TWO-DIMENSIONAL STEADY STATE FLUID FLOW PREDICTION USING CONVOLUTIONAL NEURAL NETWORKS

Steve Guerin, Dimitrios Baltimas, and Kareemullah Shaik

ABSTRACT

There is a wide spectrum of applications in the engineering domain that require solutions which need to be approximated through fitting partial differential equations in different spatio-temporal conditions. In the fluid dynamics world, applications like fluid movement around various sized objects has been thoroughly modeled using computational fluid dynamics (CFD) solvers. The optimization process that involves CFDs is time-expensive as the design complexity increases. Data driven approaches involving neural networks have been developed to alleviate the trade-off between the model complexity and the accuracy of the simulations. We explore three Convolutional Neural Network architectures for performance in steady-state flow prediction around arbitrary obstacles. We find that models with significant dimensionality reduction efficiently learn global context for flow and accurately predict fluid velocities.

1. INTRODUCTION

Modeling a natural phenomenon by exploiting its underlying dynamics generates a set of partial differential equations whose solutions need to be adequately approximated under a specific set of spatial and temporal conditions. These either define the energy offered to the system and/or the boundary specifics (edges, surface conditions etc.).

In fluid flow applications the partial differential equation at hand is the Navier-Stokes equation that models viscous fluid flow. This can be seen as a balance equation that arises from applying Newton’s second law of motion to fluid flow, together with the assumption that the stress in the fluid is the sum of a diffusing viscous term (proportional to the gradient of velocity) and a pressure term.

\[ \rho \left( \frac{\partial u}{\partial t} + u \nabla u \right) = \nabla p + \nabla \tau + \mathbf{g} \rho \]

- \( \frac{\partial u}{\partial t} \) is the material derivative
- \( \rho \) is the density
- \( u \) is the flow velocity
- \( p \) is the fluid pressure
- \( \tau \) is the deviatoric stress tensor of the material
- \( \mathbf{g} \) represents accelerations acting on the continuum

CFD solvers are designed to simulate the physical interaction of liquids and gases with surfaces defined by prescribed boundary conditions by approximating the Navier-Stokes solution discussed above. CFDs also utilize parameters that prescribe the fluid at hand with behavioral properties, such as the Reynolds’s number. This number is a dimensionless quantity in fluid mechanics used to help predict flow patterns in different fluid flow situations. At low Reynolds numbers, flows tend to be dominated by laminar (sheet-like) flow, while at high Reynolds numbers turbulence prevails. This parameter is provided to the CFD by the engineer, as the engineer forecasts a probable flow pattern and asks from the CFD to simulate for solutions at this part of the parameter space.

Optimization processes that involve the use of CFDs for all the iterations needed to meet the specifications are lengthy and require field specific expertise to generate reliable and applicable results (i.e. provide Reynold’s number that is close to expected flow pattern). In our approach, a data driven model will be utilised as a surrogate for the optimization process. This model can de-convolve the time efficiency - accuracy trade-off of the parameter-grid search during the first iterations of any new design. At the latter stages the CFD can be then utilised to fine-tune the parameters.

2. RELATED WORK

Over the past years numerous efforts have been presented in the fluid-dynamics bibliography as potential surrogates to the typical CFD-oriented simulations to generate the fluid flow over arbitrary spatial conditions i.e. obstacles, anomalies etc.

Convolutional neural networks have been in the epicenter of this effort since they have been extensively used for steady state flow prediction [1] and can achieve orders of magnitude faster runtime at the initial steps of a design. For data with strong spatial and/or temporal dependencies CNNs have been known to learn invariant high-level features that are informative for supervised tasks [2] while performing impressively in learning arbitrary geometry representations [3,4]. CNNs are the core models in all the presented schemes used to properly learn the underlying dynamics of the fluid flow pattern. CNNs are used as blocks in Encoder-Decoders (Fig.1) were used to end-to-end map the input to the target fluid flow snapshot.

