Министерство образования и науки Российской Федерации
Московский физико-технический институт (государственный университет)
Факультет управления и прикладной математики
Вычислительный центр им. А. А. Дородницына РАН
Кафедра "Интеллектуальные системы"

03.04.01 Прикладные математика и физика

Сопоставление изображений при условии значительных смещений и изменений среды

Автор: .....
Швец Михаил Юрьевич

Научный руководитель: .....
Лемпицкий Виктор Сергеевич
Профессор, к. ф.-м. н.

Научный руководитель: .....
Lorenzo Torresani
Associate Professor, PhD

Москва

Июнь 2017

Аннотация

Точная оценка положения платформы в пространстве является трудной задачей автономного вождения. Несмотря на то, что на борту установлены датчики GPS, во многих случаях они не обеспечивают точную локализацию. Тем не менее, приближенное положение может быть использовано для составления запроса к базе данных и получения изображения, отображающего ту же сцену, которая в текущий момент наблюдается с камеры, установленной на платформе. Таким образом, формируется пара изображений, которые, вообще говоря, могут иметь существенные различия, несмотря на то, что содержат одну и ту же сцену. Во-первых, смещение камеры на несколько метров приводит к большим смещениям объектов и фона в изображениях, что приводит к невозможности использования методов локального поиска. Во-вторых, два изображения могут происходить из разных точек времени, внося не только изменения освещенности, но и значительные изменения окружающей среды, такие как снег на тротуарах, здания, построенные или разрушенные на втором снимке по сравнению с первым, движущиеся объекты, присутствующие только на одном из изображений. Мы формируем такие пары изображений, формулируем и предлагаем методы решения нескольких задач сопоставления изображений. В работе представлены методы компьютерного зрения, основанные на глубоком обучении, позволяющие сопоставлять пары изображений и уточнять положение автономной платформы.

Основная часть этой работы посвящена проблеме оценки поточечных соответствий между двумя изображениями. Мы предлагаем несколько архитектур нейронных сетей. Признаковые представления, которые извлекаются из обоих изображений, включают в себя как глобальный контекст, так и локальную информацию, необходимую для предсказания целевых величин. Мы изучаем эффект деформации признаковых представлений, в отличие от классической деформации исходных изображений и показываем преимущества такого подхода. Большие синтетические коллекции используются для обучения моделей. Дальнейшая тонкая настройка выполняется для сохранения этих знаний и адаптации к изображениям из реального мира. В работе получены результаты на парах изображений из одного проезда (при одинаковых погодных условиях и условиях освещения), а также на парах из разных проездов (при существенно различных погодных условиях, например, разных временах года). По аналогии с задачей выделения объектов на изображении, мы дискретизируем пространство векторов соответствий, предсказывая метку дискретного множества классов совместно с уточнением предсказания внутри выбранного класса. Кроме того, предлагается модель для прогнозирования вида сцены при различных условиях, кодирующая эти условия вектором небольшой размерности. Полученное векторное пространство используется для удаления сезонной составляющей. Основываясь на этих исследованиях, мы показываем, что решение промежуточных задач позволяет предсказывать положение платформы точнее, чем в случае непосредственного прогнозирования. Мы оцениваем геометрическое смещение в сценариях с двумя и тремя степенями свободы. Мы демонстрируем применимость разработанных моделей и оцениваем их эффективность на хорошо известных базах данных оптического потока, дополнительно расширяя результаты до сложных межсезонных данных.

Оглавление

Список иллюстраций

Список таблиц

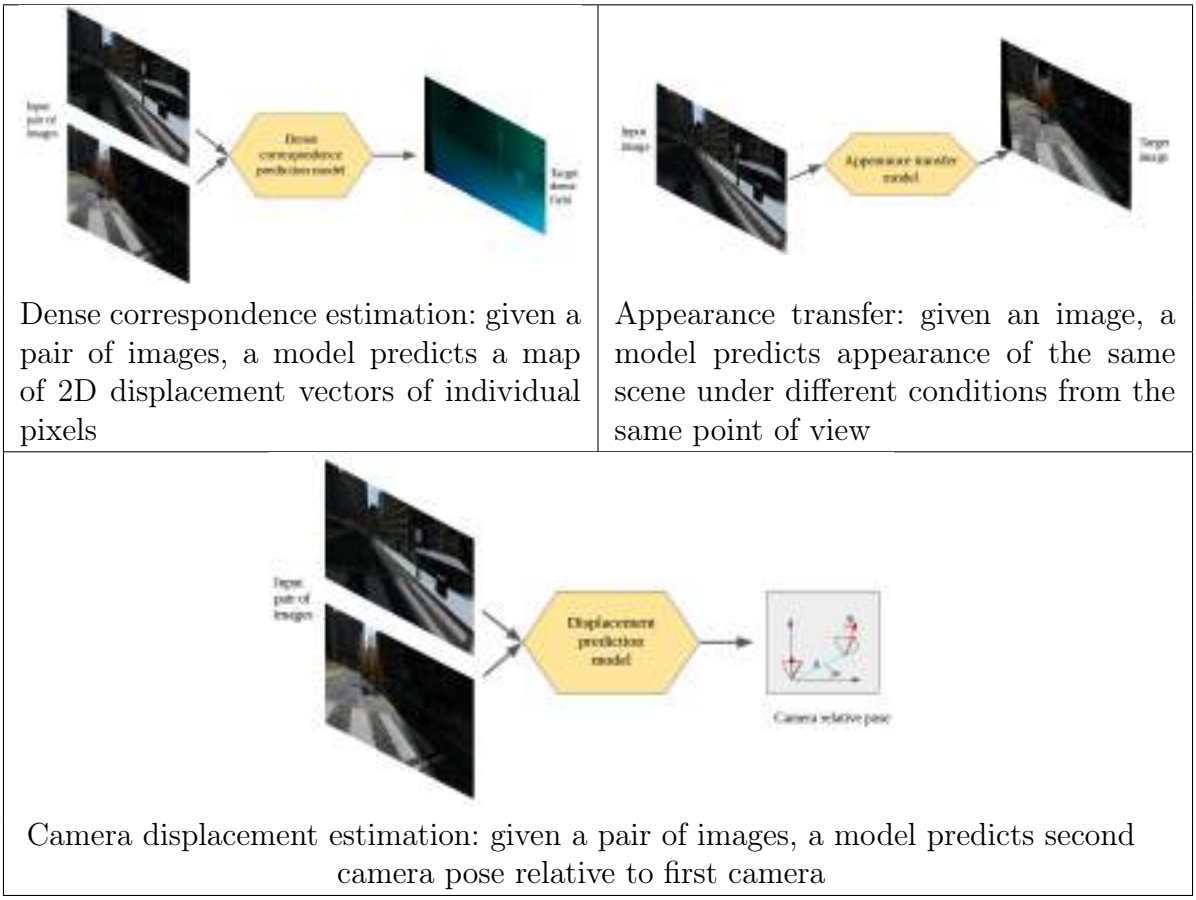
Глава 1

Введение

Vision-based image matching, place recognition and localization are becoming increasingly viable components of navigation systems for autonomous robots and driving scenarios. However, attaining robustness to large objects displacements and variations in appearances such as weather, season, time of day and camera viewpoint remains a major challenge. First of all, pixels that describe the same object may be in distant parts of an image plane for two cameras that capture the same scene from different positions, which would make methods that analyze local neighbourhoods futile. Also, on a pixel-wise level objects may look drastically different: pavement would be gray in summer but white from snow in winter; a wall would be red in light but brown in shadow. On the other hand, localization using monocular vision and GPS positioning is a prospective task, as required sensors are cheap. We stress that GPS measurements alone are not sufficient because of measurement coarseness, especially in urban settings with high buildings and massive concrete structures, quarries, etc.

In this thesis we present a range of methods, making a step towards cross-appearance image matching, relative motion understanding and localization refinement for an autonomous vehicle with visual information obtained from a single camera (see examples of several problems being solved in Figure 1-1). We assume that the vehicle follows only those routes, which have been traversed before, and that a database of images with precise coordinates is created and stored. This database allows to retrieve an image and solve a problem of finding camera pose relative to this image afterwards. For example, knowing GPS coordinates one can

retrieve nearest image from the database and estimate the geometric displacement between two single-camera images: the one currently observed by the platform and the one from the database.



Dense correspondence estimation: given a pair of images, a model predicts a map of 2D displacement vectors of individual pixels

Appearance transfer: given an image, a model predicts appearance of the same scene under different conditions from the same point of view

Camera displacement estimation: given a pair of images, a model predicts second camera pose relative to first camera

Рис. 1-1: Examples of problems that are solved in our work

The problem of finding relative camera pose between image frames is traditionally solved through sparse correspondence estimation between sets of interest points. Usually, image pixel is assumed to be an interest point if it represents edge, corner or other local singularity. Manually designed vector of features (usually based on analyzing image gradients in a local neighbourhood) is extracted to describe each interest point. Then, matching algorithm is developed to align interest points sets between images and finally incorrect matches are removed after checking additional constraints. Executing this approach for cross-appearance image pairs yields matching results that are usually insufficient for localization refinement. Same approach is applied when reconstructing structure from motion and performing alignment

between 3D point clouds that gives more precise results. Unfortunately, it is too time-consuming, especially for self-driving and robotics scenarios.

Single-shot estimation is an important task because of several reasons. Firstly, real-time performance is strictly required for driving scenarios, where huge amounts of information are processed in every moment of time. Secondly, single-shot target prediction gives strong priors, which can be further incorporated into systems that account for temporal information. For example, it is possible to perform Bayesian inference on the graph of platform positions given single-shot estimations. Finally, it is important for making a cold start, for instance when the platform is lost or system reboot takes place. In out work we propose several deep learning based single-shot prediction models.

Deep learning techniques allow to directly analyze raw images, circumventing steps of interest points detection and hand-crafting features for these points. Deep convolutional neural networks have recently been proven to succeed in various classification and regression tasks. Posing localization task in form of a regression problem is a straightforward approach to try. However, 3D motion and rules of projective geometry are challenging to learn in an end-to-end fashion without additional supervision. Revisiting traditional methods of localization, which are described above, we aim to predict correspondences between individual pixels to further enable relative pose prediction. We go from sparse estimation to dense prediction of target displacement vector in each pixel of the image.

Dense target prediction is an essence of fully convolutional neural networks. These models are capable of incorporating global context, predicting target values in each pixel. Successfully used for single-image prediction tasks (semantic segmentation, edges estimation, etc.), fully convolutional models are now explored in application to problems that include image pairs and sequences analysis (optical flow, odometry, etc.).

We adapt fully convolutional models for cross-appearance correspondence estimation and appearance transfer, providing pairs of images as an input and elaborating on how to extract features, fuse information from two images, preserve fine-grained information during global context prediction and deal with ambiguities. We pose traditional notion of image warping in context of convolutional neural networks to deform tensor representations of images, producing more precise estimates. Upon pixel-wise prediction models we build

another model to estimate relative geometrical displacements between cameras and show localization refinement.

Computational experiments are carried out firstly on synthetic driving data, after which the obtained knowledge is transferred to real-world data for driving scenario optical flow prediction and for cross-appearance robot localization.

The structure of the rest of this thesis goes as follows:

- Chapter 2 provides an overview of related works in the field. We describe recent progress of deep learning methods and links to geometrical and matching problems. We take a look at traditional optical flow pipelines and methods of camera pose estimation. Appearance transfer problems are also discussed. Chapter 2 ends with publicly available datasets analysis.

- Chapter 3 gives and overview of existing methods and describes the novelty of our work. Firstly, we describe neural networks concepts and projective geometry notions. Then, correspondence prediction problem is stated and a pipeline is drawn on high level. The following sections introduce details, improvements and changes to this general pipeline. Then, we show how to restate the problem to classification with further refinement step. We address problems of appearance prediction and show how it can be used for improving quality of correspondence prediction model. Finally, geometrical displacement estimation is build upon the combination of these methods.

- Computational experiments are described in Chapter 4. The evaluation is performed on synthetic and real datasets. We provide quantitative measures of endpoint average errors for predicted correspondence fields, Euclidean norm errors in 3D space for displacement problems and in the space of RGB for appearance transfer problems. Additionally we show the qualitative results and provide useful insights.

- Chapter 5 summarizes the work and discusses potentials for future work.

Глава 2

Обзор области

Challenges for outdoor image-based camera motion understanding include large displacements of objects [9, 49, 32] and large environmental changes, including illumination, weather, seasons, time of day [5]. Appearance of the scene under these assumptions may differ drastically. Large displacements and large environmental changes are usually addressed separately: optical flow and odometry papers assume constant environmental conditions, while appearance transfer papers [30, 22] assume fixed camera position. The works addressing both challenges simultaneously [32] provide coarse matching accuracy, which is insufficient for localization refinement.

2.1 Localization

Appearance-based localization across seasons has been considered in [6] from metric point of view with use of stereo imaging. Basing on stereo pair streams authors in [12] propose to look at "experiences"about places to construct a "plastic map"to deal with environmental changes instead of keeping conventional map. Nevertheless, these approaches only deal with discrete grid, providing limited localization accuracy and requiring a search over environment map. In this thesis we propose single-shot methods for image matching and localization from a pair of images to deliver localization refinement. Cross-appearance single-shot correspondence estimation methods were developed to match different instances of one object [32, 51]. Although qualitatively impressive alignment is achieved in these works, average endpoint

errors are much higher than required in driving scenarios. We aim to reduce the errors to make consequent precise camera localization feasible.

Typically localization problem is being addressed with simultaneous localization and mapping (SLAM) methods, in particular the monocular ones [38]. Those mainly include matching features (usually hand-crafted) of some key points [50]. Traditional pipeline for key points correspondence estimation has several steps and establishes only sparse matches, as it is done in [34]. Firstly, key points are extracted from each image. These are locally-special points, for example those having sharp image gradient change. Secondly, these points are encoded into vectors, which are constructed based on local neighbourhood analysis. Finally, matching scheme is introduced between two obtained sparse sets to establish correspondences and incorrect matches are eliminated based on local geometrical constraints. With respect to epipolar constraints, essential and fundamental matrices are robustly estimated as in [50] to gain knowledge about camera motion. Another matching approach is proposed in [49], linking deep learning methods with traditional patch-based sparse correspondence methods.

2.2 Neural networks

Recent progress of deep neural networks [28, 47] enables us to move away from typical SLAM pipeline, as modern architectures are capable of learning internal features representation without need to choose any markers and generate hand-crafted features. Deep neural networks have already been trained on localization tasks like loop closure [15], place recognition [46] etc. Convolutional features for place recognition have also been examined in [45].

Convolutional neural network approach was used to deal with pairs of images in [1] by training a model in an end-to-end fashion to predict 6DOF relative camera displacement. Here the problem was posed as a classification task with coarse discretization of the space into bins. Optical flow regression is handled in [14], which inspires our correspondence prediction methods. We go beyond this approach, predicting correspondences for driving scenarios between seasons of the year.

Cross-seasonal matching is usually addressed in landmark recognition [11] and image retrieval [3], which provides coarse localization, or in structure from motion [7], where 3D

structure is reconstructed and kept in memory, being slow and requiring a lot of computational resources. Structure from motion approach is leveraged for global image-based localizaition in [37, 31]. These methods are also usually based on traditional pipeline of extracting special feature descriptors and matching them with point cloud. Unlike [2], in our work we do not assume landmarks to be explicitly specified, but want to build the internal representation.

Impressive localization results were achieved in [26], where the authors consider a very challenging task of directly regressing six degree of freedom (6dof) camera position from a single image with a CNN. In [25] Bayesian extension of that system is provided.

Probabilistic framework for navigation tasks was also introduced in [13]. Direct regression approach was also tried in [43], where RGBD input was required and regression forests were used for indoor scenes. Same authors have published a sequence of other papers on the topic [48, 17, 19], using RGBD input and following machine learning methods: random forests, RANSAC, randomized ferns. Basing on those results another group has built a system to estimate a pose of an object in RGBD image using CNN to obtain value of energy [8]. Monocular systems have also been developed, but mainly for indoor environment and with dependence on markers [29].

2.3 Predicting appearance

Appearance transfer is performed in [30] through local linear transforms for each pixel, where transforms dictionary is learned in a supervised fashion. Separate dictionary is learned for each season in [39] basing on superpixels.

Our idea to combine features extracted from an image with a low-dimensional vector resembles similar idea in [52]. In this work, the appearance of the scene is predicted from a point of view different from the reference camera position. The conditions are assumed to be exactly the same, which leads to leveraging natural idea of image warping by predicting target flow field. On the contrary, we predict the appearance of the scene from the same point of view under different conditions. Moreover, in our setting these small-dimensional vectors corresponding to target image are unknown, so we jointly predict those together with the image appearance. No supervisory signal is used during training except raw image pairs.

2.4 Datasets

Optical flow estimation is a traditional research field with several popular publicly available datasets. In this thesis we use KITTI 2012 [16] and KITTI 2015 [36] datasets, which provide around two hundred training pairs each together with ground truth optical flow field. FlyingChairs dataset [14] consists of 22872 synthetic images of chairs rendered from 3D models and places on randomly selected background from real photos.

