



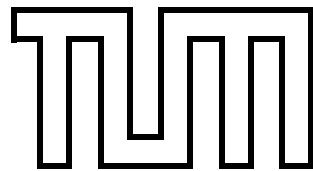
FAKULTÄT FÜR INFORMATIK  
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

**Perceptual Losses for Deep  
Learning on Fluid Simulations**

Hanfeng Wu





# FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatik

## Perceptual Losses for Deep Learning on Fluid Simulations

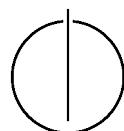
## Perceptual Losses für Deep Learning von Flüssigkeitssimulationen

Author: Hanfeng Wu

Supervisor: Prof. Dr. Nils Thürey

Advisor: M.Sc. Georg Kohl

Date: 15.09.2021





I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.09.2021

Hanfeng Wu



# Acknowledgements

First I want to thank Prof. Nils Thürey and my advisor Georg Kohl, who offered me this great opportunity to work on this interesting topic. They both have inspired me a lot in this 4-month journey. I also want to thank Prof. Kiwon Um, who provided me the primitive code of the Solver in the Loop project and helped me to understand the project better. Without the help from anyone of them, I wouldn't have accomplished this thesis.

## Abstract

This thesis studies the integration of perceptual losses into several methods or approaches that are related to fluid simulation. Perceptual losses are used to compare high level differences, while traditional loss functions can only compute pixel-level differences reliably.

First, some general concepts of fluid simulations, deep learning are discussed. Afterwards previous work on perceptual losses is discussed, which mainly focuses on image-related tasks like [1, 2, 3, 4, 5, 6]. Next, common methods for data comparison, which is an essential aspect of every perceptual loss function, are compared and characteristics highlighted. Next, we explain in detail several tasks we focus on such as Solver in the Loop and super-resolution. Then, in the experiments we integrate perceptual losses into these tasks in order to show that in the context of fluid simulation, the integration of some pretrained perceptual losses should outperform the traditional loss functions.

Eventually we combine the results of Solver in the Loop and super-resolution models together to achieve a lower time cost comparing to running a traditional fluid simulation.

## Zusammenfassung

Diese Arbeit untersucht die Integration von Perceptual Losses in verschiedene Methoden oder Ansätze, die im Zusammenhang mit Fluidsimulationen stehen. Perceptual Losses werden verwendet, um großflächige Unterschiede zu vergleichen, während herkömmliche Loss Funktionen nur Unterschiede auf Pixelebene zuverlässig berechnen können.

Zunächst werden einige allgemeine Konzepte von Fluidsimulationen und Deep Learning erläutert. Anschließend werden bisherige Publikationen zu Perceptual Losses diskutiert, die sich hauptsächlich auf Aufgaben aus der Bildverarbeitung konzentrieren, wie etwa [1, 2, 3, 4, 5, 6]. Als nächstes werden gängie Methoden für den Datenvergleich, der ein wesentlicher Aspekt jedes Perceptual Losses ist, verglichen und deren Eigenschaften hervorgehoben. Danach erklären wir detailliert die verschiedenen Aufgaben auf die wir uns konzentrieren, wie Solver in the Loop und super-resolution. Dann integrieren wir in den Experimenten Perceptual Losses in diese Aufgaben, um zu zeigen, dass im Kontext von Fluidsimulation die Integration einiger Perceptual Losses traditionelle Formulierungen von Losses übertrifft.

Abschließend kombinieren wir die Ergebnisse von den Modellen für Solver in the Loop und super-resolution, um im Vergleich zu einer herkömmlichen Fluidsimulation eine geringere Laufzeit zu erzielen.

# Contents

<b>1</b>	<b>Introductions</b>	<b>1</b>
1.1	Fluid simulation . . . . .	1
1.1.1	Navier-Stokes equations . . . . .	1
1.1.2	Grid Stucture . . . . .	2
1.1.3	Advection . . . . .	2
1.2	Deep learning tasks . . . . .	4
1.2.1	Autoencoder . . . . .	4
1.2.2	Super-resolution . . . . .	4
1.2.3	Solver in the Loop . . . . .	4
<b>2</b>	<b>Related Work</b>	<b>5</b>
<b>3</b>	<b>Metrics and Perceptual Losses</b>	<b>6</b>
3.1	Mean Square Error . . . . .	6
3.2	LSiM . . . . .	7
3.3	VGG-16 loss network . . . . .	9
3.4	Lpips . . . . .	10
<b>4</b>	<b>Datasets</b>	<b>11</b>
4.1	Autoencoder . . . . .	11
4.2	Super-resolution . . . . .	11
4.3	Solver in the Loop . . . . .	12
4.3.1	Simulation tasks . . . . .	12
4.3.2	Generation of datasets . . . . .	14
<b>5</b>	<b>Tasks and Experiments Setup</b>	<b>15</b>
5.1	Autoencoder . . . . .	15
5.1.1	Network . . . . .	15
5.1.2	Application of loss functions . . . . .	15
5.2	Super-resolution . . . . .	16
5.2.1	Network . . . . .	16

5.2.2	Application of loss functions . . . . .	18
5.3	Solver in the Loop . . . . .	18
5.3.1	Network . . . . .	18
5.3.2	Training Set . . . . .	19
5.3.3	Application of loss functions . . . . .	20
<b>6</b>	<b>Results and Analysis</b>	<b>22</b>
6.1	Comparison . . . . .	22
6.1.1	Autoencoder . . . . .	22
6.1.2	Super-resolution . . . . .	24
6.1.3	Solver in the loop . . . . .	29
6.1.4	Time cost evaluation . . . . .	33
6.2	Limitations . . . . .	35
<b>7</b>	<b>Conclusion</b>	<b>37</b>
7.1	Future work . . . . .	37
7.2	Summary . . . . .	38

# Chapter 1

## Introductions

Our main purpose of this thesis is to demonstrate the advantage of adopting the perceptual loss functions in various deep learning tasks on fluid simulation. We try to show quantitatively and qualitatively that by integrating suitable perceptual loss functions in such tasks can outperform the original model trained on traditional loss functions such as MSE. First we want to give a brief introduction to the core concepts of fluid simulation and their related tasks with deep learning techniques.

### 1.1 Fluid simulation

#### 1.1.1 Navier-Stokes equations

The state-of-the-art fluid simulation is based on the famous incompressible equation Navier-Stokes equations

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \quad (1.1)$$

$$\nabla \cdot \vec{u} = 0 \quad (1.2)$$

where  $\nabla$  denotes the gradient,  $\nabla \cdot$  denotes the divergence,  $\vec{u}$  denotes the velocity of the fluid,  $t$  denotes the time step,  $\rho$  denotes the density of the fluid,  $p$  denotes the pressure,  $\vec{g}$  denotes the gravity and  $\nu$  denotes the viscosity of the fluid.

The equation (1.1) is actually a transformation of the  $\vec{F} = m\vec{a}$ [7] and the equation (1.2) describes the incompressibility of the fluid. There are two ways to regard the fluid in the simulation, which are Lagrangian point of view and Eulerian point of view. Lagrangian point view focuses on the behavior of

the fluid parcels, while Eulerian point of view focuses on the velocity fields, pressure fields that vary in time and space. We prefer to use Lagrangian point of view because the existing NS-equations focus more on the physical properties of the fluid parcel rather than describing the mathematical properties of the field. In order to adopt the Lagrangian point of view, we need a grid structure to store the properties of our fluid parcels.

### 1.1.2 Grid Structure

The way of storing the velocity and the density is based on two different grid structure. We store the density in the center of each cell, we call it centered grid or scalar grid. However for storing the velocity, we sample them at the face centers of each cell[8], while the pressure is stored in the center of each grid. In such way we can more easily compute the inflow and outflow of each grid for each direction. As a trade off for such grid structure, the data format would be more complex than normal centered grid. We usually have to stack each velocity array of each dimension together and also adopt them individually in some computations(e.g. computing loss function in deep learning).

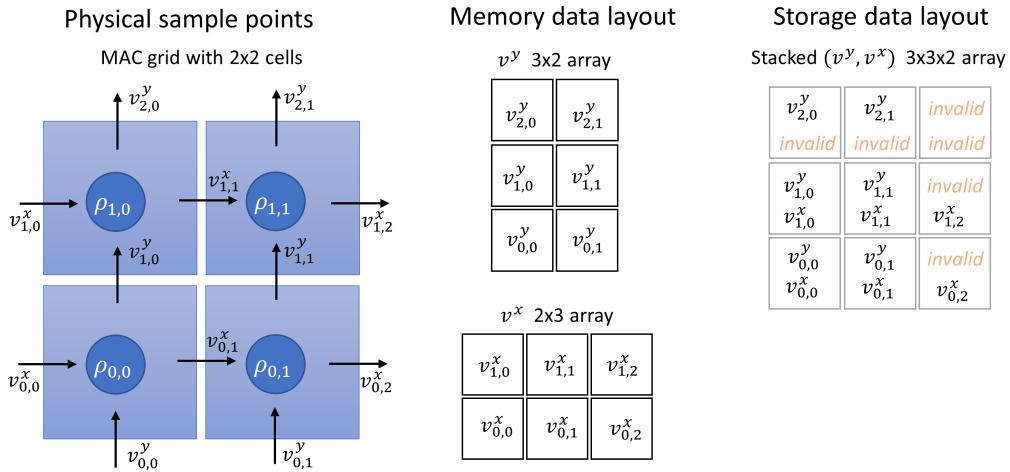


Figure 1.1: Staggered grid format [9]

### 1.1.3 Advection

With the knowledge base of the NS-equation and grid structures we now need the advection algorithm to run the simulation. We adopt the semi-lagrangian advection algorithm. In the lagrangian point of view, if we want

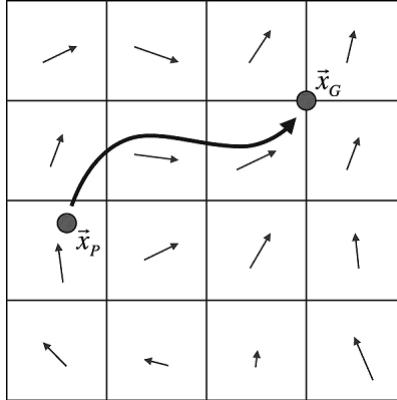
**Algorithm 1**

Incompressible fluid simulation algorithm

```

 $\nabla \cdot \vec{u} \leftarrow 0$  (start with an initial velocity field  $\vec{u}^0$  with the property)
choose a time step  $\Delta t$ 
for  $n \leftarrow 0, 1, 2, \dots$  do
     $\vec{u}^A \leftarrow \text{advect}(\vec{u}^n, \vec{u}^n, \Delta t)$ 
    add the extra force  $\vec{u}^B \leftarrow \vec{u}^A + \Delta t \vec{g}$ 
     $\vec{u}^{n+1} \leftarrow \text{make\_incompressible}(\Delta t, \vec{u}^B)$ 
end for
    
```

to compute the grid value of the  $n+1$  time step of grid position  $\vec{x}_G$ , we should figure out which particle actually flows to the grid  $\vec{x}_G$  from time step  $n$ . If we name the grid position of the particle as  $\vec{x}_P$ , then the new particle at time step  $n+1$  in  $\vec{x}_G$  should have the same physical property as the particle at time step  $n$  in  $\vec{x}_P$



**Figure 1.2:** advection[10]

we first use the euler equation to calculate the  $\vec{x}_P$  as a step before of  $\vec{x}_G$

$$\vec{x}_P = \vec{x}_G - \Delta t \frac{d\vec{x}_G}{dt} \quad (1.3)$$

because  $\frac{d\vec{x}}{dt} = \vec{u}(\vec{x})$  we have

$$\vec{x}_P = \vec{x}_G - \Delta t \vec{u}(\vec{x}_G) \quad (1.4)$$

where  $\vec{u}(\vec{x}_G)$  denotes the velocity sampled at the position  $x_G$ , with the advection we can update the velocity, pressure and density of the whole vector field.

with all the knowledge above, we can formulate a sequence to conduct the basic fluid simulation.(see Algorithm 1)

## 1.2 Deep learning tasks

In this section, we introduce several fluid simulation related tasks which we focus on in this thesis.

### 1.2.1 Autoencoder

The Autoencoder model has a bottleneck shape, which aims to produce the same output as the input. Once trained, the Autoencoder model can be separated into two parts, namely the encoder and the decoder, which can compress the image with the encoder and later decompress the image with the decoder to its former shape.

### 1.2.2 Super-resolution

With the super-resolution model, we can upscale the low-resolution images to high-resolution images. The model is able to up-sample fluid simulation images if we feed similar data to train the model. In this paper, we apply the super-resolution model at the end of the Solver in the Loop model in order to boost the fluid simulation. Meanwhile, the simulation still obtain a similar output to the traditional simulation.

### 1.2.3 Solver in the Loop

When we want to perform our fluid simulation on a lower-resolution domain to decrease the time cost, we always suffer from the numerical error comparing to the simulation running on the higher-resolution domain. It is reasonable because with the same initial state, the vector field on the higher domain has richer information than the one sampled in the lower domain. Um et al. developed a mechanism to learn the difference between the 2 different domains to reduce the numerical error. We use the simulation from a high-resolution domain as the ground truth and make another same simulation with a lower resolution to approach our reference.

# Chapter 2

## Related Work

**Perceptual losses related** The idea of jumping out of the traditional metrics based on  $L^p$  norm has firstly begun by [12], who developed the structure similarity index. Yet it was still not a perceptual loss and can not extract any deep features of the images. The situation has been improved by the discovery of CNN. CNNs are able to extract a lot of deep features, structures and patterns from the image datasets (for example in 2014 [13] has proposed the VGG structure which achieved 92.7% top-5 test accuracy in ImageNet), therefore more and more works tend to use perceptual loss to train their models[2, 3, 4, 5, 6]. In 2016 [1] used intermediate layer comparison from the trained VGG-16 network to evaluate their model for both super-resolution task and style-transfer task. in 2018 [14] developed a new metric used for quality assessment as well as perceptual loss function. In 2020 [15] proposed a new perceptual metric focusing on the numerical evaluation data.

**Fluid simulation related** The PDE model of adopting NS-equation in fluid simulation has firstly been brought up by [8] in 1965. Then PDE models with machine learning started to be popular [16, 17, 18]. More recently deep learning started to play an important role in fluid dynamics[19]. In 2019, Bar-Sinai et al. successfully integrate deep learning method into advection-diffusion problems to infer stencils. With the development of Phiflow [21], beside performing physical simulations, we are able to integrate different deep learning backends (Tensorflow and Pytorch), so that we can track the gradient of the vectors that are stored in different grid formats. With the help of Phiflow, [11] used Phiflow based deep learning methods to reduce the numerical error when performing fluid simulations. In 2020 Kim et al. developed a neural style transfer approach from images to 3D fluids formulated in a Lagrangian viewpoint. Instead of using grid-based structure, Kim et al. used particles for style transfer, which significantly reduce the training time.

