LEARNING TO ADAPT FROM FEW EXAMPLES

Eunbyung Park

A thesis submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2019

Approved by:

Alexander C. Berg

Tamara Berg

Jan-Michael Frahm

Marc Niethammer

Jimei Yang

ABSTRACT

Eunbyung Park: Learning to Adapt from Few Examples
(Under the direction of Alexander C. Berg)

Despite huge progress in artificial intelligence, the ability to quickly learn from few examples is still far short of that of a human. With the goal of building machines with this capability, learning-to-learn or meta-learning has begun to emerge with promising results. I present the effectiveness and techniques that improve existing meta-learning methods in the context of visual object tracking, few-shot classification, and few-shot reinforcement learning setup.

The visual object trackers that use online adaptation are improved. The core contribution is an offline meta-learning-based method to adjust the initial deep networks used in online adaptation-based tracking. The meta learning is driven by the goal of deep networks that can quickly be adapted to robustly model a particular target in future frames. Ideally the resulting models focus on features that are useful for future frames, and avoid overfitting to background clutter, small parts of the target, or noise. Experimental results on standard benchmarks, OTB2015 and VOT2016, show that the meta-learned trackers improve speed, accuracy, and robustness.

It is observed that learning curvature information can achieve better generalization and fast model adaptation. Based on the model-agnostic meta-learner (MAML), learning to transform the gradients in the inner optimization such that the transformed gradients achieve better generalization performance to a new task. For training large scale neural networks, the decomposition of the curvature matrix into smaller matrices are proposed and this capture the dependencies of the model's parameters with a series of tensor products. Experimental results show significant improvements on classification tasks and promising results on reinforcement learning tasks. Finally, an analysis that explains better generalization performance with the meta-trained curvature is presented.

Generative models given few examples are explored in the context of novel 3D view synthesis—given a single view of an object in an arbitrary pose, the goal is to synthesize an image of the object after a specified transformation of viewpoint. Instead of taking a 'blank slate' approach, information presented in an input image is used. First, the parts of the geometry visible both in the input and novel views are explicitly inferred, and then the remaining synthesis problem becomes image completion task. In addition to the new network structure, training with a combination of adversarial and perceptual loss results in a reduction in common artifacts of novel view synthesis such as distortions and holes, while successfully generating high frequency details and preserving visual aspects of the input image. Both qualitative and quantitative results show the proposed method achieves significantly better results compared to existing methods.

"This is dedicated to my wife, Eunha Kim who sacrificed to support and encourage me to pursue and finish this dissertation"

ACKNOWLEDGEMENTS

First and foremost I want to thank my advisor Alex Berg. It has been an great opportunity to be his Ph.D. student. Although we had some disagreements from time to time, I could learn a lot of lessons during my phd studies. Without his guidance, it is hard to imagine for me to be what I am right now. Now I am flying out of his arms, I do really hope we can be a close friend and a collaborator to support each other for the rest of our lives. Thank you Alex.

I would also like to extend my gratitude to wonderful committee members: Tamara Berg, Jan-Michael Frahm, Marc Niethammer, Jimei Yang. Their fruitful comments and suppors helped me a lot to refine the thesis. For Tamara who always has brilliant ideas, for Marc who gave me the opportunity to be TA for your courses and collaborations, for Jan who was always present at computer vision lab and provided us great research environment, for Jimei who gave me the promising research direction and served my committee, is so appreciated. Thank you all.

I would like to thank my collaborators and mentors during my phd. I had a great pleasure working with Junier Olivar. We closely worked together in my later work about meta-curvature. It has been great discussions and ended up having great results and findings. Whenever I have some questions and topics to discuss, his door has been always open to me.

I like to thank Ali Eslami who was the mentor of my internship at DeepMind. I remember the 5 months there was the period I learned the most and it was amazing experience to work with brilliant minds. I remember the moment when I fixed the frustrating and stinky bug of our codebase, he put a little present on my desk to celebrate. It was touching moment and I felt we are not just co-workers but also real friends. He also continuously introduced many researchers so that I can ask and discuss, which really helped us to make great progress. I sincerely thank his attitude and genuine investment on the internship project.

I want to thank Matthew Hausknecht who was the mentor at Microsoft Research during my internship. We worked together to set up a new project on the topics that are new to both of us, we had endless discussions and I really appreciate your effors to support my internship project.

In addition, I must thank to Jimei Yang, Ersin Yumer, Duygu Ceylan, who were my incredible mentors at Adobe Research. If I have to pick the best weekly meeting during my phd, I would definitely pick the one we had together. We easily went over the reserved time all the time, and we had so much fruitful ideas out of the meeting. Thanks for giving me the opportunity to know about the joy of discussion.

Special thanks to my computer vision lab colleagues who were always there, providing me with helpful discussions, proofreading, and the pleasant learning environment: Phil Ammirato, Akash Bapat, Sangwoo Cho, Marc Eder, Cheng-Yang Fu, Xufeng Han, Jared Heinly, Dinghuang Ji, Hadi Kiapour, Hyo Jin Kim, John Lim, Wei Liu, Jisan Mahud, Vicente Ordonez, Patrick Poison, True Price, Johannes Schonberger, Mykhailo Shvets, Sirion Vittayakorn, Thanh Vu, Ke Wang, Zhen Wei, Yi Xu, Licheng Yu, Songxu Zhao, Enliang Zheng, Yipin Zhou. I could not imagine my life at UNC without you. I do really appreciate all of your supports and encouragements.

As an international student, there were some moments that was tough to endure. Whenever I had those issues, I always rest on my Korean friends: Jaehyun Han, Junpyo Hong, Hyounghun Kim, Hyo Jin Kim, Youngjoong Kwon, Seulki Lee, Ilwoo Lyu, Jae Sung Park, Chonhyon Park. Thanks all, it would not be possibel without your support and encouragement.

Finally, to my family, I do not know how I can express my thankfulness to all of you. For my parents who raised me with a love and support, for my brother who has been always my biggest motivation, and for my loving, supportive, and patient wife Eunha whose faithful support during long academic journey. I sincerely appreciate. I owe all of you and I will live the rest of my life to pay back to you. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: AUTOMATIC MACHINE LEARNING (META-LEARNING)

Deep learning has played a key role to make remarkable progress on artificial intelligence (AI) over the last decade. It has been successfully used on a variety of tasks in AI, e.g. image recognition (116) and generation (41), speech recognition and generation (140) , machine translation, playing challenging games (92; 126), and so on. There have been many enabling factors behind this incredible success, such as large amount of data (116), powerful computations, some of algorithmic breakthroughs (48; 55), and open sourcing efforts from both academia and industries (3; 104).

Thanks to this collective efforts toward building AI systems, we hear a myriads of successful stories. However, developing and applying current deep learning techniques to many real world problems is still challenging task itself and in most cases it requires the knowledge from human experts on this domain. Furthermore, human experts very often depends on the trial-and-error approach, which is time consuming and tedious repetitive process. Automatic machine learning (or meta-learning) has emerged from this motivation.

A typical machine learning practice is follows. First, you collect a large scale dataset for your task. Based on your dataset and task, you define or search network architectures, objective functions, optimization algorithms, hyperparameters for optimization algorithms and objective functions, and so on. Once you determine all training configurations, you initialize the parameters of the networks with well known initialization methods. Then, you train the networks until convergence on training dataset (cross-validation is popular scheme to find best hyperparameters or training configurations). After training convergence, we observe the performance on validation dataset, and we adjust the configurations and train the networks again. We keep repeating this process up until our validation performance are saturated. This process, one training episode, re-

quires often more than a day or week and there are too many factors that would potentially affect the eventual performance.

The most distinctive feature of meta-learning is that it aims to achieve better generalization in a principled way. Instead of resting on empirical intuition to adjust training configurations through cross-validation, we define a meta-objective function, which maximizes the generalization performance, to find the best training configurations. As opposed to optimization methods that usually aim to reach to the best training loss value, meta-learning's main objective is the model's generalization ability and less focus on training loss.

In this chapter, we introduce recent progress on this domain and provide high-level overview. We also provide in-depth introduction of gradient-based meta-learning, which will be used for later chapters.

1.1 Meta-learning

1.2 Categorization

This is an emerging field in machine learning community and many recent works have slightly different purpose and training setups. In this section, we categorize them into few representative meta-training algorithms and training setups.

1.2.1 Training algorithm

1.2.1.1 Bayesian optimization

Bayesian optimization has been one of the most powerful techniques algorithms for finding the best hyperparameters in deep network training. Since training deep networks has been known to be a blackbox system, it is not straightforward to extract explicit knowledge from the training process. One notable example (128) employs gaussian processes to model the target function for estimating the uncertainty and determine which point to evaluate next by considering trad-

ing off exploration and exploitation. It has been very effective, especially for low dimensional hyperparameters, e.g. learning rate and this obtained new state-of-the-art results in tuning hyperparameters for image classification tasks (58). However, it is still early stage to apply for high-dimensional hyperparameters, e.g. initialization of the network parameters. For in-depth introduction to this, we refer to the tutorial (125).

1.2.1.2 Reinforcement learning

To scale up high-dimensional hyperparameters, the reinforcement learning has been successfully used to find a network architecture (158; 159). We can optimize a network architecture that maximize validation performance and we can treat validation losses as negative rewards and we typically use policy gradients to optimize the meta-network that generates the network architecture. It also has an advantage over gradient-based methods. It does not have to unroll the large number of training steps in gradient descent steps. Thanks to recent advances in deep reinforcement learning, it obtained new state-of-the-art results on variety of tasks with the architecture founded and it gave us interesting aspects of designing network architecture.

1.2.1.3 Evolutionary method

Evolution Strategies (ES) is a set of optimization methods that can reliably use for black-box optimization. Recently, ES has been successfully adopted to train deep neural networks (42). In a nutshell, it resembles to bayesian black-box optimization. It starts with some random parameters and keeps repeating to adjust the parameters. We observe the final performance with tweaked parameters and gives higher probability to those that gave better performance. The most useful property of ES is that we can massively parallelize this process up to even thousands of populations if computations permitted. It has been adopted to find hyperparameters of the training setups, e.g. learning rate, entropy-cost coefficient in reinforcement learning (87). Recently, the network architectures produced by evolution algorithms have reached the accuracy of those directly designed by human experts or founded by reinforcement learning (30).

1.2.1.4 Gradient-based methods

Thanks to recent advances in automatic differentiation techniques, it has been popular choice to obtain gradient through unrolling gradient descent steps. Although it has potential issues such as short-horizon bias (153) and computational complexity for large unrolling steps (84), it has been successfully used in hyperparameter search (84) and meta-training initialization of the networks for few-shot classification and reinforcement learning setup (31). Throughout this dissertation, we explore and discuss the effectiveness of gradient-based methods focusing on learning the initialization of the parameters and how to adapt the gradients for better generalization and faster convergence.

1.2.2 Training setup

In meta-training, we collect few training episodes, a.k.a mini-batches in usual training setup, and construct the batches to optimize for meta-objective functions. We can differentiate the existing meta-learning works into two different scenarios. First, we can provide different tasks for every training episodes. In this case, we are optimizing training configurations for the tasks from a task distribution. We hope the resulting configurations or hyperparameters will generalize well for a new task, which can be quite different than the tasks in the meta-training.

For example, (**?** 101) have suggested to learn a optimizer instead of resting on hand-crafted optimizers such as SGD, ADAM, RMSProp, to name a few. Every training episodes, it provides a different task, e.g. different categories, different number of classes, different network architectures, and so on. They aim to learn a optimizer that can cover a variety of training setups so that resulting meta-trained optimizer can be applied to a new task or network architecture. Few-shot learning meta-training also falls into this category, (31; 109). They sample a task from a task distribution for every training episodes. For example, we have a large meta-dataset consisting of 100 categories and 600 images per category and we sample a 5-way 5-shot classification task by randomly sampling 5 categories and 5 images per category. Therefore, every training episode, we see different categories and images, which can be interpreted as different tasks, but from a task

distribution. The resulting meta-trained model can be effectively used for similar tasks but unseen categories and images.

Second, there has been many works that optimize the training configurations for a particular task. In this setup, we provide same task for every training episodes, which aims to learn an optimal training configuration that optimize generalization performance for the task. For example, hyperparameter optimization (128; 84) can fall into this category. Every training episodes, they look at the validation loss, it determines which hyperparameters and how to adjust values based on uncertainty in the hyperparameter space through gaussian process (128) or gradients from the validation loss (84).

1.3 Model-agnostic meta-learning (MAML)

MAML aims to find a transferable initialization (a prior representation) of any model such that the model can adapt quickly from the initialization and produce good generalization performance on new tasks. It is general and model-agnostic, which it can be directly applied to any learning problem and model that is trained with a gradient descent algorithm. The meta-objective is defined as validation performance after one or few step gradient updates from the model's initial parameters. By using gradient descent algorithms to optimize the meta-objective, its training algorithm usually takes the form of nested gradient updates: inner updates for model adaptation to a task and outer-updates for the model's initialization parameters. More formally,

$$\min_{\theta} \mathbb{E}_{\tau_i}[\mathcal{L}_{\text{val}}^{\tau_i}\big(\underbrace{\theta - \alpha \nabla \mathcal{L}_{\text{tr}}^{\tau_i}(\theta)}_{\text{inner udpate}}\big)], \tag{1.1}$$

where $\mathcal{L}_{\text{val}}^{\tau_i}(\cdot)$ denotes a loss function for a validation set of a task τ_i, and $\mathcal{L}_{\text{tr}}^{\tau_i}(\cdot)$ for a training set, or $\mathcal{L}_{\text{tr}}(\cdot)$ for brevity. The inner update is defined as a standard gradient descent with fixed learning rate α. For conciseness, we assume as single adaptation step, but it can be easily extended to more steps.

Starting from randomly initialized model parameters θ_0, the model parameters after after one gradient descent step can be written as

$$\theta_1 = \theta_0 - \alpha \nabla_\theta L_{\text{tr}}(\theta_0) \tag{1.2}$$

In order to optimize meta-objective function that minimize validation loss, we can take the gradient of validation loss w.r.t the initial parameters θ_0. Once we have this gradient (also called as meta-gradient), we update the initial model's parameters θ_0 with gradient descent algorithms. Therefore, simple update rules for meta-training model's initial parameters with a single gradient steps in inner update can be written as follows.

$$\theta_0 = \theta_0 - \beta \nabla_{\theta_0} L_{\text{val}}(\theta_1) \tag{1.3}$$

$$= \theta_0 - \beta \nabla_{\theta_0} L_{\text{val}}(\theta_0 - \alpha \nabla_\theta L_{\text{tr}}(\theta_0)) \tag{1.4}$$

$$= \theta_0 - \beta \nabla_\theta L_{\text{val}}(\theta_1)(\mathbf{I} - \alpha \nabla_\theta^2 L_{\text{tr}}(\theta_0)) \tag{1.5}$$

where, β and α are the learning rates for outer loop and inner loop respectively. This meta-gradients requires the gradients of the gradient, which has higher order gradient terms, e.g. second order gradients in this example. It is not scalable to compute this gradients manually. In addition, computing higher order gradients are often infeasible especially for large deep neural networks that have millions of model parameters. However, we do not have to explicitly compute or maintain the second order gradients here because eventually we will be computing matrix-vector product $\nabla_\theta L_{\text{val}}(\theta_1) \nabla_\theta^2 L_{\text{tr}}(\theta_0)$, and we can use trick from (106) that allows us to easily compute this by using automatic differentiation software packages (3; 104).

Several variations of inner update rules were suggested. Meta-SGD (78) suggested coordinate-wise learning rates, $\theta - \alpha \circ \nabla \mathcal{L}_{\text{tr}}$, where α is the learnable parameters and $\circ$ is element wise product. Recently, (8) proposed a learnable learning rate per each layers for more flexible model adapta-

tion. To alleviate computational complexity, (98) suggested an algorithm that do not require higher order gradients.

CHAPTER 2: BACKGROUND

In this chapter, we present some of background materials that are useful to formalize and understand the proposed methods in the chapter 4. First, we review the tensor algebra to formalize the proposed method. Tensor algebra is a great mathematical tool to translate complicated tensor operations into concise notation. In addition, there are many properties that we can easily use to understand the proposed methods. Secondly, we introduce some second-order optimization algorithms that are commonly used in training deep neural networks. This will be useful to understand the relationship between the proposed method and existing optimization algorithms.

2.1 Tensor Algebra

We review basics of tensor algebra that will be used to formalize the proposed method. We refer the reader to (67) for a more comprehensive review. Throughout the paper, tensors are defined as multidimensional arrays and denoted by calligraphic letters, e.g. Nth-order tensor, $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. Matrices are second-order tensors and denoted by boldface uppercase, e.g. $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$.

2.1.1 Fibers

Fibers are a higher-order generalization of matrix rows and columns. A matrix column is a mode-1 fiber and a matrix row is a mode-2 fiber. For a third order tensor $\mathcal{X}$, the mode-1 fibers of $\mathcal{X}$ are denoted as $\mathcal{X}_{:,j,k}$, where $\mathcal{X}_{i,j,k}$ to denote (i, j, k) elements and a colon is used to denote all elements of a mode.

2.1.2 Tensor unfolding

Also known as *flattening (reshaping)* or *matricization*, is the operation of arranging the elements of an higher-order tensors into a matrix. The mode-n unfolding of a Nth-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, arranges the mode-n fibers to be the columns of the matrix, denoted by $\mathcal{X}_{[n]} \in \mathbb{R}^{I_n \times I_M}$, where $I_M = \prod_{k \neq n} I_k$. The elements of the tensor, $\mathcal{X}_{i_1, i_2, \ldots, i_N}$ are mapped to $\mathcal{X}_{[n] i_n, j}$, where $j = 1 + \sum_{k \neq n, k=1}^{N} (i_k - 1) J_k$, with $J_k = \prod_{m=1, m \neq n}^{k-1} I_m$.

2.1.3 *n*-mode product

It defines the product between tensors and matrices. The n-mode product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{M} \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{M}$ and computed as

$$(\mathcal{X} \times_n \mathbf{M})_{i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N} = \sum_{i_n=1}^{I_n} \mathcal{X}_{i_1, i_2, \ldots, i_N} \mathbf{M}_{j, i_n}. \tag{2.1}$$

More concisely, it can be written as $(\mathcal{X} \times_n \mathbf{M})_{[n]} = \mathbf{M} \mathcal{X}_{[n]} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$. Despite cumbersome notation, it is simply n-mode unfolding (reshaping) followed by matrix multiplication.

2.1.4 Tucker decomposition

We can decompose a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a low rank core $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ through projection along each of its modes by projection factors $(\mathbf{U}^{(0)}, \cdots, \mathbf{U}^{(N)})$, with $\mathbf{U}^{(k)} \in \mathbb{R}^{R_k \times I_k}, k \in (0, \cdots, N)$. Formally, it can be written as

$$\mathcal{X} = \mathcal{U} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_N \mathbf{U}^{(N)}. \tag{2.2}$$

Typically, this decomposition is obtained by solving a least squares problem. For more details, we refer to (67).

2.2 Gradient descent methods

We also review of basics of gradient descent methods. We refer the reader to (99) for a more comprehensive review. Gradient descent methods, more specifically stochastic variations, have dominated for training large scale deep neural networks. With reverse-mode automatic differentiation technique, a.k.a back-propagation algorithm, we can easily compute the gradients and apply simple mini-batch stochastic gradient descent algorithm. It turns out stochastic variations have a great generalization performance compared to batch gradient descent (98).

2.2.1 First order methods

The most notable example of first order methods for deep learning is the stochastic gradient descent with momentum. First order methods cannot capture the higher-order dependencies between the parameters (by definition), and it focuses on the history of gradients for individual components. With momentum, it computes the gradients based on exponential average over certain amount of history for each elements of gradients. Therefore, some local gradient with specific mini-batch gave a bad direction, it has a chance to fix this based on the history of the gradients. ADAM (66) combines second order information to accelerate the convergence speed. It maintains the history of inverse of square of gradients for every elements, which then will be used to multiply the current gradients. We can interpret this as a diagonal approximation of online natural gradient descent algorithm, which will be discussed in the next section.

2.2.2 Second order methods

The biggest motivation of second order methods is that first-order optimization such as standard gradient descent performs poorly if the Hessian of a loss function is ill-conditioned, e.g. a long narrow valley loss surface. There are a plethora of works that try to accelerate gradient descent by considering local curvatures. Typical mechanic of second order methods is to approximate loss function at θ_t with surrogate quadratic function, and minimize this surrogate function

at θ_t instead of original loss function. We provide two representative algorithms, Newton method and natural gradient method.

2.2.2.1 Newton method

We perform second-order taylor approximation of loss function at θ_t.

$$L(\theta_t + d) \approx L(\theta_t) + \nabla L(\theta_t)^\top d + \frac{1}{2} d^\top \nabla_\theta^2 L(\theta_t) d, \tag{2.3}$$

where d is the direction we take to minimize the loss. Then, we minimize this surrogate quadratic function w.r.t d instead of original loss function. We take the gradient w.r.t d and set it to zero, we obtain analytic solution

$$-\nabla_\theta^2 L(\theta_t)^{-1} \nabla_\theta L(\theta_t) = \operatorname*{argmin}_d L(\theta_t) + \nabla L(\theta_t)^\top d + \frac{1}{2} d^\top. \tag{2.4}$$

And we can recover well-known newton update rule

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta^2 L(\theta_t)^{-1} \nabla_\theta L(\theta_t). \tag{2.5}$$

2.2.2.2 Natural gradient method

Natural gradient method was invented in (6) and it has been widely adopted in training deep neural networks. It finds a steepest descent direction in distribution space rather than parameter space by measuring KL-divergence as a distance metric. For example, if we define the loss function as negative log likelihood, e.g. a supervised classification task $\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\sim p(x,y)}[-\log p(y|x;\theta)]$, then the empirical Fisher information matrix can be defined as $\mathbf{F} = \mathbb{E}_{(x,y)\sim p(x,y)}[\nabla_\theta \log p(y|x;\theta) \nabla_\theta \log p(y|x;\theta)^\top]$. Let us consider a constrained optimization problem

$$\operatorname*{argmin}_d L(\theta_t + d) \quad \text{s.t. } \mathrm{KL}(p_{\theta_t} || p_{\theta_t + d}) = c. \tag{2.6}$$

In this optimization problem, we seek to find a direction d such that it minimize the loss value after an update with the constraint preventing from you deviating too much from the current iteration. It is encoded by setting a maximum KL divergence of a certain constant c. We can form a Lagrangian to convert into unconstrained optimization problem and perform local approximation.

$$\underset{d}{\text{argmin}}\ L(\theta_t) + \nabla_\theta L(\theta_t)^\top d + \lambda(\text{KL}(p_{\theta_t}||p_{\theta_t+d}) - c) \tag{2.7}$$

$$\approx \underset{d}{\text{argmin}}\ L(\theta_t) + \nabla_\theta L(\theta_t)^\top d + \lambda(\frac{1}{2}d^\top \mathbf{F} d - c). \tag{2.8}$$

Equation 2.8 comes from the fact that we perform second order approximation of KL divergence (Note that the Hessian of KL divergence between two different distribution is Fisher information). Similar to the derivation of Newton method, if we take the gradient w.r.t. d and set it to zero, we can obtain the solution of this surrogate function and we reach to natural gradient update rule

$$\theta_{t+1} = \theta_t - \alpha \mathbf{F}^{-1} \nabla \mathcal{L}(\theta_t). \tag{2.9}$$

2.2.2.3 Kronecker-factored approximate curvature

In large scale deep learning, it is not uncommon to have millions of parameters. Therefore, it is not feasible to compute inverse of gigantic matrix and even store the second order matrix. There have been many approximation methods to reduce these costs, the notable example is ADAM optimizer (66) who performs an online diagonal approximation of Fisher matrix. However, it is limited in terms of dependency modeling between the parameters and many existing works, e.g. block-diagonal or tri-diagonal approximations, have not successfully achieved better convergence than first order methods. Recently, (85; 45) proposed Kronecker-factored approximate curvature (K-FAC), which approximates the Fisher matrix by the Kronecker product, e.g. $\mathbf{F} \approx \mathbf{A} \otimes \mathbf{G}$, where $\mathbf{A}$ is computed from the activation of input units and $\mathbf{G}$ is computed from the gradient of output units. By using Kronecker product's properties, it can efficiently invert the approximated the Fisher information matrix.

CHAPTER 3: META-TRACKER: FAST AND ROBUST ONLINE ADAPTATION FOR VISUAL OBJECT TRACKERS

In this chapter, we improve the state-of-the-art visual object trackers with meta-learning strategies. Visual object tracking is a task that locates target objects precisely over a sequence of image frames given a target bounding box at the initial frame. In contrast to other object recognition tasks, such as object category classification and detection, in visual object tracking, instance-level discrimination is an important factor. For example, a target of interest could be one particular person in a crowd, or a specific product (e.g. coke can) in a broader category (e.g. soda cans). Therefore, an accurate object tracker should be capable of not only recognizing generic objects from background clutter and other categories of objects, but also discriminating a particular target among similar distractors that may be of the same category. Furthermore, the model learned during tracking should be flexible to account for appearance variations of the target due to viewpoint change, occlusion, and deformation.

One approach to these challenges is applying online adaptation. The model of the target during tracking, e.g. DCF (discriminative correlation filter) or binary classifier (the object vs backgrounds), is initialized at the first frame of a sequence, and then updated to be adapted to target appearance in subsequent frames (96; 129; 22; 19; 50; 82; 62; 14). With the emergence of powerful generic deep-learning representations, recent top performing trackers now leverage the best of both worlds: deep learned features and online adaptation methods. Offline-only trackers trained with deep methods have also been suggested, with promising results and high speed, but with a decrease in accuracy compared to state-of-the-art online adaptive trackers (13; 49; 132), perhaps due to difficulty finely discriminating specific instances in videos.

A common practice to combine deep learning features and online adaptation is to train a target model on top of deeply learned features, pre-trained over a large-scale dataset. These pre-

trained features have proven to be a powerful and broad representation that can recognize many generic objects, enabling effective training of target models to focus on the specified target instance. Although this type of approach has shown the best results so far, there remain several important issues to be resolved.

First, very few training examples are available. We are given a single bounding box for the target in the initial frame. In subsequent frames, trackers collect additional images, but many are redundant since they are essentially the same target and background. Furthermore, recent trends towards building deep models for target appearance (96; 129) make the problem more challenging since deep models are known to be vulnerable to overfitting on small datasets. As a consequence, a target model trained on top of deeply learned features sometimes suffers because it overfits to background clutter, small parts or features of the target, or noise. Many recent studies have proposed various methods to resolve these issues. Some include using a large number of positive and negative samples with aggressive regularizers (96), factorized convolution (19), spatio-residual modules (129), or incorporating contextual information (94).

Second, most state-of-the-art trackers spend a significant amount of time on the initial training stage (96; 129; 19). Although many works have proposed fast training methods (19; 50), this still remains a bottleneck. In many practical applications of object tracking, such as surveillance, real-time processing is required. Depending on the application, falling behind on the initial frame could mean failure on the whole task. On the other hand, an incompletely trained initial target model could affect performance on future frames, or in the worst case, result in failures on all subsequent frames. Therefore, it is highly desirable to obtain the robust target model very quickly at the initial frame.

We propose a generic and principled way of tackling these challenges. Inspired by recent meta-learning (learning to learn) studies (31; 109; 7; 119; 78; 4), we seek to learn how to obtain the target model. The key idea is to train the target model in a way that generalizes well over future frames. In all previous works (96; 129; 22; 19; 50; 82; 62; 14), the target model is trained to minimize a loss function on the current frame. Even if the model reaches an optimal solution,

it does not necessarily mean it would work well for future frames. Instead, we suggest to use error signals from future frames. During the meta-training phase, we aim to find a generic initial representation and gradient directions that enable the target model to focus on features that are useful for future frames. Also, this meta-training phase helps to avoid overfitting to distractors in the current frame. In addition, by enforcing the number of update iterations during meta-training, the resulting networks train significantly faster during the initialization.

Our proposed approach can be applied to any learning based tracker with minor modifications. We select two *state-of-the-art trackers*, MDNet (96), from the classifier based tracker (tracking-by-detection) category, and CREST (129), a correlation based tracker. Experimental results show that our meta-learned version of these trackers can adapt very quickly—just one iteration—for the first frame while improving accuracy and robustness. Note that this is done even without employing some of the hand engineered training techniques, sophisticated architectural design, and hyperparameter choices of the original trackers. In short, we present an easy way to make very good trackers even better without too much effort, and demonstrate its success on two different tracking architectures, indicating potentially general applicability.

3.1 Meta-Learning for Visual Object Trackers

In this section, we explain the proposed generalizable meta-training framework for visual object trackers. The details for applying this to each tracker are found in Section 3.2.

