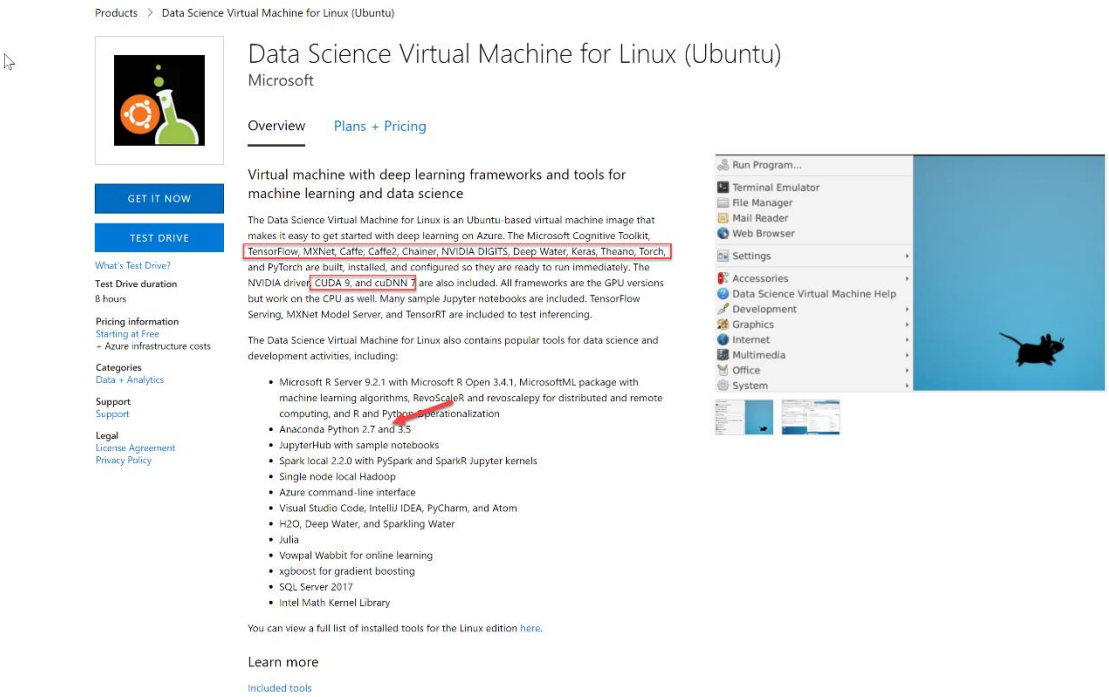


Image Data Science Virtual Machine Linux (Ubuntu) summary

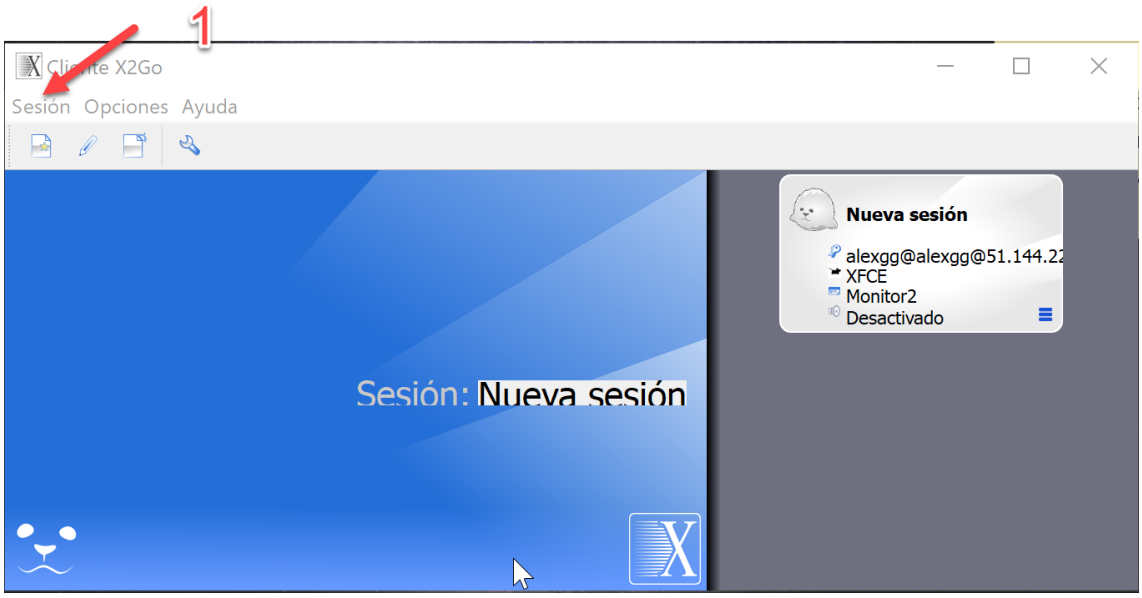
We can see that this VM support many of the current deep learning environments and comes with CUDA and cuDNN installed, although many times without configuring paths. In the first post we explain how to configure these tools paths in our pc. For that case we only need to have Python environment 2.7, this VM default use 3.5 env. Also, we will not have to necessarily use a graphic card to develope this classifier of Artworks.

Therefore, the idea of develope this classifier in a virtual machine of this dimension is not the best idea or profitable.

In the case that we want to use this option we have two options to develope the classifier project, one is connecting directly with PUTTY to the virtual machine and command line to organize and launch the commands to configure anaconda and the rest of the project. Another option is to connect by X2GO, it provides us UI rather friendly and close to a desktop interface. Be free to choose!!

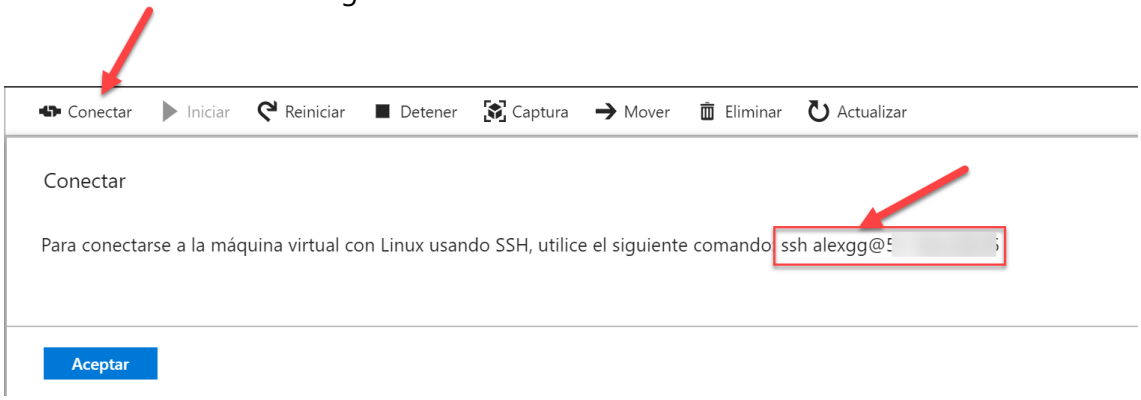
Option X2GO

We need craate a new session first

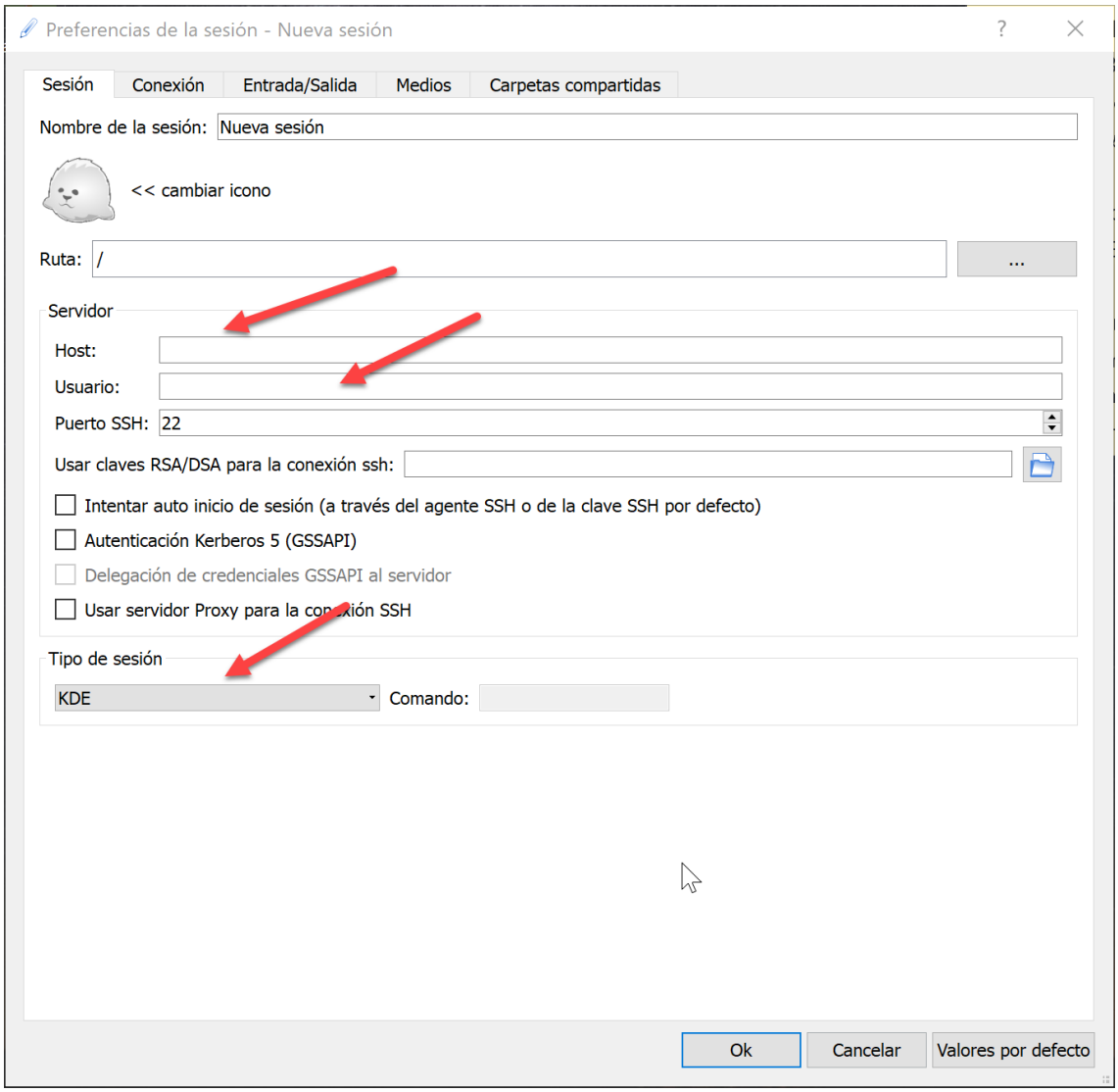


Then we need to fill this form:

1. **Host**: The host that azure give us to connect

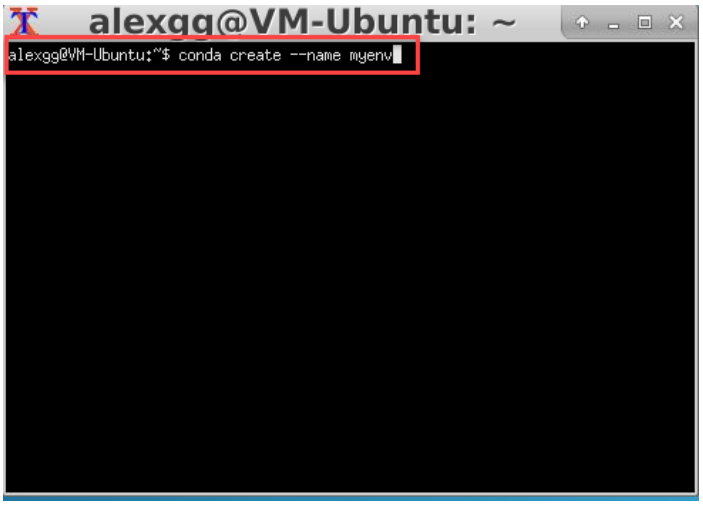


2. **User:** The user that you assigned when you create your machine on Azure
3. **Port SSH:** 22
4. **Type of session:** Change KDE to KFCE

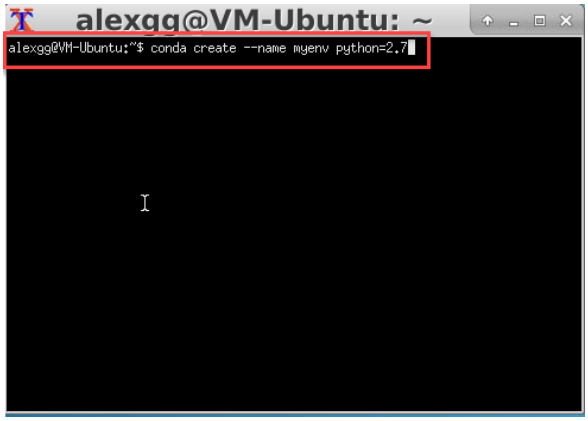


Since we have X2GO with our session set up we can start to start our project!

The first thing we must do is configure our environment in python with the libraries that our project files need. To create the environment variable, we can launch this command:



But there we are not assigning any library or what version of python we want, by default it will not assign the latest version and for the case of this project we do not need it but the 2.7. Therefore we need to write this command:



Once we have our environment already configured in anaconda with python 2.7 we will be able to start running our .py files that will be explained later.

Our PC or Laptop

To carry out the project from your PC, first you must download and install Anaconda:

https://www.anaconda.com/download/

It is very easy to start with python. Thanks to anaconda we can set up our environments easily and it's done in the same way that we explained earlier. Here you have a link to the anaconda documentation, it explains you how to configure your environments:

https://conda.io/docs/user-guide/tasks/manage-environments.html

2. Download this tutorial's repository from GitHub

Download the full repository located on this page, scroll to the top and click Clone or Download and extract all the contents directly into the C:\ directory. This establishes a specific directory structure that will be used for the rest of the post. At this point, your project folder should looks like:

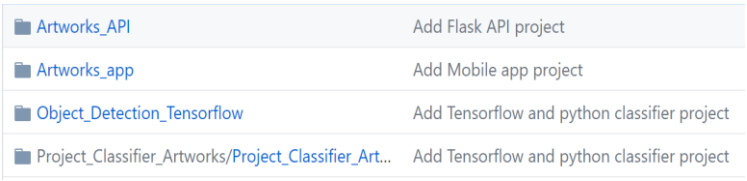


Image Github repository solutions. API, App, Object Detection and Object classifer.

Solutions:

- Artworks API with Flask
- Artworks Xamarin App
- *Artworks Classifier*

The first two solutions (API and App) we will explain in the last sections of this tutorial. The latter is the one that we will explain more in detail.

Artworks Classifier Solution

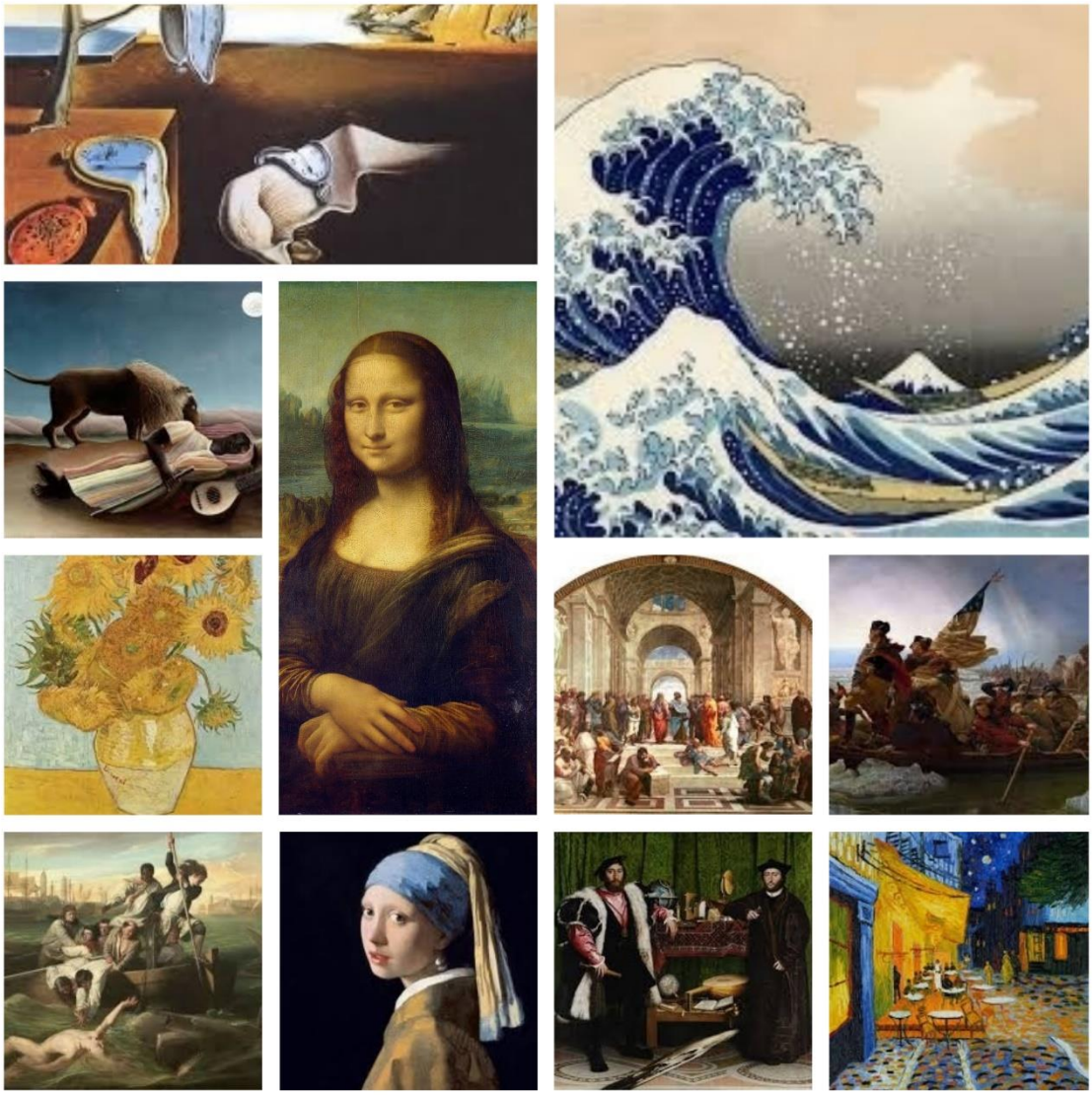


Image many artworks of the classsifier

This folder contains the images of 20 artworks, python files needed to train artworks classifier. It also contains a CoreML model if you wanted to use on iOS project with Xcode, it was generated with Coremltools library

If you want to practice training your own artwork classifer, you can leave all the files as they are. You can follow along with this tutorial to see how each of the files were generated, and then run the training. You will still need to generate the K-Means Cluster model and SVM model as described in nexts steps.