# Chapter 3

## Metrics and Perceptual Losses

Perceptual loss functions are used when comparing two images. They are different from traditional loss functions. For instance, MSE or MAE only compute the low level pixel differences. Perceptual losses on the other hand, are normally forward networks trained for specific tasks. For example, VGG-16 network was trained for massive image classification, and LSiM was trained on smoke and fluid simulations. When adopting those perceptual loss networks, we can either compute the MSE loss between the intermediate layers of the trained network, or pass through the trained perceptual loss network depending on the original goal of the chosen network. These perceptual loss networks try to retrieve the high level differences between the images, hence will usually have a better training performance than the traditional loss functions.

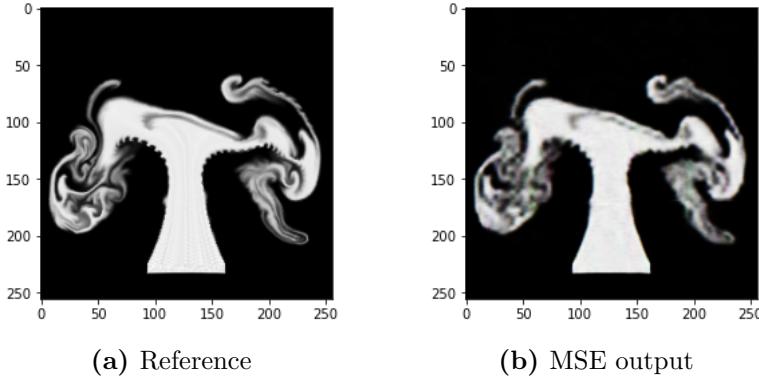
In this thesis we mainly focus on comparing the performance between traditional loss function MSE and perceptual loss functions LSiM, VGG-16 and Lpips. In this chapter we will cover these loss functions that we adopted in this paper and briefly explain their characteristics and their drawbacks. More precise integration and results will be covered in the Chapter 5.

### 3.1 Mean Square Error

MSE is one of the most widely used loss functions.

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \quad (3.1)$$

Where  $Y$  is the reference tensor and  $\hat{Y}$  is the predicted tensor based on the trainable parameters of the model. The optimal case is that the predicted tensor  $\hat{Y}$  is exactly the same as the reference tensor  $Y$ . We can see that



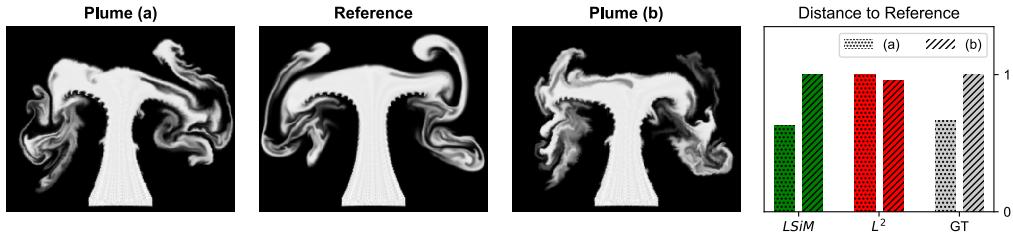
the core idea for the MSE is very general so that this loss function can be applied almost for any models, however the MSE loss function is not suitable for some image related tasks. If the two inputs of the MSE loss function is merely a image with its core content shifted by some pixels, the MSE will produce a very high value, which is not representitive, in another word, the MSE does not have shift or rotate invariance.

Besides, MSE computes the square error of the two tensors, which means that MSE is also prone to the outliers, beacause when it has a very large outlier in the tensor, the result of MSE loss function will be strongly influenced, so in most image construction cases, in order to achieve a lower square error, the output will be a relative blur one compared with the original. Because a sharper image means that the difference between pixel values is huge, which will result in the high square error. On contrary, a blur image means that the pixel values have smaller distance, which will more likely achieve a lower square error. When MSE being adopted, the loss function would rather have a blur area than having a sharp outlier.

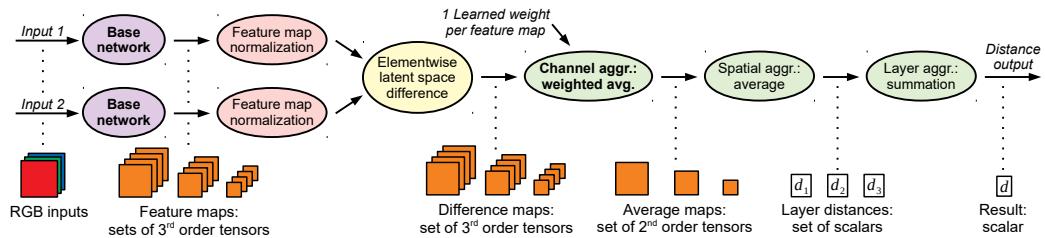
## 3.2 LSiM

LSiM [15] is a neural network-based approach that computes a stable and generalizing metric. Due to the unreliability of the MSE loss in numerical simulation tasks, LSiM has been established. It aims to demonstrate the performance of CNN-based evaluations on numerical simulation tasks. The data used to train LSiM are generated with known partial differential equations (PDEs).

With figure 3.2, we can tell that the  $L^2$  loss function consider plume (b) a closer image to the reference, while the fact is that plume (a) has a smaller distance to the reference image, which is exactly predicted by the LSiM. Notice that the so called ground truth in this case is determined by



**Figure 3.2:** The loss computed between two different plume images and the reference with LSiM and  $L^2$  metrics. While the GT is the ground truth distance determined by the distance of the initial parameters in the data generation.



**Figure 3.3:** The Network structure of LSiM

shifting some initial parameters of the simulation. If the initial parameters are more closer to those of the reference simulation, then the GT distance will be respectively smaller.

The LSiM itself is also a CNN-based metric, see figure 3.3. The input of the network must be two batches of images with 3 channels.

As a metric, it also holds the metric properties  $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{I}$ :

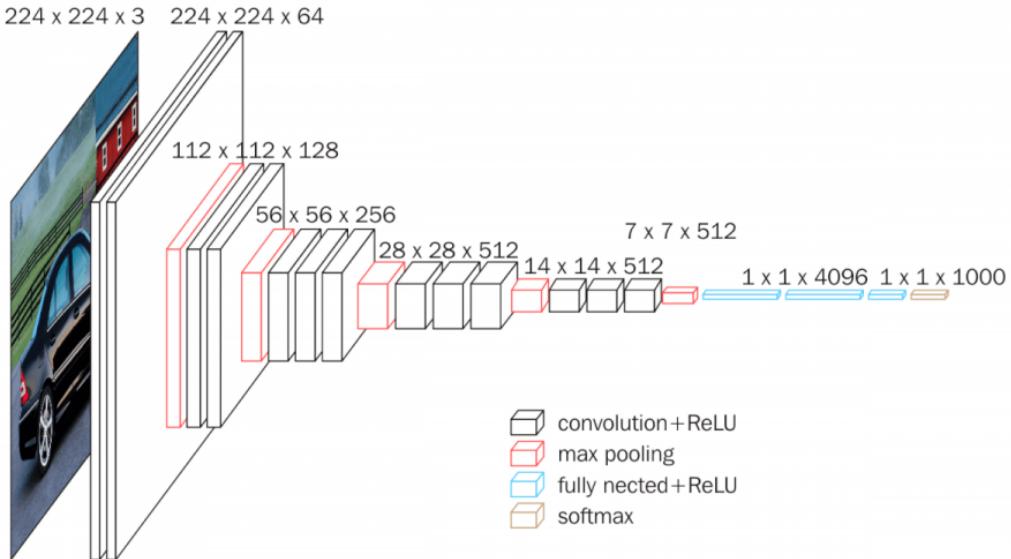
$$m(\mathbf{x}, \mathbf{y}) \geq 0 \quad \text{non-negativity} \quad (3.2)$$

$$m(\mathbf{x}, \mathbf{y}) = m(\mathbf{y}, \mathbf{x}) \quad \text{symmetry} \quad (3.3)$$

$$m(\mathbf{x}, \mathbf{y}) \leq m(\mathbf{x}, \mathbf{z}) + m(\mathbf{z}, \mathbf{y}) \quad \text{triangle ineq.} \quad (3.4)$$

$$m(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y} \quad \text{identity of indisc.} \quad (3.5)$$

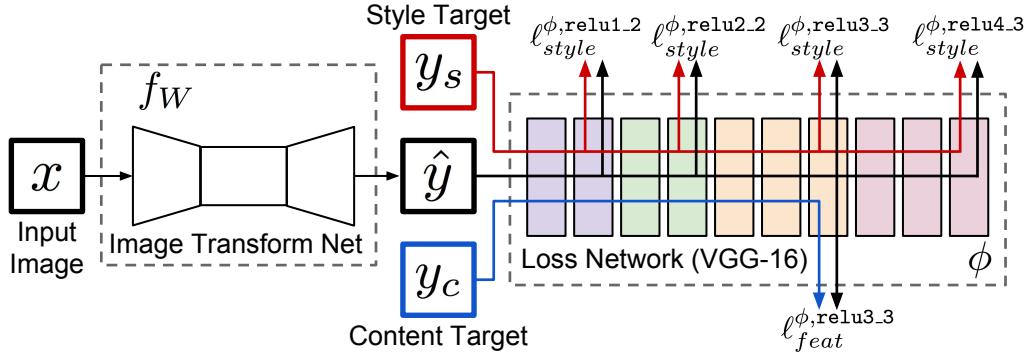
In terms of application, LSiM is already itself a perceptual loss network. According to Kohl et al., we need to form a dictionary consisting of two image inputs. When we feed the dictionary into the LSiM network, the output will be the loss value.



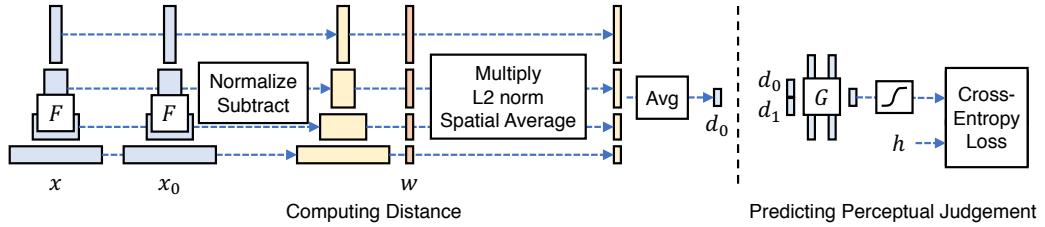
**Figure 3.4:** The network structure of VGG-16

### 3.3 VGG-16 loss network

VGG-16[13] is originally a CNN-based network trained for image classification problems. The author proposed 6 versions of VGG networks with different numbers of CNN layers but applied 3 fully connected layers for all of them at the end to realize the classification functionality. We adopted in the paper only the pretrained VGG-16 network(see figure 3.4) and apply it as a perceptual loss function. This pretrained model achieves 92.7% top-5 test accuracy in ILSVRC-2014, which is a challenge based on a dataset of over 14 million images belonging to 1000 classes. Although the network was not originally trained for computing the distance, we can still use transfer learning to use it as a perceptual loss. In the paper[1], they chose to extract the third intermediate layer outputs and compute the MSE loss between them to realize the perceptual loss functionality(shown in figure 3.5). Apart from that we have also tested extracting the first, second or the fourth intermediate layers to calculate the MSE loss in order to compare the performance with the current method. We found out that computing MSE between the third one has the best performance in our training task.



**Figure 3.5:** How to extract the intermediate layer outputs of pretrained VGG-16 network to compute the loss.[1]



**Figure 3.6:** The Lpips training setup: The distance is computed between the ground truth patch  $x$  and the distorted patch  $x_0$  or  $x_1$  based on the chosen network architecture. The perceptual distance is then trained on a small network  $G$ . The ground truth of the model outputs is the Psychophysical Similarity Measurement  $h$ , and the training is based on the Cross Entropy Loss[14]

### 3.4 Lpips

Lpips as in Learned Perceptual Image Patch Similarity[14] is also a trained network to compare the high level image features based on various distortions. According to the author, they adopted distortions like CNN-based distortions, super-resolution, frame interpolation and so on for the datasets. Based on that, they use several psychophysical similarity measurements on the datasets, trying to approach the human judgements to the images. For the network architecture, they evaluate the SqueezeNet, AlexNet, and VGG architectures. We can consider Lpips as an advanced version of VGG-16, because we do not only consider certain intermediate layer outputs, moreover we take advantage of more intermediate layer outputs and the perceptual judgement, which will be a more reliable than just computing their MSE difference. We in this project choose to use the AlexNet based Lpips.

# Chapter 4

## Datasets

In this chapter we will cover the various datasets that we have adopted for training, validation and testing in the following three tasks.

### 4.1 Autoencoder

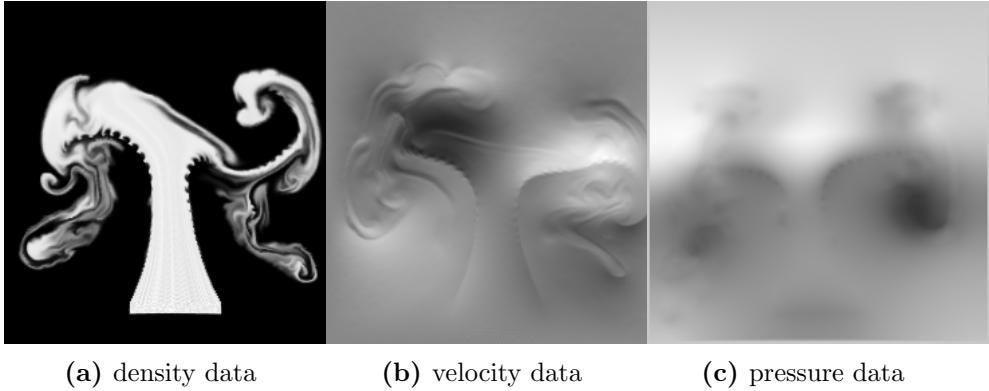
We set up the Autoencoder mainly to have a rough overview of the performance of the perceptual loss functions. Hence, we choose to use the numerical simulation data with the original size of 256\*256, on which the LSiM model was trained.

Autoencoder can be trained without supervision, because the training data is exactly the reference, so we do not need to do any modification to complete the training. However, in order to accelerate the training, we made 2 following modifications:

- since the numerical simulation data are mostly gray scale, which means we can sample them with one channel to boost our training and replicate the channel when certain loss functions demand their inputs in RGB channels. Such techniques can also be applied on any models with gray scale data, you can see more details about it in the Section 5.1.2.
- in order to reduce the GPU usage, we downsample the 256\*256 training data to the size of 128\*128.

### 4.2 Super-resolution

The data we used in the super-resolution task came from various sources. We first used the training data of LSiM model, because these are all numerical



**Figure 4.1:** Training set of LSiM model

simulation data including the velocity, density and pressure images, so we recon that LSiM can outperform the other metrics.

