NanoFlowNet

Real-time optical flow estimation on a nano quadcopter

R.J. Bouwmeester

Delft University of Technology

TU Delft

NanoFlowNet

Real-time optical flow estimation on a nano quadcopter

by

R.J. Bouwmeester

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday October 13, 2022 at 01:00 PM.

Thesis committee:	Prof. dr. ir. G. C. H. E. de Croon,	TU Delft, MAVLab,	supervisor
	Assoc. Prof. dr. ir. J. C. van Gemert,	TU Delft, Computer Vision Lab	
	Dr. ir. C. de Wagter,	TU Delft, MAVLab	
	Ir. F. Paredes-Vallés,	TU Delft, MAVLab,	supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

This report on real-time convolutional neural networks for optical flow estimation on nano quadcopters is the result of my MSc graduation research project at the Delft University technology. I have conducted this research under the supervision of Prof. dr. Guido C.H.E. de Croon and Federico Paredes-Vallés at the MAVLab. The project has been an excellent combination of my interests in machine learning and robotics while maintaining a strong link with my prior education in aerospace engineering. The opportunity to see our ideas come to life has been incredible.

The document is structured in two parts. The first part contains a scientific paper. I'm proud to say we have submitted an altered version of this article to ICRA 2023. The second part includes a supplementary literature review, providing background information to the research article.

I want to thank Nilay Sheth, who helped me in the early stages of the project by effectively giving me a crash course in embedded hard- and software engineering. I want to thank the people at Bitcraze, who wholeheartedly welcomed me for an internship to develop these embedded skills further and introduced me to Malmö falafel. I want to thank my supervisors and collaborators, Guido and Federico, for their contagious passion; it would not have been the same without your intelligent contributions. I am incredibly grateful for my family, who have always supported me in my every decision. Finally, I would like to thank my girlfriend for all her valuable advice; you make me a better and happier person.

R.J. Bouwmeester
Delft, September 2022

Contents

I

Research paper

NanoFlowNet: Real-time Optical Flow CNN on a Nano Quadcopter

Rik J. Bouwmeester, Federico Paredes-Vallés and Guido C.H.E. de Croon

***Abstract*— Nano quadcopters are small, agile, and cheap platforms well suited for deployment in narrow, cluttered environments. Due to their limited payload, nano quadcopters are highly constrained in processing power, rendering conventional vision-based methods for autonomous navigation incompatible. Recent machine learning developments promise high-performance perception at low latency, while novel ultra-low power microcontrollers augment the visual processing power of nano quadcopters. In this work, we present NanoFlowNet, an optical flow CNN that, based on the semantic segmentation architecture STDC-Seg, achieves real-time dense optical flow estimation on edge hardware. We use motion boundary ground truth to guide the learning of optical flow, improving performance with zero impact on latency. Validation on MPI-Sintel shows the high performance of the proposed method given its constrained architecture. We implement the CNN on the ultra-low power GAP8 microcontroller and demonstrate it in an obstacle avoidance application on a 34 g Bitcraze Crazyflie nano quadcopter.**

***Index Terms*— MAV, CNN, edge AI**

I. Introduction

Due to their miniature size and weight, nano quadcopters are well suited for deployment in narrow, cluttered environments and safe to operate near humans [1]. Autonomous navigation on these platforms can enable search-and-rescue operations, monitoring, inspections, surveillance, and gas source seeking. Conventional mapping-based methods for aerial autonomy require sensors and processors that are incompatible with the nano quadcopter's constrained payload [2]. However, with the right algorithm design, nano quadcopters have been demonstrated performing complex tasks such as exploration [3], gas source seeking [4], [5], and visual navigation [6].

Given the constrained payload capabilities of MAVs, most attention has been given to monocular solutions. The camera is a lightweight and energy-efficient passive sensor that captures rich information of the environment. One of the most important monocular visual cues is optical flow, which has been extensively exploited on MAVs with higher payload for obstacle avoidance [7], depth estimation [8] and several bio-inspired methods for autonomous navigation [9]–[13].

Traditionally, the task of monocular optical flow estimation has been performed by hand-crafted methods [15], [16]. Recently, the field has been dominated by deep learning methods [17]–[28], which outperform traditional, hand-crafted methods in accuracy and latency. Although the focus has primarily been on improving performance, efforts have been made to find models of reduced size, and faster inference

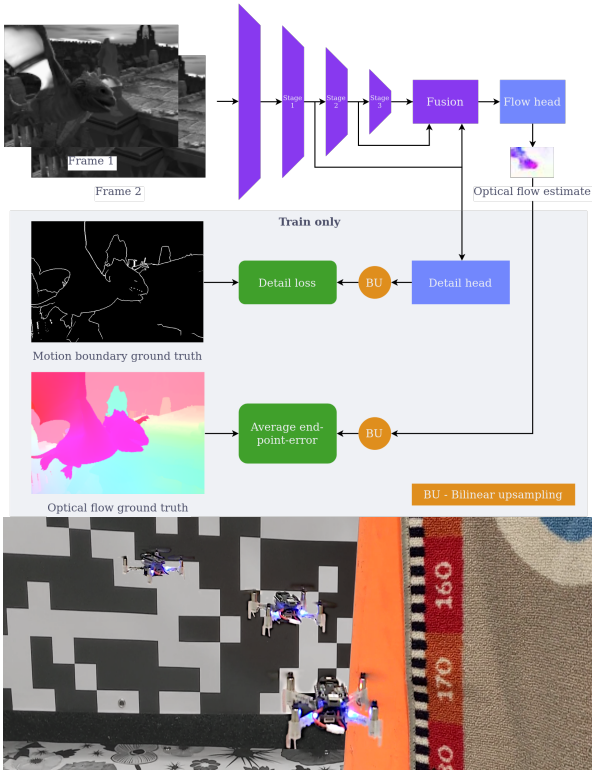


Fig. 1: *Top*: NanoFlowNet consists of i) an encoder that extracts features from the concatenated input images, ii) a fusion module that combines features from different paths, iii) a motion boundary-guided detail head, which is only enabled during training, to guide the learning with zero cost to inference latency. Stages 1-3 of the encoder each consist of a regular STDC block [14], followed by our modified strided STDC block. *Bottom*: We demonstrate NanoFlowNet in an obstacle avoidance application on a 34 gram nano quadcopter.

[18], [19], [21], [22], [25], [26], [29]. However, these methods are still computationally expensive, with network timings ranging from several to tens of FPS on modern desktop GPUs and requiring millions of parameters, rendering these networks incompatible with edge hardware.

In this work, instead of improving the accuracy of state-of-the-art approaches, we focus on inference speed and, more particularly, on the deployment of a dense optical flow network on edge devices. To this end, we present *NanoFlowNet*, a lightweight CNN architecture for optical flow estimation that, inspired by semantic segmentation network STDC-Seg [14], achieves real-time inference on a 34 gram nano quadcopter.

The key contributions of this paper are listed as follows. First, we introduce NanoFlowNet, a novel lightweight convolutional neural network architecture that performs, for the first time, real-time dense optical flow estimation on edge hardware. Through quantitative and qualitative evaluations on optical flow datasets, we validate this network, which runs at 5.5-9.3 FPS on the ultra-low power GAP8 microcontroller. Second, we show, for the first time, that using motion boundary ground truth to guide the learning of optical flow improves performance while having zero impact on inference latency. Third, we demonstrate the proposed NanoFlowNet in a real-world obstacle avoidance application on board a Bitcraze Crazyflie.

The remainder of this paper is organized as follows. Section II reviews related work in autonomous navigation on nano quadcopters and real-time inference with CNNs. In Section III, we propose our CNN architecture and pipeline. Section IV presents performance on public benchmarks, an ablation study, and a real-world obstacle avoidance implementation. Finally, in Section V, we conclude and provide further discussion.

II. Related Work

A. Autonomous navigation on nano quadcopters

The limited computational capacity of the nano quadcopter puts a constraint on the types of methods that can be used for autonomy. Methods demonstrated on-board nano quadcopters can be broadly grouped into model-based reinforcement learning for hovering [30], obstacle avoidance based on dedicated laser ranging sensors [3]–[5], and self-motion estimation using optical flow from dedicated optical flow sensors [31] or estimated with multi-camera setups [32], [33]. With monocular dense optical flow estimates, our method can exploit complex visual cues without requiring additional sensor systems and lends itself to more advanced navigational tasks.

Other methods circumvent the computational constraints of nano quadcopters by running methods off-board [34]–[36]. Our method runs entirely on board to negate reliability, latency, and security concerns of transmitted data and the required proximity to a base station.

Yet other methods rely on augmenting the computational power of the nano quadcopter. Methods based on application-specific integrated circuits (ASICs) [37]–[40] can efficiently provide information for specific tasks such as SLAM and visual-inertial odometry but have not yet been presented on a flying drone. More recently, parallel ultra-low-power (PULP) microcontrollers have introduced energy-efficient multi-core processing to parallelize visual workloads on edge devices. This work exploits the COTS AI-deck equipped with the GreenWaves GAP8 SoC. This nine-core microcontroller has been used for several end-to-end methods that integrate perception and navigation by directly regressing visual inputs through a CNN into control commands [6], [41], [42]. Instead, in our approach, we calculate optical flow, which gives us direct control over quadcopter behavior and can support multiple optical flow-based tasks to be performed simultaneously or interchangeably. Our work, motivated by these benefits, is the first to present a dense regression task on the AI-deck.

B. Real-time inference

Several low latency methods [43]–[47] utilize depthwise separable convolutions for their low computational expense. Depthwise separable convolutions effectively factorize a convolution into a depthwise and a pointwise convolution. The use of depthwise separable layers allows us to use more complex model structures on our constrained platform at the cost of reduced representational capacity per layer.

We draw inspiration from real-time semantic segmentation methods to speed up vision calculations for optical flow while retaining performance. Specifically, we draw inspiration from STDC-Seg [14]. This work is based on former study BiSeNet [48], which identifies a sacrifice of spatial information in real-time methods and improves performance by introducing a separate path designed for encoding spatial information. A feature fusion module (FFM) fuses features from the high and low-level paths. Attention refinement modules (ARM) refine features through channel attention.

STDC-Seg builds on BiSeNet and introduces the STDC module, which increases the receptive field size per layer at low computational cost. Furthermore, it replaces BiSeNet's spatial path with a train-time-only detail head and accommodating detail loss to mimic the encoded spatial details while shrinking the model and decreasing latency. The detail guidance ground truth is generated by convolving the segmentation map ground truth using a Laplacian kernel.

A few critical elements of STDC-Seg and BiSeNet have been separately investigated in the context of optical flow. AD-Net [49] shows that channel attention can benefit optical flow estimation. EDOF [50] fuses features from an edge detect net and an optical flow encoder network for detail-guided optical flow estimation. Instead, like STDC-Seg, we use edges to guide the learning only.

III. Method

We adopt the STDC-Seg network and modify it. First, we replace all regular convolutions with depthwise separable convolutions. To further reduce latency and the number of parameters, we globally reduce the number of filters by a factor four. The semantic segmentation head is replaced with an optical flow head. We introduce an even smaller model with half of NanoFlowNet's filters globally and call it NanoFlowNet-s. Further modifications to the architecture are discussed in detail in the following sections.

A. Motion boundary detail guidance

The closest analogy to the detail guidance used in STDC-Seg is generating edges from the optical flow ground truth. Instead, we replace this "edge-detect" detail guidance ground truth with motion boundary ground truth from the optical flow datasets. While closely related, we argue that using ground truth data is less prone to error than manually generated data. We adopt Focal Loss [51] to counter the class imbalance problem.

B. Strided STDC module redesign

We modify the strided STDC modules to further decrease latency. The original strided STDC module can be found in Fig. 2. The modified strided STDC module can be found in Fig. 3. We identify the first pointwise convolution in the strided STDC module as the most expensive operation. By relocating the pointwise convolution to the bottom path after the average pooling operation, we reduce the overall latency of the module while allowing an increase of the number of features in the top path and the number of features with a large receptive field size in the concatenated output. For stage 1, this results in a reduction of over 50% of the MAC operations. For stage 2 and stage 3, this results in a MAC operation reduction of over 10%.

C. Reduced input/output dimensionality

We design the network for low-resolution input and downscale all dataset's input frames, optical flow and motion boundary ground truth accordingly. The scaling factor is picked such that the resulting data resolution closely matches the target application resolution (160x112 pixels, approximately qqVGA). Horizontal and vertical scaling is identical to fix the aspect ratio to retain naturalism. This allows us to make the network shallower by dropping the first convolution altogether and thus decrease latency while maintaining feature sizes in the deepest layers. As an added benefit, working with downscaled data speeds up training. The primary drawback of reduced input resolution is the loss of information; in particular, we will miss out on small objects and small displacements that are not captured by the resolution. To compare with existing optical flow works, we benchmark performance at native dataset resolution since downscaling flow magnitudes results in lower EPE without a qualitative improvement.

Furthermore, we design our network for grayscale input images, saving two third of the onboard memory dedicated to the input frames and decreasing the cost of the first layer at a loss of input information. We convert input images from the datasets to grayscale.

IV. Experiments

A. Implementation details

All models are trained for 300 epochs on FlyingChairs2 [17], [52], a regenerated FlyingChairs dataset with motion boundary ground truth. We use the Adam optimizer [53], with learning rate $1e-3$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-07$. We fine-tune on FlyingThings3D [54] for 200 epochs with a learning rate of $1e-4$. We train with a batch size of 8.

Given the scaling and conversion to grayscale of input data, our network is not directly comparable with results reported by other works. For comparison, we retrain one of the fastest networks in literature, Flownet2-s [19], on the same data. Given the reduction in resolution, we drop the deepest two layers to maintain a reasonable feature size in the deepest layers. We name the model *Flownet2-xs*.

We run all experiments in a docker environment with TensorFlow 2.8.0, CUDA 11.2, CUDNN 8.1.0, and TensorRT 7.2.2 on an NVIDIA GeForce GTX 1070 Max-Q with batch size 1 for benchmarking latency.

B. Performance and latency on public benchmarks

We evaluate the trained networks on the MPI Sintel [55] train subset. We evaluate on both the clean and final pass. Quantitative results can be found in Table I. NanoFlowNet performs better than FlowNet2-xs, despite using less than 10% of the parameters. FlowNet2-xs does not fit on the AI-deck due to the network size. To put the achieved latency of NanoFlowNet in perspective, we execute FlowNet2-xs' first two convolutions and the final prediction layer on the

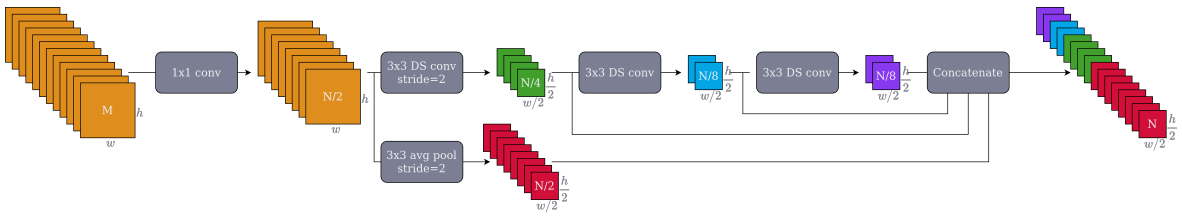


Fig. 2: Strided STDC module. The module is identical to the original strided STDC block [14], with the exception that we use depthwise separable (DS) convolutions in place of all non-pointwise convolutions. We use relu activations after all layers in the block. M is the number of input features, N is the number of output features.

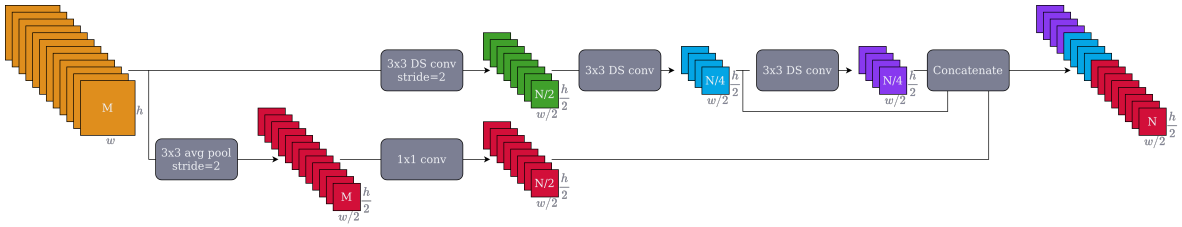
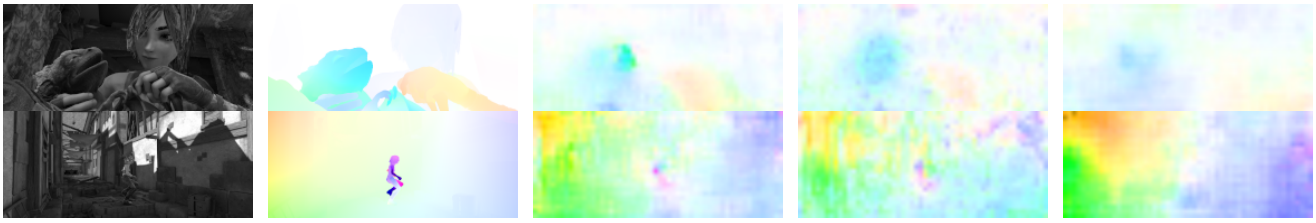


Fig. 3: Our modified strided STDC module. We reorganize the operations to reduce latency while increasing the number of features with large receptive field size in the output. M is the number of input features, N is the number of output features.

Method	MPI Sintel (train) [EPE]		Frame rate [FPS]		Parameters
	Clean	*Final*	*GPU*[1]	*GAP8*[2]	
FlowNet2-xs	9.054	9.458	150	- [3]	1,978,250
NanoFlowNet (ours)	**7.122**	**7.979**	141	5.57	170,881
NanoFlowNet-s (ours)	9.559	10.047	**151**	**9.34**	46,749

TABLE I: Results on MPI Sintel. NanoFlowNet is more accurate than FlowNet2-xs while using less than 10% of the parameters. NanoFlowNet-s is the fastest network tested. [1]At a resolution of 96x224. [2]At a resolution of 112x162, including vision thread. [3]Does not fit on on-board memory.



(a) Input frame I (b) Ground truth (c) NanoFlowNet (ours) (d) NanoFlowNet-s (ours) (e) FlowNet2-xs

Fig. 4: Qualitative comparison of optical flow estimates by NanoFlowNet(-s) and FlowNet2-xs on MPI Sintel (train) clean pass. NanoFlowNet and NanoFlowNet-s pick up on smaller moving objects, such as the person in the bottom row. NanoFlowNet-s' outputs are highly noisy.

GAP8. The three-layer architecture achieves 4.96 FPS, which is slower than running the entire NanoFlowNet (5.57 FPS). On GPU hardware, NanoFlowNet achieves comparable FPS to FlowNet2-xs. NanoFlowNet-s has lower performance than both other models but has a low parameter count with only 27% of NanoFlowNet's and 2.4% of FlowNet2-xs's parameters and is the fastest out of all the networks tested.

Qualitative results, presented in Fig. 4, confirm that NanoFlowNet makes the most accurate optical flow estimates out of the networks tested. Both NanoFlowNet and NanoFlowNet-s appear to detect displacements of smaller objects, which FlowNet2-xs misses. NanoFlowNet-s' flow estimates are highly noisy.

C. Ablation study

1) Motion boundaries detail guidance: We verify the effectiveness of motion boundary detail guidance by re-training the network with i) detail guidance based on the optical flow ground truth convolved with a Laplacian kernel and ii) disabling the detail guidance altogether. Quantitative results can be found in Table II. Motion boundary detail guidance improves results and outperforms edge detect detail guidance. All methods only affect training behavior and have identical latency.

A qualitative comparison of detail guidance methods can be found in Fig. 5. Motion boundary-guided optical flow best defines moving objects and shows the least leakage of foreground objects into the background.

2) Strided STDC module redesign: Table III shows the effects of the strided STDC module redesign. The network with the redesigned module is both faster and more accurate.

Detail guidance method	MPI Sintel (train)	
	Clean	*Final*
None	7.636	8.119
Edge detect	7.404	8.141
Motion boundaries	**7.122**	**7.979**

TABLE II: Different methods of detail guidance. Edge detect detail guidance improves MPI Sintel (train) clean pass results but slightly deteriorates results on the final pass. Motion boundary-guided learning improves results on both MPI Sintel (train) clean and final pass.

Strided STDC block	MPI Sintel (train) [EPE]		Frame rate [FPS]	
	Clean	*Final*	*GPU*[1]	*GAP8*[2]
Unmodified	7.483	8.114	136	4.84
Modified	**7.122**	**7.979**	**141**	**5.57**

TABLE III: Modification impact of strided STDC block. The modified strided STDC block improves accuracy and reduces latency on laptop GPU and the GAP8 MCU. [1]At a resolution of 96x224 [2]At a resolution of 112x162, including vision thread

(a) Input frame I (b) Ground truth optical flow (c) Motion boundary guided optical flow (d) Edge detect guided optical flow (e) No detail guidance optical flow

Fig. 5: Qualitative comparison of detail guidance methods on MPI Sintel (train) clean pass. Motion boundary-guided optical flow shows best-defined objects and characters, with the least leakage of foreground objects into the background.

