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

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 Dissertation untersucht die Integration von Wahrnehmungsverlusten in verschiedene Modelle, die sich auf die Fluidsimulation beziehen. Wahrnehmungsverluste werden verwendet, um Unterschiede auf hohem Niveau zu vergleichen, während herkömmliche Verlustfunktionen wie MSE und MAE nur die Pixelniveauunterschiede vergleichen, was roher ist.

Einige berühmte Wahrnehmungsverlustfunktionen wie der Vergleich der Zwischenschichten eines vortrainierten VGG-Netzwerks verbessern bereits seine Leistung bei einigen bildbezogenen Aufgaben.[1] Wir wollen zeigen, dass die Integration einiger vortrainierter Wahrnehmungsverluste im Kontext der Fluidsimulation übertreffen auch die traditionellen Verlustfunktionen.

Wir haben einige Fluidsimulationsaufgaben in SOL[7] getestet, um die Leistung der Percetual Loss Functions zu verbessern. Inzwischen integrieren wir solche Verlustfunktionen auch in Modelle wie Autoencoder und Superresolution, um deren Ergebnisse mit denen aus den gleichen Modellen zu vergleichen, die jedoch mit MSE-Verlustfunktionen trainiert wurden

Schließlich haben wir die Ergebnisse von SOL und Superauflösungsmodell kombiniert, um einige Fluidsimulationen zu geringeren Kosten durchzuführen, um den Vorteil der Integration von Wahrnehmungsverlusten in solchen Aufgaben zu zeigen. *Note: Insert the German translation of the English abstract here.*

Contents

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 and MAE. 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 Equation

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

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

$$\nabla \cdot \vec{u} = 0 \tag{1.2}$$

where ∇ denotes the gradient, $\nabla \cdot$ denotes the Divergence, $\vec{u}$ denotes the velocity of the fluid, t denotes the time step, ρ denotes the density of the fluid, p denotes the pressure, $\vec{g}$ denotes the gravity and ν denotes the viscosity of the fluid.

The equation (1.1) is actually a transformation of the $\vec{F} = m\vec{a}$ 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 Stucture

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 centeredgrid 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 centeredgrid. 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)

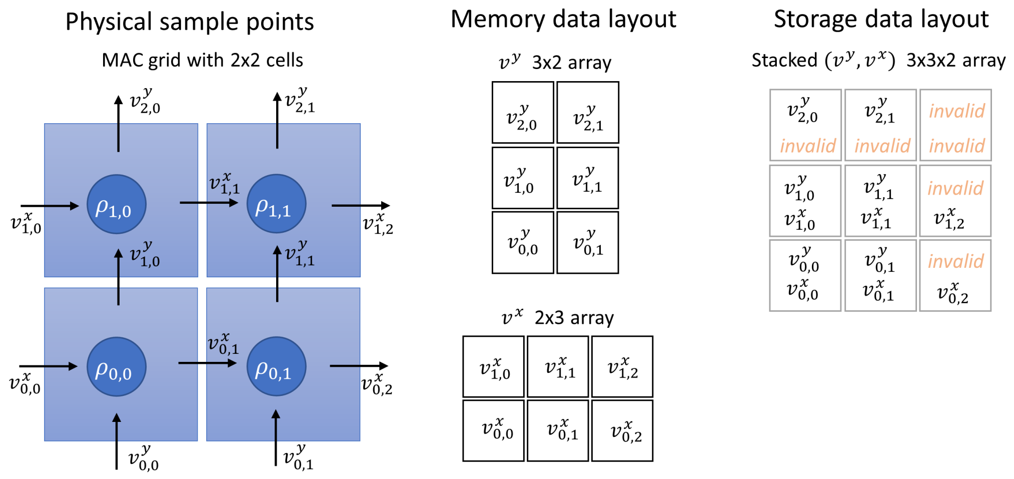

Physical sample points
MAC grid with 2x2 cells
Memory data layout
v^y 3x2 array
v^x 2x3 array
Storage data layout
Stacked (v^y, v^x) 3x3x2 array
$\rho_{1,0}$ $\rho_{1,1}$ $\rho_{0,0}$ $\rho_{0,1}$
$v^y_{2,0}$ $v^y_{2,1}$ $v^y_{1,0}$ $v^y_{1,1}$ $v^y_{0,0}$ $v^y_{0,1}$
$v^x_{1,0}$ $v^x_{1,1}$ $v^x_{1,2}$ $v^x_{0,0}$ $v^x_{0,1}$ $v^x_{0,2}$
invalid

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** $n \leftarrow 0, 1, 2, ...$ **do**
    $\vec{u}^A \leftarrow advect(\vec{u}^n, \vec{u}^n, \Delta t)$
    add the extra force $\vec{u}^B \leftarrow \vec{u}^A + \Delta t\vec{g}$
    $\vec{u}^{n+1} \leftarrow make\_incompressible(\Delta t, \vec{u}^B)$
**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+1. 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 $\vec{x_P}$

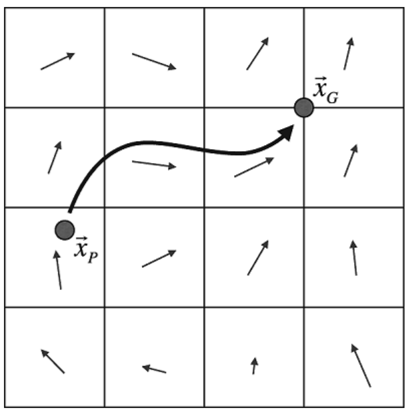


Figure 1.2: advection[10]

we first use the euler equation to calculate the $\vec{x_P}$ as a step before of $\vec{x_G}$

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

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

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

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 that we focus on in this thesis, and we explain how the deep learning techniques are integrated.