To our best knowledge, there are no established benchmarks for cross-seasonal outdoor dense correspondence prediction that would provide image pairs with reliable ground truth for correspondence fields and relative poses. In our work, we heavily use SYNTHIA [42], a synthetic driving scenario dataset, which provides rendered video sequences at 5 frames per second frequency. These sequences feature a variety of illumination and environmental changes. There are a total of 7 sequences, each coming from a run through a certain environment (highway, megapolis, small European town). Within each sequence several runs are rendered under different environmental conditions (fall, winter, summer, rain, dawn, night, sunset etc.). The virtual car has 8 cameras, which provide forward, backward, left and right views from two points. One can think of that as of stereo omnidirectional cameras, for which 8 undistorted views are already provided for simplicity. Together with the images, rich ground truth is provided: dense depth map (range from 1.5 to 50 meters) for each image, camera pose and intrinsic parameters. Depth information together with camera pose allows to establish dense correspondences for a pair of images that share field of view. Notice that is it possible in cases when a pair comes either from the same run or from a different run, because ground truth poses are provided in global coordinates, which are consistent across all runs and all sequences and allow to re-project pixels from one image onto the plane of another image.

The required ground truth information is easily extracted from computer simulators and game engines, but it is much harder in real world. As discussed above, establishing correspondences in outdoor scenes directly requires either having expensive precise sensors or immense human labour. Depth information can be obtained from LIDAR sensors. LIDAR data is usually present in driving datasets (KITTI [16, 36], The Oxford RobotCar Dataset

[35], New College Dataset [44]), but accurate projection also requires reliable camera pose information. We have already discussed that GPS sensors are not reliable enough to provide target poses. Inertial measurement units (IMUs) are used to refine relative pose of a pair from one run, given that the car has travelled small distance (usually less than 1km), but these sensors are useless when establishing relative pose across different runs. In Michigan North Campus Long-Term [10] dataset (referred to as NCLT further) large SLAM solution is pre-processed, having matching constraints from LIDAR scans across all runs and using high-accuracy RTK GPS. The dataset provides raw images from omni-directional camera, undistortion maps, absolute body frame pose, relative transformations between body frame and all sensors, LIDAR hits (which we use to construct sparse depth map).

Huge database of outdoor scenes from web-cameras (more than a billion samples and growing) was made available in [23], but the images are usually poor and not aligned, which makes it impossible to use it directly for training in our settings. Authors of [30] processed images from 101 of these web-cameras, carefully aligning all images from each camera and ensuring diversity of the images. This set is released publicly as the Transient Attributes Database, containing ground truth information about 40 attributes, grouped under 5 categories ('lighting', 'weather', 'seasons', 'subjective impression' and 'additional attributes'). We show the performance of our appearance prediction model on this dataset.

Глава 3

Методы

3.1 Notations

Here we introduce notations which are used throughout the rest of the work.

$\mathcal{X} = \{0, \ldots, 255\}^{H \times W \times 3}$ denotes the space of RGB images. If not specified explicitly, $X, Y \in \mathcal{X}$ denote images. H, W are image height and width. $\mathcal{F}$ is a space of three-dimensional tensors ($\mathcal{X} \subset \mathcal{F}$). We refer to first two dimensions that enumerate pixels, as *spatial dimensions* and last dimension that enumerates channels, as *features dimension* or *channels dimension*. D or d is a number of channels. $F \in \mathcal{F}$ denotes vector correspondence field of size $H \times W \times 2$, while M denotes matchability matrix of size $H \times W$ (definitions are provided in 3.4). W refers to four-dimensional tensor of convolutional parameters of size $K \times K \times D_{in} \times D_{out}$ (see 3.2.2). Here K is kernel size, D_{in} and D_{out} is the number of dimensions in layer's input and output.

Pixels (image plane integer coordinates) are indexed with i, j, while real-valued coordinates in camera local reference system are indexed with x, y, z. In section 3.3, where geometrical considerations are outlined, X, Y, Z, D, d are also used to denote coordinates (euclidean and homogeneous).

We use upper and lower indices in notations to index a tensor (e.g. $F^1_{i,j}$ denotes the value of correspondence field in pixel (i, j) in channel 1, which is x-displacement of this pixel) or simply for convenience reasons (e.g. D_{out} denotes number of channels in a layer's output tensor).

3.2 Neural Networks

In our work target functions are modeled with neural networks. This section describes main notions of neural networks, such as convolutional layers, receptive fields, skip-connections, etc.

3.2.1 Deep feed-forward networks

Dense feed-forward neural network is initially defined to be superposition of linear functions and nonlinearities. An instance that applies one linear function and one nonlinearity is called *layer*. Last layer is called *output layer*, while other layers are called *hidden layers*. For example, neural network with one hidden layer can be written down as a superposition

$$f(X) = \tanh(W_2 \tanh(W_1 X + b_1) + b_2). \tag{3.1}$$

In this formula X is an input vector, W_1, W_2 (matrices), b_1, b_2 (vectors) are parameters of a linear function, while tanh is a nonlinearity function. For simplicity, additive constants b are usually absent in notations after one extra feature is introduces to X which always equals 1. Nonlinearities are called *activation functions*. Notation (3.1) allows constructing a neural network with arbitrary number of layers and arbitrary activation function. Neural network is called deep when having many layers. For convenience reasons, it is possible to represent input vector as a tensor, shaping all parameters accordingly. For example, it is extremely convenient in computer vision, where an image has natural spatial dimensions (width and height) and RGB features dimension ($X \in \mathcal{X}$).

3.2.2 Convolutions

Description Deep dense feed-forward networks have excessively many parameters and allow only small input sizes because of that. Moreover, input sizes are set to be fixed, because the number of neurons in fixed. In order to overcome these limitations and to make neural networks more stable, convolutional neural networks were introduced in 1D (e.g. speech), 2D (e.g. plain images) and 3D (e.g. spatial modeling) signal processing tasks.

Each layer of a convolutional neural network for image processing usually deals with four-dimensional input and output tensors with first dimension enumerating samples in a batch, next two dimensions enumerating pixels and their spatial representations, and last dimension enumerating feature channels. We are going to describe all methods on three-dimensional tensors because samples in a batch are independent and four-dimensional representation is straightforward and only required for parallel computations.

Convolution is an operation $\circ$ over two tensors: four-dimensional parameters W and three-dimensional image representation $X \in \mathcal{X}$.

$$Y = W \circ X, \quad Y_{ij}^{d} = \sum_{u=1}^{K} \sum_{v=1}^{K} \sum_{d'=1}^{D_{in}} W_{uv}^{d'd} X_{i+u-\lceil \frac{K}{2} \rceil, j+v-\lceil \frac{K}{2} \rceil}^{d'}. \tag{3.2}$$

Here W is a tensor of shape $K \times K \times D_{in} \times D_{out}$, which is limited to be squared in spatial dimensions. Our notation follows this limitation because all kernels used in out experiments are squared, but rectangular kernels may also be used in general. Resulting tensor $Y \in \mathcal{X}$ has D_{out} feature channels.

This operation can be viewed as a traditional filtering operation applied multiple times to every channel of image X. *Strides* can be introduced to perform a step of size $s > 1$ when convolving filter with an image patch. This leads to reducing the spatial size of output s times in each dimension.

3.2.3 Fully convolutional architectures

In image processing neural networks have initially become successful in classification and regression tasks, which imply predicting one or only few target variables. Such networks have a number of convolutional layers that extract features from an image, followed by several dense layers, which fuse information from the extracted features for target prediction. Interestingly, dense layers in these networks can also be represented as convolutional layers. In order to do that one should specify a convolutional layer corresponding to first dense layer to have a kernel of same size as input spatial size, yielding output size to be 1×1. All other dense layers are set to be convolutional layers with 1×1 kernels. We call a representation of a neural network with only convolutional layers *fully convolutional*. In contrast to dense neural

networks, fully convolutional neural networks appeared to cope with tasks of estimating large number of variables (for example, per-pixel labeling). In this thesis we construct fully convolutional models for dense image matching (individual pixels displacement estimation, appearance prediction etc.).

Receptive field and its influence on flow estimation Convolutional operation (3.2) makes each neuron in a network locally connected to a $K \times K$ region of a layer's input. For feed-forward architecture *receptive field* is defined as region on initial image that influences the output of particular neuron in the network. In other words, receptive field is a set of pixels that are path-connected to the chosen neuron. For example, assume a neural network with three layers (as shown in Figure 3-1): first convolution with filter size 5, second one with filter size 2 and stride 2, and the last one with filter size 3. The receptive field of the first layer is 5 (just the same as size of the filter). Second layer has receptive field equal to 6, as each neuron in second layer aggregated information from 2 neighbouring cells with receptive size 5. Interestingly, the receptive field of the third layer appears to be equal to 10, in spite of the fact that each neuron aggregated 3 cells with receptive field of 6. The trick is in the fact that centers of pre-images of these three cells had a stride of 2 between them in X.

Eventually, when a kernel aggregates K input cells, receptive field is increased by accumulated stride multiplied by $K - 1$. Indeed, on layer l we aggregate K_l pixels from initial image X which have accumulated stride $s_{accum} = \prod_{l'=1}^{l-1} s_{l'}$ between them. Then the area of $s_{accum}\,(K_l - 1)+ 1$ is aggregated. But the receptive field on layer $l - 1$ was already q_{l-1}, which contributes to total receptive:

$$q_l = s_{accum}\,(K_l - 1) + 1 + q_{l-1} - 1.$$

Total receptive filed size (assuming $\prod \varnothing = 1$) is then

$$q = \sum_{l=1}^{L} (K_l - 1) \prod_{l'=1}^{l-1} s_{l'} \tag{3.3}$$

As we are going to discuss below, our architecture extracts features from a pair of images and fuses these features at some point. It is important to understand the receptive field of each cell of extracted feature tensor in order to ensure that is it greater than largest

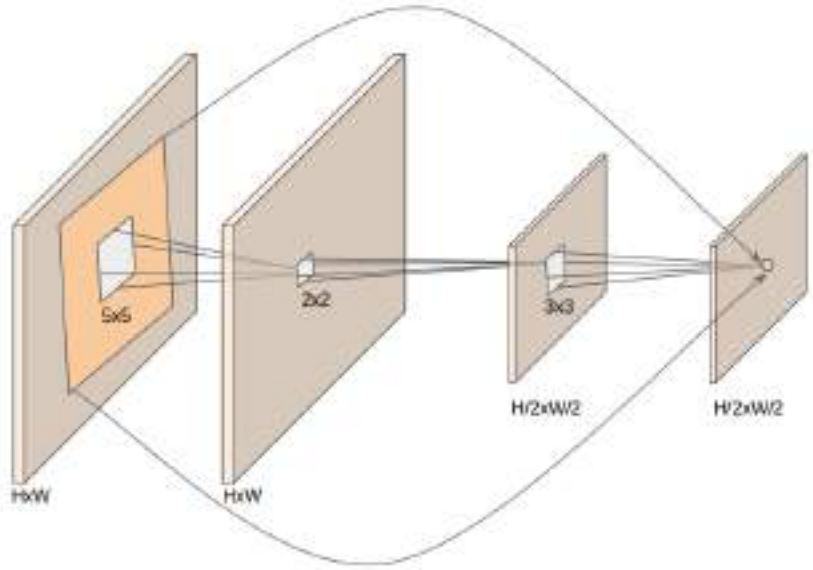


Рис. 3-1: Receptive field of CNN. Grey squares indicate convolutional kernel size, orange square indicates receptive field size of an output neuron in last layer. This output neuron aggregates information from large portion of initial input.

possible displacement between images. Otherwise it would be impossible to incorporate knowledge about large correspondences into the model. One should also keep in mind that the contribution of individual cells is decreasing from spatial position of the reference output cell to edges, although distant cells are still included into receptive field. So deep neural network is required to keep global context in each cell of extracted tensor representations.

3.2.4 Skip-connections

One drawback of deep neural networks was noticed to be in forgetting fine-grained information, which is extracted at early layers. For example, in order to predict appearance of the scene under different conditions (weather, illumination etc.), it is essential to both inject those conditions into the image and preserve details and edges of objects present in the scene. However, after a number of convolutional layers information about small details gets blurred. To overcome this issue, reusing outputs of early layers as part of inputs to deep hidden layers was proposed. Usually, a tensor from early layer is stacked to previous layer output tensor in features dimension, assuming equal spatial size.

3.3 Projective geometry

Projective geometry concepts are important when dealing with image pairs capturing the same scene. Projective transformations do not preserve shape, angles, length ratios. The one thing which is preserved is straightness of lines. It turns out, that efficient representations exists allowing to express all the required transformation in a matrix form. Only in this section we use X, Y, Z to denote 3D coordinates. Also, $\mathbf{x}$ and $\mathbf{X}$ are used to denote 2D and 3D vectors. These notations also denote the same points in homogeneous coordinates. Whether we use homogeneous coordinates or not is clear from context.

3.3.1 Homogeneous coordinates

A point is represented with its two-dimensional coordinates $(x, y) \in \mathbb{R}^2$ on the plane or with three-dimensional coordinates $(X, Y, Z) \in \mathbb{R}^3$ the Euclidean space. In two dimensions, a line is represented with three coordinates (a, b, c) and the constraint of a point line on this line is written as $ax + by + c = 0$. This leads to a three-dimensional representation of a point $(x, y, 1)$. Notice that in such a setting, $\forall k \neq 0$ vector (kx, ky, k) represents the same 2D point on the plane, because for any line the constraint holds and each point is constrained with two intersecting lines. The same reasoning can be applied to 3D case when considering intersecting planes $aX + bY + cZ + d = 0$.

Thus, *homogeneous coordinates* for a 2D point (x, y) are defined to be (kx, ky, k) for any value $k \in \mathbb{R}$. $k = 0$ represents point at infinity, while $k \neq 0$ correspond to finite points. *Homogeneous coordinates* for 3D point (X, Y, Z) are defined to be (kX, kY, kZ, k) for any $k \in \mathbb{R}$. We will write $(k_1X, k_1Y, k_1Z, k_1) \sim (k_2X, k_2Y, k_2Z, k_2)$ to denote equivalence of two points.

Points are mapped into image planes with projective transformations. Necessary and sufficient condition of a 2D mapping h to be a projective transformation is existence of a non-singular 3×3 matrix H such that $h(\mathbf{x}) = H\mathbf{x}$ for any point $\mathbf{x}$ represented in homogeneous coordinates. This assertion is a theorem, which states that projective transformations are linear transformation in space of homogeneous representations. See proof of this theorem in [20]. The same is true for 3D space for 4×4 matrix H. This allows for efficient representation

of all transformations in a matrix form.

3.3.2 Camera model

Now let's consider how image is formed, when camera captures a 3D picture (see Figure 3-2). To understand that we need a camera projection model. We assume no distortions and take standard *pinhole camera* model. In camera local coordinates plane $Z = f$, where image is formed, is called *image plane*. Here f is a camera focal distance. The projection of point (X, Y, Z) is formed at the intersection of the image plane and the line joining camera origin (focal point) and the considered point. The line from the focal point perpendicular to the image plane intersects the image plane at *principal point* with 2D coordinates (p_x, p_y) relative to this plane.

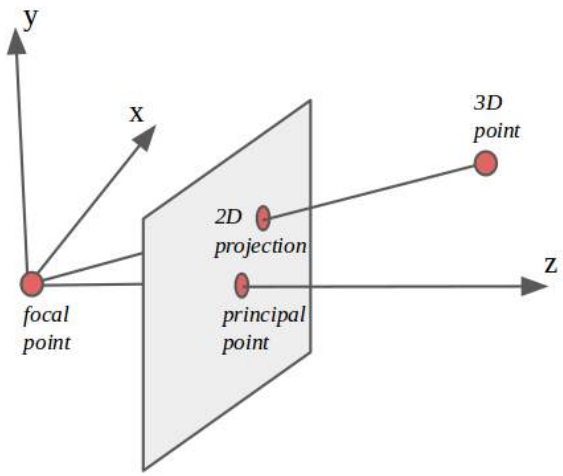


Рис. 3-2: Pinhole camera model. Camera origin coincides with coordinate system origin. Grey plane represents image plane. 3D point in world coordinates is mapped to 2D projection on the image plane.

From triangles similarity, one can write the projection rule:

$$\begin{pmatrix} X & Y & Z \end{pmatrix}^T \rightarrow \begin{pmatrix} f\frac{X}{Z} + p_x & f\frac{Y}{Z} + p_y \end{pmatrix}^T.$$

In homogeneous coordinates this means

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \to \begin{pmatrix} f\frac{X}{Z} + p_x \\ f\frac{Y}{Z} + p_y \\ 1 \end{pmatrix} \sim \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{3.4}$$

Matrix

$$K = \begin{pmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.5}$$

is called *camera calibration matrix* or *camera intrinsics matrix*. The matrix of projection in (3.4) is written as $K[I|0]$.