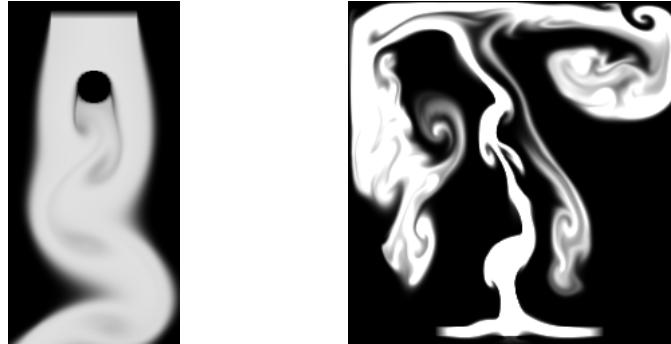
We also want to test the performance of combining Solver in the Loop and super-resolution together to compete the time cost against the simulation performing purely on the higher domain. Therefore we also feed the training data of the Solver in the Loop to the super-resolution model, so that our model can learn to super sample the plume density images when we feed the corrected simulation images on the lower domain. In this way, we are able to obtain the plume images with the same resolution as the higher domain. Apart from that, we also test the our model performance on some random videos data which are not related to numerical simulations but still can be of some assistance to our model.

To create the data set, we form a dictionary for each image. We first down-sample each training image  $I_{high}$  to a lower resolution image  $I_{downsampled}$  (e.g. images of 256\*256 to image of 64\*64), then we resize the  $I_{downsampled}$  to the same image size of the previous  $I_{high}$ , then we get a image  $I_{low}$  with the same image size but a lower resolution as the  $I_{high}$ . In the training process we treat the  $I_{low}$  as the input data of the model,  $I_{high}$  as the groud truth to compute the loss with the model output.

## 4.3 Solver in the Loop

### 4.3.1 Simulation tasks

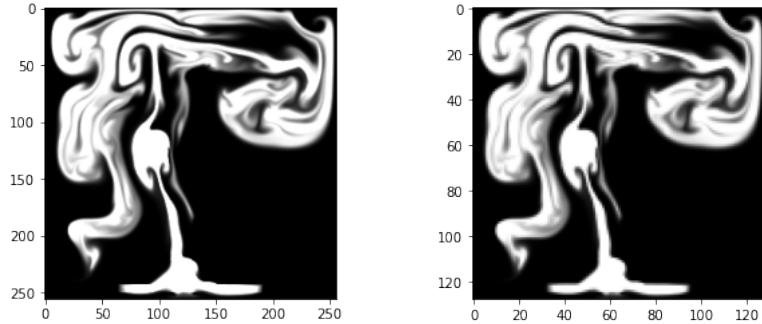
In the Solver in the Loop tasks, we have trained and tested out our model with 2 different fluid simulations, which are the 2-D karman flow and 2-D buoyancy driven flow. Um et al. has mainly trained their model on the karman flow,



(a) Karmanflow example

(b) Buoyancy driven flow example

**Figure 4.2:** Two simulation examples for Solver in the Loop



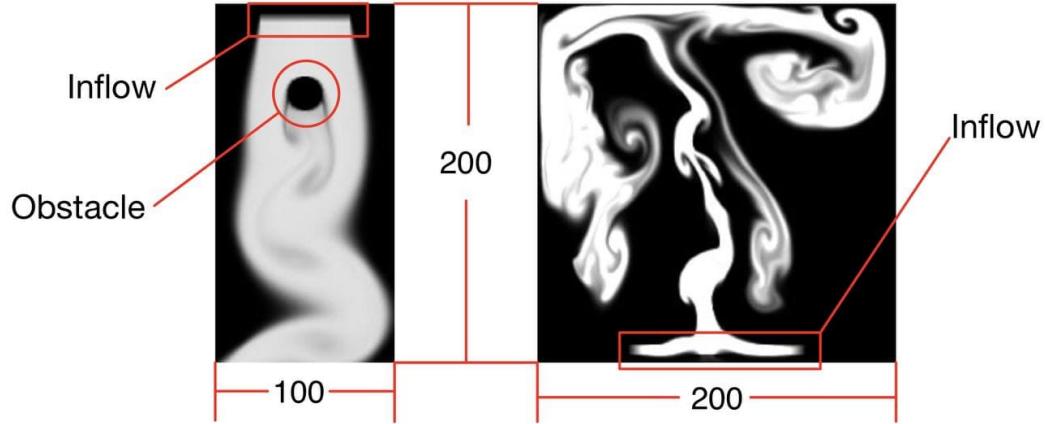
(a) Buoyancy driven flow image with 256\*256

(b) Downsampled buoyancy driven flow image with 128\*128

**Figure 4.3:** Reference images generation and downsampling

based on which we extend the model to a more complex simulation i.e. the buoyancy driven flow.

For both simulation tasks, we have prepared two datasets with different resolutions in order to compare the performance of different loss functions on different resolutions. The two resolutions we chose for the karman flow are 128\*64 and 64\*64, and for buoyancy driven flow tasks we picked 64\*64 and 128\*128. As we want to reduce the numerical error for the model trained on a lower domain, for each resolution ratio we have to firstly generate a higher resolution dataset as the reference dataset. However, we can not compare two datasets with different resolutions. Therefore we have to down-sample the higher resolution training set to a lower one that matches our training domain. Afterwards, the down-sampled dataset with rich simulation details from higher domain is our actual reference set.



**Figure 4.4:** Domain set up of karman flow and buoyancy driven flow

### 4.3.2 Generation of datasets

The tool we used to perform the fluid simulation is Phiflow[21].

For the karmanflow generation, we define an open domain with the size of  $200(\text{height}) * 100(\text{width})^1$  we set the inflow as a  $5 * 50$  rectangle at the top center, and the obstacle as a circle with radius 10 positioned in the very center of the upper half of the domain. The step size  $\Delta t$  we chose here is 1.0. For each simulation we chose to generate 1500 frames for the karman flow.

For the buoyancy driven flow generation, we define a  $200 * 200$  closed domain and set the inflow as a  $5 * 100$  rectangle at the bottom center. We generate 250 frames for the high-resolution simulation and 150 for the low-resolution one. Why we only chose 250 steps and 150 steps for the buoyancy driven flow is because we defined a closed boundary for the buoyancy flow so the simulation after 250 steps are extremely complex, and we also set the timestep to 0.1 to slow down the simulation so we will not have a chaotic frame of plumes. Which helps to make our training easier.

For both tasks, we have a total of 8 simulations with their renolds numbers ranging from  $1 * 10^5$  to  $8 * 10^5$  as the training set. As for the test set we can alter the renolds number and the seed of the simulation to generate a sets of simulation frames that are overall similar but in details deviated from the used training set.

---

<sup>1</sup>notice that domain size is not the same concept as the domain resolution. The domain size helps to locate the position of the inflow and obstacles of the simulation, while the resolution is used to define how precise we should sample our domain. Hence we could perform different resolution formats on the same domain size.

# Chapter 5

## Tasks and Experiments Setup

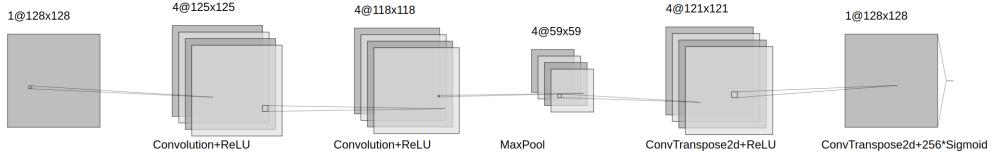
### 5.1 Autoencoder

#### 5.1.1 Network

We use a very simple bottleneck like and fully convolutional neural network for the Autoencoder(see figure 5.1). Due to the gray scale images, the input and output channel size is set to 1. We apply two convolution layers followed by one max pooling layer and two upsample convolution layer. In order to keep the tensor value within a reasonable range, we add before the final output a  $256 \times$ sigmoid layer to ensure that every output value should lie in between [0, 255]. However sigmoid should not be used in the intermediate layers, which will cause vanishing gradient problem, so we use ReLU as the activation function for the intermediate layer. The reason why we do not add a ReLU activation function before the sigmoid is because the ReLU only outputs positive values which will results in that the sigmoid function only outputs values greater than 0.5, then we will lose half of the value range of the images which is not acceptable.

#### 5.1.2 Application of loss functions

Before feeding the input image to the model and compute the loss function, we can still add on some modifications to make our training easier. Since most of our data are smoke images or velocity fields, so they are in gray scale. In this case, we can modify our network in a way that the new input channel size and the output channel size are 1, because the gray scale images only have one channel. With this modification, our model does not need to learn to align all the three RGB channels to the same value, hence it will strongly reduce the training difficulty. However, in the process of applying the loss



**Figure 5.1:** The structure of Autoencoder

function, there will be the problem that many of the perceptual loss functions only accept RGB images, which means the input tensors should have three channels. In order to fix this, we choose to replicate the channel size to three with the same value, so the loss functions will regard our gray scale images as RGB images. In this way we keep the tensors in the training process as one-channel tensors however in the loss function computation process as three-channels tensors, which increase our model performance.

In order to compute the loss functions, we need to define the reference and the model output. Now we know our first input is the model output with replicated channels, and the reference input should be the  $I_{high}$ . The rest is to define the respective metrics. For MSE loss we can directly compute it between two tensors. For VGG-16 loss function, we can feed each tensor into the model and compute MSE loss between their third intermediate outputs. For LSiM we can form a dictionary of these two tensors and pass it to the forward function of the LSiM model. For Lpips we can directly use the pretrained model and feed the two tensors into the function.

## 5.2 Super-resolution

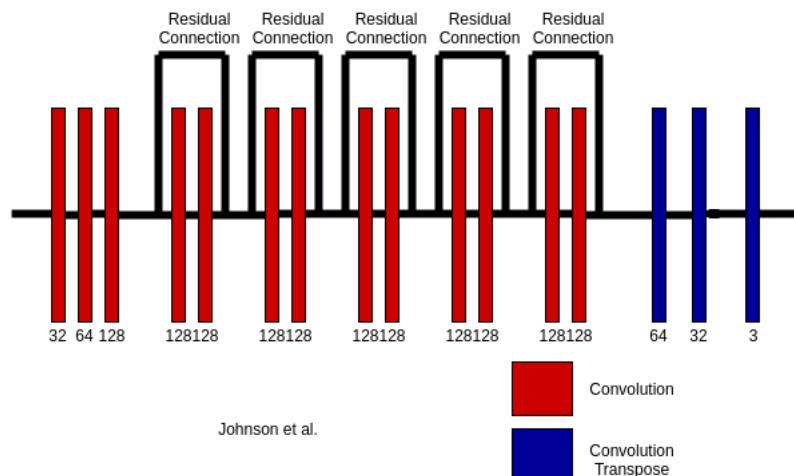
### 5.2.1 Network

We adopt the same ImageTransformNetwork proposed by Johnson et al., which is a fully convolutional network that has the same output tensor shape as the input. The network consists of three Convlayers, three UpsampleConvayers and five residual blocks.<sup>1</sup>

Since the outputs of our model is RGB images, which means the tensor

<sup>1</sup>Notice that we do not use the deconvolutional layer in the UpsampleConvayers as suggested in the paper[23], that deconvolutional layers will likely result into checkerboards artifacts due to the overlapping when doing the reverse convolution operations(see figure 5.3). So we prefer to firstly do the traditional upsampling either with Nearest-neighbor interpolation or with Bilinear interpolation then we apply the convlayer afterwards to obtain the expected up sampled size.

## 5.2. SUPER-RESOLUTION



**Figure 5.2:** The structure of ImageTransformationNetwork of [1]



(a) Using deconvolution. Heavy checker- (b) Using resize-convolution. No checker- board artifacts. board artifacts.

**Figure 5.3:** The comparison between two upsample methods[23]

values lie in the range of [0, 255], so we apply at the end of the last upsampling layer a sigmoid function and then multiply by 256. In this way, we can ensure that our output tensor value won't exceed the range. According to the model structure, the input tensor and the output tensor should share the same shape. So before we feed the low resolution data to the model, we need to first do the upsampling with some interpolation methods(we choose bilinear here) to resize our input images to the expected high resolution, then feed them to the model.

### **5.2.2 Application of loss functions**

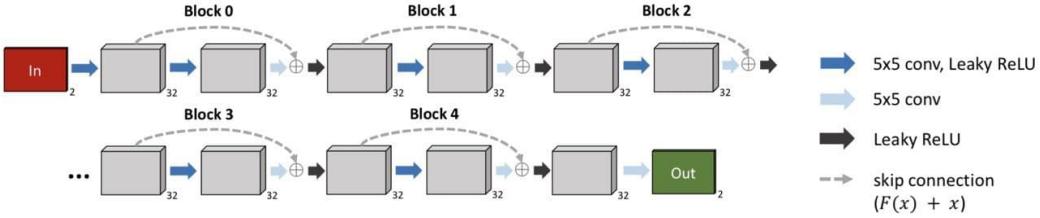
The training process of super-resolution is very similar to that in the autoencoder task. The only different part is that we in the autoencoder use the same training data as the reference data, however in the super-resolution we use the down sampled images as the training data. In terms of applying loss function, we also need to replicate the channels to suit the input shape of certain loss functions. For more details of how to deploy such modification, please turn to the Section 5.1.2.

## **5.3 Solver in the Loop**

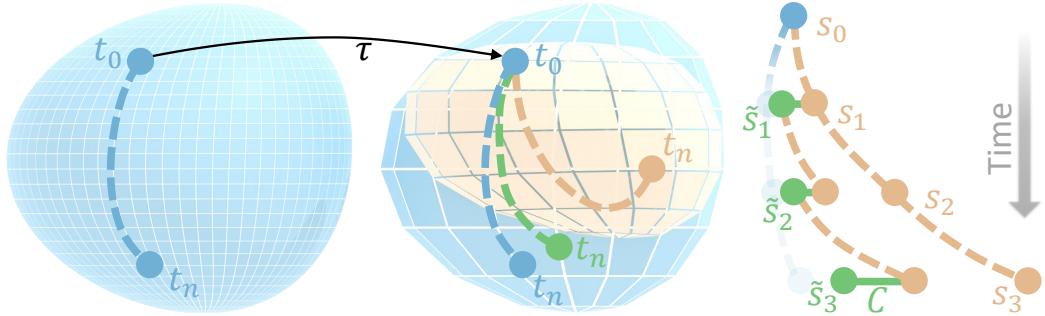
After we generate the training set and its down-sampled version, we can start the training. We will first cover the basics of the network and its training goal, then we will go into details of how to apply the loss functions.