1.2.1 Autoencoder

The Autoencoder model has a bottle neck 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 super-resolute 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 mechanisim 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

The idea of jumping out of the traditional metrics based on L^p norm has firstly begun by [11], who developed the structure similarity index. Yet it was still not a perceptual loss and can not extract any deep features of the images. This situation has changed until the discovery of CNN. Because CNNs are able to extract a lot of deep features structures and patterns from the image datasets (for example in 2014 [12] has proposed the VGG structure which achieve a very good score on the ImageNet challenge), therefore more and more works tend to use perceptual loss to evaluate 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 [13] developed a new approach to compute the perceptual loss function based on the trained deep CNNs, which, shown in his title, achieved unreasonable effectiveness. In 2020 [14] proposed a new perceptual metric which focused on the numerical evaluation data.

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 [15, 16, 17]. More recently deep learning starting to play an important role in fluid dynamics [18]. In 2019, Bar-Sinai et al. successfully integrate deep learning method into advection-diffusion problems to infer stencils. With the development of Phiflow [20], 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, [7] 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 not like traditional loss functions. For instance MSE or MAE only computes 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 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 tries to retrieve the high level differences between the images, hence will usually have a better training performance than the traditional loss functions. In a word, perceptual losses can make the model achieve better training performance in specific tasks comparing to the traditional loss functions.

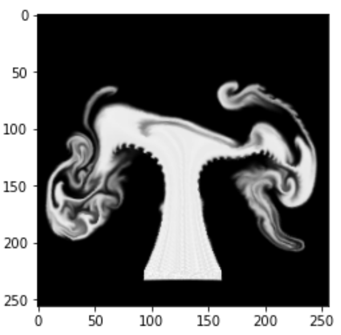
In this thesis we mainly focus on comparing the performance between traditional loss function MSE and perceptual loss functions LSIM, VGG-16 and Lpips. In this chapter we will cover these loss functions that we adopted in this paper and briefly explain their characteristics and their drawbacks. More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

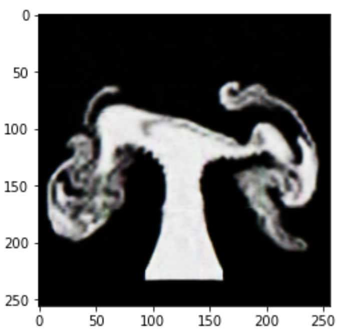
3.1 Mean Square Error

MSE is one of the most widely used loss functions.

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

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


(a) ref


(b) MSE output

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 [14] 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.

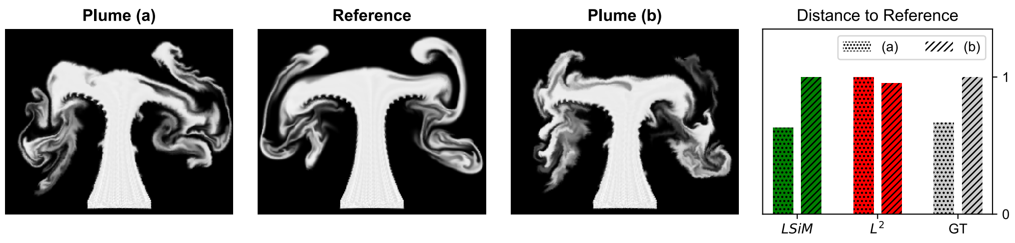


Figure 3.2: The loss computed between two different plume images and the reference with LSiM and L^2 metrics. While the GT is the ground truth distance determined by the distance of the initial parameters in the data generation.

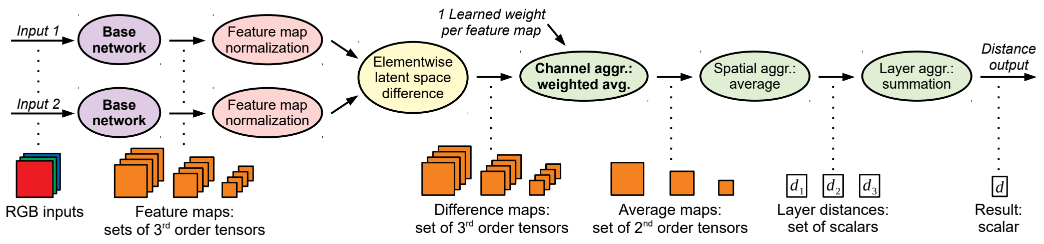

Input 1
Input 2
Base network
Base network
Feature map normalization
Feature map normalization
1 Learned weight per feature map
Elementwise latent space difference
Channel aggr.: weighted avg.
Spatial aggr.: average
Layer aggr.: summation
Distance output
RGB inputs
Feature maps: sets of 3rd order tensors
Difference maps: set of 3rd order tensors
Average maps: set of 2nd order tensors
d_1 d_2 d_3
Layer distances: set of scalars
d
Result: scalar

Figure 3.3: The Network structure of LSiM

Notice that the so called ground truth in this case is determined by 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 $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{I}$:

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

$$m(\boldsymbol{x},\boldsymbol{y}) = m(\boldsymbol{y},\boldsymbol{x}) \qquad \text{symmetry} \qquad (3.3)$$

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

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

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.

