TECHNISCHE UNIVERSITEIT DELFT

COMPUTER VISION LAB

MSc in CS

MASTER'S THESIS

Analysis Tool for Optical Flow Models

Thesis Advisor
Prof. Jan van Gemert
Computer Vision Lab
Daily co-supervisor
Sander Gielisse
Computer Vision Lab

Petter Reijalt

2025, July

Contents

1 Introduction

Optical flow refers to the computer vision task of estimating the apparent motion of objects in a scene by projecting 3D movements onto a 2D image plane. This is commonly formulated as a per-pixel correspondence problem between consecutive video frames. Accurately predicting optical flow remains a significant challenge due to various sources of ambiguity, such as lighting changes, occlusions, and reflective surfaces.

Historically, progress in optical flow has been closely tied to the development of standardized benchmark datasets. As real-life optical flow ground truth is hard to come by, often synthetic datasets are used, which provide dense optical flow ground truth. These evaluation benchmarks often solely report on basic metrics such as EPE, and depending on the benchmark occlusion and/or displacement statistics.

However, understanding the specific areas in which modern models improve requires a modular evaluation framework, one that enables controlled variation of individual scene parameters. This research contributes a dataset generation tool that allows for such controlled variation, including the ability to adjust lighting conditions, displacement magnitudes, and surface textures. Additionally, it supports learning parameter configurations that produce datasets with targeted displacement distributions and global image characteristics such as average pixel brightness.

Section 2 provides supplementary material aimed at readers who are less familiar with the domain, offering foundational concepts and current developments in optical flow research. In Section 3, we introduce the scientific paper that describes the methodology and contributions of this work to the field of optical flow.

2 Supplementary Material

2.1 Optical Flow

Optical flow is a classical computer vision task in which the goal is to estimate a 2-dimensional velocity map, which is a projection of the movement of 3-dimensional objects onto the image plane. [1] Motion analysis between consecutive frames can also constitute in a 3-dimensional motion field which is called scene flow [2]. Some of the applications of optical flow include, but not limited to, are 3d-action recognition, moving object detection, image interpolation, image superresolution and video segmantation [3].

2.1.1 Average Endpoint Error (AEPE)

When evaluating a model on a certain dataset, often the AEPE is reported. This is the mean per-pixel error, similar to the euclidean distance between the vectors:

$$AEPE = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(\mu_{st,i} - \mu_{gnd,i})^2 + (v_{st,i} - v_{gnd,i})^2},$$

where $\mu_{st,i}$ and $v_{st,i}$ are the estimated horizontal and vertical displacement vectors and $\mu_{gnd,i}$ and $v_{gnd,i}$ the ground truth vectors [4].

2.1.2 Angular error

$$\psi_E = \arccos\left(\vec{v}_c \cdot \vec{v}_e\right)$$

where $\vec{v} = \frac{1}{\sqrt{u^2+v^2+1}}$ denotes the normalised optical flow vector in *space-time*, incorporating both spatial motion components u and v, along with a temporal dimension. In this formulation, $\vec{v}_c$ and $\vec{v}_e$ represent the ground truth and estimated optical flow vectors, respectively. The metric ψ_E quantifies the *angular error*, which captures the difference in *direction* between the estimated and true pixel motion vectors, regardless of their magnitude. This angular error was historically used as the primary evaluation metric in early benchmarks such as the Yosemite sequence [5]. However, in contemporary optical flow research,

the *average AEPE* has become the more widely adopted metric, as it accounts for both the magnitude and direction of flow discrepancies, providing a more comprehensive assessment of estimation accuracy.

2.2 Classical Models

2.2.1 Horn-Schuck

One method to estimate optical flow is the Horn-Schuck method. This method makes the assumption that the brightness of an image point stays the same between consecutive frames:

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t)$$

It also assumes that the displacement and timestep are small:

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t$$

This can be rewritten as the brightness constancy equation:

$$\mathcal{E}_b = I_x u + I_y v + I_t,$$

where u and v are the horizontal and vertical displacement respectively and E_x, E_y and E_t are the change in brightness with respect to x, y, and t respectively.

However this is an underconstrained problem, as we have 2 variables and only one formula. Hence they introduced the smoothness constraint, with the underlying assumption that neighbouring pixels have a similar optical flow:

$$\mathcal{E}_c^2 = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2.$$

The goal is to minimise both constraints and this can be achieved by introducing an α^2 operator, which weighs the two errors and creates a combined error which can be minimised:

$$\mathcal{E}^2 = \iint \left(\alpha^2 \mathcal{E}_c^2 + \mathcal{E}_b^2\right) \, dx \, dy.$$

Horn-Schunck is a global approach and hence tries to minimise the error above for all pixels. This happens iteratively [6].

2.2.2 Lucas-Kanade

The Lucas-Kanade method is a local approach to solving the optical flow problem. It uses the same brightness constancy constraint as the Horn-Schunck method, but adds another constraint in the fact that within a patch the flow is assumed to be the same. First the brightness constancy equation is rewritten as follows:

$$I_x u + I_y v = -I_t,$$

Assuming a 5x5 patch in which the optical flow should stay the same results in the following 25 equations:

$$\begin{aligned} I_x(\mathbf{p}_1)u + I_y(\mathbf{p}_1)v &= -I_t(\mathbf{p}_1) \\ I_x(\mathbf{p}_2)u + I_y(\mathbf{p}_2)v &= -I_t(\mathbf{p}_2) \\ &\vdots \\ I_x(\mathbf{p}_{25})u + I_y(\mathbf{p}_{25})v &= -I_t(\mathbf{p}_{25}) \end{aligned}$$

In matrix form, this looks like, the following, with unknowns u and v which can be solved using the least-squares approximation:

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

This leads to a sparse optical flow representation as the optical flow is considered the same within a patch [7].

2.2.3 Coarse-to-fine pyramids

The methods above struggle with larger displacements, as the brightness constancy equation only holds for smaller displacements as it is based on an approximation of the taylor series. Coarse-to-fine pyramids lower the resolution of the two images to the point that every displacement is smaller than 1 pixel, and the brightness constancy equation holds again. This optical flow, obtained by for example the Lucas-Kanade method is then applied, to a higher resolution image, resulting in a warped image. This warped image can then be used to compute a new optical flow together with the higher resolution t+1 image. This results in smaller displacements which allows for the use of the brightness equation again. This new optical flow is added to the previous optical flow and used again to warp a higher-resolution image. This is done iteratively until the original resolution is reached again [8].

2.2.4 Median filtering

In [9] the creators start off with a formulation of the horn-schuck method as follows:

$$E(\mathbf{u},\mathbf{v}) = \sum_{i,j} \Big\{ \rho_D \left(I_1(i,j) - I_2(i+u_{i,j}, j+v_{i,j}) \right) + \lambda \big[\rho_S(u_{i,j} - u_{i+1,j}) + \rho_S(u_{i,j} - u_{i,j+1}) + \rho_S(v_{i,j} - v_{i+1,j}) + \rho_S(v_{i,j} - v_{i,j+1}) \big] \Big\}$$

Where the $\rho_D + \rho_S$ represent the penalty function for the brightness and the smoothness constraint respectively. They found that using median filtering reduced the End-Point-Error but increased the energy function. Median filtering is used for denoising in the optimisation stage. They changed the objective function to account for this finding:

$$
\begin{aligned}
E_A(\mathbf{u},\mathbf{v},\hat{\mathbf{u}},\hat{\mathbf{v}}) = \sum_{i,j} \Big\{ & \rho_D\left(I_1(i,j) - I_2(i+u_{i,j}, j+v_{i,j})\right) \\
& + \lambda\big[\rho_S(u_{i,j}-u_{i+1,j}) + \rho_S(u_{i,j}-u_{i,j+1}) \\
& \quad + \rho_S(v_{i,j}-v_{i+1,j}) + \rho_S(v_{i,j}-v_{i,j+1})\big]\Big\} \\
& + \lambda_2\left(\|\mathbf{u}-\hat{\mathbf{u}}\|^2 + \|\mathbf{v}-\hat{\mathbf{v}}\|^2\right) \\
& + \sum_{i,j} \sum_{(i',j')\in\mathcal{N}_{i,j}} \lambda_3\left(|\hat{u}_{i,j}-\hat{u}_{i',j'}| + |\hat{v}_{i,j}-\hat{v}_{i',j'}|\right),
\end{aligned}
$$

Here an auxiliary flow field $\hat{u}$ and $\hat{v}$ is used to ensure smoothness over a local region, in their case a 5x5 region. $\|\mathbf{u}-\hat{\mathbf{u}}\|^2 + \|\mathbf{v}-\hat{\mathbf{v}}\|^2$ is the coupling term enforces that the auxiliary flow is close to the original flow.
$\sum_{i,j}\sum_{(i',j')\in\mathcal{N}_{i,j}} \lambda_3\left(|\hat{u}_{i,j}-\hat{u}_{i',j'}| + |\hat{v}_{i,j}-\hat{v}_{i',j'}|\right)$ is the non-local term that enforces that patches in the auxiliary flow field have a similar flow. This mimics the median filtering heuristic. This can lead, however, as can be seen for the Lucas-Kanade method, to rounded edges, when a pixel's patch is dominated by pixels belonging to another object. Therefore, the pixels within a region should be weighed depending on their likelihood of belonging to the same object. So we rewrite the non-local term as follows:

$$
\sum_{i,j} \sum_{(i',j')\in\mathcal{N}_{i,j}} w_{i,j,i',j'}\left(|\hat{u}_{i,j}-\hat{u}_{i',j'}| + |\hat{v}_{i,j}-\hat{v}_{i',j'}|\right)
$$

Where w, gives higher odds to pixels deemed to belong to the same object. This w is approximated by calculating the color-value distance:

$$
\exp\left\{-\frac{|i-i'|^2+|j-j'|^2}{2\sigma_1^2} - \frac{|I(i,j)-I(i',j')|^2}{2\sigma_2^2}\right\}\frac{o(i',j')}{o(i,j)},
$$

where o is the occlusion variable and $\sigma_1 = 7$ and $\sigma_2 = 7$.