If you want to train your own object classifier, delete the following files (do not delete the folders):

- All files in \images\train

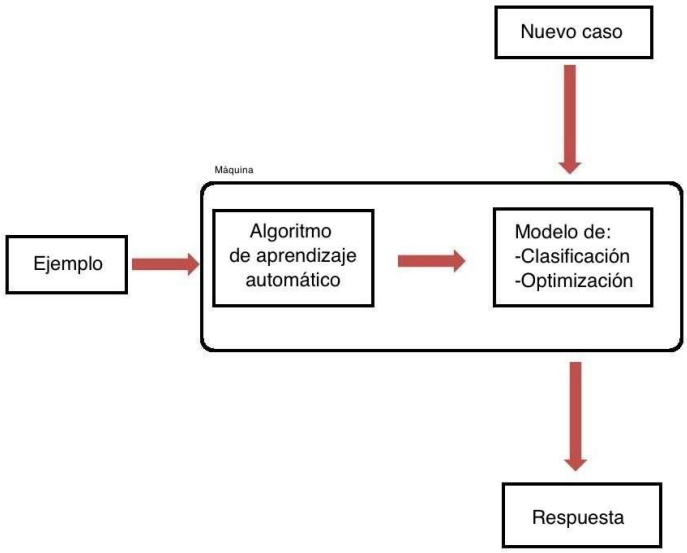
Now, you're ready to start from scratch by training your own object classifier. This publication will assume that all the files listed above are unknown and will continue to

explain how to use these files for your own training data set. Also, in this tutorial I explain important topics that are needed to complete this tutorial, knowing them we will be easier to understand the why of the processes. Such as:

- Machine learning techniques
- Algorithms
- Image descriptors
- Histograms and color histograms for artworks
- K-Mean Clustering
- Bag of Visual Words

3. Introduction to Machine Learning Techniques (Computer Vision)

Machine learning is related to computer science specifically with Artificial Intelligence. Within the latter the goal of machine learning is to create methods that allow computers to have the ability to learn.



Within machine learning we find a subarea that we will use to create this project, called *Computer Vision*, it is an area that includes different functionalities such as acquiring, processing, analyzing and understanding real-world images in order to translate an image to numerical or symbolic information understandable to a computer. The acquisition of data is achieved by means such as camera, multidimensional data, scanners ... etc.

Today this discipline is used for different purposes, such as:

- **Object detection**
- **Video analysis**
- **3D Vision**

We will only focus on the first option, detection of objects in an image. The detection of objects is a part of the artificial vision that detects objects in an image based on their visual appearance. Within the detection of objects in an image we can distinguish two phases:

- Extraction of image characteristics

    - Consists of obtaining mathematical models that summarize the content of the image. These characteristics are also called descriptors, we will comment more in detail in next sections.

- **Search for objects based on these characteristics**

    - For the object search process, we will have to elaborate the classification of said objects. There are several machine learning algorithms that will allow us to assign an identifier or label to an image. We will discuss it in future sections.

Therefore, the classification of an image is the task of assigning a label to an image of a predefined set of categories. This means that, given an input image, our task is to analyze the image, return a label that categorizes the image. This tag is usually from a predefined set.

As, for example, if in our model classifier we do an inference with this image:

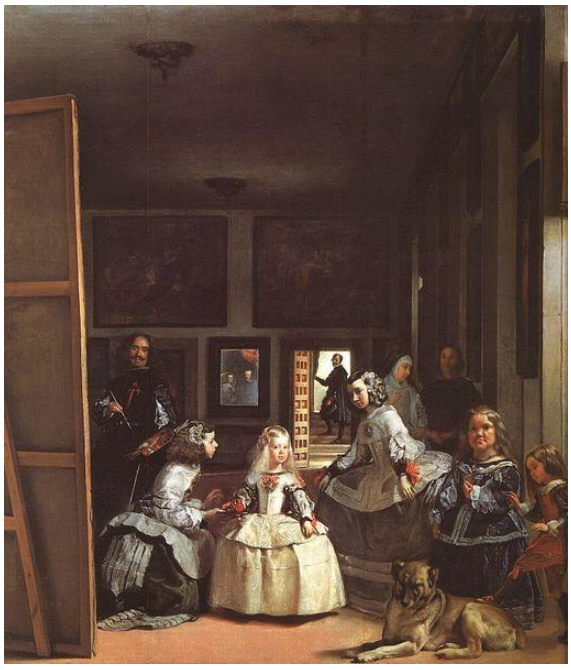


Image "Las Meninas"

It would have to return:

Label: Las Meninas

In a more formal way, given the previous image of input W x H pixels, with three channels, R (Red) G (Green) B (Blue), respectively, our objective will be to take the pixels W x H x3 = N and find out how to accurately classify the contents of the image.

In addition, we must realize that in computer vision we must give value to semantics. For a human it is trivial that he knows that the previous image is a picture of art, specifically the Meninas. But in a computer, it is not trivial, nor does it have to know it. For the computer to analyze them, it will differentiate three main properties of every image:

- **Spatial environment**
- **Color**
- **Texture**

These properties are encoded in a computer thanks to what we previously named as descriptors, each descriptor specializes in space, colors, textures, etc. Finally, based on the computer with these characterizations of space, textures and colors, you can apply automatic learning to learn how each type of image is, in our case, diverse types of art pictures of different artists.

For this we also must understand how computers represent images to analyze them. As for example if we insert the picture of the Mona Lisa or Mona Lisa in our classifier, it would represent it like this:

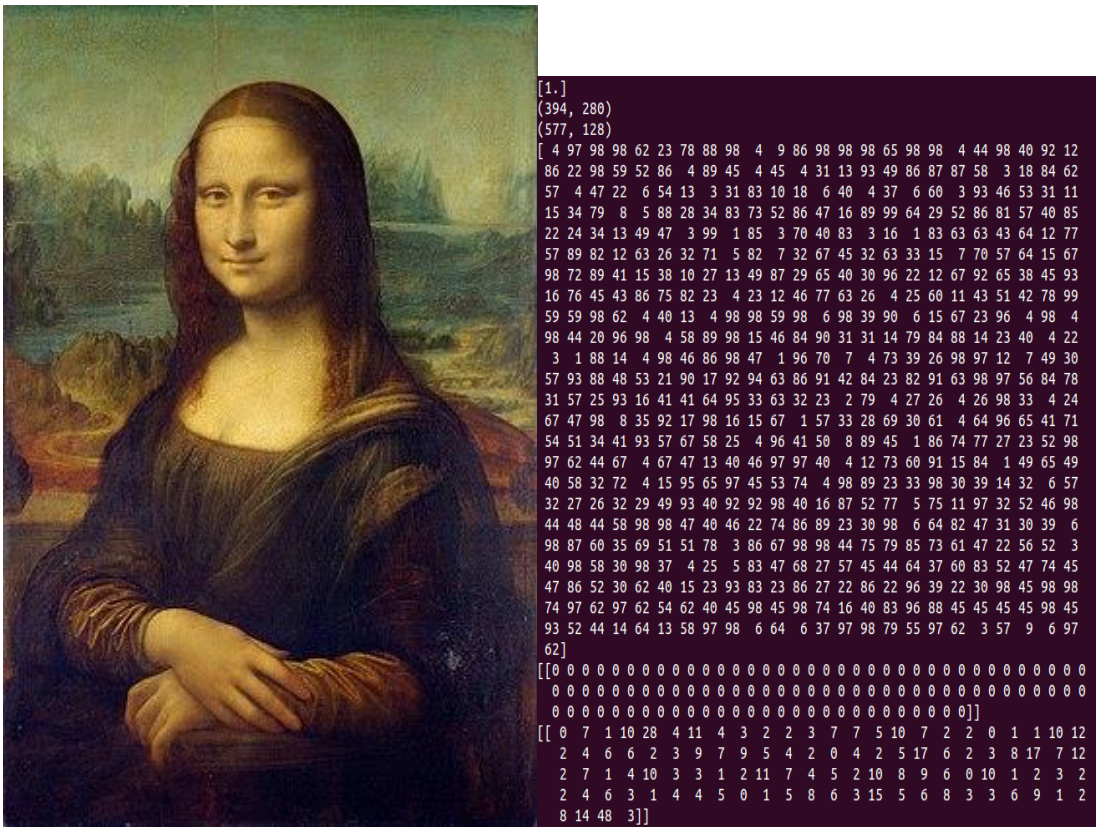


Image Giocona and her features descriptors

In addition, another section to investigate on how to build our classifier of art pictures is how the image or an object appears in an image. That is, the different points of view of a painting, different dimensions of the pictures, deformations of the image, lighting and occlusions.

1.4.2 Types of learning

When carrying out the research prior to the realization of the project we have encountered this problem, what kind of learning we want for our project. We have observed that there are three types of learning:

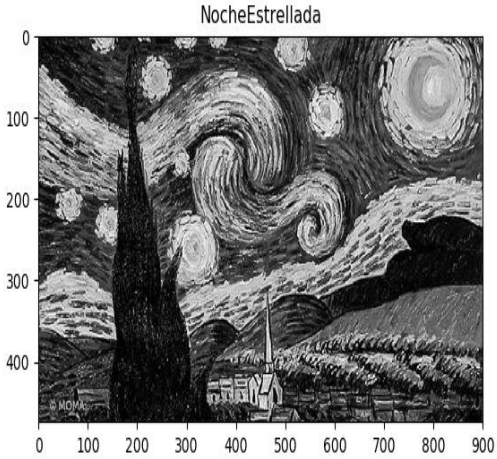
- **Supervised**
    - We have both image data (in image format or extracted feature vectors) along with the category label associated with each image so that we can teach our algorithm how each image category looks.
- **Non supervised**
    - All we have is data from the image itself, we do not have labels or associated categories that we can use to teach our algorithm to make accurate predictions or classifications.
- **Semi-supervised**
    - Try to be a middle ground between the previous two. We have a small group of our tagged image data and use that tagged information to generate more training data from the unlabelled data.

In the end we opted to use the type of supervised learning since in our system we will have an image dataset of 20 tables where the category or label is the name of the table and each category contains 21 replications of that table. Something like our classifier would be:

Label	Features Vector
Las Meninas	[...]
3 de Mayo	[...]
Maja desnuda	[...]
Noche estrellada	[...]
Mona Lisa	[...]
...	[...]

Features vector is something like this:

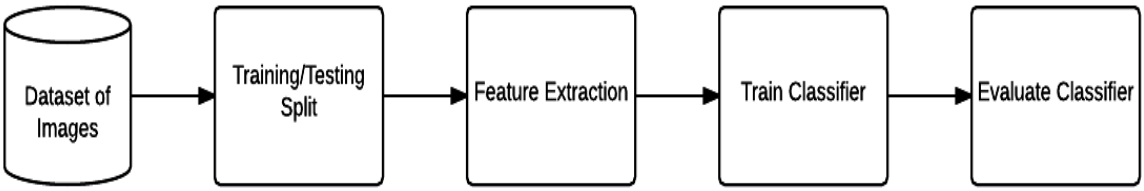
- ***Starry Night (Vincent Van Gogh):***




```
[0.]
processing  NocheEstrellada
(598, 750)
(10891, 128)
[[ 78 120  47  95  76  66 135  75  88  93 168 189  97 101 109 156 135  77
   60 145  67 125 127  86 121 130  96 149 146 186 120 170 145 111  84 151
  96  68 110 169 167  83  74  70 130 158  88  90  64  88  94  77 133  78
 204  75  53  77 129  78  84  82  73 107  96 109  58 108 221  59 169 119
 175  73  62 151 124  58 100  66  64 127 114 142 158 105 102 121 116 103
  98 186  96  74 118 138  65  45 142 106]]
```

1.4.3 Pipeline of the image classifier

Once seen how to manipulate the images and how to face our learning we will have to investigate on how to divide in processes to build our classifier of images of art pictures. One of the most common pipelines is the following one in which we distinguish 5 phases that are:



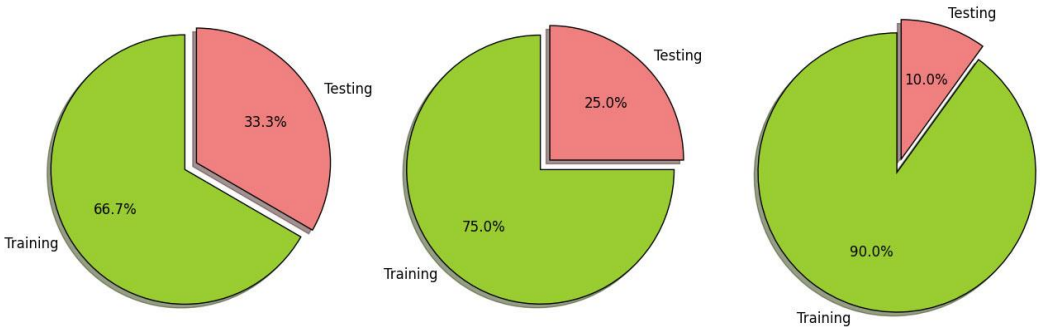
Phase 1: Structure our initial dataset

It will be necessary to create our categories each with their images and specific labeling. In this project it would be like this:

Categories = {Las Meninas, Shooting 3 of May, Gioconda, Maja Desnuda, Starry Night, …}

Phase 2: Splitting our dataset (train and test)

Once we have our initial data set, we must divide it into two parts, training set and evaluation set or tests. Our classifier uses a training set to "learn" what each category looks like by making predictions about the input data and then correct it when the predictions are incorrect. After the classifier has been trained, we can evaluate its performance in a test set.



Algorithms like the ones we use in this Random Forest Classifier project, have many configurable parameters that will help us if we configure them well to obtain optimal performance. These parameters are called hyperparameters.

Phase 3: Extract features

Once we have our final data divisions, we will need to extract functions to quantify and abstract each image. The most common options according to the previous investigation are:

- ***Color descriptors***
- ***Histogram of gradients oriented (HOG)***
- ***Histograms with local binary patterns (BRIEF, ORB, BRISK, FREAK)***
- ***Local Invariant Descriptors (SIFT, SURF, RootSIFT)***

Phase 4: Train our classification model

Given the feature vectors associated with the training data, we will be able to train our classifier. The objective here is for our classifier to learn how to recognize each of the categories in the data of our label. For this we have done an analytical study, such as Support Vector Machine, K-Nearest Neighbor.

Phase 5: Evaluate our classifier

Finally, we must evaluate our trained classifier. For each one of the vectors of characteristics in our test set, we present them to our classifier and we ask you to predict which is the label of the inserted image. Then we will have to tabulate the classifier's predictions for each point in the test set.

Finally, these classifier predictions are compared with the Ground-true label of our test set. The Ground-truth labels represent what the category really is. From there we can calculate the number of predictions that our classifier was successful and calculate aggregate reports such as Precision, Recall and F-Measure, which are used to quantify the performance of our classifier.

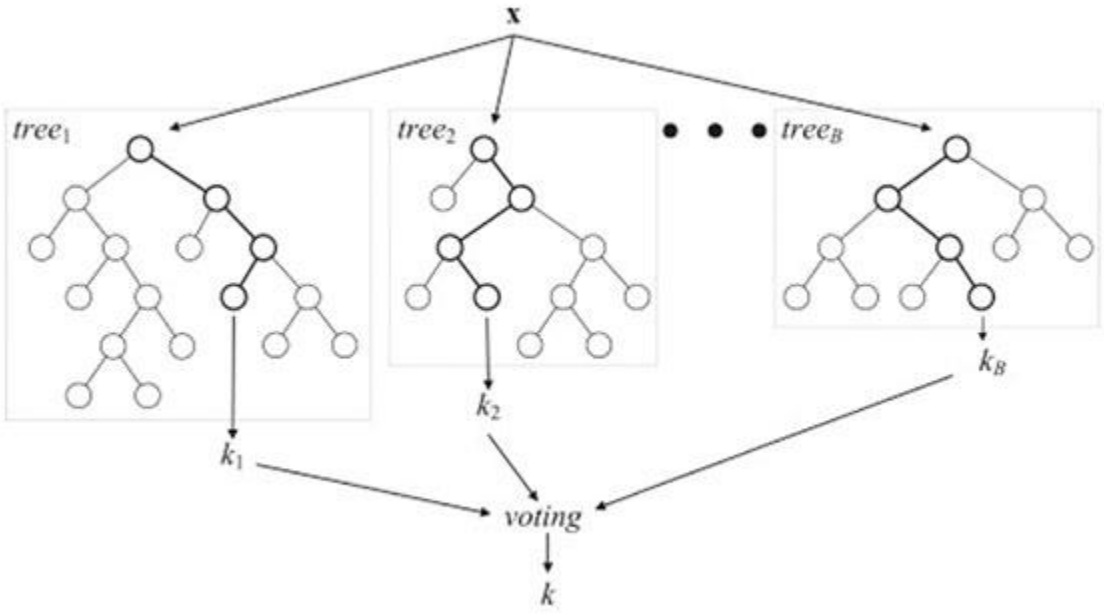
4. Classifications Algorithms (SVM and Random Forest)

We can find in the Machine Learning environment a great diversity of algorithms that we can use for our classification of art frames. After a series of tests between SVM, Decision three, K-Nearest Neighbor and Random Forests, we have chosen the latter due to the results we have found when we need to classify 5 pictorical styles. When our classifier classifies artworks, we used SVM.