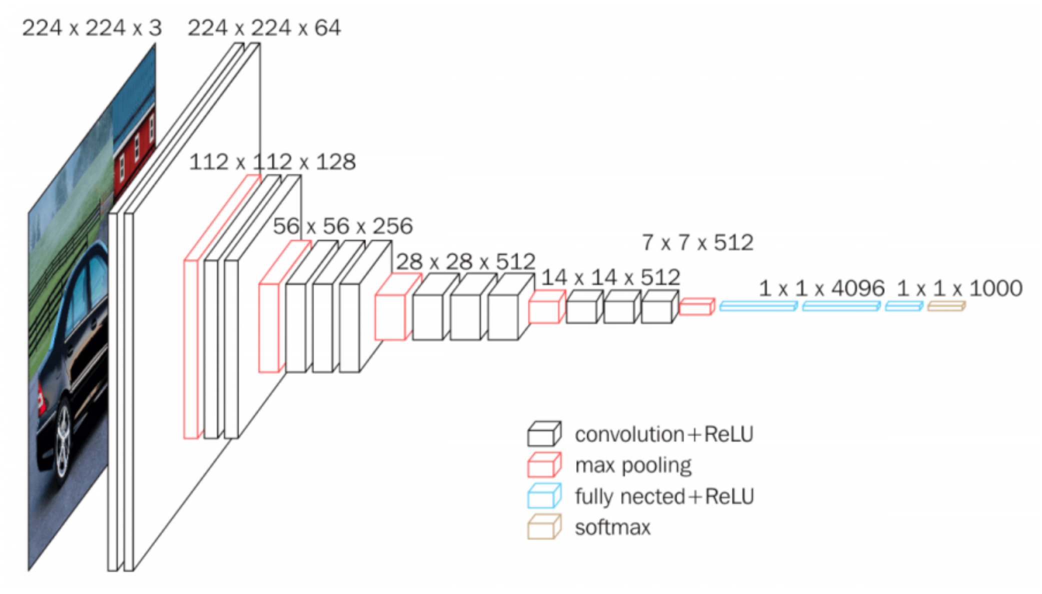


Figure 3.4: The network structure of VGG-16

3.3 VGG-16 loss network

VGG-16[12] 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 challenge based on a dataset of over 14 million images belonging to 1000 classes. Although the networks 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.

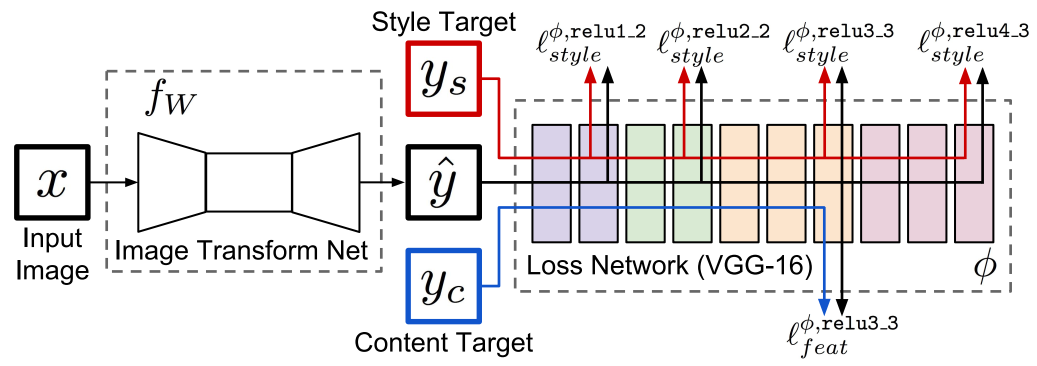

Style Target
y_s
$\ell_{style}^{\phi,\text{relu1\_2}}$
$\ell_{style}^{\phi,\text{relu2\_2}}$
$\ell_{style}^{\phi,\text{relu3\_3}}$
$\ell_{style}^{\phi,\text{relu4\_3}}$
f_W
x
Input Image
Image Transform Net
$\hat{y}$
y_c
Loss Network (VGG-16)
ϕ
Content Target
$\ell_{feat}^{\phi,\text{relu3\_3}}$

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

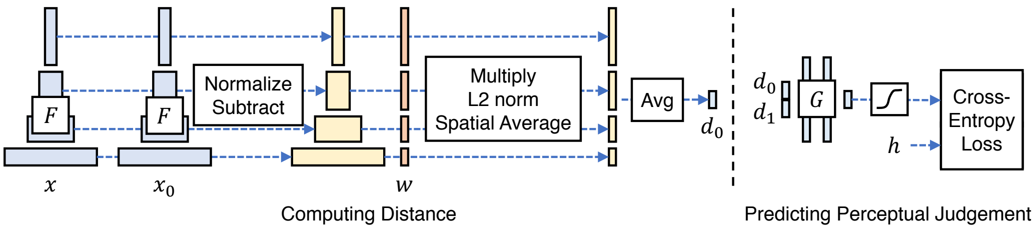


Figure 3.6: The Lpips training setup: The distance is computed between the ground truth patch x and the distorted patch x_0 or x_1 based on the chosen network architecture. The perceptual distance is then trained on a small network G. The ground truth of the model outputs is the Psychophysical Similarity Measurement h , and the training is based on the Cross Entropy Loss[13]

3.4 Lpips

Lpips as in Learned Perceptual Image Patch Similarity[13] 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, superresolution, frame interpolation and so on for the dataset. Based on that, they use several psychophysical similarity measurements on the dataset, trying to approach the human judgements to the images. For the network architecture, they evaluate the SqueezeNet, AlexNet, and VGG architectures. Comparing with the vgg-16 loss function, we can consider Lpips as a advanced version of such method, 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 comparison then 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

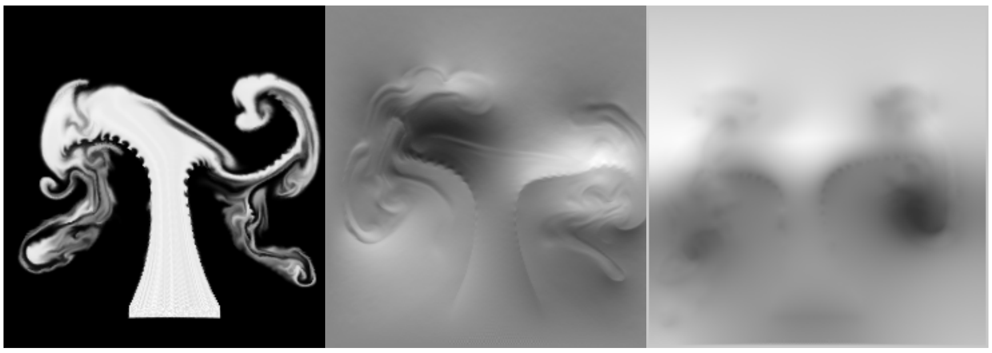
We set up the Autoencoder mainly to have a rough overview of the performance of the perceptual loss functions. Hence, we choose to use the nuerical simulation data with the original size of 256*256, on which the LSiM model was trained.

Autoencoder can be trained without the 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 RBG channels. Such techniques can also be applied on any models with grayscale data, you can see more details about it in the Section 5.2.2.
- 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 superresolution task came from various sources. We first used the training data of LSiM model, because these are all numerical



(a) density data **(b)** velocity data **(c)** pressure data

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.

