

Particle streak velocimetry using Ensemble Convolutional Neural Networks

Alexander V. Grayver · Jerome Noir

Received: - / Accepted: -

Abstract Insert your abstract here.

Keywords Streak analysis · Neural Networks · Turbulent Flow

1 Introduction

Particle Image Velocimetry (PIV) is arguably the most widely used technique to quantitatively study experimental flows [11]. A common work-flow would consist of seeding flow with luminescent particles and taking pairs of pictures with a known short time separation to capture an instantaneous flow state. By splitting a pair of images into (possibly overlapping) windows and cross-correlating between them allows one to infer the direction and magnitude of the flow. With modern computers and digital cameras, PIV has experienced wide adoption in the scientific community **add refs**. For cross-correlation to work properly, one has to ensure that the exposure time is short enough such that particles do not move more than a few pixels and the two images have clearly identifiable correlations, thus the mean motion has to remain predominantly linear. Violating this condition, for instance because of insufficient laser intensity or too fast flow, results in so called streaks - traces of particles.

Streaks in the PIV images are commonly considered an experimental failure since they render cross-correlation techniques less efficient or even inapplicable.

A. V. Grayver
Institute of Geophysics, ETH Zurich
Sonneggstrasse 5
8092 Zurich, Switzerland
Tel.: +41-44-6333154
E-mail: agrayver@erdw.ethz.ch

J. Noir
Institute of Geophysics, ETH Zurich
Sonneggstrasse 5
8092 Zurich, Switzerland
Tel.: +41-44-6337593
E-mail: jerome.noir@erdw.ethz.ch

When flow velocity imposes constraints for which the camera and light source at hand cannot capture steady particles, one either has to use a higher speed camera or/and a stronger light source. Both of these solutions quickly hit financial and safety constraints, which often cannot be overcome in academia. Although not suitable for conventional cross-correlation methods, streak images do contain information about the flow. Even though an image with streaks does not contain information about direction of the underlying flow, information on the mean velocity magnitude and azimuth can be inferred from streaks themselves. To the best of authors' knowledge, there have been no attempts to do this before, which motivated us to design a method for extracting information about the flow from streaks.

To this end, we applied an ensemble of Convolutional Neural Networks (CNN) trained on streak images to draw a statistical prediction of the displacement magnitude and corresponding azimuth. The potential of CNNs has long been recognized for applications related to the face and handwriting recognition [7, 12], although their power has been fully discovered only recently when deeper (that is, with more layers) networks became feasible to train within reasonable amount of time, mostly due to emergence of Graphic Processor Units (GPUs) [6, 5]. An interested reader is referred to a recent overview of the deep learning with important development milestones listed [8].

Before we proceed to describing methodology and results, it is worth to mention that a crucial ingredient to a success of CNNs is a suitably large training set, which a CNN is supposed to learn from without facing an over-fitting. From this perspective, streak analysis appears almost an ideal problem. Similar to PIV, we make a reasonable assumption that within a sufficiently small interrogation window velocity exhibits simple functional form (for instance, constant or linear function). This allows us to quickly generate an unlimited number of training images. Adding variability in number of streaks per window, their thickness and intensity will well mimic a realistic experimental scenario. Therefore, we can exhaustively sample the parameter space and feed the network with as many sample as needed to attain an acceptable accuracy. The specific numbers and parameters will be discussed in the corresponding sections below.

2 Methods

2.1 Problem setup

We aim at inferring the displacement and azimuth from a gray scale image of size $n \times n$ pixels depicting streaks (Figure 1). In real settings, these images are excerpts (obtained, for instance, after applying a sliding window) of a larger $N \times N$ pixels image taken by a camera. For purposes of our experiments, we chose $n = 48$, but the approach presented below is general. We assume that within $n \times n$ window, the velocity is a constant, thus streaks have the same length and direction with small variations, treated as noise. In contrast, the number of streaks ($[2, 10]$), their color intensity ($[200, 255]$, where 0 is black and 255 is white) and thickness ($[2, 4]$ pixels) vary. Additionally, streaks are allowed to go outside the image, which is often the case in real settings, thus a CNN is supposed to be resilient to such cases. Once streaks are generated, they are blurred with a Gaussian kernel of 3×3 pixels.