2.3 Deep-Learning Models

2.3.1 FlowNet (2015)

FlowNet was among the first successful deep learning-based approaches to optical flow estimation, introducing convolutional neural networks (CNNs) as an effective framework for this task. The architecture was proposed in two variants: FlowNetSimple and FlowNetCorr.

FlowNetSimple processes a pair of consecutive images by concatenating them along the channel dimension and passing them through a series of convolutional layers. These layers progressively extract hierarchical features and estimate a coarse flow field, which is subsequently refined through a decoder (or upsampling) path to produce a dense optical flow map at the original resolution.

FlowNetCorr, on the other hand, employs a two-stream architecture in which each image from the input pair is processed independently by a series of convolutional layers. The resulting feature maps are then combined using a correlation layer, which explicitly computes pairwise similarities between patches from the two feature maps, serving as a form of implicit motion matching.

While the correlation layer was initially introduced to enhance the model's capacity to learn pixel correspondences, subsequent analysis revealed that comparable or superior performance could be achieved using simpler end-to-end architectures without such explicit matching operations. This underscored the potential of purely convolutional models to implicitly learn motion representations directly from data [10].

2.3.2 SPyNet (2017)

SPyNet adopts a coarse-to-fine strategy to address large displacements in optical flow estimation, combining classical principles with modern deep learning techniques. The input images are first downsampled to construct an image pyramid, and the initial optical flow is initialised to zero at the coarsest level.

At each pyramid level, the second image is warped according to the upsampled optical flow estimated from the previous (coarser) level. A convolutional neural network (CNN) is then used to predict a residual flow correction between the warped image and the first image. Each CNN is trained independently to estimate flow refinements at its respective scale.

By progressively refining the flow from coarse to fine resolutions, the problem of large motion estimation is decomposed into a sequence of simpler, small-motion estimation tasks. This hierarchical structure significantly reduces the complexity of the learning problem at each stage, enabling the network to achieve accurate optical flow predictions with relatively lightweight CNN architectures [11].

2.3.3 DC-Flow (2017)

DC-Flow was among the first methods to construct a four-dimensional cost volume for optical flow estimation. The approach begins by downsampling the input image pair by a factor of three to reduce computational complexity. For each pixel, a feature vector is computed, capturing local appearance information. A dense matching cost volume is then constructed by evaluating the distance between feature vectors for all possible pixel correspondences across the two images.

To enhance the robustness of the matching, outlier correspondences are identified and removed from the cost volume. Pixels can now be matched by finding the lowest cost for each pixel. Subsequently, an initial optical flow field is obtained through regularisation of the remaining costs. Both forward and backward optical flows are estimated independently, and inconsistencies between them are detected and eliminated to further refine the solution.

The resulting low-resolution optical flow is then upsampled to the original resolution. Gaps introduced during this process are addressed using a combination of variational refinement and inpainting techniques, yielding the final dense flow field [12].

2.3.4 PWC-Net (2018)

PWC-Net integrates key ideas from prior optical flow models, including FlowNet, SPyNet, and DC-Flow, by employing a coarse-to-fine framework combined with the construction of partial cost volumes. Rather than relying solely on a CNN to directly regress optical flow, as in SPyNet, PWC-Net first generates a partial cost volume at each pyramid level. This cost volume encodes matching costs between features extracted from the first image and warped features from the second image, where warping is based on the upsampled flow estimate from the previous (coarser) level.

The partial cost volume, along with the features of the first image and the upsampled flow, is then provided as input to a CNN that refines the optical flow estimate at the current resolution. By leveraging cost volume representations and warping operations within a multi-scale hierarchy, PWC-Net improves flow estimation accuracy while maintaining computational efficiency [13].

2.3.5 Iterative residual refinement network (2019)

Hur et al. introduced an iterative residual refinement framework to enhance the performance of optical flow estimation models, building upon the foundations established by FlowNet and PWC-Net. In contrast to SPyNet, which employs multiple convolutional neural networks (CNNs) with independently trained weights at each pyramid level, their method utilises a single CNN whose weights are shared across multiple iterations. At each iteration, the current optical flow estimate is refined by predicting a residual correction, which is added to the previous estimate. This recurrent refinement mechanism allows the model to progressively improve its prediction, leading to higher accuracy while maintaining a compact parameter footprint. By sharing weights across iterations, the model also benefits from reduced complexity and improved generalisation [14].

2.3.6 RAFT (2020)

RAFT (Recurrent All-Pairs Field Transforms) employs a fixed-point iterative refinement strategy to estimate optical flow, similar in spirit to the iterative residual refinement networks. The method initialises a dense flow field, which is then progressively updated through multiple refinement steps. A key innovation of RAFT lies in its construction of a multi-scale, all-pairs cost volume. Instead of computing cost volumes at multiple image resolutions as in traditional pyramidal approaches, RAFT generates a single high-resolution cost volume by pooling over the last two spatial dimensions with kernel sizes of 1, 2, 4, and 8. This design enables the network to simultaneously capture both small and large displacements within a unified representation.

Another novel contribution is the use of a gated recurrent unit (GRU)-based update block, which iteratively refines the optical flow predictions. The recurrent structure allows the network to maintain a hidden state across iterations, facilitating more stable convergence and improved flow accuracy. Together, these innovations result in highly accurate, dense optical flow estimations while maintaining computational efficiency. RAFT has completely differentiable cost volume, contrary to DC-flow [15].

2.3.7 GMA (2021)

Global Motion Aggregation (GMA) is an augmentation of the RAFT framework, specifically designed to improve optical flow estimation in regions affected by occlusion, that is, pixels corresponding to objects visible in the first frame but not in the subsequent frame.

GMA introduces an attention-based mechanism, inspired by techniques commonly used in transformer architectures, to model long-range dependencies across the image. In particular, it constructs an attention matrix by computing self-similarities within the feature representations extracted from the context network. The central assumption underpinning GMA is that pixels belonging to the same object or structure exhibit similar motion patterns, even when parts of the object become occluded. By exploiting this relational information,

GMA enables the network to infer plausible motion for occluded regions based on the motion of visible, similar pixels.

The GMA unit accepts as input both the context features (providing local spatial information) and the two-dimensional motion features (reflecting the current flow estimate). Through attention-based aggregation, it produces refined motion features that encapsulate globally aggregated motion cues. These features are then fed into the recurrent update operator (e.g., GRU) during the iterative refinement of the optical flow estimate, allowing the model to integrate both local and non-local motion information at each iteration.

By incorporating GMA into the RAFT architecture, the resulting system achieves improved robustness to occlusion and enhanced flow consistency within moving objects, yielding significant performance gains across standard optical flow benchmarks [16].

2.3.8 SKFlow (2022)

SuperKernelFlow (SKFlow) employs large convolutional kernels to achieve an expanded receptive field, thereby enhancing the network's ability to estimate motion in occluded regions [17]. This design choice is motivated by the findings of Luo et al. [18], who demonstrated that increasing kernel size has a more significant impact on the receptive field than simply increasing network depth. To mitigate the computational overhead associated with large kernels, SKFlow incorporates depthwise separable convolutions, which reduce parameter count and improve efficiency. Each convolutional block in SKFlow consists of three key components: (1) a large L×L kernel to capture long-range spatial dependencies, complemented by a smaller auxiliary S×S kernel to refine local features; (2) residual connections that fuse the outputs of the large and small kernels, enhancing gradient flow and facilitating multi-scale feature aggregation; and (3) pointwise (1×1) convolutions that model cross-channel correlations following the spatial processing. This architecture effectively balances representational capacity with computational efficiency [17].

2.3.9 FlowFormer (2022)

FlowFormer is among the first architectures to effectively leverage transformer-based models for optical flow estimation. While retaining the use of cost volumes to encode pixel correspondence information, FlowFormer introduces a novel approach by projecting the cost volume into a latent feature space through a transformer-based encoder. This encoding phase captures long-range dependencies and rich context beyond local pixel neighbourhoods, addressing limitations of traditional convolutional methods. Subsequently, a decoder network operates on the latent features to predict dense optical flow fields. By utilising self-attention mechanisms within the transformer, FlowFormer achieves more globally consistent and accurate flow estimations, particularly in challenging scenarios involving large displacements or complex scene structures [19].

2.3.10 MemFlow (2024)

MemFlow is designed to capture long-range temporal dependencies across video sequences. In contrast to optical flow approaches, which operate on pairs of consecutive frames, MemFlow incorporates an aggregated historical context. By leveraging this extended temporal memory, the model aims to produce more temporally consistent and stable flow estimates for videos [20].

2.4 Evaluation Datasets

2.4.1 Yosemite (1994)

The Yosemite dataset is a synthetic optical flow benchmark comprising 8-bit grayscale image sequences with a resolution of 316 $\times$ 252 pixels. The dataset is based on aerial imagery of the Yosemite Valley, from which synthetic 3-dimensional scenes are constructed. These scenes consist of two distinct regions: the valley and the sky. Ground truth optical flow is provided exclusively for the valley region, while the sky region lacks reliable motion information due to undergoing Brownian motion.

The dataset introduces several challenges relevant to optical flow estimation, including occlusions, where portions of the scene become temporarily hidden or revealed due to camera or object motion, and a diverging flow field, typically resulting from simulated forward camera motion through the 3-dimensional scene. These characteristics make the Yosemite dataset a useful benchmark for evaluating the robustness of optical flow algorithms under conditions of depth variation. The error is reported using the angular error [5].

2.4.2 Middlebury (2007)

The Middlebury dataset introduced several significant improvements over the earlier Yosemite sequence, including the incorporation of non-rigid motion, large displacements, and more realistic surface textures. Ground truth optical flow was acquired in a controlled laboratory environment using fluorescent paint applied to scene surfaces. Image pairs were captured under both ambient and ultraviolet (UV) illumination, with the UV lighting revealing fine-grained texture patterns that were otherwise not visible.