3.1.1 Motivation

A typical tracking episode is as follows: The tracking model is adapted to a specified bounding box around the target in the initial frame of a sequence. Aggressive regularizers and fast optimization techniques are adopted to allow this adaptation/training to be done quickly so that the resulting model is robust to target variations and environment changes. Then, the tracking model is used to predict the target location in subsequent frames. Predicted target locations and

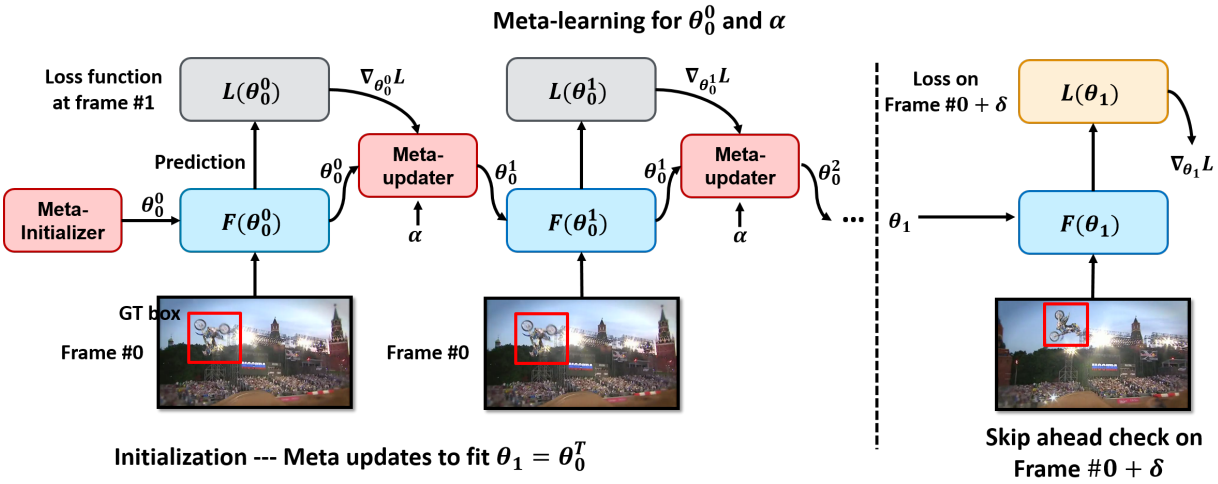


Figure 3.1: Our meta-training approach for visual object tracking: A computational graph for meta-training object trackers. For each iteration, it gets the gradient with respect to the loss after the first frame, and a meta-updater updates parameters of the tracker using those gradients. For added stability and robustness a final loss is computed using a future frame to compute the gradients w.r.t parameters of meta-initializer and meta-updater. More details in Section 3.1.

images are then stored in the database, and the models are regularly updated with collected data according to their own strategies.

A key motivation is to incorporate these actual tracking scenarios into the meta-learning process. The eventual goal of trackers is to predict the target locations in future frames. Thus, it would be desirable to learn trackers with this eventual goal. For example, if we could look at variations in future frames, then we could build more robust target models and prevent them from overfitting to the current target appearance or background clutter. We can take a step back and observe trackers running on videos, see if the trackers generalize well, and find a reason why they become distracted and adjust the adaptation procedure accordingly.

3.1.2 A general online tracker

This formulation of online tracking is made general in order to apply to a variety of trackers. Consider the key operation in a tracker, $\hat{y} = F(x, \theta)$, that takes an input x, e.g. image patches around the target or a cropped image centered on putative target from an image I, and

Algorithm 1 Meta-training object trackers algorithm

Input : Randomly initialized θ_0 and α, training dataset D
Output : θ_0^* and α^*

while not converged **do**
  $\text{grad}_{\theta_0}, \text{grad}_\alpha = \vec{0}$ ▷ Initialize to zero vector
  for all $k \in \{0, \ldots, N_{\text{mini}} - 1\}$ **do**
    $S, j, \delta \sim p(D)$ ▷ Sample a training example
    $\theta_0^0 = \theta_0$
    for all $t \in \{0, \ldots, T-1\}$ **do**
      $\hat{y}_j = F(x_j, \theta_0^t)$
      $\theta_0^{t+1} = \theta_0^t - \alpha \odot \nabla_{\theta_0^t} L(y_j, \hat{y}_j; \theta_0^t)$
    $\theta_1 = \theta_0^T$
    $\hat{y}_{j+\delta} = F(x_{j+\delta}, \theta_1)$ ▷ Apply to a future frame
    $\text{grad}_{\theta_0} = \text{grad}_{\theta_0} + \nabla_{\theta_0} L(y_{j+\delta}, \hat{y}_{j+\delta})$ ▷ Accumulate the gradients
    $\text{grad}_\alpha = \text{grad}_\alpha + \nabla_\alpha L(y_{j+\delta}, \hat{y}_{j+\delta})$
  $\theta_0 = \text{Optimizer}(\theta_0, \text{grad}_{\theta_0})$ ▷ Update θ_0
  $\alpha = \text{Optimizer}(\alpha, \text{grad}_\alpha)$ ▷ Update α

the tracker parameters θ and produces an estimate $\hat{y}$ of the label, e.g. a response map or a location in the frame that indicates the target position. For *initialization*, x_0 from the initial frame I_0 with specified y_0, we (approximately) solve for $\theta_1(x_0, y_0)$, or θ_1 for brevity, with respect to a loss, $L\left(F\left(x_0, \theta_1\right), y_0\right)$, measuring how well the model predicts the specified label. For *updates* during tracking, we take the parameters θ_j from frame $j-1$ and find $\hat{y}_j = F(x_j, \theta_j)$, then find θ_{j+1} with respect to a loss. Then, we may incorporate transforming $\hat{y}_j$ into a specific estimate of the target location as well as temporal smoothing, etc. We can write the tracking process initialized with x_0 and y_0 in an initial frame and then proceeding to track and update for frames $I_1 \ldots I_n$ as $\text{Track}\left(\theta_1(x_0, y_0), I_1, \ldots, I_n\right)$ and its output as $\hat{y}_n$, an estimate of the label in the nth frame (indicating target position) and θ_{n+1}, the model parameters after the nth frame.

3.1.3 Meta-training algorithm

Our meta-training approach has two goals. One is that initialization for a tracker on a sequence can be performed by starting with θ_0 and applying one or a very small number of itera-

tions of a update function M parameterized by α. Another goal is that the resulting tracker be accurate and robust on later frames.

The gradient-descent style update function M is parameterized by α:

$$M(\theta, \nabla_\theta L; \alpha) = \theta - \alpha \odot \nabla_\theta L \,, \tag{3.1}$$

where α is the same size as the tracker parameters θ (78), L is a loss function, and $\odot$ is element-wise product. α could be a scalar value, which might be either learnable (4) or manually fixed (31). We empirically found that having per parameter coefficients was the most effective in our settings.

Our meta-training algorithm is to find a good θ_0 and α by repeatedly sampling a video, performing initialization, applying the learned initial model to a frame slightly ahead in the sequence, and then back-propagating to update θ_0 and α. Applying the initial model to a frame slightly ahead in the sequence has two goals, the model should be robust enough to handle more than frame-to-frame variation, and if so, this should make updates during tracking fast as well if not much needs to be fixed.

After sampling a random starting frame from a random video, we perform optimization for initialization starting with $\theta_0^0 = \theta_0$ given the transformed input and output pair, (x_j, y_j). A step of optimization proceeds as

$$\theta_0^{i+1} = M(\theta_0^i, \nabla_{\theta_0^i} L(y_j, F(x_j, \theta_0^i)))\,. \tag{3.2}$$

This step can be repeated up to a predefined number of times T to find, $\theta_1(x_j, y_j) = \theta_0^T$. Then, we randomly sample a future frame $I_{j+\delta}$ and evaluate the model trained on the initial frame on that future frame to produce: $\hat{y}_{j+\delta} = F(x_{j+\delta}, \theta_1)$.

The larger δ, the larger target object variations and environment changes are incorporated into training process. Now, we can compute the loss based on the future frame and trained tracker

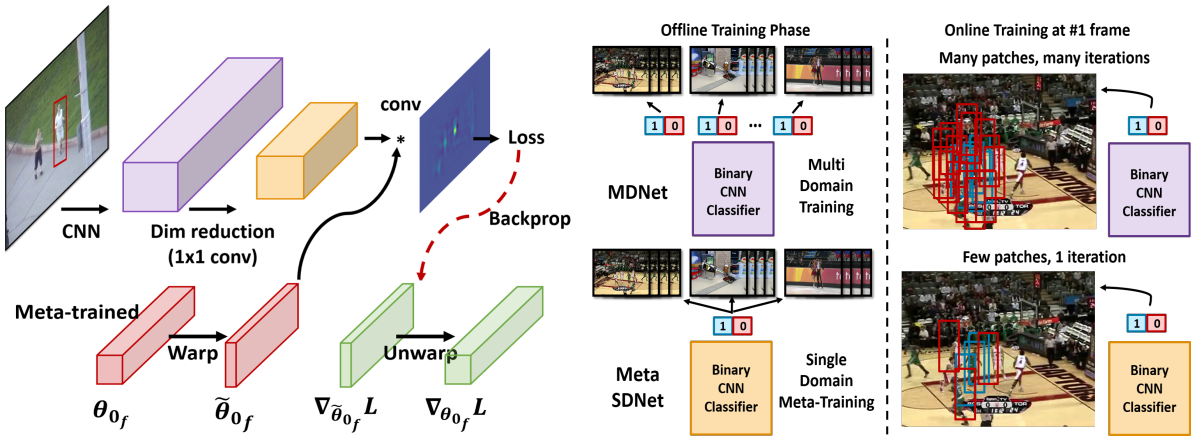
parameters. The objective function is defined as

$$\theta_0^*, \alpha^* = \underset{\theta_0,\alpha}{\operatorname{argmin}}\, \mathbb{E}_{S,j,\delta}[L(y_{j+\delta}, \hat{y}_{j+\delta})]\,. \tag{3.3}$$

We used the ADAM (66) gradient descent algorithm to optimize. Note that θ_0 and α are fixed across different episodes in a mini-batch, but $\theta_0^1, \ldots, \theta_0^T$ are changed over every episode. To compute gradients of the objective function w.r.t θ_0 and α, it is required to compute higher-order gradients (the gradients of function of gradients). This type of computation has been exploited in recent studies (31; 84; 89). We can easily compute this thanks to automatic differentiation software libraries (pyt). More details are explained in Algorithm 1.

3.1.4 Update rules for subsequent frames.

Most online trackers, including the two trackers we meta-train (Section 3.2), update the target model regularly to adjust to new examples collected by itself during tracking. We could simply use meta-trained α to update the model, $\theta_j = \theta_{j-1} - \alpha \odot \nabla_{\theta_{j-1}} L$ (only one iteration presented for brevity). However, it often diverges on longer sequences or the sequences that have very small frame-to-frame variations. We believe this is mainly because we train α for fast adaptation at the initial frame, so the values of α are relatively large, which causes unstable convergence behavior (A similar phenomenon was reported in (4) albeit in a different context). Since α is stable when it teams up with θ_0, we could define the update rules for subsequent frames as $\theta_j = \theta_0 - \alpha \odot \nabla_{\theta_0} L$, as suggested in (4). We could also combine two strategies, $\theta_j = \beta(\theta_{j-1} - \alpha \odot \nabla_{\theta_{j-1}} L) + (1-\beta)(\theta_0 - \alpha \odot \nabla_{\theta_0} L)$. Although we could resolve unstable convergence behavior with these strategies, none of these performed better than simply searching for a single learning rate. Therefore, we find a learning rate for subsequent frames and then use existing optimization algorithms to update the models as was done in the original versions of the trackers.



(a) MetaCREST (b) MDNet vs MetaSDNet

3.2 Meta-Trackers

In this section, we show how our proposed meta-learning technique can be realized in state-of-the-art trackers. We selected two different types of trackers, one from correlation based trackers, CREST (129), and one from tracking-by-detection based trackers MDNet (96).

3.2.1 Meta-training of correlation based tracker

3.2.1.1 CREST

A typical correlation filter objective is defined as follows.

$$\underset{f}{\operatorname{argmin}} \, ||y - \Phi(x) * f||^2 + \lambda ||f||^2 \,, \tag{3.4}$$

where f is the correlation filter, $*$ is the convolution operation, and Φ is a feature extractor, e.g. CNN. x is a cropped image centered on the target, and $y \in \mathbb{R}^{H \times W}$, is a gaussian shaped response map, where H and W are height and width, respectively. The cropped image is usually larger than the target object so that it can provide enough background information. Once we have the correlation filter trained, target localization at a new future frame is simply finding the coordinates (h, w) that has the maximum response value.

$$\underset{(h,w)}{\operatorname{argmax}}\, \hat{y}(h,w)\,, \tag{3.5}$$

where $\hat{y} = \Phi(x_{\text{new}}) * f$, and $\hat{y}(h,w)$ represents the element of $\hat{y}$ in (h,w) coordinates. CREST used a variation of the correlation filter objective, defined as

$$\sum_{(h,w)\in P} \frac{1}{|P|}(e^{y(h,w)}|y(h,w) - \hat{y}(h,w)|)^2 + \lambda||f||^2\,, \tag{3.6}$$

where $P = \{(h,w) \mid |y(h,w) - \hat{y}(h,w)| > 0.1\}$. This would encourage the model to focus on parts that are far from the ground truth values.

By reformulating the correlation filter as a convolutional layer, it can be effectively integrated into an CNN framework (129). This allows us to add new modules easily, since the optimization can be nicely done with standard gradient descent in end-to-end fashion. They inserted spatio-temporal residual modules to avoid target model degradation by large appearance changes. They also devised sophisticated initialization, learning rates, and weight decay regularizers, e.g. 1000 times larger weight decay parameter on spatio-temporal residual modules. Without those bells and whistles, we aim to learn a robust single layer correlation filter via proposed meta-learning process. There are two important issues for plugging CREST tracker into proposed meta-training framework, and we present our solutions in following sections.

3.2.1.2 Meta-learning dimensionality reduction

CREST used PCA to reduce the number of channels of extracted CNN features, from 512 to 64. This not only reduces computational cost, but also it helps to increase robustness of the correlation filter. PCA is performed at the initial frame and learned projection matrix are used for the rest of the sequence. This becomes an issue when meta-training the correlation filter. We seek to find a global initialization of the correlation filter for the all targets from different episodes. However, PCA would change the basis for every sequences, which makes impossible to obtain a global initialization in projected feature spaces that are changing every time. We propose to

learn to reduce dimensions of the features. In CREST, we can insert 1x1 convolution layer right after the feature extraction, the weights of this layer are also meta-learnable and jointly trained during the meta-learning process along with the correlation filter. θ_0 in proposed meta-training framework, therefore, consists of θ_{0_d} and θ_{0_f}, the parameters of dimensionality reduction and the correlation filter, respectively.

3.2.1.3 Canonical size initialization

The size of the correlation filter varies depending on the target shape and size. In order to meta-train a fixed size initialization of the correlation filter θ_{0_f}, we should resize all objects to the same size and same aspect ratio. However, it introduces distortion of the target and has been known to degrade recognition performance (79; 112). In order to fully make use of the power of the correlation filter, we propose to use canonical size initialization and its size and aspect ratio are calculated as a mean of the objects in the training dataset. Based on canonical size initialization, we warp it to the specific size taylored to the target object for each tracking episodes, $\tilde{\theta}_{0_f} = \text{Warp}(\theta_{0_f})$. We used differentiable bilinear sampling method (57) to pass through gradients all the way down to θ_{0_f}.

Putting it all together, $F(x_j, \theta)$ in our proposed meta-training framework for CREST, now takes an input a cropped image x_j from an input frame I_j, pass it through a CNN feature extractor followed by dimensionality reduction (1x1 convolution with the weight θ_{0_d}). Then, it warps the correlation filter θ_{0_f}, and finally apply warped correlation filter $\tilde{\theta}_{0_f}$ to produce a response map $\hat{y}_j$ (Figure 3.2a).

3.2.2 Meta-training of tracking-by-detection tracker

3.2.2.1 MDNet

MDNet is based on a binary CNN classifier consisting of a few of convolutional layers and fully connected layers. In the offline phase, it uses a multi-domain training technique to pre-train

the classifier. At the initial frame, it randomly initializes the last fully connected layer, and trains around 30 iterations with a large number of positive and negative patches (Figure 3.2b). Target locations in the subsequent frames are determined by average of bounding box regression outputs of top scoring positive patches. It collects positive and negative samples during the tracking process, and regularly updates the classifier. Multi-domain pre-training was a key factor to achieve robustness, and they used an aggressive dropout regularizer and different learning rates at different layers to further avoid overfitting to current target appearance. Without those techniques (the multi-domain training and regularizers), we aim to obtain robust and quickly adaptive classifier solely resting on the proposed meta-learning process.

3.2.2.2 Meta-training

It can be also easily plugged into the proposed meta-leraning framework. $F(x_j;\theta)$ takes as input image patches $x_j \in \mathbb{R}^{N\times D}$ from an input frame I_j (and $y_j \in \{0,1\}^N$ is the corresponding labels), where D is the size of the patches and N is the number of patches. Then, the patches go through a CNN classifier, and the loss function L is a simple cross entropy loss $-\sum_{k=1}^{N} y_j^k \log(F^k(x_j;\theta))$.

3.2.2.3 Label shuffling

Although a large-scale video detection dataset contains rich variation of objects in videos, the number of objects and categories are limited compared to other still image datasets. This might lead a deep CNN classifier to memorize all object instances in the dataset and classify newly seen objects as backgrounds. In order to avoid this issue, we adopted the label shuffling trick, suggested in (119). Every time we run a tracking episode, we shuffle the labels, meaning sometimes labels of positive patches become 0 instead of 1, negative patches become 1 instead of 0. This trick encourages the classifier to learn how to distinguish the target objects from background by looking at current training examples, rather than memorizing specific targets appearance.

3.3 Related Work

3.3.1 Online trackers

Many online trackers use correlation filters as the back-bone of the algorithms due to computational efficiency and discriminative power. From the early success of the MOSSE tracker (14), a large number of variations have been suggested. (50) makes it more efficient by taking advantage of circulant matrices, further improved by resolving artificial boundary issues (21; 36). Many hard cases have been tackled by using context information (94; 156), short and long-term memory (83; 54), and scale-estimation (20), just to name a few. Recently, deep learning features have begun to play an important role in correlation filters (96; 129; 22; 19; 82; 139; 76). On the other hand, tracking-by-detection approaches typically learn a classifier to pick up the positive image patches wrapping around the target object. Pioneered by (62), many learning techniques have been suggested, e.g. multiple instance learning (9), structured output SVMs (46), online boosting (43), and model ensembles (10). More recently, MDNet (96), with deep features and a deep classifier, achieved significantly higher accuracy.

3.3.2 Offline trackers

Several recent studies have shown that we can build accurate trackers without online adaptation (13; 49; 132) due to powerful deep learning features. Siamese-style networks take a small target image patch and a large search image patch, and directly regress the target location (49) or generate a response map (13) via a correlation layer (33). In order to consider temporal information, recurrent networks have also been explored in (61; 37; 42; 152).

3.3.3 Meta-learning

This is an emerging field in machine learning and its applications. Although it is not a new concept (120; 121; 52; 134), many recent works have shown very promising results along with deep learning success. (7; 16; 148; 77) attempted to replace hand-crafted optimization algorithms

with meta-learned deep networks. (109) took this idea into few shot or one shot learning problem. It aimed to learn optimal update strategies based on how accurate a learner can classify test images with few training examples when the learner follows the strategies from the meta-learner. Instead of removing existing optimization algorithms, (31) focuses on learning initialization that are most suitable for existing algorithms. (78) further learns parameters of existing optimization algorithms along with the initialization. Unlike approaches introduced above, there also have been several studies to directly predict the model parameters without going through the optimization process (152; 12; 145).

3.4 Experiments

3.4.1 Experimental setup

3.4.1.1 VOT2016

It contains 60 videos (same videos from VOT 2015 (69)). Trackers are automatically reinitialized once it drifts off the target: zero overlap between predicted bounding box and the ground truth. In this reset-based experiments, three primary measures have been used, (i) *accuracy*, (ii) *robustness* and (iii) *expected average overlap (EAO)*. The accuracy is defined as average overlap during successful tracking periods. The robustness is defined as how many times the trackers fail during tracking. The expected average overlap is an estimator of the average overlap a tracker is expected to attain on a large collection of short-term sequences.

3.4.1.2 OTB2015

It consists of 100 fully annotated video sequences. Unlike VOT2016, the one pass evaluation (OPE) is commonly used in OTB dataset (no restart at failures). The precision plots (based on the center location error) and the success plots (based on the bounding box overlap) are used to access the tracker performance.

3.4.1.3 Dataset for meta-training

We used a large scale video detection dataset (116) for meta-training both trackers. It consists of 30 object categories, which is a subset of 200 categories in the object detection dataset. Since characteristics of the dataset are slightly different from the object tracking dataset, we subsampled the dataset. First, we picked a video frame that contains a target object whose size is not larger than 60% of the image size. Then, a training video sequence is constructed by sampling all subsequent frames from that frame until the size of the target object reaches 60%. We ended up having 718 video sequences. In addition, for the experiments on OTB2015 dataset, we also used an additional 58 sequences from object tracking datasets in VOT2013,VOT2014,and VOT2015 (70), excluding the videos included in OTB2015, following MDNet's approach (96). These sequences were selected in the mini-batch selection stage with the probability 0.3. Similarly, we used 80 sequences from OTB2015, excluding the videos in VOT2016 for the experiments on VOT2016 dataset.

3.4.1.4 Baseline implementations

We selected two trackers, MDNet (96) and CREST (129). For CREST, we re-implemented our own version in python based on publicly released code written in MATLAB. We meta-trained our version. For MDNet, the authors of MDNet provide two different source codes, written in MATLAB and python, respectively. We used the latter one and called it as pyMDNet or pySDNet, depending on pre-training methods. We meta-trained pySDNet, and call it as MetaSDNet. Note that overall accuracy of pyMDNet is lower than MDNet on OTB2015 (.652 vs .678 in success rates with overlap metric). For fair comparison, we compared our MetaSDNet to pyMDNet.

3.4.1.5 Meta-training details

In MetaSDNet, we used the first three conv layers from pre-trained vgg16 as feature extractors. During meta-training, we randomly initialized the last three fc layers, and used Adam as

Table 3.1: Quantitative results on VOT2016 dataset. The numbers in legends represent the number of iterations at the initial frame. EAO (expected average overlap) - 0 to 1 scale, higher is better. A (Accuracy) - 0 to 1 scale, higher is better. R (Robustness) - 0 to N, lower is better. We ran each tracker 15 times and reported averaged scores following VOT2016 convention.

	EAO	Acc	R
MetaCREST-01	**0.317**	**0.519**	**0.932**
CREST	0.283	0.514	1.083
CREST-Base	0.249	0.502	1.383
CREST-10	0.252	0.509	1.380
CREST-05	0.262	0.510	1.298
CREST-03	0.262	0.514	1.283
CREST-01	0.259	0.505	1.277

(a) The results of MetaCREST

	EAO	Acc	R
MetaSDNet-01	**0.314**	0.526	**0.934**
pyMDNet-30	0.304	**0.540**	0.943
pyMDNet-15	0.299	0.541	0.977
pyMDNet-10	0.291	0.535	0.989
pyMDNet-05	0.254	0.523	1.198
pyMDNet-03	0.184	0.488	1.703
pyMDNet-01	0.119	0.431	2.733

(b) The results of MetaSDNet

the optimizer with learning rate 1e-4. We only updated the last three fc layers for the first 5,000 iterations and trained all layers for the rest of iterations. The learning rate was reduced to 1e-5 after 10,000 iterations, and we trained up to 15,000 iterations. For α, we initialized to 1e-4, and also used Adam with learning rate 1e-5, then was decayed to 1e-6 after 10,000 iterations. We used mini-batch size $N_{\text{mini}} = 8$. For the meta-update iteration T, larger T gave us only small improvement, so we set to 1. For each training episode, we sample one random future frame uniformly from 1 to 10 ahead. In MetaCREST, we randomly initialized θ_0 and also used Adam with learning rate 1e-6. For α, we initialized to 1e-6, and learning rate of Adam was also set to 1e-6. $N_{\text{mini}} = 8$ and meta-training iterations was 10,000 (at 50,000 iterations, the learning rate was reduced to 1e-7). We used same hyper-parameters for both OTB2015 and VOT2016 experiments. For other hyper-parameters, we mostly followed the ones in the original trackers. For more details, the code and raw results will be released.

3.4.2 Experimental results

3.4.2.1 Quantitative evaluation

Table 3.1 shows quantitative results on VOT2016. In VOT2016, EAO is considered as the main metric since it consider both accuracy and robustness. Our meta-trackers, both MetaCREST

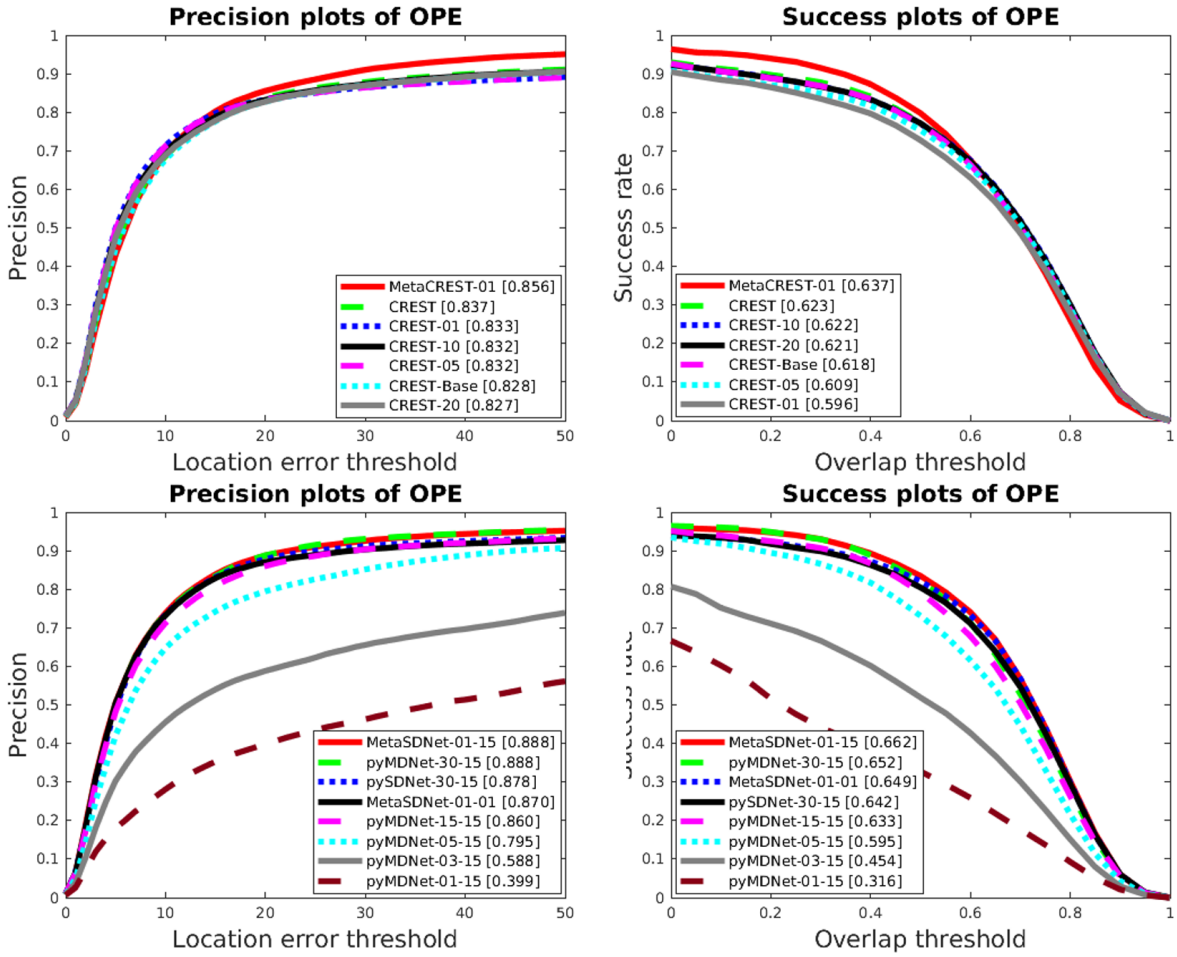


Figure 3.3: Precision and success plots over 100 sequences in OTB2015 dataset with one-pass evaluation (OPE). For CREST (top row),The numbers in legends represent the number of iterations at the initial frame, and all used 2 iterations for the subsequent model updates. For MDNet experiments (bottom row), 01-15 means, 1 training iterations at the initial frame and 15 training iterations for the subsequent model updates.

and MetaSDNet, consistently improved upon their counterparts by significant margins. Note that this is the improvement without their advanced techniques, e.g. pyMDNet with specialized multi-domain training and CREST with spatio-temporal residual modules. The performances of the accuracy metric are not very different than the original trackers. Because it computes the average overlap by only taking successful tracking periods into account, we did not change other factors that might affect the accuracy in the original trackers, e.g. scale estimation. Quantitative results on OTB2015 are depicted in Figure 3.3. Both of MetaSDNet and MetaCREST also improved

Table 3.2: Speed and performance of the initialization: The right table shows the losses of estimated response map in MetaCREST. The left table shows the accuracy of image patches in MetaSDNet. B (Before) - the performance of the initial frame before training, A (After) - the performance of the initial frame after training, LH (Lookahead) - the performance of next 5 frames after training, Time - wall clock time to train in seconds

	B	A	LH	Time(s)		B	A	LH	Time(s)
MetaCREST-01	0.48	0.04	**0.05**	**0.090**	MetaSDNet-01	0.50	0.98	**0.97**	**0.124**
CREST-01	0.95	0.82	0.87	0.395	pyMDNet-01	0.51	0.56	0.56	0.123
CREST-03	0.95	0.62	0.75	0.424	pyMDNet-03	0.51	0.79	0.78	0.373
CREST-05	0.95	0.45	0.63	0.550	pyMDNet-05	0.51	0.84	0.84	0.656
CREST-10	0.95	0.24	0.40	0.668	pyMDNet-10	0.51	0.95	0.93	1.171
CREST-20	0.95	0.18	0.31	1.048	pyMDNet-15	0.51	0.97	**0.97**	1.819
CREST-65	0.95	0.01	0.30	1.529	pyMDNet-30	0.51	0.99	**0.98**	3.508

upon their counterparts in both precision and success plots with only one iteration for initialization. Detailed results of individual sequences in both of VOT2016 and OTB2015 are presented in Appendix.