### **5.3.1 Network**

The training network we used is called MarsMoon, which is the same network that Um et al. adopted in the his project(see figure 5.4) we can see that the number of channels for both input and output is 2. This is because the input is the staggered tensor of the velocity which has two directions one represents the y direction and the other represents the x direction. The output of the network is also a staggered tensor however it can not be directly addressed as the velocity but a correction of the velocity, which can be added to the velocity calculated on the lower domain, then the corrected velocity can be applied for the next simulation step. The reason why we do not train the output towards the velocity for the next simulation step is because we want to take advantage of the physical information stored in those tensors, if we should simply get the next velocity from passing the former velocity through the network, then we would not need to perform the physical simulation,



**Figure 5.4:** The network architecture[11]

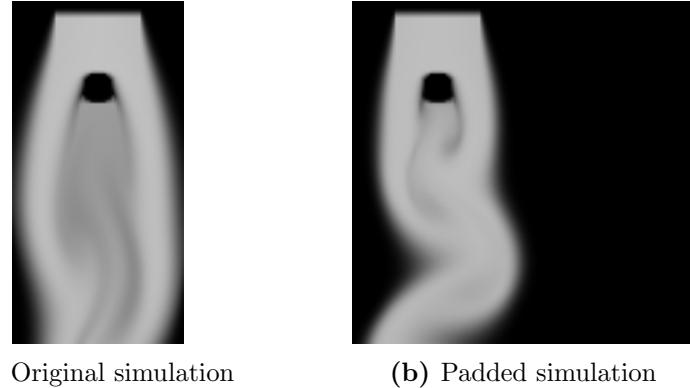


**Figure 5.5:** The training goal is to obtain the suitable correction, with which the simulation on the lower domain can reduce the numerical error. The brown one represents the source simulation of the lower domain without any corrections, the blue one represents the simulation on the higher domain which is our target, the green one is the brown one plus the trained correction, which aims to approach the blue one[11]

which is very unstable. Thus, adding the trained correction to the physical simulation output of the former velocity would at least make sure that our simulation is performing on the right track. That means our consecutive steps of the simulation are based on valid physical simulation, which can not be ensured by purely passing through the neural network.

### 5.3.2 Training Set

In karman flow simulation, we choose not to use the whole training set to train our model, because the first 1000 steps of the low-resolution karman flow have not deviated too much from the high resolution karman flow. So we only try to train the last 500 steps of the total 1500 steps to accelerate the training. We at the beginning trained the lower domain karman flow with the domain resolution of 64\*32, but we found out that the 64\*32 image for LSIM model is too small. So we decided to alter the domain to the resolution of 64\*64, correspondingly the training set also need to be regenerated on the



**Figure 5.6:** Padding plume images to match the input size of LSIM model. However, with the same parameters, there are still some differences between the padded and original karman flow simulation due to the domain size difference.

domain of 128\*128. However the size of the plume remains unchanged, we only pad the domain such that they can be applied to LSIM loss function.(see figure 5.6) For the buoyancy driven flow simulation, it only consists of 250 steps and each step has rich numerical information, so we regard the whole simulation as the training set. Besides, since the buoyancy simulation on the lower domain already has the shape of 64\*64 we don't need to add extra padding to suit the LSIM model.

### 5.3.3 Application of loss functions

In order to compute the loss we have to define the 2 inputs of our loss functions. We know our model output is the correction, however it is unreasonable to directly compare the correction with the reference simulation. So the two inputs of the loss functions should be

- $V$ : The sum of the velocity tensor on the lower-domain simulation and the trained correction .
- $\hat{V}$ : The down-sampled velocity tensor from the higher-domain simulation(reference).

For the MSE loss, which is originally used by Um et al., we can simply calculate their pixel wise square difference and compute their mean. This traditional loss function does not rely on the structure of the tensor as long as they share the same shape. So the MSE loss between two staggered tensors can be easily computed. However the perceptual loss function has

some demands on our tensor shape due to its CNN structure. For example, LSiM requires that the input tensor should consist of 3 channels to represent the RGB channels, while the size of the input tensor should be large enough to pass through several conv-layers. In our case, the output of the model is a staggered tensor with 2 channels which respectively represent x and y directions. Our solution is, that we compute the loss over each direction separately and sum them up. We firstly both extract the x direction tensor  $V_x$  and y direction tensor  $V_y$  from our model output  $V$ , then we also extract the x direction tensor  $\hat{V}_x$  and y direction tensor  $\hat{V}_y$  from the reference velocity tensor  $\hat{V}$ . Our final loss function representation looks like:

$$Loss(V, \hat{V}) = F(V_x, \hat{V}_x) + F(V_y, \hat{V}_y) \quad (5.1)$$

where  $F$  is the respective perceptual loss function. In this way, we are able to treat the velocity tensor on each direction as an image(or triplicate the channel to RGB image) and compare them with their ground truth.

# Chapter 6

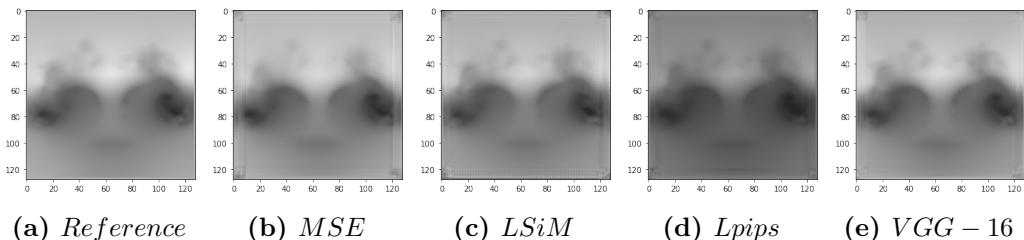
## Results and Analysis

### 6.1 Comparison

#### 6.1.1 Autoencoder

We train the autoencoder model on density, pressure and velocity data of smokes, here are some examples of the training outputs.(see figures 6.2, 6.1) The reference image as well as the training input is on the left side while the rest are different model outputs trained by their respective losses. We can see that for every output, there are some artifacts on each side of the image. The reason might be that our autoencoder model is too simple so it does not provide enough feature extractors to extract and rebuild the features of the smoke images. MSE and LSIM in this case share similar performance, while Lpips provide a darker reconstruction than the others. The model trained by VGG-16 loss function has the best output. We can infer that from the rich information on the edges of the smoke images.

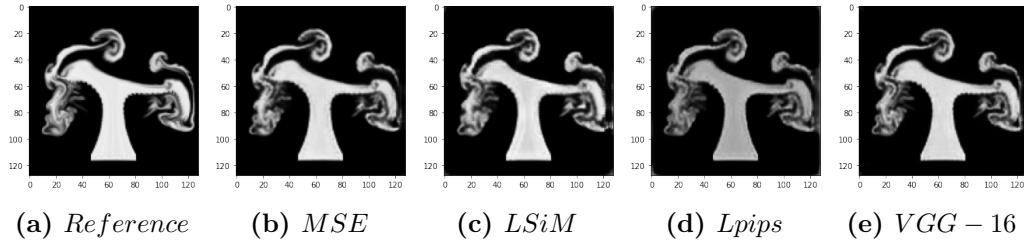
Along with the images we provide the numerical evaluations for each loss. We pick L2, L1, LSIM, MSE, SSIM, Lpips as the methods to compute the numerical distance to the reference image. We compute the average of



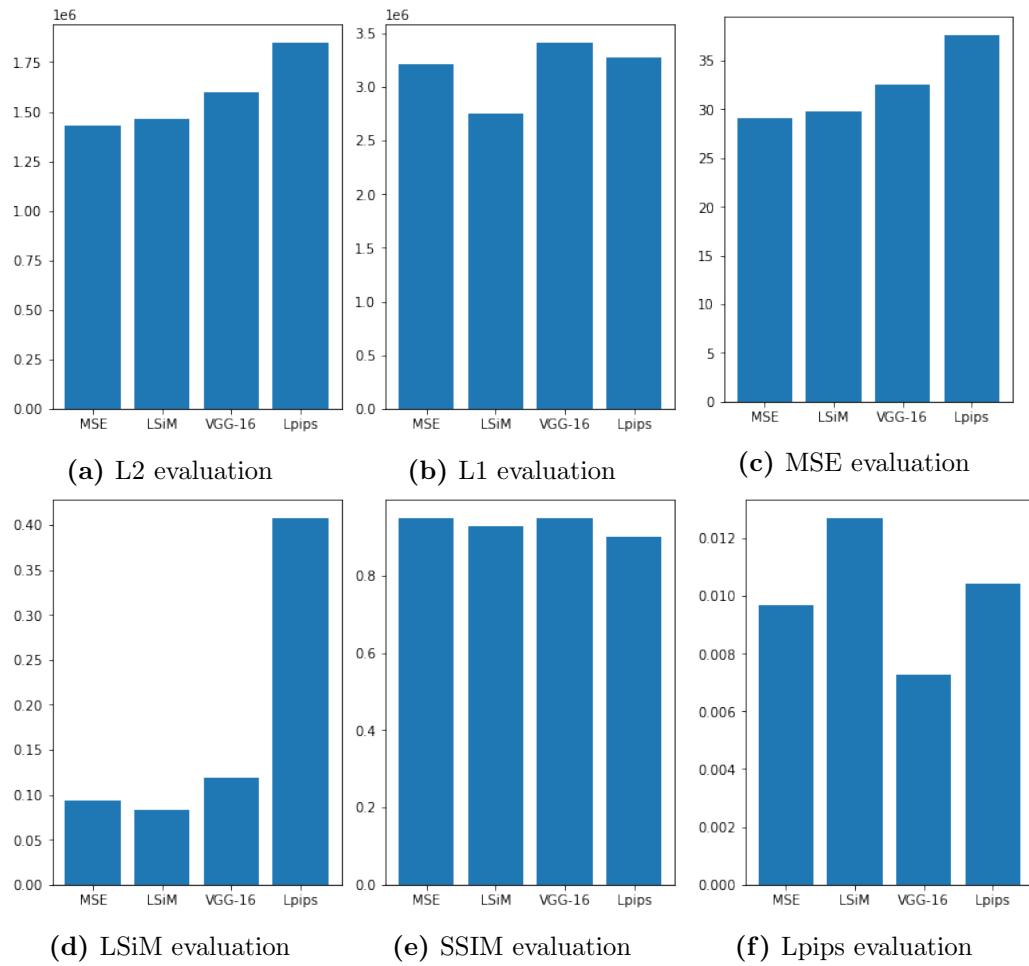
**Figure 6.1:** Pressure images reconstructions of autoencoder task

## 6.1. COMPARISON

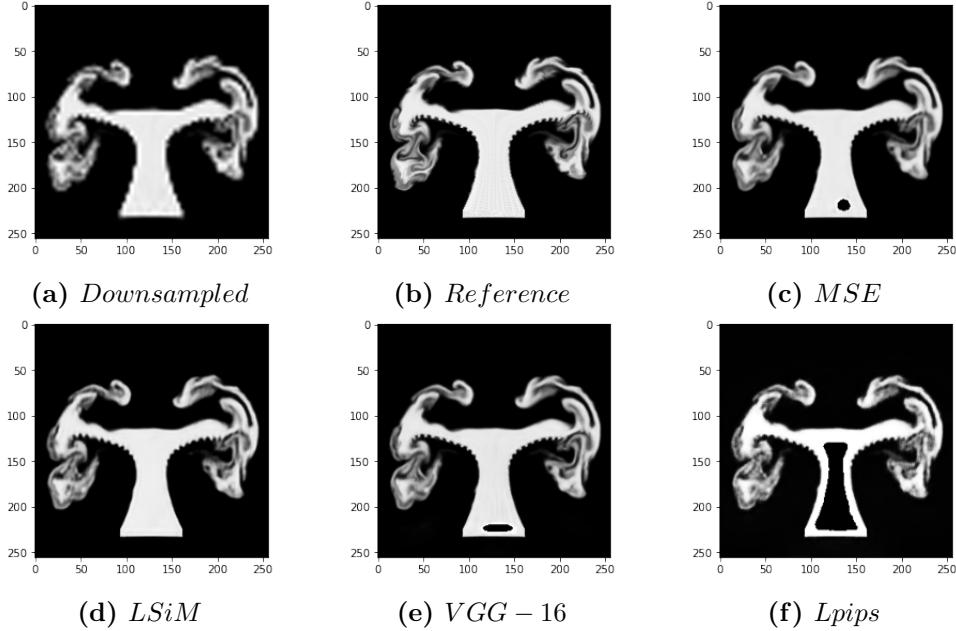
---



**Figure 6.2:** Density images reconstructions of autoencoder task



**Figure 6.3:** Numerical evaluations of each metric on the testset of autoencoder task



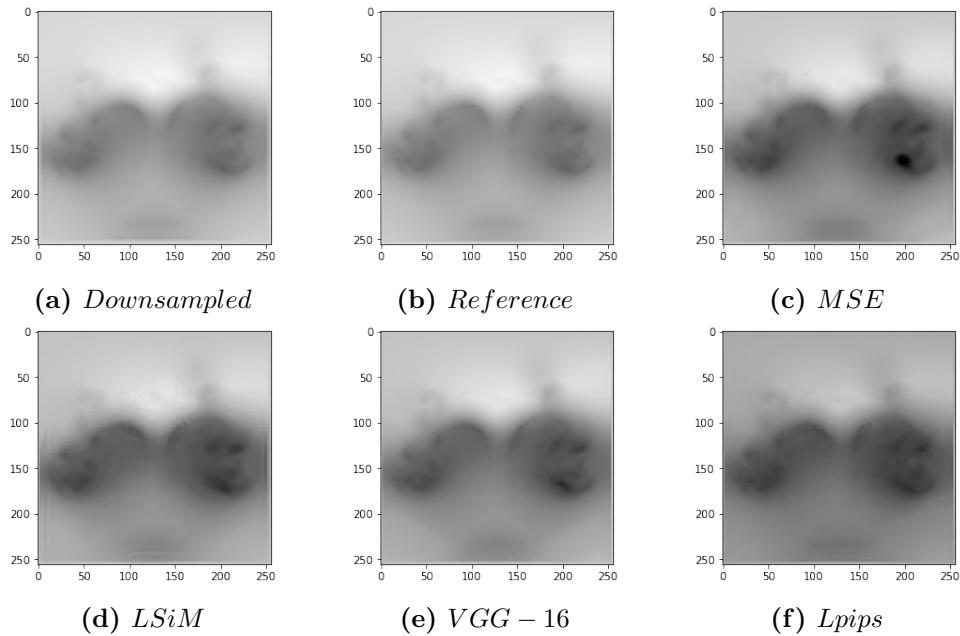
**Figure 6.4:** Smoke density images reconstructions of super-resolution task

each metric over 1056 test smoke images(see figure 6.3). The evaluations agree to our intuition of the images. MSE and LSiM achieve the best score in the evaluation when using themselves as the evaluation metrics, while sharing the same score in the SSIM evaluation. Lpips presents the worst performance among the 4 metrics and VGG-16 achieves the best scores in both SSIM evaluation and Lpips evaluation.