Mode	MPI Sintel (train) [EPE]		Frame rate [FPS]	
	Clean	*Final*	*GPU[1]*	*GAP8[2]*
Color	7.726	8.344	**141**	5.18
Grayscale	**7.122**	**7.979**	**141**	**5.57**

TABLE IV: Grayscale versus color input frame-based architectures. NanoFlowNet trained on and inferring on grayscale input images reduces EPE over a color-NanoFlowNet. The latency on the GAP8 is reduced due to reduced data transfer and a cheaper first convolution. [1]At a resolution of 96x224 [2]At a resolution of 112x162, including vision thread

3) Reduced input dimensionality: A comparison between training and inferring on grayscale images compared to color images can be found in Table IV. Our grayscale model outperforms the color variant. We hypothesize this is due to the limited capacity of the network. The latency on the GAP8 is reduced due to reduced data transfer.

D. Obstacle avoidance implementation

As a proof-of-concept, we implement NanoFlowNet on an AI-deck equipped Crazyflie 2.x for obstacle avoidance. We use the Flow deck v2 for positioning only. See Fig. 6. The total flight platform weighs 34 grams.

1) Control strategy: We implement the horizontal balance strategy [56], [57], where the quadcopter balances the optical flow in the left and right half plane. The yaw rate $\dot{\psi}$ is set based on the error between the sum of flow magnitudes in the left and right half of the optical flow estimate, see equation 1. We set gains $k_p = 0.0126$ and $k_d = 0.0018$ experimentally. The error between the sum of flow magnitudes is determined with equation 2. The derivative of the error is calculated numerically using equation 3, where Δe_{rl} is the difference in error between the current and previous estimate, and Δt is assumed to be constant. The forward velocity is set at a constant 0.2m/s.

We augment the balance strategy by implementing active oscillations (a cyclic up-down movement, $T = 2s$), resulting in additional optical flow generated across the FOV. This is particularly helpful for avoiding obstacles in the direction of horizontal travel. Up-down rather than left-right surveying favors detecting obstacles broader than taller but is simpler to combine with the left-right balance strategy. Additionally, left-right surveying requires rolling, which introduces rotational flow which does not contain depth information.

We implement the CNN and calculate e_{rl} on the GAP8 microcontroller. Calculating the flow error on the AI-deck reduces the amount of data that needs to be transmitted over UART. The calculation of the yaw rate is done on the Crazyflie 2.x, and fed into the controller. See Fig. 6 for an overview of the application.

$$\dot{\psi} = k_p e_{rl} + k_d \dot{e_{rl}} \tag{1}$$

$$e_{rl} = \sum_{\text{left}} \sqrt{u_i^2 + v_i^2} - \sum_{\text{right}} \sqrt{u_j^2 + v_j^2} \tag{2}$$

$$\dot{e_{rl}} = \frac{\Delta e_{rl}}{\Delta t} \tag{3}$$

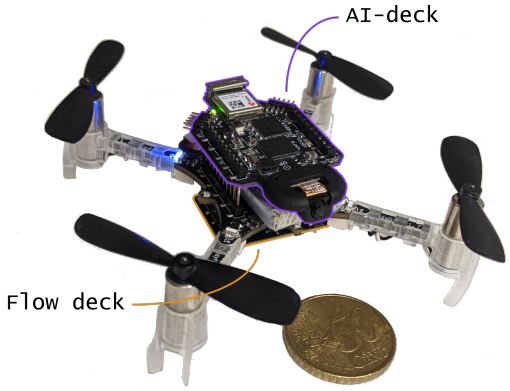


Fig. 6: We use the Crazyflie 2.x as the flight platform for our experiments. We use the AI-deck to capture images with the front-facing camera and to run optical flow inference and processing. The downward-facing optical flow deck is used for positioning only. The total flight platform weighs in at 34 grams.

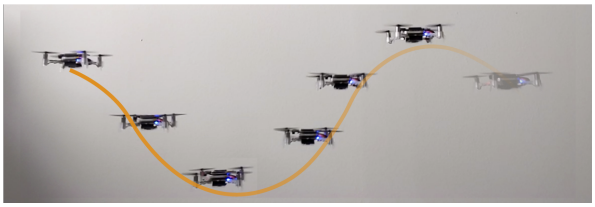
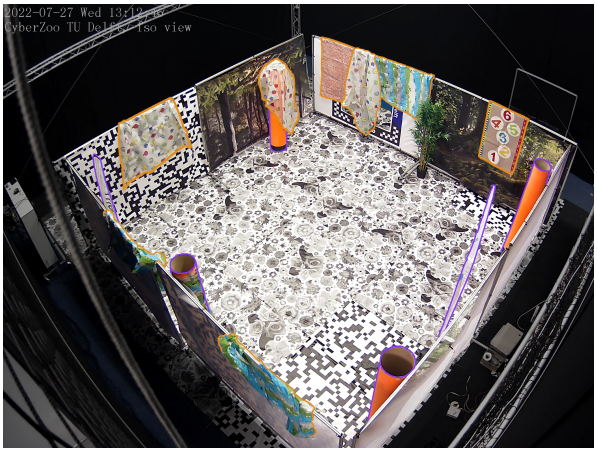


Fig. 7: Inspired by GapFlyt [8], we deliberately let the quadcopter oscillate vertically to generate additional optical flow.

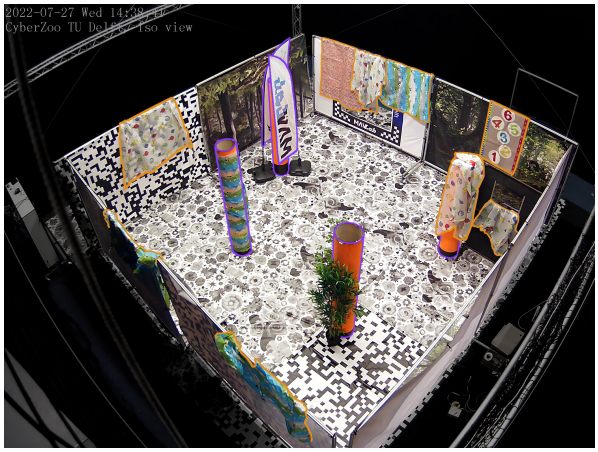
2) AI-deck implementation: The CNN processing power on the AI-deck comes from the GreenWaves Technologies GAP8. The chip is organized around the central single-core fabric controller (FC) and the eight-core cluster (CL) for parallelized workloads. For our application, we run FC@250MHz, CL@230MHz, and VDD@1.2V.

Our AI-deck is equipped with the HM01B0 monochrome camera, which supports a resolution of up to 324x324, a QVGA (244x324) window mode, a 2x2 monochrome binning mode, and cropping. For our application, we enable both the window mode and binning mode (122x162) and take a central crop of 112x160 to ensure a matched spatial resolution of upsampled and skipped features in the network architecture. At our input resolution, using grayscale versus color reduces the L2 memory usage on the AI-deck by 14%. This additional L2 memory is made available to the AutoTiler, which improves inference time by reducing the number of data transfers.

In this work, we utilize the GreenWaves Technologies GAPflow toolset for porting our CNN to the GAP8. NNTool takes a TensorFlow Lite or ONNX CNN description and maps all operations and parameters to a representation compatible with AutoTiler, the GAPflow tiling solver. We use NNTool to implement 8-bit post-training quantization to our CNN. We quantize on images from the MPI Sintel dataset and achieve an average SQNR of 10.



(a) Open environment.



(b) Cluttered environment.

Fig. 8: Overview of the two obstacle avoidance environments in the Cyber Zoo. Obstacles are outlined in purple. Texture-enhancing mats and curtains are outlined in orange.

3) Experimental set-up: The experiments are conducted in 'the Cyber Zoo', an indoor flight arena at the faculty of Aerospace Engineering at the Delft University of Technology. We compose two environments for obstacle avoidance. First, an open environment, with obstacles exclusively placed at the outline of the environment. Second, a cluttered environment, with obstacles placed throughout. Obstacles include textured and untextured poles, artificial plants, flags, or panels. Both environments are enclosed with textured panels to trap the quadcopter inside. Panel textures consist of forest texture, data matrix texture, and a drone racing gate texture. We augment the enclosure's texture in both environments with highly textured mats and curtains. Plants and flags are placed around, and textured curtains are hung on inadequately textured obstacles.

The simple proof-of-concept control algorithm has no dedicated method of dealing with head-on collisions. By placing obstacles around the perimeter of the open environment, we minimize the risk of a head-on collision with the panels as they introduce an imbalance of optical flow, even on a fully perpendicular collision path with a panel.

For each experiment, we start the quadcopter at an approximately identical location, with a varying heading. We let the quadcopter run until a collision or empty battery. We record flight positioning data with an OptiTrack Motion Capture System for post-flight analysis only and record experiments with an ISO view and top view camera.

4) Results: Flight paths extracted from logged OptiTrack position estimates are plotted on maps of the environment and can be found in Fig. 9. The control algorithm is most robust in the open environment, with the quadcopter managing to drain a full battery without crashing. In the cluttered environment, performance is more variable. Especially on occasions where obstacles are close to one another, the quadcopter tends to avoid the first obstacle successfully, only to turn straight into the second and crash into it. Adding a head-on collision detection based on FOE detection and divergence estimation (e.g., [13]) should help avoid obstacles in these cases.

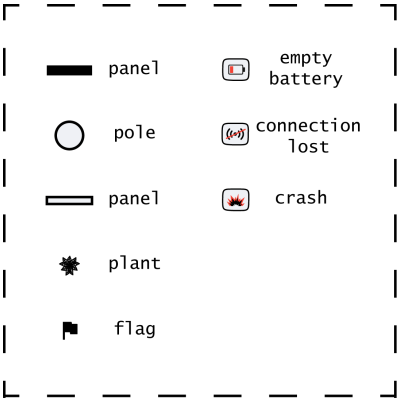
In several successful avoidances, the quadcopter initially responds weakly to the obstacle, only to turn away more harshly when the course has already been corrected sufficiently. This behavior is expected because of two reasons. First, the optical flow due to forward movement is zero at the FOE and maximum at the edge of the peripheral vision. Second, the obstacles generate optical flow across a more significant part of the FOV when near the quadcopter. This behavior could be corrected by weighing the optical flow more heavily towards the center of the image.

Another notable feature of the flight paths is that the quadcopter frequently appears to enter a spiraling path. The control algorithm is overreacting to stimuli from across the environment. Better tuning can help. Despite this, behavior is consistent, the resulting paths are still exploring the environments, and the quadcopter is able to break out of the spiraling motion by approaching a panel (Fig. 9a) or approaching an obstacle (Fig. 9b).

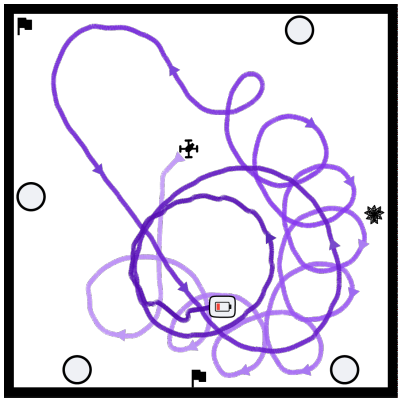
V. Conclusions & Discussion

In this work, we introduced a lightweight CNN architecture for dense optical flow estimation on edge hardware called NanoFlowNet. We achieved real-time latency on the AI-deck. Furthermore, we showed that training our network guided on motion boundaries improves performance at zero cost to latency. Finally, we implemented NanoFlowNet in a real-world obstacle avoidance application on board a Bitcraze Crazyflie nano quadcopter.

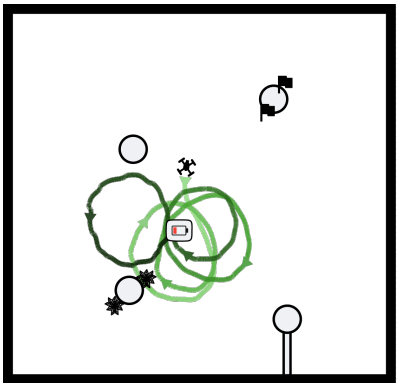
For future work, we expect to see improved performance by fine-tuning on more naturalistic datasets such as MPI Sintel, KITTI, and H1D1. We expect applications that take more advantage of the dense information in the generated optical flow field. The achieved signal to quantization noise ratio could be improved with quantization aware training (QAT) or by considering quantization in the architecture design.



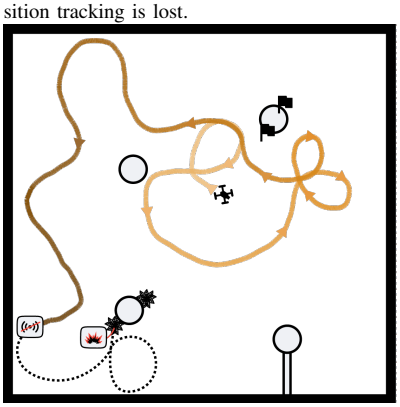
(a) Legend. Connection loss indicates that position tracking is lost.



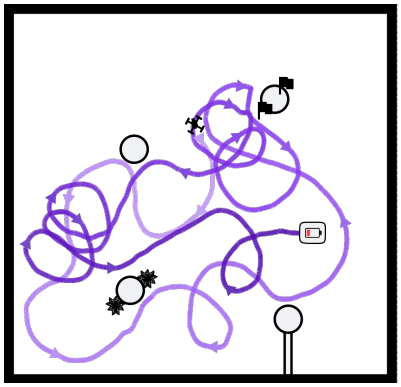
(b) Run in the open environment.



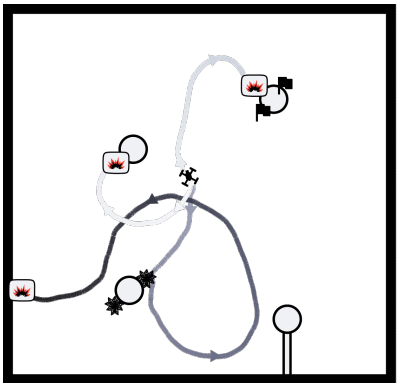
(c) Run in the cluttered environment.



(d) Run in the cluttered environment. Position tracking was lost near the end of the run, and the remainder of the flight path was sketched based on video reference.



(e) Run in the cluttered environment.



(f) Runs in the cluttered environment.

Fig. 9: Obstacle avoidance runs. Positioning logs from OptiTrack are plotted on maps of the Cyber Zoo environments. The simple proof-of-concept control strategy most successfully avoids collision in the open environment. In the cluttered environment, the limitations of the control strategy become apparent, with the quadcopter frequently colliding with a subsequent obstacle after an initial successful avoidance.

REFERENCES

[1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, may 2015.

[2] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Lujan, S. Furber, A. J. Davison, P. H. Kelly, and M. F. O'Boyle, "SLAMBench2: Multi-Objective Head-to-Head Benchmarking for Visual SLAM," in *Proceedings - IEEE International Conference on Robotics and Automation*, sep 2018, pp. 3637–3644.

[3] K. N. McGuire, C. de Wagter, K. Tuyls, H. J. Kappen, and G. C. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, oct 2019.

[4] B. P. Duisterhof, S. Krishnan, J. J. Cruz, C. R. Banbury, W. Fu, A. Faust, G. C. de Croon, and V. J. Reddi, "Tiny Robot Learning (tinyRL) for Source Seeking on a Nano Quadcopter," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2021-May, no. Icra, 2021, pp. 7242–7248.

[5] B. P. Duisterhof, S. Li, J. Burgues, V. J. Reddi, and G. C. De Croon, "Sniffy Bug: A Fully Autonomous Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments," in *IEEE International Conference on Intelligent Robots and Systems*, jul 2021, pp. 9099–9106.

[6] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, may 2019.

[7] P. Gao, D. Zhang, Q. Fang, and S. Jin, "Obstacle avoidance for micro quadrotor based on optical flow," in *Proceedings of the 29th Chinese Control and Decision Conference, CCDC 2017*, jul 2017, pp. 4033–4037.

[8] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermuller, and Y. Aloimonos, "GapFlyt: Active vision based minimalist structureless gap detection for quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2799–2806, 2018.

[9] J. Conroy, G. Gremillion, B. Ranganathan, and J. S. Humbert, "Implementation of wide-field integration of optic flow for autonomous quadrotor navigation," in *Autonomous Robots*, vol. 27, no. 3, oct 2009, pp. 189–198.

[10] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, "MAV navigation through indoor corridors using optical flow," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2010, pp. 3361–3368.

[11] G. C. De Croon, "Monocular distance estimation with optical flow maneuvers and efference copies: A stability-based strategy," *Bioinspiration and Biomimetics*, vol. 11, no. 1, jan 2016.

[12] J. R. Serres and F. Ruffier, "Optic flow-based collision-free strategies: From insects to robots," *Arthropod Structure and Development*, vol. 46, no. 5, pp. 703–717, sep 2017.

[13] G. C. de Croon, C. De Wagter, and T. Seidl, "Enhancing optical-flow-based control by learning visual appearance cues for flying robots," *Nature Machine Intelligence*, vol. 3, no. 1, pp. 33–41, jan 2021.

[14] M. Fan, S. Lai, J. Huang, X. Wei, Z. Chai, J. Luo, and X. Wei, "Rethinking BiSeNet For Real-time Semantic Segmentation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, apr 2021, pp. 9711–9720.

[15] B. D. Lucas and T. Kanade, "Iterative Image Registration Technique With an Application To Stereo Vision." in *Proc 7th Intl Joint Conf on Artificial Intelligence*, 1981, pp. 674–679.

[16] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1-3, pp. 185–203, aug 1981.

[17] A. Dosovitskiy, P. Fischery, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. V. D. Smagt, D. Cremers, and T. Brox, "FlowNet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2758–2766.

[18] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition*, nov 2017, pp. 2720–2729.

[19] 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 - 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1647–1655.

[20] S. Zhao, X. Li, and O. El Farouk Bourahla, "Deep optical flow estimation via multi-scale correspondence structure learning," in *IJCAI International Joint Conference on Artificial Intelligence*, vol. 0, jul 2017, pp. 3490–3496.

[21] D. Sun, X. Yang, M. Y. Liu, and J. Kautz, "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2017, pp. 8934–8943.

[22] T. W. Hui, X. Tang, and C. C. Loy, "LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8981–8989.

[23] Z. Yin, T. Darrell, and F. Yu, "Hierarchical discrete distribution decomposition for match density estimation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, dec 2019, pp. 6037–6046.

[24] G. Yang and D. Ramanan, "Volumetric correspondence networks for optical flow," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[25] T. W. Hui, X. Tang, and C. C. Loy, "A Lightweight Optical Flow CNN - Revisiting Data Fidelity and Regularization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 8, pp. 2555–2569, feb 2021.

[26] T. W. Hui and C. C. Loy, "LiteFlowNet3: Resolving Correspondence Ambiguity for More Accurate Optical Flow Estimation," in *European Conference on Computer Vision*, 2020, pp. 169–184.

[27] S. Zhao, Y. Sheng, Y. Dong, E. I. Chang, and Y. Xu, "Maskflownet: Asymmetric feature matching with learnable occlusion mask," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, mar 2020, pp. 6277–6286.

[28] Z. Teed and J. Deng, "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow (Extended Abstract)," in *European Conference on Computer Vision*, aug 2020, pp. 402–419.

[29] J. Hur and S. Roth, "Iterative residual refinement for joint optical flow and occlusion estimation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, apr 2019, pp. 5747–5756.

[30] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-Level Control of a Quadrotor with Deep Model-Based Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, jan 2019.

[31] A. Briod, J.-C. Zufferey, and D. Floreano, "Optic-Flow Based Control of a 46g Quadrotor," in *Workshop on Vision-based Closed-Loop Control and Navigation of Micro Helicopters in GPS-denied Environments, IROS 2013*, 2013.

[32] R. J. Moore, K. Dantu, G. L. Barrows, and R. Nagpal, "Autonomous MAV guidance with a lightweight omnidirectional vision sensor," in *Proceedings - IEEE International Conference on Robotics and Automation*, sep 2014, pp. 3856–3861.

[33] K. McGuire, G. De Croon, C. De Wagter, K. Tuyls, and H. Kappen, "Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone," in *IEEE Robotics and Automation Letters*, vol. 2, no. 2, apr 2017, pp. 1070–1076.

[34] O. Dunkley, J. J. Engel, J. Sturm, and D. Cremers, "Visual-Inertial Navigation for a Camera-Equipped 25g Nano-Quadrotor," in *IROS2014 aerial open source robotics workshop*, 2014, p. 2.

[35] F. Candan, A. Beke, and T. Kumbasar, "Design and Deployment of Fuzzy PID Controllers to the nano quadcopter Crazyflie 2.0," in *2018 IEEE (SMC) International Conference on Innovations in Intelligent Systems and Applications, INISTA 2018*, sep 2018.

[36] A. Anwar and A. Raychowdhury, "Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning," *IEEE Access*, vol. 8, pp. 26 549–26 560, oct 2020.

[37] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mW Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, apr 2019.

[38] Z. Li, Y. Chen, L. Gong, L. Liu, D. Sylvester, D. Blaauw, and H. S. Kim, "An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration," in *IEEE International Solid-State Circuits Conference*, mar 2019, pp. 134–136.

[39] M. Hosseini and T. Mohsenin, "Binary Precision Neural Network Manycore Accelerator," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 17, no. 2, pp. 1–27, apr 2021.

[40] N. K. Manjunath, A. Shiri, M. Hosseini, B. Prakash, N. R. Waytowich, and T. Mohsenin, "An Energy Efficient EdgeAI Autoencoder Accelerator for Reinforcement Learning," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 182–195, jan 2021.

