Characterization of turbulent structures in tokamaks

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

***Abstract*—The goal of this project is to use machine learning to analyze GPI measures and estimate the Z-axis velocity of the structures.**

To perform this task, we have implemented a convolutional neural network (CNN) to predict the speed of the plasma, on a synthetic dataset that we generated ourselves and real data provided to us by the lab.

We have tuned some hyperparameters to optimize the models and obtain the best results possible. We were able to successfully train and optimize a CNN for each dataset, and overall we were able to have good estimation of the speeds on the real data.

I. Introduction

A tokamak is a device which uses magnetic fields to confine hot plasma in the shape of a torus [1] (See Fig. 1). At the edges of the core plasma, turbulence generate coherent filamentary structures, also called blobs, that can be ejected at high speed.

A technique used to study these turbulences is called Gas Puff Imaging (GPI). It generally consists of injecting a puff of neutral gas in this region, which will be excited by the plasma. This will create light emission, that are then collected tangentially to the magnetic field lines, allowing imaging of the turbulent structures.(See Fig. 2)

GPI analysis is an 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 network could analyze these images.

The primary objective was to measure the z-axis velocities of the moving, turbulent structures captured by GPI. We have decided to opt for a convolutional neural network working on sequences of frames, as it is a good approach for complex tasks and convenient for image sequence analysis and velocity estimation [2].

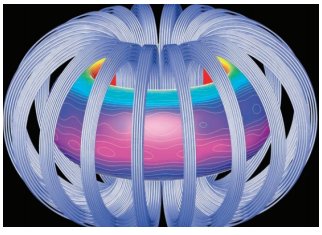

Fig. 1: Geometry of a tokamak [3], which has the shape of a torus.

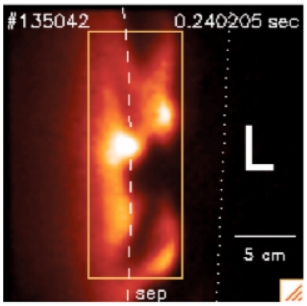

Fig. 2: GPI images from the tokamak, where gas is excited by the plasma. [4]

II. Data and preprocessing

A. Real Data Description

The labeled data was given to us from the lab in 9 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 (of 12x10xl where l is the length of `t_window`) of GPI measures, corresponding to the brightness of each 12 by 10 frames, 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 margin in the estimation of vz
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 pixel 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 columns, which means this column will have flows going both ways. Thus, this column will have two speeds associated to it.

It wasn't possible for the lab to provide good estimates of these two speeds in all datasets, so the 13th column is set to NaN if there isn't an accurate estimation. We had to ignore these sets as that would have meant changing our approach too much; those concerned 3 out of the 9 different files. Finally, the direction of the plasma relative to the shear layer is consistent: the left side always has plasma going down (i.e. negative values), while the right side always going up (i.e. positive values).

In total, we have access to 278 datapoints with the provided datasets.

B. Synthetic Data Description

We have decided to generate artificial data mainly because it would give us a controlled environment with no noise or imprecision in the labels, and also because it would make for a clearer model to train and analyze.

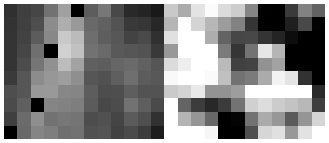

Fig. 3: Real image on the left, synthetic image on the right

We have decided to generate the data in the following way: we draw a large canvas (480x480), and spawn gaussian arrays of varying random small sizes (264 arrays of 100 to 260 pixels) in random places. A ratio of these arrays are also set to negative values (around 33%). To draw a frame, all the arrays are summed in the canvas, and values are limited to between 0 and 255 to stay in an 8 bit grayscale. Then, a smaller window at the center is taken and downsampled until it reaches the final size of 10x12, same as the real data.

At creation, these arrays are assigned a vertical speed given their horizontal coordinate by an hyperbolic tangent, with several settings to tweak its behaviour. To update their position at each iteration, the speed is simply added to their position. 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 the final frames will be put, with the labels, and various other information for debuging and for reproducibility purposes (like a speedplot or a list of settings). At the beginning of the script there is a list of variables used as settings to tweak the behaviour of the generated dataset.

