MNRAS **000**, 1–8 (2015) Preprint 21 July 2020 Compiled using MNRAS LaTeX style file v3.0

Closed Loop Predictive Control with Convolutional Neural Networks

Swanson, R.[1]⋆ A. N. Other,[2] Third Author[2,3] and Fourth Author[3]
[1]*Royal Astronomical Society, Burlington House, Piccadilly, London W1J 0BQ, UK*
[2]*Department, Institution, Street Address, City Postal Code, Country*
[3]*Another Department, Different Institution, Street Address, City Postal Code, Country*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT
Predictive wavefront control is an important and quickly developing area of adaptive optics (AO). Through the prediction of future wavefront effects we can undo inherent AO system servo lag caused by the measurement, computation, and application of the wavefront correction. This lag can impact the final delivered science image, including reduced contrast, and inhibits our ability to reliably use faint guidestars. We summarize here a method for predictive control based on convolutional long-short term memory neural networks. Unlike previous methods which showed results based on offline images, we demonstrate their closed loop performance in simulation and show that it can both reduce effects induced by servo lag and push the faint end of reliable control with natural guidestars, improving strehl performance compared to classical methods by over 55% for magnitude 16 guidestars on an 8-meter telescope. We further demonstrate its performance on our experimental bench, showing the first experimental results for our method. Finally, we discuss the requirements and difficulties of implementing on large telescopes.

Key words: keyword1 – keyword2 – keyword3

1 INTRODUCTION

1.1 Opening Introductions

- talk about general status of AO systems and their GENERAL limitations (i.e. sky coverage, servo lag, i.e. things AO systems encounter generally that we are trying to overcome)
- CNNs can provide a unique approach at de-noising systems and extrapolation (i.e. prediction), so has potential to improve aspects of an AO system in a general by exploiting these two things
- Some work with CNNs in the past are ...; they did ....
- In this paper we apply this type of CNN, with the aims of testing the de-noising and extrapolation suppositions made above
- Consider capping off by summarizing what each section will tackle (this is a theme in these papers sometimes) end the paragraph by saying we also discuss the future of such an approach and its application for both real life systems and predictive control for high order systems with bright magnitude guidestars

1.2 Neural Networks in Adaptive Optics

Neural networks, and more recently deep convolutional neural networks (CNNs) are now well established as an effective and robust tool for many image processing problems (LeCun et al. (2015)). Their ability to learn complex and robust functions largely comes from the rich features that can be learned with a combination of using many 2D convolutions and having a very large set of training data to learn from. These methods have been adopted in many areas of scientific imaging, including the medical and astronomical sciences (e.g., Ronneberger et al. (2015); Dieleman et al. (2015)). However, while their ability to analyze, process, and improve astronomical image data have been well shown, their ability to improve the instrumentation tools themselves has been relatively overlooked.

One successful application of neural networks astronomical instrumentation has been in the field of adaptive optics, where traditional densely-connected neural networks were applied to improve on the problem of off-axis anisoplanatism (Gendron et al. (2011); Osborn et al. (2012)). In this work the authors show a successful application of their model in an on-sky environment by training on wavefront sensor slope measurements. However, because their networks lacked spatial and temporal reasoning capabilities, they weren't able to leverage information across neighbouring slope measurements or learn persistent patterns across wavefront measurements.

1.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs; Goodfellow et al. (2014)) are a more recent development in training neural networks which have greatly advanced our methods for generating realistic looking images. The key to their success is training two networks simultaneously: the generative network which learns to create realistic images, and the discriminator which learns do distinguish between real and fake images. As the two networks train, they compete against each

⋆ E-mail: mn@ras.org.uk (KTS)

other which helps the generator learn to create more realistic images that can fool the discriminator.

Typically this technique is employed to generate new, previously unseen, images either from a random initialization (Goodfellow et al. (2014)) or conditioned on some prior data (Liang et al. (2017)). Similarly, GANs have also found success in re-creating images based on the style of another (Zhu et al. (2017)), or to create more realistic training data for real-life systems (Shrivastava et al. (2017)). Here we use GANs in a novel way to facilitate training a closed loop integrator (which can be sensitive to new types of input), the discriminator acts as a prior on the output of the predictive network be statistically similar to the training data. Therefore when the loop is closed, and predictions from the network are fed back to the network as input, the input is still similar to the training data that it learned on.

1.4 Long-Short Term Memory Networks

One area where CNNs prove less efficient is for temporally correlated data. While it is possible to incorporate time dependencies in our data by passing a large batch of data through the network at once, this proves to be much more computationally expensive and therefore not ideal for applications which are run at very high speeds.

Recurrent neural networks, and later improved by Long Short Term Networks (LSTM; Hochreiter & Schmidhuber (1997)), were created to solve this weakness in traditional neural networks by passing information from the last data point to the next in time. Furthermore, in the case of LSTMs, the network creates and holds a hidden state at each layer in the network which is updated as new data is passed through the network. This enables the network to pick out features and patterns over time which it finds relevant and produce better results as more is passed through the network; this is accomplished by learning functions which choose what parts of the new data to add to the state, and which information can be forgotten. For a more detailed and illustrated description, see (Olah (2015)).

