Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Advanced Data Augmentation for the RAFT Optical Flow Approach

Sebastian Fritsch

Course of Study:	Data Science
Examiner:	Prof. Dr.-Ing. Andrés Bruhn
Supervisor:	Azin Jahedi, M.Sc.
Commenced:	January 12, 2021
Completed:	July 12, 2021

Abstract

We add several new augmentation methods to RAFT, a deep learning architecture that is used to calculate the optical flow between two sequential images. Because RAFT is trained using supervised learning, it requires annotated training data that not only contains image sequences but also the corresponding ground truth optical flow. Since the optical flow cannot be automatically generated from arbitrary image sequences, synthetic data sets are created to train these networks. One drawback of these data sets is their small size and low variety of optical flows they contain. To increase this variety, one option is to use data augmentation techniques to modify the training samples before feeding them to the network. These augmentations can change the images of a sample on the pixel level, but also modify the geometry of these images and hence the optical flow as well. We conduct experiments during each training phase to find out which kind of augmentation at which intensity is able to increase the accuracy of the trained model when estimating the optical flow of MPI-Sintel. Furthermore we compare this accuracy to that achieved by the original RAFT implementation. We find out that it depends on the specific training phase which kind of augmentation and which intensity is beneficial for the model's performance. The model that uses our augmentations is able to beat the original RAFT implementation after both are trained on FlyingChairs and after both are trained FlyingChairs and FlyingThings3D afterwards. When using these models to estimate the optical flow of KITTI-15, these models then perform worse, which shows that ideal augmentation settings are dependent on the target data set. The results after training on MPI-Sintel in the third phase show that adding these augmentations does not necessarily improve the model's performance, as the model that uses advanced augmentations doesn't manage to beat the original RAFT implementation.

Contents

1 Introduction

One of the main challenges in current computer vision research is the accurate computation of optical flow. Optical flow is the apparent motion of one ore more objects that is perceived by an observer, triggered either by actual movement of the objects, by movement of the observer or by a combination of both. In the context of computer vision, optical flow refers to the movement of a pixel in the image plane between two consecutive frames in an image sequence as can be observed in Figure 1.1. This movement is described by a two-dimensional vector where the first component describes horizontal movement while the second component describes vertical movement.

Since every pixel of an image has its own flow vector, a visualization of the optical flow field of an entire image using these vector arrows quickly becomes unreadable. That's why usually the vector directions are colour coded, where the direction of the vector corresponds to a colour and the magnitude of the movement corresponds to the intensity of this colour. Figure 1.2 shows the colour codification that we use in this thesis, while Figure 1.3 shows two consecutive images and the corresponding optical flow between those images, colour coded as described above.

One of the ways to estimate the optical flow between two consecutive images is with the help of Convolutional Neural Network (CNN) architectures. These networks first require training on large, annotated data sets to afterwards be able to estimate the optical flow field of new image sequences. The data sets contain lots of training samples, each consisting of a pair of sequential images and the corresponding ground truth optical flow field between these images. In order to achieve a good accuracy when estimating the optical flow, the architectures need to generalize well and have to be

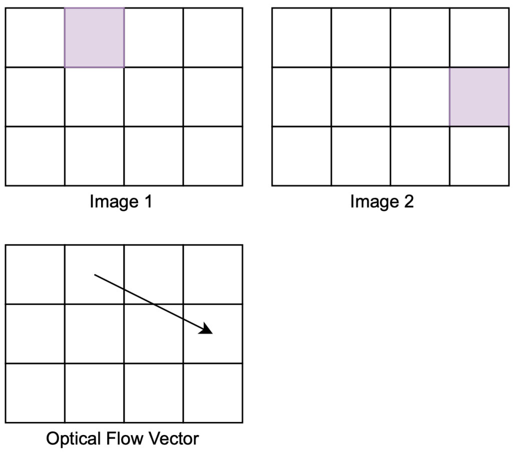


Figure 1.1: Example of the optical flow vector of a pixel.

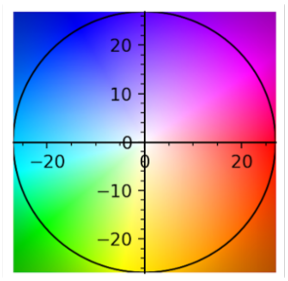
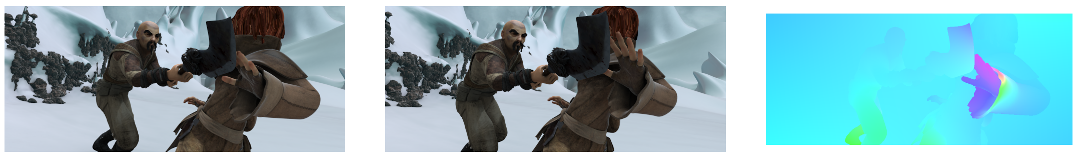


Figure 1.2: Colour coding for optical flow vectors.



(a) Image 1. **(b)** Image 2. **(c)** Colour coded optical flow.

Figure 1.3: Example of an image pair and its colour coded flow field.

trained with a large variety of optical flows. One way to increase this variety of optical flows that is found in the data sets is to use data augmentation. This procedure randomly modifies the training image pairs and the corresponding optical flow before feeding them into the network.

In this thesis, we study the impact of adding various data augmentation methods to the RAFT (Recurrent All-Pairs Field Transforms) architecture [TD20], a deep learning architecture released in 2020. This network is trained sequentially on several different data sets, and while RAFT already uses light data augmentations in every training phase, we want to explore whether we can improve the network's performance by using more advanced data augmentation methods. The question we want to answer is which combination of data augmentation methods during which training phase is able to improve the performance of the network.

In this thesis we first will give a brief overview of earlier approaches and deep learning architectures that were created to estimate optical flow, as well as describe RAFT's functionality. Furthermore we will introduce the data sets that are used for training RAFT. Then we describe in detail the data augmentations that we add to the RAFT architecture and our approach of conducting the experiments. During each of these experiments we will use one ore more of our data augmentations during the training of a model and afterwards evaluate the model's performance. Lastly we present our results and findings.

2 Related Work

2.1 Optical Flow Estimation

Conventional approaches have long treated the estimation of optical flow as an energy minimization problem [HS81]. These methods' accuracy in estimating optical flow is often restricted by the fact that they have to make very approximate assumptions about image brightness changes and the spatial structure of the flow.

In 2015, Fischer *et al.* [DFI+15] propose the first deep learning approach to estimating optical flow, building two CNN networks, FlowNetS and FlowNetC. While these aren't able to match the performance of conventional approaches, they already establish a multi-phase training schedule that consists of first training on a large synthetic data set and afterwards finetuning the model on another data set that has similar properties to the target data set. The networks already make use of several methods of data augmentation during training: translation, rotation and scaling of the images and the optical flow field and changing the brightness, contrast, gamma and colour of the images, as well as adding Gaussian noise. Ilg *et al.* [IMS+17] stack several FlowNet networks into a large one, called FlowNet 2.0. While their data augmentations stay unchanged compared to FlowNet, they add another synthetic data set to the training schedule and note that the order in which the data sets are presented to the network during the learning phase influences its performance. The resulting network manages to beat conventional approaches but the architecture of several stacked networks leads to very high memory requirements and complicated training procedures.

Further progress is made by embedding classical principles into the network architecture: Spynet [RB17] uses coarse to fine spatial pyramids to improve its performance and create a model that is much smaller than FlowNet 2.0. Their data augmentations include rotations, scaling, changing brightness, contrast and saturation as well as adding Gaussian noise. Sun *et al.* [SYLK18] present PWC-Net, which instead of using image pyramids uses learned feature pyramids of the images. The resulting network is 17 times smaller than Flownet 2.0, while offering even better performance. Their training schedule and use of data augmentations is similar to the FlowNet networks. Hur and Roth [HR19] propose IRR, a network based on PWC that iteratively refines its optical flow estimation by reusing the same network block with shared weights while again using FlowNet's data augmentation scheme.

Another method to calculate optical flow is to use an unsupervised learning approach. Yu *et al.* [YHD16] introduce a model that, similarly to conventional methods, combines a data constancy term with a spatial smoothness term. While the performance of these networks is currently not as good as that of models that use a supervised approach, they can be trained on arbitrarily large data sets that do not need to include ground truth optical flow. As a consequence these unsupervised methods rarely use data augmentations, but Liu *et al.* [LZH+20] propose a framework that uses augmented training samples as a regularization technique.

2.2 Introduction to RAFT

In this thesis we will use the RAFT (Recurrent All-Pairs Field Transforms) architecture to estimate optical flow. This architecture also uses deep learning techniques, was introduced in 2020 and achieved great results on various benchmarks for optical flow [TD20]. Compared to other network architectures, RAFT in its original implementation only uses a very limited number of data augmentations during training. It consists of three stages:

1. feature extraction
2. computation of visual similarity
3. iteratively updating the optical flow.

Figure 2.1 gives on overview of the RAFT architecture.

The features are extracted from the input images using a convolutional neural network. This feature encoder network extracts 256 features per pixel from both input images. The resulting feature maps have 1/8 of the resolution of the input images. The context encoder, which is another network with an identical architecture, extracts features only from the first image.

In the second step, a correlation layer is constructed by computing the dot product of every vector of the first feature map and every vector of the second vector map. The higher the value of a dot product between two feature vectors is, the more similar the underlying pixels are. The last two dimensions of the resulting four-dimensional (horizontal position of vector 1, vertical position of vector 1, horizontal position of vector 2, vertical position of vector 2) correlation volume are then pooled at multiple scales to construct a set of volumes.

These multi-scale correlation volumes are then used in the next step, where a flow field is initialized at zero and an update operator iteratively updates this flow field by retrieving values from the correlation volumes. When applied to video or data sets that contain more than two sequential images, the user can choose not to initialize the flow field at zero but to instead use the flow field of the previous two images to acquire a better starting point. This process is called warm start.

Since the resulting optical flow field only has 1/8 of the resolution of the input images, the final step is to upsample the optical flow to the original image dimensions.

2.3 Data Sets

Deep learning networks that rely on supervised training require large datasets that contain image pairs and the corresponding ground truth optical flow between these images. When training a neural network, its parameters are first set randomly and the image pairs of the training samples are fed into the network. The resulting estimated optical flow is then compared to ground truth optical flow of the training samples by using a loss function. The goal is to minimize this loss function, so that the estimated flow and the ground truth optical flow are as close as possible. Since the loss function is dependent on all parameters of the network, minimizing this function requires adjusting these parameters, which is done iteratively using gradient descent and backpropagation.

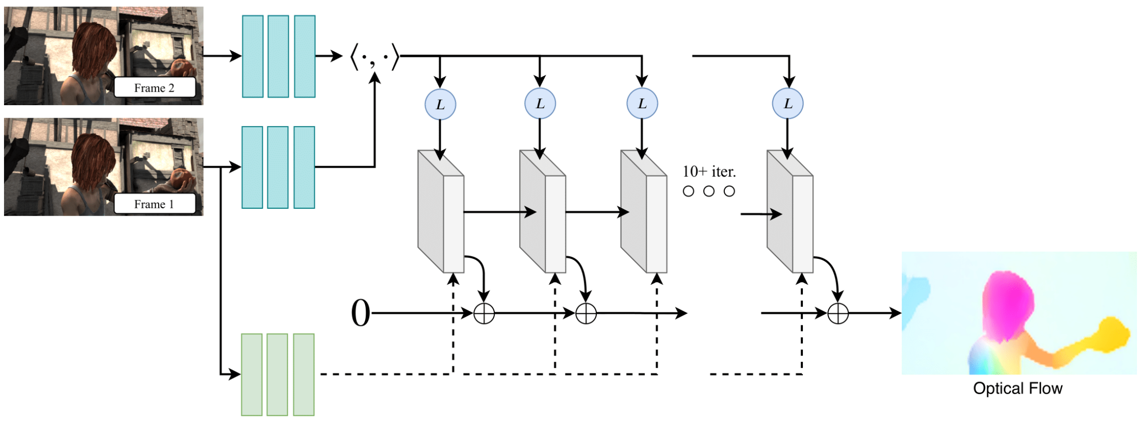


Figure 2.1: Overview of the RAFT network. Image source: [TD20].

In many other areas of computer vision like segmentation, object recognition etc., the ground truth can be captured by specialized sensors or via manual labeling. This means that datasets of arbitrarily large sizes can be created. In the case of optical flow, this is not the case. There are no sensors that can automatically capture the ground truth optical flow of an image sequence and since the optical flow can differ from pixel to pixel, manual labeling of the flow is not feasible even for low resolution images. That's why the data sets that are used when training the network have to be created synthetically. This section introduces the data sets that we use when training the RAFT network.

MPI-Sintel

The MPI-Sintel data set [BWSB12] was created by using scenes from the open source 3D animated short film ”Sintel”. It contains two versions for each of the 1,041 training samples: the more challenging final pass includes atmospheric effects and motion blur to mimic natural scenes, while the clean pass does not.

FlyingChairs

The FlyingChairs data set was created for training the Flownet network [DFI+15]. It consists of more than 22,000 image pairs and the corresponding ground truth optical flow fields. The images are random background images from Flickr, which are overlayed with renderings of 3D chair models. These chairs move between the first and the second image, while the background image also slightly moves. Both of these movements are planar, so there is no size increase or decrease of the chair models between the two images.

FlyingThings3D

The FlyingThings3D data set [MIH+16], which includes around 40,000 samples, is at first sight similar to the FlyingChairs data set: the images contain rendering of 3D objects in front of static backgrounds. The difference is that the objects show true 3D motion and there is a greater variety

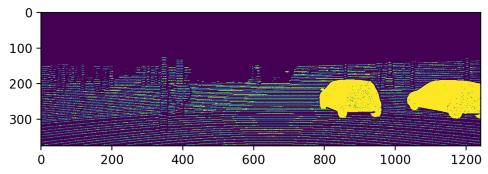


Figure 2.2: Example of the sparse optical flow of a KITTI-15 sample. Only yellow pixels contain valid ground truth optical flow.

of objects. Like the Sintel data set, the data set includes two versions of every sample: the final pass includes additional effects like depth-of-field blur, motion blur, sunlight glare and gamma curve manipulation. The clean pass does not include these effects.

KITTI-15

The KITTI-15 dataset [MG15] consists of 200 training samples. These are real world traffic scenes, where the ground truth optical flow was obtained by simultaneously recording the scenes with a camera and a 3D laser scanner. One drawback of this technique is that the optical flow of distant object cannot be captured in this way, hence there is only sparse ground truth optical flow available. This means that in contrast to the data sets described above, the optical flow is only available for a small number of pixels of the images, as observable in Figure 2.2: ground truth optical flow is only available for the yellow pixels.