To compute ground truth flow, a local brute-force search was performed on high-resolution UV images to match small image patches between successive frames, enabling high-precision correspondence estimation. Additionally, the Middlebury benchmark introduced the AEPE metric alongside the previously used angular error, providing a more comprehensive evaluation of optical flow accuracy by quantifying both direction and magnitude discrepancies between estimated and ground truth flow vectors [21].

2.4.3 MPI Sintel (2012)

The MPI Sintel dataset is a synthetic optical flow benchmark derived from the open-source animated short film "Sintel," produced by the Durian Open Movie Project. The original Blender scene files are openly available, which enables the generation of accurate ground truth optical flow data. Compared to earlier benchmarks such as the Middlebury dataset, MPI Sintel introduces increased

complexity through features such as large motion, specular reflections, and motion blur.

The dataset is structured into multiple rendering passes of increasing realism and difficulty. The albedo pass represents the simplest version, containing only intrinsic surface colors without any shading. The clean pass incorporates basic shading effects, while the final pass adds further visual complexity, including motion blur, atmospheric effects, and specular highlights. This progression allows for a more thorough evaluation of optical flow algorithms under varying levels of photorealism and scene complexity [22].

2.4.4 KITTI-15 (2015)

The KITTI dataset comprises real-world data collected using a sensor-equipped vehicle that rode through the streets of Karlsruhe, Germany. The vehicle was outfitted with various sensors, including cameras and a 3-dimensional laser scanner, enabling the acquisition of high-resolution imagery and spatial information. This dataset provides valuable ground truth for a range of computer vision tasks such as optical flow estimation, object detection, and object tracking [23] To generate ground truth for optical flow, 3-dimensional CAD models are aligned with the 3-dimensional point clouds obtained from the laser scanner, enabling the precise simulation of object motion and corresponding pixel displacements in the image plane [24].

2.4.5 Spring (2023)

The Spring dataset is a synthetic benchmark designed for the evaluation of scene flow and optical flow algorithms. It offers significant improvements over previous datasets by providing high-resolution, high-detail imagery that enables more precise assessment of motion estimation techniques. Specifically, the dataset includes 6,000 HD image pairs at a resolution of 1920×1080 pixels.

To facilitate more accurate evaluation of fine-grained motion and subpixel accuracy, the corresponding ground truth for optical and scene flow is gener-

ated at an even higher resolution of 3840 × 2160 pixels. Similar to the MPI Sintel dataset, the scenes in Spring are created using Blender, based on assets from the open-source animated film Spring. This approach allows for the generation of dense, pixel-accurate ground truth data under a wide range of visual conditions and motion complexities [25].

2.5 Pretraining Datasets

Contemporary optical flow models are typically pretrained on large-scale synthetic datasets before being evaluated on more realistic benchmark datasets. These pretraining datasets are often not photorealistic, yet they have demonstrated high effectiveness in generalising well to real-world scenarios, often outperforming models trained exclusively on realistic data [26].

2.5.1 FlyingChairs (2015)

FlyingChairs is a synthetic optical flow dataset comprising 22,872 image pairs, generated by compositing 3-dimensional chair models onto a collection of 2-dimensional background images sourced from stock photography. To simulate motion, 2-dimensional affine transformations are independently applied to both the background and the foreground chair objects. As a result of the planar nature of these transformations, only a single visible side of each chair is maintained throughout the sequence, limiting the representation of full 3-dimensional motion or (self-)occlusions.

To enhance model generalisation and mitigate overfitting, the dataset includes data augmentation techniques, such as the addition of Gaussian noise, brightness variations, and other photometric perturbations. Despite its synthetic nature and limited realism, FlyingChairs has proven to be an effective pretraining dataset for learning optical flow representations, particularly when followed by fine-tuning on more realistic benchmarks [10].

2.5.2 FlyingThings3D (2016)

FlyingThings3D extends the concept introduced by FlyingChairs by incorporating realistic 3-dimensional object models, thereby enabling the generation of both optical flow and scene flow ground truth. Unlike its predecessor, which applies 2-dimensional affine transformations, FlyingThings3D simulates realistic 3-dimensional motion by applying rigid translations and rotations to both the virtual camera and scene objects along linear 3-dimensional trajectories.

The dataset contains approximately 25,000 stereo image pairs, all rendered using Blender, which facilitates pixel-accurate ground truth for optical flow, disparity, and scene flow. Each scene includes between 5 and 20 dynamic objects, randomly sampled from a subset of 32,872 models from the ShapeNet database. The background environment is procedurally generated by populating the scene with randomly placed geometric primitives, such as cuboids and cylinders, which are then textured.

FlyingThings3D thus provides a more physically grounded and diverse synthetic dataset for training and evaluating models on dense motion estimation tasks in 3-dimensional space [27].

2.5.3 AutoFlow (2021)

Autoflow is a dynamic dataset designed to optimise performance based on a target dataset. This optimisation is achieved through learnable hyperparameters that govern the motion, shape, and appearance of objects within the dataset. The construction of scenes involves sampling a random background and overlaying it with polygons. These polygons are characterised by a random number of sides, may contain holes, and can have either smooth or non-smooth edges. The optical flow field is sampled randomly, and objects are warped accordingly. The motion of objects can be rigid, encompassing scale, rotation, and translation, or non-rigid, achieved through a bilinear grid warp.

Empirical findings indicate that scenes with 3 or 4 objects, particularly those with smooth edges, yield the best results. For datasets such as Sintel and Kitti, the application of a bilinear grid warp on objects was found to

be crucial. Additionally, the introduction of fog and motion blur resulted in significant improvements in terms of Average End-Point Error (AEPE) [26].

2.6 Challenges

2.6.1 Large displacements

Large displacements can occur when objects move rapidly relative to the camera. Capturing large displacements in optical flow estimation necessitates a correspondingly large receptive field, which increases computational complexity.

2.6.2 Aperture problem

The aperture problem refers to the inherent ambiguity in estimating motion based on a limited local image region. When only a small portion of a moving object is visible, such as a straight edge, the motion can only be determined perpendicular to the edge, making the true motion direction indeterminate. A classic illustration of this phenomenon is the barber pole illusion, where the perceived motion of diagonal stripes becomes ambiguous due to the constrained viewing aperture [3].

2.6.3 Occlusion

Occlusion in optical flow refers to the phenomenon where a pixel that is visible in the reference frame becomes invisible in the subsequent frame. This may occur due to self-occlusion during object rotation, as parts of the object obscure themselves, or due to inter-object occlusion, where another object obstructs the view. Occlusions can also result from scene dynamics, such as objects moving out of the camera's field of view. This makes for an example task, as they violate most classical constraints, such as the brightness constancy equation.

In some cases, optical flow can still be inferred through contextual cues, such as the motion of visible regions of a partially occluded object. However,

when an object moves entirely out of the frame, the absence of visual information renders accurate flow estimation infeasible, even for human observers.

2.6.4 Brightness changes

Variations in illumination, such as objects entering shadowed regions, introduce additional complexity to optical flow estimation. These changes violate the brightness constancy assumption, which underpins many traditional flow algorithms, thereby complicating the task of establishing reliable pixel correspondences.

2.6.5 Textureless regions

Textureless regions present a significant challenge for optical flow estimation, as the lack of distinctive visual features impedes reliable pixel matching. In such cases, algorithms must rely on broader contextual information or regularisation strategies to infer motion, compensating for the absence of local texture cues.

2.7 Deep Learning

Deep learning can be seen as a subset of machine learning, which can be seen as a subset of artificial intelligence.

2.7.1 Neural networks

Neural networks are computational models inspired by the structure and function of the biological brain. They aim to replicate the brain's ability to learn and adapt by utilising interconnected processing units known as artificial neurons. These neurons are organised into structured layers and are connected through synaptic weights. Information is propagated through the network via activation functions that determine the output of each neuron based on its input. As illustrated in Figure 1, the network architecture consists of multiple layers, with a total of 26 learnable parameters in this example, comprising 21 weights and 5 biases. These parameters are adjusted during the learning

process using the backpropagation algorithm, which minimises the prediction error by computing gradients and updating the weights accordingly [28].

2.7.1.1 Feedforward networks

Feedforward networks are neural networks that do not use the output toward the input.

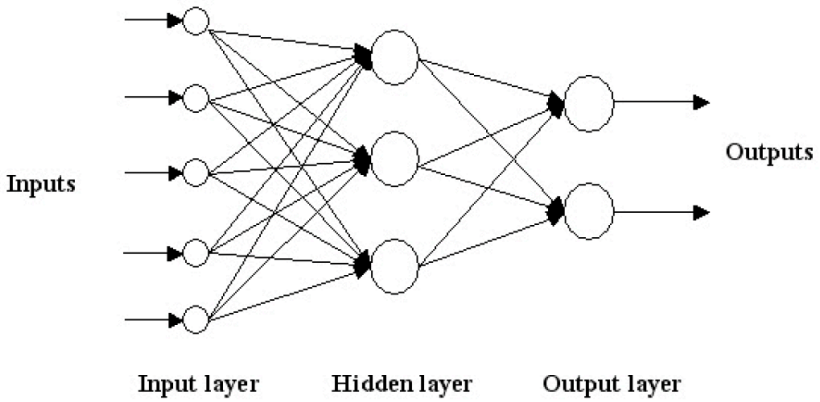

Figure 1: Example of a Feedforward Network

This is a fully connected network, as every neuron is connected with every other neuron, using weights [28].

2.7.1.2 Stochastic Gradient Descent

To update the weights, they are adjusted in the opposite direction of the gradient of the loss function with respect to the weights. Instead of computing this gradient using the entire dataset, a small subset of samples (a mini-batch) is used to approximate the gradient. This approach, known as stochastic gradient descent (SGD) or mini-batch gradient descent, is computationally more efficient than standard (batch) gradient descent, which uses all training samples to compute the exact gradient at each iteration.