In cases where the data is both spatially and temporally correlated, such as images, these methods can be further improved by using convolutional filters (Xingjian et al. (2015)) such as those found in a typical CNN. These types of networks are aptly named convolutional LSTM networks and have been successfully used in a variety of cases such as video and motion prediction (Finn et al. (2016); Lotter et al. (2016)) and are therefore a powerful tool for analyzing atmospheric phase perturbations which have strong spatial features and are highly temporally correlated.

2 DEEP PREDICTIVE CONTROL

2.1 Network Description

Our network was designed with to exploit the natural spatial and temporal structure found in atmospheric conditions while operating as efficiently as possible to try to meet the high operating speed of modern AO systems. To achieve these design goals we use a densely connected convolutional neural network (CNN) operating on pseudo-open-loop (POL) slopes as seen in Figure 1. Densely connected CNNs have been recently shown to learn meaningful convolutional features while maintaining a smaller number of learned parameters. The skip connection, which connects the input and output of the network, reduces the complexity required of the network output by requiring the network to only generate the small change required between the input data and the future ground truth slopes. We choose to do our inference on the slope data due to it being the only available information from the atmospheric wavefront in typical AO telemetry. The large reduction in spatial dimensionality compared to full resolution atmosphere also reduces computational complexity while preserving spatial and temporal relationships in the data. This assumption holds true if a bijection between a wavefront and its calculated slopes exists which we assume to be true for this type of AO system.

As input, a set of x and y direction slopes are stacked and passed through the network as an $[N_x \times N_y \times 2]$ data point where N_x, N_y are the number of lenslets in the x and y direction. Once passed through the network, the output slopes can be converted into commands with the pre-calibrated command matrix and applied directly onto the DM.

While we have found that this network to be sufficient for producing slopes which very accurately match the ground truth data, in practice the network was insufficient for closing an AO loop. This is due to statistical differences in the distribution of data from the input, ground truth, and network output slopes which resulted in a network that performs very well for a short amount of time in closed loop before diverging. As the loop is closed with the output from our network, the new input to the network in subsequent iterations of the loop will look less and less like the training data used to create the network.

One solution would be to reduce the number of learnable paramters of our network to avoid this overfitting to the ground truth data. However, this in turn reduces the networks accuracy resulting in very minor improvements over classical methods. As a simple solution to this problem we include an additional discriminative network and loss function as a prior to encourage the network to output slopes which are close to the ground truth while statistically resembling the input data. In this way the slopes in our closed loop integrator will appear similar to the training data even if we are in entirely new simulation environments.

The discriminative network takes a set of slopes as input and attempts to discern whether those slopes were part of the original training data set or a set of slopes produced by out network. We then train both the predictive and discriminative networks simultaneously, forcing our predictive network to both predict the future slopes while also fooling the discriminator while also training the discriminator to better discern between the two. This game of cat and mouse continues until we have reached a saddle point in our loss function where the predictive network is as accurate as possible while the discriminator can not reliably tell the two sets of input apart.

The discriminative network, as shown in Figure 3, is a simple series of strided convolutional layers which take the input slopes and apply increasing numbers of convolutional filters at each layer while halving the spatial resolution at each step. This results in a final output from the convolutional layers of size $[512 \times 1]$ which is then fed into a fully connected layer which takes a learned, weighted sum of the outputs to produce a single output value of 0 or 1. Here an output of 0 means the network believes the input slopes were produced by our predictive network while an output of 1 means the network thinks it was from the original set of training data.

Each layer of the predictive and discriminative networks contain a convolutional layer with 3×3 pixel filters, a batch norm layer, and is activated with the PReLU non-linear activation layer (He et al. (2015)). The PReLU activation function introduces a learned parameter α at each layer allowing the network to adapt its activation

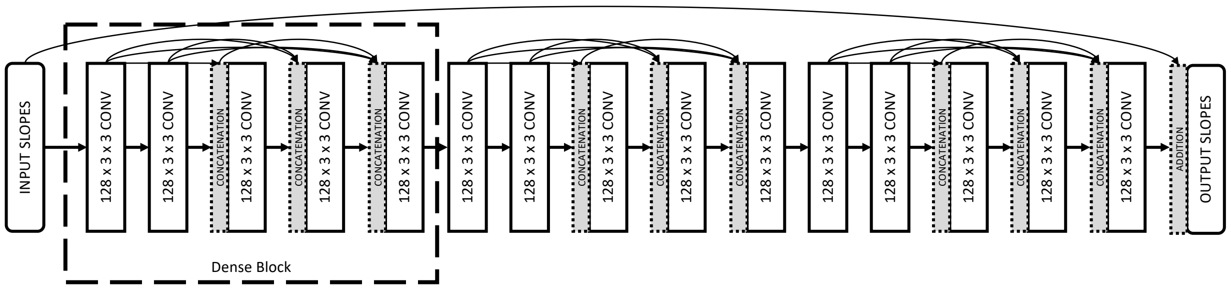


Figure 1. Overview of our predictive convolutional neural network. At each time step the latest 20 measured POL slopes are passed into the network which predicts the upcoming slopes corresponding to those from two iterations forward in time. The network consists of three densely connected blocks wherein each layer concatenates its output to the input channels of all future convolutional layers within its block. Each layer consists of 32, 3×3 convolutional filters which are applied to all of its input channels after which a PreLU non-linear activation function is applied.

