University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer science

Optimizing Omnimotion: Enhancing Efficiency and Speed in Dense Full-Pixel Tracking

Supervisor
Prof. Lamberto Ballan
University of Padova

Co-supervisor
Prof. Paolo Favaro
Unversity of Bern

Master Candidate
Sanjar Tlepin

Student ID
2041606

Academic Year
2024-2025

"Just like relationships, real-world challenges in technology aren't as seamless as they appear in movies or on TV. There's no simple 'will it work, won't it?' that ends in perfect resolution. Many ideas and systems fail because they weren't designed with the right foundation, and even successful ones face setbacks. But through it all, I've never become discouraged. I still believe in the potential of innovation—whether it's in refining algorithms or creating something new. The key to success is perseverance. True progress happens when someone steps up to face the challenges head-on and fights for the vision, believing it can work. And if the idea is right, and with some luck, it will succeed."

— Dr. Percival Ulysses Cox, M.D.

Abstract

Motion estimation in computer vision (CV) presents significant challenges, particularly in tracking points across occlusions. The Omnimotion model, introduced in the paper "Tracking Everything Everywhere All at Once," addresses this issue by mapping video points from local frames into a global 3D representation, allowing for accurate tracking. This model leverages Invertible Neural Networks (INNs) and Neural Radiance Fields (NeRF) to maintain visibility across frames, but its computationally intensive structure and reliance on RAFT for pre-processing demand significant computational resources and lengthy training times.

This study proposes several optimizations to reduce these computational dependency. By eliminating non-essential loss functions and introducing an embedding mechanism for internal MLP layers, training complexity and time were significantly reduced. Additionally, the implementation of Tiny-CUDA and weight freezing techniques further enhanced performance, cutting memory usage by 50% and improving training speed by nearly twofold. Although a slight decrease in model quality was observed, these optimizations expand the potential use of Omnimotion for larger datasets and real-time applications.

Contents

Abstract v

List of figures ix

List of tables xi

Listing of acronyms xiii

1 Introduction 1

2 Background 3
- 2.1 Deep Learning . . . 3
  - 2.1.1 Perceptron . . . 3
  - 2.1.2 Simple Neural Network . . . 4
  - 2.1.3 Deep Neural Network . . . 5
  - 2.1.4 Loss Functions . . . 6
  - 2.1.5 Gradient Descent . . . 6
  - 2.1.6 Convolutional Neural Networks . . . 7
  - 2.1.7 Training Procedures . . . 10
- 2.2 Computer vision . . . 12
  - 2.2.1 Optical flow . . . 12
  - 2.2.2 Lucas-Kanade Method . . . 14
  - 2.2.3 Recurrent All-Pairs Field transforms . . . 15

3 The omnimotion architecture 19
- 3.1 Pre-processing . . . 19
  - 3.1.1 Self-distillation with no labels . . . 21
- 3.2 Dense Pixel Tracking . . . 23
  - 3.2.1 The Neural Radiance Fields . . . 24
  - 3.2.2 3D mapping . . . 25
  - 3.2.3 Frame-to-frame motion . . . 28
- 3.3 Optimization Techniques . . . 29
  - 3.3.1 Loss Function Optimization . . . 29
  - 3.3.2 Hard-Mining Strategy . . . 31
  - 3.3.3 Implementation Details . . . 31

4 Problem Statement and Methodology 33
- 4.1 Training Challenges . . . 34
  - 4.1.1 Loss Minimization . . . 34
- 4.2 Pre-processing enhancement . . . 34
  - 4.2.1 Substitution to the optical flow measurement method . . . 34
  - 4.2.2 Update of the appearance check techniques . . . 35

4.2.3 Hard-Mining Approach 36
4.3 Feature Embedding 36
4.4 Model Freezing Techniques 38
4.5 Tiny CUDA Neural Networks 39

5 Model Development 43

6 Results 45
6.1 Dataset Description 45
6.2 Metrics 46
6.3 Evaluation 46

7 Discussions 49
7.0.1 Other solution 52

References 55

Acknowledgments 59

Listing of figures

2.1 A simple mathematical model for a neuron. [1] 4
2.2 Fully Connected Layer. 5
2.3 LeNet-5 architecture [2]. 8
2.4 Convolution operation on a 5x5 input with a 3x3 kernel. 8
2.5 Max Pooling with 2x2 filters and stride 2. 9
2.6 Optical flow example overview. 12
2.7 Example of the neighborhood for Lucas-Kanade method [3]. 14
2.8 RAFT method overview. [4] 15
2.9 Building correlation volumes [4] 17

3.1 (a) Cycle consistency and (b) Chaining. 20
3.2 Erroneous correspondences check. [5] 20
3.3 DINO example. [6] 22
3.4 DINO in the case of one single pair of views (x1, x2). [7] 23
3.5 Omnimotion method overview. [5] 23
3.6 Neural Radiance Fields method overview. [8] 24
3.7 Invertible Neural Network. [9] 26
3.8 Network architecture for the mapping network $M\theta$. [5] 27
3.9 Positional encoding $M\theta$. 27
3.10 Distortion loss. [10] 31

4.1 Quantitative comparison of the CoTracker with PIPs++ and TAPIR [11] 35
4.2 CoTracker architecture. [12] 35
4.3 Quantitative comparison of the DINOv2 with DINO [13] 36
4.4 Example of usage of positional encoding [8] 37
4.5 Coordinates representation: normal NeRF, Voxels, Tri-plane [14] 38
4.6 Multiresolution hash encoding in 2D. [15] 39
4.7 "Fully fused" multi-layer perceptron. [15] 40

5.1 Our model overview. 44

7.1 Color comparison. 50
7.2 Comparison of RAFT-based and CoTracker-based Omnimotion. 51
7.3 CaDeX++ Method Overview. [16] 52
7.4 Architecture of CaDeX++. [16] 53

Listing of tables

6.1 Performance comparison between ”Omnimotion” and the models. 46
6.2 Time&Memory usage of the Omnimotion, No-Smooth and Encoded model. 47
6.3 Performance comparison between ”Omnimotion” and ”Encoded” models. 47
6.4 Performance comparison between ”Omnimotion” and ”Freeze” models. 47
6.5 Performance comparison of models across videos: ”Omnimotion”, ”No smooth”, ”Encoding”, ”Freeze” and ”TCNN”. 48

7.1 Performance comparison of models across videos: ”Omnimotion” with RAFT and ”TCNN” with CoTracker on 40, 000 epochs. 52

Listing of acronyms

CNN ..... Convolutional Neural Networks
NLP ..... Natural Language Processing
DL ..... Deep Learning
ML ..... Machine Learning
ViT ..... Vision Transformers
SG ..... Stop-Gradient
EMA ..... Exponential Moving Average
NeRF ..... The Neural Radiance Fields
DINO ..... Self-Distillation with no Labels
MLP ..... Multilayer Percepton
INN ..... Invertible Neural Networks
Real-NVP ..... Real-valued Non-volume Preserving
MAE ..... Mean Absolute Error
MSE ..... Mean Squared Error
GPU ..... Graphics Processing Unit
RAFT ..... Recurrent All-Pairs Field Transforms
ReLU ..... Rectified Linear Unit
RGB ..... Red Green Blue
TCNN ..... Tiny CUDA Neural Networks
CS ..... Computer Science
NRC ..... Neural Radiance Caching
AI ..... Artificial Intelligence
OF ..... Optical flow
ADAM ..... Adaptive Moment Estimation
RMSP ..... Root Mean Square Propagation
RMSprop ..... Root mean square prop
DAVIS ..... Densely Annotated Video Segmentation
TAPIR ..... Tracking Any Point with per-frame Initialization and temporal Refinement
PIPs ..... Persistent Independent Particles
NRC ..... Neural Radiance Caching

1 Introduction

In the field of computer vision, the accurate estimation of motion from video data has seen remarkable advancements over recent years. This progress is particularly evident in applications such as autonomous driving, robotics, and augmented reality, where precise motion tracking of objects is essential for system functionality. A persistent challenge, however, arises when objects in the scene occlude one another, temporarily obstructing their visibility. The ability to maintain accurate tracking during such occlusions remains a significant hurdle, despite the development of numerous sophisticated methodologies.

One of the most promising contributions to overcoming this challenge is the Omnimotion model. Omnimotion's approach to motion estimation and occlusion handling involves mapping all observed points in a video sequence into a unified 3D representation. This technique ensures that the visibility status of each point is maintained over time, even across occlusions. The model's use of Invertible Neural Networks and Neural Radiance Fields allows for the precise mapping of points between multiple local and single global volumes, enabling reliable motion tracking based on color and pixel density. However, despite its strengths, Omnimotion is limited by its considerable computational demands.

The complexity of Omnimotion is twofold: not only does the model itself require extensive training, but its reliance on the RAFT model for pre-processing further complicates the pipeline. RAFT introduces additional training overhead, and any inaccuracies at this stage can propagate through the system. Moreover, the high computational cost associated with training Omnimotion limits its scalability, making it impractical for use on larger datasets or in real-time applications. This computational burden, alongside the inability to reuse pre-trained models for different video sequences, presents a significant barrier to the widespread adoption of the model.

This thesis addresses these limitations by proposing several optimization strategies aimed at reducing the computational load of Omnimotion without sacrificing accuracy. Possible modifications to the pre-processing by swapping RAFT with more efficient and accurate solutions were discussed and tested. Through a careful analysis of the model's architecture, specific loss functions with minimal impact on final outcomes were identified and removed, leading to reductions in training time. A new holding mechanism named "freezing" were proposed

which improved the training speed. Furthermore, a novel embedding mechanism for 3D coordinates was introduced, which simplified the structure of the model's MLPs and resulted in faster processing. Additionally, the integration of Tiny-CUDA, a modern method for optimizing GPU computations, substantially decreased both memory usage and training duration.

The optimizations presented in this work offer a pathway to more efficient motion estimation, making models like Omnimotion more viable for real-time applications and large-scale deployments. These improvements not only contribute to advancing the state of motion tracking technology but also provide practical solutions for reducing the computational requirements of complex CV models.

2 Background

2.1 Deep Learning

Deep Learning (DL), a specialized branch of Machine Learning, has become a groundbreaking area that was inspired by the biological neural system of living things and the human brain in particular. Fundamentally, Machine Learning evolves around crafting algorithms that can make the recognition of patterns and have the ability to make informed decisions grounded in data. Deep learning constitutes a substantial progression in this domain, utilizing neural networks [17] composed of multiple layers (deep neural networks) to autonomously extract detailed features and complex representations from unprocessed data (e.g., identifying patterns in images). These aforementioned technologies and methodologies are positioned within the field of supervised learning. In this type of learning, the constructed architecture is defined by working with an features-target pair, x and $\hat{y}$, respectively. Subsequently, after the training phase, it is expected that the architecture will be capable of delivering accurate predictions when presented with new, previously unseen data.

2.1.1 Perceptron

The **perceptron** is a fundamental type of artificial neural network, inspired by the structure and function of biological neurons. The artificial neuron serves as the fundamental unit in the learning process, accepting inputs (denoted as x_n), applying corresponding weights (denoted as w_n), and processing them through a nonlinear activation function (denoted as $g(\cdot)$) to generate an output $\mathbf{X}$.

As illustrated in Figure 2.1, the operation of the artificial neuron can be broken down into essential steps, which together constitute the forward propagation of information. These key steps are the computation of weighted sum of its inputs and the application of a nonlinear activation function. Additionally, we add a bias (parameter

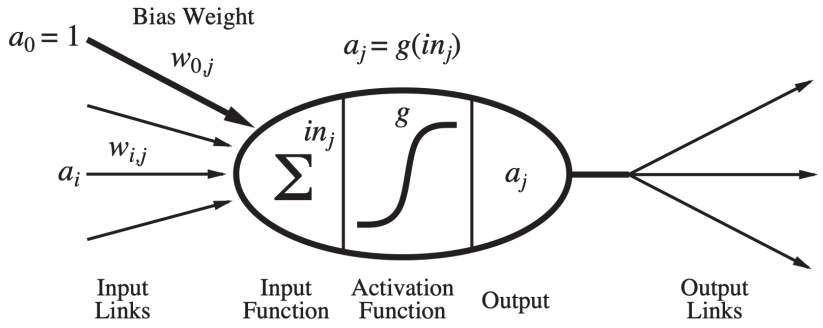


Figure 2.1: A simple mathematical model for a neuron. [1]

denoted as b) to the weighted sum of inputs before applying the activation function, allowing the model to better fit the data by shifting the activation function. Final neuron state mathematically can be explained as follows:

$$z = g(\sum_{i=1}^{n} w_i \cdot x_i) + b \tag{2.1}$$

The activation function determines the threshold that decides whether the perceptron should be activated. The use of a non-linear activation function is of great importance, as it introduces non-linearity into the learning process, thereby enabling neural networks to approximate complex functions. Some of the most commonly used activation functions in perceptrons are the step function, the Sigmoid function [18], and the Rectified Linear Unit [19].

2.1.2 Simple Neural Network

When multiple perceptrons are arranged together and connected to each other, they form a network through which input data flows. This process, known as **forward propagation**, follows the previously described steps. Normally this stack divided into 3 main sections: input, hidden layer and output. The addition of a hidden layer between the input and output functions can be viewed as a "transformation layer." This layer facilitates the transformation of the input from its original form into a novel, potentially more intricate, and abstract representation. The introduction of such a complex layer enables the system to learn and comprehend more complex data, thereby enabling the final function to perform with greater accuracy.

In Figure 2.2 demonstrated how this process is following, where $W^{(1)}$ and $W^{(2)}$ can explained as matrices of weights. Now the Equation 2.1 can be modified as follows:

$$z_i = b_i^{(1)} + \sum_{j=1}^{n} w_{j,i}^{(1)} \cdot x_j \tag{2.2}$$

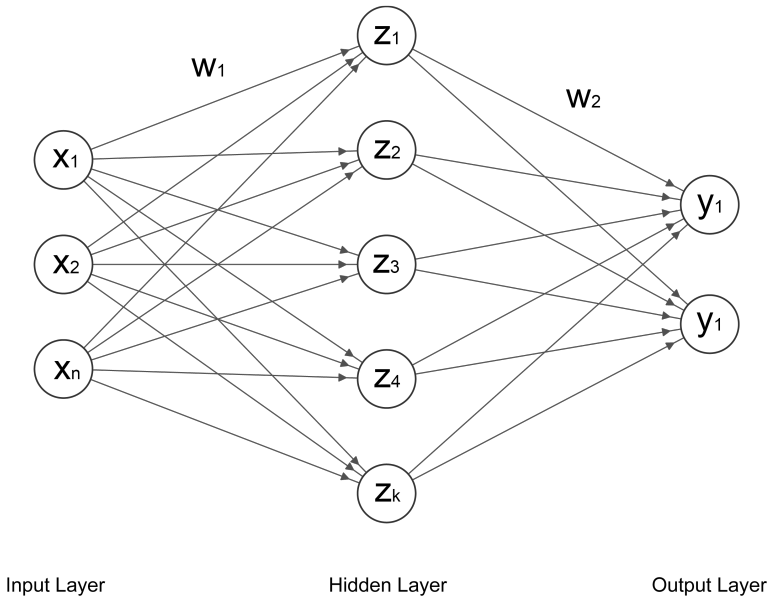


Figure 2.2: Fully Connected Layer.

And the result at the output will counts as follow:

$$\hat{y}_i = g(b_i^{(2)} + \sum_{i=1}^{k} g(z_j) w_{j,i}^{(2)}) \tag{2.3}$$

2.1.3 Deep Neural Network

The key distinction between the neural network and the deep neural network is the presence of multiple layers in the hidden states. Arranged in a stacked configuration comprising multiple layers, with each subsequent layer built upon the output of the previous one.

Each layer $k-1$ is fully connected to the subsequent layer k, resulting in the standard three-step calculation process for the preceding layers. The general formula for calculating the neuron state is defined as follows:

$$z_{k,i} = b_i^{(k)} + \sum_{j=1}^{n} g(z_{k-1,j}) \cdot w_{j,i}^{(k)} \tag{2.4}$$

2.1.4 Loss Functions

As it stated previously the goal of Machine learning in general is to find an algorithm (function) which is able to learn from the data and give a correct answer according to the task (e.g. function that will classify objects in the picture). And the main goal of the neural network is to find such algorithm as well. The term "learning" in machine learning refers to the process of adjusting the model's parameters to minimize the discrepancy (or loss) between the actual data and its predictions. To achieve this, the network calculates the loss, which quantifies the divergence between the predicted output $\hat{y}$ and the target value $\mathbf{y}$.

The selection of an appropriate loss function is of paramount importance in the training of neural networks, as it has a significant impact on the model's performance and convergence. This decision should be made with due consideration of the characteristics of the dataset and the specific objectives of the deep learning task.

To measure the performance of the network (i.e. how good predictions are) we use the Cost function. This function can be described as the *"cost"* of making mistakes in predictions. The loss function measures the error for a single data point, while the cost function is typically the average loss over the entire dataset. Although these terms are often used interchangeably, the cost function generally refers to the aggregate loss across the dataset.

Depending on the task, different loss functions can be chosen. As for instance, mean absolute error [20] and mean squared error [21] are frequently employed in regression problems, whereas binary cross-entropy [22] is the preferred approach for binary classification tasks. In the context of multi-class classification, Categorical Cross-Entropy is frequently employed due to its efficacy in managing multiple output classes. Conversely, Focal Loss can be advantageous for focusing on challenging examples, such as those with complex class distributions.

