Latent GAN: using a latent space-based GAN for rapid forecasting of CFD models

Jamal Afzali, César Quilodrán Casas, and Rossella Arcucci

Data Science Institute,
Imperial College London, London, UK

Abstract. The focus of this study is to simulate realistic fluid flow, through Machine Learning techniques that could be utilised in real-time forecasting of urban air pollution. We propose a novel *Latent GAN* architecture which looks at combining an AutoEncoder with a Generative Adversarial Network to predict fluid flow at the proceeding timestep of a given input, whilst keeping computational costs low. This architecture is applied to tracer flows and velocity fields around an urban city. We present a pair of AutoEncoders capable of dimensionality reduction of 3 orders of magnitude. Further, we present a pair of Generator models capable of performing real-time forecasting of tracer flows and velocity fields. We demonstrate that the models, as well as the latent spaces generated, learn and retain meaningful physical features of the domain. Despite the domain of this project being that of computational fluid dynamics, the *Latent GAN* architecture is designed to be generalisable such that it can be applied to other dynamical systems.

Keywords: Generative adversarial networks · Reduced order models · Urban air pollution.

1 Introduction

Computational Fluid Dynamics (CFD) concerns itself with using applied mathematics and/or computational software to resolve fluid flows in a domain. A crucial component of CFD are the Navier-Stokes (NS) equations; these describe the motions of incompressible fluid flow. A major drawback of CFD is the extreme difficulty in solving the NS equations due to their non-linearity. Another is the high dimensionality involved. Although the introduction of computers has allowed researchers to automate the calculations involved, with current hardware limitations, the computational complexity involved is far too great to solve in a reasonable amount of time. The revolution of Machine Learning (ML) in recent years has been invaluable for the field of CFD [6]. Previous data-driven studies tend to represent the fluids using linear basis functions such as principal component analysis (PCA). However, CFD is a non-linear problem and so more complex methods are required.

The aim of this study is the development of a neural network that could be utilised in real-time forecasting of urban air pollution in London. Accurate

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

and fast forecasts of air pollution simulations have tremendous potential for healthcare, especially to explore the impact of exposure of individuals to air pollution. Generalising to the wider field of CFD, historically velocity fields and turbulent flows have been extremely difficult to resolve. The non-linearity of the problem leads to complex solutions that are far too difficult to solve analytically, with numerical solutions being too slow to do within a reasonable amount of time. A wider objective of this project is to train a surrogate generative model capable of predicting velocity fields in real-time.

In this study, we propose a novel *Latent GAN* architecture that looks at combining a Convolutional AutoEncoder (CAE) with a Generative Adversarial Network (GAN) [7] to produce surrogate generative models capable of predicting fluid flows such as tracers and velocity fields, whilst keeping computational costs low. The CAE focuses on reducing the dimensionality of given data samples before passing it to the GAN, which attempts to predict outputs that are a single timestep ahead. Despite the domain for this study being that of CFD, the network architecture is designed to be general such that it could be applied to many other dynamical domains that aren't necessarily related to CFD.

One study [8] of interest looks at combining an autoencoder (AE) with a generative model to predict fluid flows, which shows promising results. Another model of interest is the variational autoencoder (VAE)/GAN [9] which looks at generations of random faces using a combination of VAEs with GANs. Further, the Structural and Denoising Generative Adversarial Network (SD-GAN) [5] makes use of a piece-wise loss function to guide its GAN generations. This is useful as we also want to restrict the generations of our GANs; namely, to predict at the proceeding timestep. Other studies that have successfully combined fluid predictions with machine learning include: an application to a realistic case in China [17], application to unsteady flows [15], prediction of flow fields with deep convolutional recurrent autoencoders [16], and for data assimilation [3] with ROMs indoors [2] and outdoors [13,10]. Furthermore, [12] used adversarial training to improve the divergence of the data-driven forecast prediction over time and achieve better compression from full-space to latent space of urban air pollution CFD simulations.