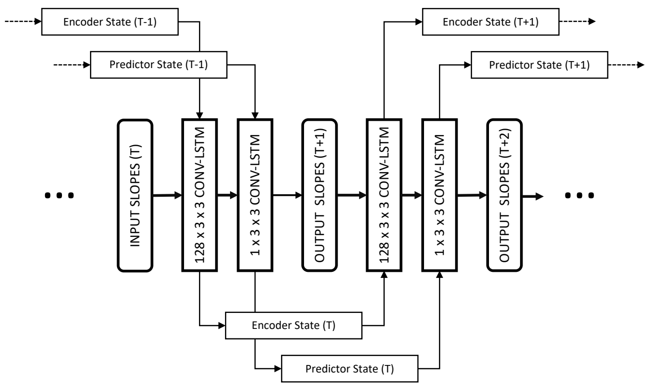


Figure 2. Overview of our predictive convolutional LSTM network. At each time step the current POL slopes and previous network states are passed into the network which predicts the upcoming slopes corresponding to those from two iterations forward in time. A single step of the LSTM network consists of two convolutional LSTM blocks, the encoder which takes a set of input slopes and convolves them with 128 pre-trained filters, outputting an encoded set of features and a state for the given iteration. Similarly, the predictor layer takes the encoded features from the encoder layer and predicts the next slopes based on that and the previous predictor state. In addition to each layers output, they also update their current state to be used in the next iteration.

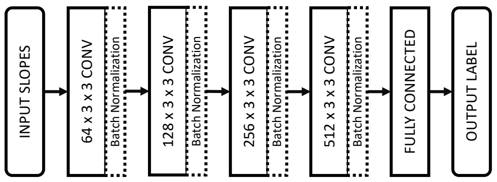


Figure 3. Overview of the discriminate (adversarial) network. During training this network attempts to label an input set of slopes as either "real" or "fake". Here "fake" refers to slopes output from our predictive network and "real" slopes refer to those passed into the network during training. Simultaneously, our network must learn to trick this network into believing its output is "real". This adversarial learning leads our network to output slopes which it would also recognize as input, allowing our network to act as a stable AO controller. The network itself consists of four 3×3 convolutional layers each an increasing number of filters. The final layer is a fully connected layer which maps the previous layers output to a single scalar value, 0 for "fake" or 1 for "real".

functions to the data during training but is fixed during evaluation. More formally it can be described as,

$$f(y_i) = \begin{cases} y_i & \text{if } y_i \geq 0 \\ \alpha_i y_i & \text{if } y_i < 0 \end{cases} \tag{1}$$

where y_i is the input channel of the i^{th} layer of the network and α_i is the learned slope of its non-linear activation. While the discriminative network increases the training complexity, it is not required during inference after the weights of the network are fixed and therefore has no impact on the run-time of the final model.

3 TRAINING

3.1 Effect of Adversarial Prior

3.2 prioreffect

As previously described, we use an additional adversarial prior to train our generative networks.

3.3 Simulation Settings

For both training and testing we simulate the Gemini telescope; an 8-metre class telescope with a 16×16 lenslet array operating in POL at 800 Hz with one additional frame of servo lag. We operated the AO with an R-band natural guide star, three layer atmosphere, and K-band science camera over a wide range of NGS magnitudes. All simulations were implemented and run using the OOMAO adaptive optics simulation software package Conan & Correia (2014).

To generate training data 20,000 independent simulations were run for 500 loop iterations each. For better sampling of possible simulation settings we randomly sample the r_0, wind speed, and direction from normal distributions, and uniformly sample an NGS magnitude between 8 and 16 for each simulation. This exposes our network to a wide variety of data during training, helping it generalize for all simulation parameters, avoid overfitting to certain conditions, and therefore work suitably well under any set of conditions.

At each simulation time step t we save the current frame-delayed POL slopes, $s(t)$, computed from the classical integrator and the current ground-truth “best fit” slopes $s^*(t)$, computed from the true atmosphere projected onto the DM. Given the current atmo-

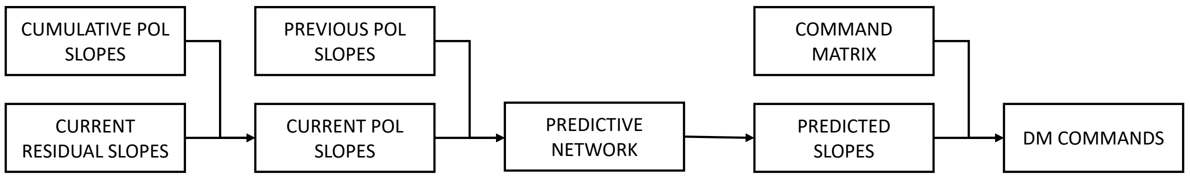


Figure 4. One iteration of our simulation when applying our predictive control algorithm. First, the POL slopes are calculated as usual. Those slopes are then passed to our learned predictive network along, with a fixed number of previous POL slopes, to predict the corrected slopes to be applied at the current iteration. To generate the accompanying mirror commands, we multiply the corrected slopes with the previously calibrated DM command matrix, resulting in the slopes to be applied to the DM.

