TDGS Pooler

a novel, learnable pooler

presented by Bredariol Francesco

How did I approach the project

Literature review

and implementation of
existing pooler

Implementation

of the novel and learnable
TDGS pooler

Evaluation

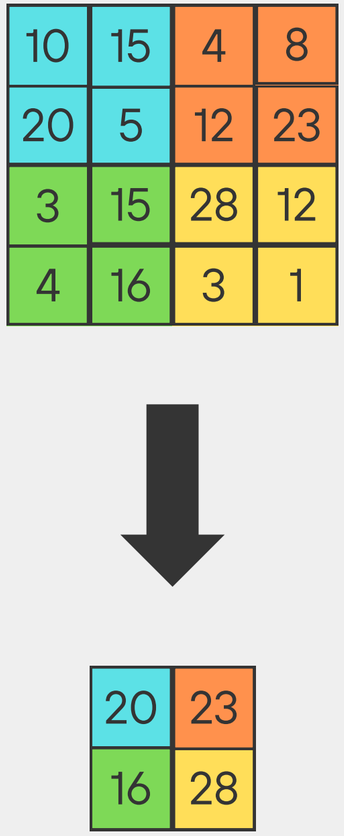
on classical dataset as
MNIST and CIFAR10

Literature review

- **Mixing Pooler**
- **Gated Pooler**
- **Stochastic Pooler**
- **SP3 Pooler**

Based on

1. *"A Comparison of Pooling Methods for Convolutional Neural Networks"* (Zafar et al. 2022)
2. "Generalizing Pooling Functions in CNNs: Mixed, Gated, and Tree" (Lee et al. 2017)
3. "Stochastic Pooling for Regularization of Deep Convolutional Neural Networks" (Zeiler & Fergus, 2013)
4. "S3Pool: Pooling with Stochastic Spatial Sampling" (Zhai et al. 2017)



example of max pooling.
the dimension is reducted from 4x4 to 2x2
while all the peaks (maximum values) are retained.

What is a pooling function?

A pooling function is a mathematical operation used primarily in convolutional neural networks (CNNs) to reduce the spatial size (width and height) of feature maps while retaining important information. It helps in making the model more computationally efficient and less sensitive to small translations in the input.

MIXING POOLER

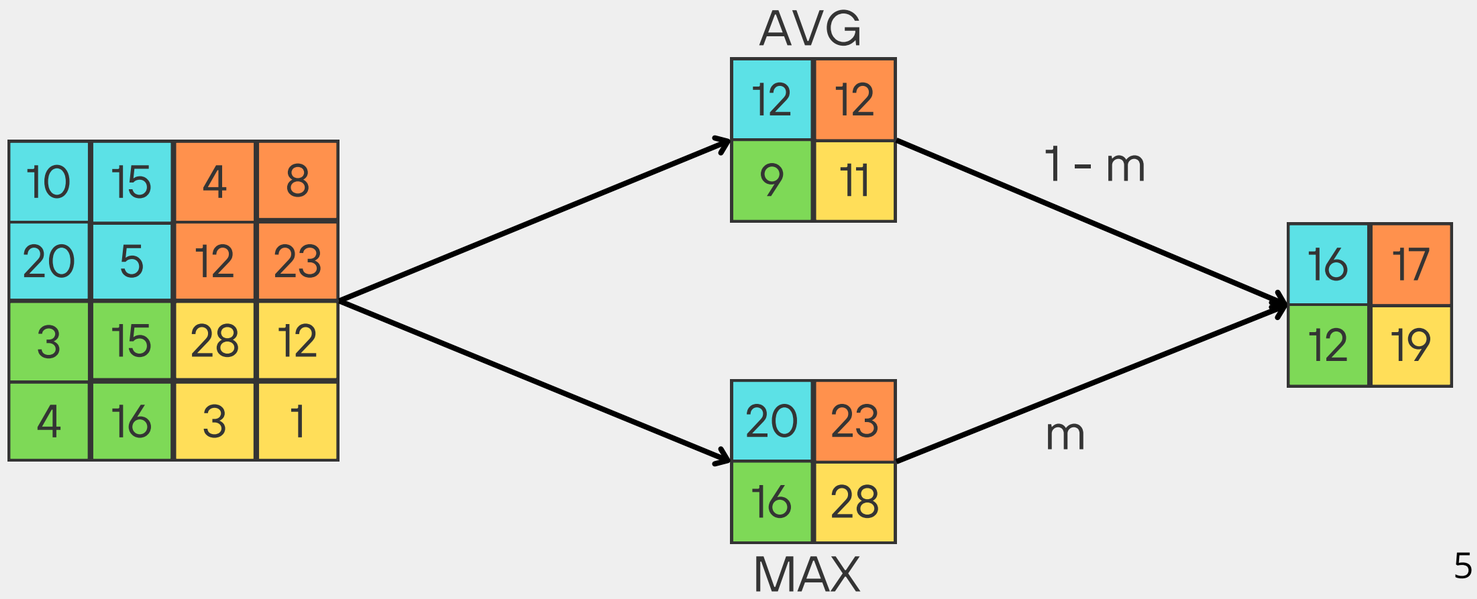
Given a pooling patch Rj, the output of the pooling function Sj is

