Using Deep Learning to Predict Solar Far-Side Magnetic Flux From Far-Side EUV Flux

Team member: Ruizhu Chen (SUID: rzchen)
Code link: https://github.com/rzchen26/CS230_project

1. Introduction.

Space weather influences human activities such as spacecraft and astronauts, satellite communication, commercial airlines, etc. Forecasting of space weather is important, and the most fundamental observable in need is solar magnetic field, which is the ultimate root of all solar activities. While the solar magnetic field on the near-side (side facing Earth) is monitored, the far-side magnetic field is not available but as important, because first, the Sun is rotating, whatever happens on far-side will be facing us in a few days; second, whole-sun inputs are necessary for certain space-weather related simulations.

Although the far-side magnetic flux is not observed, the far-side extreme ultraviolet (EUV) flux is observed by ***NASA/STEREO*** mission, therefore, a "translation" between EUV flux and magnetic flux is valuable, which can be learned using near-side observations. In this work, I take the near-side EUV image as input (X), and use neural network to output the predicted near-side magnetic flux image ($\hat{Y}$). By comparing with the observed near-side magnetic flux image (Y), I train the neural network, which can be applied to translate far-side EUV images into the far-side magnetic flux images in the long run, for forecasting or research purpose.

2. Related works

Previously, the far-side magnetic field can be generated by magnetic flux transport models[1,2,3], which basically uses the magnetic fields observed certain days ago when they haven't rotated to the far-side yet and migrate them. However, this method cannot capture the evolvements of active region. Now, for the first time, we propose the idea to use the EUV flux to generate proxies of magnetic flux, by machine learning techniques. The relation between EUV flux and magnetic flux has only been analyzed statistically[3], where an approximate power law is found between the two total fluxes in sunspot areas. Now, we're to learn a direct image-to-image relation between these quantities by use of neuro network. Image-to-image networks are widely used in machine learning, with many models being developed. To choose a model, I notice a particular task of this work is that both the general pattern and the fine structures needs to be reproduced, so I adopt the "Unet" neuro network model[4,5], which combines local and global spatial information effectively.

3. Data Collection and Preprocessing

I obtain a total of about 10,000 pairs of images, with abut 600 each used as test and development sets. The raw data are from ***NASA/SDO*** mission[6,7], with 7-year continuous data publicly available from http://jsoc.stanford.edu/. Raw images are $4096\times4096\times1$,

they are (1) prescreened for bad images, (2) calibrated in coordinates so each pair are of exactly same time and location, (3) downsized to 512×512×1, (4) log scaled and normalized, (5) randomly flipped during training for data augmentation. Part of the processing are illustrated below for one date (2013 Aug 1).

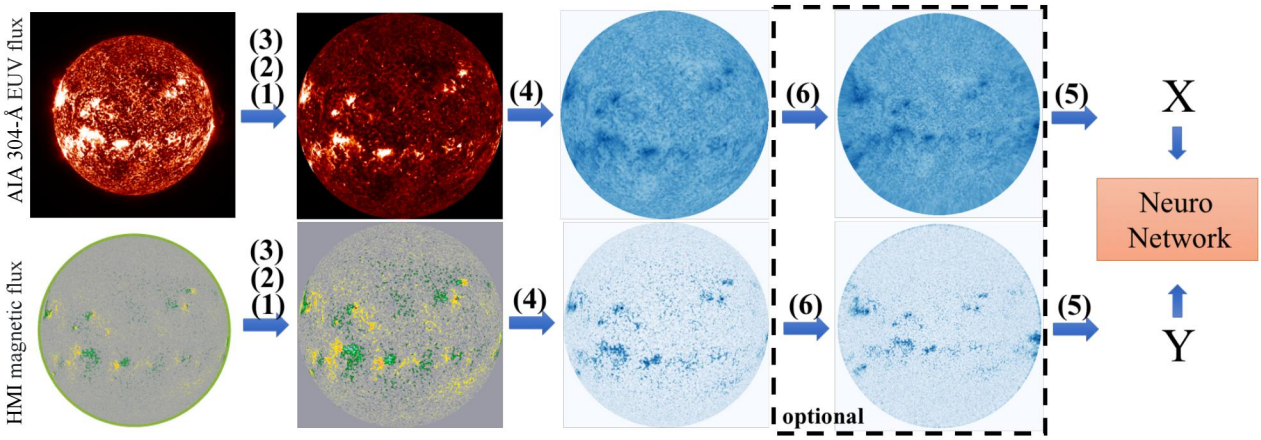

Figure 1. Data processing pipeline. The left most are the raw images.