2.7.1.3 Activation Function

Every neuron uses an activation function on input. In this manner, non-linearity can be introduced:

$$h = g(W^\top x + c)$$

where g is the activation function. One example is a ReLU, which makes every negative number 0, and all the other numbers stay the same ($f(x) = max(0, x)$)

2.7.1.4 Backpropagation

Backpropagation is used to update the weights. By using gradient descent the weights and biases of the neurons are changed to converge to a local optimum. The error of the output neuron j can be computed as the squared difference between the outcome y_j and the desired d_j outcome:

$$e_j(n) = d_j - y_j(n)$$

$$\varepsilon_j(n) = \frac{1}{2} e_j^2(n)$$

The average error over all the training examples can be computed as follows:

$$\varepsilon_{av} = \frac{1}{N} \sum_{n=1}^{N} \varepsilon(n)$$

The function of the output neuron can be written as follows, where w_{j0} is equal to the bias b_j:

$$y_j(n) = f\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n) \right)$$

We want to know $\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$, which is the partial derivative of the error with respect to the weights and biases, which will tell us how we should update the weights and biases. This partial derivative can be written, using the chain rule, as follows:

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial w_{ji}(n)}$$

The first two derivatives can be obtained trivially:

$$\frac{\partial \varepsilon(n)}{\partial e_j(n)} = e_j(n)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

$$\frac{\partial y_j(n)}{\partial w_{ji}(n)} = f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) \cdot \frac{\partial \left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right)}{\partial w_{ji}(n)}$$

$$= f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) \cdot y_i(n)$$

$$\text{where} \quad f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) = \frac{\partial f\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right)}{\partial \left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right)}$$

[28]

2.7.2 Overfitting

Overfitting occurs when a model doesn't work well on unseen data, as it's generalisation capability is limited as it suits only the training data. To combat overfitting in neural networks various strategies can be deployed.

2.7.2.1 Dropout

During training, dropout stochastically deactivates a subset of neurons, thereby reducing inter-neuronal dependency and encouraging the learning of more robust and independent feature representations. This regularisation mechanism improves the model's ability to generalise to unseen data [29].

2.7.2.2 Regularisation

To prevent overly complex representations that may hinder generalisation,

several regularisation strategies can be employed. For instance, early stopping restricts the number of training epochs to reduce overfitting. Similarly, constraining the number of neurons limits the representational capacity of the model, thereby encouraging better generalisation performance [30].

2.7.3 Residual networks

Residual connections, as introduced in residual neural networks (ResNets), function by adding the input of a given block to its output, thereby enabling the network to learn a residual mapping rather than the direct transformation. Let $H(x)$ denote the underlying function the network aims to approximate. Instead of directly learning $H(x)$, the residual block learns $F(x) = H(x) - x$, so that the overall output becomes $F(x) + x$.

This formulation has been shown to alleviate the degradation problem in very deep networks, where additional layers can actually lead to higher training error. It is hypothesised that learning the residual function is easier than learning the original unreferenced mapping, particularly when the optimal transformation is close to the identity function. In such cases, driving the residual function $F(x)$ towards zero effectively enables the network to preserve the input, facilitating gradient flow and improving convergence in deep architectures [31].

2.7.4 Convolutional Neural Networks

Convolutional Neural Networks (ConvNets) are designed to process data with a grid-like topology, such as images and videos, by leveraging spatial hierarchies through the use of local connections and shared weights. In the context of image analysis, the initial layers of a ConvNet typically consist of convolution and pooling operations.

In the convolutional layers, a set of learnable filters (or kernels) is applied to the input feature maps to produce new feature maps that capture local spatial patterns. Each filter learns to detect a specific type of feature (e.g., edges, textures, or shapes), and different filters in a layer are responsible for

capturing different aspects of the input. These filters are applied across the entire input space via weight sharing, which reduces the number of parameters and allows the network to learn spatially invariant features. Following convolution, pooling layers, most commonly max pooling, are used to downsample the feature maps. This reduces the spatial resolution and computational load, while also providing a degree of translation invariance. Pooling aggregates information over small neighbourhoods, enabling the network to become more robust to small translations and distortions in the input. The hierarchical nature of ConvNets allows them to learn increasingly complex features across layers: lower layers capture simple local patterns, while higher layers integrate these into more abstract representations. This coarse-to-fine architecture exploits the compositional structure of visual data, where higher-level patterns (motifs) are composed of lower-level features [32].

2.7.4.1 Network In Network

A 1×1 convolution operates by computing a weighted linear combination across all input channels at each spatial location, effectively performing an element-wise dot product between the filter weights and the channel vector at that position. This operation is typically followed by a non-linear activation function, such as ReLU. When multiple 1×1 filters are applied in parallel, the result is analogous to applying a fully connected layer independently at each spatial location. This approach enables increased representational power and non-linearity, and is referred to as a "Network in Network" architecture, as introduced by [33].

2.7.4.2 Inception modules

In traditional convolutional neural networks, the architectural operations within each layer, such as the presence of pooling or the choice of convolutional kernel size, are predefined and fixed across the network. In contrast, the Inception module introduces a more flexible design by performing multiple operations

in parallel within a layer, including 1×1, 3×3, and 5×5 convolutions, as well as max-pooling. The outputs of these parallel paths are concatenated along the channel dimension, allowing the network to learn, through training, which receptive field sizes and operations are most relevant at different depths. This adaptive capability enables the model to dynamically select appropriate feature extraction strategies depending on the level of abstraction, as proposed by [34].

2.7.4.3 Depthwise Separable Convolutions

Xception employs depthwise separable convolutions, a factorised form of standard convolution that decomposes the operation into two distinct steps. First, pointwise convolutions (typically 1×1 kernels) are applied independently at each spatial location to capture cross-channel correlations. Subsequently, depthwise convolutions (e.g., 3×3 or 5×5) are performed separately for each channel to model spatial correlations. This architectural design effectively decouples the learning of spatial and inter-channel features, leading to improved computational efficiency and performance [35].

2.7.5 Transformer Architecture

The transformer architecture is a neural network framework that leverages multi-head attention mechanisms and has achieved state-of-the-art performance in diverse domains such as natural language processing and computer vision.

It follows an encoder–decoder structure, where both encoder and decoder are composed of six stacked layers, as originally proposed in the seminal work *Attention is All You Need*[36]. Each layer consists of a multi-head self-attention module, followed by a feed-forward neural network.

2.7.5.1 Attention Mechanism

An attention mechanism defines a mapping between a query vector and a set of key–value pairs, producing a weighted output representation. Formally, the

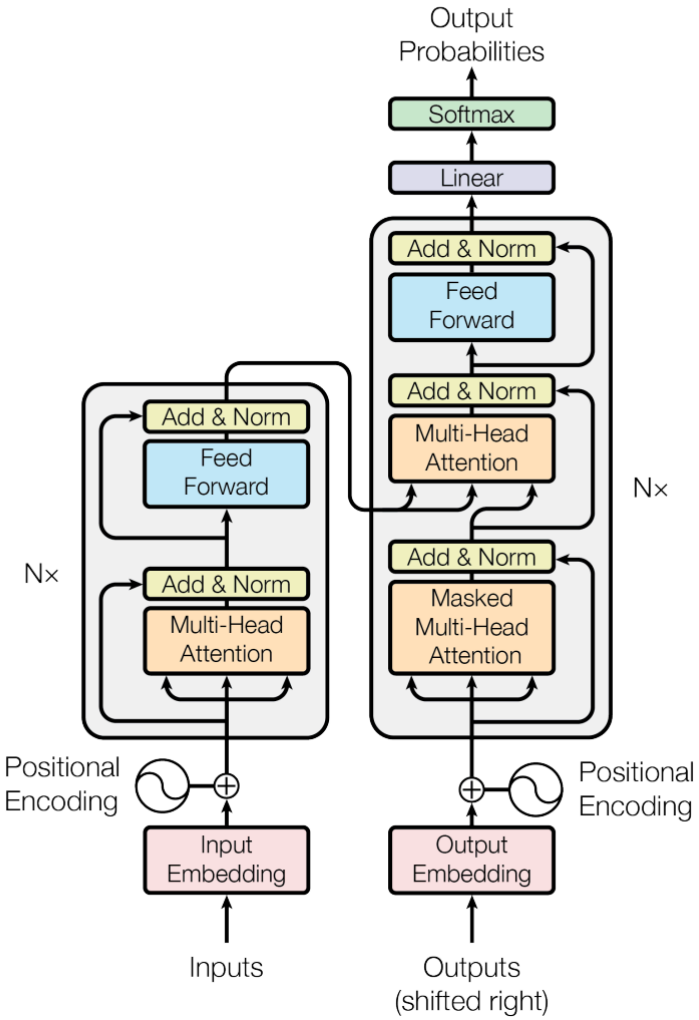

Figure 2: The standard transformer architecture, as introduced by Vaswani et al.[36].

scaled dot-product attention is given by[36]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where Q, K, and V denote the query, key, and value matrices respectively.

3 Scientific Article

The scientific article is on the next page.

FlyingDutchman: An Optical Flow Analysis Tool

Petter Reijalt[1], Sander Gielisse[1], and Jan van Gemert[1]

[1]Computer Vision Lab, Delft University of Technology, The Netherlands

Abstract

Much progress in optical flow research has been driven by benchmark datasets. However, these datasets provide only limited feedback on the underlying causes of architectural failures, typically restricted to metrics such as end-point error (EPE), occlusion statistics, and large-displacement ranges. This leads to imprecise claims regarding areas consecutive models have improved upon. In this paper, we present an analysis tool that enables the generation of customizable datasets, allowing controlled variation in displacement size, camera corruptions, luminance, and other factors. We demonstrate the utility of this tool by analyzing the behaviour of different architectures under varying displacement sizes and in low-light settings.

