

ISTANBUL AYDIN UNIVERSITY

PROGRESS REPORT

**AI TRAFFIC SIGNS RECOGNITION**

ABDEL AZIZ KAMO MEGNA (Y2013.140009)

Supervisor: Dr HALUK GUMUSKAYA

24th January, 2021

1. Summary of Weekly Studies

First week

Task 1: Choice of subject and search and read for scientific articles related to this subject

* AI signs recognition: it is our default project given by the Doctor.

**Second** week

Task 2: Review of the Course Project

* Download the database German Traffic Sign Recognition Benchmark (GTSRB) dataset, which contains thousands of images of 43 different kinds of road signs from http://www.gumuskaya.com/gtsrb.zip and unzip it
* Understanding the dataset
* Download the distribution code from http://www.gumuskaya.com/traffic.zip and unzip it

Third week

Task 3: Implementation

* Understanding the implementation of the code
* Install the requirements tolds: opencv-python for image processing, scikit-learn for ML-related functions and tensorflow for neural networks.
* Reading for the tensorflow and opencv documentation
* Complete the implementation of load\_data and get\_model functions in traffic.py following the specifications given by the Advisor.

**Fourth** week

Task 4: Code for learn\_signs.py

* Import of libraries
* Set parameters
* Import of the images
* Read CSV file(labelFile)
* Display some samples images of all the classes and a bar chart showing number of images for each category
* Processing the images
* Convolutional neural networks model
* Train the model and plot the result
* Save the model

**Fifth** week

Task 5: Code for recognize.py

* Import of libraries
* Setup the video camera
* Load our model
* Write getClassName function
* Predict image

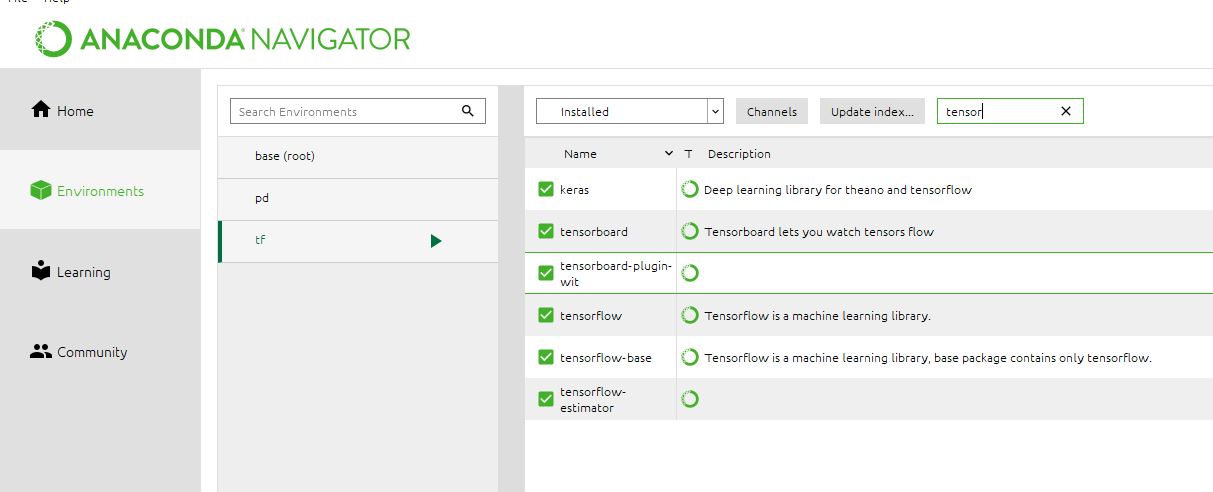
**Sixth** week

Task 6: writhing papers and video

* Project paper
* Progress report
* Video record

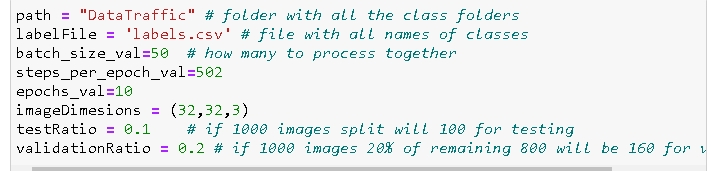
1. Analysis, Design and Implementation Records

For installing our different dependences, we created a new environment calls ‘‘tf’’ in which we done it.