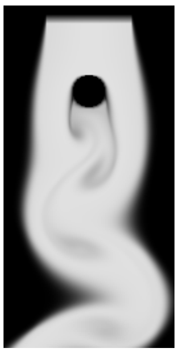
We also want to test the performance of combining Solver in the loop and superresolution together to compete against the time cost with performing the simulation purely on the higher domain. Therefore we also feed the training data of the Solver in the Loop to the superresolution model, so that our model can learn to super sampling the plume density images when we feed the corrected simulation images on the lower domain. In this way, we are able to abtain 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 downsample 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

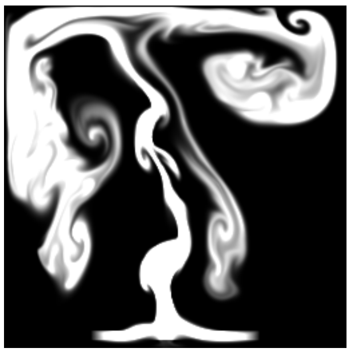
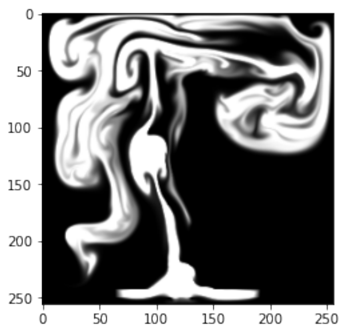
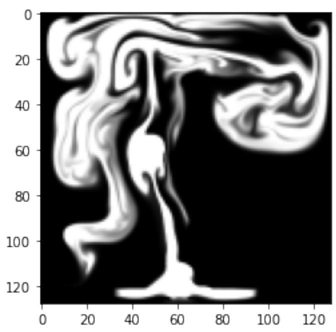

(b) Buoyancy driven flow example

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*32, 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.

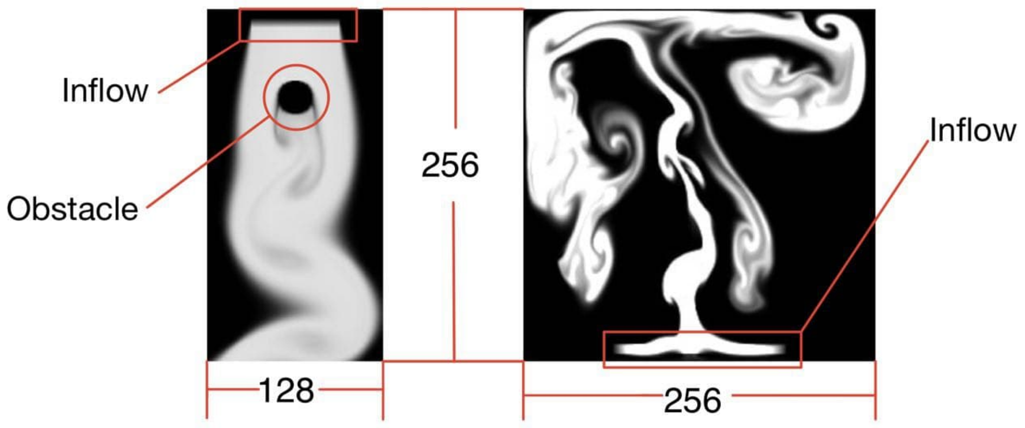


Figure 4.4: Domain set up of karman flow and buoyancy driven flow

4.3.2 Phiflow setup

The tool we used to perform the fluid simulation is Phiflow[20].

For the karmanflow generation, we define an open domain with the size of 200(height) * 100(width). **(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 percise we should sample our domain. Hence we could perform different resolution formats on the same domain size.)** 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 Δt 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 each simulation. Why we only chose 250 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.

Chapter 5

Tasks and Experiments Setup

5.1 Autoencoder

5.1.1 Network

We used a very simple bottle-neck like and fully convolutional neural network for the Autoencoder(see figure 5.1). Due to the grey scale images, the input and output channel size is set to 1. We apply two convlution layers followed by one max pooling layer and two upsample convolution layer(i.e. the ConvTranspose2d layer). In order to keep the tensor value within a reasonable range, we add before the final output a 256*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 don not add a ReLU activation function before the sigmoid is because the ReLU only outputs positive values which wil results in that the sigmoid function will then only output values greater then 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 modification to make our training eaiser. Since most of our data are smoke images or velocity fields, so they are in grey scale. In this case, we can modify our network in the way that the new input channel size and the output channel size are 1, because the grey scale images only have one channel. With this modification, our model does not need to learn to align all the three RBG channels to the same value, hence it will strongly reduce the training difficulty. However, in the process of applying

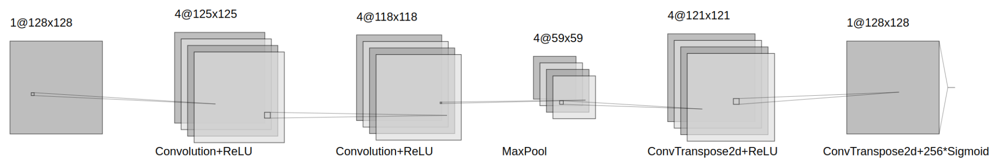


Figure 5.1: The structure of Autoencoder

the loss function, there will be the problem that many of the perceptual loss functions only accept RBG images that the input tensors should have three channels. In order to fix this, we choose to replicate the channel size to three with the same values, so the loss functions will regard our grey scale images as RBG 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 functions, 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 Superresolution

5.2.1 Network

We adopt the same ImageTranformNetwork proposed by Johnson et al., which is a fully convolutional network that has the same output tensor as the input. The network consists of three Convlayers, three UpsampleConvlayers and five residual blocks.(Notice that we do not use the deconvolutional layer in the UpsampleConvlayers as suggested in the paper[22], that deconvolutional layers will likely result into checkerboards artifacts due to the overlapping when doing the reverse convolution operations. So we prefer to fistly do the traditional upsampling either with Nearest-neighbor interpolation or with Bilinear interpolation then we apply the convlayer afterwards to obtain the expected upsampled size.)