1. Introduction

Optical flow describes the pixel–wise correspondence between consecutive image frames and underpins numerous applications, including autonomous driving, robotics, and motion detection [1]. Optical flow can be difficult to estimate due to numerous challenges under which, occlusions [2], low-light settings [3], untextured surfaces [2], and large displacements [2]. As in many computer-vision tasks, model performance has advanced rapidly, largely propelled by benchmark datasets, that are either real-word or synthetic. Because dense ground-truth annotations (annotations for every pixel in the frame) are difficult to obtain in real-world settings, the majority of these datasets are synthetic. Unfortunately, most benchmarks provide only general feedback, typically limited to overall displacement and occlusion metrics, offering little insight into the specific failure modes of a model, e.g. KITTI [4], Sintel [5], Spring [6] etc. The effects of specific confounding variables in vacuum remain often unknown. Consequently, recent optical flow papers seldom pinpoint which specific areas they improve, instead reporting a generic performance gain. Alternatively, one can also make claims without providing evidence to support them. For example, FlowFormer makes the claim that its attention mechanism better handles challenging cases such as occluded regions and large displacements [7]. A similar claim is made by the authors of FlowFormerPlusPlus, who speculate that their design improves robustness to noise, large displacements, and occlusions [8]. However, these assertions are presented without direct evidence, reporting instead on EPE scores from evaluation datasets, where multiple confounding factors are at play.

In this paper we introduce an analysis tool, that makes it easier for researchers to substantiate these claims. By isolating variables, we can research the effects of large displacements. Hence can more accurately pin-point improvements across different models. Our analysis tool can automatically generate optical flow datasets with fine-grained control over scene parameters such as object displacement, camera motion, lighting conditions, blur, and surface textures.

By using Differential Evolution (DE), we learn the scene parameters that lead to expressive quantities such as average pixel displacement and average pixel brightness. Which makes it easier to draw concrete conclusions.

The framework is written in Python on top of the Blender API[1]. The code is available here or at the following url https://github.com/Petter6/FlyingDutchman.

In order to test the claims of better long range dependency and noise robustness made by the FlowFormer [7] and FlowFormerPlusPlus [8] papers, and in order to demonstrate the capabilities of the analysis tool, we issue the following research question, which we aim to answer using our analysis tool:

- How do varying magnitudes of optical flow vectors affect model performance?
- To what extent do low-light settings influence the accuracy of optical flow predictions?

2. Related works

We evaluate four seminal optical flow architectures: RAFT [9], GMA [2], FlowFormer [7], and Flow-

[1]https://docs.blender.org/api/current/index.html

Former++ [8]. RAFT introduced a dense all-pairs cost volume together with iterative refinement via a recurrent update operator, establishing a strong baseline across standard benchmarks [9]. GMA builds on the RAFT paradigm by aggregating global motion cues with attention mechanisms to better capture long-range dependencies and reason about occlusions [2]. FlowFormer implements an attention module, in order to make better use of the cost-volume than RAFT it claims [7]. Aming to improve handling longe-range relationships. FlowFormer++ further pretrains the transformer using masked cost-volume autoencoding, which is claimed to enhance robustness to noise and to improve performance on large-motion regimes [8].

Kubric [10] is a dataset rendering pipeline, which combines realistic physics via PyBullet and Blender for rendering. In this manner, ground truth for different computer vision tasks, such as Optical Flow, can be created. Allowing among others to change the velocity and the location of the objects, and camera movements. They show that their tool can be used to create either a benchmark dataset, or can be used to create additional training data, where they show to be competitive with AutoFlow [11]. Models pre-trained using Kubric are shown to outperform models pre-trained with FlyingChairs [12] on KITTI-15 [13] and Sintel.

Carla simulator is a simulator that can be used to generate ground truth from a simulated vehicle. Thereby allowing for capturing realistic optical flow ground truth in the autonomous vehicle domain [14]. This comes however with a few challenges, as it provides backwards optical flow compared to the normal forward optical flow; additionally the precision of the flow is limited to 10 digits [15].

RobustSpring extends the Spring dataset by introducing distortions such as rain and blur, enabling the evaluation of optical flow models under controlled corruptions. The study reports that transformer-based architectures achieve the best overall performance across most corruption types, although their accuracy decreases under noise distortions. The authors attribute this to the reliance on global matching. In contrast, stacked architectures such as FlowNet2 perform remarkably well on noise corruptions, likely due to their iterative refinement process [16].

Mayer et al. [17] investigated the key characteristics of good synthetic pretraining datasets, focusing on factors such as lighting, displacement distributions, and textures. Their findings indicate that networks trained with dynamic lighting perform poorly when evaluated on scenes with static lighting. The study further emphasizes that diversity in the training data is more critical than realism, and that incorporating camera distortions into the dataset significantly improves performance.

Zheng et al. [3] researched the effects of low-light scenes on FlowNet models and PWC-net. They did this by creating their own dataset and creating various levels of darkness with accompanying noise levels. The EPE shows an overall increasing trend with darkness, though not strictly monotonically, as occasional decreases occur between consecutive levels [3]. [3].

3. Analysis tool

The analysis tool is implemented in Python using the Blender Python API (bpy) bpy version 4.4.0, which allows for headless rendering and full programmatic control over the Blender suite.

Backgrounds

Two types of backgrounds are integrated into the pipeline: a set of eight 2D HDRI images and one fully modeled 3D market scene. The latter enables the rendering of complete and realistic optical flow fields, whereas the former allows for the isolation of foreground objects in a simplified setting that is well-suited for analyzing motion statistics.

The HDRI images are sourced from PolyHaven[2], and the 3D supermarket model is obtained from Creazilla[3].

Objects

To introduce dynamic, moving elements into the scenes, we make use of the SynthDet dataset, which was originally developed for Unity and which we ported to Blender. We reduced the dataset size to 50 objects and provide two variants: one with textures and one without. This enables a controlled investigation of the role of texture in influencing the end-point error. [4].

Configuration file

The rendering process is configurable through a structured JSON file, which defines parameters such as object count, camera motion, lighting conditions, and image resolution. A detailed overview of these configurable parameters is presented in Table 1.

Learning Configuration File

Blender units can be unintuitive in the context of optical flow. For example object motion is expressed in Blender units and the power of lighting sources is specified in watts. We developed a tool that maps these parameters to more interpretable and expressive quantities such as the average pixel displacement and average pixel intensity.

We employ an evolutionary algorithm, Differential Evolution. To infer the scene parameters that yield

[2]https://polyhaven.com/hdris

[3]https://creazilla.com/media/3d-model/65069/mini-market-scene

[4]https://github.com/Unity-Technologies/SynthDet

Parameter	Explanation
scene.num_scenes	Number of scenes to generate in this dataset.
scene.num_obj	Number of objects placed per scene.
scene.seed	Random seed for deterministic scene generation.
scene.depth	Minimum and maximum distance of objects relative to the camera.
camera.position_start	Initial camera position $[x, y, z]$ in blender units.
camera.rotation_start	Initial camera rotation [roll, pitch, yaw] in radians.
camera.shutter_speed	Physical shutter speed (seconds).
camera.translation	Per-frame camera translation vector.
camera.rotation_offset	Per-frame camera rotation offset (radians).
lighting.lighting_color	RGB range applied to lights.
lighting.3d_scene_light_intensities	Intensity of a point light.
objects.textures_enabled	Toggle for textured materials on imported meshes.
motion.mean_rotation	Mean object rotation per frame (radians).
motion.sigma_rotation	Standard deviation of object rotation.
motion.mean_translation	Mean object translation vector $[x, y, z]$.
motion.sigma_translation.x	Standard deviation of object translation.
effects.fog_percentage	Density of fog (0–1).
effects.inverted_colors	Produce negative-coloured images.
render.resolution.x	Horizontal render resolution (pixels).
render.resolution.y	Vertical render resolution (pixels).
stats.calc_occlusion	Export per-pixel occlusion mask images.
stats.calc_displacement	Calculate displacement distributions.

Table 1: Blender2Flow configuration parameters.

a desired average displacement magnitude or luminance:

$$(R + G + B)/(3 * 255))$$

This algorithm iteratively generates scenes and measures statistics such as luminance and displacement magnitude. Through repeated evaluations, it learns parameter settings that produce target scenes, for example with an average displacement of 10 pixels within a predefined error tolerance. The accuracy of this process depends on the available computational resources. The displacement size only based on the pixels that display a moving object, ie. flow $\neq 0$; pixels with a flow of zero are hence excluded.

Output flow

An example scene and optical flow map can be seen here 1. The flow is calculated using the vector pass. The vector pass, can be enabled during rendering and produces 4 vectors. 2 vectors that describe the backwards optical flow (the optical flow relative to the previous frame) and 2 vectors for the forward flow, which we use. The vector pass normally provides the information used by the motion blur node. The code for the flow extraction from the exr-file can be found here. After the u and v component are extracted the v component needs to be inverted as a downward y-flow is considered positive when it comes to flow maps. The Z and W vector are chosen as they represent the forward flow as opposed to the backward flow.

Format folder output

The user can choose to either use a Sintel format for the dataset output or a KITTI format. Sintel uses an image resolution of 1024×436 pixels. KITTI uses a resolution of 1242 × 376 pixels [4]

Output Statistics

Occlusion

Furthermore, it is possible to calculate which pixels are occluded. This can be achieved by casting a ray through each pixel in the initial frame and recording the corresponding object face. In the subsequent frame, a ray is cast through each pixel displaced by the optical flow vector. If the intersected face, or the object itself, differs from the initial frame, the pixel is marked as occluded. This procedure yields fairly accurate results, although edge cases remain, for example, when a pixel becomes occluded by its own face but is nevertheless classified as visible.