In another approach [5], custom loss functions are educatedly engineered in order to push the Encoder-Decoder to at-
3. DATA

3.1. Computational Fluid Dynamics Simulation

There are no widely available datasets for fluid flow around arbitrary objects, derived either through experimentation or CFD simulation. For this reason, we developed our own dataset using an open source Lattice Boltzmann Solver called PyLBM [11]. PyLBM is a flexible package that can be used for a wide variety of transport problems in 1D, 2D, and 3D. The internal representation of geometry in PyLBM relies on construction from simpler primitives such as circles and triangles. To generate our dataset, we simulated flow around randomly generated and placed triangles. The vertices of these triangles were saved for processing prior to training as described in 3.2. To obtain laminar flow that converged reliably to a steady state, we reduced fluid velocity until the Reynolds number was low ($Re \sim 2$). In the course of our simulations, we generate significant amounts of velocity data prior to reaching steady state. While we did not attempt to predict these transient flows, that problem is an important and promising application of machine learning approaches. The resolution of the simulation was $128 \times 64$, with boundary conditions as follows: constant fluid velocity at the left (inlet) boundary, constant pressure at right (exit) boundary, and bounce-back condition at top and bottom boundaries.

For each triangle, the simulation was processed until the magnitude of local acceleration at every point in the domain was sufficiently small ($< 10^{-3} \cdot v_{inlet}/timestep$), when the simulation was determined to have reached a steady state. The X and Y velocity fields were then stored as a stack, and served as the $2 \times 128 \times 64$ label for the model. An example of the X and Y velocity fields is given in Figure 2.

1000 scenes were generated as described above, with 800 of those randomly chosen as a training set, and 200 as a validation set.

3.2. Data Preprocessing

The input features for our model were geometric representations of our obstacles. While these obstacles are represented in a sparse way using vertices within the simulation, we desired a more flexible representation that would allow us to test our model on arbitrary geometry. To do this, we initially generate a binary presence map sampled on the Cartesian grid of our domain indicating the presence or absence of an obstacle. The information within the perceptive field of a convolution kernel far from the triangle boundaries, is identical - all 1s or 0s. The binary representation therefore struggles in carrying geometric information far from the obstacle without requiring very deep networks.

For these reasons, we further process the binary presence map into a Signed Distance Function (SDF). To define this function, first a level-set is chosen, consisting of points on the boundary of solid objects (in this case, our obstacle and the top and bottom walls). The value of the function at any point in the Cartesian grid is then the shortest distance from the point to the level set, signed such that inside the boundary the values are negative and outside, positive. An illustration of the binary presence map for a random geometry and the SDF for that geometry is given in Figure 3.

4. METHODS

This project explored various permutations of Convolutional Neural Network Architectures. A neural network, broadly defined, is a directed computational graph that is comprised of nodes organized in a series of layers. The nodes in the network are connected such that all nodes in a given hidden layer are a linear combination of all nodes in the previous layer. The outputs of all nodes are passed through a non-linearity called the activation function, which provides the neural net with more expressivity. Theoretically, a single layer network with enough nodes can approximate any real-valued continuous function in $R^n$ [10]. Functionally, wide networks become impractical due to the fact that the size of the network increases exponentially with layer width ($O(n^2)$) and therefore standard neural nets no longer become practical for data with high dimensionality.

Because of this phenomenon, convolutional neural networks have become the de-facto network architecture for high-dimensional and spatially related data. The network is named after it’s core operation, the convolution. This
mathematical operation effectively acts as a form of template matching with the input data, where the convolution kernel slides across the input map creating an activation map that is passed to the following layer:

\[
\sum_{n=-l}^{+l} \sum_{m=-k}^{+k} h(n, m) \cdot x(i - l, j - m)
\]

What makes the convolutional neural network effective is the shared weights across a layer. For a input shape of 100 x 100 and hidden layer of the same size, the network need only store \(O(n \text{ filters} \cdot K^2)\) weights (K is the kernel size) compared to \(O(100^4)\). This allows the network to contain more layers and process higher dimensional data, increasing the network utility.