We require only one iteration at the initial frame to outperform the original trackers. We also performed the experiments with more than one iteration, but the performance gain was not significant. On the other hand, MDNet takes 30 iterations to converge at the initial frame as reported in their paper, and fewer iterations caused serious performance degradation. This confirms that getting a robust target model at the initial frame is very important for subsequent frames. For CREST, performance drop was not significant as MDNet, but it was still more than 10 iterations to reach to its maximum performance. MDNet updates the model 15 iterations for subsequent frames at every 10 frames regularly (or when it failed, meaning its score is below a predefined threshold).

3.4.2.2 Speed and performance of the initialization

We reported the wall clock time speed at the initial frame in Table 3.2, on a single TITAN-X GPU. In CREST, in addition to feature extraction, there are two more computational bottlenecks. The first is the convolution with correlation filters. Larger objects means larger filters and more computations. We reported average time across all 100 sequences. Another heavy computation

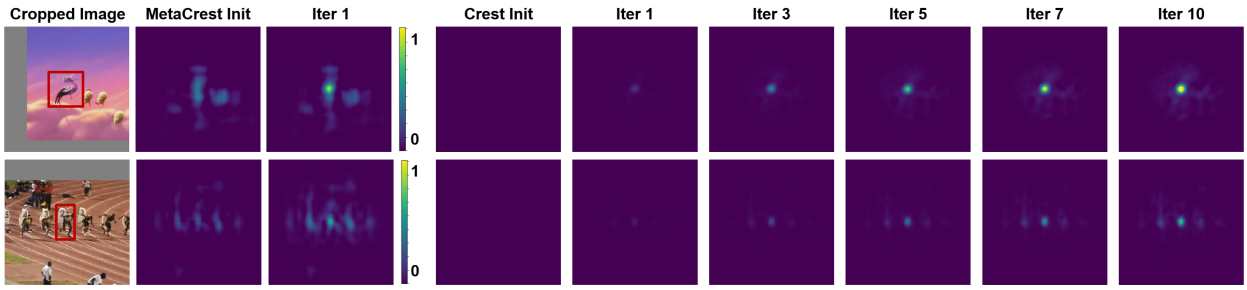


Figure 3.4: Visualizations of response maps in CREST: Left three columns represents the image patch at the initial frame, response map with meta-learned initial correlation filters θ_{0_f}, response map after updating 1 iteration with learned α, respectively. The rest of seven columns on the right shows response maps after updating the model up to 10 iterations.

comes from PCA at the initial frame. It also depends on the size of the objects. Larger objects lead to larger center cropped images, features, and more computation in PCA.

MDNet requires many positive and negative patches, and also many model update iterations to converge. A large part of the computation comes from extracting CNN features for every patch. MetaSDNet needs only a few training patches and can achieve 30x speedup (0.124 vs 3.508), while improving accuracy. If we used more compact CNNs for feature extractions, the speed could have been in the range of real-time processing. For subsequent frames in MDNet, model update time is of less concern because MDNet only updates the last 3 fully connected layers, which are relatively faster than feature extractors. The features are extracted at every frame, stored in a database, and update the model every 10 frames. Therefore, the actual computation is well distributed across every frames.

We also showed the performance of the initialization to see the effectiveness of our approach (in Table 3.2. We measured the performance with learned initialization. After initial training, we measure the performance on the first frame and 5 future frames to see generalizability of trackers. MetaSDNet achieved very high accuracy after only one iteration, but accuracy of pyMDNet after one iteration was barely above guessing (guessing is 50% and all negative prediction is 75% accuracy since sampling ratio was 1:3 between positive and negative samples). The effectiveness is more apparent in MetaCREST. MetaCREST-01 without any updates gave already close per-

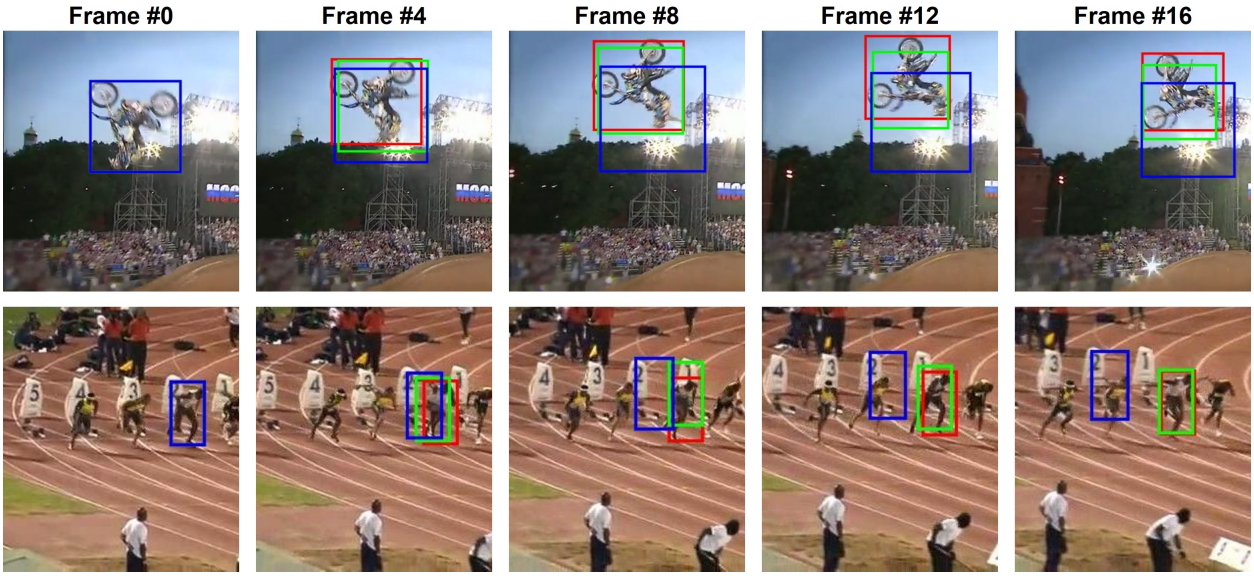


Figure 3.5: Qualitative examples: tracking results at early stage of MotorRolling (top) and Bolt2 (bottom) sequences in OTB2015 dataset. Color coded boxes: ground Truth (Red), MetaCREST-01 (Green) and CREST (Blue).

formance to CREST-05 after training (0.48 vs 0.45). In original CREST tracker, they train the model until it reaches a loss of 0.02, which corresponds to an average 65 iterations. However, its generalizability at future frames is limited compared to ours (.05 vs .30). Although this is not directly proportional to eventual tracking performance, we believe this is clear evidence that our meta-training algorithm based on future frames is indeed effective, as also supported by overall tracking performance.

3.4.2.3 Visualization of response maps

We visualized response maps in MetaCREST at the initial frame (Figure 3.4). A meta-learned initialization, θ_0 should be capable of learning generic objectness or visual saliency. At the same time, it should not be instance specific. It turns out that is the case. The second column in Figure 3.4 shows response maps by applying correlation filters to the cropped image (first column) with θ_0. Without any training, it already generates high response values on some locations where there are objects. But, more importantly, there is no clear maximum. After one iteration, the max-

imum is clearly located at the center of the response map. In contrast to MetaCREST, CREST consumes more iterations to produce high response values on the target.

3.4.2.4 Qualitative examples of robust initialization

In Figure 3.5, we present some examples where MetaCrest overcomes some of the issues in the original CREST. In MotorRolling sequence (top row), CREST was distracted by a horizontal line from the forest in the background. CREST easily reached to 0.0000 loss defined in Equation 3.6 at the initial frame, as opposed to 0.1255 in MetaCREST. This is a strong evidence that an optimal solution does not necessarily mean good generalizability on future frames. In contrast, MetaCREST, generalizes well to future frames, despite not finding an optimal solution at the current frame. In Bolt2 sequence (bottom row), CREST also reached to 0.0000 loss (vs 0.0534 in MetaCREST). In a similar way, a top left part in the bounding box was the distractor. MetaCREST could easily ignore the background clutter and focused on the object in the center of the bounding box.

CHAPTER 4: META-CURVATURE

Despite huge progress in artificial intelligence, the ability to quickly learn from few examples is still far short of that of a human. We are capable of utilizing prior knowledge from past experiences to efficiently learn new concepts or skills. With the goal of building machines with this capability, *learning-to-learn* or *meta-learning* has begun to emerge with promising results.

One notable example is model-agnostic meta-learning (MAML) (31; 98), which has shown its effectiveness on various few-shot learning tasks. It formalizes *learning-to-learn* as a meta objective function and optimizes it with respect to a model's initial parameters. Through the meta-training procedure, the resulting model's initial parameters become a very good prior representation and the model can quickly adapt to new tasks or skills through one or more gradient steps with a few data examples. Although this end-to-end approach, using standard gradient descent as the inner optimization algorithm, was theoretically shown to approximate any learning algorithm (32), recent experiments indicate that the choice of the inner-loop optimization algorithm affects performance (78; 8; 44).

Given the sensitivity to the inner-loop optimization algorithm, second order optimization methods (or preconditioning the gradients) are worth considering. They have been extensively studied and have shown their practical benefits in terms of faster convergence rates (99), an important aspect of few-shot learning. In addition, the problems of computational and spatial complexity for training deep networks can be effectively handled thanks to recent approximation techniques (85; 115). Nevertheless, there are issues with using second order methods in its current form as an inner loop optimizer in the meta-learning framework. First, they do not usually consider generalization performance. They compute local curvatures based on training losses and move along the local curvatures as far as possible. This can be very harmful, especially in the

few-shot learning setup, because it can overfit easily and quickly. In addition, it may be computationally more expensive since third order derivatives are needed for the outer loop optimizations when the second order derivatives are presented in the inner loop.

In this work, we propose to learn a curvature for better generalization and faster model adaptation in the meta-learning framework, we call *meta-curvature*. The key intuition behind MAML is that there are some representations are broadly applicable to all tasks. In the same spirit, we hypothesize that there are some curvatures that are broadly applicable to many tasks. Curvatures are determined by the model's parameters, network architectures, loss functions, and training data. Assuming new tasks are distributed from the similar distribution as meta-training distribution, there may exist common curvatures that can be obtained through meta-training procedure. The resulting meta-curvatures that are represented by a second-order tensor (matrix), coupled with the simultaneously meta-trained model's initial parameters, will transform the gradients by multiplying the curvature matrix, such that the updated model has better performance on new tasks with fewer gradient steps. In order to efficiently capture the dependencies between all gradient coordinates for large networks, we design a multilinear mapping consisting of a series of tensor-products to transform the gradients. It also considers layer specific structures, e.g. convolutional layers, to effectively reflects our inductive bias. In addition, meta-curvature can be easily plugged into existing meta-learning frameworks like MAML without additional, burdensome higher-order gradients.

We demonstrate the effectiveness of our proposed method on the few-shot learning tasks done by (143; 109; 31). Experimental results show consistent improvements on few-shot classification tasks on MiniImagenet datasets. In the most extreme few-shot learning setup, which is one-shot one-gradient step, the proposed method outperformed the baselines by a large margin. We also found that it gave us much faster convergence rates of the meta-training. In addition, we perform few-shot reinforcement learning experiments and show promising results compared to its strong baselines.

4.1 Method

First, we present a simple and efficient form of the meta-curvature computation through the lens of tensor algebra. Then, we present a matrix-vector product view to provide intuitive idea of the connection to the second order matrices. And, we discuss the relationships to other methods in computational aspects.

4.1.1 Tensor product view

We consider neural networks as our models. With a slight abuse of notation, let the model's parameters $\mathcal{W}^l \in \mathbb{R}^{C^l_{out} \times C^l_{in} \times d^l}$ and its gradients of loss function $\mathcal{G}^l \in \mathbb{R}^{C^l_{out} \times C^l_{in} \times d^l}$, at each layers l. To avoid cluttered notation, we will omit the superscript l. We choose superscripts and dimensions with convolutional layers in mind, but the method can be easily extended to other layers, e.g. recurrent layers. C_{out}, C_{in}, and d are the number of output channels, the number of input channels, and the filter size respectively. d is height $\times$ width in convolutional layers and 1 in fully connected layers. We also define meta-curvature matrices, $\mathbf{M}_o \in \mathbb{R}^{C_{out} \times C_{out}}$, $\mathbf{M}_i \in \mathbb{R}^{C_{in} \times C_{in}}$, and $\mathbf{M}_f \in \mathbb{R}^{d \times d}$. Now a meta-curvature function takes a multidimensional tensor as an input and has all meta-curvature matrices as learnable parameters.

$$\mathrm{MC}(\mathcal{G}) = \mathcal{G} \times_3 \mathbf{M}_f \times_2 \mathbf{M}_i \times_1 \mathbf{M}_o. \tag{4.1}$$

Figure 4.1 (top) shows an example of computational illustration with an input tensor $\mathcal{G} \in \mathbb{R}^{2 \times 3 \times d}$. The main motivation of this structure is to capture the second-order dependencies between all model's parameters with minimal computational and space overhead. First, it performs linear transformations for all 3-mode fibers of $\mathcal{G}$. In other words, $\mathbf{M}_f$ captures the parameter dependencies between the elements within a 3-mode fiber, e.g. all gradient elements in a channel of a convolutional filter. Secondly, the 2-mode product models the dependencies between 3-mode fibers computed from the previous stage. All 3-mode fibers are updated by linear combinations of other 3-mode fibers belonging to the same output channel (linear combinations of 3-mode

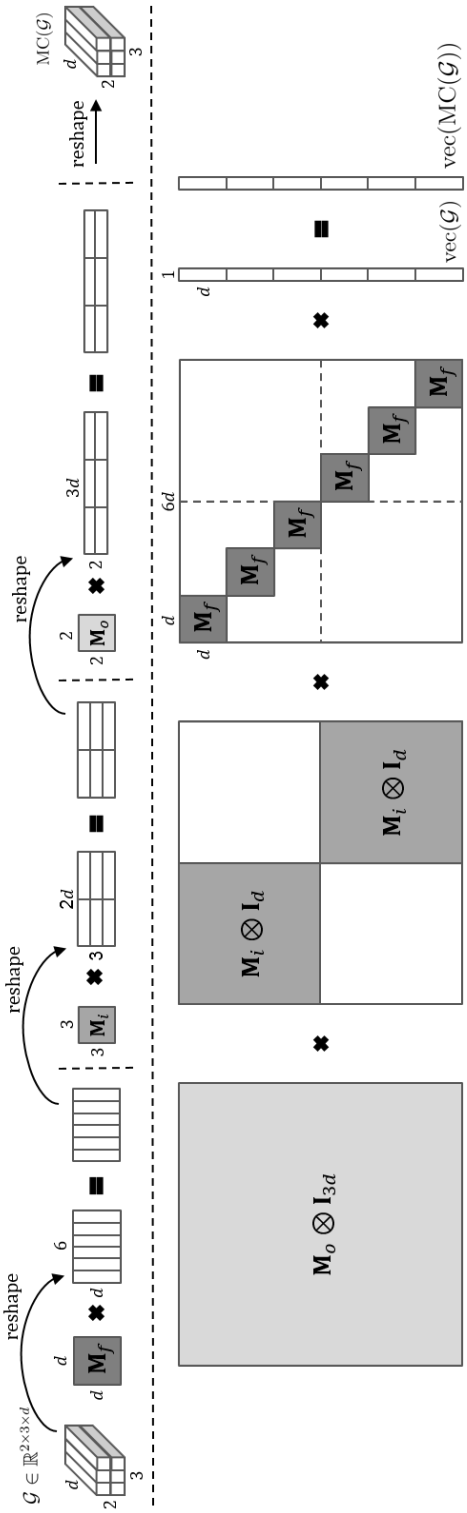


Figure 4.1: An example of meta-curvature computational illustration with $\mathcal{G} \in \mathbb{R}^{2\times3\times d}$. Top: tensor algebra view, Bottom: matrix-vector product view.

fibers in a convolutional filter). Finally, the 1-mode product is performed in order to model the dependencies between the gradients of all convolutional filters. Similarly, the gradients of all convolutional filters are updated by linear combinations of gradients of other convolutional filters.

A useful property of n-mode products is the fact that the order of the multiplications is irrelevant for distinct modes in a series of multiplications. For example, $\mathcal{G} \times_3 \mathbf{M}_f \times_2 \mathbf{M}_i \times_1 \mathbf{M}_o = \mathcal{G} \times_1 \mathbf{M}_o \times_2 \mathbf{M}_i \times_3 \mathbf{M}_f$. Thus, the proposed method indeed examines the dependencies of the elements in the gradient all together.

4.1.2 Matrix-vector product view

We can also view the proposed meta-curvature computation as a matrix-vector product analogous to that from other second order methods. We can expand the meta-curvature matrices as follows.

$$\widehat{\mathbf{M}_o} = \mathbf{M}_o \otimes \mathbf{I}_{C_{in}} \otimes \mathbf{I}_d, \tag{4.2}$$

$$\widehat{\mathbf{M}_i} = \mathbf{I}_{C_{out}} \otimes \mathbf{M}_i \otimes \mathbf{I}_d, \tag{4.3}$$

$$\widehat{\mathbf{M}_f} = \mathbf{I}_{C_{out}} \otimes \mathbf{I}_{C_{in}} \otimes \mathbf{M}_f, \tag{4.4}$$

where $\otimes$ is the Kronecker product, $\mathbf{I}_k$ is k dimensional identity matrix, and the three expanded matrices are all same size $\widehat{\mathbf{M}_o}, \widehat{\mathbf{M}_i}, \widehat{\mathbf{M}_f} \in \mathbb{R}^{C_{out}C_{in}d \times C_{out}C_{in}d}$. Now we can transform the gradients with the meta-curvature as

$$\text{vec}(\text{MC}(\mathcal{G})) = \mathbf{M}_{mc}\text{vec}(\mathcal{G}), \tag{4.5}$$

where $\mathbf{M}_{mc} = \widehat{\mathbf{M}_o}\widehat{\mathbf{M}_i}\widehat{\mathbf{M}_f}$. Unlike the normal matrices do not hold commutative property, the expanded matrices satisfy, e.g. $\widehat{\mathbf{M}_o}\widehat{\mathbf{M}_i}\widehat{\mathbf{M}_f} = \widehat{\mathbf{M}_f}\widehat{\mathbf{M}_i}\widehat{\mathbf{M}_o}$. Thus, $\mathbf{M}_{mc}$ models the dependencies of the model parameters all together. Note that we can also write $\mathbf{M}_{mc} = \mathbf{M}_o \otimes \mathbf{M}_i \otimes \mathbf{M}_f$ with the tensor algebra notations, but this is non-commutative, $\mathbf{M}_o \otimes \mathbf{M}_i \otimes \mathbf{M}_f \neq \mathbf{M}_f \otimes \mathbf{M}_i \otimes \mathbf{M}_o$.

Figure 4.1 (bottom) shows a computational illustration. $\widehat{\mathbf{M}_f}\text{vec}(\mathcal{G})$, which is equivalent computation to $\mathcal{G} \times_3 \mathbf{M}_f$, can be interpreted as a giant matrix-vector multiplication with block diagonal matrix, where each block shares same meta-curvature matrix $\mathbf{M}_f$. It resembles the block diagonal approximation strategies in some second-order methods for training deep networks, but as we are interested in learning meta-curvature matrices, no approximation is involved. And matrix-vector product with $\widehat{\mathbf{M}_o}$ and $\widehat{\mathbf{M}_i}$ are used to capture inter-parameter dependencies and are equivalent to 2-mode and 3-mode products of Eq. 4.1.

4.1.3 Relationship to other methods

We note that the suggested method is computationally related to Tucker decomposition (67), which decomposes a tensor into low rank cores with projection factors and aims to closely reconstruct the original tensor. We maintain full rank gradient tensors, however, and our main goal is to transform the gradients for better generalization. Recently, (68) proposed to learn the projection factors in Tucker decomposition for fully connected layers in deep networks. Again, their goal was to find the low rank approximations of fully connected layers for saving computational and spatial cost.

We also note that the computation of meta-curvature is closely related to Kronecker-factored Approximate Curvature (K-FAC) (85; 45), which approximates the Fisher information matrix by the Kronecker product, e.g. $\mathbf{F} \approx \mathbf{A} \otimes \mathbf{G}$, where $\mathbf{A}$ is computed from the activation of input units and $\mathbf{G}$ is computed from the gradient of output units. Its main goal is to approximate the Fisher such that matrix vector products between its inversion and the gradient can be computed efficiently. However, especially in convolutional layers, we found that maintaining $\mathbf{A} \in \mathbb{R}^{C_{in}d \times C_{in}d}$ was quite expensive both computationally and spatially even for smaller networks. In addition, when we applied this factorization scheme to meta-curvature, it tends to easily overfit to meta-training set. On the contrary, we maintain two separated matrices, $\mathbf{M}_i \in \mathbb{R}^{C_{in} \times C_{in}}$ and $\mathbf{M}_f \in \mathbb{R}^{d \times d}$, which allows us to avoid overfitting and heavy computation. More importantly, we learn meta-curvature matrices to improve generalization instead of directly computing them

Algorithm 2 Training MAML with the meta-curvature for few-shot supervised learning

Input : task distribution $p(\mathcal{T})$, learning rate α, β

Initialize $\mathbf{M}_o, \mathbf{M}_i, \mathbf{M}_f = \mathbf{I}$
while not converged **do**
    Sample batch of tasks $\tau_i \sim p(\mathcal{T})$
    for all τ_i **do do**
        $\theta^{\tau_i} = \theta - \alpha \mathbf{M}_{mc} \nabla \mathcal{L}_{\text{tr}}^{\tau_i}(\theta)$ ▷ Assuming one gradient step
    $\theta \leftarrow \text{ADAM}\big(\theta, \beta, \nabla_\theta \sum_{\tau_i} \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i})\big)$
    $\mathbf{M}_o \leftarrow \text{ADAM}\big(\mathbf{M}_o, \beta, \nabla_{\mathbf{M}_o} \sum_{\tau_i} \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i})\big)$
    $\mathbf{M}_i \leftarrow \text{ADAM}\big(\mathbf{M}_i, \beta, \nabla_{\mathbf{M}_i} \sum_{\tau_i} \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i})\big)$
    $\mathbf{M}_f \leftarrow \text{ADAM}\big(\mathbf{M}_f, \beta, \nabla_{\mathbf{M}_f} \sum_{\tau_i} \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i})\big)$

from the activation and the gradient of training loss. Also, we do not require expensive matrix inversions.

4.1.4 Meta-training

We follow a typical meta-training algorithm and replace the gradients in the inner update rule with the projected gradient. We initialize all meta-curvature matrices as identity matrices so that the projected gradients are equal to the original ones at the beginning. We used the ADAM (66) optimizer for the outer loop optimization and update the model's initial parameters and meta-curvatures simultaneously. To avoid cluttered notation, we assumed the model has only one layer. It is straightforward to extend to multiple layers (Alg. 2).

4.2 Analysis

In this section, we will explore how a meta-trained matrix $\mathbf{M}_{mc}$, or $\mathbf{M}$ for brevity, can operate for better generalization. Let us take the gradient of meta-objective w.r.t $\mathbf{M}$ for a task τ_i. With the inner update rule $\theta - \alpha \mathbf{M} \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_i}(\theta)$, and by applying chain rule,

$$\nabla_{\mathbf{M}} \mathcal{L}_{\text{val}}^{\tau_i}(\mathbf{M}) = -\alpha \nabla_\theta \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i}) \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_i}(\theta)^\top, \tag{4.6}$$

where θ^{τ_i} is the parameter for the task τ_i after the inner update. It is the outer product between the gradients of validation loss and training loss. Note that there is a significant connection to the Fisher information matrix. For a task τ_i, if we define the loss function as negative log likelihood, e.g. a supervised classification task $\mathcal{L}^{\tau_i}(\theta) = \mathbb{E}_{(x,y)\sim p(\tau_i)}[-\log_\theta p(y|x)]$, then the empirical Fisher information matrix can be defined as $\mathbf{F} = \mathbb{E}_{(x,y)\sim p(\tau_i)}[\nabla_\theta \log_\theta p(y|x) \nabla_\theta \log_\theta p(y|x)^\top]$. There are three clear distinctions. First, the training and validation sets are treated separately in the meta-gradient $\nabla_{\mathbf{M}} \mathcal{L}^{\tau_i}_{\text{val}}$, while the empirical Fisher is computed with only training set (validation set is not available during training). Secondly, the gradient of the validation set is evaluated at new parameters θ^{τ_i} after the inner update in the meta-gradient. Finally, the Fisher information matrix is positive semi-definite by construction, but it is not the case for the meta-gradient. This is an attractive property since it guarantees that the transformed gradient is always a descent direction. However, we mainly care about generalization performance in this work and the descent direction of the training loss gradient does not necessarily mean it would decrease the validation loss. Hence, we rather not force this property in this work, but leave it for future work.

Now let us consider what the meta-gradient can do for good generalization performance. Given a fixed point θ and a meta training set $\mathcal{T} = \{\tau_i\}$, standard gradient descent from an initialization $\mathbf{M}_0$, gives the following update.

$$\mathbf{M}_T = \mathbf{M}_0 - \beta \sum_{i=1}^{|\mathcal{T}|} \nabla_{\mathbf{M}} \mathcal{L}^{\tau_i}_{\text{val}}(\mathbf{M}_{i-1}) \tag{4.7}$$

$$= \mathbf{M}_0 + \alpha\beta \sum_{i=1}^{|\mathcal{T}|} \nabla_\theta \mathcal{L}^{\tau_i}_{\text{val}}(\theta^{\tau_i}(\mathbf{M}_{i-1})) \nabla_\theta \mathcal{L}^{\tau_i}_{\text{tr}}(\theta)^\top, \tag{4.8}$$

$$\text{where} \quad \theta^{\tau_i}(\mathbf{M}_{i-1}) = \theta - \alpha \mathbf{M}_{i-1} \nabla_\theta \mathcal{L}^{\tau_i}_{\text{tr}}(\theta), \tag{4.9}$$

or θ^{τ_i} for brevity. α and β are fixed inner/outer learning rates respectively. Here, we assume a standard gradient descent for simplicity. But the argument extends to other advanced gradient algorithms, such as momentum and ADAM, since we can replace the gradients with some exponentially weighted moving averages according to the methods.

Note that this has close connection to *online natural gradient* (102; 115), e.g. $\mathbf{F}_{t-1} = (1-\gamma)\mathbf{F}_{t-1} + \gamma g_t g_t^\top$, where g_t is the gradient at the t-th iteration. It is also related to the rank-1 update in quasi-Newton methods, such as the *SR1 method* (99). In short, meta-training aims to update $\mathbf{M}$ for each meta-training batch.

We apply $\mathbf{M}_T$ to the gradients of a new task, giving the transformed gradients

$$\mathbf{M}_T \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_{\text{new}}}(\theta) = \big(\mathbf{M}_0 + \alpha\beta \sum_{i=1}^{|\mathcal{T}|} \nabla_\theta \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i}) \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_i}(\theta)^\top\big) \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_{\text{new}}}(\theta) \tag{4.10}$$

$$= \mathbf{M}_0 \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_{\text{new}}}(\theta) + \beta \sum_{i=1}^{|\mathcal{T}|} \big(\nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_i}(\theta)^\top \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_{\text{new}}}(\theta)\big) \alpha \nabla_\theta \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i}) \tag{4.11}$$

$$= \mathbf{M}_0 \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_{\text{new}}}(\theta) + \beta \sum_{i=1}^{|\mathcal{T}|} \big(\underbrace{\nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_i}(\theta)^\top \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_{\text{new}}}(\theta)}_{\text{A. Gradient similarity}}\big)\big(\underbrace{\alpha \nabla_\theta \mathcal{L}_{\text{val}}^{\tau_i}(\theta) + \mathcal{O}(\alpha^2)}_{\text{B. Taylor expansion}}\big). \tag{4.12}$$

Given $\mathbf{M}_0 = \mathbf{I}$, the second term in the R.H.S. of Eq. 4.12 can represent the transformed gradient direction for the new task. For Eq. 4.12, we used the Taylor expansion of vector-valued function, $\nabla_\theta \mathcal{L}_{\text{val}}^{\tau_i}(\theta^{\tau_i}) \approx \nabla_\theta \mathcal{L}_{\text{val}}^{\tau_i}(\theta) + \nabla_\theta^2 \mathcal{L}_{\text{val}}^{\tau_i}(\theta)(\theta - \alpha \mathbf{M}_{i-1} \nabla_\theta \mathcal{L}_{\text{tr}}^{\tau_i}(\theta) - \theta)$.