In summary, in this paper we

- propose a novel *Latent GAN* architecture that combines a CAE along with a GAN to produce a model that is capable of predicting fluid flow. The Generator is restricted to generate data at the proceeding timestep, given an input data sample. The architecture has been designed to be generalisable such that it can be applied to any dynamical system.
- demonstrate the ability to encode fluid data on unstructured meshes down to a latent space with several orders of reduction through the use of a CAE, an architecture typically used on structured domains that do not have real physical meanings, such as images.
- present a pair of Generator models, trained via the *Latent GAN* architecture, that are capable of predicting tracer flows and velocity fields at the proceeding timestep of a given input.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

2 Latent GAN

The *Latent GAN* architecture looks at combing a Convolutional AutoEncoder with a Generative Adversarial Network to predict the proceeding timestep, given an input data sample. A high level overview of the model architecture proposed is displayed in Figure 1. The AutoEncoder comprised of an Encoder and a Generator, as shown. The Encoder model reduces dimensionality of the data down to a reduced latent space. This is done to reduce the computational complexity when training the GAN and to attempt to reduce the instability that is inherit when training GANs.

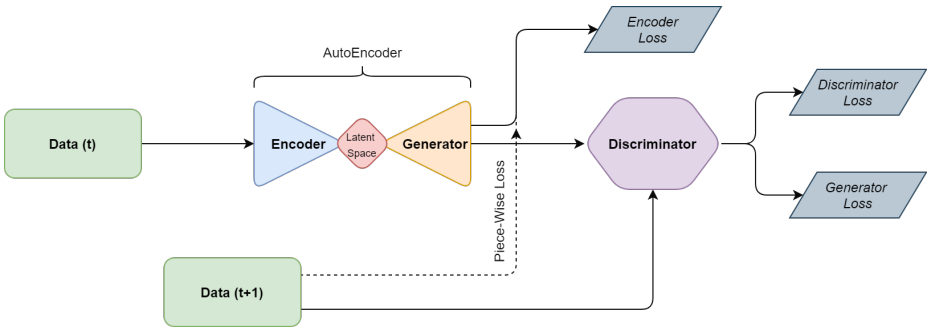

Fig. 1. High level overview of *Latent GAN* architecture.

Initially, we attempted to train all three models in a single training process. Whilst in theory this is expected to work, in practice the training process showed extreme instability across all three models. This is understandable as we were essentially trying to find three solutions simultaneously with targets that were constantly moving as each of the three networks were updated. A further, less obvious, issue with this technique is that an unintuitive latent space is being generated. As the Encoder is encoding w.r.t timestep t and the Generator is decoding w.r.t timestep $t+1$, this meant the latent space became a strange cross between both timesteps. Whilst this may not have been an issue for scope of this particular project, it meant the AE could not be used alone as it provided no useful latent space. Instead, we opted to train the AE and GANs separately. The AE training process is performed, as standard, with a Mean Squared Error (MSE) loss being calculated. Once a stable AE is trained, the Decoder is dropped, and the Encoder is implemented as a fixed standalone pre-trained model into the *Latent GAN* architecture displayed previously in Figure 1.

The input of the *Latent GAN* network is a data sample at time t. This is reduced into a latent space representation via the pre-trained Encoder. The output is then passed through the Generator which attempts to reconstruct the data back to its original size, but also incrementing the time by one. The generated outputs, along with real data at time $t+1$ are then fed into the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

Discriminator for classification. As per standard GAN training [7], Binary Cross Entropy (BCE) losses were calculated for the Discriminator and Generator, based on whether the generated outputs had successfully "tricked" the Discriminator. In addition, a separate MSE is calculated for the Generator between its generations and the ground truths at $t+1$. This is done to guide our Generator to produce a specific output, and not one that only appeared to be plausibly from the dataset as per regular GANs.