HD1K

The HD1K benchmark suite [KNH+16] contains a data set that consists of more than 1000 training samples. Similar to the KITTI-15 data set, the images show diverse traffic scenarios. These real world scenes were captured during different times of the day and can include lens flare effects, reflections and other weather effects. As is the case with KITTI-15, the training samples only offer sparse ground truth optical flow.

3 Adding Advanced Data Augmentations to RAFT

The fact that all data sets introduced in the previous section are mostly synthetic and also relatively small presents a problem: the variety of different optical flows that a network is trained on might be too low, and in turn it may very well be the case that a model overfits this data. Overfitting means that while the trained model explains the training samples very well, its generalization capability is weak and so accuracy of optical flow estimations of new data may be low. Because the generalization capability is crucial for the performance of a model, we have to find ways to prevent overfitting and increase the variety of optical flows that a network trains on. One way to achieve this is to use data augmentations.

In this chapter, the different types of data augmentations that we implement for use during training the network are introduced. We implement the augmentations in Pytorch[1] to be compatible to the RAFT network: during training, the image pairs and optical flow are converted into tensors. Our data augmentations are then applied to these tensors and afterwards they are fed into the RAFT network. Furthermore, using Pytorch enables us the use of its torchvision library which includes many methods that facilitate the construction of our data augmentations.

There exist a great number of other data augmentations that we have not included. Our selection is mostly based on those successfully used in FlowNet [DFI+15], Flownet 2.0 [IMS+17], PWC-Net [SYLK18] and VCN [YR19] and is more varied than that of the RAFT original implementation. Generally, we can distinguish two kinds of data augmentations: geometric augmentations and photometric augmentations.

3.1 Geometric Augmentations

Geometric augmentations change the geometry of one or both of the images. Since the optical flow describes the movement of all pixels between the first and the second image, any changes in geometry of any of these images hence also inevitably change the optical flow. This means that using these augmentations we can increase the variety of optical flow geometries that the network learns during training.

[1] www.pytorch.org

3.1.1 Random Flipping

There are two flipping augmentations: horizontal flipping and vertical flipping. These are straight forward: both images are flipped along the horizontal or the vertical axis of the images. Additionally, the optical flow has to be flipped as well: in the case of horizontal flipping, the x-components of the optical flow have to be flipped, while the y-components stay the same. In the case of vertical flipping, the y-components are flipped and the x-components stay the same. Figure 3.1 shows an example of an image pair and its corresponding optical flow in its original orientation (first row of images), in a horizontally flipped orientation (second row of images) and in a vertically flipped orientation (third row of images).

Once a flipping augmentation is active, the probability of a training sample being flipped can be adjusted. In our experiments, the probability that a training sample (consisting of the image pair and its flow) is flipped is 50 percent. That means with both flipping augmentations active, the probabilities are evenly distributed among the four cases:

- sample is not flipped
- sample is only flipped horizontally
- sample is only flipped vertically
- sample is flipped both horizontally and vertically

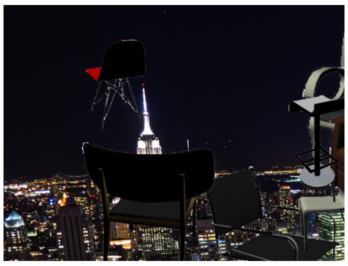
3.1.2 Random Scaling

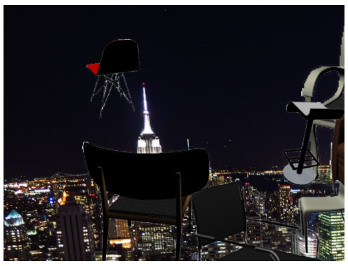
The scaling augmentation can increase or decrease the dimensions of the images and flow field by multiplying the height and width with a scaling factor and then resizing the images and flow field to these new dimensions. In our case, this resizing is achieved via the torchvision library, using bilinear interpolation. After the resizing of the images and the optical flow field, all flow vectors of the optical flow field needs to be adjusted to account for the new dimensions: The x-component of each flow vector is multiplied with the ratio between the new and the old image width, while the y-component of each flow vector is multiplied with the ratio between the new and old image height. The scaling factor that is applied to each training sample is a random variable that is uniformly distributed in a range that we specify. Via this range we can influence the intensity of the augmentation. There are two ways to augment an image using scaling:

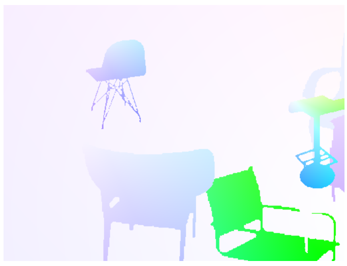
- Random symmetric scaling: both images are rescaled
- Random asymmetric scaling: only the second image is rescaled

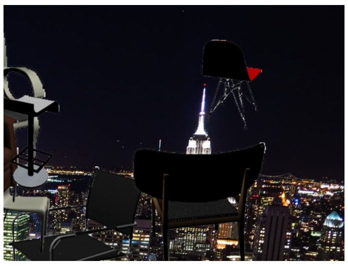
Random Symmetric Scaling

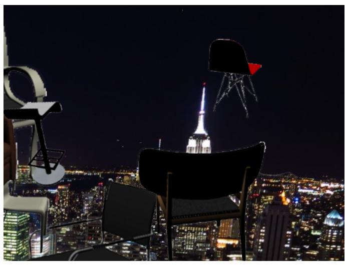
RAFT requires training samples whose dimensions are divisible by 8. This means that before feeding the image pairs and their corresponding optical flow of a training sample into the network, the images and their flow field always have to be cropped to such dimensions. This crop size is fixed for every training phase and is not changed by any augmentation. The white rectangle in the images in Figure 3.2 signifies the border of the image that will be cropped at this arbitrary position and then fed into the network. By using the scaling augmentation, we can increase or decrease the

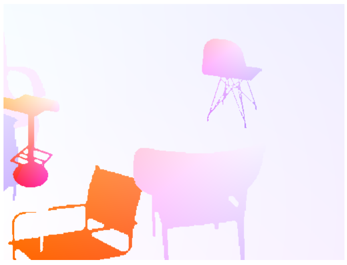

(a) Original image 1.

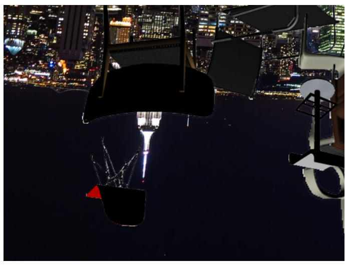

(b) Original image 2.

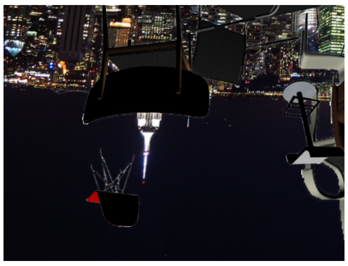

(c) Original optical flow.


(d) Horizontally flipped image 1.


(e) Horizontally flipped image 2.


(f) Horizontally flipped optical flow.


(g) Vertically flipped image 1.


(h) Vertically flipped image 2.

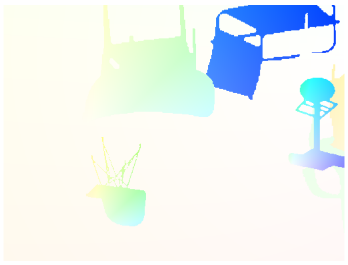
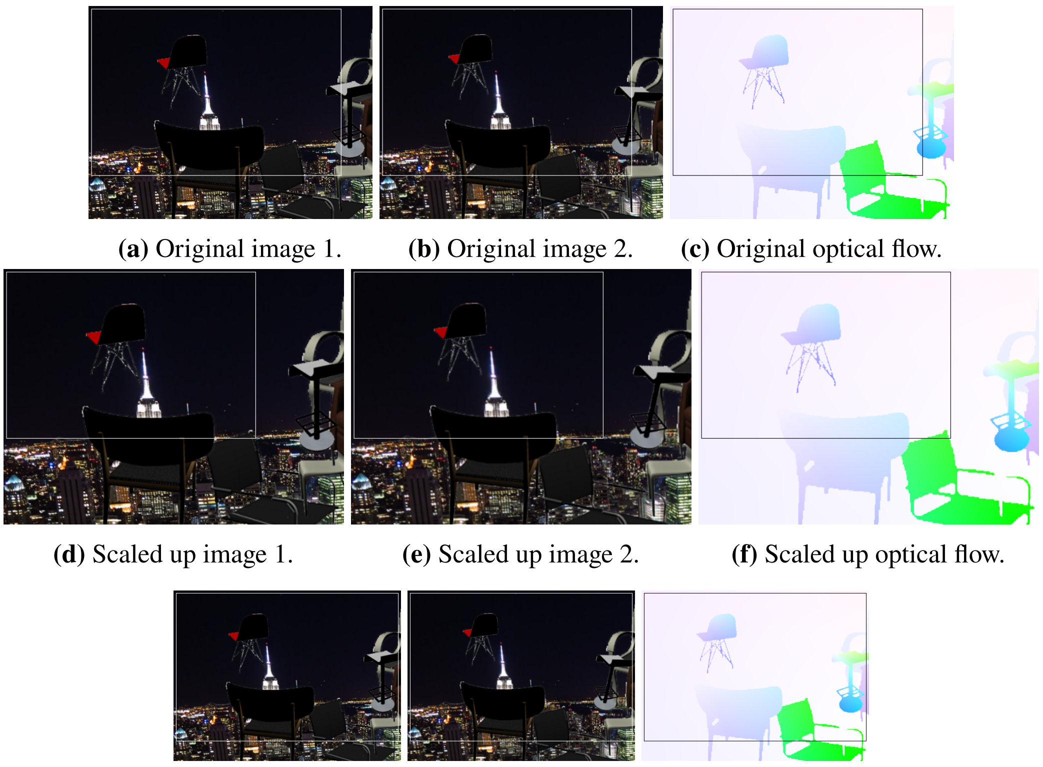

(i) Vertically flipped optical flow.

Figure 3.1: Horizontal and vertical flipping.

amount of objects that will be part of this cropped image. The middle image row of Figure 3.2 shows both images and the optical flow after their dimensions were increased. As the cropping size is unaffected by this, the resulting cropped image now contains less objects than before, for example only a small part of the chair on the lower right is now part of the cropped image. Conversely, in the bottom image row of Figure 3.2 the dimensions of the images and optical flow were decreased. This results in the cropped image containing more objects and flow information than the original, non augmented image.

The minimal scaling factor that can be applied to the images depends on the fixed crop size: in case that the resulting dimensions after rescaling would be smaller than the crop size, the factor is adjusted so that both the new height and width of the image are at least as large as the corresponding crop width and height.


(a) Original image 1. (b) Original image 2. (c) Original optical flow.

(d) Scaled up image 1. (e) Scaled up image 2. (f) Scaled up optical flow.

(g) Scaled down image 1. (h) Scaled down image 2. (i) Scaled down optical flow.

Figure 3.2: Symmetric scaling.

Random Asymmetric Scaling

In asymmetric scaling, only the size of the second image is increased or decreased. If the size of the second image is increased while the crop size stays the same, then as before fewer objects are part of the cropped image, and the size of the remaining objects inside the cropped image increases. When there is no scaling in the first image, then the motion between the first and second image is akin to zooming in, i.e. the objects in the cropped space appear larger in the second image than they do in the first image.

Conversely, if the scale of the second image is decreased, the objects in the cropped images then appear smaller in the second image than in the first image and the motion is akin to zooming out.

To apply this augmentation, first the second image is scaled up or down using a scaling factor and bilinear interpolation. Then the optical flow field has to be adjusted:
Let $v = (x, y)$ the displacement vector between the position of an arbitrary but fixed pixel between image 1 and image 2.
Let $v' = (x', y')$ the displacement vector between the original position of the pixel in image 2 and the position of the pixel in the scaled image 2'. This displacement vector is dependent on two things:

- The scaling factor, since the larger the re-scaled image is, the larger is the displacement vector between the pixel's position in the original and the re-scaled version of the image and vice versa.
- The distance of the pixel from the center of the image. The closer a pixel is from the center of the image, the less its position is affected by any resizing.

Let V the resulting displacement vector between the pixel in image 1 and the pixel in the scaled image 2'.

$$V = v + v' = (x + x', y + y')$$

After the adjustment of all flow vectors, the image pairs and the flow field are cropped.

Figure 3.3 shows both variants in which asymmetric scaling can be used. In the middle image row, the size of the second image is decreased, which means the objects inside the (unchanged) cropped area now appear smaller than they do in image 1. The resulting flow shows this zooming out motion.

In the bottom row of Figure 3.3, the size of the second image is increased, which leads to the objects in the cropped area now appearing larger in image 2 than they do in image 1. Now the resulting flow shows this zooming in motion.

3.1.3 Random Rotation

The rotation augmentation rotates the images and the optical flow field of a training sample around the center. The angle of the rotation of each training sample is a random variable that is uniformly distributed in a range that we specify. Through choosing this range we influence the intensity of the augmentation. There are two ways to augment a training sample using rotation:

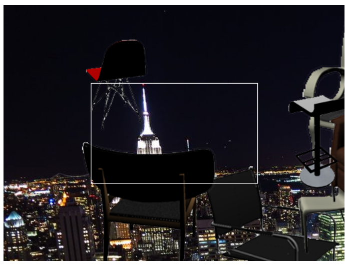
- Random symmetric rotation: both images are rotated the same amount
- Random asymmetric rotation: only the second image is rotated

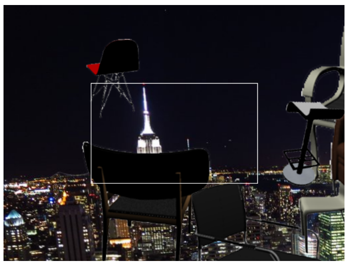
Random Symmetric Rotation

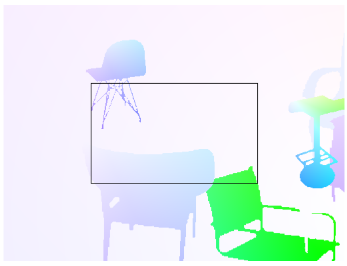
The first step in applying this augmentation is rotating the image pairs and the optical flow field around a random angle θ. While after this step the position of each flow vector in the rotated flow field is correct, as it is on the same coordinates as the pixel that it relates to on the rotated images, the flow vectors themselves now have to be rotated as well so that they again point in the correct direction. This is done by multiplying each flow vector with a rotation matrix R:

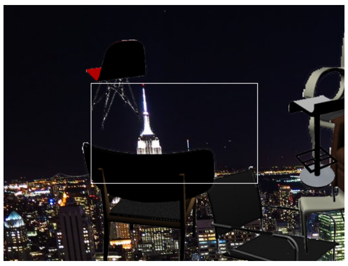
$$R = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

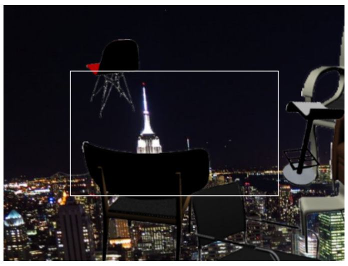
The rotation above results in images that contain black borders, as visible in the second row of figure 3.4. To eliminate these borders, the images and the optical flow field are scaled up (using the same technique as the scaling augmentation described above) with a factor that results in an

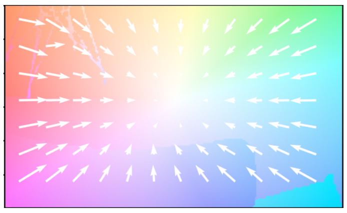

(a) Original image 1.