Simulation Parameters		Values
Telescope	Diameter	8 m
	Sampling Frequency	800 Hz
	Frame Delay	2 Frames
	WFS Order	16×16
	WFS Readout Noise	$\approx 0e^-$
	DM Order	17×17
	Pupil Shape	Gemini Pupil
Three Layer Atmosphere	r_0	$\mathcal{N}(0.15, 0.02)$ cm
	Layers	3
	Altitudes	0 km 4 km 10 km
	Fractional r_0	0.7 0.25 0.05
	Wind Speeds	$\mathcal{N}(5, 2.5)$ km/s $\mathcal{N}(10, 5)$ km/s $\mathcal{N}(25, 10)$ km/s
	Wind Directions	$[0, 2\pi)$ rad
Science	Science Camera Band	K
	NGS Band	R
	NGS Magnitude	[8, 16]
	POL Gain	0.3

Table 1. Simulation parameters used for training our neural networks.

spheric phase $\Phi(t)$, as well as the command matrix M and influence function F of our calibrated DM, this can easily be calculated as,

$$s^*(t) = -\frac{1}{2}M^{-1}\left(F^{-1}\Phi(t)\right) \tag{2}$$

These two sets of data, s and s^*, then represent our input and target pairs during training. From each simulation we save 5 evenly spaced sets of 50 continuous time steps, ignoring the initial 100 time steps to remove any data generated while the loop wasn't fully closed, for a total set of 100,000 training pairs. Please refer to Table 1 for further details.

3.4 Training

During training we take 20 sequential iterations of noisy input slopes s from time t to $t + 20$ and similarly, the ground truth slopes s^* for time $t + 22$. The noisy slopes are concatenated and passed through the predictive network $\mathcal{P}$ which outputs a single set of slopes corresponding to time $t + 22$. From this output we compute the data error term as the ℓ_1 loss between $\mathcal{P}(s)$ and s^*,

$$\mathcal{L}_D = ||\mathcal{P}(s) - s^*||_1 \tag{3}$$

As described in Section e also include an adversarial loss to impose a prior on the network outputs to be as similar as possible to the training data. This takes the form of a binary Cross-Entropy loss function, penalizing outputs from the predictive network which are not labeled by the discriminative network $\mathcal{D}$ as the training data,

$$\mathcal{L}_A = -log\left(\mathcal{D}\left(\mathcal{P}\left(s\right)\right)\right). \tag{4}$$

At each training step the total loss of our predictive network is then,

$$\mathcal{L}_P = \mathcal{L}_P + \gamma\mathcal{L}_D = ||\mathcal{P}(s) - s^*||_1 - \gamma log\left(\mathcal{D}\left(\mathcal{P}\left(s\right)\right)\right) \tag{5}$$

where γ weights the importance of the two loss terms. By increasing the value of γ we put higher weight on the adversarial loss and, in turn, decrease the networks power to perfectly recreate the ground truth data. The weighting is therefore crucial for finding the appropriate balance between the two terms. Based on the results of our experiments we find that a value of $\gamma = 5$ works best and is used throughout this work.

Simultaneous to the training of our predictive network, $\mathcal{P}$, we update the descriminative network at each step based on the current weights of $\mathcal{P}$, $\theta_{\mathcal{P}}$. Similar to $\mathcal{L}_D$, $\mathcal{D}$ must maximize the likelihood that it labels both the input training data and the output of $\mathcal{P}$ correctly which can be succinctly described by,

$$\min_{\theta_{\mathcal{P}}}\max_{\theta_{\mathcal{D}}} \mathbf{E}\left[log\left(\mathcal{P}_{\theta_{\mathcal{P}}}\left(s\right)\right)\right] + \mathbf{E}\left[log\left(1 - \mathcal{D}_{\theta_{\mathcal{D}}}\left(s\right)\right)\right] \tag{6}$$

Our network was coded and trained using the Tensorflow machine learning software package. All optimizations were performed with a batch size of 32 using the Adam optimizer (Kingma & Ba (2014)) with an initial learning rate of $1e^{-4}$. The Adam optimizer is widely regarded as the current best method for optimizing deep neural networks, due in large part to its ability to tune the learning rate for each variable individually, and proved to work well in our experiments.