3.3.3 Camera pose

Usually points are represented in a coordinate system, different from camera local system (*world coordinate system*), so camera origin is not situated at $(0, 0, 0)$ and camera axis orientation does not coincide with coordinate axis. In order to represent a 3D point in camera local coordinates, transformation involving rotation and translation should be applied:

$$\mathbf{X}_{cam} = R(\mathbf{X} - \mathbf{C}),$$

where R is a 3×3 rotation matrix and $\mathbf{C}$ is a vector of camera focal point position. In homogeneous coordinates

$$\mathbf{x}_{cam} = \begin{pmatrix} R & -R\mathbf{C} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}.$$

This 4×4 matrix represents camera position in world coordinates. We will refer to it as *camera pose matrix* P. When two camera pose matrices P_1 and P_2 are given, second

camera pose relative to first camera is $P_2 P_1^{-1}$. This assertion easily follows camera pose definition. Indeed, second camera relative pose is a matrix which defines a projection from first camera local frame to second camera local frame. To compute this projection, we first project points from first camera to world coordinates with P_1^{-1} and then use P_2 to project the obtained points to the reference coordinate system. We will use this fact when computing ground truth values for camera relative displacement. Now we will more precisely look at the intrinsic projective geometry between two views, namely, epipolar geometry.

3.3.4 Epipolar geometry

After the image is formed, the information about only two degrees of freedom is preserved (remember that projection matrix is of size 3×4, destroying one degree of freedom after its application). In order to reproject pixels from an image onto the plane of another image, *depth* information (Z coordinate of the pixel in camera local system) needs to be stored separately. In driving scenarios this information can be sparsely retrieved from LIDAR sensors.

Assume the same point $\mathbf{X}$ in the world coordinates being projected as $\mathbf{x}$ onto first image plane and as $\mathbf{x}'$ onto second image plane. 3×3 matrix F which sets constraint on $\mathbf{x}$ and $\mathbf{x}'$ in form of

$$\mathbf{x}'^T F \mathbf{x} = 0, \tag{3.6}$$

is called *fundamental matrix*. This constraint comes from the fact that the point, both its projections and both cameras focal points are coplanar (see Figure 3-3). Point of intersection of a line that joins the two focal points is called *epipole*. There are two epipoles $\mathbf{e}_1$ and $\mathbf{e}_2$ – one in each image plane. As it can be seen from the picture, for each point $\mathbf{x}$ in the first image, there exists a whole *epipolar line* $\mathbf{l}'$ in the other image. This line can be represented as a cross product $\mathbf{l}' = \mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{x}'$, where $[\mathbf{e}']_\times$ is a 3×3 matrix of cross product. Earlier we stated that there exists a linear transformation H such that $\mathbf{x}' = H\mathbf{x}$. Then $\mathbf{l}' = [\mathbf{e}']_\times H\mathbf{x} = F\mathbf{x}$, where fundamental matrix is defined. Accounting for the fact that $\mathbf{x}'$ belongs to the line $\mathbf{l}'$ ($\mathbf{x}'^T \mathbf{l}' = 0$), we arrive at (3.6).

In the following sections we are going to discuss the problem of finding dense correspondences,

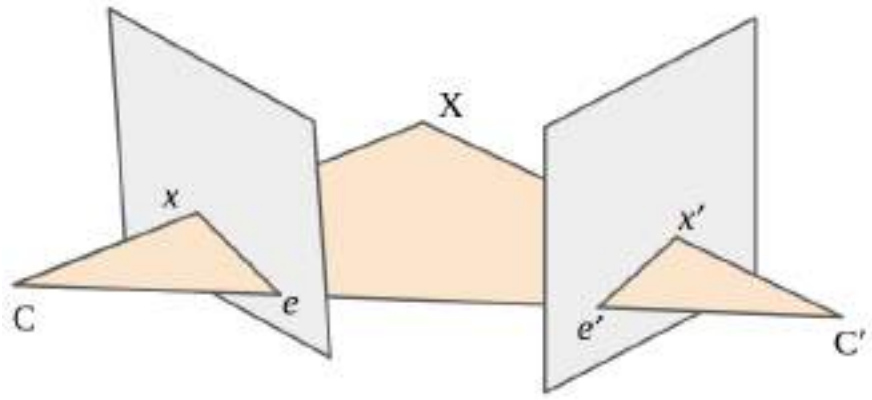


Рис. 3-3: Epipolar constraints. Coplanarity of 3D point X, its projections x, x', cameras origins C, C' and epipoles e, e' is shown in pink.

a solution for which provides an over-determined set of pixel pairs $(\mathbf{x}, \mathbf{x}')$. Fundamental matrix can be estimated from the obtained set, for example, with a five-point algorithm [40]. Together with camera calibration matrix (3.5), fundamental matrix is sufficient to reconstruct relative pose of the cameras up to a translation scale, after cherality constraints are checked (see [20]).

3.4 Correspondence field

In order to understand motion patterns between two images that share field of view, pixel-wise *correspondences* information should be taken into account. Below we discuss methods for predicting *dense* correspondences, which means that prediction is made in each pixel. In B we show how to augment data when dealing with image pairs and dense correspondence targets.

It is natural to represent dense correspondences in a tensor $F \in \mathcal{F}$ of two channels, where in each point displacement vector of corresponding pixel in the first image relative to second image is stored. This means, for two images X and Y, F is such tensor that the following holds:

$$X^d_{i,j} \sim Y^d_{i+F^1_{i,j}, j+F^2_{i,j}}, \quad \forall (i,j) : M_{i,j} = 1, \tag{3.7}$$

where $\sim$ denotes that corresponding pixels from X and Y are images of the same 3D

point (although in RGB space they may be distant because of environmental changes); M is a binary matrix that specifies whether for point (i, j) in image X corresponding point in Y exists (discussed in the following section).

One can encounter representation (3.7) named *flow field* in literature. In our work we write *correspondence field*, as traditionally optical flow estimation assumes static environmental conditions, which is not the case in our experiments.

When visualizing ground truth and predicted correspondence fields we use color coding shown in Figure 3-4.

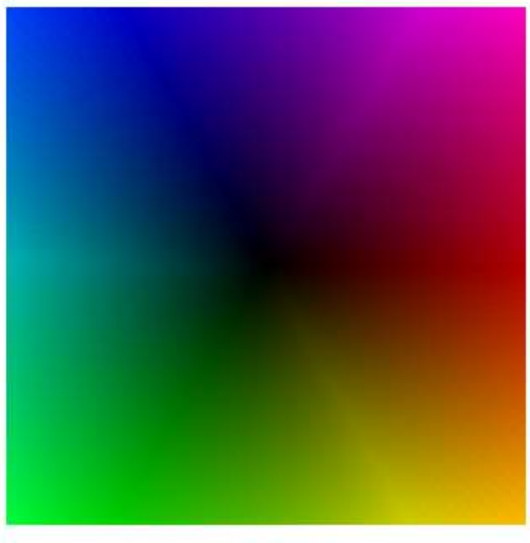


Рис. 3-4: Correspondence field color coding. Displacement of each pixel is a vector from the center to that pixel.

A traditional pipeline for correspondence fields estimation includes hand-crafted features extraction with consequent matching. In our experiments for comparison we provide performance of SIFT Flow [32], so here we provide basic ideas of the approach. The model predicts dense correspondences, directly estimating a displacement vector for each pixel. Firstly, SIFT [34] feature extractor is applied to the image densely. That is, in each pixel 64×64 neighbourhood is divided into 4×4 cells and a descriptor of length 128 is obtained after gradient orientation histogram having 8 bins is built. The obtained $H \times W \times 128$ tensor is called SIFT image. This image is a tensor representation of both inputs X and Y, which is similar to tensor representation extracted by hidden layers of a neural network. Secondly,

matching objective is introduced to make SIFT descriptors match along the flow vectors while preserving smoothness. This objective induces a factor-graph and inference is performed through applying dual-layer loopy belief propagation. Several heuristics are introduced to gradually speed up the process: decoupling smoothness term, coarse-to-fine matching. SIFT Flow pipeline enables to grasp large displacements because matching is performed globally. On the other hand, SIFT descriptors incorporate only local information, which may be detrimental for establishing correspondences in large monotonic regions (e.g. road, walls) or thin objects (e.g. lampposts).

Matchability

Predicting correspondence field densely is an ill-posed problem in general case, because some of the pixels are not visible in the second image due to occlusions, objects leaving the scene, ambiguities etc. Complex scenes are usually cluttered and some objects obscure the others: a tree can occlude part of the building in the first image, which is nevertheless visible in the second image due to 3D motion nature. Some objects can appear or leave the scene because of limited camera field of view. Also, some objects can be present or absent only in one image because of the temporal distance between images: buildings can be constructed or destroyed, moving objects can come and go from the scene. Moreover, it is hardly possible to formalize movement of some pixels, for example of those belonging to sky. In order to resolve this issue, *matchability matrix* is introduced in (3.7) following [51]. Binary random variable is associated with each pixel of the first image, indicating its visibility in the second image (0 meaning not visible and 1 meaning visible). In our experiments we model matchability matrix with a fully convolutional network branch with two neurons softmax activation function on the top (predicting probabilities of each pixels to be matchable).

3.5 Correspondence estimation from a pair of images

For the task of correspondence field prediction we construct a fully-convolutioinal neural network, which takes a pair of images as an input and predicts correspondence field and matchability matrix. In B.2 we also pose correspondence prediction as a classification task

and present a model which predicts discrete variables in each pixel and performs further refinement step.

3.5.1 Two-stream network

When given an input pair of images, one approach to fuse available information might be in stacking these two RGB images along the color dimension and feeding the obtained six-dimensional tensor into a neural network as proposed in [14]. Let's consider what happens at early layers of such neural network. For example, the output of the first convolutional layer represents the result of (3.2) applied directly to the tensor of stacked images. This is a three-dimensional tensor, where feature vector for each pixel is formed after convolving a kernel with a patch extracted from the input tensor. Notice that this input patch is a stack of two patches extracted directly from two images. However, due to large displacements these patches are taken from different parts of the scene, which means that we encode a pair of scene points that in general case have no relation to each other into one feature vector. Understanding of this situation leads to an idea of separate processing of each image before stacking features that already encompass global context.

We propose splitting neural network into two identical parts which share the parameters values, each processing one image from a pair. In this way, meaningful features are built for both images separately. This idea is implemented as a siamese sequence of convolutional and pooling layers, which output spatially downscaled feature tensors. Only at this stage do we concatenate the obtained representations, because each feature vector now encapsulates global context.

Following this approach, the architectures used in our experiments have the following pipeline: parallel features extraction with weights sharing, features fusing through concatenation, target values prediction. The whole model is trained end to end, containing only differentiable operations. In the following section we separately discuss computational graph peculiarities which allow to make models more robust and make predictions more accurate.

3.5.2 Skip-connections

As it was already mentioned, in order to increase prediction accuracy it is important for the model to be able to reconstruct fine-grained details of initial images. For example, edges in a correspondence field are strongly correlated with objects edges on the initial image. Deep neural network does not have explicit mechanisms to store this information and may smooth out important details after several convolutional layers. But it is possible to keep this information by introducing new edges in a computational graph. Straightforward approach is to concatenate output tensors from early layers to input tensors of deep layers in order to provide new, rich with fine-grained information, inputs.

Another option is to extract information from pooling layers, because these layers are exactly the points of information loss in the computational graph. The idea is to keep tensor of arg max indices that appeared in each pooling layer and perform deconvolutions, putting information in places of those indices, leaving other cells empty (zero), in contrast to copying the information or putting it into upper-left positions. This approach was shown to be efficient for semantic segmentation task in [4].

This type of skip-connections is used in the architecture shown in Figure 3-5, which we refer to as *FlowMatchNet*. Tensors depending only on first image are shown in gray, while those depending only on second image are in red. In yellow we show those tensors for which computational graph includes both first and second image. Other colors for tensors will usually denote prediction targets. Arrows show layer input-output tensors relations inside the neural network. Skip-connections are also indicated with arrows. Tensors which are located close to each other (with no arrows between, but having a small gap) are computed in a feed-forward fashion from left to right. Adjoined tensors (having no gap in figure) are concatenated in features axis. Matchability prediction branch, if present, is identical to correspondence prediction branch with the only difference that last layer has softmax activation function. Skip-connection arrows leading to matchability decoder part are not shown on picture for readability. Notice that these two branches do not share parameters (numerical values are different and trained separately) in contrast to features extraction branches, where the two streams have exactly the same sets and numerical values of parameters.

We will use the same style for visualizing architectures further in text.

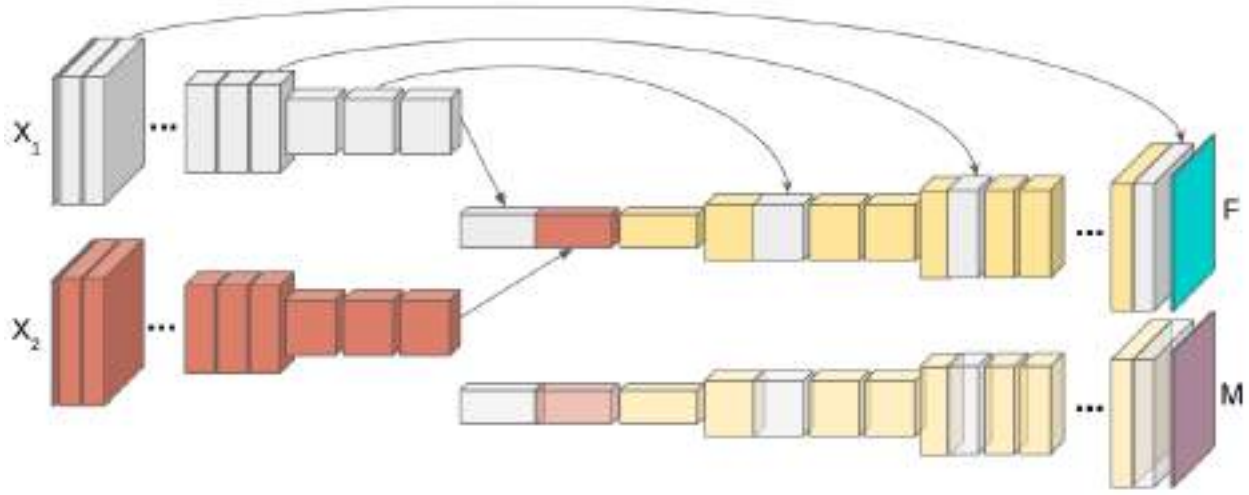


Рис. 3-5: FlowMatchNet architecture with two prediction branches: correspondence F and matchability M. Grey color refers to tensors depending only on first image; red color – to those depending only on second image; yellow color – to those depending on both (information fused). Arrows represent computational graph connections. M branch is identical to F branch (connections are omitted for readability). Two conv-conv-pool and two unpool-conv-conv blocks are hidden with dots on the left and on the right respectively.

3.5.3 Coarse to fine approach

Although correspondence map provides much stronger signal than camera pose, direct correspondence map prediction is still a difficult task. In order to make it simpler for the model to learn, we leverage coarse to fine approach, starting early predictions at small scales from intermediate feature representations. The scale of prediction comes from a network structure naturally, because pooling layers, which downsample tensor representation in spatial dimensions, are present in all our architectures. That is, we predict coarse correspondence field to match 32 times downsampled target map after fifth pooling layer, because its tensor representation has $\frac{H}{2^5} \times \frac{W}{2^5}$ shape when all pooling layers have stride 2. When training the network, additive members are added to the loss function to constrain all the predictions to match the target map. Effectively, these coarse to fine maps serve as regularizes at training stage, providing additional supervision to intermediate network layers and constraining features which they output.

3.5.4 Warping

When dense correspondence field is known, it allows to perform warping operation. Warping A is an operator that is defined over second image X from a pair and correspondence field F. Notice that after displacing pixel (i, j) by F new coordinates are $i + F^1_{i,j}, j + F^2_{i,j} \in \mathbb{R}$, which are not integers in general case. In order to form a valid warped image, interpolation to discrete grid is required, so the result or warping operation $A(X)$ is:

$$A(X) = Y: \quad Y_{i,j} = interpolate\left(X_{i+F^1_{i,j}, j+F^2_{i,j}}\right). \tag{3.8}$$

Using bilinear interpolation to a discrete grid in (3.8) is a differentiable operation, as shown in [24], although it involves rounding obtained coordinated to nearest integer.

Warping representations Earlier we have understood that skip-connection are required to enable fine-grained predictions. However, simple concatenation of tensors from early layers will lead to the same issue that was discussed in 3.5.1: the model will be forced to fuse information from local patches which do not have any relation to each other. We propose to firstly distort tensor representation before concatenation in order to fuse information properly.

Notice that warping operation (3.8) is defined on arbitrary-sized 3D tensor $Z \in \mathcal{F}$, provided that correspondence field F has the same spatial size (which can be achieved by resizing using nearest-neighbout or bilinear interpolation).

In the previous section we discussed coarse-to-fine approaches, so we can construct an architecture that would predict coarse correspondence fields in its hidden layers. These fields can be used to warp tensors from early layers. The resulting architecture, referred to as *WarpNet*, is shown in Figure 3-6. Matchability branch, if present, is kept as in FlowMatchNet.

