Characterization of turbulent flows in tokamaks

Julien Hu, Matthieu Masouyé and Sébastien Ollquist
Department of Computer Science, EPFL, Switzerland

***Abstract*—A tokamak is an object in the form of a torus in which we make plasma turn at very high speeds in order to generate energy. The goal is then to push gas into the tokamak while plasma is turning and collect images of it in order to detect the regions where turbulence is the highest. This whole process is called Gas Puff Imaging (GPI) and the concerned specific regions of turbulence are called blobs. We will in this report describe the machine learning techniques we have implemented in order to help identifying the direction of the velocity of the particles in a selected turbulent region.**

I. Introduction

Gas Puff Imaging (GPI) is a technique that involves pushing gas in a tokamak in order to study the turbulence present at the edge of magnetically confined plasmas. It uses a puff of neutral gas so we can increase the local light emission level in order to improve optical imaging of the space-time structure of the edge plasma turbulence.

The primary goal of this project is to understand and implement machine learning methods that can help us detect the shear layer more easily, that is, the separation at which the direction of the flow of particles changes. The idea is to implement a 3D Convolutional Neural Network on the data set that can analyze a sequence of images easily, all this in order to facilitate the distinguishibility of the different flows [1].

II. Data importation and manipulation

A. Initial data

We were initially given a matlab file with all necessary data to use for the project. The data given consists of four sets described down below. In order to prepare the data to become an input parameter, we have created a python script that reads the data from the .mat file and creates a numpy array from each part of the given file. Then, these arrays are written into four separate csv files, one for each category. The four relevant data sets given are:

1) `r_arr` a 10 by 12 matrix of radiuses.
2) `z_arr` the corresponding array of the z-axis values.
3) `t_window` the array of times at which a particular value was measured.
4) `brt_arr` the 3d-array (10x12x200000) of brightnesses.

We will be interested in the `brt_arr` which contains each individual experiment that has been done accross a one second time lapse. The frequency at which the images are taken is a fixed $2MHz$, thus one frame is taken every 500 nano-seconds approximately. Each frame will be analyzed by our algorithm in order to determine the velocity of each particle in the frame and its corresponding direction.

B. Follow up data

Then, we were given more precise data and in bigger quantity in order for us to improve the training of the model. In complement of the data we have already received, we were given additional information required for the training such as:

1) `vz_from_wk` the vertical speed for each column of data, given in km/s.
2) `vz_err_from_wk` the error of the velocity estimation.
3) `col_r` the average r-coordinate for each column.

This time, both `vz` arrays have a $(13,)$ dimension because the middle column contained both positive and negative velocities, as its closest to the shear layer.

We were also given a small python script that loaded the data into python so that we could use it directly for the training. However, the script was in Python 2.7 so we had to modify it a little in order for it to be compatible with our current Python version, which is 3.6.

C. How to treat it ?

The best solution to analyze such data is to create a 3D convolutional neural net (CNN), method that is widely used for video analysis, a security camera footage for example. A CNN consists of processing layers that are used to reduce an image to its key features in order for it to be more easily classified. These three layers are:

1) the convolution layer, in which the images are translated into processable data,
2) the pooling layer which progressively reduces the spatial size of the data in order to reduce the amount of parameters and computation in the network, and
3) the fully connected layer, in which after all the necessary computation is done, the output layers are flattened, the probabilities identified are analyzed ans the output is assigned a value.

III. Synthetic data generation

We decided to create a script that generate artificial data for two reasons: First is that the real dataset would take some time for the lab to prepare, and then more importantly it would give us a controlled environment with no noise or imprecision. With the advices of the lab, we generate these data in the following way: We draw a large canvas (typically 480x480), and spawn in randomly gaussian arrays of varying size; A portion of these arrays are negative. To draw a frame, all the arrays are summed in the canvas; then, a smaller window at the center is taken and downsized until it reaches the final size of 10x12. To update the position of these arrays, they are

attributed a vertical speed given their horizontale coordinate, and at each frame their new positions are calculated by adding this speed. The function used to assign these speeds is a tanh, with several settings to modify it's behavior. To get the labels, we can simply average this speed function over each column of the final window.

Everytime the script is run, it will generate a new folder inside data/, where it will put the final frames, the labels, and various other infos for debuging and reproducability purpose (like a speedplot, a list of settings, and a video of all the frames).

This method gets reasonably close to how the real data looks, and we can get away with a few simplification: for instance, the shear layer is always vertical, and the speed we want is only per column; so we only need to deal with speed in Z.

IV. Model training

A. Finding the correct architecture

This part essentially contains the research we have done in order to choose the correct architecture for the project. The model we have decided to use is based on ResNet, which is a classic neural network that helps solving computer vision tasks [2]. As we have to analyze a sequence of images, this model is the perfect solution for us.

B. Training the model on synthetic data

In this section we will describe how we have proceeded to train the model on the synthetic data in order to prepare it for the real data later. Using our data loader script, we import the data we want to use and define a training set, a validation set and a test set.

a) Training: This part essentially consists of two main phases: a forward phase and a backward phase. In the forward phase, the input goes through the whole network and in the backward phase, an algorithm is used (back propagation algorithm) to derive the gradients and update the weights of the model. We perform multiple iterations of the training algorithm which are called "epochs" in order to train the model better and decrease the loss as much as possible.

b) Evaluation: This part is meant to evaluate how good we have done in the training part. We do this by computing the proportion of correct predictions done.

c) Testing:

C. Training the model on the real data

V. Conclusion

References

[1] M. J. Park and M. Sacchi, "Automatic velocity analysis using convolutional neural network and transfer learning," 2019. [Online]. Available: https://www.researchgate.net/publication/336339562_Automatic_velocity_analysis_using_convolutional_neural_network_and_transfer_learning

[2] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?" *arXiv preprint*, vol. arXiv:1711.09577, 2017.