$$S_j = \sigma(\lambda)\max(a_i) + (1-\sigma(\lambda))\frac{1}{|R_j|}\sum_{i\in R_j} a_i$$

where sigma stands for the sigmoid activation.

LEARNABLE BUT STATIC!

example of mixing pooling.
1) average pooling
2) max pooling
3) mixer parameter (m) = 0.5
For a visualization purpose actual values are truncated.



GATED POOLER

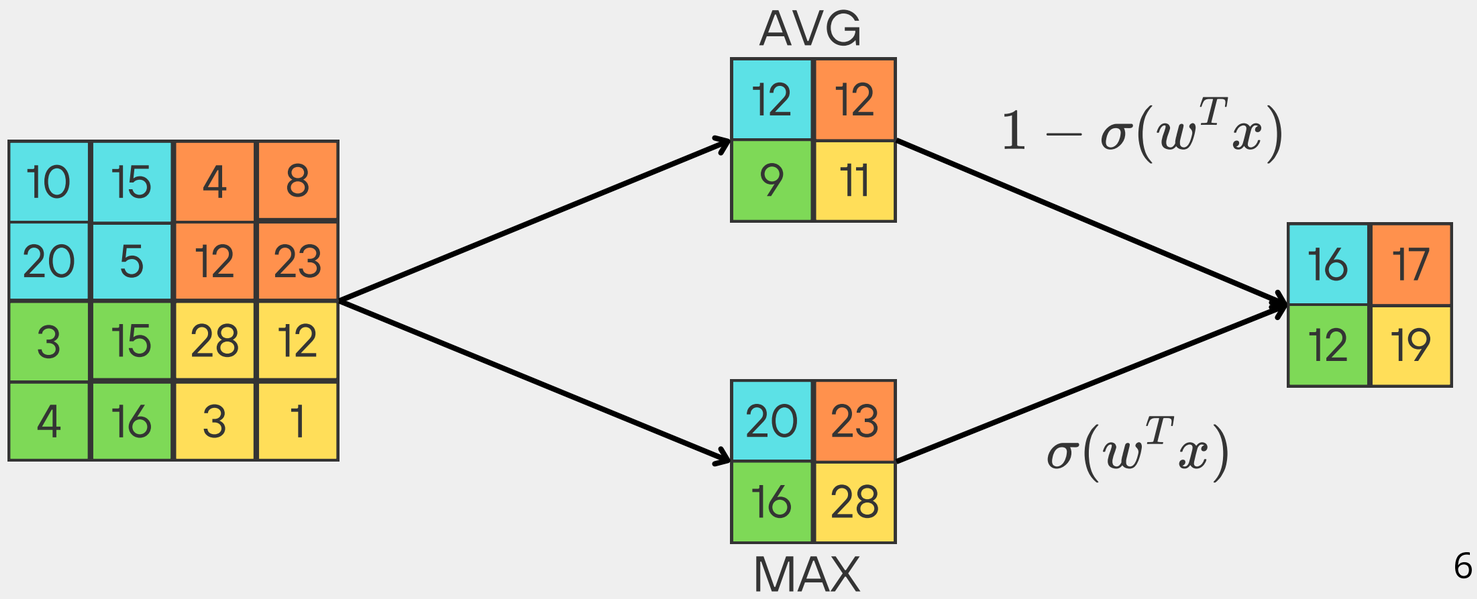
Given a pooling patch Rj, the output of the pooling function Sj is

$$S_j = \sigma(\omega^T R_j)\max(a_i) + (1 - \sigma(\omega^T R_j))\frac{1}{|R_j|}\sum_{i \in R_j} a_i$$

where sigma stands for the sigmoid activation.