4 RESULTS

4.1 Testing

After training, we tested our network by using its output to close the loop in 500 additional simulations. Each simulation was randomly initialized with the same range of parameter values used for the

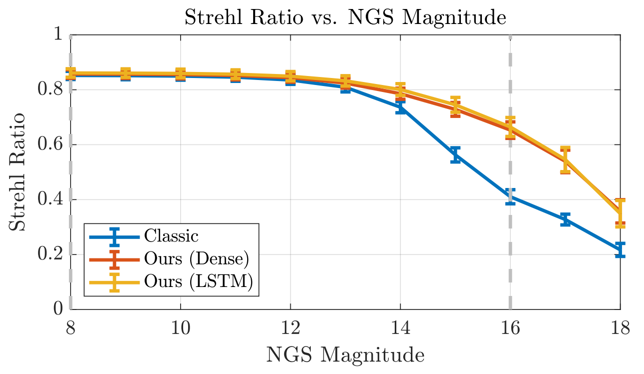


Figure 5. Closed loop performance of our network compared to a classical integrator as the NGS magnitude is varied. Our network shows considerable performance increase for faint NGS, but continues to show improvements even under ideal NGS conditions.

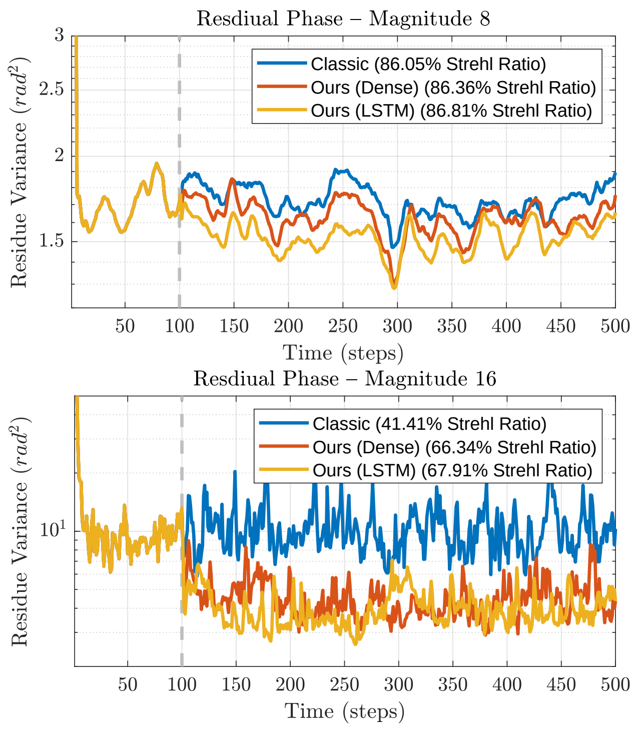


Figure 6. Residual Wavefront Analysis: Example residual measurements over the course of a simulation comparing our predictive control with a classical integrator for a magnitude 8 (top) and magnitude 16 (bottom) NGS. Our network begins predicting after 100 iterations (indicated with a dashed veritcal line) and immediately shows improvements over the classical method. Strehl ratio values are included in the legend parenthesis.

training simulations. The one exception being we run the test simulations for 1,000 time steps to demonstrate their ability to learn and operate over much larger time sequences than those used to train the network.

Because our network was trained only on converged, closed loop data, we use the classical POL integrator for the first 70 time steps before using the output from our network to close the loop. However, to initialize the state of the network we pass the computed slopes through the network starting from the fiftieth iteration.

To compare the performance of our network with the classical method, we ran the same 500 simulations with both methods

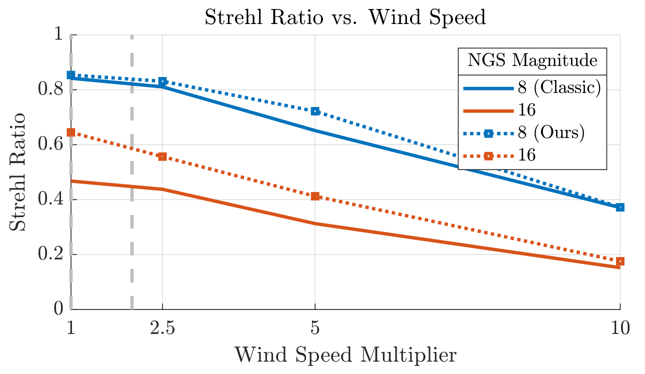


Figure 7. Closed loop performance of our network compared to a classical integrator as the wind speed is varied. While holding all other simulation variables constant, we increase the r_0 from 10 to 20cm to see its impact on Strehl ratio. Our network is better able to cope with very low values of r_0 by taking into account a history of previous slopes.

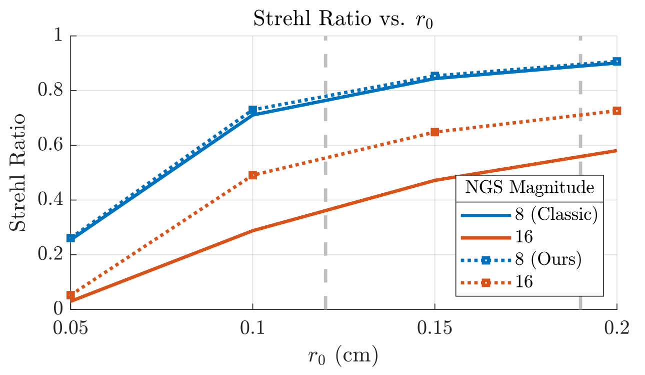


Figure 8. Closed loop performance of our network compared to a classical integrator as the r_0 is varied. While holding all other variables constant, we increase the wind speed by up to a factor of three, to show how both integrators are affected by increasing winds.

initialized with identical simulation parameters. In this way we can directly compare their performance for each simulation in addition to the aggregate performance across all simulations.