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

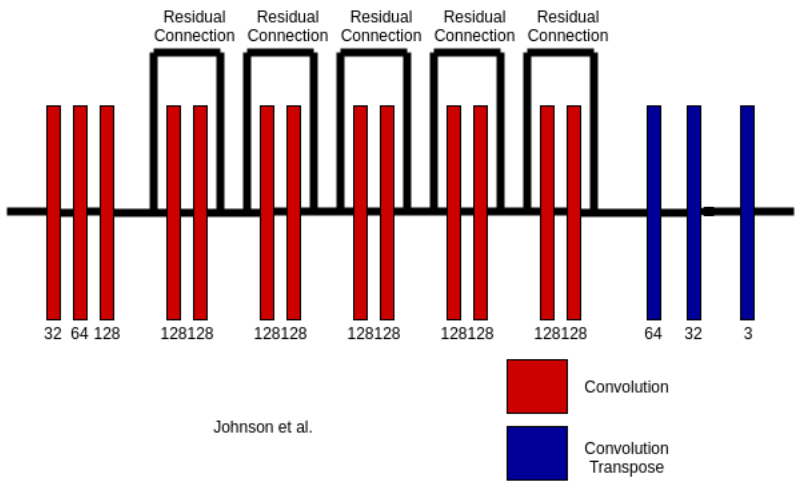
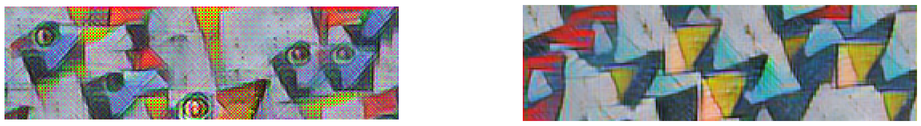


Figure 5.2: The structure of ImageTranformationNetwork of [1]



(a) Using deconvolution. Heavy checkerboard artifacts. **(b)** Using resize-convolution. No checkerboard artifacts.

Figure 5.3: The comparison between two upsample methods[22]

values lie in the range of [0, 255], so we apply at the end of the last upsampling layer a sigmoid function and times 256. In such way, we can ensure that our output tensor value won't exceed the range. Since the input tensor and the output tensor 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 Applying 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 downresoluted 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 have generated 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 function.

5.3.1 Network

The training network we used is called MarsMoon, which is the same network that [7] 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, which is very unstable. Thus, adding the trained correction to the physical

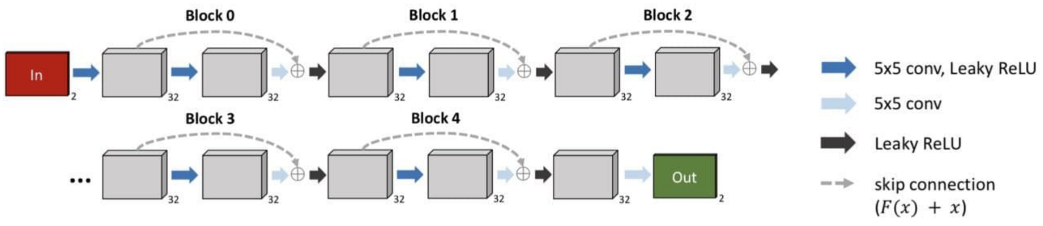


Figure 5.4: The network architecture[7]

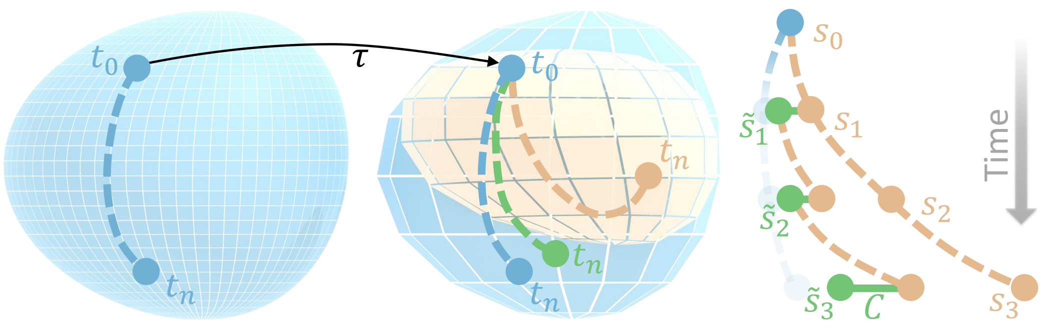


Figure 5.5: The training goal is to obtain the suitable correction, with which the simulation on the lower domain can reduce the numerical error. The brown one represents the source simulation of the lower domain without any corrections, the blue one represents the simulation on the higher domain which is our target, the green one is the brown one plus the trained correction, which aims to approach the blue one[7]

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, as in the first several steps of karman flow do not have much difference between high and low domains. So we only try to train the last 1000 steps of the total 1500 steps. When we train the lower domain 64*32 karman flow, we also find out that the 64*32 image for LSiM model is too small, so we decide to alter the domain to the resolution of 64*64, correspondingly the training set also need to be retrained on the 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)

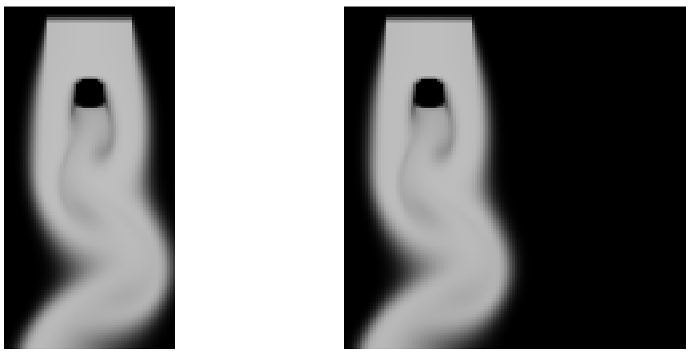


Figure 5.6: Padding plume image to match the input size of LSiM model

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

For the MSE loss, which is exactly used by Um et al. in his project, 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 RBG 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:

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

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 RBG image) and compare them with their ground truth.