This method gets reasonably close to how the real data looks like. You can see a side-to-side comparison in Fig. 3. In the precedent description, we have overseen a few simplifications in the behaviour of the synthetic dataset. For instance, the shear layer is always vertical but the real data's shear layer is slightly curved. Another important distinction is that the speed of the structures doesn't only go alongside the z-axis in the plasma. Simulating these aspects has been judged too complex for the stage of this project.

In total, we have generated a total of 60 datapoints of 2000 frames each. We wanted to generate more, but we were limited by the computational power of our personal computers.

C. Data Manipulation and Preprocessing

We have implemented two different data loaders, that are used to organize and load real or synthetic data into the CNN.

Both data loaders organize the frames into 2000-frames datapoints (10x12x2000 values), and shuffle these datapoints using a fixed seed to homogenize the different scenarios. These datapoints are separated into 3 datasets: training, validation and testing sets, with default ratios of 64%, 16% and 20% respectively. They will be used in the training and valudation phases, with testing only being used to evaluate our best model at the end.

The last transformation these datapoints need is to be converted from a 1-channel greyscale to 3-channel, to be compatible with the architectures we used. This was done simply by duplicating the single channel across the two others; this shouldn't pose any problem, because the model shouldn't depend on specific colors.

The differences between the two data loaders are where and how they fetch the frames, as they are stored in a different way on disk. Moreover, the dataloader working on real data has to rescale the values of the array to be 8-bit greyscale images instead of raw brightness measures. This isn't a problem for the synthetic dataloader as the synthetic data was already generated in a convenient way.

Due to time constraints, we could only use part of all the sets for the real data, that is, `t_window`, `r_arr`, `z_arr` and `vz_err_from_wk` were not taken into consideration for our algorithm.

Finally, the reason we chose to work with inputs of 2000 images is that on one side more images were pushing the limits of our computers in terms of time and memory. On the other, the lab estimated that an entire structure would take approximately 18'000 frames to cross the image completely, so 2'000 frames should mean it moves at least one pixel, as the images have a height of 10 pixels.

III. Model training

A. The CNN architectures

To perform our regression task, we have decided to work on the architectures defined for video classification in PyTorch, for simplicity and efficiency. We trust the architectures have already been optimized and tested. It is hence easier and safer to use them rather than develop our own network. The architectures considered were: *ResNet 3D*, *ResNet Mixed Convolution* and *ResNet (2+1)D* [5]. ResNet is a classic neural network that helps solving computer vision tasks [6]. As we have to analyze a sequence of images, this architecture is appropriate for our problem.

These networks consist of multiple processing layers that perform different operations to process the input, extract and recognise its key features (measurable property or characteristic), but also predict the desired outputs. These layers are:

- The convolutional layer, in which features from input images or feature maps are extracted by applying filters.
- The pooling layer, that is similar to the convolutional layer, but performs a specific function, such as taking the maximum or average value in a region, to reduce the complexity and amount of computation of the network
- The fully connected layer, where the network regroups all information from the final feature maps, and generates the output.

The main difference between the three architectures lies in the convolutional layers, where filters are applied distinctively:

- ResNet 3D performs a 3D convolution, the 3D filters are convolved both over time and space dimensions.
- ResNet Mixed Convolution starts with 3D convolution, then in later layers, uses 2D convolution, as at higher level of abstraction, motion or temporal modeling may not be necessary.

- ResNet (2+1)D seperates the 3D convolution into two steps, a 2D convolution in the space dimension followed by a 1D convolution in time, to reduce the computation cost [7].

B. The training procedure

We start training it on the synthetic data so that we can get a good impression of how well it performs. We then train it on the real data.

The training loop consists of two main phases: the training phase, where the model is first trained from the training set. The model learns how to recognize the key features, and will update its weights (learnable parameters) with the gradient calculted on its performance on the dataset.

Then, during the validation phase, the model simply predicts the output for the validation set, and computes the loss. This part is to give an estimation how good the model performs at each iteration, to detect any signs of overfitting.

Each step is repeated for 30 epochs. The last validation results are used to compare different instances of the model and choose the best hyperparameters. After optimizing and selecting the best performing model, it is then tested on the testing set, which contains data the model has never been executed with. The testing phase is only to provide us with an evaluation of the final model.