4.2 Results

In Figure 5 we show the average Strehl ratio performance for both our method and the classical POL integrator. Our method clearly improves the overall Strehl ratio across all star magnitudes; not only at the faint end where noise dominates and its effects are most significant but also for bright sources where servo-lag and aliasing are the largest sources of non-fitting related error.

Figure 6 shows the the residual wavefront error over time for a magnitude 8 and 16 NGS simulation using both integrators. Our model immediately improves the residual variance after the necessary burn-in period for the closed loop integrator to reach a relatively steady-state (indicated with a dashed vertical line). These improvements are further increased as the seeing conditions become worse by either increasing wind speeds or decreased r_0 of the system. Figure 7 compares our method with the classical integrator as the wind speed increases for all three atmospheric layers while r_0 is held constant. For all NGS magnitudes and windspeeds our method shows improvements over the classical method, with the performance gap increasing as the winds grow stronger, and for high wind speeds our method improves the quality of our seeing by the same factor as a decrease of 2 NGS magnitudes. Similarly, Figure 8 shows the performance of the two methods as the r_0 varies when wind speed and direction are held constant. Again, we see that at nearly ideal seeing conditions our method shows some strehl ratio improvements with the performance gap increasing as the seeing becomes worse at lower values of r_0.

4.3 Frequency Analysis

To better understand the performance gains of our method, we investigate the average frequency content of the residual atmospheric wavefront over the course of a simulation. At each time step we calculate the power spectral density (PSD) of the residual wavefront, applying a Hamming filter to account for high frequencies caused by the telescope pupil, and then average over the entire simulation.

Figure 10 shows two example comparison plot for extended simulations over 10 seconds with NGS magnitude of 8 and 16. The ratio map, shown on the right, is the ratio between the classical method and ours. While the two PSDs may appear similar, the ratio image shows at which 2D spatial frequencies the two methods differ. Of note, our network improves smearing of the central PSD core even at low magnitudes, indicating a reduction in servo-lag and therefore an improvement in contrast. At higher magnitudes our method improves across all frequencies indicating an improvement in noise suppression and aliasing in addition to servo-lag.

5 CONCLUSIONS

5.1 Hardware Implementation

While we have proven our method in simulation, the next step is to show bench results to verify our claims. As is often the case when moving from simulation to hardware many issues may arise due to differences in the data. However, due to the simplicity and low spatial dimensionality of the data required by our method, we do not forsee substantial issues beyond potentially acquiring additional training data from our hardware.

However, looking beyond simple hardware verification towards real-time application more work will be required to incorporate these types of methods into AO pipelines. Due to the high speeds at which they are operated, careful consideration will be required to achieve real-time speeds. Currently, our method runs on the order of 200Hz without any GPU code optimization. Luckily, there is a growing body of work dedicated to optimizing trained networks He et al. (2017), more efficient compilation of hardware instructions[1], and custom hardware for very high-speed inference of deep learning models Lacey et al. (2016), which are quickly accelerating the real-time operating speeds for these types of networks. Other changes to the network, such as adjusting the number of layers, filters, or dense blocks can also be made to reach the run-time budget for a given project.

[1] e.g., NVidia's TensorRT https://developer.nvidia.com/tensorrt

ACKNOWLEDGEMENTS

The Acknowledgements section is not numbered. Here you can thank helpful colleagues, acknowledge funding agencies, telescopes and facilities used etc. Try to keep it short.

REFERENCES

Conan R., Correia C., 2014, in Adaptive optics systems IV. p. 91486C

Dieleman S., Willett K. W., Dambre J., 2015, Monthly notices of the royal astronomical society, 450, 1441

Finn C., Goodfellow I., Levine S., 2016, in Advances in neural information processing systems. pp 64–72

Gendron E., et al., 2011, Astronomy & Astrophysics, 529, L2

Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., 2014, in Advances in neural information processing systems. pp 2672–2680

He K., Zhang X., Ren S., Sun J., 2015, in Proceedings of the IEEE international conference on computer vision. pp 1026–1034

He Y., Zhang X., Sun J., 2017, in Proceedings of the IEEE International Conference on Computer Vision. pp 1389–1397

Hochreiter S., Schmidhuber J., 1997, Neural computation, 9, 1735

Kingma D. P., Ba J., 2014, arXiv preprint arXiv:1412.6980

Lacey G., Taylor G. W., Areibi S., 2016, arXiv preprint arXiv:1602.04283

LeCun Y., Bengio Y., Hinton G., 2015, nature, 521, 436

Liang X., Lee L., Dai W., Xing E. P., 2017, in Proceedings of the IEEE International Conference on Computer Vision. pp 1744–1752