To summarise, the losses for each network were as follows:

$$L_{AE}(\boldsymbol{x}_t) = \min\left[\frac{1}{N}\sum_{i=1}^{N}\left[AE(\boldsymbol{x}_t) - \boldsymbol{x}_t\right]^2\right] \tag{1}$$

$$L_D(\boldsymbol{x}_t) = \max\Big[\log D(\boldsymbol{x}_{t+1}) + \log\left(1 - D(G(E(\boldsymbol{x}_t)))\right)\Big] \tag{2}$$

$$L_G(\boldsymbol{x}_t) = \max\Big[\log D(G(E(\boldsymbol{x}_t)))\Big] + \min\left[\frac{1}{N}\sum_{i=1}^{N}\left[G(E(\boldsymbol{x}_t)) - \boldsymbol{x}_{t+1}\right]^2\right] \tag{3}$$

where AE represents a forward pass through both Encoder and Decoder architectures, E represents the Encoder alone, G represents the Generator and D represents the Discriminator. A full Algorithm of both training processes can be seen in section 3.1.

3 Models

The full codebase constructed as part of this study can be found at the following repository:
`https://github.com/DL-WG/LatentGAN`

3.1 Algorithms

For the AutoEncoder, training is as described in Algorithm 1. The inputs to this were unstructured tracer / velocity field data arrays at time t.

For the GAN, training is as described in Algorithm 2. The inputs to this were unstructured tracer / velocity fields at time t, and $t+1$ for the Generator loss.

It is worth noting that in Algorithm 2 Line 23, the order of magnitude of the two components of the Generator's error are starkly different. Typical values for the BCE loss are of 1 order of magnitude, whereas for the MSE error, we'd hope for this to be as small as possible. From our experimentations, we noticed that some MSE errors reaches as small as 10^{-7}. Therefore, an α coefficient is included with the MSE to try to balance the scales of the errors to be of the same order. If we were to perform the standard addition of both components ($\alpha = 1$), the MSE error would become irrelevant and ignored when calculating the gradients for gradient descent. For example, say $errBCE = 2.3$ and $errMSE = 0.0000006$, then $errG = 2.3 + 0.0000006 = 2.3000006 \approx 2.3$. This then means the Generator is just learning to become a standard GAN Generator, and ignores information about $t+1$.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

Algorithm 1: AutoEncoder Training

```
Instantiate Encoder (E) and Decoder (D) architectures;
Initialise E and D weights from normal distribution;
Instantiate DataLoader;
for epoch ← 0 to max_epoch do
    /* iterate over all batches in DataLoader                    */
    for data in DataLoader do
        Zero gradients in Encoder and Decoder;
        output ← E(data); // Forward pass through Encoder
        output ← D(output); // Forward pass through Decoder
        loss ← MSE(data, output); // Calculate loss
        Calculate gradients;
        Update weights using Optimiser step;
    end
end
Save models;
```

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

Algorithm 2: GAN Training

Instantiate Encoder (**E**), Generator (**G**) and Discriminator (**D**) architectures;
Load pretrained **E** weights;
Initialise **G** and **D** weights from normal distribution;
Instantiate DataLoader;

real ← 1 ;
fake ← 0 ;

for *epoch* ← 0 **to** *max_epoch* **do**
    `/* iterate over all batches in DataLoader */`
    for *data* **in** *DataLoader* **do**
        data_incr ← *(data* samples incremented by 1*)*;
        Zero gradients in **D** and **G***;*
        `/* Discriminator Training */`
        `/* Pass all-real batch */`
        outputD_real ← **D***(data) ;*
        errD_real ← BCE*(outputD_real, real) ;*
        `/* Pass all-fake batch */`
        outputE ← **E***(data) ;* `// Do not track gradients here`
        outputG ← **G***(outputE) ;*
        outputD_fake ← **D***(outputG) ;*
        errD_fake ← BCE*(outputD_fake, fake) ;*
        errD ← *errD_real* + *errD_fake ;*
        Calculate gradients*;*
        Update **D** weights using Optimiser step*;*
        `/* Generator Training */`
        output ← **D***(outputG) ;* `// Pass outputG through updated D`
        errBCE ← BCE*(output, real) ;* `// Use real labels as suggested by Goodfellow`
        errMSE ← MSE*(data_incr, outputG) ;*
        errG ← *errBCE* + α *errMSE ;* `// See comment below`
        Calculate gradients*;*
        Update **G** weights using Optimiser step*;*
    end
end
Save models*;*

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

3.2 Model architectures

Note that all LeakyReLU layers had a *negative_slope* value of 0.2 and *in_place* set to True. Tables 1, 2, and 3 show the final hyperparameters used. The visualisation of the networks are shown in Figures 2 and 3.