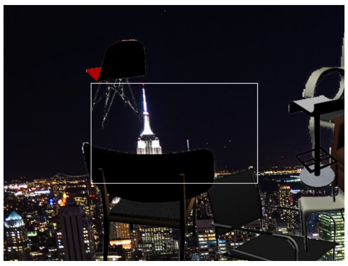

(b) Original image 2.

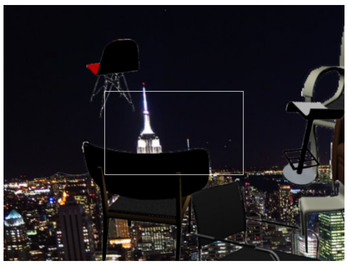

(c) Original optical flow.


(d) Image 1, unchanged.


(e) Image 2, decreased scale.


(f) Cropped resulting optical flow.


(g) Image 1, unchanged.


(h) Image 2, increased scale.

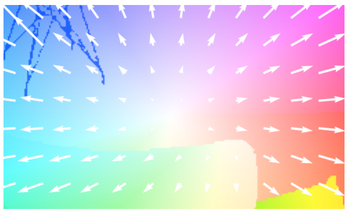
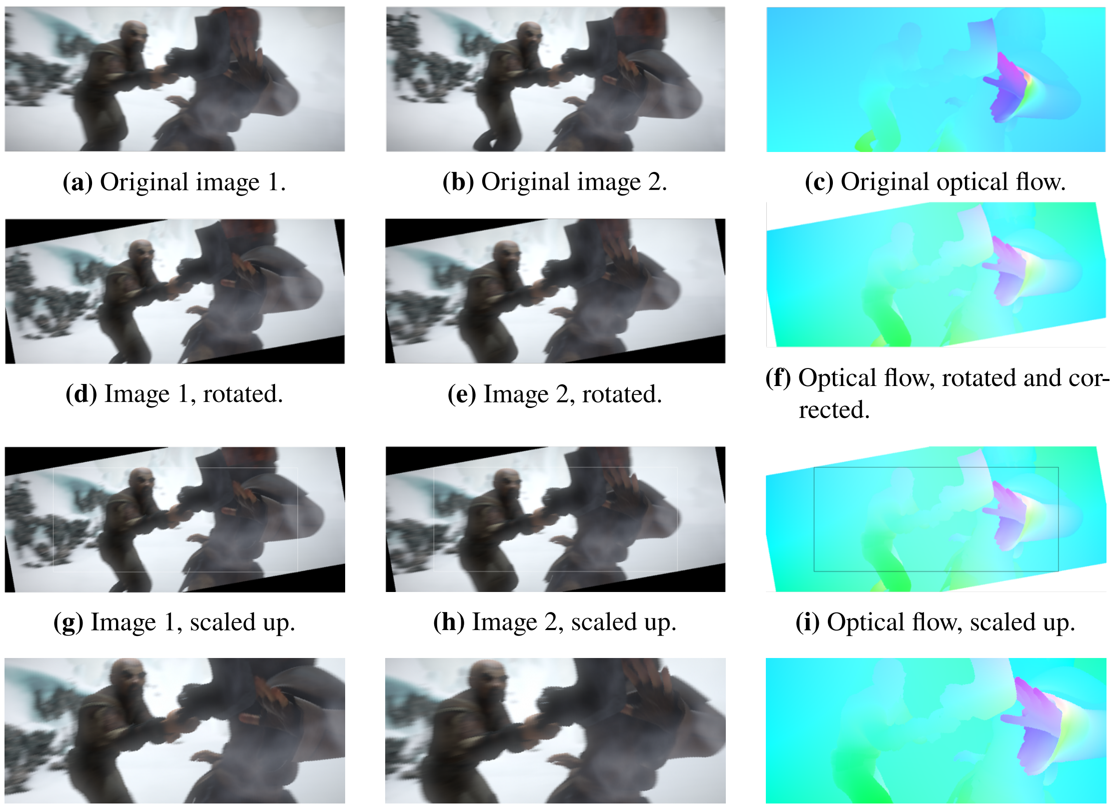

(i) Cropped resulting optical flow.

Figure 3.3: Asymmetric Scaling.

image size that allows us to crop out an area of the original image dimensions that doesn't contain any black borders. The third row of Figure 3.4 shows the images and the optical flow at their new increased size, with the white rectangle inside the images (and the black rectangle inside the optical flow) having the same dimensions as the original image pairs and optical flow. In the final step, these border free images (that have the same dimensions as the original images) are then cropped out, as shown in the bottom row of Figure 3.4.

Random Asymmetric Rotation

In this augmentation, only the second image is rotated. After this rotation, the flow field has to be adjusted:
Let $v = (x, y)$ the displacement vector of the position of a pixel between image 1 and image 2.
Let $v' = (x', y')$ the displacement vector between the original position of the pixel in image 2 and the position of the pixel in the rotated image 2'. Similarly to the asymmetric scaling augmentation, this displacement vector v' is influenced by two things:



(a) Original image 1. **(b)** Original image 2. **(c)** Original optical flow.

(d) Image 1, rotated. **(e)** Image 2, rotated. **(f)** Optical flow, rotated and corrected.

(g) Image 1, scaled up. **(h)** Image 2, scaled up. **(i)** Optical flow, scaled up.

(j) Image 1, cropped. **(k)** Image 2, cropped. **(l)** Optical flow, cropped.

Figure 3.4: Symmetric Rotation.

- The amount of rotation. The higher this amount is, the larger is the distance (and hence the displacement vector) between the original position of the pixel and the position in the rotated image.
- The distance of the pixel from the rotation center, which in our augmentation is the center of the image. The larger this distance is, the stronger the pixel is displaced by the rotation. The opposite is true as well: a pixel right in the center of the image is not influenced by any amount of rotation. This means that its displacement vector v' is zero.

Let V the resulting vector between the pixel in image 1 and the pixel in the rotated image 2'.

$$V = v + v' = (x + x', y + y')$$

The flow field is hence adjusted so that every vector has this correct orientation. As in the case of symmetric rotation, the second, rotated image now contains black borders. That's why at this point, similarly to the case of symmetric rotation, both images as well as the optical flow are now again scaled up in order to be able to crop out an area that has the same dimensions as the original images and optical flow but doesn't include any black borders. The middle row of images of Figure 3.5 shows these scaled up images and flow, with the white rectangles having the same dimension as the

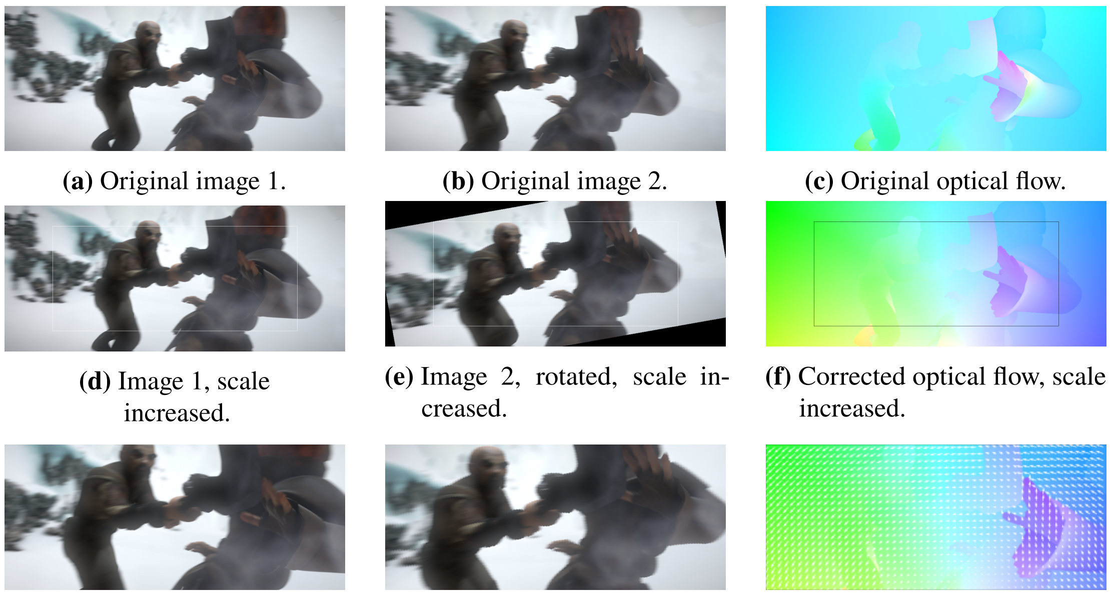

(a) Original image 1. **(b)** Original image 2. **(c)** Original optical flow. **(d)** Image 1, scale increased. **(e)** Image 2, rotated, scale increased. **(f)** Corrected optical flow, scale increased. **(g)** Image 1, cropped. **(h)** Image 2, cropped. **(i)** Cropped resulting flow.

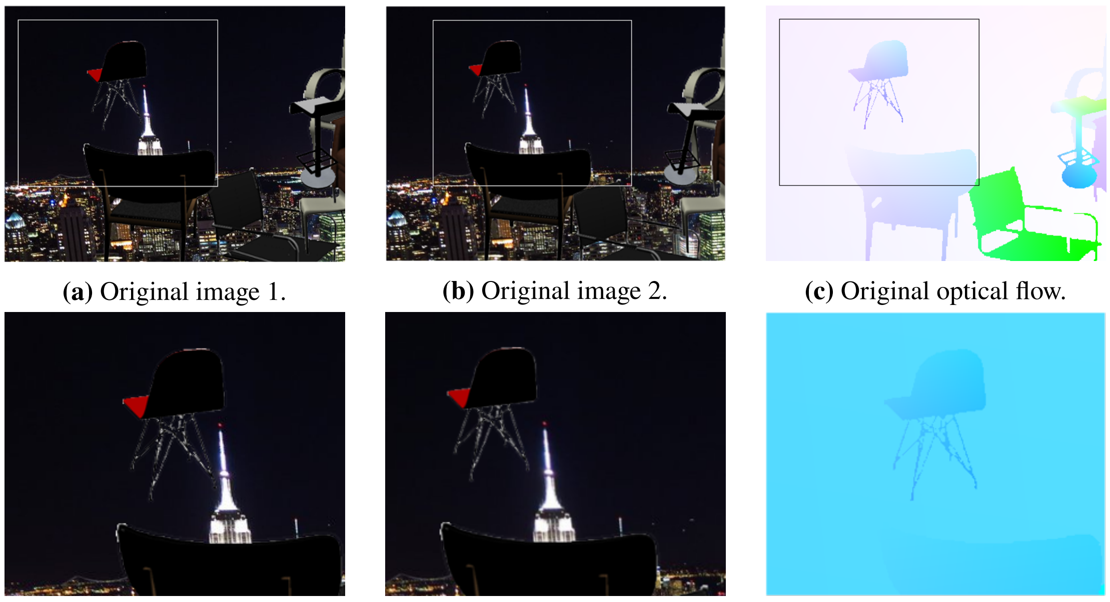
Figure 3.5: Asymmetric Rotation.

original images. Lastly, the images and optical flow field are cropped, as shown in the bottom row of Figure 3.5. The resulting optical flow shows the rotation between the two images: the further away a pixel is from the rotation center (i.e the image center), the stronger the rotation influences the pixel's optical flow vector.

3.1.4 Random Translation

The final geometric data augmentation is the translation augmentation. In this augmentation, the two images are cropped at two different positions. In the upper row of Figure 3.6, we can see that the cropped area (as indicated by the white rectangles) in the first image and the optical flow field is located to the left of the cropped area of the second image. After cropping the images and flow field, this results in the movement of the objects inside the cropped images in this direction: In Figure 3.6, we can observe that the objects traveled to the left between image 1 and image 2. The flow is corrected by adding the offsets between the two crop positions: the horizontal offset is added to the x-component of each flow vector of the flow field, while the vertical offset is added to the y-component of each flow vector of the flow field. Hence the resulting flow field shows the translative movement that was induced by the different cropping positions between the first and second image.

The amount of offset between the first and second image is a random varible uniformly distributed in a range that we specify. The choice of range allows us to influence the intensity of this augmentation. Of course the possible offsets are also limited by the (random) cropping position of the first image and the crop size itself: if for example the horizontal cropping position of the first image is at the left border of the image, then the horizontal offset between the two cropping positions cannot be negative, as this would mean the second image would be cropped outside of its borders. Furthermore



(a) Original image 1. **(b)** Original image 2. **(c)** Original optical flow.

(d) Cropped image 1. **(e)** Cropped image 2. **(f)** Cropped resulting flow.

Figure 3.6: Translation

if an image has a width of x pixels and a crop width of c pixels, the maximal offset between the two cropped images is $x - c$ pixels, and is further reduced if the horizontal cropping position of the first image is not directly at the image border.

3.2 Photometric Augmentations

In contrast to geometric augmentations, photometric augmentations simply change the image properties of one or both of the images at the pixel level. Since there is no change in geometry of the images, the optical flow field does not have to be adjusted when applying any of these augmentations.

3.2.1 Random Brightness Shifting

There are two different augmentations that shift the brightness of the images:

- Random additive brightness shift
- Random multiplicative brightness shift

Both can be used in two ways: either both images are shifted the same amount or only the second image is shifted.