Random Forest

We have investigated a bit how this algorithm works in detail. It was created and introduced into the scientific community by Leo Brieman in his 2001 article, Random Forests is one of those algorithms that many scientists still do not believe works, many say it is an elegant and straightforward way to perform a classification, others They say that there are multiple Decision trees but with a touch of randomness that clearly increases their accuracy.

From the satisfactory results and the good comments on it, it has been decided in this section to show the functioning of said algorithm. The Random Forests are a type of method of classification by sets, instead of using a single classifier as we have done in the tests with SVM and K-NN, this uses multiple classifiers that are added in one called goal classifier. In our case we will build multiple decision trees in a forest and then we will use our forest to make predictions.



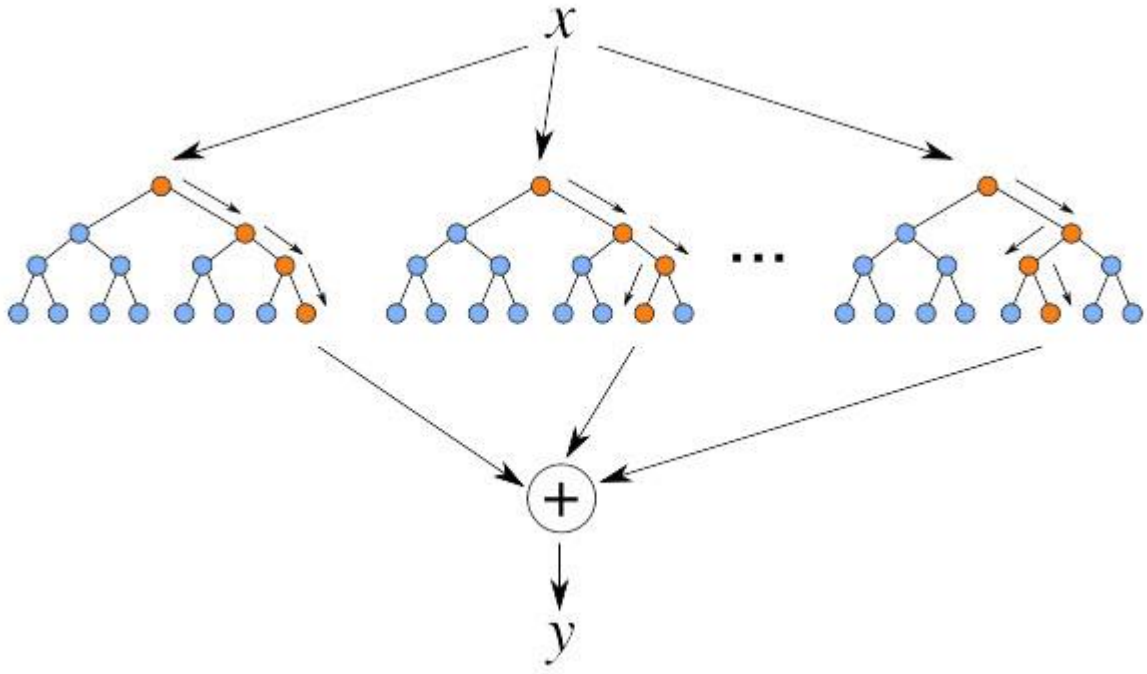
As you can see in the previous figure, our random forest consists of multiple grouped decision trees. Each decision tree "votes" on what it believes is the final classification. These votes are tabulated by the classifying goal, and the category with the most votes is the final classification.

You have to make an appointment to the "Jensen Inequality" to understand a large part of how Random Forests works. Dietterich's seminal work (2000) details the theory of why ensemble methods can generally obtain greater precision than a single model alone. This work depends on Jensen's Inequality, which is known as "diversity" or "decomposition of ambiguity" in the machine learning literature.

The formal definition of Jensen's Inequality states that the combined (average) convex set will have an error that is less than or equal to the average error of the individual models. It may be that an individual model has an error smaller than the average of all the models, but since there is no criterion that we can use to "select" this model, we can be sure that the average of all the models will not be worse than the Select any random individual model.

Another crucial factor is boostrapping or randomization injection. These classifiers train each individual decision tree in an initial sample of the original training data. Boostrapping is used to improve the accuracy of machine learning algorithms while reducing the risk of overfitting.

In the following figure we simulate the selection of "votes" created in the nodes of our classifier. We pass a feature input vector and through each of the decision trees, you will receive the votes of the class label of each of the trees, then a count of the votes will be done to show the final classification or prediction.



In conclusion, we have used this classifier, since it is a set method that consists of multiple decision trees. The ensemble methods, such as the Random Forests, tend to

obtain greater precision than other classifiers, since they averaged the results of each individual model. The reason why this average works is due to Jensen's Inequality.

Randomness is introduced in the selection of training data and in the selection of the characteristic column when training each tree in the forest. As Brieman discussed in his article Random forests, performing these two levels of random sampling helps (1) avoid overfitting and (2) generates a more precise classifier.

Support Vector Machine

The reason why SVMs are so popular is because they have quite solid theoretical foundations. The hyperparameters are still being improved, but in general, launching an SVM to a problem is an effective way to quickly obtain a prediction or a good result for a given problem. However, with what has given me problems with the adjustment of the parameters, if you want to obtain an optimal result you need to play with them and adjust it to the maximum to our problem, for this I found a Python library *GridSearchCV* that helps you to program the parameters that you want to touch and automate a workout executing all the possible combinations of said hyperparameters, with this we will save a lot of time in going testing different values one by one.

Types of SVM

We will only explain how the linear SVM type worksLinear separability

Linear separability

In order to explain SVMs, we should first start with the concept of linear separability. A set of data is linearly separable if we can draw a straight line that clearly separates all data points in class #1 from all data points belonging to class #2:

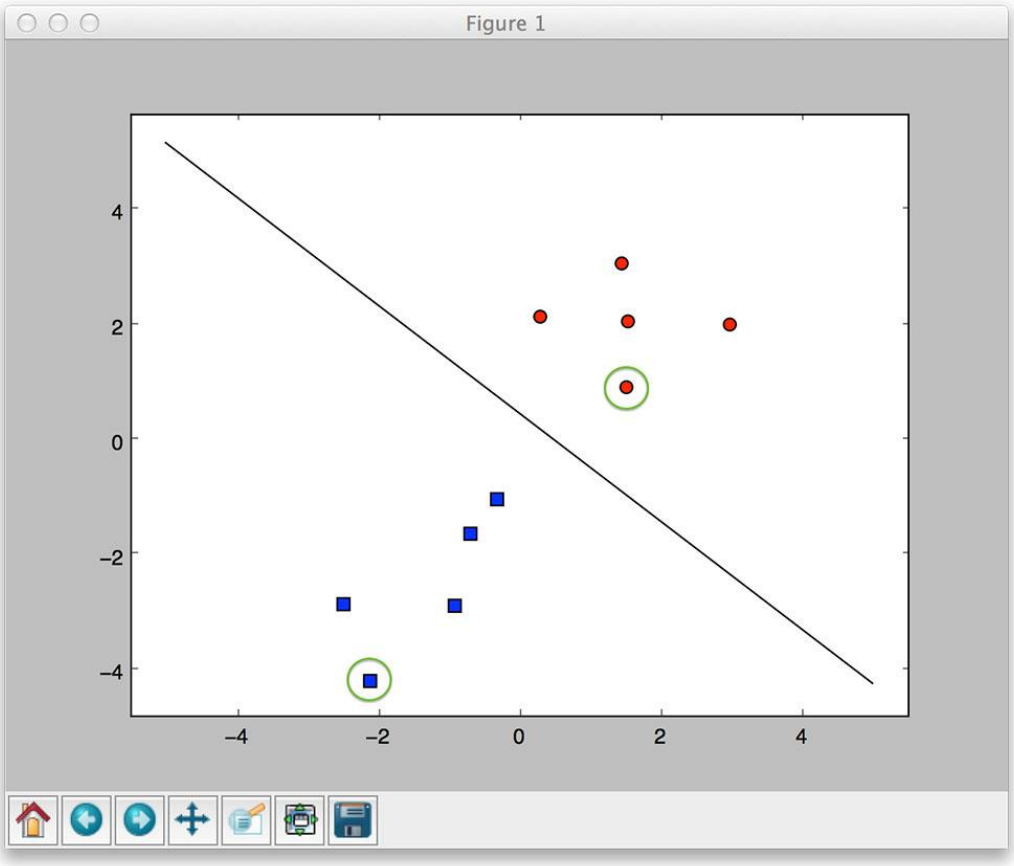


Image. Given our decision boundary, I am more confident that the highlighted square is indeed a square, because it is farther away from the decision boundary than the circle is.

Take a few seconds and examine this plot and convince yourself that there is no way to draw a single straight line that cleanly divides the data points, so all blue squares are on one side of the line and all red circles on the other. Since we cannot do that, this is an example of data points that are not linear separable.

**Note: As we'll see later in this lesson, we'll be able to solve this problem using the kernel trick.*

In the case of Plots, A and B, the line used to separate the data is called the separating hyperplane. In 2D space, this is just a simple line. In a 3D space, we end up with a plane. And in spaces > 3 dimensions, we have a hyperplane.

Regardless of whether we have a line, plane, or a hyperplane, this separation is our decision boundary or the boundary we use to decide if a data point is a blue rectangle or a red circle. All data points for a given class will lay on one side of the decision boundary, and all data points for the second class on the other.

Keeping this in mind, wouldn't it be nice if we could construct a classifier where the farther a point is from the decision boundary, the more confident we are about its prediction?

5. Images Descriptors

A very important part of our project, not to say the main one, is to know how to extract the descriptors of the images in the best way and efficiently possible. Before going into explaining the world of descriptors, basic concepts in the field of computer vision will be briefly explained and many professionals overlook such concepts, such as image descriptors, feature descriptors and feature vectors. All these terms are very similar, but nevertheless it is very important to understand the difference between them.

To begin with, a feature vector is simply a list of numbers used to abstractly quantify the content of an image. Characteristic vectors are transmitted to other computer vision programs, such as the creation of an automatic learning classifier to recognize the contents of the image using feature vectors or comparing feature vectors for similarity when constructing an image search engine. To extract feature vectors from an image, we can use image descriptors or feature descriptors.

An image descriptor quantifies the complete image and returns a vector of characteristics per image. The descriptions of the images tend to be simple and intuitive to understand but may lack the ability to distinguish between different objects in the images.

On the contrary, a feature descriptor quantifies many regions of an image, returning multiple feature vectors per image. Feature descriptions tend to be much more powerful than simple image descriptors and more robust to changes in rotation, translation and point of view of the input image.

An impediment that arose when extracting feature descriptors is that not only do we have to store several feature vectors per image, which increases our storage overhead, but we also need to apply methods such as Bag of Visual Words to take the multiple vectors of characteristics extracted from an image and condensed into a single vector of characteristics. This Bag Visual Words technique will be explained in another section.

Within the descriptors there are several types, but for our case we will use the local invariant descriptors, consisting mainly of:

- *SURF* (Speeded Up Robust Features), Keypoint detector DoG
- *SIFT* (Scale-invariant Feature Transform), Keypoint Detector Fast-Hessian

We will explain the operation of each one of them with an example of execution with the art pictures and we will explain the reason of our choice of SIFT through results represented in graphs.

Understanding Local features

The most usual when analyzing an image above is to use image descriptors to complete image, this leaves us with a global quantification of the image. However, a

global quantification of the image means that each pixel of the image is included in the calculation of the vector characteristics and may not always be the most appropriate.

Suppose for a second that we were tasked with building a computer vision system to automatically identify the covers of books. Our system would take a photo of a artwork cover captured from a mobile device such as an iPhone or Android, extract features from the image, and then compare the features to a set of artworks covers in a database.

We can use HOG for resolve this problem? Or LBP's?

The problem with these descriptors is that they are all global image, and if we use it we will end up by quantifying image regions that do not interest us, such as people who are in front of the frames. Including these regions in our vector calculation can dramatically divert the vector of output characteristics and we run the risk of not being able to correctly identify the art box.

The solution to this is to use local characteristics, where we only describe small local areas of the image that are considered interesting instead of the whole image. These regions must be unique, easily compared and carry some kind of semantic meaning in relation to the contents of the image.

**Note: At the highest level, a "feature" is a region of an image that is both unique and easily recognizable.*

Keypoint detection and feature extraction

The process of finding and describing interesting regions of an image is broken down into two phases: keypoint detection and feature extraction.

The first phase is to find the "interesting" regions of an image. These regions could be edges, corners, "blobs", or regions of an image where the pixel intensities are approximately uniform. There are many different algorithms that we'll study that can find and detect these "interesting" regions — but in all cases, we call these regions keypoints. At the very core, keypoints are simply the (x, y)-coordinates of the interesting, salient regions of an image.

Then for each of our keypoints, we must describe and quantify the region of the image surrounding the keypoint by extracting a feature vector. This process of extracting multiple feature vectors, one for each keypoint, is called feature extraction. Again, there are many different algorithms we can use for feature extraction, and we'll be studying many of them in this module.

However, up until this point, we have had a one-to-one correspondence between images and feature vectors. For each input image, we would receive one feature vector out. However, now we are inputting an image and receiving multiple feature vectors out. If we have multiple feature vectors for an image, how do we compare them? And how do we know which ones to compare?

As we'll find out later in this tutorial, the answer is to use either keypoint matching or the ***bag-of-visual-words model***

Research on SIFT and SURF

First, we explain what SIFT is. SIFT descriptor is a lot easier to understand than the Difference of Gaussian (DoG) keypoint detector also proposed by David Lowe in his 1999 ICCV paper, Object recognition from local scale-invariant features. The SIFT feature description algorithm requires a set of input keypoints. Then, for each of the input keypoints, SIFT takes the 16 x 16-pixel region surrounding the center pixel of the keypoint region.

The SURF descriptor is a characteristic vector extraction technique developed by Bay in its 2006 ECCV document. It is very similar to SIFT, but it has two main advantages with respect to SIFT.

1. The first advantage is that SURF is faster to calculate than SIFT, so it is more suitable for real-time applications.

2. The second advantage is that SURF is only half the size of the SIFT descriptor. SIFT returns a feature vector of 128-dim and SURF returns a vector of 64-dim.

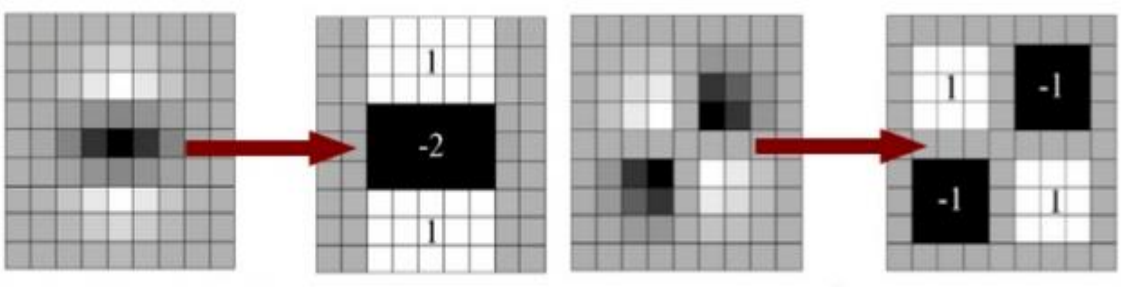
Having understood its advantages, we have decided to set an objective and it is to understand why SURF works and because it obtains such satisfactory results, for this we have had to understand a keypoint descriptor called Fast Hessian. Today, both the SURF image description algorithm and the Fast Hessian keypoint descriptor ended up calling both by the scientific community "SURF" although they may be confusing.

Initially before SIFT and SURF were introduced there was a trend in which the proposed algorithms included a keypoint detector and an image descriptor, so sometimes we see both keypoint detectors and image descriptors share the same name.

But how does the Fast Hessian keypoint descriptor work?

The motivation of Fast Hessian and SURF came from the slowness of DoG and SIFT. The computer vision researchers wanted a faster keypoint detector and image descriptor.