The term A of Eq. 4.12 is the inner product between the gradients of meta-training losses and new test losses. We can simply interpret this as how similar the gradient directions between two different tasks. This has been explicitly used in continual learning or multi-task learning setup to consider task similarity (27; 80; 113). When we have a loss function in the form of finite sums, this term can be also interpreted as a kernel similarity between the respective sets of gradients (see Eq. 4 of (93)).

With the first term in B of Eq. 4.12, we compute a linear combination of the gradients of validation losses from the meta-validation set. Its weighting factors are computed based on the similarities between the tasks from the meta-training set and the new task as explained above. Therefore, we essentially perform a soft nearest neighbor voting to find the direction among the validation gradients from the meta-validation set. Given the new task, the gradient may lead the model to overfit (or underfit). However, the proposed method will extract the knowledge from

the past experiences and find the gradients that gave us good validation performance during the meta-training process.

4.3 Related Work

4.3.1 Meta-learning

Meta-learning's most distinctive feature is that it aims to achieve better generalization in a principled way. (158; 159) learns to search neural architectures so that the generated architectures can maximize the accuracy of held-out sets. (88) trains optimizers such that trained optimizers can train child neural networks minimizing generalization error. Learning hyperparameters are also explored to find the best hyperparameters to maximize the validation performance (84; 81). Model-agnostic meta-learning (MAML) highlighted the importance of the model's initial parameters for better generalization (32) and there have been many extensions to improve the framework, e.g. for continuous adaptation (5), better credit assignment (114), and robustness (65). In this work, we improve the inner update optimizers by learning a curvature for better generalization and fast model adaptation.

Meta-SGD (78) suggests to learn coordinate-wise learning rates. We can interpret it as an diagonal approximation to meta-curvature in a similar vein to recent adaptive learning rates methods, such as (135; 66; 28), performing diagonal approximations of second-order matrices. Recently, (8) suggested to learn layer-wise learning rates through the meta-training. However, both methods do not consider the dependencies between the parameters, which is our main focus of this work.

4.3.2 Few-shot classification

As a good test bed to evaluate few-shot learning, huge progress has been made in the few-shot classification task. Triggered by (143), many recent studies have focused on discovering effective inductive bias on classification task. For example, network architectures that perform

nearest neighbor search (143; 127) were suggested. Some improved the performance by modeling the interactions or correlation between training examples (90; 38; 131; 103; 95). In order to overcome the nature of few-shot learning, the generative models have been suggested to augment the training data (124; 144) or generate model parameters for the specified task (117; 107). The state-of-the-art results are achieved by additionally training 64-way classification task for pre-training (107; 117; 103) with larger ResNet models (107; 117; 95; 90). In this work, our focus is to improve the model-agnostic few-shot learner that is broadly applicable to other tasks, e.g. reinforcement learning setup.

4.3.3 Learning optimizers

Our proposed method may fall within the *learning optimizer* category (109; 7; 149). They also take as input the gradient and transform it via a neural network to achieve better convergence behavior. However, their main focus is to capture the training dynamics of individual gradient coordinates (109; 7) or to obtain a generic optimizer that is broadly applicable for different datasets and architectures (149; 7). On the other hand, we meta-learn a curvature coupled with the models initialization parameters. We focus on a fast adaptation scenario requiring a small number of gradient steps. Therefore, our method does not consider a history of the gradients, which enables us to avoid considering a complex recurrent architecture. Finally, our approach is well connected to existing second order methods while learned optimizers are not easily interpretable since the gradient passes through nonlinear and multilayer recurrent neural networks.

4.4 Experiments

We evaluate the proposed method on a synthetic data few-shot regression task, few-shot image classification tasks on Omniglot and MiniImagenet datasets, and few-shot reinforcement learning tasks. We test two variants of the meta-curvature. The first one, named as MC1, we fixed the $Mo = I$ in Eq 4.1. The second one, named as MC2, we learn all three meta-curvature matrices. We also report results on few-shot reinforcement learning in appendices.

Table 4.1: Few-shot regression results on sinusoidal functions.

Method	5-shot	10-shot	20-shot
MAML	0.686 ± 0.070	0.435 ± 0.039	0.228 ± 0.024
Meta-SGD	0.482 ± 0.061	0.258 ± 0.026	0.127 ± 0.013
LayerLR	0.528 ± 0.068	0.269 ± 0.027	0.134 ± 0.014
MC1	0.426 ± 0.054	0.239 ± 0.025	0.125 ± 0.013
MC2	$\mathbf{0.405 \pm 0.048}$	$\mathbf{0.201 \pm 0.020}$	$\mathbf{0.112 \pm 0.011}$

4.4.1 Few-shot regression

To begin with, we perform a simple regression of sinusoidal functions following (31; 78). During the meta-training process, sinusoidal functions are sampled, where the amplitude and phase are varied within $[0.1, 5.0]$ and $[0, \pi]$ respectively. The network architecture and all hyper-parameters are same as (31) and we only introduce the suggested meta-curvature. We reported the mean squared error with 95% confidence interval after one gradient step in Figure 4.1.

Experimental setup We used the same experimental setups in (31). During training and testing, the amplitude and the phase vary within $[0.1, 5.0]$ and $[0, \pi]$ respectively, and data points are sampled from uniform distribution $[-5, 5]$. We used one gradient step with the fixed learning rate 0.01 and Adam was used for meta-training with the outer loop learning rate 0.001. We used the same network architecture, which has two 40 dimension fully connected layers with ReLU activation. We randomly sampled 25 tasks for every iterations and trained 70000 iterations. We reported the performance from the trained model that had the minimum meta-training loss value. (31) reported the MSE for 5-shot setting, and we could reproduced the results. (78) has slightly different settings, so the MSE are not directly comparable to theirs.

Qualitative results: We provide qualitative results of few-shot regression task on sinusoidal functions in Figure 4.2. The star shape markers are the few data points for training, and we draw the curves based on each methods, MAML, Meta-SGD, and the proposed MC2. The left column is 5-shot and the right column is 10-shot experiments.

Table 4.2: Few-shot classification results on Omniglot dataset. † denotes 3 model ensemble.

	5-way 1-shot	5-way 5-shot	20-way 1-shot	20-way 5-shot
SNAIL (91)	99.07 ± 0.16	99.78 ± 0.09	97.64 ± 0.30	99.36 ± 0.18
GNN (39)	99.2	99.7	97.4	99.0
MAML	98.7 ± 0.4	99.9 ± 0.1	95.8 ± 0.3	98.9 ± 0.2
Meta-SGD	99.53 ± 0.26	99.93 ± 0.09	95.93 ± 0.38	98.97 ± 0.19
MAML++† (8)	99.47	99.93	97.65 ± 0.05	99.33 ± 0.03
MC1	99.47 ± 0.27	99.57 ± 0.12	97.60 ± 0.29	99.23 ± 0.08
MC2	99.77 ± 0.17	99.79 ± 0.10	97.86 ± 0.26	99.24 ± 0.07
MC2†	$\mathbf{99.97 \pm 0.06}$	99.89 ± 0.06	$\mathbf{99.12 \pm 0.16}$	$\mathbf{99.65 \pm 0.05}$

4.4.2 Few-shot classification on Omniglot

The Omniglot dataset consists of handwritten characters from 50 different languages and 1632 different characters. It has been widely used to evaluate few-shot classification performance. We follow the experimental protocol in (31) and all hyperparameters and network architecture are same as (31). We used the same experimental setups in (31). The N-way K-shot Omniglot task setup is as follows. We pick N unseen character classes out of entire dataset, and randomly sample K training images belonging to each character classes. Therefore, one training task would be to have NK images in total. The goal of the task is to classify images that are not part of training images out of N categories. Out of 1623 characters, we used 1100 characters for training, 100 characters for validation, and remaining 423 characters for testing. The network architecture is 4 convolutional layers with 64 filters and 1 fully connected layer for the final classification. We only used one inner gradient step with 0.4 learning rate for all meta-curvature experiments for training and testing. The batch size was set to 32 (5-way) and 16 (20-way), and outer loop learning rate is 0.001 and we trained 60000 iterations.

Except 5-shot 5-way setting, our simple 4 layers CNN with meta-curvatures outperform all MAML variants and also achieve state-of-the-art results without additional specialized architectures, such as attention module (SNAIL (91)) or relational module (GNN (39)). We provide the training curves in Figure 4.4 and our methods converge much faster and achieve higher accuracy.

4.4.3 Few-shot classification on miniImagenet and tieredImagenet

Datasets: The miniImagenet dataset was proposed by (143; 109) and it consists of 100 subclasses out of 1000 classes in the original dataset (64 training classes, 12 validation classes, 24 test classes). The tieredImagenet dataset (111) is a larger subset, composed of 608 classes and reduce the semantic similarity between train/val/test splits by considering high-level categories.

baseline CNNs: We used 4 layers convolutional neural network with the batch normalization followed by a fully connected layer for the final classification. In order to increase the capacity of the network, we increased the filter size up to 128. We found that the model with the larger filter seriously overfit (also reported in (31)). To avoid overfitting, we applied data augmentation techniques suggested in (18; 23). For a fair comparison to (8), we also reported the results of model ensemble. Throughout the meta-training, we saved the model regularly and picked 3 models that have the best accuracy on the meta-validation dataset. We re-implemented all three baselines and performed the experiments with the same settings.

Fig. 4.4 and Table 4.3 shows the results of baseline CNNs experiments on miniImagenet. MC1 and MC2 outperformed all other baselines for all different experiment settings. Surprisingly, not only do MC reaches to higher accuracy at convergence, but also both showed much faster convergence rates of the meta-training. Our methods share the same benefits as second order methods although we do not approximate any Hessian or Fisher matrices. Also our methods are very robust to hyperparameter settings while we had to extensively perform hyperparameter search to make other MAML variants work. Usually, MC2 outperforms MC1 because the more fine-grained meta-curvature enable us to effectively increase the model's capacity.

WRN-28-10 features and MLP: To the best of our knowledge, (117; 107) are current state-of-the-art methods that use pretrained WRN-28-10 (155) network (Wide Resiaul Network). They found that 5-way or 20-way classification was not good enough to obtain good features, so they first pre-trained 64-way classification task on entire meta-training set as a feature extractor network. And, they fine-tuned the network top of the feature extractor following few-shot classification experimental setup. We evaluated our methods on this setting by adding one hidden layer

Table 4.3: Few-shot classification results on miniImagenet test set (5-way classification) with baseline 4 layer CNNs. * is from the original papers. † denotes 3 model ensembles.

	1-shot		5-shot	
Inner steps	1 step	5 step	1 step	5 step
*MAML	·	48.7 ± 1.84	·	63.1 ± 0.92
*Meta-SGD	50.47 ± 1.87	·	64.03 ± 0.94	·
*MAML++†	51.05 ± 0.31	52.15 ± 0.26	·	68.32 ± 0.44
MAML	46.28 ± 0.89	48.85 ± 0.88	59.26 ± 0.72	63.92 ± 0.74
Meta-SGD	49.87 ± 0.87	48.99 ± 0.86	66.35 ± 0.72	63.84 ± 0.71
LayerLR	50.04 ± 0.87	50.55 ± 0.87	65.06 ± 0.71	66.64 ± 0.69
MC1	53.37 ± 0.88	53.74 ± 0.84	$\mathbf{68.47 \pm 0.69}$	$\mathbf{68.01 \pm 0.73}$
MC2	$\mathbf{54.23 \pm 0.88}$	$\mathbf{54.08 \pm 0.93}$	67.94 ± 0.71	$\mathbf{67.99 \pm 0.73}$
MC2†	$\mathbf{54.90 \pm 0.90}$	$\mathbf{55.73 \pm 0.94}$	$\mathbf{69.46 \pm 0.70}$	$\mathbf{70.33 \pm 0.72}$

MLP followed by a softmax classifier and our method again improved MAML variants by a large margin. Despite our best attempts, we could not find a good hyperparameters to train original MAML in this setting. Although our main goal is to push how much a simple gradient transformation in the inner loop optimization can improve general and broadly applicable MAML frameworks, our methods performs favorably against state-of-the-art methods. Our methods outperform task dependent weight generating approach suggested in (107). We also note that our results are also very competitive with the very latest state of the art using WRN-28-10 features, LEO (117). However, LEO uses many other components (e.g. a relation net, weight generator, etc.) which we do not incorporate for simplicity. (75) recently achieved the best result on miniImagenet dataset. It is not directly comparable since they used different backbone network, 15-shot meta-training scheme, and many regularization and augmentation techniques.

4.4.4 Visualization

Fig. 4.5 is a visualization of meta-trained meta-curvature matrices for 5-way 1-shot classification task. To visualize the full matrix, $\mathbf{M}_{mc}$, we picked up the matrices from the first convolutional layer in the small model (filter size 64). Therefore with the 3 color input channels,

Table 4.4: The results on miniImagenet and tieredImagenet with WRN-28-10 features. ‡ indicates that both meta-train and meta-validation are used during meta-training.

	miniImagenet		tieredImagenet	
(107)‡	59.60 ± 0.41	73.74 ± 0.19	·	·
LEO (center)‡ (117)	61.76 ± 0.08	77.59 ± 0.12	66.33 ± 0.05	81.44 ± 0.09
LEO (multiview)‡ (117)	63.97 ± 0.20	79.49 ± 0.70	·	·
MetaOptNet-SVM‡† (75)	64.09 ± 0.62	80.00 ± 0.45	65.81 ± 0.74	81.75 ± 0.53
Meta-SGD (center)	56.58 ± 0.21	68.84 ± 0.19	59.75 ± 0.25	69.04 ± 0.22
MC2 (center)	61.22 ± 0.10	75.92 ± 0.17	66.20 ± 0.10	82.21 ± 0.08
MC2 (center)‡	$\mathbf{61.85 \pm 0.10}$	77.02 ± 0.11	$\mathbf{67.21 \pm 0.10}$	$\mathbf{82.61 \pm 0.08}$
MC2 (multiview)‡	$\mathbf{64.40 \pm 0.10}$	$\mathbf{80.21 \pm 0.10}$	·	·

$\mathbf{M}_f \in \mathbb{R}^{9\times 9}$, $\mathbf{M}_i \in \mathbb{R}^{3\times 3}$, $\mathbf{M}_o \in \mathbb{R}^{64\times 64}$, and $\mathbf{M}_{mc} \in \mathbb{R}^{1728\times 1728}$. The diagonal elements are high values, mostly > 0.5. Interestingly, there are also a lot of off-diagonal elements > 0.5 or < -0.5. Thus, they capture the dependencies between the gradients.

4.4.5 Few-shot reinforcement learning

The goal of few-shot learning in reinforcement learning (RL) is that an agent can quickly adapt to a new task with little prior experience. A distinct feature from the few-shot supervised learning task is that the RL objective is not generally differentiable. Therefore, we use policy gradient methods to estimate the gradient both for inner and outer loop gradients. In addition, policy gradient methods are generally on-policy, which means that the training data depends on the agents initial policy. Therefore, the initial policy (with the meta-learned initial parameters) needs to explore as diverse experiences as possible to get proper feedback from a new task. We described the method and interpretation with respect to supervised classification tasks, but it can be easily modified to RL setting.

4.4.5.1 Experimental setup

We tested our method on complex high-dimensional locomotion tasks with the MuJoCo simulator (136). Most of the settings are based on (31) for fair comparison. We consider two

simulated robots (HalfCheetah and Walker2d) and two types of task environments (to run in a forward/backward direction or a particular velocity). The network architecture is two hidden layers of size 100 with ReLU activations for both. We used the standard linear feature baseline estimator. We evaluated the performance after one policy gradient step with 20 trajectories. We compare against MAML-TRPO and MAML-PPO. In the original MAML, TRPO (122) was used as the outer loop optimizer but we found out that using PPO (123) consistently outperformed the TRPO. MAML-PPO is also computationally more efficient since MAML-TRPO requires third-order gradients (or computed by hessian-vector product instead). To the best of our knowledge, MAML-PPO has not been tested on this setup. We evaluated two variations of meta-curvature similar to the classification setup, MC1 and MC2, and used PPO as the meta-optimizer.

4.4.5.2 Experimental results

Fig. 4.6 shows the rewards obtained after one step policy gradient update. In the HalfCheetahDir experiment, our methods outperformed both strong baselines. MC1-PPO reached the same performance of a strong baseline, MAML-PPO three times faster. In HalfCheetahVel and Walker2dDir experiments, both MC2-PPO and MAML-PPO reached nearly the same performance, but in a more sample efficient manner. For Walker2dVel, MAML-TRPO showed the fastest convergence at the earlier meta-training stage, but our meta-curvatures outperformed eventually. In this setting, most of the rewards come from the survival reward (the agent gets 1.0 reward for every step if they do not fall over). All methods were able to survive throughout the episode, but our methods run better at a given velocity. One thing we noticed that it stops obtaining more rewards and starts to degrade the performance in Walker2dDir experiment. The recently proposed approach (114) may alleviate this issue through better credit assignment in the meta-gradients. Combining it would be interesting direction to be explored.

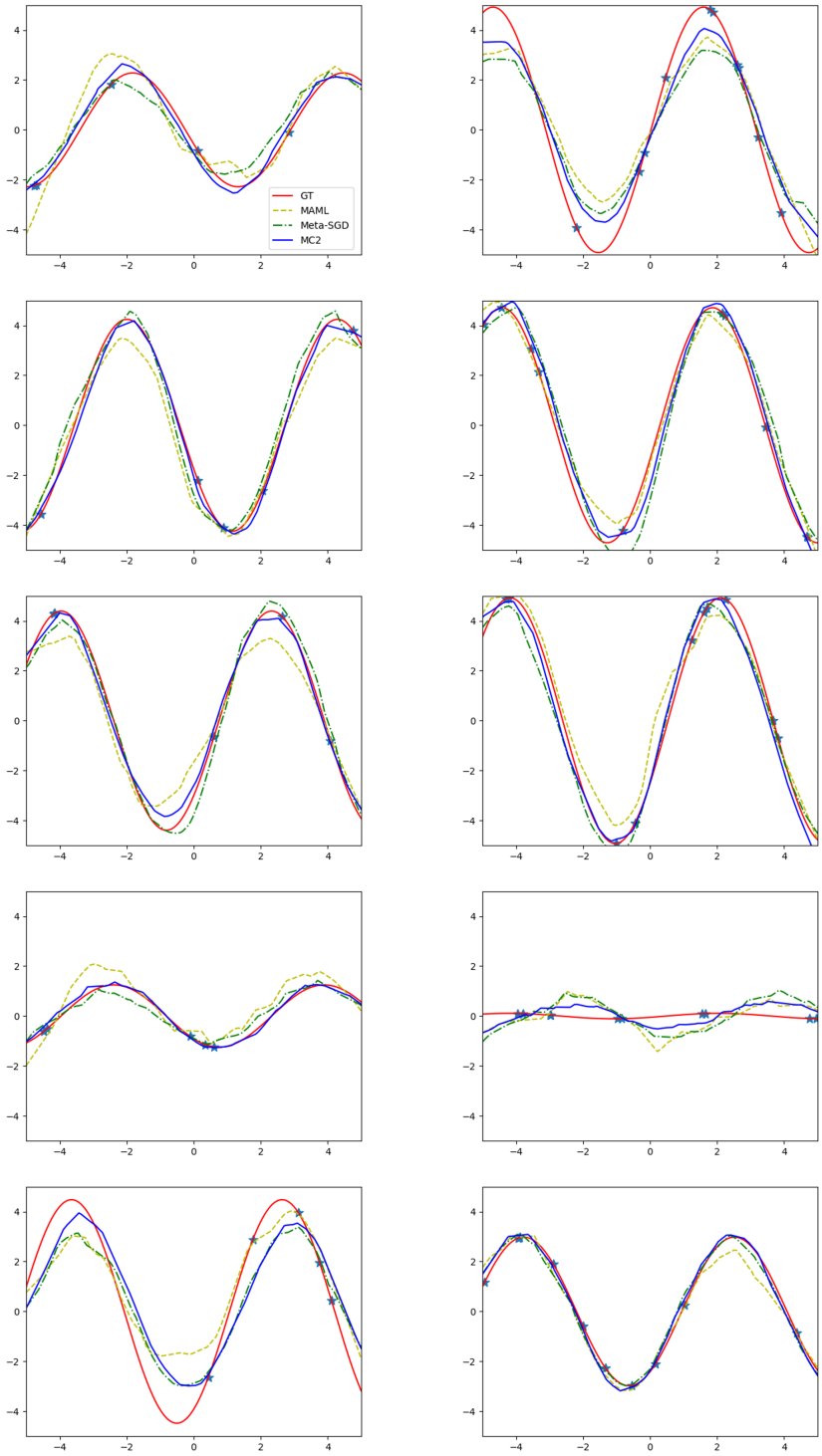


Figure 4.2: Qualitative results of few-shot regression on sinusoidal functions. The left column - 5 shot, The right column - 10 shot

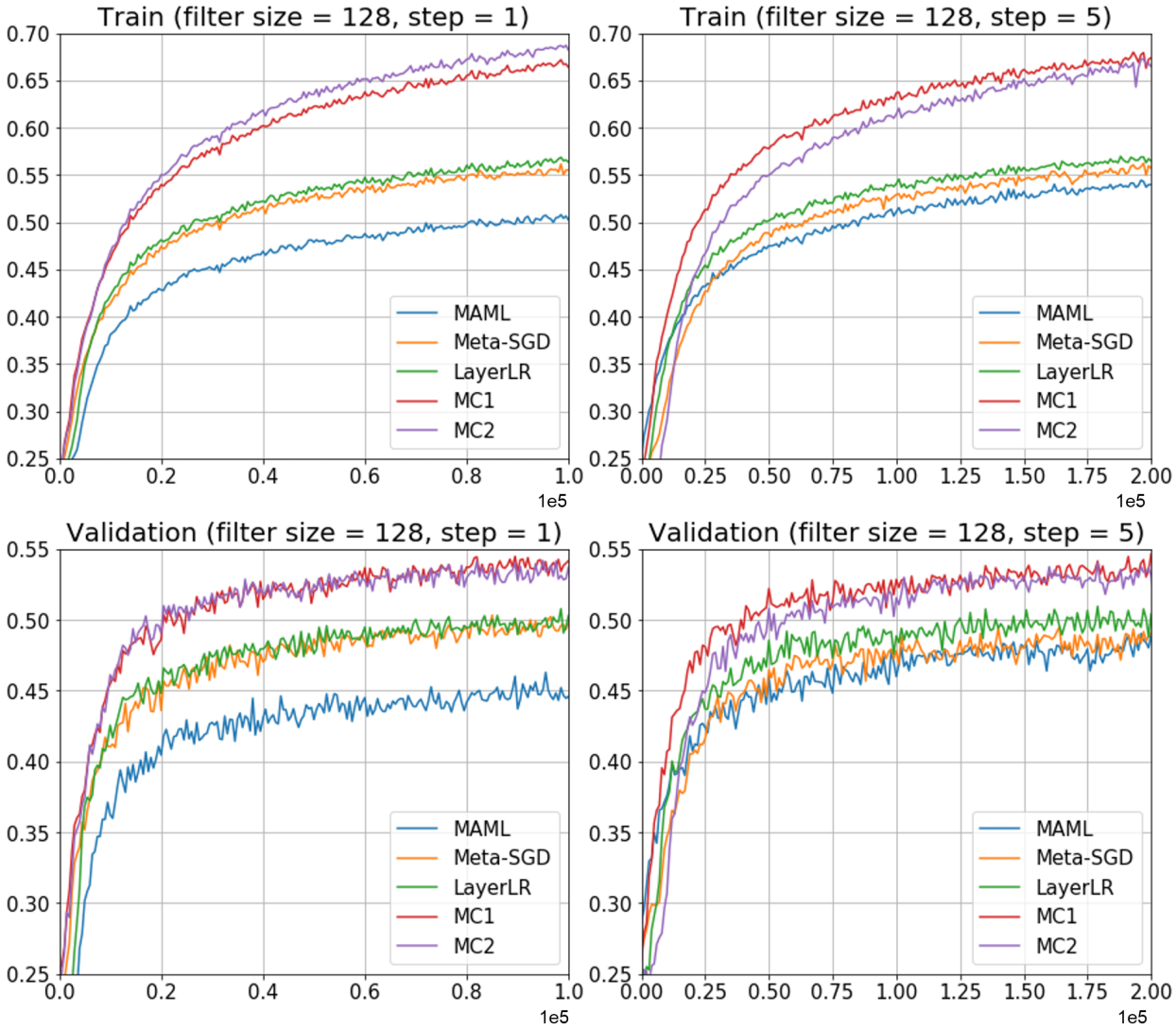


Figure 4.3: Experimental results of one-shot classification. Top row - training accuracy after the model update (1 or 5 steps). Bottom row - validation accuracy after the model update (1 or 5 steps). Y-axis: accuracy. X-axis: meta-training iterations.

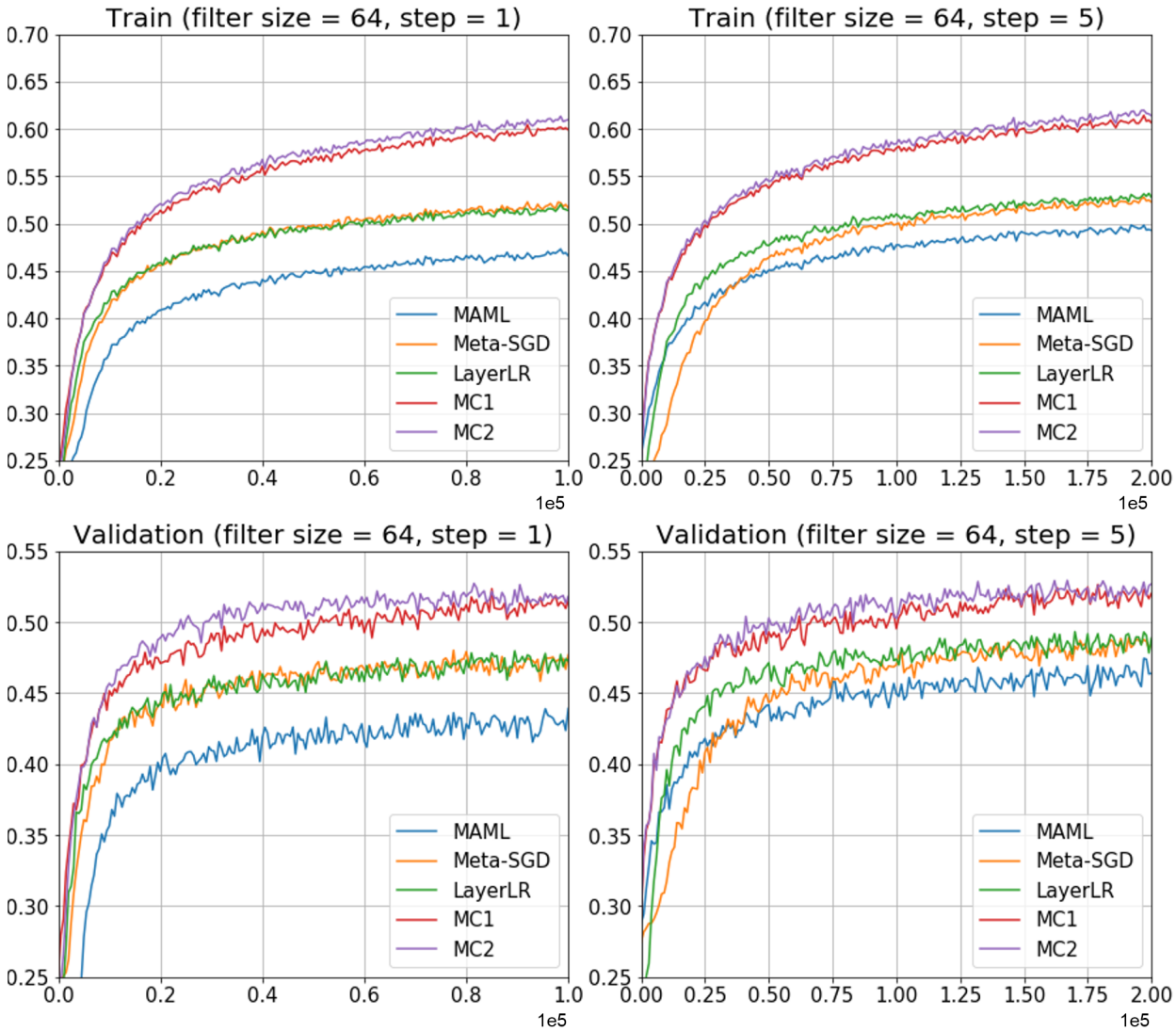


Figure 4.4: Experimental results of one-shot classification. Top row - training accuracy after the model update (1 or 5 steps). Bottom row - validation accuracy after the model update (1 or 5 steps). Y-axis: accuracy. X-axis: meta-training iterations.

