Learning New Features for Effective Dense Matching Using Meta-Level Deep Neural Networks

ZHANG Feihu

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

Supervised by

Prof. Benjamin W. Wah

Abstract of thesis entitled:

Learning New Features for Effective Dense Matching Using Meta-Level Deep Neural Networks
Submitted by ZHANG Feihu
for the degree of Master of Philosophy
at The Chinese University of Hong Kong in XX 2017

In dense Matching (including stereo matching and optical flow), nearly all existing approaches are based on simple features, such as gray or RGB color and gradient or simple transformations like census, to calculate matching costs. These features do not perform well in complex scenes that may involve radiometric changes, noises, overexposure and/or textureless regions. Various problems may appear, such as wrong matching at the pixel or region level, flattening/breaking of edges and/or even entire structural collapse.

In this thesis, we first propose two fundamental principles based on the consistency and distinctiveness of features to identify the real good feature for dense matching and show that almost all existing problems in dense matching are caused by features that violate one or both of these principles. Then, to systematically learn good features for dense matching, we formulate these two principles as a general multi-objective optimization for the learning system and studied the trade-off between the two principles. Finally, to overcome the problem that traditional learning systems (*e.g.* convolutional neural

networks) only tries to find a model with fixed parameters that optimize the average behavior over all inputs, without considering data-specific properties (*e.g*. variances in texture or illuminance conditions), we proposed a new meta-learning method which could learn and assimilate continuous meta-knowledge into the base model to make the weights/parameters adaptive to different inputs without losing generalization capability. The proposed meta-level learning strategy is applied to deep neural networks (DNNs) which finally works as the feature extractor to find the effective features for dense matching.

By using two-frame optical flow and stereo matching as applications, we verify the effectiveness of the proposed feature principles and objective model. Our experimental results show that the features learned can significantly improve the performance of state-of-the-art dense matching algorithms and based on the KITTI benchmarks, our method ranks first on the two stereo benchmarks by the end of 2016 and is the best among existing two-frame optical-flow algorithms on flow benchmarks. Besides, by comparing with the state-of-the-art DNN architectures (*e.g*. inception net, residual net *etc*.), the feature extractors implemented by our MLNN achieves lower error rates in both stereo matching and optical flow.

Acknowledgement

I would like to express my sincere gratitude and appreciation to my supervisors, Prof. Benjamin W. Wah. I gain too much from their guidance not only on knowledge and attitude in doing research, but also on the presentation, teaching, and English writing skills. I will always be grateful for their supervision, encouragement and support at all levels.

I am grateful to my thesis committee members, Prof. Jiaya JIA and Prof. Hanqiu SUN for their helpful comments and suggestions about this thesis. My special thanks to Prof. Long QUAN who kindly served as the external committee for this thesis.

I also thank my predecessors, Jingxi Xu, Weicong Liu, Furui Liu and many others who provide all kinds of helps during my Mphil study.

Last but not least, I want to thank my families. Without their deep love and constant support, this thesis would never have been completed.

Contents

List of Figures

List of Tables

Chapter 1

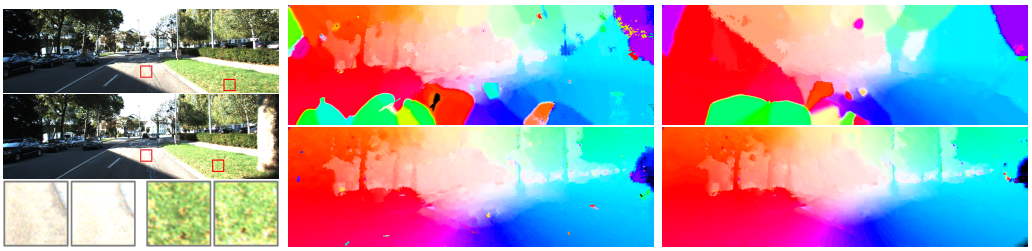
Introduction

In this chapter, we give an overview of the feature extraction in dense matching and focus on the challenges, motivations and our major contributions to this topic. After that, we describe the organization of the thesis.

1.1 Motivation

Stereo matching, optical flow and other dense-matching applications have always been hot issues in computer vision. In the past, a number of methods have been developed to solve these problems. These methods consist of three steps: extracting features and their descriptors, computing the matching cost and/or aggregation [19,49,57,75], and applying matching algorithms [8, 25, 65] to minimize some energy functions.

In recent years, there is a lot of attention on the last two steps. However, little has been done on feature extraction that is critical in dense-matching. The most popular features for stereo matching and optical flow are still limited to some kind of color space or gradi-



(a) Inputs examples (b) CostFilter [57] (c) PMBP [8]

Figure 1.1: An illustration of the Consistency Principle using optical flow as an example. Results on flow field are visualized by the color coding technique used in [18]. (a) Input views (from KITTI dataset [18]). Enlarged red windows, illustrating color inconsistencies, are usually caused by radiometric variations, view-angle changes and over exposure. (b) Original CostFilter [57] with simple color+gradient features and the improved results using our proposed feature. (c) Original and improved PMBP [8] by using 32-channel fast-flow features to improve these matching methods. (See Section 5.2 for a description of the features.)

ent values [26]. Although these simple features are fast to compute and flexible (like in scaling and subpixel interpolation), they are easily influenced by radiometric changes, noise, overexposure and the scene environment (as shown in Fig. 1.1). This is also the major reason why some of the best methods that work well on benchmarks of simple indoor scenes [59] report limited success on benchmarks of complicated outdoor scenes [18]. On the other hand, the popular sparse features used for shape matching and object detection (including SIFT [47] and SURF [7]) which are scale and radiometric invariant met their limitations on performance improvement and flexibility when directly applied to dense matching. These methods were not designed for dense matching from the beginning, and they usually involve a complex step on label densifying [79].

Instead of developing new features for dense matching, some recent methods [17,76] introduce convolutional neural networks (CNNs) to compare the similarity of a pair of patches and use the similarity

score as the matching cost. These help achieve high accuracy when used in some stereo matching methods [76]. To address their high computational cost, Zbontar *et al*. proposed a faster framework with some sacrifice in accuracy [77]. However, as its time complexity depends on the displacement space, it still cannot be used for optical flow and other complex algorithms, such as continuous matching with slanted surface or subpixel accuracy. (The time complexity is $O(KNM)$, with size K of displacement space, N pixels, and computation complexity M of CNN.)

The primary problem in developing better dense matching algorithms is to identify good features. There has been little work in this area. A direct approach [26, 27] collects the error rates when employing one type of features in a specific algorithm. The rates, however, are not useful for designing feature extractors because they cannot provide quantitative information on the features of each pixel and/or region. Also, it is impractical to use them as targets because not only is it time consuming to run matching algorithms during feature extraction, the error rate of one matching algorithm cannot represent the feature's performance in other algorithms.

1.2 Principles to Identify Good Features

To identify the real good feature for dense matching, we identify two fundamental principles on good features that each pixel should possess in order to be effective for dense matching. These principles help understand the requirements on good features for dense matching, as well as identifying the weaknesses of existing algorithms (as they violate one or both of these principles).

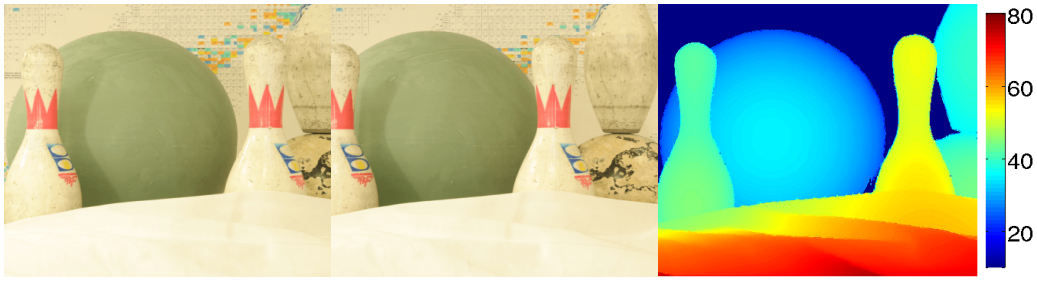
The first principle, the *Consistency Principle*, states that a feature point (a pixel/location where the feature performs well) should own the same or similar feature descriptors (such as RGB values when color is used as the feature) when it appears in different image views (such as the left and right views of a stereo pair). For example, many existing features like color or gradient are highly influenced by noise, radiometric variance, scaling change, translation and/or rotation. As illustrated in Fig. 1.1, such external disturbances can easily break the consistency of features between different views.

The second principle, the *Distinctiveness Principle*, states that a feature point should be different enough with respect to other points/pixels in its surrounding regions. For instance, when using color as the feature, pixels at a corner are unique, whereas those in smooth regions are not distinct. In large smooth regions, the principle is violated in many state-of-the-art features [8, 9, 43] because the color of all the pixels in these regions are very similar. As illustrated in Fig. 1.2, the lack of distinctiveness lead to wrong matches in these regions.

The above two principles can guide us in finding good features. To facilitate the search of such features, we formulate a multi-objective optimization that incorporates both principles as objective:

$$\min_{\Phi \in C} F(\mathbf{X}, \Phi), \tag{1.1}$$

where, $\mathbf{X}$ is the training data/images, Φ is the feature extractor and C is the search space for Φ. The definition of the objective function F gives the answer to the question that what is the real good feature.



(a) Left View (b) Right View (c) Ground Truth

(d) PatchMatch [9] (e) MC-CNN [76] (f) [9] with Our Feature

Figure 1.2: An illustration of the Distinctiveness Principle using stereo matching as an example. (a) Left and (b) right views from Middlebury dataset [59]. (c) Ground truth (with disparity values visualized). (d) Original patchmatch stereo [9] with color+gradient features. (e) Poor subpixel accuracy produced by MC-CNN [76] that cannot be used for continuous stereo matching. (f) Significant improvements when our 32-channel fast stereo features are embedded in the original patchmatch algorithm [9].

1.3 Meta-Level Learning for Feature Extraction

Above model can be used to systematically learn good features. While, in such a model Eq.(1.1), the feature extractor Φ is another key element. It defines the searching space of the solutions and would heavily influence the features/solutions' performance. In the traditional learning systems (*e.g.* CNNs), the model are always trained by optimizing the objective across all training data in order to find a model with fixed structure and weights. It only optimizes the average behavior across all training data, without specializing to their variations. That is in the above model, it's impossible for one kind

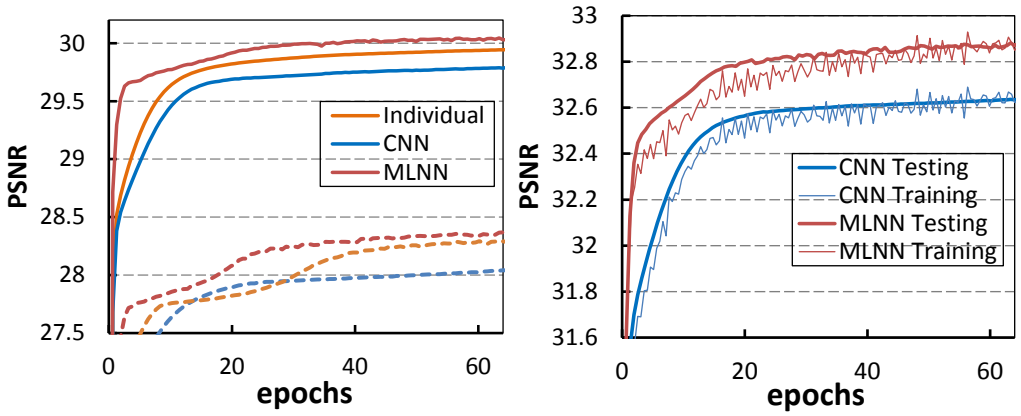


Figure 1.3: Using image super-resolution (Appendix A) to illustrate the benefit of dynamic model adaptation to inputs. *Left*: Test errors for each type of inputs with a mixture of nature images (solid lines) and screenshots (dashed lines) as training data. Our proposed meta-level system (MLNN) (red lines) with model adaptation performs better than training after static manual classification (orange lines) and original CNNs without data classification (blue lines). *Right*: Training and testing results for a mixture of five types of images.

of Φ to perform well in all regions with different texture and illuminance conditions.

As a comparison, when we manually classify training data into multiple classes according to their meta-level differences, and train each to learn a unique model, the results can be significantly improved (as shown in Fig. 1.3). Namely, the objective model is further improved as:

$$\min_{\phi_i \in C} \sum_{0 \le i < K} F(\mathbf{X}_i, \phi_i), \tag{1.2}$$

where, ϕ_i is the model/feature extractor for each kind of input $\mathbf{X}_i$.

Obviously, such an approach is onerous. In dense matching, where each pixel is an input as well as a target, it is impossible to manually classify them. There are another two alternative solutions.

First, we can train a second classifier to categorize the data before selecting the proper model as done in many meta-learning systems

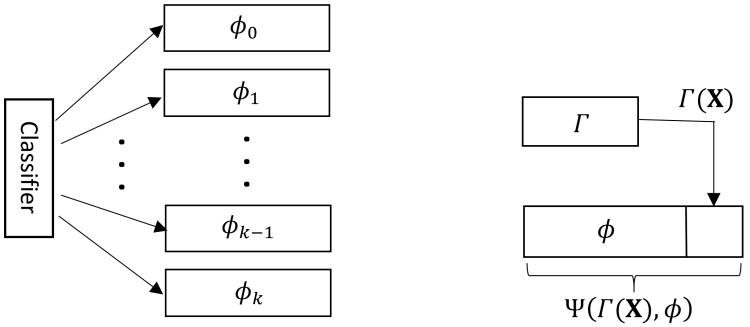


Figure 1.4: Comparisons between model Eq.(1.3) and model Eq.(1.4). *Left*: Classification results help to select the proper model. The classification results are discrete and exclusive. The error could be easily accumulated step by step. *Right*: The proposed model learned continuous meta-knowledge $\Gamma(\mathbf{X})$ and directly embedded it into the generalization part Φ to make it adaptive. Γ and Φ are learned simultaneously. There is no accumulated errors.

[42, 62]. Namely,

$$\min_{\phi_i \in C} F(\mathbf{X}, \mathbf{\Phi}^T \cdot \mathbf{C}(\mathbf{X})), \tag{1.3}$$

where, $\mathbf{\Phi}^T = [\phi_0, \phi_1, ...\phi_k]$ are the models/feature extractors for K kind of input $\mathbf{X}$. The classifier $\mathbf{C}$ is pre-trained with an one-hot decision vector (*e.g.* $\mathbf{C}(\mathbf{X}_0) = [1, 0, ...0, ...]^T$) as the output.

However, the classification results are discrete and exclusive and it is hard for the new classifier to accurately classify every input, leading to the possible selection of an improper model. Moreover, the number of classes is unknown and can be infinitely many.

Another better solution is done in this thesis, which is to realize the simultaneous automatic classification and dynamic model selection with generalization. This is realized though learning and assimilating a flow of continuous meta-information into the base model. So, the objective is finally improved as

$$\min_{\Phi \in C} F(\mathbf{X}, \Psi(\Gamma(\mathbf{X}), \Phi)), \tag{1.4}$$

where, Γ is used to learn the meta-knowledge, and Ψ combines the learned meta-information directly into the base feature extractor Φ to make it adaptive to different kinds of inputs without losing generalization. As shown in Fig.1.4, $\Psi(\Gamma(\mathbf{X}), \Phi)$ is learned and optimized simultaneously and a flow of continuous meta-knowledge is directly combined into the feature extractor to make it adaptive. There is no classification or accumulation of errors. Therefore, it is far better than models 1.1-1.3.

1.4 Summary of the Contributions

The major contributions in this thesis lie in that 1) we propose two fundamental principles which helps to identify the real good feature for dense matching; 2) we formulate these two principles as a multi-objective model to learn the effective features for dense matching and study the trade-off between these two principles; 3) we propose a new meta-level learning system which learns and assimilates continuous meta-knowledge into the feature extractor to realize its adaptive behaviors for different inputs and different regions.

The features learned through the proposed method possess the following good properties. a) *High accuracy with robustness to noise and radiometric changes*. Our evaluations on benchmarks show that our feature-based method outperforms the current best. b) *Flexible for use in almost all existing dense-matching algorithms*. These are similar to RGB values (*e.g*. scaling and subpixel interpolation are possible for continuous and pyramid-matching algorithms). c) *Fast speed*. The feature extractor consists of several convolutional layers and can be implemented efficiently (*e.g*., a 16-channel

fast feature can be extracted in 20 fps). Also, the new features do not increase the time complexity of the original matching algorithms.

1.5 Organization of the Thesis

The rest of thesis is organized as follows. After reviewing previous work and their challenges in Chapter 2. In Chapter 3, we propose and analyze the two fundamental principles (Section 3.1) and these two principles are formulated as objective function F of Eq.(1.4) (Section 3.2). Chapter 4 describes the theory of metal-level learning system which is used to build the adaptive feature extractor of the general model Eq.(1.4). Then, the implementation of the whole system is presented in Chapter 5. The experimental results in Section 6 demonstrate that our approach is effective and efficient. Finally, Section 7 concludes the thesis.

□ **End of chapter.**

Chapter 2

Background Study

In this chapter, we give a literature review on the three aspects of the proposed general model of Eq.(1.4): 1) feature extraction in dense matching which helps to understand the principles and objective function F proposed in the next section. 2) weakness of the existing meta-learning and dynamic neural networks and the strength of the proposed meta-level neural networks (MLNN) which will finally work as the feature extractor of the general model and 3) methods to optimize a multi-objective model which are the necessary for us to optimize the model and study the trade-off.

2.1 Feature Extraction in Dense Matching

In this section, we discuss the related knowledge in developing objectives for feature extraction, namely what is the real good feature for dense matching.

Traditinally, a dense matching framework usually consists of three steps: extracting local feature descriptors, calculating and aggregating matching costs, and matching the descriptors by minimizing

some energy functions. Based on the available features, matching cost can be defined as the distance or difference between two feature descriptors. Common functions used include the sum of absolute or squared distances (SAD/SSD), and normalized cross-correlation (NCC) as well as their mixed or truncated versions [27].

Cost aggregation and matching algorithms have been well studied in the past. Many powerful approaches have been proposed, including fast cost-aggregation methods [19, 49, 52, 57, 75] and matching algorithms [6, 8, 25, 65]. They are effective and perform well under some simple synthetic or indoor scenes. However, they may perform poorly when applied to complicated outdoor scenes (for instance, the KITTI datasets [5]) because the simple color or gradient-based features they employ for calculating matching costs cannot address the nuances in complex scenes. Various problems, such as wrong matches in pixels or regions, flattening/breaking in edges, and/or collapse of the whole structure, may arise when involving radiometric changes, noises, overexposure and/or textureless regions.

In general, features form the foundation of dense matching approaches. In state-of-the-art dense matching schemes, features are more important than the matching-cost function used. As shown in Fig's 1.1 and 1.2, good features can lead to impressive results, whereas improper ones may lead to structural collapses. To this end, we focus in this thesis on the development of methods for finding good features.

As discussed above, one simple and widely used feature is the color-space (*e.g.*, RGB) and/or gradient values. Its biggest advantage is that it can be flexibly and efficiently used in every dense matching scheme. For instance, it can be used for scaling and sub-

pixel interpolations to produce sub-pixel accuracy in some continuous and pyramid dense matching approaches [8, 43, 72]. However, it is easily influenced by radiometric changes, noise, overexposure and textureless regions (as shown in Fig. 1.1).

As a compensation, some illuminance invariant features have been proposed, including Laplacian of Gaussian, photometric correlation by bilateral filter [3], and Adaptive Normalized Cross Correlation (ANCC) [24]. However, they are still not robust enough for challenging outdoor scenes. Moreover, they may bring outliers and smoothness effects to some object boundaries.

Some approaches have introduced radiometric and scale invariant SIFT [47] or SURF [7] to their dense matching algorithms [79]. These features have been successfully used in object detection or scene recognition. However, for dense matching, they are not reliable in every region because it is necessary to employ a complex densifying procedure to generate dense matching maps. This requirement limits their effectiveness and accuracy improvements.

Zbontar, *et al*. are the first to introduce CNNs to compute the matching-cost matrix [76]. Their method uses a trained siamese CNN architecture to learn the patch similarity score. It skips the feature-extraction step and directly uses the score as the matching cost. Such a method has largely improved the accuracy of some algorithms (like SGM [25]). However, it sacrifices many good properties of the traditional feature-based framework; for instance, it cannot be used in continuous or pyramid matching algorithms [8, 9, 73]. Also, its time complexity and memory requirement rely on the displacement space and are issues in many large displacement dense matching applications.