Lotter W., Kreiman G., Cox D., 2016, arXiv preprint arXiv:1605.08104

Olah C., 2015

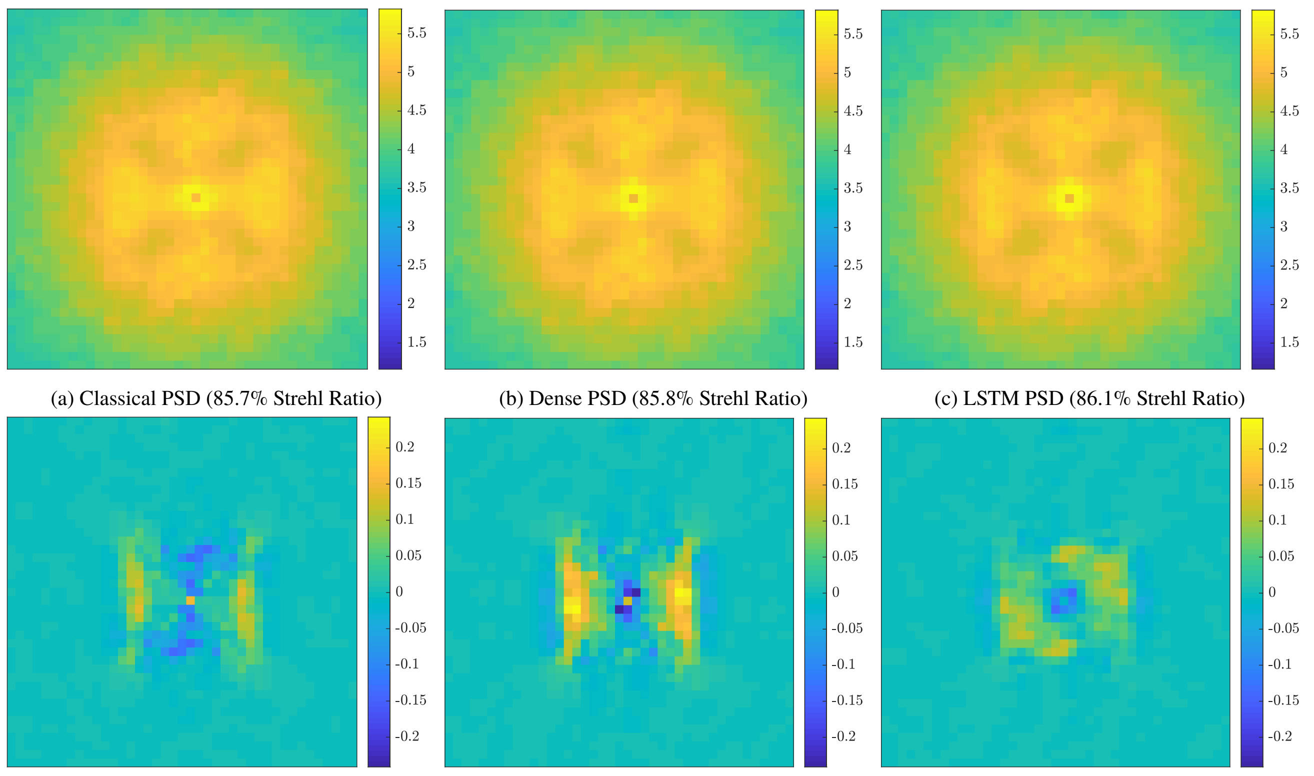
Osborn J., Juez F. J. D. C., Guzman D., Butterley T., Myers R., Guesalaga A., Laine J., 2012, Optics express, 20, 2420

Ronneberger O., Fischer P., Brox T., 2015, in International Conference on Medical image computing and computer-assisted intervention. pp 234–241

Shrivastava A., Pfister T., Tuzel O., Susskind J., Wang W., Webb R., 2017, in Proceedings of the IEEE conference on computer vision and pattern recognition. pp 2107–2116

Xingjian S., Chen Z., Wang H., Yeung D.-Y., Wong W.-K., Woo W.-c., 2015, in Advances in neural information processing systems. pp 802–810

Zhu J.-Y., Park T., Isola P., Efros A. A., 2017, in Proceedings of the IEEE international conference on computer vision. pp 2223–2232


(a) Classical PSD (85.7% Strehl Ratio) (b) Dense PSD (85.8% Strehl Ratio) (c) LSTM PSD (86.1% Strehl Ratio)

(d) Classical/Dense Ratio Image (e) Classical/LSTM Ratio Image (f) Dense/LSTM Ratio Image

Figure 9. Magnitude 8 Residual Wavefront Frequency Analysis: Log-scale power spectral density (PSD) plots (left, centre) and PSD ratio image (right) of the residual wavefronts comparing our predictive method and the classical integrator over the DM control radius. Our method clearly shows improvements near the PSD core, which are attributed to servo-lag, and near the edge of the control radius due to aliasing. At higher magnitudes we see suppression across nearly all frequencies due to the network removing noise from the measurements which dominate in the faint regime.