The principal objective of training a neural network is to identify the set of parameters $\mathbf{W}$ (e.g., weights and biases), which minimises the discrepancy between the predicted output $\hat{y}$ and the true target $\mathbf{y}$. This is accomplished by minimizing a cost function $J(\mathbf{W})$, which represents the aggregate loss across the entire dataset. In mathematical terms, the cost function $J(\mathbf{W})$ is often defined as the average of the individual loss functions across all training examples.

$$J(\mathbf{W}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, \mathbf{Y}^{(i)}) \tag{2.5}$$

where:

- m is the number of training examples,
- $\mathcal{L}(\hat{a}^{(i)}, \mathbf{A}^{(i)})$ is the loss function for a single example, which measures the error between the predicted output $\hat{y}^{(i)}$ and the true label $\mathbf{Y}^{(i)}$.

Minimizing this cost function is the central task of the learning algorithm. The smaller the cost function, the better the network's predictions will align with the true targets.

2.1.5 Gradient Descent

In order to minimize the cost function $J(\mathbf{W})$, we use an optimization algorithm called *gradient descent*. Gradient descent is an iterative method that adjusts the network's parameters $\mathbf{W}$ in the direction that reduces the cost function the most quickly.

The key idea behind gradient descent is to compute the gradient (partial derivatives Equation 2.6) of the cost function with respect to the network's parameters.

$$\frac{\partial f}{\partial \mathbf{x}} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right) \tag{2.6}$$

The gradient indicates the direction of the steepest increase in the cost function. Therefore, to minimize the cost function, we update the parameters in the opposite direction of the gradient.

The update rule for gradient descent is given by:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} J(\mathbf{W}) \tag{2.7}$$

where:

- $\mathbf{W}$ represents the parameters (weights and biases) of the network,
- η is the learning rate, a small positive value that controls the size of the step,
- $\nabla_{\mathbf{W}} J(\mathbf{W})$ is the gradient of the cost function with respect to the parameters $\mathbf{W}$.

The gradient $\nabla_{\mathbf{W}} J(\mathbf{W})$ is calculated using backpropagation, an algorithm that computes the partial derivatives of the cost function with respect to each parameter by efficiently applying the chain rule of calculus (Equation 2.8). Once the gradient is calculated, the parameters are updated to reduce the cost function, and this process repeats iteratively until convergence (i.e., when the cost function reaches a minimum or sufficiently low value).

$$\frac{d}{dx}\left[f(u)\right] = \frac{d}{du}\left[f(u)\right]\frac{du}{dx} \tag{2.8}$$

Example

For a simple neural network with a MSE cost function:

$$J(\mathbf{W}) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - \mathbf{y}^{(i)}\right)^2 \tag{2.9}$$

the gradient descent update rule would involve calculating the partial derivative of $J(\mathbf{W})$ with respect to each weight w_{ij}, and adjusting the weights accordingly:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial J(\mathbf{W})}{\partial w_{ij}} \tag{2.10}$$

This process continues iteratively until the model reaches a point where the cost function is minimized, yielding the optimal set of parameters $\mathbf{W}$ that allow the network to make accurate predictions.

2.1.6 Convolutional Neural Networks

A Convolutional Neural Network is a specialized type of neural network characterized by the use of convolutional layers, which are particularly effective for tasks such as image recognition. CNNs are particularly effective at identifying patterns in grid-like data structures, such as images, where spatial hierarchies are important. They were first

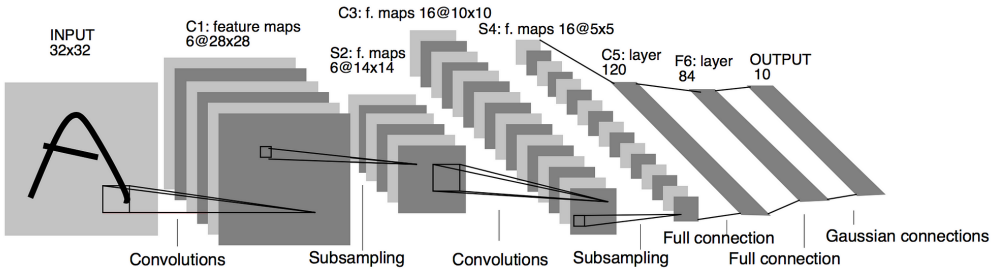


Figure 2.3: LeNet-5 architecture [2].

introduced by Yann LeCun [23] to address the problem of handwritten digit recognition. In addition to their traditional use in image processing, CNNs have also proven effective in complex audio-related tasks, including sound analysis [24] and voice classification [25].

In a basic Convolutional Neural Network, feature extraction is achieved through a sequence of layers. This sequence begins with a convolutional layer applied to the input data, followed by an activation function—typically the ReLU and concludes with a fully connected layer. Optinally, it can be used a pooling layer that reduces the dimensionality of the data. An example of one of the earliest CNN architectures is illustrated in Figure 2.3.

Convolutional Layer

Convolutional layer is the core building block of a CNN (see a Figure 2.4). It performs a convolution operation, which involves sliding a filter (or kernel) over the input data to produce feature maps. Mathematically, the convolution operation can be expressed as:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n) \tag{2.11}$$

where:

- I is the input image,
- K is the convolutional kernel (filter),
- i and j are the coordinates of the output feature map.

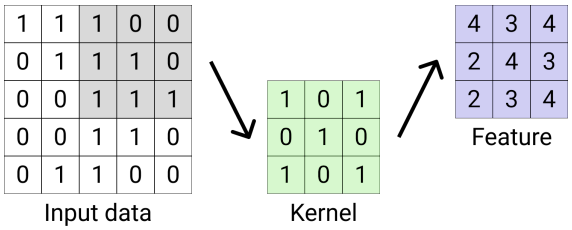


Figure 2.4: Convolution operation on a 5x5 input with a 3x3 kernel.

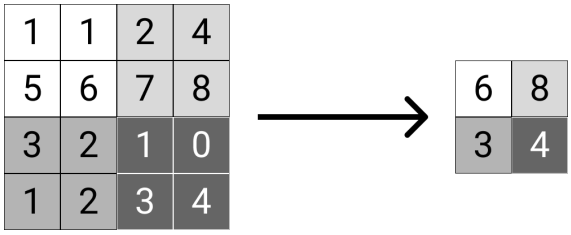


Figure 2.5: Max Pooling with 2x2 filters and stride 2.

The filter K is applied to each position of the input I, resulting in a feature map that highlights specific features like edges or textures.

The convolution operation enables the network to learn spatial hierarchies and detect increasingly complex features in higher layers. Filters vary in size and number, and their weights are learned during the training process through backpropagation.

Pooling layers

Pooling layers are used to reduce the spatial dimensions of feature maps produced by convolutional layers (see a Figure 2.5). This reduction helps to decrease computational complexity, reduce the number of parameters, and control overfitting.

The most common pooling operation is max pooling. For a pooling window of size 2×2, max pooling can be defined as:

$$P(i,j) = \max_{(m,n)\in W} F(i+m, j+n) \tag{2.12}$$

where:

- F is the feature map,
- W is the pooling window,
- P is the output of the pooling operation.

The window W slides over the feature map, retaining only the maximum value within each window.

Another pooling method is average pooling, which computes the average value within each sub-region:

$$P(i,j) = \frac{1}{|W|} \sum_{(m,n)\in W} F(i+m, j+n) \tag{2.13}$$

where $|W|$ is the number of elements in the pooling window.

Pooling layers help to make the network more robust to translations and distortions in the input data by providing translational invariance, meaning the network becomes less sensitive to the exact position of features in the input.

Together, convolutional and pooling layers enable CNNs to efficiently extract and represent features at various levels of abstraction, leading to improved performance in tasks like image and audio recognition.

2.1.7 Training Procedures

In the conclusion of the DL overview we need to introduce the concepts of training the model, which can be used in code implementation. The principal objective during the training of a network is to guarantee that the model produces accurate predictions on new, previously unseen data. Nevertheless, this can prove challenging, particularly in the case of deep networks, where overfitting may occur. Overfitting is a typical problem of deep networks and it can be explained when model is tends to remember the input data, rather to learn patterns. In other words, in overfitting model will always give correct answer on the seen input data, but has high chance to fail on predicting the unseen samples.

The most famous method which deals with overfitting problem is to split dataset into several groups of different proportions (in case of supervised learning) – training and validation (optionally test). Model learns on the training set and always checks their quality by making predictions on the validation set. Final quality of the model can be given by checking how good is model in working with unseen test set.

Due to enormous sizes of dataset, the model trains on some sample of the data (batch) at once. Model has to finish one circle across all data and it denotes as epoch. Furthermore, training is frequently segmented into folds for cross-validation. Cross-validation assesses the model's performance across disparate data subsets, thereby mitigating the risk of overfitting or results that are excessively specific to a single data partition. It is customary to utilise discrete training and validation sets for each fold, thereby enhancing the robustness of the evaluation and preventing the model from memorising patterns that are unique to a single data partition.

The training process also makes use of an **optimizer**, which is an algorithmic tool designed to address the issue of reducing the value of the loss function. In many cases, optimisers make changes to the weights or learning rates of a model to minimize the loss function more effectively. One of the most commonly used optimisers, which is also utilized in the Omnimotion model, is Adaptive Moment Estimation (Adam) [26]. Adam is an advanced algorithmic technique for optimization, belonging to the class of gradient descent algorithms. It is particularly effective when applied to large-scale problems involving substantial amounts of data or parameters. Adam requires less memory and is more computationally efficient than other algorithms of a similar nature, making it a popular choice in deep learning applications.

Adam can be understood as a combination of two powerful gradient descent methodologies: Momentum and Root Mean Square Propagation (RMSProp).

Momentum

The concept of momentum in optimization algorithms is inspired by the physical principle of momentum in classical mechanics. In gradient descent, momentum is used to accelerate the convergence of the optimization process by allowing the optimizer to build up speed in directions with consistent gradients. Specifically, momentum introduces a term that takes into account the past gradients when updating the parameters. This helps to smooth out the oscillations in the gradient descent path and speeds up the convergence in the relevant direction.

In traditional gradient descent, each update to the parameters is solely based on the gradient of the loss function with respect to the parameters at the current iteration. However, with momentum, the update also includes a fraction of the update from the previous step, effectively "remembering" the direction in which the optimizer was moving. This memory allows the optimizer to push through shallow valleys or ravines in the loss landscape and

Algorithm 2.1 Adam: Stochastic Optimization Algorithm [26]

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ # Initialize 1st moment vector
$v_0 \leftarrow 0$ # Initialize 2nd moment vector
$t \leftarrow 0$ # Initialize timestep
while θ_t not converged
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ # Get gradients w.r.t. stochastic objective at timestep t
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ # Update biased first moment estimate
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ # Update biased second raw moment estimate
    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ # Compute bias-corrected first moment estimate
    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ # Compute bias-corrected second raw moment estimate
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\varepsilon}$ # Update parameters
end while
return θ_t # Resulting parameters

reach the minimum more quickly.

Root Mean Square Propagation

RMSProp is another enhancement to standard gradient descent, designed to adapt the learning rate for each parameter individually. The core idea behind RMSProp is to normalize the gradient by dividing it by an exponentially decaying average of its recent magnitude. This ensures that the learning rate is adjusted based on how steep or flat the loss landscape is for each parameter.

RMSProp effectively deals with the problem of varying gradients by maintaining a running average of the squared gradients for each parameter. This allows the optimizer to dampen the learning rate in directions where the gradients are large, preventing the optimizer from taking too large steps, and to increase the learning rate in directions with small gradients, enabling faster convergence. This approach is particularly useful in scenarios where the gradients differ significantly across different dimensions of the parameter space, which is common in deep learning models.

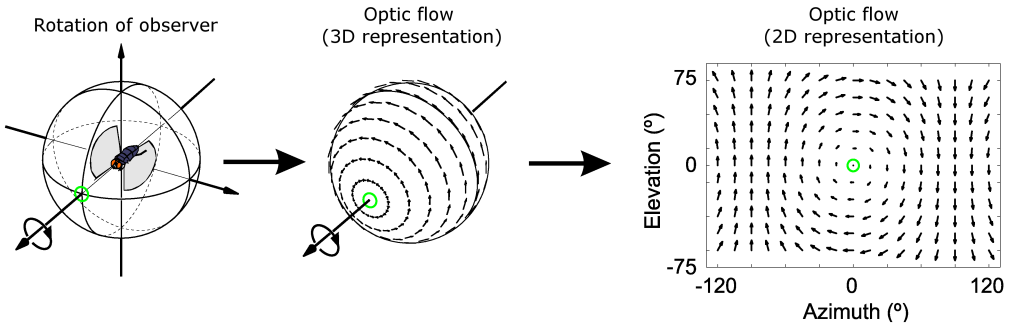


Figure 2.6: Optical flow example overview.

2.2 Computer vision

2.2.1 Optical flow

Computer vision is a rapidly evolving sub-field of Artificial Intelligence and Computer Science, dedicated to enabling machines to interpret and understand visual information from the world. This field encompasses a variety of techniques and technologies that allow computers to process and analyze images and video data, emulating the human visual system's ability to recognize and interpret visual stimuli.

CV leverages a spectrum of techniques ranging from traditional image processing methods to modern machine learning approaches. In recent years, deep learning, particularly convolutional neural networks, has revolutionized the field by significantly enhancing the accuracy and efficiency of visual recognition tasks. Deep learning models excel in learning complex patterns and representations from large datasets, driving advancements in object detection, image classification, and more [27] [28] [29] [30] [31].

Optical flow (OF) is a concept in computer vision and image processing that refers to the pattern of apparent motion of objects, surfaces, and edges in a visual scene, based on the movement of pixels between successive frames in a video sequence. It is used to estimate the motion of objects and the camera itself within the scene.

The motion field refers to the projection of the actual 3D velocities of objects in the scene onto the 2D image plane. While optical flow represents the apparent motion of pixel intensities between frames, the motion field reflects the true motion of objects as they move in the 3D space relative to the observer. In an ideal scenario, the optical flow would perfectly match the motion field. However, in practice, optical flow is often an approximation of the motion field due to limitations such as occlusions, noise, and changes in lighting. Discrepancies between optical flow and the motion field can arise when the assumptions behind optical flow computation—such as brightness constancy and small motion—are violated.

Optical flow is determined by examining the spatio-temporal patterns of moving objects projected onto an image plane. The value of optical flow at a pixel indicates how much the pixel has shifted between consecutive frames, effectively allowing us to measure the relative motion between objects and the observer. To compute optical flow, the technique relies on the Intensity Coherence assumption, which posits that the brightness of a specific point remains constant or nearly constant across two successive images. Essentially, two key assumptions are necessary for effectively utilizing optical flow:

- The intensities of the pixel of an object do not change between consecutive frames
- The pixels that are neighbors (concecutive or have small time difference) have a similar motion

Imagine we have a coordinate (x, y) of particular pixel on the 2D frame of the video at the moment t and we knew that at the moment $t + \delta t$ the coordinates will also change by δ. Optical flow can be measured as:

$$(u, v) = (\frac{\delta x}{\delta t}, \frac{\delta y}{\delta t}) \tag{2.14}$$

where u and v represent the velocity of pixel movement in the x and y directions, respectively, and are components of the optical flow.

Taking into consideration first assumption about the intensity we get:

$$\begin{aligned} I(x, y, t) &\mapsto I(x + \delta x, y + \delta y, z + \delta z) \\ I(x, y, t) &= I(x + \delta x, y + \delta y, z + \delta z) \end{aligned} \tag{2.15}$$

Now let's defined a function $f(x + \delta x)$ and use Taylor Series on it:

$$f(x + \delta x) = f(x) + \frac{\delta f}{\delta x} \cdot \delta_x + \frac{\delta^2 f}{\delta x^2} \cdot \frac{\delta x^2}{2!} + \cdots + \frac{\delta^n f}{\delta x^n} \cdot \frac{\delta x^n}{n!} \tag{2.16}$$

From the second assumption we can re-write it as follows:

$$f(x + \delta x) = f(x) + \frac{\delta f}{\delta x} \cdot \delta_x + O(\delta x^2) \tag{2.17}$$

where $O(\delta x^2) \mapsto 0$

The initial function f based on three variables (x, y, t) so the formula would looks like:

$$\begin{aligned} f(x + \delta x, y + \delta y, t + \delta t) &= f(x, y, t) + \frac{\partial f}{\partial x} \cdot \delta x + \frac{\partial f}{\partial y} \cdot \delta y + \frac{\partial f}{\partial t} \cdot \delta t, \\ I(x + \delta x, y + \delta y, t + \delta t) &= I(x, y, t) + \frac{\partial I}{\partial x} \cdot \delta x + \frac{\partial I}{\partial y} \cdot \delta y + \frac{\partial I}{\partial t} \cdot \delta t, \\ I(x + \delta x, y + \delta y, t + \delta t) &= I(x, y, t) + I_x \cdot \delta x + I_y \cdot \delta y + I_t \cdot \delta t. \end{aligned} \tag{2.18}$$

If we subtract Equation 2.16 from Equation 2.18, then divide it by δt and take limit as $\delta t \mapsto 0$ we will get a **Constraint Equitation**:

$$\begin{aligned} I_x \frac{\delta x}{\delta t} + I_y \frac{\delta y}{\delta t} + I &= 0 \\ I_x u + I_y v + I_t &= 0 \end{aligned} \tag{2.19}$$

The problem is that in this expression (u, v) are not known or it is not possible to find a solution with traditional methods. Several methods have been proposed to solve this problem and the most used is the Lucas-Kanade method [32].

2.2.2 Lucas-Kanade Method

Optical flow estimation problem is an under constraint problem (we have one Equation 2.19 and two unknown parameters). So the Lucas-Kanade method takes an assumption, that OF in a very small neighborhood in the scene is the same for all points within that neighborhood (Figure 2.7).