In short, existing dense-matching applications generally find good features by trying them one after another and by adjusting their parameters in order to achieve good performance on some benchmarks. This is the reason why one effective matching scheme may degenerate greatly when applied to a new untested environment/dataset. This has happened to many schemes (*e.g.* [8, 9, 43, 73]) that cannot produce good results on the challenging KITTI dataset, although they have done well on others.

In this thesis, we identify two fundamental principles on good features that each pixel should possess in order to be effective for dense matching. These principles help understand the requirements on good features for dense matching. Namely, we first identify what is the real good features for dense matching through the proposed consistency and distinctiveness principles and then formulate these two principles into an objective function to automatically learn the real good features for dense matching without relying on trial-and-error strategy.

2.2 Meta-Learning and Dynamic Neural Networks

Besides the development of the objectives function mentioned in previous section, another key elements in feature extraction system is the formation/definition of the feature extractor, since it defines the search space of the solutions and heavily influences the performance of the features found by the system. When using traditional learning systems (*e.g.* CNNs), feature extractor Φ is always trained by optimizing the objective across all training data in order to find a model with fixed structure and weights. Therefore, it's impossi-

ble for one kind of feature extractor to perform well in all regions with different texture and illuminance conditions. In this section, we analysis the existing methods which aim to realize the automatically classification and model selection. They are the meta-learning system and dynamic neural network models (for which the weights of the model are varying along with the inputs.)

2.2.1 Meta-Learning Systems

In applications with diverse features, it is desirable to have algorithms that dynamically adapt to the features [58, 63, 64]. Meta-learning [42, 62] is an approach that focuses on learning the characteristics of a problem or its inputs that allows the system to select a suitable model for new or unseen scenarios. By accumulating meta-knowledge [70], meta-learning systems build self-adaptive learners using algorithms that improve their bias dynamically.

Many existing approaches combine known solution models into an integrated system and use meta-knowledge to select a proper model. Chan *et al*. [10] proposed to combine the results of multiple learning algorithms, each applied to a different set of training data, in order for the system to adapt to diverse situations. Alexandros *et al*. [1] used decision trees as inducers at the meta-learning level and mapped dataset characteristics to inducer performance in order to adapt inducers to datasets. Ali *et al*. [2] used meta-learning in automatic kernel selection for support vector machines. Ferrari *et al*. [16] developed a new method on distance-based problem characterization and ranking combination for selecting clustering algorithms.

When applied to NNs, existing meta-learning approaches can be used to select among learned models in different scenarios. Such an approach is inadequate because it is hard to enumerate all possible cases, and an incorrect selection may result in even worse performance.

2.2.2 Dynamic Neural Networks

Besides meta-learning systems, dynamic learning has been studied for a long time. Recent work explored the idea of introducing more flexibility in the network structure and their weights. Jaderberg *et al*. [31] proposed the Spatial Transformer that allows the spatial manipulation of data in a NN. Kalchbrenner *et al*. [35] proposed structure-dynamic k-Max Pooling in order to handle input sentences of varying lengths.

There were recent studies on weight-dynamic NNs. Noh *et al*. [54] introduced a layer to generate dynamic outputs and to supply them as parameters of another fully connected layer. Klein *et al*. [37] designed a dynamic convolutional layer and used it for short-range weather prediction. It used one independent branches to learn the dynamic weights and then use these weights for convolutional manipulation. However, in these work, a large amounts of parameter need to be learned to construct the dynamic weights for the fully connected or convolutional layers. Bert and Xu *et al*. [32] proposed dynamic filter networks that generated different filters with adaptive weights. This scheme do not need a large amount of parameters to learn the adaptive weights. However, the capacities of these filters are not always reliable compared with the original CNNs.

To get similar modeling power as CNNs, Klein *et al*. [37] and Noh *et al*. [54] need a complex model to generate enough parameters to run the convolutional computation. As a result, the size of the model is often too large for existing back propagation solvers, leading to overfitting or getting stuck in local minima. For example, when we implement the dynamic convolutional layer [37] for feature extraction at each pixel, the network needs a hundred times of parameters in order to construct a structure comparable to the original CNNs. Such a network can easily get stuck into an all-zero local minimum, and the memory for storing hidden layers increases quadratically.

Different with all of them, in this thesis, we propose to combine continuous meta-knowledge directly into the base model to realize the dynamic properties of the filter kernels/weights without losing the generalization capabilities. More importantly, the meta-knowledge could be learned with only a small part of extra parameters (usually 5-10%), most of the parameters are still used to guarantee the generalization of the learned solutions. When applied this strategy into the deep neural network, we are able to design a new feature extractor which could realize the adaptive behavior over different inputs/regions without losing generalization capabilities.

2.3 Multi-Objective Optimization and Pareto Optimality

In this thesis the general model is formulated as multi-objective. So, in this section, we discuss the possible methods for multi-objective

optimization.

A general multi-objective optimization can be defined as follows:

$$\min_{\phi \in C} F = \begin{bmatrix} f_1(\phi) \\ ... \\ f_k(\phi) \end{bmatrix}. \tag{2.1}$$

Solving Eq. (2.1) amounts to finding a representative set of Pareto optimal solutions. *Pareto optimality* or *Pareto efficiency* is a state of allocation of resources from which it is impossible to reallocate in order to make any one objective better off without making at least another objective worse off. The *Pareto frontier* is the set of all Pareto efficient solutions that do not dominate each other. The following four classes of methods are widely recognized approaches.

A) *Classical methods based on scalarizing.* Scalarizing Eq. (2.1) entails reformulating it into a single-objective optimization problem in such a way that optimal solutions to the single-objective problem are Pareto optimal solutions to the original multi-objective problem [29]. Using different scalarization parameters, different Pareto optimal solutions can be produced. There are two popular scalarization methods.

a) Linear scalarization (weighted-sum) methods [11]:

$$\min_{\phi \in C} F = \sum_{i=0}^{k} \omega_i f_i(\phi), \tag{2.2}$$

which can be extended to more general non-linear forms.

b) ϵ-constraint methods [11] optimize one of the objectives and reformulates the remaining as constraints:

$$\begin{aligned} &\min_{\phi \in C} \ f_j(\phi) \\ &\text{s.t.} \quad f_i(\phi) \leq \epsilon_i, i = 1...k, i \neq j, \ \phi \in C. \end{aligned} \tag{2.3}$$

B) *Methods based on lexicographic ordering* assume that the multiple objectives in Eq. (2.1) can be ranked in order of importance, with f_1 being the most important to the decision maker and f_j, the least important. The following sequence of optimization problems are then solved one at a time:

$$\begin{aligned} &\min_{\phi \in C} \ f_l(\phi) \\ &\text{s.t.} \quad f_j \leq \mathbf{y}_j^*, \ j = 1...k, \ j \neq l, \ \phi \in C, \end{aligned} \tag{2.4}$$

where $\mathbf{y}_j^*$ is the optimal value of the above problem with $l = j$.

C) *Methods based on evolutionary multi-objective optimization (EMO)* simulate the natural evolution by an iterative computation process. An initial population is first created according to a predefined scheme. Then a loop (generation) consisting of evaluation, selection, recombination, and/or mutation is executed a number of times until some termination condition is met. The best individuals left in the population are output as Pareto optimal solutions. Examples of popular EMO algorithms include NSGA-II [12] and SPEA-2 [36].

D) *Other methods.* In some special cases, no-preference methods [78] and interactive methods [51] can be used.

In this thesis, the feature extractor is defined as DNNs and trained by a back-propagation solver with mini-batch gradient descent. As gradient values must be propagated backward in every iteration, only the formulations based on weighted sum in Eq. (2.2) and ϵ-constraint in Eq. (2.3) are applicable. Using a weighted sum is easy because its gradient can be directly calculated. However, for the ϵ-constraint method, it is impossible to discard solutions out of the ϵ-constraint in the mini-batch-based training scheme. The only possibility is to

give a heavy penalty when $f_i(\phi) > \epsilon_i$ and zero penalty otherwise in order to make most of the training samples within the ϵ-constraint. This is similar to the widely used hinge-loss method.

□ **End of chapter.**

Chapter 3

Objective for Feature Extraction

In this chapter, we present the formation of the objective function F of the general model Eq.(1.4):$\min_{\Phi \in C} F(\mathbf{X}, \Psi(\Gamma(\mathbf{X}), \Phi))$. We first present details of the consistency and distinctiveness principles in Section 3.1 which help to identify what is the real good feature for dense matching. Then, in the Section 3.2, the two principles are further formulated as a multi-objective optimization which is used as the objective function F. (The formulation of the feature extractor is presented in the next chapter.)

3.1 Metrics of Features Satisfying Principles

To develop superior features for dense matching, we first present the metrics on features that satisfy the consistency and distinctiveness principles discussed in Chapter 1.

Φ – feature extractor;
$\Phi(p)$ – feature descriptor at pixel p;
l_0 – ground truth displacement;
l_i – candidate displacement in label space S ($l_i \in S$);
(*e.g.* for stereo, l_i is the candidate disparity value).

For stereo matching, stereo images are well rectified, with displacements in the x direction. In contrast, displacements for optical flow occur in both the x and y directions. In evaluating features for dense matching, there must be a related image pair (such as a stereo pair) and a correspondence map that describes the displacement between the corresponding pixels in the two images. Such a correspondence map provides the ground-truth displacement l_0 at each pixel.

In all stereo-matching and optical-flow datasets, ground truths are usually available and obtained by radar (for natural images). New features can be learned and validated on the training data based on the ground truths and then generalized to test data with the ground truths hidden.

For any pixel p in one view and q_i in another ($q_i = p + l_i$ as shown in Fig. 3.1), let $d(p, q_i)$ be the distance between the feature descriptors of p and q_i. For convenience, we set:

$$d(p, q_i) = d(\Phi(p), \Phi(q_i)) = d(p, p + l_i) = d(p, l_i). \tag{3.1}$$

Although there are many popular functions for $d(p, q_i)$, such as $L-1$ distance $\|\Phi(p) - \Phi(q_i)\|_1$ and $L - 2$ distance $\|\Phi(p) - \Phi(q_i)\|_2$, we introduce in Eq. (5.2) of Section 5.2 our own distance functions that can be specialized for dense matching applications and tailored to different feature extractors.

We then define the metrics for the two principles below.

1) *Consistency Principle.* The principle requires a feature point to own the same or similar feature descriptors when it appears in different views. For $d(p, q)$ defined in Eq. (3.1), the principle can be stated in terms of the consistence measure.

$$f_1(\Phi) = d(p, q) = d(p, l_0) = 0 \text{ or } \approx 0. \tag{3.2}$$

To increase the consistency of features between different views, f_1 must be as small as possible. In practice, large $d(p, q)$ in some regions will lead to poor accuracy in matching algorithms. For example, regions with noises and illuminance changes always have larger average $d(p, q)$.

2) *Distinctiveness Principle.* For pixel p in an image, let $p_i \neq p \in \Omega_p$, where Ω_p is the surrounding region (called *distinctive region*) centered at p in the same image (to be defined more formally later). The principle requires a feature to be distinct enough with respect to other pixels in its surrounding region. It can be stated formally as follows:

$$\forall\, p_i \in \Omega_p,\ p_i \neq p,\ d(p, p_i) > m, \tag{3.3}$$

where m is the threshold related to the vision system. For instance, the threshold is known as the just-noticeable difference (JND) in human vision systems [44,81]. There are plenty of studies that focus on JND thresholds, which decide whether $d(p, p_i)$ is "good." Note that the performance of a feature is highly influenced by the size of its distinctive region $|\Omega_p|$ (the larger the better). For example, Eq. (3.3) will not be satisfied for pixels in a smooth region and $|\Omega_p| \approx 0$.

Since the consistency principle defines a relation between different views, whereas the distinctiveness principle states a condition

Figure 3.1: An illustration of the definition in Eq. (3.5). Here, $q = p + l_0$ is the best matched location for p. The red pixel $q_j = p + l_j$ is the closest outlier to q. Hence, the width and height of Ω_p is $\gamma = \|q - q_j\|_\infty = \|l_0 - l_j\|_\infty$.

between a pixel and its neighborhood in the same view, we can deduce from the consistency principle $d(p_i, q_i) \approx 0 \Rightarrow d(p, q_i) \approx d(p, p_i)$ and $d(p, q) \approx 0$, where $q = p + l_0$ is the best matched location for p. Based on Eq. (3.2), we have $d(p, q_i) = d(p, l_i)$ and $d(p, q) = d(p, l_0)$. The distinctiveness principle between different views can be formulated as follows.

$$d(p, l_i) - d(p, l_0) > m' \;\; \forall q_i = p + l_i \in \Omega_q, \; l_i \neq l_0, \tag{3.4}$$

where m' is the new JND threshold that varies in different matching algorithms.

As shown in Fig. 3.1, we define the *distinctive region* $\Omega_p = \Omega_q$ with width/height of $2\gamma^*$ and ground truth displacement l_0 as:

$$\Omega_p = \{p + l_i \mid l_i \in S, \; \|l_i - l_0\|_\infty < \gamma^*\} \text{ with } \gamma^* = \max \gamma$$
$$\text{s.t. } d(p, l_i) - d(p, l_0) > m', \forall \, \|l_i - l_0\|_\infty < \gamma, l_i \neq l_0 \in S.$$

We now define f_2 as a metric (normalized to the [0,1] range) for the distinctiveness principle.

$$f_2(\Phi) = |\Omega_p|/|S|, \tag{3.5}$$

Table 3.1: Error rates under different precisions based on the results of MC-CNN [76] from the KITTI 2012 stereo benchmarks. The results illustrate that more errors appear around ground truths.

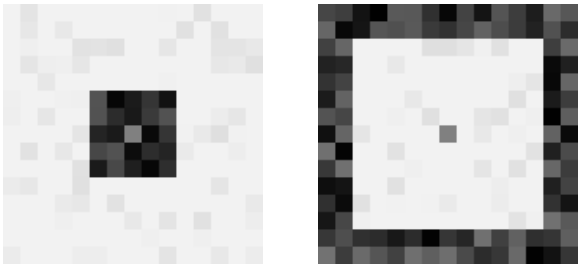
Error Threshold	Error of All Regions	Increased Error Rate
2 pixels	5.45 %	1.82%
3 pixels	3.63 %	0.78%
4 pixels	2.85 %	0.46%
5 pixels	2.39 %	–

where $|S|$ is the number of candidates in the search space S. The value is better if it is larger. For example, when it is 1, the feature at p is distinctive in the entire displacement space.

Before understanding the definition of f_2, two important properties of any dense matching algorithms must be emphasized. Firstly, outliers (for pixel p, l_j is an outlier displacement for p if $d(p, l_j) - d(p, l_0) \leq m'$) are more likely to appear around a ground truth because pixels close to each other in an image usually share similar texture, illuminance and color conditions that make them hard to differentiate. Secondly, an outlier close to a ground truth has significant influence on the accuracy of matching, not because of its high similarity to the ground truth but also their spatial proximity. These facts make it hard to find correct matches among outliers close to a ground truth.

Table 3.1 illustrates the above observations on some dense matching results. It shows that 1.82% pixels are matched at the wrong locations 2-3 pixels away from the ground truths; 0.78% are wrongly matched 3-4 pixels away; but only 0.46% are wrongly matched 4-5 pixels away.

The definition of f_2 is designed based on the properties above. The value of f_2 is decided by the outlier closest to the ground truth



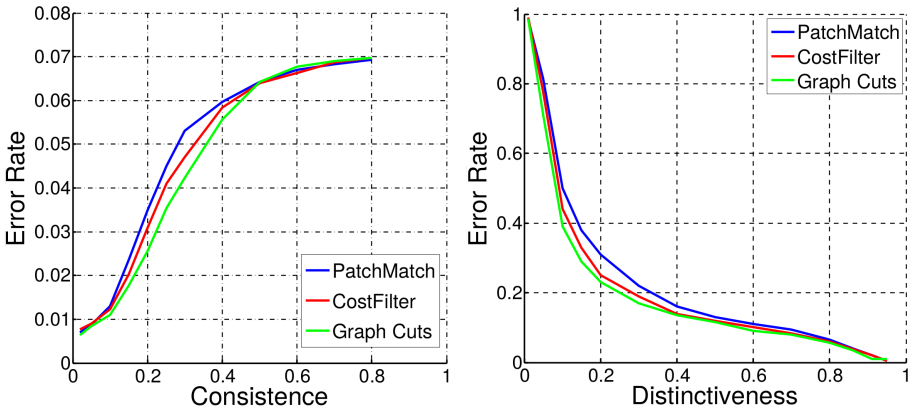
(a) (b)

Figure 3.2: An illustration of the distinctiveness principle with two matching-cost maps. Each grid corresponds to one candidate that matches q_i for p, with its color representing the value of $d(p, q_i)$ (the brighter the larger). Our target is to find the center grid corresponding to ground truth q_0, where any grid darker than the center is an outlier. (a) $f_2 = 0$, with 24 outliers close to the ground truth. (b) $f_2 \approx 0.5$, with more than 100 outliers that are all far from the ground truth.

and defines the size of the distinctive region in which there is no outlier.

As illustrated in Fig. 3.2(a) with $f_2 = 0$, although there are only 24 outliers, it is hard for any matching algorithm to find the center best matches. However, in Fig. 3.2(b) with $f_2 \approx 0.5$, there are more than 100 outliers, but it is much easier to find the center best matches. When the candidate displacement drops into the distinctive region, the matching algorithm will be guided to converge to the center pixel during cost aggregation.

Both principles are indispensable for characterizing good features. As an illustration, we collect statistics on the effect of one when that of the other is controlled. For example, we measure the effect on $d(p, q)$ by keeping the left view fixed and by adding noise and radiometric changes to the other. We then collect the data from those regions where distinctiveness is well satisfied. Fig. 3.3(a) shows how changing $f_1(\Phi)$ can influence the performance of a feature (in



(a) Effect of consistency (b) Effect of distinctiveness

Figure 3.3: Effects of each principle on accuracy of the matching results. (a) The effect of the consistency principle (f_1) on accuracy of the stereo matching results. Only regions where the distinctiveness principle is well satisfied is counted ($f_2 \geq 0.95$). (b) The influence of the distinctiveness principle (f_2) on result accuracy. Only regions where the consistency principle is well satisfied is counted ($f_1 \leq 0.02$). We use the evaluation method suggested in [26, 27] to collect the accuracy data. Different lines represent different matching algorithms, including a local method Costfilter [57], a global method graph cuts and a continuous matching algorithm Patchmatch [9].

terms of result accuracy) when $f_2(\Phi)$ is fixed. The accuracy statistics is collected using the evaluation method suggested in [26, 27]. Likewise, we choose those regions where the consistency principle is satisfied and then collect the accuracy of these regions to see its relationship to f_2. Fig. 3.3(b) shows the trend on result accuracy due to changing $f_2(\Phi)$ when $f_1(\Phi)$ is limited to a small value.

3.2 Optimization Model for Finding Good Features

Based on f_1 in Eq. (3.2) and f_2 in Eq. (3.5), we can formulate the feature extraction problem as a multi-objective optimization over C, a set of possible calculations/transformations (such as Laplacian of

Gaussian (LoG) and Sobel operator) over pixels:

$$\min_{\Phi \in C} F = \begin{bmatrix} f_1(\Phi) \\ -f_2(\Phi) \end{bmatrix}. \tag{3.6}$$

Eq. (3.6) defines the objective on good features for dense matching. Its solution $\{\Phi_1, ...\Phi_n\}$ (feature extractors) is a set of Pareto optima that are decision vectors whose objective cannot be improved in any dimension without degrading the other. Specifically, solution Φ_1 dominates Φ_2 ($\Phi_1 \succ \Phi_2$) when

$$\begin{aligned} & f_1(\Phi_1) < f_1(\Phi_2) \;\&\; f_2(\Phi_1) \geq f_2(\Phi_2) \\ \text{or} \quad & f_1(\Phi_1) \leq f_1(\Phi_2) \;\&\; f_2(\Phi_1) > f_2(\Phi_2). \end{aligned} \tag{3.7}$$

Solutions that are not dominated by others are called non-dominated, and the set of all such solutions form a Pareto-optimal set or frontier. Our target is to systematically find one or more Pareto-optimal solutions for Eq. (3.6).

In Eq. (3.6), f_1 can be easily computed when given the feature descriptors Φ. However, f_2 is expensive to evaluate because, as defined in Eq. (3.5), we need to traverse all candidate displacement values in the whole displacement space S in order to calculate f_2 for each pixel. Moreover, f_2 is non-differentiable with respect to Φ.

To reduce the complexity of calculating f_2, we develop f_2' to replace f_2 by using the following trend information on f_2. According to Eq. (3.5), we have the following observations.

1) A higher $f_2(\Phi)$ comes with a lower outlier rate. As shown in Fig. 3.1, Ω_p and f_2 for p are decided by the closest outlier l_j to ground truth displacement l_0. Given any outlier distribution, f_2 is, therefore, directly proportional to the outlier rate.