APPENDIX A: TRAINING PSEUDO-CODE

Algorithm 1: Network Training Procedure

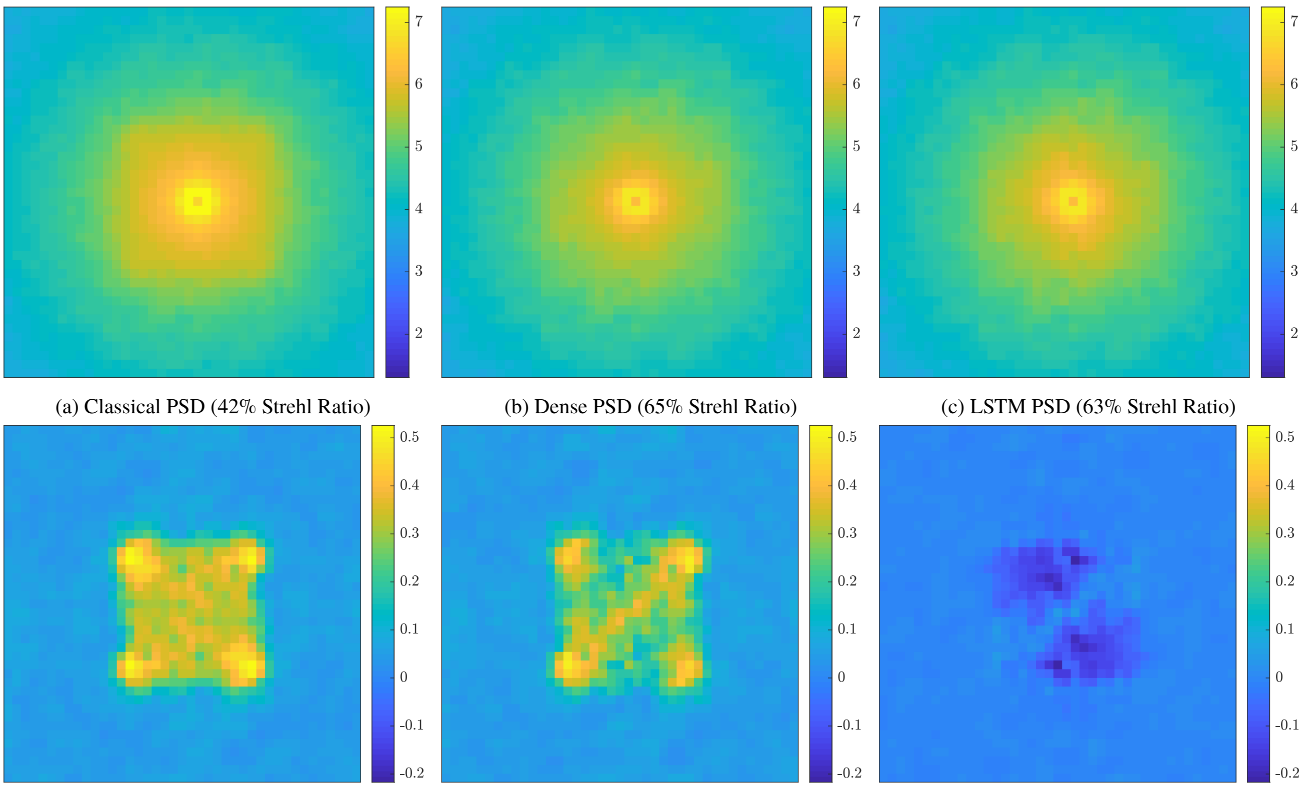
```
while Training do
    /* Sample next batch of training data */
    nextInput = nextBatch(trainingData)
    nextGT = nextBatch(groundtruthData)

    /* Get prediction with current network */
    prediction = predictionNet(nextInput)

    /* Get discriminator labels for both our
       network output and the current
       training data */
    pLabel = discriminatorNet(prediction)
    dLabel = discriminatorNet(nextInput)

    /* Calculate mean squared error between
       prediction and ground truth slopes as
       well as its cross-entropy loss against
       the "real" label */
    pLoss = MSE(prediction, nextGT) + CEL(pLabel, $\vec{1}$)
    /* Calculate cross-entropy loss between
       the network input and output and their
       respective labels */
    dLoss = CEL(pLabel, $\vec{0}$) + CEL(dLabel, $\vec{1}$)

    /* Use the calculated loss functions to
       update both networks */
    discriminatorNet.backPropagate(dLoss)
    predictionNet.backPropagate(pLoss)
end
```



(a) Classical PSD (42% Strehl Ratio) (b) Dense PSD (65% Strehl Ratio) (c) LSTM PSD (63% Strehl Ratio)

(d) Classical/Dense Ratio Image (e) Classical/LSTM Ratio Image (f) Dense/LSTM Ratio Image

Figure 10. Magnitude 16 Residual Wavefront Frequency Analysis: Log-scale power spectral density (PSD) plots (left, centre) and PSD ratio image (right) of the residual wavefronts comparing our predictive method and the classical integrator over the DM control radius. Our method clearly shows improvements near the PSD core, which are attributed to servo-lag, and near the edge of the control radius due to aliasing. At higher magnitudes we see suppression across nearly all frequencies due to the network removing noise from the measurements which dominate in the faint regime.

This paper has been typeset from a TEX/LATEX file prepared by the author.