[41] D. Palossi, F. Conti, and L. Benini, "An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-UAVs," in *Proceedings - 15th Annual International Conference on Distributed Computing in Sensor Systems, DCOSS 2019*, may 2019, pp. 604–611.

[42] D. Palossi, N. Zimmerman, A. Burrello, F. Conti, H. Muller, L. M. Gambardella, L. Benini, A. Giusti, and J. Guzzi, "Fully Onboard AI-Powered Human-Drone Pose Estimation on Ultralow-Power Autonomous Flying Nano-UAVs," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 1913–1929, feb 2022.

[43] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, oct 2017, pp. 1800–1807.

[44] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," apr 2017.

[45] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, jul 2018, pp. 6848–6856.

[46] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, jan 2018, pp. 4510–4520.

[47] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann, "BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs," *CVPR Workshop on Computer Vision for Augmented and Virtual Reality*, jul 2019.

[48] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "BiSeNet: Bilateral segmentation network for real-time semantic segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 325–341.

[49] M. Zhai, X. Xiang, R. Zhang, N. Lv, and A. E. Saddik, "Ad-net: Attention Guided Network for Optical Flow Estimation Using Dilated Convolution," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, may 2019, pp. 2207–2211.

[50] G. Zuo, C. Zhang, J. Tong, D. Gong, and M. You, "Edge Detection-Based Optical Flow Estimation Method," in *2021 IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems, CYBER 2021*, jul 2021, pp. 873–878.

[51] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, aug 2020.

[52] E. Ilg, T. Saikia, M. Keuper, and T. Brox, "Occlusions, Motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 614–630.

[53] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, dec 2015.

[54] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation," in *Proceedings*

of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-Decem, dec 2016, pp. 4040–4048.

[55] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012, pp. 611–625.

[56] K. Souhila and A. Karim, "Optical flow based robot obstacle avoidance," *International Journal of Advanced Robotic Systems*, vol. 4, no. 1, p. 2, mar 2007.

[57] G. Cho, J. Kim, and H. Oh, "Vision-based obstacle avoidance strategies for MAVs using optical flows in 3-D textured environments," *Sensors*, vol. 19, no. 11, p. 2523, jun 2019.

II

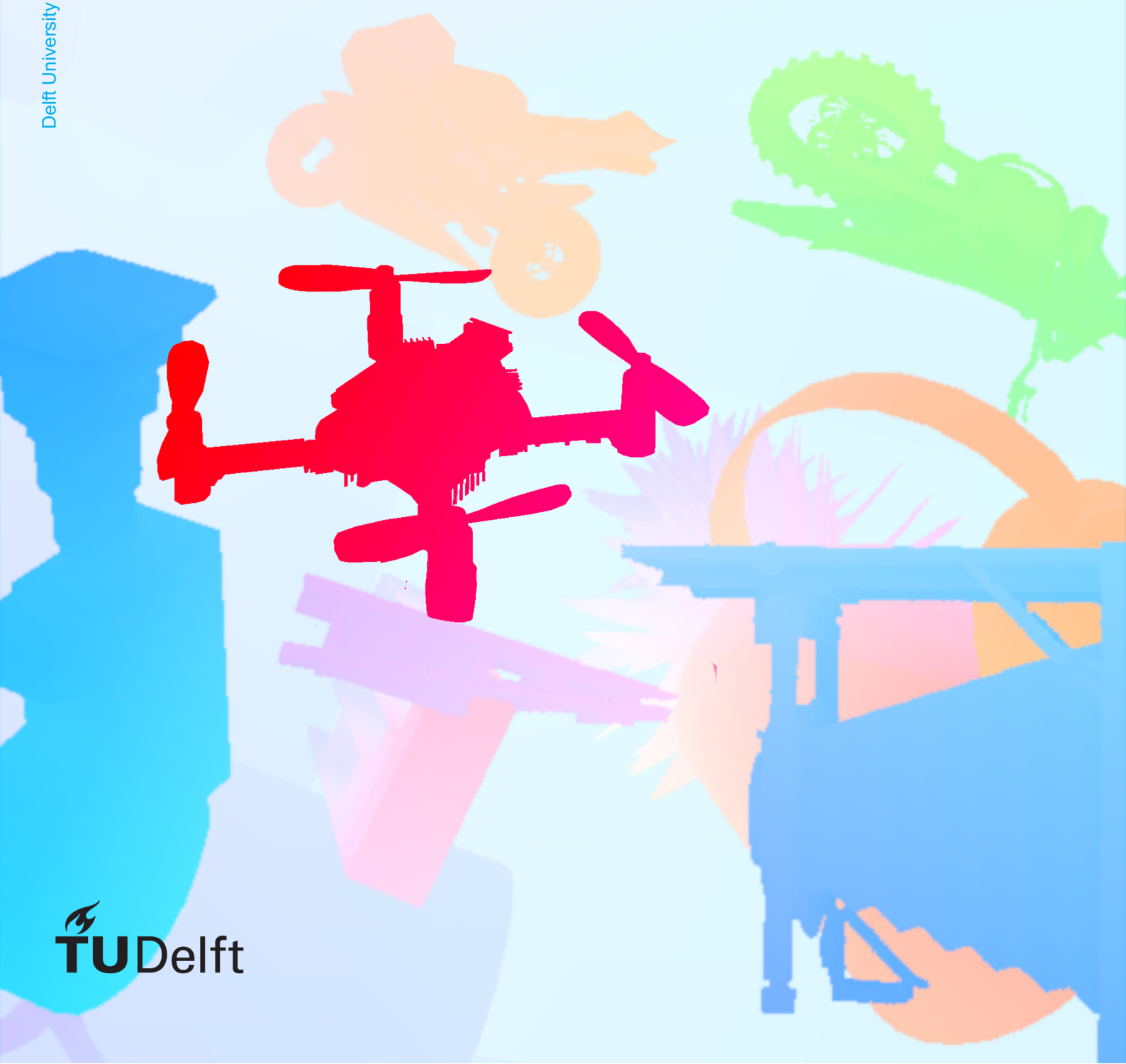
Literature research

Literature Research

Lightweight optical flow CNNs for application on a nano quadcopter

R.J. Bouwmeester

Delft University of Technology



TU Delft

Literature research

Lightweight optical flow CNNs for application on a nano quadcopter

by

R.J. Bouwmeester

in partial fulfillment of the requirements for the degree of
Master of Science Aerospace Engineering
at the Delft University of Technology

supervised by
Prof. dr. ir. G.C.H.E. de Croon, Control and Simulation, LR, TU Delft
Ir. F. Paredes Vallés, Control and Simulation, LR, TU Delft

Abstract

Nano quadcopters are miniature (<100 mm) micro aerial vehicles that are well accommodated for navigation in highly constrained spaces. Optical flow estimation on these flight platforms can be used for (self)motion estimation, monocular depth estimation, tracking, visual odometry, and autonomous navigation. The field of optical flow estimation is dominated by convolutional neural networks (CNNs), both in terms of accuracy and latency. These networks are of substantial size, ranging from 1E+06 to 2E+08 parameters, and achieving a throughput of several to tens of frames per second (FPS) on modern high-end desktop computer graphics processing units (GPUs). Preliminary results show that, in spite of internet of things (IoT) hardware and kernel improvements, the current state of the art optical flow CNNs have a too low throughput for application on a nano quadcopter. A literature survey is proposed, covering i) supervised, end-to-end optical flow estimation CNNs, ii) CNNs from closely related fields designed for inference on IoT hardware, and iii) recent IoT hardware developments that contribute to faster CNN inference on nano quadcopter. We highlight insights that contribute to a CNN architecture with a latency-accuracy trade-off more in favor of high throughput than those architectures presented in current literature. A proposal to research low latency optical flow CNNs for on-board inference is presented.

Contents

List of Figures

List of Tables

Nomenclature

CNN	Convolutional neural network
EPE	End-point error
FPS	Frames per second
GPU	Graphics processing unit
GRU	Gated recurrent unit
ICP	Iterative closest point
IoT	Internet of things
MAC	Multiply-accumulate operations
MAV	Micro aerial vehicle
PULP	Parallel ultra-low power
RISC	Reduced instruction set computer
RNN	Recurrent neural network
SDK	Software development kit
SoC	System on a chip

Chapter 1

Introduction

Optical flow is a concept first introduced by Gibson et al. [1]. It describes the apparent visual variations caused by relative motion between the observer and its surroundings. Nano quadcopters are miniature (<100 mm) micro aerial vehicles that are well accommodated for navigation in highly constrained spaces. Optical flow estimation on these flight platforms can power (self)motion estimation, monocular depth estimation [2], tracking, visual odometry, and obstacle avoidance [3].
The field of optical flow estimation has been long dominated by variational methods, introduced by Horn et al. [4]. However, like many computer vision fields, optical flow has been significantly affected by recent developments in deep learning. Several proposed methods use convolutional neural networks (CNNs) as feature extractors [5–8]. The first fully end-to-end optical flow CNN was proposed by Dosovitskiy et al. [9]. Their network, *FlowNet*, achieved state-of-the-art accuracy among real-time methods, but failed to match the most accurate of the variational methods. This network has been the first in a series of quickly improving networks, and the current state-of-the-art of flow estimation is dominated by end-to-end CNNs. These networks are of substantial size, ranging from 1E+06 to 2E+08 parameters, and achieving a throughput of several to tens of frames per second (FPS) on modern high-end desktop computer graphics processing units (GPUs).

Simultaneously, advancements in reduced instruction set computer (RISC) microprocessors and kernels have been increasingly enabling embedded devices to employ neural networks. The ultra-low power GAP8 [10] by GreenWaves technologies[1] has been designed for CNN inference. Improved neural network kernels [11, 12] allow for even faster inference. Palossi et al. [13] propose *PULP-Shield*, an expansion deck for a CrazyFlie 2.0. The PULP-Shield is based on a GAP8 SoC, and deployed an autonomous navigation CNN onboard. In collaboration with ETH Zurich, BitCraze[2] released a commercial off-the-shelf expansion deck, the *AI-deck*, for their CrazyFlie 2.X nano quadcopter series, based on the design of the PULP-shield, allowing faster network inference on the nano quadcopter. Most proposed optical flow networks in literature focus on improving prediction accuracy while maintaining throughput on high-end desktop computer hardware in the order of 10s of FPS. Despite RISC performance improvements, real-time application of optical flow CNNs on a GAP8 requires much higher throughput.

[1] `https://greenwaves-technologies.com/`
[2] `https://www.bitcraze.io/`

The objective of this research is to design an end-to-end CNN architecture for real-time optical flow estimation on a nano quadcopter. Real-time optical flow estimation on a nano quadcopter can power (self-)motion estimation, monocular depth estimation, autonomous navigation, tracking, and visual odometry, requiring only the camera most quadcopters are already equipped with.
This work might contribute to the field with insights into optical flow CNN miniaturization. The extremely limited hardware brings the project back to the essentials of optical flow estimation, with no room to brute force a result. Its contributions may be extended to optical flow CNNs in general, and other computer vision CNN tasks such as pose estimation, semantic segmentation, super-resolution.

1.1 Scope

There are a large number of methods for optical flow estimation. This review only concerns optical flow estimation using CNNs. Insights from traditional (e.g., variational) methods of optical flow estimation can contribute to methods using CNNs, but the pool of articles concerning optical flow estimation using CNNs is vast and must be prioritized.

Several multi-frame networks are proposed [14–17]. These networks can yield accuracy improvements by using more than two input images, but the significant performance impact makes these methods unsuitable for the very limited hardware on nano quadcopters. This review will as such not explore multi-frame networks.

The flight platform considered in this document is a CrazyFlie 2.0, equipped with an AI-deck, designed for neural network inference. This literature review will be based on the computational constraints of this specific platform. The literature review proposes a comparison between this and other popular flying computational platforms.

1.2 Report Structure

In Part I, a literature review is provided. Chapter 2 introduces this work's definition of optical flow, optical flow evaluation metrics, optical flow datasets, the state-of-the-art CNN architectures and training techniques. Finally, an overview of state-of-the-art benchmarking performance is provided and discussed. Chapter 3 treats literature from related CNN tasks that contribute to lower latency for application on edge devices. In chapter 4, the flight and computing platform used in this research are presented. A comparison with similar platforms is proposed.
In part II, a research proposal is presented. In chapter 5, research questions and objectives for the remainder of the project are given. Chapter 6 introduces the methodology that will be used to answer these research questions and achieve the objectives given, it also details the experimental set-up used to implement the methodology and lists the expected results, how to interpret them, how to verify and validate the work, and the relevance of the outcome of the experiments.
Part III covers preliminary results. In chapter 7, the inference time of an encoder from one of the fundamental optical flow CNNs, FlowNetS, is benchmarked on the GAP8. Two

insights that follow from the literature review are applied and benchmarked, too. Finally, the report is concluded in chapter 8.

Chapter 2

Optical Flow Estimation

2.1 Optical Flow Definition

Optical flow is a concept first introduced by Gibson et al. [1]. It describes the apparent visual variations caused by relative motion between the observer and its surroundings. In a computer science context, optical flow is described as a series of vectors that describes the movement of pixels between two input images, a source and target image. There is a distinction between dense optical flow, for which optical flow is calculated for every pixel in the image frame, and sparse optical flow, for which optical flow is calculated for certain detected features. Dense optical flow estimation is more computationally-intensive but the complete representation of optical flow allows it to be deployed for a much wider variety of applications than sparse optical flow. This work covers dense optical flow, which from here on is often simply referred to as optical flow. In optical flow, the motion field [18] of the 3D motion as projected on our observer is determined, rather than the actual 3D motion of the scene (*scene flow*). As a direct result of being the flow of a projection, the resulting flow vectors give information on the ratio between the relative velocity between observer and object and the distance between them. With an external source of either velocity or distance information, the opposing quantity in the ratio can be estimated.

2.2 Optical Flow Estimation Challenges

Occlusions are a large and inevitable problem in optical flow estimation. It is in the nature of the problem that several pixels in the source or target image are occluded in the other image, leading to ghosting effects and a higher estimation error.

A limited receptive field size can cause the aperture problem, where without capturing certain contours of an object, its motion cannot be unambiguously detected. In the context of optical flow estimation, this can translate to missing flow information parallel to the flow direction of objects with large untextured surfaces or ambiguous direction of motion.

Most optical flow estimation methods have low robustness when it comes to illumination changes. Classical methods involve a brightness constancy constraint, and CNNs are trained on datasets without illumination changes. However, in most applications, illumination changes should be minimal between two frames.

2.3 Optical Flow Evaluation Metrics

A commonly used metric for benchmarking and training in dense optical flow is the end-point error (*EPE*). The EPE is the Euclidean distance between the predicted flow vectors (v_p) and the ground truth vectors (v_{gt}), as in equation 2.1. In most works, the EPE is synonymous with the end-point error averaged over all pixels. Another common metric is the F_β score. F_1 is relatively common in benchmarking, and F_2 is commonly used as training loss. This metric uses type I and type II errors (true/false positive/negative - t/f p/n). Any F_β score is calculated as in equation 2.2. Some papers [19–21] report more detailed accuracy metrics, such as the EPE for different flow velocities and distances from motion boundaries. These metrics provide insight into the proposed networks' strengths and weaknesses (e.g., difficulty estimating high-velocity optical flow) and a such are vital in the design of any network.
Low latency and high throughput is crucial to this project, considering the step down in computing power from desktop computers to nano quadcopters. Latency or inference time is the time between inputting an image (pair) and receiving an output flow. The inverse of that is the refresh rate or the number of frames per second (FPS). These metrics are highly dependent on the hardware the algorithm is run on.
The required accuracy (e.g., *EPE*, F_1) and inference time depend highly on the application.

$$EPE = |v_{gt} - v_p| \tag{2.1}$$

$$F_\beta = \frac{(1 + \beta^2 \cdot tp}{(1 + \beta^2) \cdot tp + \beta^2 \cdot fn + fp} \tag{2.2}$$

2.4 Optical Flow Datasets

As there is no way to directly and accurately extrapolate optical flow ground truth from a random image sequence, the challenge of obtaining such data has been the origin of many papers. The first effort to generate the optical flow ground truth data required to evaluate optical flow algorithms is by Barron et al. [22]. The data consists of real images with parametric transformations applied and synthetic image pairs, for which ground truth optical flow is extracted from construction parameters. To address the lack of large displacements and lack of discontinuities (Barron et al.'s flow sequences contain mostly smooth flow fields, without motion borders), Baker et al. propose the *Middlebury* dataset [23]. Several image pairs are generated using real images, shot in a strictly controlled environment, providing ground truth. To compensate for Middlebury's lack of size, complexity, and variety, Butler et al. propose the *MPI Sintel* dataset [24]. The dataset is derived from the open-source 3D animated film *Sintel* and includes long sequences, large motions, and realistic lighting effects. The dataset is rendered at three levels of complexity. An *Albedo Pass* with flat, unshaded surfaces and constant albedo, a *Clean Pass* with smooth shading and specular reflections, and a *Final Pass* with all effects such as blur due to depth of field, motion, and atmospheric effects. The sequences are compared to real video footage to make the data as naturalistic as possible. Naturally, the Final Pass is the most naturalistic out of the three render complexity options. Geiger, Lenz, et al. propose the *KITTI* dataset [25], comprising several real image pairs recorded from a car. Semi-dense (50%) ground truth is generated using an iterative closest point

(ICP) algorithm, using the car's equipped laser scanner. To provide ground truth data for scene flow, Menze et al. propose a new KITTI dataset [26], generated using raw data provided by Geiger, Lenz, et al. The dataset also includes significantly more image pairs and, like the original KITTI dataset, optical flow ground truth. To discern between the two datasets, the publishing year is appended, resulting in *KITTI 2012* and *KITTI 2015*, respectively.

Table 2.1: An overview of common optical flow datasets. (* varying resolutions, indicated resolution accurate for most frames)

Dataset	***No. training frames***	***No. test frames***	***Resolution***	***Optical flow ground truth density***
FlyingChairs	22872	-	512x384	100%
FlyingThings3D	21818	4248	960x540	100%
MPI Sintel	1064	564	1024x436	100%
KITTI 2012	194	195	1240x376	50%
KITTI 2015	800	800	1242x375	50%
Middlebury	74	-	640x480*	100% (for 8 frames)

The Middlebury, MPI Sintel, and KITTI datasets' size is limited, particularly in the context of training optical flow CNNs. To compensate for the lack of available training data, Dosovitsky et al. [9] propose a new synthetic training dataset, with over 20 times the image pairs with ground truth optical flow than any previous dataset, called *FlyingChairs*. The dataset consists of images with one or several chairs projected on a real image as background. The image pair is generated by applying affine transformations to both the chairs and the background. Ground truth is generated using the associated transformation parameters. Mayer et al. follow a similar approach and aim as Butler et al. to propose *FlyingThings3D* [27], which is contains more than twenty times the training frames of the MPI Sintel dataset. FlyingThings3D is created using the free and open source creation suite *Blender*[1], allowing true 3D motion (in contrast to FlyingChairs) and the extraction of ground truth optical flow. Table 2.1 contains an overview of the treated datasets and the number of training/test frames, the image resolution and the density of the optical flow ground truth.
Figures 2.2 through 2.6 show the first image of an image pair and the corresponding optical flow ground truth for several of the mentioned datasets. The optical flow is visualized using the color coding of figure 2.1 or similar, where e.g. blue corresponds to a up and left moving pixel, and the intensity of the color corresponds to the magnitude of the displacement. There are slight inconsistencies between papers in the used color coding. Several works use a vertically flipped version of figure 2.1, or a gradient from white to full intensity colors, rather than from black.