Assumption: For each pixel, assume Motion Field and hence Optical flow (u, v), is constant within a small neighborhood W:

- that's for all points $(k, l) \in W$,
- so $I_x(k,l)u + I_y(k,l)v + I_t(k,l) = 0$.

Figure 2.7: Example of the neighborhood for Lucas-Kanade method [3].

This equations are made for some point k, l, but since we have multiple number of points within the neighborhood we would get a system of equations (also its matrices form):

$$\forall (k,l) \in W : I_x(k,l)u + I_y(k,l)v + I_t(k,l) = 0$$

$$\underbrace{\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ I_x(k,l) & I_y(k,l) \\ \vdots & \vdots \\ I_x(n,n) & I_y(n,n) \end{bmatrix}}_{\substack{\text{A} \\ \text{(known)} \\ n^2 \times 2}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\substack{\text{u} \\ \text{(unknown)} \\ 2 \times 1}} = \underbrace{\begin{bmatrix} I_t(1,1) \\ I_t(k,l) \\ \vdots \\ I_t(n,n) \end{bmatrix}}_{\substack{\text{B} \\ \text{(known)} \\ n^2 \times 1}} \tag{2.20}$$

where n is a size of W window.

Now we have n^2 equations, 2 unknowns $\mapsto$ Find Least Squares Solutions. We need to solve linear system $Au = B$. We take $A^T Au = A^T B$ (Least-Squares using Pseudo-Inverse). In matrix form it looks as follow (without

k,l indices for simplicty):

$$\underbrace{\begin{bmatrix} \sum_w I_x I_x & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w I_y I_y \end{bmatrix}}_{\substack{A^T A \\ \text{(known)} \\ 2 \times 2}} \quad \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\substack{\text{u} \\ \text{(unknown)} \\ 2 \times 1}} = \underbrace{\begin{bmatrix} -\sum_w I_x I_t \\ -\sum_w I_y I_t \end{bmatrix}}_{\substack{A^T B \\ \text{(known)} \\ 2 \times 1}} \tag{2.21}$$

Which brings us to the final and simple equation to solve:

$$Au = B \qquad\qquad u = (A^T A)^{-1} A^T B \tag{2.22}$$

But there are several requirements that has to be done so the formula will be correct:

- $A^T A$ must be invertible (i.e. $def(A^T A) \neq 0$,
- $A^T A$ must be well-conditioned:
  - if λ_1 and λ_2 are eigenvalues of $A^T A$, then
  - $\lambda_1 > \varepsilon$ and $\lambda_2 > \varepsilon$,
  - $\lambda_1 \geq \lambda_2$ but not $\lambda_1 \ll \lambda_2$,

when λ_1 and λ_2 are very small we cannot estimate OF.

These concepts of Lucas-Kanade were evolved and improved in various of researches in Optical Flow area. One of the most is Recurrent All-Pairs Field transforms, which is used as base for the Omnimotion model. The RAFT model outputs optical flow and serves as a pre-processing stage in the Omnimotion model.

2.2.3 Recurrent All-Pairs Field transforms

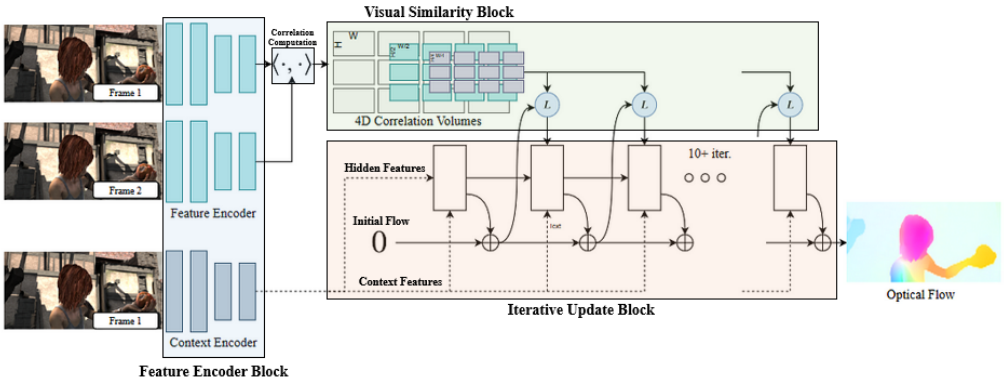


Figure 2.8: RAFT method overview. [4]

Usually, optical flow methods have problems with correctness of estimating dense displacement fields between image pairs, depending strongly on manually crafted optimization problems. These methods typically balance two main components: a data term that aligns visually similar regions and a regularization term that enforces plausible motion patterns. While this approach has been effective in many scenarios, its progress is limited by the challenge of designing an optimization objective capable of handling diverse and complex edge cases, such as occlusions, rapid motion, or textureless regions.

Deep learning has emerged as a promising alternative. Rather than formulating an optimisation problem as it was before in OF, deep learning scientists train networks to predict optical flow directly. DL techniques have demonstrated performance that is on a par with the best traditional methods while being significantly faster during inference [33] [34] [35] [36] [37]. Looking ahead, the key challenge is to develop architectures that not only improve performance and ease of training but also generalise well to new and diverse scenes.

In light of these considerations, the *State-of-the-art* solution, namely, Recurrent all-pairs field transforms [4] was published.

RAFT is a deep network architecture which solves the problem of OF and can be splitted into three main sections (see the Figure 2.8):

- The first stage of the process is the extraction of a feature vector for each pixel of two given frames, which is achieved through the use of a feature encoder.
- The second stage involves the generation of a 4D correlation volume for all pairs of feature vectors, with subsequent pooling of shapes 1, 2, 4, 8 to produce lower resolution volumes.
- The third stage comprises the utilisation of a recurrent GRU-based [38] update operator (Iterative Update Block in the Figure 2.8), which retrieves values from the correlation volumes from the second frame (out of 2 in input) and iteratively updates a flow field that is initially set to zero.

Feature Extraction

Extracting feature vector from the two successive input images is applied by using a CNN. The feature encoder (FE) network is engaged to both I_1 and I_2 and maps the input images to dense feature maps at a lower resolution. Encoder (denoted as g_θ) outputs features at $1/8$ resolution $g_\theta : \mathbb{R}^{H\times W\times 3} \rightarrow \mathbb{R}^{H/8\times W/8\times D}$ where set $D = 256$. The FE consists of 6 residual blocks, 2 at $1/2$ resolution, 2 at $1/4$ resolution, and 2 at $1/8$ resolution (more details in the supplementary material).

As an addition on that the context network (denoted as h_θ) is being used, which is another FE network with the same structure. The main difference is that it extracts features only from the first input image I_1. Together, the g_θ and the h_θ performed once and define the first stage of the RAFT model.

Computing Visual Similarity

RAFT calculates visual similarity by creating a complete correlation volume between every pair of images. For image features $g_\theta(I_1) \in \mathbb{R}^{H\times W\times D}$ and $g_\theta(I_2) \in \mathbb{R}^{H\times W\times D}$, this correlation volume is generated by performing

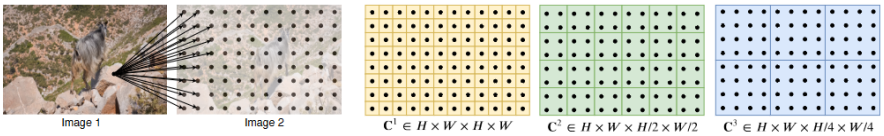


Figure 2.9: Building correlation volumes [4]

the dot product across all feature tensor pairs. The resulting correlation volume C can be computed efficiently with a single matrix multiplication.

$$C(g_\theta(I_1), g_\theta(I_2)) \in \mathbb{R}^{H\times W\times H\times W}, \quad C_{ijkl} = \sum_h g_\theta(I_1)_{ijh} \cdot g_\theta(I_2)_{klh} \tag{2.23}$$

The model creates a 4-layer pyramid $\{C^1, C^2, C^3, C^4\}$ by averaging over the last two dimensions of the correlation volume using kernel sizes of 1, 2, 4, and 8, and applying a pyramid stride of $\lfloor \text{floor} \rfloor 2^k$ (see Figure 2.9. Consequently, each volume C^k has dimensions $H \times W \times H/2^k \times W/2^k$. This pyramid of volumes captures information about both large and small displacements. By preserving the first two dimensions (corresponding to I_1), high-resolution details are maintained, enabling the method to detect the movements of small, fast-moving objects effectively.

Correlation Lookup

The authors introduced a lookup operator $\mathcal{L}_C$ that creates a feature map by indexing from the correlation pyramid. For a given current estimate of optical flow (f^1, f^2), each pixel $x = (u, v)$ in I_1 is mapped to its estimated correspondence in I_2 as $x' = (u + f^1(u), v + f^2(v))$. A local grid is then defined around this point x'.

$$\mathcal{N}(x') = \{x' + \text{dx} \mid \text{dx} \in \mathbb{Z}^2, \|\text{dx}\|_1 \leq r\} \tag{2.24}$$

The set of integer offsets within a radius of r units around x', using the $L1$ distance, defines a local neighborhood $\mathcal{N}(x')$. This neighborhood is used to index into the correlation volume, with bilinear sampling applied since $\mathcal{N}(x')$ is a grid of real numbers.

Lookups are performed across all pyramid levels, with the correlation volume at level k, C^k, being indexed using the grid $\mathcal{N}(x'/2^k)$. A constant radius across levels implies that lower levels provide a broader context. For instance, at the lowest level $k = 4$, a radius of 4 corresponds to a range of 256 pixels in the original resolution. The volumes from each level are concatenated to form a single feature map.

Efficient Computation for High-Resolution Images

The all-pairs correlation computation scales as $O(N^2)$, where N is the number of pixels. While this is computationally expensive, it needs to be computed only once and reused in subsequent iterations. Alternatively, an implementation that scales as $O(NM)$ can be used, which leverages the linearity of the inner product and average pooling. For instance, considering the cost volume at level m, C^m_{ijkl}, between the image pairs $g_\theta(I_1)$ and $g_\theta(I_2)$, this

approach can be more efficient.

$$C^m_{ijkl} = \frac{1}{2^{2m}} \sum_p^{2^m} \sum_q^{2^m} (g^{(1)}_{i,j}, g^{(2)}_{2^m k+p, 2^m l+q}) = (g^{(1)}_{i,j}, \frac{1}{2^{2m}} \sum_p^{2^m} \sum_q^{2^m} g^{(2)}_{2^m k+p, 2^m l+q}) \tag{2.25}$$

The value at C^m_{ijkl} is derived from the average correlation response over a $2^m \times 2^m$ grid. This value can be computed as the inner product between the feature vectors $g_\theta(I_1)$ and $g_\theta(I_2)$ after pooling with a $2^m \times 2^m$ kernel.

In the alternative implementation, instead of precomputing all correlations, they precompute the pooled image feature maps. Correlation values are computed on demand during each iteration, resulting in $O(NM)$ complexity.

Empirically, precomputing all pairs is manageable and not a bottleneck due to efficient GPU matrix operations. For videos with a resolution of $1088x1920$, it accounts for only 17% of the total inference time. However, the alternative method can be employed if precomputation becomes a limitation.

Iterative Updates

The update operator estimates a sequence of flow values $\{f_1, ..., f_T\}$ starting from $f_0 = 0$. In each iteration, it generates an update direction Δf which is added to the current estimate, yielding $f_{k+1} = \Delta f + f_k$.

The update operator processes the flow, correlation, and a latent hidden state to produce the update Δf and an updated hidden state. Its architecture is designed to resemble optimization steps, incorporating techniques like tied weights and bounded activations to promote convergence. The operator is trained to ensure that the sequence converges to a fixed point $f_k \to f^*$.

3

The omnimotion architecture

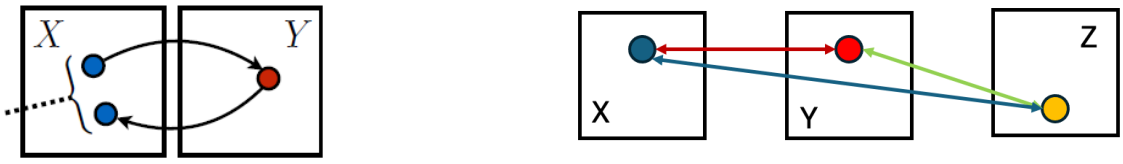
The Omnimotion [5] system was presented at the International Conference on Computer Vision in 2023 as a student paper, entitled "*Tracking Everything Everywhere, All at Once*" . As previously stated, the primary distinction between this tracking model and others is its capacity to estimate the full-length motion of all pixels in the input video. The authors highlight that their solution allows for the tracking of every point in a video, even when occluded, and ensures "*cycle consistency*", which is reflected in the name "*Everything, Everywhere.*" An optimised representation per video is employed to jointly solve for the motion of the entire video, a process described as "*All at Once*". This is a crucial aspect of the approach, as most existing solutions in this area tend to focus on the occlusal aspect, particularly in short-term occlusion situations. The initial dense tracking model of Omnimotion can be divided into two main sections: pre-processing and Omnimotion. These are further divided into several inner stages.

3.1 Pre-processing

In the pre-processing stage, the RAFT produce optical flow to get pairwise correspondences. Such correspondences are used as a ground truth for the following training stage. Optical flow is employed in the training phase as a fundamental motion estimation technique. The authors modified the order of making predictions at the RAFT model. At the computing flow field stage between base frame **i** and target frame **j**, flow prediction is employed for the frame preceding the target frame, **j-1**, when feasible. This modification to the prediction process had a beneficial effect on the quality of predictions for long-distance frames, but further improvements in quality are still required. Besides RAFT model authors also other models to produce optical flow, such as TapNet [39].

However, RAFT has limitations in terms of noise and occlusion handling, which affect its suitability as the most optimal solution for optical flow. As an enhancement, the authors proposed a series of filter techniques to remove false flows and to keep the correct ones.

The filtering stage of the pre-processing phase ensures the consistency of both the location and the appearance track predicted by RAFT. Cycle consistency represents an additional control mechanism for ensuring the accuracy of the correspondences. The common description of this control is illustrated in **Figure 4.2a**, which depicts the mapping of a point from frame X to frame Y. In the case of backward mapping, the objective is to return the point to its original location in frame X with the minimum possible difference. In the original paper, a threshold of 3 pixels was identified as an acceptable error margin.



(a) Cycle consistency.

(b) Chaining.

Figure 3.1: (a) Cycle consistency and (b) Chaining.

Despite the cycle consistency check, the authors report in fact that there are still common cases when some pairs can be misclassified or even lost as it displayed in **Figure 3.2**.

As the results indicated, there were instances of spurious correspondence. Such errors are possible due to the inherent limitations of the cycle check process. Flow networks may encounter difficulties in estimating motion for regions that undergo significant deformation between the two input frames. Instead, they may resort to motion interpolation from surrounding regions, effectively "*locking on*" to the background. Such cases may evade the cycle consistency check and subsequently impair the prospective quality of the prediction. In order to address this issue, the authors of the original paper incorporated an additional filtering stage named "*appearance check*", which will be described later.

After the appearance has been done, they introduced the **double cycle consistency** – a straightforward strategy for the detection of reliable flow in occluded regions and return it back to the ground truth. For the classic cycle consistency for each pixel, two distinct flows are computed: the forward flow to a target frame (a), the cycle flow (flow back to the source frame from the target pixel) (b). As addition for double cycle consistency they compute a second forward flow (c), whereby the consistency between (b) and (c) forms a secondary, supplementary one.

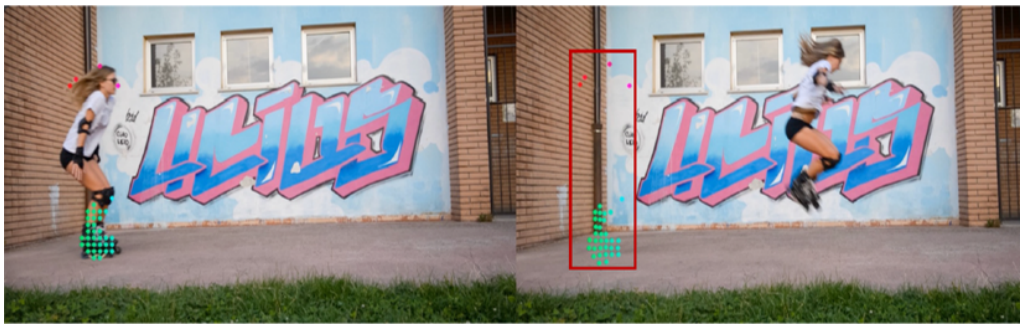


Figure 3.2: Erroneous correspondences check. [5]

Pixels where (a) and (b) are inconsistent but (b) and (c) are consistent are identified as occluded pixels. This approach proved effective in identifying reliable flows for occluded regions, particularly when the two frames were in close proximity. Consequently, correspondences spanning a temporal distance of less than three frames were permitted to bypass cycle consistency checks.

An optional 'chaining' mechanism was introduced as a means of controlling the mapping of points across multiple consecutive frames. This is illustrated in Figure 7.1b, which also provides an explanation of the underlying concept. In essence, the point selected from frame X which is mapped to the point Y can be mapped to the point in the frame Z if and only if the points in frames Y and Z have a mapping between them.

3.1.1 Self-distillation with no labels

Self-**Di**stillation with **no** labels (DINO) [40] is a self-supervised learning method for training Vision Transformers [41] and other neural networks without the need for labelled data. It was first presented by Facebook in the paper "Emerging Properties in Self-Supervised Vision Transformers". DINO is based on the combination of the ViTs and employs self-distillation.