2) A high $f_2(\Phi)$ also means that all the outliers are far from ground truth l_0; that is, the value of γ^* in Eq. (3.5) should be as large as possible.

Hence, to increase f_2, we can decrease the outlier rate as well as control the distribution of outliers in order to make them all far from the ground truth. Fortunately, these targets can be easily achieved using $f_2'(\Phi)$ defined for each pixel p:

$$
\begin{aligned}
f_2'(\Phi) \;&=\; \frac{1}{|U|}\frac{\sum_{l_j\in U} w_j h(l_j)}{\sum_{l_j\in U} w_j} \qquad (3.8)\\
\text{where}\quad & h(l_j) = \delta\cdot h_1(l_j) + (1-\delta)\cdot h_2(l_j)\\
& h_1(l_j) = -\tau\cdot\log(d(p,l_j) - d(p,l_0) + \tau)\\
& h_2(l_j) = -\frac{\tau\,(d(p,l_j) - d(p,l_0) + \tau)}{\epsilon}\\
& \qquad\qquad -\tau\cdot\log(\epsilon) + \tau\\
& \delta = \begin{cases} 1 & \text{if } d(p,l_j) - d(p,l_0) + \tau > \epsilon \\ 0 & \text{otherwise} \end{cases}\\
& w_j = \exp\left(-\frac{\|l_j - l_0\|_\infty}{r}\right).
\end{aligned}
$$

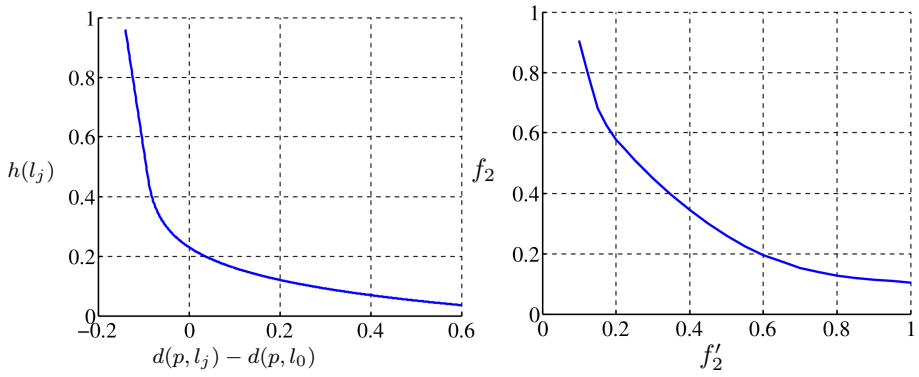
Here, $U \subseteq S$ is a small subset from the whole displacement space S chosen to reduce the high computational cost of f_2. $|U|$ represents the number of samples in U. We use a small sample displacement set $U \subseteq S$ to improve training efficiency. In this thesis, $U = U_1 \cup U_2$ contains two parts: U_1 is a sample set of integer-precision displacements and U_2 a set of floating/sub-pixel displacements. $d(p, l_j)$ and $d(p, l_0)$ can be easily achieved if $l_j \in U_1$. If $l_j' \in U_2$, we use the bilinear interpolation to obtain $d(p, l_j')$ and $d(p, l_0')$. (l_0' is also floating-precision if $l_j' \in U_2$.)

w_j is a popular spatial Gaussian weight for controlling the distri-

bution of outliers (to give different penalties to make them far from l_0); τ is used to adjust the penalty to sample l_j; and h is a piecewise function of h_1 and h_2 chosen as follows. h_1 is a slightly altered $\log$ function of the hinge/margin $\Delta d = d(p, l_j) - d(p, l_0)$: it gives large penalties if Δd is small or negative, and small penalties otherwise. h_2 is a supplementary function for h_1: it has the same value and derivative/gradient at the intersection point with h_1, and helps set an upper bound to the derivative/gradient of h_1 to avoid it becoming infinity. ϵ is a very small value chosen to avoid h_2 being infinity or an imaginary number. As $d(p, l) = d(\Phi(p), \Phi(p + l))$ is differentiable with respect to Φ, h and f_2' are continuous and differentiable, although they are both piecewise functions. In short, the components of f_2' are chosen to give large penalties to closer outliers, while having small penalties to non-outliers.

Fig. 3.4(a) shows that $h(l_j)$ is effective for reducing the outlier rate. When l_j is an outlier, it means that $d(p, l_j) - d(p, l_0)$ is small or even negative, and $h(l_j)$ gives it a higher penalty. On the other hand, as different matching algorithms have different JND (m'), it is hard to set a fixed value for m'. Instead of setting a fixed value like hinge loss, the definition of $h(l_j)$ can increase generalizability of the features learned and remove the influence of m' in our method.

Fig. 3.4(b) shows that f_2 is monotonically decreasing with increasing f_2'. We calculate f_2' by sampling p and l_j from the KITTI datasets with the condition of $f_1{<}0.02$. f_2 is then calculated by brute-force sampling. Both the sampled values of f_2 and f_2' are then averaged over a set of pixels. In this and the following experiments, we used $r{=}10$, $\tau{=}0.1$, $\epsilon{=}0.1\tau$. These parameters are used to adjust the penalty and gradient of the function, as they help limit the maxi-



(a) Trend of $h(l_j)$ (b) Trend of f_2' wrt f_2

Figure 3.4: An illustration of the monotonic behavior of f_2' with respect to f_2. (a) Monotonic trend of $h(l_j)$ with respect to $d(p, l_j) - d(p, l_0)$. (b) Corresponding monotonic relation between f_2 and f_2'. Data for f_2 and f_2' is collected by sampling from the KITTI datasets. To calculate f_2, we use $m' = 0$. Note that different m' does not change the monotonic behavior.

mum of the derivative/gradient's absolute value $\|\frac{\partial(f_2')}{\partial(\Delta d)}\|=\frac{\tau}{\epsilon}=10$ and set the baseline $\|\frac{\partial(f_2')}{\partial(\Delta d)}\|=1$ if $\Delta d=0$. These choices are not unique, and other reasonable values can also be used.

Finally the original objective in Eq. (3.6) is transformed to include f_1 and f_2' that are differentiable with respect to Φ:

$$\min_{\Phi \in C} F = \begin{bmatrix} f_1(\Phi) \\ f_2'(\Phi) \end{bmatrix}. \tag{3.9}$$

F of above model is used as the objective function in the general models Eq.(1.4):$\min_{\Phi \in C} F(\mathbf{X}, \Psi(\Gamma(\mathbf{X}), \Phi))$. To get a complete definition of the proposed model Eq.(1.4), the rest work is to give the formation of the feature extractor which will be discussed in the next chapter.

□ **End of chapter.**

Chapter 4

Supplementary Meta-Learning

In the general model Eq.(1.4): $\min_{\Phi \in C} F(\mathbf{X}, \Psi(\Gamma(\mathbf{X}), \Phi))$. Feature extractor is another key element which significantly influence the performance of the whole learning system. In this chapter, we will discuss the proposed supplementary meta-learning method which is used to make the feature extractor adaptive to different inputs. Namely, we will talk about the formulation of $\Psi(\Gamma(\mathbf{X}), \Phi)$.

To make the feature extractor adaptive, we need to realize simultaneous automatic classification and dynamic model selection with generalization, we propose to use meta-learning to integrate the two steps together. Meta-learning systems [70] adapt to specific situations by dynamically searching for the best learning strategy in response to data diversity. It differs from base-level learning (like CNNs) in terms of adaptation: meta-level learning studies how to choose the right model dynamically, as opposed to base-level learning whose model is fixed with a priori assumptions (or inductive bias [53]).

These years, deep neural networks (DNNs) have been shown effective in feature extraction for image classification, recognition and

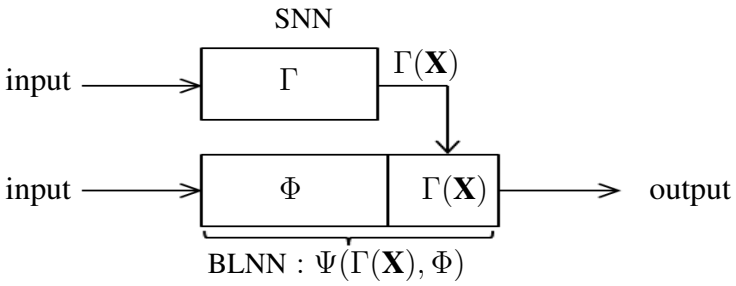


Figure 4.1: Proposed meta-learning NN (MLNN) using SNN (with function Γ and $\Gamma(\mathbf{X})$ outputs) and BLNN (Ψ combines base Φ and $\Gamma(\mathbf{X})$ together).

detection *etc*. In this thesis, the feature extractor is also defined as DNNs. In the following section, we will describe the metal-level learning strategy and its application in CNNs.

Fig. 4.1 depicts our proposed *meta-learning neural network* (MLNN) and its relationship with the general model (Eq.1.4). The top *supplementary NN* (SNN) with a small number of weights constructs the function Γ which learns and outputs meta-level information $\Gamma(\mathbf{X})$ to the *base-level NN* (BLNN) that uses it as part of its weights. The SNN is a traditional NN that abstracts meta-information which is important for the BLNN to adapt to different inputs. The BLNN maintains generalization using base traditional NN Φ, and realizes the dynamic behaviors with the meta-information $\Gamma(\mathbf{X})$ from SNN. It combine the meta-information into the base NN and achieve the adaptive feature model $\Psi(\Gamma(\mathbf{X}), \Phi)$. Such a system can be trained by backward propagation and achieves far better performance when compared to traditional CNNs, especially on complex problems with large data diversity.

4.1 Assimilation of the Continuous Meta-Knowledge

In a traditional NN like CNN, we can assume feature extractor $\Phi = \{\mathbf{W}, \mathbf{b}\}$, with $\mathbf{W}$ as the convolutional filter, and $\mathbf{b}$ as the bias vector. The output $\mathbf{Y}_i$ for input $\mathbf{X}_i$ of the layer can be represented as follows,

$$\mathbf{Y}_i = \mathbf{W} \cdot \mathbf{X}_i + \mathbf{b}, \tag{4.1}$$

The problem is that in above model, the solver optimizes an average behavior over input $\mathbf{X} = \{\mathbf{X}_0, ..., \mathbf{X}_n\}$ to result in fixed $\mathbf{W}$ and $\mathbf{b}$ independent of input. That is, all inputs share the same model despite their individual features.

Let $\Gamma(\mathbf{X}_i)$ in Fig. 4.1 be the learned meta-knowledge for $\mathbf{X}_i$, where Γ is the function to be learned. To combine $\Gamma(\mathbf{X}_i)$ into the original convolutional model, we split it into two parts as follows.

$$\Gamma(\mathbf{X}_i) = \{\Gamma_{\mathbf{w}}(\mathbf{X}_i), \Gamma_{\mathbf{b}}(\mathbf{X}_i)\}. \tag{4.2}$$

We then combine Γ into the two parts of Eq. (4.1) by the simple dot product and addition.

$$\begin{aligned} &\mathbf{Y}_i = [\Gamma_{\mathbf{w}}(\mathbf{X}_i) \cdot \mathbf{W}]\mathbf{X}_i + [\Gamma_{\mathbf{b}}(\mathbf{X}_i) + \mathbf{b}] \\ &\text{where } \mathbf{W}_i = \Gamma_{\mathbf{w}}(\mathbf{X}_i) \cdot \mathbf{W}; \quad \mathbf{b}_i = \Gamma_{\mathbf{b}}(\mathbf{X}_i) + \mathbf{b}. \end{aligned} \tag{4.3}$$

The meta-knowledge introduced in Eq. (4.3) allows different situations to be classified accordingly. The fixed part $\{\mathbf{W}, \mathbf{b}\}$ in Eq. (4.1) enables generalization of each class to unseen cases. The model, however, can adapt to different inputs, as the meta-knowledge captured by SNN and realized as $\{\Gamma_{\mathbf{w}}, \Gamma_{\mathbf{b}}\}$ will change with different inputs. Since the learning of both parts is done simultaneously in our model, the classification and generalization of each input can be realized in an integrated manner.

4.2 Matrix Setting

We discuss in this subsection the setting of $\{\Gamma_{\mathbf{w}}, \Gamma_{\mathbf{b}}\}$ in Eq. (4.3). For $\{\mathbf{W}, \mathbf{b}\}$ in Φ, we assume $\mathbf{W}$ is an $N \times M$ matrix, and $\mathbf{b}$ is an $N \times 1$ vector. To keep the matrix sizes consistent, we set $\{\Gamma_{\mathbf{w}}(\mathbf{X}_i), \Gamma_{\mathbf{b}}(\mathbf{X}_i)\}$ as an $N \times N$ matrix and an $N \times 1$ vector, respectively.

Since the output of $\Gamma_{\mathbf{b}}$ is an $N \times 1$ vector that can be easily learned, we focus on $\Gamma_{\mathbf{w}}$. Learning such a function is expensive as it needs many new parameters to fit $\Gamma_{\mathbf{w}}$ and N^2 memory to store intermediate and final outputs. To reduce the burden in learning, we need to simplify $\Gamma_{\mathbf{w}}$ by reducing the number of parameter to learn the function Γ. There are three possibilities here: a) using a matrix with repeated elements, *e.g*. circulant matrix, b) choosing a matrix with special element distribution, *e.g*. each row satisfies the Gaussian distribution that requires learning its mean and standard deviation, c) employing a sparse matrix.

We encountered some difficulties when using the first two alternatives. We observe that convergence is not stable during learning, the convergent speed is even slower than the original CNNs and it can easily get stuck in local minima. This could be caused by the dependence of matrix elements that counteract each other. For example, when using a matrix with repeated elements in a convolutional model, each element is used to adjust more than one filters. This is also the case when elements have a special distribution.

We, therefore, focus on using sparse matrices with independent elements. Learning in this case is more efficient as the number of elements is small, although the structure of $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$ has significant effects on complexity.

To better utilize the learned meta-knowledge, it is obvious that $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$ should have full rank, namely, $rank(\Gamma_{\mathbf{w}}(\mathbf{X}_i)) = N$. For $\mathbf{W}_i = (\mathbf{w}_1, ..., \mathbf{w}_n)^T$, we know that $rank(\mathbf{W}_i) \leq rank(\Gamma_{\mathbf{w}}(\mathbf{X}_i))$. If $rank(\Gamma_{\mathbf{w}}(\mathbf{X}_i)) < N$, then there exists $j \neq k$ such that $\mathbf{w}_k = \alpha \mathbf{w}_j$, which makes $\mathbf{w}_k$ homogeneous with $\mathbf{w}_j$ and contributes nothing new to the model. For example, in an extreme case where $rank(\Gamma_{\mathbf{w}}(\mathbf{X}_i)) = 1$, $(N-1)/N$ of the elements in $\mathbf{W}_i$ are redundant because all the N filters are homogeneous. This setting performs no better than directly using a $1 \times M$ matrix as $\mathbf{W}_i$. During learning, although we cannot control the values of the weights, we can eliminate redundant information in $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$ to reduce the complexity in learning.

In sparse matrices, the band matrix is a special case that always has full rank regardless of how its bandwidth is changed. We, therefore, use a band matrix to implement $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$, as it is easy to control the model complexity by adjusting the bandwidth while keeping the full-rank property.

The following is a band matrix with bandwidth 1 (with both lower and upper bandwidths of 1),

$$\begin{bmatrix} w_{00} & w_{01} & & & & & \mathbf{0} \\ w_{10} & w_{11} & w_{12} & & & & \\ & & \ddots & & & & \\ & & w_{i,i-1} & w_{i,i} & w_{i,i+1} & & \\ & & & \ddots & & & \\ \mathbf{0} & & & & & w_{n,n-1} & w_{n,n} \end{bmatrix}.$$

Such a band matrix has many good properties when realizing DNNs. Firstly, we only need to learn no more than $3N$ parameters in a 1-band matrix ($5N$ for a 2-band matrix). Secondly, learning is fast and memory efficient in both forward and backward propagations, since band matrices lend themselves to more efficient computations than dense ones. Finally and most importantly, band matrices are

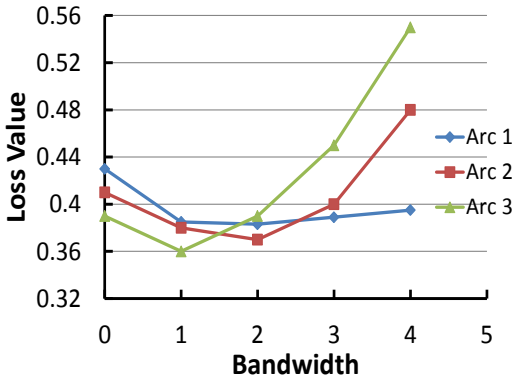


Figure 4.2: Image supper resolution (Appendix A) used to illustrate the effect of bandwidth on loss value. The graph plots the performance (loss values) of different trained network architectures (Arcs 1-3) as a function of bandwidth.

more flexible and efficient in utilizing the learned elements when compared to other sparse matrices, as they always have full-rank regardless of the bandwidth.

When implementing the output of $\Gamma_{\mathbf{w}}$ as a band matrix, its bandwidth will significantly affect the speed, memory usage and accuracy in learning. When bandwidth is increased, more elements will need to be learned and the SNN becomes more complex. Limited by the ability of back propagation, it will soon lead to either overfitting or getting stuck in local minima. Fig. 4.2 illustrates this phenomenon in which learning performance reaches an optimum at a particular bandwidth. In most cases, the optimum is at a bandwidth between 1 and 3, with little difference among them. As a result, we set the bandwidth to 1 in all our experiments.

Now, the whole model Eq.(1.4): $\min_{\Phi \in C} F(\mathbf{X}, \Psi(\Gamma(\mathbf{X}), \Phi))$ has been clearly defined. In the next chapter, we will describe the implementation of the model.

□ **End of chapter.**

Chapter 5

Implementation of the Framework

With the well defined general model: $\min_{\Phi \in C} F(\mathbf{X}, \Psi(\Gamma(\mathbf{X}), \Phi))$. In this chapter, we give a detailed implementation of the general model. We fist define the loss function for training in Section 5.1 which is achieved from the multi-objective F. Then, in Section 5.2, we present two kinds of DNN architectures as the feature extractor Φ, including a five-block fast architecture and a five-block accurate architecture. Consequently, we describe the detailed implementation of each block of the DNN architectures. We studied two kinds of implementations of the NN blocks. They are the CNN implementation and the MLNN implementation (Section 5.2.2). Finally, Section 5.3 gives the detailed settings of the learning algorithms.

5.1 Loss Functions

In the whole learning system, to solve Eq. (3.9), we employ the following loss function ℓ to learn the feature extractor. For the input

left view I and the corresponding right view I',

$$\ell \;=\; \frac{1}{|I|}\sum_{p\in I}\left((1-\lambda)f_2' + \lambda f_1^3\right), \tag{5.1}$$

where $|I|$ is the number of valid training samples, and f_1 and f_2' are defined in Eq's (3.2) and (3.8), respectively. We use λf_1^3 to control the consistency principle in order to achieve different non-dominated solutions. Different solutions can be found by adjusting λ and/or by changing the form of f_1^3.

Our loss function follows the weighted-sum method while using a cubic form of f_1. This realizes the target by using a small penalty when $f_1(\Phi)$ is relatively small and a heavy penalty when $f_1(\Phi)$ is relatively large. By giving different values to λ, we can obtain a set of non-dominated solutions.

Before deciding on the form of the loss function, we tried $d = 1, 2, 3, 4$, respectively, in f_1^d. In each case, we collected several solutions by adjusting λ. Finally, we found that the current cubic form works the best.

In solving Eq. (3.9), it is impossible to get all the non-dominated solutions on the continuous Pareto frontier, which does not have a closed form. Due to the high cost in training DNNs, we find four non-dominated solutions for each architecture. More solutions can be obtained by changing λ and by training the DNN again. These solutions are non-dominated with respect to each other; that is, no one solution is better than another, and each may perform differently under different matching algorithms. For example, in some local matching algorithms with the Winner-Take-All strategy (such as CostFilter [57]), the feature when setting $\lambda = 0$ gets the best accuracy. In other global matching algorithms (such as PMBP [8]),

spatial propagation (such as PatchMatch [9]) and pyramid matching (pyramid-MDP [72]), features by setting λ in $[0.4, 0.6]$ perform a little better.

5.2 Implementation of the Feature Extractors

Recently, DNNs trained by back-propagation [40] have been found to perform well on large-scale computer-vision tasks, with superior performance on feature extraction for classification and recognition [21, 46]. In this section, we develop a method based on DNNs to solve Eq. (3.9).