Random Additive Brightness Shifting

In this augmentation for any training sample a shifting value is randomly chosen from a normal distribution. The properties of this distribution, i.e. its mean and its standard deviation, are set beforehand and influence the augmentation's intensity. As we deal with 8 bit images, the values of all three colour channels in any pixel of the image are in a range between 0 and 255. The channel values are normalized by division through 255 and afterwards the random value is added to all channels for every pixel of the image. Afterwards we multiply all channel values with 255. If a channel value is larger than 255 or smaller than 0 after the addition of the random shifting value, the channel value is then set to 255 or 0 respectively. Since the added shifting value is the same across all pixels, all pixels of the image are shifted the same amount. If this value is positive, the image gets brighter, as shown in in Image 3.7b, if it is negative, the image gets darker as shown in Image 3.7c.

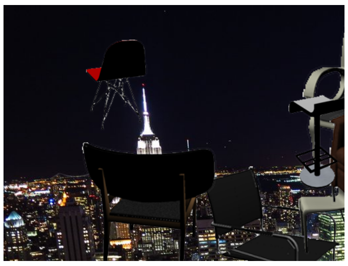
Random Multiplicative Brightness Shifting

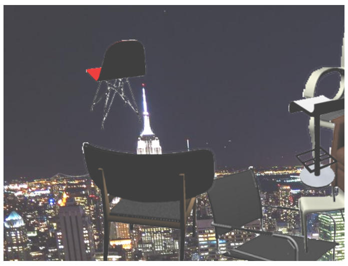
In this augmentation for any training sample a factor is randomly picked from a uniformly distributed range. Increasing or decreasing this range allows us to influence the augmentation's intensity. As above the colour channel values are normalized by dividing them by 255 and then lie in a range between 0 and 1. Then they are multiplied with the random factor and afterwards again multiplied with 255. A factor > 1 increases the brightness of each pixel, while a factor < 1 decreases its brightness. This multiplication has a different effect than the addition of a value before: bright pixels with high channel values are influenced much stronger by this multiplication than pixels that are darker and have low channel values. In Image 3.7e for example we can observe that while the brighter areas get much brighter, the dark background stays dark. This is not the case in the additive brightness augmentation, where an increase in brightness also increased the dark background, as seen in Image 3.7b.

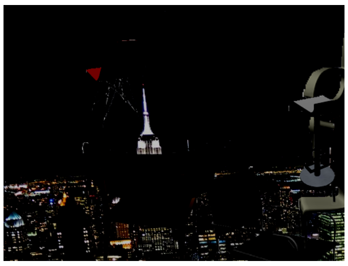
3.2.2 Random Contrast Shifting

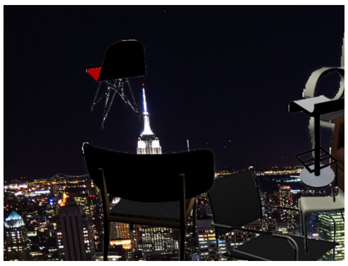
This augmentation allows us to increase or decrease the contrast of either both images or of only the second image. For any training sample a factor is chosen randomly out of uniformly distributed range that we specify. Then the average gray value of the image is computed. Then the image is adjusted in the following way:
Let c the randomly chosen contrast factor.
Let g the average gray value of the image.
Let $f(x, y)$ the colour values of the image at the position (x, y) before the application of the augmentation.
Let $\mathrm{out}(x, y)$ the colour values of the image at the position (x, y) after the application of the augmentation.

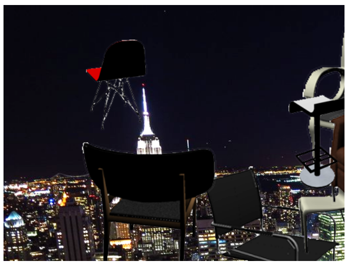
$$\mathrm{out}(x, y) = c * f(x, y) - (1 - c) * g$$


(a) Original image.


(b) Added brightness, value +0.25.


(c) Added brightness, value -0.25.


(d) Original image.


(e) Multiplicated brightness, factor 1.25

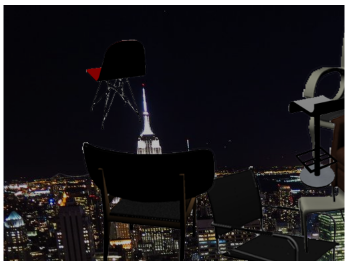
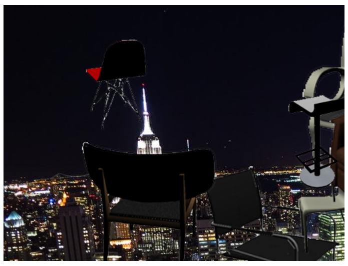
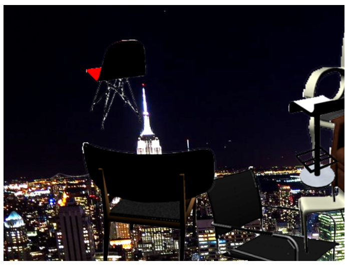

(f) Multiplicated brightness, factor 0.75

Figure 3.7: Additive and multiplicative brightness augmenations.


(a) Original image.


(b) Contrast augmentation, factor 1.5.

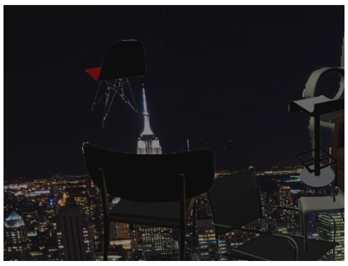

(c) Contrast augmentation, factor 0.5.

Figure 3.8: Contrast augmentation.

If the factor is larger than 1, the contrast of the image is increased, as visible in Image 3.8b. If the factor is between 0 and 1, the contrast of the image is decreased, as visible in Image 3.8c.
If the factor is 0, then every pixel of the resulting image has the average gray value, so there is no more contrast.

3.2.3 Random Gamma Shifting

With this augmentation we can change the gamma property of either both or the second one of the images of a training sample. For every training sample a value is randomly picked out of a uniformly distributed range we specify. The channel values are divided by 255 so that each one lies in a range between 0 and 1. Afterwards the channel values are raised by the random value.

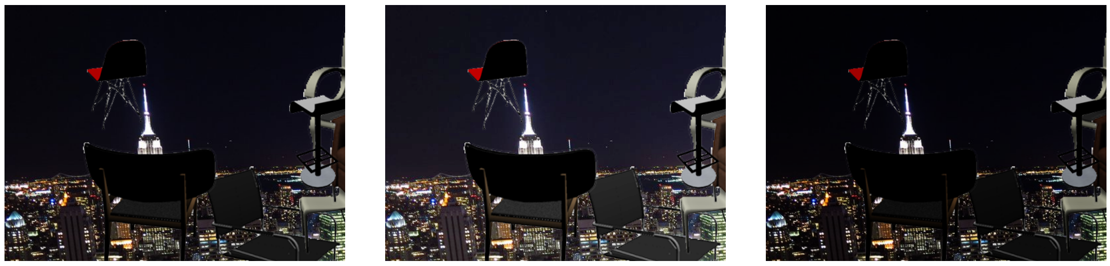

(a) Original image **(b)** Gamma augmentation, value = 0.75 **(c)** Gamma augmentation, value = 1.25

Figure 3.9: Gamma augmentation.

Let g the randomly chosen value.
Let $f(x, y)$ the colour values of the image at the position (x, y) before the application of the augmentation.
Let $\text{out}(x, y)$ the colour values of the image at the position (x, y) after the application af the augmentation.

$$\text{out}(x, y) = f(x, y)^g$$

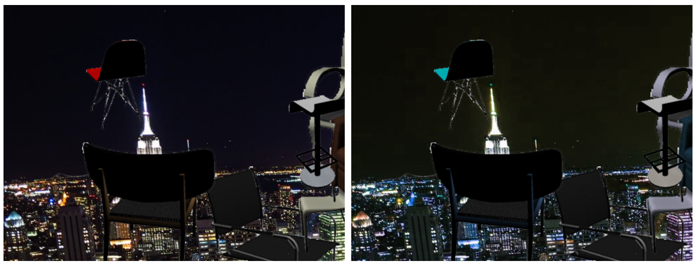
Afterwards the channel values are again multiplied with 255. As visible in Image 3.9b, values smaller than 1 make dark areas of the image lighter. This can be useful to increase the visibility of details in dark areas. Gamma values larger than 1 make dark areas darker, as visible in Image 3.9. This can be useful to improve the contrast when overall the image is very light.

3.2.4 Random Hue Shifting

This augmentation changes the hue of both images. To achieve this, the image is first converted into an HSV image. In this representation, the hue (H) channel is represented by a colour circle and every hue value corresponds to an angle on this circle. A value is then randomly chosen out of a specified uniformly distributed range and the hue values of all pixels are then shifted along the circle by this value. Afterwards, the image is converted back into the RGB representation. Figure 3.10 shows an example of this augmentation.

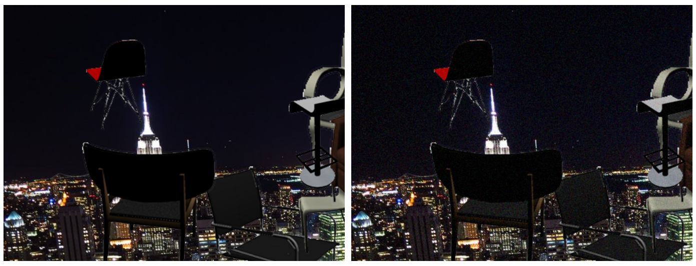
3.2.5 Random Additive Gaussian Noise

This augmentation adds noise to all three colour channels of the images. For every training sample we set up the Gaussian distribution of the noise by setting its mean as zero and randomly picking its standard deviation out of a uniformly distributed range we specify. This means that while every training sample is augmented with Gaussian-distributed noise, the standard deviation of the distribution of this noise differs between the training samples. For each colour channel of each pixel, a value is randomly picked out of the resulting Gaussian distribution and then added to the



(a) Original image. **(b)** Hue augmentation, value = 0.5

Figure 3.10: Hue augmentation.



(a) Original image. **(b)** Added Gaussian noise with a standard deviation of 15.

Figure 3.11: Additive Gaussian noise augmentation.

value of the colour channel. To make sure that all resulting values stay between 0 and 255, all resulting negative values are set to 0 and all resulting values larger than 255 are set to 255. Figure 3.11 gives an example of added noise.

3.2.6 Motion Blurring

Generally, motion blur is generated by convolving an image with a box function in the direction of the motion. Due to the fact that the final pass of the MPI-Sintel data contains motion blur effects, we want to add a motion blur augmentation during training in the hope to improve the network's performance when estimating the optical flow of image pairs that include motion blur. Since in our training images often times different objects move in different directions, ideally several kernels (one for each movement direction present in the image) would be generated and only areas close to a moving object would be convoluted with the correct kernel for the object's direction of movement. Unfortunately such an approach isn't viable when the procedure is used for augmenting training data, since in this case the speed of applying an augmentation is paramount and our attempts to implement the above behaviour were too slow to be used during training. This is why we implement two different motion blur augmentations:

- Random motion blur
- Nonrandom motion blur

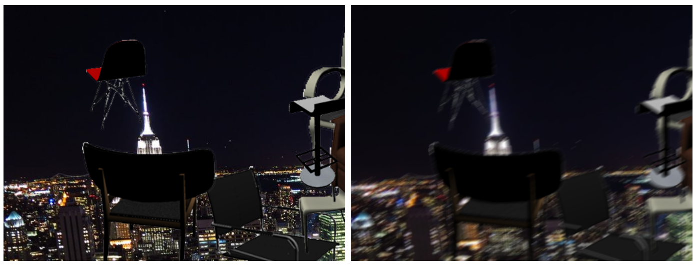

(a) Original image. **(b)** Applied motion blur.

Figure 3.12: Motion blur augmentation.

Random Motion Blurring

In this augmentation, we use the Motion Blur augmentation from the Albumentations framework[2]. This augmentation randomly chooses a direction and generates the corresponding kernel for this direction. Afterwards, the image is convolved with this kernel. Since both images of a training sample are augmented individually in this case, both are blurred in different directions.

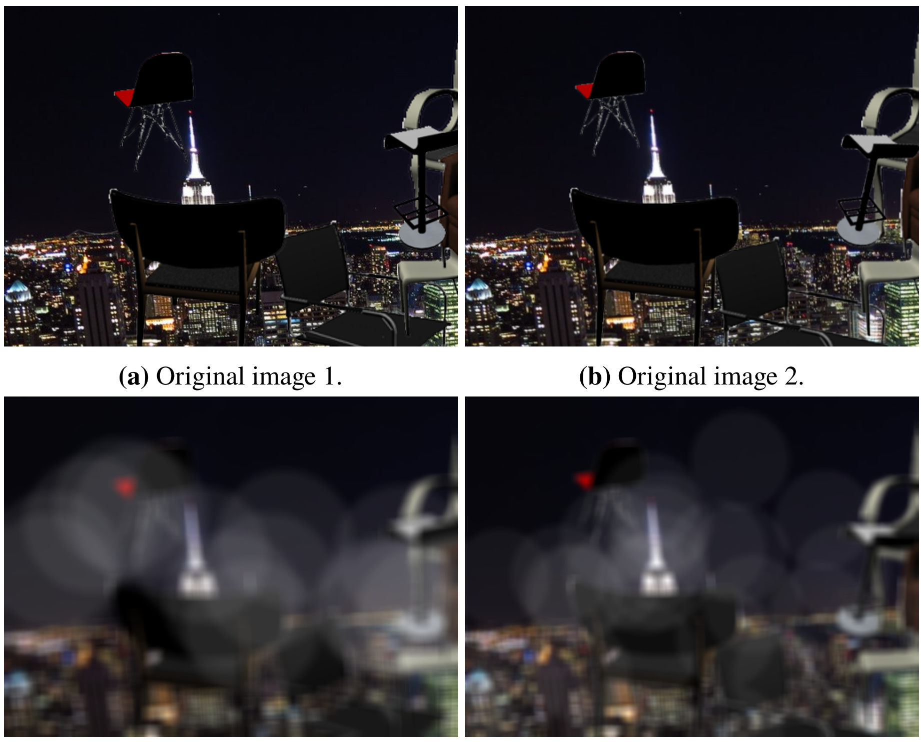
Nonrandom Motion Blurring

Our own motion blur augmentation uses the optical flow field to first calculate the average direction of all optical flow vectors that are larger then a set threshold. For example if this threshold is zero, then the average direction is the average of all optical flow vectors of the flow field. But if it is 20, it is the average direction of all optical flow vectors of the image that are larger than 20. This may be advantageous when we have a fast object that goes in one direction but there's slow background movement that goes into another direction. With this threshold, the slow background movement is ignored and the blur will be applied in the direction of the fast moving object. The resulting direction is used to build a box kernel which is then convolved with both images. In our experiments we set the threshold to 20. If there are no flow vectors that exceed this threshold, no blurring is applied. Figure 3.12 shows an example of an image that was augmented with nonrandom motion blur.