Chapter 6

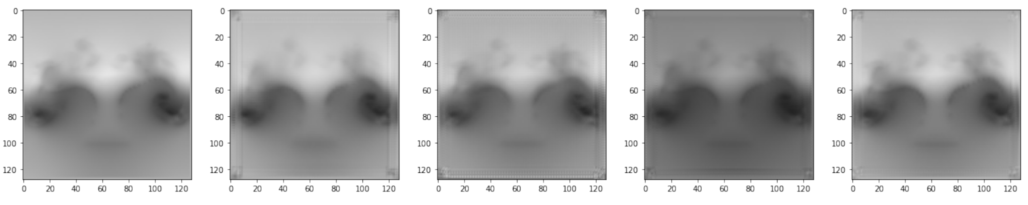
Results and Analysis

6.1 Comparison

6.1.1 Autoencoder

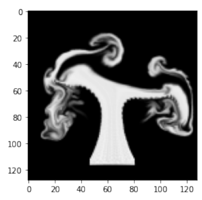
We train the autoencoder model on density, pressure and velocity data of smokes, here are some examples of the training outputs. The reference image as well as the training input is on the left side while the rest are different model outputs trained by their respective losses. We can see that for every output, there are some blur area on the sides. 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 each

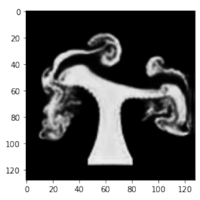


(a) *Reference* **(b)** *MSE* **(c)** *LSiM* **(d)** *Lpips* **(e)** $VGG-16$

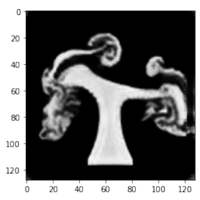
Figure 6.1: Pressure images



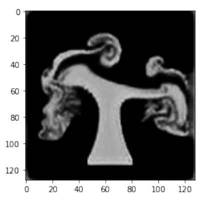
(a) $Reference$



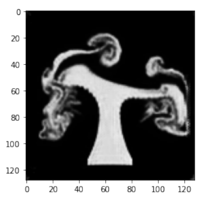
(b) MSE



(c) $LSiM$

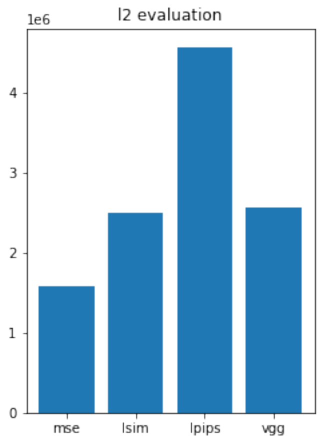


(d) $Lpips$

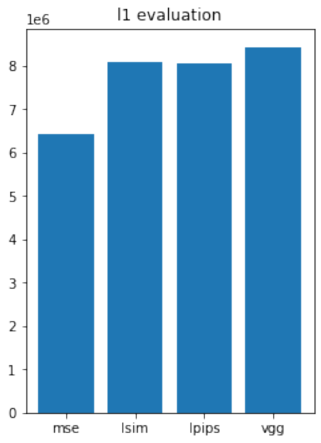


(e) $VGG - 16$

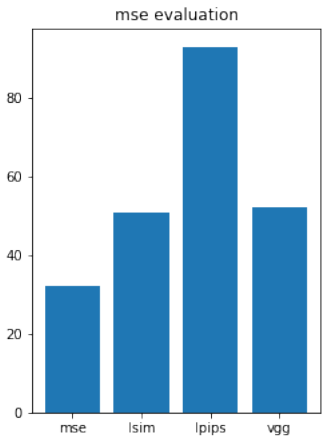
Figure 6.2: Density images



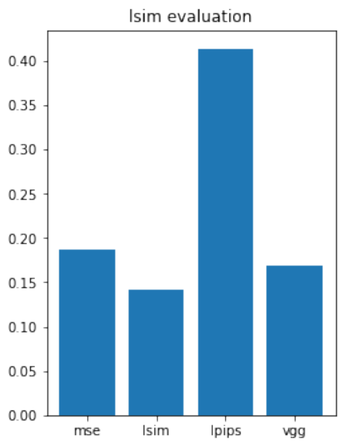
(a) L2 evaluation



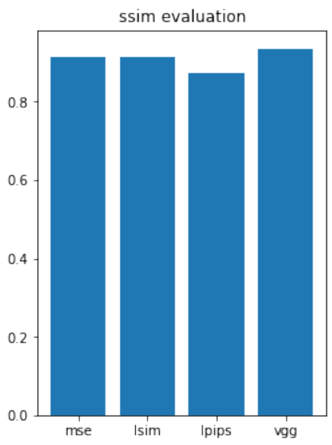
(b) L1 evaluation



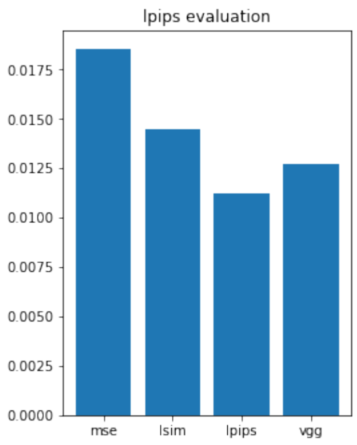
(c) MSE evaluation



(d) LSiM evaluation



(e) SSIM evaluation



(f) Lpips evaluation

Figure 6.3: Numerical evaluation of each metrics on testset of autoencoder

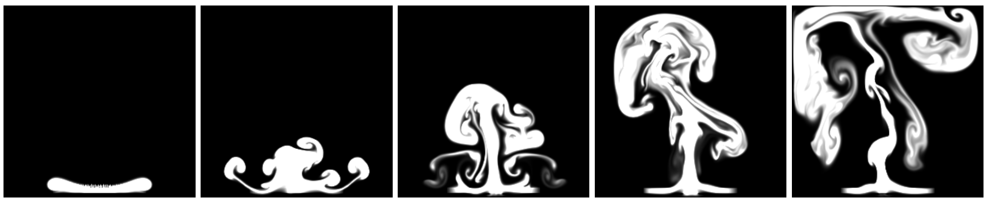

(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.4: Reference simulation

metric over 1056 test smoke images. The evaluations agree to our intuition of the images. MSE and LSiM achieve the best score in the evaluation using their selves, 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

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 evaluation, we compute the distance of 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.