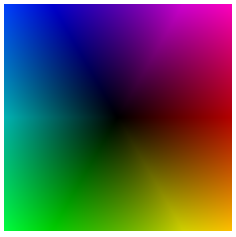
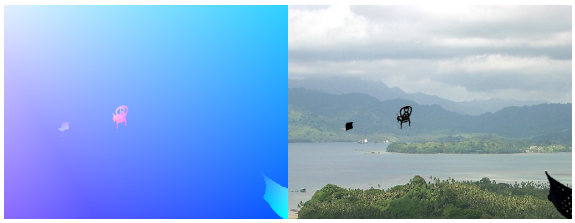


Figure 2.1: Optical flow color coding

[1] `https://www.blender.org/`

(a) Flow ground truth (b) Image 1

Figure 2.2: Middlebury [23]



(a) Flow ground truth (b) Image 1

Figure 2.3: FlyingChairs [9]

(a) Flow ground truth (b) Image 1 (Final) (c) Image 1 (Albedo) (d) Image 1 (Clean)

Figure 2.4: MPI Sintel [24]

(a) Flow ground truth (b) Image 1

Figure 2.5: KITTI 2015 [26]

(a) Flow ground truth (b) Image 1

Figure 2.6: FlyingThings3D [27]

2.5 Supervised End-to-end CNN Optical Flow Estimation

The first proposed convolutional neural network to effectively estimate optical flow, trained end-to-end, is *FlowNet* [9]. The model is based on a U-Net [28] shaped convolutional neural network. Dosovitskiy et al. propose two variants of the network; *FlowNetS*, which concatenates two images to a single input, and *FlowNetC*, which inputs each image to either one of two identical encoder parts and employs a correlation layer to combine the resulting feature maps. The architecture of FlowNetS can be seen in figure 2.7. FlowNetC's correlation layer promises more accurate results to compensate for its significantly longer inference time. FlowNetC's architecture is similar to the FlowNetS' architecture in figure 2.7, but with the input images entering separate but identical encoders up to `c3`, after which they are combined in the correlation layer. The rest of the architecture is identical. The correlation layer calculates the correlation between image patches. With equation 2.3, the correlation between two such patches, centered around $\mathbf{x}_1$ and $\mathbf{x}_2$, and with patch size $2k+1$ by $2k+1$, can be calculated. A limit is put on the maximum displacement between compared patches, and a stride is introduced to limit the computational expense. The four-dimensional results ($w \times h \times w \times h$) of the correlation layer are organized into a three-dimensional tensor by folding out the displacement into the channel dimension.
Best benchmark results were obtained by first extensively training on the relatively simple but large FlyingChairs dataset, and then finetuning on a dataset of the benchmarking environment, rather than train on the environment directly, or training on a mix of datasets. Dosovitskiy et al. show that directly training on the more realistic data by skipping training on FlyingChairs deteriorated results. They hypothesize that by training on the simpler datasets, the networks learn the base concepts, which the network has a hard time learning if more complex visuals are introduced from the start. Neither FlowNetS nor FlowNetC achieve state of the art accuracy but have low inference time compared to most traditional methods, thus providing a promising result to spark further research.
De Jong et al. [29] show that reducing deepest level kernel sizes of FlowNetS reduces the receptive field size and thus the network's ability to solve the aperture problem. They also show that the decoder part of the network is, to a certain extent, able to extrapolate motion cues to the center of the moving object.

$$c(\mathbf{x}_1, \mathbf{x}_2) = \sum \langle \mathbf{f}_1(\mathbf{x}_1 + \mathbf{o}), \mathbf{f}_2(\mathbf{x}_2 + \mathbf{o}) \rangle \tag{2.3}$$
$$\mathbf{o} \in [-k, k] \times [-k, k]$$

Ilg et al. [30] improve on FlowNet in several ways. They propose several new networks, built out of one or several stacked units of FlowNetS and/or FlowNetC (e.g., *FlowNet2-SS*). Stacking the networks helps refine the flow. Based on FlowNetS, a network to better deal with small displacements, called *FlowNet-SD*, is proposed. The network replaces the first three convolutions of FlowNetS with relatively large kernel size with several smaller kernel size convolutions with a lower stride. Convolutions are added between transposed convolutions in the decoder part to smoothen the estimates. This architecture effectively deals with noise. Combining a several stacked units (*FlowNet2-CSS*) and a small displacement unit (FlowNet-SD) results in *FlowNet2*. This final proposed network achieved state of the art accuracy, with significantly lower latency than most classical methods.
The authors make a trade-off between flow prediction accuracy and inference time by varying the number of channels for FlowNetS, for all layers' channels multiplied with the

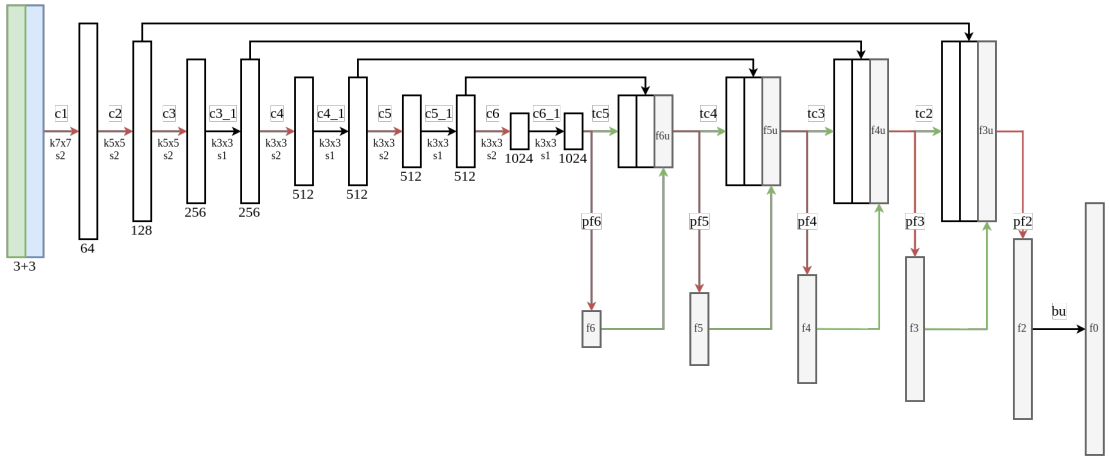


Figure 2.7: FlowNetS schematic architecture. Convolutions `c`, kernel sizes `k`, strides `s`, flow prediction convolutions `pf`, predicted flows `f`, upsampled predicted flows `fu`, transpose convolutions `tc`, and bilinear upsampling layer `bu`. The numbers under the feature maps indicate the number of channels.

same factor, simultaneously, and with the prediction accuracy in EPE measured only for Sintel Clean during training. Below a factor of 3/8, the latency improvement is small, and the EPE increases more significantly. The results are visualized in figure 2.8. Networks with 3/8th the number of hidden layer channels are named *FlowNet2-s* and *FlowNet2-c*. The paper proposes several training practices (improved learning rate scheduling, use of additional training dataset FlyingThings3D [27]) that significantly improve the results of this network, and FlowNetC and FlowNetS alike, and have become standard for the field.

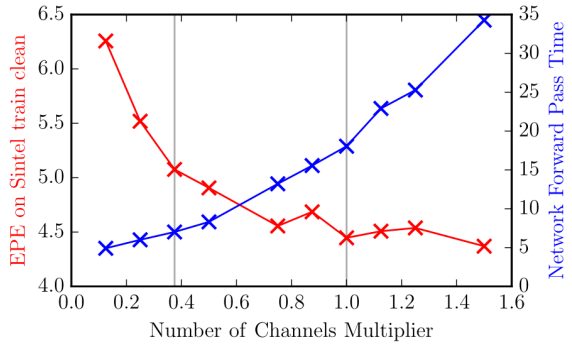


Figure 2.8: "Accuracy and run-time of FlowNetS depending on the network width. The multiplier 1 corresponds to the width of the original FlowNet architecture. Wider networks do not improve the accuracy. For fast execution times, a factor of $\frac{3}{8}$ is a good choice. Timings are from an Nvidia GTX 1080." [30]

Ranjan et al. propose a network that treats optical flow in a coarse-to-fine pyramid structure called *SPyNet* [19]. The network is built up out of five pyramid layers, with a further downsampled input image each coarser layer. The flow estimate of a coarser layer is both used to warp the and is concatenated onto one of the two input images to the

next, finer layer. The warping allows the next, finer layer to compute only the residual flow. Thanks to the *image pyramid* structure, the individual networks do not need to handle large motion, which solves the need for a big receptive field per layer. Each layer predicts flow using a sub-network consisting of five regular convolutions with shallow feature maps (FlowNet has 64 feature maps after a single convolution, whereas SpyNet has 64 feature maps at most), and large kernel size compared to FlowNet. Using no stride and zero padding, each convolution retains the same spatial resolution. Feature maps channel width goes from shallow to deep to shallow. Layers can be trained individually, but flow predictions of coarser layers must be accounted for. The architecture reduces the number of model parameters by 96%, compared to FlowNet, while achieving equal or slightly better results. Inference time is slightly lower than that of FlowNetS.
The low number of parameters is particularly relevant for memory-limited applications. The reduction is mostly obtained due to the residual structure of the network. A smaller part of the reduction is due to the direct implementation of a warping layer, not requiring the network to learn it. This still remains one of the networks with the lowest number of parameters.

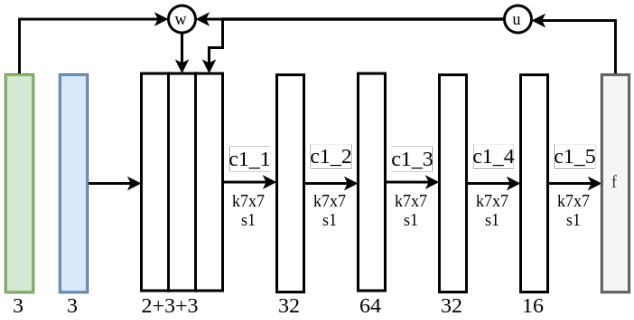


Figure 2.9: SPyNet architecture schematic of a single pyramid layer. Convolutions `c`, kernel sizes `k`, strides `s`, predicted flows `f`, warping operations `w`, upsampling operations `u`. The numbers under the feature maps indicate the number of channels. The network initializes the lowest spatial resolution flow estimate at 0.

Shanshan Zhao et al. propose a multi-scale correspondence structure learning (*MSCSL*) approach [31]. Pyramid-shaped architectures with coarse to fine layers have a learned fixed correspondence range per layer, which is undesirable as video by definition has varying properties and scale of displacements. These architectures also propagate error from the coarser to finer layers. The proposed network deals with these problems by concatenating all different (coarse to fine) dependency maps onto one of two input frames and making a flow prediction out of this, rather than only taking the finest prediction (that only carries the information of coarser predictions through image or feature warping as in, e.g., SPyNet, and not concatenating). Rather than using downsampled images as in SPyNet, the network uses downsampled feature maps using a series of convolutions followed by, for each representation, a different convolution with varying stride, with or without pooling to generate varying spatial resolution representations. The network uses gated recurrent units (GRUs) from the field of recurrent neural networks (RNNs) to incrementally update a stack of intra-level dependency maps. Rather than using fully connected layers, these

GRUs use convolutions.

Flow prediction results are slightly better than for SPyNet. Still, whether using a *feature pyramid* rather than an image pyramid, the resolutions for error propagation, or the reintroduction of a correlation layer contribute most to the improved results is unclear.

Concurrently, Sun et al. propose *PWC-Net* [20], which also employs feature maps generated with convolutions, rather than downsampled images, as inputs to the coarser layers. Unlike MSCSL's concurrent convolutions, this feature pyramid is generated with sequential convolutions. Similar to SPyNet, the predicted flow from coarser levels is used to warp one of the two feature maps. Every PWC-Net level has a cost volume layer, which holds the matching cost for matching pixels in the two frames, defined as the correlation between the feature map of the first, and warped feature map of the second input image. This layer is processed to estimate the flow. Having a correlation layer at every pyramid level allows for a smaller matching distance, especially since feature maps are warped to each-other, to improve run-time performance. Flow estimation feature maps are twice as deep as SPyNet, and in contrary start deep (128 channels) and reduce throughout. The convolutional layers use *DenseNet* [32] connections, shown to be easier to train and produce more accurate results. A smaller network variant without DenseNet connections, *PWC-Net-small*, is proposed. DenseNet layers improve results when fine-tuning is applied and deteriorate results otherwise. The second last flow prediction is fed through a context network that refines the flow with dilated convolutions, allowing the use of more contextual information [33, 34]. PWC-Net is the first network to outperform traditional methods, both in prediction accuracy and run-time performance.

In a follow-up article [35], Sun et al. improve the training procedure and achieve significantly better results. The retrained model is named *PWC-Net+* to distinguish between the two differently trained but otherwise identical networks. The training improvements include horizontal flips, the removal of added Gaussian noise and a sudden increase in learning rate halfway through training. They also retrain FlowNetC and achieve better results on some benchmarks than reported for FlowNet2 (which includes FlowNetC as a sub-network), proving the importance of training. The retrained FlowNetC and FlowNetS architecture are named *FlowNetC+* and *FlowNetS+*, respectively.

Hui et al. propose *LiteFlowNet* [36], which has a similar architecture to PWC-Net. The differences lie in the addition of flow regularizing convolutions. This allows per-layer regularization using flow estimate, feature maps and an occlusion probability map. These convolutions sharpen boundaries, and smooth the rest of the flow estimate. In a follow-up paper, Hui et al. propose *LiteFlowNet2* [37], with modifications to the final flow inference that improves both run-time performance and accuracy. In the third update, *LiteFlowNet3* [38], accuracy is improved by imposing local flow consistency conditions and removing outliers in the cost volume at the cost of inference speed.

The 'lite' in the name is mostly true when comparing to FlowNet2. LiteFlowNet achieves significantly better results than FlowNet2 while achieving more than a third decrease in inference time. However, PWC-Net achieves similar flow prediction results with another three times decrease in inference time. LiteFlowNet2 is the best performing out of the three variants in terms of latency, with approximately 130% of the inference time of PWC-Net, while achieving better flow prediction results in most benchmarks. The authors propose *LiteFlowNetX*, a lightweight version of LiteFlowNet. This architecture

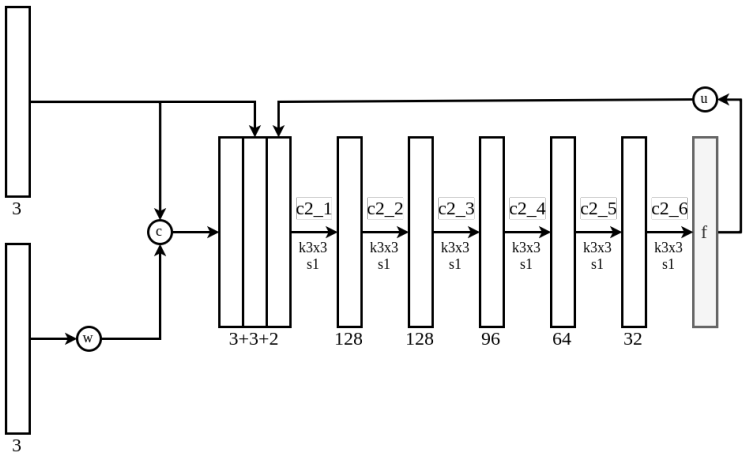


Figure 2.10: PWC-Net architecture schematic of a single pyramid layer. Convolutions `c`, kernel sizes `k`, strides `s`, predicted flows `f`, warping operations `w`, correlation layer construction operations `c`, and upsampling operations `u`. The numbers under the feature maps indicate the number of channels. The network initializes the lowest spatial resolution flow estimate at 0. Context network (at highest spatial resolution pyramid layer) not shown.

has fewer parameters than SPyNet, while outperforming it. The reported run-times for LiteFlowNetX are only slightly lower than those for LiteFlowNet2.

In particular LiteFlowNet2's modifications of the final flow inference are relevant. By measuring the EPE and run-time at individual pyramid levels, the authors were able to identify an exponential increase in run-time per layer and identified that such an exponential increase in run-time was not worth the relatively small increase in EPE of the final flow inference level (of the highest spatial resolution feature pyramid level). The authors remove this layer of the network, and to compensate for the loss of EPE, add two convolutional layers to all remaining flow decoders. This shows the power of detailed performance metrics.

Yang et al. [39] propose several simple modifications relating to the cost volume calculations. They promise fewer parameters and lower computation time at no loss of accuracy by approximating 4-dimensional convolutional operations with 2-dimensional convolutions. Their proposed *VCN* network can handle true volumetric (4D) processing of cost volumes at increased efficiency. Additionally, they propose multi-channel cost volumes to handle multi-channel pixel matching. The proposed network requires significantly fewer training iterations, few parameters, and promises lower GFLOPs while achieving state of the art accuracy. However, the lower GFLOPs currently do not translate to lower inference time, as kernel and hardware support for convolutions and poolings of non-standard shapes is limited.

Shengyu Zhao et al. propose *MaskFlowNet* [40], which improves flow estimation accuracy at negligible run-time performance cost by filtering the image immediately after feature warping with a learnable occlusion map (without occlusion ground truth). The final

flow and occlusion predictions are concatenated and fed into a refinement network that further refines the flow with the occlusion predictions. An alternative network, without refinement, called *MaskFlowNet-S* is proposed. MaskFlowNet-S has a similar architecture and run-time performance as PWC-Net but achieves higher accuracy predictions. MaskFlowNet improves on these results at a considerable cost of run-time performance. The authors hypothesize that the learnable occlusion maps only contribute significantly to the achieved accuracy when used in the refinement network. Feature warping and occlusion masking is done on one of each pyramid level's two input feature maps. The authors show that the network benefits from the big differences between the two input feature maps that result from the asymmetry in the architecture, despite that the fact that these feature maps are to be compared.

Yin et al. propose the probabilistic framework *HD*^{3} [41]. This framework generates the global match density of pixel correspondence between input images. Where all previously mentioned networks directly regress into optical flow, this framework uses the global match density to convert to optical flow. Naturally, this framework can output the confidence of an estimation. Such information is relevant, but run-time performance is poor (although the authors do not disclose what hardware is used to achieve the reported timings, a questionable decision). Flow prediction accuracy, while good, is not worth the high run-time.

Hur et al. propose an iterative refinement scheme called *IRR* [42], which can be applied to numerous base optical flow architectures. The refinement scheme shares parameters between refinement iterations and can improve flow estimation accuracy while maintaining the same number of parameters. The refinement scheme is applied to both PWC-Net (*PWC-Net+IRR*) and FlowNetS (*FlowNetS+IRR*[2]). For PWC-Net-IRR, the authors lower the number of parameters by 2.39M while significantly increasing the prediction accuracy. Each refinement step uses the predicted flow from the previous step to warp the feature maps of one of the two input images. Of course, PWC-Net already employs such refinement between pyramid levels, but with varying weights per decoder level. PWC-Net+IRR replaces the various per-level decoders with a single, shared decoder and iterates over the pyramid levels. The context network is now used every iteration, at every pyramid level. A single 1x1 convolution is added at the decoder's start to equalize the number of channels that enter the encoder for every pyramid level.
The final proposed networks, *IRR-PWC* and *IRR-Flownet*, include bi-directional flow estimation and occlusion prediction (trained on ground truth occlusion). These additions improve flow prediction accuracy at the cost of using 1.87 and 1.40 times the parameters of PWC-Net+IRR and FlowNetS+IRR, respectively. The authors do not mention run-time whatsoever. FlowNetS+IRR must take significantly longer to run, as each iteration requires almost a full pass of the network, except for part of the encoder. PWC-Net+IRR should see a small increase in run-time, with the addition of the 1x1 convolution and context network at every pyramid level. The findings of this work can be applied to refine flow at the cost of run-time performance while maintaining the number of parameters and limiting memory usage.

Bar-Haim et al. [21] show the powerful effects of regularization and data augmentation on optical flow CNN training. Their presented network *ScopeFlow* uses IRR-PWC with a new

[2]Hur et al. do not name this network, this name is in line with naming for PWC-Net variant

proposed cropping augmentation and generally applicable training schedule, significantly improving results without a change in architecture. The authors propose future research to apply the augmentation scheme and training schedules to other architectures, such as FlowNet-like ones.

The authors identify a large bias towards pixels in the center of an image when fixed size cropping is applied, see figure 2.11. Pixels of fast moving objects are often closer to the borders of an image, and thus sampled much less frequently. The authors minimize the sampling bias by introducing a varying zoom range and random crop. Additionally, removing of regularization (such as random noise, weight decay) during fine-tuning, helped improve results further.

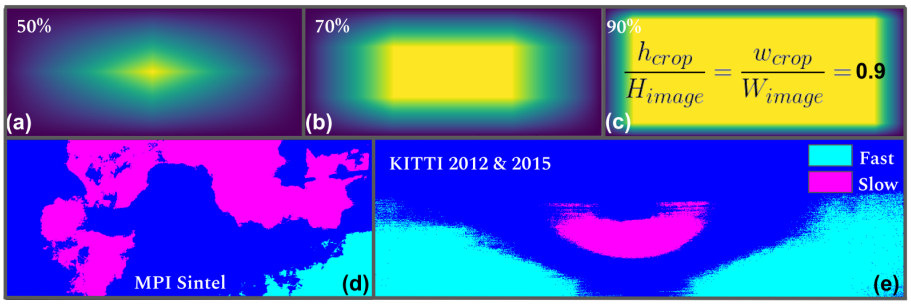


Figure 2.11: "Sampling bias caused by fixed size random crops. (a),(b),(c): Pixel sampling probability maps for a fixed sized crop, with ratios of 50%, 70% and 90% respectively, for each axis. The probability to sample a marginal pixel shrinks drastically with the crop size. (d),(e): areas with strong prevalence for motion categories. High velocities tend to start from lower corners, while small ones tend to occur in the middle and upper part of the scene." [21]

Teed et al. propose *RAFT* [43], which uses convolutional GRUs to incrementally update the flow prediction using 4D correlation volumes. Input images are fed into separate feature encoders, a single 4D correlation volume is created, and pooling is used to create reduced spatial resolution correlation volumes. Both the 4D correlation volumes and the feature maps from a separate context encoder are fed into a series of GRUs. Unlike Zhao et al.'s MSCSL approach [31] (which interestingly, the authors do not refer to), Teed et al. use GRUs to directly update a flow prediction, initiated at zero or with the previous frame pair's flow prediction (warm-start). The idea of using RNNs on the feature maps generated by a CNN draws similarities with IRR, which also iteratively refines the flow. However, rather than using the varying resolution 4D correlation volumes to generate flow at varying resolutions, RAFT consistently updates a single, full-resolution flow prediction. This reduces error propagation, improves fast flow detection, and vastly (a factor 35 compared to FlowNet2) reduces the number of required training epochs.

De Jong et al. [29] exploit feature visualization to show that over half of deepest-level features of *FlowNetS* can be represented by Gabor filters. Detect translational, dilation, rotation, occlusion filters. This work provides insights into how CNNs actually estimate optical flow, which most papers overlook. Gabor filters could be fed directly into the network, effectively shrinking the networks size.