LEARNABLE and RESPONSIVE!

example of gated pooling.
1) average pooling
2) max pooling
3) sigmoid(wx) = 0.5
For a visualization purpose actual values are truncated.



STOCHASTIC POOLER

Given a pooling patch Rj, we define a probability distribution on its indexes as follows

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

NOTHING TO LEARN,
REDUCE OVERFITTING,
CAN HAVE VARIOUS FORM

then the output of the pooling function Sj is

$$s_j = a_l \text{ where } l \sim Mult(1, \pi(R_j))$$

example of stochastic pooling.

10	15	4	8
20	5	12	23
3	15	28	12
4	16	3	1

$Mult(1, \pi(R_j))$

15	23
4	28

SP3 POOLER

1. Perform a MaxPooling2d with no dimensionality reduction
2. Divide the input feature map into a grid of non-overlapping blocks (e.g., 4×4 patches)
3. Randomly sample a fixed number of rows and columns from each block (e.g., 2 out of 4)
4. Select the elements at the intersections of the sampled rows and columns
5. Repeat for each block, and combine all sampled patches into the downsampled output

NOTHING TO LEARN,
PREVENT OVERFITTING,
IMLPICIT AUGMENTATION

example of SP3 pooling.
1) max pool 2x2 with stride 1
2) stochastic downsampling over columns and rows

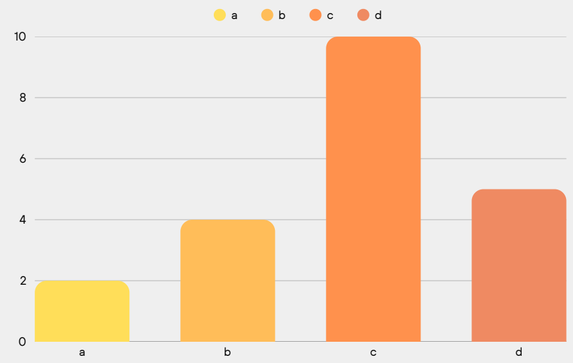
TDGS POOLER

Temperature Driven Gumbel Softmax Pooler:

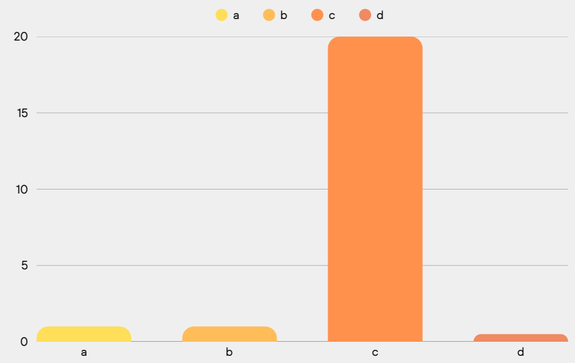
In this novel pooling operator the idea is to let a learnable temperature parameter T to model the SoftMax distribution.

- T low (<1) : sharpened distribution centered in the max value
- T high (>1) : softer distribution similar to uniform distribution

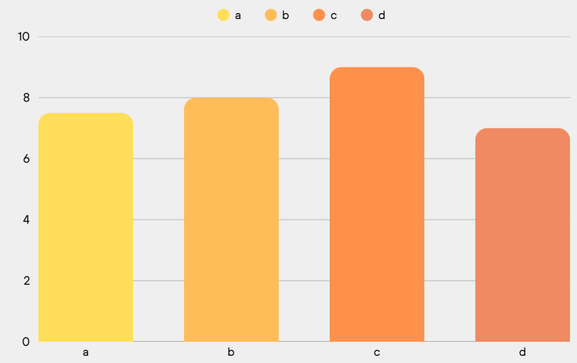
The temperature parameter serves as balancer between deterministic behavior (which is somehow related to exploitation, MaxPool) and stochastic behavior (which is somehow related to exploration, Uniform StochasticPool).