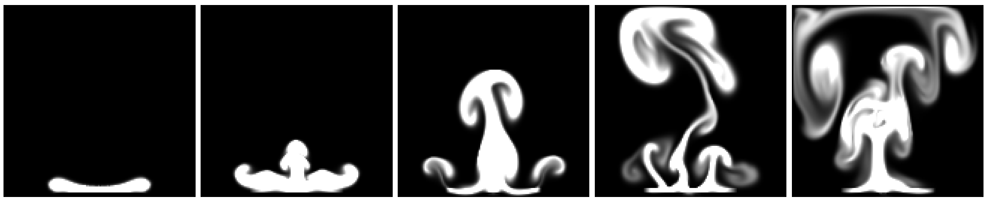
- Karman flow
- Buoyancy driven flow

We first pick a dataset from our training set as the test set with renolds number of 2e5(note the renolds numbers of out training set range from 1e5 to 8e5)

However when we generate a test set with the renolds number of 2.5e5 which was not included in the training set, the results are then different.

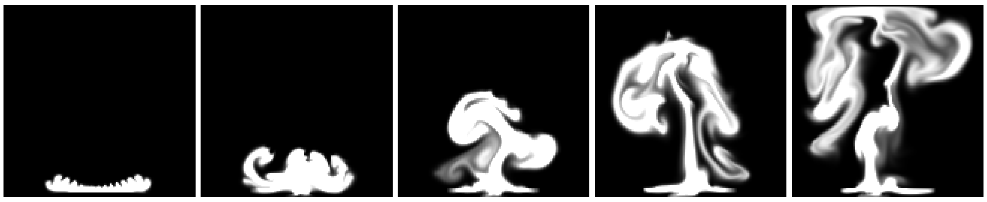
6.1.4 Time cost evaluation

6.2 Limitaions



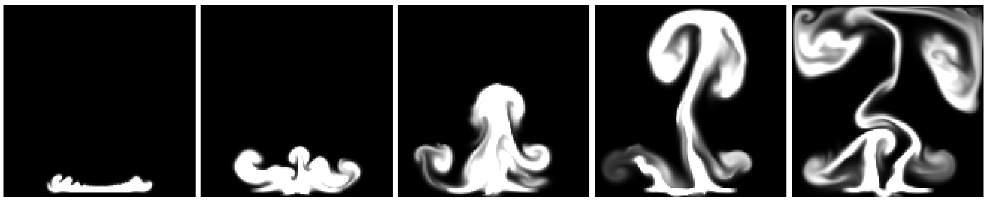
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.5: Source simulation



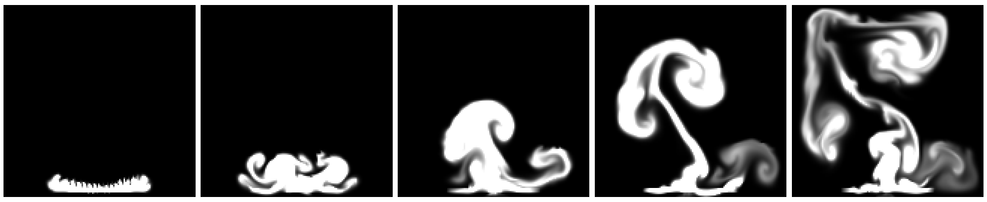
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.6: simulation with lsim



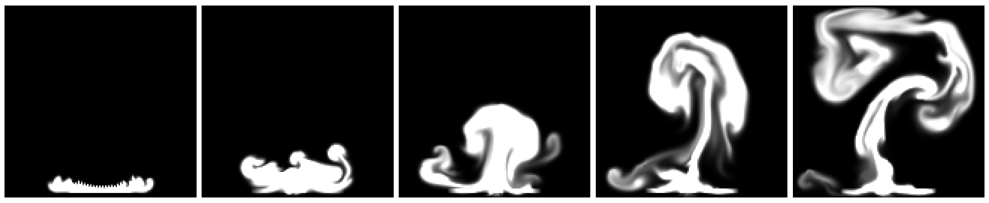
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.7: simulation with mse



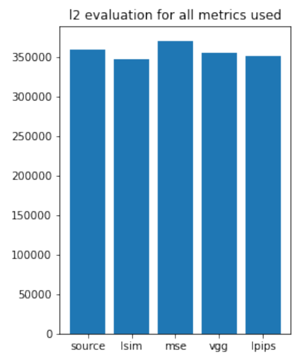
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.8: simulation with vgg