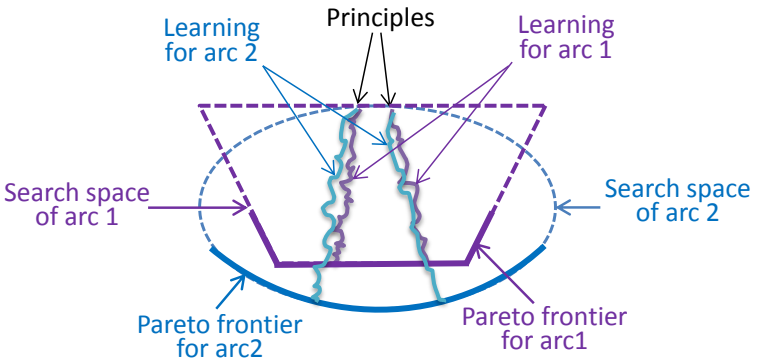


Figure 5.1: An illustration of the orthogonal relationship among the principles on consistency and distinctiveness, network architectures, as well as training/optimization procedures in our learning system.

In the search space defined by a given DNN architecture, the multi-objective formulation in Eq. (3.9) identifies the possible tradeoffs between the two principles in this search space. The network optimization/training procedure is used to find a path in this space towards those solutions that are on or close to the Pareto frontier with the best tradeoffs. That is, different DNN architectures will

lead to different Pareto frontiers as well as tradeoffs. Fig. 5.1 illustrates the orthogonal relationship among the principles, the DNN architecture, and the learning algorithm. The architectural setting defines the corresponding search space and limits where solutions can be found.

Among the principles, the architectures and the learning algorithms, we find the architectures to be the most flexible because they are always designed empirically. More advanced architectures usually lead to an expanded solution space and better solutions. However, when limited by time and computational resources, it is impossible to try all possible architectures one by one in order to find the best solution.

5.2.1 DNN Architectures Studied

Fig. 5.2 shows our proposed DNN architectures for stereo and optical flows, and Table 5.1 lists the configurations of two versions, one designed for speed and the other for accuracy.

For the version designed for speed, we employ a one-branch structure for which both views share the same model for feature extraction. The input data can be any $k \times k$ $(k > 10)$ sub-images. We find that this increases the consistency of the features obtained and helps training converge faster. With only one branch, we do not need to exchange the left and right views and compute the feature maps twice to obtain the matching results for both views. Sigmoid function is used after the last layer to restrict the range of values to (0,1). This will not affect accuracy but will accelerate convergence.

For the version designed for accuracy, we employ a symmetric

Table 5.1: Configurations of the DNN architectures studied for stereo and optical flows. Two versions, one designed for speed and the other for accuracy, are used. In these architectures, batch normalization [30] are embedded before each ReLU and Max-Pooling layer.

Layer Set	Stereo			
	One-branch Fast		Two-branches Accurate	
	Layer	shape	Layer	shape
→⓪	input	$1 \times k \times k$	input	$1 \times 11 \times 11$
			Normalize	$1 \times 11 \times 11$
⓪→①	3×3 block	$n \times (k\text{-}2) \times (k\text{-}2)$	3×3 block	$n \times 9 \times 9$
	ReLU	—	ReLU	—
①→②	3×3 block	$n \times (k\text{-}4) \times (k\text{-}4)$	3×3 block	$n \times 7 \times 7$
	ReLU	—	ReLU	—
②→③	3×3 block	$n \times (k\text{-}6) \times (k\text{-}6)$	3×3 block	$n \times 5 \times 5$
	ReLU	—	ReLU	—
③→④	3×3 block	$n \times (k\text{-}8) \times (k\text{-}8)$	3×3 block	$n \times 3 \times 3$
	ReLU	—	ReLU	—
④→⑤	3×3 block	$n \times (k\text{-}10) \times (k\text{-}10)$	3×3 block	$n \times 1 \times 1$
	Sigmoid	—		

Layer Set	Optical Flow			
	One-branch Fast		Two-branches Accurate	
	Layer	shape	Layer	shape
→⓪	input	$1 \times k \times k$	input	$1 \times 31 \times 31$
			Normalize	$1 \times 31 \times 31$
⓪→①	5×5 block	$n \times (k\text{-}4) \times (k\text{-}4)$	3×3 block	$n \times 29 \times 29$
	ReLU	—	MaxPooling	$n \times 15 \times 15$
①→②	5×5 block	$n \times (k\text{-}8) \times (k\text{-}8)$	3×3 block	$n \times 13 \times 13$
	ReLU	—	MaxPooling	$n \times 7 \times 7$
②→③	5×5 block	$n \times (k\text{-}12) \times (k\text{-}12)$	3×3 block	$n \times 5 \times 5$
	ReLU	—	MaxPooling	$n \times 3 \times 3$
③→④	5×5 block	$n \times (k\text{-}16) \times (k\text{-}16)$	3×3 block	$n \times 1 \times 1$
	ReLU	—	ReLU	—
④→⑤	5×5 block	$n \times (k\text{-}20) \times (k\text{-}20)$		
	Sigmoid	—		

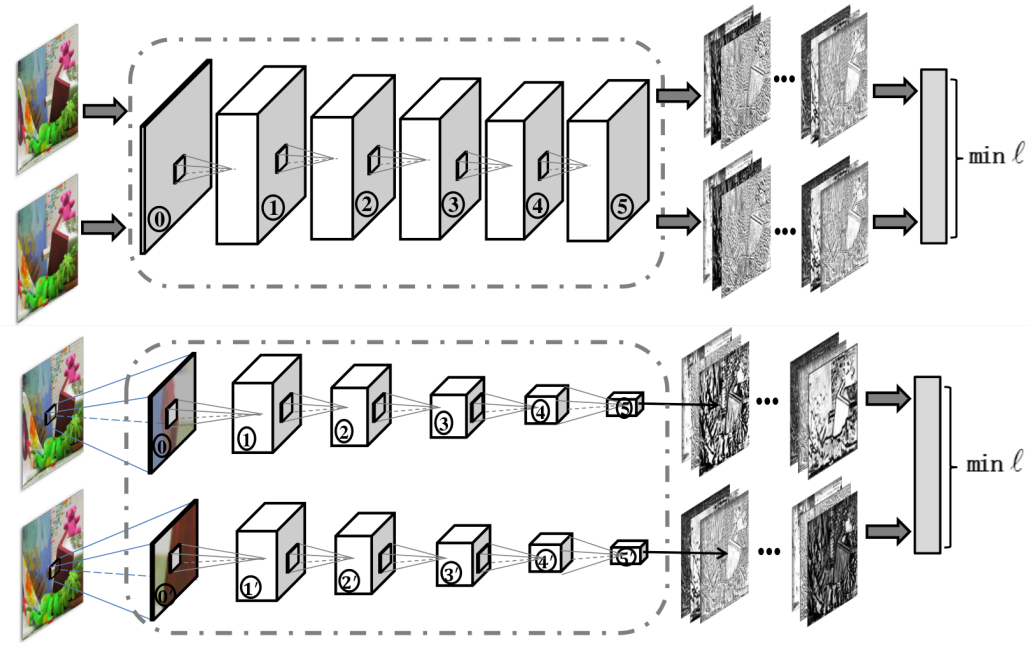


Figure 5.2: DNN architectures used as feature extractors. *Upper panel*: fast architecture, with left and right views using the same network as feature extractors. The inputs can be images of any size. *Lower panel*: two-branch accurate architecture, with both left and right views having its own network branch as feature extractor. Inputs must be 11×11 patches; $M \times N$ images must be sliced to MN patches before input.

two-branch architecture, one for the left view and another for the right. Its inputs are fixed-size patch pairs at each pixel. Before they are input to the network, we reduce the influence of radiometric changes between the views by normalizing both patches (subtracting the mean and dividing by the standard deviation). After training, to extract features for the whole images, mirror-padding is used in each convolutional layer for pixels in image borders.

Based on the fast and accurate DNN architectures, we can define the distance-measurement function (cost function) of two feature de-

scriptors $d(p, l_i) = d(\Phi(p), \Phi(p + l_i)) = d(\Phi(p), \Phi(q_i))$ as follows:

$$d(p, l_i) = \begin{cases} \frac{1}{2} \left\| \frac{\Phi(p)}{\|\Phi(p)\|_2} - \frac{\Phi(p+l_i)}{\|\Phi(p+l_i)\|_2} \right\|_2^2 & \text{fast} \\ \text{sigmoid}\left(\alpha \begin{bmatrix} \Phi(p) \\ \Phi(p + l_i) \end{bmatrix} + \beta \right) & \text{accurate.} \end{cases} \tag{5.2}$$

For the fast (*resp*., accurate) architecture, we use the first (*resp.*, second) form as the cost function. Here, α and β are both learned parameters that can be implemented using one fully connected layer. The time complexities of both functions in Eq. (5.2) are $O(n)$, where n is the number of channels in the feature descriptor. More complex forms are not employed, as they highly influence a matching algorithm's complexity.

These two functions both restrict the range of distance values to [0,1]. There are two main reasons for using this range. Firstly, restricting the distribution of distance values will help adapt them easier in different matching algorithms, as it is easy to adjust parameter settings in different matching algorithms (such as setting the balance between matching-cost and smoothness terms). Secondly and more importantly, this helps reduce the influence of some extreme outliers. Without restricting the matching cost, $d(p, l_0)$ of the best matches in some difficult cases can be very large and will significantly influence the neighborhoods in cost aggregation of matching algorithms, leading a whole region to wrong matches.

5.2.2 Implementation of MLNN Block

For each of the five NN blocks in both fast and accuracte architectures, we studied two kinds of implementation, the base CNN implementation and the implementation of the proposed MLNN. In above

DNN architectures (both fast and accurate), the five NN blocks can be easily implemented as CNN layers. For example, in this thesis, we studied the architectures where each of the block is implemented as one $[3 \times 3, n]$ convolutional layers for 3×3 blocks or $[5 \times 5, n]$ convolutional layers for 5×5 blocks. For this CNN implementaion, we studied the tradeoffs between the proposed principles and verify its effectiveness of the principles in the experiments.

Besides the base CNN implementation, in this section, we will focus on the implementation of the advanced MLNN block. Different with the CNN block, we replace the convolutional layer with Eq.(4.3): $\mathbf{Y}_i = [\Gamma_{\mathbf{w}}(\mathbf{X}_i) \cdot \mathbf{W}]\mathbf{X}_i + [\Gamma_{\mathbf{b}}(\mathbf{X}_i) + \mathbf{b}]$, which combine the meta-knowledge into the base NN and realize the adaptive behavior of the filter kernels/weigths and the bias term. Namely this section talks about the implementation of $\Psi(\Gamma(\mathbf{X}), \Phi)$.

The original Eq.(4.3) is not flexible when used in DNNs. Here, we simplify the model to facilitate the implementation in existing deep-learning platforms by employing simple transformations to $\Gamma_{\mathbf{w}}$ and $\Gamma_{\mathbf{b}}$. We first assume $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$ and $\Gamma_{\mathbf{b}}(\mathbf{X}_i)$ to be near linear transformation of $\mathbf{X}_i$, which can be implemented by several convolutional layers. Since $\Gamma_{\mathbf{b}}$ has no fixed form, we give it a new form:

$$\Gamma_{\mathbf{b}}(\mathbf{X}_i) = [(\Gamma_{\mathbf{w}}(\mathbf{X}_i) - \mathbf{I}) \cdot \mathbf{b}]^T + \Gamma'_{\mathbf{b}}(\mathbf{X}_i). \tag{5.3}$$

Note that $\Gamma'_{\mathbf{b}}(\mathbf{X}_i)$ and $[(\Gamma_{\mathbf{w}}(\mathbf{X}_i) - \mathbf{I}) \cdot \mathbf{b}]^T$ can be treated as a base shift vector. In that case, $\Gamma'_{\mathbf{b}}(\mathbf{X}_i)$ is still dynamic to inputs but now becomes the target of learning. As the original $\Gamma_{\mathbf{b}}$ and $\Gamma_{\mathbf{w}}$ are learned as a set of convolutional layers, $\Gamma'_{\mathbf{b}}$ can also be learned using these layers.

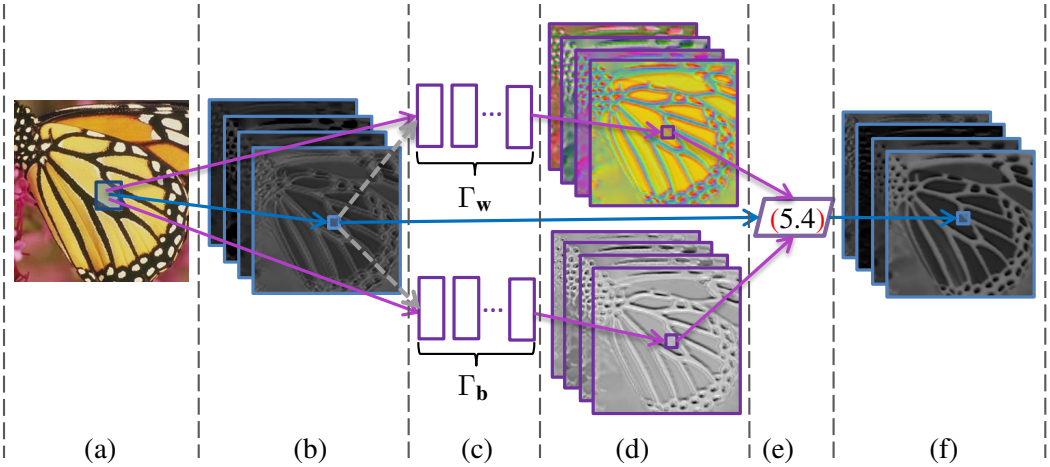


Figure 5.3: Implementation of MLNN, where pixel-level meta knowledge $\{\Gamma_{\mathbf{w}}(\mathbf{X}_i), \Gamma_{\mathbf{b}}(\mathbf{X}_i)\}$ is extracted in SNN by two sub-branches. (a) Input sample; (b) Output of ω_1; (c) Γ: two-branch SNN; (d) Learned meta-knowledge $\Gamma(\mathbf{X}_i)$: $\{\Gamma_{\mathbf{w}}(\mathbf{X}_i), \Gamma_{\mathbf{b}}(\mathbf{X}_i)\}$; (e) Combining the meta-knowledge $\Gamma(\mathbf{X}_i)$ by Eq. (5.4); (f) Output of MLNN. For implementation of Eq. (5.5), the input to SNN in Stage (c) is changed to the gray dashed arrows.

By substituting Eq. (5.3) into Eq. (4.3), we have

$$\mathbf{Y}_i = \Gamma_{\mathbf{w}}(\mathbf{X}_i) \cdot (\mathbf{W}\mathbf{X}_i + \mathbf{b}) + \Gamma'_{\mathbf{b}}(\mathbf{X}_i). \tag{5.4}$$

We further introduce a linear transformation of $\mathbf{X}_i$ to $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$ and $\Gamma'_{\mathbf{b}}(\mathbf{X}_i)$,

$$\begin{aligned} \mathbf{Y}_i = \Gamma'_{\mathbf{w}}(\mathbf{W}\mathbf{X}_i + \mathbf{b}) \cdot (\mathbf{W}\mathbf{X}_i + \mathbf{b}) + \Gamma''_{\mathbf{b}}(\mathbf{W}\mathbf{X}_i + \mathbf{b}) \\ \text{where } \Gamma_{\mathbf{w}}(\mathbf{X}_i) = \Gamma'_{\mathbf{w}}(\mathbf{W} \cdot \mathbf{X}_i + \mathbf{b}) \\ \text{and } \Gamma'_{\mathbf{b}}(\mathbf{X}_i) = \Gamma''_{\mathbf{b}}(\mathbf{W} \cdot \mathbf{X}_i + \mathbf{b}). \end{aligned} \tag{5.5}$$

As indicated above, the fixed part $\mathbf{W}\mathbf{X}_i + \mathbf{b}$ in Eq's (5.4)-(5.5) can be implemented by convolutional layers. In this thesis, we assume that $\Gamma'_{\mathbf{w}}$, $\Gamma'_{\mathbf{b}}$ and $\Gamma''_{\mathbf{b}}$ can all be fitted by some convolutional layers. We further restrict the values of $\Gamma'_{\mathbf{w}}(\mathbf{X}_i)$ in a range $(-1,1)$ using an activation function like $Tanh$ to avoid the influence of the initialization of the parameters in SNN. This also helps limit the search

space and improves convergence, while avoiding getting stuck in local minima. Finally, the outputs of $\Gamma'_{\mathbf{w}}$ are reshaped to the required band matrices.

Fig. 5.3 shows that $\mathbf{W}\mathbf{X}_i + \mathbf{b}$ can be achieved in BLNN as Φ. We further implement SNN using two sub-branches for $\Gamma'_{\mathbf{w}}$ and $\Gamma'_{\mathbf{b}}/\Gamma''_{\mathbf{b}}$, respectively. The difference between their implementations is that Eq. (5.4) needs to use the output of the previous layers in Stage (a) as inputs to the sub-branches, whereas Eq. (5.5) directly uses the output of the major branch in Stage (b). When the BLNN layer is a $k \times k$ convolutional layer, to guarantee shape and range consistency, the two branches of SNN also need similar $k \times k$ convolutional scope. For Eq. (5.5), we can always use 1×1 filters to improve efficiency. For these reasons, we use Eq. (5.5) to realize the dynamic model in our experiments. Eq. (5.4) will only be used in fully-connected layers or when the inputs of SNN is different from those of BLNN.

5.3 Details of the Learning Algorithm

This section present the details of the learning algorithms including the processing of the training data and the parameter settings during the learning course.

5.3.1 Preprocessing of training data

Training data are constructed from the KITTI stereo and optical-flow datasets [18]. For the fast architecture, each whole image is normalized by subtracting its mean and dividing by its standard deviation. As mentioned before, the fast architecture can use any size of sub-

Table 5.2: Approaches and parameter ranges for training-data augmentation on the KITTI datasets

Types	Parameters	Ranges	
		stereo	optical flow
Radiometric	$contrast$	$[0.8, 1.2]$	$[0.8, 1.2]$
	$constrast\_dif$	$[-0.15, 0.15]$	$[-0.25, 0.25]$
	$brightness$	$[-0.3, 0.3]$	$[-0.3, 0.3]$
	$brightness\_dif$	$[-0.2, 0.2]$	$[-0.4, 0.4]$
Scale&Rotation	$scale$	$[0.9, 1.1]$	$[0.75, 1.25]$
	$scale\_dif$	$[-0.1, 0.1]$	$[-0.25, 0.25]$
	$angle$	$[-20°, 20°]$	$[-30°, 30°]$
	$angle\_dif$	$[-10°, 10°]$	$[-20°, 20°]$
Noise	$mean$	$[-0.05, 0.05]$	$[-0.05, 0.05]$
	$stdev$	$[0, 0.2]$	$[0, 0.2]$

images as inputs. To ensure the same size of training-image pairs, all training images are sliced into overlapping 71×71 sub-image pairs. The center of each pair is required to be highly matched (*e.g.* the KITTI 2012 dataset contains 194 image pairs that can be sliced to about 200,000 sub-image pairs). In contract, for the accurate architecture, training data are sliced into 11×11 (stereo) or 31×31 (flow) patch pairs as in [76]. During the training process, all invalid regions and pixels, such as occlusions and pixels without ground-truth displacements, are neglected.

5.3.2 Data-set augmentation

Augmenting the training data set is commonly used to reduce generalization errors. To make the learned feature robust to imperfections such as noise, illuminance changes and view rotations, for sub-images I^l and I^r of the, respectively, left and right views, we control their radiometric variance by $I^l_n = I^l \times \text{constrast} + \text{brightness}$ and $I^r_n = I^r \times (\text{constrast} + \text{constrast\_dif}) + (\text{brightness} + \text{brightness\_dif})$.

Further, to make the feature extractor robust to noise, Gaussian noises (controlled by their mean and standard deviation) are randomly added to each sub-image pairs during training. Likewise, each pixel in the training data is scaled and rotated by the scale and rotation parameters. Table 5.2 shows the ranges in which the random values are taken for these radiometric, noise, scaling and rotation parameters.

5.3.3 Learning platform and parameters

We modified the CNN platform Caffe [33] to implement our own learning procedures. For training-data sampling, we set $U = \{l_i | l_i \in S, |l_i - l_0|_\infty > 3\}$ and the sampling numbers $|U_1| = 2$ and $|U_2| = 1$. Mini-batch gradient descent with the momentum term set to 0.9 is used to minimize the loss. The batch size is set at 256 (*resp.*, 64) for the accurate (*resp.*, fast) architecture. The models are trained for a total of 800K iterations with the learning rate initially set to 0.01, and then decreased by a factor of 10 at the 300K$^{\text{th}}$ and the 600K$^{\text{th}}$ iterations.

