Characterization of turbulent flows in tokamaks

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

Abstract—**A tokamak is a type of plasma generator, inside which plasma turns at very high speeds. The plasma is confined in the shape of a torus via magnetic fields, at the edges of which it becomes turbulent; this is called the shear layer. These turbulences might create "blobs" of plasma, that can be ejected at high speed and damage the tokamak.**

Gas Puff Imaging (GPI) is a technique used to study these turbulences: it consists in launching a puff of neutral gas near the shear layer, to generate light emissions when mixing up with the plasma and thus allowing imaging of the turbulent structures.

Throughout this project, there will be mention of R-axis and Z-axis: those are the radial and height defined from the center of the torus' ring. When looking at GPI images, you can imagine them to be equivalent to the X and Y axis, respectively.

I. INTRODUCTION

Gas Puff Imaging analysis is a ongoing task, and can yield important results. The Swiss Plasma Center lab at EPFL was interested in the new possibilities machine learning could bring, and together we worked to test how well a neural net could analyze the GPI images. The primary goal of this project is to understand and implement machine learning methods that take GPI images and correctly measure the z-axis velocity (parallel to the shear layer) of the moving, turbulent structures.

II. DATA AND PREPROCESSING

A. Synthetic Data

We decided to create a script that generates artificial data for two reasons:

1) First of all, the real dataset would take some time for the lab to prepare, but
2) more importantly it would give us a controlled environment with no noise or imprecision in the labels.

This overall means that we could get a good idea of how well our architecture works, and later try to use it in combination with the real dataset to help it learn.

With the advices of the lab directors, we decided to generate this data in the following way: we draw a large canvas (typically 480x480), and spawn in randomly gaussian arrays of varying size. Note that 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 horizontal coordinate, and at each frame their new positions are calculated by adding this speed. The function used to assign these speeds is the hyperbolic tangent, with several settings to modify its 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 information for debuging and for reproducibility purposes (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 like, and we can get away with a few simplifications. For instance, the shear layer is always vertical, and the speed we want is only per column, so we therefore only need to deal with the speed in z.

B. Real Data Description

The labeled data was given to us from the lab in pickle files, inside which there are the following sets:

1) `shot` The shot number.
2) `t_window` An array of timestamps at which the frames were measured (in seconds)
3) `brt_arr` A 3D-array (12x10xl) where l is the length of `t_window`) of brightnesses of each frame, measured in $mW/cm^2/ster$)
4) `r_arr` The array of r-coordinate of each view, in meters.
5) `z_arr` The similar array of z-coordinate of each view, also in meters.
6) `vz_from_wk` The vertical speed for each column of data, in km/s.
7) `vz_err_from_wk` The error of the estimation of vz, and
8) `col_r` the average r-coordinate of each column.

Due to the measurement method, four specific "pixels" of `brt_arr` are set to NaN permanently. The `r_arr` and `z_arr` describe the exact position of each pixels of `brt_arr` in a 2d space, as they are not perfectly spaced. `vz_from_wk` will constitute the labels for the algorithm.

Importantly, there are 13 values in `vz_from_wk`, `vz_err_from_wk` and `col_r` while there are only 12 columns per frames: this is because the shear layer we want to calculate is inside one of the column, which means this column will have flows going both way. Thus, this column will have two speeds associated to it. It also means that `col_r` has two identical values, as they concern the same column. Note that it wasn't possible for the lab to provide good estimates of these two speeds in all datasets, and the 13th column will be set to NaN if they couldn't get an accurate estimation.

C. Data Manipulation and Preprocessing

We implemented two different Dataloaders for using real or synthetic data:

1) The synthetic one doesn't do anything special, as the synthetic data is already generated in a convenient way. It simply fetches all the frames and labels, organizes them in datapoints compatible with the architecture, and shuffles the order of these datapoints to homogenize scenarios.
2) The one working with real data, in addition, has to rescale the brightness of the frames to 8-bit greyscale images, and organize the labels to be coherent.

Due to time constraints, we only could use part of all these sets: `t_window`, `r_arr`, `z_arr` and `vz_err_from_wk` were not taken into consideration for our algorithm. Moreover, we had to ignore datasets where the 13th column was missing as described above, as that would have required us to change our approach to semi-supervised learning.

D. 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 and the output is assigned a value.

III. 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 [1]. 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.
1) In the forward phase, the input traverses the whole network for the first time.
2) 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

IV. CONCLUSION

REFERENCES

[1] 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.