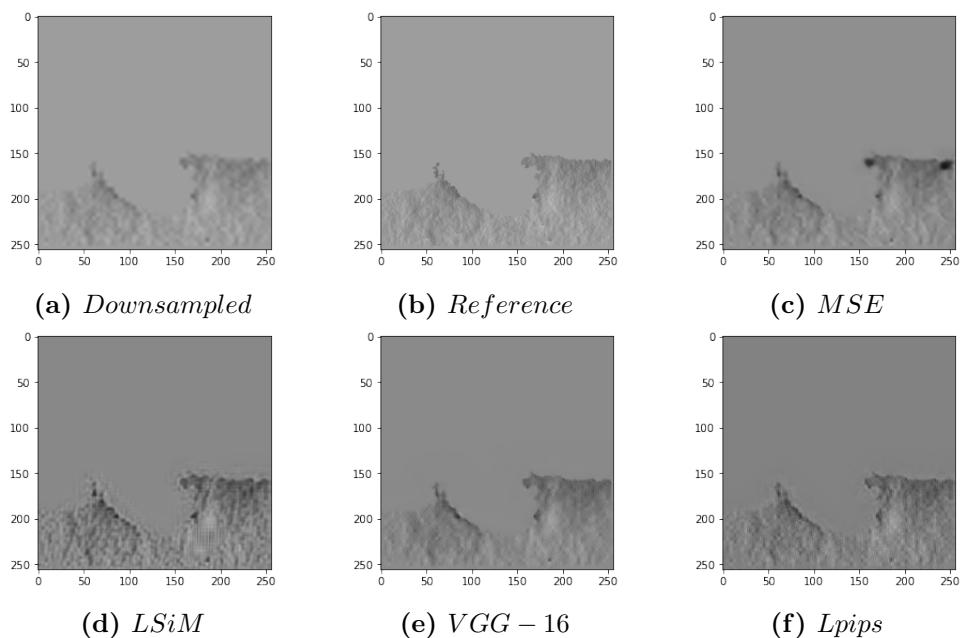
### 6.1.2 Super-resolution

We train our ImageTransformNet based super-resolution model on the data combined by smoke and liquid data. Both smoke and liquid data consist of density, velocity and pressure images. We sample them all in gray scale hence only one channel. We can see in terms of smoke density images(see figure 6.4), every other metric has some artifacts within the plumes, while LSiM reconstructs the downsampled images well. This problem can be solved if we choose to train our model solely on the smoke density data. However the model will then lose its generality. As for the liquid velocity images(see figure 6.6), we can see that MSE generates a more blur image comparing to the others. Because MSE metric is prone to outliers so it tends to achieve a blur image to reduce the loss value. The other two metrics LSiM and Lpips, on the other hand, tend to generate a sharper image to approach the

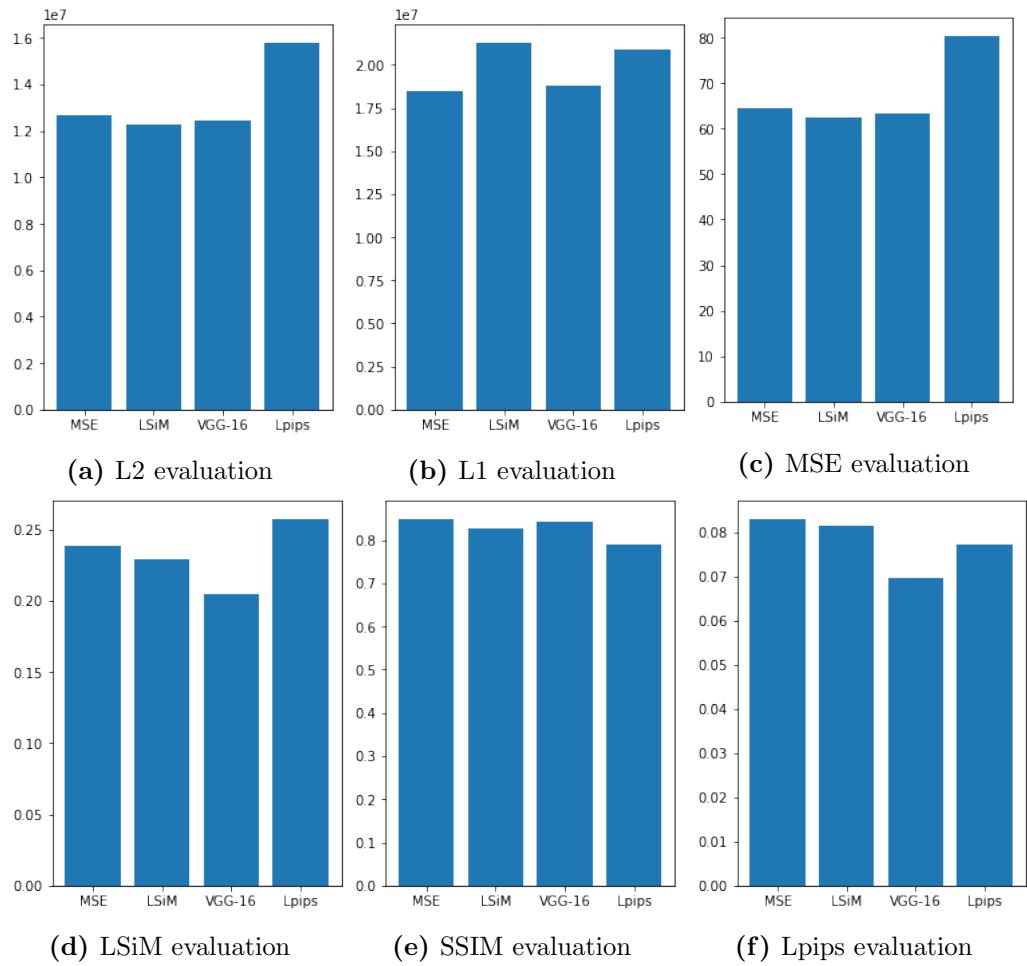
## 6.1. COMPARISON



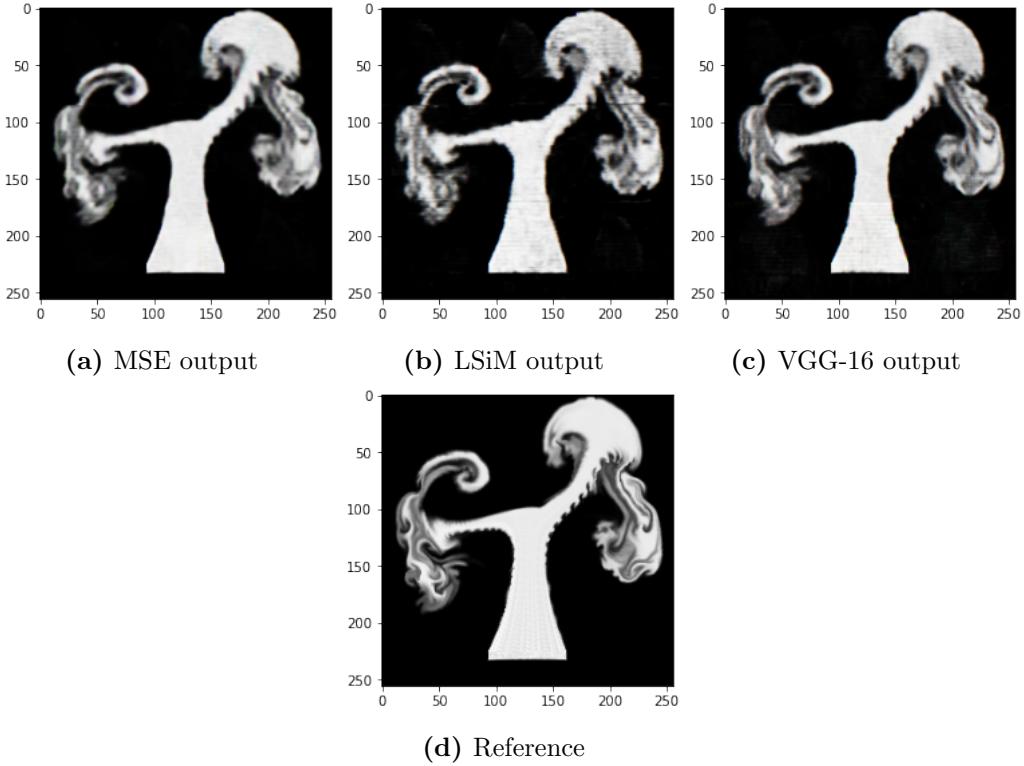
**Figure 6.5:** Smoke Pressure images reconstructions of super-resolution task



**Figure 6.6:** Liquid velocity images reconstructions of super-resolution task



**Figure 6.7:** Numerical evaluations of each metric on test set of the super-resolution task

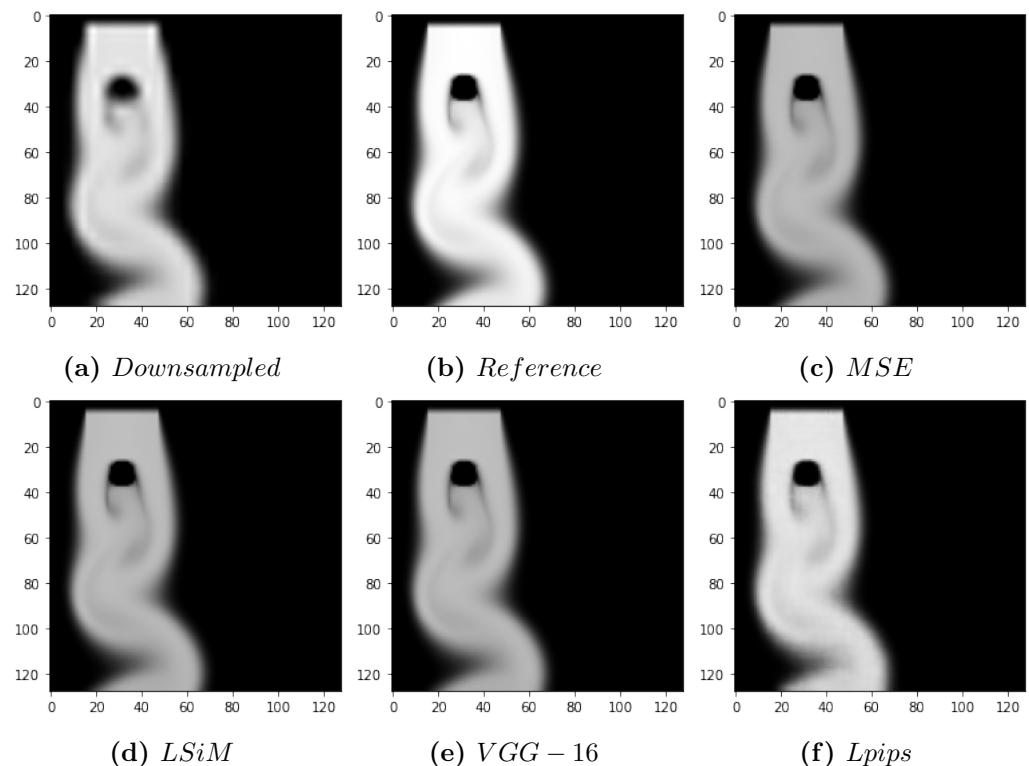


**Figure 6.8:** Besides we have also tested the RGB channels model of ImageTransformNet on the density data. We can see when trained with LSiM metric, the output image has very obvious artifacts that tends to divide the images into 9 parts. This problem can be solved by sampling the images in gray scale and adopt the one channel ImageTransformNet model

reference. In this case, VGG-16 metric achieves the best performance.

We also run the numerical evaluations(see figure 6.7) on all the models we trained. We use 6 metrics to compute the distance between each model outputs and the references. According to the charts, we can tell that Lpips has the worst performance over all metrics, which is similar to the result of the autoencoder task. The other three metrics share the similar results in L2 and SSIM evaluation. VGG-16 achieves the best score in LSiM evaluation, LSiM itself achieves the second best and MSE the third. Overall VGG-16 has the best performance in terms of the numerical evaluations.

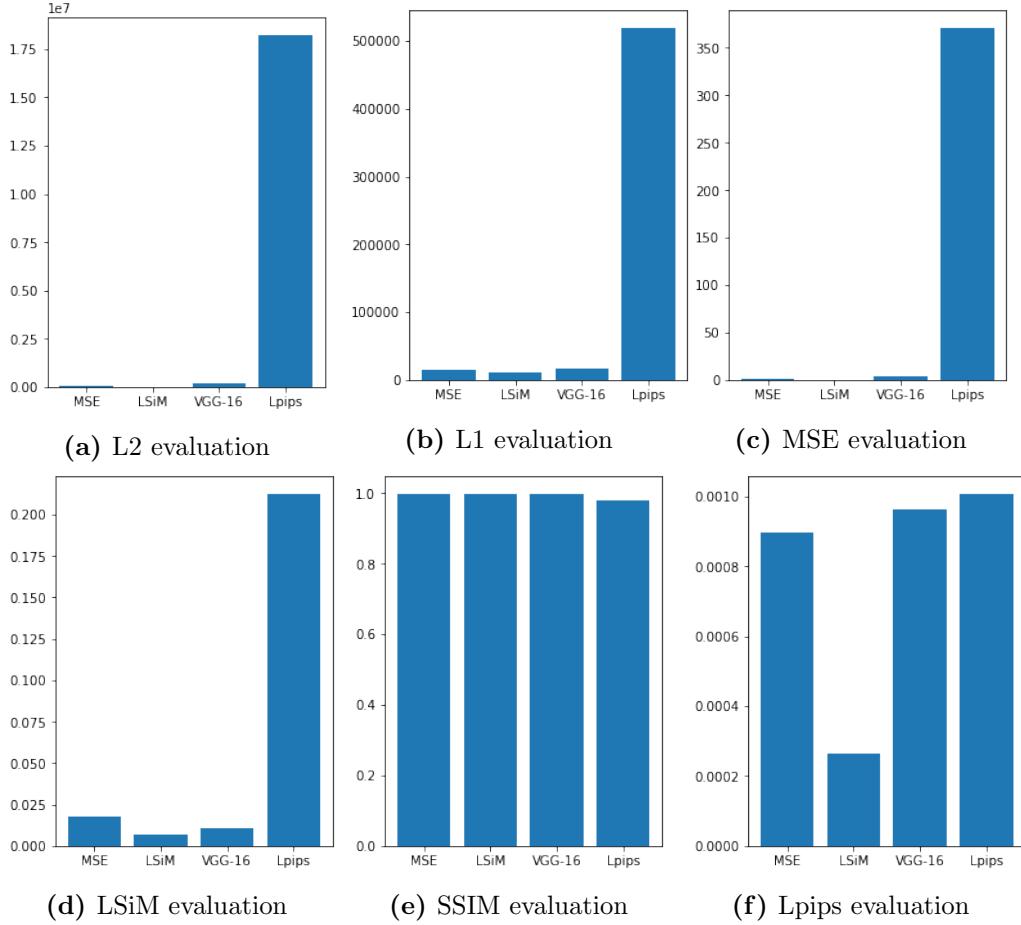
We also test the karman flow data(see figure 6.9) from Solver in the Loop in the super-resolution model. The numerical evaluation is quite surprising(see figure 6.10). Despite Lpips outputs have the closest color to the reference, it has the worst score over all the evaluations. LSiM achieves the



**Figure 6.9:** Karman flow images reconstructions from super-resolution models trained by different metrics

## 6.1. COMPARISON

---



**Figure 6.10:** Numerical evaluations of each metric on test set of the super-resolution task trained on karman flow data

best score in all evaluations. However intuitively, we can hardly tell the difference between the MSE, LSiM and VGG-16 outputs.