3.2.7 Fog

Here again the fact that the final pass of the MPI-Sintel data set includes weather effects such as fog motivates us to include a fog augmentation during training. We use the fog augmentation of the Albumentations framework, which tries to emulate fog by working as follows: at several random coordinates of an image, transparent white circles are added one by one. If parts of circles overlap, the transparency in this overlapping area is reduced. Finally, the complete image is blurred. Since both images of a training sample are augmented separately, the coordinates of the random fog circles differ between the two images. The bottom row of Figure 3.13 shows this: the left image is the first image of the image pair and the right image is the second image.

[2]www.albumentations.ai



(a) Original image 1. **(b)** Original image 2.

(c) Image 1 after fog augmentation. **(d)** Image 2 after fog augmentation.

Figure 3.13: Fog augmenation.

3.3 Augmentations of the Original RAFT Implementation

The original RAFT implementation only uses a comparably limited number of data augmentations. This section gives an overview and briefly explains any augmentations that are used in the original implementation but are not used by ours.

Geometric Augmentations

The original RAFT implementation uses three different geometric augmentations:

- Horizontal and Vertical Flipping
- Symmetric Scaling
- Symmetric Stretching

The Flipping and Symmetric Scaling augmentations work just like our versions described above. The only geometric augmentation we don't use in our implementation is the symmetric stretching augmentation. This augmentation is very similar to the symmetric scaling augmentation as both change the dimensions of the images. The difference is that while in the scaling augmentation, both height and width of the images are scaled by the same factor, here height and width are scaled with different factors. This means that the ratio between height and width is different after applying this augmentation.

Photometric Augmentations

There are four photometric augmentations in the original implementation of RAFT:

- Multiplicative brightness shifting
- Contrast shifting
- Hue shifting
- Saturation shifting
- Addition of occlusions

The brightness, contrast and hue augmentations again work like in our implementation described above. Additionally there's an augmentation which changes the saturation of the images. All four can also be used either symmetrically, i.e. on both images with the same parameters, or asymmetrically, where both images are augmented with different parameters and intensity. The decision which mode of the two is used is randomized: in 80 percent of all training samples, symmetric augmentation is used, while the remaining 20 percent are augmented asymmetrically.

The addition of occlusions is another augmentation that is not part of our implementation. This augmentation changes the second image of a training sample pair: The colour channel values inside several rectangular areas of random dimensions at random positions are replaced with the average colour of the image. This way these rectangles simulate occlusions, i.e. areas that are visible in the first image but are no longer visible in the second image because they may be covered by other objects.

4 Experimentation Approach

In this section, we describe the experimentation process we conduct to answer our research question: can we improve RAFT's performance by adding augmentation methods? Specifically, we test for every phase of our training pipeline what combinations of data augmentations and which intensities of these augmentations improve the final accuracy of the trained model the most.

4.1 Training Protocol

Like the original RAFT implementation, our network is trained in three phases: first we train the network on large and relatively simple data and then we finetune on a complex dataset:

1. Phase 1: Training on the FlyingChairs data set
2. Phase 2: Training on the FlyingThings3D dataset (both clean and final pass)
3. Phase 3: Training on a mix of several data sets: MPI-Sintel (both clean and final pass), FlyingThings3D (clean pass only), KITTI-15 and HD1K.

A training step includes

- sampling batches of image pairs and their corresponding optical flow from the data set
- applying specific data augmentations at specific intensities to the images and the optical flow
- cropping the images and optical flow to a dimensions that are divisible by 8
- feeding the images and optical flow into the network
- adjusting the network's parameters via a loss function and backpropagation

We conduct experiments in all three training phases. In each experiment, we activate certain data augmentations at certain intensities and then train a model on the phase-specific training set using these settings. Afterwards, the performance of this model is evaluated on the phase-specific evaluation set. Our goal is to find a combination of data augmentations and their intensities on every phase so that a model which is trained on all three of these phases achieves a better accuracy on MPI-Sintel than the original RAFT implementation. All experiments are carried out using a single NVIDIA RTX 3090 GPU.

Since our final goal is to build a network that works well on the MPI-Sintel data set, we generally evaluate every model after every experiment on the full MPI-Sintel training split. The only exception to this is the third training phase. Since in this phase we already use the MPI-Sintel training split to train our model, we cannot also use it to evaluate the performance of the trained model. Training and evaluating a model on the same data set doesn't give any insights on the generalization capabilities on the network, because very good performance in the evaluation could simply be a result of overfitting the model to the data. That's why in the third phase we split the MPI-Sintel training split into two disjoint sets, as described by Zhao *et al.* [ZSD+20]. The first one, that we call ”eval”, contains 225 of the 1041 image pairs of the original MPI-Sintel training split and is used for evaluating the models. The second one, called ”train”, contains the rest of the image pairs and is used to train the network in place of the original, full MPI-Sintel training split. Table 4.1 gives an overview of the data sets and evaluation set used for each training phase.

Phase	Training set(s)	Evaluation set
1	FlyingChairs	MPI-Sintel training split
2	FlyingThings3D	MPI-Sintel training split
3	MPI-Sintel training split subset ”train”	MPI-Sintel training split subset ”eval”
	& FlyingThings3D clean pass	
	& KITTI-15	
	& HD1K	

Table 4.1: Overview of the training and evaluation sets.

To measure the performance of a trained model, we use the average endpoint error (EPE). To calculate the EPE, a trained model first estimates the optical flow of all image pairs of an evaluation data set. These evaluation data sets also include the corresponding ground truth optical flow for each image pair. For every pixel of an image, the Euclidian distance between the ground truth optical flow vector of this pixel and the estimated optical flow is calculated. These distances are then averaged over all pixels of the image.

Let $V_{\text{gt, x}}$ the horizontal component of the ground truth optical flow of a pixel.
Let $V_{\text{gt, y}}$ the vertical component of the ground truth optical flow of this pixel.
Let $V_{\text{est, x}}$ the horizontal component of the estimated optical flow of this pixel.
Let $V_{\text{est, y}}$ the vertical component of the estimated optical flow of this pixel.

$$\text{EPE} = \frac{\sum_{\text{pixels}} \sqrt{(V_{\text{gt, x}} - V_{\text{est, x}})^2 + (V_{\text{gt, y}} - V_{\text{est, y}})^2}}{\text{number of pixels}}$$

Finally, the EPE is averaged over all samples of the evaluation data set.

4.2 Decreasing Training Time

Several issues prevent us from evaluating the influence of every single data augmentation separately. Any adjustment of the intensity of a data augmentation again necessitates time consuming training of the network, which limits the number of experiments that are achievable in a given time frame. Even if the ideal intensity for this augmentation was found, this would not guarantee that this intensity is ideal once another augmentation is added to the network. This interdependency leads to a number of experiments that is not feasible.

The goal is to decrease the number and time consumption of individual experiments while still finding well performing augmentation combinations and intensities. This influenced several design decisions of our experimentation process.

4.2.1 Grouping of Data Augmentations

To decrease the number of experiments, we group several data augmentations into data augmentation classes. Any augmentations that are part of such a class are never activated and evaluated individually. Instead, we add all augmentations of a class to the network at the same time. Furthermore, when studying the effects of intensity changes, we also change the intensity of all augmentations of the class at the same time. There are two augmentation classes.

Geometric Augmentation Class

This class contains all of our geometric augmentations:

- Flipping, horizontally and vertically
- Scaling, symmetrically and asymmetrically
- Rotation, symmetrically and asymmetrically
- Translation

Photometric Augmentation Class

This class contains most of our photometric augmentations:

- Additive and multiplicative brightness changes, symmetric and asymmetric
- Contrast changes, symmetric and asymmetric
- Gamma changes, symmetric and asymmetric
- Hue changes

The rest of our photometric augmentations (additive Gaussian noise, fog and motion blurring) are not part of any class and hence are activated and evaluated individually.

While we lose the ability to evaluate the impact of any single data augmentation method that is part of a class, splitting the augmentations up into these two classes at least leaves us the ability to differentiate between the impact of augmentations that do change the flow and those that don't. The reason that we do not include motion blur and fog into our photometric augmentation class is that contrary to the other augmentations, these two haven't been used in previous works. It is therefore deemed important to be able to evaluate their usefulness individually without diluting their impact by combining them into a class.

4.2.2 Fixed Augmentation Order

Another design decision that is made is to fix the order in which augmentations or augmentation classes are added. In any of the three training phases, we always use the following ordering:

- Addition of the geometric augmentation class and evaluating the impact at different intensities
- Addition of the photometric augmentation class and evaluating the impact at different intensities
- Evaluation the impact of different cropping sizes
- Addition of Gaussian noise and evaluating its impact at different intensities
- Addition and evaluation of fog, motion blur and a combination of both

Keeping this order fixed reduces the number of experiments, with the drawback of preventing us from further insights, for example since the photometric augmentations are always added after the geometric augmentations we don't conduct any experiments that only use photometric augmentations without using geometric augmentations and hence cannot judge the impact of activating only photometric augmentations.

4.2.3 Usage of RAFT-S Instead of Full RAFT

To decrease the actual training time during an experiment, most experiments are conducted using the RAFT-S network architecture. While the feature and context encoders remain unchanged between the two smaller RAFT-S and the full RAFT architecture, the RAFT-S network uses a simpler architecture to iteratively update the optical flow. While this leads to a less accurate final network, the number of parameters and hence the training time are significantly reduced: the original RAFT network has around 4.8 million parameters, the smaller RAFT-S network only has around 1 million parameters. In our case, training on RAFT-S is roughly twice as fast as training on the full RAFT model. Figure 4.1 shows a comparison of both architectures.

In our experiments we use RAFT-S to find well performing augmentation combinations during all training phases. Once these are found, at the end of each training phase we train a full RAFT model using these augmentations. This way we can directly compare the performance of our model to that of the original RAFT implementation, which always uses the full RAFT architecture.

4.3 Experimentation Process

This section describes the process to find well-performing augmentation settings. It is used in all three of our training phases. The only difference is the model that is trained during the experiments: In the first phase, every experiment trains a model from scratch. At the end of this phase we will have acquired one model that performs the best.

This model is then always the basis for any experiment in the second phase: for every experiment, this model (which was already trained in the first phase) is copied and then trained with the experiment-specific settings in the second phase. The same holds true for the experiments of our

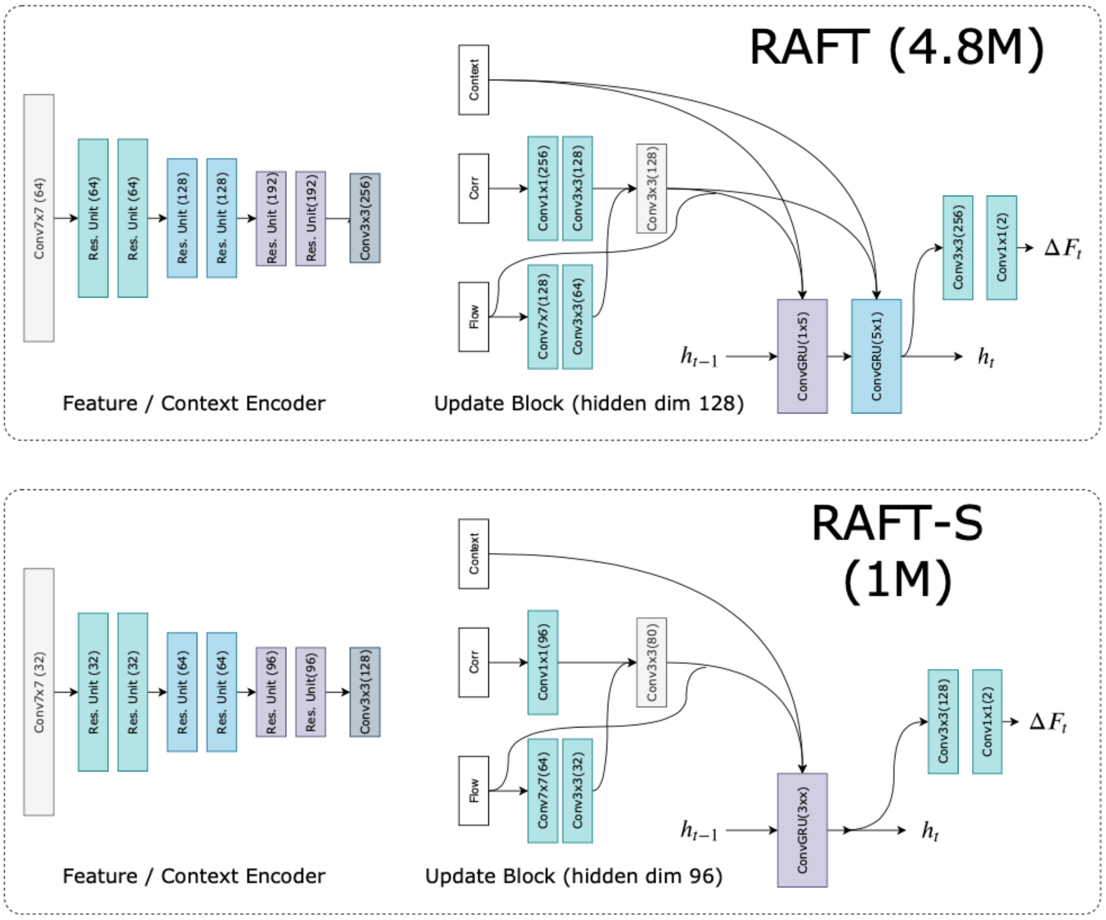


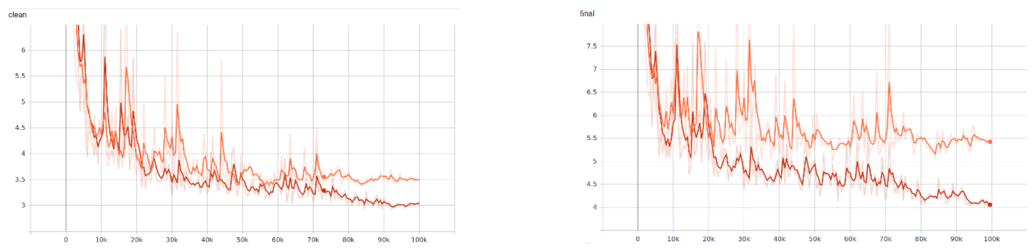
Figure 4.1: Comparison af the RAFT and RAFT-S architectures. Image source: [TD20].

third phase: after completion of the experiments of the second phase we will have acquired a model that performs best after having been trained on the first and second data sets. This is then copied and further trained for each of our experiments of the third stage.