To accomplish the task of predicting steady state flow fields from SDF representations of the simulation domain, we examined multiple fully convolutional architectures. These models included a feed forward encoder-decoder architecture, a modified ResNet-18 architecture, and a U-Net architecture. The encoder-decoder network and UNet were chosen for their demonstrated capabilities for image in - image out tasks while the ResNet-18 was chosen as benchmark due to its widespread using in visual computing tasks. Each model was trained and tuned independently and their performance was compared. The mean-squared error metric was used as a loss function during training and as the performance metric during validation. Results were also visually examined to look for artifacts.

4.1. Encoder-Decoder Model

The feed forward encoder decoder architecture consisted of a series of convolutional layers followed by a downsampling layer that reduced the activation map by a factor of 4. Each convolutional layer had \(N\) filters and 3x3 kernels, with \(N\) being a tuned hyperparameter. Downsampling was accomplished using strided 4x4 convolutions. The pattern of convolution followed by downsampling was repeated until reaching the bottleneck layer. After reaching the bottleneck layer the data is passed through an upsampling layer where the feature map and number of filters are increased until reaching the output layer. Upsampling was accomplished using strided 4x4 transpose convolutions. Four encoder-decoder architectures were tested: ‘Tiny’ with 6 layers with 1 dimension-reducing step, ‘Straight’, with 13 layers with no reducing step, ‘Short’, which had 9 layers with 2 reducing steps, and ‘Long’, which had 13 layers with 4 reducing steps. The ‘Long’ model, which would become our baseline for the Encoder-Decoder model, is diagrammed in Figure 4.

4.2. ResNet-18 Model

The ResNet Architecture capitalizes on an approach used frequently in the forecasting space: learning a residual function. Often times it is easier to approximate the difference between the function \(F(x)\) and the input \(x\) rather than the function itself. The residual function \(H(x)\) is functionally implemented through blocks of the architecture. A block consists of a series of convolution layers and a skip connection that combine the the input to the block with the output of the convolutional layers. The skip connections also allow gradient information to flow backwards to earlier layers which allows for significantly deeper networks. Because the depth of the ResNet, it was thought that the effective receptive field of the network at the higher levels would be large enough to process semiglobal information despite the size 3 kernels used in the convolution.

4.3. UNet Model

The UNet Architecture was initially developed for tasks involving semantic segmentation [9], but has been adapted for applications that seek to leverage both local and global information. The architecture accomplishes this task by modifying the encoder-decoder model. At each downsample, a skip connection is generated and passed through to the equivalent upsampling layer in the decoder portion of the network. This allows the local features from encoding to be combined with the context features generated in the decoding portion.

5. RESULTS

Our models were evaluated primarily on the basis of mean squared error(MSE) loss between the model velocity output and the ground truth. The ResNet-18 and UNet architectures were not tuned for architectural variations, but acted as a
baseline for our more tuned Encoder-Decoder model. To determine the best Encoder-Decoder model for training on our larger dataset, we conducted a brief search over the following hyperparameters: model architectures as described, number of filters per layer, number, learning rate, weight decay of L2 regularization. The loss of each of these models on our validation set is plotted below in Figure 6.

![Fig. 6: Validation loss vs epoch and hyperparameter](image)

Based on these results, we arrived at a baseline Encoder-Decoder model described in Table 1. This model was used to produce the results seen in Figure 7.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>13-layer AE ('Long')</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Number of epochs</td>
<td>400</td>
</tr>
<tr>
<td>Filters per layer</td>
<td>64</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Baseline Encoder-Decoder hyperparameters

Smaller numbers of filters per layer tended to result in a simpler model that trained slower, while larger numbers of filters tended to overfit and have an increasing loss at later layers. Higher learning rates also tended to result in larger persistent validation losses, while low learning rates remained slow to train. L2 Regularization either performed very similarly to no regularization or caused training to fail totally. The ‘Tiny’ and ‘Straight’ models tended to learn the global context poorly, and only the significant dimensionality reduction in the ‘short’ and ultimately the ‘Long’ model produced the best results.