The Fast Hessian is based on the same principles as DoG in that keypoints must be repeatable and recognizable at different scales of an image. However, instead of calculating the Gauss Difference explicitly as done in DoG, Bay proposed that instead we approximate the Gaasian Differences step using what is called Haar Wavelets and integral images. We will not go into detail about this technique, but in the following image you can see how it would become a theory:



And from the previous result of building a matrix:

$$H = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix}$$

A region will be marked with a keypoint if the candidate pixel score is 3 x 3 x 3 times greater than the neighbor. Unlike SIFT, this time we are only interested in maximums, not maximums or minimums. The use of Fast Hessian for our application is the best choice since it is very appropriate for a real-time approach. In the following image we have tested an image of "The persistence of memory" of Dali executing on this Fast Hessian and extracting its keypoints:

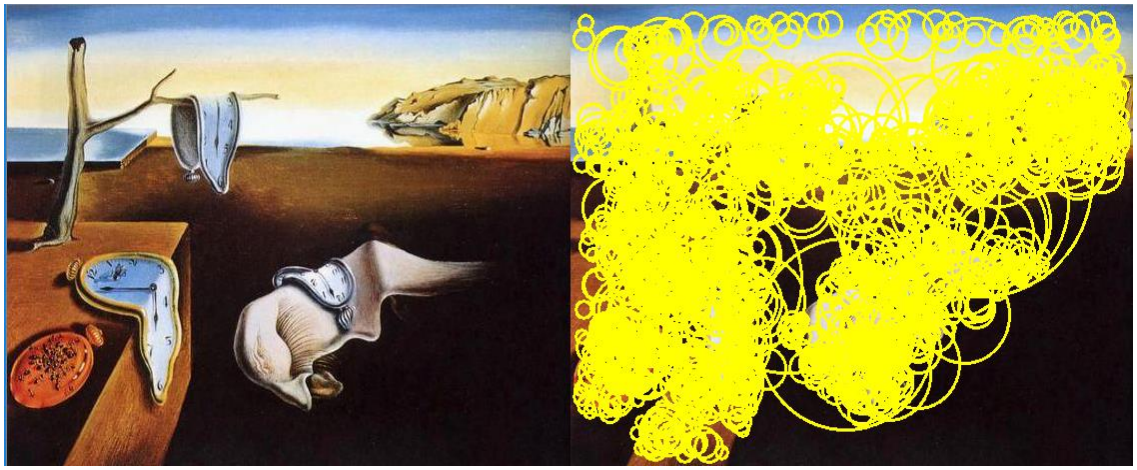
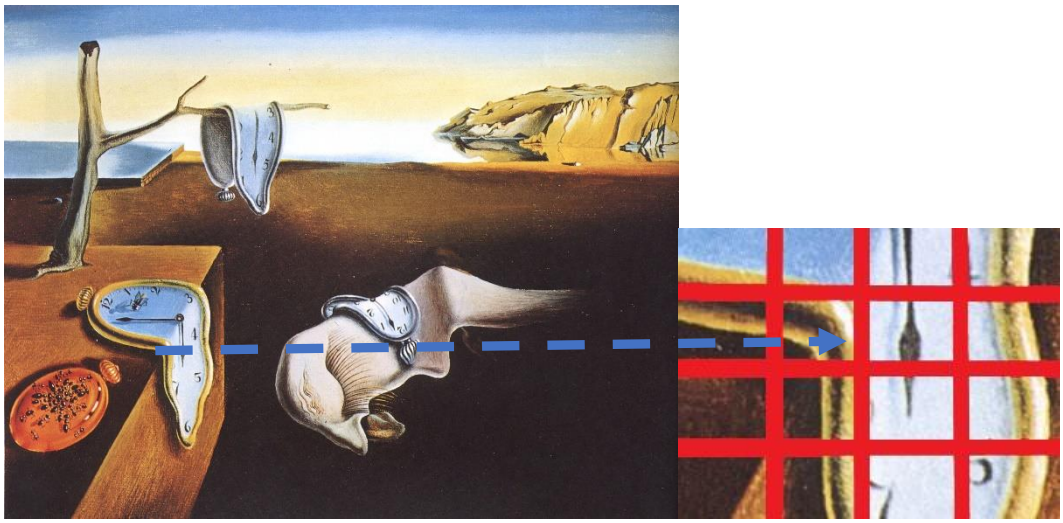
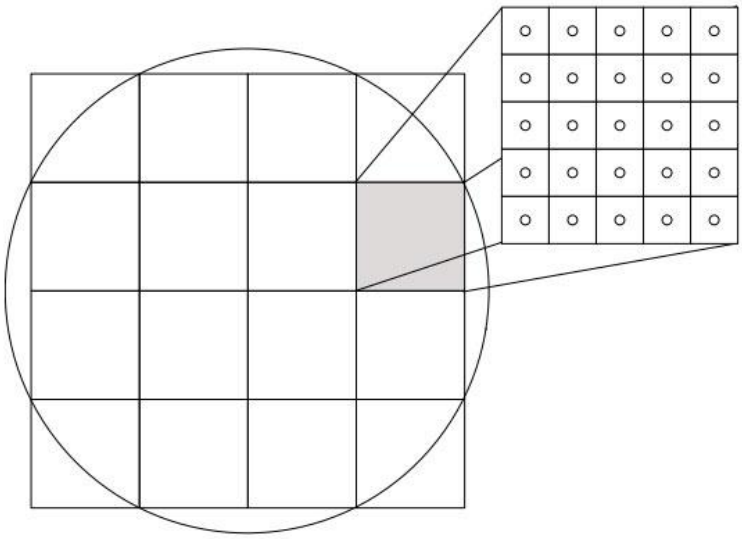


Image Fast Hessian Keypoints of "Persistence of memory"

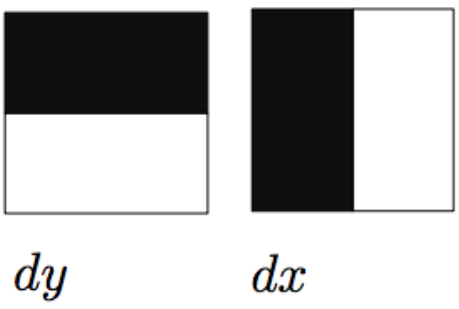
Once these keypoints are extracted our image description algorithm, SURF, we will go through each one of the keypoints obtained and we will divide the keypoint region into 4 x 4 sub-areas, just like in SIFT.



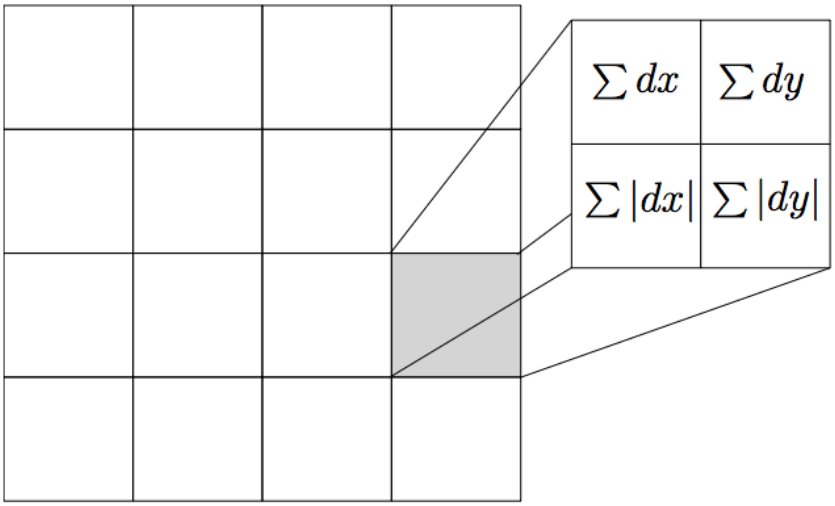
From this step is when SIFT and SURF begin to differentiate. SURF for each of these sub-areas of 4 x 4 extracts by means of the Haar Wavelet sample points of 5 x 5.



And for each extracted cell, both directions X and Y are processed by Haar Wavelets. This result is known as d_ {x} and d_ {y}.



Now that we have d_ {x} and d_ {y}, we will extract their weights by means of what is called as Gaussian Kernel as in SIFT. The results furthest from the center of the keypoint will contribute less to the vector of final characteristics, however, the results that are closer to the center of the keypoint will contribute to the vector of final characteristics. And finally, SURF to finish its process and finally process its characteristics vector for each sub-area of 4 x 4 process this formula:



Therefore, as we said at the beginning we have 4 x 4 = 16 subareas, returning a 4-dim vector to us. These characteristic vectors with 4-dim will concatenate with each other, giving rise to 16 x 4 = 64-dim of the characteristic vector of SURF. Below are results with paintings by Velázquez (Las Meninas) and DaVinci (The Last Supper).

Image SURF keypoints of "Las Meninas"

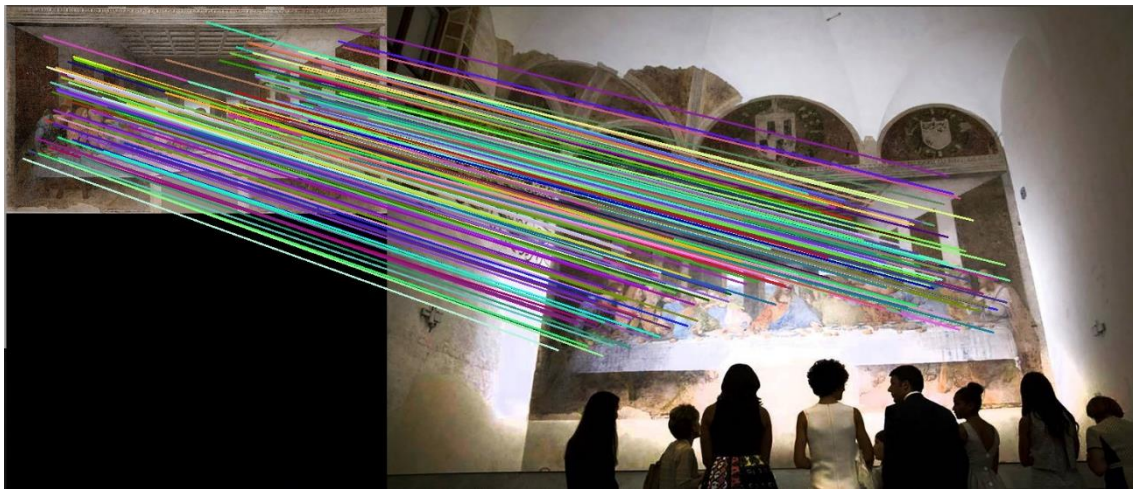


Image SURF Keypoints of "The Last Supper"

6. Histograms and color histograms for artworks

Within the fundamental concepts in computer vision we must give weight to the histograms. So, what exactly is a histogram? A histogram represents the distribution of pixel intensities (whether color or gray- scale) in an image. It can be visualized as a graph (or plot) that gives a high-level intuition of the intensity (pixel value) distribution. We are going to assume a RGB color space in this example, so these pixel values will be in the range of 0 to 255. And now what is a color histogram? Simply put, a color histogram counts the number of times a given pixel intensity (or range of pixel intensities) occurs in an image. Using a color histogram, we can express the actual distribution or "amount" of each color in an image. The counts for each color/color range are then used as our feature vector.

Image Athens School artwork

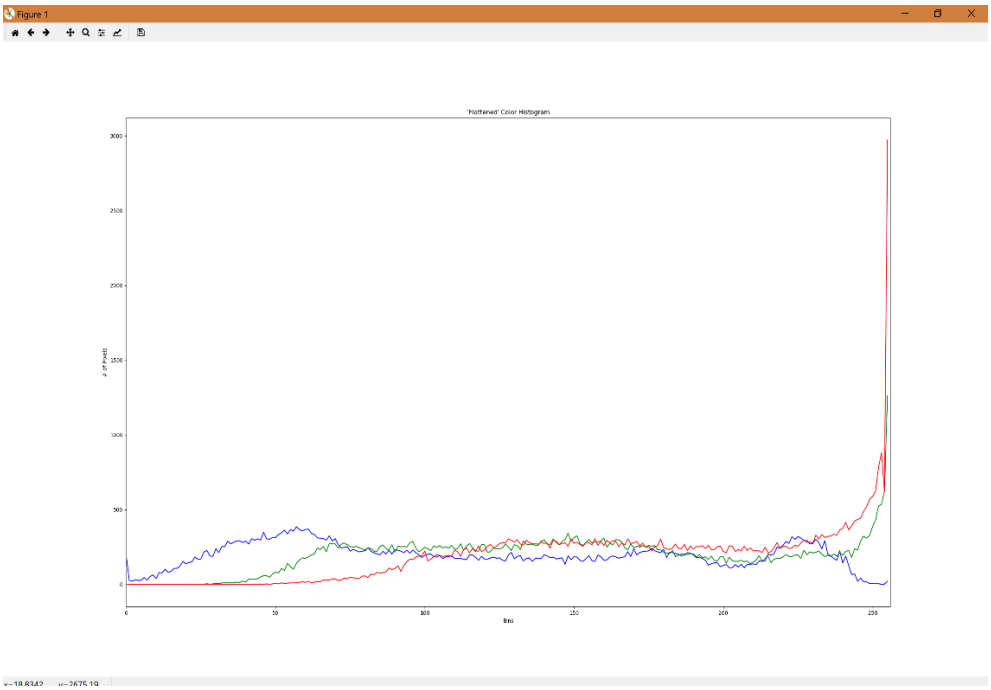


Image Histogram of Athens School artwork

We see there is a sharp peak in green and red histogram around bin 200, the end of the histogram represents that the men on the right with the toga with "green" and "red" tonality contain many pixels with that color range. We see that most of red, green, blue pixels are container in all the range except in final when blue decrease.

*Note: *I'm intentionally revealing which image I used to generate this histogram. I'm simply demonstrating my thought process as I look at a histogram. Being able to interpret and understand the data you are looking at, without necessarily knowing its source, is a good skill to have in computer vision.*

Now in the image below que can see a grapchic of computing 2d color histograms for each combination of the red, green, and blue channels. First is a 2D color histogram for the Green and Blue channels, the second for Green and Red, and the third for Blue and Red.

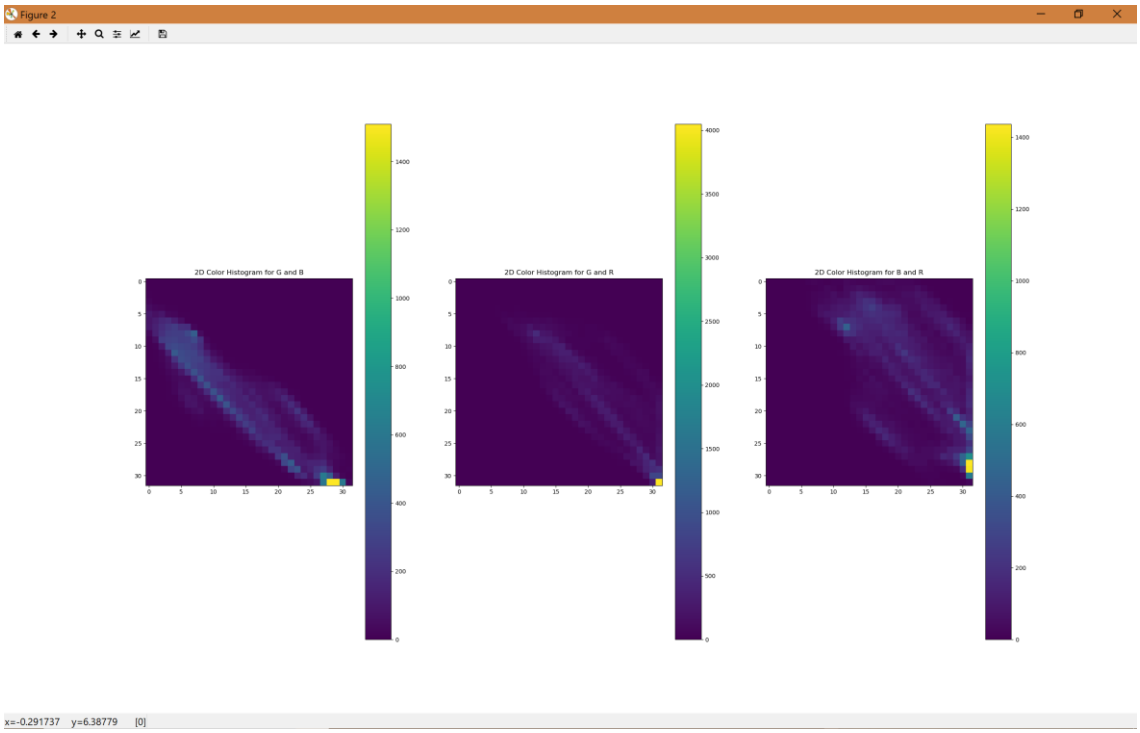


Image computing 2d color histograms for each combination of the red, green, and blue channels.

7. K-Means Clustering

Once we understand how we extract the color histograms thanks to K-means we can group colors and know which the predominant ones in each painting, artist or pictorial style are. In addition to relating a feeling with the colors that we extract from a painting, such as *happy* or *sad*.

There are many clustering algorithms in ML but k-means after investigation I discovered that it is the most popular, most used and easiest to understand clustering algorithm. The K-Means algorithm is a type of unsupervised learning algorithm (no label/category information associated with the images/feature vector), I was explained this concept before in *Machine learning techniques chapter*.

Clustering algorithms seek to learn, from the properties of the data, an optimal division or discrete labeling of groups of points. Scientist call K-means because it finds "K" unique clusters where the center of each cluster(centroid) is the mean of all values in the cluster. The overall goal of use k-means is to put similar points in a cluster and dissimilar data points in a different cluster. For more information about clustering techniques enter in this link.

Many clustering algorithms are available in Scikit-Learn and elsewhere, but perhaps the simplest to understand is an algorithm known as k-means clustering, which is implemented in *sklearn.cluster.KMeans*.

Applications of k-means in computer vision

In my project the k-means algorithm can also be used to extract the dominant colors of an artworks like extract the top 5 colours used by Ziem Felix in his 20 more relevant artworks. However, we can also extract features for only one artwork or a style of art, only changing the dataset.

Image example of artwork of Ziem Felix (Romanticism)

Here I show you just one picture of Felix Ziem but in almost all we can see the same colors, intuitively we can distinguish the colors that predominate in this author but thanks to K-means we can get the correct colors like the following:

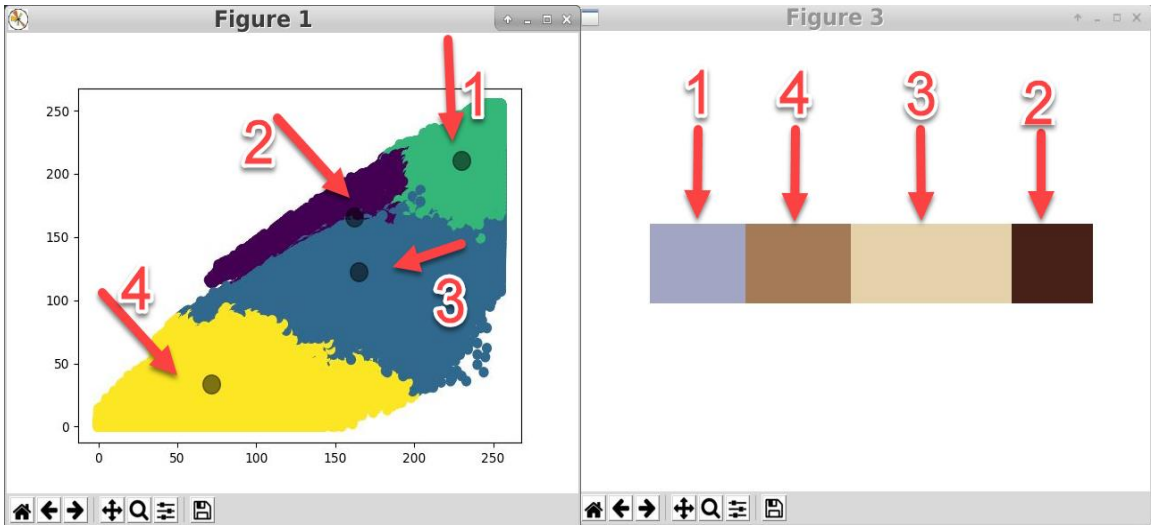


Image Top 4 Colors predominant in Felix Ziem artworks

You will think it is complicated but in a few lines of code knowing what we do with python we get it quickly. Only we need to take the raw pixel intensities of the image dataset as our data points, pass them on to k-means, and let the clustering algorithm determine the dominant colors. That's all!