Step 1: Minimizing The Number Of Training Iterations

The evaluation of the impact of an augmentation or the change of the augmentation's intensity or parameters in a training phase requires two steps: firstly, training a network with the augmentation active and set at the desired intensity and secondly evaluating this trained network on the evaluation set. This means that every single adjustment of an augmentation requires training a network. The original implementation of RAFT uses 100.000 training iterations for all three training phases.

To decrease the required training time for each experiment and thus allow a greater number of experiments, we try to decrease the number of iterations used in training as much as possible while still keeping this number high enough so that the difference between using and not using the augmentations stays recognizable. This is done in two steps: First the network is trained without any augmentations for 100.000 iterations. After every 500 iterations of training the network, the



(**a**) Clean pass. (**b**) Final pass.

Figure 4.2: Evaluations of a model trained on FlyingChairs with geometric augmentations (red) and one without any augmentations (orange) on the MPI-Sintel training split every 500 iterations.

network is evaluated on the corresponding evaluation data set. In the second step, we now activate the geometric augmentations and train a new network for 100.000 iterations while evaluating every 500 iterations. This way we acquire the EPE of two different networks for each 500 steps and can judge after which number of iterations the data augmentation starts making an impact on the EPE and especially the point at which the difference between the EPE's of both networks is similar to the difference of EPE's after the full 100.000 iterations. We fix this number as the number of training iterations for all of the following experiments of the current phase.

Figure 4.2 shows the EPEs of two different networks for the first phase (training on FlyingChairs). The orange graphs show the EPEs of the network trained without any augmentations, the red graphs show the EPEs of the network trained with activated geometric augmentations. The left image shows the EPE of both networks on Sintel's clean pass every 500 iterations, the right image shows the EPE on Sintel's final pass every 500 iterations. In this example, we observe that at 30.000 iterations, both in the clean and final pass the network that uses the geometric data augmentations achieves a higher accuracy than the network that doesn't use any augmentations, so consequently the experiments in this training phase are all conducted using 30.000 iterations.

Step 2: Setting the Baseline EPE

The EPE of a network using no augmentations during training, trained for the number of iterations set in the previous step, is our first baseline EPE. From this point on, the performance of every network that is trained is compared against this baseline and if a network manages a better EPE than the current baseline, the baseline is updated as described below.

Step 3: Adding Geometric Augmentations

Now we train a network using all augmentations that are part of the geometric augmentation class. The intensity for each of these augmentations for the first experiment are based on those used in the PWC network [SYLK18]. We call this intensity level **0**.

- Horizontal flipping active, probability 50 %
- Vertical flipping active, probability 50 %

- Symmetric scaling active, scaling factor range: 0.75 to 1.5
- Asymmetric scaling active, scaling factor range: 0.95 to 1.05
- Symmetric rotation active, rotation range: -10 to 10 degrees
- Asymmetric rotation active, rotation range: -1 to 1 degree
- Max translation active, range: 0 to 2 pixels

This resulting network is then evaluated on our evaluation set. Now there are two possible outcomes:

Outcome 1

The first possibility is that the resulting EPE is worse than our baseline EPE (i.e the network that was trained without any augmentations). In this case we train another network with decreased intensities, called **-**:

- Horizontal flipping active, probability 50 %
- Vertical flipping active, probability 50 %
- Symmetric scaling active, scaling factor range: 0.9 to 1.1
- Asymmetric scaling active, scaling factor range: 0.95 to 1.05
- Symmetric rotation active, rotation range: -5 to 5 degrees
- Asymmetric rotation active, rotation range: -1 to 1 degree
- Max translation active, range: 0 to 1 pixels

If this network still performs worse than the network without any augmentations (i.e. doesn't manage to beat the baseline EPE), none of the augmentations of the geometric augmentations class will be activated for the rest of this training phase and the experiments continue with the next augmentation class. If on the other hand this network (with activated geometric augmentations but decreased intensities of these augmentations) performs better than our baseline (i.e its EPE is lower than our baseline EPE), the augmentations stay activated for the rest of the experiments in the current phase and the baseline EPE is updated to the EPE of this network. The experiments then also continue with the next augmentation class.

Outcome 2

The second possibility is that the EPE of this network that uses intesity level **0** is lower than our current EPE baseline. In this case we update our EPE baseline to the EPE achieved by the network and then train another network, now using increased intensities of the augmentations, called **+**. This procedure then continues until a further increase of intensities results in an EPE that is worse than the current EPE baseline. At this point, experiments on geometric augmentations are completed and the data augmentations of the geometric augmentation class stay activated at their best performing intensity for the rest of the experiments during that phase.

Intensity level **+**

- Horizontal flipping active, probability 50 %
- Vertical flipping active, probability 50 %
- Symmetric scaling active, scaling factor range: 0.5 to 1.75
- Asymmetric scaling active, scaling factor range: 0.9 to 1.1
- Symmetric rotation active, rotation range: -15 to 15 degrees
- Asymmetric rotation active, rotation range: -3 to 3 degree
- Max translation active, range: 0 to 2 pixels

Intensity level **++**

- Horizontal flipping active, probability 50 %
- Vertical flipping active, probability 50 %
- Symmetric scaling active, scaling factor range: 0.3 to 2
- Asymmetric scaling active, scaling factor range: 0.8 to 1.2
- Symmetric rotation active, rotation range: -20 to 20 degrees
- Asymmetric rotation active, rotation range: -5 to 5 degree
- Max translation active, range: 0 to 5 pixels

Intensity level **+++**

- Horizontal flipping active, probability 50 %
- Vertical flipping active, probability 50 %
- Symmetric scaling active, scaling factor range: 0.2 to 2.5
- Asymmetric scaling active, scaling factor range: 0.6 to 1.4
- Symmetric rotation active, rotation range: -25 to 25 degrees
- Asymmetric rotation active, rotation range: -8 to 8 degree
- Max translation active, range: 0 to 10 pixels

Step 4: Adding Photometric Augmentations

This step is similar to the previous step, but we now add the data augmentations of our photometric augmentations class. The intensities of the first experiment again are based on those used on the PWC-Net, again named intensity level **0**:

- Symmetric additive brightness shift, range: 0 - 0.15
- Asymmetric additive brightness shift, range: 0 - 0.015
- Symmetric Gamma shift, range: 0.8 - 1.2
- Asymmetric Gamma shift, range: 0.98 - 1.02
- Symmetric multiplicative brightness shift, range: 0.8 - 1.2
- Asymmetric multiplicative brightness shift, range: 0.95 - 1.05
- Symmetric contrast shift, range: 0.8 - 1.2
- Asymmetric contrast shift, range: 0
- Symmetric hue shift, range -0.05 - 0.05

Again, if the EPE of the resulting network is worse then our current baseline EPE, there is another try of using decreased intensities:

Intensity level **-**

- Symmetric additive brightness shift, range: 0 - 0.1
- Asymmetric additive brightness shift, range: 0 - 0.01
- Symmetric Gamma shift, range: 0.9 - 1.1
- Asymmetric Gamma shift, range: .99 - 1.01
- Symmetric multiplicative brightness shift, range: 0.9 - 1.1
- Asymmetric multiplicative brightness shift, range: 0.98 - 1.02
- Symmetric contrast shift, range: 0.9 - 1.1
- Asymmetric contrast shift, range: 0
- Symmetric hue shift, range -0.025 - 0.25

If the EPE of this network is still worse than our baseline EPE, we continue with the next step and no photometric augmentations will be activated during this training phase. If the EPE of this network is better than our baseline EPE, we update our current baseline EPE and the photometric augmentations will stay activated at this low intensity.

Like in the previous step, if the EPE of this first experiment is lower than our current baseline (i.e. the EPE of the network with the best performing geometric augmentations), the baseline EPE is updated and other models with increasing intensities are trained until the baseline EPE no longer improves.

Photometric intensity level **+**

- Symmetric additive brightness shift, range: -0.15 - 0.15
- Asymmetric additive brightness shift, range: -0.015 - 0.015
- Symmetric Gamma shift, range: 0.5 - 1.5
- Asymmetric Gamma shift, range: 0.9 - 1.1
- Symmetric multiplicative brightness shift, range: 0.8 - 1.2
- Asymmetric multiplicative brightness shift, range: 0.95 - 1.05
- Symmetric contrast shift, range: 0.6 - 1.4
- Asymmetric contrast shift, range: 0.95 - 1.05
- Symmetric hue shift, range -0.05 - 0.05

Photometric intensity level **++**

- Symmetric additive brightness shift, range: -0.25 - 0.25
- Asymmetric additive brightness shift, range: -0.025 - 0.025
- Symmetric Gamma shift, range: 0.25 - 1.75
- Asymmetric Gamma shift, range: 0.8 - 1.2
- Symmetric multiplicative brightness shift, range: 0.5 - 1.5
- Asymmetric multiplicative brightness shift, range: 0.9 - 1.1
- Symmetric contrast shift, range: 0.5 - 1.5
- Asymmetric contrast shift, range: 0.8 - 1.2
- Symmetric hue shift, range -0.1 - 0.1

Step 5: Influence of Crop Size

In this step we explore the influence that the crop size has on the performance of the network. This is motivated by Bar-Haim and Wolf [BW20], who found out that using the maximum possible crop size during training increases the accuracy of the network. Their experiments were done on the IRR-PWC network architecture. During this step we decrease the crop size to check whether this holds true for RAFT as well.

Up until this point, our crop size is always close to the actual size of the images of this phase, which means we crop as little as possible. The images are only cropped in a way that both width and height are divisible by 8, which is necessary for RAFT to work.

The first experiment decreases the crop size to around 90 percent of the base values (while maintaining height and width which are divisible by 8). If this network achieves a lower EPE than the current baseline EPE, the baseline EPE is updated to this new EPE and further cropping experiments are done, where the crop size is again reduced by 10 percent of the original dimensions, until the performance of a network is worse than the current baseline. The resulting, best performing crop size is then kept for the rest of the experiments of this training phase.

Step 6: Activation of Additive Gaussian Noise

Now we activate additive Gaussian noise. In the first experiment a network is trained using a range between 0 and 5, which means that for every training sample the standard deviation of the Gaussian noise added to the image pairs of the training sample is uniformly distributed between these values.

If this network performs worse then our baseline EPE, the additive Gaussian noise augmentation is turned off and experiments continue with the next step. If on the other hand this network performs better then our baseline EPE, the baseline EPE is updated and the upper value of the noise range is increased by 5. This procedure is repeated until the performance is worse than the current baseline EPE. The noise range that resulted in the lowest EPE is then kept for the rest of the experiments of this phase.

Step 7: Adding Fog or Motion Blur

In this final step, we evaluate the influence of the fog and motion blur augmentations by comparing the performance of four networks:

1. Activation of random motion blur
2. Activation of nonrandom motion blur
3. Activation of fog
4. Activation of both fog and the better performing type of motion blur

This way we can evaluate the impact of all three of these augmentations individually and also the impact of a combination of fog and motion blur. If the best performing of these 4 networks manages a lower EPE than a network without any of these augmentations, then the fog and motion blur settings of this network are kept.

After this step, the experiments of this training phase are finished. The resulting model is copied and these copies are used as the starting point for every experiment of the next training phase. This means that in training phase 2 (training on FlyingThings3D), we always do experiments with the model that was already trained on FlyingChairs in phase 1, and in training phase 3 we always use the model trained both on FlyingChairs (in phase 1) and FlyingThings3D (in phase 2).

4.3.1 Comparing our Network with the Original RAFT Implementation

After completing each phase, we train two full RAFT models (instead of the small RAFT-S model used during the experiments to find well performing augmentation settings) for the full number of iterations on the phase's training set: one using the augmentations that we determined during the experiments that lead to the lowest EPE and another one using the augmentations of the original RAFT implementation. Even though the original RAFT implementation uses single precision during training, we use mixed precision during training of both networks, which is a little bit faster. The performance of these two models is then compared by evaluating them on the phase's evaluation set.

5 Results and Discussion

In this chapter we present the results of the conducted experiments of all three phases. The full results of the experiments conducted with a small RAFT-S model are included in the appendix. Unless specifed otherwise, we always use the cold start option when calculating the optical flow during evaluation.

5.1 Training Phase 1: FlyingChairs

In this phase we need to use 30,000 iterations to observe a difference in the network's performance when using no augmentation and when using our geometric data augmentations. That's why during this phase all experiments on the small RAFT-S network are conducted using this amount of iterations. Table 5.1 gives an overview of the results of this phase.

Augmentations	Parameters/Intensity Level	baseline EPE after 30k steps	
		clean pass	final pass
None		3.497	5.241
+ Geometric Augmentations	+	3.106	4.273
+ Photometric Augmentations	+	3.07	3.955
+ Crop Size	100 %	3.07	3.955
+ Additive Gaussian noise	[0, 5]	2.983	3.932
+ Fog/Motion Blur	Nonrandom Motion Blur, no fog	2.96	3.825

Table 5.1: Evaluation on the MPI-Sintel training split after training on FlyingChairs, small models.

The addition of the geometric data augmentations improves the accuracy of the network in both the clean and the final pass. The best performance is achieved using the intensity level +, which means using higher intensities than used in the PWC-Net implementation. In the clean pass the addition of this augmentation class improves the accuracy by around 11 percent, in the final pass the EPE is even 18 percent smaller than when using no augmentation. The reason for these large improvements might be that the FlyingChairs data set includes only very limited types of object motions. The strong geometric augmentations change that and allow the network to learn much more varied types of motions.

The photometric augmentation further improves the performance, again using intensity level +, and the activation of additive Gaussian noise with a standard deviation range between 0 and 5 further increases the network's accuracy as well. While the fog augmentation decreases the network's performance and is hence omitted, the nonrandom motion blur augmentation does indeed improve the networks performance on both the clean and final pass, even though the clean pass of the MPI-Sintel training set does not include any motion blur.

Overall, the RAFT networks benefits from using strong augmentation intensities when using the FlyingChairs data set during its first training phase. With the exception of the fog augmentation, all of our implemented augmentation methods are being used here and help to improve the network's performance.

Any reduction of crop sizes during this phase decreases the performance, which means there should be as litte cropping as possible to maximize the network's performance.