2.5.1 Self-reported Results

Given the memory and computing power constraints of a nano quadcopter, the most important insights from literature are those relating to a decrease in run-time and number of parameters. Although authors rarely report memory usage, a decrease in number of parameters translates to a decrease in memory usage, too. Theoretically, an insight that causes an increase in flow estimation accuracy at a fixed parameter and/or run-time budget could be used instead for a decreased parameter and/or run-time budget at a fixed flow estimation accuracy, but given the highly non-linear nature of CNNs, it is not possible to extrapolate such information. In this section, the most revealing networks from literature are discussed with respect to this trade-off between the run-time budget, parameter budget and flow estimation accuracy.

Self-reported run-times, number of parameters, and benchmarked inference accuracy are available for most proposed networks. Though these are gathered on varying (but all comparatively powerful) hardware, they provide some insight. For many network configurations, only training results are reported, with test results being reported only for the best-performing configurations. The MPI Sintel Final and the KITTI datasets are among the most naturalistic of all datasets considered. They are thus most comparable to real-life scenarios that a nano quadcopter might encounter. An overview of self-reported results, inference-times, and the number of parameters for supervised, end-to-end optical flow CNNs can be found in table 2.2. A dash (-) indicates that the value is not reported. A citation indicates a data source other than the original paper. Figures 2.12 through 2.15 show the benchmark results for MPI Sintel Clean, MPI Sintel Final, KITTI 2012, and KITTI 2015, respectively. In each of the graphs, a Pareto front is identified (blue line). Note that most authors do not vary parameters or do an ablation study, and thus that by varying parameters of individual networks, the drawn Pareto front may change. E.g., we do not know how SPyNet would perform if it had a parameter budget similar to PWC-Net. The dots are grouped and colored by base architecture. For example; PWC-Net, PWC-Net-small, IRR-PWC, PWC-Net + IRR are all colored the same.

`ft` indicates that the network was fine-tuned (trained) on the benchmarked dataset. These values should be disregarded when looking at training results, as the network was trained on the same data. For fine-tuned networks, only consider test results. The trend of testing networks on the train set of a benchmark is odd in itself. It is valid, as long as the network is not trained on that train set, but at the same time, it is unnecessary given the large test sets that are included in the same benchmarks and confusing to readers. Many authors actually report results for networks that were fine-tuned on the same data they are tested on, likely because of the resulting low error (which is more likely from overfitting than anything else).

FlowNet2-s has the lowest run-time of all networks considered. This method's number of parameters is not reported but should be considerably lower than for FlowNetS. However, even if the factor of 3/8th of the channels of FlowNetS would apply directly to the number of parameters, the resulting network would still have ten times the number of parameters of SPyNet and approximately 1.4 times the parameters of PWC-Net. None of the FlowNet-like networks would contribute to the identified Pareto front in the parameter-related graphs, even when giving FlowNet2-s a favorable estimation of 3/8th of the parameters. Clearly, FlowNet-like networks are straightforward and might even run well if small enough, but are not memory friendly. FlowNet2 is so large and slow that

it is out of the limits of all graphs. The drop in run-time for FlowNetS+ compared to FlowNetS in figure X is caused by excluding reading and writing times from the run-time benchmarks by Sun et al. Dosovitskiy et al. do not elaborate on how run-times are measured. A significant boost in flow estimate accuracy is achieved by retraining the networks (FlowNetS+, FlowNetC+, retrained by Ilg et al.), but the results are far from the state of the art.

The used GPU for inference, if reported, can be found in table 2.2. The only serious outlier is the NVIDIA K80 used for SPyNet, which results in a higher run-time than expected for the comparatively small and simple network. Comparing figure 2.9 and 2.10, SPyNet has a similar architecture as PWC-Net, but with fewer channels and no correlation layer. If anything, it should have a lower run-time. If so, it could be a serious competitor among the lightweight methods. It would be interesting to see how much flow estimation accuracy could increase by implementing an IRR-like framework and, being one of the first end-to-end networks proposed, better training. Additionally, it could be interesting to see a SPyNet with a feature rather than an image pyramid. This would effectively be a low channel PWC-Net, without correlation layer and context network.

The importance of proper training becomes apparent from the benchmark results. PWC-Net+ outperforms PWC-Net on all reported benchmarks, except on Sintel Clean. ScopeFlow (only test results are available, which, unlike vice versa, is forgivable, good training results might only indicate overfitting) outperforms the identical IRR-PWC architecture in all benchmarks.

LiteFlowNet architectures are not the best in their class when it comes to just run-time or number of parameters, but they are pretty solid when both are considered simultaneously. LiteFlowNetX has impressive accuracy for one of the networks with the fewest parameters, but it does not translate to a low inference time. This is likely due to the correlation layer, which has no learnable parameters but still consumes time to run. The only source that gives an indication of the time consumption of a correlation layer is the official GitHub repository for VCN[3]. The authors state that approximately 8.7% of the run-time is due to the correlation layer, compared to 4.7% due to the feature encoder. It would be interesting to see a LiteFlowNetX variant that skips the final flow inference layer, like LiteFlowNet2, which saw a significant run-time performance improvement over LiteFlowNet.

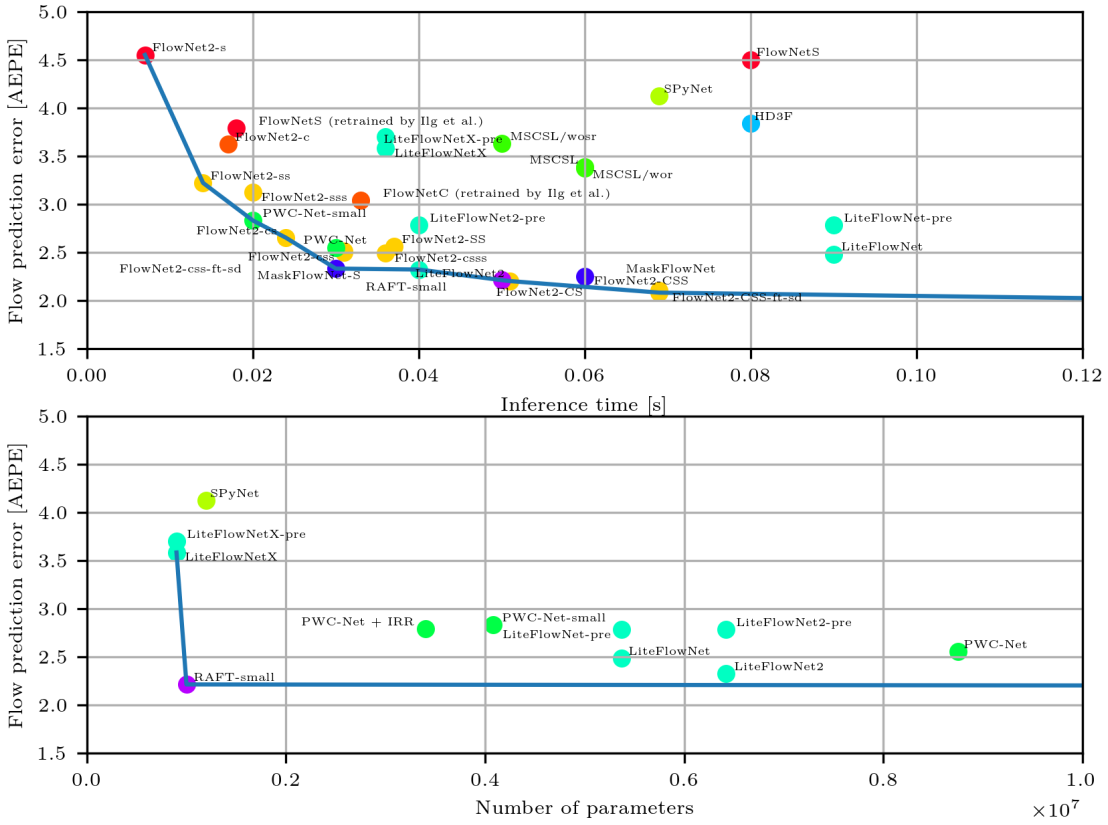
RAFT is currently the most promising network in terms of parameter efficiency. In the number of parameters vs. flow prediction error graphs, RAFT achieves lower flow prediction errors of other Pareto points at a minimal cost in the number of parameters, resulting in nearly vertical lines in the identified Pareto front. It is not unthinkable that the entire Pareto front in these graphs could be dominated by the method, given more data points (e.g., by varying hyperparameter settings or the number of channels). The parameter efficiency does not directly translate to low run-times, but there is still room to reduce RAFT-small's network size further. The poor run-time performance, while having high parameter efficiency, likely originates from the construction of the 4D correlation volume. It is a pity no test results are available for RAFT-small.

[3] `https://github.com/gengshan-y/VCN`

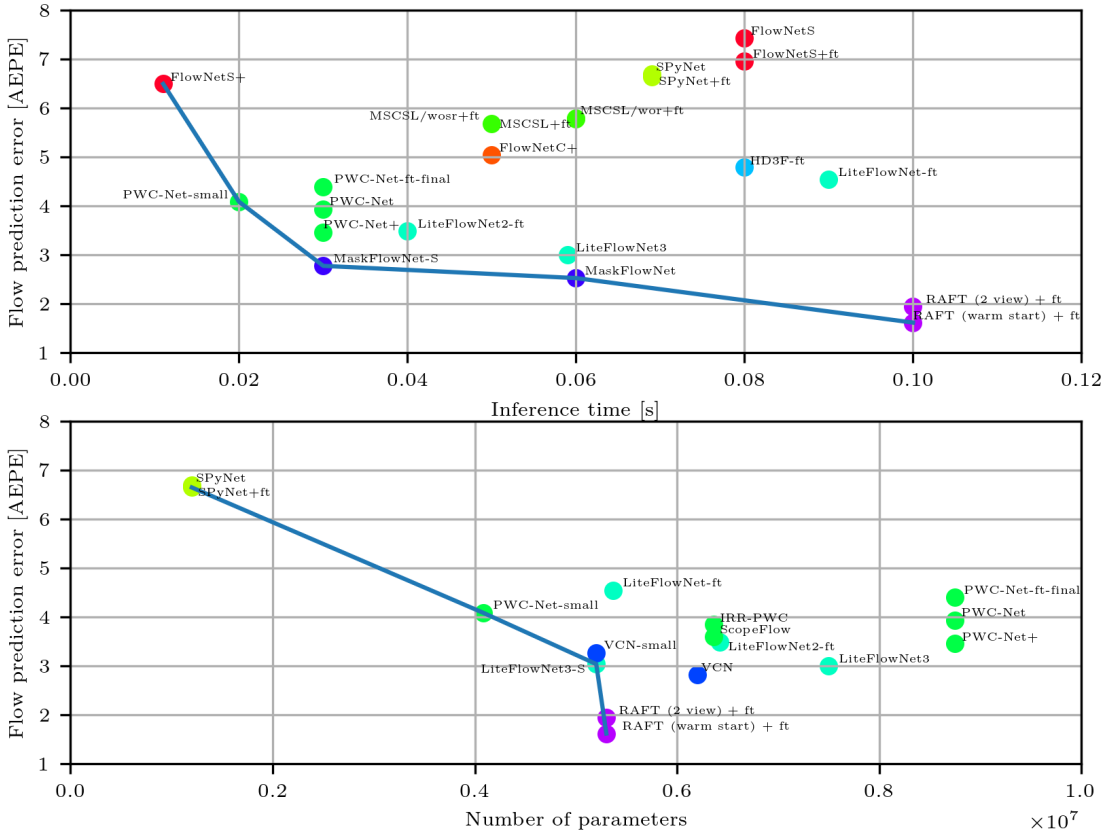
MaskFlowNet performs very well on Sintel Final and KITTI. Sadly, the authors do not disclose any information on the number of parameters. As such, the network is hard to compare with other methods.

Table 2.2: Self-reported benchmark results, number of parameters, run-time performance. Parentheses indicate the network was trained on the same benchmark dataset.

Network	*Sintel clean*		*Sintel final*		*KITTI 2012*		*KITTI 2015*			*Hardware*	*Inference time [s]*	*FPS*	*Number of parameters*
	EPE train	*EPE test*	*EPE train*	*EPE test*	*EPE train*	*EPE test*	*EPE train*	*F1-all train*	*F1-all test*				
FlowNetS	4.50	7.42	5.45	8.43	8.26	-	-	-	-	TITAN	0.080	13	3.21E+07
FlowNetS+ft	(3.66)	6.96	(4.44)	7.76	7.52	9.1	-	-	-	TITAN	0.080	13	3.21E+07
FlowNetC	4.31	7.28	5.87	8.81	9.35	-	-	-	-	TITAN	0.150	7	3.26E+07
FlowNetC+ft	(3.78)	6.85	(5.28)	8.51	8.79	-	-	-	-	TITAN	0.150	7	3.26E+07
FlowNetS (retrained [30])	3.79	-	-	-	-	-	-	-	-	1080	0.018	56	3.21E+07
FlowNetC (retrained [30])	3.04	-	-	-	-	-	-	-	-	1080	0.033	30	3.26E+07
FlowNetS+	(2.80)	6.49	(2.76)	6.54	-	-	-	-	-	TITAN X	0.011	88	-
FlowNetC+	(2.31)	5.04	(2.34)	5.47	-	-	-	-	-	TITAN X	0.050	20	-
FlowNet2-s	4.55	-	5.21	-	8.89	-	16.42	56.81	-	1080	0.007	143	-
FlowNet2-ss	3.22	-	3.85	-	5.45	-	12.84	41.03	-	1080	0.014	71	-
FlowNet2-sss	3.12	-	-	-	-	-	-	-	-	1080	0.020	50	-
FlowNet2-SS	2.56	-	-	-	-	-	-	-	-	1080	0.037	27	-
FlowNet2-c	3.62	-	-	-	-	-	-	-	-	1080	0.017	59	-
FlowNet2-cs	2.65	-	-	-	-	-	-	-	-	1080	0.024	42	-
FlowNet2-css	2.51	-	3.54	-	4.49	-	11.01	35.19	-	1080	0.031	32	-
FlowNet2-css-ft-sd	2.50	-	3.50	-	4.71	-	11.18	34.10	-	1080	0.031	32	-
FlowNet2-csss	2.49	-	-	-	-	-	-	-	-	1080	0.036	28	-
FlowNet2-CS	2.20	-	-	-	-	-	-	-	-	1080	0.051	20	-
FlowNet2-CSS	2.10	-	3.23	-	3.55	-	8.94	29.77	-	1080	0.069	14	-
FlowNet2-CSS-ft-sd	2.08	-	3.17	-	4.05	-	10.07	30.73	-	1080	0.069	14	-
FlowNet2	2.02	3.96	3.14	6.02	4.09	-	10.06	30.37	-	1080	0.123	8	1.62E+08
FlowNet2-ft-sintel	(1.45)	4.16	(2.01)	5.74	3.61	-	9.84	28.20	-	1080	0.123	8	1.62E+08
FlowNet2-ft-kitti	3.43	-	4.66	-	(1.28)	1.8	(2.30)	(8.61)	11.48	1080	0.123	8	1.62E+08
SPyNet	4.12	6.69	5.57	8.43	9.12	-	-	-	-	K80	0.069	14	1.20E+06
SPyNet+ft	(3.17)	6.64	(4.32)	8.36	4.13	4.7	-	-	-	K80	0.069	14	1.20E+06
MSCSL/wosr	3.63	-	4.93	-	5.98	-	-	-	-	TITAN X	0.050	20	-
MSCSL/wosr+ft	(3.18)	5.68	(4.21)	7.49	5.89	6.9	-	-	-	TITAN X	0.050	20	-
MSCSL/wor	3.37	-	4.72	-	5.80	-	-	-	-	TITAN X	0.060	17	-
MSCSL/wor+ft	(3.07)	5.79	(4.16)	7.42	5.87	6.8	-	-	-	TITAN X	0.060	17	-
MSCSL	3.39	-	4.70	-	5.87	-	-	-	-	TITAN X	0.060	17	-
MSCSL+ft	(3.07)	5.78	(4.15)	7.12	5.77	7.1	-	-	-	TITAN X	0.060	17	-
PWC-Net-small	2.83	4.08	-	-	-	-	-	-	-	TITAN X	0.020	50	4.08E+06
PWC-Net-small-ft	(2.27)	5.05	(2.45)	5.32	-	-	-	-	-	TITAN X	0.020	50	4.08E+06
PWC-Net	2.55	3.93	-	-	4.14	-	10.35	33.67	-	TITAN X	0.030	33	8.75E+06
PWC-Net-ft	(1.70)	3.86	(2.21)	5.13	(1.08)	1.7	(1.45)	(7.59)	7.90	TITAN X	0.030	33	8.75E+06
PWC-Net-ft-final	(2.02)	4.39	(2.08)	5.04	-	-	-	-	-	TITAN X	0.030	33	8.75E+06
LiteFlowNetX-pre	3.70	-	4.82	-	6.81	-	16.64	36.64	-	1080	0.036	28	9.00E+05
LiteFlowNetX	3.58	-	4.79	-	6.38	-	15.81	34.90	-	1080	0.036	28	9.00E+05
LiteFlowNet-pre	2.78	-	4.17	-	4.56	-	11.58	32.59	-	1080	0.090	11	5.37E+06
LiteFlowNet	2.48	-	4.04	-	4.00	-	10.39	28.50	-	1080	0.090	11	5.37E+06
LiteFlowNet-ft	(1.35)	4.54	(1.78)	5.38	(1.05)	1.6	(1.62)	(5.58)	9.38	1080	0.090	11	5.37E+06
HD3F	3.84	-	8.77	-	4.65	-	13.17	23.99	-	-	0.080	13	-
HD3F-ft	(1.70)	4.79	(1.17)	4.67	(1.05)	1.4	(1.31)	(4.10)	6.55	-	0.080	13	-
IRR-PWC	(1.92)	3.84	(2.51)	4.58	-	-	(1.63)	(5.32)	7.17	1080Ti	0.200	5	6.36E+06
IRR-FlowNet	3.32		4.92	-	-	-	-	-	-	-	-	-	4.51E+07
PWC-Net + IRR	2.79		4.10	-	-	-	-	-	-	-	-	-	3.40E+06
FlowNetS + IRR	3.77	-	5.00	-	-	-	-	-	-	-	-	-	3.21E+07
LiteFlowNet2-pre	2.78	-	4.14	-	4.11	-	11.31	32.12	-	1080	0.040	25	6.42E+06
LiteFlowNet2	2.32	-	3.85	-	3.77	-	9.83	28.45	-	1080	0.040	25	6.42E+06
LiteFlowNet2-ft	(1.41)	3.48	(1.83)	4.69	(0.95)	1.4	(1.33)	(4.32)	7.62	1080	0.040	25	6.42E+06
LiteFlowNet3	(1.32)	2.99	(1.76)	4.45	(0.91)	1.3	(1.26)	(3.82)	7.34	1080	0.059	17	7.50E+06
LiteFlowNet3-S	(1.43)	3.03	(1.90)	4.53	(0.94)	1.3	(1.39)	(4.35)	7.22	-	-	-	5.20E+06
PWC-Net+	(1.71)	3.45	(2.34)	4.60	-	1.4	-	-	7.72	TITAN X	0.030	33	8.75E+06
VCN-small	(1.84)	3.26	(2.44)	4.73	-	-	(1.41)	(5.50)	7.74	-	-	-	5.20E+06
VCN	(1.66)	2.81	(2.24)	4.40	-	-	(1.16)	(4.10)	6.30	1080Ti	0.260	4	6.20E+06
MaskFlowNet-S	2.33	2.77	3.72	4.38	3.21	1.1	-	23.58	6.81	TITAN Xp	0.030	33	-
MaskFlowNet	2.25	2.52	3.61	4.17	2.94	1.1	-	23.14	6.11	TITAN Xp	0.060	17	-
ScopeFlow	-	3.59	-	4.10	-	1.3	-	-	6.82	TITAN X	-	-	6.36E+06
RAFT (2 view) + ft	(0.76)	1.94	(1.22)	3.18	-	-	(0.63)	(1.50)	5.10	1080Ti	0.100	10	5.30E+06
RAFT (warm start) + ft	(0.77)	1.61	(1.27)	2.86	-	-	-	-	-	1080Ti	0.100	10	5.30E+06
RAFT-small	2.21	-	3.35	-	-	-	7.51	26.90	-	1080Ti	0.050	20	1.00E+06