Table 1. Model architecture for Encoder.

Layer Name	Kernel Size	Stride	Padding
Conv1D	4	2	1
LeakyReLU			
BatchNorm1D			
Conv1D	4	2	1
LeakyReLU			
BatchNorm1D			
Conv1D	4	2	1
LeakyReLU			
BatchNorm1D			
Conv1D	4	2	1
LeakyReLU			
BatchNorm1D			
Conv1D	4	2	1

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

Table 2. Model architecture for Decoder and Generator.

Layer Name	Kernel Size	Stride	Padding	Output Padding
ConvTranspose1D	4	2	1	0
BatchNorm1D				
LeakyReLU				
ConvTranspose1D	4	2	1	1
BatchNorm1D				
LeakyReLU				
ConvTranspose1D	4	2	1	0
BatchNorm1D				
LeakyReLU				
ConvTranspose1D	4	2	1	0
BatchNorm1D				
LeakyReLU				
ConvTranspose1D	4	2	1	0
Tanh				

Table 3. Model architecture for Discriminator.

Layer Name	Kernel Size	Stride	Padding
Conv1D	4	8	1
LeakyReLU			
Conv1D	4	8	1
BatchNorm1D			
LeakyReLU			
Conv1D	4	8	1
BatchNorm1D			
LeakyReLU			
Conv1D	4	8	1
BatchNorm1D			
LeakyReLU			
Conv1D	4	4	1
BatchNorm1D			
LeakyReLU			
Conv1D	4	4	1
BatchNorm1D			
LeakyReLU			
Conv1D	1	2	1
Sigmoid			

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

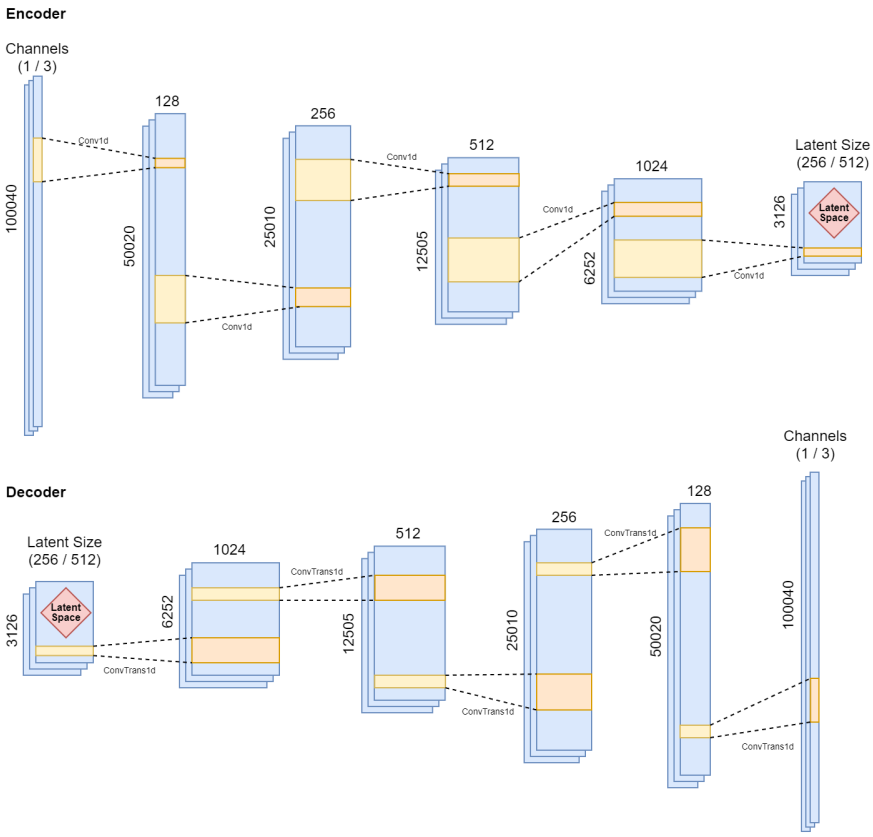

Fig. 2. Encoder and Decoder model visualisation.