T = 1
(histogram)



T = 0.1
(max)



T = 10
(uniform)

TDGS PROBLEM

We want to sample from a distribution parameterized by the parameter T.
The main problem is that it is not feasible to optimize the parameter of a probability distribution while sampling from it using backpropagation: it is non-differentiable.

Reparameterization trick

Keep it simple: if we want to perform sampling and optimization from a gaussian.
Math comes in help and rememeber us that

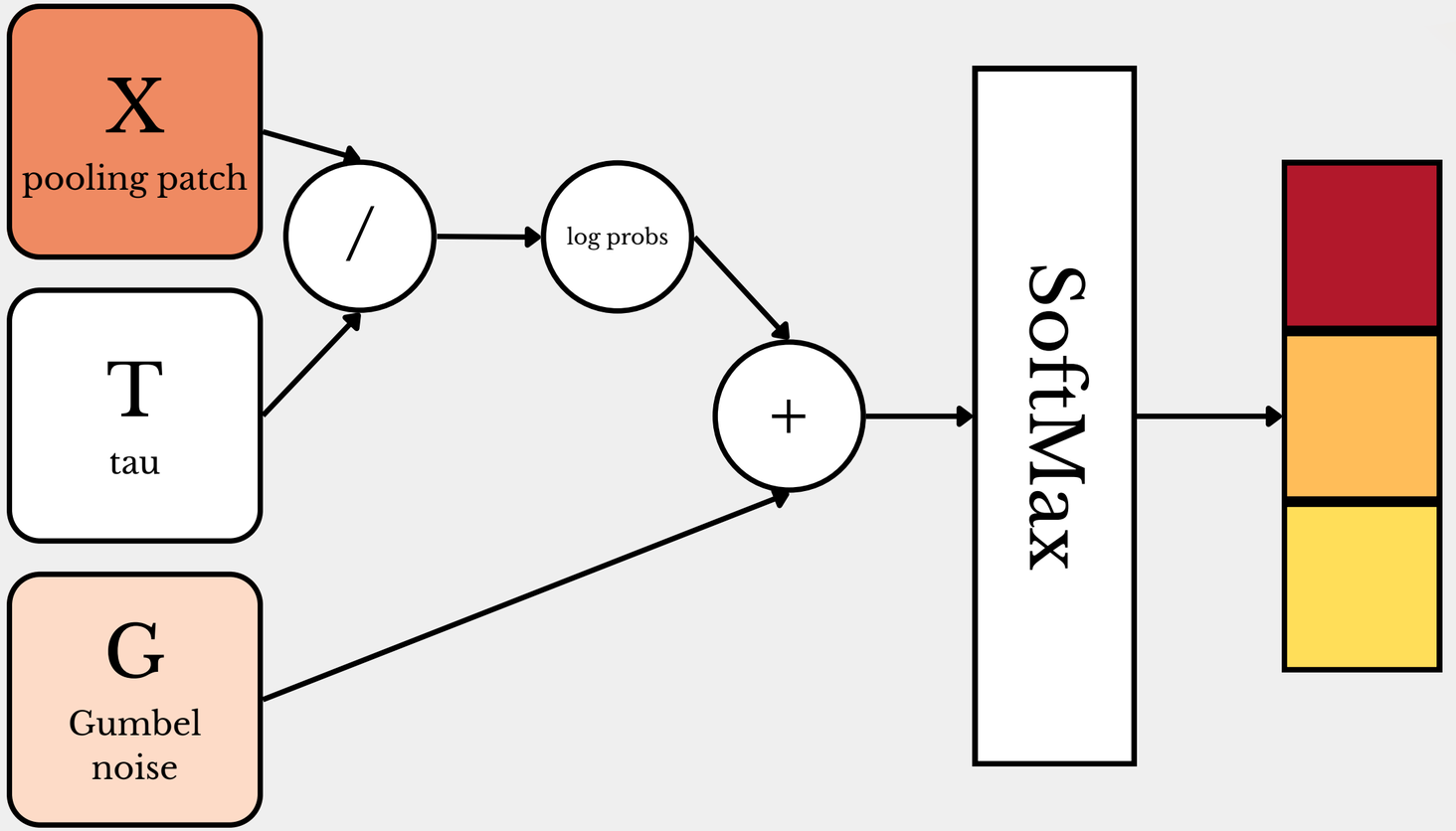
$$z = \mu + \sigma * \epsilon \text{ where } \epsilon \sim N(0, 1)$$

Now this is fully differentiable.

