Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehr- und Forschungsgebiet für Informatik VIII (Computer Vision)
Prof. Dr. Bastian Leibe

Master Thesis

Pedestrian line counting by probabilistic combination of flow and appearance information

István Sárándi
Matriculation Number: 328707

April 2015

Erstgutachter: **Prof. Dr. Bastian Leibe**
Zweitgutachter: **Prof. Dr. Leif Kobbelt**

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

Aachen, April 16, 2015

(István Sárándi)

Contents

Chapter 1

Introduction

Computer vision is a quickly progressing field which aims at automatically extracting high-level information from visual input, or metaphorically speaking, designing computer systems that can see. It is a particularly challenging task to automatically make sense of videos of dynamic scenes, such as those of outdoor pedestrian crowd behavior. This thesis is about a standard problem formulation in pedestrian video analysis called pedestrian line counting. In this problem, we have a static surveillance camera observing a scene and our goal is to count how many people cross a predefined line in the image over time.

Specifically, we are interested in solving the line counting problem in high-occlusion, densely crowded scenes, where the camera is positioned slightly above the height of the people, but its viewing direction still has a large horizontal component (as opposed to overhead cameras on the top of gates and doors, looking straight downwards).

Such a system can be of interest to several groups of end users. Shop owners may want to count their customers, shopping malls could analyze the dependence between time of day and the pedestrian flow in different areas of the mall. Organizers of events may be interested in the speeds with which different queues at different entrances are progressing. Advertisement companies may want to charge more for billboards where there is more pedestrian traffic. Organizers of demonstrations may want to know how many supporters they had. Local governments could repair and improve pavements with high pedestrian traffic.

We note that anyone who deploys video surveillance systems for monitoring people's behavior has to consider the privacy implications. Unlike in tracking or in recognition, here we are not directly interested in where the individual people are and what they look like, or even what they are doing in particular, we are only concerned with their count. The fact that a computer vision algorithm does not extract or process such high-level information has been presented as an advantage by Chan et al.[CLV08] Senior et al.[SPH+05] have proposed a conceptual framework for handling computer processed surveillance data and classified aggregate statistics, such as the number of people, among the least privacy-sensitive information in their layered model. Further information on balancing intelligent surveillance and privacy can be found in [Ora11],

[FCD+11] and [DV05].

1.1 Related Work

There has been a lot of research interest in crowd analysis in recent years. In this section we outline the main approaches found in the literature.

1.1.1 Line Counting

Several authors have proposed methods for line counting, with different degrees of sophistication. Almost all of them make use the same underlying tools of background subtraction, optical flow, simple features (e.g. number of foreground pixels, edges) and some estimation method (such as regression), but there is a large variety as to how these tools (or steps) are combined and there are many idiosyncratic steps in each. This makes it hard to unify them into a principled survey based on a single conceptual framework. Therefore the following categorization is somewhat fluid.

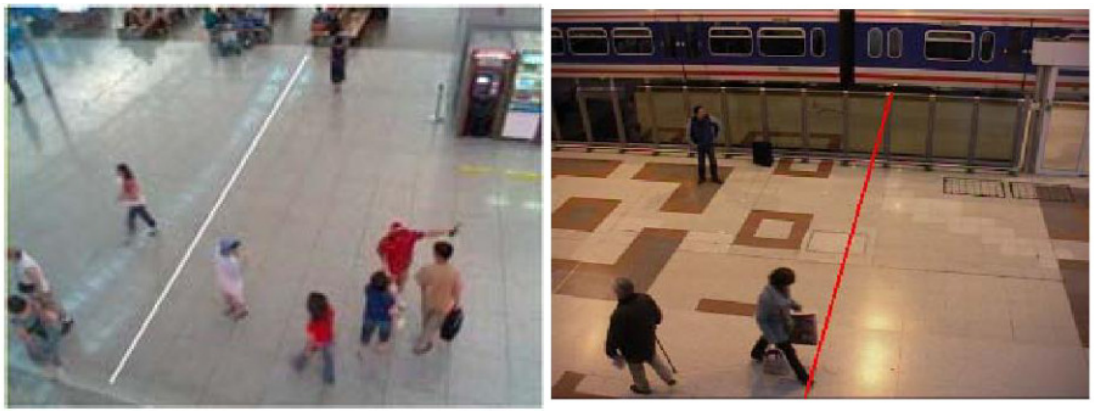

Figure 1.1: Line counting task encountered in [LKK07].

One of the simplest approaches was proposed by Lee et al.[LKK07]. Their method consists of computing a weighted count of foreground pixels that cross the line (or "virtual gate", as they call it, see Figure 1.1). The weights are the product of the optical flow component perpendicular to the line and a perspective normalization factor depending on the position in the image. Finally they multiply this weighted count by a hand-tuned constant factor to get the number of people who have crossed the line.

Using an overhead camera setup, Chen et al.[CCC06] perform tracking on the foreground blobs, and use color histograms to associate the blobs frame by frame. This approach is mainly tailored to scenes where blobs usually contain individual people and only occasionally a few of them. Very dense crowds would be problematic for such a blob-tracking-based approach.

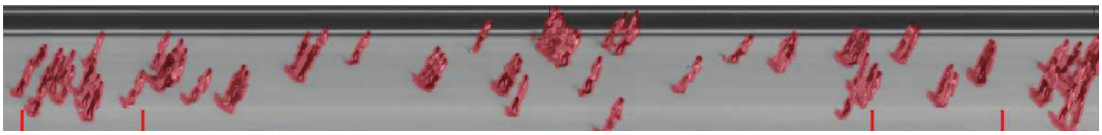

Figure 1.2: A spatiotemporal slice computed by Ma et al.[MC13] on the LHI dataset, with pedestrians highlighted in red.

Several methods make use of the spatiotemporal slicing technique. First of all, we can think of videos as three-dimensional entities with two spatial and one time dimension. In this view, a spatially fixed line-of-interest (LOI) corresponds to a 2D plane in space-time. To create the spatiotemporal slice, we take the pixel data on this plane, in other words we sample the line in each frame and stack these 1 pixel wide samples side-by-side (see Figure 1.2). Such slices are well-known from racing events, where they are used to determine the winner in tight cases. The spatiotemporal slice can be analyzed using the standard computer vision methods of extracting features and applying machine learning methods to create a predictor of the people counts. Broadly speaking, there are blob-based and window-based methods for creating regression samples based on the spatiotemporal slice.

1.1.1.1 Blob-Based Methods

In blob-based methods, the spatiotemporal slice is segmented to pedestrian and non-pedestrian regions and each blob (connected component) of the people segment is used as a sample for learning. The input variables of the regression are features extracted from the blob and the desired output is the ground truth number of people crossings in the blob. Such an approach is used by Yu et al.[YGYB14] They use nothing more than the spatiotemporal slice of the foreground mask (computed by background subtraction). Using a support vector machine (SVM), they classify each blob of the spatiotemporal slice of the mask as either "individual" (single pedestrian) or "crowd" (multiple pedestrians) by shape descriptor features. Finally, they apply mean-shift clustering on each "crowd" blob and estimate the number of people in the blob by the number of clusters found. This relies heavily on the assumption that the blob shapes contain enough information to recognize the separate people in multi-pedestrian blobs.

Cong et al. introduced the flow mosaicking[YHSY09] approach, which is a blob-based method but does not directly work on the spatiotemporal slice. Instead, their main idea is to vary the width of the area taken from each frame, based on the speed of the instantaneous optical flow over the line. They call the resulting blobs flow mosaics, since they are built from different-sized crops of different frames. They describe each blob by a set of features and learn a regression on them. We will come back to this method in Chapter 3, to create a baseline solution.

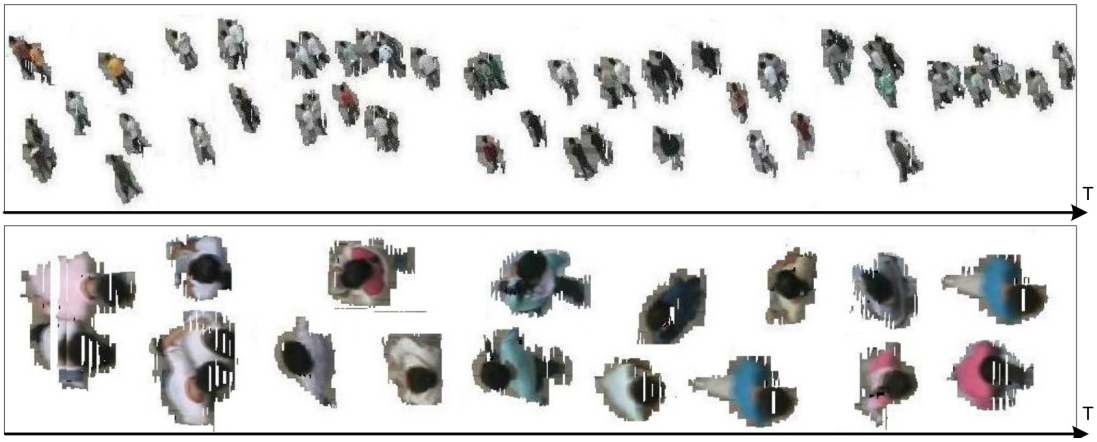

Figure 1.3: The flow mosaicking approach, introduced by Cong et al.[YHSY09] creates a similar image to the spatiotemporal slice, but it does not always stack 1 pixel wide parts of the frames, but varies the width adaptively depending on the flow speed.

1.1.1.2 Window-Based Methods

The other broad category of approaches are the window-based ones. In these, a regression sample is generated for each position of a sliding temporal window over the spatiotemporal slice. An important approach in this category was developed by Ma et al.[MC13]. They segment the crowd to two moving directions by a mixture of dynamic textures method. Then they densely extract local histograms of oriented gradients (local HOGs) from the moving segments. These local HOGs are clustered into visual words. Now, a temporal sliding window of size 238 is used to aggregate the visual words into a histogram (a so called *bag-of-visual-words*) and this histogram is taken as the final feature vector describing the temporal window. To convert from the window estimations to instantaneous estimations, they use an integer programming formulation.

1.1.2 Region Counting

Pedestrian region counting (or ROI counting, from *region-of-interest*) is the task of estimating the number of pedestrians in image regions.

The two main kinds of region counting approaches in the literature are detection-based and regression-based ones. Detection-based methods use a pedestrian detector and count the number of resulting detections. They tend to be computationally complex and only work reliably in moderately occluded scenes.

In regression-based methods, the image is first converted to a feature vector by feature extraction and a regression model is trained that maps from the feature vector to the target variable (pedestrian count). Some authors have used qualitative crowdedness levels ([WLC10] uses 5 levels) instead of directly predicting the number of people.

Figure 1.4: Video recorded from an overhead camera.[BMB08]. Such scenes present considerably less difficulty, due to the uniform size of people as seen from above, and almost no occlusion.

1.1.3 Combination of Multiple Counts

A method which combines the results of multiple counting lines has been proposed by Barandiaran et al.[BMB08]. Their idea is to run blob-based line counting on several parallel lines independently and combine their output by a voting scheme. However, this is used in the context of an overhead camera and not in a sideways scenario, as we would need for our main topic. Indeed, the method depends heavily on the fact that people usually arrive one-by-one, with gaps in-between (see Figure 1.4). Although they devised some simple workarounds for connected groups of people (splitting the group after a certain size threshold), the method is hard to generalize to the sideways looking setup where blobs do not have this simple correspondence to pedestrians.

The idea of combining line counting with region counting has not been investigated in detail yet. Cong et al.[YHSY09] proposed a way to construct a region counter from line counters by summing up the incoming and outgoing pedestrians over the sides of the region. The obvious problem with this is error drift. Without having access to an estimation of the current number of people in the region, small errors made in line counting can quickly accumulate. Their remedy is that they observe the region and every time the foreground mask is empty, they set the region count to zero. Of course this relies on there being such moments when there is no foreground present. This is hardly realistic in a crowded scene. We will investigate a possible combination approach in Chapter 6, where we use a full-fledged region counter to try and keep the errors from piling up.

1.2 Overview and Contributions

In this thesis we propose and implement a new window-based line counting method (based on earlier related work) and evaluate it on a new and more challenging dataset than currently found in the literature. The method is based on regression using low-level features extracted from multiple overlapping blocks in a temporal sliding window over the spatiotemporal slice.

Besides this, we will also consider the relations between region counting and line counting by comparing the output of a region counting method with the counts that we can derive from line counting. For this we describe a region counting method in Chapter 5, mainly based on [CLGX12]. We describe

We also propose new evaluation measures in Chapter 7 that generalize the precision and recall used in information retrieval and binary classification to continuous instantaneous pedestrian flow estimations.

Chapter 2

Preliminaries

In this chapter, we present the general-purpose techniques of background subtraction, optical flow and regression. We will use these in the pedestrian counting algorithm but they are not specific to this application.

2.1 Background Subtraction

Finding objects of interest in images is a difficult vision problem in the general case. If, however, we have a video from a static camera setup, background subtraction (also known as figure-ground segmentation) techniques can help us find surprising or unexpected-looking regions that hopefully correspond to semantically interesting objects in the scene. The *foreground mask* for a given frame of the video is a binary mask in which each pixel indicates whether the pixel is currently part of a dynamic foreground object or a (fairly) static background (see Figure 2.1).

Computing such a mask can be a valuable intermediate step in vision algorithms, as it allows us to restrict some of the further processing steps to the foreground region. This, of course, assumes that the distinction between an interesting foreground and a boring background is meaningful for our particular high-level problem. The pedestrian counting task lends itself easily to such a distinction, since the moving people usually stand out from the background.

A useful survey on different background subtraction approaches can be found in [TKBM99] and [Elg11].

The simplest way to obtain a foreground mask is to compute the thresholded absolute difference between the current frame and a fixed background image I_{ref} that contains no foreground objects.

$$F_t(x,y) = \begin{cases} 1, & \text{if } \left|I_t(x,y) - I_{ref}(x,y)\right| > \tau \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

where F_t is the foreground mask at time t, I_t is the frame and τ is a threshold value.

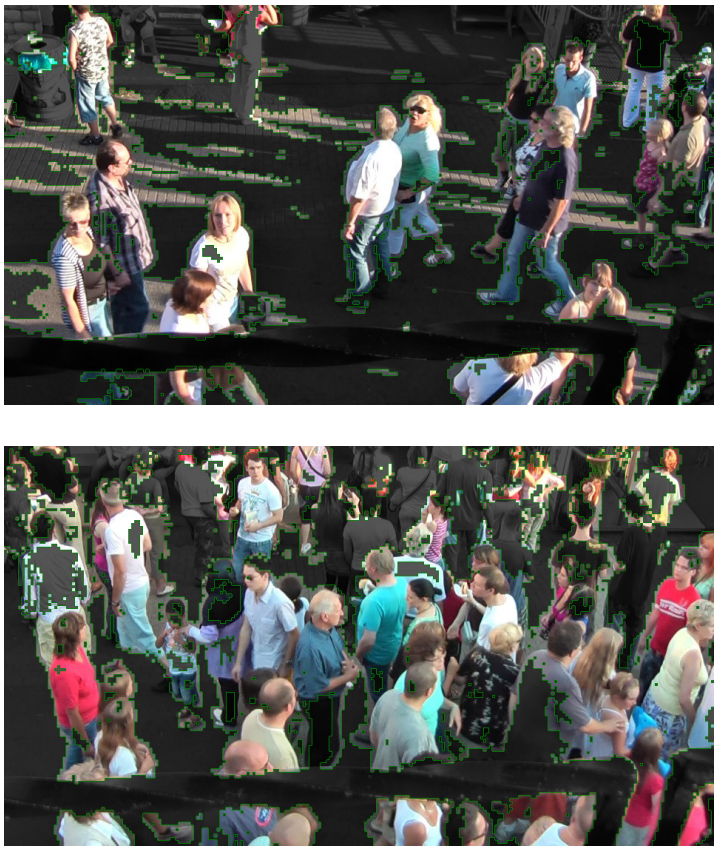

Figure 2.1: Examples of background subtraction results. The highlighted area is detected as foreground (also marked by green outline). In the top image, we can see some falsely detected bright sunny spots and at the top of the bottom image we can see non-moving people blended into the background.

Unfortunately in practice the background is not perfectly stationary and may itself change over time. Drawing the precise line between irrelevant changes in the background as opposed to the appearance of interesting foreground objects can be difficult and we need to make trade-offs. In outdoor scenes, the following changes can be considered uninteresting:

- *Illumination changes:* The intensity and color of pixels can change slowly over the course of the day and may suddenly change when clouds occlude the sun. Strong shadows may appear under direct sunlight.
- *Small movements:* Background objects such as flags or tree branches in the wind may make small movements on the order of a few pixels. The camera may also shake slightly due to the wind, moving the whole image by a small offset.
- *Background objects introduced or removed:* There may be some uninteresting objects that occasionally get repositioned in the background. The new position

of the object (and the "hole" at the old position) should eventually be accepted into the background.

Advanced background subtraction techniques can tolerate such changes to a certain degree, by using adaptive models that get adjusted with each new frame.

2.1.1 Probabilistic Methods

Most of the adaptive background models can be described probabilistically as an attempt to approximate the posterior probability of a pixel being background given the current frame and all previous frames.

$$F_t(x,y) = \begin{cases} 1, & \text{if } P\left(B_{t,x,y} \middle| I_t, I_{t-1}, ..., I_1\right) < \tau \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

where $B_{t,x,y}$ is the event (in probability terminology) that the pixel at position (x,y) at time t is part of the background. This probability can be represented using an incrementally updated model M_t and a feature extraction function Φ that computes local features such as gradients, intensity or color features:

$$F_t(x,y) = \begin{cases} 1, & \text{if } P\left(B_{t,x,y} \middle| \Phi(I_t)(x,y), M_{t-1}\right) < \tau \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

$$M_0 = initialModel \tag{2.4}$$

$$M_t = updateModel(M_{t-1}, \Phi(I_t)) \tag{2.5}$$

In practice, the posterior probability is often replaced by the likelihood, i.e.

$$F_t(x,y) = \begin{cases} 1, & \text{if } P\left(\Phi(I_t)(x,y) \middle| B_{t,x,y}, M_{t-1}\right) < \tau' \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

This formulation is equivalent to the posterior-based approach under the following assumptions:

- The features at the foreground pixels have a uniform distribution:

$$P\left(\Phi(I_t)(x,y) \middle| \neg B_{t,x,y}, M_{t-1}\right) \equiv c \tag{2.7}$$

- Whether a pixel at time t is background or not is independent of time, position and observations up to time $t-1$:

$$P\left(B_{t,x,y} \middle| M_{t-1}\right) \equiv b \tag{2.8}$$

Indeed, in this case the posterior $s(t,x,y)$ and the likelihood $l(t,x,y)$ are in a fixed monotonic functional relationship:

$$s(t,x,y) = P\left(B_{t,x,y}\middle|\Phi(I_t)(x,y), M_{t-1}\right) = \tag{2.9}$$

$$= \frac{P\left(\Phi(I_t)(x,y)\middle|B_{t,x,y}, M_{t-1}\right) P\left(B_{t,x,y}\middle|M_{t-1}\right) P\left(M_{t-1}\right)}{num + P\left(\Phi(I_t)(x,y)\middle|\neg B_{t,x,y}, M_{t-1}\right) P\left(\neg B_{t,x,y}\middle|M_{t-1}\right) P\left(M_{t-1}\right)} = \tag{2.10}$$

$$= \frac{l(t,x,y)\cdot b}{l(t,x,y)\cdot b + c\cdot(1-b)} \tag{2.11}$$

$$l(t,x,y) \propto \frac{s(t,x,y)}{1-s(t,x,y)}, \tag{2.12}$$

where *num* refers to the numerator of the fraction that it is part of.

Therefore, thresholding the posterior is equivalent to thresholding the likelihood if the above assumptions are met.

To apply such probabilistic background subtraction techniques, we need to choose the functional form of the likelihood, the feature mapping Φ, and we need to specify how to update the model incrementally. There are two main families of likelihood representations to choose from: parametric and nonparametric models. Essentially, parametric models keep the "size" of the stored model M_t fixed and independent of the number of samples that have been observed. By contrast, nonparametric models grow with each new observation and the distribution is not assumed to have any particular form.