3.5.5 Loss model

Both FlowMatchNet and WarpNet are trained with ground truth F and M supervision. In this section we denote $\hat{F}$ and $\hat{M}$ to be the model's predictions. Total loss is defined as

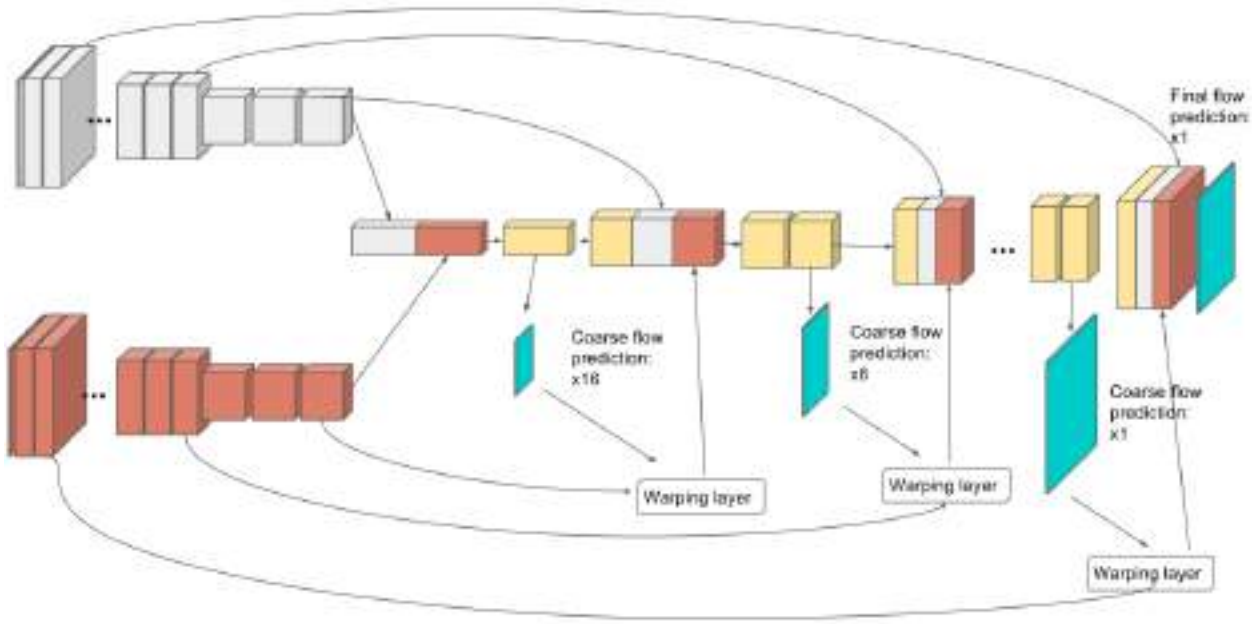


Рис. 3-6: WarpNet architecture. Arrows show computational graph connections. Blue tensors represent correspondence fields predicted at different scales. Representations of second image are bilineary distorted inside warping layers with coarse fields. Dots hide two blocks ($\times 2$ and $\times 4$ scales) in both cases.

$$\mathcal{L}\left(F, M, \hat{F}, \hat{M}\right) = \lambda \mathcal{L}_{corresp}\left(F, M, \hat{F}\right) + \mathcal{L}_{match}\left(M, \hat{M}\right), \tag{3.9}$$

where λ is a meta-parameter, $\mathcal{L}_{match}$ is an arithmetic average of per-pixel binary cross-entropy loss over all pixels. For FlowMatchNet $\mathcal{L}_{corresp}$ is an arithmetic average of a per-pixel correspondence loss function over matchable pixels:

$$\mathcal{L}_{corresp}\left(F, M, \hat{F}\right) = \frac{1}{N_{match}} \sum_{p:M(p)=1} \ell\left(F(p), \hat{F}(p)\right), \quad N_{match} = |\{p : M(p) = 1\}|. \tag{3.10}$$

For WarpNet $\mathcal{L}_{corresp}$ is a weighted sum of losses on each predicted map. These losses are of the same structure as (3.10).

It is natural to choose ℓ to be L_2 squared loss $\ell\left(F(p), \hat{F}(p)\right) = \|F(p) - \hat{F}(p)\|_2^2$. However, individual pixels displacements values range from zero to several hundreds. For instance, a histogram of absolute displacement values on SYNTHIA dataset is given in Figure A-1. Choosing L_2 squared loss leads to noisy predictions and noisy training curve, which we observed in our experiments. In order to avoid introducing noise into our model, we choose

truncating this loss with meta-parameter coefficient T:

$$\ell\left(F(p), \hat{F}(p)\right) = \min\left(\|F(p) - \hat{F}(p)\|_2^2, T^2\right).$$

3.6 Cross-seasonal prediction

3.6.1 Modeling conditions and predicting appearance

When traversing the same scene several times with a big time gap, several things change dramatically. We have already discussed that some objects may be present or absent in one image and the overall appearance of the scene may vary. Here we visit the latter problem, introducing a notion of appearance conditions definition and presenting a model that makes a step towards unifying the appearances of the same scene.

Appearance changes of outdoor scenes are physically conditioned by relatively few factors: position of the sun, cloud cover, precipitations. Knowledge of these factors would allow to model changes in objects colors because of illumination, weather (e.g. snow on pavements) and shades. However, some of these factors are hard to formalize to be numerical parameters. Moreover, as discussed in 2.3, few datasets are available with reach hand-labeled information, none of them large-scale.

In order to avoid the requirement of ground truth information about the mentioned factors for each image, we propose an architecture in encoder-decoder fashion, which combines scene objects from the first image and appearance conditions from the second image. We define conditions to be a small-dimensional vector, which is sufficient to reconstruct appearance of second image under extracted conditions. This is an image that would have been obtained by the camera in position identical to first camera position with environmental condition identical to second image. Putting it another way, the aim is to reconstruct warped first image given second image and the conditions.

Formally, let X, Y be a pair of images. Let $c : \mathcal{X} \to \mathbb{R}^d$ be a function that encodes the conditions in a small-dimensional vector given an image. We require the existence of

reconstruction function $r : \mathcal{X} \times \mathbb{R}^d \to \mathcal{X}$ such that

$$r(Y, c(X)) \approx W(X, p_Y).$$

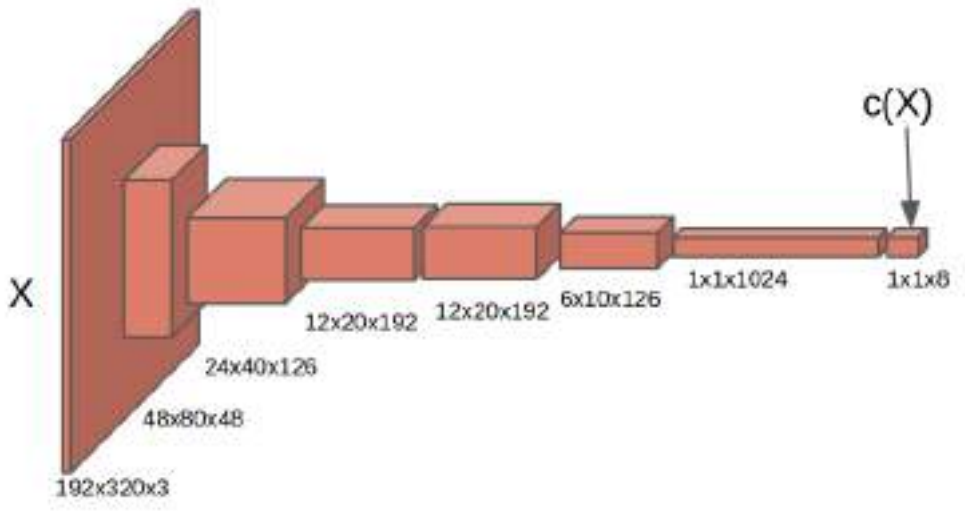


Рис. 3-7: Architecture modeling $c(X)$: conditions encoder. Takes RGB image as an input and outputs d numbers after seven convolutional layers.

We model c with a neural network. The architecture should output d numbers when given an image. Usually, this is done with a number of convolutional layers, followed by several dense layers. We generally follow this approach, but pose all dense layers as convolutional, setting the appropriate kernel size for the first of those layers to make the output of spatial size 1×1 as discussed in 3.2.3. This slight modification only keeps the d numbers in features dimension, having the exactly the same number of parameters and exactly the same training and inference equations. The proposed architecture is shown in Figure 3-7. Convolutional layers correspond to AlexNet [28] first five layers, batch normalization operations are inserted after each layer. Nonlinear rectifier function (ReLU) is used everywhere except last layer, which is linear. We choose $d = 8$ in the experiments.

While modeling r we introduce another function $f : \mathcal{X} \to \mathcal{F}$ which refers to second image feature extractor. At the end, to produce the desired output, we extract features from Y with f, forming the tensor of shape $h \times w \times d_{feat}$ and combine it with conditions tensor of shape $1 \times 1 \times d$, simply repeating the values in first and second dimensions and stacking

both tensors along last dimension. Architecture of feature extractor is shown in Figure 3-8.

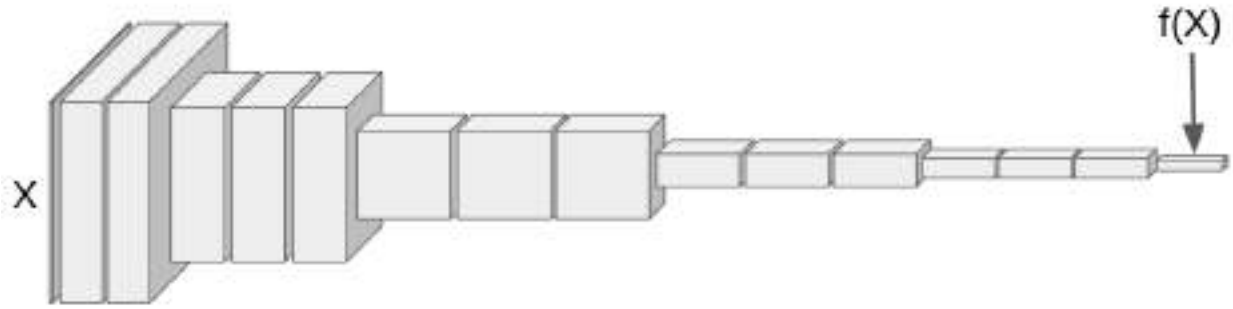


Рис. 3-8: Image feature extractor $f(X)$: deep encoder-type convolutional neural network. Has five hidden pooling layers, which yields a tensor of $\times 32$ smaller spatial resolution compared to input image X.

Figure 3-9 shows the way r and c are merged into one architecture which is fully differentiable. The only required input is a pair of perfectly aligned images X and Y or a pair of images X and Y together with full information for warping X, that allows training without knowing any numerical ground truth values of any environmental factors. Alignment can be achieved in case of fixed camera (web-camera datasets) or in case of knowing depth and pose information that allow re-projecting one of the images onto the plane of another. This information if required to construct a loss function, penalizing output deviation from $W(X, p_Y)$. Because output prediction models the first image, we select mean pixel-wise L_2 loss in color space. We refer to this architecture as *CondNet*.

3.6.2 Fusing conditions model with correspondence estimation

The model described in previous section effectively allows to walk in space of image appearances, injecting d-dimensional parameter instead of input obtained from $c(X)$. Intuitively, modeling the scene appearance under different conditions should help to improve the quality of establishing correspondences between two images because of the following consideration. Pairs capturing the same scene under same conditions are easier than those capturing the same scene under different conditions. Truly, optical flow problem is much easier than cross-appearance correspondence problem.

We once again stress that perfect image alignment or warping information was required

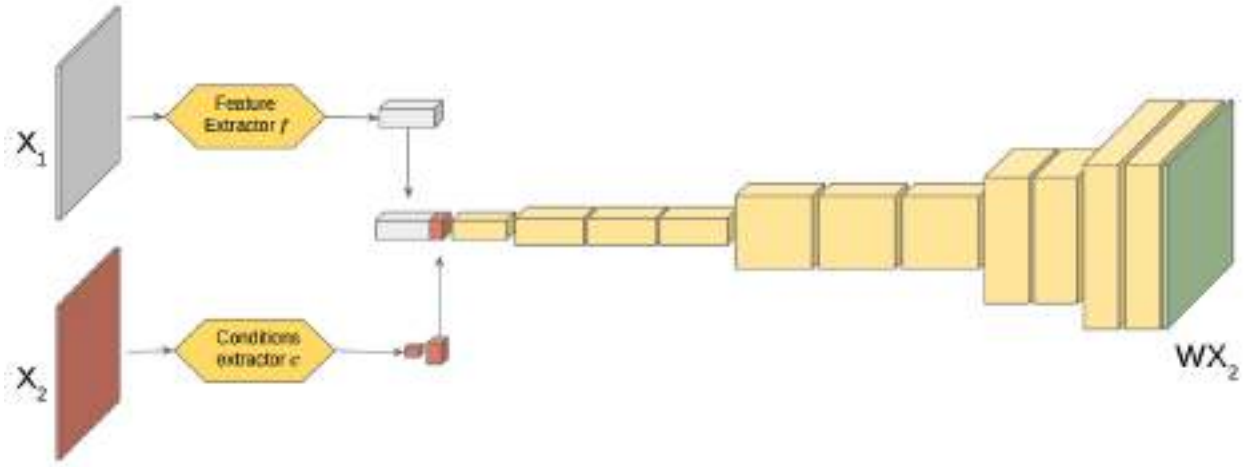


Рис. 3-9: CondNet architecture. Combines features encoder from first image, conditions encoder from second image, fusing concatenation layer and reconstruction branch. Target output is second image warped onto the plane of first image.

only during training stage to compute loss value. In other words, when solving the task of predicting correspondences, the model that extracts conditions from X using c and applies those to Y using r, no pixels displacement is introduced. So,

$$F(X, Y) = F(X, r(Y, c(X))).$$

This idea leads to simple concept of stacking two models. First, predict appearance of Y under conditions from X and then establish correspondences between the resulting image and X.

In fact for any conditions vector c

$$F(X, Y) = F(X, r(Y, c)).$$

Because FlowMatch architecture extracts features from both images with Siamese network part, it is easier to match images from the same domain. Previously, even if $\hat{Y} = r(Y, c(X))$ closely resembled $W(X)$, it was still not real image. Thus, we would extract features from real X and realistic $\hat{Y}$ with exactly the same network part. It is better to extract features from the same domain. We propose to feed $r(X, \mathbf{0})$ and $r(Y, \mathbf{0})$ as FlowMatch input pair, simply reducing both images to zero-conditions appearance.

The architecture modeling this idea is constructed in Figure A-9. In the same figure we show another approach which also exploits CondNet model as a part of computational graph. The idea is to regularize features that are extracted from the bottleneck of the encode-decoder type correspondence prediction architecture. In order to enable good cross-seasonal predictions, it is natural to require independence of image representations, which are fused in the bottleneck, from appearance conditions. For example, if instead of X_2 another image was provided that captures the scene from exactly the same point but under different conditions, the result should stay exactly the same. To regularize the model with this constraint, we introduce additive loss term, which will penalize the difference between features extracted from X_2 and those extracted from $r(X_2, c(X_1))$.

3.7 Displacement estimation

Having discussed methods that estimate correspondences between two images at pixel level, we describe approaches which leverage this information for camera geometrical displacement estimation.

Angle and direction estimation Correspondence information forms an over-determined system that is sufficient to estimate fundamental matrix F (3.6) (for example RANSAC estimator can be used for five-point algorithm [40]). Using camera intrinsics matrices K and K', essential matrix is defined as $E = K'^T F K$. This matrix allows only to recover relative displacement up to scale, so that the direction and the angle are recovered, but the magnitude is not.

Relative camera pose can be estimated from the essential matrix E through SVD decomposition. Supposing $E = Udiag(1, 1, 0)V^T$, there are two possible factorizations $E = SR$ with $S = UZU^T$ and $R = UWV^T$ or $R = UW^TV^T$, where

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

R here is a rotation matrix. Also, last column of U is collinear to the direction of displacement. This means, up to scale, two options are possible for displacement direction (U_3 or $-U_3$). Proof is provided in [20].

Of the obtained four options, only one is left after checking *cherality constraints*, which reproject matched points into their 3D preimages and ensure that obtained points lie in front of both cameras.

Neural network for displacement estimation The model is required to estimate magnitude of geometrical displacement between cameras. It is also possible to construct regression on any number of chosen degrees of freedom.

We have already discussed the architecture $c(X)$ which predicts d numbers from an input tensor in 3.6.1. In driving scenarios it is natural to choose $d = 2$ (magnitude and azimuth prediction) or $d = 3$ (magnitude, azimuth and camera yaw prediction), because the platform always stays on the road and has no freedom in Z dimension and camera is fixed on the car, so roll and pitch angle are fixed. Together with correspondence prediction model, whole architecture can be fine-tuned end-to-end (only using displacement ground truth information as a supervision signal). We call this model *DispNet*.