Displacement magnitude

Due availability of accurate flow data it is possible to calculate displacement magnitudes and creating a corresponding histograme. This is done by calculating the L2-norm:

$$\text{mag} = \sqrt{u^2 + v^2}$$

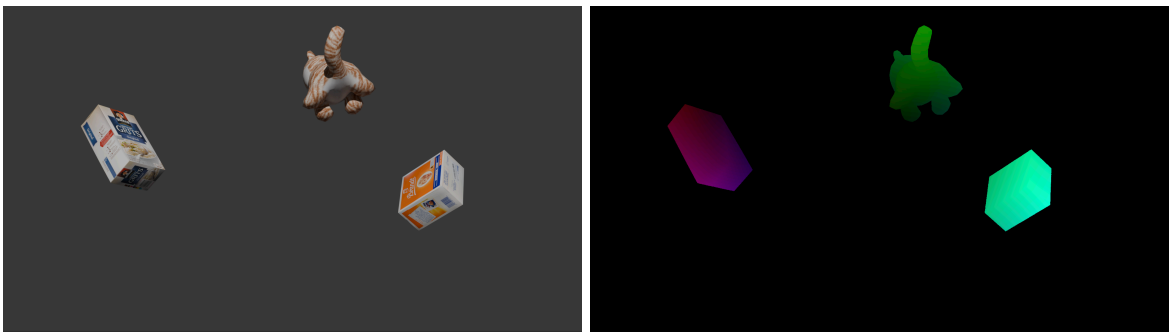

Figure 1: Example scene with corresponding optical flow, without a background

where u is the vertical flow for every pixel and v is the horizontal flow. This metric is used to answer the first RQ, and is used by the DE algorithm to converge to a mean displacement size.

4. Experiments

Large Displacements

Our first research question concerns the effect of larger displacement magnitudes on different models. Since optical flow networks are trained on datasets with varying displacement distributions, they may be biased toward either different size displacements. Sintel has an average displacement of 13.5 pixels [18], FlyingThings3D [19] is around 38 pixels, FlyingChairs around 19 [19], and KITTI-15 around 2.8 pixels [20]. Also different architectures tend to refine long range relationships differently [17]. To isolate the impact of object motion, we generated 50 short video sequences in which a randomly-chosen single object moves by 10 pixels between consecutive frames, while both the background and the object appearance remain fixed. The background is an HDRI and hence has no detectable optical flow. The object only moves in the spaces and does not rotate. No extra lighting is added and only the standard HDRI background lighting is used. For each sequence, we computed the EPE between the base frame and the reference frame at increasing pixel displacements (i.e., 10 px, 20 px, ..., 90 px). We evaluated four representative architectures that mark recent milestones in optical flow research, as layed out in Table 4. Here, C+T+S+K+H denotes training on Chairs, Things, KITTI, Sintel, and HD1K. An example scene is shown in Figure 4, where four consecutive frames are displayed with a displacement of 10 pixels between each pair, enabling controlled experiments with progressively larger displacements. The results are reported in Table 2. As shown in Figure 2, all models exhibit a similar trajectory with respect to the increase in EPE as displacement grows. Make note of the RAFT architecture, which loses track of some objects at the 80-pixel mark, resulting in remarkably steeper path. Notably, all lowest-EPE checkpoints correspond to models trained with the C+T+K+S+H regime, highlighting the importance of a varied training set for this purpose 7. In contrast, models trained solely on KITTI perform the worst on GMA, FlowFormerPlusPlus and Raft 7. This can be attributed both to the synthetic nature of our dataset versus the real-world nature of KITTI, and to the fact that KITTI contains very few large displacements, with an average displacement of 2.8 pixels [20]. GMA seems to perform slightly better than the other architectures.

Low-light

We further investigated the role of luminance, i.e., the amount of light in the scene, and the point at which performance begins to degrade. Many current-day benchmarks lack low-light scenes in their datasets, and often models are not built for this specific scenario [3]. Low-light scenes come with an increased amount of noise, which increases the difficulty of the task [3].

Specifically, we calculated the intensity needed of four lights in a 3D market scene required to achieve an average luminance of 1 across all pixels in frame. These same settings are then kept the same for the second frame. The light intensities were then scaled to reach average luminance values of 2, 3, 4, and 5. Higher luminance values were also tested, but no additional performance degradation was observed. We created 100 scenes and recorded the average EPE of all models.

An example scene with the different luminance levels for Scene 0 is shown in Figure 5. Here the same object is shown only for different luminance levels. The performance trajectories of the best checkpoints of all architectures are presented in Figure 3, where GMA and FlowFormer++ exhibit nearly identical behavior. Figure 6 also shows that architectures trained exclusively on Chairs and KITTI perform poorly. For the Chairs dataset, this can partly be attributed to the absence of dynamic lighting sources in the training data.

5. Discussion

We introduce an optical flow analysis tool that generates datasets via a Blender-based pipeline. Scene parameters are optimized with Differential Evolution

Architecture	Model	10px	20px	30px	40px	50px	60px	70px	80px	90px
RAFT	raft-chairs	0.075	0.084	0.093	0.102	0.116	0.135	0.148	0.166	0.207
RAFT	raft-kitti	0.048	0.068	0.095	0.122	0.189	0.266	0.382	0.462	0.585
RAFT	raft-sintel	0.021	0.027	0.032	0.036	0.042	0.047	0.053	0.063	0.068
RAFT	raft-small	0.095	0.115	0.132	0.150	0.169	0.190	0.209	0.229	0.252
RAFT	raft-things	0.046	0.051	0.055	0.060	0.064	0.069	0.073	0.079	0.083
GMA	gma-chairs	0.035	0.050	0.066	0.079	0.092	0.106	0.127	0.145	0.160
GMA	gma-kitti	0.039	0.054	0.080	0.112	0.176	0.259	0.373	0.454	0.555
GMA	gma-sintel	0.015	0.021	0.026	0.030	0.036	0.040	0.045	0.051	0.057
GMA	gma-things	0.036	0.040	0.045	0.049	0.054	0.059	0.064	0.069	0.074
FlowFormer	chairs_small	0.042	0.071	0.096	0.129	0.170	0.206	0.267	0.309	0.346
FlowFormer	chairs	0.089	0.114	0.137	0.171	0.312	0.430	0.519	0.618	0.703
FlowFormer	kitti	0.065	0.091	0.125	0.159	0.182	0.241	0.321	0.445	0.556
FlowFormer	sintel_small	0.018	0.024	0.029	0.035	0.041	0.047	0.054	0.060	0.067
FlowFormer	sintel	0.019	0.025	0.031	0.036	0.042	0.048	0.055	0.061	0.068
FlowFormer	things_kitti	0.031	0.035	0.040	0.044	0.049	0.053	0.058	0.063	0.067
FlowFormer	things_small	0.029	0.034	0.039	0.044	0.049	0.054	0.060	0.065	0.071
FlowFormer	things	0.028	0.034	0.039	0.044	0.050	0.056	0.063	0.069	0.077
FlowFormer++	chairs	0.038	0.062	0.085	0.121	0.166	0.213	0.249	0.287	0.333
FlowFormer++	kitti	0.065	0.096	0.138	0.166	0.197	0.254	0.300	0.341	0.407
FlowFormer++	sintel	0.020	0.025	0.031	0.036	0.041	0.047	0.052	0.058	0.063
FlowFormer++	things_288960	0.035	0.040	0.045	0.050	0.056	0.062	0.068	0.074	0.080
FlowFormer++	things	0.033	0.040	0.048	0.055	0.062	0.070	0.078	0.085	0.093

Table 2: End-point error (EPE) for different models across displacement magnitudes (10–90 pixels).

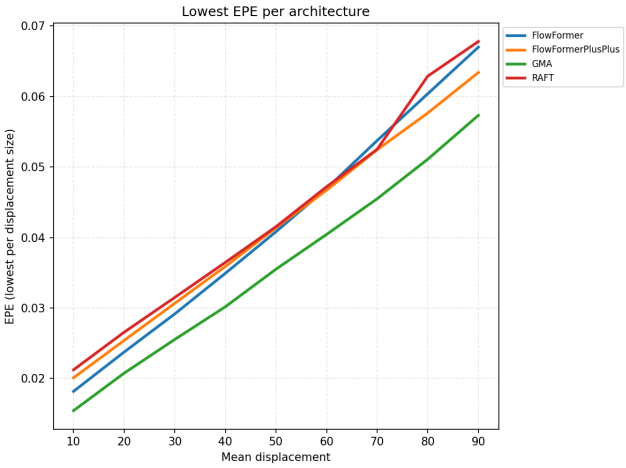

Figure 2: Lowest EPE per architecture per displacement size

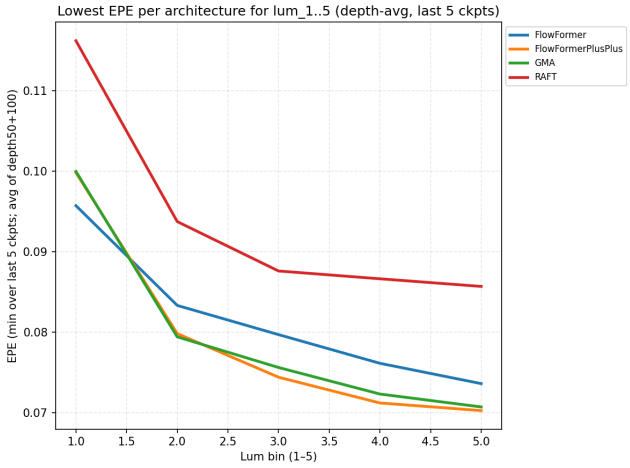

Figure 3: Lowest EPE per architecture per luminance level

(DE), enabling controlled ablations to isolate variables and relate them to the end-point error (EPE).

The claims made by FlowFormerPlusPlus regarding superior performance in noise handling and large displacements cannot be substantiated using our tool. For displacements, only marginal improvements over the FlowFormer architecture are observed. With respect to luminance, FlowFormerPlusPlus performs slightly better in brighter scenes (2–5

FlowFormer outperforms RAFT but does not demonstrate improvements over GMA. In fact, GMA performs better than FlowFormer in handling larger displacements.