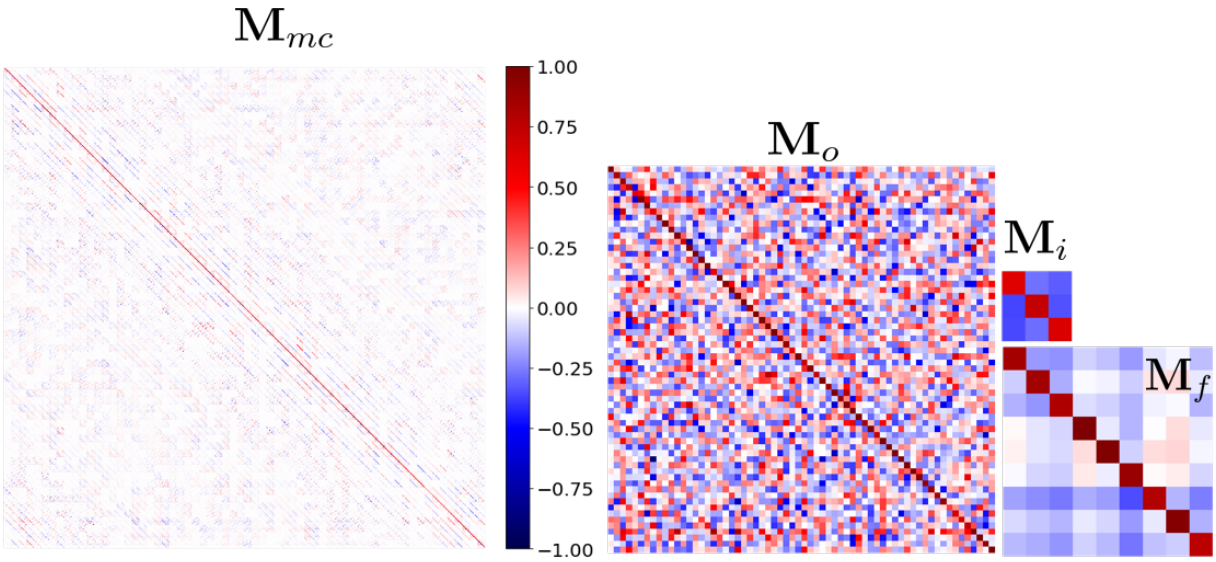


Figure 4.5: Visualization of meta-curvature matrices. We clipped the values $[-1, 1]$ for better visualization (Best viewed in color)

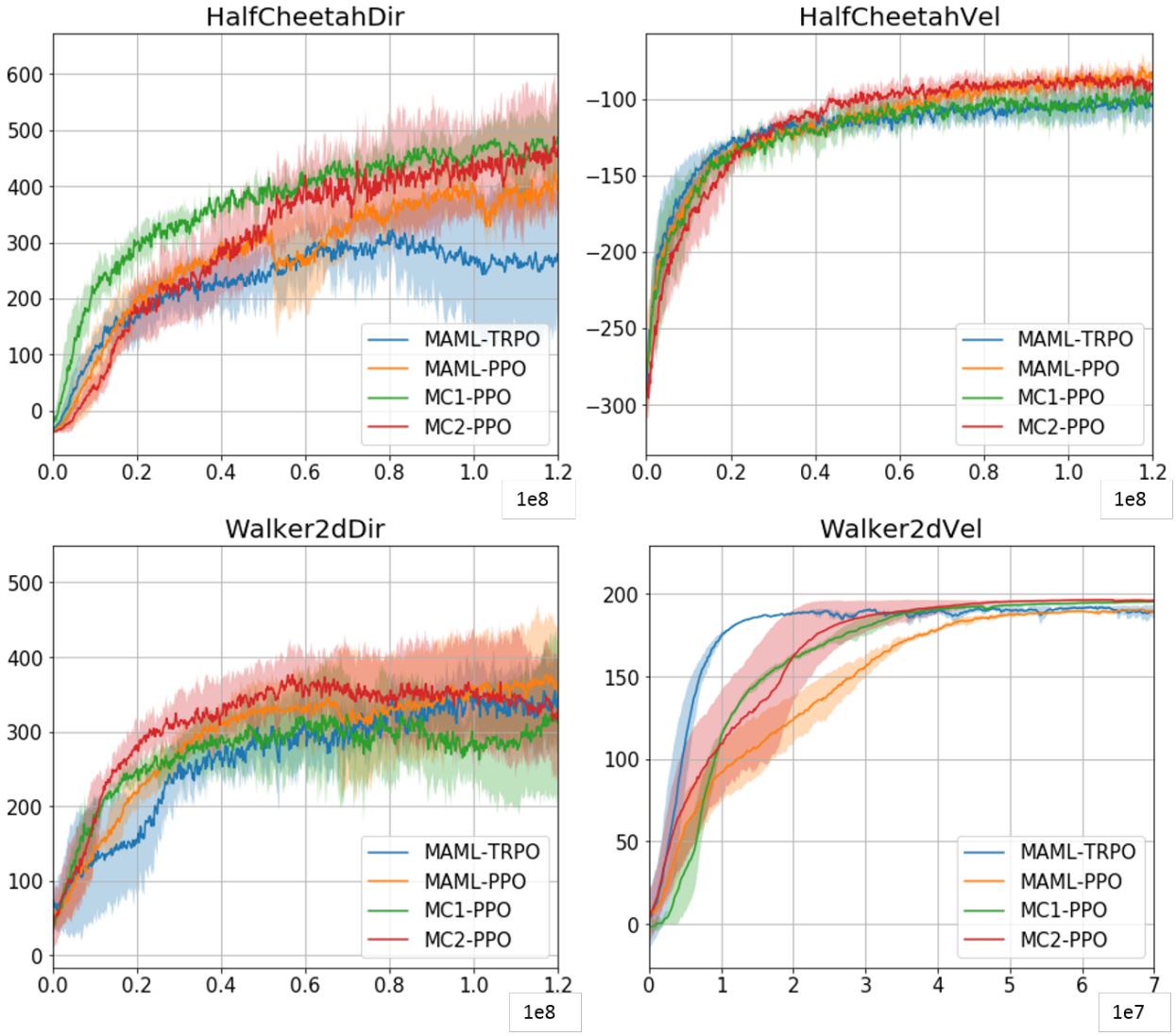


Figure 4.6: Reinforcement learning experimental results. Y-axis: rewards after the model updates. X-axis: meta-training steps. We performed at least three runs with random seeds and the curves are averaged over them.

CHAPTER 5: LEARNING TO GENERATE GIVEN FEW EXAMPLES

In this chapter, we explore generative models given few examples. We consider the problem of novel 3D view synthesis—given a single view of an object in an arbitrary pose, the goal is to synthesize an image of the object after a specified transformation of viewpoint. It has a variety of practical applications in computer vision, graphics, and robotics. As an image-based rendering technique (63), it allows placing a virtual object on a background with a desired pose or manipulating virtual objects in the scene (64). Also, multiple generated 2D views form an efficient representation for 3D reconstruction (133). In robotics, synthesized novel views give the robot a better understanding of unseen parts of the object through 3D reconstruction, which will be helpful for grasp planning (142).

This problem is generally challenging due to unspecified input viewing angle and the ambiguities of 3D shape observed in only a single view. In particular inferring the appearances of unobserved parts of the object that are not visible in the input view is necessary for novel view synthesis. Our approach attacks all of these challenges, but our contributions focus on the later aspect, dealing with disoccluded appearance in novel views and outputting highly-detailed synthetic images.

Given the eventual approach we will take, using a carefully constructed deep network, we can consider related work on dense prediction with encoder-decoder methods to see what makes the structure of the novel 3D view synthesis problem different. In particular, there is a lack of pixel-to-pixel correspondences between the input and output view. This, combined with large chunks of missing data due to occlusion, makes novel view synthesis fundamentally different than other dense prediction or generation tasks that have shown promising results with deep networks (100; 25; 60). Although the input and desired output views may have similar low-level

image statistics, enforcing such constraints directly is difficult. For example, skip or residual connections, are not immediately applicable as the input and output have significantly different global shapes. Hence, previous 3D novel view synthesis approaches (151; 133) have not been able to match the visual quality of geometry-based methods that exploit strong correspondence.

The geometry-based methods are an alternative to pure generation, and have been demonstrated in (53; 64; 110). Such approaches estimate the underlying 3D structure of the object and apply geometric transformation to pixels in the input (e.g. performing depth-estimation followed by 3D transformation of each pixel (40)). When successful, geometric transformation approaches can very accurately transfer original colors, textures, and local features to corresponding new locations in the target view. However, such approaches are fundamentally unable to hallucinate where new parts are revealed due to disocclusion. Furthermore, even for the visible geometry precisely estimating the 3D shape or equivalently the precise pixel-to-pixel correspondence between input and synthesized view is still challenging and failures can result in distorted output images.

In order to bring some of the power of explicit correspondence to deep-learning-based generation of novel views, the recent appearance flow network (AFN) (157) trains a convolutional encoder-decoder to learn how to move pixels without requiring explicit access to the underlying 3D geometry. Our work goes further in order to integrate more explicit reasoning about 3D transformation, hallucinate missing sections, and clean-up the final generated image producing significant improvements of realism, accuracy, and detail for synthesized views.

To achieve this we present *a holistic approach to novel view synthesis by grounding the generation process on viewpoint transformation*. Our approach first predicts the transformation of existing pixels from the input view to the view to be synthesized, as well as a visibility map, exploiting the learned view dependency. We use the transformation result matted with the predicted visibility map to condition the generation process. The image generator not only hallucinates the missing parts but also refines regions that suffer from distortion or unrealistic details due to the imperfect transformation prediction. This holistic pipeline alleviates some difficulties in novel view synthesis by explicitly using transformation for the parts where there are strong cues.

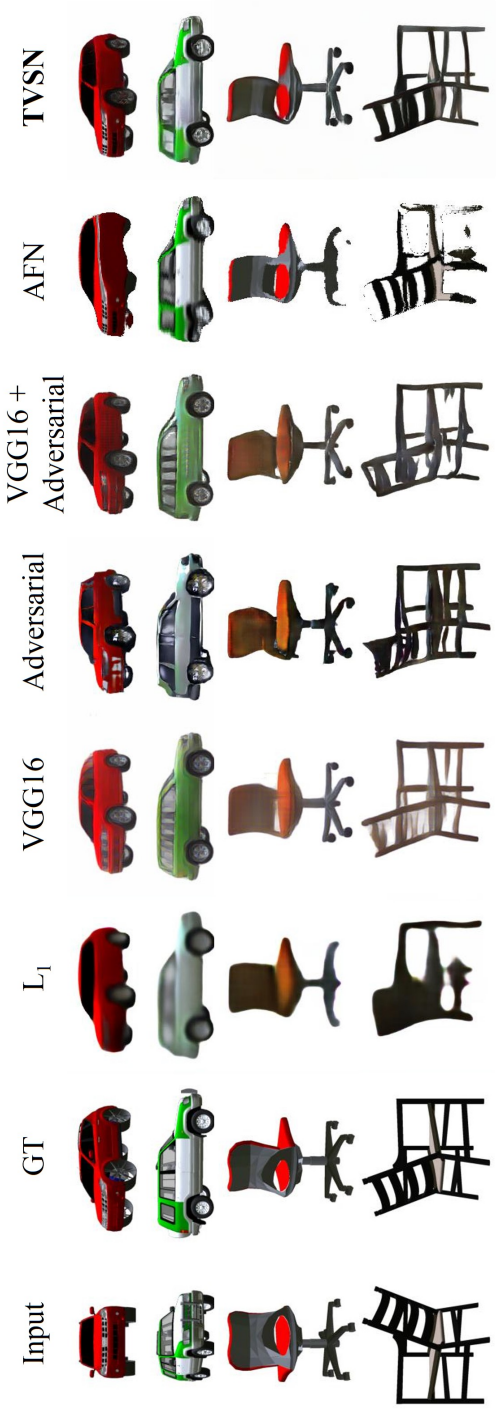


Figure 5.1: Results on test images from 3D ShapeNet dataset (15). 1st-input, 2nd-ground truth. From 3rd to 6th are deep encoder-decoder networks with different losses. (3rd-L_1 norm (133), 4th-feature reconstruction loss with pretrained VGG16 network (60; 74; 137; 73), 5th-adversarial loss with feature matching (41; 108; 118; 24), 6th-the combined loss). 7th-appearance flow network (AFN) (157). **8th-ours(TVSN)**.

We propose an architecture composed of two consecutive convolutional encoder-decoder networks. First, we introduce a disocclusion aware appearance flow network (DOAFN) to predict the visibility map and the intermediate transformation result. Our second encoder-decoder network is an image completion network which takes the matted transformation as an input and completes and refines the novel view with a combined adversarial and feature-reconstruction loss. A wide range of experiments on synthetic and real images show that the proposed technique achieves significant improvement compared to existing methods.

Our main contributions are:

- We propose a holistic image generation pipeline that explicitly predicts how pixels from the input will be transformed and *where there is disocclusion* in the output that needs to be filled, converting the remaining synthesis problem into one of image completion and repair.
- We design a disocclusion aware appearance flow network that relocates existing pixels in the input view along with predicting a visibility map.
- We show that using loss networks with a term considering how well recognition-style features are reconstructed, combined with L_1 loss on pixel values during training, improves synthesized image quality and detail.

5.1 Transformation-Grounded View Synthesis

Novel view synthesis could be seen as a combination of the following three scenarios: 1) pixels in the input view that remain visible in the target view are moved to their corresponding positions; 2) remaining pixels in the input view disappear due to occlusions; and 3) previously unseen pixels are revealed or disoccluded in the target view. We replicate this process via a neural network as shown in Figure 5.2. Specifically, we propose a disocclusion-aware appearance flow network (5.1.1) to transform the pixels of the input view that remain visible. A subsequent generative completion network (5.1.2) then hallucinates the unseen pixels of the target view given these transformed pixels.

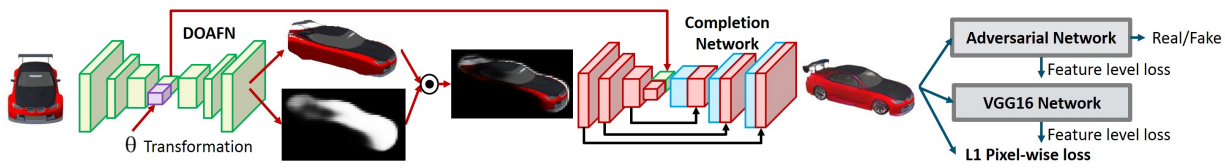


Figure 5.2: Transformation-grounded view synthesis network(TVSN). Given an input image and a target transformation (5.1.1), our disocclusion-aware appearance flow network (DOAFN) transforms the input view by relocating pixels that are visible both in the input and target view. The image completion network, then, performs hallucination and refinement on this intermediate result(5.1.2). For training, the final output is also fed into two different loss networks in order to measure similarity against ground truth target view (5.1.2.1).

5.1.1 Disocclusion-aware Appearance Flow Network

Recently proposed appearance flow network (AFN) (157) learns how to move pixels from an input to a target view. The key component of the AFN is a differentiable image sampling layer introduced in (56). Precisely, the network first predicts a dense flow field that maps the pixels in the target view, I_{t}, to the source image, I_{s}. Then, sampling kernels are applied to get the pixel value for each spatial location in I_{t}. Using a bilinear sampling kernel, the output pixel value at spatial location $I_{\mathrm{t}}^{i,j}$ equals to:

$$\sum_{(h,w)\in N} I_{\mathrm{s}}^{h,w} \max(0, 1 - |F_y^{i,j} - h|) \max(0, 1 - |F_x^{i,j} - w|), \tag{5.1}$$

where F is the flow predicted by the deep convolutional encoder-decoder network (see the first half of Figure 5.2). $F_x^{i,j}$ and $F_y^{i,j}$ indicate the x and y coordinates of one target location. N denotes the 4-pixel neighborhood of $(F_y^{i,j}, F_x^{i,j})$.

The key difference between our disocclusion aware appearance flow network (DOAFN) and the AFN is the prediction of an additional visibility map which encodes the parts that need to be removed due to occlusion. The original AFN synthesizes the entire target view, including the disoccluded parts, with pixels of the input view, e.g. 1st row of AFN results in Figure 5.1. However, such disoccluded parts might get filled with wrong content, resulting in implausible results, especially for cases where a large portion of the output view is not seen in the input view.

Figure 5.3: Visibility maps for different rotations: the first column in the first row is an input image. Remaining columns show output images and corresponding masks for rotations from 20 to 340 degrees in 20 degree intervals. The second, third and fourth rows show visibility maps M_{vis}, symmetry-aware visibility maps $M_{\text{s-vis}}$, and background masks M_{bg}, respectively. The input image is in the pose of 0 elevation and 20 azimuth. The visibility maps for the rotations from 160 to 340 show the largest difference between M_{vis} and $M_{\text{s-vis}}$. For example, $M_{\text{s-vis}}$ shows the opposite side of the car as visible and allows it to be filled in by the network based on the visible side.

Such imperfect results would provide misleading information to a successive image generation network. Motivated by this observation, we propose to predict a visibility map that masks such problematic regions in the transformed image:

$$I_{\text{doafn}} = I_{\text{afn}} \odot M_{\text{vis}}, \tag{5.2}$$

where $M_{\text{vis}} \in [0,1]^{H \times W}$. To achieve this, we define the ground truth visibility maps according to the 3D object geometry as described next.

5.1.1.1 Visibility map

Let $M_{\text{vis}} \in \mathbb{R}^{H \times W}$ be the visibility map for the target view, given source image I_{s} and desired transformation parameter θ. The mapping value for a pixel in the target view corresponding to a spatial location (i, j) in I_{s} is defined as follows:

$$M_{\text{vis}}^{(PR(\theta)\mathbf{x}_{\text{s}}^{(i,j)})^h,(PR(\theta)\mathbf{x}_{\text{s}}^{(i,j)})^w} = \begin{cases} 1 & \mathbf{c}^\top R(\theta)\mathbf{n}_{\text{s}}^{(i,j)} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

$\mathbf{x}_{\text{s}}^{(i,j)} \in \mathbb{R}^4$ is the 3D object coordinates and $\mathbf{n}_{\text{s}}^{(i,j)} \in \mathbb{R}^4$ is the surface normal corresponding to location (i, j) in I_{s}, both represented in homogeneous coordinates. Since we use synthetic render-

ings of 3D CAD models, we have access to ground truth object coordinates and surface normals. $R(\theta) \in \mathbb{R}^{3\times4}$ is the rotation matrix given the transformation parameter θ and $P \in \mathbb{R}^{3\times3}$ is the perspective projection matrix. The superscripts h and w denote the target image coordinates in y and x axis respectively after perspective projection. $\mathbf{c} \in \mathbb{R}^3$ is the 3D camera center. In order to compute the target image coordinates for each pixel in I_{s}, we first obtain the 3D object coordinates corresponding to this pixel and then apply the desired 3D transformation and perspective projection. The mapping value of the target image coordinate is 1 if and only if the dot product between the viewing vector and surface normal is positive, i.e. the corresponding 3D point is pointing towards the camera.

5.1.1.2 Symmetry-aware visibility map

Many common object categories exhibit reflectional symmetry, e.g. cars, chairs, tables etc. AFN implicitly exploits this characteristic to ease the synthesis of large viewpoint changes. To fully take advantage of symmetry in our DOAFN, we propose to use a symmetry-aware visibility map. Assuming that objects are symmetric with respect to the xy-plane, a symmetry-aware visibility map M_{sym} is computed by applying Equation 5.3 to the z-flipped object coordinates and surface normals. The final mapping for a pixel in the target view corresponding to spatial location (i, j) is then defined as:

$$M^{i,j}_{\text{s-vis}} = \mathbb{1}\left[M^{i,j}_{\text{sym}} + M^{i,j}_{\text{vis}} > 0\right] \tag{5.4}$$

5.1.1.3 Background mask

Explicit decoupling of the foreground object is necessary to deal with real images with natural background. In addition to parts of the object being disoccluded in the target view, different views of the object occlude different portions of the background posing additional challenges. For example, transforming a side view to be frontal exposes parts of the background occluded by the two ends of the car. In our approach, we define the *foreground* as the region that covers pixels of the object in both input view and output view. The rest of the image belongs to the *background*

and should remain unchanged in both views. We thus introduce a unified background mask,

$$M_{\text{bg}}^{i,j} = \mathbb{1}\big[B_{\text{s}}^{i,j} + B_{\text{t}}^{i,j} > 0\big], \tag{5.5}$$

where B_{s} and B_{t} are the background masks of the source and target images respectively. Ground truth background masks are easily obtained from 3D models. Examples of background masks are presented in Figure 5.3.

When integrated with the (symmetry-aware) visibility map, the final output of DOAFN becomes:

$$I_{\text{doafn}} = I_{\text{s}} \odot M_{\text{bg}} + I_{\text{afn}} \odot M_{\text{s-vis}} \tag{5.6}$$

5.1.2 View Completion Network

Traditional image completion or hole filling methods often exploit local image information (29; 11; 147) and have shown impressive results for filling small holes or texture synthesis. In our setting, however, sometimes more than half of the content in the novel view is not visible in the input image, constituting a big challenge for local patch based methods. To address this challenge, we propose another encoder-decoder network, capable of utilizing both local and global context, to complete the transformed view inferred by DOAFN.

Our view completion network is composed of an "hourglass" architecture similar to (97), with a bottleneck-to-bottleneck identity mapping layer from DOAFN to the hourglass (see Figure 5.2). This network has three essential characteristics. First, being conditioned on the high-level features of DOFAN, it can generate content that have consistent attributes with the given input view, especially when large chunk of pixels are dis-occluded. Second, the output of DOAFN is already in the desired viewpoint with important low-level information, such as colors and local textures, preserved under transformation. Thus, it is possible to utilize skip connections to propagate this low-level information from the encoder directly to later layers of the decoder. Third, the view completion network not only hallucinates disoccluded regions but also fixes artifacts such as

distortions or unrealistic details. The output quality of DOAFN heavily depends on the input viewpoint and desired transformation, resulting in imperfect flow in certain cases. The encoder-decoder nature of the image generation network is well-suited to fix such cases. Precisely, while the encoder is capable of recognizing undesired parts in the DOAFN output, the decoder refines these parts with realistic content.

5.1.2.1 Loss networks

The idea of using deep networks as a loss function for image generation has been proposed in (74; 137; 60; 24). Precisely, an image generated by a network is passed as an input to an accompanied network which evaluates the discrepancy (the feature distance) between the generation result and ground truth. We use the *VGG16* network for calculating the feature reconstruction losses from a number of layers, which is referred as *perceptual loss*.

We tried both a pre-trained loss network and a network with random weights as suggested in (47; 138). However, we got perceptually poor results with random weights, concluding that the weights of the loss network indeed matter.

On the other hand, adversarial training (41) has been phenomenally successful for training the loss network at the same time of training the image generation network. We experimented with a similar adversarial loss network as in (108) while adopting the idea of feature matching presented in (118) to make the training process more stable.

We realized that the characteristics of generated images with these two kinds of loss networks, perceptual and adversarial, are complementary. Thus, we combined them together with the standard image reconstruction loss (L_1) to maximize performance. Finally, we added total variation regularization term (60), which was useful to refine the image:

$$-\log D(G(I_{\mathrm{s}})) + \alpha L_2(F_D(G(I_{\mathrm{s}})), F_D(I_{\mathrm{t}}))+ \\ \beta L_2(F_{vgg}(G(I_{\mathrm{s}})), F_{vgg}(I_{\mathrm{t}})) + \gamma L_1(I_{\mathrm{s}}, I_{\mathrm{t}}) + \lambda L_{TV}(G(I_{\mathrm{s}})) \quad (5.7)$$

I_{s}, $G(I_{\mathrm{s}})$ and I_{t} is the input, generated output and corresponding target image, respectively. $\log(D)$ is log likelihood of generated image $G(I_{\mathrm{s}})$ being a real image, estimated by adversarially trained loss network, called discriminator D. In practice, minimizing $-\log D(G(I_{\mathrm{s}}))$ has shown better gradient behaviour than minimizing $\log D(1 - G(I_{\mathrm{s}}))$.

F_D and F_{vgg} are the features extracted from the discriminator and *VGG16* loss networks respectively. We found that concatenated features from the first to the third convolutional layers are the most effective. L_1 and L_2 are ℓ_1 and ℓ_2 norms of two same size inputs divided by the size of the inputs. In sum, both generated images $G(I_{\mathrm{s}})$ and ground truth image I_{t} are fed into D and *VGG16* loss networks, and we extract the features, and compute averaged euclidean distance between these two.

The discriminator D is simultaneously trained along with G via alternative optimization scheme proposed in (41). The loss function for the discriminator is

$$-\log D(I_{\mathrm{s}}) - \log(1 - D(G(I_{\mathrm{s}}))) \tag{5.8}$$

We empirically found that $\alpha = 100$, $\beta = 0.001$, $\gamma = 1$, and $\lambda = 0.0001$ are good hyper-parameters and fixed them for the entire experiments.

5.2 Experiments

5.2.1 Training Setup

We use rendered images of 3D models from ShapeNet (15) both for training and testing. We use the entire *car* category (7497 models) and a subset of the *chair* category (698 models) with sufficient texture. For each model, we render images from a total of 54 viewpoints corresponding to 3 different elevations (0, 10, and 20) and 18 azimuth angles (sampled in the range $[0, 340]$ with 20-degree increments). The desired transformation is encoded as a 17-D one-hot vector corresponding to one of the rotation angles between input and output views in the range $[20, 340]$. Note that we did not encode 0 degree as it is the identical mapping. For each category, 80% of

Table 5.1: We compare our method (*TVSN(DOAFN)*) to several baselines: (i) a single-stage encoder-decoder network trained with different loss functions: L_1 (L_1), feature reconstruction loss using VGG16 (*VGG16*), adversarial (*Adv*), and combination of the latter two (*VGG16+Adv*), (ii) a variant of our approach that does not use a visibility map (*TVSN(AFN)*).

	car		chair	
	L_1	SSIM	L_1	SSIM
L_1(133)	.168	.884	.248	**.895**
VGG	.228	.870	.283	**.895**
Adv	.208	.865	.241	.885
VGG+Adv	.194	.872	.242	.888
AFN(157)	.146	.906	.240	.891
TVSN(AFN)	**.132**	**.910**	**.229**	**.895**
TVSN(DOAFN)	**.133**	**.910**	**.230**	**.894**

3D models are used for training, which leaves over 5 million training pairs (input view-desired transformation) for the car category and 0.5 million for the chair category. We randomly sample input viewpoints, desired transformations from the rest 20% of 3D models to generate a total of $20,000$ testing instances for each category. Both input and output images are of size $256 \times 256 \times 3$.

We first train DOAFN, and then the view completion network while DOAFN is fixed. After the completion network fully converges, we fine-tune both networks end-to-end. However, this last fine-tuning stage does not show notable improvements. We use mini-batches of size 25 and 15 for DOAFN and the completion network respectively. The learning rate is initialized as 10^{-4} and is reduced to 10^{-5} after 10^5 iterations. For adversarial training, we adjust the update schedule (two iterations for generator and one iteration for discriminator in one cycle) to balance the discriminator and the generator.

5.2.2 Results

We discuss our main findings in the rest of this section and refer the reader to the supplementary material for more results. We utilize the standard L_1 mean pixel-wise error and the structural similarity index measure (SSIM) (146; 86) for evaluation. When computing the L_1 error, we normalize the pixel values resulting in errors in the range $[0, 1]$, lower numbers corresponding to

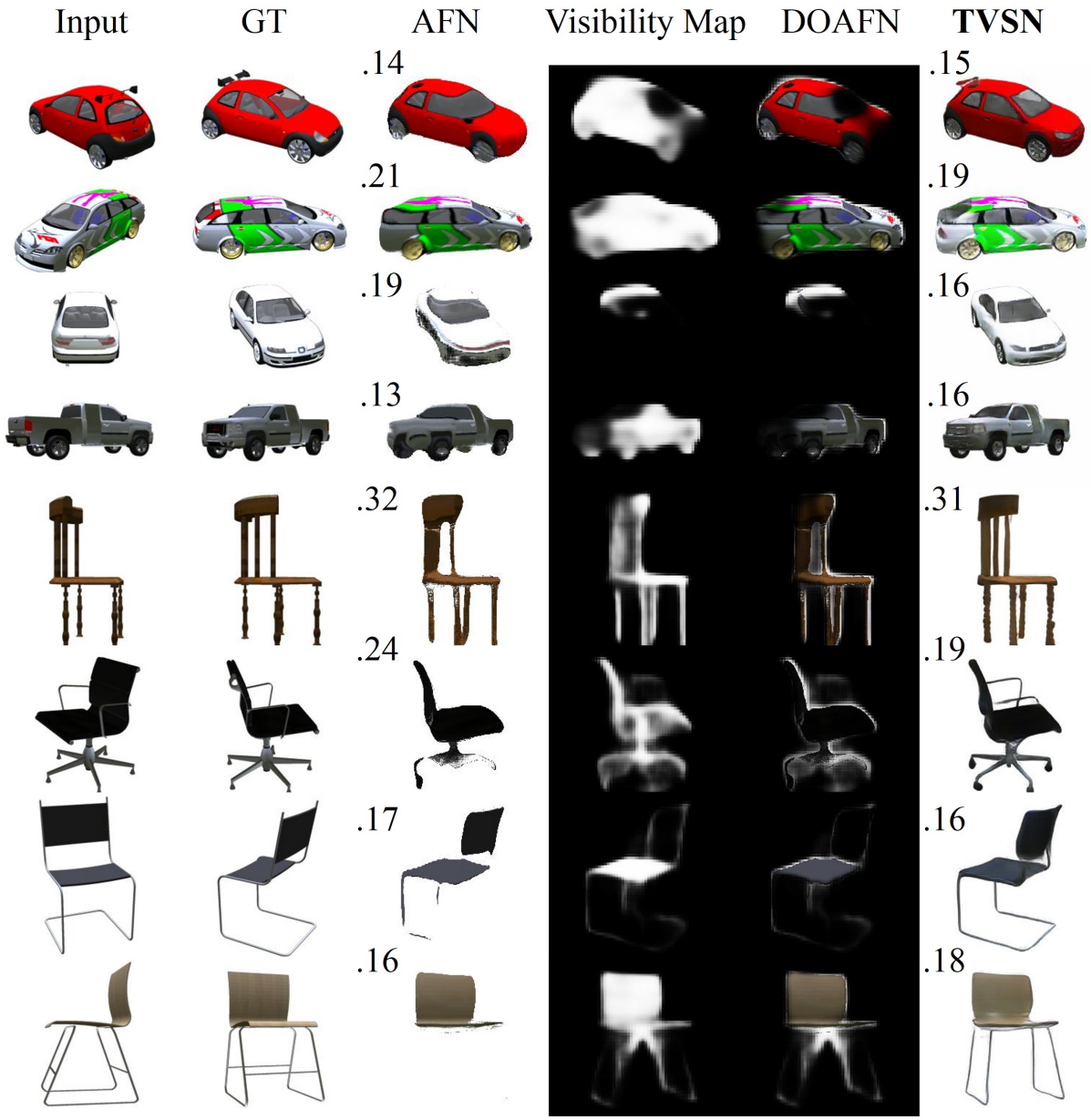


Figure 5.4: Results on synthetic data from ShapeNet. We show the input, ground truth output (GT), results for AFN and our method (TVSN) along with the L_1 error. We also provide the intermediate output (visibility map and output of DOAFN).

better results. SSIM is in the range $[-1, 1]$ where higher values indicate more structural similarity.