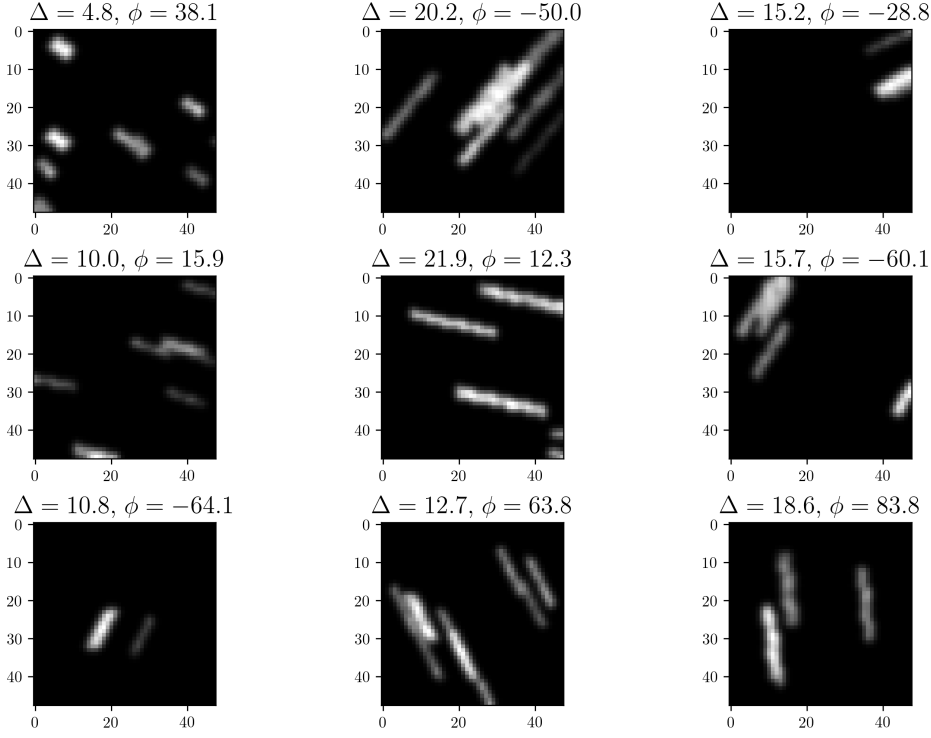


Fig. 1 Subset of generated streak images used for training and validation with corresponding displacements (in pixels) and azimuth angles (in degrees) given in titles.

The maximum displacement, r_{\max} , of a particle within an image is assumed to be half of the chosen window size, that is $r_{\max} = 24$ pixels in our case. Each streak is thus characterized by two numbers Δ, ϕ , denoting displacement and azimuth, respectively.

To generate train and validation images, we sampled Δ and ϕ from the uniform distributions

$$\Delta \sim U(1, r_{\max}) \quad (1)$$

and

$$\phi \sim U(-\pi/2 + \delta, \pi/2), \quad (2)$$

respectively. Here, $\phi = 0$ corresponds to the positive x -axis and increases clockwise. Note that limits for ϕ cover only half of the circle minus $\delta = 5^\circ$ on one end to avoid ambiguity with respect to the direction. Figure 1 shows a selection of streak images generated using the stated procedure. The outlined procedure enables generation of millions of images in just a few minutes on a regular PC.

2.2 Network architecture and implementation

We aim at building a regression CNN that, given a single window image, outputs two real valued numbers, corresponding to the displacement and angle. To this

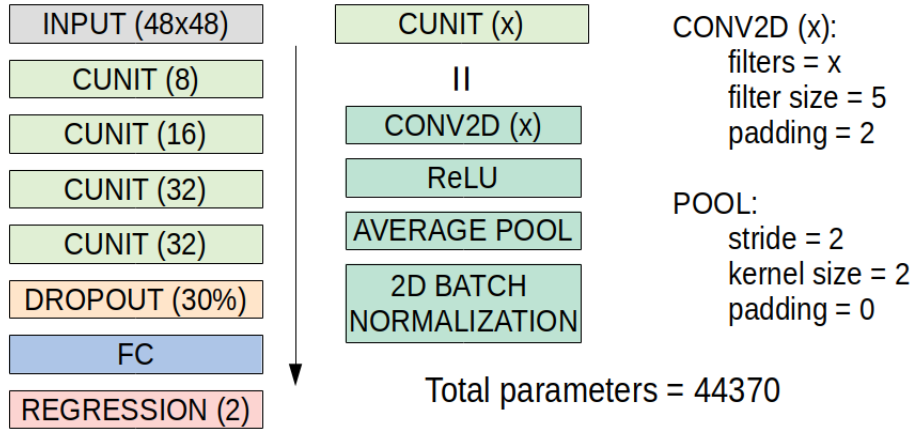


Fig. 2 Architecture of the network. Left: sequence of layers and corresponding parameters in the brackets. Each CUNIT layer consists of four actual layers listed in the middle column. Relevant layer parameters are given on the right. Total number of learnable parameters in the network is 44370.