Example of extract Top 3 colors in only one artwork:

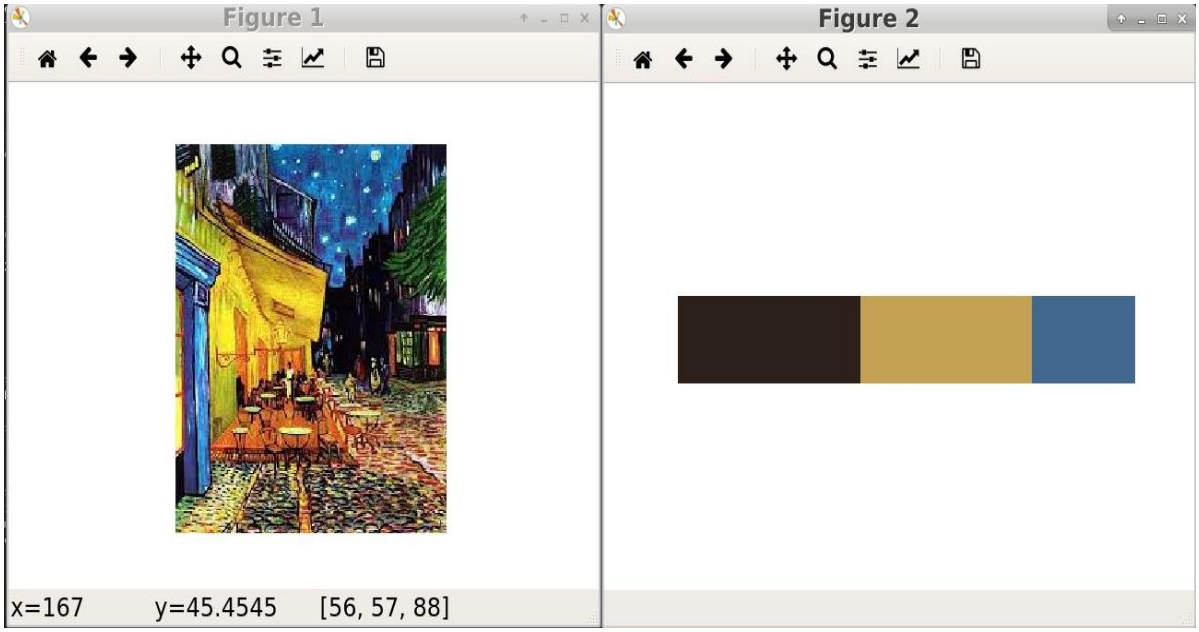


Image Vincent Van Gogh artwork Top 3 [n_clusters] colors (Cafe Terrace)

Image Salvador Dali artwork Top 5 [n_clusters] colors (Persistence of memory)

In addition, reading a little about color analysis articles, several modern artists and current psychologists have in many cases managed to understand what feelings and emotions had the artists or pictorial styles. Graphics like these explain the feelings and aptitudes of some colors and with the extraction of these colors, thanks to K-Means, we can relate them to this table.

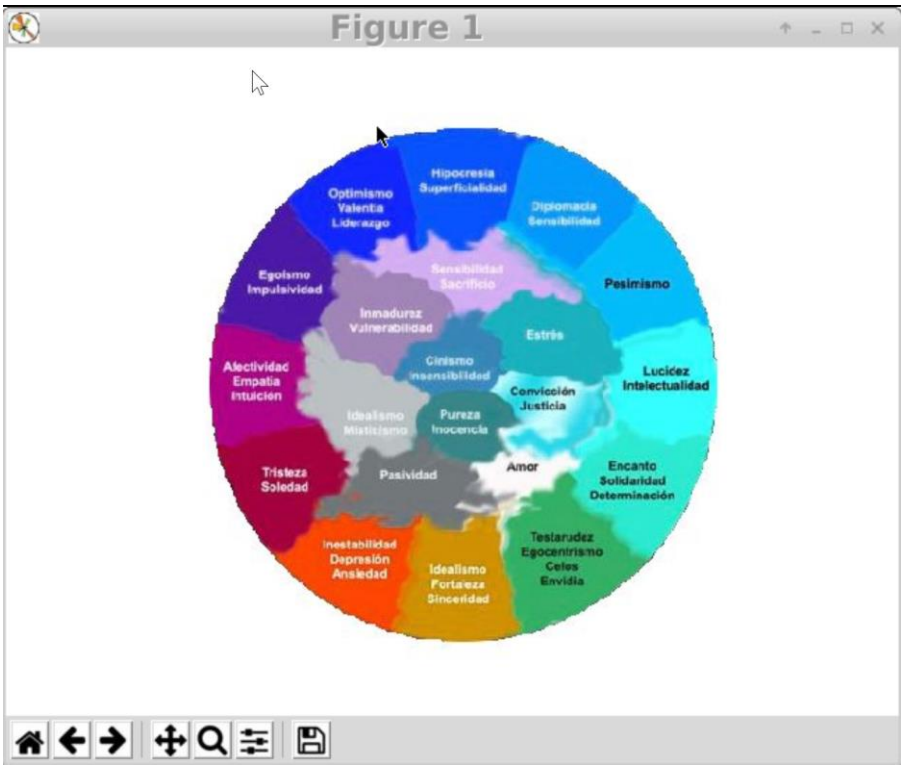


Image example plot of color emotions

However, the most popular usage of k-means in computer vision/machine learning is the bag-of-visual words (BOVW) model where we:

- Extract SIFT/SURF (or other local invariant feature vectors) from a dataset of images.
- Cluster the SIFT/SURF features to form a "codebook".
- Quantize the SIFT/SURF features from each image into a histogram that counts the number of times each "visual word" appears.

8. Classification with Bag of Visual Words

This section is one of the most important! In the previous sections we have explained several very important concepts in computer vision to understand now the processes that are needed to carry out our image classifier, specifically artworks. But before I start to explain the classifier code, I will explain above what it means Bag of Visual words.

What is BOVW?

To begin explaining what this technique consists of I would like to mention the world of natural language processing or also known as NLP, where our intention is to compare and analyze multiple documents. Each document has a lot of different words and in a certain order. With this technique we ignore the order

and simply throw the words in a "bag". Then once all the words inside the bag we can analyze the occurrences of each word. Finally, with the NLP, each document becomes a histogram of word counts and can be used as characteristics for a ML process.

Visual Words

In the section on image descriptors, explain what they are and how they are generated (SIFT, SURF). If we think of an image as a document of "words" generated by SIFT, we can extend the Bag Visual Words model to classify images instead of text documents. We can imagine that a SIFT descriptor or "Visual Word" represents an object of the picture, as for example a descriptor can be the eye of the Mona Lisa.

These descriptors of SIFT have variations so we must have some grouping method to group words that represent the same. For example, all the characteristics of the eye of the Gioconda must go to the same cluster or container.

However, the characteristics of SIFT are not as literal as saying "eye of the Gioconda". We can not agrpar them according to a human definition but mathematically. For this the descriptions of SIFT are 128-dimensional vices so we can simply make a matrix with each SIFT descriptor in our training set as its own row, and 128 columns for each of the dimensions of the SIFT characteristics. Once all this has been done, we will have to connect that matrix to an aggregating algorithm, such as the K-Means explained above.

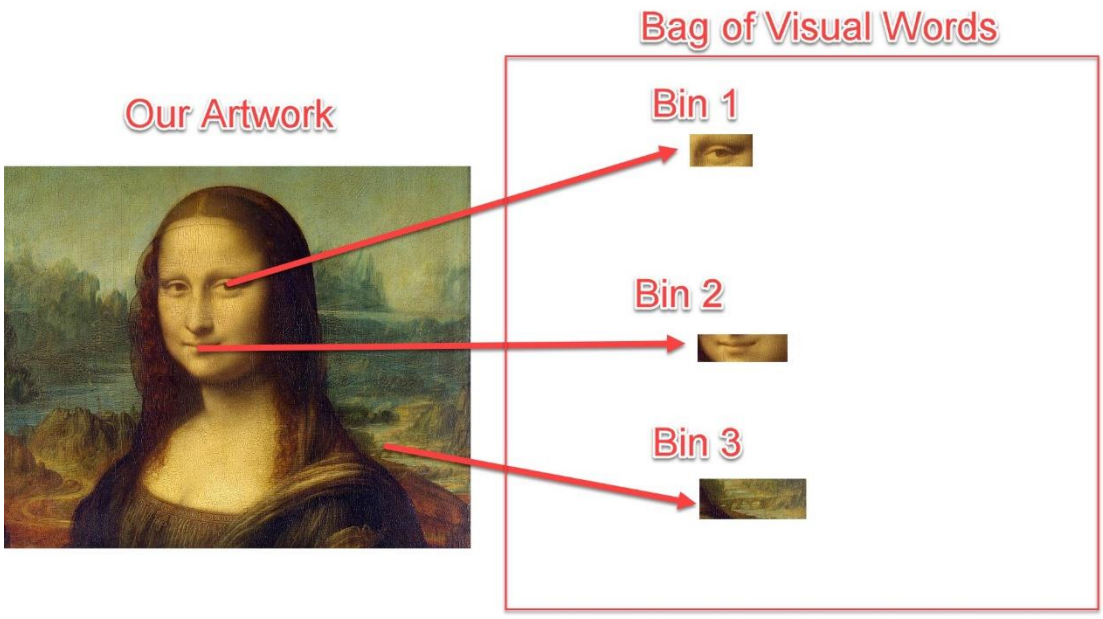


Image Example Bag Visual Words

Next we go through each individual image, and assign all of its SIFT descriptors to the bin they belong in. All the “eye” SIFT descriptors will be converted from a 128-

dimensional SIFT vector to a bin label like “eye” or “Bin number 4”. Finally we make a histogram for each image by summing the number of features for each codeword. For example, with K=3, we might get a total of 1 eye feature, 3 mouth features, and 5 bridge features for image number 1, a different distribution for image number 2, and so on.

**Note: Remember, this is just a metaphor: real SIFT feature clusters won't have such a human-meaningful definition.*

At this point we have converted images with varying numbers of SIFT features into K features. We can feed the matrix of M observations and K features into a classifier like Random Forest, AdaBoost or SVC as our X, image labels as our y, and it ought to be able to predict image labels from images with some degree of accuracy.

But how can I know how many bins I need? And what is K?

For this artworks classifier, I performed a grid search across a range of K values and compared the scores of classifiers for each K. The code is on github and references other files in the repo.

**Note: Determining what is K to use for K-Means depending on your project and how long it takes to do a grid search, you might want try differents methods.*

Prepare our classifier

Once we have understood the theory, we move on to the practical part to train our classifier and obtain our model to use it in our mobile application developed in Xamarin. The first thing we must do now is to understand each of the files that make up our solution of the object classifier.

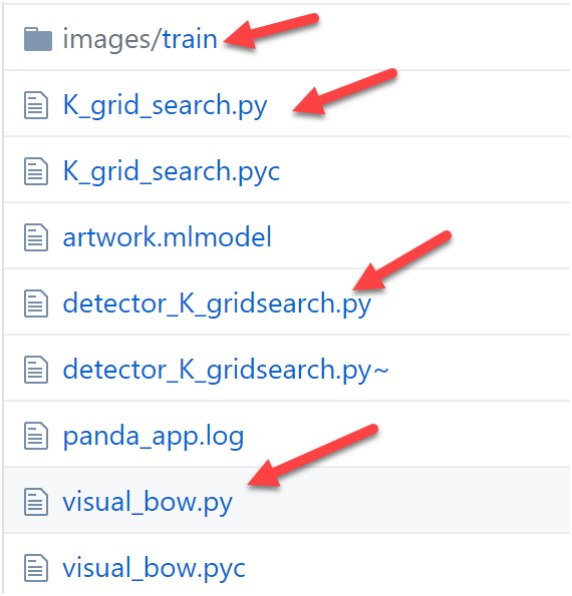


Image repository files

We can see that it consists of a folder images and three python files

- *K_grid_search.py*
- *detector_K_gridsearch.py*
- *visual_bow.py*

We will not have to modify anything of the code, it will only depend on each one the modification of the dataset to change the thematic of images to classify.

Image Dataset folder

Training

From the project directory, issue the following command to begin training:

```
python  detector_K_gridsearch.py –train_path images/train
```

If everything has been set up correctly, python will initialize the training. When training begins, it will look like this:

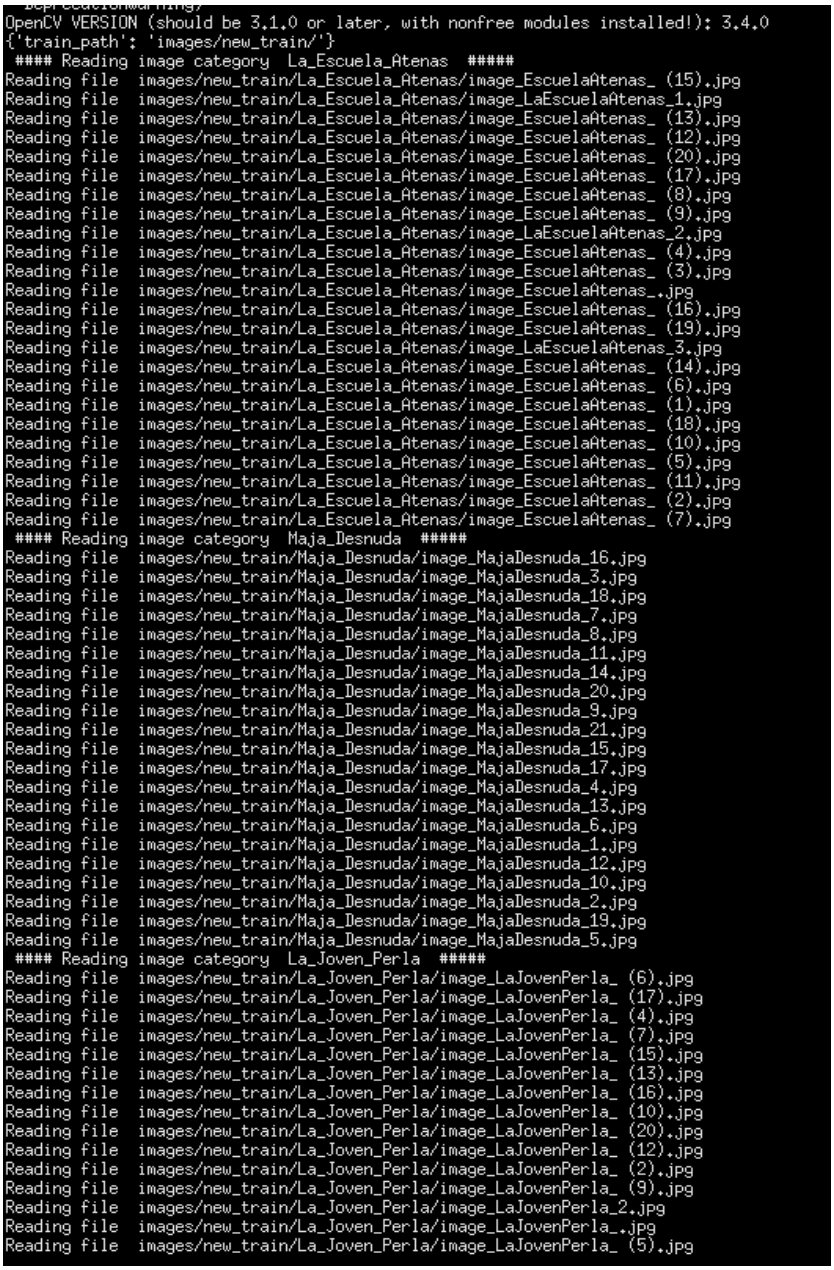


Image Reading all files of differents folders of artworks

Then after reading all the files, the program start to extract features descriptors for each image, in the image below I show you the example of Moulin Gallete Artwork, in which we see that the 21 images that we have of this picture now we have extracted their descriptors of characteristics:

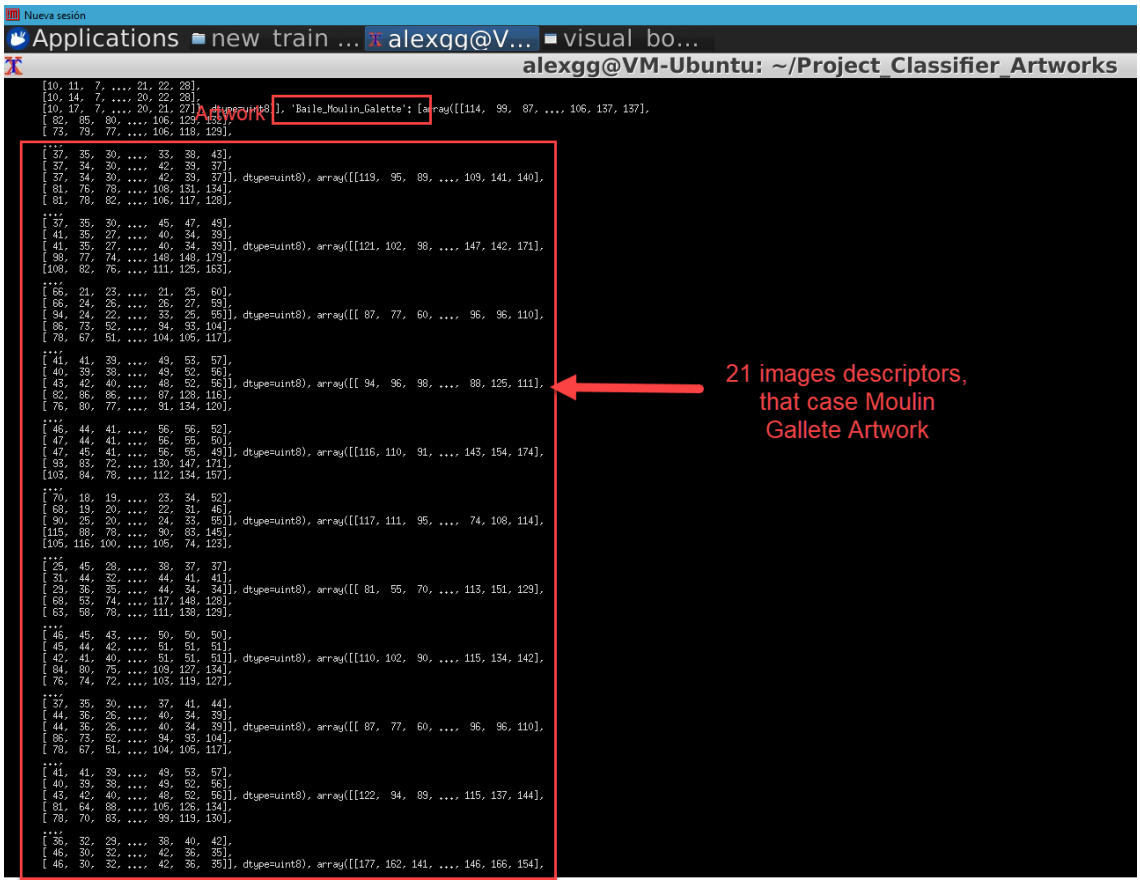


Image Array of descriptors of Moulin Gallete Artwork

```
generating SIFT descriptors for 18 artworks
Computing Features for  3Mayo
Computing Features for  Terraza_Cafe
Computing Features for  Terraza_Saint_Adresse
Computing Features for  El_Almuerzo_Remeros
Computing Features for  Mona_lisa
Computing Features for  Persistencia_Memoria
Computing Features for  La_Gitana_Dormida
Computing Features for  La_Gran_Ola_Kanagawa
Computing Features for  Los_Embajadores
Computing Features for  Noche_Estrellada
Computing Features for  Girasoles
Computing Features for  Washington_Cruzando_Delaware
Computing Features for  El_Caballero_Mano_Pecho
Computing Features for  Baile_Moulin_Galette
Computing Features for  Maja_Desnuda
Computing Features for  Las_Meninas
Computing Features for  La_Escuela_Atenas
Computing Features for  La_Joven_Perla
SIFT descriptors generated.
```

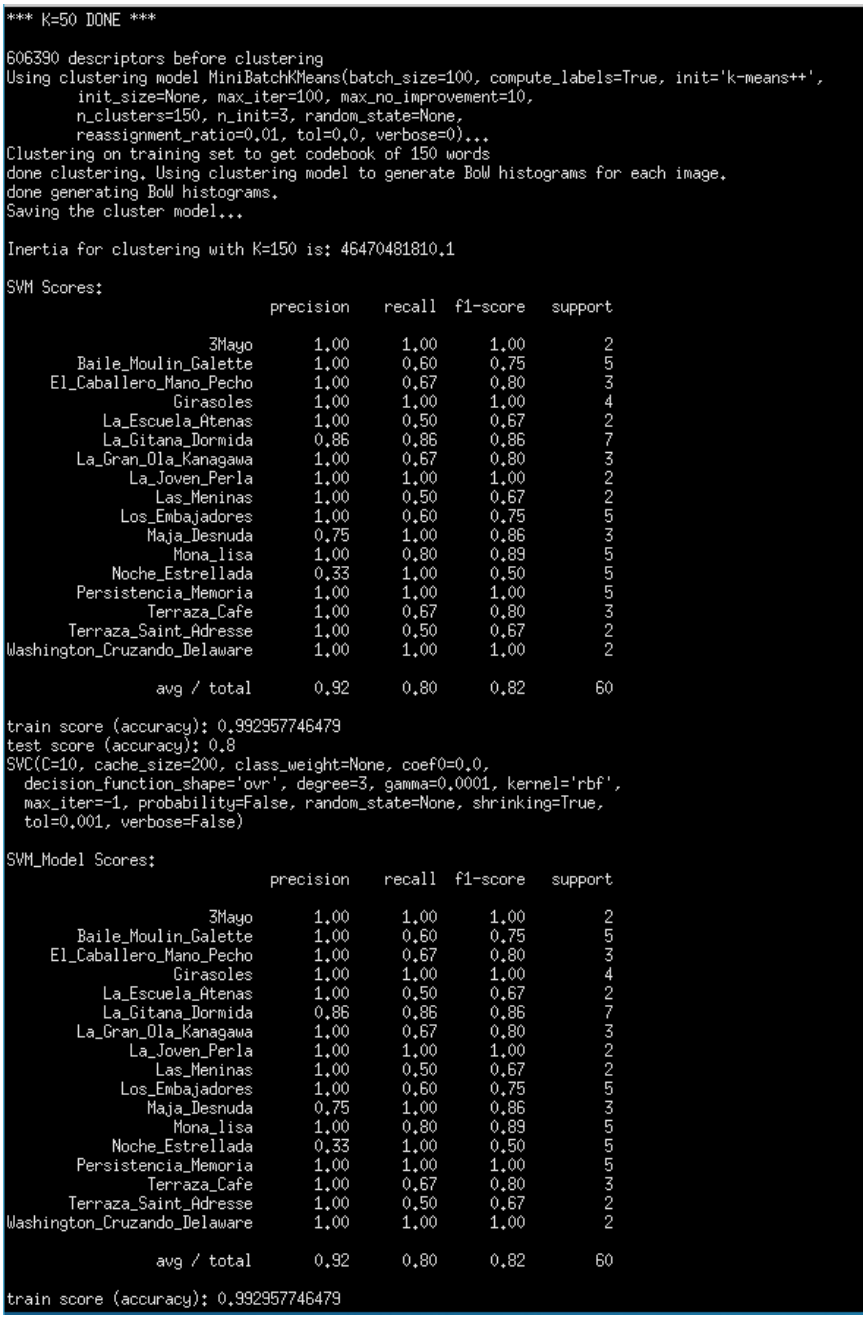
When we have finished generating the descriptors of each image, we begin to group the SIFT descriptors of each group of images to then perform the clustering and start creating our word notebook. Our codebook will increase with K, thanks to GridSearchCV we can do several tests with different values of K to see which is the most

optimal value for our bag of visual words. We will start with K = 50, K = 150, K = 300, K = 500 words.

```
SIFT descriptors generated.
404 404
Train-test-val split: 284 training rows, 60 test rows, 60 validation rows
481770 descriptors before clustering
Using clustering model MiniBatchKMeans(batch_size=100, compute_labels=True, init='k-means++',
        init_size=None, max_iter=100, max_no_improvement=10, n_clusters=50,
        n_init=3, random_state=None, reassignment_ratio=0.01, tol=0.0,
        verbose=0)...
Clustering on training set to get codebook of 50 words
```

```
Clustering on training set to get codebook of 50 words
done clustering. Using clustering model to generate BoW histograms for each image.
done generating BoW histograms.
Saving the cluster model...

Inertia for clustering with K=50 is: 51904088021.8

SVM Scores:
                          precision    recall  f1-score   support

                   3Mayo       1.00      0.50      0.67         2
       Baile_Moulin_Galette       1.00      0.60      0.75         5
    El_Caballero_Mano_Pecho       1.00      0.67      0.80         3
                Girasoles       1.00      1.00      1.00         4
          La_Escuela_Atenas       1.00      0.50      0.67         2
          La_Gitana_Dormida       0.86      0.86      0.86         7
       La_Gran_Ola_Kanagawa       1.00      0.67      0.80         3
            La_Joven_Perla       1.00      0.50      0.67         2
              Las_Meninas       0.50      0.50      0.50         2
          Los_Embajadores       1.00      0.60      0.75         5
             Maja_Desnuda       1.00      1.00      1.00         3
                Mona_lisa       1.00      0.80      0.89         5
           Noche_Estrellada       0.31      1.00      0.48         5
       Persistencia_Memoria       0.83      1.00      0.91         5
              Terraza_Cafe       1.00      0.33      0.50         3
      Terraza_Saint_Adresse       0.50      0.50      0.50         2
Washington_Cruzando_Delaware       1.00      1.00      1.00         2

              avg / total       0.88      0.75      0.77        60

train score (accuracy): 0.992957746479
test score (accuracy): 0.75
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.0001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

SVM_Model Scores:
                          precision    recall  f1-score   support

                   3Mayo       1.00      0.50      0.67         2
       Baile_Moulin_Galette       1.00      0.60      0.75         5
    El_Caballero_Mano_Pecho       1.00      0.67      0.80         3
                Girasoles       1.00      1.00      1.00         4
          La_Escuela_Atenas       1.00      0.50      0.67         2
          La_Gitana_Dormida       0.86      0.86      0.86         7
       La_Gran_Ola_Kanagawa       1.00      0.67      0.80         3
            La_Joven_Perla       1.00      0.50      0.67         2
              Las_Meninas       0.50      0.50      0.50         2
          Los_Embajadores       1.00      0.60      0.75         5
             Maja_Desnuda       0.75      1.00      0.86         3
                Mona_lisa       1.00      0.80      0.89         5
           Noche_Estrellada       0.31      1.00      0.48         5
       Persistencia_Memoria       1.00      1.00      1.00         5
              Terraza_Cafe       1.00      0.33      0.50         3
      Terraza_Saint_Adresse       0.50      0.50      0.50         2
Washington_Cruzando_Delaware       1.00      1.00      1.00         2

              avg / total       0.88      0.75      0.77        60

train score (accuracy): 0.992957746479
test score (accuracy): 0.75
Saving the trained model...
```

Image Codebook K=50



```
*** K=50 DONE ***

606390 descriptors before clustering
Using clustering model MiniBatchKMeans(batch_size=100, compute_labels=True, init='k-means++',
        init_size=None, max_iter=100, max_no_improvement=10,
        n_clusters=150, n_init=3, random_state=None,
        reassignment_ratio=0.01, tol=0.0, verbose=0)...
Clustering on training set to get codebook of 150 words
done clustering. Using clustering model to generate BoW histograms for each image.
done generating BoW histograms.
Saving the cluster model...

Inertia for clustering with K=150 is: 46470481810.1

SVM Scores:
                           precision    recall  f1-score   support

                    3Mayo       1.00      1.00      1.00         2
       Baile_Moulin_Galette       1.00      0.60      0.75         5
    El_Caballero_Mano_Pecho       1.00      0.67      0.80         3
                Girasoles       1.00      1.00      1.00         4
         La_Escuela_Atenas       1.00      0.50      0.67         2
         La_Gitana_Dormida       0.86      0.86      0.86         7
       La_Gran_Ola_Kanagawa       1.00      0.67      0.80         3
            La_Joven_Perla       1.00      1.00      1.00         2
              Las_Meninas       1.00      0.50      0.67         2
          Los_Embajadores       1.00      0.60      0.75         5
             Maja_Desnuda       0.75      1.00      0.86         3
                Mona_lisa       1.00      0.80      0.89         5
          Noche_Estrellada       0.33      1.00      0.50         5
       Persistencia_Memoria       1.00      1.00      1.00         5
              Terraza_Cafe       1.00      0.67      0.80         3
      Terraza_Saint_Adresse       1.00      0.50      0.67         2
Washington_Cruzando_Delaware       1.00      1.00      1.00         2

              avg / total       0.92      0.80      0.82        60

train score (accuracy): 0.992957746479
test score (accuracy): 0.8
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.0001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

SVM_Model Scores:
                           precision    recall  f1-score   support

                    3Mayo       1.00      1.00      1.00         2
       Baile_Moulin_Galette       1.00      0.60      0.75         5
    El_Caballero_Mano_Pecho       1.00      0.67      0.80         3
                Girasoles       1.00      1.00      1.00         4
         La_Escuela_Atenas       1.00      0.50      0.67         2
         La_Gitana_Dormida       0.86      0.86      0.86         7
       La_Gran_Ola_Kanagawa       1.00      0.67      0.80         3
            La_Joven_Perla       1.00      1.00      1.00         2
              Las_Meninas       1.00      0.50      0.67         2
          Los_Embajadores       1.00      0.60      0.75         5
             Maja_Desnuda       0.75      1.00      0.86         3
                Mona_lisa       1.00      0.80      0.89         5
          Noche_Estrellada       0.33      1.00      0.50         5
       Persistencia_Memoria       1.00      1.00      1.00         5
              Terraza_Cafe       1.00      0.67      0.80         3
      Terraza_Saint_Adresse       1.00      0.50      0.67         2
Washington_Cruzando_Delaware       1.00      1.00      1.00         2

              avg / total       0.92      0.80      0.82        60

train score (accuracy): 0.992957746479
```

Image Codebook K=150

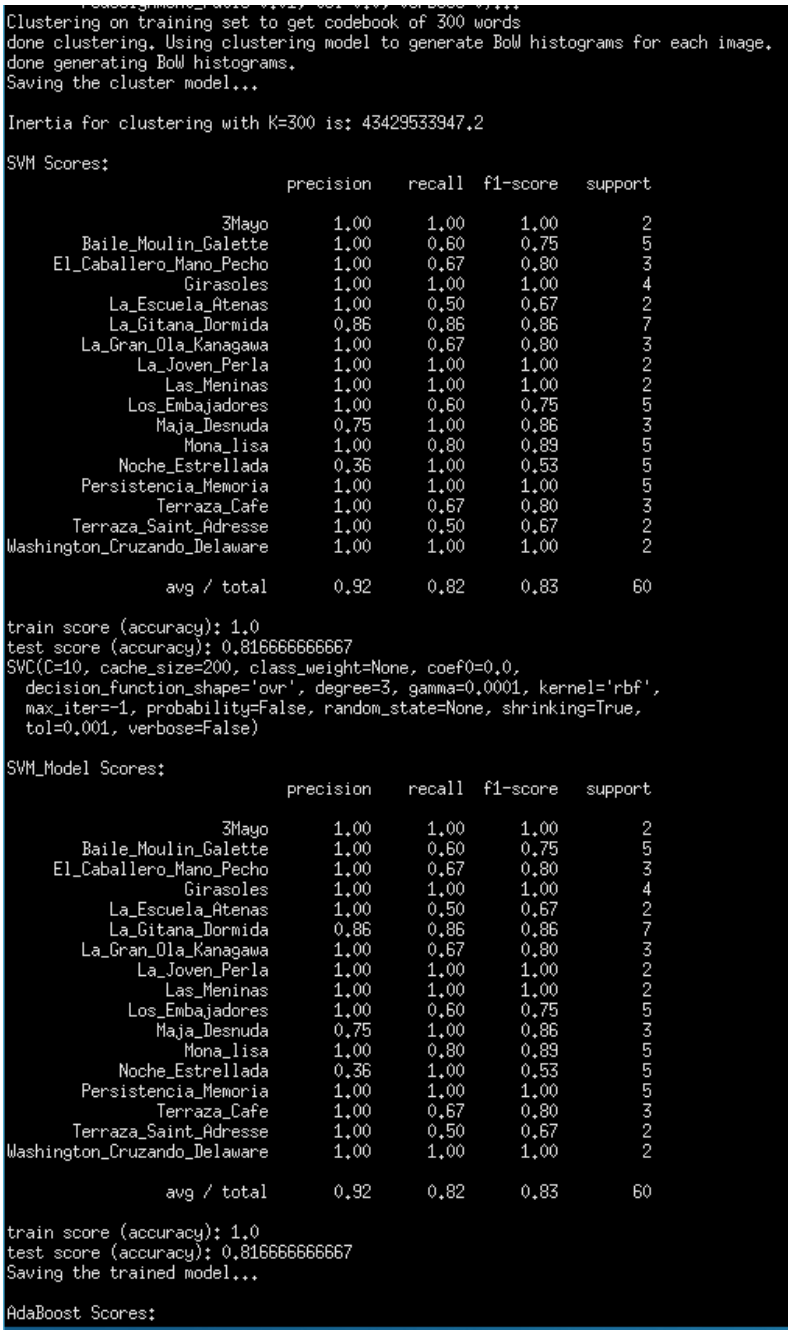


Image Codebook K=300

```
606390 descriptors before clustering
Using clustering model MiniBatchKMeans(batch_size=100, compute_labels=True, init='k-means++',
       init_size=None, max_iter=100, max_no_improvement=10,
       n_clusters=500, n_init=3, random_state=None,
       reassignment_ratio=0.01, tol=0.0, verbose=0)...
Clustering on training set to get codebook of 500 words
done clustering. Using clustering model to generate BoW histograms for each image.
done generating BoW histograms.
Saving the cluster model...

Inertia for clustering with K=500 is: 41399274297.2

SVM Scores:
                          precision    recall  f1-score   support

                   3Mayo       1.00      1.00      1.00         2
       Baile_Moulin_Galette       1.00      0.60      0.75         5
    El_Caballero_Mano_Pecho       1.00      0.67      0.80         3
                Girasoles       1.00      1.00      1.00         4
          La_Escuela_Atenas       1.00      0.50      0.67         2
          La_Gitana_Dormida       0.86      0.86      0.86         7
       La_Gran_Ola_Kanagawa       1.00      0.67      0.80         3
            La_Joven_Perla       1.00      1.00      1.00         2
               Las_Meninas       0.67      1.00      0.80         2
           Los_Embajadores       1.00      0.80      0.89         5
              Maja_Desnuda       0.75      1.00      0.86         3
                 Mona_lisa       1.00      0.80      0.89         5
          Noche_Estrellada       0.42      1.00      0.59         5
       Persistencia_Memoria       1.00      1.00      1.00         5
              Terraza_Cafe       1.00      0.67      0.80         3
      Terraza_Saint_Adresse       1.00      0.50      0.67         2
Washington_Cruzando_Delaware       1.00      1.00      1.00         2

               avg / total       0.91      0.83      0.84        60

train score (accuracy): 1.0
test score (accuracy): 0.833333333333
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.0001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

SVM_Model Scores:
                          precision    recall  f1-score   support

                   3Mayo       1.00      1.00      1.00         2
       Baile_Moulin_Galette       1.00      0.60      0.75         5
    El_Caballero_Mano_Pecho       1.00      0.67      0.80         3
                Girasoles       1.00      1.00      1.00         4
          La_Escuela_Atenas       1.00      0.50      0.67         2
          La_Gitana_Dormida       1.00      0.86      0.92         7
       La_Gran_Ola_Kanagawa       1.00      0.67      0.80         3
            La_Joven_Perla       1.00      1.00      1.00         2
               Las_Meninas       0.67      1.00      0.80         2
           Los_Embajadores       1.00      0.80      0.89         5
              Maja_Desnuda       0.60      1.00      0.75         3
                 Mona_lisa       1.00      0.80      0.89         5
          Noche_Estrellada       0.42      1.00      0.59         5
       Persistencia_Memoria       1.00      1.00      1.00         5
              Terraza_Cafe       1.00      0.67      0.80         3
      Terraza_Saint_Adresse       1.00      0.50      0.67         2
Washington_Cruzando_Delaware       1.00      1.00      1.00         2

               avg / total       0.92      0.83      0.85        60

train score (accuracy): 1.0
test score (accuracy): 0.833333333333
Saving the trained model...
```

Image Codebook K=500

```
* * *
Scored grid search with metric: "accuracy"
For K = 50:     SVM 0.800000    AdaBoost 0.100000       K-Means Inertia 52713981541.140625
For K = 150:    SVM 0.850000    AdaBoost 0.050000       K-Means Inertia 47210966943.468750
For K = 300:    SVM 0.883333    AdaBoost 0.066667       K-Means Inertia 44197771739.828125
For K = 500:    SVM 0.916667    AdaBoost 0.050000       K-Means Inertia 42118644821.703125
(gurus) alexgg@VM-Ubuntu:~/Project_Classifier_Artworks$
```

Image Comparison result of GridSearchCV for value "K" beetween SVM and AdaBoost

At the end of the process we will automatically save, thanks to GridSearchCV, the best SVM model in this case, which gives us a result of accuracy 0.91 and also save the model of our cluster. Two very necessary files for our API.