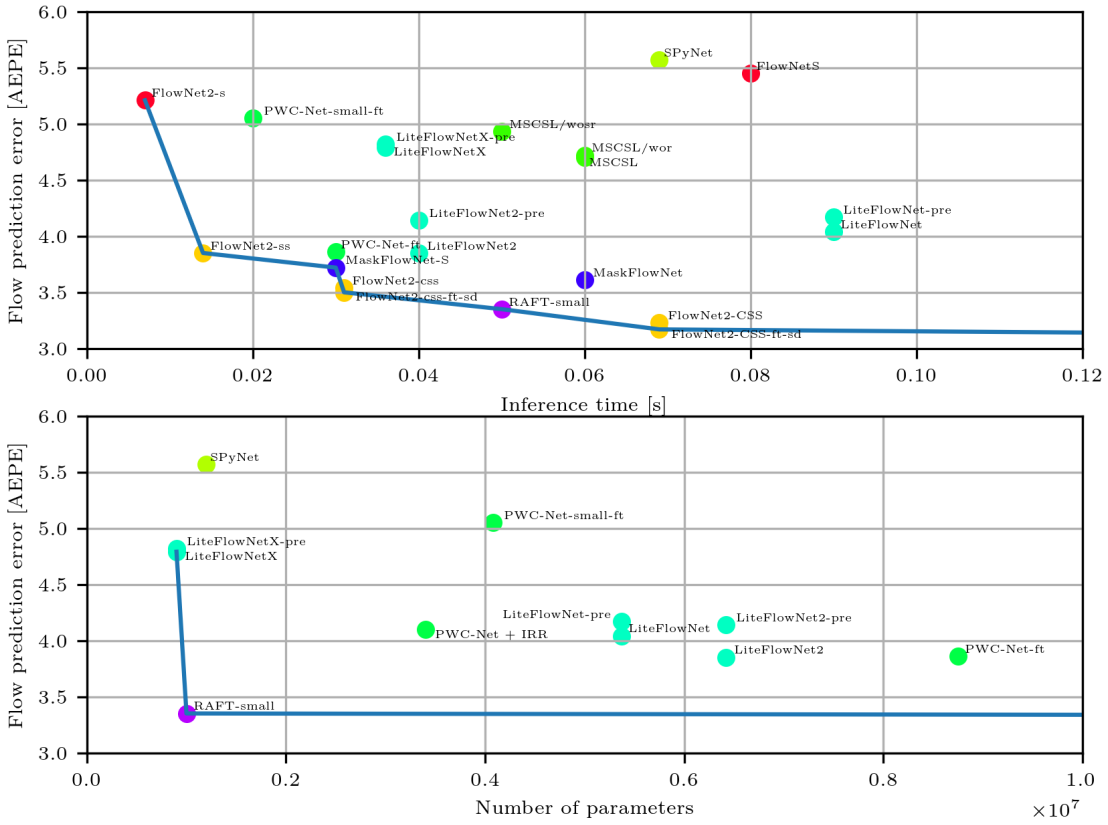


(a) Train

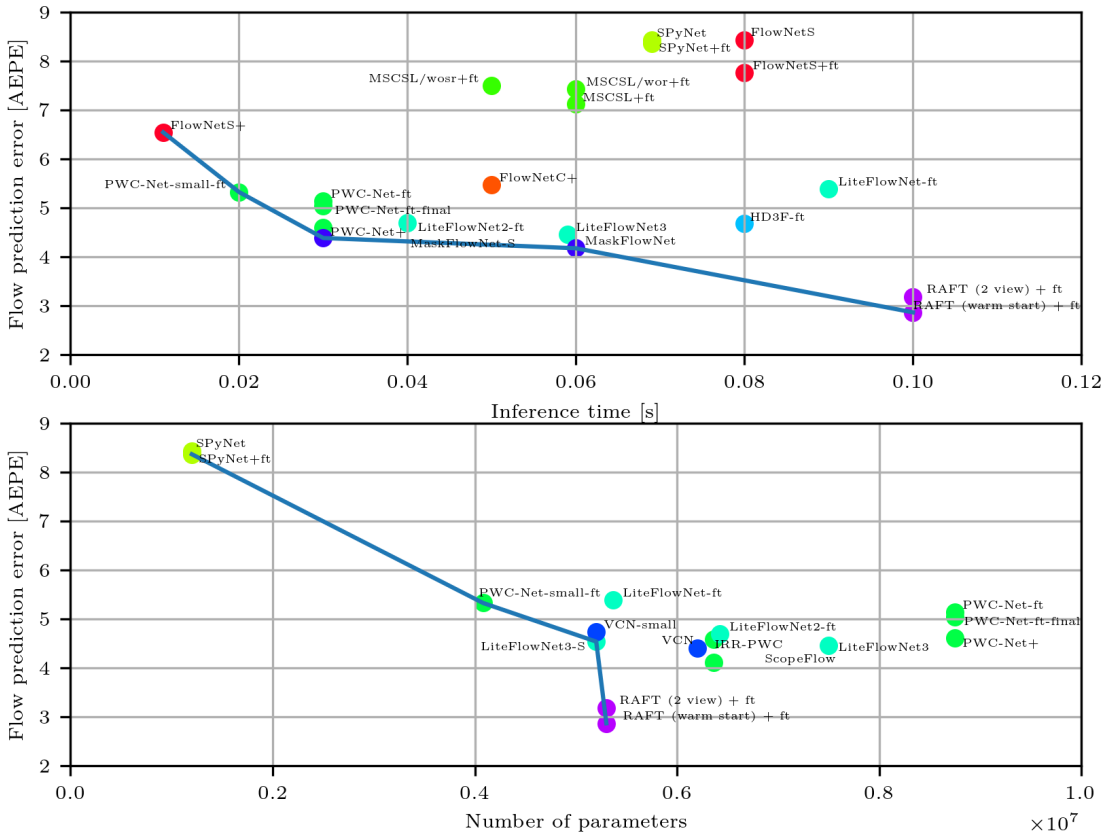


(b) Test

Figure 2.12: MPI Sintel Clean self-reported benchmark results

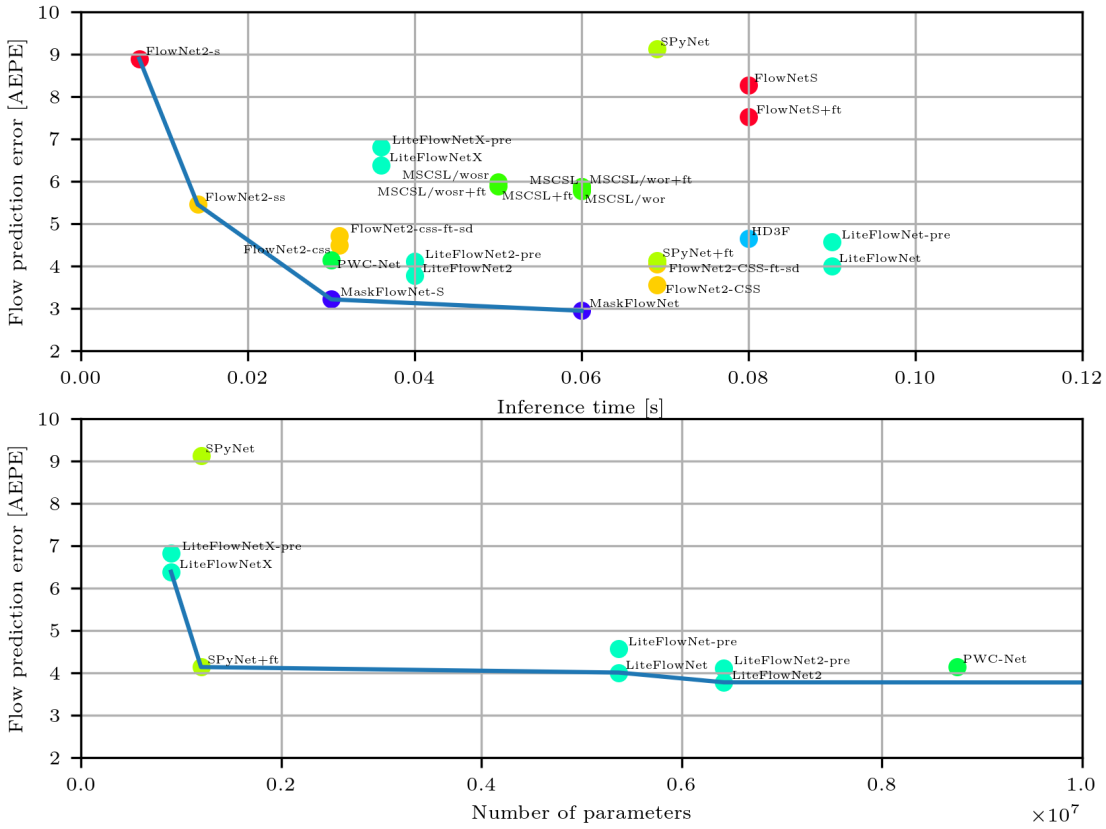


(a) Train

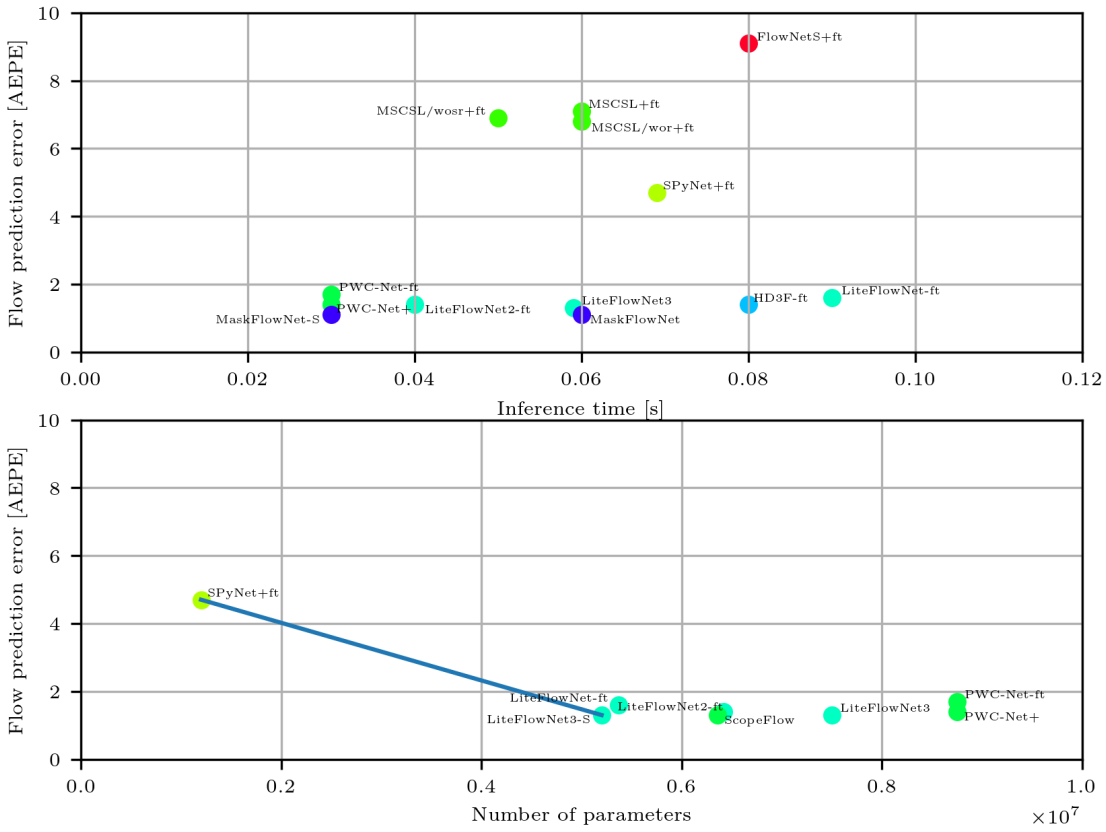


(b) Test

Figure 2.13: MPI Sintel Final self-reported benchmark results

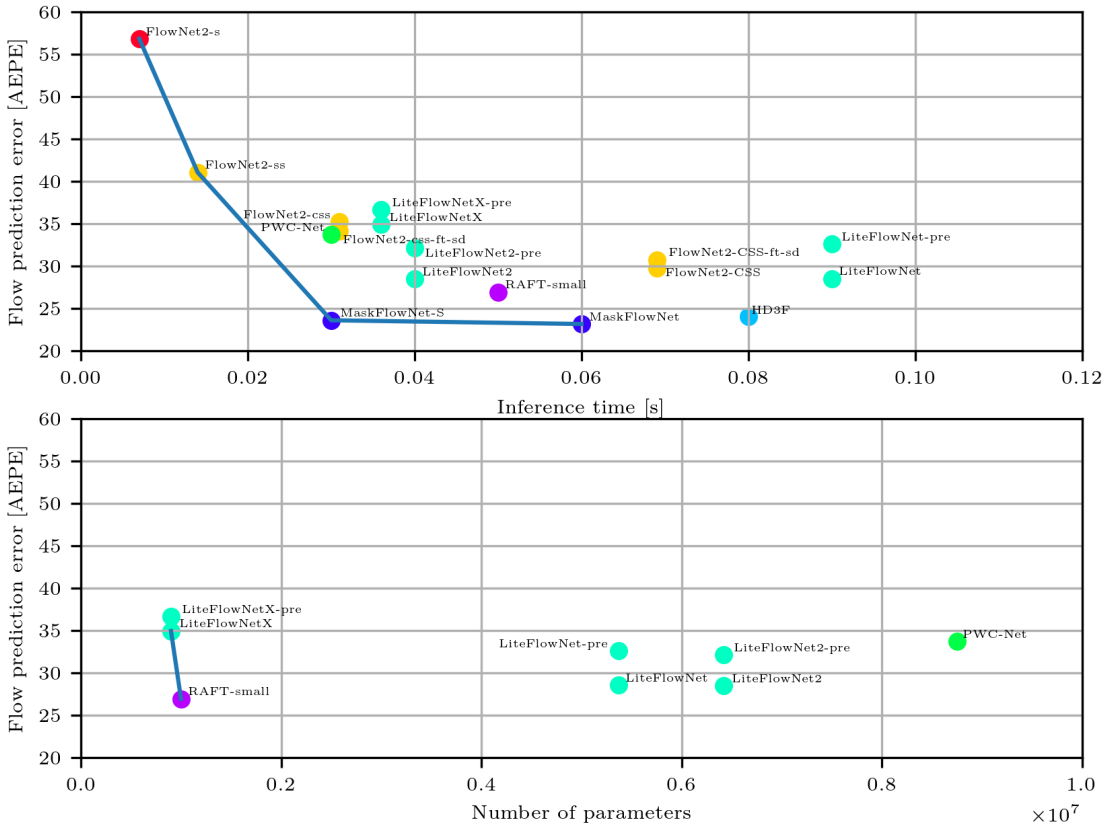


(a) Train

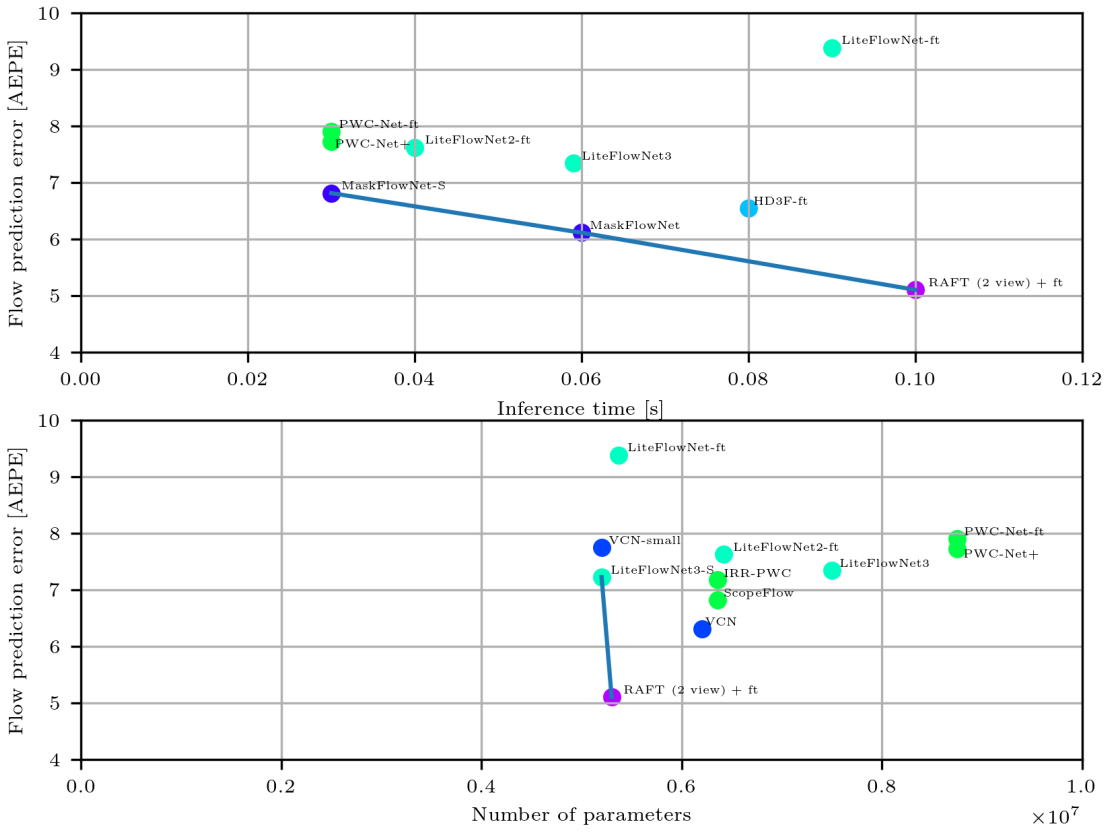


(b) Test

Figure 2.14: KITTI 2012 self-reported benchmark results



(a) F1 train



(b) F1 test

Figure 2.15: KITTI 2015 self-reported benchmark results

2.6 Self-supervised End-to-end CNN Optical Flow Estimation

One of the biggest challenges of supervised optical flow CNNs is the disrepancy between the synthetic training data and the target domain, resulting in much poorer accuracy in target domain applications. Several self-supervised networks are proposed to train the CNNs directly on representative domain data while lacking ground truth optical flow. These methods are slightly behind in accuracy on established benchmarks but promise better results on target domains for which there is no available ground truth data. The value of such frameworks highly depends on the desired application of the flow network. Often, these methods are an extension of existing CNN architectures.

Ahmadi et al. [44] propose a CNN architecture (USCNN) that is self-supervised by a cost function inspired by the classical Horn et al. [4] method. By minimizing the cost function, the network learns to estimate optical flow. It closely resembles a teacher-student learning paradigm, where the network learns from a classical method. The CNN architecture is very similar to SPyNet, where one of two input images is warped with an upsampled flow estimate from a lower resolution iteration. The flow estimate network used is much larger than SPyNet, and has an encoder-decoder architecture quite similar in shape and size to FlowNet. The only self-reported benchmarks are on MPI Sintel and UCF101, for which no ground truth optical flow is available, hence results are compared with the state of the art EpicFlow. The very brief report does not cover the number of parameters or run-time performance.

Yu et al. [45] apply a self-supervised loss function to FlowNetS. The loss is a combination of a traditional photometric loss, comparing the target image, warped to source using the flow prediction, with the source image, and a smoothness loss function. The approach allows for training on raw KITTI data (80k image pairs, larger than any dataset treated in section 2.4), for which ground truth optical flow is unavailable. The only self-reported benchmarks are on FlyingChairs and KITTI 2012. Compared to the regular FlowNetS, the self-supervised method by Yu et al. performs significantly worse on FlyingChairs and slightly worse on KITTI 2012, except for non-occluded pixels, where it performs better. Concurrently, Ren et al. [46] apply a very similar loss function to an architecture heavily inspired by FlowNetS.

Zhu et al. [47] propose a novel architecture inspired by DenseNet [48]. The architecture has a typical encoder-decoder shape, but features several *dense blocks* between convolutions. These blocks are built up out of several convolutions with skip connections between all layers. These blocks help combat overfitting, and require fewer parameters. A simple photometric loss function is used, see equation 2.4, where I_1' is the target image I_2, inversely warped with the flow estimate, and N the number of pixels, and $\rho(x)$ the generalized Charbonnier penalty ($\rho(x) = (x^2 + \epsilon^2)^\alpha$).

$$L_{\text{reconst}} = \frac{1}{N} \sum_{i,j}^{N} \rho \left(I_1(i,j) - I_1'(i,j) \right) \tag{2.4}$$

In another work, Zhu et al. [32] propose a framework that uses the classical FlowFields method, which at that point was still state of the art, to generate proxy 'ground truth'

data. Further refinement done is with a photometric loss. Urban et al. [49] show that distillation methods, similar to the one presented by Zhu et al., can result in student models that are shallower than the teacher model they mimic. Still, student models need sufficient representational capacity to learn functions of comparable accuracy. Bucila et al. [50] show that a small network with 1000x fewer parameters, can be trained to closely approach the accuracy of an ensemble of architectures.

Given the fact that the proxy loss function dictates what the network will learn, Meister et al. [51] propose a novel occlusion-aware loss, through a bidirectional flow estimation (where input images are exchanged) and the use of a higher order smoothness loss. Occlusions can be determined from bidirectional flow estimation, since occlusions from source to target image, are the inverse of disocclusions from target to source image. By warping the source image with the estimated flow, disocclusions can be detected. The photometric loss is applied only in non-occluded areas. Inspired by Ilg et al., the authors use FlowNetC, FlowNet2-CS, and FlowNet2-CSS as base architectures for *UnFlow-C*, *UnFlow-CS*, and *UnFlow-CSS*, respectively. Concurrently, Wang et al. [52] propose a very similar framework. The difference lies in that Wang et al. use a modified FlowNetS as base architecture. The modification is that flow estimates in the decoder are used to warp downsampled input images, which are then upsampled to the next flow prediction resolution and concatenated to the information flow in the decoder.

Chapter 3

CNNs on the Edge

Several deep learning fields closely related to optical flow have investigated the application of networks on embedded hardware. To the best of our knowledge, no such works has been conducted for optical flow. In this section findings from other CNN tasks that reduce latency and are applicable to optical flow networks are discussed.