end, we have created a network with the architecture shown in Figure 2. As can be seen, four convolutional units form the core of the network, each consisting of a convolutional layer, activation layer (in this case, rectified linear unit), average pooling layer served to downsample the input followed by a batch normalization layer. An increasing depth of the convolutional layers is a common practice which is aimed at giving a network freedom to learn more complex features in the input.

To prevent overfitting and improve training speed, we use batch normalization at the end of each convolution unit [4] and dropout with the 30% drop fraction [13]. Additionally, following common practices, the outputs are standardized and centered. The input grayscale images have a single eight bits channel. We normalized them by 255 to stay in the $(0, 1)$ range.

Whereas there may exist more efficient architectures, we found that for our problem increasing depth of the network has not improved overall performance significantly, nor did increasing and/or decreasing filter sizes. Additionally, we found that, in contrast to majority of reported CNN architectures, using the average pooling instead of the max pooling results in higher accuracy.

To increase accuracy and robustness of the prediction, we trained an ensemble consisting of ten networks [3]. All networks have the same architecture and parameters, but were trained using different randomly chosen initial weights [1, 2] and random shuffling to form a unique sequence of batches supplied to an optimizer. The outputs of all models are averaged to get the final prediction. An ensemble also enables us to calculate a variance, thereby providing a proxy for the uncertainty of a prediction.

Finally, the network was implemented using the PyTorch library [9]. The complete implementation can be found using the link below.

3 Results

3.1 Network training and accuracy

We generated one million images (e.g., Figure 1) of which 75% and 25% were used for training and validation, respectively. Each network in the ensemble has been trained for 100 epochs. The ADAM optimizer with an initial learning rate value of $1e-3$ was used. The learning rate was halved if no sufficient decrease in the validation loss was observed for the past 10 epochs. To make the training more resistant to potential outliers (e.g. imagine a situation when all streaks cross the window), we chose to minimize the Huber loss [1] rather than conventional least-squares functional. Huber loss represent a hybrid $L_2 - L_1$ distribution with a high tolerance to the presence of long tails in the original distribution. Training and validation losses for all networks are shown in Figure XX. We see a typical behaviour of exponential drop in the loss during first epochs, followed by slow flattening curves. No pathologies between training and validation loss were observed, specifically the validation loss remains a bit higher than the training loss, indicating that the network learns some generic features of the dataset. Training of each network took 90 minutes on a single CPU-GPU system. This modest time suggests that training networks for window sizes larger than the one assumed here is feasible.

Figure 3 shows RMSE values for all networks separately as well as the RMSE of the CNN ensemble. It is evident that on the ensemble performs up to 20% better than a single network. CNN ensemble achieves the accuracy of 1.05 pixels for the displacement and 8.5° for the angle predictions, respectively. The latter may seem like a large error, however one should realize that we have the 48 pixels window where limitations in recognizing small rotations are inherent, especially when streaks are short.

Figure 4 show a random selection of validation images with the corresponding CNN ensemble predictions and true values. We see that the ensemble CNN has learned to make accurate prediction even for complex situations when some streaks overlap, join or leave the window. Additionally, Figure 5 shows nine worst predictions for the displacement. Most of these cases are associated with situations where originally longer streaks have all been displaced outside the window or multiple streaks coincidentally joined and formed what appears to be a longer streak. Note that such situations do not represent a failure of the CNN, but rather limitation of the window-based approach adopted in this work. Such situations are likely to occur in experimental images and unfortunately they leave little chance for a CNN (or any other sliding window approach) to make an accurate prediction. It is worth to note that despite inaccurate displacement prediction, the angle is predicted accurately in most of these cases. We will discuss possible ways to mitigate this in the Conclusions section.

3.2 Application to a numerical simulation

After confirming the excellent performance of the network on the validation set, we turn to a discrete numerical simulation (DNS) of a complex turbulent flow as presented recently by [10]. Figure 6 shows original and window-averaged displacement maps calculated from the DNS velocity field with the 50% window overlap.