An example of a simple parametric model for the likelihood is a Gaussian (normal) distribution at each pixel. Practice shows, however, that the likelihood is usually multimodal. Gaussian Mixture Models can help to a certain extent but it is hard to justify any apriori choice for the number of mixture components.

2.1.1.1 Kernel Density Estimation

The most prominent non-parametric model for background subtraction is kernel density estimation (KDE). This is a general technique for estimating a continuous density function $p(x)$ from discrete observations $x_1,\ldots,x_t$. The estimation $\hat{p}$ is given by the formula

$$\hat{p}(x) = \frac{1}{N}\sum_{i=0}^{t} k(x_i,x) \tag{2.13}$$

where $k(x,y)$ is a so called kernel function, that is non-negative and $\int_{-\infty}^{\infty} k(x,y)dx = 1,\ \forall\, y$. The kernel function usually has radial symmetry, with the Gaussian kernel being the most common:

$$k_{Gauss}(x,y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-y)^\top (x-y)}{2\sigma^2}\right) \tag{2.14}$$

The formula 2.13 can also be interpreted as a mixture distribution, where each previous sample has equal probability of generating a noisy copy of itself, the kernel function describing the noise distribution. (Note: These kernel functions should not be confused with the kernel used in machine learning (kernel ridge regression, kernel SVM etc.)

In the background subtraction case we would like adapt to the changing background conditions therefore we want our samples to concentrate on recent times. One method is to simply keep the T most recent samples (i.e. use a first-in-first-out approach). This estimation in our previous notation becomes:

$$P\left(\Phi(I_t)(x,y)\middle|B_{t,x,y}, M_{t-1}\right) := \frac{1}{\min(T, t-1)} \sum_{i=\max(t-T,0)}^{t-1} k(\Phi(I_i)(x,y), \Phi(I_t)(x,y)) \tag{2.15}$$

and the model data M_t is simply the sets of the last T previously observed features at every pixel.

2.1.1.2 Exponential Forgetting By Random Replacement

The first-in-first-out sample handling explained above has the disadvantage that all information about pixels we saw before T frames ago gets forgotten. This is unfortunate if objects (or people) move and then stand still for some time. In these cases they blend into the background, which is expected and understandable. However, when they leave, the original background is now detected as foreground, since we have no samples of the original background any more. Therefore, it is beneficial to retain some of the older samples as well.

It is very simple to modify the sample handling such that the system remembers past samples with exponentially decaying probability $e^{-\alpha t}$, where t denotes the sample's age. For this, we simply discard a random previous sample at every sample insertion (with uniform distribution). From the point of view of an individual sample, surviving t insertion rounds has probability $\left(1-\frac{1}{T}\right)^t = e^{-\alpha t}$, with $\alpha = \log T - \log(T-1)$, so it is truly an exponentially forgetting setup.

2.1.1.3 Pixel Features

The pixel features $\Phi(I_t)$ can be anything that we can expect to discriminate well between foreground and background. We choose color and gradients as features. Color is represented in HSV (hue, saturation, value) color space, which has several advantages over e.g. RGB space. For one, the value (brightness) component can be treated

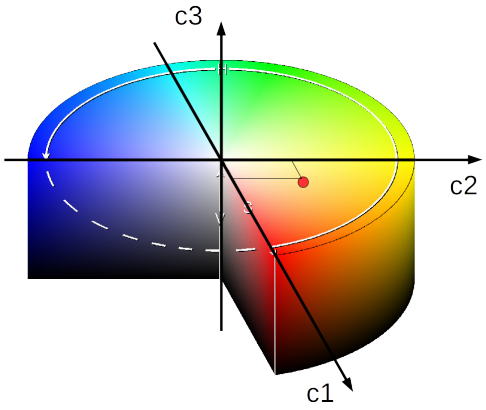

Figure 2.2: The HSV color space is represented as a cylinder and the Cartesian coordinates c_1, c_2, c_3 used as color features.

separately, which can help suppress false foreground detections in shadows or other illumination changes. Furthermore, the hue and saturation components correspond to perceptual color qualities that are useful when quantifying color differences.

Since the hue and saturation components of the HSV space are polar coordinates, and the hue wraps back to itself after 360 degrees, we cannot straightforwardly use these coordinates with the Gaussian kernel, which computes squared differences. Instead, we represent the HSV space as a cylinder and use the Cartesian coordinates of the points in the cylinder as our features (see Figure 2.2). More precisely, the color is represented by the following values:

$$c_1 = S \cdot \cos(H) \tag{2.16}$$
$$c_2 = S \cdot \sin(H) \tag{2.17}$$
$$c_3 = V \tag{2.18}$$

The two gradient features are obtained applying a vertically and a horizontally oriented Sobel filter.

2.1.1.4 Combating Small Shifts

Small shifts of background pixels may happen due to the camera shaking slightly or some objects moving (perhaps in the wind). To avoid false foreground detection in these cases, we evaluate the likelihood of a new sample $\Phi(I_t)(x,y)$ being background against every neighboring pixel's background model as well, and then we take the maximum value as our probability estimate. This means that if a background object is shifted by only one pixel, it will still be recognized as background, even if the position where it moved to has no similar samples stored. Elgammal et al. describe this method in [EHD00], and even propose further refinements. In our experience this method already works sufficiently well.

From a theoretical point of view, we should notice that taking the maximum over several probability distributions will lead to a "distribution" whose integral (over all possible pixel features) is larger than one. However, we have to remember that our real aim is not to model the likelihood of the features under the background model, but to give the probability of a pixel being background given the observations. Therefore, we can interpret the larger-than-one integrals as simply setting higher prior probabilities for the backgroundness of pixels, whose neighbors have highly non-overlapping background models. Specifically, using our previous notation, this effect corresponds to a non-constant $P\left(B_{t,x,y}\middle|M_{t-1}\right)$ probability (in Equation 2.10). In other words, the past observations can make us anticipate that some parts of the image are more likely to be background in the next frame than other parts, even before seeing that next frame.

2.2 Line Optical Flow

In the pedestrian line counting problem we are concerned about the movement of people. A useful way to capture this is by optical flow, which is a standard task in computer vision. Given an image sequence, a solution to the optical flow problem is a sequence of vector fields (called *flow fields*), each describing the movement between one frame and the next. Ideally the flow field should correspond to the actual movement vectors of the real scene objects as projected onto the image plane. There are several algorithms for generating such a flow field.

2.2.1 Problem Formulation

In our task, we only need to compute the flows over a one-dimensional line (see examples in Figure 2.3. Based on the work of Horn and Schunk[HS81] and Brox et al.[BPW04], in the following we formulate the one-dimensional dense optical flow problem as a variational optimization problem and give the equations for solving it.

Our aim is to find the optical flow over a line L connecting the points $\mathbf{p}_1$ and $\mathbf{p}_2$ in the image[i]. Let its length be $\ell = \|\mathbf{p}_2 - \mathbf{p}_1\|$, its direction $\mathbf{d} = \frac{\mathbf{p}_2 - \mathbf{p}_1}{\ell}$. Its points can then be represented by the function[ii]

$$\mathbf{p}(\cdot) : [0, \ell] \to \mathbb{R}^2 \quad s \mapsto \mathbf{p}_1 + s \cdot \mathbf{d} \tag{2.19}$$

The optical flow over the line is a vector function $\mathbf{f}(\cdot) : [0, \ell] \to \mathbb{R}^2$. Its two components can also be denoted as $\mathbf{f} = (u\ v)^\top$. To quantify how good an optical flow field is we will use the following energy functional:

[i] Vectors are denoted by boldface.

[ii] In this chapter, when referring to a function as a mathematical object itself, we use the $f(\cdot)$ notation. Otherwise a lone f refers to the function's output for some unspecified or implied input.

Figure 2.3: Examples of line optical flow solutions.

$$J[\mathbf{f}(\cdot)] = \int_0^\ell \mathcal{L}(\mathbf{p}(s), \mathbf{f}(s), \mathbf{f}'(s))\, ds \tag{2.20}$$

$$= \int_0^\ell \Psi\big(E^2_{data}(\mathbf{p}(s), \mathbf{f}(s))\big) + \alpha \cdot \Psi\big(\left\|\mathbf{f}'(s)\right\|^2\big) + \beta \cdot \Psi\big(\|\mathbf{f}(s)\|^2\big)\, ds, \tag{2.21}$$

where $\Psi(x) = \sqrt{x+\varepsilon^2}$, $\varepsilon = 10^{-4}$.

In this formula, E_{data} describes how well the flow field fits to the image contents (pixel value constancy). The second term encourages a smooth solution by penalizing the magnitude of the derivative of the flow along the line. The third term penalizes the magnitude of the flow itself, so it suppresses unnecessarily long flow vectors. The α and β values are hand-tuned constants specifying the strength of these penalties.

The penalizer $\Psi(x)$ applied to squared arguments approximates the absolute value function $\Psi(x^2) \approx |x|$, while being differentiable everywhere and therefore easier to handle in optimization. This kind of error norm is called a TV-L1 (total variation) regularizer and can yield robust estimations. Compared to the more usual and straight-

forward squared norm, Ψ punishes large values more gracefully and it is less prone to prefer an overall bad solution for the sake of a few outliers.

A simple choice for defining the data fit term E_{data} is to take the difference to the warped next frame:

$$E_{data}(\mathbf{p},\mathbf{f}) = I(\mathbf{p}+\mathbf{f}, t+1) - I(\mathbf{p},t) \tag{2.22}$$

However, we may have multiple channels in our images. These can be color channels (such as RGB), gradients or anything else that we can assume to stay constant with movement. In this case we can treat $\mathbf{I}(\cdot)$ as a vector function and use the Euclidean distance instead of the difference:

$$E_{data}(\mathbf{p},\mathbf{f}) = \|\mathbf{I}(\mathbf{p}+\mathbf{f}, t+1) - \mathbf{I}(\mathbf{p},t)\| = \sqrt{\sum_{c=1}^{C} \left(I_c(\mathbf{p}+\mathbf{f}, t+1) - I_c(\mathbf{p},t)\right)^2}, \tag{2.23}$$

where C is the number of channels.

A necessary condition for a minimizer of the energy functional is that it must fulfill the Euler-Lagrange equations along the whole line:

$$\begin{aligned} 0 &= \frac{\partial}{\partial u}\mathcal{L} - \frac{\partial}{\partial s}\frac{\partial}{\partial u'}\mathcal{L} \\ 0 &= \frac{\partial}{\partial v}\mathcal{L} - \frac{\partial}{\partial s}\frac{\partial}{\partial v'}\mathcal{L} \end{aligned} \tag{2.24}$$

Or with vector notation

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{f}}\mathcal{L} - \frac{\partial}{\partial s}\frac{\partial}{\partial \mathbf{f}'}\mathcal{L} \tag{2.25}$$

This expands to

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{f}}\Psi(E_{data}^2(\mathbf{f})) - \alpha\frac{\partial}{\partial s}\frac{\partial}{\partial \mathbf{f}'}\Psi\left(\left\|\mathbf{f}'\right\|^2\right) + \beta\frac{\partial}{\partial \mathbf{f}}\Psi\left(\|\mathbf{f}\|^2\right) \tag{2.26}$$

Expanding the vector norms for the first equation we get:

$$0 = \frac{\partial}{\partial u}\Psi(E_{data}^2(u,v)) - \alpha\frac{\partial}{\partial s}\frac{\partial}{\partial u'}\Psi\left((u')^2 + (v')^2\right) + \beta\frac{\partial}{\partial u}\Psi\left(u^2+v^2\right) \tag{2.27}$$

Applying the chain rule of differentiation:

$$0 = \Psi'(E_{data}^2(u,v)) \cdot 2E_{data}(u,v) \cdot \frac{\partial}{\partial u}E_{data}(u,v) \tag{2.28}$$

$$- \alpha \cdot \frac{\partial}{\partial s}\left(\Psi'\left((u')^2 + (v')^2\right) \cdot 2u'\right) \tag{2.29}$$

$$+ \beta \cdot \Psi'\left(u^2+v^2\right) \cdot 2u \tag{2.30}$$

The derivative of Ψ is $\Psi'(x) = \frac{1}{\sqrt{x+\varepsilon^2}}$.
Now let us introduce the following notations:

$$\psi_{data}(u,v) := \Psi'(E_{data}^2(u,v)) = \frac{1}{\sqrt{E_{data}^2(u,v)+\varepsilon^2}} \tag{2.31}$$

$$\psi_{norm}(u,v) := \Psi'\left(u^2+v^2\right) = \frac{1}{\sqrt{u^2+v^2+\varepsilon^2}} \tag{2.32}$$

Finally, we can rewrite Equation 2.28 with these new notations and return back to vector notation (also dividing by 2):

$$\mathbf{0} = \psi_{data}(\mathbf{f}) \cdot E_{data}(\mathbf{f}) \cdot \frac{\partial}{\partial \mathbf{f}} E_{data}(\mathbf{f}) - \alpha \cdot \frac{\partial}{\partial s}\left(\psi_{norm}(\mathbf{f}') \cdot \mathbf{f}'\right) + \beta \cdot \psi_{norm}(\mathbf{f}) \cdot \mathbf{f} \tag{2.33}$$

We now have a non-linear equation system.

2.2.2 Numerical Solution

We solve this by an iterative method. Suppose we are after iteration k, which resulted in the flow function $\mathbf{f}^{(k)}(\cdot) = \begin{pmatrix} u^{(k)}(\cdot) \\ v^{(k)}(\cdot) \end{pmatrix}$. We now want to find a good $\mathbf{f}^{(k+1)}(\cdot)$ for the result of the next iteration. We first linearize the data term around the previous solution. Let $\delta\mathbf{f}^{(k)} := \mathbf{f}^{(k+1)} - \mathbf{f}^{(k)}$.

$$E_{data}\left(\mathbf{f}^{(k+1)}\right) = E_{data}\left(\mathbf{f}^{(k)} + \delta\mathbf{f}^{(k)}\right) \approx E_{data}\left(\mathbf{f}^{(k)}\right) + \frac{\partial}{\partial \mathbf{f}} E_{data}(\mathbf{f})\Big|_{\mathbf{f}=\mathbf{f}^{(k)}} \cdot \delta\mathbf{f}^{(k)} \tag{2.34}$$

(Note: The location s along the line is omitted for readability, but it is always implied. These equations apply along the whole line.)

If we only use one image channel, the derivative of E_{data} is simply the warped gradient in the next frame:

$$\frac{\partial}{\partial \mathbf{f}} E_{data}(\mathbf{f}) = \frac{\partial}{\partial \mathbf{f}}\Big(I(\mathbf{p}+\mathbf{f}, t+1) - I(\mathbf{p}, t)\Big) = \frac{\partial}{\partial \mathbf{x}} I(\mathbf{x}, t+1)\Big|_{\mathbf{x}=\mathbf{p}+\mathbf{f}}, \tag{2.35}$$

Using the multichannel E_{data} formula, this derivative becomes a more complicated expression but it is still simple to compute:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{f}} E_{data}(\mathbf{f}) &= \frac{\partial}{\partial \mathbf{f}} \left\|\mathbf{I}(\mathbf{p}+\mathbf{f}\cdot\Delta t, t+\Delta t) - \mathbf{I}(\mathbf{p}, t)\right\| \\ &= \frac{\sum_{c=1}^{C} 2\Big(I_c(\mathbf{p}+\mathbf{f}, t+1) - I_c(\mathbf{p}, t)\Big) \cdot \frac{\partial}{\partial \mathbf{x}} I_c(\mathbf{x}, t+1)\Big|_{\mathbf{x}=\mathbf{p}+\mathbf{f}}}{E_{data}(\mathbf{f})} \end{aligned} \tag{2.36}$$

Let us introduce some more notation:

$$I_{diff}^{(k)} := E_{data}\left(\mathbf{f}^{(k)}\right) \tag{2.37}$$

$$\mathbf{I}_x^{(k)} := \frac{\partial}{\partial \mathbf{f}} E_{data}(\mathbf{f})\Big|_{\mathbf{f}=\mathbf{f}^{(k)}} \tag{2.38}$$

Now we can rewrite Equation 2.33 by adding the iteration indices and plugging in the linearized data term.

$$\begin{aligned}\mathbf{0} =& \psi_{data}\left(\mathbf{f}^{(k+1)}\right) \cdot \left(I_{diff}^{(k)} + \mathbf{I}_x^{(k)} \delta\mathbf{f}^{(k)}\right) \cdot \mathbf{I}_x^{(k)} \\ & - \alpha \cdot \frac{\partial}{\partial s}\left(\psi_{norm}\left(\mathbf{f}'^{(k+1)}\right) \cdot \mathbf{f}'^{(k+1)}\right) \\ & + \beta \cdot \psi_{norm}\left(\mathbf{f}^{(k+1)}\right) \cdot \mathbf{f}^{(k+1)}\end{aligned} \tag{2.39}$$

The second term contains derivatives along the line (w.r.t. s). These derivatives are approximated numerically by half-step central differences: $y'(x) \approx y(x+1/2) - y(x-1/2)$.

For this part we shall consider the position s explicitly and introduce:

$$\psi_{sm}^{(k+1)}(s) = \psi_{norm}\left(\mathbf{f}'^{(k+1)}(s)\right), \tag{2.40}$$

where with a slight abuse of notation we write $\mathbf{f}(s)$ instead of $\mathbf{f}(\mathbf{p}(s))$ for clarity. The second term in Equation 2.39 becomes (disregarding the α factor and with the $^{(k+1)}$ superscripts implied):

$$\begin{aligned}&\frac{\partial}{\partial s}(\psi_{sm}(s) \cdot \mathbf{f}'(s)) = \\ &= \psi_{sm}(s+1/2) \cdot \mathbf{f}'(s+1/2) - \psi_{sm}(s-1/2) \cdot \mathbf{f}'(s-1/2) \\ &= \psi_{sm}(s+1/2) \cdot (\mathbf{f}(s+1) - u(s)) - \psi_{sm}(s-1/2) \cdot (\mathbf{f}(s) - u(s-1)) \\ &= \psi_{sm}(s+1/2) \cdot (\mathbf{f}(s+1) - u(s)) + \psi_{sm}(s-1/2) \cdot (\mathbf{f}(s-1) - \mathbf{f}(s))\end{aligned} \tag{2.41}$$

Now let us reintroduce the superscripts, because we will make use of the fact that $\mathbf{f}^{(k+1)} = \mathbf{f}^{(k)} + \delta\mathbf{f}^{(k)}$.

$$
\begin{aligned}
\ldots = \quad & \psi_{sm}^{(k+1)}(s+1/2)\cdot(\mathbf{f}^{(k+1)}(s+1)-\mathbf{f}^{(k)}(s)-\delta\mathbf{f}^{(k)}(s)) \\
& +\psi_{sm}^{(k+1)}(s-1/2)\cdot(\mathbf{f}^{(k+1)}(s-1)-\mathbf{f}^{(k)}(s)-\delta\mathbf{f}^{(k)}(s)) \\
= \quad & \psi_{sm}^{(k+1)}(s+1/2)\cdot(\mathbf{f}^{(k+1)}(s+1)-\mathbf{f}^{(k)}(s)) \\
& +\psi_{sm}^{(k+1)}(s-1/2)\cdot(\mathbf{f}^{(k+1)}(s-1)-\mathbf{f}^{(k)}(s)) \\
& -\left(\psi_{sm}^{(k+1)}(s+1/2)+\psi_{sm}^{(k+1)}(s-1/2)\right)\cdot\delta\mathbf{f}^{(k)}(s) \\
= & \sum_{i\in\{-1,+1\}}\psi_{sm}^{k+1}\left(s+\frac{i}{2}\right)\cdot\left(\mathbf{f}^{(k+1)}(s+i)+\delta\mathbf{f}^{(k)}(s+i)-\mathbf{f}^{(k)}(s)\right) \\
& -\left(\sum_{i\in\{-1,+1\}}\psi_{sm}^{k+1}\left(s+\frac{i}{2}\right)\right)\cdot\delta\mathbf{f}^{(k)}(s)
\end{aligned}
\tag{2.42}
$$

These ψ_{sm} terms are also approximated by central differences:

$$
\psi_{sm}(s+1/2)=\frac{1}{\sqrt{\|\mathbf{f}'(s+1/2)\|^2+\varepsilon^2}}=\frac{1}{\sqrt{\|\mathbf{f}(s+1)-\mathbf{f}(s)\|^2+\varepsilon^2}} \tag{2.43}
$$

Our system is still nonlinear in $\delta\mathbf{f}^{(k)}$ because of ψ_{data}, ψ_{sm} and ψ_{norm}. In order to get a linear system, we introduce a nested iterative process with iteration index l. In the inside of this iterative process we "constantize" (approximate as constant function, i.e. keep fixed) the ψ values, so the equation system becomes linear. Let us denote the fixed value of $\psi_{norm}(\mathbf{f}^{(k+1)})$ as ψ_{magn}. After the linear system is solved, the ψ values are updated and we enter the next nested iteration $l+1$.