4 Testing and Evaluation on a real test case

The dataset used to train the network is that of an urban city comprising of tracers with dimensionality $100,040$ and velocity fields with dimensionality $3 \times 100,040$. Note that all screenshots displayed showcase a 2D slice of the 3D domain, using ParaView [1].

Using DCGAN [14] as a basis for the GAN with modifications made to fit the specific dataset, the Decoder is a duplicate of the Generator and the Encoder is designed to be an inverse of this. All networks were run for 1000 epochs, taking around 24 hours for the AutoEncoders and around 27 hours for the GANs. We saved the networks every 200 epochs, and the best of these were chosen.

The final tracer AE, trained for 600 epochs, is capable of reducing the dimensionality down to 256, whilst achieving an average MSE loss of 7.68×10^{-7}.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

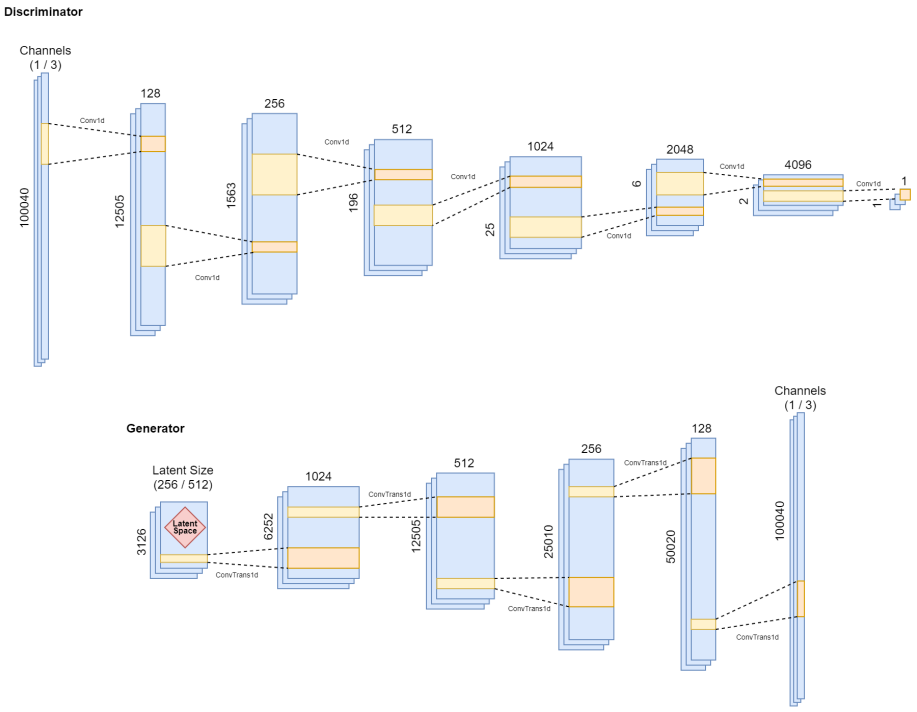

Fig. 3. Discriminator and Generator model visualisation.

Showcased in Figure 4 is an AE reconstruction of timestep $t = 3000$ from our test set along with its ground truth. We see similar reconstructions, with flow shapes and intensities maintained. Its worth noting that the final timestep used to train the tracer models is $t = 988$, demonstrating that the AE network has learnt meaningful physical features of the dataset and is generalisable to unseen data. The final tracer *Latent GAN* is trained for 400 epochs, achieving an average MSE loss of 7.14×10^{-6} as shown in Figure 8. An example of interpolation results can be viewed in Figure 5. The network is fed in timestep $t = 3000$ and a prediction is made for timestep $t = 3001$. We see that the shapes of the tracer are correctly predicted along with the intensities. Despite this however, we notice that some artifacts are created around the domain with a somewhat patchy prediction of the actual flow. This is perhaps a side effect caused by the unstructured meshes used, leading to the convolutional layers incorrectly learning features in the data arrays that are not present in the actual flow.