Wofk et al. [53] propose *FastDepth*. This work combines several findings from other works to design a monocular depth estimation with very low latency. The network architecture used is similar to those used in optical flow estimation, in particular to FlowNetS. *MobileNet* [54] is used as encoder, which uses depthwise separable convolutions. These convolutions can be seen as two separate convolutions. The first convolution has kernels with a length of 1 in the channel dimension, effectively iterating over one channel of the image at a time. This convolution is followed by a 1x1 - or pointwise convolution that convolves all input channels. This reduced the number of multiply-accumulate operations (MAC) by a factor $h \times w$, where h and w are the height and width (spatial resolution) of input feature map, respectively. The use of depthwise separable convolutions is expanded to the decoder. The use of hardware-specific TVM [55] compiler further reduces latency. *NetAdapt* [56] is used to prune the network. It empirically and iteratively tweaks the number of channels in all layers, by picking the network with the best trade-off between accuracy and a user-defined metric (such as latency on a nano quadcopter), out of a series of generated networks based on a reference model.

Dilated convolutions, first applied in the field of semantic segmentation by Yu et al. [34], introduce a dilation between kernel values, effectively creating a checkerboard-like filter. Using dilated convolutions increases the receptive field size without the need of a larger kernel and without loss of spatial resolution. They can be used as a replacement for upsampling/transposed convolutional layers. They have seen limited application in optical flow. Zhu et al. [57] replace `c3_1`, `c4_1`, `c5_1` of FlowNetS (figure 2.7) with dilated convolutions. The resulting increased receptive field allows the authors to remove the entire encoder, except for the final flow predicting convolution `pf2`. The authors promise improved run-time performance. Zhai et al. [58] apply dilated convolutions to limit computational expense and improve flow estimation for large motion.

Chapter 4

Flight & Computing Platform

Advancements in reduced instruction set computer (RISC) microprocessors and kernels have been increasingly enabling embedded devices to employ neural networks. The ultra-low power GAP8 [10] by GreenWaves technologies[1] has been designed for CNN inference. Improved neural network kernels [11, 12] allow for even faster inference. Palossi et al. [13] proposed *PULP-Shield*, an expansion deck for a CrazyFlie 2.0. The PULP-Shield is based on a GAP8 SoC, and succeeded in running an autonomous navigation CNN onboard. In collaboration with ETH Zurich, BitCraze[2] released a commercial off-the-shelf expansion deck, the AI-deck, for their CrazyFlie 2.X nano quadcopter series, based on the design of the PULP-shield, allowing faster network inference on a nano quadcopter. An overview of GAP8 speedups compared to the pure RISC-V ISA is available online[3]. With the hardware being novel, the only CNN inference time benchmark available is of SqueezeNet [59], taking approximately 400 ms for a single inference [60] of the 0.36 GFLOP[4] network.

4.1 Technical Specifications

The BitCraze AI-deck features a nona-core 32-bit GAP8 SoC, and an ESP32 NINA-W102 WiFi module. The GAP8 SoC features 1+8 RISC-V ISA cores. One core to rule them all, and 8 to perform tasks in parallel. It features 64kB level 1 memory and 512kB level 2 memory. The CrazyFlie 2.X nano quadcopter features 512 Mbit HyperFlash and 64 Mbit HyperRAM, up to 16Mb of which the GAP8 can access. The ESP32 NINA WiFi will be used to stream results and/or image data back to a personal computer. The AI-deck is equipped with a Himax HM01B0 ultra low power 320×320 Bayer RGB camera that can be used to feed images into an optical flow network.

The AI-deck can be mounted on the CrazyFlie 2.X nano quadcopter series. It adds 4.4g to the 27g flight platform. The dimensions of the combination of the quadcopter and the expansion deck do not exceed the dimensions of the quadcopter itself. The CrazyFlie 2.X measures 92x92x29mm (excluding rotors).

[1] `https://greenwaves-technologies.com/`
[2] `https://www.bitcraze.io/`
[3] `https://greenwaves-technologies.com/gap8-cnn-benchmarks/`
[4] `https://github.com/albanie/convnet-burden`

4.2 Comparison With Other Flight Platforms

The CrazyFlie 2.X is classified as a nano quadcopter. Drones of this class of micro aerial vehicles (MAVs) are very small in size (<100mm in width and height) and are very lightweight. Due to strict size, weight and power consumption constraints these nano quadcopters are conventionally equipped with chips of comparatively limited computational power. The AI-deck brings significantly faster CNN inference to the small hardware.
Larger flight platforms may carry more computing power, or have their computational power extended similar as done by the AI-deck. The NVIDIA Jetson series of GPUs can bring significant computational power to larger flight platforms, but require a much more powerful (and thus larger) flight platform. Sanket et al. [61] use optical flow to estimate depth and detect a gap in a wall. Including the gap detection algorithm, the used NVIDIA Jetson TX2 achieves a run-time of 0.12 s for SpyNet and a run-time of 1.00 s for FlowNet2. FlowNet2 having the largest self-reported inference time of all considered networks. These results indicate that such hardware might benefit from latency improvements, but is already capable of running existing optical flow CNNs. The TX2 weighs 85 grams, 20 times the weight of the AI-deck. Considering the implications for the required flight platform (mounted on a 380x328x89mm, 500g Parrot Bebop 2) and power consumption, this justifies the categorization into a separate class. For an indication of the inference times on the AI-deck, refer to chapter 7.

Chapter 5

Research Question & Objective

This chapter consists out of two sections. First, in section 5.1, the research questions are formulated. In section 5.2, the research objective is proposed.

5.1 Research Question(s)

How can an end-to-end convolutional neural network for dense optical flow estimation for on-board, real-time deployment on a nano quadcopter be designed?

1. What is the trade-off between memory usage, run-time performance and optical flow prediction accuracy?
    - What is the trade-off between flow prediction accuracy at varying flow velocities?
    - What is the trade-off between flow prediction accuracy for varying distances from motion boundaries?
    - How can the representational capacity of a tiny convolutional neural network be maximized while maintaining the run-time and memory budget of a nano quadcopter?
2. How does the approach perform compared to traditional methods of optical flow estimation on nano quadcopters in terms of optical flow prediction accuracy, run-time performance and memory usage?

5.2 Research Objective

The main research objective of this thesis is:

> "To achieve real-time, on-board, dense optical flow estimation on a nano quadcopter by means of an end-to-end convolutional neural network".

Chapter 6

Research Plan

In this chapter, a research plan is presented to answer the research questions. First, in section 6.1, the methodology is introduced. Next, in section 6.2 the accompanying experimental set-up is introduced. This mainly covers the software and hardware used to conduct the experiments from the previous section. In section 6.3, the expected results and how to interpret them, the verification and validation of the experiments, and the relevance of the work are discussed.

6.1 Methodology

The process of designing the architecture can be split up into the five following major steps, which are further explained in their respective sections below.

1. Define performance metrics
2. Set-up pipeline
3. Define baseline model
4. Identify bottlenecks
5. Update, iterate

6.1.1 Define performance metrics

The first step is to define what performance metrics will be measured, and what results for these metrics are desired. To get proper insight into the strengths and weaknesses of each base network, more than just the EPE at the final flow prediction should be measured. Metrics such as EPE for varying flow velocities, EPE for varying distances from image boundaries, and EPE for all levels of flow prediction (if applicable) should be benchmarked from an early stage.
Considering the applications and high mobility of nano quadcopters, a slight preference goes to low EPE in high velocity flow. Predictions far away from motion boundaries should be sufficiently accurate in an obstacle avoidance application. All these error metrics shall be minimized, while sticking to the memory and run-time budget that come with the AI-deck and possible applications.

6.1.2 Set-up pipeline

Next, before any architecture is implemented, a pipeline will be set up to both train and evaluate the model using the chosen performance metrics. This includes a data pipeline, that loads training images and ground truth flows, and a training pipeline in the deep learning framework of choice (see chapter 6.2). Considering the high number of epochs optical flow networks are trained with, it is worth investing in an efficient data and training pipeline. The training pipeline should support early stopping [62] and store the latest state of the network, so that training can be resumed after a stop.
Run-time performance is another important metric in this project. Whereas inference performance achieved on other hardware (such as on a desktop computer) is relevant, and metrics such as FLOPs do give insight into the run-time performance of networks and hardware alike, for a proper indication of the power of the AI-deck some physical tests must be done. After all, kernels have a large influence on the achieved framerate [39]. Hence, architectures should be benchmarked on the AI-deck itself. For simplicity, these can be randomly initialized, as this does not affect run-time significantly (unless training is accompanied by heavy pruning). Run-time performance could be measured on a desktop computer to compare with previous iterations, but ultimately should be tested on the AI-deck for any candidate architecture.

6.1.3 Define baseline model

A baseline model will be designed, based on literature and considering the memory and run-time performance limitations of the AI-deck. The earlier executed benchmarks will give an indication of the possibilities. On the very limited hardware there is no room to compensate for an unsuitable architecture or improper training/hyperparameter selection with a larger network architecture.

6.1.4 Identify bottlenecks

After training the baseline model, the earlier discussed metrics will be used to measure network performance, and identify shortcomings.

6.1.5 Update, iterate

The identified bottlenecks will be used to improve the design. The goal is to minimize the error metrics, while keeping to a memory and run-time budget. According to GoodFellow et al. [62], the performance of a network is dictated by "the representational capacity of the model, the ability of the learning algorithm to successfully minimize the cost function used to train the model, and the degree to which the cost function and training procedure regularize the model".
Using a higher number of nodes in a network (in CNNs this could mean increasing spatial resolution and/or number of channels per feature map) is a direct way of increasing the network's representational capacity, but a direct way of increasing the run-time and memory usage at the same time. Ilg et al. [30] showed that reducing the number of network channels is a direct way of speeding up a network, at a loss of flow prediction accuracy. Reducing the number of channels could even be applied at the input, e.g. by reducing the number of input image channels to 2 or 1 (greyscale), at a considerable loss of information. Other parameters that can be varied are spatial resolution (which depends on the

used stride, use of pooling, input/output resolution), and more fundamental architectural decisions such as the number of convolutional layers/depth of the network, or more optical flow specific architecture choices such as the number of multi-level flow predictions or use of correlation/warping operations.
The reduction of output resolution is already used in networks from literature, such as FlowNet, where bilinear upsampling is used to upscale the highest resolution flow prediction 4 times to bring it back to input resolution. The bilinear upsampling step does not add additional flow information so might just as well be left out for practical applications.

One of the most important aspects is to properly train the networks. As shown by PWC-Net+ [35] and ScopeFlow [21], different training methods can significantly improves results. An IRR-like iterator can be added if there is still room for a slight increase in run-time, but the memory/parameter budget is filled. Further pruning can be applied reduce the number of parameters Quantization can improve run-time performance and memory usage significantly, at a small cost in accuracy.

There are several methods of automatically tweaking hyperparameters, such as Network Architecture Search. Automatic hyperparameter tweaking is not realistic for optical flow networks due to the very high required number of training epochs.

6.2 Experimental Set-up

6.2.1 Network Architecture Design and Training

As the provided GAP SDK (see section 6.2.2) can use TensorFlow Lite models as input, the most straightforward method of network design and training is using TensorFlow. The trained TensorFlow networks can be converted to TensorFlow Lite models directly using the TensorFlow Lite Converter.
Several GitHub repositories exist for TensorFlow implementations of optical flow networks, from which the data and training pipelines could be adapted to a novel network architecture. These repositories have been mostly abandoned and are left outdated.

With the release of GAP SDK `v3.8.0` support for Open Neural Network Exchange (ONNX) filetypes has been released. This enables the use of other deep learning frameworks than TensorFlow. Without direct support for ONNX, models would have to be converted to ONNX, to TensorFlow, to TensorFlow Lite. This number of conversions is prone to causing errors and incompatibility.
There are several PyTorch implementations that are kept up to date, and can be used to quickly implement an efficient pipeline.

6.2.2 GAP SDK

GreenWaves technologies has provided an SDK online[1] for the development and execution of software on the GAP8 processor. It allows for conversion of TensorFlow Lite and ONNX models for execution on the GAP8 processor.

[1] `https://github.com/GreenWaves-Technologies/gap_sdk`

The conversion of a graph to GAP8 executable code is done in several steps, that can be automated. This is intentional, to avoid a black box approach where debugging is more difficult. Figure 6.1 shows the workflow (or GAPflow) split up in these steps.
NNTool can load a TensorFlow Lite graph, with or without quantization. NNTool has the ability to do the quantization itself. Key to both quantization during TensorFlow Lite conversion and in NNTool is to feed the quantizer representative data. This could be actual image data, or randomly generated tensors. NNTool can convert the TensorFlow Lite graph into an Autotiler Model, written in C, that the AutoTiler can interpret. This C code contains the architecture and network parameters, and the paths to the separately stored constant tensors (e.g. network weights). Certain combinations of layers can be fused together to use optimized kernels for these combinations in AutoTiler.
The AutoTiler loads the AutoTiler Model and constant tensors, and produces NN optimized code. This tool automatically assigns memory across L1, L2 and L3. The generated code contains a network construct, destruct and inference function.
Together with user made application code (in C), which calls the AutoTiler generated functions, the NN optimized code can be compiled and flashed to the GAP8 processor or run in the GVSOC simulation.

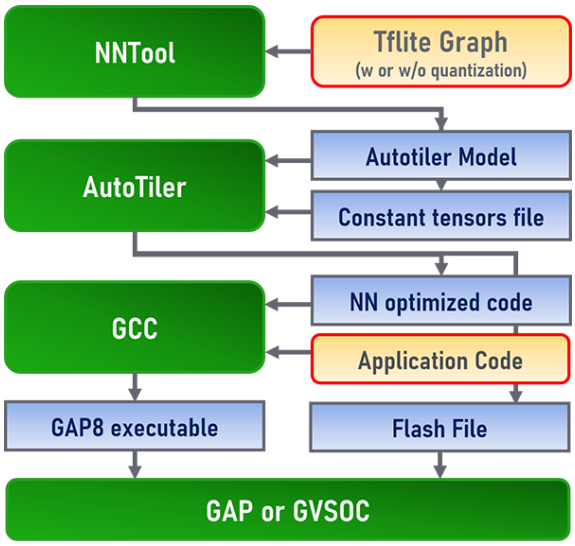


Figure 6.1: GAPflow [63]

6.2.3 GAP Unsupported Operations

At the time of writing there are several unsupported operands that are very common in optical flow CNNs.

Transposed convolutions
Also known as deconvolutions, upconvolutions. These are common in decoder networks. Effectively they provide learnable upsampling, and could be seen as a regular convolution with a fractional stride. A workaround is to use bilinear upsampling, followed by a regular convolution. This workaround, however, requires four times the operations of a transposed convolution [64].

Concatenations with multiple versions of the same input
These are widely used in optical flow literature for feature concatenation. These concatenated skip connections carry information from the encoder to the decoder part of the network in order to preserve spatial information that might be lost in the encoder, which reduces the spatial resolution with convolutions. Another very common use of concatenations is flow prediction concatenation, where lower resolution flow predictions are upsampled and concatenated to a higher spatial resolution feature map in the decoder. Information could be carried through the network with additions instead of concatenations as done in residual networks [65], but this requires tensors of equal dimensions. The skip connections lend themselves well to such a workaround, but flow predictions are 2-channel tensors, and can as such not be added to numerous-channel feature maps. Without this operation, conventional architectures are a no-go. On the bright side, conventional networks with numerous flow predictions at varying resolution would never be achievable due to hardware constraints.

Image warping, correlation layer
A large number of networks use image warping (SPyNet, PWC-Net, ScopeFlow, MaskFlowNet, etc) and/or a correlation layer (PWC-Net, ScopeFlow, MaskFlowNet, RAFT, etc). These operations are quite specific to optical flow, so there is not a large chance they will be implemented to the supported operands in the future. There are other ways of implementing such layers. A manual image warping operation or correlation layer can be written in the application code, or kernels can be written following the instructions provided by GreenWaves. However, the performance implications (and limited project time) should be considered.

Resize
Upscaling of flow estimates occurs in all treated networks. Currently, (bilinear) resizing layers are only supported for TensorFlow Lite graph imports.

All unsupported operations have been requested for implementation with GreenWaves.

6.2.4 Hardware

The BitCraze AI-deck features a nona-core 32-bit GAP8 SoC, and an ESP32 NINA WiFi module. It can be mounted on the CrazyFlie 2.X nano quadcopter series. An overview of GAP8 speedups compared to the pure RISC-V ISA is available online[2]. With the hardware being novel, the only CNN inference time benchmark available is of SqueezeNet [59], taking approximately $400ms$ for a single inference [60] of the $0.36GFLOP$[3] network. The GAP8 SoC features 1+8 RISC-V ISA cores. One core to rule them all, and 8 to perform tasks in parallel. It features 64kB level 1 memory and 512kB level 2 memory. The CrazyFlie 2.X nano quadcopter features 512 Mbit HyperFlash and 64 Mbit HyperRAM, up to 16Mb of which the GAP8 can access. The ESP32 NINA WiFi will be used to stream results and/or image data back to a personal computer. To flash to the AI-deck, the Olimex ARM-USB-TINY-H with 20 to 10 pin adapter will be used. The AI-deck can be powered with a Micro-USB connector.

[2] `https://greenwaves-technologies.com/gap8-cnn-benchmarks/`
[3] `https://github.com/albanie/convnet-burden`

Accuracy and run-time performance can be evaluated not only on-board, but also on a desktop computer. That is; accuracy or run-time performance gain on desktop will translate to a similar improvement of results on-board. This will be used to quickly test new features without having to run through the entire GAP SDK. Promising changes can be more quickly implemented on-board by automating the GAP SDK workflow using makefiles.

Evaluations can be done using images loaded into memory, or using the camera feed. Run-time benchmarking must be done carefully, making sure to exclude influence of the data pipeline, and using the correct level of optimization and disabling debugging features for GAP8 application, for proper evaluation.

6.3 Results, outcome, relevance

6.3.1 Future Result Interpretation

During benchmarking accuracy and run-time performance will be measured. This will result in *EPE*, F_1, F_2, inference time $[s]$ measurements. These can be used to select network features to trade-off accuracy and run-time performance. More specific accuracy metrics such as accuracy for varying flow velocity and distance from motion boundaries can be measured. By measuring all these metrics at different cnn levels (as they are in training), these metrics can provide insights into receptive field size, and occlusion behavior, respectively. The results will be compared to SOTA optical flow CNNs, but also with traditional methods.

6.3.2 Verification & Validation

Verification of the models will be done in a number of ways. Programming will be checked with code inspection, which is only realistically possible with well organized, structured code. Visualization will be used throughout the network development process as a verification tool. Tools such as Netron[4] can visualize both the TensorFlow and the TensorFlow Lite models. Both TensorFlow and NNTool can list network operations, the number of parameters associated, input/output dimensions. Visualizing the output at varying locations of the network gives a qualitative performance indication. A full focus on quantitative results might result in overlooking otherwise obvious errors. Visualizations are a quick way to identify mistakes in the network architecture, and can be used throughout the workflow. Visualizing the worst outcomes can give insight in the cause of the poor performance. The train and test error should be tracked, to identify whether the network is overfitting to the train data. If the network seems to underfit, a much smaller dataset can be used to train to identify whether or not there is a software error or the network lacks representational capacity.

The used synthetic training datasets will be split into a training and validation part. Considering the high number of training epochs ($\mathcal{O}(1E6)$) in literature cross-validation is not a realistic option considering the time to train. Finally, the flight tests will validate whether the network can predict flow in real life.

[4] `https://netron.app/`

6.3.3 Relevance

This work will provide insights into tiny optical flow networks. It will investigate how far optical flow networks can be shrunk, and still provide accurate flow estimations. It will tell if novel miniature hardware is powerful enough to estimate optical flow with convolutional neural networks. If achieved, it will propose the first optical flow net to run on a nano quadcopter.
The results might not only apply to tiny optical flow nets but tiny CNNs for computer vision tasks in general. Some computer vision tasks in particular are very closely related to optical flow, such as pose estimation, depth estimation, possibly allowing such networks to run on miniature hardware.