The linear equation for the u component is the following:

$$
\begin{aligned}
0 = & \psi_{data}^{(k,l)}\cdot\left(I_{diff}^{(k)}+I_{x_1}^{(k)}\cdot\delta u^{(k,l+1)}+I_{x_2}^{(k)}\cdot\delta v^{(k,l+1)}\right)\cdot I_{x_1}^{(k)} \\
& -\alpha\cdot\sum_{i\in\{-1,+1\}}\psi_{sm}^{(k,l)}\left(s+\frac{i}{2}\right)\cdot\left(u^{(k)}(s+i)+\delta u^{(k,l+1)}(s+i)-u^{(k)}\right) \\
& +\alpha\cdot\left(\sum_{i\in\{-1,+1\}}\psi_{sm}^{(k,l)}\left(s+\frac{i}{2}\right)\right)\cdot\delta u^{(k,l+1)} \\
& +\beta\cdot\psi_{magn}^{(k,l)}\cdot\left(u^{(k)}+\delta u^{(k,l+1)}\right)
\end{aligned}
\tag{2.44}
$$

We have now arrived at a linear equation system in which there is an equation like the above one for each pixel along the line and also an analogous set of equations for the v components of the flow vectors. The unknowns are the $\delta u^{(k,l+1)}$ and $\delta v^{(k,l+1)}$ values again for each pixel. This is a very large system and we cannot solve it with

the straightforward, matrix decomposition-based methods for solving linear systems. Instead, we use an iterative algorithm called successive over-relaxation (SOR).

In short, the SOR method considers each equation in the system one after the other and adjusts a single variable each time. The readjustment is done by considering that single equation alone. For this the equation is rearranged to the form $0 = a \cdot x + b$. The straightforward method (called Gauss-Seidel method) would be to set $x_{new} = \frac{-b}{a}$. The more sophisticated SOR method specifies

$$x_{new} = x_{prev} + \omega \cdot \left(\frac{-b}{a} - x_{prev} \right) = \omega \cdot \frac{-b}{a} + (1 - \omega) \cdot x_{prev}. \tag{2.45}$$

This method pushes the value of x from x_{prev} towards satisfying the equation under consideration (which would be at $\frac{-b}{a}$) but multiplies this step with the factor ω so that convergence can be made faster or more cautious depending on the characteristics of the problem at hand.

This is done for each variable and the whole cycle is repeated several times. This repetition cycle is a new nested iteration level in our overall algorithm, marked by iteration index m. The equation format required for SOR becomes the following in our case.

$$0 = A_u^{(k,l)} \cdot \delta u^{(k,l+1)} + B_u^{(k,l)} \tag{2.46}$$

$$0 = A_v^{(k,l)} \cdot \delta v^{(k,l+1)} + B_v^{(k,l)} \tag{2.47}$$

By reordering Equation 2.44, we get the following for the u component (it is analogous for v):

$$A_u^{(k,l)} = \psi_{data}^{(k,l)} \cdot \left(I_{x_1}^{(k)} \right)^2 + \alpha \sum_{i \in \{-1,+1\}} \psi_{sm}^{(k,l)} (s + i/2) + \beta \cdot \psi_{magn}^{(k,l)} \tag{2.48}$$

$$\begin{aligned} B_u^{(k,l)} = {} & \psi_{data}^{(k,l)} \cdot \left(I_{x_1}^{(k)} \cdot I_{diff}^{(k)} + I_{x_1}^{(k)} \cdot I_{x_2}^{(k)} \cdot \delta v^{(k,l+1)} \right) \\ & - \alpha \sum_{i \in \{-1,+1\}} \psi_{sm}^{(k,l)} (s + i/2) \cdot \left(u^{(k)}(s+i) + \delta u^{(k,l+1)}(s+i) - u^{(k)} \right) \\ & + \beta \cdot \psi_{magn}^{(k,l)} \cdot u^{(k)} \end{aligned} \tag{2.49}$$

The SOR adjustment formula is:

$$\delta u_{new}^{(k,l+1)} = \delta u_{prev}^{(k,l+1)} + \omega \cdot \left(\frac{-B_u^{(k,l)}}{A_u^{(k,l)}} - \delta u_{prev}^{(k,l+1)} \right) \tag{2.50}$$

We can notice that certain factors appear in A and B that can be precomputed at the beginning of iteration $k+1$:

$$I_{x_i x_i}^{(k)} = \left(I_{x_i}^{(k)}\right)^2 \tag{2.51}$$

$$I_{x_1 x_2}^{(k)} = I_{x_1}^{(k)} \cdot I_{x_2}^{(k)} \tag{2.52}$$

$$I_{diff x_i}^{(k)} = I_{diff}^{(k)} \cdot I_{x_i}^{(k)} \tag{2.53}$$

2.2.3 Coarse-To-Fine Estimation

In itself, the above described algorithm can only make accurate flow vector adjustments on the order of around one pixel. To handle larger displacements, we use the well-known coarse-to-fine estimation scheme. We first repeatedly halve the image resolution until the expected displacements are on the order of one pixel. We solve the optical flow problem on the coarsest level with the initial flows set to zero. We then scale the solution up bz a factor of two, run the solver again (with with flows initialized to the scaled up solution from the coarsest scale), and we repeat this until we have the estimation on the finest level.

2.3 Regression

Regression is one of the main supervised learning tasks in machine learning, the field concerned with automatically extracting generalizable patterns from observed data. In regression, this pattern is an underlying functional relationship between some input and output variable(s). Specifically, we are given *training examples* $\{\mathbf{x}_i, y_i\}_{i=1}^N$, $\mathbf{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}$ and we want to predict the output y_* corresponding to some newly observed *test input* $\mathbf{x}_*$ that we have not seen before.

The challenge in regression is to capture a core input-output relationship that will stay consistent across training and test time as opposed to disturbances due to unknown factors and coincidences (collectively modeled as noise). Different regression algorithms (and their different parametrizations) make different prior assumptions regarding the underlying function and the noise. The algorithms that we consider are ridge regression, kernel ridge regression and noisy-input Gaussian process regression. Ridge regression is chosen for being a very fast and simple baseline, kernel ridge regression for its appealing theoretical foundations (including the Gaussian process interpretation) and good performance, and finally the noisy-input Gaussian process can give us better uncertainty estimations and can weight the importance of training examples adaptively based on target values.

2.3.1 Bayesian Formulation

All of these regression methods can be expressed in a basic Bayesian probabilistic setting, laid out in more detail in [BN06]. First, let us assume there is no noise in $\mathbf{x}$

Algorithm 1 Line optical flow computation (single scale)

function SINGLESCALEFLOW($I_1, I_2, \mathbf{f}_{init}, \mathbf{p_1}, \mathbf{p_2}, \alpha, \beta, \omega, \varepsilon, K, L, M$)
  $\mathbf{f} \leftarrow \mathbf{f}_{init}$
  $G_2 \leftarrow$ SOBELFILTER(I_2)

  // Loop in whose iterations E_{data} is linearized:
  for $k \in \{0, \ldots, K-1\}$ **do**
    $\left(I_{x_1x_1},\ I_{x_1x_2},\ I_{x_2x_2},\ I_{diffx_1},\ I_{diffx_2}\right) \leftarrow$ LINEARIZEDATATERM($I_1, I_2, G_2, \mathbf{f}$)
    $\delta\mathbf{f} \leftarrow \mathbf{0}$

    // Loop in whose iterations Ψ_{data}, Ψ_{sm} and Ψ_{magn} are constantized:
    for $l \in \{0, \ldots, L-1\}$ **do**
      $\hat{\mathbf{f}} \leftarrow \mathbf{f} + \delta\mathbf{f}$
      for $s \in \{0, \ldots, \lfloor \ell \rfloor\}$ **do**
        // calculate Ψ_{data} according to equation 2.31
        $\Psi_{data}[s] \leftarrow$ CALCPSIDATA($I_1, I_2, G_2, \hat{\mathbf{f}}, s, \varepsilon$)
        $\Psi_{sm}[s] \leftarrow \left(\left\|\hat{\mathbf{f}}[s+1] - \hat{\mathbf{f}}[s]\right\|^2 + \varepsilon^2\right)^{-1/2}$
        $\Psi_{magn}[s] \leftarrow \left(\left\|\hat{\mathbf{f}}[s]\right\|^2 + \varepsilon^2\right)^{-1/2}$
      end for

      // Successive over-relaxation cycles:
      for $m \in \{0, \ldots, M-1\}$ **do**
        // One SOR cycle:
        for $s \in \{0, \ldots, \lfloor \ell \rfloor\}$ **do**
          Calculate A_u, B_u according to Equations 2.48 and 2.49
          $\delta u[s] \leftarrow \delta u[s] + \omega \cdot (\frac{-B_u}{A_u} - \delta u[s])$
          Calculate A_v, B_v analogously
          $\delta v[s] \leftarrow \delta v[s] + \omega \cdot (\frac{-B_v}{A_v} - \delta v[s])$
        end for
      end for
    end for

    $\mathbf{f} \leftarrow \mathbf{f} + \delta\mathbf{f}$

  end for
  return f
end function

Algorithm 2 Line optical flow computation (single scale)

function LINEARIZEDATATERM($I_1, I_2, G_2, \mathbf{f}$)
  $I_{x_1x_1}$, $I_{x_1x_2}$, $I_{x_2x_2}$, I_{diffx_1}, $I_{diffx_2} \leftarrow 0$
  for $s \in \{0, \ldots, \lfloor \ell \rfloor\}$ **do**
    $E_{data} \leftarrow 0$
    $I_{diffx_1}[s] \leftarrow 0$
    $I_{diffx_2}[s] \leftarrow 0$
    for $c \in \{0, \ldots, C-1\}$ **do**
      $\Delta \leftarrow I_{2,c}(\mathbf{p}(s) + \mathbf{f}[s]) - I_{1,c}(\mathbf{p}(s))$
      $E_{data} \leftarrow E_{data} + \Delta^2$
      // Below, $G_{2,c,1}$ stands for the first component of the
      // gradient vector of image channel c in the second frame
      $I_{diffx_1}[s] \leftarrow I_{diffx_1}[s] + \Delta \cdot G_{2,c,1}(\mathbf{p}(s) + \mathbf{f}[s])$
      $I_{diffx_2}[s] \leftarrow I_{diffx_2}[s] + \Delta \cdot G_{2,c,2}(\mathbf{p}(s) + \mathbf{f}[s])$
    end for
    $I_{x_1x_1}[s] \leftarrow I_{diffx_1}[s]^2 / E_{data}$
    $I_{x_1x_2}[s] \leftarrow I_{diffx_1}[s] I_{diffx_2}[s] / E_{data}$
    $I_{x_2x_2}[s] \leftarrow I_{diffx_2}[s]^2 / E_{data}$

  end for
  return $\left(I_{x_1x_1}, I_{x_1x_2}, I_{x_2x_2}, I_{diffx_1}, I_{diffx_2}\right)$
end function

and there is additive normally distributed noise in y:

$$\begin{aligned} y &= f(\mathbf{x}) + \varepsilon \\ \varepsilon &\sim \mathcal{N}(0, \sigma_n^2) \end{aligned} \tag{2.54}$$

Furthermore, we assume that f is a linear function in some feature space with mapping function $\phi : \mathbb{R}^m \to \mathbb{R}^d$ and let us model our prior uncertainty about the linear weight coefficients with normally distributed priors:

$$\begin{aligned} f(\mathbf{x}) &= \phi(\mathbf{x})^\top \mathbf{w} \\ \mathbf{w} &\sim \mathcal{N}(\mathbf{0}, \sigma_p^2 I) \end{aligned} \tag{2.55}$$

Let X be an $n \times m$ matrix where each row contains a training input, let X_* be the same for the test inputs, Y the training outputs (single column), Y_* the test outputs (these are random variables), Φ the training inputs in feature space (again row-wise) and Φ_* the test inputs in feature space.

Using the above assumptions, one can marginalize over $\mathbf{w}$ analytically and derive that the predictive distribution of the test outputs Y_* is a multivariate normal distribution with the following mean and covariance:

$$\mathbb{E}(Y_*|X_*, X, Y) = \Phi_* \left(\Phi^\top \Phi + \frac{\sigma_n^2}{\sigma_p^2} I \right)^{-1} \Phi^\top Y \tag{2.56}$$

$$Cov(Y_*|X_*, X, Y) = \sigma_n^2 I + \sigma_n^2 \Phi_* \left(\Phi^\top \Phi + \frac{\sigma_n^2}{\sigma_p^2} I \right)^{-1} \Phi_*^\top \tag{2.57}$$

$$\tag{2.58}$$

This is already a useful result. If we set $\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$, i.e. add an extra dimension with constant 1, this mean prediction formula coincides with the formula for ridge regression.

However, this result can be rearranged to a form that is often more advantageous. The quickest way to do so is to consider the Woodbury matrix identity with substitution $A = \lambda I, U = \Phi^\top, V = \Phi, C = I$:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{2.59}$$

$$(\Phi^\top \Phi + \lambda I)^{-1} = \lambda^{-1} \left(I - \Phi^\top (\Phi \Phi^\top + \lambda I)^{-1} \Phi \right) \tag{2.60}$$

A second identity that we need can be derived by multiplying both sides by $\Phi^\top$ from the right and rearranging:

$$(\Phi^\top\Phi+\lambda I)^{-1}\Phi^\top = \lambda^{-1}\left(I-\Phi^\top(\Phi\Phi^\top+\lambda I)^{-1}\Phi\right)\Phi^\top \tag{2.61}$$
$$= \lambda^{-1}\Phi^\top - \lambda^{-1}\Phi^\top(\Phi\Phi^\top+\lambda I)^{-1}\Phi\Phi^\top \tag{2.62}$$
$$= \lambda^{-1}\Phi^\top(\Phi\Phi^\top+\lambda I)^{-1}\left((\Phi\Phi^\top+\lambda I)-\Phi\Phi^\top\right) \tag{2.63}$$
$$= \Phi^\top(\Phi\Phi^\top+\lambda I)^{-1} \tag{2.64}$$

Setting $\lambda = \frac{\sigma_n^2}{\sigma_p^2}$ we can use equation 2.64 on the expectation and equation 2.60 on the covariance to get:

$$\mathbb{E}(Y_*|X_*,X,Y) = \Phi_*\Phi^\top\left(\Phi\Phi^\top+\frac{\sigma_n^2}{\sigma_p^2}I\right)^{-1}Y \tag{2.65}$$

$$Cov(Y_*|X_*,X,Y) = \sigma_n^2 I+\sigma_p^2\left(\Phi_*\Phi_*^\top-\Phi_*\Phi^\top\left(\Phi\Phi^\top+\frac{\sigma_n^2}{\sigma_p^2}I\right)^{-1}\Phi\Phi_*^\top\right) \tag{2.66}$$

Now we notice that the feature vectors $\phi(\mathbf{x})$ only appear in the form of scalar products with other feature vectors. This is the key property in all kernelizable learning algorithms because it allows us to sidestep direct considerations of the linear weights $\mathbf{w}$ and the feature space. Instead, let us define the *kernel function* as

$$k:\mathbb{R}^m\times\mathbb{R}^m\to\mathbb{R}\quad k(\mathbf{x},\mathbf{x}') = \sigma_p^2\phi(\mathbf{x})^\top\phi(\mathbf{x}') \tag{2.67}$$

If we introduce the kernel matrices $K=\sigma_p^2\Phi\Phi^\top$ and $K_*=\sigma_p^2\Phi_*\Phi^\top$, we can rewrite the predictive mean and covariance:

$$\mathbb{E}(Y_*|X_*,X,Y) = K_*\left(K+\sigma_n^2 I\right)^{-1}Y \tag{2.68}$$
$$Cov(Y_*|X_*,X,Y) = \sigma_n^2 I+K_{**}-K_*\left(K+\sigma_n^2 I\right)^{-1}K_*^\top \tag{2.69}$$

In kernel learning algorithms we specify the kernel function directly instead of defining ϕ. The definition as a replacement for the inner product in feature space does not give us guidance on what could be a good kernel function. We get a more enlightening interpretation by noticing that

$$k(\mathbf{x},\mathbf{x}') = Cov(f(\mathbf{x}),f(\mathbf{x}')), \tag{2.70}$$

which can be proved by marginalizing over the linear weights $\mathbf{w}$. This shows that the kernel function can be interpreted informally as a kind of similarity function. Therefore, a popular choice for the kernel function is the *squared exponential kernel*:

$$k_{SE}(\mathbf{x},\mathbf{x}') = \sigma_p^2 \exp(-\gamma\, \mathbf{x}^\top \mathbf{x}') \tag{2.71}$$

By using this kernel, we assume that proximity in the input space implies proximity in the output space, which is a sensible default assumption for most applications. It also turns out that using this kernel function means that we implicitly set ϕ to be a function with infinite dimensional output, meaning that the implicit linear regression takes place in an infinite dimensional feature space, something that we could not realize without the kernel trick.

2.3.2 Gaussian Process Interpretation

Another interpretation for the above results is called the Gaussian process (GP) regression view, which does not make any reference to the linear weights $\mathbf{w}$, and works directly using priors over entire functions instead. An excellent introduction to Gaussian process learning can be found in [RW05].

A Gaussian process $\mathcal{GP}(m(\cdot),k(\cdot,\cdot))$ is a stochastic process, i.e. an infinite collection of random variables, with the defining property that any finite set of the random variables has a joint multivariate normal distribution. The parameter $m(\cdot)$ is a the *mean function* and $k(\cdot,\cdot)$ is the *covariance function* of the process. The random variables of the process correspond to the outputs of the function f at each point in the input space. One can also think of the GP as a generalization of the multivariate normal distribution for vectors with infinitely many components.

Working directly with distributions over functions can be regarded as more elegant than starting out with linear regression in an unspecified feature space.

Note that in Gaussian process literature, the predictive covariance formula (2.69) is usually given for the function values $f(\mathbf{x})$ and not for the noisy outputs y, so that formula does not contain the $\sigma_n^2 I$ term.

Some other parameter notations usually encountered in kernel ridge regression, Gaussian process and Bayesian linear regression literature are:

$$\alpha = \frac{1}{\sigma_p^2} \qquad \beta = \frac{1}{\sigma_n^2} \qquad \lambda = \frac{\sigma_n^2}{\sigma_p^2} = \frac{\alpha}{\beta} \qquad C = \frac{\sigma_p^2}{2\sigma_n^2} = \frac{1}{2\lambda} \tag{2.72}$$

2.3.3 Noisy Input Gaussian Process Regression

One shortcoming of the basic Gaussian process regression is that it does not consider the training output values when estimating the prediction uncertainty. The uncertainty solely depends on the distribution of training inputs. This means that a test input that is close to many training inputs will yield a low-uncertainty prediction, even if the nearby training data points have widely varying, almost contradictory target values. Seeing such a high variability in nearby outputs, common sense would dictate that any new prediction in that area has high uncertainty.

Another unfortunate consequence appears in case of a multi-output regression task since the estimated uncertainty will be equal for all outputs (assuming that the outputs are equivalent a priori, before seeing the training set). This is especially problematic when learning the left and right pedestrian flows on a line, since their uncertainties are not always the same (consider occlusion effects for example, when one direction is clearly visible and the other is more uncertain).