□ **End of chapter.**

Chapter 6

Experimental Results

In this chapter, we present the experimental design and results. The experiments include two parts: 1) experiments with CNN implementation which mainly study the trade-off between the proposed consistency and distinctiveness principles and evaluate the feature's performance with dense matching algorithms on the KITTI datasets and benchmarks; 2) experiments which compare the features found by MLNN implementation and other state-of-the-art base NNs (*e.g.* residual and inception network architecture).

6.1 Parameter settings

In experiments of Section 6.2, the 3×3 or 5×5 blocks are directly implemented as $[3 \times 3, n]$ or $[5 \times 5, n]$ convolutional layers. More advanced architectures are evaluated in the Section 6.3. There are two key parameters in our learning system. The first is n, the number of channels in the DNN architecture. It denotes the length of the feature descriptor that highly influences the accuracy and efficiency of the features found. We set n to different integers (64, 128, 256) and

train the DNN to get the feature descriptors. Section 6.2.4 analyzes the effect of n on complexity. The second key parameter is λ in Eq. (5.1) that balances the weights between consistency and distinctiveness. Different λ will lead to different non-dominated solutions. In our experiments, we set λ is to $\{0.8, 0.6, 0.4, 0\}$ to get four sets of non-dominated features.

All the other parameters, including learning rate, training iterations, batch-size and data augmentation parameters, are set empirically. Reasonable settings are decided using trial-and-error, background experience, and suggestions from related works. Until now, there is no easy way to automatically set the best values for these parameters in learning DNNs.

6.2 Evaluations of Features Found by CNN implementation

In this section, we first compare the performance (error rates of the results) of our feature found by CNN implementation with other existing features in different kinds of matching algorithms (Section 6.2.1). Then, we study the tradeoffs between the consistency and distinctiveness principles, analyze and evaluate features through these two principles (Section 6.2.2). Finally, we evaluate the features' performance (error rates) on the KITTI benchmarks (Section 6.2.3) and analyze the complexity and efficiency of the system and the feature extractor (Section 6.2.4).

6.2.1 Comparisons and Evaluations in Different Matching Algorithms

The learned features (in 256 channels) are evaluated in various existing dense matching (stereo matching or two-frame optical-flow) algorithms. The features found by our method are flexible and can be used for subpixel-location interpolation and size/shape scaling for each channel, allowing them to be used in almost all existing dense matching algorithms. One or two algorithms are selected for each of the 5 existing classical frameworks and their performance is compared using different features. The matching algorithms studied include cost-filtering methods [57], global-matching algorithms [65], semi-global matching algorithms [25], continuous (subpixel accuracy/slanted surface) matching algorithms [8, 9], and pyramid-matching algorithms [72]. The features (or the matching-cost method) for comparisons include Census (used in [25]), color, gradient, color+gradient [9], Dense SIFT [47]/SURF [7] and MC-CNN matching cost [76].

In our evaluations, test data are chosen from the KITTI 2012 (stereo and flow) dataset: 194 training-image pairs are randomly divided into 4 sets (each with 48 or 49 two-view pairs), with 3 sets chosen for training and the rest for testing.

Table 6.1 presents the performance results, which are evaluated by the 3-pixel-threshold error rate of the non-occluded regions and

Table 6.1: Performance evaluation (in error rate %) and comparisons of different features in various matching algorithms

Features	CostFilter [57]		PatchMatch [9]		PMBP [8]		SGM [25]	MDP [72]
	stereo	flow	stereo	flow	stereo	flow	stereo	flow
color	30.19	39.94	19.92	23.9	17.99	21.79	11.75	15.9
gradient	23.1	28.31	14.81	15.51	14.1	14.61	6.91	13.41
color+grad	21.51	27.5	14.25	13.35	12.95	10.5	7.01	11.15
census	11.54	19.1	×	×	×	×	5.17	×
SIFT*	9.13	17.51	×	×	×	×	4.92	×
SURF*	9.91	18.19	×	×	×	×	×	×
mc-cnn-fast	6.49	×	×	×	×	×	3.41	×
mc-cnn-acc	6.04	×	×	×	×	×	3.09	×
Our fast	6.01	10.47	4.74	5.74	4.64	5.46	3.24	5.84
Our acc	**5.67**	**9.34**	**4.37**	**5.17**	**3.83**	**4.81**	**3.01**	**5.17**

$*$ Keypoint-detection step is discarded for SIFT [47] and SURF [7] features (refer to the dense SIFT implementation in [69]). Descriptors are generated at each pixel in order to run above dense matching algorithms.

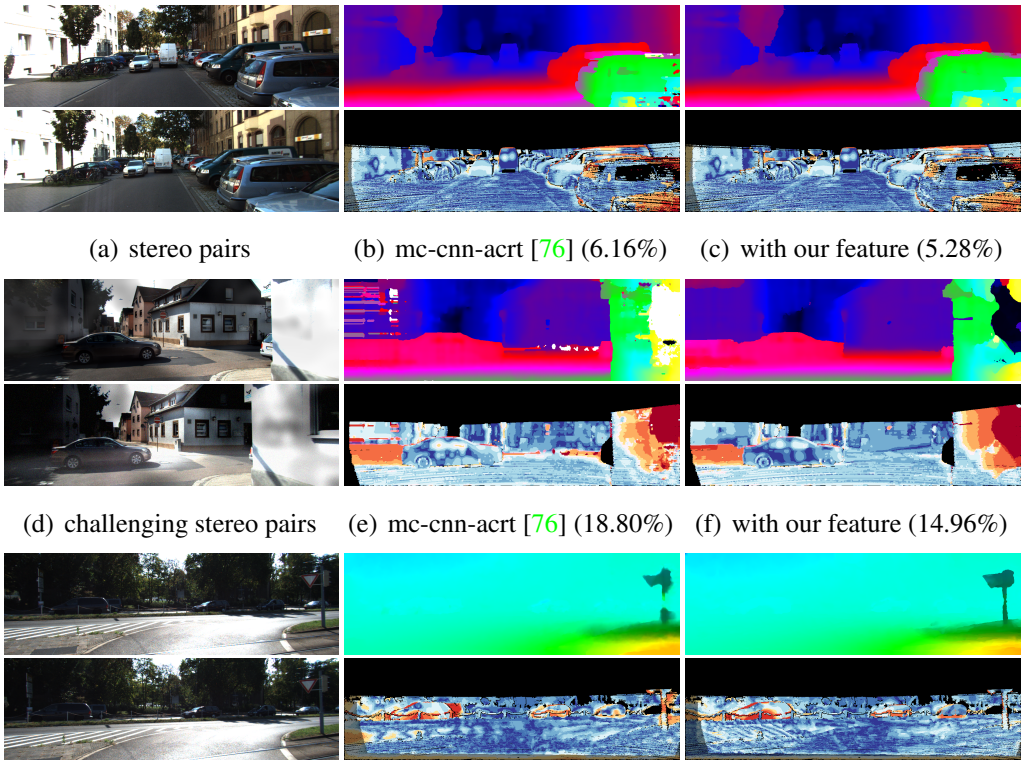
calculated by:

$$er = \frac{1}{|I|}\sum_{p\in I}\delta_p \tag{6.1}$$

$$\delta_p = \begin{cases} 1 & \text{if } \|l(p) - l_0(p)\|_2 > 3 \\ 0 & \text{otherwise,} \end{cases}$$

where $|I|$ is the number of valid pixels in an image; $l(p)$ is the matching result at pixel p; and $l_0(p)$ is the ground-truth displacement at p. The results show that our features found can significantly improve the performance of existing dense matching algorithms, especially with respect to popular features like color, gradient or census.

When compared to the current best MC-CNN matching-cost computation [76], our features perform better in all matching algorithms tested. Fig. 6.1 illustrates the improvements using examples from the KITTI datasets. It shows that matching algorithms using our features perform better in some large smooth regions such as cars.



(a) stereo pairs (b) mc-cnn-acrt [76] (6.16%) (c) with our feature (5.28%)

(d) challenging stereo pairs (e) mc-cnn-acrt [76] (18.80%) (f) with our feature (14.96%)

(g) optical flow pairs (h) PMBP [8] (7.99%) (i) with our feature (4.88%)

Figure 6.1: Improvements using features found by our method. (b)∼(c), (e)∼(f) and (h)∼(i) are visualized disparity/optical-flow results (upper: color coded by KITTI stereo/flow benchmarks) and visualized error map (lower) corresponding to stereo/flow pairs (a) (d) (g), respectively. 3-pixel threshold error rates are reported in the captions. Gaussian noises (only right view) and irregular illuminance variations (both views) are added to the stereo pair of (d) to evaluate the robustness of the features to noise and illuminance variations.

Further, our features are more robust to illuminance variations and noises. More importantly, they are general and can be used in all 9 stereo and optical-flow matching algorithms. In contrast, MC-CNN [76] can only be used in the three integer-precision stereo-matching algorithms due to its time complexity, as it needs to use all possible patch pairs to calculate the matching-cost matrix. This requirement is impractical for optical flow and continuous or pyramid matching algorithms.

Table 6.2: Comparison of consistency and distinctiveness with respect to different features on the KITTI 2012 dataset

Feature Types	Stereo Matching		Optical Flow	
	f_1	f_2	f_1	f_2
Color	0.14	0.17	0.23	0.09
Gradient	0.10	0.29	0.16	0.14
Grad+Color	0.11	0.31	0.19	0.17
Census	0.08	0.49	0.15	0.28
dense sift	0.06	0.57	0.11	0.39
dense surf	0.07	0.52	0.13	0.32
MC-CNN fast [76]	0.20	0.77	×	×
MC-CNN acc [76]	0.13	0.80	×	×
Our Four Sets of	0.05	0.76	0.06	0.51
Non-dominated	0.09	0.79	0.11	0.55
Fast Features	0.15	0.81	0.17	0.58
Found	0.24	0.83	0.25	0.60
Our Four Sets of	0.07	0.80	0.05	0.57
Non-dominated	0.11	0.83	0.1	0.62
Accurate Features	0.16	0.86	0.16	0.65
Found	0.24	0.88	0.25	0.68

6.2.2 Analyzing Features through Consistency and Distinctiveness

In this subsection, we compare different features and explain why our features learned can outperform other state-of-the-art approaches. Features are compared with respect to how they satisfy the consistency and the distinctiveness principles by calculating f_1 (Eq. (3.2)) and f_2 (Eq. (3.5)). For f_1, we first normalize their values by subtracting $\min(f_1)$ and dividing by $\max(f_1) - \min(f_1)$, before comparing the average normalized values. Then, f_2 is calculated by assuming integer displacements. JND m' in f_2 is set to zero, as different values of m' do not change the relative partial order of solutions as well as the monotonic behavior of f_2 (as illustrated in Fig. 3.4). For exam-

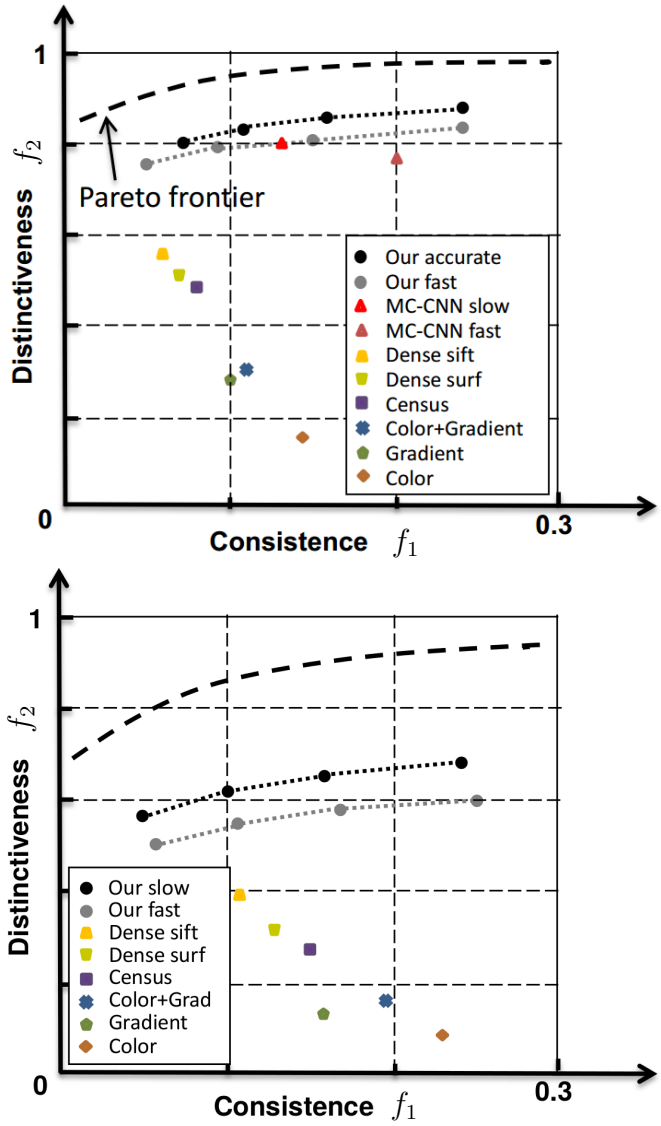


Figure 6.2: Referring to details in Section 6.2.2, an illustration of Pareto optimality and a comparison of the solutions found in stereo matching. The x axis shows the consistence f_1 (Eq. (3.2)), the smaller the better. The y axis shows the distinctiveness f_2 (Eq. (3.5)), the larger the better. The target of the search is to find the Pareto optimal frontier with trade-offs between f_1 and f_2. The results of ten feature methods are plotted, and four non-dominated solutions with each of our CNN solvers are shown. Data is collected using the KITTI datasets [5] and shown in Table 6.2. The dash curve illustrates a hypothetical Pareto frontier for some CNN architecture.

ple, our feature always dominates the others regardless of whether $m' = 0$ or $m' = 0.1$.

Table 6.2 shows the values of f_1 and f_2 when setting λ to 0.8, 0.6, 0.4 and 0, respectively, to get four non-dominated solutions on the KITTI 2012 dataset. Fig's 6.2 further depict all the (f_1, f_2) tuples. The results show that the features found by our method are the closest to the Pareto frontier that achieve the best tradeoffs between consistency and distinctiveness. For stereo matching, we get the following dominance order: our feature (fast or accurate) $\succ$ MC-CNN (fast or accurate) $\succ$ dense sift $\succ$ dense surf $\succ$ Census $\succ$ gradient (or color+gradient) $\succ$ color. Such a dominance order is also visible in Fig. 6.2. For optical flows, we get similar conclusions.

6.2.3 Evaluations of Our Features Using KITTI Benchmarks

We evaluate our features using the stereo and optical-flow benchmarks by embedding it in one of the existing matching frameworks. For stereo evaluations, the whole stereo framework in [76] with the matching-cost matrix calculated by our accurate 256-channel features are employed. The depth edges are further refined by a weighted median filter (with radius=11, $\epsilon = 0.05$).

Table 6.3 shows the evaluation results on the stereo benchmarks. When evaluated on non-occlusion areas, our feature with the SGM stereo matching algorithm (CNNF+SGM) ranks top in the KITTI 2012 and 2015 stereo benchmarks.[1]

[1]With respect to the public KITTI benchmarks at http://www.cvlibs.net/datasets/kitti/index.php, our visualized results and ranking data are available at (by clicking to access each) (a) 2012 stereo benchmarks; (b) 2015 stereo benchmarks; (c) 2012 flow bechmark; and (d) 2015 flow benchmarks.

Table 6.3: Stereo matching evaluations (non-occlusion areas) on two KITTI stereo benchmarks with ranking data, error rates and run times reported. The stereo matching algorithm [76] with our learned feature achieves the current best accuracy.

KITTI 2012 Benchmark			
Method	Rank	Error Rate	Run Time (Device)
Our CNNF+SGM	**1**	**2.28%**	71s (GPU TESLA K40)
PCPB [60]	2	2.36%	68s (GPU Titan X)
Displets v2 [20]	3	2.37%	265s (CPU >8 cores)
MC-CNN-acrt [76]	5	2.43%	67s (GPU Titan X)
cfusion [55]	6	2.46%	70s (GPU Titan X)
KITTI 2015 Benchmark			
Method	Rank	Error Rate	Run Time (Device)
Our CNNF+SGM	**1**	**3.04%**	71s (GPU TESLA K40)
Displets v2 [20]	2	3.09%	265s (CPU >8 cores)
PCPB [60]	3	3.17%	68s (GPU Titan X)
MC-CNN-acrt [76]	4	3.33%	67s (GPU Titan X)

For optical-flow evaluations, we only test our features on classical two-frame optical-flow applications. Since our method does not use extra information like multi-view frames, stereo/depth and/or semantic segmentation results, for fair comparison, methods like motion stereo flows and multi-view flows are not compared. PMBP [8] with slanted surface assumption (just like the PMBP-stereo) are implemented to collect the initial results for both the left and right views. Consistency checking as that in [76] is used to find occlusions and invalid regions which are recovered by an iterative weighted median filter (with radius=20, ϵ=0.075). (We applied the weighted median filter several times until all the invalid regions are filled with displacement values.) Finally, we employ a 5×5 median filter and a bilateral filter (with radius=11, ϵ=0.1). The above three post-refinement steps are repeated twice to collect the final results for benchmark evaluations.

Table 6.4: Optical-flow evaluations on two KITTI flow benchmarks with ranking data, error rates and run times reported. The optical-flow algorithm [8] with our learned feature achieves the current best accuracy among state-of-the-art two-frame optical-flow algorithms.

KITTI 2012 Benchmark			
Method	Rank	Error Rate	Running Time (Device)
Our CNNF+PMBP	**11**	**4.70%**	30min (CPU 1 core)
PatchBatch-s	12	4.81%	60s (GPU)
CNN-HPM [4]	13	4.89%	23s (GPU)
PatchBatch [17]	15	5.29%	50s (GPU)
PH-Flow [74]	21	5.76%	800s (CPU)

KITTI 2015 Benchmark			
Method	Rank	Error Rate	Run Time (Device)
Our CNNF+PMBP	**4**	**12.26%**	45min (CPU 1 core)
SOF [50]	7	16.81%	6min(CPU)
JFS [28]	8	17.07%	13min (CPU)
PatchBatch [17]	14	21.69%	50s (GPU)

The original PMBP [8] was run on a CPU with one thread. Its computational complexity is KNR^2, where K is a constant, N is the number of pixels in an image, and R is the patch width. It is slow and cannot be parallelized due to the spatial-propagation step. We use PMBP instead of other improved algorithms based on PMBP because it is the first and the most classical version among these algorithms. There are some faster versions, such as GC-LSL [67] and SPMBP [43] that achieved speed through limiting the candidate displacement space or discarding the spatial-propagation step. Our features can be easily introduced in all these algorithms to improve their accuracy. When our features are embedded, using half of the patch width slightly affects accuracy, but the whole algorithm can be sped up by $2 \sim 3$ times.

Table 6.4 shows the ranks and the error-rate information. Our method ranks 11^{th} on the KITTI 2012 benchmarks and 4^{th} on the

Table 6.5: Efficiency and time complexity of our feature extractors without the matching algorithm. The running times are elapsed times for just one view (resolution: 1242×375). The numbers in brackets are the CPU run times.

Feature Extractor	Time Complexity	Running Time		
		$n = 16$	$n = 64$	$n = 128$
fast-stereo	$O(4n^2N)$	0.05s (4s)	0.15s (10s)	0.3s (19s)
fast-flow	$O(4n^2N)$	0.1s (7s)	0.4s (21s)	0.7s (37s)
acc-stereo	$O(84n^2N)$	3s	12s	21s
acc-flow	$O(195n^2N)$	7s	23s	35s

Table 6.6: Elapsed time (1-core CPU) when embedded in different matching algorithms.

Matching Alogirthm	Original Running Time	Running Time With Our Feature		
		$n = 16$	$n = 64$	$n = 128$
CostFilter [57]	8s	13s	27s	41s
PMBP flow [8]	800s	500s	1300s	2100s
Particle BP [65]	5s	7s	15s	27s
SGM [25]	3s	5s	11s	20s

2015 optical-flow benchmarks. More importantly, the results show that our method is the current best two-frame optical-flow method without using extra multi-view, motion stereo or semantic segmentation reinforcement.

6.2.4 Analysis of Time Complexities

In this subsection, we analyze the time complexities of our feature-extraction procedure and those of using it in existing matching algorithms. Each convolutional manipulation of one pixel in a channel is set as one basic unit of computation time (while ignoring activations, like ReLU, max-pooling and Sigmoid computations due to their negligible times). For example, for an n-channel fast-feature extractor, we have five convolutional layers for an N-pixel image.

According to the DNN architectures in Table 5.1, the time complexity for the fast-feature extractor is $O(n^2N)$, as it needs $(4n^2 + n)N$ units of convolutional computations. Table 6.5 lists the time complexities and elapsed times for all the feature extractors.