Vision Transformers represent a neural network architecture that adapts the Transformer model, originally designed for Natural Language Processing, to image recognition tasks. In contrast to traditional Convolutional Neural Networks, vision transformers employ a partitioning of the image into smaller patches, which are then treated as a sequence of tokens analogous to words in a sentence. Each patch is embedded into a vector, and positional encodings are added to ensure the preservation of spatial information. The self-attention mechanism within the Transformer enables the model to focus on different parts of the image, capturing global context and relationships between distant patches. This approach allows ViTs to excel at understanding complex visual patterns that span across the image. While ViTs require large datasets and significant computational resources to train effectively, they have demonstrated competitive performance with, and commonly exceeding, traditional CNNs. Their scalability and flexibility make them suitable for various vision tasks beyond classification, such as segmentation and detection.

Another method is used in DINO, so called Knowledge distillation [42] is implemented. It is a training technique whereby a smaller, more efficient "*student*" model is trained to emulate the behaviour of a larger, more complex "*teacher*" model. The teacher model, which is typically highly accurate but computationally expensive, provides "*soft*" predictions (probabilities across classes) that the student model learns to replicate. By concentrating on the knowledge embedded in the teacher's output, the student model is capable of attaining a comparable level of performance while being considerably smaller and faster.

In the DINO model they modified classical knowledge distillation approach. They changed the manner in which the input data (e.g. an image) is presented. The input data is divided into 2 half and several little. The teacher receives one of these 50% samples of the original input in the form of a patch (a portion of the image), while the student receives a randomly picked smaller proportion or second half (not the same as teacher).This allocation enables the model to learn the global context from the teacher and small local details from the student. This technique is particularly beneficial for deploying models in resource-constrained environments, as it allows for high accuracy with reduced computational overhead.

The DINO scheme is depicted in **Figure 3.3**. As in the case of knowledge distillation, the two models in question are of the same vision transformer architecture, but differ in terms of the parameters used: the student and

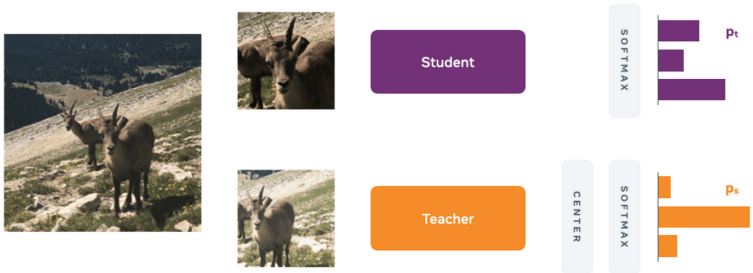


Figure 3.3: DINO example. [6]

teacher models. The DINO model applies two distinct, randomly selected transformations of the input image to the student and teacher networks. The teacher is a "*momentum-based*" teacher, which signifies that its weights are an exponentially weighted average of the student's weights. This concept was first introduced in the paper "*Momentum Contrast for Unsupervised Visual Representation Learning*" as a means of preventing mode collapse, which is a phenomenon whereby the teacher and student models become identical and produce the same embeddings regardless of the input. The weight update for the teacher is illustrated in Equation 3.1, with λ following a cosine schedule from 0.996 to 1 during training, as originally presented in the paper. As illustrated in **Figure 3.4**, the model incorporates an additional step involving centring with a mean computed over the batch. Each network generates a K-dimensional feature, which is then normalised with a temperature SoftMax over the feature dimension. The similarity between the two features is subsequently measured with a cross-entropy loss function. A stop-gradient operator is applied to the teacher network, enabling the propagation of gradients solely through the student network. The teacher parameters are updated with an exponential moving average of the student parameters.

$$\theta_t \leftarrow \lambda\theta_t + (1-\lambda)\theta_s \tag{3.1}$$

To sum up, at the pre-processing stage, the model is required to generate optical flow by applying the RAFT model. This is done in order to collect pairwise correspondences, based on the mappings that have been constructed in the main part of the model. However, prior to the subsequent section, it is necessary to filter the data in order to ensure 'cycle consistency'. However, this filtering method is not guaranteed to be optimal, as it may fail to identify instances of rapid motion that cannot be detected through cycle consistency evaluations. This issue can be addressed by utilising DINO as an 'appearance check'. An optional chaining process may be employed to enhance the accuracy of the motion estimation, which serves as the ground truth.

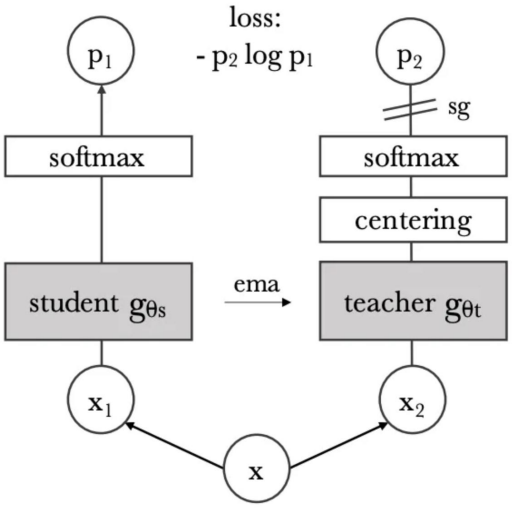


Figure 3.4: DINO in the case of one single pair of views (x1, x2). [7]

3.2 Dense Pixel Tracking

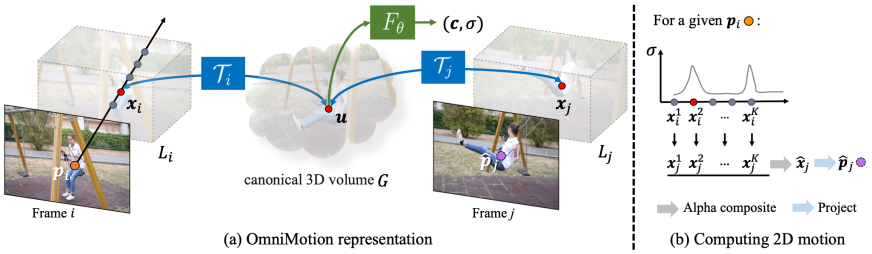


Figure 3.5: Omnimotion method overview. [5]

Once the model has generated pairwise optical flow and filtered it during the pre-processing stage, the Omnimotion phase commences. The objective of the Omnimotion system is to produce a dense (i.e. pixel-level) and long-range motion estimation based on the ground truth made by another tracking network at the pre-processing phase. The primary distinction between existing solutions and this approach is that it enables tracking through occlusions while maintaining global consistency in estimated motion trajectories. The authors assert the presence of cycle and appearance consistency during the pre-processing stage, while the main part substantiates the complete and global consistency of estimated motion trajectories. To achieve this, they proposed the concept of a global canonical 3D volume (3D global representation of the scene), as illustrated in **Figure 3.5**.

The canonical volume can be defined as a data structure that encodes the trajectories of all scene points in single space. This representation enables the maintenance of accurate tracks despite occlusions. In other words, Omnimotion is creating 3D bijections between local volume to canonical volume (volume based on exact frame

at moment of time i). However, it is not feasible to conceptualise the representation of empirical data as a series of immutable, sequential layers. Similarly, the full-detailed 3D reconstruction process requires a considerable amount of power and memory resources, which presents an ill-posed problem. More important that such issues arise in the context of static scenes, while Omnimotion is capable of handling the dynamic scenes derived from video sequences. The authors' solution to this problem is based on creating a so-called "quasi 3D" representation as the canonical volume, which is not-full ground truth 3D representation. This is not an absolute true 3D physical representation, but it still can guarantee globalness and completeness. Furthermore, this solution seems reasonable given, that the majority of the input data is an incomplete portion set of some scene with a limited number of angles captured by regular video recording devices, which is not giving wholesome information about the real scene.

The Omnimotion system employs the video content to represent it as a canonical volume **G** (as illustrated in **Figure 3.5**), which serves as a three-dimensional atlas of the observed scene. As with NeRF, the authors define a Coordinate-Based Network, F_θ, over volume **G**, which maps each coordinate $u \in G$ from canonical volume to colour and density (σ). In combination with the 3D bijections, it allows tracking of surfaces over multiple frames as well as reasoning about occlusion relationships. The colour stored in the canonical volume allows Omnimotion to compute a photometric loss during optimisation.

3.2.1 The Neural Radiance Fields

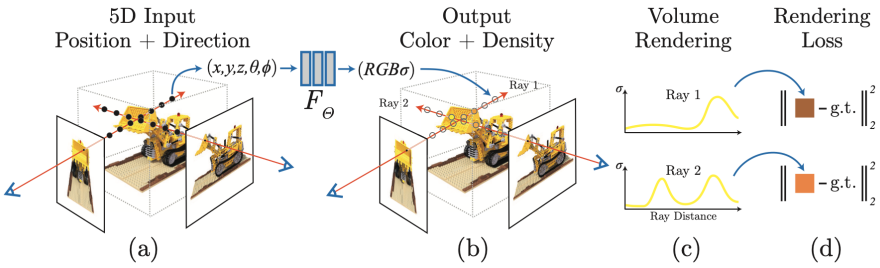


Figure 3.6: Neural Radiance Fields method overview. [8]

The **N**eural **R**adiance **F**ields [8] algorithm represents a deep learning approach for the generation of three-dimensional scene representations from two-dimensional images (see **Figure 3.6**). A continuous scene is represented as a 5D vector-valued function, with the input comprising a 3D location (x, y, z) and a 2D viewing direction (θ,φ), and the output consisting of an emitted colour (r, g, b) and volume density (σ). In practice, the function is the direction expressed as a three-dimensional Cartesian unit vector, designated as d. The continuous five-dimensional scene representation is approximated by an MLP network, designated as F_θ, which maps from the input five-dimensional coordinate to the corresponding volume density and directional emitted colour. The weights of the network θ are optimised to achieve this mapping. The result of NeRF is a realistic image that reflects the scene from the camera's specific viewpoint.

The objective is to ensure multiview consistency by limiting the network to predicting the volume density σ as a function of the location x alone, while allowing the RGB colour to be predicted as a function of both location

and viewing direction. To achieve this, the MLP F_θ initially processes the input 3D coordinate x with eight fully-connected layers (utilising ReLU activations and 256 channels per layer), subsequently generating σ and a 256-dimensional feature vector. This feature vector is then concatenated with the camera ray's viewing direction and passed to one additional fully-connected layer (employing a ReLU activation and 128 channels), which outputs the view-dependent RGB colour.

To generate a two-dimensional image from the three-dimensional representation, the NeRF algorithm utilises established volume rendering techniques. This is accomplished by sampling a multitude of points along rays projected from the camera and calculating the cumulative contribution of each sampled point based on its predicted volume density and radiance. The volume density $\sigma(x)$ can be understood as the differential probability that a ray will end at an infinitesimally small particle located at position x. The anticipated colour C(r) of camera ray $r(t) = o + td$ (where **o** is initial position of the camera, **d** is direction vector and **t** is an investigated point on that vector), with near and far limits t_n and t_f is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt \quad \text{where} T(t) = exp(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\, ds) \tag{3.2}$$

In this context, the function **T(t)** is used to denote the *accumulated transmittance* along the ray from t_n to t. This can be understood as the probability that the ray travels from t_n to t without hitting any other particle. In order to render a view from our continuous neural radiance field, it is necessary to estimate this integral **C(r)** for a camera ray traced through each pixel of the desired virtual camera.

3.2.2 3D mapping

The authors presented a system of continuous bijective mapping, designated as T_i, which was employed to map 3D points, x_i, from each local coordinate frame L_i, to the canonical 3D coordinate frame, as $u = T_i(x_i)$. These points were taken from the rendering of NeRF rays Figure 3.5. In this context, **i** represents a time state index (i.e. index of frame). From the figure and definition, it is evident that the coordinate **u** is time-independent and serves as a globally consistent "*index*" for a specific scene point or 3D trajectory across time. By composing these bijective mappings and their inverses, it is possible to map a 3D point from one local 3D coordinate frame (L_i) to another frame (L_j):

$$x_j = T_j^{-1} \circ T_i(x_i) \tag{3.3}$$

where T_j and T_i are the same network with different hidden code.

Invertible Neural Networks

Bijective mappings ensure that the correspondences between three-dimensional points in individual frames are cycle consistent, as they all originate from the same canonical point. In order to enable the creation of highly expressive mappings that can effectively capture the nuances of real-world motion, they represent these bijections using Invertible Neural Networks [43]. In light of recent advances in homeomorphic shape modelling, the authors have elected to employ **Real**-valued **N**on-**v**olume **P**reserving [44], largely due to its uncomplicated formulation and analytic invertibility.

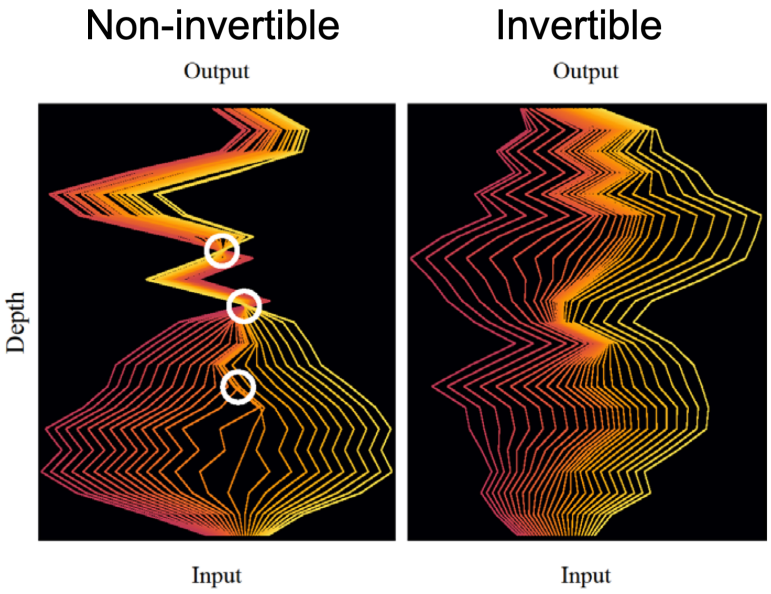


Figure 3.7: Invertible Neural Network. [9]

Invertible neural networks constitute a specific category of neural networks, designed in such a way that their transformations are invertible (see **Figure 3.5**). This implies that, based on the output of the network, it is possible to reconstruct the original input by applying the inverse of the transformation. In other words, the network is not only searching for the forward function F_θ from input space $\mathbf{X}$ to output space $\mathbf{Z}$, but also for the backward function F_θ^{-1} (see Equation 3.4). In contrast to traditional neural networks, where information may be lost during the forward pass, Invertible Neural Networks guarantee the preservation of all information, thereby enabling a one-to-one mapping between input and output.

$$\begin{aligned} F_\theta : \mathbb{R}^d &\mapsto \mathbb{R}^d & \qquad F_\theta^{-1} : \mathbb{R}^d &\mapsto \mathbb{R}^d \\ x &\mapsto z & z &\mapsto x \end{aligned} \tag{3.4}$$

Real-NVP is responsible for constructing bijective mappings through the utilisation of simple bijective transformations, otherwise known as affine coupling layers (see in **Figure 3.8**). The learnable component in each affine coupling layer is an MLP that computes a scale and a translation from a frame latent code ψ_i (i.e. representation of time index of the frame) and the initial portion of the input coordinates x_i and y_i. Subsequently, the scale and translation are applied to the second part of the input coordinate z_i. This process is then repeated for each of the other coordinates. The MLP network in each affine coupling layer has three layers with 256 channels. It was found that the fitting ability of the MLP was improved by the application of positional encoding to the MLP's input coordinates, and the number of frequencies was set to four.

The Omnimotion team used six affine coupling layers from Real-NVP, but modified them to also condition on a per-frame latent code, ψ_i. Subsequently, all invertible mappings T_i are parameterised by the identical Invertible Network $M\theta$ (**Figure 3.5**), albeit with disparate latent codes: $T_i(\cdot) = M\theta(\cdot; \psi_i)$. Further enhancements

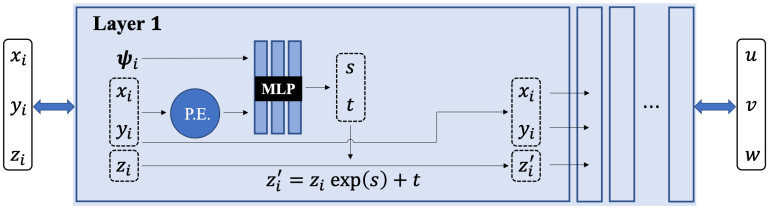


Figure 3.8: Network architecture for the mapping network $M\theta$. [5]

to the layer are applicable when utilising embedding over time-sensitive operations, facilitating an upscale from a dimension of size 1 to a dimension of size 128. Embedding represents a conventional deep learning methodology employed when the input data exhibits a limited shape, with the objective of augmenting the quantity of learnable data. These alterations are exemplified in Figure 7. The initial one-dimensional value (blue point) can be represented as a set of diverse periodic functions at that point (green points). Typically embedding can be reached by applying periodic functions (sin, cos, etc.) and different variations of them as it seen in **Figure 3.9**.

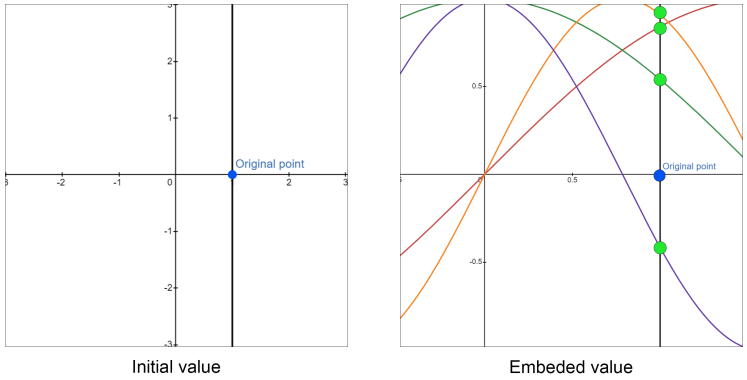


Figure 3.9: Positional encoding $M\theta$.