Chapter 7

FlowNetS encoder on the GreenWaves GAP8 SoC

Having set up most of the pipeline[1], some capacity testing of the GAP8 SoC can be done. One of the most straightforward networks treated in this document is FlowNetS, consisting of simple convolutions and transposed convolutions. Seeing that both transposed convolutions and resize layers are currently unsupported (GreenWaves confirmed that bilinear upsampling support would be added soon), only the encoders of these networks can be implemented, for now, directly followed by a single flow estimating convolution (reducing the number of channels to two).
The architecture of FlowNetS can be found in figure 2.7. The architecture of FlowNet2-s is identical, but the number of channels per feature map is multiplied by 3/8.
The run-time is measured and averaged over 10 runs. The assigned available memory is 4.88E+04, 2.50E+05, and 8.39E+06 bytes for L1, L2, and L3. Note that the overhead of kernels and application code fill a significant part of the memory. The inference input is fixed at a 160x160 pixels image, half of the 320x320 pixels of the AI-deck camera.
The GAP8 has a difficult time running large kernel sizes. The first convolution of FlowNetS with its 7x7 kernel size has an inference time of 459 ms. The first convolution of FlowNet2-s with a 7x7 kernel takes 172 ms of inference time. By reducing the kernel size to 3x3, the convolution takes only 20% and 12% of the time, respectively. The number of parameters drops by 2/3 approximately. This allows for more convolutional layers before abysmal performance is reached. Benchmark results for FlowNetS and FlowNet2-s can be found in table 7.1 and 7.2, respectively. Benchmarking of FlowNetS was halted after adding layer `c4_1` and run-time approached a second.
It can be seen that the number of parameters gives an approximation of the memory occupied but is not the only factor. Comparing the FlowNetS encoder architecture with convolutions up to `c3_1` with the FlowNet2-s encoder architecture with convolutions up to `c5`. The former has 9.67E+05 parameters, the latter 9.70E+05. The total amount of occupied memory is 1.65E+06 and 1.35E+05 bytes, respectively. The former network has fewer parameters, but occupies more memory, and takes three times longer to run.
It can be seen that, at least for a FlowNet-like architecture, the first concern will be run-time performance. There is sufficient memory to store a 3.30E+06 parameter network, with 60% L3 memory to spare. Although L3 memory is slower, there is no obvious drop in run-time performance when the Autotiler assigns memory to it. The effects of where

[1] Available on GitHub

memory is tiled could be further investigated by tiling a small, identical network to L1, L2, or L3 memory, exclusively.
Run-times for the nominal 7x7 kernel in the first convolution are comparable to a network with six 3x3 kernel convolutions with 100 times the parameters. For improved performance, it is better to replace a single large kernel convolution with several smaller kernel convolutions, which deliver a larger receptive field. The reduction of kernel size comes at the cost of expressiveness [66]. The reduction in channels between FlowNetS and FlowNet2-s delivers large memory usage and run-time performance improvements.

Implementing depthwise separable convolutions as done by Wofk et al. [53] reduces the number of parameters by a factor 10, and the average run-time by over 90%.

Table 7.1: FlowNetS encoder with 3x3 kernel size - number of parameters, GAP8 SoC run-time performance and memory usage

c1	c2	c3	c3_1	c4	c4_1	c5	c5_1	c6	c6_1	Parameters [-]	Average run-time [ms]	Refresh rate [FPS]	L1 memory usage [bytes]	L1 memory usage [%]	L2 memory usage [bytes]	L2 memory usage [%]	L3 memory usage [bytes]	L3 memory usage [%]
x										4.67E+03	75	13	4.84E+04	99	2.42E+04	10	4.10E+05	5
x	x									7.97E+04	206	5	4.85E+04	99	2.21E+05	88	4.86E+05	6
x	x	x								3.77E+05	352	3	4.85E+04	99	2.33E+05	93	7.78E+05	9
x	x	x	x							9.67E+05	548	2	4.85E+04	99	2.35E+05	94	1.37E+06	16
x	x	x	x	x						2.15E+06	686	1	4.85E+04	99	2.42E+05	97	2.56E+06	30
x	x	x	x	x	x					4.51E+06	903	1	4.85E+04	99	2.45E+05	98	4.92E+06	59

Table 7.2: FlowNet2-s encoder with 3x3 kernel size - number of parameters, GAP8 SoC run-time performance and memory usage

c1	c2	c3	c3_1	c4	c4_1	c5	c5_1	c6	c6_1	Parameters [-]	Average run-time [ms]	Refresh rate [FPS]	L1 memory usage [bytes]	L1 memory usage [%]	L2 memory usage [bytes]	L2 memory usage [%]	L3 memory usage [bytes]	L3 memory usage [%]
x										1.75E+03	22	46	4.86E+04	100	1.56E+05	62	0.00E+00	0
x	x									1.26E+04	44	22	4.73E+04	97	1.74E+05	70	7.68E+04	1
x	x	x								5.50E+04	89	11	4.88E+04	100	2.17E+05	87	7.91E+04	1
x	x	x	x							1.38E+05	117	9	4.88E+04	100	2.18E+05	87	1.63E+05	2
x	x	x	x	x						3.06E+05	137	7	4.88E+04	100	2.50E+05	100	3.85E+05	5
x	x	x	x	x	x					6.38E+05	167	6	4.88E+04	100	2.50E+05	100	7.18E+05	9
x	x	x	x	x	x	x				9.70E+05	182	5	4.88E+04	100	2.50E+05	100	1.05E+06	13
x	x	x	x	x	x	x	x			1.30E+06	195	5	4.88E+04	100	2.50E+05	100	1.38E+06	16
x	x	x	x	x	x	x	x	x		1.97E+06	221	5	4.88E+04	100	2.50E+05	100	2.05E+06	24
x	x	x	x	x	x	x	x	x	x	3.30E+06	256	4	4.88E+04	100	2.50E+05	100	3.38E+06	40

Table 7.3: FlowNet2-s encoder with 3x3 kernel size and depthwise separable convolutions - number of parameters, GAP8 SoC run-time performance and memory usage

c1	c2	c3	c3_1	c4	c4_1	c5	c5_1	c6	c6_1	Parameters [-]	Average run-time [ms]	Refresh rate [FPS]	L1 memory usage [bytes]	L1 memory usage [%]	L2 memory usage [bytes]	L2 memory usage [%]	L3 memory usage [bytes]	L3 memory usage [%]
x	x	x	x	x	x	x	x	x	x	3.76E+05	19	53	4.85E+04	99	2.50E+05	100	3.91E+05	5

Chapter 8

Conclusion

There is currently no dense optical flow CNN suitable to run on nano quadcopter hardware. A literature review was conducted, in which it was found that the SOTA focuses on improving prediction accuracy while maintaining several to 10s of FPS on modern desktop computer hardware. These networks are unsuitable for running on nano quadcopter hardware.
Three classes of architectures can be identified in the literature. The first are U-Net shaped networks. These networks are straightforward, versatile but not memory friendly. Pyramid-shaped networks iterate flow predictions over an image or feature pyramid in a coarse to fine manner. Lower spatial resolution flow estimates are used to warp one of two input images or feature maps to the next, higher-resolution layer. By sharing a single decoder between pyramid levels, a decrease in the number of parameters can be achieved. The third class of architectures employ GRUs to iteratively update the optical flow. Several papers are dedicated to improved training, data augmentation, and regularization habits. Self-reported benchmark results of all discussed architectures are compared.

From the knowledge gap and literature review follows the following main research question:

How can optical flow estimation using end-to-end convolutional neural networks on nano quadcopters be achieved?

A framework was laid out to design a base optical flow CNN architecture and identify and mitigate bottlenecks in the architecture optimization. Broadly, the steps to be taken are:

1. Define performance metrics
2. Set-up pipeline
3. Define baseline model
4. Identify bottlenecks
5. Update, iterate

A proper selection of performance metrics is vital in evaluating and monitoring the process. Hui et al. [37] made large improvements to their network by monitoring the flow prediction accuracy at several locations in the network. A proper pipeline can speed up

the process of updating, iterating and measuring data. A baseline will be defined based on insights from the literature review. The measured performance metrics will indicate bottlenecks in the tested architecture. Several options are presented for updating the network, depending on the type of bottleneck. Generally, this implies a trade-off between the network's representational capacity, run-time performance, and the number of parameters.

The architecture will be designed for deployment on the CrazyFlie 2.X nano quadcopter, equipped with an AI-deck for faster CNN inference. The architecture can be designed in any deep learning framework that supports the open neural network exchange (ONNX) filetype. Several up-to-date GitHub repositories are available for optical flow CNN training, written for PyTorch. The trained networks can be evaluated and benchmarked on a personal computer or converted for use on the AI-deck using GAP SDK by GreenWaves Technologies. Application code must be written in c to deploy the network. The supported operations are limited by the kernels available for the GAP8 SoC that runs on the AI-deck.

The project will provide insights into the miniaturization of optical flow networks and related computer vision fields and propose the first dense optical flow net to run on a nano quadcopter. A nano quadcopter can use dense optical flow for a large variety of applications including autonomous navigation, (self-)motion estimation, depth estimation and obstacle avoidance.

Bibliography

[1] James J. Gibson and Leonard Carmichael. *The Perception of the Visual World*. Houghton Mifflin, 1950, p. 235. DOI: 10.2307/426044.

[2] Guido C.H.E. De Croon. "Monocular distance estimation with optical flow maneuvers and efference copies: A stability-based strategy". In: *Bioinspiration and Biomimetics* 11.1 (Jan. 2016). ISSN: 17483190. DOI: 10.1088/1748-3190/11/1/016004.

[3] Andrew J Barry and Leslie A Kolodziejski Chair. *High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo Signature redacted LIBRARIES ARGhivti P o6I*. Tech. rep. 2016. URL: https://dspace.mit.edu/handle/1721.1/103718.

[4] Berthold K. Horn and Brian G Schunck. "Determining Optical Flow". In: *Techniques and Applications of Image Understanding*. Vol. 0281. 12. SPIE, Nov. 1981, pp. 319–331. DOI: 10.1117/12.965761.

[5] Min Bai et al. *Exploiting semantic information and deep matching for optical flow*. 2016. DOI: 10.1007/978-3-319-46466-4_10. arXiv: 1604.01827.

[6] Christian Bailer et al. *CNN-based patch matching for optical flow with thresholded hinge embedding loss*. July 2017. DOI: 10.1109/CVPR.2017.290. arXiv: 1607.08064.

[7] David Gadot and Lior Wolf. "PatchBatch: A Batch Augmented Loss for Optical Flow". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-Decem. 2016, pp. 4236–4245. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.459. arXiv: 1512.01815.

[8] Fatma Güney et al. "Deep discrete flow". In: *Springer* 10114 LNCS (2017), pp. 207–224. ISSN: 16113349. DOI: 10.1007/978-3-319-54190-7_13.

[9] Alexey Dosovitskiy et al. "FlowNet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. Vol. 2015 Inter. 2015, pp. 2758–2766. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.316. arXiv: 1504.06852.

[10] Eric Flamand et al. "GAP-8: A RISC-V SoC for AI at the Edge of the IoT". In: *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*. Vol. 2018-July. Institute of Electrical and Electronics Engineers Inc., Aug. 2018. ISBN: 9781538674796. DOI: 10.1109/ASAP.2018.8445101.

[11] Liangzhen Lai, Naveen Suda, and Vikas Chandra. *CMSIS-NN: Efficient neural network kernels for arm cortex-M CPUs*. Jan. 2018. arXiv: 1801.06601.

[12] Angelo Garofalo et al. *PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors*. Feb. 2019. DOI: 10.1098/rsta.2019.0155. arXiv: 1908.11263v1.

[13] Daniele Palossi et al. "A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones". In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 8357–8371. ISSN: 23274662. DOI: 10.1109/JIOT.2019.2917066. arXiv: 1805.01831.

[14] Zhile Ren et al. "A fusion approach for multi-frame optical flow estimation". In: *Proceedings - 2019 IEEE Winter Conference on Applications of Computer Vision, WACV 2019*. 2019, pp. 2077–2086. ISBN: 9781728119755. DOI: 10.1109/WACV.2019.00225.

[15] Pengpeng Liu et al. "Selflow: Self-supervised learning of optical flow". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2019-June. Apr. 2019, pp. 4566–4575. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00470. arXiv: 1904.09117.

[16] Daniel Maurer and Andres Bruhn. "Proflow: Learning to predict optical flow". In: *British Machine Vision Conference 2018, BMVC 2018*. BMVA Press, 2019. arXiv: 1806.00800.

[17] Michal Neoral, Jan Šochman, and Jiří Matas. "Continual Occlusion and Optical Flow Estimation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11364 LNCS. Springer Verlag, 2019, pp. 159–174. ISBN: 9783030208691. DOI: 10.1007/978-3-030-20870-7_10. arXiv: 1811.01602.

[18] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. "Optical flow modeling and computation: A survey". In: *Computer Vision and Image Understanding* 134 (2015), pp. 1–21. ISSN: 1090235X. DOI: 10.1016/j.cviu.2015.02.008.

[19] Anurag Ranjan and Michael J. Black. "Optical flow estimation using a spatial pyramid network". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vol. 2017-Janua. Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 2720–2729. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.291. arXiv: 1611.00850.

[20] Deqing Sun et al. "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2017, pp. 8934–8943. ISBN: 9781538664209. DOI: 10.1109/CVPR.2018.00931. arXiv: 1709.02371.

[21] Aviram Bar-Haim and Lior Wolf. "ScopeFlow: Dynamic scene scoping for optical flow". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers (IEEE), Feb. 2020, pp. 7995–8004. DOI: 10.1109/CVPR42600.2020.00802. arXiv: 2002.10770.

[22] J L Barron, D J Fleet, and S S Beauchemin. "Performance of optical flow techniques". In: *International Journal of Computer Vision* 12.1 (1994), pp. 43–77. ISSN: 09205691. DOI: 10.1007/BF01420984.

[23] Simon Baker et al. "A database and evaluation methodology for optical flow". In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31. ISSN: 09205691. DOI: 10.1007/s11263-010-0390-2.

[24] Daniel J. Butler et al. "A Naturalistic Open Source Movie for Optical Flow Evaluation". In: vol. 7577 LNCS. PART 6. 2012, pp. 611–625. ISBN: 9783642337826. DOI: 10.1007/978-3-642-33783-3_44.

[25] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? the KITTI vision benchmark suite". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361. ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6248074.

[26] Moritz Menze and Andreas Geiger. "Object scene flow for autonomous vehicles". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 07-12-June. 2015, pp. 3061–3070. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298925.

[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 Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-Decem. IEEE Computer Society, Dec. 2016, pp. 4040–4048. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.438. arXiv: 1512.02134.

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9351. Springer Verlag, 2015, pp. 234–241. ISBN: 9783319245737. DOI: 10.1007/978-3-319-24574-4_28. arXiv: 1505.04597.

[29] D. B. de Jong, F. Paredes-Vallés, and G. C. H. E. de Croon. "How Do Neural Networks Estimate Optical Flow? A Neuropsychology-Inspired Study". In: *arxiv.org* (2020). arXiv: 2004.09317.

[30] Eddy Ilg et al. "FlowNet 2.0: Evolution of optical flow estimation with deep networks". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vol. 2017-Janua. 2017, pp. 1647–1655. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.179. arXiv: 1612.01925.

[31] Shanshan Zhao, Xi Li, and Omar El Farouk Bourahla. "Deep optical flow estimation via multi-scale correspondence structure learning". In: *IJCAI International Joint Conference on Artificial Intelligence*. July 2017, pp. 3490–3496. ISBN: 9780999241103. DOI: 10.24963/ijcai.2017/488. arXiv: 1707.07301.

[32] Yi Zhu and Shawn Newsam. "DenseNet for dense flow". In: *Proceedings - International Conference on Image Processing, ICIP* 2017-Septe (2018), pp. 790–794. ISSN: 15224880. DOI: 10.1109/ICIP.2017.8296389. arXiv: 1707.06316.

[33] Liang Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4 (Apr. 2018), pp. 834–848. ISSN: 01628828. DOI: 10.1109/TPAMI.2017.2699184. arXiv: 1606.00915.

[34] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, Nov. 2016. arXiv: 1511.07122.

[35] Deqing Sun et al. "Models Matter, so Does Training: An Empirical Study of CNNs for Optical Flow Estimation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.6 (Sept. 2018), pp. 1408–1423. ISSN: 19393539. DOI: 10.1109/TPAMI.2019.2894353. arXiv: 1809.05571.

[36] Tak Wai Hui, Xiaoou Tang, and Chen Change Loy. "LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8981–8989. ISBN: 9781538664209. DOI: 10.1109/CVPR.2018.00936. arXiv: 1805.07036.

[37] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. "A Lightweight Optical Flow CNN - Revisiting Data Fidelity and Regularization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Feb. 2020), pp. 1–1. ISSN: 0162-8828. DOI: 10.1109/tpami.2020.2976928. arXiv: 1903.07414.

[38] Tak-Wai Hui. *Supplementary Material for LiteFlowNet3: Resolving Correspondence Ambiguity for More Accurate Optical Flow Estimation 1 Color Code for Visualizing Flow Fields*. Tech. rep. 2020. URL: https://github.com/twhui/LiteFlowNet3.

[39] Gengshan Yang and Deva Ramanan. "Volumetric correspondence networks for optical flow". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019. URL: http://papers.nips.cc/paper/8367-volumetric-correspondence-networks-for-optical-flow.

[40] Shengyu Zhao et al. "MaskFlownet: Asymmetric Feature Matching With Learnable Occlusion Mask". In: *openaccess.thecvf.com*. Institute of Electrical and Electronics Engineers (IEEE), Mar. 2020, pp. 6277–6286. DOI: 10.1109/cvpr42600.2020.00631. arXiv: 2003.10955.

[41] Zhichao Yin, Trevor Darrell, and Fisher Yu. "Hierarchical discrete distribution decomposition for match density estimation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2019-June. IEEE Computer Society, Dec. 2019, pp. 6037–6046. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00620. arXiv: 1812.06264.

[42] Junhwa Hur and Stefan Roth. "Iterative residual refinement for joint optical flow and occlusion estimation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2019-June. IEEE Computer Society, Apr. 2019, pp. 5747–5756. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00590. arXiv: 1904.05290.

[43] Zachary Teed and Jia Deng. *RAFT: Recurrent all-pairs field transforms for optical flow*. Mar. 2020. arXiv: 2003.12039.

[44] Aria Ahmadi and Ioannis Patras. "Unsupervised convolutional neural networks for motion estimation". In: *Proceedings - International Conference on Image Processing, ICIP*. Vol. 2016-Augus. 2016, pp. 1629–1633. ISBN: 9781467399616. DOI: 10.1109/ICIP.2016.7532634. arXiv: 1601.06087.

[45] Jason J. Yu, Adam W. Harley, and Konstantinos G. Derpanis. "Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9915 LNCS. Springer Verlag, 2016, pp. 3–10. ISBN: 9783319494081. DOI: 10.1007/978-3-319-49409-8_1. arXiv: 1608.05842.

[46] Zhe Ren et al. "Unsupervised deep learning for optical flow estimation". In: *31st AAAI Conference on Artificial Intelligence, AAAI 2017*. 2017, pp. 1495–1501. URL: www.aaai.org.

[47] Yi Zhu et al. "Guided optical flow learning". In: *arXiv* (2017). ISSN: 23318422. arXiv: 1702.02295.

[48] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vol. 2017-Janua. Institute of Electrical and Electronics Engineers Inc., Aug. 2017, pp. 2261–2269. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.243. arXiv: 1608.06993.

[49] Gregor Urban et al. "Do deep convolutional nets really need to be deep and convolutional?" In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, Mar. 2017. arXiv: 1603.05691.

[50] Cristian Bucilă, Rich Caruana, and Alexandra Niculescu-Mizil. "Model compression". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Vol. 2006. 2006, pp. 535–541. ISBN: 1595933395. DOI: 10.1145/1150402.1150464.

[51] Simon Meister, Junhwa Hur, and Stefan Roth. "UnFlow: Unsupervised learning of optical flow with a bidirectional census loss". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. AAAI press, 2018, pp. 7251–7259. ISBN: 9781577358008. arXiv: 1711.07837.

[52] Yang Wang et al. "Occlusion Aware Unsupervised Learning of Optical Flow". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4884–4893. ISBN: 9781538664209. DOI: 10.1109/CVPR.2018.00513. arXiv: 1711.05890.

[53] Diana Wofk et al. "FastDepth: Fast monocular depth estimation on embedded systems". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 6101–6108. ISBN: 9781538660263. DOI: 10.1109/ICRA.2019.8794182. arXiv: 1903.03273.

[54] Andrew G. Howard et al. *MobileNets: Efficient convolutional neural networks for mobile vision applications*. Apr. 2017. arXiv: 1704.04861.

[55] Tianqi Chen et al. "TVM: An automated end-to-end optimizing compiler for deep learning". In: *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018*. USENIX Association, Feb. 2007, pp. 579–594. ISBN: 9781939133083. arXiv: 1802.04799.

[56] Yanchao Yang and Stefano Soatto. "Conditional prior networks for optical flow". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11219 LNCS. 2018, pp. 282–298. ISBN: 9783030012663. DOI: 10.1007/978-3-030-01267-0_17. arXiv: 1807.10378.

[57] Yi Zhu and Shawn Newsam. *Learning optical flow via dilated networks and occlusion reasoning*. 2018. DOI: 10.1109/ICIP.2018.8451790. arXiv: 1805.02733.

[58] Mingliang Zhai et al. "Ad-net: Attention Guided Network for Optical Flow Estimation Using Dilated Convolution". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. Vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 2207–2211. ISBN: 9781479981311. DOI: 10.1109/ICASSP.2019.8682229.

[59] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and less than 0.5MB model size". In: (2016). arXiv: 1602.07360.

[60] *Face Identification on a mW power budget*. Dec. 2019. URL: https://greenwaves-technologies.com/face_reid_on_gap8/.

[61] Nitin J. Sanket et al. "GapFlyt: Active vision based minimalist structure-less gap detection for quadrotor flight". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 2799–2806. ISSN: 23773766. DOI: 10.1109/LRA.2018.2843445.

[62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[63] *NN Quick Start Guide*. June 2020. URL: https://greenwaves-technologies.com/sdk-manuals/nn_quick_start_guide/.

[64] Zbigniew Wojna et al. "The Devil is in the Decoder: Classification, Regression and GANs". In: *International Journal of Computer Vision* 127.11-12 (2019), pp. 1694–1706. ISSN: 15731405. DOI: 10.1007/s11263-019-01170-8. arXiv: 1707.05847.

[65] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-Decem. IEEE Computer Society, Dec. 2016, pp. 770–778. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

[66] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. Tech. rep. arXiv: 1512.00567v3.