### 6.1.3 Solver in the loop

After training lots of models on both buoyancy driven flow and the karman flow of different resolutions, we have following results. In terms of numerical evaluations, we compute the distance by various metrics(L1, L2, MSE, LSiM, SSIM, Lpips) between the simulation corrected by the models trained by different loss functions, and the down-sampled reference simulation. We compute the distance between each corresponding frames pair and compute their average as the result.

**Karman flow** From the comparisons from the figures, we can see that the source simulation(see figure 2) deviates from the reference (see figure 1) after 1000 steps. Lpips (see figure 5) obviously has the worst performance, the plume is shifting to the right and there are noticeable artifacts near the obstacle. MSE, LSIM and VGG-16(see figures 3 , 4 , 6) share similar outputs after step 1000.

In our numerical evaluations(see figure 6.11), we only compare the simulation frames after step 1000 with their reference frames. We didn't compare the whole 1500 steps because we believe that considering all steps is not suitable since the reference images of the training only come from the latter 500 simulation steps. In terms of L1, L2 and MSE comparisons, we can see all corrected models have better scores than the source simulation. In the other evaluations, LSIM achieves overall the best score, followed by the VGG-16. MSE is not better than the source simulation, which is deviated from Um et al.'s work. We think the deviation may result from that we didn't choose a large multiple steps size<sup>1</sup>, or a large training epoch. Still we prove that LSIM or VGG-16 outperform MSE in terms of this specific task.

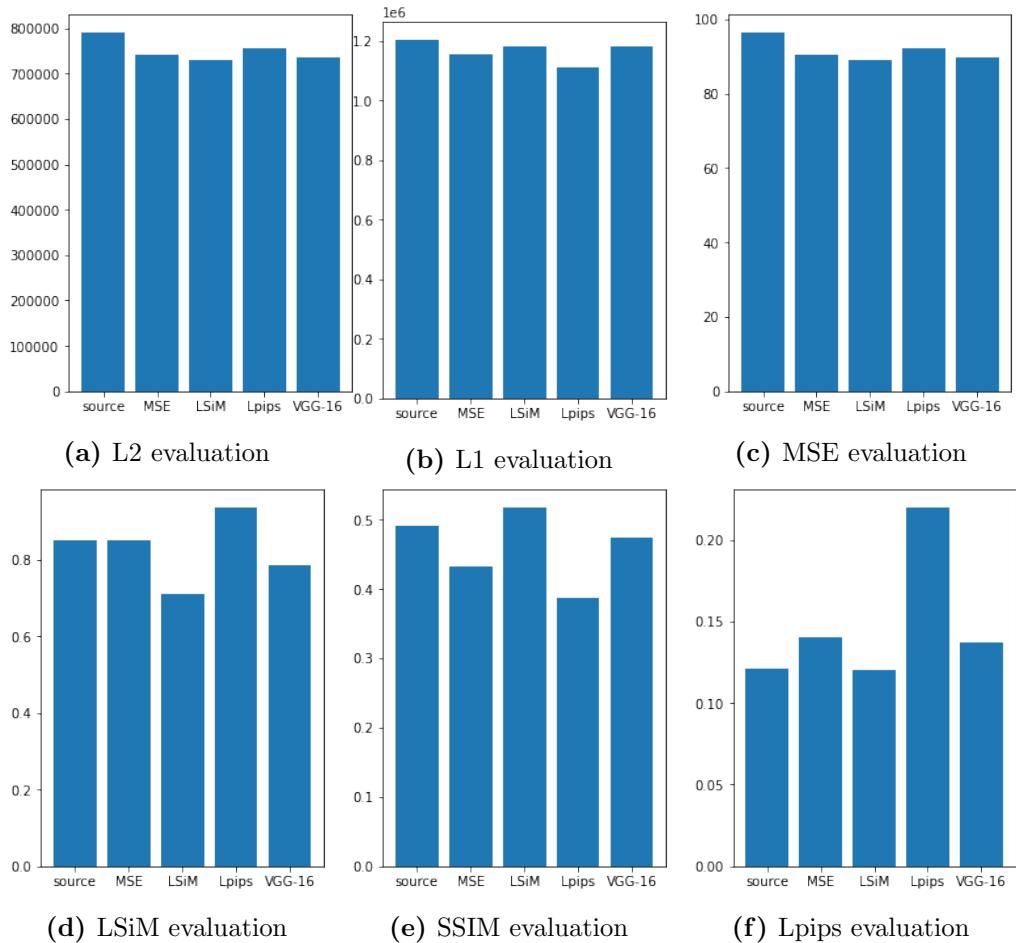
We also tested the karman flow on a lower resolution, whose simulation was running on a 64\*64 domain. The numerical evaluations(see figure 6.12) are a little different from the training on a higher domain(see figure 6.11). The outputs from all trained model(see figure 7) are intuitively similar. We only show the last frame of each model output including the reference and the source simulation, but we add the respective gifs as the hyperlink in the caption. We can see on a lower domain, Lpips metric outperform the others, while LSIM is the worst over all 4 metrics. However all the metrics outperform the source simulation in terms of MSE, L2, SSIM and Lpips evaluation, especially in Lpips and L2(MSE) evaluation. Every corrected model has made a strong difference comparing to the source simulation.

**Buoyancy driven flow** We first run the evaluation on a test set of the buoyancy driven flow on the higher domain(128\*128). We can see from the simulation images, the corrected simulation(see figures 10, 11, 12) can hardly approach the shape of the reference images(see figure 8) Only the corrected simulation trained by Lpips(see figure 13) shares a little similarity with the reference. The same concept is also revealed in the numerical evaluation of these models(see figure 6.13). Except Lpips, other metrics can not even outperform the uncorrected source simulation.

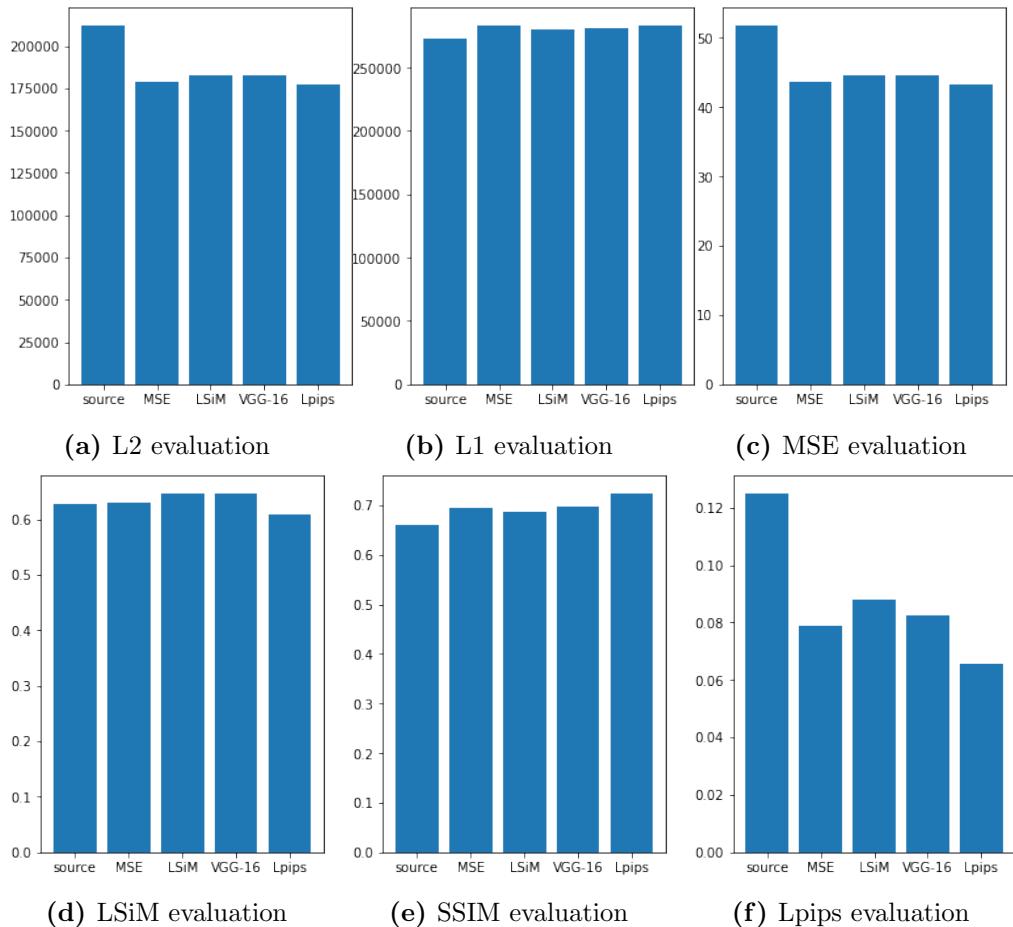
---

<sup>1</sup>The multiple steps size is the number of the consecutive frames we focus on in one training step. It's similar to batch size, the larger the multiple steps size is, the more GPU memory it requires.

## 6.1. COMPARISON



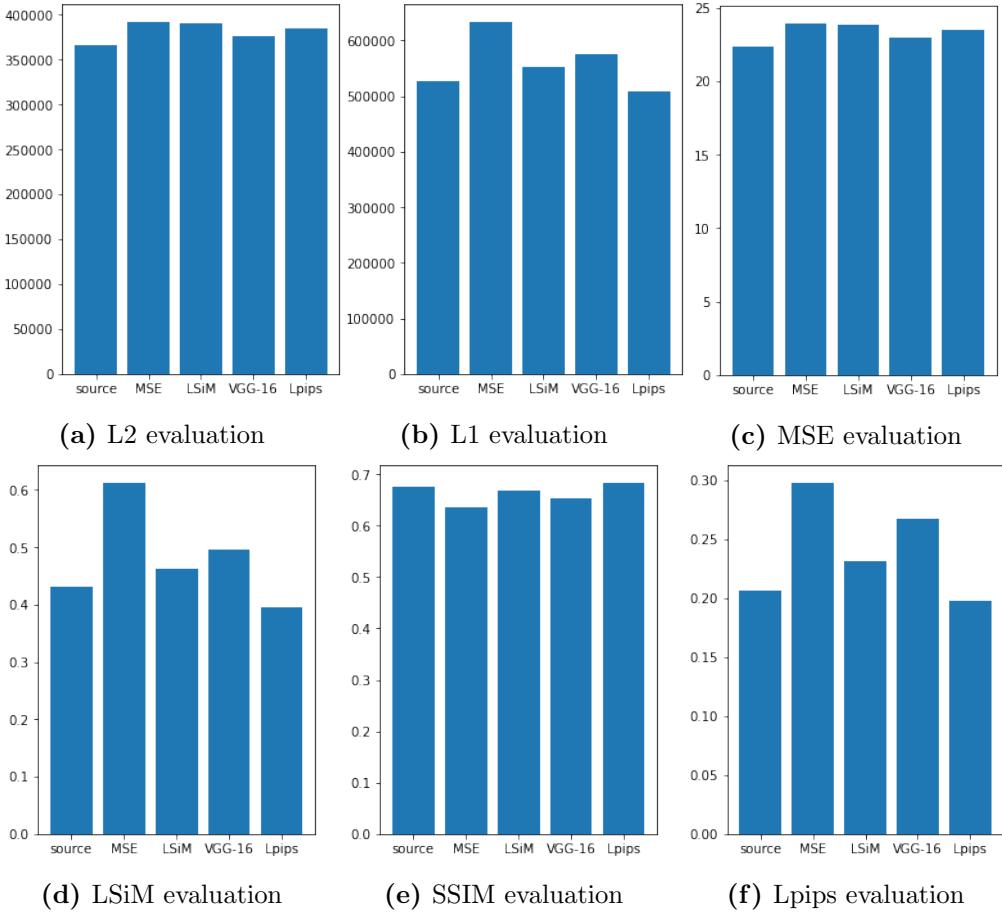
**Figure 6.11:** Karman flow numerical evaluations of each metric on test set with the reference on the  $128 \times 64$  domain



**Figure 6.12:** Karman flow numerical evaluations of each metric on test set with the reference on the 64\*64 domain

## 6.1. COMPARISON

---



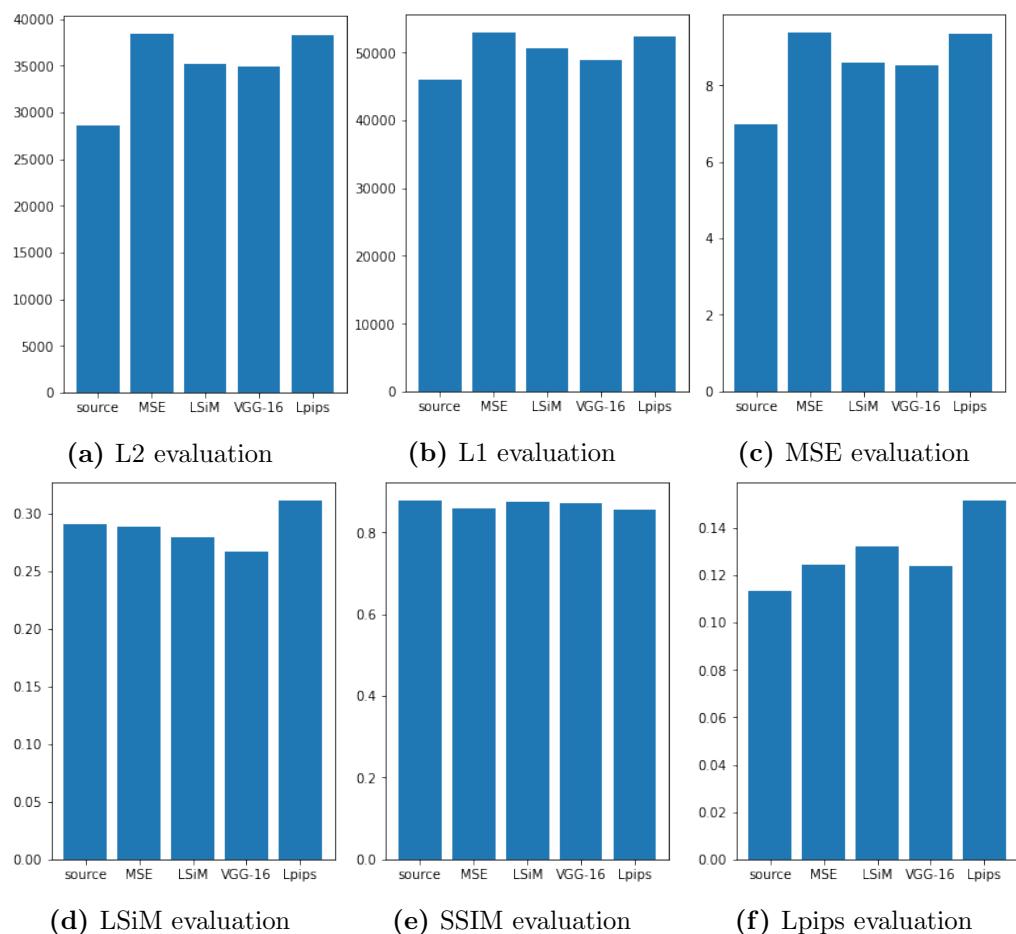
**Figure 6.13:** Buoyancy driven flow numerical evaluations of each metric on test set with the domain resolution of 128\*128

For completeness we also turn to the buoyancy driven flow on the lower domain(64\*64)(see figure 14), the result is similar or even worse. We can see from the numerical evaluation(see figure 6.14), except for the LSiM evaluation, the source simulation outperform all the corrected simulation. However, if we only look at the comparisons between the corrected simulation, we can see that generally LSiM and VGG-16 metrics achieve better scores than MSE.