The positional encoding implemented by feature extraction with Gabor-Net model [45].

GaborNet

GaborNet is a neural network architecture that integrates the principles of Gabor filters into its design, making it particularly effective for tasks involving pattern recognition and image processing. Gabor filters are widely recognized for their ability to capture spatial frequency characteristics and orientations in images, mimicking the human visual system's early stages of processing.

Gabor Filters

Gabor filters are linear filters used for edge detection, texture analysis, and feature extraction in images. Each Gabor filter is a sinusoidal plane wave modulated by a Gaussian envelope, and it is defined by the following equation:

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi\frac{x'}{\lambda} + \psi\right) \tag{3.5}$$

where:

- x' and y' are the coordinates after rotation by the angle θ,
- λ is the wavelength of the sinusoidal factor,
- θ is the orientation of the normal to the parallel stripes of the Gabor function,
- ψ is the phase offset,
- σ is the standard deviation of the Gaussian envelope, and
- γ is the spatial aspect ratio, specifying the ellipticity of the support of the Gabor function.

In GaborNet, the layers are designed to emulate the functionality of Gabor filters. The first layer applies a set of Gabor filters to the input, which allows the network to extract multi-scale, multi-orientation features directly from the data. This preprocessing step enhances the network's ability to capture texture, edges, and other spatially localized features.

The output of the Gabor filter layer is then passed through conventional neural network layers, such as convolutional layers, pooling layers, and fully connected layers. These layers further process the extracted features, enabling the network to perform complex tasks like classification, segmentation, and detection.

3.2.3 Frame-to-frame motion

In light of the previous explanation of representation, the authors proceed to elucidate the methodology by which they computed two-dimensional motion for any query pixel, designated as p_i, within a given frame, designated as **i**. The underlying principle may be understood by considering the process of "lifting" the query pixel to three-dimensional space through the sampling of points along a ray, which are then "mapped." Three-dimensional points are mapped to a target frame **j** using bijections (T_i from local frame i to canonical space and T_j from canonical volume to local frame j, as it was explained in Equation 3.3) , which are then rendered through alpha compositing. Finally, the points are projected back to two-dimensional space to obtain a putative correspondence ((b) in Figure 3.5).

In particular, it is assumed that camera motion is subsumed by the local-canonical bijections T_i, a fixed, orthographic camera is employed. The ray at p_i can then be defined as $r_i(z) = o_i + zd$, where $o_i = [p_i, 0]$ and $d = [0, 0, 1]$. We sample K samples on the ray z_i^k, which are equivalent to appending a set of depth values $\{z_i^k\}_{k=1}^K$ to p_i. Despite not representing a physical camera ray, this approach captures the notion of multiple surfaces at each pixel and is sufficient for handling occlusion.

Subsequently, the densities and colours for these samples are obtained by mapping them to the canonical space and then querying the density network (GaborNet) F_θ. For illustrative purposes, the k-th sample x_i^k is taken as an example, whereby its density and colour can be expressed as:

$$(\sigma_k, c_k) = F_\theta(M_\theta(x_i^k; \psi_i)) \tag{3.6}$$

Additionally, each sample can be mapped from the initial ray to a corresponding 3D location x_j^k in frame j (Equation 3.3).

The correspondences $\hat{x}_j^k$ from all samples can now be aggregated to produce a single correspondence $\hat{x}_j$. This aggregation is analogous to the aggregation of the colours of sample points in NeRF, whereby alpha compositing is employed, with the alpha value for the k-th sample defined as $\alpha_k = 1 - \exp(-\sigma_k)$. The resulting $\hat{x}_j$ is then computed as:

$$\hat{x}_j = \sum_{k=1}^{K} T_k \alpha_k x_j^k \quad \text{where} \quad T_k = \prod_{l=1}^{k-1} (1 - \alpha_l) \tag{3.7}$$

A comparable process is employed to composite c^k in order to obtain the imagespace colour $\hat{C}_i$ for p_i. Subsequently, x_j is projected using our stationary orthographic camera model, thereby yielding the predicted 2D corresponding location $\hat{p}_j$ for the query location p_i.

3.3 Optimization Techniques

The Omnimotion optimisation process accepts as input a video sequence and a set of noisy correspondence predictions (derived from an existing method) as a guide, and produces a comprehensive, globally consistent motion estimate for the entire video. As previously stated, the pre-processing stage generates these flows and applies filtering and checks. Despite the filtering process, the resulting flow fields remain noisy and inconsistent. The objective of the optimisation method is to consolidate these noisy, incomplete pairwise motions into complete and accurate long-range motions.

3.3.1 Loss Function Optimization

During the learning stage, Omnimotion employs a variety of loss functions. The primary loss function is a flow loss, which is designed to minimise the Mean Absolute Error between the predicted flow $\hat{f}_{i\to j} = \hat{p}_j - p_i$ and the supervised input flow $f_{i\to j}$, which is derived at the pre-processing stage by the RAFT model and represents the optimised representation of the model itself:

$$\mathcal{L}_{\text{flo}} = \sum_{f_{i\to j} \in \omega_f} \|\hat{f}_{i\to j} - f_{i\to j}\|_1 \tag{3.8}$$

where ω_f is the set of all the filtered pairwise flows. The aforementioned loss can be attributed to the model's objective of achieving the lowest possible error rate in flow estimation, given that even minor discrepancies in the prediction of movement have the potential to impact the overall prediction. Furthermore, Omnimotion minimise

a photometric loss, defined as the Mean Squared Error between the predicted colour and the actual colour $\hat{C}_i$ and the observed colour C_i in the source video frame:

$$\mathcal{L}_{\text{pho}} = \sum_{(i,\mathbf{p})\in\omega_p} \|\hat{C}_i(p) - C_i(p)\|_2^2 \tag{3.9}$$

where ω_p is the set of all pixel locations over all frames. The application of MSE in this context can be elucidated by the observation that while the chromatic composition of an image remains unaltered, its visibility is contingent upon the prevailing light conditions within the scene. It can be reasonably assumed that the minor alterations will not have a significant impact on the ultimate outcome. Finally, in order to guarantee temporal coherence of the 3D motion estimated by M_θ, a regularisation term is applied which penalises large accelerations. Given a sampled 3D location xi in frame i, it is mapped to frames i-1 and i+1 using Equation 3.3, resulting in 3D points x_{i-1} and x_{i+1}, respectively. Subsequently, the 3D acceleration is minimised in accordance with the following equation:

$$\mathcal{L}_{\text{req}} = \sum_{(i,\mathbf{p})\in\omega_x} \|x_{i+1} + x_{i-1} - 2 * x_i\|_1 \tag{3.10}$$

In this context, ω_x represents the union of local 3D spaces for all frames. The final combined loss can be expressed as follows:

$$\mathcal{L} = \mathcal{L}_{\text{flo}} + \lambda_{\text{pho}}\mathcal{L}_{\text{pho}} + \lambda_{\text{req}}\mathcal{L}_{\text{req}} \tag{3.11}$$

The rationale behind this optimisation is to utilise the bijections to a single canonical volume G, photo consistency, and the inherent spatiotemporal smoothness afforded by the coordinate-based networks M_θ and F_θ to harmonise disparate pairwise flow and supplement absent content in the correspondence graphs.

Furthermore, the authors of Omnimotion put forth two loss functions, which were subsequently discussed in the appendix of the paper. One of these additional losses was part of the photometric loss function, $\mathcal{L}_{\text{pho}}$, which serves as an auxiliary loss term that oversees the relative colour between a pair of pixels in a given frame:

$$\mathcal{L}_{\text{pgrad}} = \|(\hat{C}_i(p_1) - \hat{C}_i(p_2)) - (C_i(p_1) - C_i(p_2))\|_1 \tag{3.12}$$

The term '$(\hat{C}_i(p_1) - \hat{C}_i(p_2))$' represents the difference in predicted colour between a pair of pixels, whereas '$(C_i(p_1) - C_i(p_2))$' denotes the corresponding difference between ground-truth observations. This loss bears resemblance to spatial smoothness regularisations or gradient losses that have been employed in prior work as supplements to pixel reconstruction losses. However, in contrast to these previous approaches, it is computed between pairs of randomly sampled pixels, which may be distant from one another, rather than between adjacent pixels. Furthermore, the same gradient loss is applied to the flow prediction. Author's findings indicate that the incorporation of these gradient losses enhances the spatial consistency of estimates and, more broadly, optimises the training process. Additionally, the authors utilise the distortion loss, as introduced in **mip-NeRF 360** [10], to mitigate the occurrence of floaters.

The fundamental principle of distortion loss can be observed in **Figure 3.10**. As with standard NeRF, a ray is characterised by peaks, indicating high density. The zero values between peaks represent transparency, and typically, the majority of the information conveyed by the ray is transparency, which is of limited consequence.

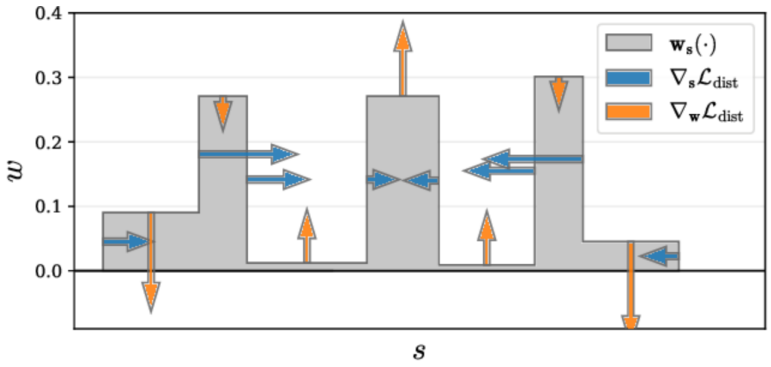


Figure 3.10: Distortion loss. [10]

Consequently, in the mip-NeRF 360 model (an enhancement of mip-NeRF [46], which is an advancement of classical NeRF), the concept of consolidating all peaks into a single global peak is introduced.

3.3.2 Hard-Mining Strategy

The comprehensive pairwise flow input facilitates the availability of a greater quantity of valuable motion data during the optimisation process. However, this method, particularly when combined with flow filtering, can result in an uneven distribution of motion samples in dynamic regions. It is often the case that rigid background areas retain a considerable number of reliable pairwise correspondences. In contrast, fast-moving or deforming foreground objects tend to have a smaller number of reliable correspondences after filtering, particularly between distant frames. This imbalance can result in the network prioritising dominant, simpler background motions while neglecting the more complex moving objects, which constitute a smaller portion of the supervisory signal. To address this challenge, the authors propose a straightforward approach for identifying and addressing difficult examples during training. Specifically, we periodically store flow predictions and generate error maps by calculating the Euclidean distance between the predicted and input flows. During optimization, they adjust sampling to prioritise regions with higher error rates. These error maps are computed from consecutive frames, where the supervisory optical flow is assumed to be the most accurate.

3.3.3 Implementation Details

As previously stated, the mapping network M_θ comprises six affine coupling layers. Positional encoding with four frequencies is applied to each layer's input coordinates prior to the computation of the scale and translation. A single two-layer MLP with 256 channels, implemented as a GaborNet, is used to compute the latent code ψ_i for each frame i. The input to this MLP is the time t_i. The dimensionality of the latent code ψ_i is 128. The canonical representation F_θ is also implemented as a GaborNet, but with three layers of 512 channels.

In the training stage, the default setting for batch sampling involved the selection of 256 correspondences from eight pairs of images, resulting in a total of 1024 correspondences. Additionally, 32 points were sampled along a ray.

4

Problem Statement and Methodology

The Omnimotion model exhibited exemplary performance and precision on existing datasets when benchmarked against other contemporary solutions. In order to verify the findings, we conducted an identical series of experiments and obtained comparable outcomes. A further noteworthy observation was the considerable memory usage and overall processing time required by the Omnimotion pipeline, which encompassed both the pre-processing stage and the primary computational phase. While the pre-processing stage primarily involved executing RAFT with filtering, which typically took between one and two hours, the main Omnimotion processing phase took longer. Furthermore, this stage required a considerable amount of graphics processing unit (GPU) memory to run efficiently. These factors raised concerns about the scalability of Omnimotion for more extensive, real-time applications where both processing time and memory allocation could become significant bottlenecks.

In the case of models such as Omnimotion, it is of great importance to have access to the weights of the original, unaltered versions, as this provides a baseline for comparison when evaluating the results of any subsequent updates. Following this initial trial, it became apparent that the primary objective in subsequent iterations was to optimise the Omnimotion model in terms of both time and memory consumption. Attaining this optimization would facilitate the testing of the model by users with less powerful hardware, enabling them to explore its potential in their own tasks. Furthermore, enhancing the model's efficiency could facilitate its adoption across a wider range of applications. But the most important part of such work is to keep the same level of accuracy so the improvements in time were not reached by significantly lowering the quality of the model.

Due to complexity of the Omnimotion system, a decision was taken to decompose the model into its constituent parts in order to ascertain which of the steps could be worked on and accelerated in order to improve the overall efficiency of the model.

4.1 Training Challenges

4.1.1 Loss Minimization

The initial insight gained from the original paper indicated that the Omnimotion model exhibited a notably high numbers of losses during the optimisation process. This issue has the potential to result in a significant increase in the model's overall complexity, which could subsequently lead to an increase in hardware demands and an extension of the time required for training. A comprehensive empirical assessment was conducted to ascertain the impact of each loss function on multiple benchmarks. It was established that the loss associated with the regularisation term (see Equation 3.10) has the same running time as the backward propagation function per iteration. This observation attracted our attention, given that the majority of loss functions apart from these two have a value below 0.01 second per iteration. Furthermore, backward propagation represents an essential component of the training process, and thus warranted closer examination. To address this problem, the loss functions related to the regularisation term was eliminated from the training phase, representing the first enhancement made to the model. As result, it was observed that this specific loss function (denoted as "smoothness loss") resulted in a notable increase in the computational burden, as it necessitated extensive calculations. Such regularisation term seems too computationally expensive for it needs, but following optimisation and return the term back might be possible. Overall, the original Omnimotion model was observed to run for approximately 10 hours on the dataset, whereas the version without the "no smoothness" loss function ran for less than 6 hours (see Table 7.1).

The benchmarks indicated a slight decline in quality, as evidenced by a reduction in occlusion accuracy from 89 to 87. However, not all evaluations exhibited a decrease, and the observed change was not always statistically significant (see the Table 6.1). Removing the loss function inevitably resulted in a reduction in the overall quality of the model. Nevertheless, in our case, we observed an improvement in computational efficiency. As previously discussed, this approach is particularly advantageous in scenarios where regular usage or production is a priority.

4.2 Pre-processing enhancement

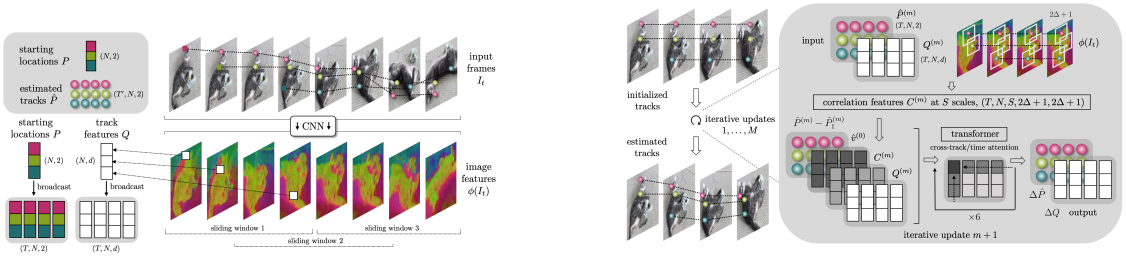
4.2.1 Substitution to the optical flow measurement method

A section of the Omnimotion publication was dedicated to the discussion of prospective enhancements to the existing model, a concept that has also prompted us to pursue these developments. The authors emphasise the fact that the RAFT can be switched to another model capable of producing the optical flow. In the course of our research, we identified a number of alternative models, including TAPIR [47], PIPs [11], CoTracker [12], and various adaptations of RAFT.

Our final selection was CoTracker , which proved to be the most dominant among the alternatives (Figure 4.1):

- PIPs demonstrated sub optimal performance in handling noise, as evidenced by the Omnimotion results.
- RAFT exhibited limitations in covering a wide range of scenarios without additional filtering or checks.
- TAPIR demonstrated proficiency in general tracking but exhibited a decline in overall quality when confronted with occlusions.

Figure 4.1: Quantitative comparison of the CoTracker with PIPs++ and TAPIR [11]



(a) CoTracker architecture.

(b) Visualization of one sliding window with M iterative updates.

Figure 4.2: CoTracker architecture. [12]

CoTracker is a transformer-based iterative algorithm, which can works online, designed to jointly track dense points across frames throughout a video sequence (see the pipline Figure 4.2). Based on identification of image features, track features and correlation features can achieves state-of-the-art performance on standard tracking benchmarks, frequently by a significant margin. It is capable of tracking through occlusions and when points move out of the field of view, even over hundreds of frames. CoTracker can handle highly diverse point configurations (target and support) and can track a large number of points simultaneously. The transformer's flexible architecture also allows for potential future extensions, such as integrating 3D reconstruction capabilities.