Глава 4

Эксперименты

This chapter describes computational experiments, in which we train and evaluate the proposed models. All experiments are set in Python. GPU implementation of neural networks architectures is done in Theano and Lasagne. In all our experiments weights are initialized uniformly as in [18]. Adam optimizer [27] is used for training the models. Unless otherwise specified, qualitative results (visualization of models performance) are provided on validation samples, which were never shown during training.

4.1 Training schedules

4.1.1 Datasets

We start our experiments on SYNTHIA dataset, referring to samples at 320×192 resolution as $SYNTHIA$, and to samples at 160×92 resolution as $SYNTHIA_{small}$. We also add "same"in a superscript to show that both images in a pair come from the same route traverse (e.g. $SYNTHIA^{same}_{small}$).

As discussed in section 2.4, SYNTHIA dataset does not contain ground truth flow fields for image pairs, but rather provides depth maps for each image together with camera intrinsic parameters. This information is enough to re-project points of static objects (buildings, lamppost) from one image plane to another (see (3.4)). Moving objects' motion can not be extracted from the available information, but in this thesis we are mainly interested in

predicting target values between different traverses of the same route, which means ground truth motion for these objects is undefined. Leveraging semantic information of the dataset, pixels belonging to moving objects can be factored out from the loss function during training. The drawback of this approach can be seen when fine-tuning models on optical flow datasets (e.g. KITTI dataset), where errors tend to be high on moving objects. In order to expand a model beyond learning only background (single rigid body) motion, we pre-training on FlyingChairs dataset, which contains individual objects' motion information, before going for SYNTHIA with further fine-tuning on KITTI.

For evaluation on KITTI dataset, we split KITTI2015 into training (80% of samples) and validation (20% or 40 images), also including KITTI2012 into training set (making 354 training samples in total). Original training-validation split is used for FlyingChairs dataset. Simply splitting images for SYNTHIA and NCLT into training and validation sets would be unfair, as same objects would appear in both sets, although captured from different positions. For both dataset we project global coordinates onto map and separate regions for training and validation, ensuring absence of training scenes in validation set.

Sampling from SYNTHIA and NCLT is performed non-uniformly, because these datasets represent continuous route traverses. While usually in computer vision tasks samples are taken from dataset with equal probability, here a platform can pass different parts of the environment at different speed or pass them different number of times. In both cases, training pairs will not be balanced across scenes of the environment. Moreover, if the platform waits for the traffic light, big number of training pairs will be generated with camera displacement being close to zero. We want the distribution to be balances both *across scenes* and *across possible camera displacements* (within some range, for which images have sufficient overlap of fields of view).

In SYNTHIA dataset the platform traverses exactly the same route several times, which means each route can be mapped to a line segment as shown on Figure 4-1. All images captured along the route are represented as points on the segment. Example in the figure shows a route that has been traversed twice (gray and yellow points represent first and second traverses). To pick the first image (*an anchor*) a point is uniformly sampled from the line segment. Then, nearest image is picked to be an anchor. Second image is picked

with the same procedure, when new segment is a sub-segment containing all images in a neighbourhood of the anchor. This procedure ensures sampling of pairs that balances visible environment scenes and cameras relative displacement. NCLT platform not always travels exactly the same route, so we sample first image with probability inversely proportional to the number of pairs this image is associated to.

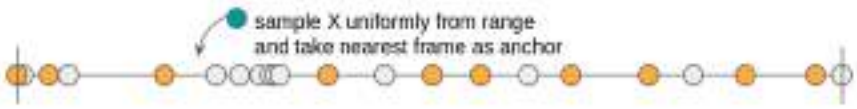


Рис. 4-1: Sampling from SYNTHIA dataset. Camera poses from two traverses (yellow and grey) are mapped to a line segment. Point X is sampled uniformly from the segment after which first image (*anchor*) is picked. In the neighbourhood of the anchor the same procedure is repeated to pick second image. This procedure ensures diversity of scenes and relative displacements in training set.

4.1.2 Pre-training with ground truth warping

WarpNet architecture experiences a problem of cold start. This can be seen from Figure 4-2, where we show training set average endpoint error for first 20 thousand iterations of FlowMatchNet and WarpNet. The curve for WarpNet goes above the curve for FlowMatchNet on the early stages, because on these stages coarse correspondence fields prediction is poor. Remembering that those fields are used for warping intermediate tensor representations, we understand that large noise is introduced to the model from the beginning, which dumps the speed of convergence.

To overcome this negative effect one can perform first iterations, providing ground truth correspondence fields for warping operations. We do not remove coarse prediction branches or change the loss model, only injecting ground truth values in those places, where coarse predictions are used in regular mode. The effect of this change is also displayed in Figure 4-2. After 20 thousand iterations, we switch training mode to regular regime. We use this schedule to train WarpNet architecture in our experiments.

We stress that this section describe only first part of training iterations and is aimed at developing effective pre-training method. FlowMatchNet seeming superiority at this point

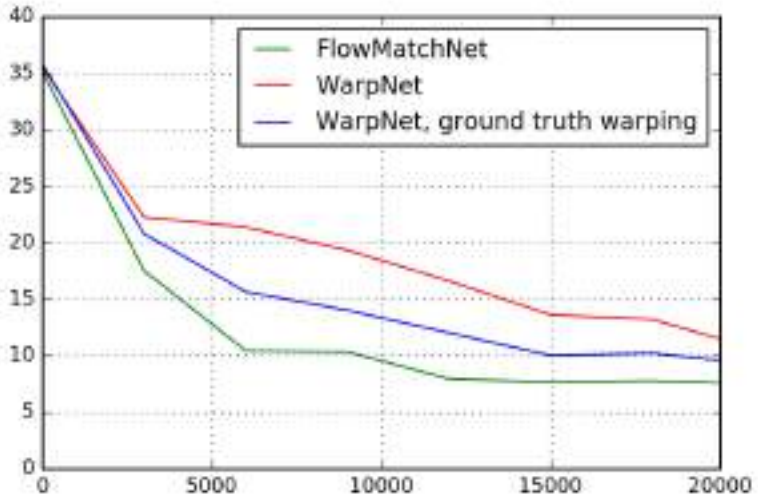


Рис. 4-2: Average EPE (endpoint error) convergence for first 20 thousand training iterations. FlowMatchNet starts converging faster than WarpNet, because random coarse predictions are used for warping at early stages. To improve WarpNet convergence, we start warping with ground truth F for the first 20 thousand iterations, switching to regular mode afterwards.

does not survive after several hundreds thousands of iterations, which we will describe in the following sections.

4.2 Correspondence prediction

Now let's describe and analyze the results of correspondence prediction methods. In this section we provide results for FlowMatchNet and WarpNet. While training was done with the loss described in Chapter 3, in this section we report endpoint mean errors (EPE) for interpretability.

4.2.1 Comparison

From violin plots in Figure 4-3 we can clearly see that WarpNet architecture performs substantially better than FlowMatchNet, having 3.2 mean per-pixel EPE compared to 6.0. Both distributions have heavy tales, which is natural due to large ground truth displacements. Figure A-2 shows per-image mean EPE dependency on mean ground truth. Notice a "fracture"in FlowMatchNet plot, which means that the model can not cope well with large displacements.

For WarpNet the plot does not have such a loss pattern, although mean EPE slightly grows all the way from zero to large ground truth displacements. Therefore, coarse-to-fine approach together with warping operations introduce strong regularization to the model, making it grasp large pixels displacement.

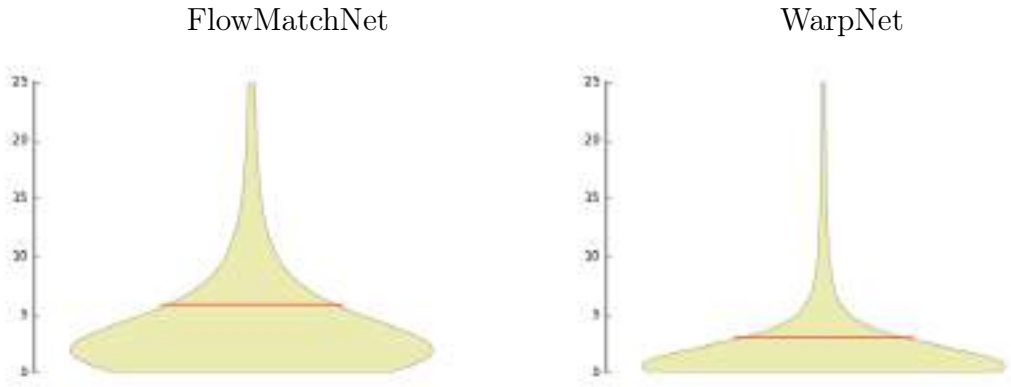


Рис. 4-3: Violin plots for FlowMatch and WarpNet: distribution of endpoint errors (EPE). Mean value is shown in red. Plot is truncated at 25 pixels to better represent bottom part

Figure also shows one validation sample from SYNTHIA with challenging rainy conditions and EPE heat map for both architectures on this pair of images. WarpNet grasps road pixels better as well as pixels of small or thin objects such as tree (these objects usually have large displacements as they are closer to camera).

Training and validation EPE for different datasets are aggregated in Table 4.2.

FlowMatchNet and WarpNet have a number of differences as described in Chapter 3, so from comparison provided in this section it may not be clear whether warping operations play big role in WarpNet being much better than FlowMatchNet. To provide direct evidence of warping layers impact, in the following section we study WarpNet in comparison to the same architecture without warping layers and also to an architecture which refines its predictions in an iterative manner.

4.2.2 Warping layers effect

A neural network having the same architectural structure as WarpNet but with no warping operations is called *base architecture* in this section. Below we discuss superiority of WarpNet

over base architecture and also show that simply introducing warping layers gives better results than performing iterative refinement.

WarpNet follows coarse to fine idea of predicting a flow, reusing its coarse predictions to warp sample representation. Another similar approach involves predicting flow at full scale, then stacking the obtained prediction with image pair and run through *refinement network* [21]. Notice that refinement network can be applied multiple times in an iterative fashion, because it inputs image pair and predicted field from previous iteration and again outputs predicted field (refined), which can be again stacked with the image pair to form new input. The first part of the joint architecture is a bit different as it takes only image pair as an input, having no prior information about correspondence field.

To make a fair comparison, we select base architecture to be the first part of the final architecture. We train three architectures: only base network (B), base network followed by one step of iterative network (BR(1)) and base network followed by two steps of iterative network BR(2). In the latter model iterative network weights are not shared, i.e. the two steps have separate set of parameters. When stacking an iterative part on top of already trained model, we preserve the weights of the bottom part and start with training only the top part (last iterative step). We also fine-tune the whole architecture end-to-end after having trained both parts (top and bottom). We write '+f' when showing fine-tuned results.

We perform training on SYNTHIA$^{same}_{small}$ and provide the results in Table 4.1 to prove the concept of warping layers approach being superior to iterative approach. From this table one can see that WarpNet has considerably smaller overfitting gap ($1.54 - 1.12 = 0.42$) compared to all other architectures present in the table (e.g. $2.53 - 1.23 = 1.30$ for base network). Also, even though WarpNet is approximately 1.5 times slower than the base network, it is almost twice faster than architecture with two iterative refinement steps. Thus, introducing warping operations helps to generalize the model to new (validation) data with reasonable loss in run time.

4.2.3 Visualization

Visualization of WarpNet performance on NCLT dataset is provided in Figure A-3. In each example, a pair of images is shown together with ground truth correspondence field, predicted

	WarpNet	B	BR(1)	BR(1) + f	BR(2)	BR(2) + f
Training EPE	1.12	1.23	1.14	1.08	1.03	0.94
Validatons EPE	1.54	2.53	2.32	2.35	2.19	2.13
Forward run time (ms per pair)	33	24	42	-	64	-

Таблица 4.1: Comparison of WarpNet with iterative architecture. "B"denotes base architecture (same structure, no warping); "BR(k)"denotes base architecture with $k = 1$ or 2 refinements. Each refinement copy is added preserving already trained weights for bottom layers. Weights are not shared for refinement steps. "+f"denotes fine-tuning end-to-end. WarpNet overfitting gap is smaller on validation. WarpNet is faster than BR(1) and BR(2).

correspondence field (full and masked with ground truth mask of LIDAR hits) and EPE heat map. Largest errors are achieved on pixels corresponding to road, branches and moving objects. Firstly, moving objects ground truth information is incorrect. Because dataset only provides depth maps, having no semantic information, moving objects can not be factored out, so when we reproject their depth maps, we introduce noise into the dataset. Accounting for the fact that LIDAR does not capture most of pixels of the road, it is hard for the model to grasp its motion pattern. Branches on the trees also have substantial noise in ground truth data as LIDAR depth estimations have finite resolution and branches are very thin, so reprojection may be incorrect. Therefore, inaccuracy of ground truth data is a limitation in this case.

Matching results, obtained with WarpNet, for random pixels are shown in Figure A-4 for NCLT, in Figure A-5 for KITTI. On the KITTI dataset motion of moving objects is learned successfully, which can be seen from point to point matches between the parts of cars.

4.2.4 Endpoint errors

In this section we aggregate results for correspondence prediction of the discussed architectures in Table 4.2. We also show SIFT Flow [32] performance for comparison (the method was discussed in 3.4).

	SYNTHIA	NCLT	Flying Chairs	KITTI
WarpNet	3.2	14.8	2.1	6.4
FlowMatchNet	6.0	17.0	3.2	8.0
SIFT Flow [32]	16.2	29.6	4.4	21.4
Zero prediction	29.2	44.3	10.7	31.3

Таблица 4.2: Endpoint average error results. WarpNet is better than FlowMatchNet across all datasets. SIFT flow and zero prediction results (average magnitude) are given for comparison.

4.3 Appearance prediction

We trained CondNet architecture for 200 thousand iterations on SYNTHIA until training loss convergence.

The only image-level label available represents environmental conditions that were modeled during the run. Although we never used this label during training, it is natural to suppose that seasons are encoded in vectors, which are extracted from raw images by c. In Figure A-6 we show t-SNE 2D projections for these eight-dimensional conditions vectors. Images labeled as 'dawn', 'winter', 'night', 'rain' form well-separated clouds, while 'summer', 'spring' and 'fall' are mixed. Indeed, these three seasons are usually indistinguishable in urban environment. This means that seasonal information is encoded into conditions.

Figure A-7 shows two random examples of CondNet performance. First row represents a pair of images, second row provides the result of CondNet prediction together with the heat map of mean absolute difference between the predicted image and warped second image. As shown in Table 4.3, error (absolute difference averaged over pixels and RGB channels) drops to 12 on training set, equaling 13 on test sample, so only minor over-fitting takes place.

We also show the results on Transient Attributes Database [30]. 21 of 101 web-cameras are put off for validation, leaving 80 cameras for training. Table 4.3 shows mean CondNet error, which is L_2 norm of pixel-wise difference between ground truth X_2 and predicted appearance of X_1 under the conditions extracted from X_2. Over-fitting to the training set is clearly visible because of the small number of samples in the training set. Qualitative results for validation set are given in Figure A-8. The model learns to predict well large details, such as color of the sky and major spots of light. Unfortunately, having only 80 cameras in the training set does not provide diverse enough set to model the appearance of small attributes.

Dataset	Training	Validation
SYNTHIA	12	13
Transient Attributes	19	28

Таблица 4.3: CondNet mean absolute difference error. Good generalization is achieved on SYNTHIA due to diverse enough training set. Overfitting on Transient Attributes is substantial as training set consists of only 80 scenes

Conditions model effect on correspondence estimation

We provide the results for both ideas from 3.6.2 (preliminary prediction and regularization) for SYNTHIA dataset in Table 4.4. In both cases, training error drops, but validation error is higher. In other words, introducing CondNet into this general pipeline only deteriorates the results and widens the overfitting gap. Fine-tuning the whole architecture in an end-to-end fashion with small learning rate also does not help to reduce the overfitting gap.

	FlowMatchNet	FlowMatchNet + CondNet(p)	FlowMatchNet + CondNet(r)
Training	4.5	3.9	4.0
Validation	6.0	7.2	7.1

Таблица 4.4: CondNet effect on correspondence estimation deteriorates the result on validation, widening the overfitting gap

4.4 Geometrical displacement estimation

Finally, we provide the results for camera displacement estimation on SYNTHIA and NCLT datasets. For SYNTHIA magnitude is measured in Unity units, while for NCLT – in meters. Angles errors are measured in degrees from 0 to 180.

On SYNTHIA, we report DispNet model combined with WarpNet for magnitude and azimuth prediction. RANSAC algorithm applied to full correspondence field gives worse results both for camera orientation and displacement magnitude. Because of the fact that the platform travels along the straight lane most of the time, camera displacement yaw angle is equal to zero for such pairs. That is why 3DOF prediction has shown worse results in our experiments compared to 2DOF, which we are reporting in Table 4.5. Azimuth mean error might be misleading, because when camera motion is small, any predicted value of

azimuth will not lead to major displacement error as long as magnitude is predicted correctly. We achieve substantial improvement of platform localization with mean displacement error equaling 0.20 while mean ground truth displacement is 1.75 Unity units.