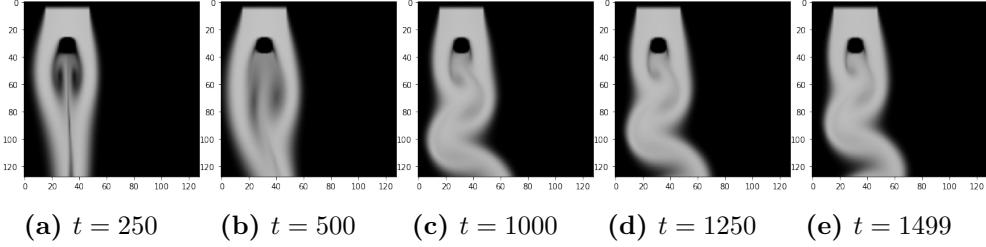
### 6.1.4 Time cost evaluation

In this evaluation we combine the SOL model and the super-resolution model for karman flow on the domain of 64\*64. We want to demonstrate the time cost and the quality difference between

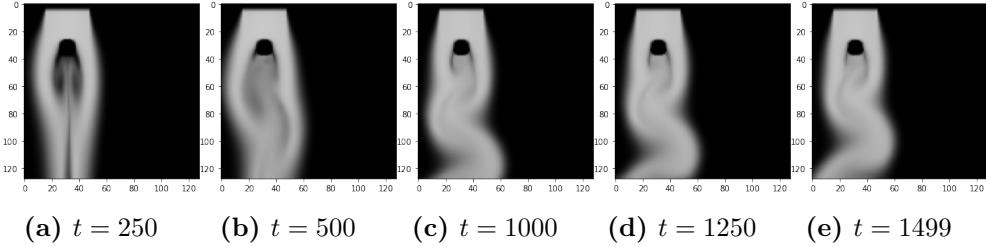
- running the pure physical simulation on the higher-resolution domain(128\*128),



**Figure 6.14:** Buoyancy driven flow numerical evaluations of each metric on test set with the reference on the 64\*64 domain



**Figure 6.15:** Karman flow reference simulation [gif here](#)



**Figure 6.16:** Karman flow lower-resolution simulation + SOL model + super-resolution model [gif here](#)

without any deep learning models involved (see figure 6.15)

- running first the corrected physical simulation on the lower-resolution domain( $64 \times 64$ ) by the pretrained SOL model and then apply the super-resolution model to the same resolution of the higher domain( $128 \times 128$ ) (see figure 6.16)

The outputs are satisfying. We can see barely any artifacts from the outputs and the simulation is similar to the reference. We average the time cost of generating the 2 simulations. We have 14 min 6 sec on average for generating 1500 images of the reference(see figure 6.15) and 4 min 23 sec for generating the second simulation(see figure 6.16). The time cost reduced by 70% if we choose to generate a corrected low-resolution simulation and adopt the super-resolution model rather than performing the pure physical simulation on the higher-resolution domain. Meanwhile, the outputs are similar in terms of both the quality and the shape of the plumes.

## 6.2 Limitations

**Autoencoder** We think the network we build for the autoencoder task is rather small and shallow. It only consists of 4 feature extraction layers, so the distance between using perceptual losses and traditional loss functions may

not be explicitly revealed. We only choose to train the model that handles the gray scale images, while in more complicated use cases there could be fluid images with RGB channels. For such conditions we need to train a more complicated model.

**Super-resolution** When we use the data sets that consist all the density, velocity and pressure images, the model seems very hard to generalize for such data sets. We can notice lots of artifacts within the density reconstruction plumes. We might need to add some regularization to the network to let the model apply to the data sets with high variations. We also tested the RGB-channel version of the network. The outputs of the network still consist of some artifacts. We can see vaguely that the outputs are divided into 9 parts, we may need to redesign the network to eliminate such artifacts.

**Solver in the Loop** Due to the GPU memory limitation (we used a GTX 1080Ti with 11GiB of memory to train the SOL model), we are unable to choose a higher multiple steps size to train the model.<sup>2</sup> We choose the multi step size in both buoyancy driven flow and karman flow as 8. Um et al. used some similar multiple steps sizes for karman flow, however 128 for the buoyancy driven flow. We think that the reason why our buoyancy driven flow is not working as expected may come from that we choose a too small multiple steps size.

---

<sup>2</sup>The multiple steps size is the number of the consecutive frames we focus on in one training step. It's similar to batch size, the larger the multiple steps size is, the more GPU memory it requires.

# Chapter 7

## Conclusion

### 7.1 Future work

We can try to extend the model of super-resolution and autoencoder to the RGB channel and test on fluid data with RGB-channels. Adding more regularization to the training would also be an improvement.

For the Solver in the Loop model, we can choose to upsample the model outputs and compare them with the reference instead of downsampling the reference to compare with the model outputs. In such way, we do not need to pad our low-resolution domain to suit the minimal input size of the LSiM model. Besides, we can consider all the 1500 simulation steps of karman flow and enlarge the simulation multi steps and data sets to train a more general model. We can also include more fluid simulation types(not just karman flow and buoyancy driven flow) to the data sets and train them on a more higher-resolution domain to see whether the distance between MSE and perceptual losses becomes bigger.

Moreover we can try to compare the performance of perceptual losses in other tasks, e.g. inpaintings[24], style transfer for the fluids[22] etc. Especially in [22], the data sets are fluids with RGB-channels. We can try out different perceptual loss functions for computing the loss of shapes and the loss of styles, which is similar to[1]'s model architecture(see figure 3.5).

Using the super-resolution model + Solver in the Loop model to accelerate the numerical simulation is also a very interesting topic. The current version only supports very limited simulation types(karman flow). In order to add more generality to the models, we need to expand our datasets to include more numerical simulation data for training both the super-resolution model and the Solver in the Loop model. Further more we can expand such models to 3-D version. With the huge decrease of simulation time, we may be able

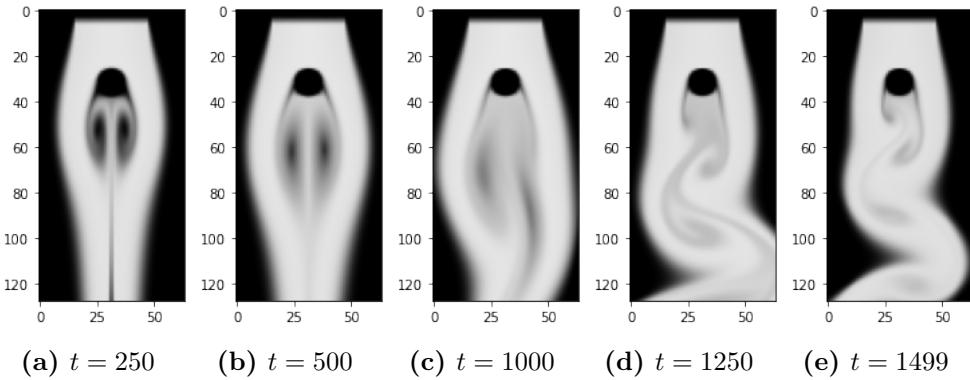
to develop real-time Physics-based simulation in the future.

## **7.2 Summary**

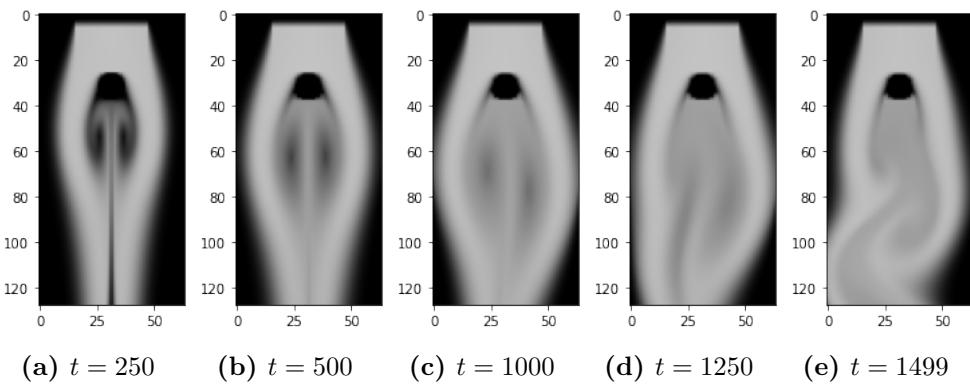
In this thesis, we successfully proved the advantage of using perceptual losses in some fluid simulation related tasks. However, in some tasks, using perceptual losses is not necessarily better than using traditional loss functions. There is no one dominant perceptual loss function for every task. For the super-resolution model(see figure 6.7) on the smoke and liquid data sets, the VGG-16 demonstrates the best performance over all the 4 metrics. While for the same model on the karman flow data sets, LSiM achieves the best score(see figure 6.10). In the super-resolution task, we also notice that the distance between using MSE and perceptual losses is smaller when we train the model that handles gray scale images(see figure 6.4) rather than RGB-channel images(see figure 6.8). The similar result also appears in the Solver in the Loop task. After running tests on both the karman flow and buoyancy driven flow, we can see that the perceptual losses tend to perform better in more complicated conditions. Both for the karman flow(see figures 6.11 vs 6.12) and buoyancy driven flow(see figures 6.13 vs 6.14), the numerical distance between using perceptual losses and MSE is bigger when running the simulation on the higher domain. Similarly, if we compare the higher resolution simulation between karman flow and buoyancy driven flow(see figures 6.11 vs 6.13, 6.12 vs 6.14), we can tell that the numerical distance between using perceptual losses and MSE is bigger when running the buoyancy driven flow simulation. Despite the fact that some of the corrected simulations are not even better than the source simulation, we can still judge in terms of loss functions, that perceptual losses tend to perform better under more complicated conditions.

## 7.2. SUMMARY

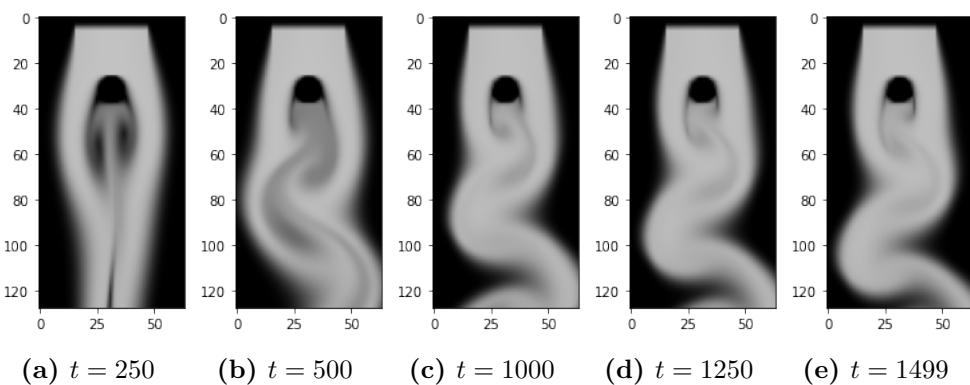
---



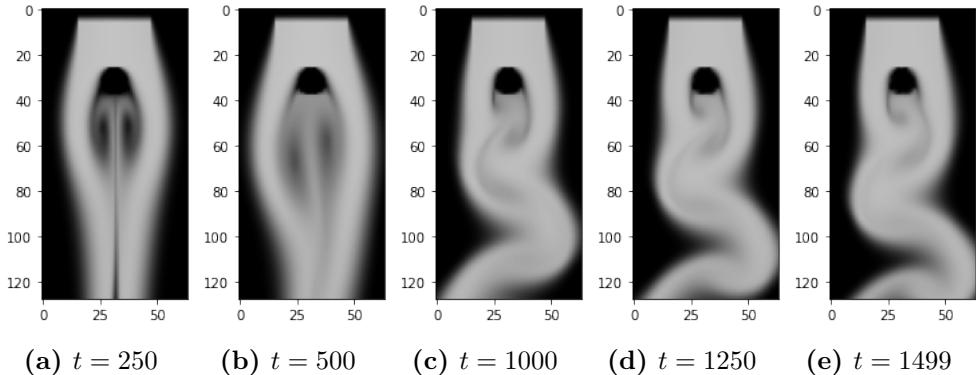
**Figure 1:** Karman flow reference simulation [gif here](#)



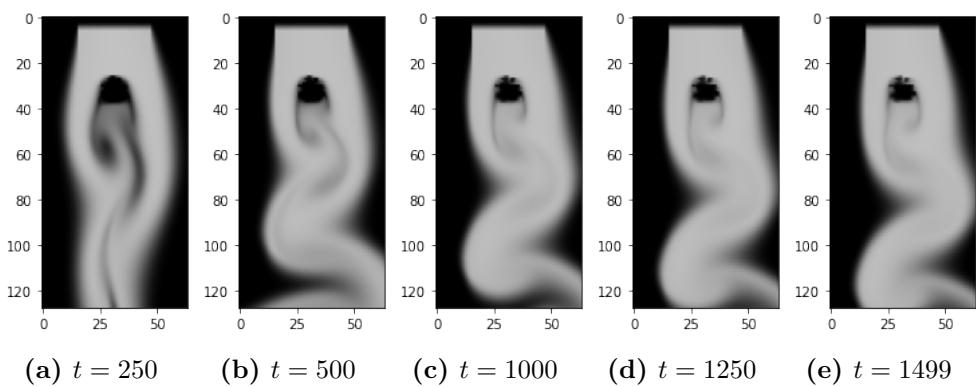
**Figure 2:** Karman flow source simulation without correction [gif here](#)



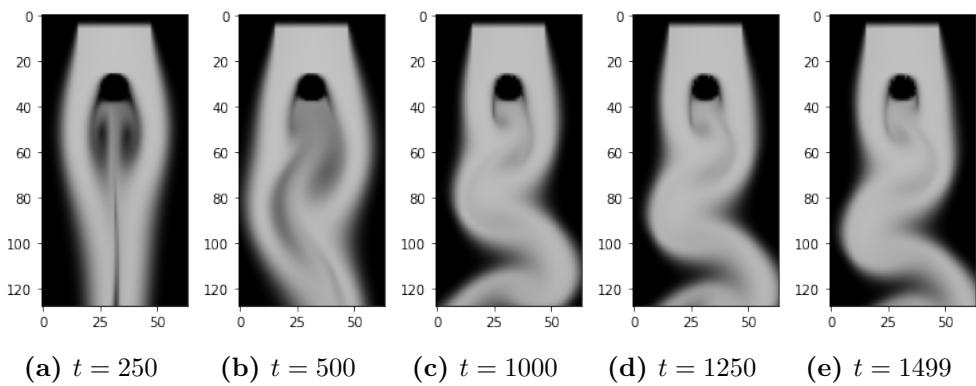
**Figure 3:** Karman flow corrected simulation trained with MSE on the  $128 \times 64$  domain [gif here](#)