The log-scaling in (4) is applied to make the problem less non-linear. The original images (both X and Y) are largely exponentially distributed in pixels, which would make the extreme bright points in images dominating the loss function, and also make the network relying on the non-linear part of activation function. Both X and Y are log scaled, so as to be close to normal distribution, and then normalized. The transformation is:

$$\textbf{Scaled data} = \textbf{Log}(\,|\textbf{original data}| + 1)/10$$

Besides the above processing, I also generate a parallel data set for comparison, on which one more step, (6) Postel's transformation is applied. This is to test one concern I had, that our data are spherical data with non-uniform pixel size -- a pixel near the limb is 10 times larger in actual size than a pixel in the center, so the weight share of convolutional neuro-network may not be appropriate. The transformation stretches the images to nearly (but not exactly) uniform in actual pixel-size. I briefly discuss whether this transformation is necessary in Section 5.

4. Method

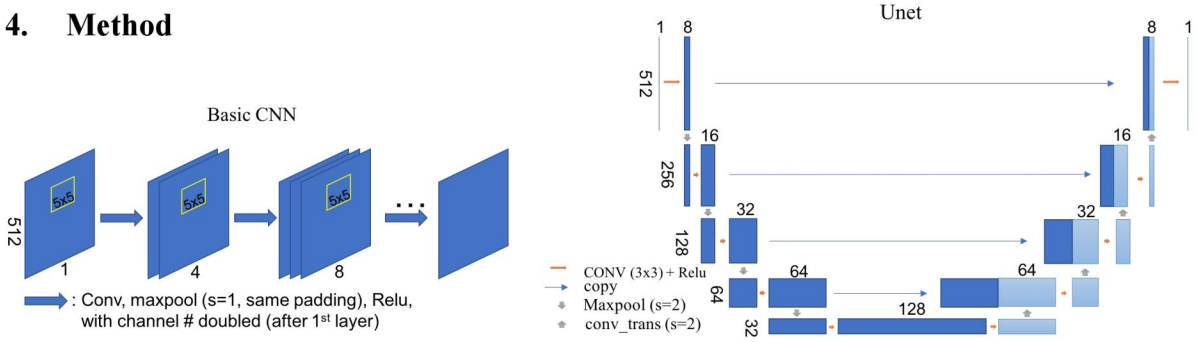

Figure 2. Architectures of neuro networks using basic CNN(left) and Unet (right).

Basic sequential CNNs was first used for initial trials, and are kept here as a reference for comparison; for the final product, Unet-architecture neuro network is used. Both networks, with their channel number, kernel size, and the image size of each layer, are illustrated in the above figure labeled. Basically, the sequential CNN applies convolution to the image and add a little non-liner effect in each layer when the it processes; while the Unet architecture does one more thing, that as the images downsize, the high-resolution data are saved and will be fed into later layers again when the images are up-sampled to the same size. This way, intermediate images of different sizes are relatively independently connected, so that small-scale and large-scale features can be combined as the target image requires.

I apply gradient descent with Adam algorithm and Batch normalization to minimize the Loss function, which is defined as the mean squared error $\frac{1}{2}||(\hat{\boldsymbol{Y}} - \mathbf{Y})||^2$, to minimize the difference between the prediction and the target on pixel-to-pixel level. The primary metrics used is the cosine similarity between $\hat{\boldsymbol{Y}}$ and $\mathbf{Y}$.

5. Experiments and Results

5.1 Hyper-parameters

During training, the primary hypermeter I tuned is the learning rate, which start from 0.01, then manually decayed when obvious zig-zag of loss function appears, after approximately 100 epochs (depending on models). Most of time alpha is between 0.01 or 0.001. I also tested the balance between the number of layer vs. number of channel vs. input image size, all of which helps performance but are limited by computation power and storage. It is found in my case a larger input image size is more efficient, then the number of layer, and then the number of channels.