5.2.2.1 Comparisons

We first evaluate our approach on synthetic data and compare to AFN. Figure 5.4 shows qualitative results.[1] We note that while our method completes the disoccluded parts consistently with the input view, AFN generates unrealistic content (front and rear parts of the cars in the 1st and 2nd rows). Our method also corrects geometric distortions induced by AFN (3rd and 4th rows) and better captures the lighting (2nd row). For the chair category, AFN often fails to generate thin structures such as legs due to the small number of pixels in these regions contributing to the loss function. On the other hand, both perceptual and adversarial loss help to complete the missing legs as they contribute significantly to the perception of the overall shape. In order to evaluate the importance of the visibility map, we compare against a variant of our approach which directly provides the output of AFN to the view completion network without masking. (For clarity, we will refer to our method as *TVSN(DOAFN)* and to this baseline as *TVSN(AFN)*.) Furthermore, we also implement a single-stage convolutional encoder-decoder network as proposed in (133) and train it with various loss functions: L_1 loss (L_1), feature reconstruction loss using VGG16 (VGG16), adversarial loss (Adv), and combination of the latter two (VGG16+Adv). We provide quantitative and visual results in Table 5.1 and Figure 5.1 respectively. We note that, although commonly used, L_1 and SSIM metrics are not fully correlated with human perception. While our method is clearly better than the L_1 baseline (133), both methods get comparable SSIM scores.

We observe that both TVSN(AFN) and TVSN(DOAFN) perform similarly with respect to L_1 and SSIM metrics demonstrating that the view completion network in general successfully refines the output of AFN. However, in certain cases severe artifacts observed in the AFN output, especially in the disoccluded parts, get smoothly integrated in the completion results as shown

[1] The results from the original AFN (157) paper are not directly comparable due to the different image size. In addition, since the complete source code was not available at the time of paper submission, we re-implemented this method by consulting the authors.

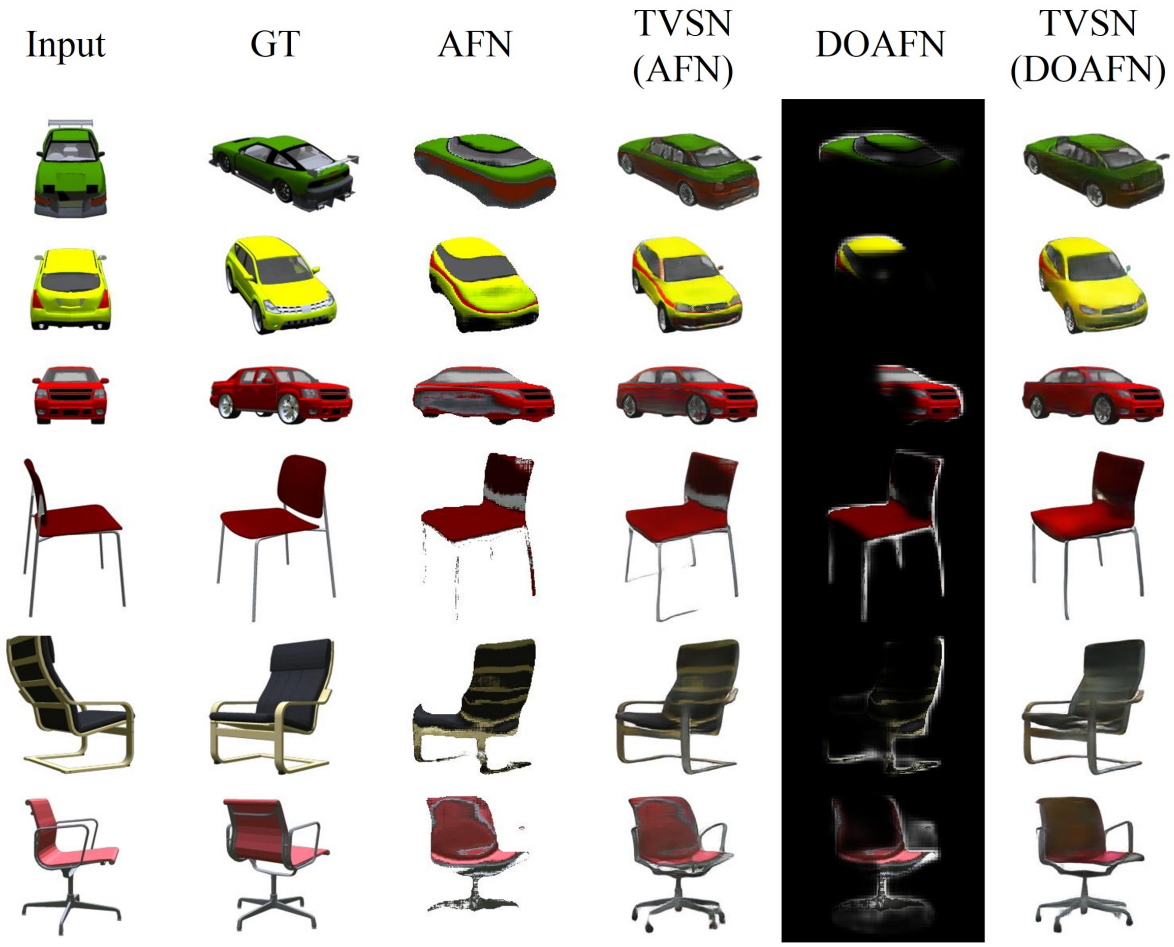


Figure 5.5: When a visibility map is not utilized (TVSN(AFN)), severe artifacts observed in the AFN output get integrated into the final results. By masking out such artifacts, our method (TVSN(DOAFN)) relies purely on the view completion network to generate plausible results.

in Figure 5.5. In contrast, the visibility map masks out those artifacts and thus TVSN(DOAFN) relies completely on the view completion network to hallucinate these parts in a realistic and consistent manner.

5.2.2.2 Evaluation of the Loss Networks

We train our network utilizing the feature reconstruction loss of VGG16 and the adversarial loss. We evaluate the effect of each loss by training our network with each of them only and provide visual results in Figure 5.6. It is well-known that the adversarial loss is effective in gen-

Figure 5.6: We evaluate the effect of using only parts of our system, VGG16 in TVSN(VGG16), and adversarial loss in TVSN(Adversarial), as opposed to our method, TVSN(VGG16+Adversarial) that uses both.

erating realistic and sharp images as opposed to standard pixel-wise loss functions. However, some artifacts such as colors and details inconsistent with the input view are still observed. For the VGG16 loss, we experimented with different feature choices and empirically found that the combination of the features from the first three layers with total variation regularization is the most effective. Although the VGG16 perceptual loss is capable of generating high quality images for low-level tasks such as super-resolution, it has not yet been fully explored for pure image generation tasks as required for hallucinating disoccluded parts. Thus, this loss still suffers from the blurry output problem whereas combination of both VGG16 and adversarial losses results in the most effective configuration.

5.2.3 360 degree rotations and 3D reconstruction

Inferring 3D geometry of an object from a single image is the holy-grail of computer vision research. Recent approaches using deep networks commonly use a voxelized 3D reconstruction as output (17; 150). However, computational and spatial complexities of using such voxelized

representations in standard encoder-decoder networks significantly limits the output resolution, e.g. 32^3 or 64^3.

Inspired by (133), we exploit the capability of our method in generating novel views for reconstruction purposes. Specifically, we generate multiple novel views from the input image to cover a full 360 rotation around the object sampled at 20-degree intervals. We then run a multi-view reconstruction algorithm (35) on these images using the ground truth relative camera poses to obtain a dense point cloud. We use the open source OpenMVS library (ope) to reconstruct a textured mesh from this point cloud. Figure 5.7 shows multi-view images generated by AFN and our method whereas Figure 5.8 demonstrates examples of reconstructed 3D models from these images. By generating views consistent in terms of geometry and details, our method results in significantly better quality textured meshes.

5.2.4 3D Object Rotations in Real Images

In order to generalize our approach to handle real images, we generate training data by compositing synthetic renderings with random backgrounds (130). We pick $10,000$ random images from the SUN397 dataset(130) and randomly crop them to be of size $256 \times 256 \times 3$. Although this simple approach fails to generate realistic images, e.g. due to inconsistent lighting and viewpoint, it is effective in enabling the network to recognize the contours of the objects in complex background. In Figure 5.9, we show several novel view synthesis examples from real images obtained from the internet.

While our initial experiments show promising results, further investigation is necessary to improve performance. Most importantly, more advanced physically based rendering techniques are required to model complex light interactions in the real world (e.g. reflections from the environment onto the object surface). In addition, it is necessary to sample more viewpoints (both azimuth and elevation) to handle viewpoint variations in real data. Finally, to provide a seamless break from the original image, an object segmentation module is desirable so that the missing

pixels in background can be separately filled in by alternative methods, such as patch-based inpainting methods (11) or pixel-wise autoregressive models (141).

5.3 Related Work

5.3.1 Geometry-based view synthesis

A large body of work benefits from implicit or explicit geometric reasoning to address the novel view synthesis problem. When multiple images are available, multi-view stereo algorithms (35) are applicable to explicitly reconstruct the 3D scene which can then be utilized to synthesize novel views. An alternative approach recently proposed by Flynn et al. (34) uses deep networks to learn to directly interpolate between neighboring views. Ji et al. (59) propose to rectify the two view images first with estimated homography by deep networks, and then synthesize middle view images with another deep networks. In case of single input view, Garg et al. (40) propose to first predict a depth map and then synthesize the novel view by transforming each reconstructed 3D point in the depth map. However, all these approaches only utilize the information available in the input views and thus fail in case of disocclusion. Our method, on the other hand, not only takes advantage of implicit geometry estimation but also infers the parts of disocclusion.

Another line of geometry-based methods utilize large internet collections of 3D models which are shown to cover wide variety for certain real world object categories (64; 110). Given an input image, these methods first identify the most similar 3D model in a database and fit to the image either by 3D pose estimation (110) or manual interactive annotation (64). The 3D information is then utilized to synthesize novel views. While such methods generate high quality results when sufficiently similar 3D models exist, they are often limited by the variation of 3D models found in the database. In contrast, our approach utilizes 3D models only for training generation networks that directly synthesize novel views from an image.

5.3.2 Image generation networks

One of the first convolutional networks capable of generating realistic images of objects is proposed in (26), but the network requires explicitly factored representations of object type, viewpoint and color, and thus is not able to generalize to unseen objects. The problem of generating novel views of an object from a single image is addressed in (151; 71; 133) using deep convolutional encoder-decoder networks. Due to the challenges of disentangling the factors from single-view and the use of globally smooth pixel-wise similarity measures (e.g. L_1 or L_2 norm), the generation results tend to be blurry and low in resolution.

An alternative to learning disentangled or invariant factors is the use of equivariant representations, i.e. transformations of input data which facilitate downstream decision making. Transforming auto-encoders are coined by Hinton et al. (51) to learn both 2D and 3D transformations of simple objects. Spatial transformer networks (56) further introduce differentiable image sampling techniques to enable in-network parameter-free transformations. In the 3D case, flow fields are learned to transform input 3D mesh to the target shape (154) or input view to the desired output view (157). However, direct transformations are clearly upper-bounded by the input itself. To generate novel 3D views, our work grounds a generation network on the learned transformations to hallucinate disoccluded pixels.

Recently, a number of image generation methods introduce the idea of using pre-trained deep networks as loss function, referred as perceptual loss, to measure the feature similarities from multiple semantic levels (60; 74; 137; 73). The generation results from these works well preserve the object structure but are often accompanied with artifacts such as aliasing. At the same time, generative adversarial networks (41; 108), introduce a discriminator network, which is adversarially trained with the generator network to tell apart the generated images from the real ones. The discriminator encapsulates natural image statistics of all orders in a real/fake label, but its min-max training often leads to local minimum, and thus local distortions or painting-stroke effects are commonly observed in their generated images. Our work uses a combined loss

function that takes advantages of both the structure-preserving property of perceptual loss and the rich textures of adversarial loss (See Fig. 5.1).

Deep networks have also been explored for image completion purposes. Examples of proposed methods include image in-painting with deep networks (105) and sequential parts-by-parts generation for image completion (72). Such methods assume the given partial input is correct and focus only on completion. In our case, however, we do not have access to a perfect intermediate result. Instead, we rely on the generation network both to hallucinate missing regions and also refine any distortions that occur due to inaccurate per-pixel transformation prediction.

Figure 5.7: Results of 360 degree rotations

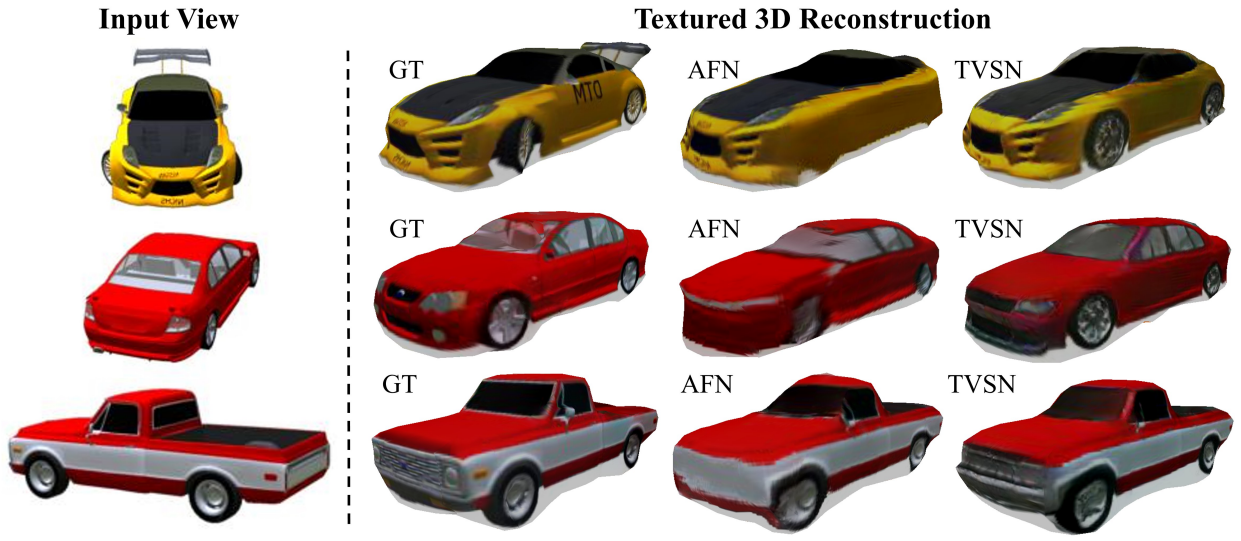


Figure 5.8: We run a multi-view stereo algorithm to generate textured 3D reconstructions from a set of images generated by AFN and our TVSN approach. We provide the reconstructions obtained from ground truth images (GT) for reference.

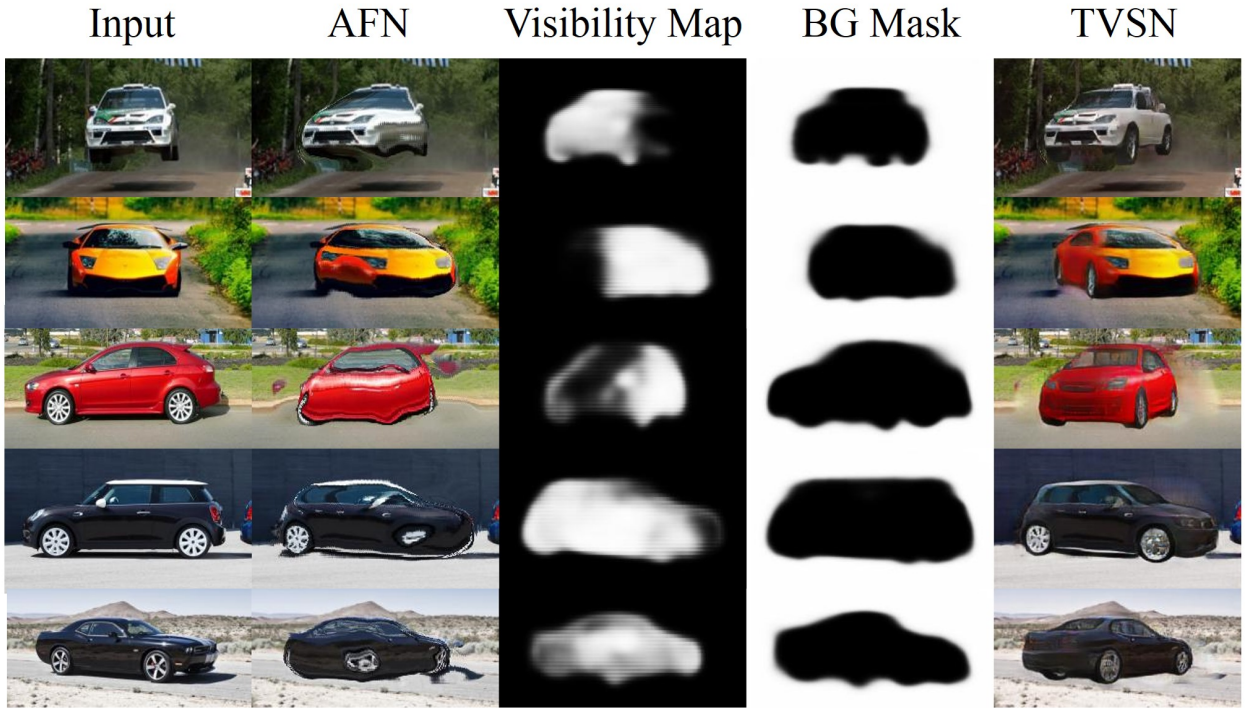


Figure 5.9: We show novel view synthesis results on real internet images along with the predicted visibility map and the background mask.

CHAPTER 6: APPENDICES

6.1 Meta-Trackers

6.1.1 More visualizations of response maps in MetaCREST

Figure 6.1.

6.1.2 Detailed results on VOT2016

We present detailed results of MetaCREST (Table 6.1 and 6.2) and MetaSDNet (Table 6.3 and 6.4) on VOT2016 dataset. Both accuracy and robustness table are generated from VOT2016 toolkit. For original CREST tracker, we could not get the same results as reported in their paper (the performance we could get is lower). In the main text, we reported the results from their paper and we omitted detailed results of CREST since they are not available. We provided other results, CREST-Base, CREST-10, CREST-05, CREST-03, and CREST-01.

6.1.3 Detailed results on OTB2015

We present detailed results of MetaCREST (Table 6.5 and 6.6) and MetaSDNet (Table 6.7 and 6.8) on OTB2015 dataset. It shows the results of individual sequences in success plots.

Figure 6.1: More visualizations of response maps in MetaCREST: Left three columns represents a cropped image centered on the target at the initial frame, response map with meta-learned initial correlation filters θ_0, response map after updating 1 iteration with meta-learned α, respectively. The rest of six columns on the right shows response maps of CREST after updating the model up to 10 iterations.

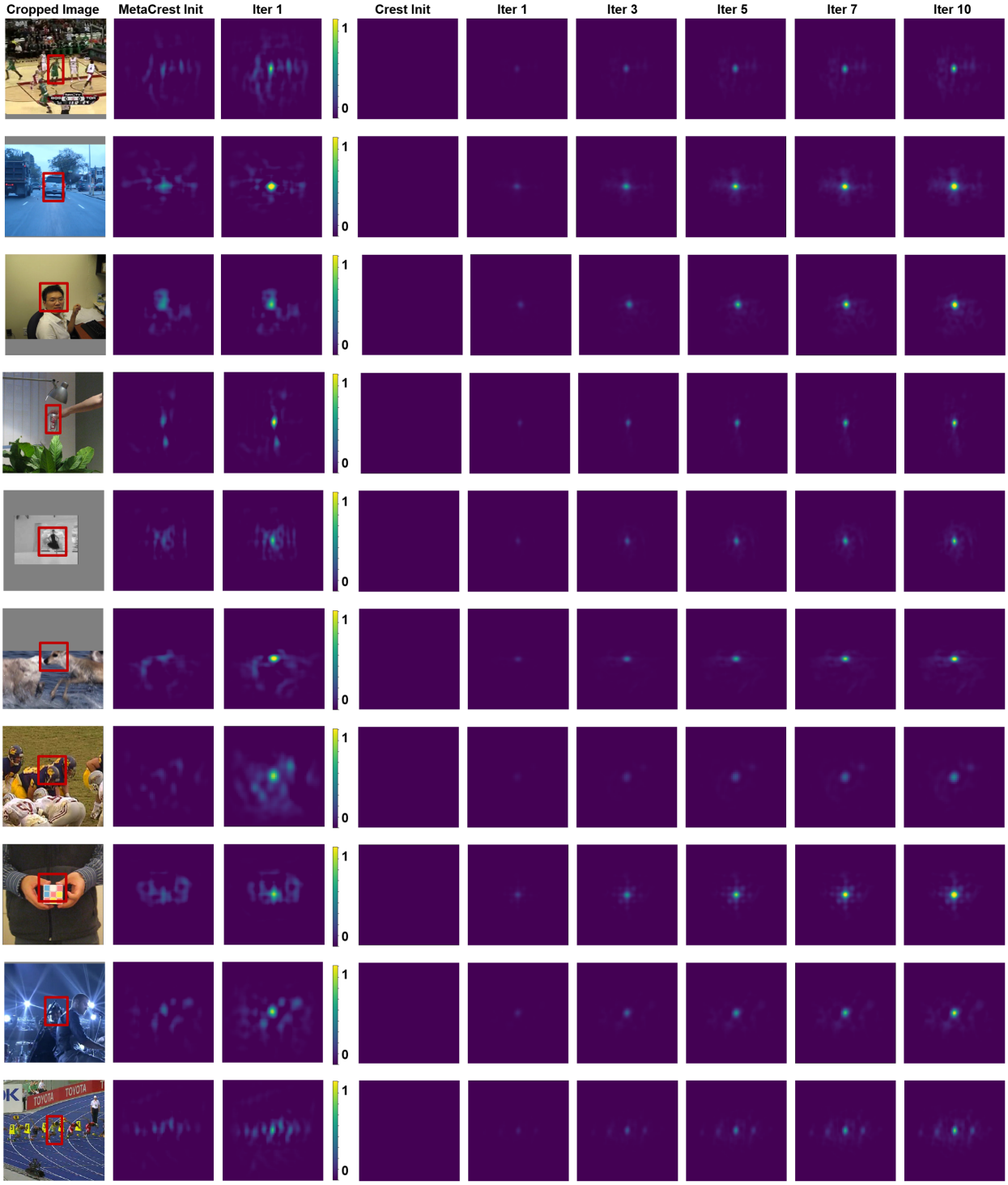


Table 6.1: Detailed results of MetaCREST on VOT2016 — Accuracy.

	MetaCREST-01	CREST-Base	CREST-10	CREST-05	CREST-03	CREST-01
bag	0.45	0.45	0.45	0.41	0.37	0.31
ball1	0.8	0.8	0.8	0.81	0.8	0.8
ball2	0.44	0.31	0.38	0.3	0.47	0.45
basketball	0.6	0.55	0.55	0.6	0.62	0.61
birds1	0.52	0.53	0.52	0.5	0.53	0.53
birds2	0.29	0.28	0.32	0.28	0.28	0.28
blanket	0.58	0.61	0.61	0.64	0.62	0.61
bmx	0.24	0.42	0.42	0.42	0.43	0.42
bolt1	0.52	0.66	0.67	0.67	0.68	0.68
bolt2	0.56	0.48	0.57	0.58	0.58	0.57
book	0.41	0.39	0.38	0.39	0.37	0.46
butterfly	0.45	0.4	0.4	0.41	0.4	0.41
car1	0.74	0.74	0.75	0.76	0.75	0.74
car2	0.77	0.73	0.74	0.75	0.77	0.78
crossing	0.71	0.69	0.69	0.7	0.71	0.71
dinosaur	0.41	0.29	0.28	0.32	0.32	0.43
fernando	0.38	0.35	0.32	0.4	0.41	0.42
fish1	0.43	0.4	0.42	0.53	0.52	0.52
fish2	0.34	0.42	0.39	0.34	0.34	0.33
fish3	0.6	0.61	0.61	0.59	0.58	0.58
fish4	0.46	0.43	0.31	0.42	0.43	0.43
girl	0.63	0.68	0.68	0.69	0.69	0.69
glove	0.52	0.52	0.53	0.51	0.53	0.55
godfather	0.48	0.48	0.48	0.48	0.48	0.48
graduate	0.38	0.47	0.48	0.49	0.46	0.47
gymnastics1	0.37	0.38	0.36	0.37	0.39	0.39
gymnastics2	0.5	0.5	0.49	0.52	0.5	0.5
gymnastics3	0.29	0.39	0.35	0.37	0.33	0.29
gymnastics4	0.5	0.44	0.43	0.42	0.43	0.44
hand	0.43	0.4	0.38	0.42	0.41	0.44
handball1	0.6	0.57	0.6	0.62	0.62	0.57
handball2	0.46	0.49	0.52	0.52	0.54	0.44
helicopter	0.38	0.51	0.51	0.49	0.47	0.47
iceskater1	0.51	0.52	0.52	0.56	0.56	0.54
iceskater2	0.58	0.53	0.53	0.53	0.55	0.54
leaves	0.39	0.13	0.13	0.3	0.3	0.29
marching	0.74	0.75	0.75	0.74	0.74	0.75
matrix	0.66	0.55	0.55	0.61	0.61	0.54
motocross1	0.47	0.42	0.42	0.42	0.39	0.39
motocross2	0.45	0.42	0.42	0.49	0.48	0.54
nature	0.46	0.3	0.3	0.32	0.29	0.32
octopus	0.4	0.32	0.33	0.44	0.43	0.37
pedestrian1	0.73	0.68	0.72	0.7	0.69	0.68
pedestrian2	0.3	0.45	0.49	0.38	0.37	0.34
rabbit	0.41	0.23	0.33	0.32	0.41	0.29
racing	0.52	0.42	0.42	0.4	0.4	0.41
road	0.58	0.58	0.57	0.62	0.64	0.67
shaking	0.66	0.64	0.69	0.73	0.76	0.68
sheep	0.49	0.5	0.5	0.5	0.52	0.53
singer1	0.6	0.65	0.68	0.61	0.57	0.41
singer2	0.69	0.65	0.6	0.54	0.66	0.64
singer3	0.4	0.23	0.25	0.16	0.15	0.15
soccer1	0.52	0.48	0.53	0.53	0.55	0.51
soccer2	0.56	0.6	0.57	0.57	0.57	0.57
soldier	0.39	0.37	0.36	0.36	0.37	0.4
sphere	0.49	0.46	0.44	0.41	0.39	0.41
tiger	0.71	0.68	0.68	0.67	0.68	0.66
traffic	0.79	0.79	0.8	0.8	0.76	0.73
tunnel	0.52	0.67	0.68	0.43	0.41	0.38
wiper	0.62	0.65	0.65	0.64	0.65	0.65
mean	0.51	0.5	0.51	0.51	0.51	0.5
weighted_mean	0.52	0.52	0.52	0.52	0.52	0.52

Table 6.2: Detailed results of MetaCREST on VOT2016 — Robustness (The number of failures).