**Figure 4:** Karman flow corrected simulation trained with LSIM on the 128\*64 domain [gif here](#)

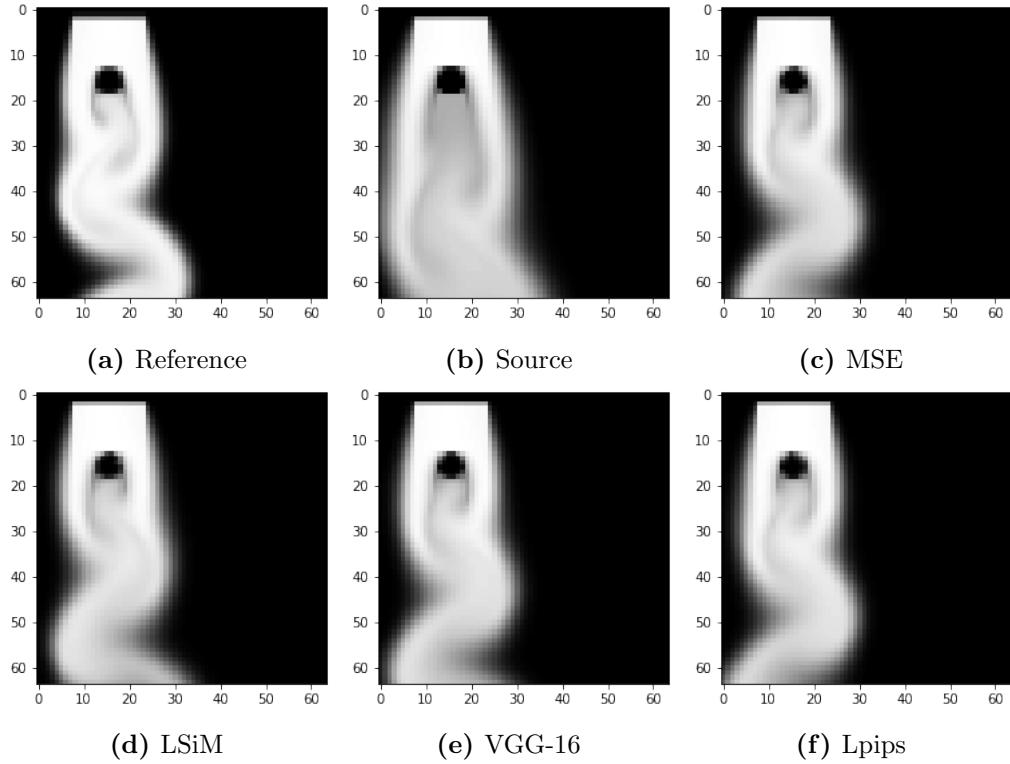


**Figure 5:** Karman flow corrected simulation trained with Lpips on the 128\*64 domain [gif here](#)

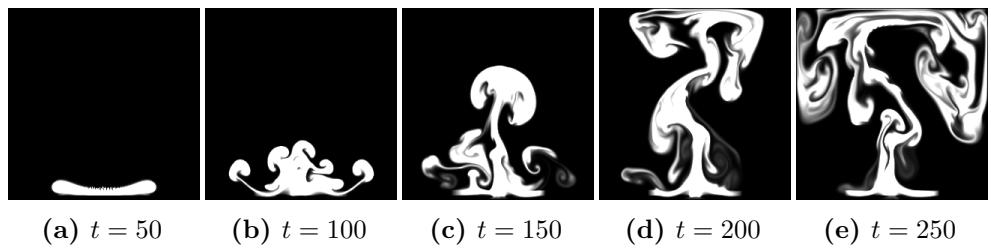


**Figure 6:** Karman flow corrected simulation trained with VGG-16 on the 128\*64 domain [gif here](#)

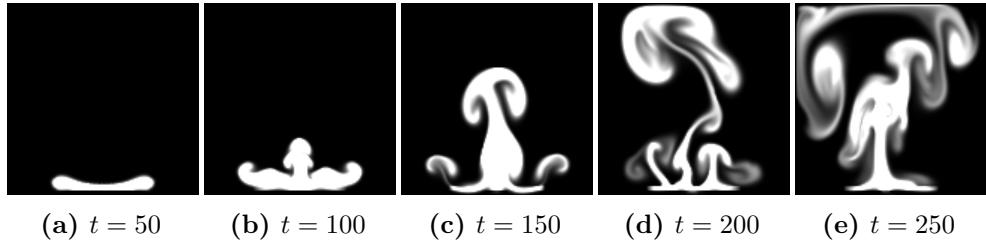
## 7.2. SUMMARY



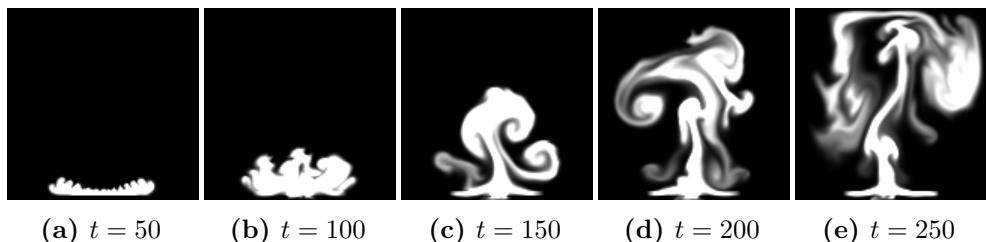
**Figure 7:** Karman flow corrected simulation trained with each loss on the 64\*64 domain comparing with the reference [all gifs here](#)



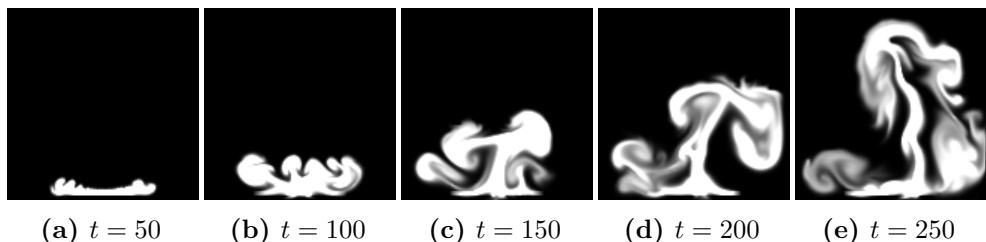
**Figure 8:** Reference simulation of buoyancy driven flow on the 128\*128 domain [gif here](#)



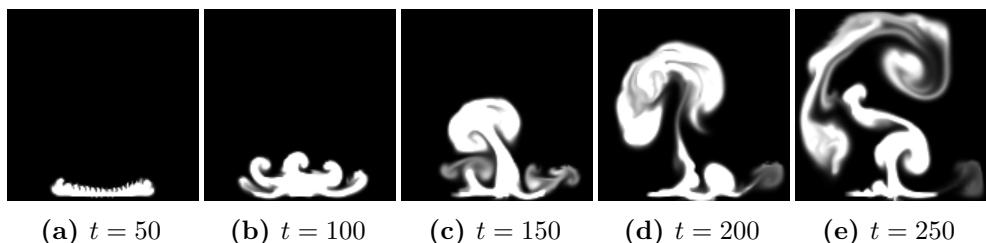
**Figure 9:** Source simulation of buoyancy driven flow on the 128\*128 domain [gif here](#)



**Figure 10:** Buoyancy driven flow corrected simulation trained with LSiM on the 128\*128 domain [gif here](#)

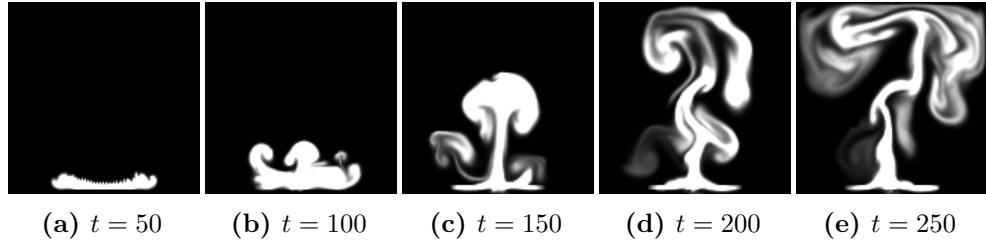


**Figure 11:** Buoyancy driven flow corrected simulation trained with MSE on the 128\*128 domain [gif here](#)

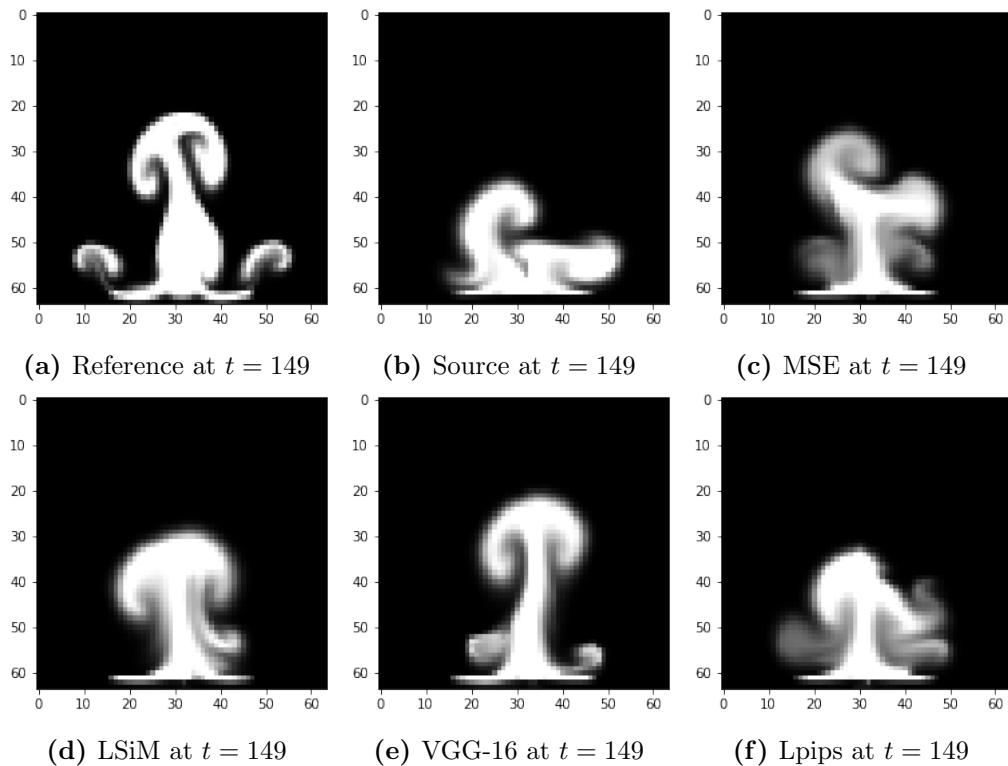


**Figure 12:** Buoyancy driven flow corrected simulation trained with VGG-16 on the 128\*128 domain [gif here](#)

## 7.2. SUMMARY



**Figure 13:** Buoyancy driven flow corrected simulation trained with Lpips on the 128\*128 domain [gif here](#)



**Figure 14:** Buoyancy driven flow corrected simulation trained with each metric on the 64\*64 domain comparing with the reference and the source [all gifs here](#)

# Bibliography

- [1] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.
- [2] Seyed Ali Amirshahi, Marius Pedersen, and Stella X. Yu. Image Quality Assessment by Comparing CNN Features between Images. *Journal of Imaging Sience and Technology*, 60(6), 2016. doi: 10.2352/J.ImagingSci.Tech.2016.60.6.060410.
- [3] Alexander Berardino, Johannes Balle, Valero Laparra, and Eero Simoncelli. Eigen-Distortions of Hierarchical Representations. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, volume 30, 2017. URL <http://arxiv.org/abs/1710.02266>.
- [4] Sebastian Bosse, Dominique Maniry, Klaus-Robert Mueller, Thomas Wiegand, and Wojciech Samek. Neural Network-Based Full-Reference Image Quality Assessment. In *2016 Picture Coding Symposium (PCS)*, 2016. doi: 10.1109/PCS.2016.7906376.
- [5] Le Kang, Peng Ye, Yi Li, and David Doermann. Convolutional Neural Networks for No-Reference Image Quality Assessment. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1733–1740, 2014. doi: 10.1109/CVPR.2014.224.
- [6] Jongyoo Kim and Sanghoon Lee. Deep Learning of Human Visual Sensitivity in Image Quality Assessment Framework. In *30TH IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, pages 1969–1977, 2017. doi: 10.1109/CVPR.2017.213.
- [7] Robert Bridson and Matthias Müller. Fluid simulation: Siggraph 2007 course notes video files associated with this course are available from the citation page. pages 1–81, 08 2007. doi: 10.1145/1281500.1281681.

---

## BIBLIOGRAPHY

- [8] Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189, 1965. doi: 10.1063/1.1761178. URL <https://aip.scitation.org/doi/abs/10.1063/1.1761178>.
- [9] Staggered grid explanation. [https://tum-pbs.github.io/PhiFlow/Staggered\\_Grids.html](https://tum-pbs.github.io/PhiFlow/Staggered_Grids.html).
- [10] Semi-lagrangian advection. <https://yangwc.com/2019/05/01/fluidSimulation/>.
- [11] Kiwon Um, Robert Brand, Yun Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. *Advances in Neural Information Processing Systems*, 2020.
- [12] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.
- [13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [14] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.
- [15] Georg Kohl, Kiwon Um, and Nils Thuerey. Learning similarity metrics for numerical simulations. In *International Conference on Machine Learning*, pages 5349–5360. PMLR, 2020.
- [16] James P Crutchfield and Bruce S McNamara. Equations of motion from a data series. *Complex systems*, 1(417-452):121, 1987.
- [17] Ioannis G Kevrekidis, C William Gear, James M Hyman, Panagiotis G Kevrekidid, Olof Runborg, Constantinos Theodoropoulos, et al. Equation-free, coarse-grained multiscale computation: Enabling mocroscopic simulators to perform system-level analysis. *Communications in Mathematical Sciences*, 1(4):715–762, 2003.

## BIBLIOGRAPHY

---

- [18] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.
- [19] J. Nathan Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017. doi: 10.1017/jfm.2016.803.
- [20] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.
- [21] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.
- [22] Byungsoo Kim, Vinicius C Azevedo, Markus Gross, and Barbara Solenthaler. Lagrangian neural style transfer for fluids. *ACM Transactions on Graphics (TOG)*, 39(4):52–1, 2020.
- [23] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.
- [24] Pascal Laube, Michael Grunwald, Matthias O. Franz, and Georg Umlauf. Image inpainting for high-resolution textures using CNN texture synthesis. *CoRR*, abs/1712.03111, 2017. URL <http://arxiv.org/abs/1712.03111>.