4.2.2 Update of the appearance check techniques

The superior performance of the CoTracker model enables the elimination of additional filtering stages that were previously required in the pre-processing phase of Omnimotion. Nonetheless, it remains essential to supply highly accurate data for the tracking stage, as even CoTracker may still be affected by incorrect pairings, as illustrated in Figure 3.2. Consequently, the appearance verification conducted by the DINO model continues to be valuable for eliminating erroneous pairs. However, rather than relying on the classical DINO model, we propose the adoption of the updated DINOv2 [48], which demonstrates substantial improvements in quality compared to its

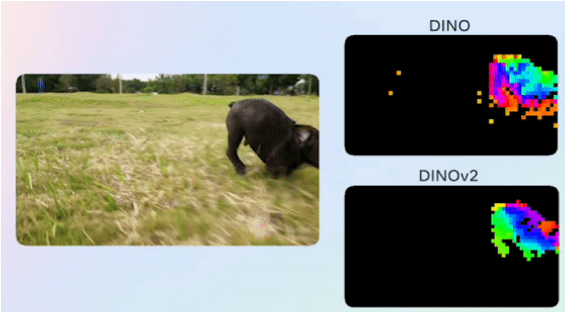


Figure 4.3: Quantitative comparison of the DINOv2 with DINO [13]

predecessor (see Figure 4.3).

Current pre-training methods demonstrates, particularly self-supervised approaches, that they can yield high-quality features when trained on sufficiently curated data from diverse sources. The authors revisit and integrate various techniques to enhance their pre-training process, focusing on scaling both the data and model sizes. Their technical contributions primarily address accelerating and stabilizing large-scale training. To improve data quality, they propose an automatic pipeline for creating a dedicated, diverse, and curated image dataset, contrasting with the typically uncurated data used in existing self-supervised methods. On the model front, they train a Vision Transformer with 1 billion parameters and subsequently distill it into a series of smaller models that outperform the best general-purpose features, such as OpenCLIP, across most benchmarks at both the image and pixel levels.

4.2.3 Hard-Mining Approach

The proposed solution, which involves the use of a specific hard-mining technique during the optimisation stage, does not address the issue of rapid changes within the thresholds. In the original implementation, the error map was retained. At each 20, 000 epoch, the region with the highest error (calculated the Euclidean distance between the predicted and input flows) was cashed and model with 50% chance selected between random points or such error map. During empirical analysis, the aforementioned solution was observed to be ineffective in addressing the aforementioned 'extreme' scenarios (i.e. sequences with significant movements). We have removed the random element in choosing the section to sample and have instead employed a simple summation of distances. This approach allows for the highest error section to be identified on each 20, 000 epoch.

4.3 Feature Embedding

Subsequently, the research proceeded to investigate the internal models of Omnimotion with the objective of elucidating their structural and functional characteristics. The analysis commenced with an examination of the network designated as "**color mlp**", which exhibited characteristics analogous to those of a $NeRF - like$ model. This specific network is configured to accept three-dimensional coordinates as input and subsequently return both the colour, represented in the RGB format, as well as the density. In its default configuration, the network comprises three layers, each containing 512 hidden units. This structure provides a foundation for the model's

ability to capture and process spatial information, allowing it to accurately render the colour and density outputs from the given $3D$ coordinates.

The considerable number of trainable parameters, which exceeded 100 million, prompted the investigation of methods to reduce the model's complexity. This included the reduction of the number of layers and the number of hidden units. A reduction in the number of parameters resulted in a notable decrease in both training time and memory usage, thereby reducing the computational resources required. The original $NeRF - family$ model is constrained by a lack of sufficient input data, which impairs its capacity to train effectively. This results in an unnecessarily complex structure with a vast number of parameters. In the course of our investigation and analysis of existing enhancements to *NeRF*, we identified an efficient solution through the use of embeddings. This approach had previously been successfully implemented in other work for time-series variables like ψ_i, where dimensionality was increased to enhance performance. By applying a similar embedding strategy, the model's ability to process input data can be improved without significantly increasing the computational burden, offering a promising path forward for optimising both accuracy and efficiency.

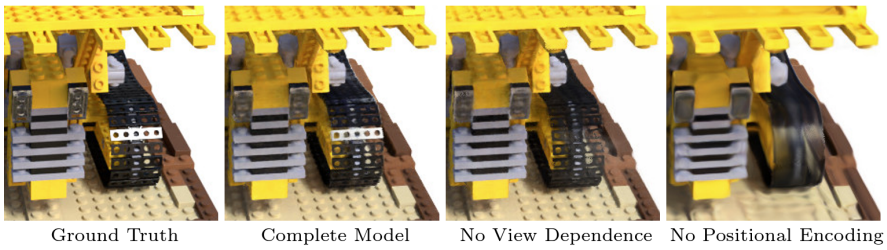


Figure 4.4: Example of usage of positional encoding [8]

In original implementation of the NeRF model there is a stage of positional encoding (see Figure 4.4), however it is missing in the Omnimotion implementation. In the course of our research, we encountered a hitherto unknown representation, designated a 'tri-plane', which was introduced in the context of TriPlaneNet [14] (see **Figure 4.5**). An Encoder for EG3D Inversion. This discovery offered valuable insight into the potential for more effective structuring of spatial representations. Furthermore, it was noted that other researchers have been engaged in efforts to enhance the encoding process for three-dimensional coordinates within the context of NeRF-related tasks. In one of the papers examined, significant developments in encoder technology were discussed, with particular emphasis on the ongoing trend of optimising the representation of coordinates. The majority of recent innovations appear to focus on modifying and refining these representations by introducing new, more efficient spaces to enhance the embedding process. This method of transforming the spatial coordinates into higher-dimensional embeddings offers a way to capture more complex relationships in 3D data, ultimately improving model performance in tasks like scene reconstruction and view synthesis. Such strategies are indicative of the continuous evolution of encoder architecture and the vital role they play in advancing 3D modelling and neural rendering techniques.

However, the original propose of the Tri-plane method for GANs models[49] made us refuce to implement such novel solution. Consequently, we elected to utilise a more conventional and established methodology, a

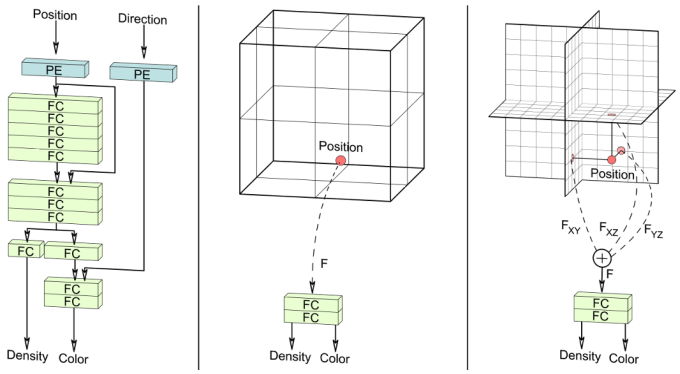


Figure 4.5: Coordinates representation: normal NeRF, Voxels, Tri-plane [14]

classical positional encoding approach. That approach entails the application of periodic functions (such as sin and cos) to each of the three-dimensional coordinates. As we stated it earlier, the objective of this method is to encode the spatial information in a manner that enables the network to more readily discern the intricate details of the scene while circumventing the inefficiencies associated with more complex or resource-intensive encoding schemes. By leveraging these periodic functions, we seek to achieve a balance between simplicity and effectiveness, maintaining efficient training times without compromising the quality of the model's predictions.

In each iteration of the **color mlp** algorithm, a set of sins a cons is applied to the three-dimensional input coordinates. This set comprises twenty-one functions per coordinate, resulting in a sixfold increase in the dimensionality of the input data, from three to sixty-six. Furthermore, parallel computing techniques have been incorporated into the position encoding process, with all three coordinates undergoing encoding simultaneously. The size of the encoded vector was selected as optimal due to the observation that a higher encoded vector size tends to result in a higher failure rate in prediction. This approach reduces the overall time required for training the colour mlp model. The modifications to the input data allow us to reduce the number of hidden states from 3 to 2. As a final configuration of the **color mlp** we have 3 layers with 256 hidden states. The overall time improvement can be see in the Table 7.1, but by applying encoding we decrease training time by 20% from previous encasement, while remain in the the same level of accuracy.

4.4 Model Freezing Techniques

Moreover, our investigation into the internal workings of neural networks has enabled us to gain a deeper understanding of their operational dynamics. As previously discussed, when a reversible neural network is active, the concept of time is represented using positional encoding, facilitated by an internal network named as '**feature mlp**'. The principal objective of this internal network is to identify patterns within the time representation of the latent code ψ_t. This pattern recognition process is repeated at each training epoch. In view of the fixed nature of this input, it was recognised that encoding and training over all 100, 000 epochs would not be practical or efficient. It was concluded that such a lengthy training process would not yield proportional improvements and would instead be an unnecessary expenditure of computational resources.

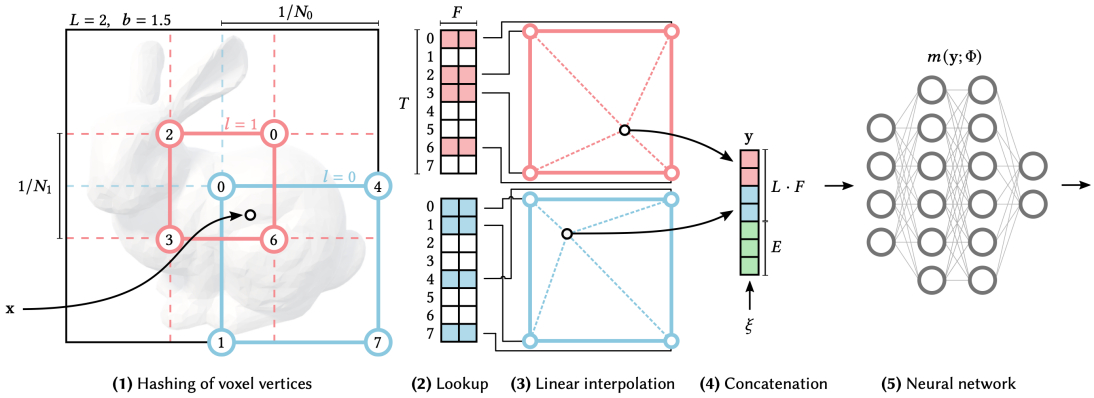


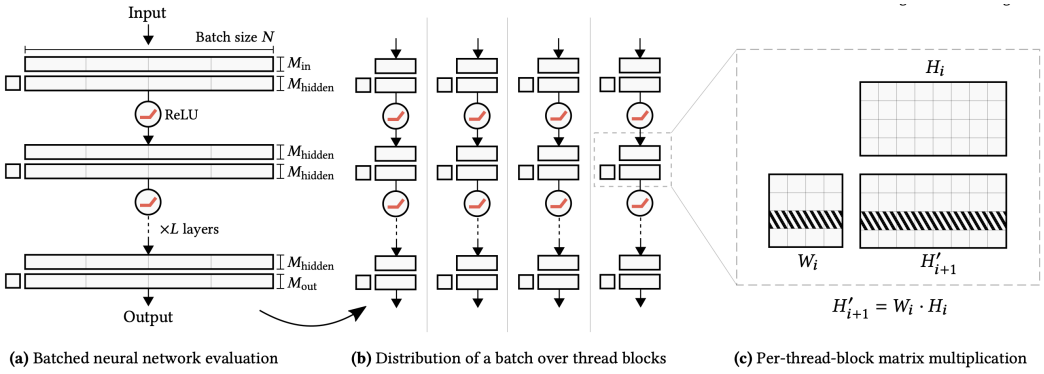
Figure 4.6: Multiresolution hash encoding in 2D. [15]

The empirical analysis demonstrated that following a number of epochs, the gradient of the 'feature mlp' was unable to achieve convergence. In contrast to the anticipated incremental improvement, the model displayed erratic behaviour, exhibiting arbitrary steps without any discernible increase in overall accuracy. In order to address this issue, a solution was proposed which involved the manual setting of a specific epoch threshold (our choice end up with 40000 epoch). Once the pre-established threshold has been exceeded, the 'feature mlp' will maintain the final gradient values of ψ, which are referred to as the '**global** ψ'. Instead of continuing to repeatedly invoke the network, the stored gradient values could be utilised for further computations. This modification has resulted in moderate enhancements in both runtime efficiency and memory usage, while preserving the accuracy of the model. The approach entails "freezing" a specific outcome of the 'feature mlp' training process and utilising it as a global value for subsequent operations. The streamlined methodology not only optimises resource consumption but also reduces the unnecessary computational load that would otherwise arise from repeatedly recalculating the gradients, thus enhancing the overall performance of the model without compromising its effectiveness.

4.5 Tiny CUDA Neural Networks

Further researchers into the 'feature MLP' network revealed that the quality of the network does not necessarily have a direct effect on the overall accuracy of the Omnimotion model. This finding prompted a more comprehensive investigation into potential solutions to address the issue of extending the network's runtime. Given that the network incorporates an additional layer of positional encoding, our research sought to identify a more efficient approach to this challenge. During previous researching process, we encountered a substantial enhancement to the NeRF framework. In particular, the authors of the paper entitled '*Instant Neural Graphics Primitives with a Multiresolution Hash Encoding*' from NVIDIA [15] introduced a flexible and efficient multiresolution hash encoding technique. This method offers a cost-effective solution for encoding input data, allowing for the streamlined parameterisation of neural graphics primitives using Convolutional Neural Networks. The proposed technique simplifies the process of managing complex graphics.

Figure 4.6 presents a detailed visual representation of the encoding process discussed earlier. To begin with, for a given input coordinate x, the neighboring voxels across L resolution levels are identified, and these are assigned



(a) Batched neural network evaluation **(b)** Distribution of a batch over thread blocks **(c)** Per-thread-block matrix multiplication

Figure 4.7: "Fully fused" multi-layer perceptron. [15]

specific indices based on their integer coordinates (1). Following this, for each of these corner indices, the corresponding F-dimensional feature vectors are retrieved from the hash tables θ_l and are then linearly interpolated (3), taking into account the relative position of x within the $l-th$ voxel. As part of the concatenation process (4), the results from all resolution levels are combined with the auxiliary inputs, denoted by the vector variable $\xi \in \mathbb{R}^E$, which depends on both the number of inputs, E, and the number of features, F. This results in the encoded MLP input, represented by the vector variable $y \in \mathbb{R}^{LF+E}$, which is then evaluated at the final stage (5). For the training phase, loss gradients are propagated backwards through the MLP (5), the concatenation process (4), and the linear interpolation (3), before being accumulated in the looked-up feature vectors to update the model parameters.

In other words, each input is divided into voxels of varying shapes, with each point x having a distinct coordinate within several voxels. For each voxel, the features of that coordinate are extracted and linearly interpolated. These features are then concatenated to form an input vector for the MLP, which encodes them.

In the course of our research, we identified an existing open-source implementation of the hash encoding method called '*Tiny CUDA*', which I integrated into my improvement. This resource provided a practical foundation that allowed for the application of the encoding technique, thus facilitating further experimentation and analysis within the framework of my project. Additionally, such implementation includes '*Real-time Neural Radiance Caching (NRC) for Path Tracing*' [50].

NRC is real-time neural radiance caching method for path-traced global illumination. The system is designed by the NVIDIA as well and able to handle fully dynamic scenes, and makes no assumptions about the lighting, geometry, and materials. The data-driven nature of this approach sidesteps many difficulties of caching algorithms, such as locating, interpolating, and updating cache points. Since pre-training neural networks to handle novel, dynamic scenes is a formidable generalization challenge, the authors do away with pre-training and instead achieve generalization via adaptation, i.e. they opt for training the radiance cache while rendering.

Description of such cashing can be explained as follows (see **Figure 4.7**). The evaluation of a multi-layer perceptron for a large batch of inputs , such as $N \approx 2^{21}$, for a 1920 × 1080 frame, typically alternates between weight-matrix multiplication and element-wise application of an activation function. In order to optimise the process, NVIDIA divides the batch into 128-element wide sections, with each section processed by a thread block. Given the narrow structure of MLP, with 64 neurons per layer, the weight matrices and intermediate activations (64×128) can be efficiently stored in shared memory. The aforementioned memory management techniques are

instrumental in facilitating the performance enhancements observed in our fully fused approach.

The matrix multiplication within each thread block transforms the output from layer H_i into the pre-activated layer H_{i+1}. This transformation is performed in blocks of 16×16 elements, which aligns with the capabilities of the hardware-accelerated half-precision matrix multiplier (TensorCore [51]). Each thread block is responsible for computing one block-row of size 16×128 of H'_{i+1}. This is achieved by first loading the corresponding striped weights, of size 16×64, from W_i into registers. Subsequently, the aforementioned weights are multiplied by all 64×16 block-columns of H_i, thereby ensuring that the weight matrix is fetched from global memory on a single occasion. This approach minimises memory access and accelerates processing. The results are stored in shared memory for rapid access in the subsequent computation steps.

Tiny-CUDA accepts vectors of dimensions 2, 3 and 4 as an input. Given that the time is a single digit, additional variants can be considered, such as:

$$(t_i, \log t_i + 1) \qquad (t_i, \sin t_i, \cos t_i) \tag{4.1}$$

5
Model Development

Our solutions and ideas on improvement were described in the previous chapter 4. Here we describe our final model and how it differs from the classical Omnimotion.

The pre-processing section was maintained in its original form, but concepts that were introduced in the previous chapter 4 will be discussed in detail in chapter 7.

In the main section, several modifications are made. Firstly, the classical NeRF method is adopted, with the addition of positional encoding for the inner NeRF-family network. We applied a parallel computing by applying the set of sin and cons. This modification allows for an increase in the input data, thereby enhancing the network's ability to learn the principles of the scene. Furthermore, this improvement enables the reduction in the size of the network, which can lead to improved memory usage.