The final velocity field AE, trained for 1000 epochs, reduces the dimensionality down to 512, whilst achieving an average MSE loss of 1.49×10^{-2}. We immediately notice the degraded performance compared to that of the tracers. This is, however, to be expected as the velocity fields are far more complex, with 3 channels instead of 1 for the tracers, as well as having flows occurring across the entire domain

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

instead of just the centre. Figure 6 displays AE reconstructions for $t = 980$, where the final timestep to train the model is $t = 899$. We find that the reconstructions appear to be very similar, maintaining shapes and intensities. The final velocity field *Latent GAN* is also trained for 1000 epochs, achieving an average MSE loss of 2.31×10^{-2} as shown in Figure 9. Note that the initial 25 timesteps have been removed for better visualisation. Interpolated results of these can be viewed in Figure 7, where the network is fed in $t = 980$ and a prediction of $t = 981$ is generated. Similarly to the tracers, we find that the prediction correctly maintained intensities as well as flow shapes. We do note however, that the predictions here also suffer from patchy generations.

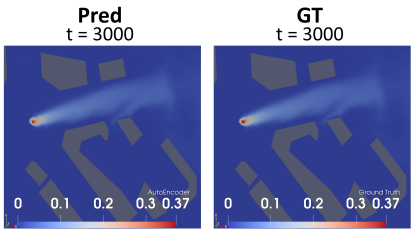

Fig. 4. Tracer AutoEncoder reconstruction.

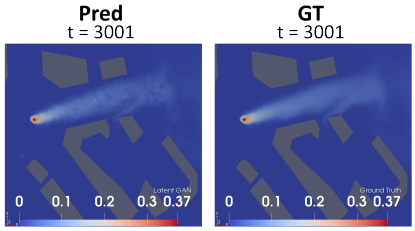

Fig. 5. Tracer *Latent GAN* prediction given input $t = 3000$

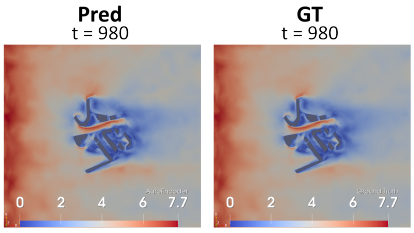

Fig. 6. Velocity field AutoEncoder reconstruction

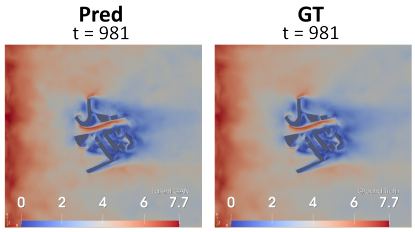

Fig. 7. Velocity field *Latent GAN* prediction given input $t = 980$

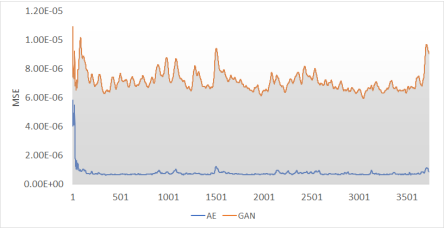

Fig. 8. Tracer MSE losses

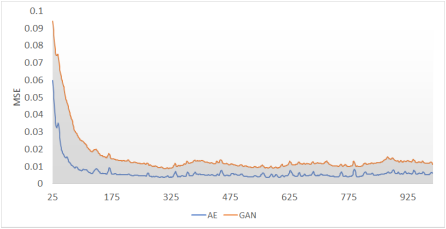

Fig. 9. Velocity field MSE losses

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

Generating real timesteps, where the actual physical equations were solved, took around 172 seconds using an *E5-2650 v3 CPU* with 250GB RAM. Predictions using the *Latent GAN* models took only 1.1 seconds and 1.3 seconds for tracers and velocities respectively using a single core of an *i7-4790k* with 16GB RAM. Unfortunately, due to limitations we were unable to reproduce these on the same hardware, although its worth noting that the i7 has around a 15% better single core performance. Running on a NVIDIA CUDA [11] enabled GPU, a GTX 970 with 4GB of vRAM, took 0.25 and 0.3 seconds for tracers/velocities respectively.