5.2 Statistics

	Model	Loss (train)	Loss (dev)	Loss (test)	**Similarity** (train)	**Similarity** (dev)	**Similarity** (test)
Basic CNN	2-layer (a)	0.005446	0.005457	0.005365	0.853590	0.854364	0.855026
	3-layer	0.004908	0.004845	0.004770	0.867097	0.866880	0.869422
	4-layer (b)	0.004839	0.004775	0.004703	0.868996	0.867097	0.869991
Unet	4-Layer	0.003722	0.003758	0.003715	0.897562	0.896574	0.896705
	5-Layer(c)	**0.003476**	**0.003510**	**0.003505**	**0.904201**	**0.903930**	**0.902053**

Table 1. Performance of models on train/dev/test sets, respectively. The best model is highlighted. Results of the (a)(b)(c) model will be visualized in Section 5.3.

The statistics of the Loss (mean squared error) and metrics (cosine similarity) using the spherical dataset (without Postel's projection) are shown in the table below. For basic CNN, I've tested 2, 3, and 4 (blocks of) hidden layers with doubly increasing channels to

see how well the problem would work with a simple structure. For Unet, the 5 Layer means the image sizes are downsized and upsized 5 times each, from 512 to 32 and back. It is equivalent to 10 layers if counting the sequential convolutional layers. The 4-Layer Unet used starts from the (downsized) 256x256 image directly, for faster computation. Unet turns out more efficient in speed because of the down-sampling. The 5-Layer Unet is comparable in integration speed with the 4-layer basic CNN, despite that it actually has 10 sequential convolution layers.

From the statistics, clearly the Unets works better than basic CNNs, and the best is the 5-Layer Unet. The mean squared error is around 0.0035 (out of 1), which is quite good for regression task, and the cosine similarity between prediction and target is above 0.9. Unet is expected to work better because the Unet handles the local and global information more effectively, which is suitable for our problem. Also, I didn't see obvious overfitting in this case, since the performance in train/dev/test sets are similar.

5.3 Comparison with and without Postel's projection

I compare the performances of each model on the spherical data set vs. the dataset with extra coordinates transformation. For each model, two set of weights are trained separately for the two dataset, but with transfer learning in the initialization of the second dataset using results from the first dataset, to save computation. From the table below, it is shown that the transformation helps when the model is very simple, i.e. basic and shallow CNNSs, but when the model is complex enough to learn the geometry, this transformation is unnecessary. In particular, when the model is complex enough, the transformation seems to even hurt the performance. This is not because the transformation itself, but because the side effect of oversampling the limb data, which will be explained in Section 5.3 when the results are visualized.

Model		Loss		Similarity	
		Spherical	**Projected**	**Spherical**	**Projected**
Basic CNN	2-layer (a)	0.005457	**0.004761**	0.853590	**0.870785**
	3-layer	0.004845	**0.004556**	0.867097	**0.874010**
	4-layer (b)	0.004775	**0.004549**	0.868996	**0.876295**
Unet	4-Layer	**0.003758**	0.003962	**0.897562**	0.890148
	5-Layer(c)	**0.003510**	0.003907	**0.904201**	0.891776

Table 2. Comparision of performance of data sets with and without Postel's projection.

5.4 Visualization

I pick a random example in test set (2013 Aug 1) to show the predicted images vs the target images. I show results of three selected model from simple to complex: (a) 2-layer CNN (b) 4-layer CNN (c) 5-Layer Unet. The more complex the network is (between columns), the more details and fine structures are captured.

The visualization also compares results with and without the coordinates transformation (between rows). When the model is too simple (a,b), the spherical results didn't quite capture the background small-scale features which should shrink from center to limb, while the Postel projection has made the problem easier by approximately uniform the pixel size. However, the transformation over samples too much near the limb in both the X and Y images, as can be seen in Figure 1, lowering the quality of the input data near the limb, and therefore the learning of limb area performs poorly, as seen below. Therefore, Postel's projection is not used in the final result.