A possible remedy for these problems is to use the noisy-input Gaussian process, introduced by McHutchon and Rasmussen[MR11]. In this modified formulation, the inputs are also assumed to be affected by normally distributed noise and the functional relationship that we model exists between the input and output if we disregard the noise:

$$y - \varepsilon_1 = f(\mathbf{x} - \varepsilon_2) \tag{2.73}$$
$$\varepsilon_1 \sim \mathcal{N}(0, \sigma_n^2) \tag{2.74}$$
$$\varepsilon_2 \sim \mathcal{N}(\mathbf{0}, \Sigma_i) \tag{2.75}$$

Unfortunately, computing the exact posterior for this formulation is intractable. The approximation proposed by the authors is to linearize $f(\mathbf{x} - \varepsilon_2)$ with respect to $\mathbf{x}$:

$$f(\mathbf{x} - \varepsilon_2) \approx f(\mathbf{x}) - \varepsilon_2^\top f'(\mathbf{x}) \tag{2.76}$$

Computing the exact mean and covariance when using this approximative formula is still too computationally expensive. Instead, McHutchon and Rasmussen ignore the uncertainty in the derivative by simply taking the derivative *of the expected value* of $f(\mathbf{x})$. The resulting formulas are:

$$\mathbb{E}(Y_*|X_*, X, Y) = K_* \left(K + \sigma_n^2 I + \Delta_{\bar{f}}^\top \Sigma_i \Delta_{\bar{f}} \right)^{-1} Y \tag{2.77}$$
$$Cov(Y_*|X_*, X, Y) = \sigma_n^2 I + \Delta_{\bar{f}_*}^\top \Sigma_i \Delta_{\bar{f}_*} + K_{**} - K_* \left(K + \sigma_n^2 I + \Delta_{\bar{f}}^\top \Sigma_i \Delta_{\bar{f}} \right)^{-1} K_*^\top \tag{2.78}$$

where $\Delta_{\bar{f}}$ are $n \times m$ matrices containing row-wise the gradients of the predictive mean w.r.t. at each training input and $\Delta_{\bar{f}_*}$ is the same for test inputs. The formula is equivalent to adaptively assuming higher output noise for the training (and test) outputs where the gradient of the output is higher. Essentially, the uncertainty in the inputs has been converted into additional uncertainty in the outputs.

Intuitively, this method "trusts" training samples less in those regions, where the desired targets have very diverse values. It is important to see that this method also affects the mean prediction, it isn't simply a tweak of the uncertainty prediction. (Note that the formula in the original paper specifies the covariance for the noiseless output f, and therefore has less terms than this formulation for the noisy output.)

The above formulas are still hard to handle computationally because the $\Delta_{\bar{f}}$ refer to the gradients of the whole expression that they are part of. Instead of solving this complicated differential equation, the authors propose an iterative scheme, in which the gradients are first set to zero and then get updated by computing the gradient of the resulting formula for the predictive mean. This means that we first solve the classic Gaussian process regression task without any input noise considerations and then use an analytically expressible formula for the gradient of the mean prediction function to get the corrective term $\Delta_{\bar{f}}^{\top}\Sigma_i\Delta_{\bar{f}}$. Then we solve the Gaussian process problem with this modified noise term. The authors suggest that a single iteration can already yield a good approximation, but we could go on and compute the gradients again, plug in the new values and solve the GP, etc.

To use this method, one needs to specify the input noise covariance Σ_i. One can use any hyperparameter tuning method for this purpose, including maximizing the marginal likelihood, grid search, random search, etc. We will set it as $\sigma_i I$, i.e. each input variable is assumed to be affected by the same amount of independent noise, and find an appropriate value for σ_i by the same procedure that we use for setting σ_n and σ_f.

Chapter 3

Baseline Based on Flow Mosaicking

In this chapter, we formulate a slightly modified version of the flow mosaicking method proposed by Cong et al. in [YHSY09], for use as a baseline. This is a line counting method and its main idea is similar to the spatiotemporal slicing technique. The key difference is that in the flow mosaicking approach we don't always sample and stack a one-pixel-wide part of each frame (i.e. the pixels belonging to the line-of-interest), but the width of the extracted image part depends on the speed of the pedestrians.

3.1 Original Formulation

First, one needs to compute line optical flows. Cong et al. do this by discrete optimization with a dynamic programming approach. Then the following steps are performed

- Compute the optical flow for the counting line.
- Threshold the (one-dimensional) line flow and segment it to moving directions. Both directions are processed independently in the further steps.
- Take the connected components in both of the direction masks (still one-dimensional).
- Discard small components.
- Compute the mean optical flow (orthogonal to the line) in each component.
- For each component, extract a region from the frame that is centered on the counting line, and its position along the line is determined by the component's position. The width of this region is chosen as the mean optical flow in the component.
- Stack these extracted regions side-by-side to build so called flow mosaics (blobs).
- Stop stacking onto the same flow mosaic once it reaches a given size, and start a new one instead.

- Extract features from each flow mosaic and learn a regression on them. The two features are: perspective weighted number of foreground pixels, and the perspective weighted number of edge pixels (in the foreground mask).

3.2 Modifications to the Approach

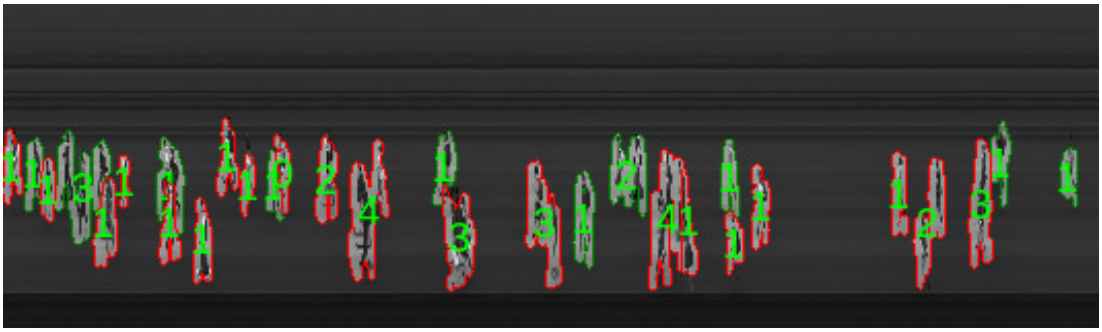

Figure 3.1: The blobs and the ground truth counts on the spatiotemporal slice, created using our version of flow mosaicking. Red outline: rightwards, green outline: leftwards.

First, we use our own implementation of line optical flow as described in Chapter 2 instead of the dynamic programming approach by Cong et al.

Furthermore, there is a problem with naively building flow mosaics that we need to consider. Since we extract image regions with varying width, the image does not "grow" uniformly in all parts of the line. In some parts of the line we add wider crops than in others. This means that the resulting flow mosaics may overlap with each other and overwrite each other's values in the flow mosaic image.

Managing the splitting and merging of different flow mosaics is also considerably complicated if our blobs are irregularly shaped.

Instead, we create an equivalent formulation of the flow mosaicking approach that does not explicitly build mosaics as images but extracts and accumulates the same features directly. The original formulation would have us lay out the flow mosaics onto an image and go over it in a second pass, calculating the features. By contrast, we turn this algorithm into a blob-based spatiotemporal slicing-based method, where the blob features are computed using data outside the actual slice image i.e. a wider area around the line in the original frame.

In more detail, we first find the connected components of the direction mask in the spatiotemporal slice. Since the original approach does not grow the flow mosaics over a given size, we also break up these connected components such that the corresponding flow mosaics would not exceed this limit.

Then we process each resulting smaller blob S to extract the features. This is done by taking one column of the spatiotemporal slice at a time. The connected components

of *S* inside this 1 pixel wide part are then found. For each connected component of the column, we determine the average optical flow speed and extract a rectangle around the line from the corresponding original frame with the width chosen as the average speed. Then the perspective weighted foreground pixels and edge pixels are computed in this extracted part of the original frame and these values are summed up to arrive at the final two features describing blob *S*. See Figure 3.1 for examples of resulting blobs.

Once we have the blob features, we learn regression models separately for the two movement directions.

At test time we extract the blobs and their features exactly as in the training phase and use the regression to predict the number of pedestrian line crossings in each. Then we convert the blob-wise results to a continuous flow estimation by distributing the result of each blob uniformly over the time range covered by the blob.

Chapter 4

Line-Based Pedestrian Counting

In this chapter we discuss how we can predict the pedestrian flow over a line segment by observing the line's local neighborhood over time. For this we first discuss the representation of this problem in terms of regression and choose a method to learn the regression model. By doing this, we convert a vision problem to the standard machine learning framework, allowing us to use well-established learning techniques.

Seen from a high-level perspective, our basic task is the following. We are given a training set containing an RGB video, a line segment (line-of-interest, LOI) $L = (\mathbf{p}_1, \mathbf{p}_2)$ and ground-truth person trajectory annotations for the video. At test time, we are given another RGB video taken from the same position and we have to estimate the cumulative number of leftwards and rightwards people-crossings over the line L as a function of time.

4.1 Spatiotemporal Slicing

To represent this vision task as a regression problem, we need a method to create input/ouput samples based on the video data. For this, we first introduce the concept of a spatiotemporal slice. A spatiotemporal slice is an image where the horizontal direction corresponds to time and the vertical direction corresponds to the position on the line. Computing a spatiotemporal line means sampling the line (a 1 pixel wide area of the image) and stacking these samples side by side (see top image in Figure 4.2).

Directly taking the spatiotemporal slice of the RGB video and passing the result through standard feature extractors may seem to be the straightforward approach, but it is not a good idea, since the slice shows severe distortions due to varying speed and angle of the pedestrians crossing. Instead, we first preprocess the video frame by frame to extract multiple local features and then we create a slice through each of the resulting preprocessed videos. This preprocessing yields the following:

- Background subtraction masks
- Canny edges

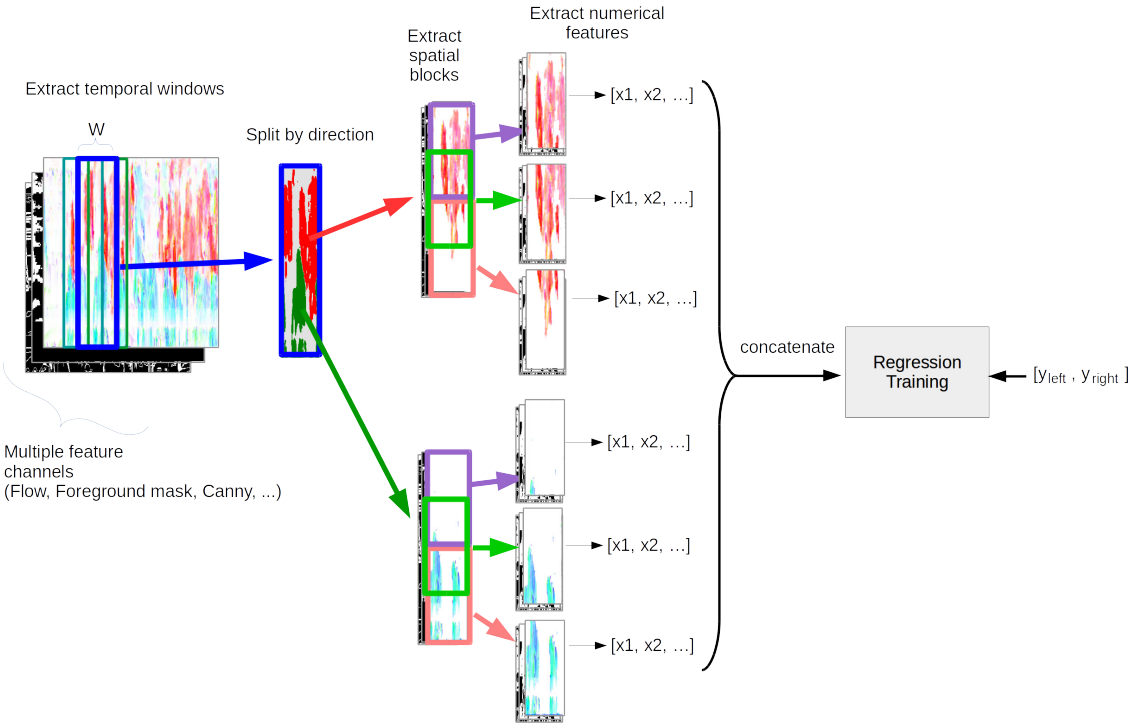

Figure 4.1: Overview of the window-based feature extraction method.

- Optical flows

The optical flow and the background subtraction masks are computed as explained in Chapter 2 (*Preliminaries*). After taking the spatiotemporal slice for each of these transformed frame sequences, the result can be thought of as a multi-channel spatiotemporal slice.

From this multichannel slice, we use a sliding-window-based approach to yield the input/output samples for regression. The sliding window has size w and step s, so that the ith window covers the time interval $(i \cdot s, i \cdot s + w)$. The ground truth (desired) output for the regression sample corresponding to a given temporal window is the number of people crossing the line in the given time interval.

4.2 Feature Extraction

Each of the (overlapping) temporal windows are described using a feature vector. This vector is the concatenation of a descriptor of the leftward moving segment and the rightward moving one. Therefore, we first need to segment the slice to leftward moving, rightward moving and non-moving (stationary) regions. After suppressing noise by median and Gaussian filtering, we threshold the absolute value of the optical flow

perpendicular to the line as follows (denoting the flow slice as **f** and the unit-length leftward pointing normal vector of the counting line as **n**):

$$Direction(t,s) = \begin{cases} Left, & \text{if } \mathbf{n}\cdot\mathbf{f}(t,s) > \tau \\ Right, & \text{if } \mathbf{n}\cdot\mathbf{f}(t,s) < -\tau \\ Stationary, & \text{otherwise} \end{cases} \tag{4.1}$$

The descriptor vectors of both the left and right moving segments are concatenated from block descriptors. Each block encompasses a certain section of the line and the blocks overlap (see Figure 4.1 for a visual explanation). This results in finer-grained locality information for the learning stage, compared to simply computing global features in the whole temporal window.

We extract three features from each block (masked with a direction mask):

- Sum of the optical flow component perpendicular to the line
- Number of foreground pixels
- Number of Canny edge pixels

In total, we have $2 \cdot N_{blocks} \cdot 3$ features describing each temporal window (2 directions, N_{blocks} blocks for each direction and 3 features per block).

Once the regression has been trained and run on a test set, we convert the resulting window-based estimations to a more detailed frame-by-frame estimate, so that the final result is independent of the choice of the window size and the particular method. This is done by averaging the nearby window estimates for each frame.

4.3 Data Normalization

Before applying the regression, we normalize the input data. The mean and standard deviation of each feature is estimated on the training set. We subtract the mean and divide each feature by its standard deviation, so that the resulting features all have zero mean and unit variance. The means and standard deviations are retained from training and the same values are used to transform the test data as well.

4.4 Data Augmentation

We augment the dataset by the mirrored copies of each sequence (mirrored around the line-of-interest). This can help avoid overfitting to an unbalanced training set which may, by chance, have significantly more people going in one direction than in the other.

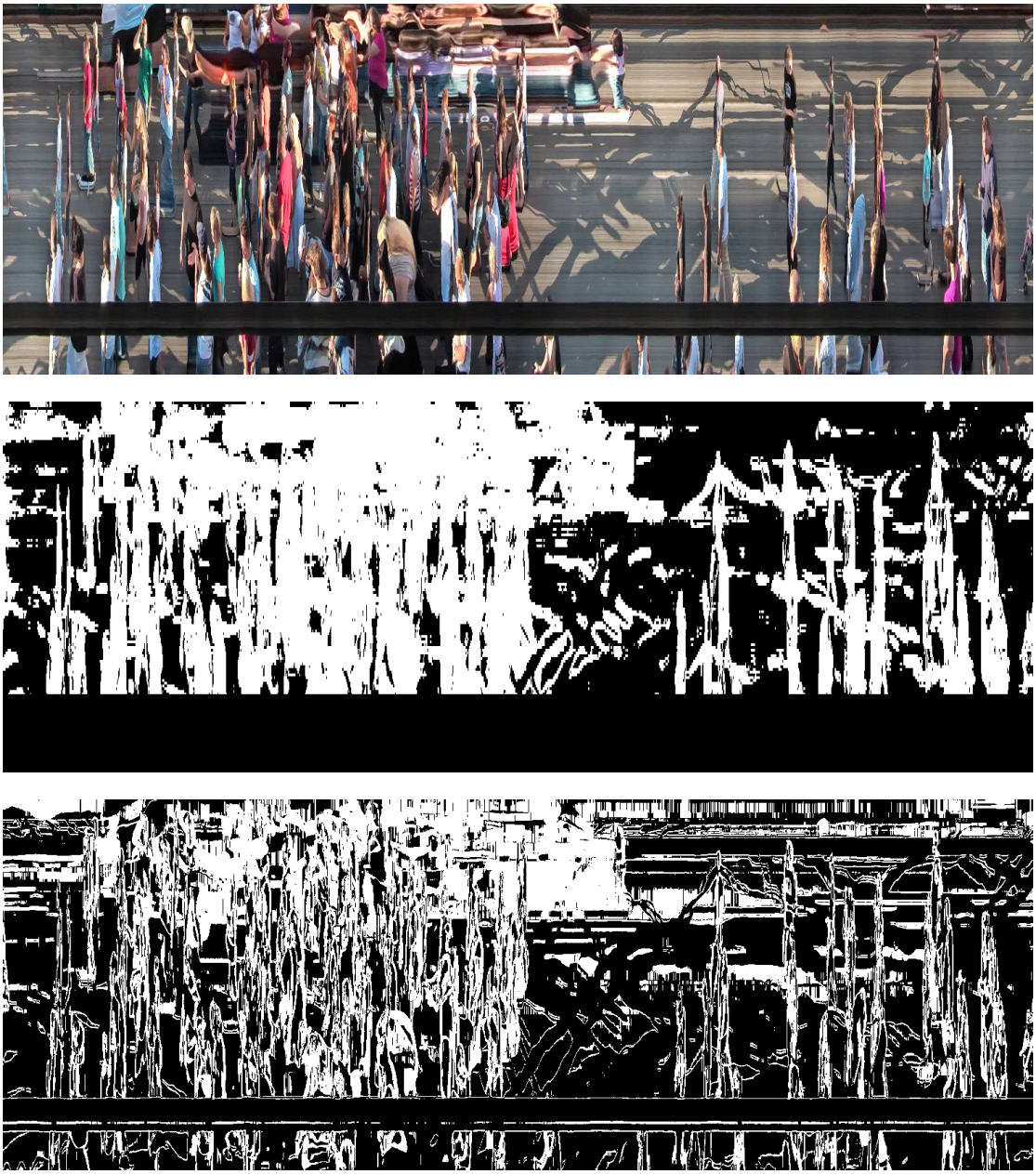

Figure 4.2: Spatiotemporal slices. *Top:* RGB slice, *Middle:* Foreground mask slice, *Bottom:* Canny edge slice.