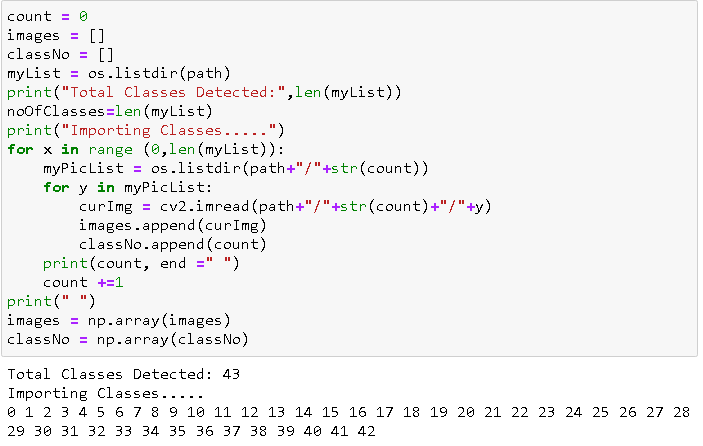


**Implementation of learning\_signs.py**

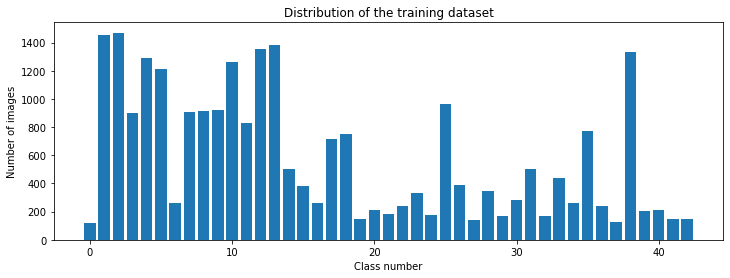
After setup the parameters of our program such as the dataset path, label file path, epochs\_val which is the iteration for our training phase, testRadio=0.1 (10%) which represent the number of data that will be used for testing, validationRtio=0.2 (20%) for the validation and the rest of 70% for training.



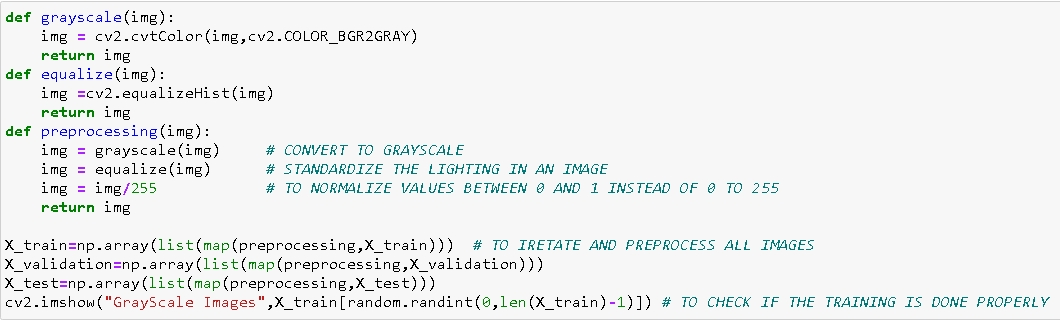
Then we import the images, the cod will automatically detect how many classes they are and t will put them all in one matrix.



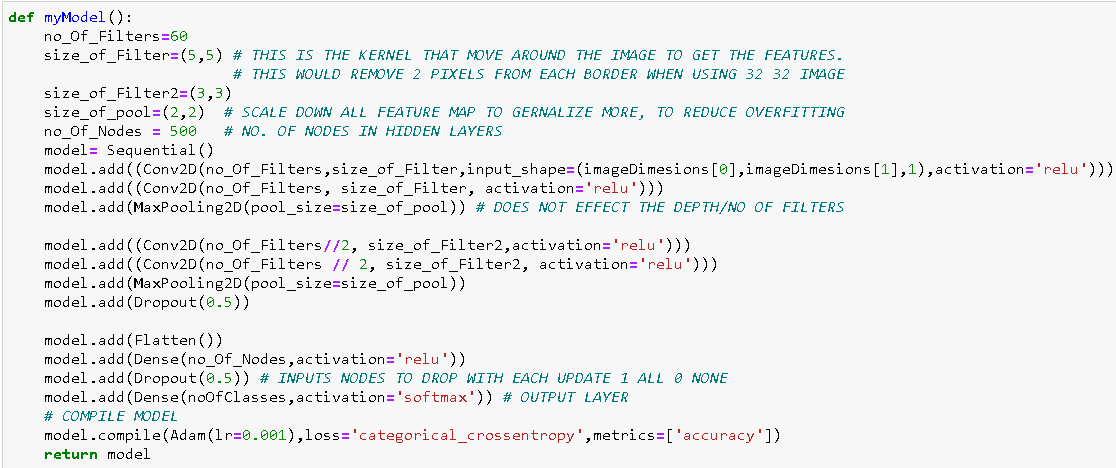
In this image below, we can see the distribution of the training dataset showing us each class with his corresponding number of images. We notice that we do not have the same number of image in each class. We can see that we have about 100 of images for the first class, and then we have about 30000 images for the second so the distribution is not even, we might get good classification for one class and bad classification for other. So we can see in the end of this project how much dataset is important to have good classification.