C. Finding the correct architecture

To find the most suitable model for our task, we have decided to train and evaluate it with our synthetic dataset. We see from the last validation results from table I, that ResNet 3D offers the best results. Moreover, while training the models afterwards on the real dataset, our computers were unable to handle the ResNet MixedConv and ResNet 2+1D, as the memory load is too heavy with some set of hyperparameters, and would restrain us for optimization. We therefore selected and optimized ResNet 3D.

Architecture	Validation loss
ResNet 3D	1.311
ResNet MixedConv	3.792
ResNet 2+1D	3.284

TABLE I: Last validation results for the architectures after 30 epochs, with learning rate 0.02 and batch size 3

After choosing the architecture to work on, we also needed to get the optimal hyperparameters. The hyperparameters we chose to improve the model with were:

- The learning rate *lr*, which describes how much should it update itself based on the error. Choosing a learning rate that is too high can cause the model to converge too quickly to a sub-optimal solution or overfit the training data, while a too low learning rate results in a long training process, or can even cause the model to be stuck.
- *batch_size*, which defines the number of samples that will be trained at the same time, and affects how the gradient is calculated when correcting the weights. In short, bigger batches means the gradient should fluctuate less and be more accurate, at the expense of memory required. But it doesn't necessarily means a better model, as a noisier gradient might lead to a more robust model.

The loss function used to evaluate the model is the MSE (Mean Square Error) loss function, as it is the most commonly used function for regression tasks. As for the optimizer, we decide to choose SGD. SGD (Stochastic Gradient Descent) is an algorithm used to approximate the gradients with a subset of the data instead of the whole dataset, to reduce the computational burden. We have also used a scheduler, which causes the learning rate to decrease after some epochs, so that the model converges safely. The γ has been set to 0.1 and the step_size to 10, as we didn't have time to optimize them. The seeds are the same for all the models.

IV. Results

	0.005	0.01	0.02	0.03	0.04	0.05	0.06
2	3.77	2.81	1.50	2.54			
4	4.01	2.74	1.25	1.05	1.59		
8	4.25		3.77	1.16	1.02	2.48	
16			5.76	4.02	3.97	3.12	4.00

TABLE II: Batch size (column) vs learning rate (rows) on synthetic data

	0.001	0.005	0.01	0.02	0.03	0.04
2			7.653	2.394	1.863	3.089
4			0.377	0.268	0.256	0.359
8	0.213	0.093	0.111	0.133	0.247	
16	0.153	0.116	0.126	0.375		

TABLE III: Batch size (column) vs learning rate (rows) on the real data

The results of our model on the synthetic data can be found in table II, and the ones on real data in table III. The losses and models are from the last epoch of the algorithm. Note that not all cases have been tested. In fact, we stopped increasing the learning rate when the loss got worse. As for the batch size, we were again limited by our computing power, and 16 was the most we could run with.

It is also important to note that the losses of both datasets are not strictly comparable; indeed, the synthetic dataset and real dataset do not use the same scale for measuring speed in their labels; they are however close.

A. Synthetic Dataset

The losses throughout the epochs are a bit unstable: they often oscillate while converging, and only calm down after the decrease of the learning rate, at the tenth epoch. A good example of this is in Fig 4. We think this is due to both the fact that we don't have that many datapoints, and the fact that the different datasets have too many variations between them; the model has trouble learning it all.

Another sign that shows that our synthetic dataset is too complex for the amount of datapoints is that the model does

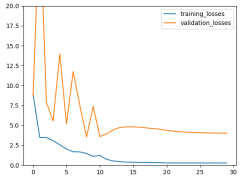

Fig. 4: Training and validation losses on synthetic dataset with learning rate 0.06 and batch size 16, we observe some oscillation until epoch 10.