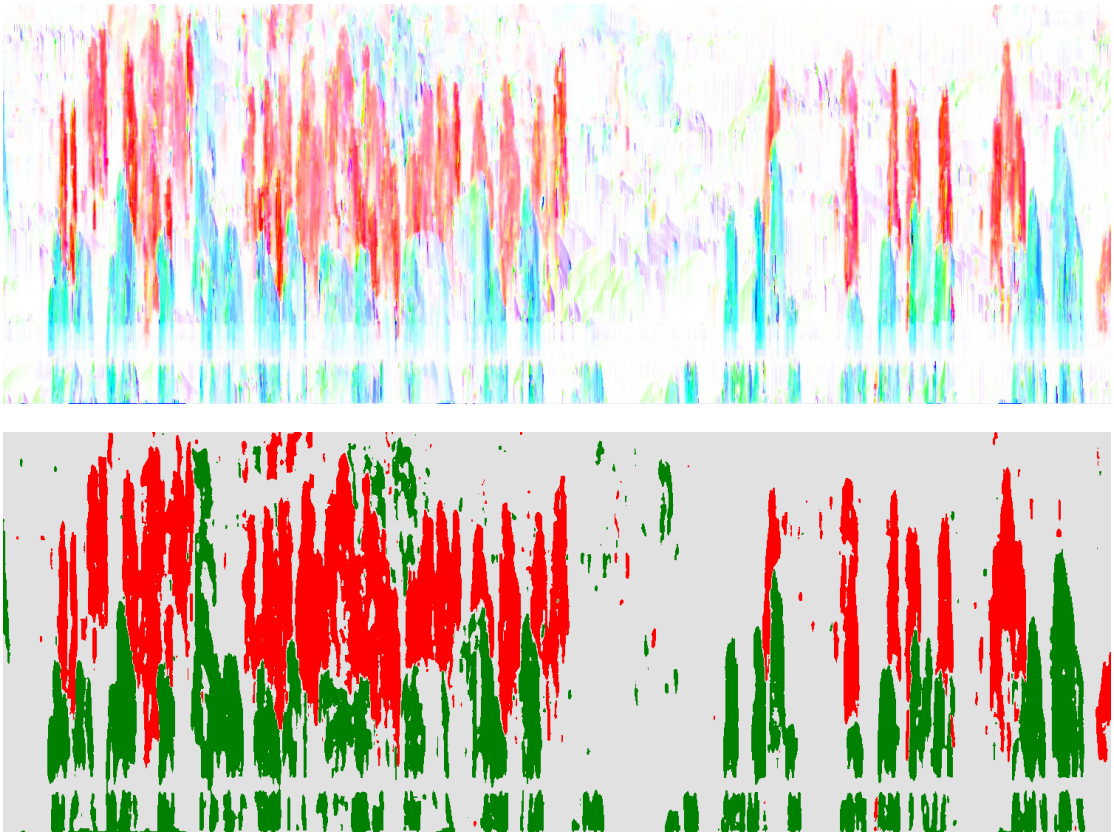

Figure 4.3: *Top:* Optical flow slice. Each flow vector is represented by a color in hue, saturation, value (HSV) color space, with the flow direction shown as hue and the magnitude show as saturation (the V component is fixed at the 255). *Bottom:* Segmentation of the optical flow by thresholding (red: leftwards, green: rightwards).

Chapter 5

Region-Based Pedestrian Counting

Region-counting (or ROI-counting, standing for *region-of-interest*) is a task in visual crowd analysis, where the number of pedestrians inside a given region has to be estimated for each frame of a video. In this thesis our main focus is on the line counting task but region counting approaches can also useful to us for two reasons. First, we can compare their failure modes and whether the "hard cases" are the same for both, giving us more insight and guidance for future research. Second, we may combine the line counting and the region counting results into a single consistent solution, hoping that they will mutually improve each other in the process (we will come back to this idea in Chapter 6).

5.1 Overview

We subdivide the area of the frame to several rectangular regions using a grid. The task is then to estimate the number of people in each of these regions in each frame. Our method for region counting is based on the work of Chen et al.[CLGX12] and Chan et al.[CLV08]. Both are low-level feature-based regression approaches.

There are two main steps to counting by low-level feature regression. First, the feature vector is assembled for each frame, and second, some learning algorithm is applied to learn a regression from the feature vector to the desired output (people count). We can choose from multiple approaches when it comes to building a regression model. The three main options are:

- Learn a separate, independent model for each grid region. This method uses no context information and can suffer from too little training data.

- Learn a single region model that is trained with the data from every grid region and is also used for prediction in every grid region. This information sharing can be beneficial if the regions are similar enough to be handled by the same parameter settings.

- Concatenate the feature vectors from all regions and learn a multi-output regression that maps from the long concatenated vector to the vector of the pedestrian counts in all regions. This is the main idea in Chen et al.'s paper and it can be used to uncover correlations across regions (mainly neighboring ones), since some visual information in one cell can be indicative of a person being in another cell.

We will use the second option, since our regions are similar enough to benefit from learning from each other's training data.

5.2 Feature Extraction

Feature extraction starts with the already familiar background subtraction step. Then three types of features are extracted from each region independently: foreground mask-based features (e.g. foreground area), edge-based features (e.g. edge orientation histogram) and texture-based features (using gray-level co-occurrence matrices). To reduce perspective distortion effects, pixels are weighted depending on their scale when calculating the features.

In more detail, we extract the following features from the foreground mask that we obtain by background subtraction.

- *Foreground area:* the number of pixels in the foreground mask
- *Perimeter length:* The number of pixels on the perimeter of the foreground. The perimeter can be computed using simple morphological operations on the mask.
- *Perimeter-to-area ratio:* The perimeter length divided by the foreground area. A high value indicates a complicated, "squiggly" perimeter.
- *Perimeter orientation histogram:* A filter bank of oriented ellipse-shaped Gaussians is applied at each perimeter pixel and the maximum response is recorded for each, which corresponds to the local orientation of the perimeter. Then a histogram of these orientations is computed over the whole region.

Edges are obtained using the Canny algorithm and we compute the following features based on the edge map:

- *Edge pixel count:* The number of edge pixels.
- *Edge orientation histogram:* Computed the same way as the perimeter orientation histogram.
- *Minkowski fractal dimension:* The Minkowski fractal dimension is used to estimate the degree of space filling of a curve, in this case the foreground edges. As described in [aMCLV99], the Minkowski dimension can be approximated using

a simple algorithm. The morphological operation of dilation is applied to the image with successively larger and larger disk structuring elements, so that the edges become thicker (called Minkowski sausages). For each of these dilated images we compute the size of the filled area. This area increases with growing disk size until it reaches the area of the whole image. The rate of growth of the filled area A for growing disk diameter d is used to estimate the Minkowski dimension. First, a log-log transformation is applied:

$$(d,A) \mapsto (\log(d), \log(A)) \tag{5.1}$$

Then least-squares linear regression is used to obtain the slope of this log-log plot resulting in the approximation

$$\begin{aligned} \log(A) &= w_1 \cdot \log(d) + w_0 \\ A &= \exp(w_0) \cdot d^{w_1} \end{aligned} \tag{5.2}$$

Finally, the Minkowski dimension m is approximated as

$$m = 2 - w_1. \tag{5.3}$$

Both Chan et al. and Chen et al. use texture features based on gray-level co-occurrence matrices (GLCM) for region counting.

Given a distance d and an angle θ, the gray-level co-occurrence matrix of an image I (quantized to q gray levels) is defined as

$$G(d,\theta) \in \mathbb{N}^{q \times q} \tag{5.4}$$

$$\begin{aligned} G(d,\theta)_{ij} = \Bigg| \Big\{ \big((x_1,y_1),(x_2,y_2)\big) \;\Big|\; & dist\big((x_1,y_1),(x_2,y_2)\big) = d \\ & \text{and } \operatorname{atan2}(y_2-y_1, x_2-x_1) = \theta \\ & \text{and } I(x_1,y_1) = i \text{ and } I(x_2,y_2) = j \Big\} \Bigg| \end{aligned} \tag{5.5}$$

The GLCM describes how many times two pixels that are at the given distance and angle to each other have the gray level i and j, respectively. The GLCM is computed for $d = 1$ and $\theta \in \{0°, 45°, 90°, 135°\}$.

Haralick et al.[HSD73] have defined 14 features that can be computed from a GLCM. We use the same 3 of them as Chan et al.:

- *Homogeneity:* $H = \sum_{i,j} \frac{G_{ij}}{1+(i-j)^2}$
- *Energy:* $E = \sum_{i,j} G_{ij}^2$
- *Entropy:* $S = -\sum_{i,j} G_{ij} \log G_{ij}$

5.3 Regression

Having constructed the feature vectors as explained above, we can apply any regression algorithm to learn the correspondence between the features and the region counts.

5.3.1 Postprocessing

The frame-by-frame region counts are usually susceptible to some independent random effects as the feature extraction methods are not perfectly robust. For instance, the Canny edges will look quite different in each frame due to the thresholding step (a whole edge piece can suddenly disappear if the gradients decrease under the threshold). Therefore, we smooth the predicted region count time series with a median filter of size 3.

Chapter 6

Combining Line-Based and Region-Based Counting

Seeing that line-based and region-based counting methods use rather different aspects of the crowd appearance, the question occurs naturally whether we could improve the overall counting performance by combining them together. To do this, we place a set of N equidistant parallel lines over the image. Besides counting the pedestrian flow over these lines, we also run the region counting procedure on the $N-1$ regions between the lines.

Theoretically, once we know the initial number of people in each region, the crowd flows over the lines should exactly determine all the subsequent region counts, due to the "conservation of people". Therefore, the region counts contain a strict subset of the information in the line counts (as long as people don't enter or exit the scene through the top or the bottom of the video frame). However, the error characteristics of line-based counting and region-based counting are quite different. If we were to use a line-based method to calculate the region counts, we would need to integrate the line flows over time, which results in cumulating the errors. Region counting, on the other hand, gives us estimates of the actual number of people in each frame independently and therefore does not cumulate the error over time. Thus, by modifying the line flow results so that they are more consistent with the region-based estimates we may eliminate some of the effects of error drift.

6.1 Problem Formulation

First, we introduce some notation. Assume that we have run a line counter and its estimate for the instantaneous crowd flow over time is given by the function $\hat{\ell}_{i,d}(t)$, for the ith line and in direction $d \in \{L,R\}$ (left/right). We have also run the region counter and it resulted in the region count estimates $\hat{r}_i(t)$ for the ith region. Now we are looking for a combined solution that has two parts: new *cumulative* line flow functions $c_{i,d}(t)$ and new initial region counts s_i. By finding such a new solution, we automatically

create new region count estimates $q_i(t)$ according to the summation formula:

$$q_i(t) = s_i + c_{i,R}(t) - c_{i,L}(t) + c_{i+1,L}(t) - c_{i+1,R}(t) \tag{6.1}$$

At the same time, of course, the derivative of $c_{i,d}(t)$ is automatically a new estimate for the instantaneous line flows. We would like to choose $c_{i,d}(t)$ such that $c'_{i,d}(t) \approx \hat{\ell}_{i,d}(t)$ and $q_i(t) \approx \hat{r}_i(t)$.

This can be expressed as a variational optimization problem similar to the one we encountered when computing the line optical flow.

$$\begin{aligned} E(c,s) = \int_0^T \sum_{i=0}^{N} \sum_{d\in\{1,-1\}} \alpha_{i,d}(t) \left(c'_{i,d}(t) - \hat{\ell}_{i,d}(t)\right)^2 + \sum_{i=0}^{N-1} \gamma_i(t) \left(q_i(t) - \hat{r}_i(t)\right)^2 dt + \\ + \sum_{i=0}^{N-1} \rho_i \cdot (s_i - \bar{r})^2 \end{aligned} \tag{6.2}$$

Additionally we have the boundary conditions $c_{i,d}(0) = 0$, i.e. the cumulative flows begin at 0. The $\alpha_{i,d}$ and the $\gamma_i(t)$ are the penalty factors for deviations from the line flows and the region counts that the regression models predict independently. The initial region counts are also penalized according to their squared difference from the expected region count (expected before observing the test sets), with weighting factors ρ_i.

A local minimum solution to this problem must fulfill the Euler-Lagrange equation for the cumulative flow functions:

$$0 = \frac{\partial}{\partial c_{i,d}} \mathcal{L} - \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial}{\partial c'_{i,d}} \mathcal{L} \tag{6.3}$$

where $\mathcal{L}$ stands for the integrand in Equation 6.2.

Let us expand these terms. Notice that $c_{i,d}$ appears as a term in q_i and q_{i-1}. By encoding the directions d so that left has the numerical value 1 and right is -1, we get the following expression for the first term:

$$\frac{\partial}{\partial c_{i,d}} \mathcal{L} = \frac{\partial}{\partial c_{i,d}} \left(\gamma_{i-1} \cdot (q_{i-1} - \hat{r}_{i-1})^2 + \gamma_i \cdot (q_i - \hat{r}_i)^2 \right) \tag{6.4}$$

$$= 2\left(-d\gamma_{i-1} \cdot (q_i - \hat{r}_i) + d\gamma_i \cdot (q_{i-1} - \hat{r}_{i-1})\right) \tag{6.5}$$

For the second term:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial}{\partial c'_{i,d}} \mathcal{L} &= \frac{\mathrm{d}}{\mathrm{d}t} \left(2\alpha_{i,d}(t) \cdot \left(c'_{i,d}(t) - \hat{\ell}_{i,d}(t)\right)\right) \\ &= 2\alpha'_{i,d}(t) \cdot \left(c'_{i,d}(t) - \hat{\ell}_{i,d}(t)\right) + 2\alpha_{i,d}(t) \cdot \left(c''_{i,d}(t) - \hat{\ell}'_{i,d}(t)\right) \end{aligned} \tag{6.6}$$

Furthermore, we need to set the partial derivatives with respect to the initial region counts s_i to zero as well:

$$0 = \frac{\partial}{\partial s_i} E(c,s) \tag{6.7}$$

$$0 = \rho_i(s_i - \bar{r}_i) + \int_0^T \gamma_i(t)\,(s_i + c_{i,R}(t) - c_{i,L}(t) + c_{i+1,L}(t) - c_{i+1,R}(t) - \hat{r}_i(t))\,dt \tag{6.8}$$

$$0 = s_i \cdot \left(\rho_i + \int_0^T \gamma_i(t)\right) - \rho_i \bar{r}_i + \int_0^T \gamma_i(t)\,(c_{i,R}(t) - c_{i,L}(t) + c_{i+1,L}(t) - c_{i+1,R}(t) - \hat{r}_i(t))\,dt \tag{6.9}$$

6.2 Numerical Solution

The above equations, when discretized to the pixel grid, result in a linear equation system. The system is too large to be solved exactly, therefore we use the successive over-relaxation method that we have used in Section 2.2 for optical flow optimization.

6.3 Discussion

We can notice from Equation 6.2 that the originally estimated region counts only affect the combined solution through a difference with $q_i(t)$. Each $q_i(t)$ contains terms from different two lines, but the sum is only influenced by the net flow over the lines $c_{i,L} - c_{i,R}$ (see Equation 6.1). This means that this optimization procedure can only effectively change the net flows but cannot specifically target the two directions.

In fact, if we did not use time-, line-, and direction-specific penalty factors ($\alpha_{i,d}(t)$ and $\gamma_i(t)$), the optimizer would always prefer to split the required net flow changes evenly between the two directions as that leads to the least sum of squared penalty. By using different penalty factors, we let the less certain predictions be modified more than the more certain ones. Specifically, we set

$$\alpha_{i,d}(t) = \frac{1}{\sigma^2_{\ell,i,d}(t)} \tag{6.10}$$

$$\gamma_i(t) = \frac{1}{\sigma^2_{r,i}(t)}, \tag{6.11}$$

i.e. the reciprocals of the estimated variance of the line counter's and the region counter's prediction. This choice is not arbitrary. In fact, we can interpret this whole optimization procedure in a probabilistic framework as the maximization of a posterior probability.

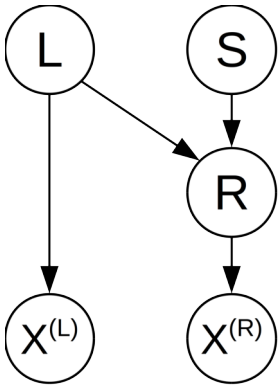

Figure 6.1: Bayes network depicting the high-level dependence structure of the line and region combination problem. L stands for all pedestrian line counts (at every frame and every line together), S for the initial region counts, R for all region counts, finally $X^{(L)}$ and $X^{(R)}$ denote the features that we extract in the line counter and the region counter algorithm respectively. L and S determine R unambiguously.

6.4 Probabilistic Interpretation

We can describe the above optimization problem in terms of the following probabilistic model. By having trained regression models for line and region counting, we get the ability to give predictive distributions of the form $p(\ell_{i,d}(t)|x_i^{(\ell)}(t))$, where $x_i^{(\ell)}(t)$ stands for the features that we extract from the video in order to predict the line flow at time t for the ith line. Similarly, for the region counts we can give the distribution $p(r_i(t)|x_i^{(r)}(t))$. If we make the simplifying assumption that the line flows for each line and time are predicted independently, the joint predictive distribution over all line flows is also available to us by simply taking the product of each predictive distribution that we get from the line counting regression model. Furthermore, we may also suppose that the predictive distributions coming from the region counting regression models are independent.

By the above assumptions, we can regard the problem from a more abstract perspective, using higher-level random variables that abstract away the specific times, lines and regions and instead stand for whole sequences of the lower-level random variables. In particular, let the variable L denote all line flows over all lines and all times, R all the region counts, $X^{(\ell)}$ all the line counting features in the test sequence and $X^{(r)}$ all the region counting features. Furthermore let S denote the initial region counts in all regions. We know that L and S exactly determine R (denote this functional relationship as $R(L,S)$) and we have the predictive distributions $p(R|X^{(r)})$ and $p(L|X^{(\ell)})$ from the regression models. We are interested in $p(L,S|X^{(\ell)},X^{(r)})$, i.e. the true line flows (and initial region counts) considering all our observations, both from lines and regions. We can derive this distribution as follows:

$$p(L,S|X^{(\ell)},X^{(r)}) \propto p(X^{(\ell)},X^{(r)}|L,S)p(L)p(S) \tag{6.12}$$
$$= p(X^{(\ell)}|L)p(X^{(r)}|R(L,S))p(L)p(S) \tag{6.13}$$
$$\propto \frac{p(L|X^{(\ell)})}{p(L)}\frac{p(R(L,S)|X^{(r)})}{p(R(L,S))}p(L)p(S) \tag{6.14}$$
$$= \frac{p(L|X^{(\ell)})p(R(L,S)|X^{(r)})p(S)}{p(R(L,S))}, \tag{6.15}$$

where all distributions are also implicitly conditioned on the training data. By our assumptions,

$$p(L|X^{(\ell)}) = \prod_{i=0}^{N}\prod_{d\in\{1,-1\}}^{N}\prod_{t=0}^{T} p(\ell_{i,d}(t)|x_i^{(\ell)}(t)) \tag{6.16}$$
$$p(R|X^{(r)}) = \prod_{i=0}^{N-1}\prod_{t=0}^{T} p(r_i(t)|x_i^{(r)}(t)) \tag{6.17}$$

The $p(S)$ and $p(R(L,S))$ express how the initial and later region counts are estimated to be distributed after observing the training set but before seeing the test set.

The fact that $p(R(L,S))$ appears in the denominator of 6.15 is an artifact of the overly strong assumption that the individual predictive distributions that the line counter gives us can be simply multiplied together to yield the joint predictive distribution of all lines at every point in time. The actual joint predictive distribution would have to account for dependencies between the flows over different lines at different times, implicitly enforcing constraints that, for example, keep many people from accumulating in a small region. If we really modeled such dependencies in the line counters, the denominator $p(R(L,S))$ would be necessary to avoid "double counting" of the post-training distribution over region counts: once in the region counter and once implicitly in the line counter. In our case, however, using the denominator would simply reward unexpected region counts, which is clearly undesirable.

We may model $p(S)$ by a normal distribution whose mean is the empirical mean of the region counts in the training set and whose variance is the empirical variance as seen in the training set (multiplied by an adjustable factor). The predictive distributions given by the regressions are already normal. Overall, this means that the negative logarithm of the distribution consists of weighted sums of squared differences, and indeed that is what we required to be minimized in Equation 6.2. In this sense, minimizing that optimization criterion can be loosely interpreted as a maximum a posteriori estimation.

Chapter 7

Evaluation

In this chapter we will look at the evaluation results of the counting methods introduced in the previous chapters. Before giving the actual results, we first describe the datasets and the evaluation metrics that we used.

7.1 Datasets

There are few publicly available annotated videos of crowded scenes where line counting algorithms can be evaluated. In this section we describe one example, the UCSD vidd dataset and we introduce our new, more crowded video called crange_top.

7.1.1 UCSD Vidd

The UCSD vidd1 sequence[CV08] was recorded at a walkway of the University of California, San Diego. It mainly features pedestrians walking individually and with few occlusions. The pedestrians walk at fairly constant speeds and lighting conditions do not vary much (see example frames in Figure 7.1). Therefore it is not a particularly challenging dataset. Nevertheless, we list our results on it, since it is commonly used and it is still informative to see how easy it is to adapt our method to a different scene.