GUMBEL SOFTMAX

Similarly to the reparameterization trick, the gumbel softmax trick operates not on gaussian but on categorical distributions, producing the same output of a softmax.

MATHEMATICAL PROOF



This is what I actually do

TDGS ADJUSTMENTS

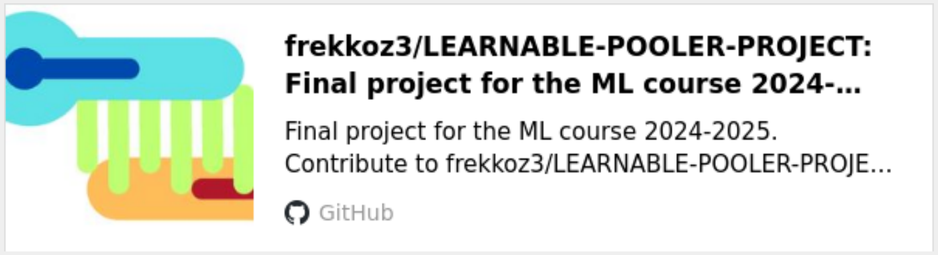
02

Even if everything could work just like that, I added some little adjustments

- The temperature used while computing the log probabilities is not the raw temperature but a rectified one, to make it strictly positive
- A little epsilon is also added to the rectified temperature in order to prevent numerical errors
- Temperature is initialized from an Uniform distribution in [0.5, 1.5] multiplied by an initial value
- During inference a weighted sum is performed as output (related to what done in StochasticPooling)
- To obtain elements of the same magnitude, we are gonna normalize the vector before dividing it by the temperatures

IMPLEMENTATION

Everything is full implemented using PyTorch (the Gumbel Softmax is already existing).
The full implementation is available at my GitHub (link below).
Project.ipynb is a fully working wrapper for everything discussed until now, check it out.



TEMPERATURE INIT. and EVALUATION

03

To tune the initialization of the temperature and to evaluate how the pooler works a simple CNN is being developed composed as follows:

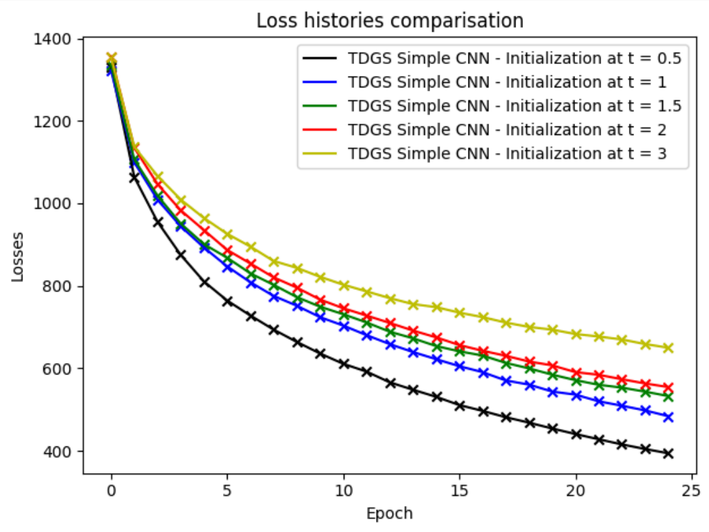
- conv2d
- pooler
- conv2d
- pooler
- conv2d
- pooler
- linear
- linear

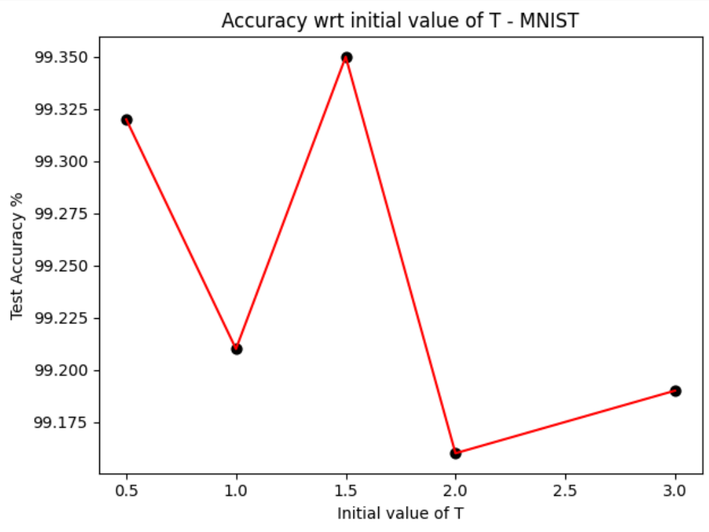
as activation function ReLU is used.
As dataset are used MNIST and CIFAR10.