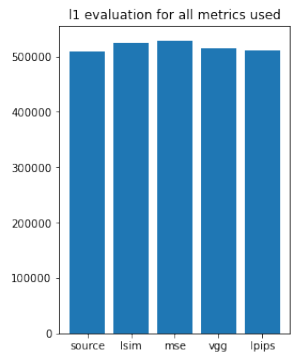


(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

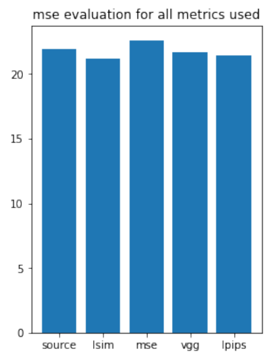
Figure 6.9: simulation with lpips



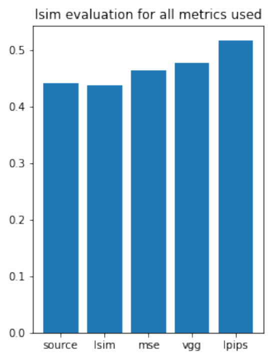
(a) l2 evaluation



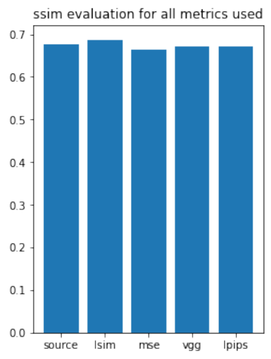
(b) l1 evaluation



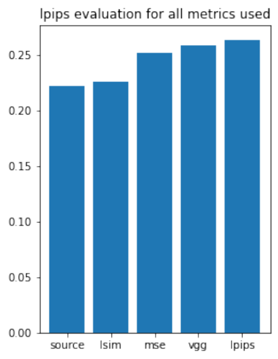
(c) mse evaluation



(d) lsim evaluation

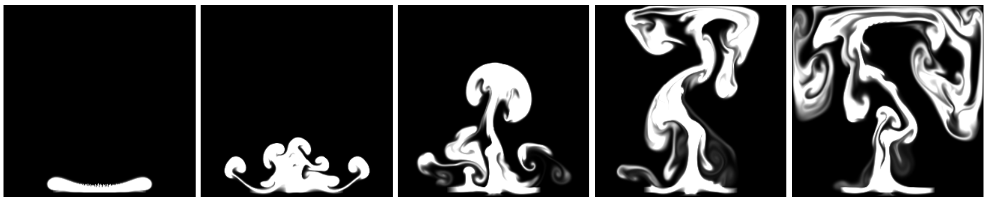


(e) ssim evaluation



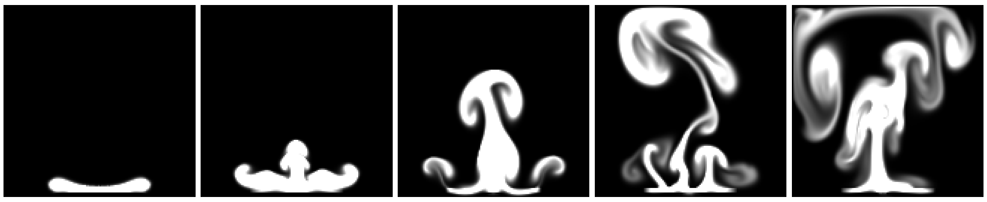
(f) lpips evaluation

Figure 6.10: Numerical evaluation of each metrics on training set



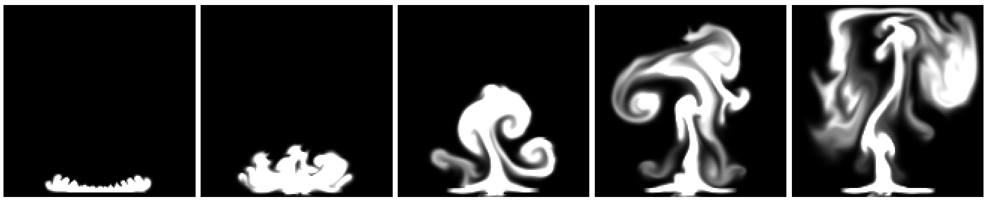
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.11: Reference simulation



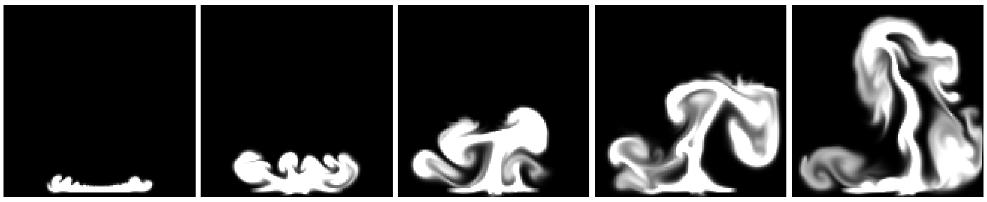
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.12: Source simulation



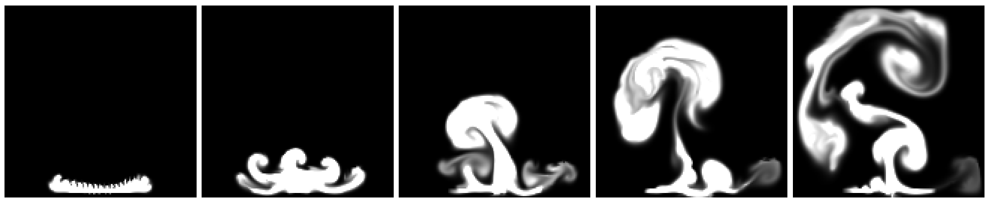
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.13: simulation with lsim



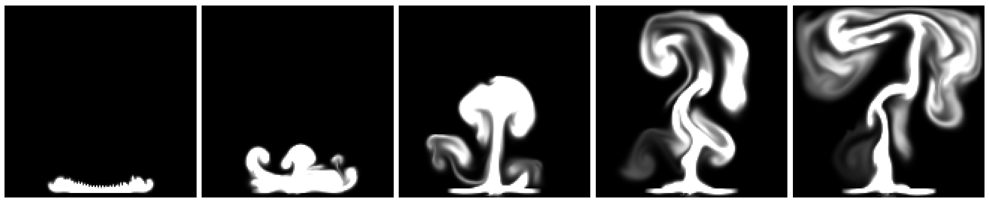
(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.14: simulation with mse



(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.15: simulation with vgg



(a) $t = 50$ **(b)** $t = 100$ **(c)** $t = 150$ **(d)** $t = 200$ **(e)** $t = 250$

Figure 6.16: simulation with lpips

Chapter 7

Conclusion

7.1 Future work

7.2 Summary

Appendix A

e.g. Questionnaire

Note: If you have large models, additional evaluation data like questionnaires or non summarized results, put them into the appendix.

List of Tables

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.Technol.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] 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.

[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`.

[10] Semi-lagrangian advection. `https://yangwc.com/2019/05/01/fluidSimulation/`.

[11] 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.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] 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.

[14] Georg Kohl, Kiwon Um, and Nils Thuerey. Learning similarity metrics for numerical simulations. In *International Conference on Machine Learning*, pages 5349–5360. PMLR, 2020.

[15] James P Crutchfield and Bruce S McNamara. Equations of motion from a data series. *Complex systems*, 1(417-452):121, 1987.

[16] 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.

[17] 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.

[18] J. Nathan Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017. doi: 10.1017/jfm.2016.803.

[19] 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.

[20] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

[21] 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.

[22] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.