# Traffic Sign Recognition

---

TARGET : **Build a Traffic Sign Recognition Project**

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

No Pictures are here present, please run the differents cells to visualize the results step by step

This document described perfectly the differents steps of the evaluation : it's more a detailled design of the project and to undertand the design choice!

Every step of the "traffic sign recognition" is perfectly explain in the SW in commentaries

4 Steps are here useful :

---

STEP 0 /Preliminary :

- Librairies import (pandas, numpy, matplotlib, etc.....)
- Load the 3 Sets : Training/Validation/Test Set

---

STEP 1 : - Global Diagnostic Information (for the project) :

```
- Number of training examples = 34799

- Number of validation examples = 4410

- Number of testing examples = 12630

- Image data shape = (32, 32, 3)

- Number of classes = 43
```

- Specific Diagnostic information (specific for every image) :
    - the size of this image are: [26 25]
    - the boundary coordinate of this image are : [ 6 5 21 20]
- Data Vizualization :
    - Traffic Sign Vizualization : for every classes (43 type of sign) : an image from this class is attached
    - Class Distrubution : 3 plot/histogram (in x : number of classes : 0 ->43, and in y : number of images for this class) are available
    - 3 plot /histogram for every data set (training/validation/test)

---

STEP 2 : - Preprocessing the data set (2 tasks) :

```
- shuffle the image (to be sure that the order of the images is not important)

- image normalization for all the images to be sure that we start with the good
initial condition
(this last fonction is implemente in CNN_fonction.py)
```

---

- Tensorflow (Hyperparameters and variable initialization) :
    - variable initialization :

hyperparameters : the most important informations for the training

the learn rate, the number of EPOCH, the batch size and the drop probabilities for the full connected layer

- variable initialization :

  the placeholder initialization for the features (x), the labels (y), the dropout ans the one_hot encoding

---

- SW Pipeline for the training Set (Forward Propagation) :

  - description of the SW pipeline for the traing set, the target is to progressively to decrease the w and the h and increase the depth

  - it's a small model : the 1x1 convolution to decrease the depth is not used, because the depth is always very small

  - the normal distribution is mean=0, and the variance=0.1.

  - this part is very good described in the notebook. You can find here the differents stage of the treatment :

| **Layer** | **Description** |
|---|---|
| Input | 32x32x3 RGB image |
| Convolution 3x3 | 1x1 stride, valid padding, outputs 28x28x6 |
| RELU | Activation |
| MAX POOLING | 2x2 , output 14x14x6 |
| Convolution 3x3 | 1x1 stride, valid padding, outputs 10x10x6 |
| RELU | Activation |
| MAX POOLING | 2x2, output 5x5x6 |
| CONVERSION:1vector | input 5x5x6, output 400 |
| FULL CONNECTED | 400-> 120 |
| RELU | Activation |
| DROPOUT | 40% deleted to delete Overfitting |

| Layer | Description |
|---|---|
| FULL CONNECTED | 120-> 84 |
| RELU | Activation |
| DROPOUT | 40% deleted to decrease the Overfitting |
| FULL CONNECTED | 84-> 43 |
| RELU | Activation |
| DROPOUT | 40% deleted to decrease Overfitting |

---

- Forward propagation of the training set in 4 setps :
    - run the forward propagation to calculate the output of the SW pipeline
    - comparaison of the image at the beginning with the SW pipeline output
    - makes the mean of the cross entropy of all the the images
    - Adams optimizer to minimize the loss fonction (identical with the stochastic gradient descent)

| Function | Description |
|---|---|
| RUN FORWARD | forward propagation |
| CROSS ENTROPY/Softmax | comparaison input pipeline/output pipeline |
| Loss operation | mean cross entopy |
| ADAMS OPTIMIZATION | comparable Stochastic Grad.Descent |
| Training operation | minimize the loss operation with the ADAMS optimizater |

---

- SW Pipeline for the model evaluation (not the same as the training)
    - measure the prediction : comparaison prediction from the one_hot and the probablity from the softmax

- mean calculation of the accuracy with the mean of the individual prediction save the results

Finally the fonction "evaluate" is implemented (the generic fonction is implemented in CNN_functions.py)

---

- Train Evaluation Model (with the hyperparameters inialized at the beginning of the project) + Final Model Evaluation

  - Run sequentially every BATCH for every EPOCH and give the result for every EPOCH

  - Result : Training Set = 0.995

    Validation Set = 0.958

    Test Set = 0.938

    The result are acceptable: the validation set is very good, we don't have problem of OVERFITTING!

    The overfitting problem was resolved with the DROPOUT.

    The others opportunities as the "EARLY STOP" or "the L2 Regularizaton" has been not used. The learning rate decay is not used.