On NCLT we also report the results for DispNet model combined with WarpNet. Again, when predicting 3DOF (azimuth, magnitude, yaw), results for azimuth and magnitude are deteriorated compared to 2DOF prediction. On the other hand, yaw angle is predicted better in 3DOF mode than when predicting it as a single target.

	SYNTHIA	NCLT
Azimuth error, °	4.7	10.0
Yaw error, °	-	2.2
Magnitude error	0.11	0.56
Displacement error	0.20	0.76
Average ground truth displacement	1.75	2.62

Таблица 4.5: Displacement estimation results. Displacement error represents Euclidean norm of localization error. DispNet for 2DOF prediction is reported on SYNTHIA, because yaw angle changes are not diversely represented in training set. Substantial localization refinement is achieved from 1.75 to 0.20 Unity units. For NCLT localization uncertainty reduces from 2.62 to 0.76.

Глава 5

Заключение

In this thesis we have addressed cross-appearance image matching problems in presence of large displacements, proving applicability of deep learning based methods. On the way to solving these problems, we have based the proposed methods on recent progress in the field, at the same time embracing basic knowledge in this work. We build bridges between deep learning and traditional localization approaches, introducing differentiable warping operations for arbitrary tensor representations as well as solving localization problem through establishing correspondences between pixels.

We discuss architectural pipeline and its variations together with training schedules. We show how different models are combined to form a fully-differentiable model, which can be trained end-to-end. Comprehensive analysis is provided helping to obtain deep insights. The proposed pipeline can be naturally expanded with post-processing methods basing on graphical models, shallow neural networks etc.

Synthetic data sets were leveraged, which has been proven to provide rich prior information about the scene motion and appearance patterns. However, cross-seasonal driving data with minimal noise is required to perform confident fine-tuning of the models. We have shown substantial error decrease in real outdoor scenarios, which, however, have not yet reached the desired accuracy. Collection of large cross-appearance matching datasets in real domain together with improving the proposed models is a direction of future work. Structure from motion methods may be used to construct large collections of cross-seasonal driving data, where precise ground truth poses in world coordinates will be present together with sparse

or dense pixels depth information.

Our work can be expanded with exploring semantic information that would allow better understanding of correspondence fields. The problem of semantic segmentation is much easier, because it involves only one image as an input and recent methods show impressive performance for driving scenarios.

Speed of training and speed of forward inference are essential things to address, because the model needs to be fast to cope with real-time driving and robotics scenarios. Moreover, convergence is achieved after hundreds of thousands iterations, which take days or weeks of GPU-time. Faster architectures should be considered and training process should be controlled carefully.

Литература

[1] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 37–45, 2015.

[2] Shervin Ardeshir, Amir Roshan Zamir, Alejandro Torroella, and Mubarak Shah. Gis-assisted object detection and geospatial localization. In *European Conference on Computer Vision*, pages 602–617. Springer, 2014.

[3] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *European conference on computer vision*, pages 584–599. Springer, 2014.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[5] Aayush Bansal, Hernán Badino, and Daniel Huber. Understanding how camera configuration and environmental conditions affect appearance-based localization. In *Intelligent Vehicles Symposium Proceedings*, pages 800–807. IEEE, 2014.

[6] Chris Beall and Frank Dellaert. Appearance-based localization across seasons in a metric map. *6th PPNIV, Chicago, USA*, 2014.

[7] Alessandro Bergamo, Sudipta N Sinha, and Lorenzo Torresani. Leveraging structure from motion to learn discriminative codebooks for scalable landmark classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 763–770, 2013.

[8] Andreas Breitenmoser, Laurent Kneip, and Roland Siegwart. A monocular vision-based system for 6d relative robot localization. In *Intelligent Robots and Systems (IROS)*, pages 79–85. IEEE, 2011.

[9] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):500–513, 2011.

[10] Nicholas Carlevaris-Bianco, Arash K Ushani, and Ryan M Eustice. University of michigan north campus long-term vision and lidar dataset. *The International Journal of Robotics Research*, 35(9):1023–1035, 2016.

[11] David M Chen, Georges Baatz, Kevin Köser, Sam S Tsai, Ramakrishna Vedantham, Timo Pylvänäinen, Kimmo Roimela, Xin Chen, Jeff Bach, Marc Pollefeys, et al. City-scale landmark identification on mobile devices. In *Computer Vision and Pattern Recognition (CVPR)*, pages 737–744. IEEE, 2011.

[12] Winston Churchill and Paul Newman. Experience-based navigation for long-term localisation. *The International Journal of Robotics Research*, 32(14):1645–1661, 2013.

[13] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[14] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[15] Xiang Gao and Tao Zhang. Unsupervised learning to detect loops using deep neural networks for visual SLAM system. *Autonomous Robots*, pages 1–18, 2015.

[16] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[17] Ben Glocker, Shahram Izadi, Jamie Shotton, and Antonio Criminisi. Real-time rgb-d camera relocalization. In *Mixed and Augmented Reality (ISMAR)*, pages 173–179. IEEE, 2013.

[18] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[19] Abner Guzman-Rivera, Pushmeet Kohli, Ben Glocker, Jamie Shotton, Toby Sharp, Andrew Fitzgibbon, and Shahram Izadi. Multi-output learning for camera relocalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1114–1121, 2014.

[20] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[21] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint arXiv:1612.01925*, 2016.

[22] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[23] Nathan Jacobs, Nathaniel Roman, and Robert Pless. Consistent temporal variations in many outdoor scenes. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1–6. IEEE, 2007.

[24] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[25] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *Robotics and Automation (ICRA)*, pages 4762–4769. IEEE, 2016.

[26] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Convolutional networks for real-time 6-dof camera relocalization. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[27] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[29] Alexander Krull, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. Learning analysis-by-synthesis for 6d pose estimation in rgb-d images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 954–962, 2015.

[30] Pierre-Yves Laffont, Zhile Ren, Xiaofeng Tao, Chao Qian, and James Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM Transactions on Graphics (TOG)*, 33(4):149, 2014.

[31] Hyon Lim, Sudipta N Sinha, Michael F Cohen, and Matthew Uyttendaele. Real-time image-based 6-dof localization in large-scale environments. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1043–1050. IEEE, 2012.

[32] Ce Liu, Jenny Yuen, and Antonio Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):978–994, 2011.

[33] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

[34] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[35] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.

[36] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[37] Sven Middelberg, Torsten Sattler, Ole Untzelmann, and Leif Kobbelt. Scalable 6-dof localization on mobile devices. In *European conference on computer vision*, pages 268–283. Springer, 2014.

[38] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[39] Peer Neubert, Niko Sunderhauf, and Peter Protzel. Appearance change prediction for long-term navigation across seasons. In *Mobile Robots (ECMR)*, pages 198–203. IEEE, 2013.

[40] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[42] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

[43] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.

[44] Mike Smith, Ian Baldwin, Winston Churchill, Rohan Paul, and Paul Newman. The new college vision and laser data set. *The International Journal of Robotics Research*, 28(5):595–599, 2009.

[45] Niko Sünderhauf, Sareh Shirazi, Feras Dayoub, Ben Upcroft, and Michael Milford. On the performance of convnet features for place recognition. In *Intelligent Robots and Systems (IROS)*, pages 4297–4304. IEEE, 2015.

[46] Niko Sunderhauf, Sareh Shirazi, Adam Jacobson, Feras Dayoub, Edward Pepperell, Ben Upcroft, and Michael Milford. Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. *Proceedings of Robotics: Science and Systems XII*, 2015.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[48] Julien Valentin, Matthias Nießner, Jamie Shotton, Andrew Fitzgibbon, Shahram Izadi, and Philip HS Torr. Exploiting uncertainty in regression forests for accurate camera

relocalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4400–4408, 2015.

[49] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1385–1392, 2013.

[50] Wei Zhang and Jana Kosecka. Image based localization in urban environments. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, pages 33–40. IEEE, 2006.

[51] Tinghui Zhou, Philipp Krahenbuhl, Mathieu Aubry, Qixing Huang, and Alexei A Efros. Learning dense correspondence via 3d-guided cycle consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 117–126, 2016.

[52] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *European Conference on Computer Vision*, pages 286–301. Springer, 2016.

Приложение А

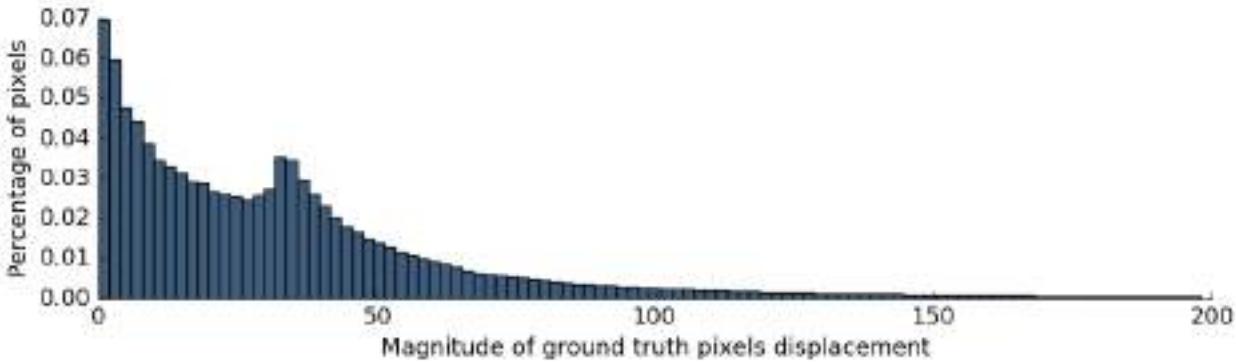
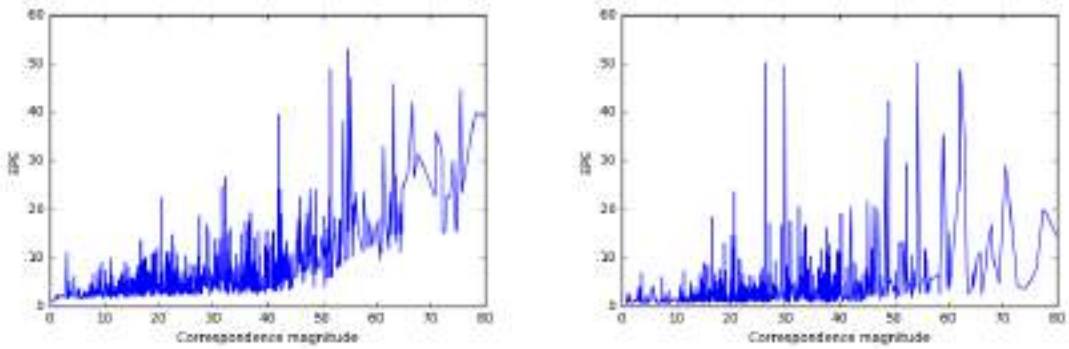
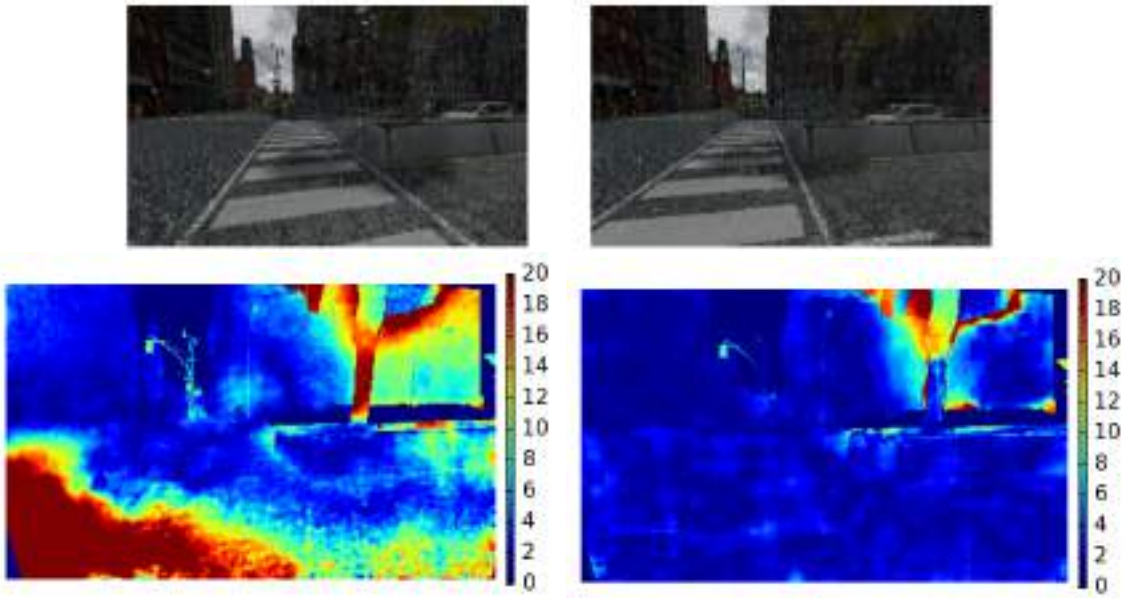


Рис. А-1: Distribution of pixel displacement magnitude on SYNTHIA: fraction of pixels having corresponding ground truth magnitude

FlowMatchNet WarpNet



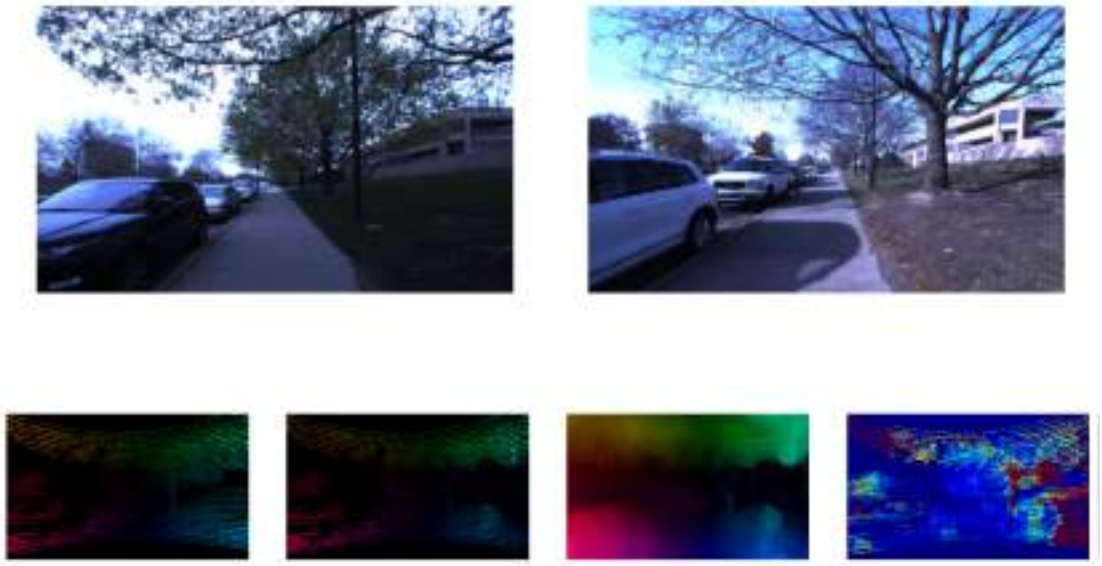
Per-image average EPE dependency on correspondence magnitude. Notice a "fracture"in FlowMatchNet plot, which means that the model can not cope well with large displacements. For WarpNet the plot does not have such a loss pattern, although mean EPE slightly grows all the way from zero to large ground truth displacements. Therefore, coarse-to-fine approach together with warping operations introduce strong regularization to the model, making it grasp large pixels displacement.



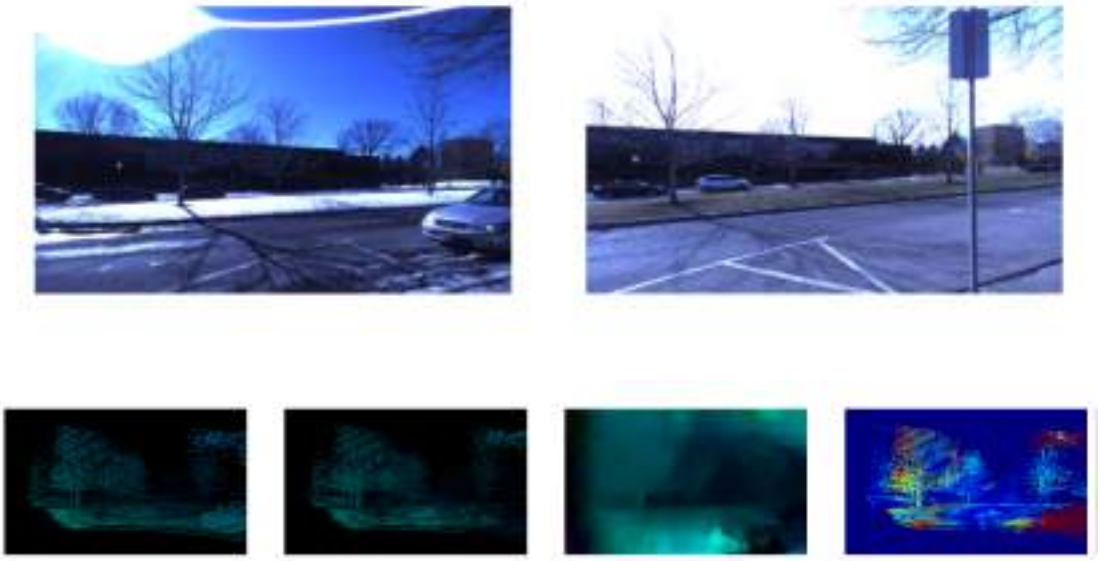
EPE heat maps EPE heat map for SYNTHIA [42] validation sample. WarpNet grasps road pixels better as well as pixels of small or thin objects such as a tree (these objects usually have large displacements as they are closer to camera).