7.1.2 Crange Top

The main video sequence that we use for evaluation is called crange_top and it has been provided by the German Fire Protection Association (see example frames in Figure 7.2. It is a several hours long, full-HD resolution, RGB color video at 25 frames-per-second. We annotated some parts of this video by specifying the full trajectories of the head center-point of each pedestrian. Using these ground truth trajectories, we can determine the pedestrian line flows and region counts for any line or region. We also specify for each person in each frame whether they are occluded, by which we mean that at least 50% of the head is hidden behind something.

Figure 7.1: Example frames from the UCSD vidd dataset.

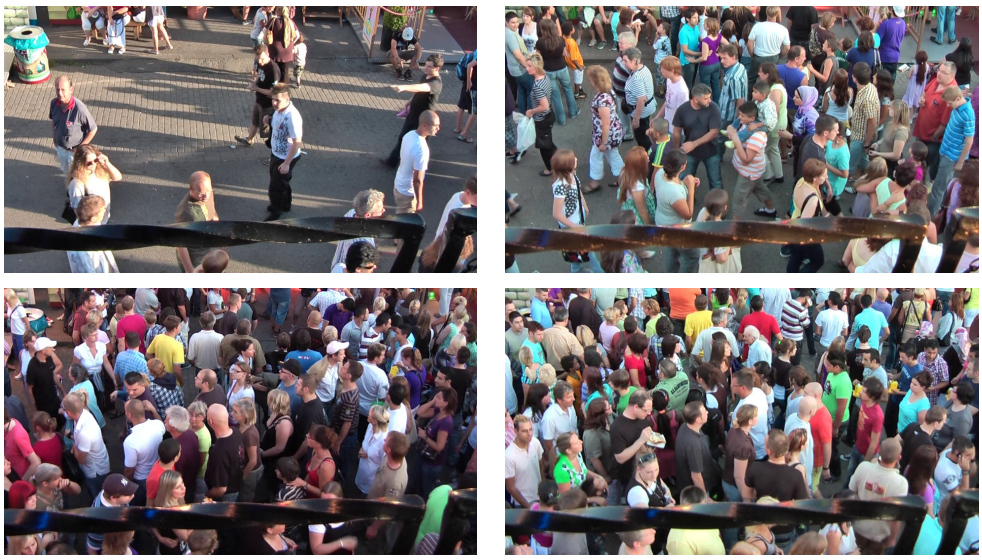

Figure 7.2: Example frames from the crange_top dataset. Note the high level of occlusion and varying crowd density.

Since annotation is very time consuming, we have chosen (for annotation) a 3 minute long continuous part and 14 additional 10-second fragments of the video scattered over a more than 1 hour 40 minutes long timespan. Our goal was to include fragments depicting largely different crowd densities, ranging from less than 20 people per frame to over 90 (as opposed to 6 in the UCSD vidd video). We use 11 of the short fragments as training and 3 as validation (for hyperparameter tuning). The long continuous part is reserved for testing so we can observe how the error develops over extended times.

To decide which fragments to use as training or validation, we grouped them to 3 categories based on crowdedness (average people count per frame), and took one fragment from each for inclusion in the validation set.

More information about the evaluation datasets can be seen in Table 7.1.

Dataset name	Offset in video	Length	Mean count	Mean flow	Unique pedestrians	Occlusion
ucsd_vidd1	n/a	06:40	6.5	63.6	349	n/a
crange_top1	00:14:00	03:00	20.0	43.5	339	9%
crange_top_f11	00:23:25	00:10	17.8	39.3	32	5%
crange_top_f12	00:48:45	00:10	24.1	20.7	37	3%
crange_top_f13 *(v)*	00:40:45	00:10	29.4	45.3	51	5%
crange_top_f14	01:00:25	00:10	35.6	26.7	49	4%
crange_top_f21	00:53:05	00:10	43.0	72.9	69	4%
crange_top_f22	01:26:05	00:10	49.0	66.0	68	6%
crange_top_f23	01:10:45	00:10	50.5	62.7	72	5%
crange_top_f24	01:04:05	00:10	57.3	53.4	76	7%
crange_top_f25 *(v)*	01:03:25	00:10	59.8	70.8	84	9%
crange_top_f31	01:33:45	00:10	65.4	74.7	93	12%
crange_top_f32	01:42:05	00:10	68.2	79.8	87	4%
crange_top_f33 *(v)*	01:35:05	00:10	79.7	58.8	103	8%
crange_top_f34	01:21:25	00:10	86.1	36.9	109	13%
crange_top_f35	01:54:05	00:10	92.7	57.0	118	11%

Table 7.1: Evaluation datasets. Mean count refers the the average pedestrian count in the frames and mean flow is the average of the mean flow to left and to the right right (in pedestrian/minute units). The Occlusion column gives the percentage of pedestrian instances whose head is hidden behind something to more than 50%. The datasets marked with *(v)* are used as validation sets.

7.2 Evaluation Measures

To assess the accuracy of a counting algorithm (or a specific parametrization of it), we need to define precise evaluation measures. Their purpose is two-fold. Most importantly they indicate how well the algorithm is expected to perform in the wild from an application point of view. Additionally, evaluation measures are also needed for internal tuning of the algorithm's hyperparameters. Ideally, an evaluation measure should be relevant, i.e. not far removed from the aim of the system, easy-to-interpret); and generalizable, i.e. descriptive of the algorithm and not the peculiarities of the test set.

7.2.1 Cumulative Measures

In our task, the goal is to yield accurate cumulative line counts after extended periods of time. Therefore, the most relevant (and simplest) evaluation measure is to consider the final cumulative count returned by the algorithm and take its absolute difference from the ground truth value. We can also divide it by the ground truth desired value to get the final relative error. Let $\ell(t)$ and $\hat{\ell}(t)$ denote the ground truth and the predicted instantaneous flow at time t, respectively. The definition of the error measures are then:

Figure 7.3: Partitioning the crange_top dataset. Dark blue marks the 11 training parts (10 seconds each), red denotes the validation parts used for hyperparameter calibration and green is the final test set (3 minutes long). Light blue parts are unannotated.

$$FinalAbsErr = \left| \int_0^T \hat{\ell}(t) - \ell(t) dt \right| \tag{7.1}$$

$$FinalRelErr = \frac{FinalAbsErr}{\int_0^T \ell(t) dt} \tag{7.2}$$

Unfortunately, this measure hides much of the information about how the error develops over time and reduces many different effects to a single number. For example, it cannot differentiate between the case when a single anomalous moment introduces a large error but otherwise the counting is accurate versus the error being evenly distributed over time. It also results in spurious error cancellations, where any over-counting in the beginning can be compensated by any under-counting at distant points in time. While this *is* how cumulative counting behaves "in the wild", these effects make the evaluation less robust and less informative. Error cancellation also means that the result does not grow linearly with time, so simply normalizing by T would create systematic bias that underestimates the accuracy over longer time spans.

At the other extreme, we have the mean instantaneous absolute error, where we take the absolute value inside the integral:

$$MeanInstAbsErr = \frac{1}{T} \int_0^T \left| \hat{\ell}(t) - \ell(t) \right| dt \tag{7.3}$$

The problem with this formulation is that it leaves no room for error cancellation at all. Remember that we are not interested in exact frame-for-frame accurate people counts. If the estimations are time-shifted by some frames, it still performs well and should attain a high evaluation score. The mean instantaneous absolute error would count such discrepancies twice, instead of not at all. Therefore, we should choose a solution between the two extremes by using a fixed-size sliding temporal window, inside of which we allow error-cancellation:

$$MeanWindowAbsErr(W) = \frac{1}{T-W} \int_0^{T-W} \frac{1}{W} \left| \int_w^{w+W} \hat{\ell}(t) - \ell(t) dt \right| dw \qquad (7.4)$$

Unfortunately, this is only informative of the error over a W-sized time window, which, due to the lack of long annotated sequences, must be much shorter than the timespans in a real-life application (seconds versus hours). To estimate the behavior over a longer timespan, we need to model the error probabilistically.

7.2.2 Probabilistic Error Modeling

Assuming that the cumulative error can be modeled as the sum of many independent errors cumulating over time, it exhibits random walk behavior. More precisely, the cumulative error CE after t units of time is distributed normally according to

$$CE(t) \sim \mathcal{N}(t \cdot \mu_{err},\, t \cdot \sigma_{err}^2), \qquad (7.5)$$

where μ_{err} is the mean (signed) error of the prediction over a unit of time and σ_{err} is the standard deviation of the error over a unit of time. Ideally μ_{err} should be zero, but it may happen that there is a discrepancy between the training and the test set making the algorithm systematically over- or under-estimate the pedestrian flow. This may be, for example, due to different crowd density, different occlusion patterns or different flow speed than previously observed. We can estimate these parameters by choosing an appropriate window size W and extracting all time windows with this length and computing the empirical mean and variance of the cumulative error over these windows. The windows should be large enough so that their errors can be assumed to be independent (given μ_{err} and σ_{err}) but small enough so that enough of them can fit into the test set length for their sum to be modelable as normally distributed.

For illustration purposes, we also compute the cumulative absolute error over overlapping temporal windows of various sizes and plot the average of this cumulative absolute error as a function of window size. This curve is useful to gain more qualitative insight but hard to summarize numerically to a single number beyond the μ_{err} and σ_{err} that we discussed above.

Let us now turn back to the mean window absolute error (Equation 7.4). We can reinterpret this as the mean instantaneous absolute error computed on the box-filtered estimation and ground truth:

$$
\begin{aligned}
MeanWindowAbsErr(W) &= \frac{1}{T-W} \int_0^{T-W} \frac{1}{W} \left| \int_w^{w+W} \hat{\ell}(t) - \ell(t) dt \right| dw \\
&= \frac{1}{T-W} \int_{W/2}^{T-W/2} \left| \frac{1}{W} \int_{w-W/2}^{w+W/2} \hat{\ell}(t) dt - \frac{1}{W} \int_{w-W/2}^{w+W/2} \ell(t) dt \right| dw \\
&= \frac{1}{T-W} \int_{W/2}^{T-W/2} \left| (\mathrm{rect}_W * \hat{\ell})(w) - (\mathrm{rect}_W * \ell)(w) \right| dw
\end{aligned}
\tag{7.6}
$$

This means that short-term error-cancellation is achieved by blurring both functions. As a metaphor, we can understand this as if each (infinitesimal) piece of ground truth flow would plant requests in a fixed neighborhood around it and the estimation flows would (independently) plant replies in their own neighborhoods. The error measure is then the amount of miscommunication (i.e. unanswered requests or unrequested replies) over a unit of time. This is a rather crude approach because the two functions cannot "seek out" one another, precisely because this measure handles the two functions independently. In the next section, we discuss an evaluation approach where the "requests" and "replies" find each other adaptively.

7.2.3 Evaluation By Flow-Association (Precision, Recall)

Instead of taking the above, time-centered view on the error, we can also evaluate algorithms using a flow-centered (or pedestrian-centered) approach, where we associate pieces of ground truth flow with pieces of estimated flow. By this, we can quantify how much of the ground truth flow (in turn how many pedestrians) is correctly "found" by the algorithm. This can be viewed as a kind of detection evaluation, where we can compute true positives, false positives and the usual quantities in information retrieval evaluation, such as precision and recall.

This idea is based on the paper of Ma et al.[MC13], who use a so called *recall-distance curve* to describe their line counting results. Their algorithm returns a list of estimated crossing times as opposed to a continuous flow like in our case, so the detection-like interpretation is more natural in their formulation. To create the recall-distance curve, they try to associate each ground truth crossing event to a predicted crossing event within a given time distance and plot the recall as a function of the time distance. The recall is defined as the ratio of the number of true positives and the number of relevant (desired, ground truth) items. In this case it refers to the percentage of ground truth crossings that have successfully been associated to a predicted crossing among all ground truth crossings.

We devised a generalized version of the above approach that is applicable to continuous flow estimations. In this evaluation method, we associate pieces of desired and predicted flows to each other by an iterative approach. Throughout the algorithm, we maintain the frame-for-frame unassociated flows both for the ground-truth and the predicted flows. In the beginning all flows are unassociated. The first step is to take the frame-by-frame minimum (i.e. intersection) of the desired and the predicted flows and consider this shared amount of flow to be associated and thus subtract this amount from the unassociated flows (both predicted and desired).

In subsequent steps, each piece of desired flow starts to "seek out" pieces of prediction flows nearby that it can get associated with. In each step the distance d of this search is incremented by one. More precisely we filter the ground truth flow sequence with the convolution kernel $[1,0,...,0,1]$ (having $2d+1$ elements), resulting in an "association request" array, expressing how much desired flow is ready to get associated with some predicted flow (frame-for-frame). We take the minimum of the unassociated prediction flow and the association request array to determine how much of the requests was answered and can get associated in this step. These are associated with the desired flows that *originally* requested them (i.e. the ones that are at distance d in either time direction of the particular piece of found flow), proportional to their flow values.

The steps are repeated until the distance is equal to a prespecified maximum. Now we can count how much of the desired flows have been associated with predicted flows (by subtracting the sum of the unassociated desired flows from the sum of all desired flows). These are interpreted as the true positive counts when computing the recall (Rec). We can also calculate the precision ($Prec$) and the F1-measure (F_1):

$$F_1 = \frac{2Prec \cdot Rec}{Prec + Rec}, \tag{7.7}$$

7.2.3.1 Handling Negative Predicted Flows

We have no explicit mechanism that would forbid the pedestrian line flow predictor to give negative-valued flows in its prediction. Of course, in information retrieval or in binary classification we never have a negative number of retrieved documents or detected items, so precision and recall could give nonsense results. Therefore we adjust the above explained computations such that negative-valued flows do not cause evaluation problems.

The key idea is that the negative-valued flows should be canceled out using nearby positive-valued flows first. We can accept the negative-valued flow if the long term sums are not affected, i.e. if there is enough positive-valued flow for compensation nearby. This kind of cancellation should have priority over the "requests" of the desired flows. In other words, we should be reluctant to declare a piece of predicted flow as true positive as long as there are negative-valued flows nearby.

In the end, any negative-valued flows that could not be canceled by nearby positive-valued flows get added up (with positive sign) to the amount of predicted flow in the precision formula. This is necessary, so that we can guarantee that the precision will be between zero and one. The precise details can be read in Algorithm 3.

7.3 Hyperparameter Optimization

There are numerous parameters in the methods that we have introduced and described in the previous chapters and we have not defined any formal way to choose them. There are examples among representation parameters, such as the temporal sliding window size, feature parameters, such as the movement segmentation threshold or the optical flow penalty factors, regression hyperparameters such as the noise variance in the Gaussian process, etc. The space of possible combinations is enormous, so one needs to use some heuristic approaches.

Some of these hyperparameters were tuned by direct visual feedback (by creating graphical interfaces) and moving the parameters until the result visually looks the best. These include the optical flow parameters, Canny edge parameters, etc.

For other parameters, we trained different models on the training set and evaluated the results on the validation set using $F_1(37)$, i.e. the F1-measure with maximal flow association distance set to 37 frames (approx. 1.5 seconds at 25 fps). For choosing the parameter settings to try, we used a combination of trial-and-error, creativity and the Python library called Hyperopt created by Bergstra et al.[BYC13] This software can suggest new parameters based on past experiment results, optimizing for expected improvement.

7.4 Line Counting Results

In this section we describe the results of that we obtained for the line counting methods. First, we look at the performance of our window-based method on the challenging crange_top dataset, then we also consider the our baseline of the modified flow mosaicking and the UCSD vidd dataset.

We subdivide the image area to 8 regions and use the 7 inner lines for counting (see Figure 7.4). In training, we build a single model from the data collected over all 7 lines in all the training video(s). We then evaluate the performance of this predictor over the 7 lines in the test set. The evaluation measures are computed by averaging the performance on the 7 lines. For the precision, recall and F1 measures, we use micro-averaging.

Algorithm 3 Evaluation algorithm by flow-association (ℓ is the array of the desired instantaneous flows, $\hat{\ell}$ contains the predicted flows)

function ASSOCIATE-FLOWS($\ell, \hat{\ell}, maxDist$)
  // Yet Unassociated Desired and Unassociated Prediction flows
  $UD \leftarrow \ell$
  $UP \leftarrow \hat{\ell}$

  // Subtract the common part of the prediction and the ground truth
  $UP \leftarrow UP - \min(UP,\ UD)$
  // The desired flow only gets associated with positive parts
  $UD \leftarrow UD - \max(0,\ \min(UP,\ UD))$
  // Negative parts go to a different buffer, they will have priority in canceling with positive flows
  $UN \leftarrow -\min(0,\ \min(UP,\ UD))$
  for $dist \in \{1, \ldots, maxDist\}$ **do**
    // Create "requests" by filtering with array of size $2 \cdot dist + 1$
    $R_d \leftarrow UD[t] * [1, 0, ..., 0, 1]$
    $R_n \leftarrow UN[t] * [1, 0, ..., 0, 1]$

    // The formerly negative flows in N have priority in canceling with UP
    $A_n \leftarrow \min(UP,\ R_n)$
    $UP \leftarrow UP - A_n$

    $A_d \leftarrow \min(UP,\ R_d)$
    $UP \leftarrow UP - A_d$

    // Let each requester get their proportional part of the respective A
    // (elementwise operations, $[a:b]$ denotes range indexing)
    $UN[dist:T] \leftarrow A_n[0:(T-dist)] \cdot UN[dist:T] / R_n[0:(T-dist)]$
    $UN[0:(T-dist)] \leftarrow A_n[dist:T] \cdot UN[0:(T-dist)] / R_n[dist:T]$
    $UD[dist:T] \leftarrow A_d[0:(T-dist)] \cdot UD[dist:T] / R_d[0:(T-dist)]$
    $UD[0:(T-dist)] \leftarrow A_d[dist:T] \cdot UD[0:(T-dist)] / R_d[dist:T]$
  end for
  // TP is the amount of true positive (correctly found) flow
  $TP \leftarrow \sum_t \ell[t] - UD[t]$
  $Rec \leftarrow TP / \sum_t \ell[t]$
  $Prec \leftarrow TP / (\sum_t \hat{\ell}[t] + UN[t])$
  return $Prec,\ Rec$
end function

Figure 7.4: The counting lines used for the crange_top dataset.

7.4.1 Our Method on Crange Top

After tuning the hyperparameters of our window-based method on the crange_top dataset's validation part, we have found the following settings to yield good performance:

- Temporal window: size: 20, step: 6
- Feature extraction blocks: 5 blocks with 50% overlap between two blocks (as in 4.1, but with 5 blocks instead of 3).
- Features: the Canny feature had to be removed as it worsened the prediction quality. We use the average perpendicular (horizontal) optical flow component feature and the foreground pixel count feature in each block.
- Regression: The noisy-input Gaussian process model gives the best performance (hyperparameters $C = 10^{10}$, $\sigma_i = 0.0044$, $\gamma = 0.02$). See Table 7.2 for a comparison with classic Gaussian process and linear ridge regression. For better interpretability of the results, we also included the performance of a "Constant" regressor, which simply computes the mean of the training targets and outputs this constant value at test time.

The results on the three validation sets with this best configuration are shown in Figures 7.5, 7.6 and 7.7. Keeping these parameters, we trained a new model using both the training and the validation sets as training data and evaluated the model on the test set. The result can be seen in Figure 7.8

The quality measures based on our flow association scheme can be seen in Figure 7.9. Since the sliding window size used in training is 20, it is no surprise that these measures approximately saturate around the distance of 20 as well. In other words,

	Prec	*Rec*	F_1	*FinalAbsErr*	*FinalRelErr*	μ_{err}	σ_{err}
Linear ridge	0.828	0.894	0.859	1.695	17.13%	0.0044	0.145
Classic GP	0.831	**0.908**	0.868	1.501	15.48%	0.0051	0.135
Noisy-input GP	**0.855**	0.899	**0.877**	**1.236**	**12.62%**	0.0032	**0.133**
Constant	0.805	0.828	0.816	2.614	30.28%	**0.0030**	0.174

Table 7.2: Comparing the validation set performance of different regression algorithms (crange_top dataset). The precision, recall and F1-measure are computed with $d =$ 37 frames maximum association distance. The μ_{err} and σ_{err} error characteristics are estimated using windows of size 50.