9. Python App on Azure Web App with Flask

I try to explain you several steps that you must do it to build succesfully your Python Web App with Flask on Azure:

1. Look this doc: https://docs.microsoft.com/es-es/visualstudio/python/publishing-python-web-applications-to-azure-from-visual-studio
2. Look this doc: https://docs.microsoft.com/es-es/visualstudio/python/managing-python-on-azure-app-service

These docs will help you to do your build on azure succesfully, I attach you an image of my Flask API solution structure. But first, we need to create in visual studio a project with Flask

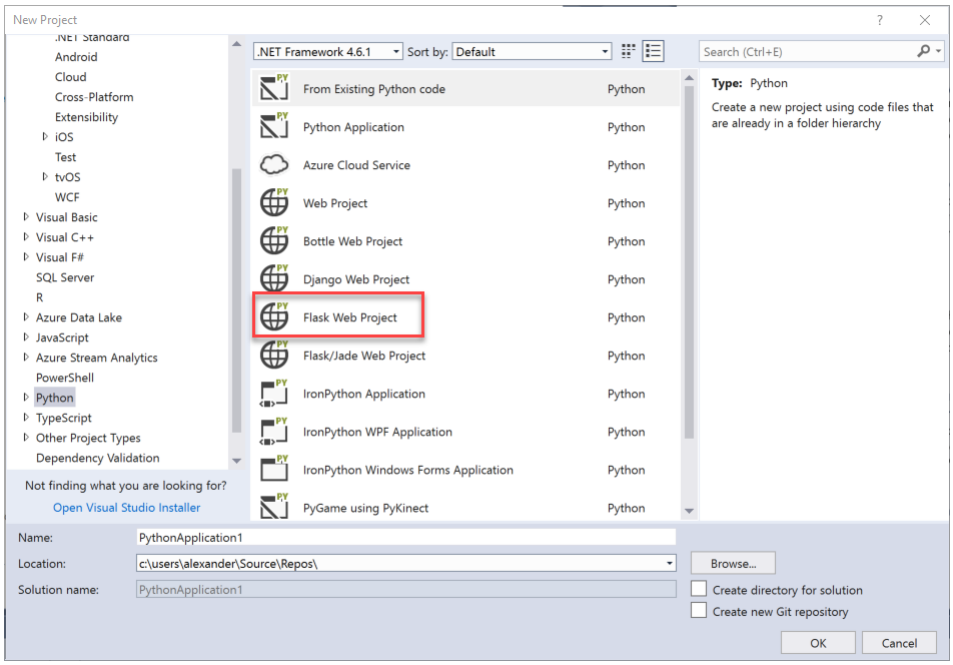


Image Flask Web Project template

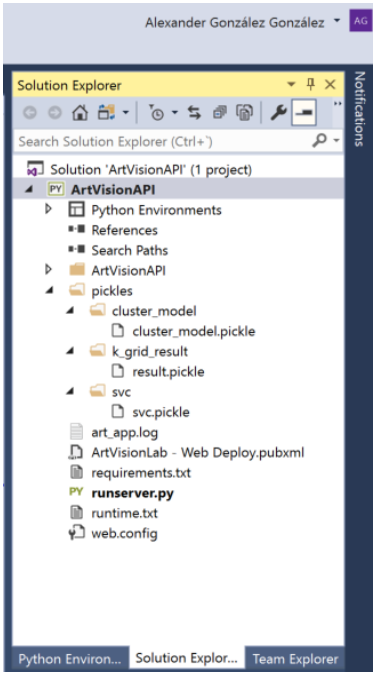


Image API Solution files

As you can see, we have several files but many of them we only need for the start-up of our solution in our azure web app. If you look, we have a folder called "*pickle*" where we have the model of our classifier made by the SVM algorithm and we also have a model created by Bag of Visual Words technique with K-Means Clustering.

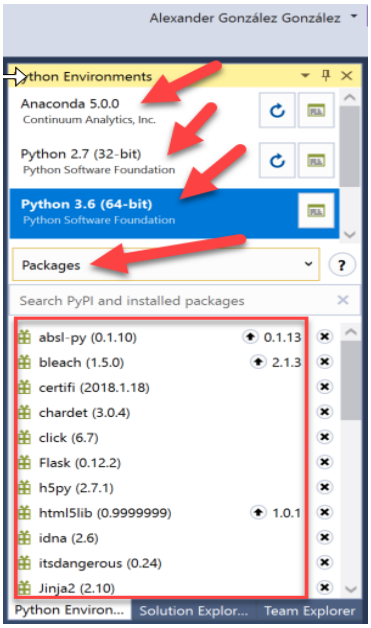


Image Python environment in VS17

In addition, we can configure our solution in "Python Environment option" and add the version of python that we want from visual studio itself and add the libraries that are important for our project, such as:

requirements.txt | runserver.py

```
absl-py==0.1.10
bleach==1.5.0
certifi==2018.1.18
chardet==3.0.4
click==6.7
Flask==0.12.2
html5lib==0.9999999
idna==2.6
itsdangerous==0.24
Jinja2==2.10
joblib==0.11
Markdown==2.6.11
MarkupSafe==1.0
numpy==1.14.1
opencv-contrib-python==3.4.0.12
pip==9.0.1
protobuf==3.5.1
PyYAML==3.12
redis==2.10.6
requests==2.18.4
scikit-learn==0.19.1
scipy==1.0.0
setuptools==28.8.0
six==1.11.0
sklearn==0.0
urllib3==1.22
Werkzeug==0.14.1
wheel==0.30.0
```

Image requirements.txt file

Do not worry about creating the requirement.txt file because you have everything in the [github repository](). Once explained this you have a series of images where I show you the web app in Azure and the options that we must activate before uploading our solution to Azure.

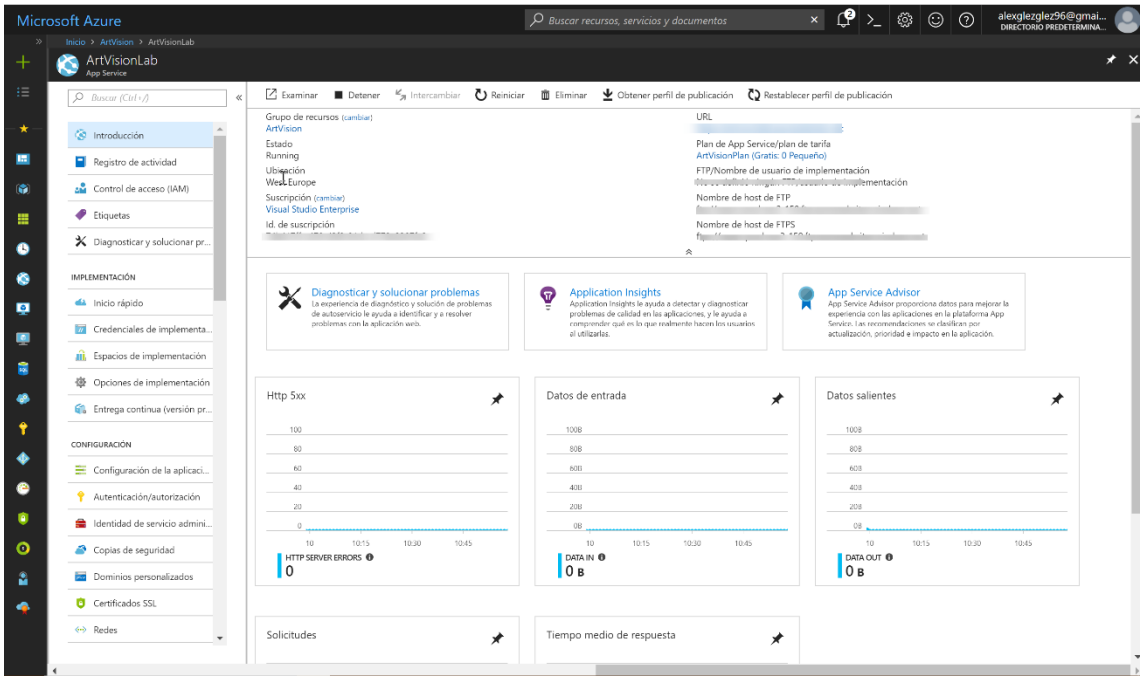


Image Azure Python Web App with Flask

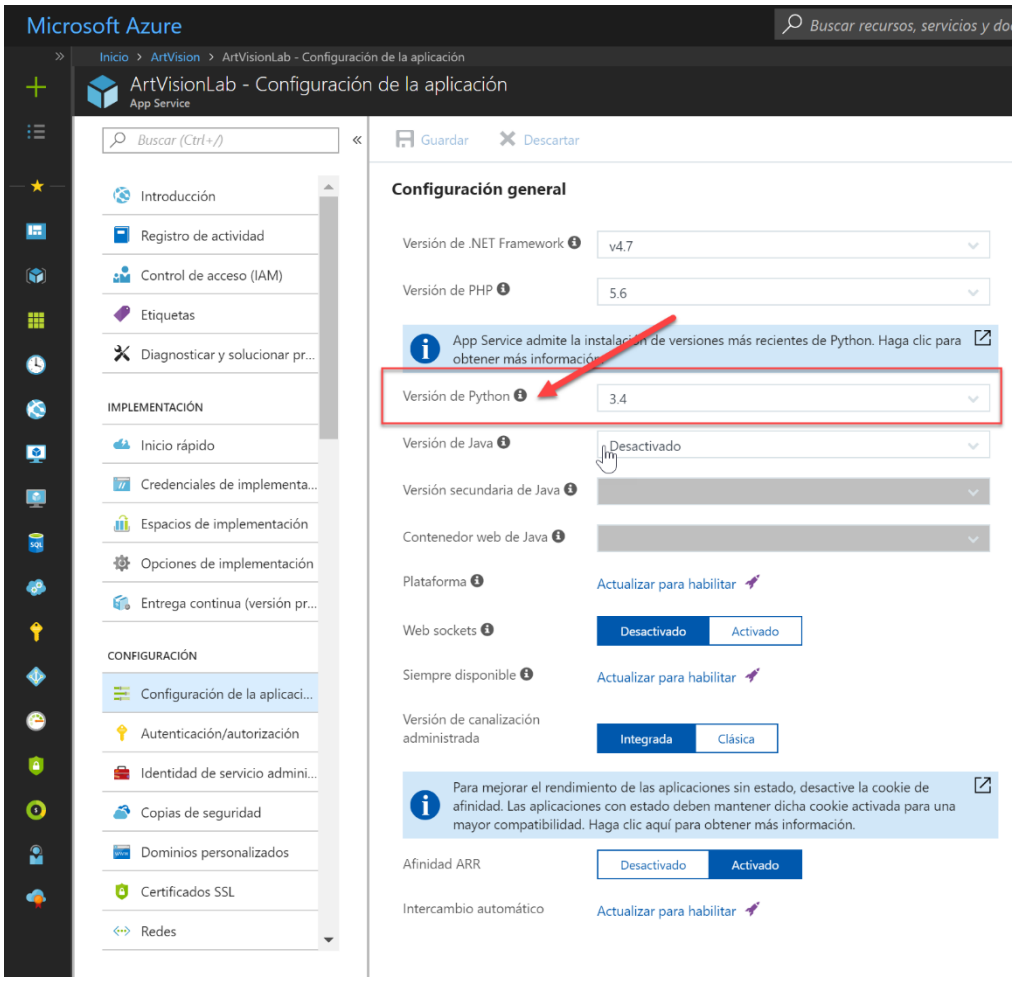


Image set up python environment version

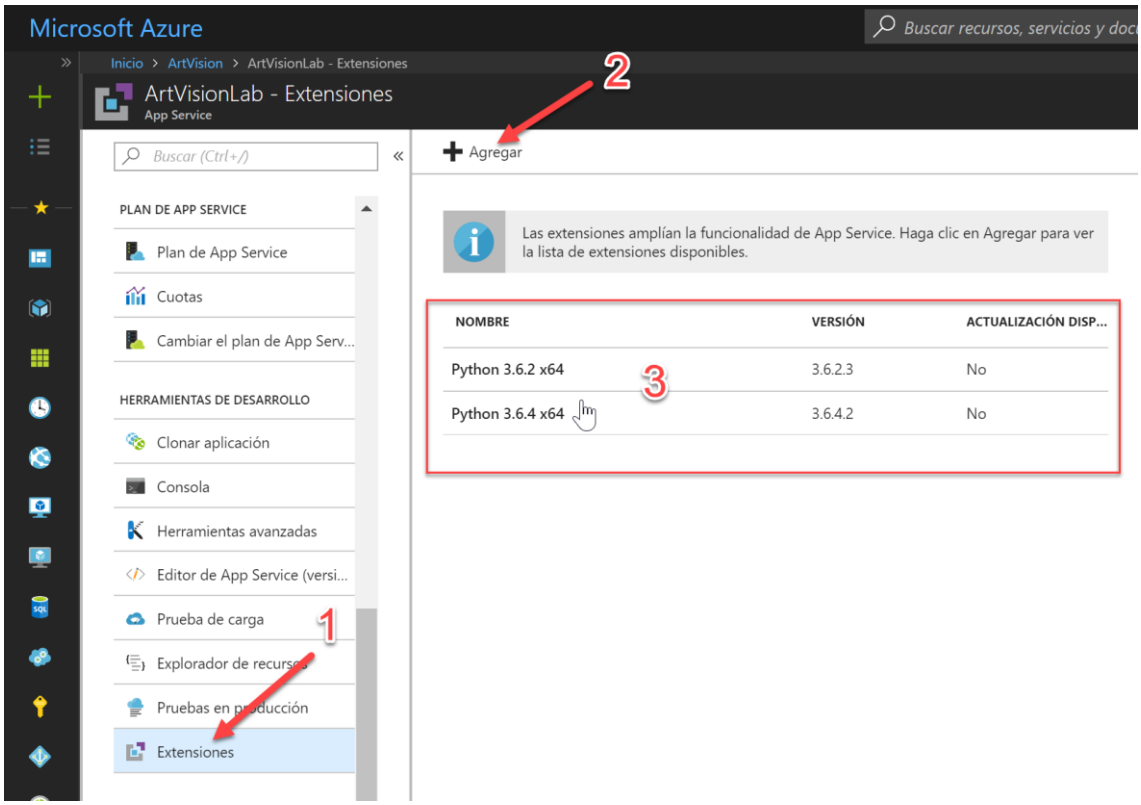


Image Add pytho to extension to our web app

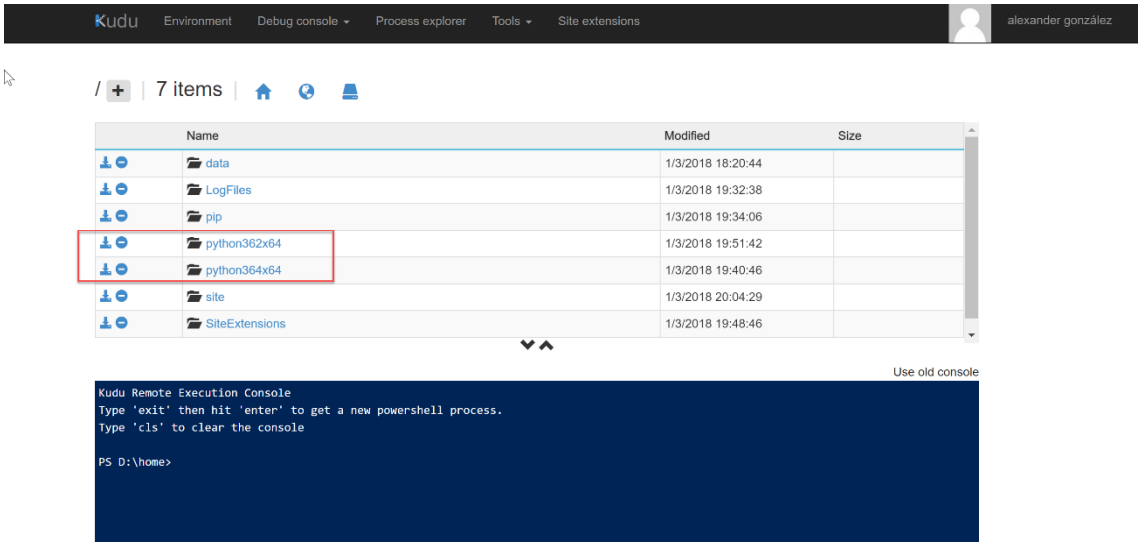


Image Location of python extension with Kudu

As I commented before to correctly upload our solution to azure we must create a series of files, like the runtime.txt. In that one you only need to write: python-3.4 for example or your version that you wanna use.

```
python-3.4
```

Another one file is runserver.py, is our index.html, you only need to change in your web.config in this line:

```
<add key="WSGI_HANDLER" value="runserver.app"/>
```

**Note: Remember that you need to set the file with .app extension.*

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This template is configured to use Python 3.5 on Azure App Service. To use a different version of Python,
or to use a hosting service other than Azure, replace the scriptProcessor path below with the path given
to you by wfastcgi-enable or your provider.