The time complexities of using our features in matching algorithms directly depend on the complexities of the cost functions in Eq. (5.2). Since both are linear with n channels of the feature descriptor, the time complexity when using our feature is $O(nM)$ if the complexity of the matching algorithm is $O(M)$. Table 6.6 compares the run times for the matching algorithms before and after using our features.

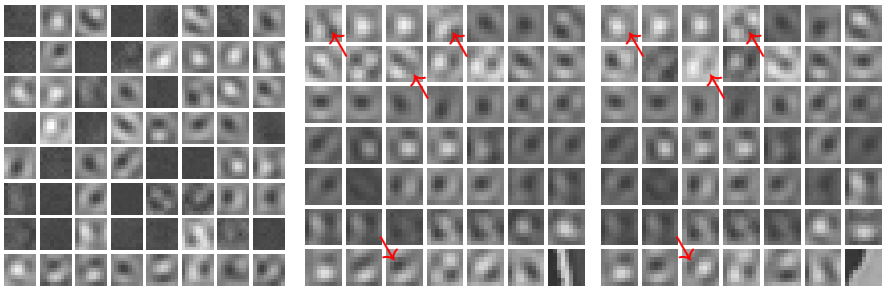
6.3 Experiments with MLNN

In this section, we studied and evaluate the MLNN implementation of the general model (Eq.1.4). We first illustrate the performance improvements of the MLNN by analyze the filter kernels and the learned meta-knowledge (Section 6.3.1), then, evaluate the features found by MLNN implementation by comparing them with some state-of-the-art base NNs (*e.g*. residual net, inception net *etc*.) in Section 6.3.2.

6.3.1 Illustration of Performance Improvement

This subsection illustrates the source of performance improvements in MLNN from two aspects.

Meta-level learning. As discussed earlier, traditional NN optimizes the average performance across all inputs, without tailoring its



(a) (b) (c)

Figure 6.3: Image Super-resolution (Appendix A) used to compare filter kernels. (a) The fixed 64 first-layer filters of the original CNN model. (b) and (c) MLNN model: adaptive filter kernels on two different input patches (last one). Most of the filters are slightly different, whereas some are significantly different (indicated by red arrows). MLNN, based on adaptive kernels, can use a simple 48 first-layer model to achieve better performance.

behavior to address the diversity of inputs. In contrast, MLNN uses SNN to learn input-specific meta-knowledge and provides this information in the form of dynamic weights to BLNN. An example of the meta-knowledge learned is visualized in Stage (d) of Fig. 5.3. The example shows that SNN learns the meta-information for each pixel, and different colors correspond to different classes of inputs.[2] By introducing meta-knowledge into BLNN, the new model can adapt itself to different types of inputs, leading to better adaptability of MLNN.

Dynamic neural networks. Our MLNN successfully introduces flexibility into a NN by using dynamic weights. As shown in Fig. 6.3, the filter kernels of MLNN are always adaptive to inputs, whereas those of traditional NNs are fixed. As a result, even when using simpler 48-channel outputs for MLNN, its performance is still better than a fixed model with 64-channel outputs.

As discussed in Sec. 2.2, existing weight-dynamic NNs are ac-

[2]More visualized examples on the meta-knowledge learned are available in the appendix.

tually special cases of MLNN shown in Fig. 4.1. They do not have a term to realize the balance between generalization (Φ) and input-adaptation (Γ). In contrast, MLNN uses a small number of additional parameters in SNN to make the model dynamic to inputs. Most of the parameters are used in the generalization term Φ to avoid overfitting and to generalize to unknown scenes.

We can find evidence of MLNN's good generalization behavior by comparing the filter kernels of different inputs. Fig. 6.3 visualizes the filter kernels of the first layer in a learned MLNN model. For the two different input patches, most of the filters are slightly different from each other and only some are significantly different. It is likely that those similar filters realize the generalization of the MLNN model, whereas those that are significantly different adapt the model's behavior to different inputs.

6.3.2 Evaluations and Comparisons of the MLNN

Here, we show that MLNN could significantly improve the accuracy of results in dense matching. MLNN's performance is also verified with many other low-level and hight level vision applications which are presented in the appendix. Each simple convolutional layer is replaced with MLNN block (as shown in Fig.6.5). We also compare it with the recently proposed residual and inception strategies. For the residual strategy, we employ more advanced 4D transformed setting [71] instead of the original setting [22]. For the inception strategy, a four-branch inception block with residual connections similar to [66] is tested.

Fig. 6.5 shows these architectural blocks. Limited by avail-

Table 6.7: Performance evaluations and comparisons of different network architectures in the feature learning system

Architectural Setting	Stereo Matching			
	param	speed (fps)	(f_1, f_2)	error rate
fast-64	0.15M	6.5	$(0.094, 0.73)$	6.31%
Res-Incep	0.14M	4.1	$(0.096, 0.76)$	6.21%
4D Res-Net	0.34M	2.8	$(0.095, 0.77)$	6.16%
MLNN-64	0.45M	2.1	$(0.093, 0.79)$	6.04%
fast-256	2.4M	0.9	$(0.091, 0.79)$	6.01%
Architectural Setting	Optical Flow			
	param	speed (fps)	(f_1, f_2)	error rate
fast-64	0.41M	2.7	$(0.12, 0.50)$	10.89%
Res-Incep	0.24M	2.3	$(0.12, 0.52)$	10.74%
4D Res-Net	0.7M	1.7	$(0.11, 0.53)$	10.67%
MLNN-64	0.89M	1.3	$(0.11, 0.55)$	10.53%
fast-256	8.2M	0.4	$(0.11, 0.55)$	10.47%

able computational resources, only fast 64-channel architectures are tested to illustrate the performance differences. Table 6.7 and Fig. 6.4 lead to the following observations.

a) 64-channel MLNN-net could already achieve similar accuracy as that of 256-channel CNN net and far better than the 64-channel CNN net. While, MLNN-net runs more than two times faster and utilizes fewer parameters in the model.

b) MLNN-net outperforms the state-of-the-art network architectures including the residual inception [66] and 4D-residual net [71] both in stereo matching and optical flow applications.

When involving large data diversity, MLNN will have big superiorities over state-of-the-art DNN architectures because the learned model are adaptive with the inputs. For feature extraction in dense matching, each pixel will become an independent target. The scale of the task will easily exceed ten million. Moreover, different regions will have variances in illuminance and texture conditions. These

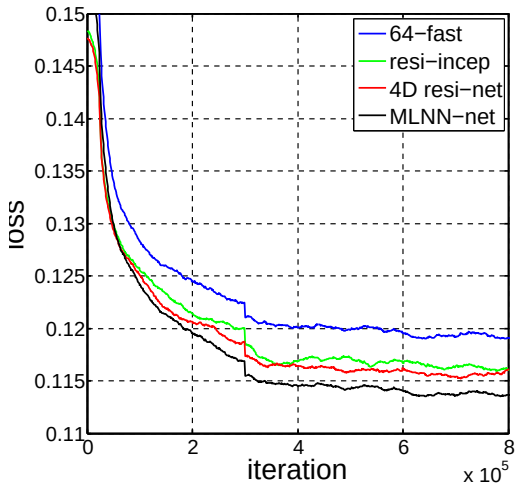


Figure 6.4: Convergence of training in different network settings as illustrated by stereo matching. 4D-residual and residual inception blocks are implemented in the original 64-channel fast architectures to replace the original five convolutional layers.

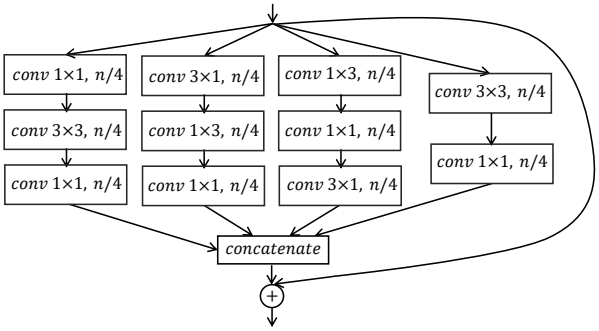
all increase the diversities in the feature extractor task. As a result, MLNN could achieve the best performance for feature extraction in dense matching even compared with several state-of-the-art network architectures.

6.4 Summary of the Experiments

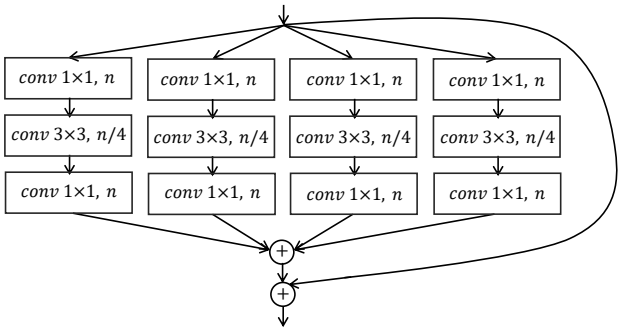
The performance comparisons with existing features under different matching algorithms show that our learned features could achieve the lower error rates in various dense matching algorithms. The study of the trade-offs between the consistency and distinctiveness principles and the evaluations of different features through the consistency and distinctiveness principles verify that either of these two principles is indispensable for finding good features for dense matching and our learned feature could dominate all other existing fea-

tures. Finally, the MLNN experiments show that the adaptive MLNN feature extractor has big superiorities over other feature extractors defined by traditional NNs.

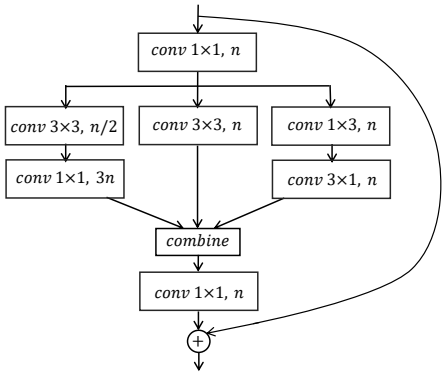
□ **End of chapter.**



(a) Residual Inception Block



(b) 4D-Residual Block



(c) MLNN Block

Figure 6.5: Advanced architectural settings studied. Each of the original-layer sets can be replaced by one of the above blocks. Batch normalization (BN) is performed right after each convolution, followed by ReLU (except the last one that is performed after adding the residual connection). Before and after all the five blocks, there is an extra $[1 \times 1, n]$ convolutional layer. (a) Inception-Residual block similar to that in [66]. (b) More advanced 4D-residual block similar to that in [71].

Chapter 7

Conclusion

This thesis solves two important problems in dense matching. a) What are good features that lead to better performance? b) How can these good features be found systematically?

For the first problem, we developed two fundamental principles, namely, the consistency and the distinctiveness principles, and then, formulated them into a general multi-objective formulation for finding good features in dense matching. We also studied the tradeoffs between these two principles and showed that either of these principles were indispensable for good features in dense matching.

For the second problem, we designed two DNN architectures (one for speed and the other for accuracy) as feature extractors and achieved the loss function from the multi-objective function to learn new features for both stereo matching and optical flow. Besides, in the formulation of the feature extractor, we solved the challenge that traditional NNs always find a model with fixed structure and weights by optimizing the average behavior across all training data without specializing to their variations using the proposed MLNN and employing into the advanced DNN feature extractors to realize

the adaptive behaviors for different inputs in different regions.

Different with existing feature methods, the proposed feature learning model and system design could be used to learn new effective features for any existing matching algorithms, including the most challenging pyramid matching scheme which has strict requirement for interpolation at sub-pixel locations. Also, the proposed MLNN which worked as feature extractors is especially beneficial when the data has large diversity, such as different image types, textures, and radiometric and noise conditions. Usually, as the data scale is increased, data diversity is more prominent, and the improvements brought by MLNN will be more pronounced. Moreover, the MLNN could not only be used to improve performance in feature extraction for dense matching, but many other applications (*e.g.* image denoising, upsampling and classification *et al.*) where data-diversity is significant (as shown in the appendix).

There are several directions which can be studied in the future. In this thesis, we have focused only on two-frame matching applications (stereo matching and optical flows). Other applications, such as multi-view matching/flow or reconstruction, can lead to further improvements. Secondly, our approach is based on existing matching frameworks and does not develop new approaches specialized for the features learned. The design of such new matching schemes by ameliorating some existing work will be beneficial. Thirdly, the limitations of our method appear in those reflection regions (such as windows of a car) that are prevalent in most existing work. In these regions, no useful matching information can be found. To further improve the performance in these regions, semantic segmentation information would be helpful (as done in [20, 61]). Finally, design-

ing new DNN architectures and learning algorithms will improve the accuracy of results, leading to better features found. Using parallel computers with GPUs will further enable effective explorations of large search spaces in manageable time.

□ **End of chapter.**

Appendix A

Experimental Verification and Analysis of MLNN

In the appendix, we provide more evidences to show the effectiveness of the proposed MLNN. First, we analyze MLNN's links to the existing polynomial and residual nets which are all used to improve the power of the DNN model (Section A.1). Then, we design more other experiments to verify the effectiveness of the proposed MLNN, including low-level image upsampling, denoising (Section A.3) and high-level image classification (Section A.4). Which shows that the proposed MLNN are beneficial when the dataset has large diversity, such as different image types, textures, and radiometric and noise conditions.

A.1 Relation to Polynomial and Residual Nets

Polynomial NNs (PNNs) [56, 68] are designed to learn a polynomial model and process more powerful capabilities to represent or classify data. The popular CNNs can be understand as a near linear system, although they use activation functions to increase their

power.

Our MLNN can be treated as one kind of well-controlled and organized PNNs. Since we utilize the dot product to combine meta-knowledge $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$ into the original convolutional layers, $\Gamma_{\mathbf{w}}(\mathbf{X}_i)$ can be represented as a near-linear transformation of $\mathbf{X}_i$. After each MLNN layer, the degree of the model will increase quadratically, which significantly increases its complexity compared with CNNs. Meanwhile, due to the well controlled architecture and parameters, MLNN overcomes the common problems of PNNs [15, 45] that are hard to implement and can easily overfit.

In a more general sense, the MLNN layer can also be used as one kind of quadratic activation function to effectively increase the searching space of the NN model.

Deep Residual Nets [23] insert shortcut connections and turn a network into its counterpart residual version. This successfully solves the convergence problem in very deep NNs and significantly improves the classification accuracy.

Similar to residual learning, we use element-wise additions to combine meta-knowledge $\Gamma_{\mathbf{b}}(\mathbf{X}_i)$. In particular, in implementing Eq. (5.4) and (5.5), when we remove the branch of $\Gamma_{\mathbf{w}}$, the SNN (as illustrated in Fig. 5.3) is directly transformed to residual connections as those in [71] and [23]. In a general sense, the residual net partially realizes dynamic learning in the bias term. We believe that our MLNN helps partially understand the performance of residual nets. On the other hand, the convergence behavior of our MLNN can also be attributed to the realization of residual learning in MLNN.

Table A.1: Baseline Network Architectures

Layer Set	Low-Level Vision			
	Arc 1 3-layer [14]	Arc 2 3-layer [14]	Arc 3 4-layer [14]	Arc 4 21-layer [80]
set 1	$\begin{bmatrix}\text{conv},9\times9,64\\ \text{ReLU},64\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},9\times9,128\\ \text{ReLU},128\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},9\times9,128\\ \text{ReLU},128\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},3\times3,64\\ \text{ReLU},64\end{bmatrix}\times1$
set 2	$\begin{bmatrix}\text{conv},1\times1,32\\ \text{ReLU},32\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},1\times1,64\\ \text{ReLU},64\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},5\times5,64\\ \text{ReLU},64\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},3\times3,64\\ \text{BatchNorm}\\ \text{ReLU},64\end{bmatrix}\times19$
set 3	$[\text{conv},5\times5,1]\times1$	$[\text{conv},5\times5,1]\times1$	$\begin{bmatrix}\text{conv},1\times1,64\\ \text{ReLU},64\end{bmatrix}\times1$	$[\text{conv},3\times3,1]\times1$
set 4			$[\text{conv},5\times5,1]\times1$	
Layer Set	High-Level Classification			
	Arc 5 4-layer [34]	Arc 6 4-layer	Alexnet [39]	
set 1	$\begin{bmatrix}\text{conv},5\times5,20\\ \text{Maxpool},2,2\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},5\times5,32\\ \text{Maxpool},2,2\\ \text{ReLU},32\\ \text{BatchNorm}\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},11\times11,4,96\\ \text{ReLU},96\\ \text{Maxpool},3,2\\ \text{BatchNorm}\end{bmatrix}\times1$	
set 2	$\begin{bmatrix}\text{conv},5\times5,50\\ \text{Maxpool},2,2\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},5\times5,32\\ \text{ReLU},32\\ \text{Maxpool},2,2\\ \text{BatchNorm}\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},5\times5,256\\ \text{ReLU},256\\ \text{Maxpool},3,2\\ \text{BatchNorm}\end{bmatrix}\times1$	
set 3	$\begin{bmatrix}\text{fc},500\\ \text{ReLU},500\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},5\times5,64\\ \text{ReLU},64\\ \text{Maxpool},2,2\end{bmatrix}\times1$	$\begin{bmatrix}\text{conv},3\times3,384\\ \text{ReLU},384\\ \text{Maxpool},3,2\end{bmatrix}\times3$	
set 4	$[\text{fc},10]\times1$	$[\text{fc},10]\times1$	$[\text{fc},4096]\times2$	
set 5			$[\text{fc},1000]\times1$	

A.2 Parameters and Network Settings

The baseline network architectures in different applications are shown in Table A.1. We replace some convolutional or fully connected layers with our MLNN layers to get new dynamic models. The performance of these new MLNN models are measured and compared with the original standard models in different applications. Fig. A.6 shows the corresponding MLNN network architectures of Arc 1-6 (Table A.1) and Alexnet [39].

Given an original $[k \times k, n]$ convolutional layer, to make sure the speed and the number of parameters won't be changed greatly, here, we talk about the parameters' setting on realizing a corresponding MLNN layer. For BLNN, we employ a $[k \times k, 3n/4]$ convolutional

layer. For SNN, in each SNN branch ($\Gamma_{\mathbf{w}}$ and $\Gamma_{\mathbf{b}}$), we use two 1×1 convolutional layers with ReLU activation between them. Specifically, we implement one $[1 \times 1, n/4]$ and one $[1 \times 1, 3n - 2]$ convolutional layer for $\Gamma_{\mathbf{w}}$. While, for $\Gamma_{\mathbf{b}}$, $[1 \times 1, n/4]$ and $[1 \times 1, n]$ layers are used. To combine $\Gamma_{\mathbf{w}}$, we first employ $TanH$ (for classification tasks) or $Sigmoid$ (for low-level vision tasks) to restrict the output of $\Gamma_{\mathbf{w}}$, and then reshape it to a 1-band matrix.

A.3 Low-Level Image Quality Improvement

In almost all low-level vision applications, each pixel will be a target and the training data can contain more than ten million samples. The data diversity is very common and significant. Therefore, dynamic MLNN model would possess apparent advantages over standard model.

A.3.1 Image Super-Resolution

Image Super-Resolution aims at recovering a high-resolution image from a single low-resolution image, is a classical problem in low-level computer vision. Dong *et al*. show that the traditional super-resolution algorithms can be replaced with a deep CNN to get better performance and faster speed [14]. Due to the convenience of collecting and controlling of training data, this is one of the most suitable applications to verify the performance of MLNN strategy.

To control the data diversity, we build mixed datasets with five kinds of different images, including the natural images, screenshots, depth images, carton images and oil paintings (with about 100,000

51×51 patches for each type of images, 9/10 for training and 1/10 for testing). We refer natural image as "dataset-1", mixture of natural image and screenshot as "dataset-2" and mixture of all five kinds of images as "dataset-5". Three kinds of network structures proposed in [14] and another much deeper 21-layer residual net [80] are used as the base models for evaluations and comparisons. In each of the comparisons, $3\times$ super-resolution is implemented. We replace the first one (for Arc 1-2), two (for Arc 3) or five (for Arc 4) original convolutional layers with our MLNN layers. All other settings in the training process are set same as those in [14]. After training for 64 epochs, we use PSNR to measure the difference between testing results and ground truths. Evaluations are shown in Table A.2. Comparisons of convergent curves of Arc 1 are shown in Fig.1.2. We can observe that 1) in all the network architectures, our MLNN would help to get 0.1-0.3 dB improvement in PSNR evaluations. 2) As the increasing of the data diversity, the improvement of our MLNN model becomes more obvious. For example, in Arc 1, MLNN only improves 0.14 dB for "dataset-1". While for "dataset-5" which has more significant data diversity, our MLNN model gets 0.23 dB improvement.