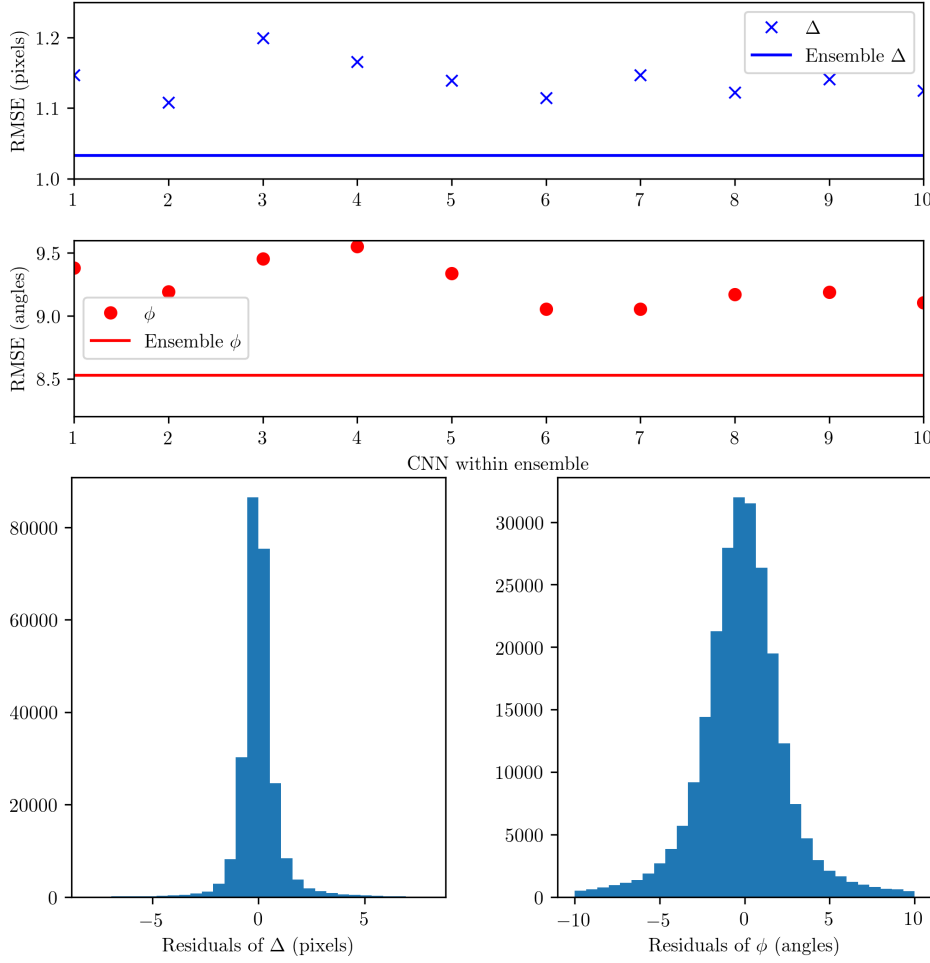


Fig. 3 Top: RMSE values of individual networks (markers) and CNN ensemble (line) for displacement Δ and angle ϕ calculated on the validation set. Bottom: histograms of the residuals for displacement and angle.

The window averaged displacement is the best reconstruction we can achieve with the 48x48 pixels window and 50% overlap used in this work.

To mitigate the effect of random streak seeding, we generated 30 streak images using the original DNS velocity (Figure 6, left) and randomly varying streak density, intensity and thickness. The trained CNN ensemble was applied to all images and the resulting averaged reconstruction of both displacement and angle are shown in Figure 7. We see that the reconstructed images recover the general structure of the flow very well. The only negative effect we see is that the ensemble CNN struggles to predict large displacements. When the displacement reached half of the window size, CNN systematically predicts smaller displacement. The reason for this is not a deficiency of the CNN, but rather the fact that for displacements as large as half the window size it becomes inevitable that most of the streaks go outside the window, thus predicting a smaller displacement is the most ratio-