	MetaCREST-01	CREST-Base	CREST-10	CREST-05	CREST-03	CREST-01
bag	0	0	0	0	0	0
ball1	0	0	0	0	0	0
ball2	1	0	0	0	1	1
basketball	1	2	1.9	1	1	1
birds1	1	1	2	2	1	1
birds2	0	0	1	0	0	0
blanket	0	0	0	0	0	0
bmx	1	0	0	0	0	0
bolt1	0	1	1.1	2	2	2
bolt2	0	0	1	1	1	2
book	2	4	6	3	3.2	5
butterfly	0	1	1	1	1	1
car1	1	1	1	1	1	1
car2	0	0	0	0	0	0
crossing	1	1	1	1	1	1
dinosaur	3	4	4	3	3	3
fernando	0.87	1	0	1	1	1
fish1	2.6	3	3.2	2	2	2
fish2	3.93	4	4	5	4	4
fish3	0	0	0	0	0	0
fish4	0	0	2	0.6	0.2	0
girl	1	1	1	1	1	1
glove	2	2	2	2	2	2
godfather	0	0	0	0	0	0
graduate	0	3	3.2	1.1	1.3	3
gymnastics1	1.67	1	2.9	2.3	5.2	5.1
gymnastics2	3	2	2	3	2	2
gymnastics3	1.53	3	2.9	2	3	3
gymnastics4	0	0	0	0	0	0
hand	7	8	7	7	7	6
handball1	0	1	0	0	0	0
handball2	3.27	5	3.4	3	3	2.9
helicopter	0	1	1	1	1	1
iceskater1	0	0	0	0	0	0
iceskater2	0.8	2	2.2	1.4	1.2	1.3
leaves	1	3	3	3	3	3
marching	0	0	0	0	0	0
matrix	1	2	2	2	2	1
motocross1	0.2	3	3	3	3	3
motocross2	0	1	0.8	0.3	0.9	0
nature	1.93	4	4	4	4	4
octopus	0	0	0	1	1	0
pedestrian1	1.87	2	1.2	1	1	2
pedestrian2	0	1	1	1	1	1
rabbit	3	5	4	4.2	3	4.1
racing	0	0	0	0	0	0
road	0	0	0	0	0	0
shaking	0	0	1	1	1	0
sheep	0	0	0	0	0	0
singer1	0	0	0	0	0	0
singer2	0	1	1	2	1	1
singer3	1.93	0	0	1	1	1
soccer1	2.93	3	2	2	1	1
soccer2	2	5	2	4	4	2
soldier	1	1	1	1	1	1
sphere	0	0	0	0	0	0
tiger	0	0	0	0	0	0
traffic	0	0	0	0	0	0.2
tunnel	0	0	0	0	0	0
wiper	0.4	0	0	0	0	0
mean	0.93	1.38	1.38	1.3	1.28	1.28
weighted_mean	0.84	1.36	1.45	1.26	1.25	1.29

Table 6.3: Detailed results of MetaSDNet on VOT2016 — Accuracy.

	MetaSDNet-01	pyMDNet-30	pyMDNet-15	pyMDNet-10	pyMDNet-05	pyMDNet-03	pyMDNet-01
bag	0.49	0.52	0.51	0.49	0.47	0.43	0.38
ball1	0.77	0.79	0.78	0.79	0.78	0.77	0.75
ball2	0.33	0.46	0.47	0.38	0.46	0.45	0.5
basketball	0.62	0.62	0.63	0.61	0.63	0.57	0.53
birds1	0.51	0.48	0.5	0.48	0.49	0.47	0.45
birds2	0.35	0.33	0.34	0.35	0.34	0.31	0.33
blanket	0.6	0.58	0.56	0.56	0.51	0.54	0.49
bmx	0.19	0.41	0.43	0.43	0.37	0.4	0.35
bolt1	0.51	0.52	0.53	0.54	0.58	0.6	0.56
bolt2	0.54	0.55	0.57	0.57	0.57	0.58	0.42
book	0.45	0.46	0.44	0.4	0.38	0.31	0.33
butterfly	0.36	0.29	0.34	0.3	0.39	0.38	0.29
car1	0.75	0.77	0.77	0.77	0.76	0.75	0.61
car2	0.71	0.75	0.75	0.75	0.72	0.7	0.5
crossing	0.67	0.69	0.67	0.67	0.65	0.67	0.56
dinosaur	0.57	0.62	0.61	0.62	0.58	0.36	0.42
fernando	0.45	0.43	0.43	0.44	0.42	0.38	0.33
fish1	0.46	0.44	0.44	0.43	0.43	0.42	0.37
fish2	0.32	0.32	0.33	0.32	0.31	0.29	0.24
fish3	0.6	0.64	0.65	0.63	0.6	0.54	0.38
fish4	0.36	0.37	0.38	0.41	0.32	0.33	0.32
girl	0.69	0.69	0.67	0.7	0.69	0.67	0.6
glove	0.49	0.52	0.49	0.5	0.48	0.38	0.38
godfather	0.46	0.45	0.45	0.45	0.44	0.4	0.4
graduate	0.51	0.53	0.52	0.52	0.49	0.43	0.4
gymnastics1	0.43	0.51	0.53	0.42	0.51	0.42	0.43
gymnastics2	0.44	0.49	0.49	0.46	0.45	0.46	0.42
gymnastics3	0.25	0.25	0.26	0.25	0.26	0.26	0.23
gymnastics4	0.46	0.48	0.49	0.47	0.45	0.43	0.36
hand	0.51	0.5	0.5	0.5	0.51	0.49	0.45
handball1	0.55	0.58	0.58	0.58	0.55	0.55	0.52
handball2	0.55	0.57	0.56	0.56	0.55	0.53	0.51
helicopter	0.57	0.49	0.5	0.49	0.44	0.4	0.37
iceskater1	0.53	0.54	0.52	0.54	0.53	0.5	0.49
iceskater2	0.55	0.55	0.54	0.53	0.48	0.49	0.42
leaves	0.29	0.32	0.32	0.4	0.31	0.28	0.28
marching	0.72	0.72	0.7	0.71	0.59	0.52	0.43
matrix	0.5	0.54	0.55	0.54	0.54	0.56	0.55
motocross1	0.47	0.48	0.48	0.47	0.47	0.46	0.39
motocross2	0.49	0.58	0.58	0.59	0.56	0.45	0.42
nature	0.49	0.44	0.43	0.39	0.43	0.4	0.42
octopus	0.57	0.59	0.58	0.58	0.59	0.56	0.47
pedestrian1	0.71	0.71	0.71	0.71	0.66	0.61	0.65
pedestrian2	0.37	0.44	0.48	0.51	0.45	0.5	0.4
rabbit	0.3	0.33	0.3	0.28	0.33	0.27	0.25
racing	0.48	0.45	0.45	0.45	0.43	0.41	0.34
road	0.45	0.47	0.47	0.45	0.48	0.47	0.45
shaking	0.6	0.58	0.54	0.58	0.59	0.59	0.57
sheep	0.53	0.54	0.53	0.53	0.48	0.45	0.47
singer1	0.58	0.57	0.6	0.56	0.58	0.59	0.5
singer2	0.59	0.64	0.64	0.65	0.64	0.57	0.48
singer3	0.28	0.26	0.27	0.26	0.28	0.28	0.26
soccer1	0.53	0.57	0.59	0.58	0.57	0.54	0.56
soccer2	0.59	0.62	0.58	0.59	0.62	0.53	0.49
soldier	0.45	0.52	0.51	0.52	0.48	0.46	0.34
sphere	0.5	0.55	0.54	0.54	0.55	0.56	0.52
tiger	0.68	0.66	0.67	0.65	0.62	0.59	0.57
traffic	0.78	0.8	0.79	0.8	0.77	0.58	0.54
tunnel	0.84	0.84	0.83	0.84	0.84	0.74	0.39
wiper	0.71	0.7	0.7	0.7	0.68	0.6	0.44
mean	0.52	0.54	0.53	0.53	0.52	0.49	0.44
weighted mean	0.54	0.55	0.55	0.54	0.53	0.5	0.45

Table 6.4: Detailed results of MetaSDNet on VOT2016 — Robustness (The number of failures).

	MetaSDNet-01	pyMDNet-30	pyMDNet-15	pyMDNet-10	pyMDNet-05	pyMDNet-03	pyMDNet-01
bag	0	0.07	0	0	0.07	0.13	0.2
ball1	0.4	0.07	0.13	0	0.2	0.33	1.6
ball2	0.67	2.27	2.33	2.67	2.67	3.07	3.33
basketball	1.27	1.07	1.27	1.33	1.8	3	5.73
birds1	0.47	2.2	1.73	1.47	1.33	2.33	3.13
birds2	0.4	0.27	0.4	0.13	0.13	0.47	0.6
blanket	0	0	0	0	0.13	0.33	1
bmx	0.87	0.13	0	0	0.13	0.2	0.47
bolt1	0.13	0	0.07	0.13	0.67	1.4	2.27
bolt2	0	0.27	0.73	0.6	0.4	1.4	1.87
book	3.4	3.53	3.27	3.53	4.73	4.47	7.4
butterfly	0.13	0	0.13	0.13	0.6	0.8	1.6
car1	1.73	2.27	2.33	1.87	2.07	2.93	2.87
car2	0	0	0	0	0.2	0.53	1
crossing	0	0.07	0.07	0.07	0.07	0.13	0.47
dinosaur	0.6	0	0	0	0.47	3.6	4.67
fernando	0.67	0.67	1.33	0.73	0.93	1.4	3.47
fish1	2.73	2.8	2.67	2.8	3.13	3.2	4.47
fish2	2.4	3	2.73	3	3.2	4.8	7.2
fish3	0.27	0.87	0.67	0.73	0.73	0.87	0.8
fish4	0.67	0.13	0.4	0.6	0.8	1.47	1.2
girl	0.07	0.07	0.13	0	0	0.33	1.53
glove	2.8	2.47	2.4	2.73	3.13	3.87	4.67
godfather	0	0	0	0	0.47	0.47	1.47
graduate	0	0.13	0.07	0.07	0.2	0.93	1.8
gymnastics1	1.53	0.67	0.6	0.87	0.73	1.13	2.93
gymnastics2	1.87	1.4	1.73	1.67	1.53	3.07	4.27
gymnastics3	1.2	1.07	1.4	1.4	1.4	2.2	3.93
gymnastics4	0.13	0	0	0.07	0.13	0.33	1.33
hand	2.53	0.8	1.53	0.93	1.2	4.07	6.6
handball1	0.93	0.6	0.93	0.73	1.93	2.13	3
handball2	1.93	2.27	2.47	2.67	2.73	3.67	7.53
helicopter	1	0.27	0.33	0.4	0.27	0.2	0.87
iceskater1	0.07	0.27	0.2	0.07	0.4	0.2	1.6
iceskater2	0.47	0	0.07	0.4	1.2	2.93	7.8
leaves	4.4	4.4	4.67	4.73	4.2	4.27	4.67
marching	0	0.13	0.2	0.33	1.27	1.6	2.27
matrix	1.8	1.53	1.4	1.4	1.87	1.6	3.53
motocross1	0.13	0	0	0	0	0.13	2.07
motocross2	0.07	0	0	0	0.07	0.73	0.93
nature	2.47	2.4	2.2	1.8	2.73	3.53	4.8
octopus	0	0	0	0	0	0.07	0.27
pedestrian1	1.2	1.07	1	1.27	1.07	1.47	2.67
pedestrian2	0	0	0.13	0	0.07	0.13	0.67
rabbit	3.2	4.27	4.67	5.07	4.53	5.27	6.53
racing	0	0	0	0	0.13	0.73	1.07
road	0	0	0	0	0	0.07	0.4
shaking	0.07	0.07	0.07	0.13	0.33	0.33	0.53
sheep	0.07	0	0	0.27	0.53	0.73	0.87
singer1	0	0	0	0	0	0.2	0.8
singer2	0.73	0.33	0.2	0.2	0.47	1.07	3.27
singer3	0.4	0.33	0.53	0.27	0.33	0.53	1
soccer1	0.73	1.47	0.93	0.8	1.47	1.27	1.73
soccer2	7.4	10.4	9.8	10.53	11.87	12.6	12.8
soldier	0.93	0.07	0.2	0.47	0.53	0.93	2
sphere	0	0	0	0	0.07	0.4	0.87
tiger	0.6	0.07	0	0	0.2	0.47	2.47
traffic	0.07	0.07	0.13	0	0	0.33	0.87
tunnel	0	0	0	0	0	0.8	1.2
wiper	0.47	0.33	0.33	0.27	0.33	0.53	1.07
mean	0.93	0.94	0.98	0.99	1.2	1.7	2.73
weighted mean	0.78	0.74	0.77	0.74	0.97	1.51	2.62

Table 6.5: Detailed results of MetaCREST on OTB2015 (BasketBall — Girl).

	MetaCREST-01	CREST	CREST-Base	CREST-20	CREST-10	CREST-05	CREST-01
BasketBall	0.658588	0.7194	0.7105	0.7286	0.7274	0.7302	0.7401
Biker	0.262911	0.3414	0.3417	0.3451	0.3484	0.2495	0.2482
Bird1	0.053922	0.3103	0.3306	0.3166	0.419	0.3757	0.3825
Bird2	0.822511	0.7638	0.7605	0.7595	0.7874	0.7937	0.8278
BlurBody	0.685201	0.6818	0.6829	0.6932	0.7039	0.6806	0.6913
BlurCar1	0.812668	0.8451	0.8452	0.8451	0.8479	0.8478	0.8498
BlurCar2	0.803337	0.8612	0.8623	0.8612	0.8584	0.8634	0.8593
BlurCar3	0.828998	0.8485	0.8494	0.851	0.8506	0.8482	0.8477
BlurCar4	0.800752	0.8383	0.8378	0.841	0.8367	0.8147	0.7949
BlurFace	0.816285	0.8355	0.8339	0.8355	0.8355	0.8362	0.8407
BlurOwl	0.780318	0.8024	0.8023	0.8014	0.7936	0.7891	0.7824
Board	0.777413	0.8172	0.8204	0.8125	0.808	0.8169	0.8037
Bolt	0.788435	0.6332	0.6197	0.638	0.6933	0.7045	0.7405
Bolt2	0.685844	0.0111	0.0111	0.0107	0.0107	0.0106	0.0106
Box	0.688159	0.7071	0.712	0.7358	0.7388	0.7323	0.7375
Boy	0.712229	0.8159	0.8128	0.8159	0.8129	0.8103	0.811
Car1	0.181559	0.2413	0.2498	0.2413	0.2537	0.2611	0.2089
Car2	0.716059	0.7891	0.7903	0.7891	0.78	0.7622	0.7537
Car24	0.558321	0.5904	0.6066	0.6208	0.6077	0.5958	0.5834
Car4	0.503505	0.6453	0.6467	0.6312	0.6517	0.6234	0.6001
CarDark	0.783473	0.748	0.7407	0.7684	0.774	0.7778	0.7955
CarScale	0.562736	0.6019	0.6005	0.6028	0.6036	0.603	0.4204
ClifBar	0.544592	0.1913	0.2028	0.2021	0.2017	0.1826	0.1918
Coke	0.610538	0.5765	0.5763	0.5765	0.5749	0.5816	0.5826
Couple	0.67483	0.5891	0.5905	0.5993	0.5963	0.602	0.5561
Coupon	0.876802	0.9131	0.9131	0.9131	0.9131	0.9134	0.9199
Crossing	0.730556	0.7548	0.7254	0.7544	0.7448	0.7016	0.6972
Crowds	0.698641	0.6595	0.669	0.6753	0.6775	0.724	0.7316
Dancer	0.722116	0.7329	0.7327	0.7395	0.7429	0.7452	0.7124
Dancer2	0.763492	0.7302	0.7302	0.7317	0.7508	0.76	0.7629
David	0.63482	0.6785	0.6785	0.7035	0.6863	0.6418	0.5828
David2	0.799415	0.7438	0.7404	0.7495	0.7693	0.7797	0.7821
David3	0.776266	0.774	0.7793	0.7681	0.7765	0.7816	0.7857
Deer	0.814219	0.7894	0.7881	0.7894	0.7887	0.784	0.7612
Diving	0.362569	0.2352	0.2343	0.3092	0.3783	0.3136	0.3154
Dog	0.463817	0.4721	0.4754	0.4814	0.5069	0.4901	0.3697
Dog1	0.677637	0.7278	0.7273	0.7298	0.7097	0.7151	0.6729
Doll	0.594906	0.6093	0.6186	0.6341	0.6214	0.6111	0.5993
DragonBaby	0.622419	0.6018	0.6018	0.6018	0.6018	0.6068	0.6372
Dudek	0.764275	0.7945	0.7937	0.7945	0.7945	0.8064	0.8035
FaceOcc1	0.717435	0.7365	0.7362	0.7365	0.7365	0.7337	0.7294
FaceOcc2	0.739151	0.7235	0.7236	0.7244	0.7433	0.7448	0.7261
Fish	0.828231	0.8302	0.8314	0.8284	0.8266	0.8203	0.8214
FleetFace	0.701893	0.6879	0.6838	0.6879	0.6879	0.687	0.6893
Football	0.730729	0.7276	0.7259	0.7276	0.7332	0.7386	0.7368
Football1	0.80888	0.6712	0.6319	0.7342	0.7831	0.8018	0.8057
Freeman1	0.588519	0.6227	0.6098	0.2888	0.267	0.2824	0.2911
Freeman3	0.508696	0.3564	0.3576	0.3552	0.3393	0.3393	0.3406
Freeman4	0.411745	0.4335	0.3969	0.3978	0.401	0.4013	0.4006
Girl	0.675333	0.7572	0.7568	0.7572	0.7538	0.7525	0.711

Table 6.6: Detailed results of MetaCREST on OTB2015 (Girl2 — Woman).

	MetaCREST-01	CREST	CREST-Base	CREST-20	CREST-10	CREST-05	CREST-01
Girl2	0.594508	0.5784	0.5672	0.5937	0.6041	0.6046	0.6139
Gym	0.512262	0.4946	0.5087	0.4976	0.5196	0.4995	0.4956
Human2	0.63796	0.6762	0.676	0.6797	0.6932	0.691	0.6977
Human3	0.667901	0.048	0.0482	0.048	0.0482	0.0492	0.0473
Human4-2	0.54537	0.3327	0.3291	0.313	0.3308	0.3355	0.3542
Human5	0.504575	0.5228	0.5215	0.5637	0.5806	0.3885	0.4031
Human6	0.338745	0.2988	0.2934	0.2893	0.2938	0.2534	0.2137
Human7	0.685143	0.7573	0.7516	0.7575	0.7558	0.7356	0.625
Human8	0.680804	0.6306	0.6291	0.7031	0.7001	0.5577	0.5141
Human9	0.61249	0.6665	0.6662	0.6937	0.6767	0.5681	0.5135
Ironman	0.448078	0.3873	0.4016	0.4013	0.393	0.42	0.4191
Jogging-1	0.733364	0.7681	0.7672	0.7844	0.7866	0.7791	0.7791
Jogging-2	0.766713	0.7763	0.7771	0.7779	0.774	0.7631	0.769
Jump	0.248634	0.0792	0.0894	0.2619	0.2806	0.1101	0.0995
Jumping	0.726609	0.6706	0.6709	0.6782	0.6895	0.6994	0.7033
KiteSurf	0.704649	0.7126	0.7149	0.7012	0.7001	0.7001	0.6956
Lemming	0.59702	0.7557	0.7552	0.758	0.7624	0.7593	0.7545
Liquor	0.83414	0.8162	0.8153	0.8181	0.8189	0.8189	0.8189
Man	0.822672	0.795	0.7978	0.8028	0.8127	0.8266	0.807
Matrix	0.595238	0.4138	0.4233	0.4071	0.3929	0.2729	0.2748
Mhyang	0.742506	0.8348	0.8355	0.8348	0.8164	0.8215	0.8024
MotorRolling	0.652729	0.5752	0.5723	0.4974	0.1687	0.1135	0.0987
MountainBike	0.677945	0.7005	0.7007	0.7005	0.7001	0.6919	0.6953
Panda	0.535333	0.5103	0.4732	0.5478	0.5699	0.5645	0.5843
RedTeam	0.518596	0.6082	0.6424	0.6442	0.6462	0.6456	0.6336
Rubik	0.666857	0.6618	0.6572	0.6627	0.6642	0.662	0.6573
Shaking	0.75499	0.7444	0.7409	0.7444	0.7293	0.7571	0.771
Singer1	0.539547	0.6697	0.674	0.6634	0.6344	0.5872	0.4001
Singer2	0.775826	0.7328	0.733	0.0364	0.037	0.0389	0.0399
Skater	0.634821	0.5952	0.5958	0.5929	0.5961	0.6074	0.6259
Skater2	0.616092	0.5734	0.591	0.6032	0.6193	0.6165	0.5872
Skating1	0.65869	0.5182	0.556	0.5248	0.5739	0.5764	0.5942
Skating2-1	0.519682	0.3528	0.351	0.3578	0.3601	0.3658	0.3614
Skating2-2	0.41488	0.4164	0.4146	0.4978	0.5029	0.4014	0.3908
Skiing	0.566725	0.4674	0.4897	0.4738	0.4856	0.4656	0.465
Soccer	0.184038	0.4298	0.4275	0.4298	0.4298	0.3907	0.4563
Subway	0.790204	0.7045	0.1782	0.7415	0.7352	0.7369	0.7429
Surfer	0.632852	0.6511	0.6494	0.6682	0.5906	0.542	0.511
Suv	0.753036	0.7996	0.8062	0.8047	0.803	0.7994	0.5684
Sylvester	0.687024	0.7601	0.7605	0.7601	0.7625	0.7647	0.7645
Tiger1	0.685096	0.6002	0.6005	0.6002	0.6002	0.6817	0.731
Tiger2	0.628571	0.5894	0.5885	0.5931	0.5926	0.5954	0.58
Toy	0.633632	0.6038	0.602	0.6094	0.6422	0.6187	0.537
Trans	0.530338	0.5518	0.5457	0.5584	0.5465	0.5495	0.5757
Trellis	0.643987	0.7381	0.737	0.7394	0.7228	0.6891	0.6656
Twinnings	0.625316	0.67	0.6607	0.6781	0.6522	0.643	0.6395
Vase	0.450185	0.4665	0.4579	0.4665	0.4672	0.4521	0.3894
Walking	0.627369	0.6448	0.6596	0.6914	0.6775	0.662	0.6511
Walking2	0.464571	0.6364	0.6253	0.6294	0.6555	0.6575	0.634
Woman	0.702002	0.7292	0.7341	0.7132	0.7192	0.7598	0.7692
Average	0.63696061	0.622858	0.617631	0.621314	0.62165	0.608597	0.595679

Table 6.7: Detailed results of MetaSDNet on OTB2015 (BasketBall — Girl).

	MetaSDNet-01	pyMDNet-30	pyMDNet-15	pyMDNet-10	pyMDNet-05	pyMDNet-03	pyMDNet-01
BasketBall	0.6004	0.6386	0.6056	0.6273	0.7412	0.6121	0.0177
Biker	0.3793	0.339	0.3722	0.3575	0.3001	0.2884	0.0956
Bird1	0.0308	0.3054	0.2381	0.3228	0.3136	0.3177	0.0524
Bird2	0.7196	0.798	0.7884	0.7388	0.7253	0.6883	0.0789
BlurBody	0.7241	0.6705	0.7849	0.6806	0.7188	0.6501	0.696
BlurCar1	0.8176	0.7969	0.8145	0.8088	0.7959	0.0022	0.1475
BlurCar2	0.8407	0.8412	0.8528	0.8423	0.7875	0.6815	0.6893
BlurCar3	0.8494	0.8297	0.8455	0.7903	0.7431	0.2721	0.0365
BlurCar4	0.8499	0.7799	0.8387	0.8183	0.5847	0.1056	0.0165
BlurFace	0.8076	0.7806	0.7734	0.7807	0.5003	0.4183	0.2497
BlurOwl	0.7754	0.7482	0.799	0.7447	0.7184	0.6763	0.7001
Board	0.7262	0.7399	0.7028	0.6973	0.4725	0.4363	0.0685
Bolt	0.7506	0.7665	0.5725	0.7642	0.7488	0.3555	0.0044
Bolt2	0.6306	0.7118	0.7109	0.7512	0.6995	0.7042	0.0112
Box	0.7216	0.6343	0.7074	0.6664	0.0342	0.675	0.6477
Boy	0.779	0.7514	0.7565	0.769	0.7727	0.7529	0.7871
Car1	0.7116	0.6764	0.681	0.5854	0.6308	0.1188	0.0047
Car2	0.8147	0.8334	0.7965	0.8258	0.7901	0.325	0.2691
Car24	0.8151	0.8405	0.8366	0.8328	0.8585	0.8365	0.854
Car4	0.791	0.7396	0.7529	0.7563	0.7284	0.7472	0.7559
CarDark	0.7741	0.748	0.8254	0.7929	0.064	0.0109	0.0108
CarScale	0.7107	0.6691	0.6576	0.674	0.6926	0.0096	0.288
ClifBar	0.5904	0.6349	0.5728	0.6319	0.6308	0.6192	0.5928
Coke	0.6246	0.4793	0.5106	0.4865	0.4117	0.2068	0.1448
Couple	0.6694	0.5759	0.5418	0.5701	0.6139	0.6449	0.0235
Coupon	0.3282	0.3328	0.3459	0.3333	0.3293	0.1067	0.2085
Crossing	0.7369	0.7294	0.7655	0.7714	0.7452	0.7298	0.7552
Crowds	0.7405	0.6823	0.0899	0.0797	0.6384	0.6351	0.0071
Dancer	0.6673	0.6083	0.6707	0.6231	0.636	0.5443	0.3439
Dancer2	0.7705	0.7083	0.7378	0.7517	0.6822	0.7063	0.0924
David	0.7676	0.7614	0.7	0.7406	0.7635	0.7598	0.4336
David2	0.7287	0.745	0.7835	0.7514	0.7467	0.774	0.1124
David3	0.716	0.7303	0.7611	0.7668	0.7551	0.0083	0.0064
Deer	0.717	0.6915	0.7612	0.7009	0.7156	0.0543	0.0235
Diving	0.3838	0.3559	0.3623	0.3652	0.355	0.0401	0.0044
Dog	0.4413	0.4511	0.5163	0.5808	0.4916	0.5114	0.5111
Dog1	0.7815	0.7451	0.7662	0.7405	0.7387	0.7279	0.7273
Doll	0.8031	0.8298	0.8238	0.8339	0.8224	0.8295	0.8236
DragonBaby	0.7012	0.7122	0.6844	0.7054	0.563	0.0923	0.1239
Dudek	0.8306	0.8251	0.8388	0.8171	0.8398	0.8352	0.8142
FaceOcc1	0.6294	0.6669	0.7112	0.7294	0.7232	0.3265	0.1794
FaceOcc2	0.7009	0.7417	0.7494	0.7481	0.7396	0.7198	0.7155
Fish	0.7749	0.8526	0.7994	0.8485	0.7777	0.8374	0.8407
FleetFace	0.6339	0.722	0.7189	0.7067	0.7224	0.722	0.3047
Football	0.7199	0.1427	0.1406	0.6771	0.6534	0.1354	0.1046
Football1	0.6802	0.7085	0.6918	0.7233	0.5798	0.0946	0.0528
Freeman1	0.6897	0.7266	0.6953	0.6952	0.6607	0.612	0.694
Freeman3	0.7696	0.7607	0.7756	0.7828	0.7909	0.548	0.5321
Freeman4	0.6303	0.6204	0.6726	0.6288	0.2028	0.689	0.6793
Girl	0.7501	0.5939	0.6645	0.6036	0.5982	0.6363	0.5794

Table 6.8: Detailed results of MetaSDNet on OTB2015 (Girl2 — Woman).