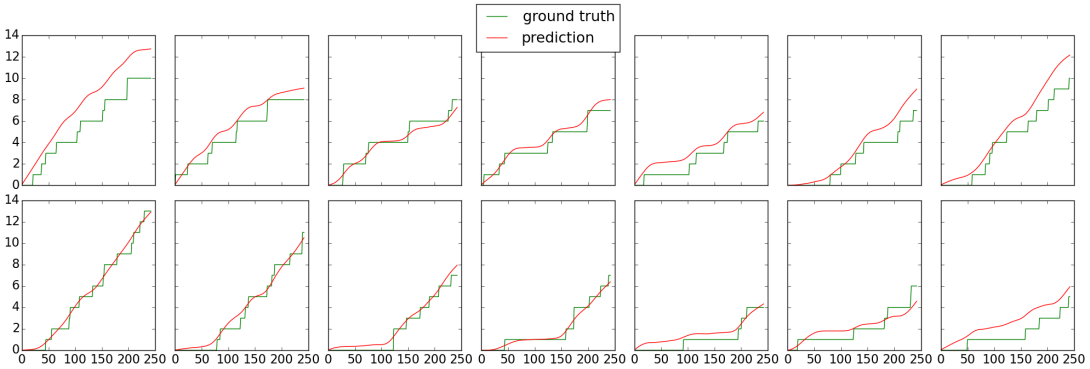

Figure 7.5: Results on the validation set crange_top_f13 (low crowdedness). Settings according to the best validation parameters. Each column shows the results for one of the 7 lines as shown in Figure 7.4. The top row shows the leftward moving counts the bottom shows the rightward counts.

the learning method has no effective resolution capability below this size, and further associations above this size are mostly coincidences and are therefore rare.

The mean window error curves (mean absolute window error plotted against window size) in Figures 7.10 and 7.11 show that the error on the crange_top dataset develops approximately according to the shape of a square root function, which is consistent with the hypothesized random walk behavior. The ends of the graphs show some anomalies but those estimations are also less reliable, since there are few very long windows to evaluate. By contrast, the UCSD vidd dataset shows almost linear growth. It is also interesting to note that the counts for left and right seem to deviate to the opposite directions from a smooth and more natural looking curve. This may show that the gained accuracy in one direction is balanced by a loss of accuracy in the other.

7.4.1.1 Data Augmentation

Mirroring the datasets creates twice as many training examples and may help avoid overfitting. We evaluated our best solution with and without additional mirrored data,

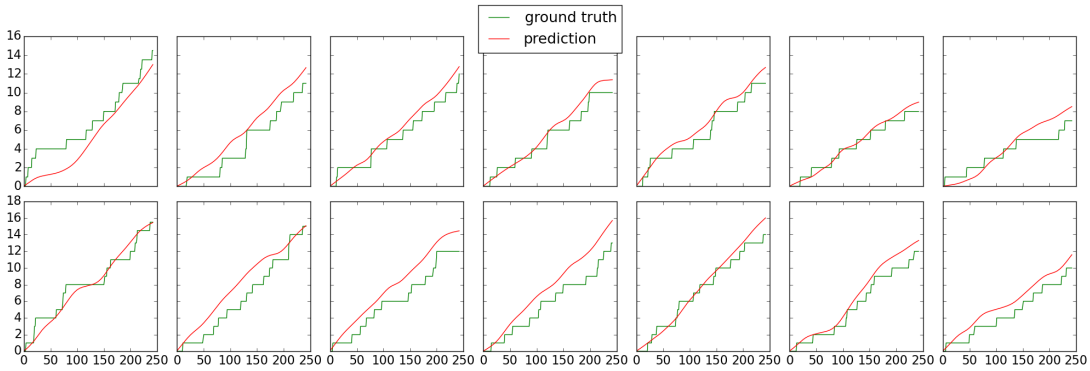

Figure 7.6: Results for crange_top_f25 (medium crowdedness), analogous to Figure 7.5.

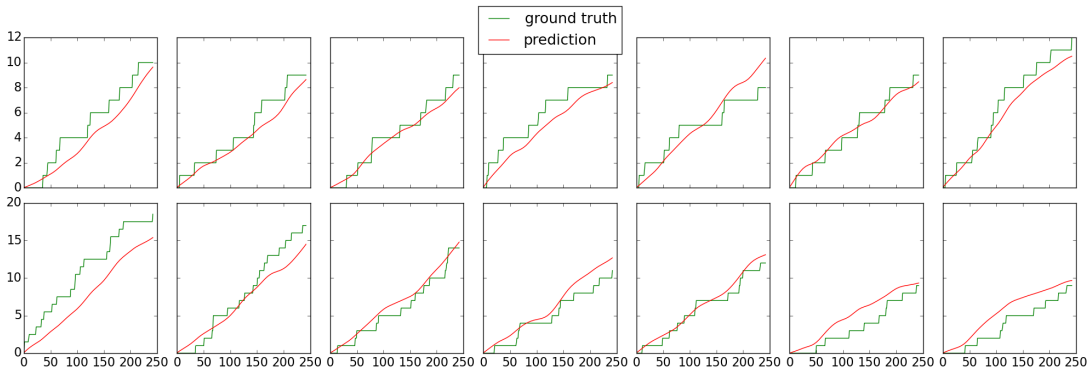

Figure 7.7: Results for crange_top_f33 (high crowdedness), analogous to Figure 7.5.

the results (Table 7.3) show that such mirroring is helpful overall.

	Prec	*Rec*	F_1	*FinalAbsErr*	*FinalRelErr*	μ_{err}	σ_{err}
Mirroring	**0.808**	0.857	**0.832**	**8.042**	**6.56%**	**0.001556**	0.123
No mirroring	0.774	**0.890**	0.828	19.970	15.91%	0.004165	**0.121**

Table 7.3: Comparing the test set performance on crange_top, with and without mirroring augmentation. The other settings are according to the best validation set performance (F1 measure).

7.4.2 Results on UCSD Vidd and Comparison with Flow Mosaicking

We also try our method on the UCSD vidd dataset. The 4000 frames are split to a 2000 frame training, a 1000 frame validation and a 1000 test part. Compared to the parameters used on crange_top, we only tweaked the low-level preprocessing parameters

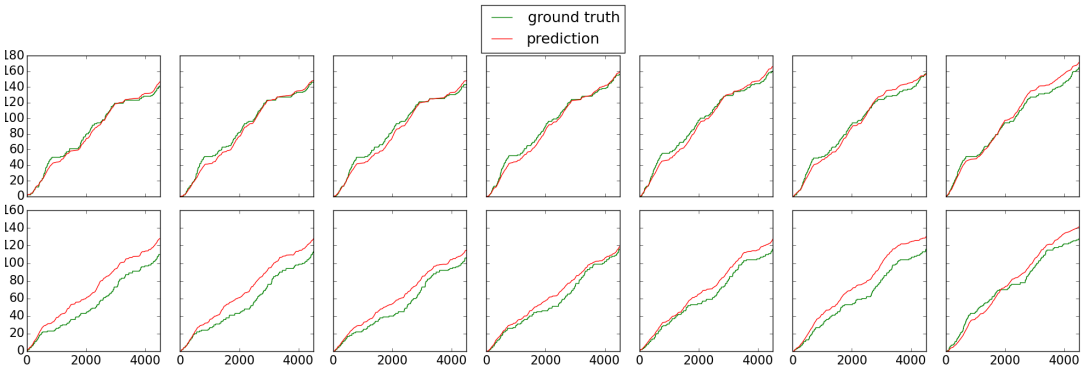

Figure 7.8: Results on the three-minute-long test set crange_top_1. Settings are chosen according to the validation set parameters. The training and the validation datasets were all used in the training of this model (of course the test set was not). Columns and rows are as in Figure 7.5.

controlling the line optical flow and the background subtraction. The other parameters (such as temporal window size, regression parameters, etc) were left at the values that worked best on the crange_top validation set.

The result on the validation part of the UCSD vidd dataset can be seen in Figure 7.12. The parameters worked very well, despite being derived from another dataset. We could not significantly improve on this result on the UCSD vidd validation set, so we used the same settings for the test as well.

The performance dropped on the test part of the video (last 1000 frames), as seen on Figure 7.12. The large error in the rightwards counts is caused by a cyclist who provided an unusually high optical flow value. Since only a few cyclists appear throughout the video, the system could not learn to handle these anomalies properly (one may solve this like [CV12], by creating 4 segments: slow left, fast left, slow right, fast right).

Our flow mosaicking implementation (as described in Chapter 3) yields the results seen in Figure 7.14. These results are considerably lower quality than the results using our window-based method. This is mainly due to the fact that some pedestrians are not even seen by the training procedure when there is no corresponding foreground blob, in which their annotation point would appear. This has been pointed out by Ma et al.[MC13] as one of their criticisms of blob-based methods. In their experience, this effect sometimes even goes unnoticed because some researchers only evaluate the method with respect to the the blobs that were found and not on the total number of people crossing. The effect is also seen from high precision (94.2%) and low recall (66.1%) values. This indicates that there are many unseen ground truth pedestrian crossings, but when the system returns a detection, then it is dependable. One could possibly improve this situation by associating ground truth pedestrian line crossings to nearby blobs, even if the point of the annotation is not part of the blob on the

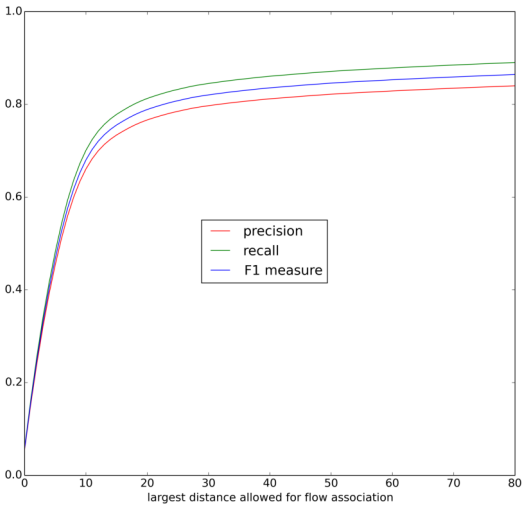

Figure 7.9: Precision, recall and F1 measure against the maximum allowed distance in flow association. Generated on the crange_top test dataset with our best method.

spatiotemporal slice.

The flow mosaicking method was unable to find meaningful blobs in the crange_top dataset and its results are omitted here.

7.5 Region Counting Results

We applied the region counter described in Chapter 5 to both datasets. Due to the high number of training samples (number of frames times number of regions), we used the linear ridge regression method to learn the mapping from the features to the region counts. Figures 7.15 and 7.16 show the results graphically, while Table 7.4 contains the quantitative results averaged over the regions.

The UCSD vidd dataset could be learned very accurately, while there are large errors for crange_top. This may be due to the fact that this video includes many pedestrians who stop to look around and hence disappear from the foreground mask, but it is also the consequence of the overall higher level of occlusion.

	Mean absolute error	Mean relative error
ucsd_vidd	0.194	0.279
crange_top	1.702	0.833

Table 7.4: Region counting results

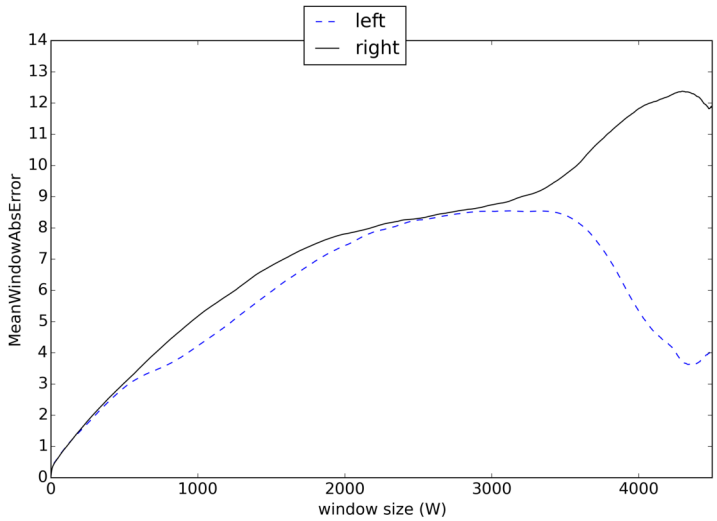

Figure 7.10: Mean window absolute error on the crange_top test dataset, as a function of the window size.

7.6 Comparison and Combination of Line and Region Counts

We examined the results of aggregating the line counts in order to produce region counts (See Figures 7.17 and 7.18). This is done by summing up the incoming flows and subtracting the outgoing flows for each region over time. Error drifting effects are clearly visible, although the shapes of the curves roughly resemble the ground truth, at least in certain parts of the data.

We attempted to use the method described in Chapter 6 to combine the region counts and the line counts into a single consistent solution. Unfortunately, it does not work as envisioned. The discretized numerical optimization does not find the optimal solution (see Figure 7.19). This is clear from the fact that both the left and the right combined cumulative flow functions deviate towards the same direction from the originally predicted (black) line. In the found solution we could trivially increase all flow estimations, bringing them nearer to the originally predicted flows, while keeping the net flow fixed. Therefore, the problem must lie in the optimization process.

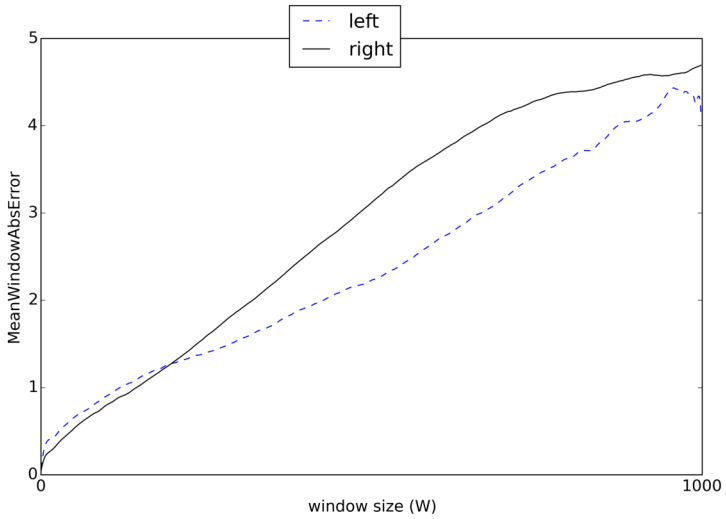

Figure 7.11: Mean window absolute error on the UCSD vidd test dataset with our method, as a function of the window size.

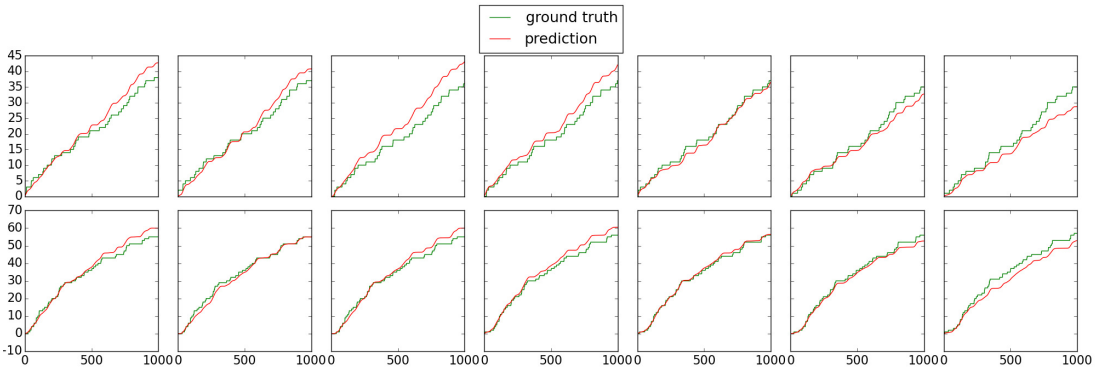

Figure 7.12: Results on the UCSD vidd validation dataset using our method, where the high-level parameters were set to the values that worked best on the crange_top dataset.

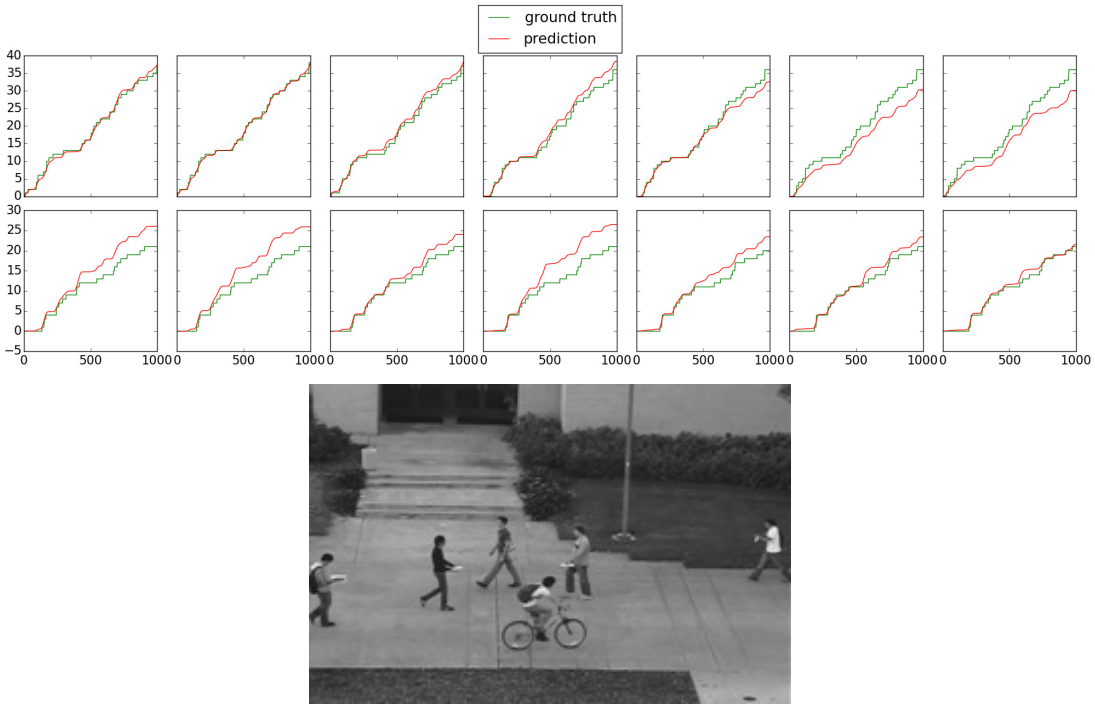

Figure 7.13: *Top:* Results on the UCSD vidd test dataset using our method. *Bottom:* Frame 407 of the test video.

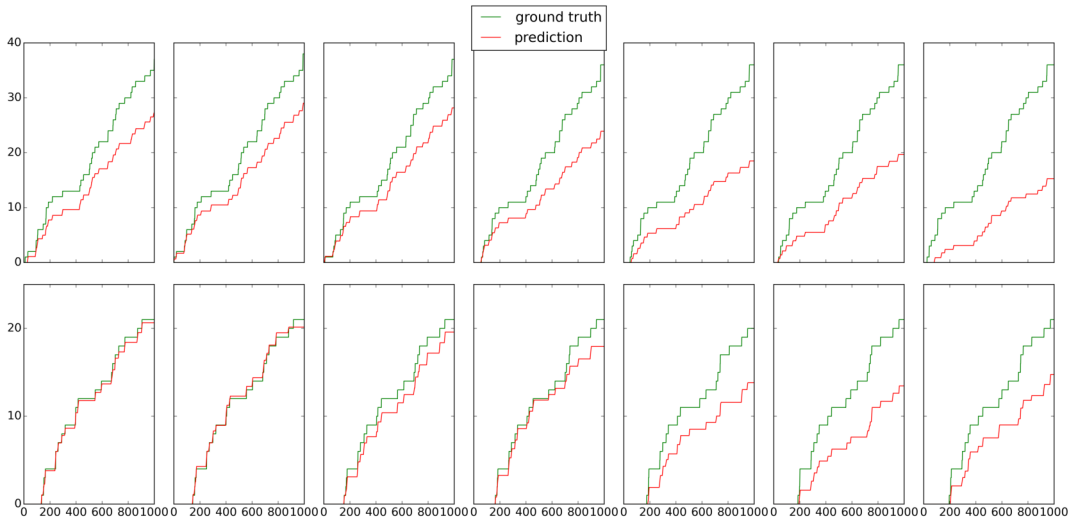

Figure 7.14: Results on the UCSD vidd test dataset using the flow mosaicking method.