Smaller models appear to be more robust to both large displacements and low-light settings compared to their larger counterparts. This trend can be seen in the flatter performance trajectory of smaller models, particularly in FlowFormer, where the Sintel-small variant achieves the best overall performance (see Fig. 6).

The GMA model performs on par with the FlowFormer models and consistently outperforms RAFT.

Across two experiments, we observe consistent gains from diverse pretraining: models trained on

Architecture	Model	1%	2%	3%	4%	5%
RAFT	raft-chairs	0.349	0.247	0.211	0.188	0.170
RAFT	raft-kitti	0.542	0.434	0.389	0.379	0.374
RAFT	raft-sintel	0.116	0.094	0.088	0.087	0.086
RAFT	raft-small	0.309	0.280	0.263	0.253	0.248
RAFT	raft-things	0.146	0.118	0.111	0.107	0.103
GMA	gma-chairs	0.281	0.219	0.187	0.173	0.162
GMA	gma-kitti	0.380	0.288	0.261	0.246	0.236
GMA	gma-sintel	0.100	0.122	0.079	0.072	0.071
GMA	gma-things	0.162	0.124	0.110	0.105	0.101
FlowFormer	chairs_small	0.359	0.331	0.315	0.304	0.292
FlowFormer	chairs	0.280	0.244	0.237	0.229	0.224
FlowFormer	kitti	0.431	0.346	0.319	0.304	0.291
FlowFormer	sintel_small	0.096	0.083	0.080	0.078	0.077
FlowFormer	sintel	0.101	0.084	0.080	0.076	0.074
FlowFormer	things_kitti	0.164	0.109	0.091	0.084	0.080
FlowFormer	things_small	0.116	0.097	0.090	0.088	0.086
FlowFormer	things	0.161	0.114	0.103	0.097	0.094
FlowFormer++	chairs	0.287	0.233	0.221	0.215	0.203
FlowFormer++	kitti	0.327	0.265	0.253	0.246	0.244
FlowFormer++	sintel	0.100	0.080	0.074	0.071	0.070
FlowFormer++	things_288960	0.184	0.111	0.094	0.086	0.082
FlowFormer++	things	0.171	0.118	0.108	0.103	0.101

Table 3: EPE of different optical flow models at different luminance thresholds

Figure 4: Four consecutive frames of a video showing 10 pixel displacements.

a mixture of KITTI, MPI-Sintel, FlyingThings3D, HD1K, and FlyingChairs outperform those trained on narrower regimes.

Regarding the first research question (RQ1), GMA emerges as the best choice for handling large displacements. Models trained exclusively on KITTI tend to underperform, while models trained on a diverse set of datasets achieve the strongest results.

For the second research question (RQ2), concerning low-light conditions, small transformer-based architectures appear most effective, as they achieve better performance in extremely dark settings.

Limitations

There are several limitations. First, the number of distinct scenes is modest (n = 50 and n=100). Losing track of a single object can induce speculative predictions and small shifts in EPE that may not reproduce across random seeds. Second, while Differential Evolution optimizes a configuration that targets a mean displacement magnitude, the realized displacement varies with the sampled supermarket item, introducing spread between small datasets. Third, scene realism is limited, especially for 2D HDRI backgrounds, complicating external validity; the 3D market experiments also used a single viewpoint, reducing diversity. Finally, we did not model camera egomotion, which is atypical for real-world capture and may limit generalizability.

References

[1] M. Zhai, X. Xiang, N. Lv, and X. Kong, "Optical flow and scene flow estimation: A survey," Pattern Recognition, vol. 114, p. 107861, 2021.

[2] S. Jiang, D. Campbell, Y. Lu, H. Li, and R. Hartley, "Learning to estimate hidden motions with global motion aggregation," in Proceedings of the IEEE/CVF international conference on computer vision, pp. 9772–9781, 2021.

[3] Y. Zheng, M. Zhang, and F. Lu, "Optical flow in the dark," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 6749–6757, 2020.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," The

Figure 5: Five frames of the same scene with luminance levels left-to-right (1% to 5%)

Architecture	Model	Size (MB)	Training Dataset(s)
RAFT	raft-chairs	21.1	FlyingChairs
RAFT	raft-kitti	21.1	KITTI
RAFT	raft-sintel	21.1	C+T+K+S+H
RAFT	raft-small	4.0	FlyingChairs + FlyingThings
RAFT	raft-things	21.1	FlyingThings3D (clean + final pass)
GMA	gma-chairs	23.8	FlyingChairs
GMA	gma-kitti	23.8	KITTI
GMA	gma-sintel	23.8	C+T+K+S+H
GMA	gma-things	23.8	FlyingThings3D (clean + final pass)
FlowFormer	chairs_small	25.0	FlyingChairs
FlowFormer	chairs	65.1	FlyingChairs
FlowFormer	kitti	65.1	KITTI
FlowFormer	sintel_small	25.0	C+T+K+S+H
FlowFormer	sintel	65.1	C+T+K+S+H
FlowFormer	things_kitti	65.1	FlyingThings3D + KITTI
FlowFormer	things_small	25.0	FlyingThings3D
FlowFormer	things	65.1	FlyingThings3D
FlowFormer++	chairs	65.0	FlyingChairs
FlowFormer++	kitti	65.0	KITTI
FlowFormer++	sintel	65.0	C+T+K+S+H
FlowFormer++	things_288960	65.0	FlyingThings3D
FlowFormer++	things	65.0	FlyingThings3D

Table 4: Overview of model checkpoints, sizes, and training datasets. C+T+K+S+H stands for FlyingChairs + FlyingThings3D + KITTI-15 + Sintel + HD1K

international journal of robotics research, vol. 32, no. 11, pp. 1231–1237, 2013.

[5] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI 12, pp. 611–625, Springer, 2012.

[6] L. Mehl, J. Schmalfuss, A. Jahedi, Y. Nalivayko, and A. Bruhn, "Spring: A high-resolution high-detail dataset and benchmark for scene flow, optical flow and stereo," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4981–4991, 2023.

[7] Z. Huang, X. Shi, C. Zhang, Q. Wang, K. C. Cheung, H. Qin, J. Dai, and H. Li, "Flowformer: A transformer architecture for optical flow," in European conference on computer vision, pp. 668–685, Springer, 2022.

[8] X. Shi, Z. Huang, D. Li, M. Zhang, K. C. Cheung, S. See, H. Qin, J. Dai, and H. Li, "Flowformer++: Masked cost volume autoencoding for pretraining optical flow estimation," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1599–1610, 2023.

[9] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16, pp. 402–419, Springer, 2020.

[10] K. Greff, F. Belletti, L. Beyer, C. Doersch, Y. Du, D. Duckworth, D. J. Fleet, D. Gnanapragasam, F. Golemo, C. Herrmann, et al., "Kubric: A scalable dataset generator," in Proceedings of the

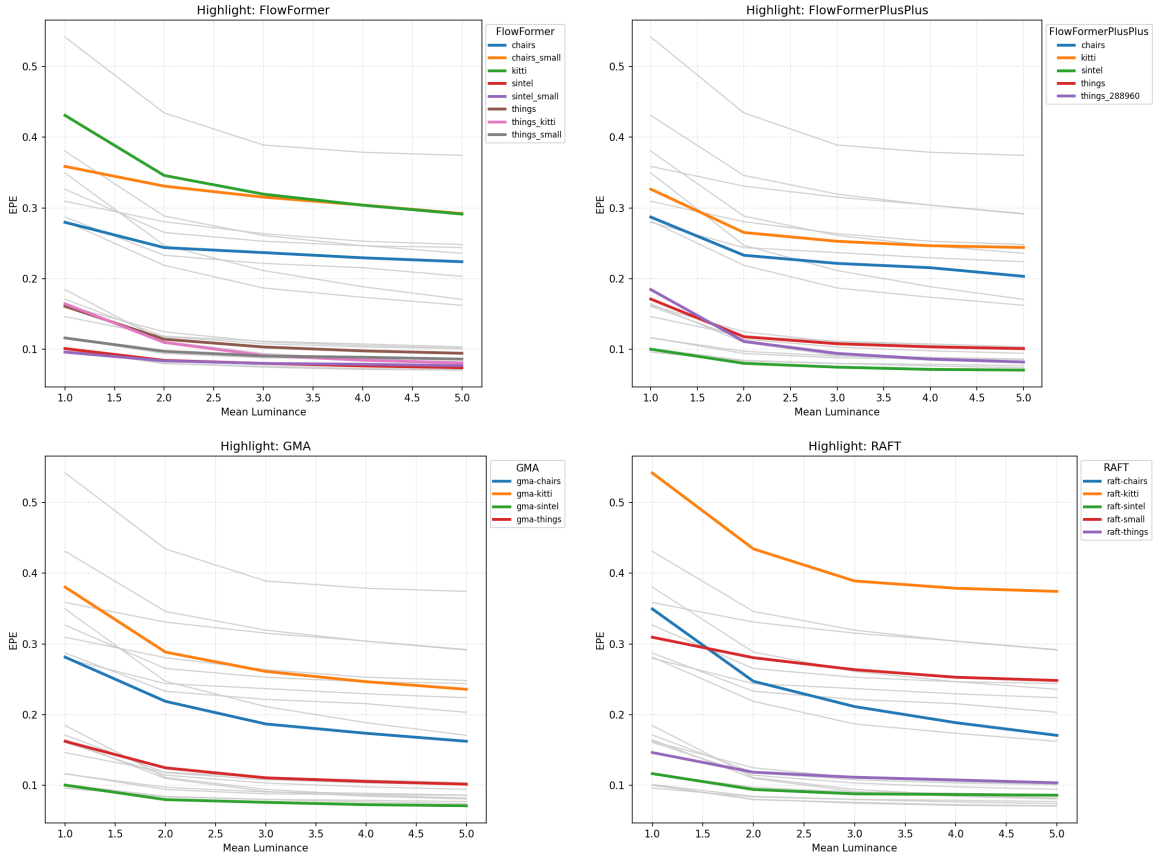

Figure 6: EPE per luminance level highlighted for every architecture

IEEE/CVF conference on computer vision and pattern recognition, pp. 3749–3761, 2022.