Further enhancements were directed towards a Real-NVP-like model of the classical Omnimotion. In the original implementation, the feature learning of encoding the time representation ψ, was performed throughout the entire training stage. From the original implementation it is possible that the same time index could be taken in the same batch, so the feature mlp can be called several times for the same input, which is inefficient. As the data set is simply a series of positional numbers representing the frames in a video, and as the steps selected by gradient descent reached the plateau after several epochs, yet consume resources during the running process. In light of these considerations, we sought to accelerate the process by implementing our idea of keeping final gradient evaluation. As from empirical analysis the threshold after which the final result of feature mlp is take was set to the $40,000$ epochs and such final evalation is named globalφ. Globalφ remains as a constant and is being called at the further epochs.

Additionally to that, we implemented our proposed idea of Tiny-CUDA. Tiny-CUDA enables the input of higher-dimensional time (e.g. instead 1-D with single number of frame as an input we put 2-D vector: $(t, \log(t+1))$), which also enhances the learning process.

At the optimisation stage, we removed smoothness loss, which represents a regularisation term which proved to be inefficient in terms of quality-time. This has a negative effect on running time without significant improve-

ments in final accuracy. Additionally, we removed the random status of choosing the most problematic section of the data as a next sample to learn on. Instead, we maintained the model training on sections which remained with the highest sum of errors over the whole training period.

The new pipeline can be seen in Figure 5.1 and the code is at GitHub Repository.

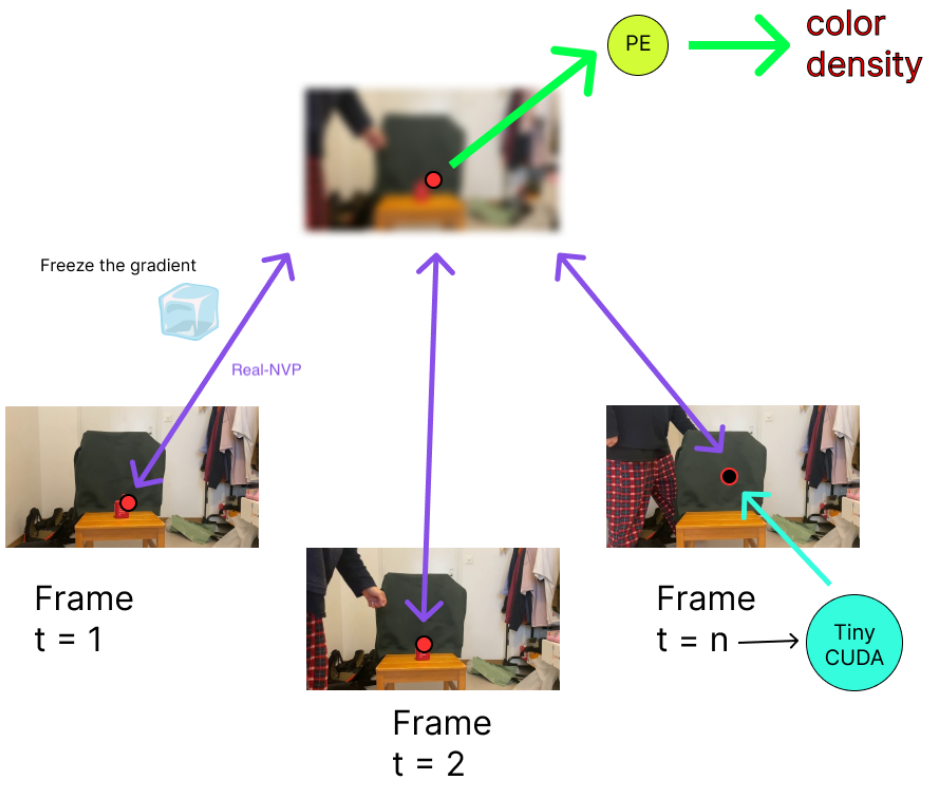


Figure 5.1: Our model overview.

6 Results

6.1 Dataset Description

The original Omnimotion model was trained using a variety of well-known datasets, including the The Densely Annotated Video Segmentation (DAVIS) dataset [52], which served as the primary dataset for our training process as well. By utilizing the DAVIS dataset as a foundational resource, we aimed to maintain consistency with the original model's methodology, thereby enabling a more accurate comparison of our results with those achieved by the original Omnimotion approach.

The DAVIS dataset is a highly esteemed resource, renowned for its superior quality and high-resolution video segmentation annotations. The dataset is available in two resolutions: The dataset is available in two resolutions: 480p and 1080p. It comprises 50 video sequences, totalling 3,455 frames, which have been meticulously annotated at the pixel level. Of these, 30 video sequences, containing 2,079 frames, have been designated for training purposes, while the remaining 20 sequences, with 1,376 frames, have been allocated for validation. This comprehensive annotation structure makes the DAVIS dataset an essential resource for evaluating video segmentation models.

Due to the constraints of our resources, we were obliged to limit the number of samples that we could compare to three, rather than the thirty that the Omnimotion team had used. In accordance with the weights that the Omnimotion team had provided, we selected samples that exhibited the lowest, highest and average prediction results, namely Black Swan, Dog and Shooting.

Furthermore, in consideration of the aforementioned rationale, the number of epochs utilized for training has been reduced in comparison to the weights provided by the authors, which were obtained following $200,000$ epochs of training. And because of special settings for final evaluation and number of epochs, we had to re-train the vanila version of the Omnimotion.

6.2 Metrics

As it was presented in the Omnimotion we kept the same metrics in use:

- $\langle \delta^x_{\text{avg}} \rangle \uparrow$: This assessment evaluates the average positional accuracy of visible points at five distinct thresholds: 1, 2, 4, 8, and 16 pixels. At each threshold, δ^x, the accuracy represents the proportion of points that are situated within δx pixels of their true position.
- AJ ↑: The Average Jaccard metric assesses both occlusion and positional accuracy using the same thresholds as those employed by δ^x_{avg}. It classifies predicted point locations into three categories: true positives, false positives, and false negatives. The AJ is defined as the ratio of true positives to the total number of points. Points that are correctly identified as visible are classified as true positives. Conversely, false positives are predicted visible points where the ground truth is either occluded or falls outside the defined threshold. Finally, false negatives are ground truth visible points that are either predicted as occluded or fall outside the threshold.
- OA ↑: The Occlusion Accuracy metric quantifies the precision of visibility and occlusion predictions for each frame.
- TC ↓: Temporal coherence is a metric that assesses the consistency of tracks over time. It does so by calculating the L2 distance between the acceleration of ground truth tracks and predicted tracks. The acceleration in question is determined by the flow difference between two consecutive frames for visible points.

6.3 Evaluation

The tests were conducted using an NVIDIA A100 GPU. Due to the constraints imposed by the utilisation of the aforementioned GPU, the original Omnimotion model was subjected to 100, 000 epochs of retraining, in contrast to the 200, 000 epochs outlined in the original paper.

The initial update to the Omnimotion model involved the removal of the regularisation term, which was designated as **"No smooth"** model. This modification resulted in a slight improvement in the quality of the new model, as evidenced by its superior performance in terms of time (see the Table 6.2). Overall quality observation is illustrated in Table 6.1.

Table 6.1: Performance comparison between "Omnimotion" and the models.

Method	AJ ↑	$\langle \delta^x_{\text{avg}} \rangle \uparrow$	OA ↑	TC ↓	Jaccard 1	Jaccard 2	Jaccard 4	Jaccard 8	Jaccard 16	Pts within 1	Pts within 2	Pts within 4	Pts within 8	Pts within 16
Dog														
Omnimotion	47.94	60.38	90.64	0.7517	8.61	21.28	42.84	79.84	87.13	16.19	35.88	61.47	91.35	96.97
No smooth	**52.94**	**65.71**	89.97	**0.7431**	**10.70**	**27.99**	**57.68**	**81.17**	**87.15**	**19.24**	**43.79**	**74.21**	**92.98**	**98.33**
Shooting														
Omnimotion	33.11	52.27	80.25	1.5851	6.64	15.78	29.25	48.02	65.83	13.97	30.77	51.30	74.03	91.29
No smooth	31.30	52.04	74.84	**1.6696**	6.61	15.57	27.85	44.82	61.67	**14.38**	**31.54**	51.08	72.69	90.52
Blackswan														
Omnimotion	68.58	80.83	98.53	0.1851	64.21	67.49	68.60	70.01	72.61	77.63	80.00	80.78	81.76	84.00
No smooth	**69.01**	**81.15**	98.53	**0.2121**	**66.29**	67.49	68.60	70.07	72.61	**79.14**	80.00	80.78	81.80	**84.04**

It is not yet possible to state with certainty that this term is entirely superfluous. This is because our tests were conducted on a sample of the entire dataset, and there may still be further tests which demonstrate that this term plays a significant role.

Following to **no smooth** version we tested the Omnimotion with encoding of 3D coordinates inside of NeRF MLP. As it was stated purpose of such enhance is time improving so the time comparison can be seen in the following table:

Method	Time	Average for 100 Epochs	Memory Usage
Omnimotion	10h30	0.32 sec	17.3 MiB
No-Smooth	6h20	0.22 sec	14.1 MiB
Encoded	5h50	0.20 sec	10.2 MiB

Table 6.2: Time&Memory usage of the Omnimotion, No-Smooth and Encoded model.

Although there has been a slight decline in accuracy, it remains a robust feature:

Table 6.3: Performance comparison between "Omnimotion" and "Encoded" models.

Method	AJ ↑	$\langle \delta^x_{avg} \rangle$ ↑	OA ↑	TC ↓	Jaccard 1	Jaccard 2	Jaccard 4	Jaccard 8	Jaccard 16	Pts within 1	Pts within 2	Pts within 4	Pts within 8	Pts within 16
Dog														
Omnimotion	47.94	60.38	90.64	0.7517	8.61	21.28	42.84	79.84	87.13	16.19	35.88	61.47	91.35	96.97
Encoded	**48.69**	**61.18**	**89.91**	**0.7566**	**7.96**	**20.57**	**47.26**	**79.76**	**87.89**	**15.06**	**34.94**	**65.81**	**91.73**	**98.37**
Shooting														
Omnimotion	33.11	52.27	80.25	1.5851	6.64	15.78	29.25	48.02	65.83	13.97	30.77	51.30	74.03	91.29
Encoded	29.93	48.15	70.98	**1.8391**	6.04	15.14	28.58	42.67	57.23	12.58	29.07	49.19	66.80	83.11
Blackswan														
Omnimotion	68.58	80.83	98.53	0.1851	64.21	67.49	68.60	70.01	72.61	77.63	80.00	80.78	81.76	84.00
Encoded	**68.06**	**79.83**	**98.74**	**0.2274**	**42.16**	**68.77**	**73.27**	**76.67**	**79.40**	**58.94**	**80.98**	**84.16**	**86.37**	**88.69**

Following to that we realised our idea of freezing final evaluation of the feature MLP which should give us winning in time. As on top of that we modify hard-mining to the sum version. Both these changes have to effect on time and quality as we can check in the following tables:

Table 6.4: Performance comparison between "Omnimotion" and "Freeze" models.

Method	AJ ↑	$\langle \delta^x_{avg} \rangle$ ↑	OA ↑	TC ↓	Jaccard 1	Jaccard 2	Jaccard 4	Jaccard 8	Jaccard 16	Pts within 1	Pts within 2	Pts within 4	Pts within 8	Pts within 16
Dog														
Omnimotion	47.94	60.38	90.64	0.7517	8.61	21.28	42.84	79.84	87.13	16.19	35.88	61.47	91.35	96.97
Freeze	51.88	61.83	91.73	0.7334	7.33	21.22	43.07	79.80	88.26	19.06	34.93	61.12	91.74	98.37
Shooting														
Omnimotion	33.11	52.27	80.25	1.5851	6.64	15.78	29.25	48.02	65.83	13.97	30.77	51.30	74.03	91.29
Freeze	29.81	46.66	70.97	2.0967	6.63	16.23	29.28	42.17	55.01	12.98	29.73	48.47	64.20	82.21
Blackswan														
Omnimotion	68.58	80.83	98.53	0.1851	64.21	67.49	68.60	70.01	72.61	77.63	80.00	80.78	81.76	84.00
Freeze	61.22	74.29	98.69	0.1900	63.03	68.50	69.73	70.01	72.89	76.82	80.78	81.97	83.97	86.54

As on top improvement we check how change the situation the TCNN implementation to the feature mlp. Overall here the all models average comparison:

Table 6.5: Performance comparison of models across videos: "Omnimotion", "No smooth", "Encoding", "Freeze" and "TCNN".

Video	AJ ↑	$\langle \partial^x_{avg} \rangle$ ↑	OA ↑	TC ↓	Time	Memory
Blackswan						
Omnimotion	68.58	80.83	98.53	0.18	10h30	17.3 MiB
No Smooth	69.01	81.15	98.53	0.21	6h20	14.1 MiB
Encoding	68.05	79.83	98.74	0.23	5h50	10.2 MiB
Freeze	61.23	74.29	98.69	0.19	5h30	10.2 MiB
Freeze + Hard	62.64	75.67	98.84	0.22	5h30	10.2 MiB
TCNN	66.74	79.05	96.83	0.24	**5h12**	8.9 MiB
TCNN + Hard	66.93	79.08	96.86	0.24	**5h12**	8.9 MiB
Dog						
Omnimotion	47.94	60.38	90.64	0.75	10h30	17.3 MiB
No Smooth	52.94	65.71	89.97	0.74	6h20	14.1 MiB
Encoding	48.69	61.18	89.91	0.76	5h50	10.2 MiB
Freeze	51.88	63.92	91.73	0.73	5h30	10.2 MiB
Freeze + Hard	48.96	62.15	89.99	0.75	5h30	10.2 MiB
TCNN	43.81	56.33	86.26	0.74	**5h12**	8.9 MiB
TCNN + Hard	46.66	58.96	87.53	0.74	**5h12**	8.9 MiB
Shooting						
Omnimotion	33.11	52.27	80.25	1.58	10h30	19.4 MiB
No Smooth	31.30	52.04	74.84	1.67	6h20	14.1 MiB
Encoding	29.93	48.15	70.96	1.84	5h50	10.2 MiB
Freeze	29.81	46.66	72.69	2.10	5h30	10.2 MiB
Freeze + Hard	28.04	45.38	76.93	3.72	5h30	10.2 MiB
TCNN	28.88	47.57	80.33	1.56	**5h12**	8.9 MiB
TCNN + Hard	28.59	46.24	80.15	1.77	**5h12**	8.9 MiB

7 Discussions

We proposed a new optimization method designed to improve the existing dense motion and pixel tracking model. The objective of our approach was to enhance the computational efficiency of the model, which resulted in a notable reduction in both processing time and memory usage. This improvement not only accelerated the processing speed but also made it feasible for the majority of users to run and test the model on their personal machines, which previously might have been constrained by hardware limitations.

Furthermore, we ensured that the overall quality of the original solution remained intact. Our optimisation preserved the moderate level of accuracy and effectiveness of the model, providing users with a more accessible yet high-performing tool. By striking this balance between efficiency and quality, our method facilitates broader adoption and usability of the dense motion/pixel-tracking model across a wider range of computational environments.

However, we made several compromises during our research. As it can been seen from the chapter 6 while winning in a running time we have loss in the overall quality. Our loss is not significant, but yet visible and during our tests we notice several points, that should be highlighted.

Regularisation term

As previously stated in chapter 4, the removal of the regularisation term resulted in a notable reduction in running time, although this was accompanied by a slight decline in quality. The results demonstrate that the overall quality can be enhanced through the removal of the term. It is important to note, however, that only one sample from the training dataset exhibits such pronounced movements, which are in line with the objective of the term. Consequently, it is unclear whether the removal of this sample is an appropriate approach. We maintain this modification due to our primary objective of accelerating the overall runtime. However, if the objective is to enhance the quality, we hypothesise that the loss should be retained or potentially combined with another loss function, as the presence of numerous losses introduces unnecessary complexity.

Embedding

The embedding implementation afforded us the opportunity to reduce the complexity of the inner MLP, which functions as a NeRF-like model. However, such a modification presents certain challenges.

At each 20, 000 epoch, the Omnimotion model also generated a visual status of the current training. It was observed that embedded visualisation has the capacity to retain details, and in such a model, tracking is conducted over a certain area, rather than precise objects. This is evident in the silhouette of the object in the video, which is more visible and precise (see Figure 7.1).

(a) Embeded color.

(b) Omnimotion color.

Figure 7.1: Color comparison.

As can be observed in the reference **??** there has been a decline in overall quality when compared to the original model. This reduction in quality can be attributed to the selection of the embedding mechanism, which was based on a straightforward positional encoding with periodic functions. However, the most expedient method was not implemented due to concerns that it might result in an even more pronounced deterioration in quality in the future. Which later in our work was proofed wrong, since new paper were published (discussed later).

Freeze

The primary objective of maintaining the final outcome of network operations is to accelerate the overall model. It is evident that there is a possibility of compromising the quality, however, our objective was to emphasise that the inner MLP was originally designed solely for the purpose of encoding time and learning from that encoding. It is unnecessary to continue running the model with limited input data (number of frames) and call the training at each epoch. As can be seen in the table, we achieved a slight improvement in time, but also lost some quality. One possible approach to finding a combination of saving quality while also removing unneeded runs is to keep track of the gradient and freeze the last value once the threshold is reached.