The results of the models are summarized in Figure 7. The ResNet-18 model performed poorly, with significant low-frequency artifacts throughout the output but especially near the obstacle. The UNet model performed well, especially at reproducing the Y velocities, but underestimates velocities near the top and bottom of the domain in the X velocity output. Finally, the Encoder-decoder model performs well at predicting both X and Y velocities, but has characteristic discontinuous artifacts downstream of the obstacle.

![Fig. 7: Example validation results for all tested models: ResNet-18, UNet, and ‘Long’ Encoder-Decoder](image)

6. CONCLUSION

We find that the UNet and a 13-layer Encoder-Decoder model performed relatively comparably at predicting steady-state flow around randomly placed triangles. We found that models that did not incorporate significant dimensionality reduction, such as the ‘Straight’ model or the ResNet-18 model had trouble learning the global context of the flow and had persistent artifacts. Without the bottleneck forcing global context, the models learned the flow field from the features in their receptive field. The depth of the networks created a large effective receptive fields, however it was not sufficient to recreate the flow fields from the local geometry representations. Given more resources, we would like to attempt solutions to a broader set of problems in flow simulation, such as the evolution of transient flows, turbulent flow, and a wider set of geometries like concave structures where more complex phenomena like flow reversal occur. Also, for 3D flow simulations the same principles can be directly translatable.
7. INDIVIDUAL CONTRIBUTIONS

Steve conducted much of the initial literature review and identified candidate architectures. He was also responsible for creating the ResNet Model and training both the ResNet and the UNet Models. Dimitrios was responsible for the preliminary UNet model and also worked on the theoretical background and presentation/report assembly. Kareem generated the data that was used for training and trained the Encoder-Decoder type models.

8. REFERENCES


Replies to critical reviews

Critiques by group 88:

Q: In the background section, although the application of flow prediction was briefly given, the topic was still too abstract and hard to give a larger picture about what the project is doing. The group may explain what kind of flow is usually viewed in this topic like liquid, gas or something.

A: You are right, and we added more insight in the introduction of this report. We are discussing about liquid fluid flow around various sized objects.

Q: In the dataset slides, only the theory behind the data was discussed. A section of the dataset maybe can be shown.

A: A more detailed description of how we used PyLBM to generate our dataset is discussed in the presentation. Hope this is much clearer now!

Critiques by group 35:

Q: Your data size seems not too big, only 1000 images. Results look good, but I believe they can still be improved greatly.

A: This is an excellent comment and something that we need to consider in future updates. The reason of the limited test set is training time since our target was to explore more models and see what performance do we get and we thought that 1000 was a reasonable number to be able to look into this.

Critiques by group 29:

Q: How different would your results be with concave shapes?

A: This is an interesting question. Short reply to this is that we do not expect the models to behave much differently with concave shapes and the reason is that all the explored models work very well in segmentation-like applications so as long as the heatmap associated with the image has well defined boundaries, either convex or concave, then the model should work well (especially the U-NET). The model would need to be trained to handle flow redirection from concave shapes in the domain, however it should be capable of processing those scenes.

Q: How different would your results be with non-triangular shapes? This would show some insight into the features that your networks are learning.

A: Continuing from the previous replies, triangles provide a combination of the 2 main geometric features that affect fluid flow. These features are continuous streamlined elements and discontinuous corners. Corners cause wakes and flow disruptions while continuous elements redirect flow with minimal disruption to the bulk flow field. Using triangles allowed us most efficiently examine how these geometric features were handled by the network. We feel that random triangular shapes are arbitrary enough so that the model learns deep enough features to be able to operate relatively well with other objects as well and this would be a nice topic of exploration for future implementations. Corners