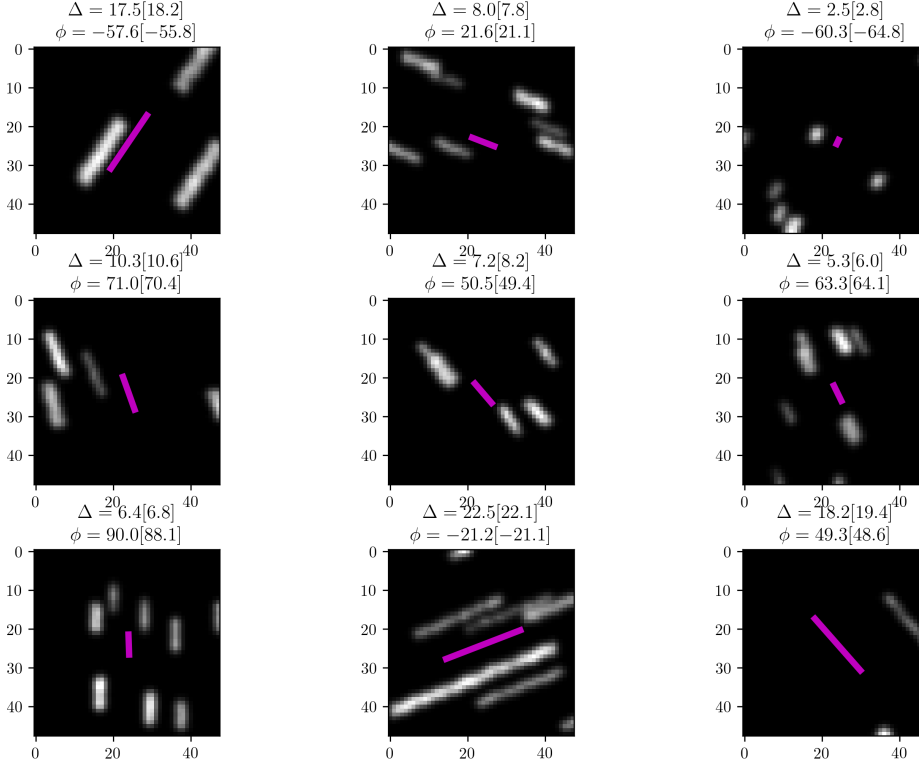


Fig. 4 Images with the predicted displacement and angle depicted as magenta lines. The true values are given in titles and predicted values in brackets.

nal output. To solve this problem, one can apply a larger window (for instance, 64x64 pixels), although this will reduce the overall resolution. In real experimental settings, it may be advantageous to apply several window sizes and either choose one or combine obtained results.

Finally, Figure 8 shows one of the streak images with true and predicted streaks for each window position. Generally, ensemble CNN performs very well with some difficulties in regions of large displacements as discussed above and across areas of vertical shearing, where angle predictions become locally less accurate.

3.3 Application to an inertial flow experiment

Ensemble CNN speed is 7 images/sec on the single NVIDIA GeForce GTX 1070Ti. Using more modern GPUs will enable real-time prediction.

4 Conclusions

We presented an open-source ensemble CNN aimed at quantitatively studying complex experimental flows using streak images. The presented approach can be