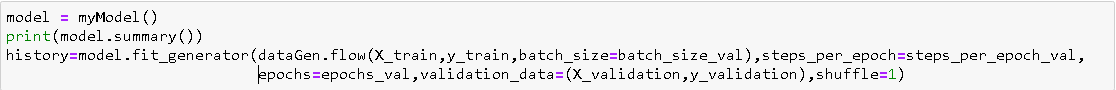
After that we are pre-processing our images, we are first converting into grayscale and then we are equalizing them, then we are normalizing the values.



Then we are going to create our model, our convolutional neural networks



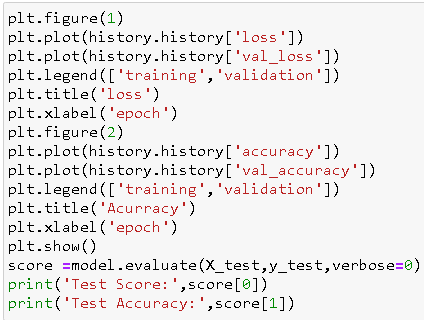
Then we train it here below

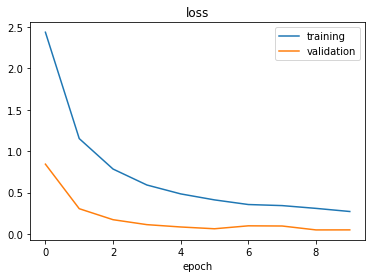


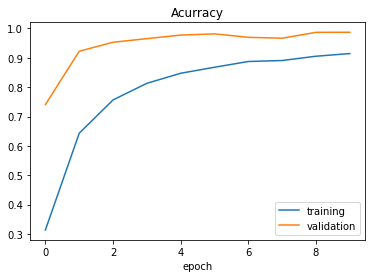
And then we save it for further use



Then we plot our result: we have the plot for loss and the plot for accuracy



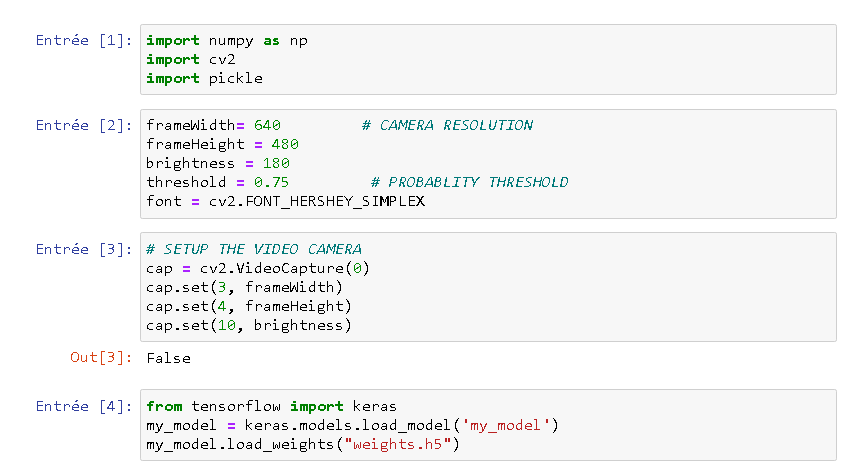




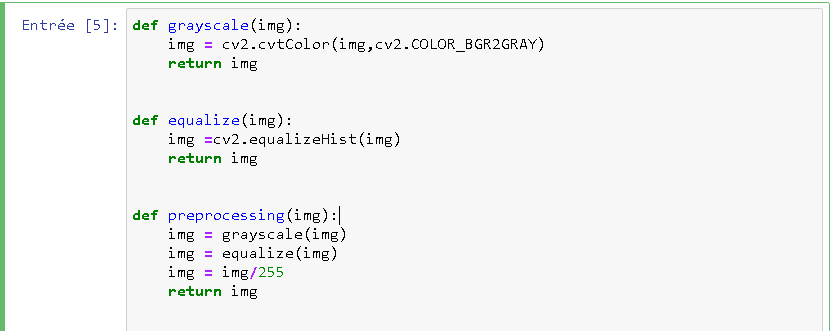
We can see that we are getting fairly good result, we can see that after about 6 ecpoch it is going at the same level so probably 6 to 8 epoch will be a good estimation of where we want to learn.

**Implementation of recognize.py**

About our recognize.py which is our test file, we stated by importing our libraries, then we setup our parameters and camera, then we load our model train as showing in the image below.