Рис. A-2: Comparison of FlowMatch and WarpNet on SYNTHIA dataset

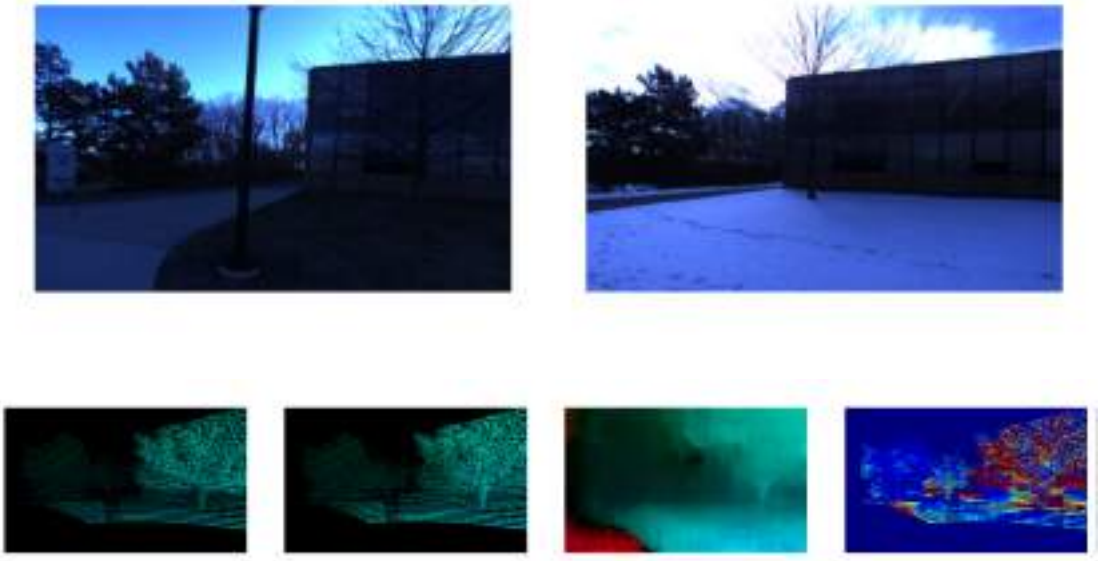
(a)



(b)



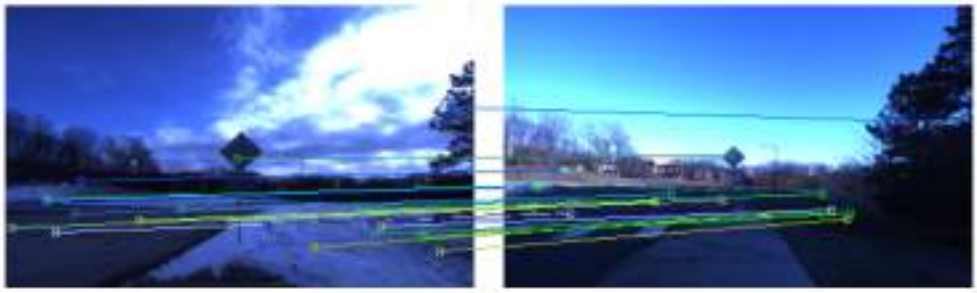
(c)



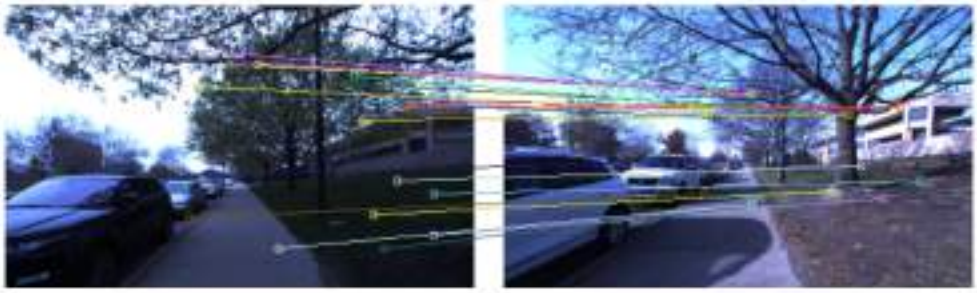
(d)

Рис. A-3: WarpNet predictions on NCLT [10]: first row shows first and second image; second row shows ground truth flow, masked prediction, prediction, endpoint error (EPE) magnitude heat map (left to right). Largest errors are achieved on pixels corresponding to road (poorly represented in training set as lower pixels are not hit by LIDAR), branches (ground truth is imprecise as branches are thin and LIDAR resolution is finite) and moving objects (wrong ground truth).

(a)



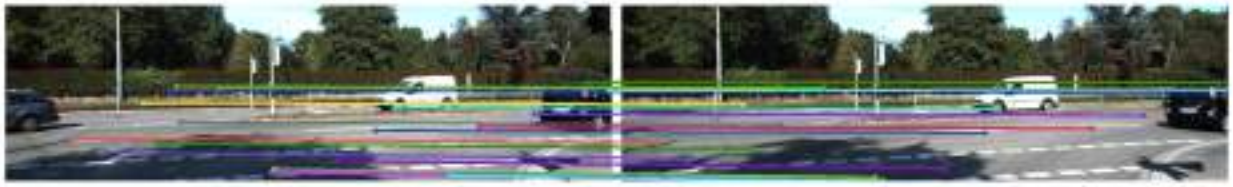
(b)



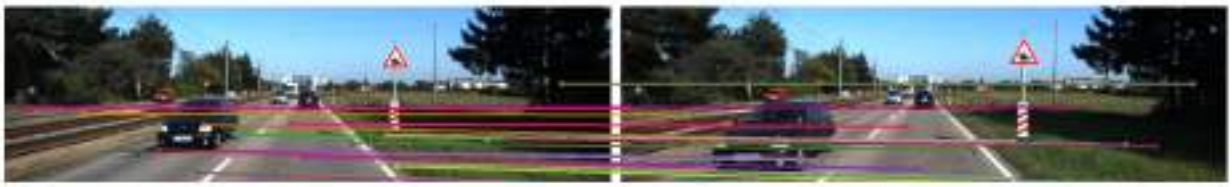
(c)

Рис. A-4: NCLT matches with WarpNet. Notice accurate matches between tree branches in (a), road sign in (b). Ground is matched incorrectly in (c) because of large motion and clutter; branches are still matched well.

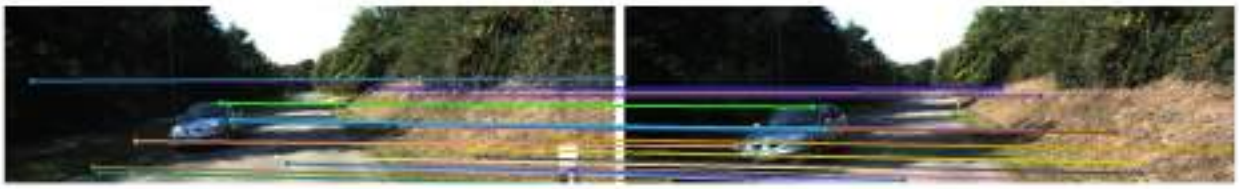
(a)



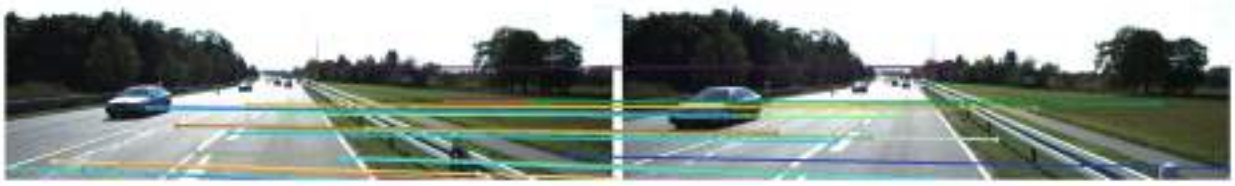
(b)



(c)



(d)



(e)

Рис. A-5: KITTI matches with WarpNet. Moving objects motion is learned successfully (accurate wheel-to-wheel correspondences in (a), (b), (c) and (e), door-to-door and roof-to-roof in (d)). Motion of a road sign, which is relatively small, is predicted accurately in (e)

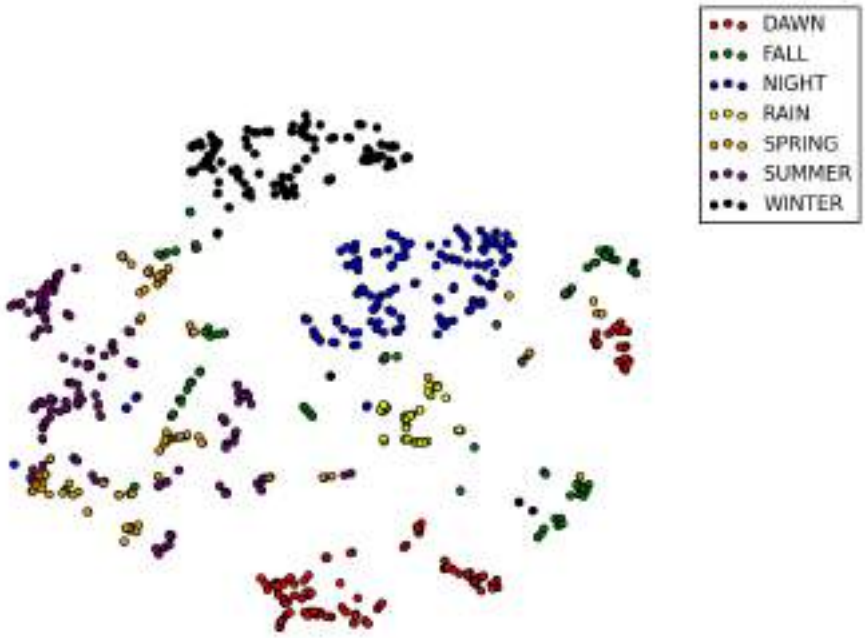
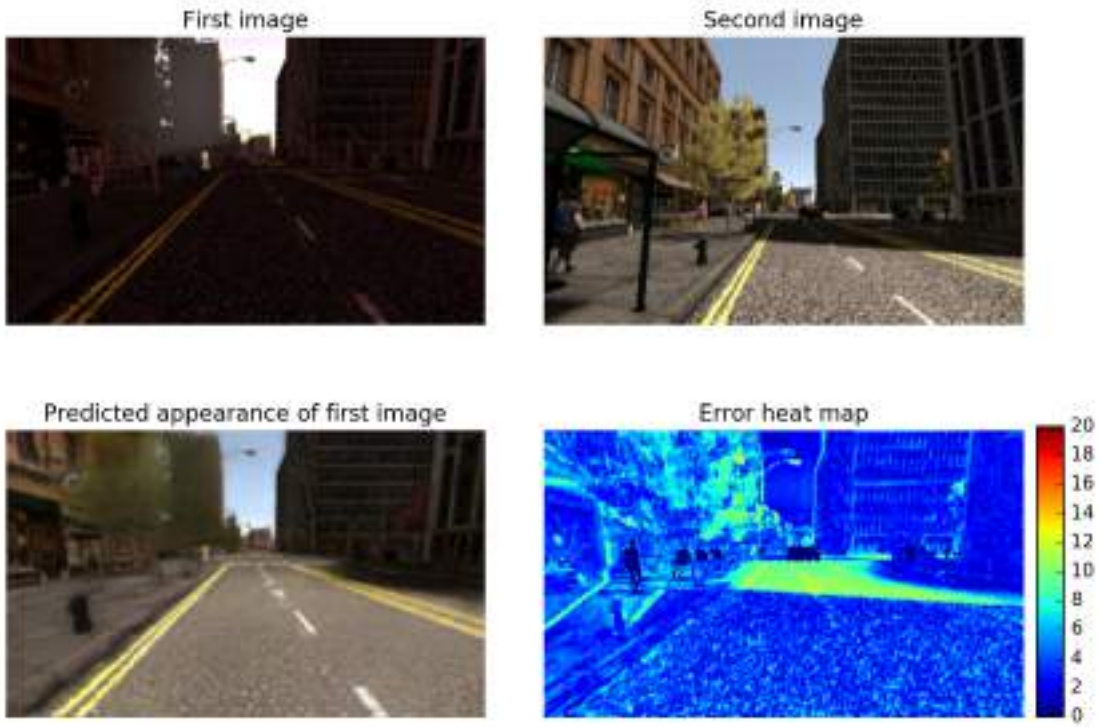
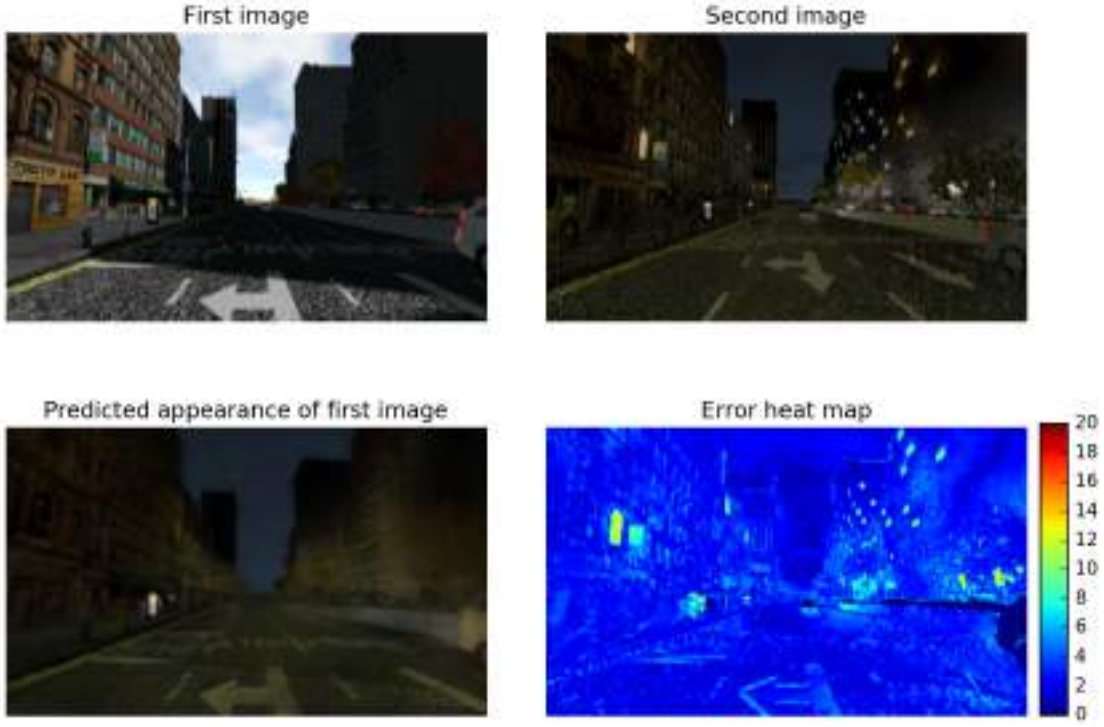


Рис. A-6: t-SNE visualization of conditions vectors on SYNTHIA. Eight-dimensional conditions are extracted from validation samples and projected to 2D. Images labeled as 'dawn', 'winter', 'night', 'rain' form well-separated clouds, while 'summer', 'spring' and 'fall' are mixed (these seasons are usually indistinguishable in urban environment). Although labels were never shown during training, CondNet learns to cluster the set accordingly.



(a)



(b)

Рис. A-7: CondNet results on SYNTHIA [42]. The model learns to remember illumination and weather conditions. Minor details (light in windows, clouds, road roughness) can not be encoded into eight-dimensional vector, so errors are high in corresponding pixels.