    A very good plot shows the results with the training accuracy and the validation accuracy.

---

SW pipeline configuration :

You can find here the hyperparameters tuning to find this result with the commentaries :

alpha = 0.001 # learning rate, to be sure that we will find a global minimum (if the alpha is to small -> we have times problem)

EPOCHS = 20 # training set, more the EPOCH is large, more the network runs better

BATCH_SIZE =128 # number of images packet, which runs in the network in the same time

dropout = 0.6 # 40% unit in the FC Layer deleted (No dropout for the Convolution and Pooling Layer)

The Beta for the Regularization L2 is not used, because the results are accpetable with the dropout functionnality.

A very small alpha to be sure that I will meet a global minimum. But this solution costs to much time,

that why alpha is not very small but not very large.

Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function.

I started with the LeNet Architecture.

The training set accuracy result was OK -> NO problem of underfitting

The validation accuracy result was not acceptable --> Overfitting problem

To resolve this problem, the droput with the probablities 0,6 has been implemented.

I have follow the recommendations :

- at the begining of the model : convolution and pooling (decrease the w and the h)
- at the end full connected layer

The Depth(MAX 16 before the conversion in unidimesional vector) was always small because the 1x1 convolution was not used!

---

The results are : Training Set = 0.995

Validation Set = 0.958

Test Set = 0.938

If the training set accuracy is no good : underfitting problem

If the training set accuracy is good but the validation set accuracy no good : overfitting problem

THE RESULTS ARE IN THE FILE ./RESULT!!!

---

STEP 3 : Test a Model on New Images

Here are five German traffic signs that I found on the web:

these images comes from the webside :
"https://www.adac.de/infotestrat/ratgeber-verkehr/verkehrszeichen/"

Traffic sign 1 /new1: Kurven left, virage gauche

traffic sign 2 /new2 : Richtzeichen, prioritary street

traffic sign 3 /new3 : Gefahren / Rutsch Risiko, sleepy street

traffic sign 4 /new4 : Halt, stop

traffic sign 5 /new5 : fahrichtung rechts, drive only right

traffic sign 6 /new6 : MAX 60 km/h, speed limite

The 6 images are present in the file ./new_images

Results for the differents classes:

Expected: [19, 12, 3, 23, 14, 33]

Obtained: [19 12 2 23 14 33]

You can see for the image 3 : I wait 3 and I predict 2 :

One mistake with the speed limit, we predict max speed 50km/h (2) and not 60 km/h (3) as written in the original image.

It's not a problem because 50<60, we read 50 km/h and not 60 km/h.

We respect the safety rules because 50 is smaller as 60.

Consequently we have 83% of positiv answers!

Finally, you can find here the results for the 6 images with the TOP5 probabilities :

opKV2(values=array([[ 1.00000000e+00, 1.97020829e-14, 2.38748747e-18, 7.55011224e-19, 9.40625311e-23], [ 1.00000000e+00, 2.03783276e-16, 4.45376582e-18, 8.44278725e-22, 1.19208659e-22], [ 9.99999881e-01, 1.68885364e-07, 4.49219217e-10, 2.03441333e-10, 4.12875974e-21], [ 1.00000000e+00, 3.65123676e-13, 1.58629037e-13, 4.37137949e-15, 9.20459322e-17], [ 9.99639153e-01, 3.60730279e-04, 6.44836149e-08, 2.30851054e-08, 4.51440485e-09], [ 1.00000000e+00, 9.92240136e-17, 1.14277005e-19, 2.60459101e-22, 2.28205595e-23]], dtype=float32),

indices=array([[19, 23, 29, 24, 21], [12, 26, 17, 41, 14], [ 2, 3, 1, 5, 13], [23, 19, 30, 20, 29], [14, 15, 9, 5, 17], [33, 35, 39, 34, 10]], dtype=int32))

It's here interestant to analyze the listake for the speed limitation : for the image 3 (MAX 60km/h),

we see :

[ 9.99999881e-01, 1.68885364e-07, 4.49219217e-10, 2.03441333e-10, 4.12875974e-21],

AND

[ 2, 3, 1, 5, 13],

The MAX speed 60 km/h (3) is the second position with the probablity: 1.68885364e-07

the model detect firstly 50km/H (2) with the probability : 9.99999881e-01

The difference is not too big.

the number 5 and 6 are very similar! We must increase the accuracy of the model for the number present in the speed limitation sign.

---

STEP 4 : (Optional) Visualizing the Neural Network (See Step 4 of the Ipython notebook for more details)

Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

This part was unfornunately not worked (timing problem).

It's plain to work this part later, because it's very important to sensibilize the people that a neural network is not a "black box" and

that we can visualize the result of every node in every layer.