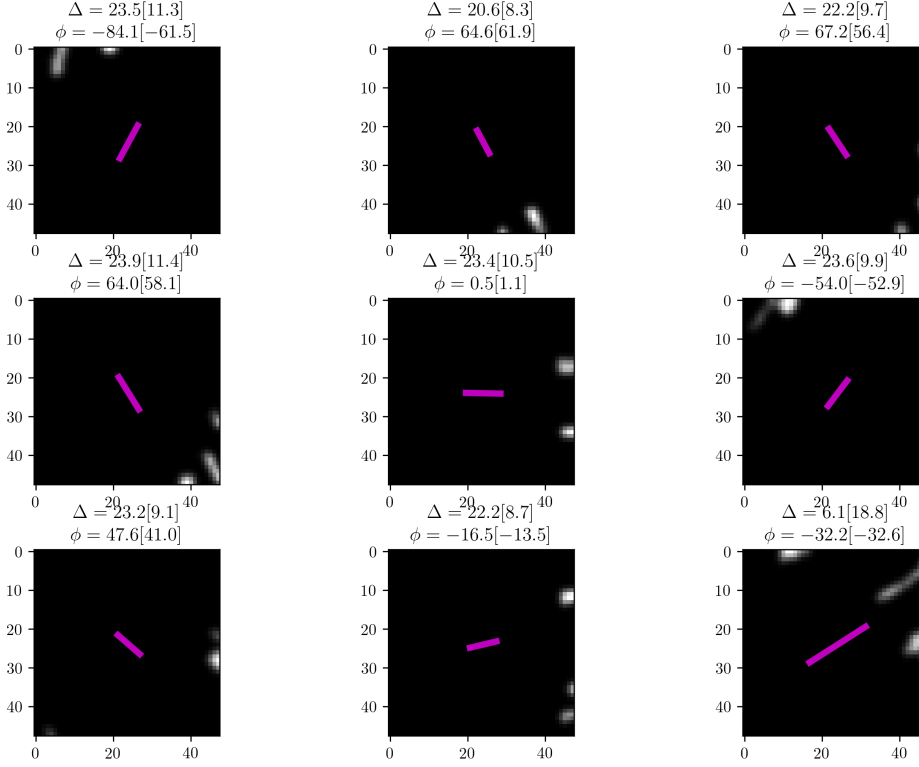


Fig. 5 Images with the predicted displacement and angle drawn as magenta lines. The true values are given in titles and predicted values in brackets.

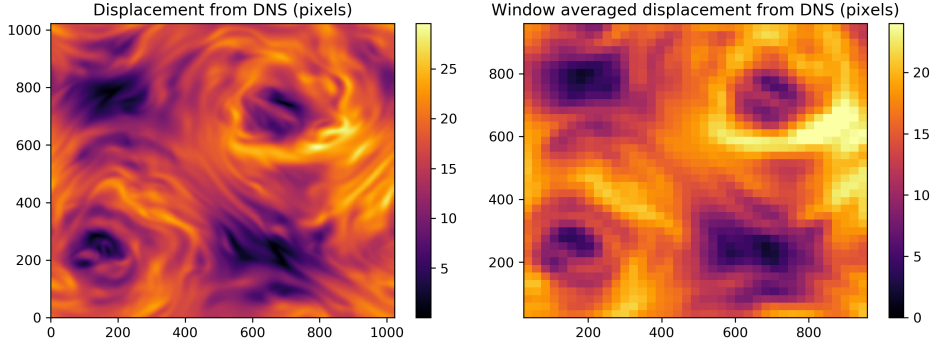


Fig. 6 Original (left) and window averaged (right) displacement maps of the DNS [10] used for validation.

applied in situations when classic PIV is inaccurate or not possible due to experimental setup limitations. The advantage of the presented approach is the ease of training/validation set generation as well as recovery of the prediction uncertainty through application of the ensemble technique. We foresee that the approach may need an experiment-dependent tailoring to achieve the best performance. Accord-