5 Summary and future work

This paper introduced *Latent GAN*. We showed that the latent spaces generated, learnt and retained meaningful physical features of the domain. Despite the domain of this project being that of CFD, *Latent GAN* is generalisable such that it can be applied to other dynamical systems.
As future work, exploiting other kinds of CAE using 3D data arrays would allow us to incorporate known real physics into the network, in the form of physics informed loss functions. A different possible route to take would be to look at the Wasserstein GAN [4]. This is a modification to the existing GAN that are generally more robust and stable, and so may allow us to train the entire *Latent GAN* in a single training process. Further directions could be to possibly extend the *Latent GAN* architecture by combining a Long-Short Term Memory (LSTM) network. Currently, only a single timestep is fed as input to the network; it would be interesting to see results where multiple timesteps are fed in instead.

Acknowledgements

This work is supported by the EPSRC Grand Challenge grant Managing Air for Green Inner Cities (MAGIC) EP/N010221/1, the EP/T003189/1 Health assessment across biological length scales for personal pollution exposure and its mitigation (INHALE), the EP/T000414/1 PREdictive Modelling with QuantIfication of UncERtainty for MultiphasE Systems (PREMIERE) and the Leonardo Centre for Sustainable Business at Imperial College London.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29

References

1. Paraview (2018)
2. Amendola, M., Arcucci, R., Mottet, L., Casas, C.Q., Fan, S., Pain, C., Linden, P., Guo, Y.K.: Data assimilation in the latent space of a neural network. arXiv preprint arXiv:2012.12056 (2020)
3. Arcucci, R., Zhu, J., Hu, S., Guo, Y.K.: Deep data assimilation: Integrating deep learning with data assimilation. Applied Sciences **11**(3), 1114 (2021)
4. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017)
5. Banerjee, S., Das, S.: Sd-gan: Structural and denoising gan reveals facial parts under occlusion. arXiv preprint arXiv:2002.08448 (2020)
6. Brunton, S.L., Noack, B.R., Koumoutsakos, P.: Annual Review of Fluid Mechanics: Machine Learning for Fluid Mechanics. Annual Reviews **52**, 477–508 (2020). https://doi.org/https://doi.org/10.1146/annurev-fluid-010719-060214
7. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
8. Kim, B., Azevedo, V.C., Thuerey, N., Kim, T., Gross, M., Solenthaler, B.: Deep fluids: A generative network for parameterized fluid simulations. In: Computer Graphics Forum. vol. 38, pp. 59–70. Wiley Online Library (2019)
9. Larsen, A.B.L., Sønderby, S.K., Larochelle, H., Winther, O.: Autoencoding beyond pixels using a learned similarity metric. arXiv preprint arXiv:1512.09300 (2015)
10. Mack, J., Arcucci, R., Molina-Solana, M., Guo, Y.K.: Attention-based convolutional autoencoders for 3d-variational data assimilation. Computer Methods in Applied Mechanics and Engineering **372**, 113291 (2020)
11. NVIDIA: Cuda, `https://developer.nvidia.com/cuda-zone`
12. Quilodrán-Casas, C., Arcucci, R., Pain, C., Guo, Y.: Adversarially trained LSTMs on reduced order models of urban air pollution simulations. arXiv preprint arXiv:2101.01568 (2021)
13. Quilodrán-Casas, C., Arcucci, R., Wu, P., Pain, C., Guo, Y.K.: A Reduced Order Deep Data Assimilation model. Physica D: Nonlinear Phenomena **412**, 132615 (2020)
14. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
15. Reddy, S.B., Magee, A.R., Jaiman, R.K., Liu, J., Xu, W., Choudhary, A., Hussain, A.: Reduced order model for unsteady fluid flows via recurrent neural networks. In: International Conference on Offshore Mechanics and Arctic Engineering. vol. 58776, p. V002T08A007. American Society of Mechanical Engineers (2019)
16. Reddy Bukka, S., Magee, A.R., Jaiman, R.K.: Deep Convolutional Recurrent Autoencoders for Flow Field Prediction. arXiv pp. arXiv–2003 (2020)
17. Xiao, D., Fang, F., Zheng, J., Pain, C., Navon, I.: Machine learning-based rapid response tools for regional air pollution modelling. Atmospheric environment **199**, 463–473 (2019)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_29