Model	EPE after 100k steps			
	Cold Start		Warm Start	
	clean pass	final pass	clean pass	final pass
RAFT original implementation	2.436	4.55	2.316	4.488
Our augmentations	2.249	3.472	2.226	3.392

Table 5.2: Evaluation on the MPI-Sintel training split after training on FlyingChairs, full RAFT models.

We use the resulting data augmentation settings to train a full RAFT model on the FlyingChairs data set. Then we compare the performance of this model against the performance of another model trained on FlyingThings using the data augmentations of RAFT's original implementation when estimating the optical flows of the MPI-Sintel training set. Table 5.2 shows the results: our network beats the original implementation both in the clean and in the final pass, which means that our data augmentations improve the networks generalization capabilities compared to the original augmentations.

5.2 Training Phase 2: FlyingThings3D

We now use the model that was trained in phase 1 on FlyingChairs and achieved the lowest EPE and further train it using the FlyingChairs3D data set. The results of the experiments of the second phase are listed in Table 5.3. Again we have to use 30,000 iterations for each experiment to make sure that any impact of a data augmentation on the performance is visible.

Contrary to the behaviour during the first training phase, the activation of the geometric augmentations deteriorates the accuracy of the network when further training the network on FlyingThings3D in the second phase. This is the case with the default intensity level **0** and even when using the low intensity level **-**, the resulting accuracy is still worse than using no geometric augmentations. Consequently, the geometric augmentations are deactivated during this training phase. One reason for this behaviour might be the fact that the FlyingThings3D data set was already designed to include a lot of variations of optical flow. The objects already do full 3D movements between two frames, so adding another layer of object movements through the use of our geometric data augmentations seems to make it more difficult for the network to learn.

Augmentations	Parameters/Intensity level	baseline EPE after 30k+30k steps	
		clean pass	final pass
None		2.439	3.492
+ Geometric Augmentations	Not activated	2.439	3.492
+ Photometric Augmentations	-	2.491	3.464
+ Crop Size	100 %	2.491	3.464
+ Additive Gaussian noise	Not activated	2.491	3.464
+ Fog/Motion Blur	Nonrandom Motion Blur, no fog	2.594	3.398

Table 5.3: Evaluation on the MPI-Sintel training split after training on FlyingChairs and FlyingThings3D, small model.

The addition of our photometric augmentations only improves things when using the low intensity level. Furthermore, only the results of MPI-Sintel's final pass improve, while the clean pass results deteriorate even when only using these low intensities. Whenever we observe this different behaviour between the clean and final pass, we always use the augmentation setting that improves the final pass, even if this means that the accuracy of the clean pass gets worse. We do this because the final pass is more challenging than the clean pass and our hope is that a model that works as well as possible on this challenging final pass is more powerful than one that works as well as possible on the clean pass.

While in the previous training phase the activation of additive Gaussian noise improved the model's performance, this is no longer the case now. Here the addition decreases the network's accuracy even when using the lowest range of standard deviations (0 to 5) and as a result no additive Gaussian noise is activated during this training phase. The fog augmentation again doesn't improve the accuracy and is also not used, while the addition of nonrandom motion blur does improve the performance at least on the final pass but actually decreases it during the clean pass. Following the strategy described above, we again keep the nonrandom motion blur augmentation activated. As before, any cropping of the images leads to decreased accuracy and hence we again crop the images as little as possible.

In summary, in this phase we see only a very limited use of data augmentations. Even those augmentations that are activated, namely the photometric augmentations, only benefit the network's performance when their intensities are chosen very low.

Model	EPE after 100k+100k steps			
	Cold Start		Warm Start	
	clean pass	final pass	clean pass	final pass
RAFT original implementation	1.504	2.713	1.424	2.689
Our augmentations	1.406	2.625	1.44	2.595

Table 5.4: Evaluation on the MPI-Sintel training split after training on FlyingChairs and FlyingThings3D, full models.

After obtaining the augmentation settings for this second phase, we apply them during training phase 2 of our full RAFT model. Furthermore, the model which uses the data augmentation settings of RAFT's original implementation is now trained on phase 2 as well.

The performance of the resulting models is again compared on the MPI-Sintel training split, as shown in Table 5.4: As expected both networks benefit from adding another training phase and deliver a greater accuracy after training for two phases instead of just one, and again our network manages to beat the original RAFT implementation in both the clean and the final pass when using the cold start.

When using the warm start, our model actually performs worse than when using the cold start when evaluating the clean pass. In fact, the original RAFT implementation beats our network in this instance. This is not the case during the final pass, where our network beats the original implementation when using the warm start. A reason for this behaviour might be the inclusion of the nonrandom motion blur augmentation. As we saw in Table 5.3, its inclusion increases the performance of the small network when evaluating the final pass, but also leads to a worse performance during the clean pass.

5.3 Training Phase 3: MPI-Sintel Mix

In this third training phase we now have to use 50,000 iterations during training the networks. The results are depicted in Table 5.5. As described above, during this training phase we no longer evaluate the network's performance using the full MPI-Sintel training split but instead we use the "eval" subset. This evaluation set seems to be a lot more challenging than the full data set, as a network trained on phase 3 without any augmentations only manages an EPE of 3.21 in the clean pass and 5.836 in the final pass.

In this training phase the activation of our geometric augmentations increases the resulting network's performance in both the clean and final pass. In fact, the best performance is achieved by using the ++ level of intensities. The same cannot be said about the addition of our photometric augmentations. Even at the lowest intensities, their activation worsens the accuracy of the network and consequently no augmentations of this augmentation class are activated.

Contrary to the findings of Bar-Haim and Wolf [BW20], decreasing the crop size in this training phase improves the accuracy of the network, at least on the final pass, albeit only slightly. Every reduction of the dimensions of 10 percent (of the original image dimensions) improves the network's performance until we reach the best performance at 60 percent. This result is surprising given that each pair of pixels in a way acts as a data sample when learning optical flow. This means using a smaller crop size leads to fewer training samples, which should be detrimental to the network's performance.

As was the case in the previous training phase, the activation of additive Gaussian noise also decreases the network's performance and is omitted. Lastly, none of the two motion blur augmentations improve the accuracy, but here for the first time the addition of the fog augmentation benefits the network's performance in the final pass, even though the activation of this augmentation deteriorates the performance during the clean pass. This is to be expected as the clean pass doesn't include any fog effects.

Augmentations	Parameters/Intensity Level	baseline EPE after 30k+30k+50k steps	
		clean pass	final pass
None		3.21	5.836
+ Geometric Augmentations	++	3.057	4.993
+ Photometric Augmentations	Not activated	3.057	4.993
+ Crop Size	60 %	3.057	4.758
+ Additive Gaussian noise	Not activated	3.057	4.758
+ Fog/Motion Blur	Fog, no motion blur	3.146	4.718

Table 5.5: Evaluation on the MPI-Sintel training split subset ”eval” after training on FlyingChairs, FlyingThings3D and the MPI-Sintel mix, small model.

To compare the performance of the full models, we now train our full RAFT network on the MPI-Sintel mix using these augmentation settings. We also train the network that uses RAFT’s original implementation data augmentations during training on this data set mix. The performance of these two networks is now evaluated using the ”eval” subset of the MPI-Sintel training split. As depicted in Table 5.6, the result is no longer clear cut: While our RAFT network beats the original implementation in the final pass, our augmentations lead to an accuracy that is worse than that of the original implementation in the clean pass. A reason for this might be our strategy to always keep augmentations if they improve the performance during the final pass evaluation, even if these augmentations deteriorate the performance during the clean pass evaluation. Both networks improve their performance when the warm start option is used, in both the clean and the final pass.

Model	EPE after 100k+100k+100k steps			
	Cold Start		Warm Start	
	clean pass	final pass	clean pass	final pass
RAFT original implementation	1.657	3.649	1.583	3.246
Our augmentations	2.049	3.169	2.037	3.044

Table 5.6: Evaluation on the MPI-Sintel ”eval” subset after training on FlyingChairs, FlyingThings3D and the MPI-Sintel mix, full models.

5.4 Evaluating the Full Models on the MPI-Sintel Benchmark

The MPI-Sintel benchmark allows the comparison of different models using a special evaluation data set, the MPI-Sintel test split. Similar to the MPI-Sintel training split, this data set contains two version of its image samples, the clean and the final pass. The difference is that the ground truth optical flow of this data set is not openly available. Instead, the samples of this evaluation data set are fed into the model that is to be benchmarked and the resulting optical flow estimations are uploaded to the MPI-Sintel website[1]. There the estimations and the ground truth optical flow are used to calculate the achieved EPE of the model for both the clean and the final pass, which is then presented on the website and can be compared to the performance of other models.

To be able to compare our network to the original RAFT implementation on this benchmark, we now train a network on the same data sets as the benchmarked original implementation, using our data augmentation settings: as before first on FlyingChairs, then on FlyingThings3D, but in the third phase we now use the full MPI-Sintel training split in the mix of data sets instead of the MPI-Sintel training subset "train" that we used in our experiments before. This is now possible because we now evaluate on the MPI-Sintel test split so we no longer have the problem of avoiding training and evaluating on the same data set.

Model	EPE after 100k+100k+100k steps			
	Cold Start		Warm Start	
	clean pass	final pass	clean pass	final pass
RAFT original implementation	1.94	3.18	1.61	2.86
Our augmentations	2.36	3.78	2.33	3.67
No augmentations in 3rd phase	2.17	3.79	1.83	3.42

Table 5.7: MPI-Sintel benchmark results after training on FlyingChairs, FlyingThings3D and the MPI-Sintel training split, full models.

As shown in table 5.7, our model does not manage to beat the original RAFT implementation on the MPI-Sintel benchmark. In both the clean pass and the final pass the accuracy of our network, trained with our augmentation settings, is worse than that of the original implementation. This is the case despite the fact that as we saw in tables 5.2 and 5.4, both a full model trained only on FlyingChairs and a full model trained on FlyingChairs and FlyingThings manage to beat the original implementation that is trained on the same data sets when we use our augmentation settings. There are a couple of issues that might be the reason for this result:

In the earlier experiments, we can optimize our augmentation parameters directly on the target data set, namely the MPI-Sintel training split in phase 1 and 2 and the "eval" subset in phase 3. Since the MPI-Sintel test split doesn't offer publicly available ground truth data, we cannot directly optimize our augmentations and their intensities on this benchmark. This is of course by design to prevent people from overfitting their models to the benchmark, but it means that we have to use another evaluation set to optimize our parameters, in our case the MPI-Sintel training split "eval" subset. It may be the case that that the impact that an augmentation has on the model's performance on this subset is not similar to the impact of this augmentation on the MPI-Sintel benchmark: We choose augmentation settings to maximize the performance in the "eval" subset, and these settings do not necessarily maximize the performance on the MPI-Sintel test split used in the benchmark.

To test this, as a further experiment we train another full network using our data augmentation settings during the first and second training phase but using no data augmentations whatsoever during training of the third phase. Intuitively, this seems to make a lot of sense because when finetuning on the third and final phase, which contains samples that are very similar to the target

[1]http://sintel.is.tue.mpg.de/

evaluation set, it may be beneficial to not augment and therefore change these training samples too much so that the finished model can learn their properties better. Also this approach is motivated by the results of Bar-Haim and Wolf [BW20], wo argue that the amount of data augmentation should decrease in later training stages.

Afterwards we again perform the MPI-Sintel benchmark using this model. As shown in the bottom row of Table 5.7, while this model also doesn't manage to beat the original RAFT implementation, it actually performs better than our original model (which does use our data augmentations during phase 3 training) in both the clean and final passes. This is despite the fact that as we saw above (Table 5.5), using no augmentations in the third phase during our experiments on the small RAFT-S model resulted in worse performance than adding augmentations, albeit when evaluating on the ”eval” subset.

Two different explanations are possible for this behaviour: firstly, it might be possible that the impact of an augmentation differs between the small RAFT-S and the full RAFT model, at least when training on this specific data set. Secondly, the MPI-Sintel training split ”eval” subset may not be suitable when trying to choose augmentation settings that perform well on the MPI-Sintel test split benchmark.

The other conclusion stemming from this additional experiment is that even though using no augmentation in the third phase during training results in a better performing model on the MPI-Sintel benchmark than using our augmentation settings, some amount of data augmentation is still necessary: The original RAFT implementation does use data augmentations during phase 3 training and beats our model that uses no augmentation on the MPI-Sintel benchmark. The reason for this might be that when using no augmentation at all during this phase, the network overfits the data and its generalization capabilities deteriorate.

5.5 KITTI-15 Performance

In all of our previous experiments for choosing augmentations and their intensities, we always use an MPI-Sintel data set to evaluate the resulting models and optimize our settings to achieve the best performance in MPI-Sintel flow estimation: in the first phase and second training phase we use the MPI-Sintel training split and in the third phase a subset of this training split. In this section we take a look at the performance of these models when they are used to estimate the optical flow of a different data set, namely KITTI-15.

Table 5.8 shows the performance of our full RAFT network, trained only on FlyingChairs using our augmentations and the performance of the original RAFT implementation trained on FlyingChairs, when estimating the optical flow of the KITTI-15 data set. The EPE column shows the average endpoint error as before. The F1-all column shows the average percentage of pixels where the endpoint error is both larger than 3 pixels and larger than 5 percent, so the lower the F1-all error is, the better is the performance of the network. As we saw earlier in table 5.2, our optimized network managed to beat the original RAFT implementation when evaluating on the MPI-Sintel training set after only training on FlyingChairs. The result when comparing the performance on KITTI-15 is not as clear cut: the network using our augmentation indeed achieves an EPE that is lower than that of the original RAFT implementation, but the F1-all error of our model is actually higher.

Model	after 100k steps	
	EPE	F1-all
RAFT original implementation	9.733	34.91
Our augmentations	9.027	36.078

Table 5.8: Evaluation on KITTI-15 after training on FlyingChairs, full models.

Training the model further on the FlyingThings3D data set leads to the results as shown in table 5.9. Now the original implementation network manages to beat our network in both the EPE and the F1-all error, even though our network was superior during the MPI-Sintel evaluation. This result shows that while we can tweak our augmentation settings to achieve better results on one target data set, these settings do not necessarily represent the optimum for any other data sets: the optimal data augmentation settings are dependent on the target data set.

Model	after 100k + 100k steps	
	EPE	F1-all
RAFT original implementation	4.836	16.535
Our augmentations	4.869	17.317

Table 5.9: Evaluation on KITTI-15 after training on FlyingChairs and FlyingThings3D, full models.

5.6 Limitations

There exist a number of caveats concerning our experimentation design and consequentially the found augmentation settings that one has to be aware of when classifying the results. In this section we address those.