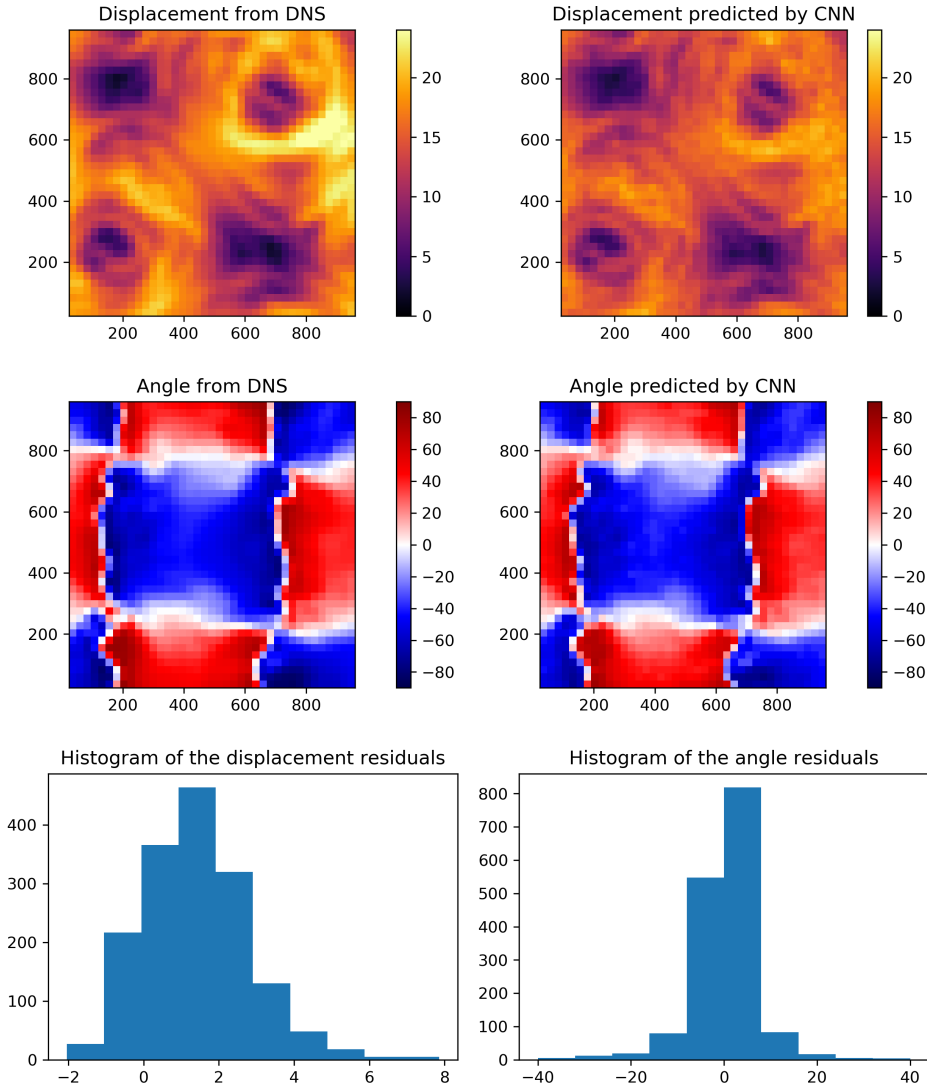


Fig. 7 Original (left) and window averaged (right) displacement maps of the DNS [10] used for validation.

ing to our experience, this only takes a single day on an average PC equipped with a modern GPU.

The potential extensions of the presented approach may include the use of multiple window sizes, more complex functional parameterizations of the velocity field within the window (e.g., polynomial). Finally, a completely different approach based on image segmentation and object localization and/or tracking can be applied to extract information about individual streaks and their evolution. We expect to investigate these techniques in details in our future works.

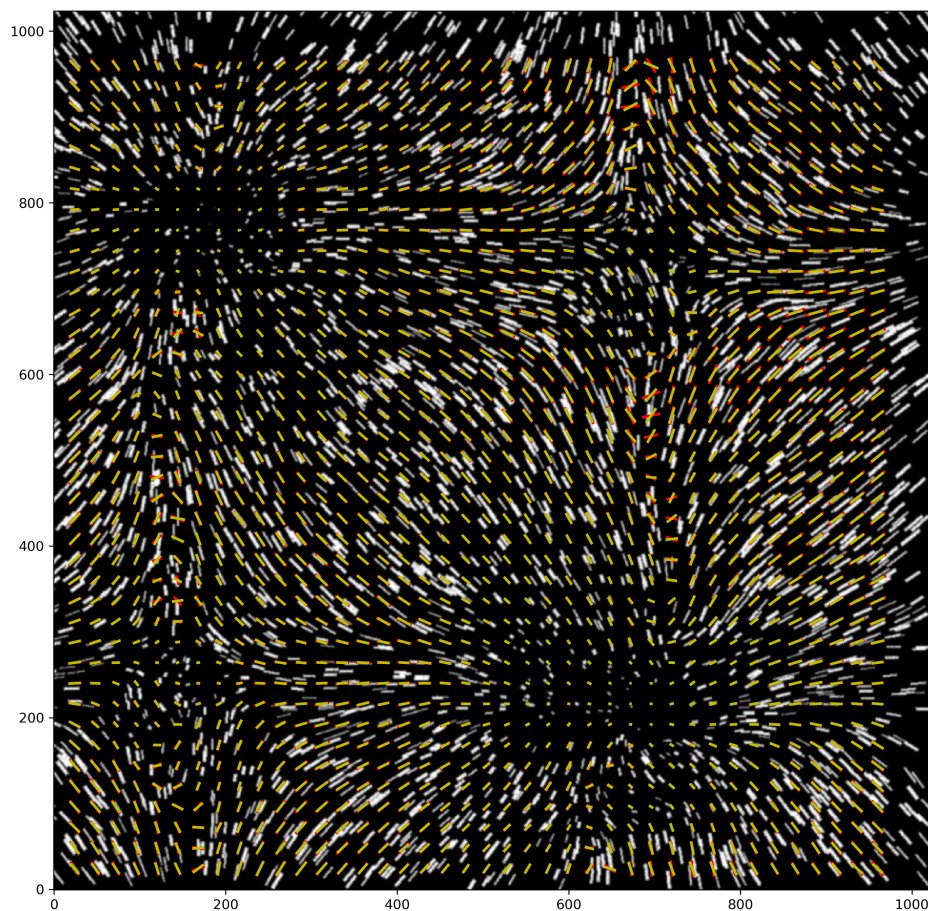


Fig. 8 Streak image generated from the DNS with true and predicted displacement and angle shown as red and yellow lines, respectively.