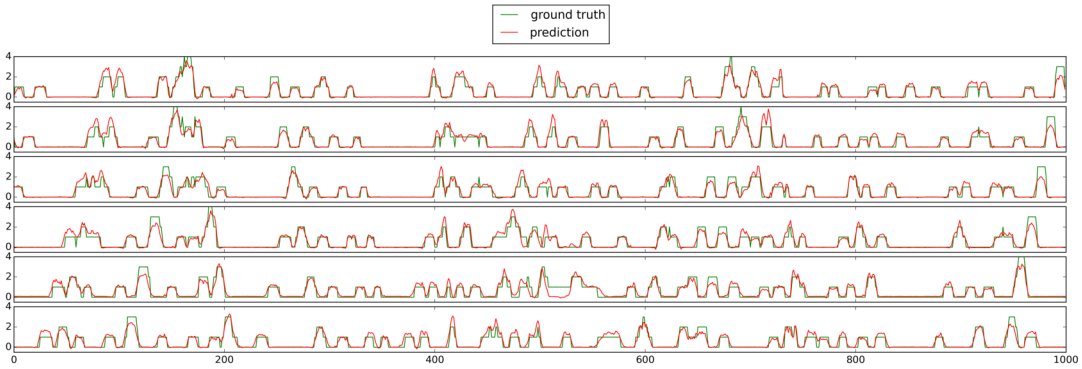

Figure 7.15: Region counting results on the UCSD vidd test dataset. Each row shows the counts for one region (of the 6 regions inbetween the 7 lines).

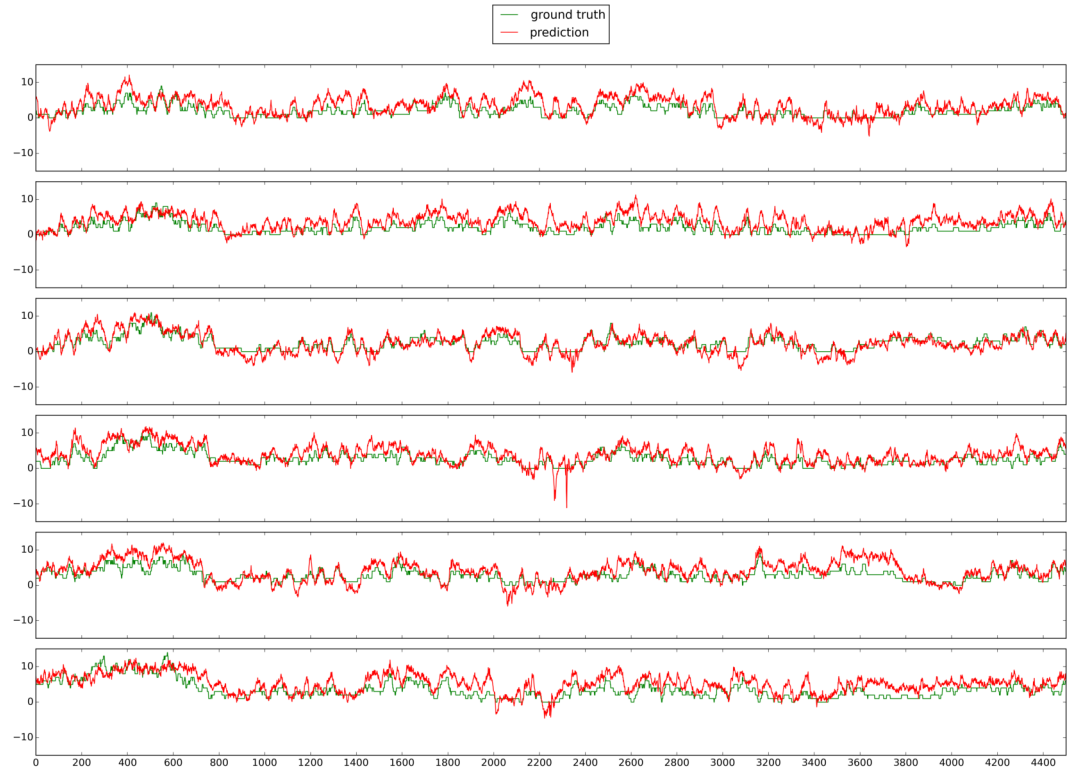

Figure 7.16: Region counting results on the crange_top test dataset.

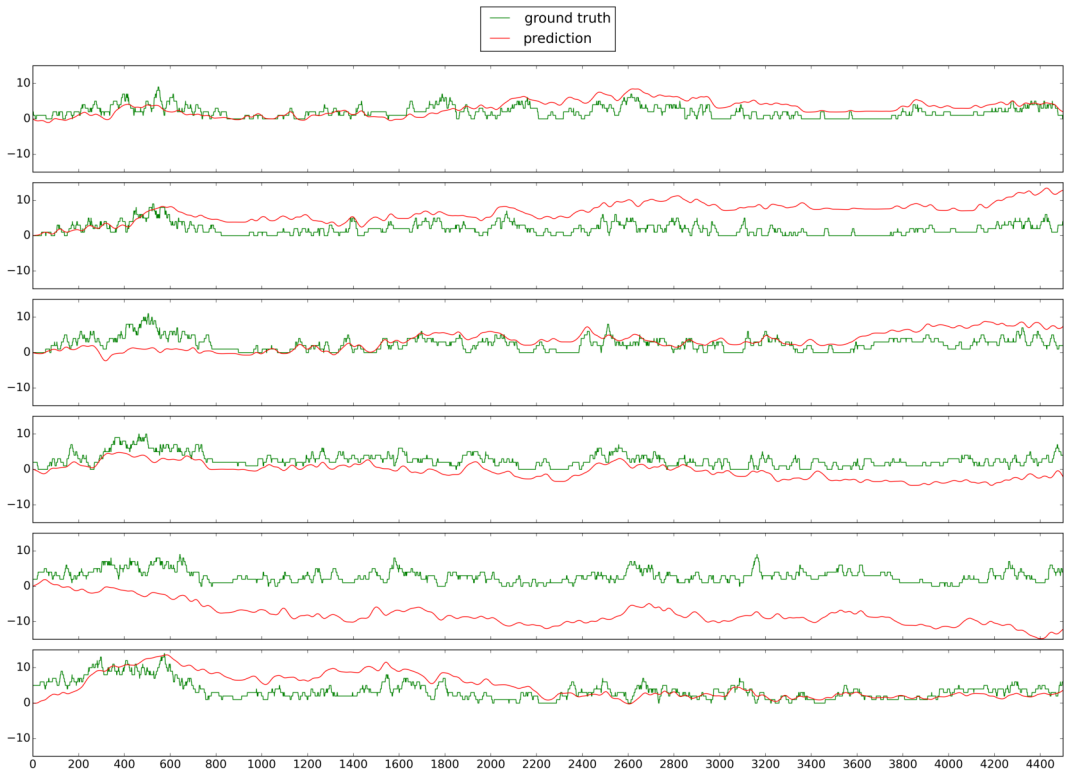

Figure 7.17: Region counts computed by aggregating the line counts on the crange_top test dataset.

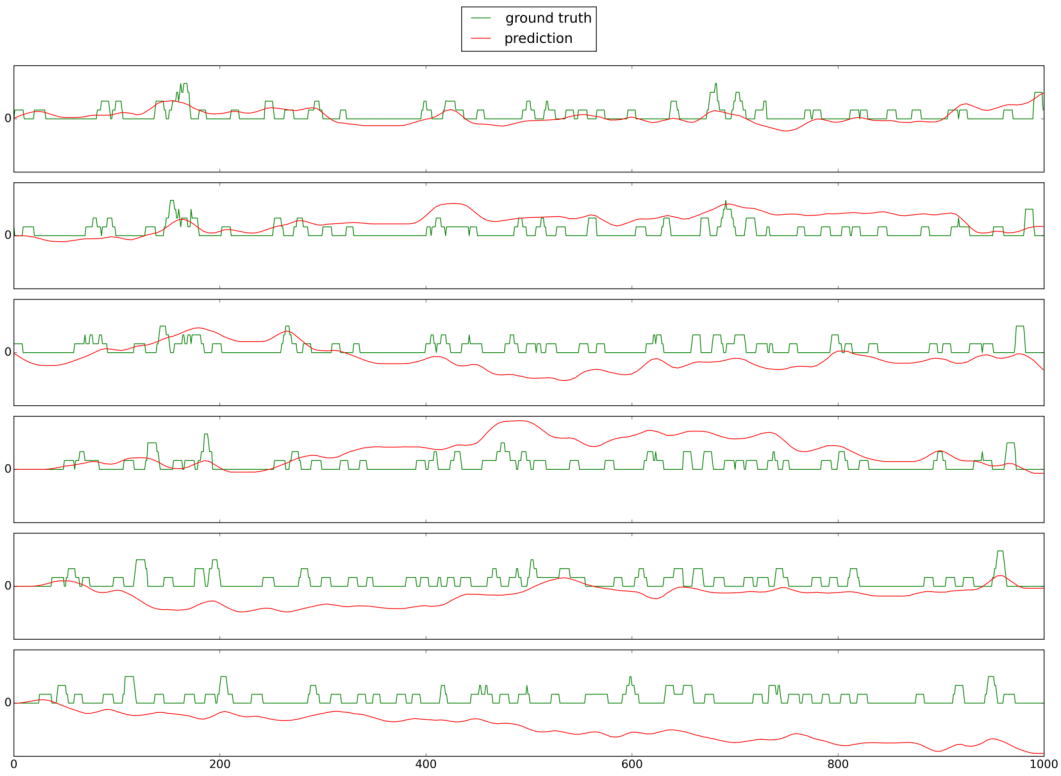

Figure 7.18: Region counts computed by aggregating the line counts on the UCSD vidd test dataset.

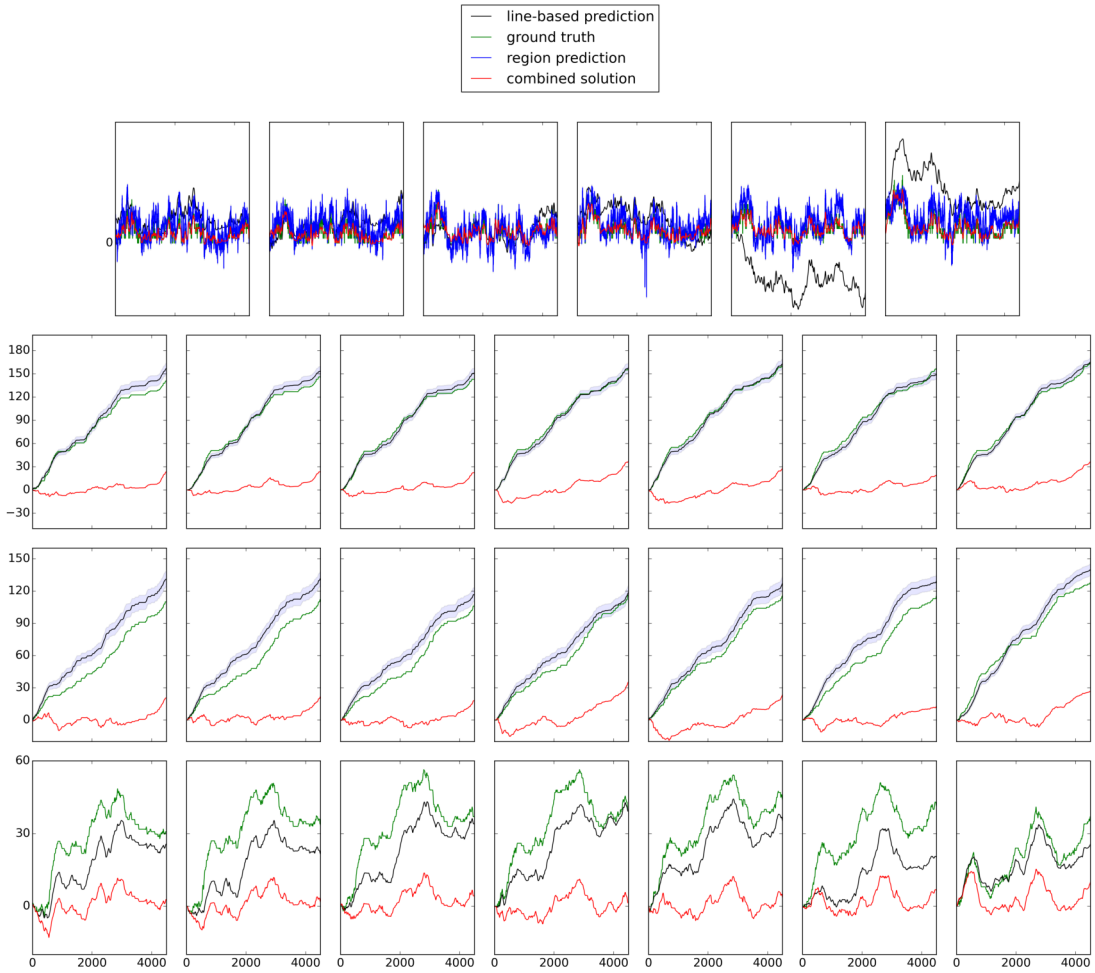

Figure 7.19: Failed attempt at the combination of line and region counts. *Top row:* Region counts, where the black line is the reconstruction from the line counts, the blue is the region counter's output, green is the ground truth and red is the combined solution. *Middle two rows:* Left and right cumulative people flows. *Bottom row:* The cumulative net flows leftwards (i.e. the difference between the curves in the second and the third row).

Chapter 8

Conclusions

In this thesis we examined the topic of pedestrian line counting in the presence of high occlusion and dense crowds. We proposed, implemented and evaluated a new spatiotemporal slice-based method that works with the simple low-level features of optical flow and background subtraction. The method provided very good results on crange_top, a new and more challenging dataset than those that have been previously used for line counting in the literature. Presumably due to the very simple and general nature of the features we used, the method also performed reasonably well on the UCSD vidd dataset without specifically tuning the high-level parameter settings. This shows the robustness of our approach.

We also considered a setup with several parallel counting lines and regions defined between the lines. While we were able to learn line and the region counts separately, our idea for their combination has not produced the hoped result of mutual improvement. Further research is necessary for creating a robust combination method.

8.1 Future Work

Future work regarding these topics should target the following:

- **Annotate more data**. We have only annotated a fraction of the crange_top dataset. With more data, we could probably improve the results significantly.

- **Design more features**. Designing more and better features is crucial to improving prediction quality. It would be especially important to look at texture features that could respond to shapes that usually appear on pedestrians, such as the circular head or the characteristic shape of shoulders or feet.

- **Line and region combinations**. We should further explore the theoretical underpinnings for combining the two kinds of measurements. The formulation in Chapter 6 still has several shortcomings, even apart from the numerical optimization difficulties.

- **Examine and improve generalization and scene invariance**. It is a promising fact that the same high-level parameters could be used for both of the datasets filmed from very different distance, having different resolution, etc. It should be further investigated how far this extends and whether scene specificities could be learned by unsupervised learning in order to avoid the need for manual annotations.

Bibliography

[aMCLV99] a.N. Marana, L. Da Fontoura Costa, R.a. Lotufo, and S.a. Velastin. Estimating crowd density with Minkowski fractal dimension. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 6, pages 3521–3524. IEEE, 1999.

[BMB08] Javier Barandiaran, Berta Murguia, and Fernando Boto. Real-Time People Counting Using Multiple Lines. In *2008 Ninth International Workshop on Image Analysis for Multimedia Interactive Services*, pages 159–162. IEEE, 2008.

[BN06] Cm Bishop and Nm Nasrabadi. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[BPW04] Thomas Brox, Nils Papenberg, and Joachim Weickert. High Accuracy Optical Flow Estimation Based on a Theory for Warping. *Computer Vision - ECCV 2004*, 4(May):25–36, 2004.

[BYC13] James Bergstra, D Yamins, and D Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proceedings of the 30th International Conference on Machine Learning*, pages 115–123, 2013.

[CCC06] TH Chen, TY Chen, and ZX Chen. An intelligent people-flow counting method for passing through a gate. *Robotics, Automation and . . .*, pages 1–6, December 2006.

[CLGX12] Ke Chen, CC Loy, S Gong, and T Xiang. Feature Mining for Localised Crowd Counting. *BMVC*, 2012.

[CLV08] Antoni B. Chan, Zhang Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[CV08] Antoni B. Chan and Nuno Vasconcelos. Modeling, clustering, and segmenting video with mixtures of dynamic textures, 2008.

[CV12] AB Chan and Nuno Vasconcelos. Counting people with low-level features and Bayesian regression. *Image Processing, IEEE ...*, 21(4):2160–2177, 2012.

[DV05] Anthony C Davies and Sergio A Velastin. A progress review of intelligent CCTV surveillance systems. *Proc. IEEE IDAACS*, pages 417–423, 2005.

[EHD00] Ahmed Elgammal, David Harwood, and Larry Davis. Non-parametric model for background subtraction. *Computer Vision—ECCV 2000*, 1843:751–767, 2000.

[Elg11] A. Elgammal. Figure-ground segmentation - pixel-based. In Thomas B. Moeslund, Adrian Hilton, Volker Krüger, and Leonid Sigal, editors, *Visual Analysis of Humans*, pages 31–51. Springer London, London, 2011.

[FCD+11] Joseph Ferenbok, Andrew Clement, A Doyle, R K Lippert, and D Lyon. Hidden Changes: From CCTV to Smart Video Surveillance. *Eyes Everywhere: The Global Growth of Camera Surveillance, A. Doyle, RK Lippert, and D. Lyon, Abingdon: Routledge*, pages 218–233, 2011.

[HS81] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. In *1981 Technical Symposium East*, volume 17, pages 319–331. International Society for Optics and Photonics, 1981.

[HSD73] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, (6):610–621, 1973.

[LKK07] GG Lee, B Kim, and WY Kim. Automatic estimation of pedestrian flow. *... Cameras, 2007. ICDSC'07. First ACM ...*, pages 291–296, September 2007.

[MC13] Zheng Ma and Antoni B. Chan. Crossing the Line: Crowd Counting by Integer Programming with Local Features. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2539–2546. IEEE, June 2013.

[MR11] Andrew McHutchon and Carl E Rasmussen. Gaussian Process Training with Input Noise. In J Shawe-Taylor, R S Zemel, P L Bartlett, F Pereira, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1341–1349. Curran Associates, Inc., 2011.

[Ora11] Jonathan Oram. Balancing surveillance between needs of privacy and security. 2011.

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). December 2005.

[SPH+05] Andrew Senior, Sharath Pankanti, Arun Hampapur, Lisa Brown, Ying Li Tian, Ahmet Ekin, Jonathan Connell, Chiao Fe Shu, and Max Lu. Enabling video privacy through computer vision. *IEEE Security and Privacy*, 3(3):50–57, May 2005.

[TKBM99] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: principles and practice of background maintenance. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 255–261 vol.1. IEEE, 1999.

[WLC10] Wenhua Ma, Lei Huang, and Changping Liu. Crowd density analysis using co-occurrence texture features. In *5th International Conference on Computer Sciences and Convergence Information Technology*, pages 170–175. IEEE, November 2010.

[YGYB14] Zhongjie Yu, Chen Gong, Jie Yang, and Li Bai. Pedestrian counting based on spatial and temporal analysis. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2432–2436. IEEE, October 2014.

[YHSY09] Yang Cong, Haifeng Gong, Song-Chun Zhu, and Yandong Tang. Flow mosaicking: Real-time pedestrian counting without scene-specific learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1093–1100. IEEE, June 2009.