For Python 2.7 on Azure App Service, the path is
"D:\home\Python27\python.exe|D:\home\Python27\wfastcgi.py"
```

```
The WSGI_HANDLER variable should be an importable variable or function (if followed by '()') that returns
your WSGI object.

See https://aka.ms/PythonOnAppService for more information.
-->
<configuration>
  <appSettings>
    <add key="PYTHONPATH" value="D:\home\site\wwwroot"/>
    <!-- The handler here is specific to Bottle; other frameworks vary. -->
    <add key="WSGI_HANDLER" value="runserver.app"/>
    <add key="WSGI_LOG" value="D:\home\LogFiles\wfastcgi.log"/>
  </appSettings>
  <system.webServer>
    <httpErrors errorMode="Detailed"></httpErrors>
    <handlers>
      <add name="PythonHandler" path="*" verb="*" modules="FastCgiModule"

scriptProcessor="D:\home\Python362x64\python.exe|D:\home\Python362x64\wfastcgi.py"
          resourceType="Unspecified" requireAccess="Script"/>
    </handlers>
  </system.webServer>
</configuration>
```

Finally, you only need to add another web config like this image in a ProyectNameFolder/static/web.config and add lines of code like the image below:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This template removes any existing handler so that the default handlers will be used for this directory.
Only handlers added by the other web.config templates, or with the name set to PythonHandler, are removed.

See https://aka.ms/PythonOnAppService for more information.
-->
<configuration>
  <system.webServer>
    <handlers>
      <remove name="PythonHandler"/>
    </handlers>
  </system.webServer>
</configuration>
```

When you have all these files correct in VS you need to publish and go to your azure web app and enter in development tools/Extension and install in your web app the env of Python that you would like to use.

Then go to kudu. In the second link that I attach you it explains you very well how to finish your deployment, you need to install in your web app via Kudu all requirements in your requirements.txt. You only need to locate your Python env that you were install previously in your web app, in my case in kudu the location is:

```
D:\home\Python362x64\python.exe
```

In these lines of code, you can see that I start to import of libraries to my solution:

```
from flask import Flask, request, render_template, jsonify
from werkzeug import secure_filename
import logging
import sys
import cv2
import numpy as np
from sklearn.externals import joblib
import os
import pickle
import json
```

Here I start to declare variables, path to my models (SVM and cluster model) and console messages to see in execution:

```
app.logger.info('\n\n* * *\n\nOpenCV version is %s. should be at least 3.1.0, with nonfree
installed.' % cv2.__version__)

app = Flask(__name__)

ALLOWED_EXTENSIONS = ['png', 'jpg', 'jpeg']

APP_DIR = os.path.dirname(os.path.realpath(__file__))

MAX_PIXEL_DIM = 2000

PICKLE_DIR = os.path.abspath(
    os.path.join(
    APP_DIR,
    './pickles/'))

LOG_PATH = os.path.abspath(os.path.join(APP_DIR,'../art_app.log'))

logging.basicConfig(filename=LOG_PATH,level=logging.DEBUG,
    format='%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]')

app.logger.setLevel(logging.DEBUG)
```

In these lines of code, I write the sentences necessary to get my models in their respectives path of my solution:

```
# TODO: make pickles for kmeans and single best classifier.
filename = os.path.join(PICKLE_DIR, 'svc\\svc.pickle')
model_svc = pickle.load(open(filename, 'rb'), encoding='latin1')
```

```
filename = os.path.join(PICKLE_DIR, 'cluster_model\\cluster_model.pickle')
clust_model = pickle.load(open(filename, 'rb'), encoding='latin1')

cluster_model = clust_model
clf = model_svc
```

Function to detect extension of input files of my API:

```
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

# upload image with curl using:
# curl -F 'file=@/home/' 'http://127.0.0.1:5000/'
```

Function to resize the input files:

```
def img_resize(img):
    height, width, _ = img.shape
    if height > width:
        # too tall
        resize_ratio = float(MAX_PIXEL_DIM)/height
    else:
        # too wide, or a square which is too big
        resize_ratio = float(MAX_PIXEL_DIM)/width

    dim = (int(resize_ratio*width), int(resize_ratio*height))

    resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
    app.logger.debug('resized to %s' % str(resized.shape))

    return resized
```

This is the more complex function cause here when I get the file, resize them we need to get features of the picture and do an inference to our cluster model, and finally get a result that we send to prediction function, the one in charge of making the inference to our SVM model:

```
def img_to_vect(img_np):
    """
    Given an image path and a trained clustering model (eg KMeans),
    generates a feature vector representing that image.
    Useful for processing new images for a classifier prediction.
    """
    # img = read_image(img_path)
    height, width, _ = img_np.shape
    app.logger.debug('Color image size - H:%i, W:%i' % (height, width))
    if height > MAX_PIXEL_DIM or width > MAX_PIXEL_DIM:
        img_np = img_resize(img_np)
```

```
    gray = cv2.cvtColor(img_np, cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    kp, desc = sift.detectAndCompute(gray, None)

    clustered_desc = cluster_model.predict(desc)
    img_bow_hist = np.bincount(clustered_desc, minlength=cluster_model.n_clusters)

    # reshape to an array containing 1 array: array[[1,2,3]]
    # to make sklearn happy (it doesn't like 1d arrays as data!)
    return img_bow_hist.reshape(1,-1)
```

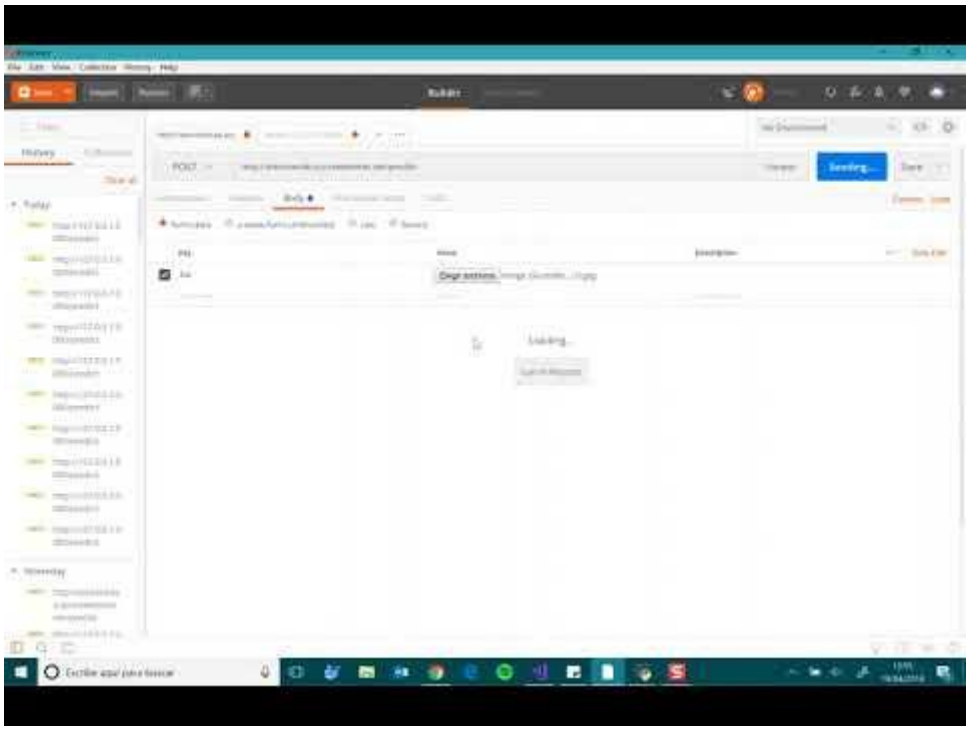
Finally, prediction function gets the result of our img_to_vect function and inference our svm model and return the result to our client:

```
def prediction(img_str):
    nparr = np.fromstring(img_str, np.uint8)
    img_np = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    # convert to K-vector of codeword frequencies
    img_vect = img_to_vect(img_np)
    prediction = clf.predict(img_vect)
    return prediction[0]
```

And here's our unique API function:

```
@app.route('/predict', methods=['POST'])
def home():
    if request.method == 'POST':
        f = request.files['file']
        if f and allowed_file(f.filename):
            filename = secure_filename(f.filename)
            app.logger.debug('got file called %s' % filename)
            lb = prediction(f.read())
            print (lb)
            json_result = json.dumps(lb.decode("utf-8"))
            return json_result
        return 'Error. Something went wrong.'
    else:
        return render_template('img_upload.jnj')


if __name__=="__main__":
    app.run(debug=True)
```

Demo with Postman (Azure Web App and local):



10. Create the mobile app on Azure Mobile App with Xamarin

La aplicación de Xamarin que he desarrollado no esta actualizada a .Net Standard, pero no creo que de problemas con la estrategia de PCL (Portable Class Library) 😉. La solución es muy sencilla, unicamente contaremos con una única pagina de interfaz de usuario y añadiremos por consiguiente los controles necesarios de los botones, que serán elegir imagen del carrete, tomar foto y clasificar.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:CustomVision"
             x:Class="CustomVision.MainPage">
    <StackLayout Padding="20">
        <Image x:Name="Img" Source="ic_launcher.png" MinimumHeightRequest="100"
MinimumWidthRequest="100"></Image>
        <Button Text="Elegir imagen" Clicked="ElegirImage"></Button>
        <Button Text="Sacar foto" Clicked="TomarFoto"></Button>
        <Button Text="Clasificar" Clicked="Clasificar"></Button>
        <Label x:Name="ResponseLabel"></Label>
        <ProgressBar x:Name="Accuracy" HeightRequest="20"/>
    </StackLayout>
</ContentPage>
```

Ademas en esta misma pagina de xaml tendremos nuestro manejadores de eventos o funciones de cada botón:

```
using Newtonsoft.Json;
```

```
using Plugin.Media;
using Plugin.Media.Abstractions;
using System;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using Xamarin.Forms;

namespace CustomVision
{
    public partial class MainPage : ContentPage
    {
        public const string ServiceApiUrl = "https://YOUR_WEB_APP_URL ";
        private MediaFile _foto = null;
        public MainPage()
        {
            InitializeComponent();
        }

        private async void ElegirImage(object sender, EventArgs e)
        {
            await CrossMedia.Current.Initialize();

            _foto = await Plugin.Media.CrossMedia.Current.PickPhotoAsync(new
PickMediaOptions());
            Img.Source = FileImageSource.FromFile(_foto.Path);
        }

        private async void TomarFoto(object sender, EventArgs e)
        {
            await CrossMedia.Current.Initialize();

            if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)
            {
                return;
            }

            var foto = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions()
            {
                PhotoSize = PhotoSize.Custom,
                CustomPhotoSize = 10,
                CompressionQuality = 92,
                Name = "image.jpg"
            });

            _foto = foto;

            if (_foto == null)
                return;

            Img.Source = FileImageSource.FromFile(_foto.Path);
```

```
    }

    private async void Clasificar(object sender, EventArgs e)
    {
        using (Acr.UserDialogs.UserDialogs.Instance.Loading("Clasificando..."))
        {
            if (_foto == null) return;

            var httpClient = new HttpClient();
            var url = ServiceApiUrl;
            var requestContent = new MultipartFormDataContent();
            var content = new StreamContent(_foto.GetStream());

            content.Headers.ContentType =
                MediaTypeHeaderValue.Parse("image/jpg");

            requestContent.Add(content, "file", "image.jpg");

            var response = await httpClient.PostAsync(url, requestContent);

            if (!response.IsSuccessStatusCode)
            {
                Acr.UserDialogs.UserDialogs.Instance.Toast("Hubo un error en la deteccion...");
                return;
            }

            var json = await response.Content.ReadAsStringAsync();

            var prediction = JsonConvert.DeserializeObject<string>(json);
            if (prediction == null)
            {
                Acr.UserDialogs.UserDialogs.Instance.Toast("Image no reconocida.");
                return;
            }
            ResponseLabel.Text = $"{prediction}";
            //Accuracy.Progress = p.Probability;
        }

        Acr.UserDialogs.UserDialogs.Instance.Toast("Clasificacion terminada...");
    }
}
```

Demo Xamarin Forms App

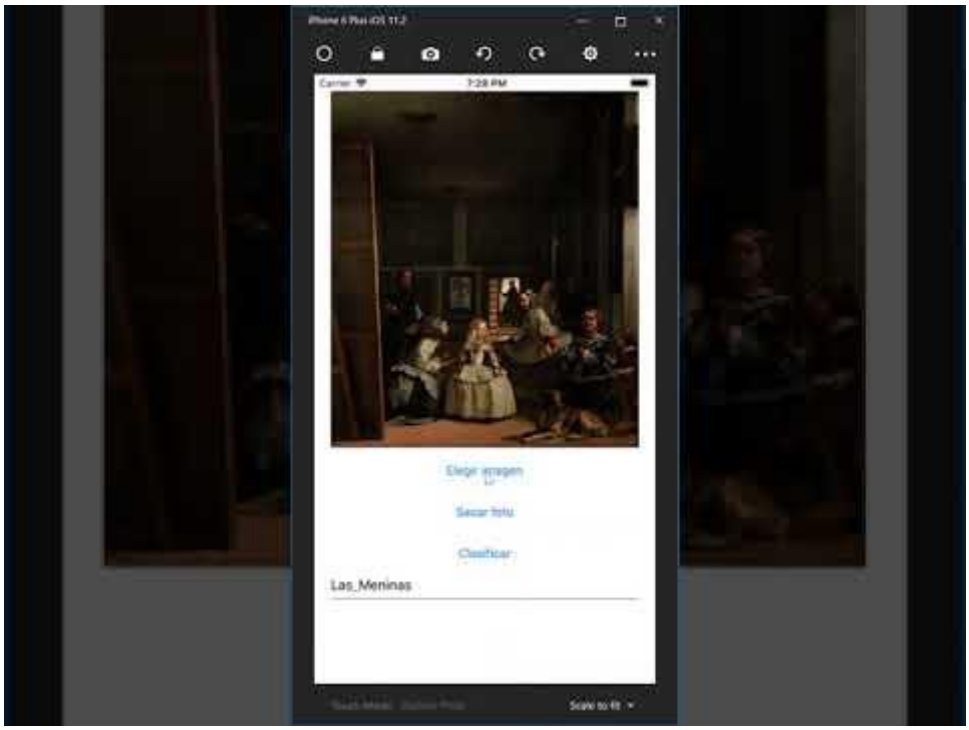


Image Test mobile app on iOS 11.2 (iPhone 6 Plus)

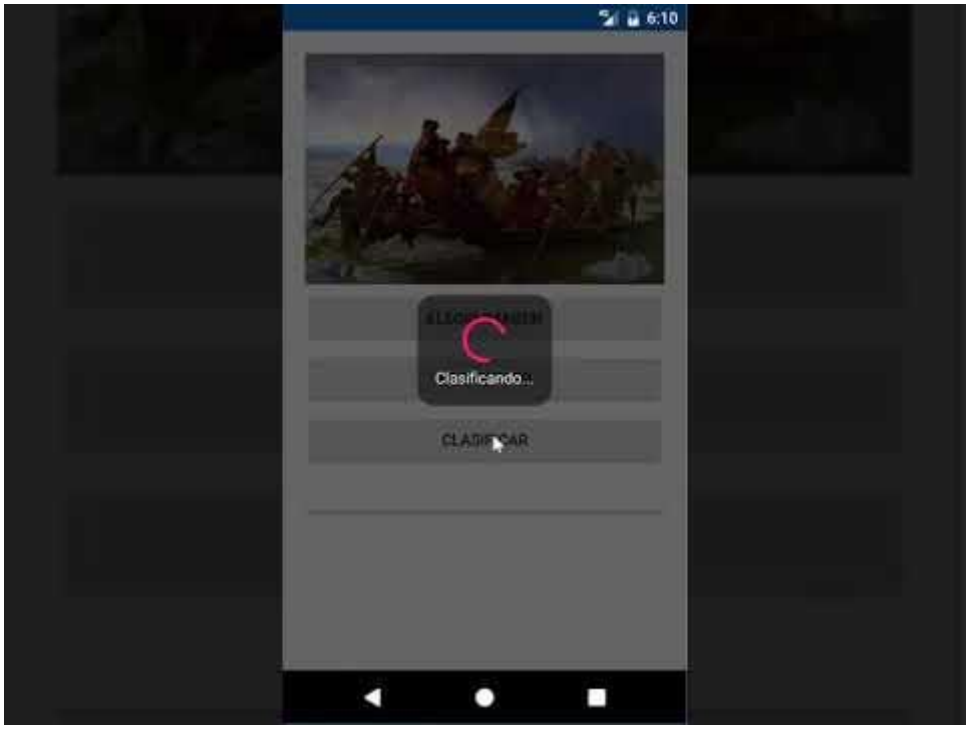


Image Test mobile app on Android 7.1 Nougat or 8.0 Oreo (Nexus -API 25-26)

11. Next Post!

Well, we have finished with the third post. Honestly, I thought that this post was going to be smaller but while I was writing it I saw that there was a lack of information to

understand key concepts when carrying out projects or computer vision solutions. I hope these explanations and sample demonstrations have been very helpful and that you have shown them a very beautiful area such as computer vision, have the possibility to manipulate the images and extract information from them.

In the next post, we will dedicate to do a comparison between the use of TensorFlow and CNTK, the results obtained, the management and usability, points of interest and final conclusions about the research that has been carried out for the development of all the posts.

Kind regards,
Alexander González (@GlezGlez96)
Microsoft Student Partner