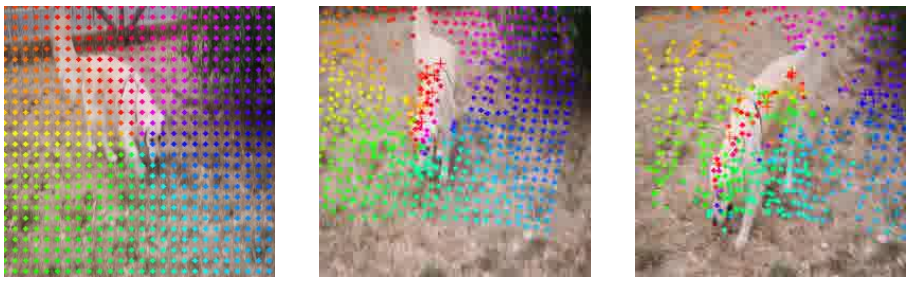
TCNN

Conversely, TCNN demonstrates superior output quality, suggesting it may offer a more effective resolution to the freezing issue, potentially leading to even better overall performance in terms of quality. Furthermore, we hypothesize that this approach could also be applied to encode 3D coordinates for the inner NeRF-family model. However, this hypothesis remains to be thoroughly investigated and validated through additional research and experimentation.

Cotracker and DINOv2

The CoTracker model was implemented as a potential substitute, yet the resulting tests yielded only modest outcomes. In order to maintain the same level of tracking (dense track as previously stated by Omnimotion), the dense mode of CoTracker is required for each frame of the video. This presents an optimization problem, as it takes the same amount of time as a pre-processing stage. Additionally, during the course of the tests, we encountered the issue of reaching the limits of our GPU. Notwithstanding these circumstances, the CoTracker model exhibited high-quality tracking while maintaining occlusion information without the necessity for additional filtering, as was the case with the RAFT model.

RAFT



CoTracker

Figure 7.2: Comparison of RAFT-based and CoTracker-based Omnimotion.

The numerical evaluation can been seen in the following table:

Table 7.1: Performance comparison of models across videos: "Omnimotion" with RAFT and "TCNN" with CoTracker on 40, 000 epochs.

Video	AJ ↑	$\langle \delta^x_{\text{avg}} \rangle$ ↑	OA ↑	TC ↓
Dog				
Omnimotion + RAFT	47.88	61.40	90.54	0.75
TCNN + CoTracker	8.71	16.74	88.92	2.48

Nevertheless, it is our supposition that with further analysis and the resolution of the aforementioned optimisation issues, the CoTracker model would significantly improve the quality of the Omniomotion results.

Furthermore, we would like to state that, in our tests, the usage of the DINOv2 model did not indicate substantial improvements. Consequently, we have decided to continue utilising the classical DINO implementation. However, it seems reasonable to maintain such filtering by employing DINO as an only check for the CoTracker results.

7.0.1 Other solution

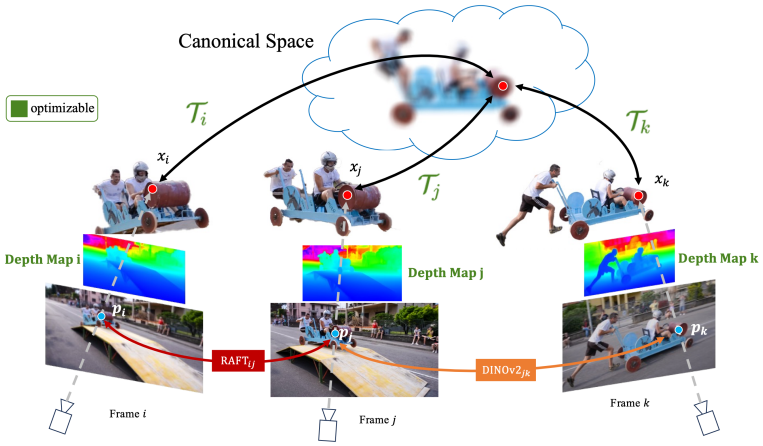


Figure 7.3: CaDeX++ Method Overview. [16]

During the course of our research, we were pleased to discover a recently developed model, entitled 'Track Everything Everywhere Fast and Robustly' [16]. The authors attempted to implement similar ideas to ours and achieved a state-of-the-art result. Key difference of that model with original Omnimotion and ours the fact that they changed the Real-NVP model with their custom modification of the CaDeX model [53]. CaDeX is a versatile approach for reconstructing dynamic surfaces and establishing correspondences. They named their novel model CaDeX++ and key difference of this model with Real-NVP is different representation inside of affine layers (see the Figure 7.4). The results demonstrate that the overall training speed is 10 times faster without a decline in accuracy, and there are even improvements in several tests. In addition to modifying the INN model, the authors

proposed the incorporation of supplementary data at the pre-processing stage. This involved the utilisation of depth measurements derived from the Zoe Depth model [54] and modifying the initial DINO model with it successor DINOv2 Figure 7.3. The incorporation of detailed depth information enhances the input data, providing the model with richer scene details and contributing to the optimisation process. In light of the various challenges encountered with the original NeRF implementation, the authors elected to replace it entirely with a depth estimation approach, which offers improvements in both efficiency and quality. Their findings indicate that this new method not only performs better in practice but also yields more robust results. However it is yet impossible to compare their approach to ours due to fact, that original code of CaDeX++ model is not yet published.

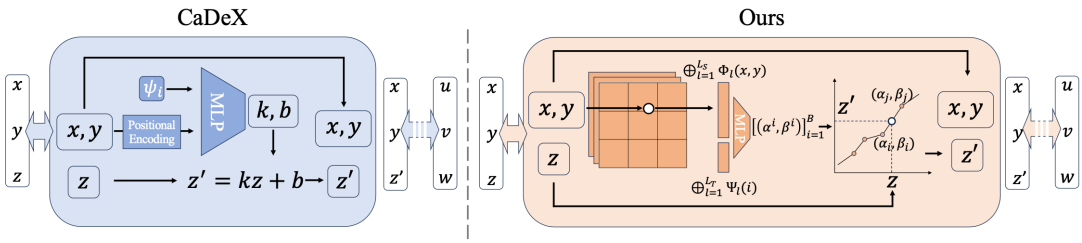


Figure 7.4: Architecture of CaDeX++. [16]

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2016.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] First Principles of Computer Vision. (2023) Lucas-kanade method | optical flow. Accessed: 2024-09-01. [Online]. Available: https://www.youtube.com/watch?v=6wMoHgpVUn8

[4] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," 2020. [Online]. Available: https://arxiv.org/abs/2003.12039

[5] Q. Wang, Y.-Y. Chang, R. Cai, Z. Li, B. Hariharan, A. Holynski, and N. Snavely, "Tracking everything everywhere all at once," in *International Conference on Computer Vision*, 2023.

[6] Meta AI, "Dino: Paws computer vision with self-supervised transformers and 10x more efficient training," 2024, accessed: 2024-09-10. [Online]. Available: https://ai.meta.com/blog/dino-paws-computer-vision-with-self-supervised-transformers-and-10x-more-efficient-training/

[7] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," 2021. [Online]. Available: https://arxiv.org/abs/2104.14294

[8] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," 2020. [Online]. Available: https://arxiv.org/abs/2003.08934

[9] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen, "Invertible residual networks," https://www.cs.toronto.edu/~duvenaud/talks/iresnet-slides.pdf, 2019, accessed: 2024-09-01.

[10] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," 2022. [Online]. Available: https://arxiv.org/abs/2111.12077

[11] A. W. Harley, Z. Fang, and K. Fragkiadaki, "Particle video revisited: Tracking through occlusions using point trajectories," 2022. [Online]. Available: https://arxiv.org/abs/2204.04153

[12] N. Karaev, I. Rocco, B. Graham, N. Neverova, A. Vedaldi, and C. Rupprecht, "Cotracker: It is better to track together," 2023. [Online]. Available: https://arxiv.org/abs/2307.07635

[13] Meta Demo Lab, "Dinov2: Learning robust visual features without supervision," 2024, accessed: 2024-09-09. [Online]. Available: https://dinov2.metademolab.com

[14] A. R. Bhattarai, M. Nießner, and A. Sevastopolsky, "Triplanenet: An encoder for eg3d inversion," 2023. [Online]. Available: https://arxiv.org/abs/2303.13497

[15] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, p. 1–15, Jul. 2022. [Online]. Available: http://dx.doi.org/10.1145/3528223.3530127

[16] Y. Song, J. Lei, Z. Wang, L. Liu, and K. Daniilidis, "Track everything everywhere fast and robustly," 2024. [Online]. Available: https://arxiv.org/abs/2403.17931

[17] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, 1943.

[18] J. Mira and F. Sandoval, Eds., *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7-9, 1995: Proceedings*. Springer, 1995.

[19] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2019. [Online]. Available: https://arxiv.org/abs/1803.08375

[20] J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C.-H. Lee, "On mean absolute error for deep neural network based vector-to-vector regression," *IEEE Signal Processing Letters*, vol. 27, p. 1485–1489, 2020. [Online]. Available: http://dx.doi.org/10.1109/LSP.2020.3016837

[21] ——, "On mean absolute error for deep neural network based vector-to-vector regression," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 5, pp. 1670–1676, 2020.

[22] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," 2018. [Online]. Available: https://arxiv.org/abs/1805.07836

[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, dec 1989. [Online]. Available: http://dx.doi.org/10.1162/neco.1989.1.4.541

[24] P. Golik, A. Tarkhov, R. Schlüter, and H. Ney, "Convolutional neural networks for acoustic modeling of raw time signal in lvcsr," in *Proceedings of INTERSPEECH*, 2015.

[25] W. Dai, C. Dai, S. Qu, J. Li, and S. Das, "Very deep convolutional neural networks for raw waveforms," *CoRR*, vol. abs/1610.00087, 2016. [Online]. Available: https://arxiv.org/abs/1610.00087

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[27] E. U. Henry, O. Emebob, and C. A. Omonhinmin, "Vision transformers in medical imaging: A review," 2022. [Online]. Available: https://arxiv.org/abs/2211.10043

[28] Z. Liu, Q. Lv, Z. Yang, Y. Li, C. H. Lee, and L. Shen, "Recent progress in transformer-based medical image analysis," *Computers in Biology and Medicine*, vol. 164, p. 107268, Sep. 2023. [Online]. Available: http://dx.doi.org/10.1016/j.compbiomed.2023.107268

[29] L. Guerra, L. Xu, P. Mozharovskyi, P. Bellavista, T. Chapuis, G. Duc, and V.-T. Nguyen, "Ai-driven intrusion detection systems (ids) on the road dataset: A comparative analysis for automotive controller area network (can)," 2024. [Online]. Available: https://arxiv.org/abs/2408.17235

[30] X. Wang, J. Peng, S. Zhang, B. Chen, Y. Wang, and Y. Guo, "A survey of face recognition," 2022. [Online]. Available: https://arxiv.org/abs/2212.13038

[31] J. Chen, W. Su, and Z. Wang, "Crowd counting with crowd attention convolutional neural network," *Neurocomputing*, vol. 382, p. 210–220, Mar. 2020. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2019.11.064

[32] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the Imaging Understanding Workshop*, 1981, pp. 121–130.

[33] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2462–2470.

[34] M. Hofinger, S. Bulo', L. Porzi, A. Knapitsch, and P. Kontschieder, "Improving optical flow on a pyramidal level," in *European Conference on Computer Vision (ECCV)*, 2020.

[35] T. Hui, X. Tang, and C. Change Loy, "Liteflownet: A lightweight convolutional neural network for optical flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8981–8989.

[36] D. Sun, X. Yang, M. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8934–8943.

[37] G. Yang and D. Ramanan, "Volumetric correspondence networks for optical flow," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 793–803.

[38] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: https://arxiv.org/abs/1412.3555

[39] S. W. Yoon, J. Seo, and J. Moon, "Tapnet: Neural network augmented with task-adaptive projection for few-shot learning," 2019. [Online]. Available: https://arxiv.org/abs/1905.06549

[40] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[41] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[42] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[43] S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe, "Bayesflow: Learning complex stochastic models with invertible neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[44] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," 2017. [Online]. Available: https://arxiv.org/abs/1605.08803

[45] A. Alekseev and A. Bobe, "Gabornet: Gabor filters with learnable parameters in deep convolutional neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1904.13204

[46] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," 2021. [Online]. Available: https://arxiv.org/abs/2103.13415

[47] C. Doersch, Y. Yang, M. Vecerik, D. Gokay, A. Gupta, Y. Aytar, J. Carreira, and A. Zisserman, "Tapir: Tracking any point with per-frame initialization and temporal refinement," 2023. [Online]. Available: https://arxiv.org/abs/2306.08637

[48] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," 2024. [Online]. Available: https://arxiv.org/abs/2304.07193

[49] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[50] T. Müller, F. Rousselle, J. Novák, and A. Keller, "Real-time neural radiance caching for path tracing," *ACM Transactions on Graphics*, vol. 40, no. 4, p. 1–16, Jul. 2021. [Online]. Available: http://dx.doi.org/10.1145/3450626.3459812

[51] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance amp; precision," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, May 2018. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2018.00091

[52] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbelaez, A. Sorkine-Hornung, and L. Van Gool, "The 2017 davis challenge on video object segmentation," *arXiv preprint arXiv:1704.00675*, 2017.

[53] J. Lei and K. Daniilidis, "Cadex: Learning canonical deformation coordinate space for dynamic surface representation via neural homeomorphism," 2022. [Online]. Available: https://arxiv.org/abs/2203.16529

[54] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller, "Zoedepth: Zero-shot transfer by combining relative and metric depth," 2023. [Online]. Available: https://arxiv.org/abs/2302.12288

Acknowledgments

Firstly, I want to express my deepest gratitude to my small but incredibly supportive family. My mother, who stood by me no matter the challenges we faced, was my rock—providing unwavering physical, financial, and emotional support. She believed in me even when I doubted myself. I also want to acknowledge my grandmother, who, I am sure, watched over me and helped in unseen ways from beyond. A special thank you to Judith, who has been like a second grandmother, always believing in me and helping me with all her strength and will. To Aunt Gulnara, thank you for always being on my side and keeping your trust in me. My heartfelt thanks go to my cats Stepan and Marsik, who brought comfort, love, and patience during one of the hardest periods of my life. Thank to my Aunt Sholpan and cousin Kamilla. Without these individuals, I would not have reached this point, and I am forever grateful.

I would like to thank the friends I made during my time in Moscow and those who have remained close even after I moved to Padua. My oldest friend, Dmitry, whom I've known since 4th grade — thank you for being such an important part of my life. To my IOP group —Daniil, Victor, Valeria, Vladimir, Gleb, Bulat, Irina, Stas, Aleksandr, and Denis—thank you for making life fun and supportive, even from a distance. Thanks to Olya, Ksenia, Daria, and Viktoria for being there with messages, calls, and attention that made a difference. My closest friend, Daria, deserves a heartfelt thank you for being a constant source of support, always ready for deep conversations that helped me through tough times. Special thanks to my friend Andrei, whose inspiration and assistance during my thesis work were invaluable in achieving the results I did. I would also like to thank Aleksandra, Kamilla, Ksenia, Julia, Anastasia, and many others who have accompanied me on my life journey.

I am incredibly grateful for the new friends I have found here in Padua, who have encouraged me to embrace Italian culture even more. A special thank you to Dan, Bianca, Angela, Elisabetta, Giorgio, Marco, Beatrice, Laura, Chiara, and Bibiana for being my new family in Padua. To Francesca and Jlenia, thank you for your close support, especially during difficult times. Margherita, thank you for being so open and kind, for introducing me to other parts of Italy, and for showing me the beauty of Arezzo. Valeria and Alina, thank you for being my connection to Kazakhstan during this experience. To Ljubica, thank you for embracing my Eastern roots and for your kindness and support from the moment we met. Daniel, thank you for being a wonderful student and making university life more adventurous. Special thanks to my Italian, French, and German professor, Katya, for helping me not only with language challenges but also guiding me in life.

I am deeply thankful to the University of Padua for giving me the incredible opportunity to participate in the SEMP program and study for 10 months at the University of Bern. This life-changing experience reaffirmed my love for Switzerland, where I met wonderful people and explored breathtaking places. I would like to thank my exchange friends —Hannah, Jones, Robin, Bara, Koen, Sonya, Daria, Polina, Lucrezia and Michelle — for their friendship and support. Special thanks to Sarah and Chiara for being both study and party companions during the main phase of my thesis, and to Michalina for teaching me the importance of sharing and filling my life with positivity. Saule, thank you for the opportunity to work as your assistant and for teaching me the challenges of

being a teacher. To Irakli, Filippo, Aram and Llukman, thank you for your life and academic advice, and for helping me approach my thesis more effectively. Finally, a big thank you to my wonderfully kind and welcoming flatmates, Nico and Elena, for filling our time together with warmth and laughter.

Most importantly, I want to acknowledge the professors who guided me throughout this academic journey. My first thanks go to my physics teacher, Anna Andreevna, for recognizing my academic potential, and to her brother for helping me navigate life in Veneto. I am grateful to Prof. Anastasia Vladimirovna for inspiring me during my bachelor's degree, and to Prof. Olga Vladimirovna for introducing me to the world of machine learning as my first supervisor. Special thanks to my current supervisors, Prof. Lamberto Ballan and Prof. Paolo Favaro — your guidance and patience during my thesis work have been instrumental in my success, and I am honored to have worked with you.

Thank you to Italy and the city of Padua for welcoming me as an international student and teaching me valuable life lessons, despite the bureaucratic challenges. Lastly, I am grateful to my first university, the Higher School of Economics in Moscow, for shaping me into the student and professional I am today, and to the University of Padua for furthering that growth and helping me achieve my goals.

PS. Thanks to my therapist, Oriola Ndreu, and Philipp Schmutz for helping me shape my problems and find solutions independently. To my spiritual guides, Mac Miller and Zach Braff, you shaped my personality and taught me how to value kindness and love in others.