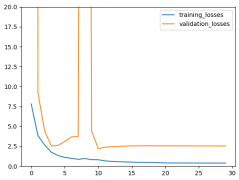

Fig. 5: Training and validation losses on synthetic dataset with learning rate 0.03 and batch size 2, a learning rate too high means the model has trouble finding the optimal parameters

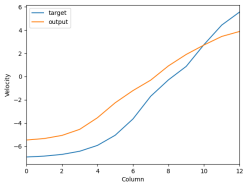

Fig. 6: Bad prediction from the best model on synthetic data, for learning rate 0.04 and batch size 8

worse with a high batch size. Higher batch sizes tend to degrade a model's ability to generalize, and a batch size of 16 was close to half the datapoints available for training. Finally, we can see the models have a hard time adapting to some of the example, as in Fig. 6

B. Real Dataset

The real dataset exhibits a strange behavior at higher batch sizes: the validation loss starts going under the training loss, as it can be seen in Fig 7. We unfortunately do not have a definite explanation, but our theory is that since there are only 6 different labels for all the real dataset, it can optimize all of them at once (on average) when the batch size is more than 6. It will find the gradient that maximizes all the different labels instead of only a few at a time. This means it will be more robust, and achieve a better validation, but it will have a harder time fitting all the training examples at once in comparison. This is supported by the fact that the validation loss is instantly under the train loss, whilst the train loss takes a while to decrease. The gradient however goes in a very robust direction from the begining of the training.

Another possibility is that the validation set has easier examples than the training set, but it is not consistent with the batch size. It should happen on all batch sizes.

Finally it might be an oversight in the loss calculation on our part. Perhaps we have forgotten a factor somewhere. This is not really consistent with the rest of the results, and the quality of the predictions, an example of which is in Fig. 8.

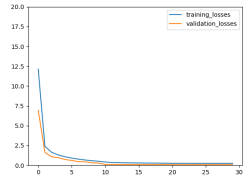

Fig. 7: Training and validation losses with learning rate 0.005 and batch size 8, unusual behavior from validation loss relative to train loss.

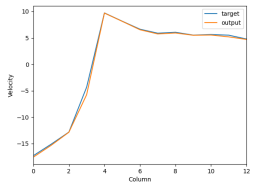

Fig. 8: Example prediction and label with learning rate 0.005 and batch size 8 on the validation set.

C. Testing

	Average testing loss	Standard deviation on the testing loss
Model on synthetic dataset, learning rate 0.04 and batch size 8	0.820	0.761
Model on real dataset, learning rate 0.005 and batch size 8	0.108	0.170

TABLE IV: Testing losses

The results of our best models on the testing set are in table IV. For both datasets, the testing loss was very near or under the best validation loss; This indicates a good robustness from our models against never-seen before datapoints.

V. Conclusion and Outlooks

To conclude, we have managed to implement, train and optimize neural networks to analyze GPI images, with the goal to estimate the average vertical velocity of the plasma structures. We also have created tools to generate synthetic data, for helping in training and optimizing these models.

We have obtained good results on the real dataset, even though the small amount of labels and the strange behaviour from the losses still ask how general that model really is. The results on the synthetic dataset were more mitigated; the model clearly learns, but there isn't enough datapoints to get truly good results out of it.

Our main obstacles were the computing power available to us, and the strict deadline for which we had to finish the project. We couldn't generate as many synthetic datapoints as we had wished, as seen above, but we were also only able to optimize some of the meta-parameters, and had to leave some like the amount of images per datapoint, the step size or the gamma unchanged.

Finally, an objective we couldn't achieve was to train and optimize a model on a mix of both datasets in an effort to correct their respective flaws. We still have coded and included

the necessary dataloader, but the model still needs to be trained and optimized.

The next step would have been concretely to: generate more synthetic data, try to get better models by playing with more hyperparameters, and finally analyze if the synthetic data can be used to enhance the performance of a model on real data.

REFERENCES

[1] "Tokamak article on wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Tokamak

[2] M. J. Park and M. Sacchi, "Automatic velocity analysis using convolutional neural network and transfer learning," 2019. [Online]. Available: https://library.seg.org/doi/10.1190/geo2018-0870.1

[3] "Tokamak consultancy." [Online]. Available: https://innovation.ox.ac.uk/academic-case-study/tokamak-consultancy/

[4] S. J. Zweben, J. L. Terry, D. P. Stotler, and R. J. Maqueda, "Invited review article: Gas puff imaging diagnostics of edge plasma turbulence in magnetic fusion devices," 2017. [Online]. Available: https://doi.org/10.1063/1.4981873

[5] "Models definition and source code." [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html#video-classification

[6] 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.

[7] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," *arXiv preprint*, 2018. [Online]. Available: https://arxiv.org/abs/1711.11248