TEMPERATURE INIT.

Various initial values have been explored.
Results are not so quite clear. Probably since the natural stochasticity of the model they can differ from one run to an other. The relevant thing is that the highest test accuracies are always obtain by t in the range [0.5, 1.5].
A better interpretation should be done analyzing how the temperature is set once the training is finished. Looking at it we can understand which type of behavior (closer to Max or closer to random uniform) it has developed.

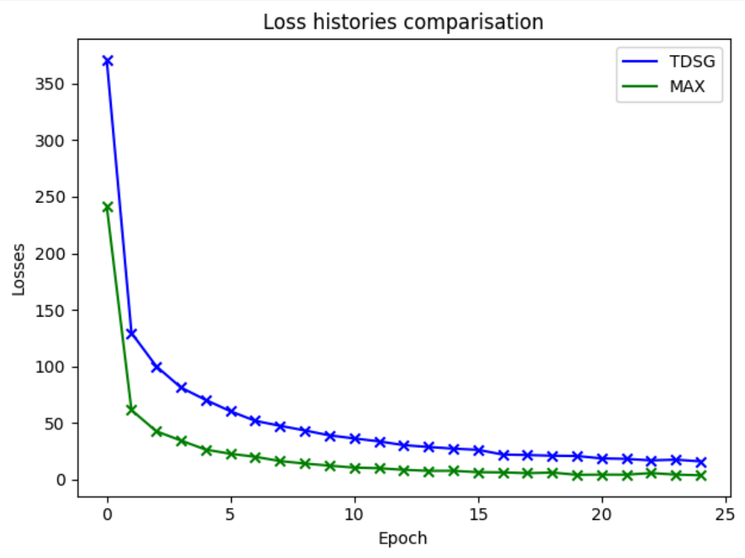
Results on MNIST



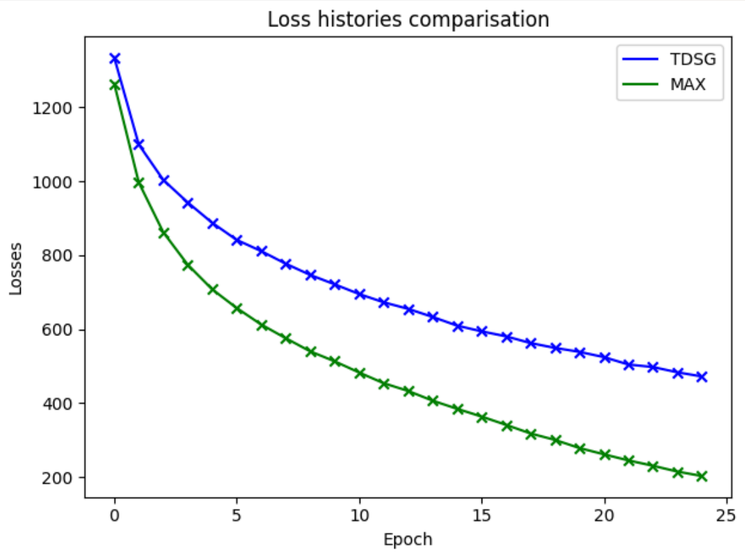


EVALUATION

In order to evaluate the pooler I test out the results against the same architecture where instead of the TDSG pooler a Max pooler is used. I fixed the initialization of the temperature to the value of 1. Here I show for both dataset the training losses histories.



MNIST



CIFAR10

EVALUATION

Here I report the accuracies obtained by the models.

MNIST

POOLER	ACCURACY
MAX	98.96%
TDGS	99.24%

CIFAR10

POOLER	ACCURACY
MAX	70.37%
TDGS	72.89%

*this is just an instance of the results. I found out that on average the TDGS gain a 0.5% on the MNIST and good 3% on the CIFAR10

Thank You