For Arc 1, Detailed results of each kind of images in "dataset-5" are shown in Table A.3. Compared with a single CNN model trained on "dataset-5", training the same CNN architecture for each type of images to get five models leads to better performance. Our MLNN performs even better in four out of five cases than CNNs trained for each kind of images. This is because MLNN learns pixel-level meta-knowledge for each pixel and can differentiate input diversities of each pixel even within a single kind of images.

Table A.2: Evaluation using image supper-resolution (PSNR: dB)

Data set	Model	Dataset-1	Dataset-2	Dataset-5
Arc 1	Original	29.97	29.01	32.64
	CNN-c	30.01	29.07	32.71
	MLNN	**30.11**	**29.20**	**32.87**
Arc 2	Original	30.05	29.08	32.73
	MLNN	**30.18**	**29.26**	**32.93**
Arc 3	Original	30.21	29.26	32.88
	MLNN	**30.31**	**29.39**	**33.03**
Arc 4	Original	30.34	29.38	32.99
	MLNN	**30.43**	**29.50**	**33.11**

Table A.3: Evaluations of each kind of images (PSNR: dB)

Data Set	CNN model	Individual model	MLNN model
natural image	29.81	29.97	**30.02**
screenshot	28.09	28.30	**28.38**
depth image	43.89	**44.23**	44.11
carton image	32.77	32.92	**33.02**
oil painting	30.22	30.32	**30.38**

We show examples of the testing results in each types of images in Fig.A.1 and A.2. Arc 1 trained on dataset-5 is used. In each type of the images, our MLNN strategy will help to achieve better results compared with original network model. Especially for "depth images" and "screenshots", the improvements of our MLNN strategy are usually larger than 0.4 dB.

A.3.2 Image Denoising

Image Denoising is another low-level vision application which benefits from the development of the DNN. With the Berkeley Segmentation Database [48] as the ground truth, we randomly add noises to construct the training and testing data. 240 images are randomly

chosen as the training set, 60 images are used for testing. Gaussian noises are randomly added with standard deviation in the range of [0, 50]. We employ simple fast 3-layer Arc 1-2 and the much deeper 21-layer residual net developed in [80] as base network settings. All other settings are kept same as those in [80] and our models are same as those in image super-resolution. The PSNR evaluations are shown in Table A.4 after training for 64 epochs. In all the three network structures, our MLNN strategy would help to get about 0.11-0.16 dB improvement compared with the original CNN models. Testing examples are shown and compared in Fig.A.3. Arc 1 is used as base architecture. MLNN strategy performs a little better by preserving more details in these results.

A.4 High-Level Image Classification

In scene classification tasks, the data diversity is not as significant as low-level vision applications since the existing training datasets only contain no more than one million samples. However, our MLNN strategy still helps to get 0.4$\sim$1.5% accuracy improvement on the following three datasets. Moreover, as the increasing of the data scale, the improvement by our MLNN is more significant.

MNIST database of handwritten digits [41] of 0-9 , has a training set of 60,000 examples, and a test set of 10,000 examples. We use a 4-layer network [34] (Arc 5 in Table A.1) as the base setting which gets a 1.11% error rate on the test set after 10 thousand iterations' training. We then change the first fully-connected layer to MLNN of Eq. (5.4), it helps to get a lower error rate of 0.71%.

Table A.4: Evaluation using image denoising (PSNR: dB)

Architecture	CNN model	MLNN model
Arc-1	28.71	**28.87**
Arc-3	28.87	**28.98**
Arc-4	29.02	**29.14**

Table A.5: Performance comparisons in image classification

Data Set	Model	Top-1 Error (%)	Top-5 Error (%)
Mnist	Arc 5	1.11	—
	MLNN	0.71	—
Cifar-10	Arc 6	18.7	—
	MLNN	18.3	—
	ResNet-44	7.61	—
	Res-MLNN-44	6.97	—
	ResNet-110	6.38	—
	Res-MLNN-110	5.82	—
ImageNet	AlexNet	42.6	19.6
	MLNN-1	42.2	19.0
	MLNN-2	41.3	18.5
	MLNN-3	41.1	18.2
	ResNet-50	24.53	7.89
	Res-MLNN-50	23.27	7.02

CIFAR-10 dataset [38] consists of 60,000 32×32 colour images in 10 classes, with 6,000 images per class (50,000 for training and 10,000 for testing). The CIFAR10 Caffe model [34] (Arc 6 in Table A.1) was trained for the CIFAR-10 classification task. Without any data augmentation, we trained it for 70,000 iterations with a batch size of 100. Table A.5 shows the error rates obtained with and without MLNN. The original Arc 6 achieves a top-1 error of 18.7%. After replacing the fist two convolutional layers by MLNN (as shown in Fig. 5.3) with nothing else changed, the error rate is reduced to 18.3%. We also tested the more complicated 44-layer and 110-layer residual nets [23]. By replacing the first 3×3 layer in each of the residual blocks by MLNN and training them with data

augmentation, the accuracy has been improved by 0.6%. Since the CIFAR-10 dataset contains only 60,000 samples, data diversity is not as obvious as that in low-level vision applications where each pixel is a target. The performance of MLNN is, therefore, limited, but we still can achieve 0.4$\sim$0.6% accuracy improvement.

ImageNet [13] is a data set with over 15 million labeled high-resolution images belonging to roughly 22,000 categories. We use the ILSVRC-2012 which is a subset of ImageNet with 1000 images in each of 1000 categories for training and another 50 in each category for validation.

We first use AlexNet [39] as the base setting of the network architecture, which contains five convolutional layers, three fully connected layers and some non-linear activations. For MLNN-1, we replace the fist fully connected layer of AlexNet by the implementation of Eq. (5.4); for MLNN-2, we change the 3^{rd} and the 4^{th} convolutional layers to the MLNN implementation of Eq. (5.5); and for MLNN-3, we include all the changes in MLNN-1 and MLNN-2. We fixed all other settings and trained the models for 310,000 iterations without data augmentation. We then tested the more advanced ResNet-50 [23] with data augmentation to increase data diversity and to avoid overfitting. We replaced the bottleneck 3×3 layer of each residual block by MLNN layers (with 16 layers changed).

Table A.5 reports the top-1 and top-5 error rates on the validation data. Convolutional nets with MLNN outperform the base models by a large margin (0.4%-1.5% in top-1 error rate and 0.6%-1.4% in top-5 error rate). Moreover, the improvement is more pronounced as we increase the number of MLNN layers from one to three in the base model.

A.4.1 Effects on Number of SNN Branches

We first study whether improvements are caused by adding extra branches to the original network. In the CNN-c model (Table A.2), we use concatenation to replace the dot product and element-wise addition in Eq. (5.5). That is, CNN-c has the same network branches and even more parameters when compared to MLNN. However, CNN-c has very limited improvements (0.04$\sim$0.06 dB) when compared to MLNN (0.14$\sim$0.23 dB). This illustrates that the improvements brought by MLNN are attributed to the its dynamic behavior with respect to each input.

Since the meta-level knowledge is split into two parts ($\mathbf{F_w}$ and $\mathbf{F_b}$) in our implementation, we test the effects of each branch in the SNN implementation individually. Firstly, we replace $\mathbf{F_w}$ by traditional fixed-weight structures and add $\mathbf{F_b}$ directly to $\mathbf{F_w}$ with the output of $\mathbf{F_b}$ adjusted to the same shape as $\mathbf{F_w}$. We find that for image upsampling, PSNR drops from 32.87 dB to 32.68 dB, and for AlexNet classification, the top-1 error rate increases by 0.61%. These results support our claim that adaptive weights are effective in MLNN. Secondly, we change $\mathbf{F_b}$ to an identity mapping [**?**]. We observe that for image upsampling, PSNR drops from 32.87 dB to 32.7 dB and for AlexNet classification, the top-1 error rate increases by 0.73%. It is obvious that both SNN branches are indispensable in our model to get good performance.

A.4.2 Effects on Number of MLNN Layers

The number of the MLNN layers also affects the model's performance. In general, as the number of MLNN layers is increased, the

accuracy of the model will first increase and then begin to drop. For example, in AlexNet [39], if we continue to change the 5^{th} convolutional layer to MLNN to get MLNN-4, we get similar accuracy as MLNN-3. However, if we further change the second fully connected layer to get MLNN-5, the accuracy begins to drop by 0.2%. This happens because after adding a new MLNN layer, the degree of the model will increase quadratically. As we ceaselessly add more MLNN layers to a model, overfitting begins to occur at some point. Our experience shows that in a neural net, we change 1/4$\sim$1/2 of the traditional layers to MLNN layers in order to get good performance.

A.5 Efficiency Comparisons and Analysis

Here, we compare the efficiency of our MLNN models with the standard CNN models. The MLNN strategy is implemented based on the deep learning platform caffe [34] with a TESLA K40C GPU. We compare the computation complexity and the speed of the standard CNN and dynamic MLNN models in Table A.6. For low-level vision applications, the computation complexity is in direct proportion to the number of the parameters in the model. During the implementation, we control the number of parameters by reducing the channels of the output after MLNN layers to offset the extra parameters used in learning meta-knowledge. As a result, the efficiency doesn't decrease too much. While for image classification, the running time of the new MLNN model is only sightly increased by 5-10%.

Table A.6: Complexity and Efficiency Comparisons

Architecture	Model	Params	FLOPs	Speed
Arc 1	Original	8k	8.0×10^3	2.5MPixel/s
	MLNN	8.5k	8.5×10^3	2.3MPixel/s
Arc 4	Original	0.7M	7.0×10^5	21KPixel/s
	MLNN	0.69M	6.9×10^5	21KPixel/s
AlexNet	Original	60M	7.3×10^8	675fps
	MLNN-2	60.6M	7.8×10^8	633fps
	MLNN-3	66M	7.9×10^8	619fps

A.6 Learned Meta-Knowledge

During the training, meta-knowledge are learned and then combined into the weights to make the model adaptive. Here, we show more examples of the learned meta-knowledge using image super-resolution trained with Arc 1 on dataset-5. Fig.A.4 illustrates the learned pixel-level meta-knowledge. The color images (third column) are the the second row of the 48×48 band matrix $\Gamma_{\mathbf{w}}(\mathbf{X})$ which has three non-zero elements (They are visualized as the values of R G and B respectively). Before visualizing, each of the three R G and B channels are normalized by reducing the minimum and then dividing the value range of the whole images. Similarly, for bias term, the second row/element of 48×1 vector $\Gamma_{\mathbf{b}}(\mathbf{X})$ is normalized and then visualized as gray images. We can observe that pixels in these images are categorized into three major classes represented by three major colors (blue, orange and black/shadow around object edges.). This means when these mate-knowledge are combined into the convolutional model, there will be around three kinds of filter kernels for the second channel of the convolutional model. Similar situation also appears in other filter kernels (other rows of $\Gamma_{\mathbf{w}}(\mathbf{X})$ and $\Gamma_{\mathbf{b}}(\mathbf{X})$). As shown in Fig.A.5, the $2 - 21th$ rows of the meta-knowledge $\Gamma_{\mathbf{w}}(\mathbf{X})$

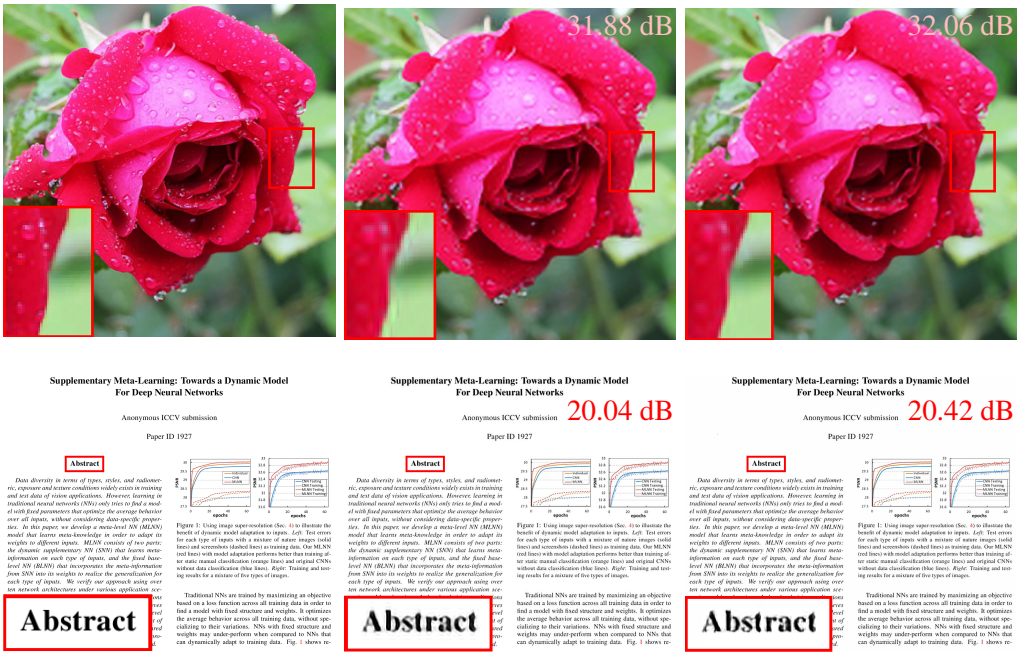
are visualized.

A.7 Conclusion

MLNN studied in this thesis introduces meta-knowledge to DNNs in order to allow them to adapt to inputs with 5%$\sim$10% extra computation costs. Meta-knowledge is first learned by SNN to differentiate various types of inputs and then combined in BLNN to make the MLNN model adaptive to inputs. Besides used for feature extraction in dense matching, the appendix further verify its performance improvement using some other several high- and low-level vision applications. By replacing some standard convolutional or fully-connected layers with our MLNN layers in a traditional NN, our new model can outperform the base model by a large margin.

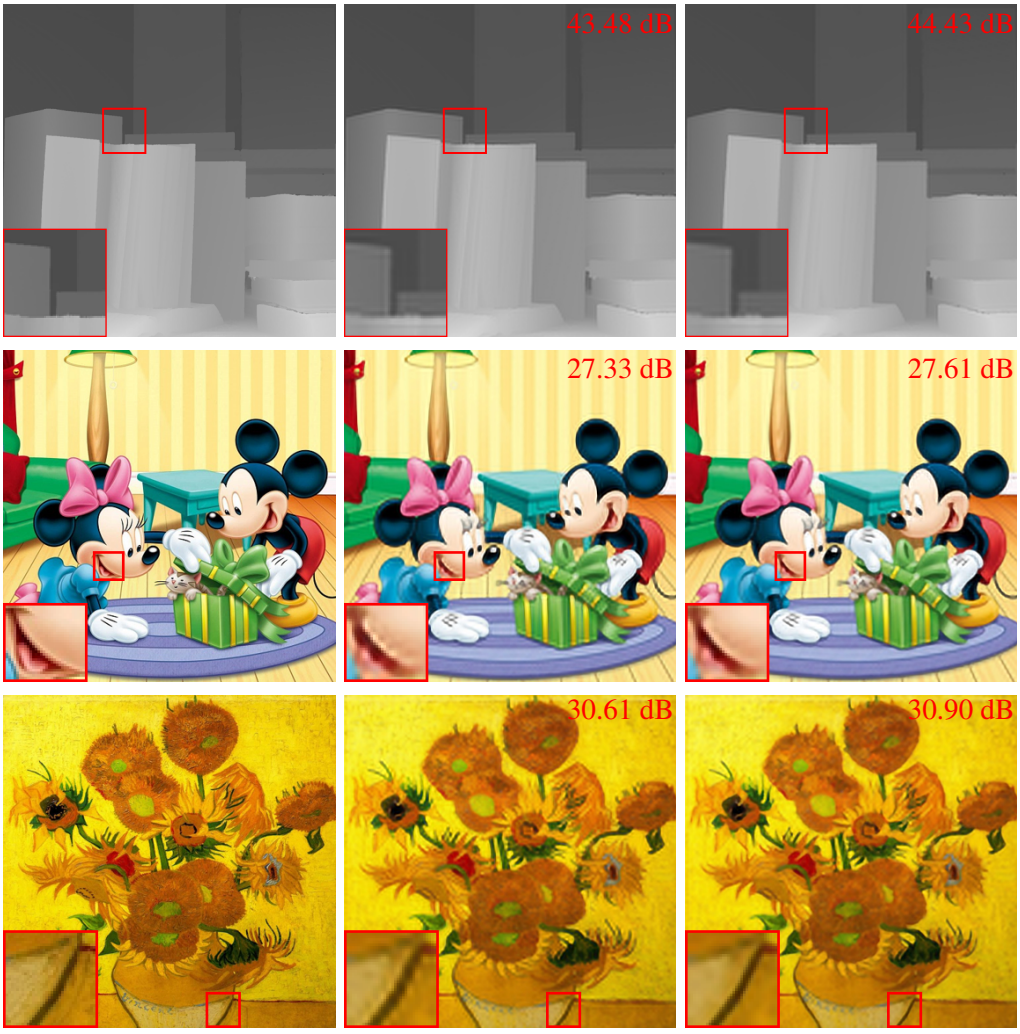
The MLNN layers are beneficial when the dataset has large diversity, such as different image types, textures, and radiometric and noise conditions. Usually, as the data scale is increased, data diversity is more prominent, and the improvements brought by MLNN will be more pronounced.

The limitation of MLNN is that in very deep NNs, overfitting may occur when too many MLNN layers are used (like in a pure MLNN net), as they significantly increase the degree of the model. As a result, MLNN layers must be used in conjunction with standard layers (convolutional or fully-connected layers) in any DNN model.



(a) ground truth (b) original Arc 1 (c) MLNN of Arc 1

Figure A.1: Testing examples of 3× super-resolution using Arc 1 trained on "dataset-5"(mixture of five types of images). An example is shown for each type of images. They (from top to bottom) are natural images, screenshots, depth images, carton images and oil paintings. Amplified details of differences (as tagged in red rectangles) are shown in bottom-left corners. In some edges, original Arc 1 will bring some artifacts. MLNN performs a little better around major edges. PSNR values (top right corner) are measured only in the first YCbCr channels.

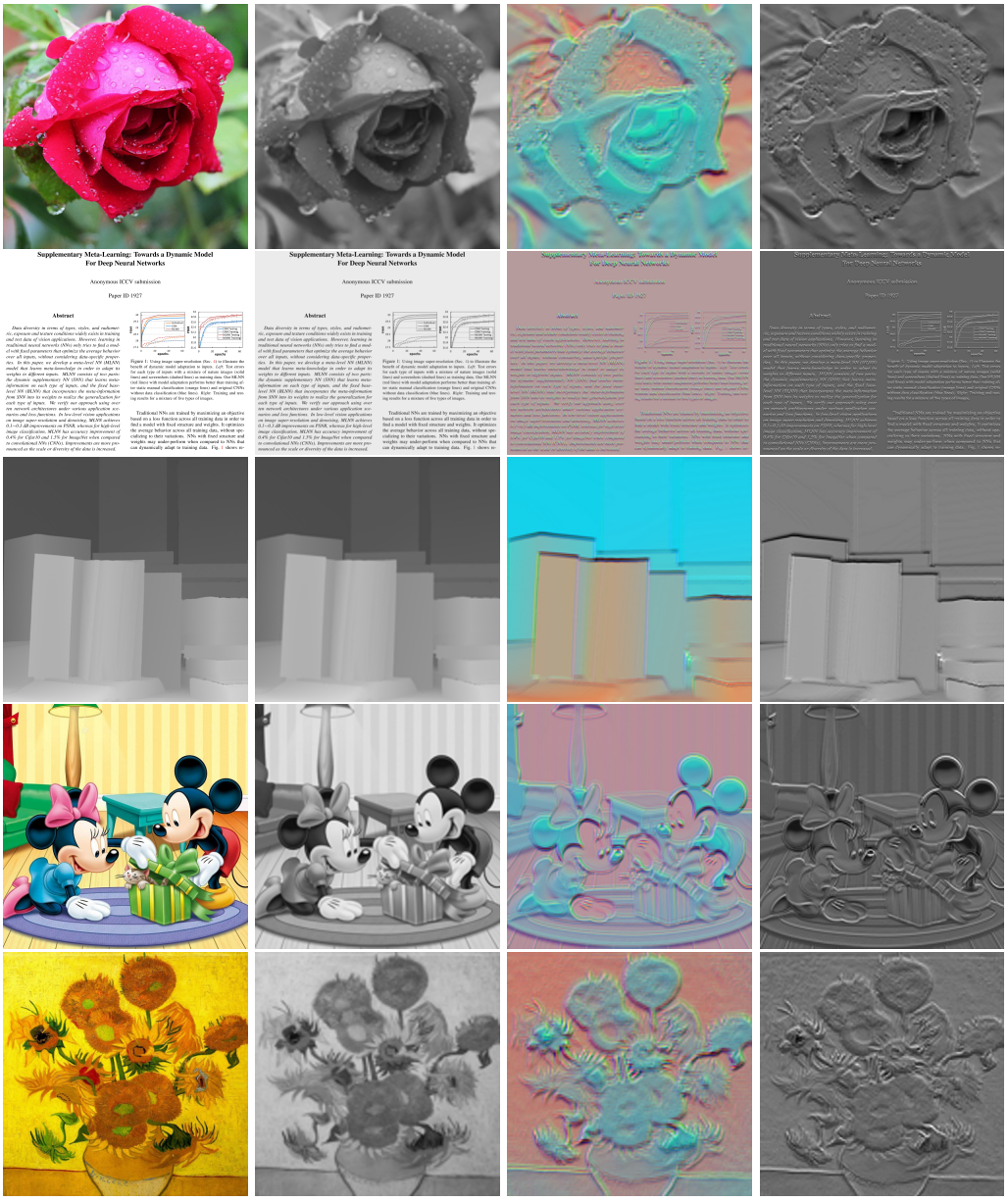


(a) ground truth (b) original Arc 1 (c) MLNN of Arc 1

Figure A.2: Continuing of Figure A.1. Testing examples of $3\times$ super-resolution using Arc 1 trained on "dataset-5"(mixture of five types of images). An example is shown for each type of images. They (from top to bottom) are natural images, screenshots, depth images, carton images and oil paintings. Amplified details of differences (as tagged in red rectangles) are shown in bottom left corner. In some edges, original Arc 1 will bring some artifacts. MLNN performs a little better around major edges. PSNR values (top right corner) are measured only in the first YCbCr channels.