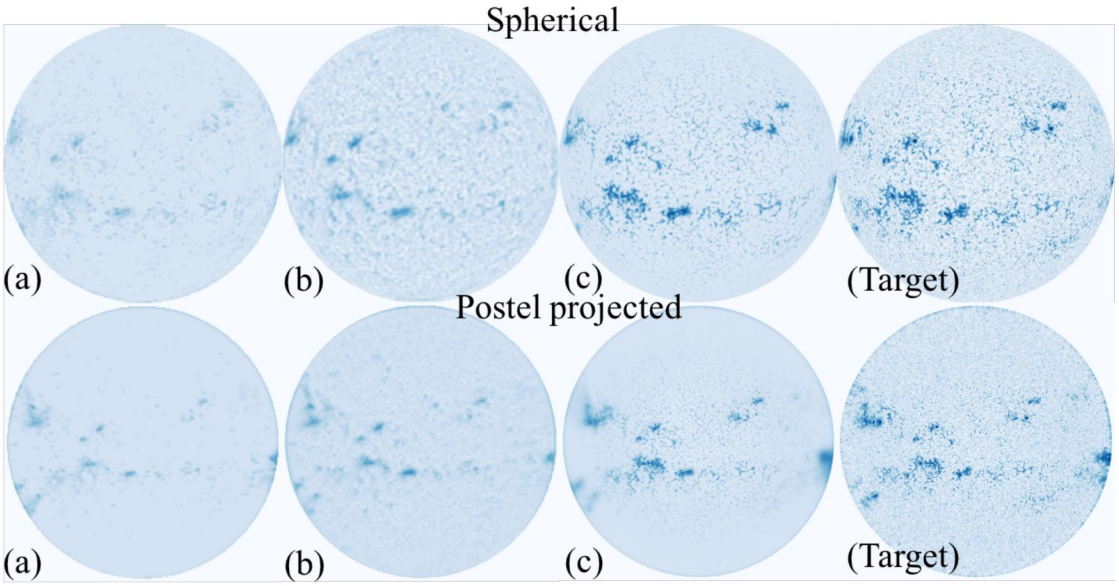

Figure 3. Predicted images for a date in test set using selected models, for input data with and without

Finally, the best prediction I got is the (c) of first row, i.e. 5-Layer Unet using spherical dataset.

6. Conclusion

Our best result, using 5-layer Unet on spherical dataset, transform the solar near-side AIA EUV (304 A) image to the near-side HMI magnetic flux image well, with a cosine similarity above 0.9 and mean squared error around 0.0035 (out of 1). This model can be applied to the far-side EUV flux maps to produce proxies of the solar far-side magnetic flux using STEREO EUV flux as well, which is valuable for space weather forecasting. Next, we'll analyze the results in scientific contents, and compare with the magnetic flux transport model, to examine the robustness of machine-learning approach in producing the far-side magnetic field proxies.

Contribution: I acknowledge helpful scientific discussions with my research advisor Dr. Junwei Zhao on the problem setup, data generation and interpretation, results analysis and future work. The technical work is done myself. The code is based on the course example code for tensorflow.[8]

Reference:

[1] Wang, Y-M., A. G. Nash, and N. R. Sheeley. "Magnetic flux transport on the Sun." Science 245.4919 (1989): 712-718.

[2] Schrijver, Carolus J., and Marc L. DeRosa. "Photospheric and heliospheric magnetic fields." Solar Physics 212.1 (2003): 165-200.

[3] Ugarte-Urra, Ignacio, et al. "Magnetic flux transport and the long-term evolution of solar active regions." The Astrophysical Journal 815.2 (2015): 90.

[4] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

[5] Çiçek, Özgün, et al. "3D U-Net: learning dense volumetric segmentation from sparse annotation." International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham, 2016.

[6] Scherrer, Philip Hanby, et al. "The helioseismic and magnetic imager (HMI) investigation for the solar dynamics observatory (SDO)." Solar Physics 275.1-2 (2012): 207-227.

[7] Lemen, James R., et al. "The atmospheric imaging assembly (AIA) on the solar dynamics observatory (SDO)." The Solar Dynamics Observatory. Springer, New York, NY, 2011. 17-40.

[8] CS230 Course Project Example Code, https://github.com/cs230-stanford/cs230-code-examples/tree/master/tensorflow/vision