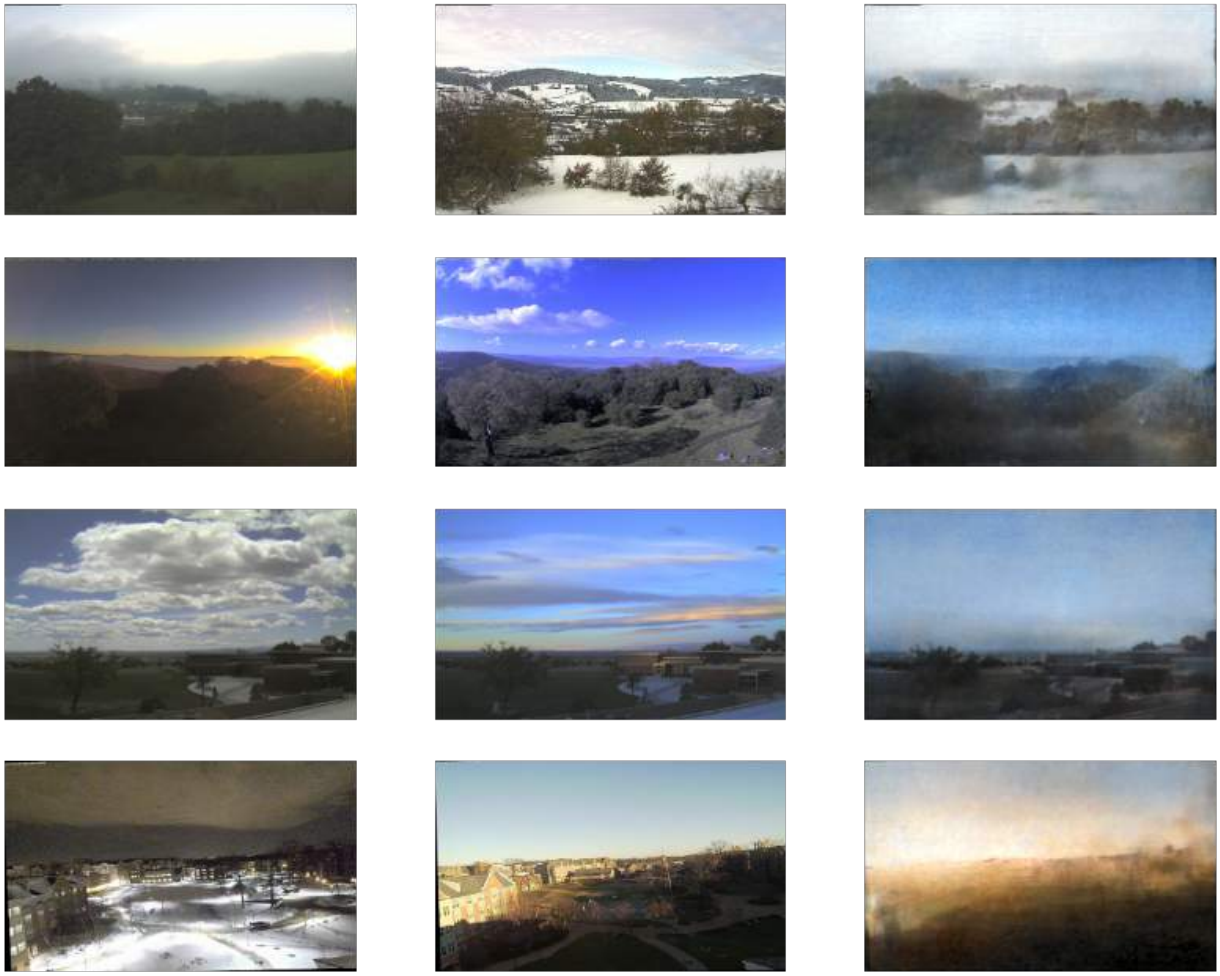
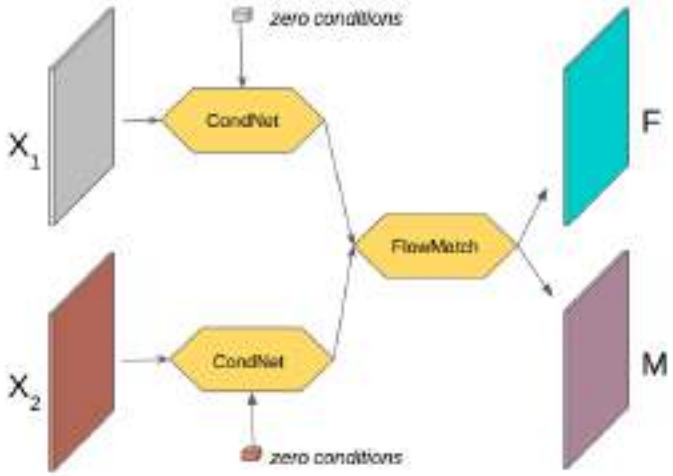
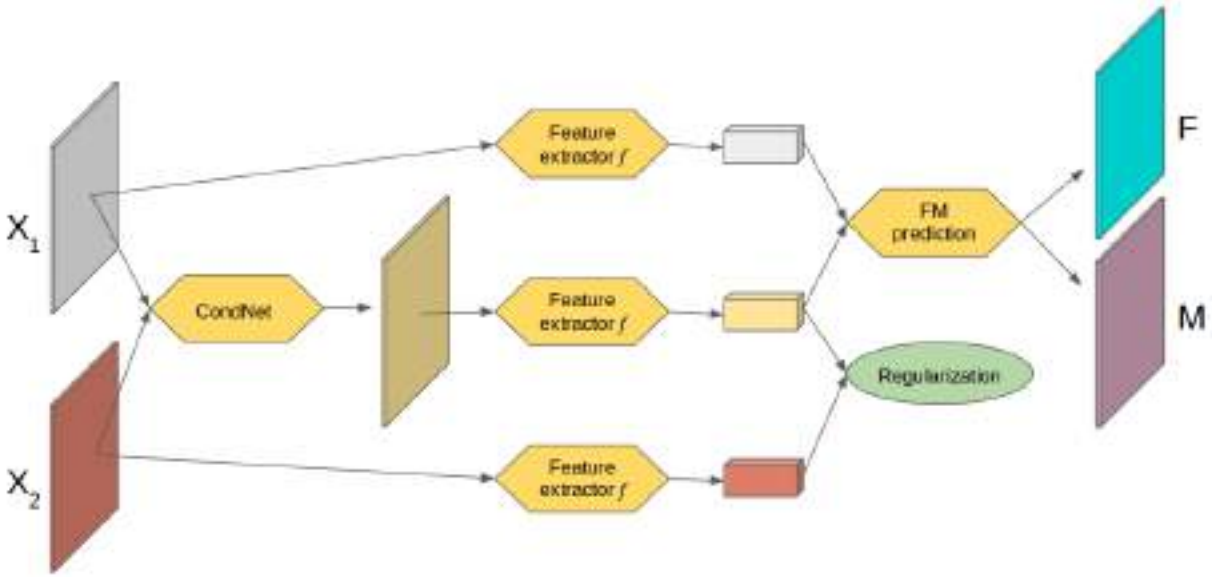


Рис. A-8: CondNet results on Transient Attributes Database [30]. Each row represents one validation sample. First two rows show pairs of images X_1 and X_2. Last row shows predicted images (X_1 under conditions from X_2).



(a) Preliminary processing of both images with CondNet providing. Zero conditions are applied. Input to FlowMatchNet is a pair of synthesized images with zeroed-out environmental conditions.



(b) Injecting CondNet as a regularizer. Features extracted from images should not depend on conditions, but only dependent on scene viewpoint. L_2 penalty is introduced for deviation of $f(X_2)$ from $f(r(X_2, c(X_1)))$.

Рис. A-9: CondNet model in correspondence prediction

Приложение B

B.1 Data Augmentation

One known problem of complex models (in terms of number of parameters) is overfitting to training data. The model is prone to learn not the hidden dependencies between input and output, but rather memorize training samples. Figure B-1 shows an example of fitting a polynomial to noisy data, where both input and target are one dimensional. The model shown in green is perfectly fit to blue training points, having zero error. However, one can judge that less complex model is “more reasonable”. Formally this means lower error on validation samples (all points that lie on blue line and were never seen during training).

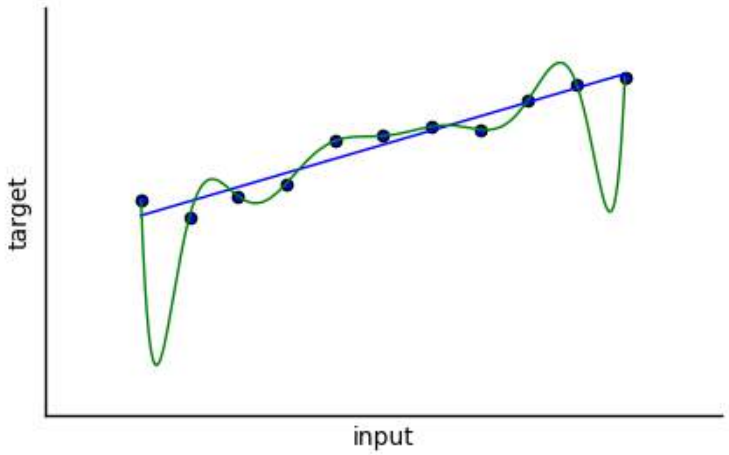


Рис. B-1: Overfitting to data: blue point show noisy observed data, blue line is a hidden target function, green line is a model experiencing overfitting. The model has memorized training samples, but is severely off in intermediate points.

As discussed above, deep neural networks have great number of parameters and are therefore facing overfitting. Various methods are known to decrease this effect and make models more robust. These include applying regularization penalties to model parameters, performing cross-validation, introducing priors in Bayesian inference, making data augmentation etc. Data augmentation in computer vision includes cropping, scaling, rotating images as well as introducing noise, blur and other artificial effects.

In order to augment training data for optical flow tasks, same parameters should be applied when augmenting input images and output flow. Below we are going to show how flow values change when applying an affine transformations to both images. Notice that if the same affine transformation T is applied to both images, then output flow change is described in terms of T:

$$F^{new}(X_1^{new}, X_2^{new}) = grid^{new}(X_2) - grid^{new}(X_1) =$$

$$= T\left(grid(X_2)\right) - T\left(grid(X_1)\right) = T\left(grid(X_2)\right) - grid(X_1)\right) = T\left(F(X_1, X_2)\right),$$

where through $grid(X)$ we denoted tensor that stores (i, j) coordinates for a corresponding pixel of X in feature channels.

On the other hand, it is also possible to introduce different type of augmentation for output flow when applying different transformations to input images. If transformation T_1 is applied to X_1 and T_2 is applied to X_2, then explicit grid processing remains in the equation:

$$F^{new}(X_1^{new}, X_2^{new}) = T_2\left(grid(X_2)\right) - T_1\left(grid(X_1)\right).$$

Notice that in order to apply a transformation to image or flow map, grid construction is still required, so computational complexity of the operation stays the same.

It is convenient to store *grid* tensor in form of matrix $G = (x, y)_{x,y=1,1}^{W,H}$ of size $2 \times WH$. Then, rotation (R), scaling (S) and translation (Tr) can be applied in a matrix form:

$$G^{new} = SRG + Tr = T(G), \tag{B.1}$$

$$T = Tr \circ R \circ S,$$

$$R = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \quad S = \begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \quad Tr = \begin{pmatrix} tr_x & tr_y \end{pmatrix}.$$

Representing grid G in homogeneous coordinates (matrix $3 \times N_{pixels}$), T becomes 2×3 transformation matrix and (B.1) is written as a single matrix multiplication: $T(G) = TG$,

$$T = \begin{pmatrix} s\cos\alpha & -s\sin\alpha & tr_x \\ s\sin\alpha & s\cos\alpha & tr_y \\ 0 & 0 & 1 \end{pmatrix}.$$

In order to obtain a valid image, which has discrete grid of pixels, bilinear interpolation should be done. To eliminate artifacts at the borders, we crop the resulting image.

B.2 Flow as classification problem

Inspired by recent methods in object detection [41, 33] we follow the idea of target space discretization. The space of flow vectors is discretized into a small set of classes and the model learns to predict both class label and adjustment to the fixed flow associated with this class.

The model is again a fully convolutional neural network with two output branches. First branch predicts class labels, having per-pixel softmax activation on $D+1$ neurons of the the last layer, where D is a number of classes the space is split into. One extra neuron accounts for non-matchable class. Second branch predicts adjustments to the flow vectors associated with each class, having $2D$ per-pixel output values.

During training step, class prediction is penalized with cross-entropy loss function, while only true class map is penalized with L_2 loss. This means that the second branch is trained to predict adjustments well only for true class. Those pixels for which correspondence vectors fall under different class do not affect the branch, which allows more exploration inside that class.

$$\mathcal{L}\left(F, M, \hat{R}, \hat{C}\right) = \lambda \mathcal{L}_{corresp}\left(F, M, \hat{R}\right) + \mathcal{L}_{match}\left(F, M, \hat{C}\right),$$

where $\mathcal{L}_{corresp}\left(F, M, \hat{R}\right)$ is an average over matchable pixels as in (3.10) and

$$\ell\left(F_{i,j}, M_{i,j}, \hat{R}_{i,j}\right) = \|F_{i,j} - \hat{R}_{i,j}^{class(F_{i,j}, M_{i,j})}\|.$$

$\mathcal{L}_{match}\left(F, M, \hat{C}\right)$ is a pixel-wise cross-entropy between true $class(F_{i,j}, M_{i,j})$ and $\hat{C}$.

$$class(F_{i,j}, M_{i,j}) = \begin{cases} d : F_{i,j} \in \Delta_d & M_{i,j} = 1 \\ D+1 & M_{i,j} = 0 \end{cases},$$

where subdivision $\mathbf{\Delta} = \{\Delta_d\}_{d=1}^{D}$ is introduced.

FlowClassifNet results Various space subdivision are feasible to discretize 2D correspondence space. In our experiments with FlowClassifNet we split output space of correspondence

vectors in a simple grid of 32 bins according to positions of each pixel in the second image.

Visualization of predicted class labels and resulting correspondence map are given in Figure B-2 for two random samples from validation set. The figure also demonstrates how different the predicted correspondence fields might be even when jointly forming good prediction. Such diversity happens because each of the fields is responsible for only a small fraction of the whole $H \times W$ output and may predict arbitrary values in other regions. However, these inharmonious predictions compensate each other when soft map assignment is performed. Indeed, switching from hard to soft assignment slightly reduces validation endpoint average error on SYNTHIA from 8.7 to 8.1. We use soft assignment in out experiments and put results into Table B.1. FlowClassifNet performs worse than FlowMatchNet, having similar structure and roughly the same number of parameters.

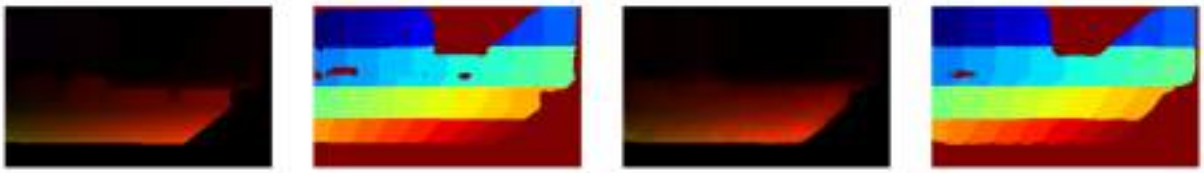
FlowClassifNet has 1×1 kernels on last convolutional layers, which models per-pixel classification in a convolutional fashion. However, in commonly used classification networks last layers encode extracted features in high-dimensional spaces (e.g. 2048 dimensions in [28] and 4096 dimensions in [41]). In our setting, we solve the classification task for $H \times W$ (tens of thousands) pixels at once, which puts limits on memory and time usage. Because of that, we do not go beyond 64 feature channels in our experiments, which might affect results quality.

Inferior performance is also caused by harsh penalizing of those pixels which lie on the border of class region. We To avoid this effect one might try defining soft distribution over classes. However, our experiments didn't show major improvement when defining probability of the pixel to be in class inversely proportional to distance to the centroid of that class.

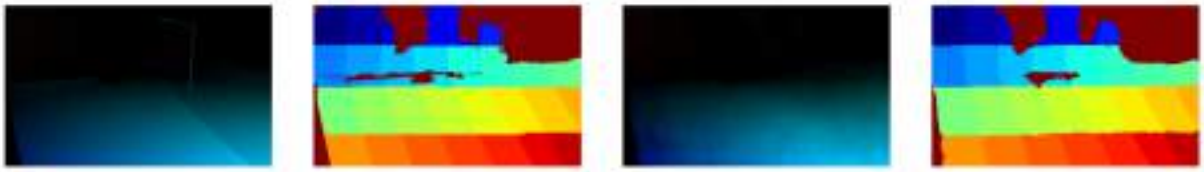
SIFT Flow	FlowMatchNet	WarpNet	FlowClassifNet
16.2	6.0	3.2	8,1

Таблица B.1: Endpoint average error results. FlowClassifNet shows inferior performance.

FlowClassifNet prediction fields grid. Correspondence fields are drastically different, although forming good prediction when voting. Such diversity happens because each of the fields is responsible for only a small fraction of the whole output.



(a)



(b)

FlowClassifNet visualization on SYNTHIA. Left to right: ground truth correspondence field, ground truth class labels, predicted correspondence field (soft voting), predicted class labels

Рис. B-2: FlowClassifNet visualization