(a) ground truth (b) noisy input (c) original Arc 1 (d) MLNN of Arc 1

Figure A.3: Testing examples of image denoising using Arc 1 trained on 240 images. Bottom-left corners are the amplified red windows to show the details of differences. MLNN seems to be able to preserve more details during the denoising process. PSNR values (top right corners) are measured in all YCbCr channels.



(a) color images (b) input to the model (c) second row of $\Gamma_{\mathbf{w}}(\mathbf{X})$ (d) second row of $\Gamma_{\mathbf{b}}(\mathbf{X})$

Figure A.4: Visualized meta-knowledge. (a) original color images. (b) the first channel of YCbCr is used as input. Before that, $3\times$ bicubic interpolation is used as done in [14]. (c) visualized second row of $\Gamma_{\mathbf{w}}(\mathbf{X})$. (d) visualized second row of $\Gamma_{\mathbf{b}}(\mathbf{X})$.

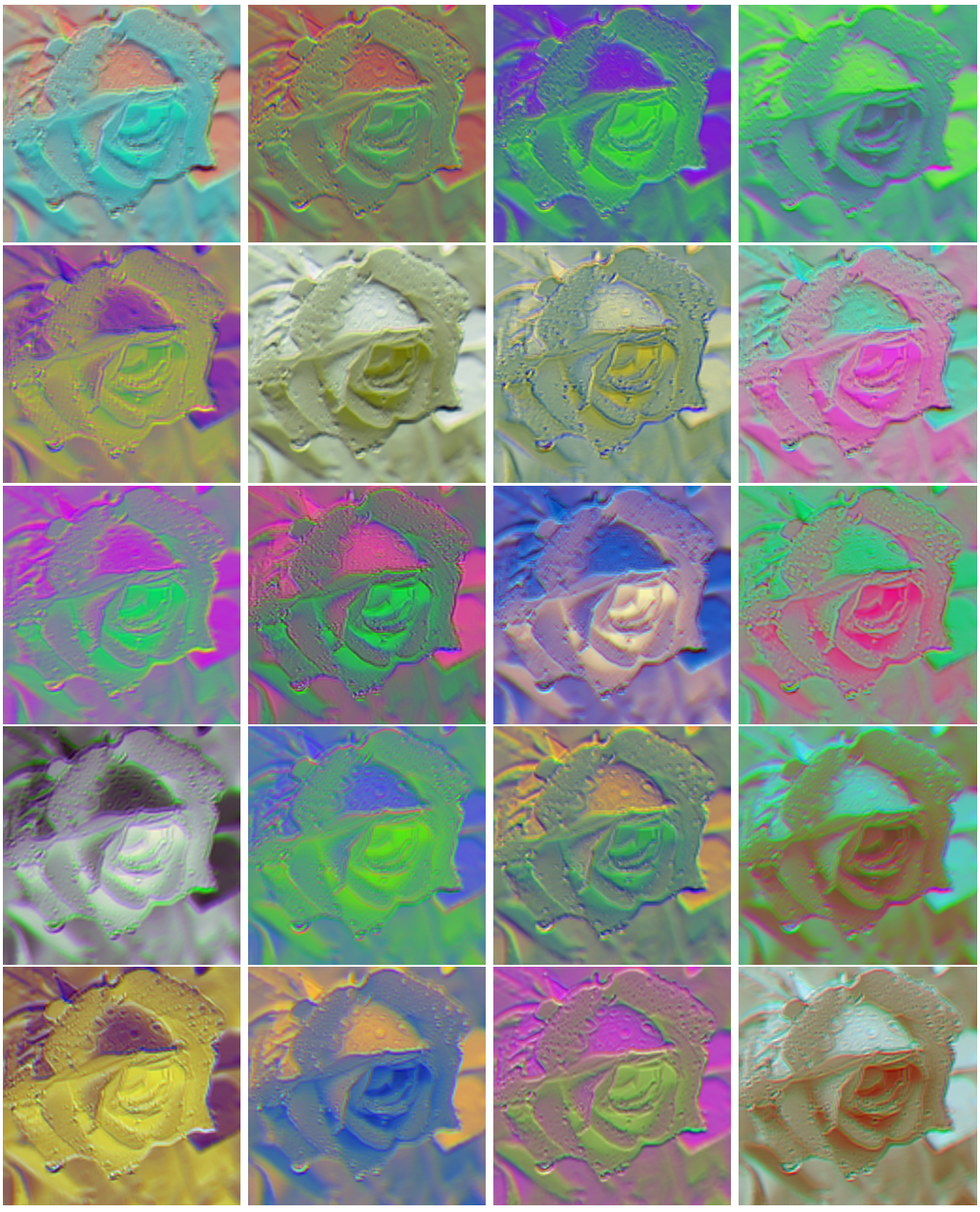
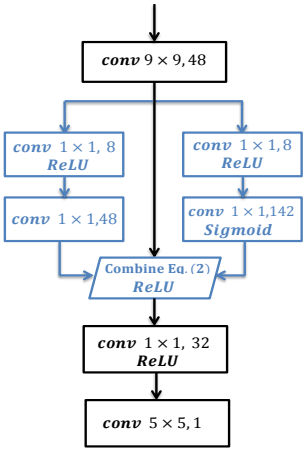
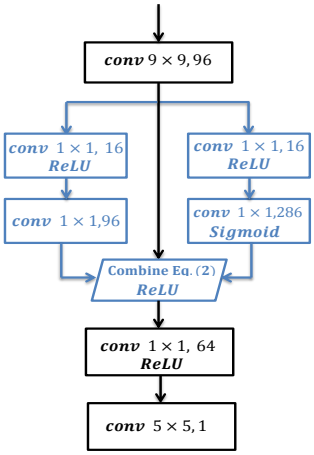


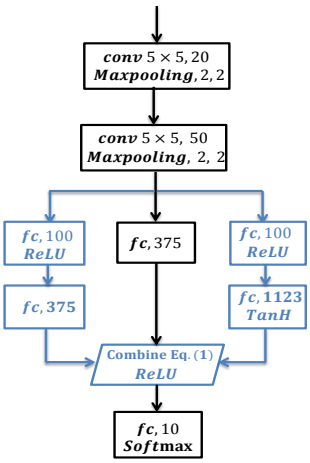
Figure A.5: Learned meta-knowledge: 2-21 row of $\Gamma_{\mathbf{w}}(\mathbf{X})$. Each row has three non-zero elements. They are visualized as R G and B values respectively.



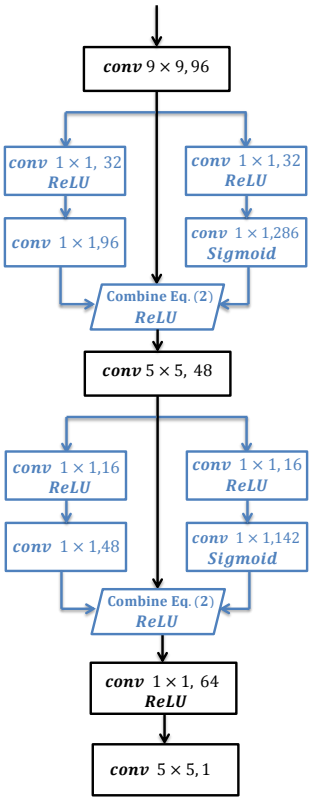
(a) MLNN for Arc 1



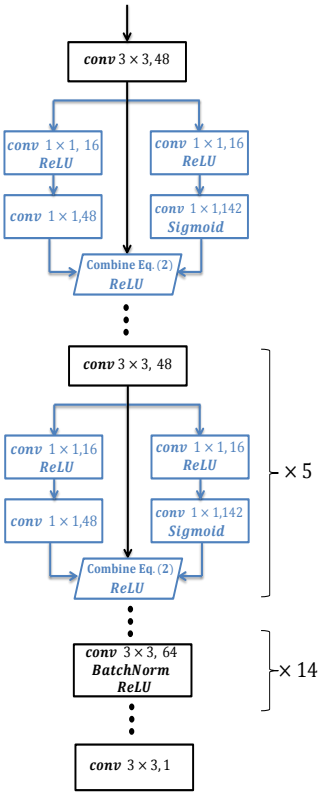
(b) MLNN for Arc 2



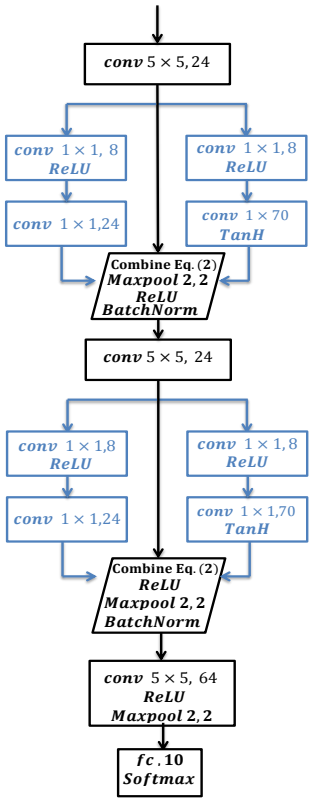
(c) MLNN for Arc 5



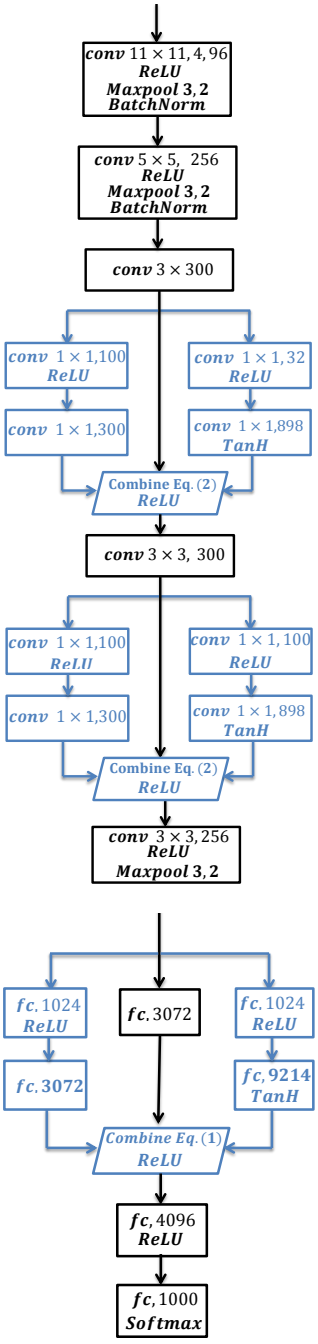
(d) MLNN for Arc 3



(e) MLNN for Arc 4



(f) MLNN for Arc 6



(g) MLNN-3 for AlexNet [39]

Figure A.6: MLNN implementation of original Arc 1-6 and AlexNet [39].

Bibliography

[1] K. Alexandros and H. Melanie. Model selection via meta-learning: a comparative study. *International Journal on Artificial Intelligence Tools*, 10(04):525–554, 2001.

[2] S. Ali and K. A. Smith-Miles. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(1):173–186, 2006.

[3] A. Ansar, A. Castano, and L. Matthies. Enhanced real-time stereo using bilateral filtering. In *Proc. 2nd Int'l Symp. on 3D Data Processing, Visualization and Transmission (3DPVT)*, pages 455–462. IEEE, 2004.

[4] C. Bailer, K. Varanasi, and D. Stricker. CNN based patch matching for optical flow with thresholded hinge loss. *arXiv preprint arXiv:1607.08064*, abs/1607.08064, 2016.

[5] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *Int'l J. of Computer Vision*, 92(1):1–31, 2011.

[6] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. on Graphics*, 28(3):24, 2009.

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008.

[8] F. Besse, C. Rother, A. Fitzgibbon, and J. Kautz. PMBP: Patchmatch belief propagation for correspondence field estimation. *Int'l J. of Computer Vision*, 110(1):2–13, 2014.

[9] M. Bleyer, C. Rhemann, and C. Rother. Patchmatch stereo-stereo matching with slanted support windows. In *British Machine Vision Conf. (BMVC)*, volume 11, pages 1–11, 2011.

[10] P. K. Chan and S. J. Stolfo. Experiments on multistrategy learning by meta-learning. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 314–323. ACM, 1993.

[11] K. Deb. Multi-objective optimization. In *Search Methodologies*, pages 403–449. Springer, 2014.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, 2002.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009.

[14] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[15] M. Donini and F. Aiolli. Learning deep kernels in the space of dot product polynomials. *Machine Learning*, pages 1–25, 2016.

[16] D. G. Ferrari and L. N. De Castro. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Information Sciences*, 301:181–194, 2015.

[17] D. Gadot and L. Wolf. Patchbatch: a batch augmented loss for optical flow. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 4236–4245, June 2016.

[18] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *Int'l J. of Robotics Research*, page 0278364913491297, 2013.

[19] M. Gong, R. Yang, L. Wang, and M. Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *Int'l J. of Computer Vision*, 75(2):283–296, 2007.

[20] F. Guney and A. Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 4165–4175, 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, June 2016.

[24] Y. S. Heo, K. M. Lee, and S. U. Lee. Illumination and camera invariant stereo matching. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008.

[25] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008.

[26] H. Hirschmüller and D. Scharstein. Evaluation of cost functions for stereo matching. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007.

[27] H. Hirschmüller and D. Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 31(9):1582–1599, 2009.

[28] J. Hur and S. Roth. Joint optical flow and temporally consistent semantic segmentation. In *ECCV Workshops*, pages 163–177, 2016.

[29] C.-L. Hwang and A. S. M. Masud. *Multiple objective decision making-methods and applications: a state-of-the-art survey*, volume 164. Springer Science & Business Media, 2012.

[30] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[31] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[32] X. Jia, B. De Brabandere, T. Tuytelaars, and L. V. Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016.

[33] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. 22nd Int'l Conf. on Multimedia*, pages 675–678. ACM, 2014.

[34] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[35] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[36] M. Kim, T. Hiroyasu, M. Miki, and S. Watanabe. SPEA2+: Improving the performance of the strength Pareto evolutionary algorithm 2. In *Parallel problem solving from nature-PPSN VIII*, pages 742–751. Springer, 2004.

[37] B. Klein, L. Wolf, and Y. Afek. A dynamic convolutional layer for short range weather prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4840–4848. IEEE, 2015.

[38] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset, 2014.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[40] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[41] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.

[42] C. Lemke, M. Budka, and B. Gabrys. Metalearning: a survey of trends and technologies. *Artificial ntelligence Review*, 44(1):117–130, 2015.

[43] Y. Li, D. Min, M. S. Brown, M. N. Do, and J. Lu. SPM-BP: Sped-up patchmatch belief propagation for continuous MRFs. In *Proc. of IEEE Int'l Conf. on Computer Vision*, pages 4006–4014, 2015.

[44] A. Liu, W. Lin, M. Paul, C. Deng, and F. Zhang. Just noticeable difference for images with decomposition model for separating edge and textured regions. *IEEE Trans. on Circuits and Systems for Video Technology*, 20(11):1648–1652, 2010.

[45] R. Livni, S. Shalev-Shwartz, and O. Shamir. An algorithm for training polynomial networks. *arXiv preprint arXiv:1304.7045*, 2013.

[46] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[47] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int'l J. of Computer Vision*, 60(2):91–110, 2004.

[48] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 416–423. IEEE, 2001.

[49] X. Mei, X. Sun, W. Dong, H. Wang, and X. Zhang. Segment-tree based cost aggregation for stereo matching. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 313–320, 2013.

[50] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3061–3070, 2015.

[51] K. Miettinen, F. Ruiz, and A. Wierzbicki. Introduction to multiobjective optimization: Interactive approaches. *Multiobjective Optimization*, pages 27–57, 2008.

[52] D. Min, J. Lu, and M. N. Do. Joint histogram-based cost aggregation for stereo matching. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 35(10):2539–2545, 2013.

[53] T. M. Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980.

[54] H. Noh, P. H. Seo, and B. Han. Image question answering using convolutional neural network with dynamic parameter prediction. *arXiv preprint arXiv:1511.05756*, 2015.

[55] V. Ntouskos and F. Pirri. Confidence driven TGV fusion. *arXiv preprint arXiv:1603.09302*, 2016.

[56] S.-K. Oh, W. Pedrycz, and B.-J. Park. Polynomial neural networks architecture: analysis and design. *Computers & Electrical Engineering*, 29(6):703–725, 2003.

[57] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3017–3024. IEEE, 2011.

[58] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[59] D. Scharstein and R. Szeliski. Middlebury stereo vision page. *Online at http://www.middlebury.edu/stereo*, 6, 2002.

[60] A. Seki and M. Pollefeys. Patch based confidence prediction for dense disparity map. In *British Machine Vision Conf. (BMVC)*, 2016.

[61] L. Sevilla-Lara, D. Sun, V. Jampani, and M. J. Black. Optical flow with semantic segmentation and localized layers. *arXiv preprint arXiv:1603.03911*, 2016.

[62] M. R. Smith, L. Mitchell, C. Giraud-Carrier, and T. Martinez. Recommending learning algorithms and their associated hyperparameters. In *Proceedings of the International Conference on Meta-learning and Algorithm Selection*, pages 39–40. CEUR-WS. org, 2014.

[63] K. A. Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 4118–4124. IEEE, 2008.

[64] K. A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

[65] J. Sun, N.-N. Zheng, and H.-Y. Shum. Stereo matching using belief propagation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(7):787–800, 2003.

[66] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-V4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[67] T. Taniai, Y. Matsushita, and T. Naemura. Graph cut based continuous stereo matching using locally shared labels. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1613–1620, 2014.

[68] I. V. Tetko, T. I. Aksenova, V. V. Volkovich, T. N. Kasheva, D. V. Filipov, W. J. Welsh, D. J. Livingstone, and A. E. Villa. Polynomial neural network for linear and non-linear model selection in quantitative-structure activity relationship studies on the internet. *SAR and QSAR in Environmental Research*, 11(3-4):263–280, 2000.

[69] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. In *Proc. 18th Int'l Conf. on Multimedia*, pages 1469–1472. ACM, 2010.

[70] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[71] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[72] L. Xu, J. Jia, and Y. Matsushita. Motion detail preserving optical flow estimation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 34(9):1744–1757, 2012.

[73] S. Xu, F. Zhang, X. He, X. Shen, and X. Zhang. PM-PM: patchmatch with Potts model for object segmentation and stereo matching. *IEEE Trans. on Image Processing*, 24(7):2182–2196, 2015.

[74] J. Yang and H. Li. Dense, accurate optical flow estimation with piecewise parametric model. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1019–1027, 2015.

[75] Q. Yang. A non-local cost aggregation method for stereo matching. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1402–1409. IEEE, 2012.

[76] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1592–1599, 2015.

[77] J. Zbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. *J. of Machine Learning Research*, 17:1–32, 2016.

[78] M. Zeleny and J. L. Cochrane. *Multiple Criteria Decision Making*. Univ. of South Carolina Press, 1973.

[79] G. Zhang, J. Jia, T.-T. Wong, and H. Bao. Consistent depth maps recovery from a video sequence. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 31(6):974–988, 2009.

[80] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *arXiv preprint arXiv:1608.03981*, 2016.

[81] Y. Zhao, Z. Chen, C. Zhu, Y.-P. Tan, and L. Yu. Binocular just-noticeable-difference model for stereoscopic images. *IEEE Signal Processing Letters*, 18(1):19–22, 2011.