[11] D. Sun, D. Vlasic, C. Herrmann, V. Jampani, M. Krainin, H. Chang, R. Zabih, W. T. Freeman, and C. Liu, "Autoflow: Learning a better training set for optical flow," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10093–10102, 2021.

[12] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in Proceedings of the IEEE international conference on computer vision, pp. 2758–2766, 2015.

[13] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3061–3070, 2015.

[14] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in Conference on robot learning, pp. 1–16, PMLR, 2017.

[15] Z. Wan, W. Zhai, Y. Cao, and Z. Zha, "Emotive: Event-guided trajectory modeling for 3d motion estimation," arXiv preprint arXiv:2503.11371, 2025.

[16] J. Schmalfuss, V. Oei, L. Mehl, M. Bartsch, S. Agnihotri, M. Keuper, and A. Bruhn, "Robustspring: Benchmarking robustness to image corruptions for optical flow, scene flow and stereo," arXiv preprint arXiv:2505.09368, 2025.

[17] N. Mayer, E. Ilg, P. Fischer, C. Hazirbas, D. Cremers, A. Dosovitskiy, and T. Brox, "What makes good synthetic training data for learning disparity and optical flow estimation?," International Journal of Computer Vision, vol. 126, pp. 942–960, 2018.

[18] S. Savian, P. Morerio, A. Del Bue, A. A. Janes, and T. Tillo, "Towards equivariant optical flow estimation with deep learning," in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 5088–5097, 2023.

[19] R. Li, R. T. Tan, L.-F. Cheong, A. I. Aviles-Rivero, Q. Fan, and C.-B. Schonlieb, "Rainflow: Optical flow under rain streaks and rain veiling effect," in Proceedings of the IEEE/CVF international conference on computer vision, pp. 7304–7313, 2019.

[20] G. Liu, H. Tian, Q. Gao, and Y. Yuan, "Enhancing visual inertial odometry with efficient dynamic perceptionnet and consistency improvement fusion," Available at SSRN 5227953.

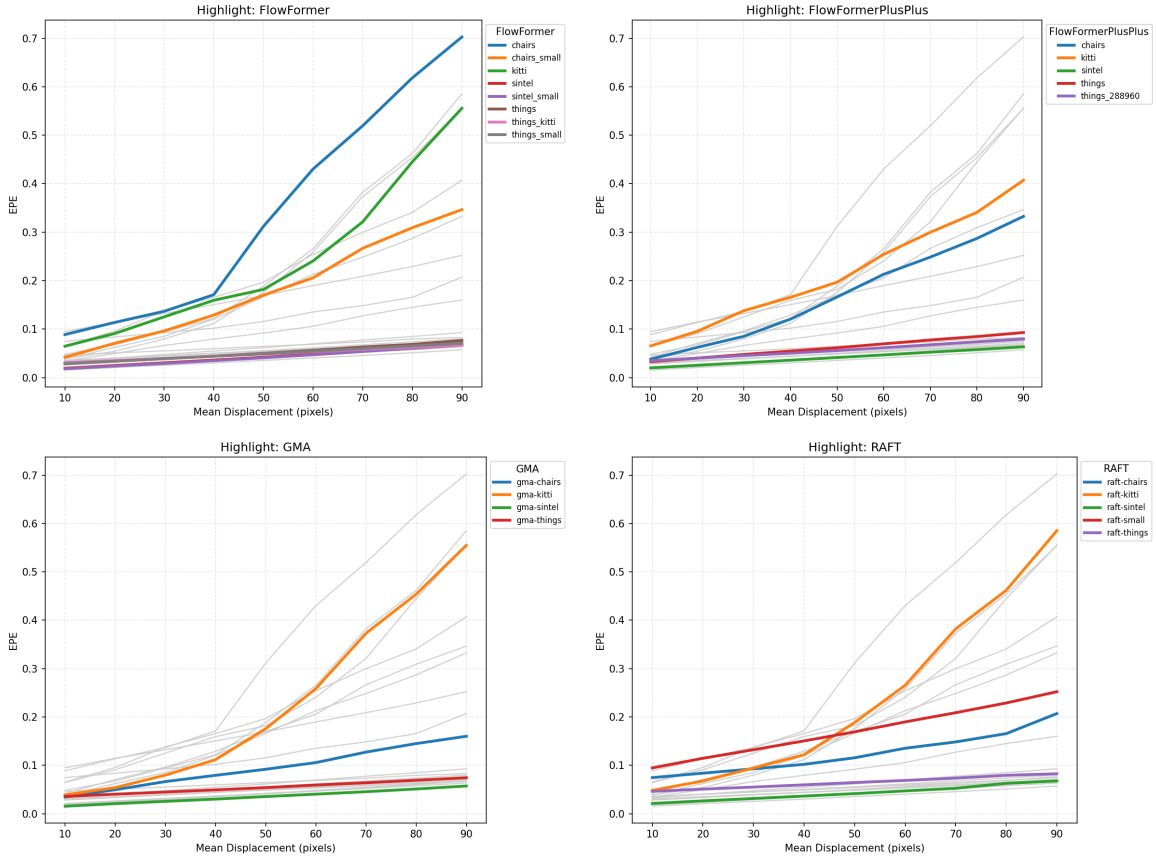

Figure 7: EPE per displacement size highlighted for every architecture

Bibliography

[1] Steven S. Beauchemin and John L. Barron. "The computation of optical flow". In: *ACM computing surveys (CSUR)* 27.3 (1995), pp. 433–466.

[2] Mingliang Zhai et al. "Optical flow and scene flow estimation: A survey". In: *Pattern Recognition* 114 (2021), p. 107861.

[3] Andrea Alfarano et al. "Estimating optical flow: A comprehensive review of the state of the art". In: *Computer Vision and Image Understanding* (2024), p. 104160.

[4] Robert Guamán-Rivera, Jose Delpiano, and Rodrigo Verschae. "Event-based optical flow: Method categorisation and review of techniques that leverage deep learning". In: *Neurocomputing* (2025), p. 129899.

[5] John L Barron, David J Fleet, and Steven S Beauchemin. "Performance of optical flow techniques". In: *International journal of computer vision* 12 (1994), pp. 43–77.

[6] Berthold KP Horn and Brian G Schunck. "Determining optical flow". In: *Artificial intelligence* 17.1-3 (1981), pp. 185–203.

[7] Carnegie Mellon University. *Lucas-Kanade Optical Flow*. Accessed: 2025-03-19. 2025. URL: `https://www.cs.cmu.edu/lectures/Lecture21.pdf`.

[8] Jean-Yves Bouguet et al. "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm". In: *Intel corporation* 5.1-10 (2001), p. 4.

[9] Deqing Sun, Stefan Roth, and Michael J Black. "Secrets of optical flow estimation and their principles". In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2432–2439.

[10] Alexey Dosovitskiy et al. "Flownet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2758–2766.

[11] Anurag Ranjan and Michael J Black. “Optical flow estimation using a spatial pyramid network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4161–4170.

[12] Jia Xu, René Ranftl, and Vladlen Koltun. “Accurate optical flow via direct cost volume processing”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1289–1297.

[13] Deqing Sun et al. “Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8934–8943.

[14] Junhwa Hur and Stefan Roth. “Iterative residual refinement for joint optical flow and occlusion estimation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 5754–5763.

[15] Zachary Teed and Jia Deng. “Raft: Recurrent all-pairs field transforms for optical flow”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer. 2020, pp. 402–419.

[16] Shihao Jiang et al. “Learning to estimate hidden motions with global motion aggregation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9772–9781.

[17] Shangkun Sun et al. “Skflow: Learning optical flow with super kernels”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 11313–11326.

[18] Wenjie Luo et al. “Understanding the effective receptive field in deep convolutional neural networks”. In: *Advances in neural information processing systems* 29 (2016).

[19] Zhaoyang Huang et al. “Flowformer: A transformer architecture for optical flow”. In: *European conference on computer vision*. Springer. 2022, pp. 668–685.

[20] Qiaole Dong and Yanwei Fu. “Memflow: Optical flow estimation and prediction with memory”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 19068–19078.

[21] Simon Baker et al. “A database and evaluation methodology for optical flow”. In: *International journal of computer vision* 92 (2011), pp. 1–31.

[22] Daniel J Butler et al. “A naturalistic open source movie for optical flow evaluation”. In: *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI 12*. Springer. 2012, pp. 611–625.

[23] Andreas Geiger et al. “Vision meets robotics: The kitti dataset”. In: *The international journal of robotics research* 32.11 (2013), pp. 1231–1237.

[24] Moritz Menze and Andreas Geiger. “Object scene flow for autonomous vehicles”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3061–3070.

[25] Lukas Mehl et al. “Spring: A high-resolution high-detail dataset and benchmark for scene flow, optical flow and stereo”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 4981–4991.

[26] Deqing Sun et al. “Autoflow: Learning a better training set for optical flow”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10093–10102.

[27] Nikolaus Mayer et al. “A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4040–4048.

[28] Murat H Sazlı. “A brief review of feed-forward neural networks”. In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006).

[29] Pierre Baldi and Peter J Sadowski. “Understanding dropout”. In: *Advances in neural information processing systems* 26 (2013).

[30] Seán Mc Loone and George Irwin. “Improving neural network training solutions using regularisation”. In: *Neurocomputing* 37.1-4 (2001), pp. 71–90.

[31] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[33] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: *arXiv preprint arXiv:1312.4400* (2013).

[34] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[35] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[36] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).