****

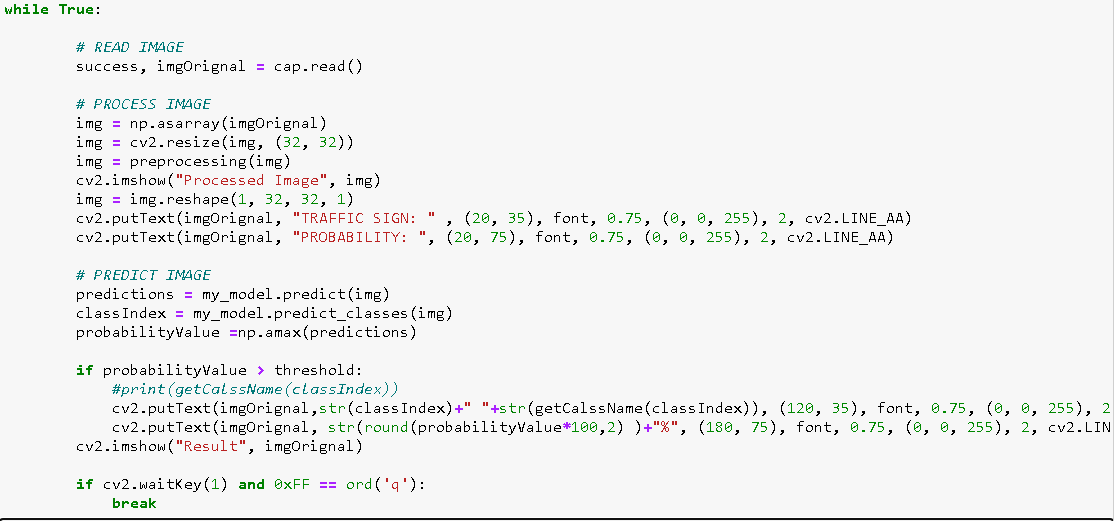
Then we are pre-processing the images as we did before the training process

****

Then we have the class name so that we can display

****

Then this is our while that we run continuously to give us our webcam image

****

1. **Architecture, Algorithms, Models and Data**

* **The Neural Network Architecture**

Using a fully connected neural network to make an image classification requires a large number of layers and neurons in the network, which increases the number of parameters leading the network to over-fitting (memorizing the training data only). The input image may also lose its pixels correlation properties since all neurons (carrying pixels values) are connected to each other [7]. Convolutional neural networks have emerged to solve these problems through their kernel filters to extract main features of the input image and then inject them into a fully connected network to define the class [7].

* **Traffic signs dataset**

A rich dataset is needed in object recognition based on neural network in order to train the system and evaluate its results. For the purpose of traffic signs classification, we used the German Traffic Sign Recognition Benchmark (GTSRB) dataset, which contains over than 35000 of images divided into 43 different classes.

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Task | Percentage of images | Shape |
| Training data | Used to train the network | 70% | 4 dimensions tensor to determine the image index in the dataset, the pixel’s row-column and the information it carries (Red Green Blue value) |
| Validation data | Allows to supervise the network performances while training it (a reduced version of testing data) | 20% |
| Testing data | Used to evaluate the final network | 10% |

Table1: dataset repartition

* **Training and testing**

The unbalanced distribution of images in the German Traffic Sign Benchmark privileges some classes over others during the training phase because they are better represented in terms of number of images. In order to make sure that the learning of the network is well performed, a data augmentation of some classes is done by applying some geometric transformations (rotation, translation, and shear mapping) on many of their images.

We split our data in training and testing phase as follow: 70 percent for the training, 10 percent for the testing and 20 percent for the validation. We done our training data after create our convolutional neural network, using tensorflow we train it ad print the result. Training our model took about 60 minutes because we set the variable epoch\_val to 10 and each epoch was about 6.1 minutes.

To improve your training time, we use three callbacks such as: **tf.keras.callbacks.EarlyStopping, tf.keras.callbacks.ModelCheckpoint and tf.keras.callbacks.TensorBoard**

1. Problems Encountered

While trying to install our dependences, I fist try to install them to the base (root) environment but it didn’t work, that’s why I created a new environment calls “tf” then I installed them inside. I was obligated to install again certain dependences such as numpy and panda which was obviously already installed in the base (root) environment.

While building and training your deep neural network model, we was faced of some of the problems:

* Training was extremely slow. To train our dataset, it took us more than hour with only 10 iterations or epochs then we found out a solution of that by using a callback which helps us to reduce the training time more than half.
* We couldn’t save our model to a pickle file by using pickle, when we were trying it, we had this error: **cannot pickle '\_thread.RLock' object**. Then we used **model.save()** to do it.

Reference

[7] L. Abdi, “Deep learning traffic sign detection, recognition and augmentation,” Proceedings of the Symposium on Applied Computing, Maroc, 2017, p. 131-136.