We always use the model which performs best in one training phase's evaluation as a base that gets trained further in the following phase. It might be possible though that a model that actually performs worse during this evaluation in the earlier training phase is a better base for the next phase and could be trained to a higher performance. Properly testing this would take a lot of experiments.

The number of iterations we use for each experiment of a training phase is chosen by comparing a model with no augmentation and one using our geometric augmentations and it is then kept for any experiment of this training phase. This means we implicitly make the assumption that this number of iterations is sufficient to show the impact of all the other augmentations during this phase as well. To remedy this in later works, if time allows it would be advantageous to compare the necessary number of iterations for all augmentations separately and choose the largest of these numbers for all experiments. A related issue is that the number of iterations that are necessary may also depend on the intensity of these augmentations. Ideally and if enough time is available, experiments should be conducted using the full number of iterations to bypass these problems.

Another issue lies in the augmentation order. We always keep the augmentation intensities of one augmentation that lead to the best performance fixed for the rest of the experiments of the training phase. Then we add another augmentation and again search its best intensity to improve the model even more. But it is possible that using another, less well performing intensity on the earlier augmentation would allow us to find another augmentation intensity for the later augmentation so that the overall resulting model performs better.

Further problems may arise from the grouping of some augmentation into classes: as we increase the intensities of several augmentations of a class at once, in case of deteriorating performance of the model after this increase it might be possible that the intensity increase is actually beneficial for all but one augmentation and the resulting performance is only worse because this one augmentation intensity increase negates all positive effects of the intensity increases of the other augmentations. So ideally and it time allows, individual evaluation of each intensity would be preferable. This may be more viable if the sheer number of data augmentations is simply decreased.

6 Conclusion and Outlook

In this thesis we explored the impact of adding advanced data augmentation to the RAFT network on its performance. To this end we implemented our data augmentations into the RAFT architecture and then followed an experimentation procedure to find the best performing data augmentation combinations and intensities for all training phases, in our case FlyingChairs, FlyingThings3D and a mix of data sets including a subset of the MPI-Sintel training split . This procedure first adds geometric augmentations and optimizes their intensities, then adds and optimizes photometric augmentations, sets the best performing crop size and additive Gaussian noise levels and finally checks the impact of fog and motion blur augmentations. The performance was usually measured on the MPI-Sintel training split, the only exception being the third phase were we only use another subset of this data set.

We found out that the network benefits from relatively strong levels of data augmentation when training on FlyingChairs. This is not the case during the second training phase on FlyingThings3D, where the best results were achieved when using almost no data augmentations. In the third and final training phase, again adding data augmentations improved the network's performance on the MPI-Sintel evaluation subset.

Even though we use the RAFT-S architecture to find optimal settings, in most cases the full models that use these settings indeed perform better than the original RAFT implementation: our model beats the original implementation's performance on the MPI-Sintel training set evaluation after training on FlyingChairs and also after additional training on FlyingThings3D. But after additional training in the third phase on the mix of data sets which includes a subset of the MPI-Sintel training split, our model only manages to beat the original implementation in the final pass of our evaluation set and not in the clean pass.

We then trained another model using our augmentation settings on the data sets that were also used during training of an original RAFT implementation model that was benchmarked on the MPI-Sintel test split: FlyingChairs, FlyingThings3D and a data set mix that now includes the full MPI-Sintel training split in the third phase. Performing this benchmark on our model shows that in both the clean and final pass, our model is unable to beat the original implementation's performance.

We conjectured that the MPI-Sintel subset that we used to find optimal augmentation parameters during the third training phase might not be suitable for this task or that the RAFT-S architecture reacts differently to augmentations than the full RAFT model and hence using this small model to optimize augmentation settings doesn't lead to the best results with the full model. We found out that using no augmentations during the third phase training results in a model that actually performs better in the MPI-Sintel benchmark than the one that uses our augmentation settings. Such a model still doesn't beat the performance of the original RAFT implementation, but this result shows that its more beneficial to use very low augmentation intensities in the final training phase instead of strong intensities.

Finally, we tested our network's performance when estimating the optical flow of KITTI-15 after one and two training phases and found out that the augmentation settings that resulted in a model that offers good performance when evaluating on the MPI-Sintel training split but worse performance when evaluating on KITTI-15. This implies that there is no single optimal augmentation setting that works very good on all different data sets.

Outlook

There are several interesting questions that could be answered by future works. Firstly an open question is how we can attain augmentation settings for the third training phase that allow us to beat the original RAFT implementation in the MPI-Sintel benchmark, since optimizing these settings on the MPI-Sintel subset as we did wasn't successful.

As described above, there are several decisions when constructing the experimentation process that can be made differently. Also the kind of experiments can be expanded: during our thesis we explore the impact of intensity changes of a data augmentation, but another possibility might be simply changing the probability with which the augmentation is applied to a training sample.

Finally, this thesis did not perform any experiments to confirm whether the sequence of training data sets that is used by the original RAFT implementation and consequently by us as well is actually optimal.

As we have seen, there are a lot of cases where advanced data augmentations can improve the performance of RAFT, but optimizing the use of these augmentations through experimentation is currently very time consuming. To circumvent this problem, automating this process or exploring other approaches like using RAFT in an unsupervised manner could be very valuable.

Bibliography

[BW20] A. Bar-Haim, L. Wolf. “ScopeFlow: Dynamic Scene Scoping for Optical Flow”. In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on pp. 38, 44, 47).

[BWSB12] D. J. Butler, J. Wulff, G. B. Stanley, M. J. Black. “A Naturalistic Open Source Movie for Optical Flow Evaluation”. In: *Computer Vision – ECCV 2012*. Springer Berlin Heidelberg, 2012, pp. 611–625 (cit. on p. 11).

[DFI+15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. v.d. Smagt, D. Cremers, T. Brox. “FlowNet: Learning Optical Flow with Convolutional Networks”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2015 (cit. on pp. 9, 11, 13).

[HR19] J. Hur, S. Roth. “Iterative Residual Refinement for Joint Optical Flow and Occlusion Estimation”. In: *CVPR*. 2019 (cit. on p. 9).

[HS81] B. K. Horn, B. G. Schunck. “Determining optical flow”. In: *Artificial Intelligence* 17.1-3 (Aug. 1981), pp. 185–203 (cit. on p. 9).

[IMS+17] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, T. Brox. “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017 (cit. on pp. 9, 13).

[KNH+16] D. Kondermann, R. Nair, K. Honauer, K. Krispin, J. Andrulis, A. Brock, B. Gussefeld, M. Rahimimoghaddam, S. Hofmann, C. Brenner, B. Jahne. “The HCI Benchmark Suite: Stereo and Flow Ground Truth with Uncertainties for Urban Autonomous Driving”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2016 (cit. on p. 12).

[LZH+20] L. Liu, J. Zhang, R. He, Y. Liu, Y. Wang, Y. Tai, D. Luo, C. Wang, J. Li, F. Huang. “Learning by Analogy: Reliable Supervision from Transformations for Unsupervised Optical Flow Estimation”. In: *IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*. 2020 (cit. on p. 9).

[MG15] M. Menze, A. Geiger. “Object scene flow for autonomous vehicles”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015 (cit. on p. 12).

[MIH+16] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, T. Brox. “A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016 (cit. on p. 11).

[RB17] A. Ranjan, M. Black. “Optical Flow Estimation using a Spatial Pyramid Network”. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017*. Piscataway, NJ, USA: IEEE, July 2017 (cit. on p. 9).

[SYLK18] D. Sun, X. Yang, M.-Y. Liu, J. Kautz. “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume”. In: *CVPR*. 2018 (cit. on pp. 9, 13, 34).

[TD20] Z. Teed, J. Deng. “RAFT: Recurrent All-Pairs Field Transforms for Optical Flow”. In: *Computer Vision – ECCV 2020*. Springer International Publishing, 2020, pp. 402–419 (cit. on pp. 8, 10, 11, 33).

[YHD16] J. J. Yu, A. W. Harley, K. G. Derpanis. “Back to Basics: Unsupervised Learning of Optical Flow via Brightness Constancy and Motion Smoothness”. In: *Computer Vision – ECCV 2016 Workshops*. Springer International Publishing, 2016, pp. 3–10 (cit. on p. 9).

[YR19] G. Yang, D. Ramanan. “Volumetric Correspondence Networks for Optical Flow”. In: *NeurIPS*. 2019 (cit. on p. 13).

[ZSD+20] S. Zhao, Y. Sheng, Y. Dong, E. I.-C. Chang, Y. Xu. “MaskFlownet: Asymmetric Feature Matching with Learnable Occlusion Mask”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 30).

All links were last followed on July 11, 2021.

A Full Results of RAFT-S Experiments

This section shows the data augmentation settings and the achieved EPE's of all individual RAFT-S networks that were trained following our experimentation process. Underlined results show that a network achieves a better performance than any previous networks.

A.1 FlyingChairs

Table A.1 lists the results of all experiments during training on the FlyingChairs data set.

Parameters					EPE after 30k steps	
Geometric A.	Photometric A.	Crop Size	Noise	Fog/MB	clean	final
None	None	100%	None	None	3.497	5.241
0	None	100%	None	None	3.324	4.644
+	None	100%	None	None	3.164	4.279
++	None	100%	None	None	3.377	4.544
+	**0**	100%	None	None	3.061	4.055
+	+	100%	None	None	3.07	3.955
+	++	100%	None	None	3.301	3.967
+	+	90%	None	None	3.099	4.077
+	+	100%	5	None	2.983	3.932
+	+	100%	15	None	3.077	3.992
+	+	100%	5	Random MB	3.132	4.033
+	+	100%	5	Nonrandom MB	2.96	3.825
+	+	100%	5	Fog	3.281	3.932
+	+	100%	5	Fog & Nonrandom MB	3.369	3.986

Table A.1: Evaluation on the MPI-Sintel training split after training on FlyingChairs, small model.

A.2 FlyingThings3D

Table A.2 lists the results of all experiments during training on the FlyingThings3D data set.

Parameters					EPE after 30k+30k steps	
Geometric A.	Photometric A.	Crop Size	Noise	Fog/MB	clean	final
None	None	100%	None	None	2.439	<u>3.492</u>
0	None	100%	None	None	2.476	3.668
-	None	100%	None	None	2.631	3.577
None	**0**	100%	None	None	2.574	3.525
None	**-**	100%	None	None	2.491	<u>3.464</u>
None	**-**	90%	None	None	2.474	3.473
None	**-**	100%	5	None	2.452	3.504
None	**-**	100%	None	Random MB	2.609	3.484
None	**-**	100%	None	Nonrandom MB	2.594	<u>3.398</u>
None	**-**	100%	None	Fog	2.674	3.53
None	**-**	100%	None	Nonrandom MB & Fog	2.629	3.474

Table A.2: Evaluation on the MPI-Sintel training split after training on FlyingChairs and FlyingThings3D, small model.

A.3 MPI-Sintel

Table A.3 lists the results of all experiments during training on the mix.

Parameters					EPE after 30k+30k+50k steps	
Geometric A.	Photometric A.	Crop Size	Noise	Fog/MB	clean	final
None	None	100%	None	None	3.21	5.836
0	None	100%	None	None	3.203	5.626
+	None	100%	None	None	3.098	5.235
++	None	100%	None	None	3.057	4.993
+++	None	100%	None	None	3.167	5.253
++	**0**	100%	None	None	3.055	5.162
++	**-**	100%	None	None	3.115	5.256
++	None	90%	None	None	3.037	4.979
++	None	80%	None	None	3.17	4.874
++	None	70%	None	None	3.099	4,874
++	None	60%	None	None	3.057	4.758
++	None	50%	None	None	3.073	4.801
++	None	60%	5	None	3.034	5.186
++	None	60%	None	Random MB	3.119	4.855
++	None	60%	None	Nonrandom MB	2.964	4.8
++	None	60%	None	Fog	3.146	4.718
++	None	60%	None	Fog & Nonrandom MB	3.358	4.73

Table A.3: Evaluation on the MPI-Sintel training split subset ”eval” after training on FlyingChairs, FlyingThings3D and the MPI-Sintel mix, small model.

B Kurzfassung

Wir fügen der RAFT Architektur, einem Deep Learning Netzwerk zur Berechung des optischen Flusses zwischen zwei aufeinanderfolgenden Bildern einer Videosequenz, neue Methoden zur Augmentation der Trainingsdaten hinzu. Da RAFT durch überwachtes Lernen trainiert wird, benötigt es Datensätze, die nicht nur Bildersequenzen sondern auch Informationen über den optischen Fluss zwischen diesen Bildersequenzen enthalten. Da sich diese Informationen nicht automatisiert aus beliebigen Bildsequenzen generieren lassen, werdem zum Training stattdessen synthetisch generierte Datensätze verwendet. Ein Nachteil dieser Datensätze ist ihre relativ geringe Größe und die dadurch beschränkte Vielfalt an Optischen Flüssen, die das Netzwerk während des Trainings erhält. Augmentationsmethoden erlauben es, diese Daten vor dem Einlesen durch das Netzwerk zu modifizieren. Dabei können diese Modifikationen auf Pixelebene erfolgen, aber auch die Geometrie der Bilder und damit den optischen Fluss zwischen den Bildern verändern. Dies erhöht die Vielfalt an Mustern, die das Netzwerk während des Lernens kennenlernt. Wir experimentieren in jeder Trainingsphase, welche Kombination von Augmentationen in welcher Stärke die Leistung eines so trainierten Modells auf dem MPI-Sintel Trainingssatz erhöht. Daneben vergleichen wir die so erzielte Leistung mit jener, die ein ursprüngliche RAFT Modell erzielt. Dabei stellen wir fest, dass die Art und Stärke der Augmentationen, die die Leistungsfähigkeit des Modells verbessern, von der Trainingsphase abhängt. Unsere Augmentationsmethoden erhöhen die Leistungsfähigkeit bei der Berechnung vom optischen Fluss des MPI-Sintel Datensatzes sowohl nach der ersten als auch nach den ersten beiden Trainingsphasen (durchgeführt auf FlyingChairs und FlyingThings3D) und schlagen dort das ursprüngliche RAFT Modell. Dieser Vorteil bleibt jedoch nicht bestehen, wenn die Modelle den optischen Fluss von KITTI-15 berechnen sollen. Die Ergebnisse nach der dritten Trainingsphase auf MPI-Sintel zeigen uns, dass das Hinzufügen weiterer Augmentationen nicht zwangsläufig die Leistungsfähigkeit eines Modells verbessert: das Netzwerk schafft es hier nicht, das ursprüngliche RAFT Modell im MPI-Sintel Benchmark zu schlagen.

Declaration