Acknowledgements We thank authors of PyTorch, matplotlib and h5py libraries for making them available, Meredith Plumley for sharing her DNS data, Adrian Tasistro-Hart for a number of insightful discussions on CNNs. The Jupyter Python notebooks of all programs used in this study can be found here <https://github.com/agrayver/streakcnn>.

References

1. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256 (2010)
2. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034 (2015)
3. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)

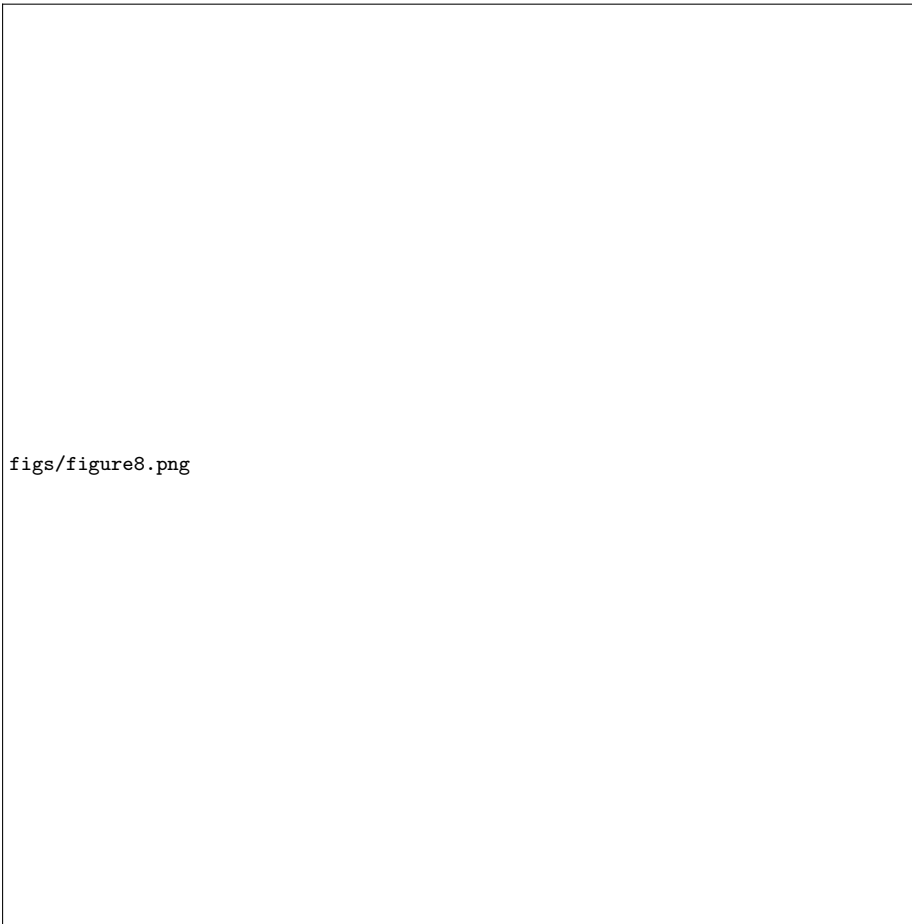


Fig. 9 Streak image generated from the DNS with true and predicted displacement and angle shown as red and yellow lines, respectively.

4. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
5. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 1725–1732 (2014)
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
7. Lawrence, S., Giles, C.L., Tsoi, A.C., Back, A.D.: Face recognition: A convolutional neural-network approach. IEEE transactions on neural networks **8**(1), 98–113 (1997)
8. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436 (2015)

9. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS-W (2017)
10. Plumley, M., Julien, K., Marti, P., Stellmach, S.: The effects of Ekman pumping on quasi-geostrophic Rayleigh–Bénard convection. *Journal of Fluid Mechanics* **803**, 51–71 (2016)
11. Raffel, M., Willert, C.E., Scarano, F., Kähler, C.J., Wereley, S.T., Kompenhans, J.: Particle image velocimetry: a practical guide. Springer (2018)
12. Simard, P.Y., Steinkraus, D., Platt, J.C., et al.: Best practices for convolutional neural networks applied to visual document analysis. In: *Icdar*, vol. 3 (2003)
13. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* **15**(1), 1929–1958 (2014)