	MetaSDNet-01	pyMDNet-30	pyMDNet-15	pyMDNet-10	pyMDNet-05	pyMDNet-03	pyMDNet-01
Girl2	0.7303	0.7339	0.7238	0.7257	0.6038	0.5872	0.6081
Gym	0.5068	0.5141	0.4567	0.5176	0.4625	0.4484	0.0254
Human2	0.7011	0.6262	0.7356	0.6797	0.6608	0.7351	0.402
Human3	0.5686	0.4716	0.0142	0.4545	0.5624	0.5327	0.0008
Human4-2	0.5927	0.6185	0.6038	0.567	0.5992	0.5804	0.0834
Human5	0.7134	0.624	0.6092	0.6533	0.7008	0.6225	0.5703
Human6	0.7504	0.694	0.7627	0.5205	0.7575	0.0121	0.003
Human7	0.7922	0.704	0.7021	0.6583	0.7718	0.795	0.5217
Human8	0.7024	0.6064	0.6399	0.5688	0.5435	0.5487	0.4799
Human9	0.664	0.5661	0.5538	0.4515	0.484	0.0073	0.0031
Ironman	0.4604	0.428	0.1816	0.4418	0.0138	0.1411	0.3368
Jogging-1	0.7098	0.7711	0.7369	0.7773	0.749	0.7217	0.4915
Jogging-2	0.7389	0.7439	0.7602	0.731	0.7565	0.7214	0.0338
Jump	0.0496	0.1593	0.251	0.1659	0.1694	0.0656	0.0582
Jumping	0.6867	0.7176	0.7157	0.6872	0.7184	0.5757	0.5276
KiteSurf	0.7098	0.7676	0.6298	0.343	0.7358	0.6865	0.7103
Lemming	0.7499	0.7108	0.7356	0.7582	0.7461	0.0215	0.0213
Liquor	0.7915	0.6435	0.7477	0.6869	0.7283	0.6757	0.1501
Man	0.8077	0.8017	0.779	0.7903	0.79	0.8088	0.8006
Matrix	0.279	0.4671	0.4476	0.4724	0.4705	0.1057	0.0705
Mhyang	0.7521	0.8094	0.8147	0.7603	0.6258	0.6084	0.674
MotorRolling	0.5839	0.5833	0.5851	0.5976	0.5738	0.1283	0.2398
MountainBike	0.7652	0.7172	0.7366	0.6433	0.7135	0.0063	0.0063
Panda	0.6096	0.5238	0.5448	0.57	0.5035	0.5199	0.4794
RedTeam	0.559	0.5711	0.6262	0.5641	0.5719	0.5234	0.2384
Rubik	0.7301	0.7106	0.7157	0.7075	0.7009	0.7092	0.0121
Shaking	0.6814	0.5986	0.6864	0.0907	0.0421	0.6373	0.0566
Singer1	0.7614	0.7159	0.7422	0.7128	0.6918	0.4938	0.6759
Singer2	0.6043	0.6906	0.0389	0.0848	0.0628	0.006	0.0034
Skater	0.5256	0.5482	0.6131	0.5643	0.5827	0.5601	0.6149
Skater2	0.5395	0.5747	0.6726	0.5111	0.417	0.0297	0.0363
Skating1	0.6006	0.6439	0.5325	0.5804	0.6102	0.5495	0.592
Skating2-1	0.481	0.5028	0.4327	0.4521	0.4986	0.4919	0.4462
Skating2-2	0.3244	0.4951	0.5316	0.3864	0.4175	0.0196	0.017
Skiing	0.4891	0.495	0.5056	0.4633	0.4762	0.3774	0.3633
Soccer	0.4948	0.4218	0.3805	0.4939	0.4021	0.3697	0.4123
Subway	0.7546	0.6754	0.6751	0.6724	0.6528	0.6727	0.6833
Surfer	0.7007	0.7285	0.719	0.7023	0.7207	0.7411	0.1836
Suv	0.744	0.7973	0.7744	0.7893	0.7716	0.6956	0.1601
Sylvester	0.7125	0.6861	0.7	0.6968	0.1434	0.2789	0.2842
Tiger1	0.6224	0.6396	0.6679	0.6355	0.4128	0.4374	0.0549
Tiger2	0.4618	0.6278	0.6971	0.6179	0.6907	0.0359	0.016
Toy	0.6755	0.6577	0.6505	0.6394	0.6424	0.6394	0.631
Trans	0.5518	0.619	0.6229	0.6194	0.2508	0.1306	0.134
Trellis	0.7806	0.6886	0.7653	0.6632	0.6988	0.6954	0.226
Twinnings	0.5667	0.7025	0.6562	0.6647	0.6228	0.2355	0.0073
Vase	0.55	0.4609	0.5083	0.5198	0.4902	0.0838	0.0931
Walking	0.7405	0.6504	0.7135	0.6823	0.6394	0.7067	0.7094
Walking2	0.7951	0.7848	0.7581	0.7666	0.7568	0.783	0.7836
Woman	0.7512	0.7577	0.7398	0.7542	0.7467	0.0019	0.0018
Average	0.6621	0.6522	0.6416	0.6328	0.5950	0.4539	0.3165

6.2 Meta-Curvature

6.2.1 Case study

In this section, we provide a case study of the linear regression example. Let $\mathbf{X}_{\text{tr}}, \mathbf{X}_{\text{val}}, \mathbf{X}_{\text{new}} \in \mathbb{R}^{m \times p}$ training, validation, and test set and their targets are $\mathbf{Y}_{\text{tr}}, \mathbf{Y}_{\text{val}}, \mathbf{Y}_{\text{new}} \in \mathbb{R}^{m}$. With the model's parameter $\theta \in \mathbb{R}^{p}$, a typical loss function for the linear regression is defined as follows.

$$J(\theta) = \frac{1}{2}\|\mathbf{Y} - \mathbf{X}\theta\|^2. \tag{6.1}$$

The gradient w.r.t the model's parameter θ is

$$\nabla_{\theta} J(\theta) = -\mathbf{X}^{\top}(\mathbf{Y} - \mathbf{X}\theta). \tag{6.2}$$

Given the meta-curvature matrix, $\mathbf{M}$, a fixed inner learning rate α, then the meta-objective function is

$$J_{\text{val}}(\theta) = \frac{1}{2}\|\mathbf{Y}_{\text{val}} - \mathbf{X}_{\text{val}}(\theta^{\text{tr}})\|^2 \tag{6.3}$$

Following the derivation from the main text and given the new test set, we perform one inner and outer optimization steps. And the transformed gradient for the new test set is as follow.

$$\mathbf{M}_{\text{new}}\nabla_\theta J_{\text{new}}(\theta) \tag{6.4}$$

$$= \mathbf{M}\nabla_\theta J_{\text{new}}(\theta) + \beta\big(\nabla_\theta J_{\text{tr}}(\theta)^\top \nabla_\theta J_{\text{new}}(\theta)\big)\alpha\nabla_\theta J_{\text{val}}(\theta^{\text{tr}}) \tag{6.5}$$

$$= \mathbf{M}\nabla_\theta J_{\text{new}}(\theta) - \beta\big[(\mathbf{Y}_{\text{tr}} - \mathbf{X}_{\text{tr}}\theta)^\top \mathbf{X}_{\text{tr}}\mathbf{X}_{\text{new}}^\top(\mathbf{Y}_{\text{new}} - \mathbf{X}_{\text{new}}\theta)\big]\alpha\nabla_\theta J_{\text{val}}(\theta^{\text{tr}}) \tag{6.6}$$

$$= \mathbf{M}\nabla_\theta J_{\text{new}}(\theta) - \beta\big[(\mathbf{Y}_{\text{tr}} - \mathbf{X}_{\text{tr}}\theta)^\top \mathbf{X}_{\text{tr}}\mathbf{X}_{\text{new}}^\top(\mathbf{Y}_{\text{new}} - \mathbf{X}_{\text{new}}\theta)\big]\alpha\big[\mathbf{X}_{\text{val}}^\top(\mathbf{Y}_{\text{val}} - \mathbf{X}_{\text{val}}\theta^{\text{tr}})\big] \tag{6.7}$$

$$= \mathbf{M}\nabla_\theta J_{\text{new}}(\theta) - \beta\big[(\mathbf{Y}_{\text{tr}} - \mathbf{X}_{\text{tr}}\theta)^\top \mathbf{X}_{\text{tr}}\mathbf{X}_{\text{new}}^\top(\mathbf{Y}_{\text{new}} - \mathbf{X}_{\text{new}}\theta)\big] \tag{6.8}$$

$$\alpha\big[\mathbf{X}_{\text{val}}^\top\mathbf{Y}_{\text{val}} - \mathbf{X}_{\text{val}}^\top\mathbf{X}_{\text{val}}(\theta - \alpha\mathbf{M}\mathbf{X}_{\text{tr}}^\top(\mathbf{Y}_{\text{tr}} - \mathbf{X}_{\text{tr}}\theta))\big] \tag{6.9}$$

$$= \mathbf{M}\nabla_\theta J_{\text{new}}(\theta) - \beta\big[\underbrace{(\mathbf{Y}_{\text{tr}} - \mathbf{X}_{\text{tr}}\theta)^\top \mathbf{X}_{\text{tr}}\mathbf{X}_{\text{new}}^\top(\mathbf{Y}_{\text{new}} - \mathbf{X}_{\text{new}}\theta)}_{\text{A}}\big] \tag{6.10}$$

$$\big[\alpha\underbrace{\mathbf{X}_{\text{val}}^\top(\mathbf{Y}_{\text{val}} - \mathbf{X}_{\text{val}}\theta)}_{\text{B}} - \underbrace{\alpha^2\mathbf{X}_{\text{val}}^\top\mathbf{X}_{\text{val}}}_{\text{C}}\underbrace{\mathbf{M}\mathbf{X}_{\text{tr}}^\top(\mathbf{Y}_{\text{tr}} - \mathbf{X}_{\text{tr}}\theta)}_{\text{D}}\big]. \tag{6.11}$$

The term A is the gradient similarity term, and in linear regression case, it is defined as a bilinear form e.g. $\mathbf{x}^\top\mathbf{A}\mathbf{y}$, where $\mathbf{A} = \mathbf{X}_{\text{tr}}\mathbf{X}_{\text{new}}^\top$. It is multiplied by both training and test residuals. $\mathbf{A}$ is related to covariance matrix, but between training set and the new test set. The term B is the validation gradient term. The terms C and D correspond to $\mathcal{O}(\alpha^2)$. Since the loss function of the linear regression has a quadratic form and its derivative has a linear form. Therefore, the Taylor expansion of the derivative has up to α^2 order. The term D is the transformed gradient and the term C is a covariance matrix of validation dataset (assuming it's centered).

6.2.2 Experimental setup

6.2.2.1 Few-shot classification

For both 5-way 1-shot and 5-way 5-shot classification, we set the batch size 4 for 1 step experiments and 2 for 5 step experiments. 15 examples per class were used for evaluating the model after updates. In total, we ran 100,000 iterations for 1 step experiments and 200,000 iterations for

2 step experiments. The inner/outer learning rates are $\beta = 0.001, \alpha = 0.01$. We apply dropout rate 0.2 in the final linear layer for only MC1 and MC2 (other methods did perform worse with dropout). For cutout data augmentation, we cut out 36×36 random crops.

6.2.2.2 Few-shot reinforcement learning

For all experiments, the inner learning rates was $\alpha = 0.1$, discount factor $\gamma = 0.99$. And we used a meta batch size of 40 tasks. For each task, the horizon is H = 200 with 20 rollouts. For TRPO optimizer, the maximum threshold of KL divergence was 0.01, the number of conjugate gradient descent steps are 10, and the maximum number of line search was 15. Every line search step, the step size was multiplied by 0.8. For PPO, we used Adam as the optimizer. The clipping threshold is 0.3 and we performed 5 gradient steps per each meta batch. In HalfCheetahVel, we uniformly sample velocities from 0 to 2. In Walker2dVel, we also uniformly sample velocities from 0 to 5. In both Walker2dDir and Walker2dVel we give additional 1.0 reward when the agent does not fall over.

6.3 Learning to generate given few examples

6.4 Detailed Network Architectures

We provide the detailed network architecture of our approach in Figure 6.2.

6.5 More examples

We provide more visual examples for *car* and *chair* categories in Figures 6.3 and 6.4 respectively. In addition to novel views synthesized by our method, we also provide the intermediate output (visibility map and output of DOAFN) as well as views synthesized by other approaches.

6.6 Test results on random backgrounds

Figure 6.5 presents test results on synthesized images with random backgrounds. Intermediate stages, such as visibility map, background mask, and outputs of DOAFN are also shown. We compare against L_1 and AFN baselines. Note that L_1 and AFN could perform better on background area if we applied similar approaches used in TVSN, which we considered backgrounds separately.

6.7 Arbitrary transformations with linear interpolations of one-hot vectors

We show an experiment on the generalization capability for arbitrary transformations. Although we have trained the network with 17 discrete transformations in the range [20,340] with 20-degree increments, our trained network can synthesize arbitrary view points with linear interpolations of one-hot vectors. For example, if [0,1,0,0,...0] and [0,0,1,0,...0] represent 40 and 60-degree transformations respectively, [0,0.5,0.5,0,...0] represents 50 degree. More formally, let $\mathbf{t} \in [0,1]^{17}$ be encoding vector for the transformation parameter $\theta \in [20, 340]$ and s be step size ($s = 20$). For a transformation parameter $i \times s \leq \theta < (i+1) \times s$, i and $i+1$ elements of the

encoding vector $\mathbf{t}$ is

$$\mathbf{t}^i = 1 - \frac{\theta - (i \times s)}{s}, \quad \mathbf{t}^{i+1} = 1 - \mathbf{t}^i \tag{6.12}$$

Figure 6.6 shows some of examples. From the third to the sixth columns, we used linearly interpolated one-hot vectors to synthesize views between two consecutive discrete views that were in the original transformation set (the second and the last columns).

6.8 More categories

We picked cars and chairs, since both span a range of interesting challenges. The car category has rich variety of reflectance and textures, various shapes, and a large number of instances. The chair category was chosen since it is a good testbed for challenging 'thin shapes', e.g. legs of chairs, and unlike cars is far from convex in shape. We also wanted to compare to previous works, which were tested mostly on cars or chairs. In order to show our approach is well generalizable to other categories, we also performed experiments for *motorcycle* and *flowerpot* categories. We followed the same experimental setup. We used the entire *motocycle*(337 models) and *flowerpot*(602 models) categories. For each category, 80% of 3D models are used for training, which leaves around 0.1 million training pairs for the *motorcycle* and 0.2 million for the *flowerpot* category. For testing, we randomly sampled instances, input viewpoints, and desired transformations from the rest 20% of 3D models. Figure 6.7 shows some of qualitative results.

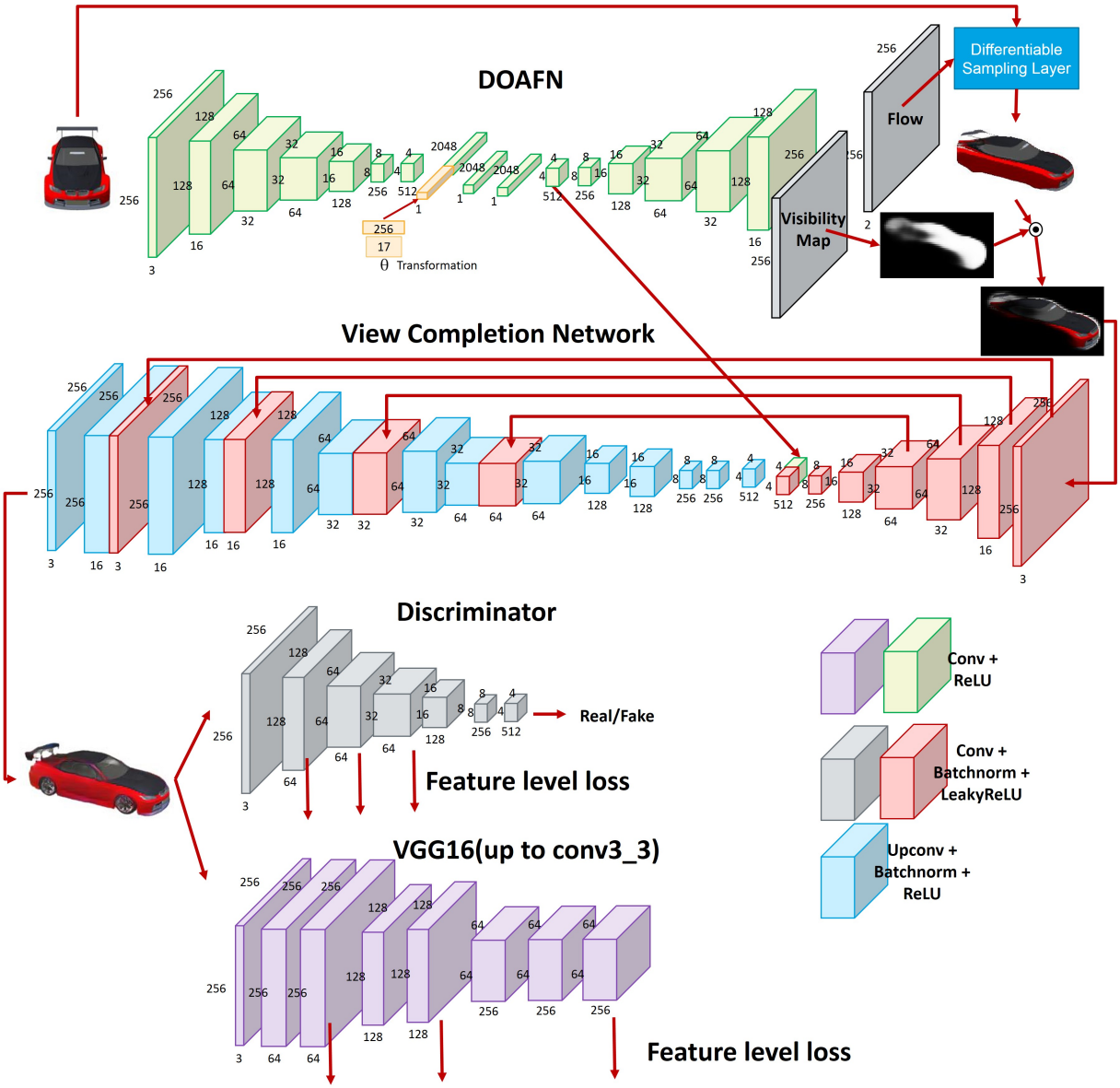


Figure 6.2: Transformation-grounded view synthesis network architecture

Figure 6.3: Results on test images from the car category (15). 1st-input, 2nd-ground truth. From 3rd to 6th are deep encoder-decoder networks with different losses. (3rd-L_1 norm (133), 4th-feature reconstruction loss with pretrained VGG16 network (60; 74; 137; 73), 5th-adversarial loss with feature matching (41; 108; 118), 6th-the combined loss). 7th-appearance flow network (AFN) (157). **8th-ours(TVSN)**.

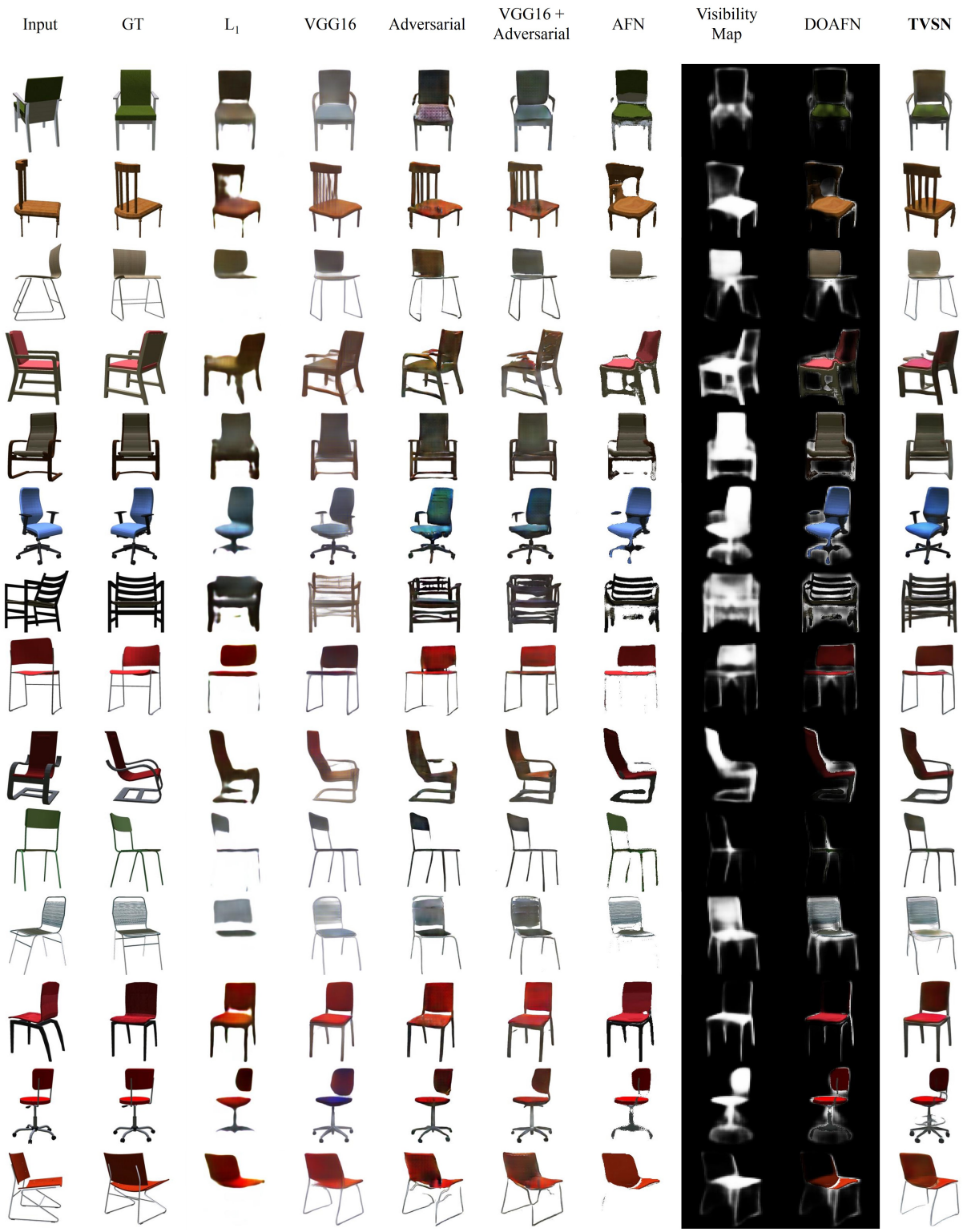


Figure 6.4: Results on test images from the car category (15). 1st-input, 2nd-ground truth. From 3rd to 6th are deep encoder-decoder networks with different losses. (3rd-L_1 norm (133), 4th-feature reconstruction loss with pretrained VGG16 network (60; 74; 137; 73), 5th-adversarial loss with feature matching (41; 108; 118), 6th-the combined loss). 7th-appearance flow network (AFN) (157). **8th-ours(TVSN)**.

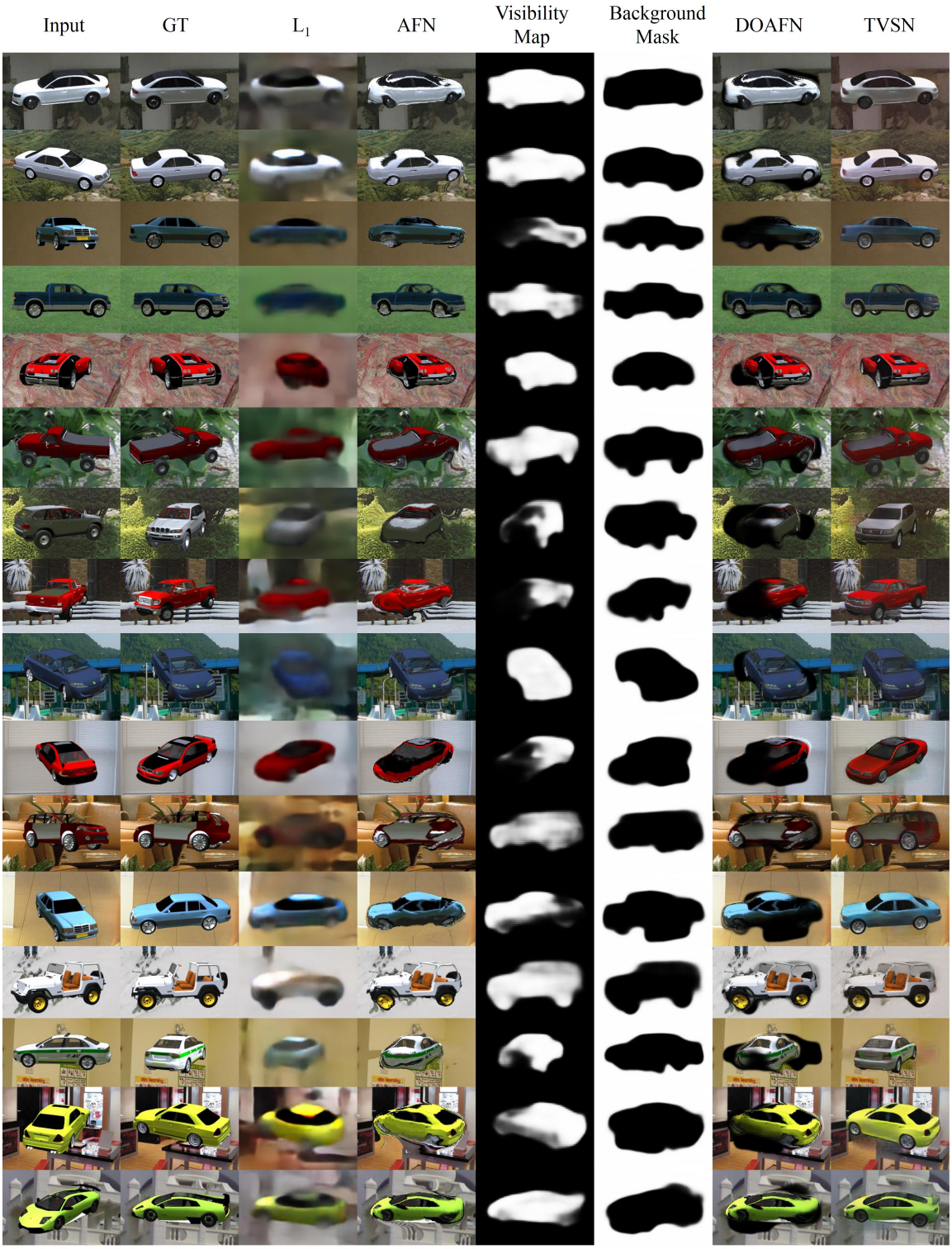


Figure 6.5: Test results on synthetic backgrounds

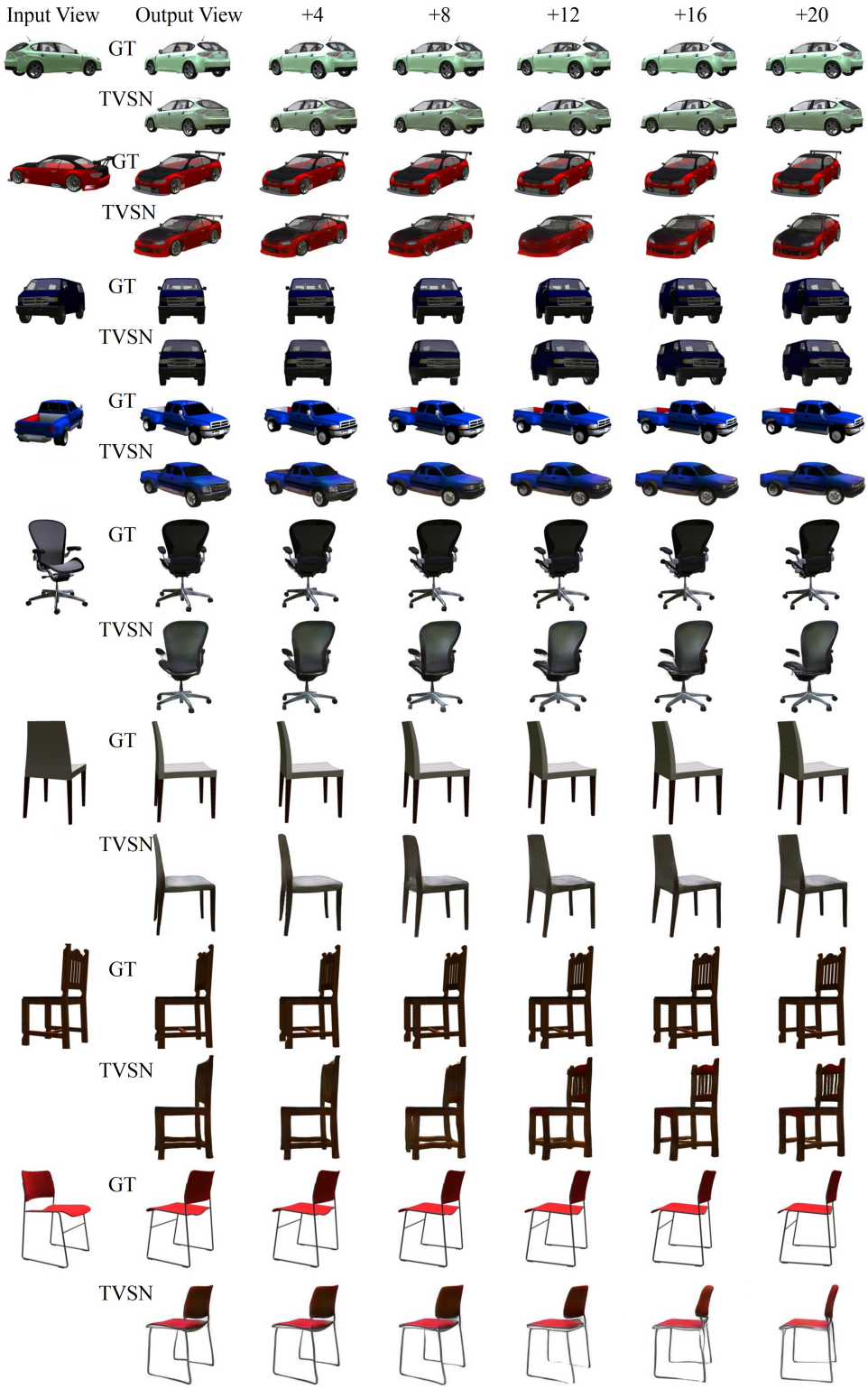


Figure 6.6: Test results of linear interpolation of one-hot vectors

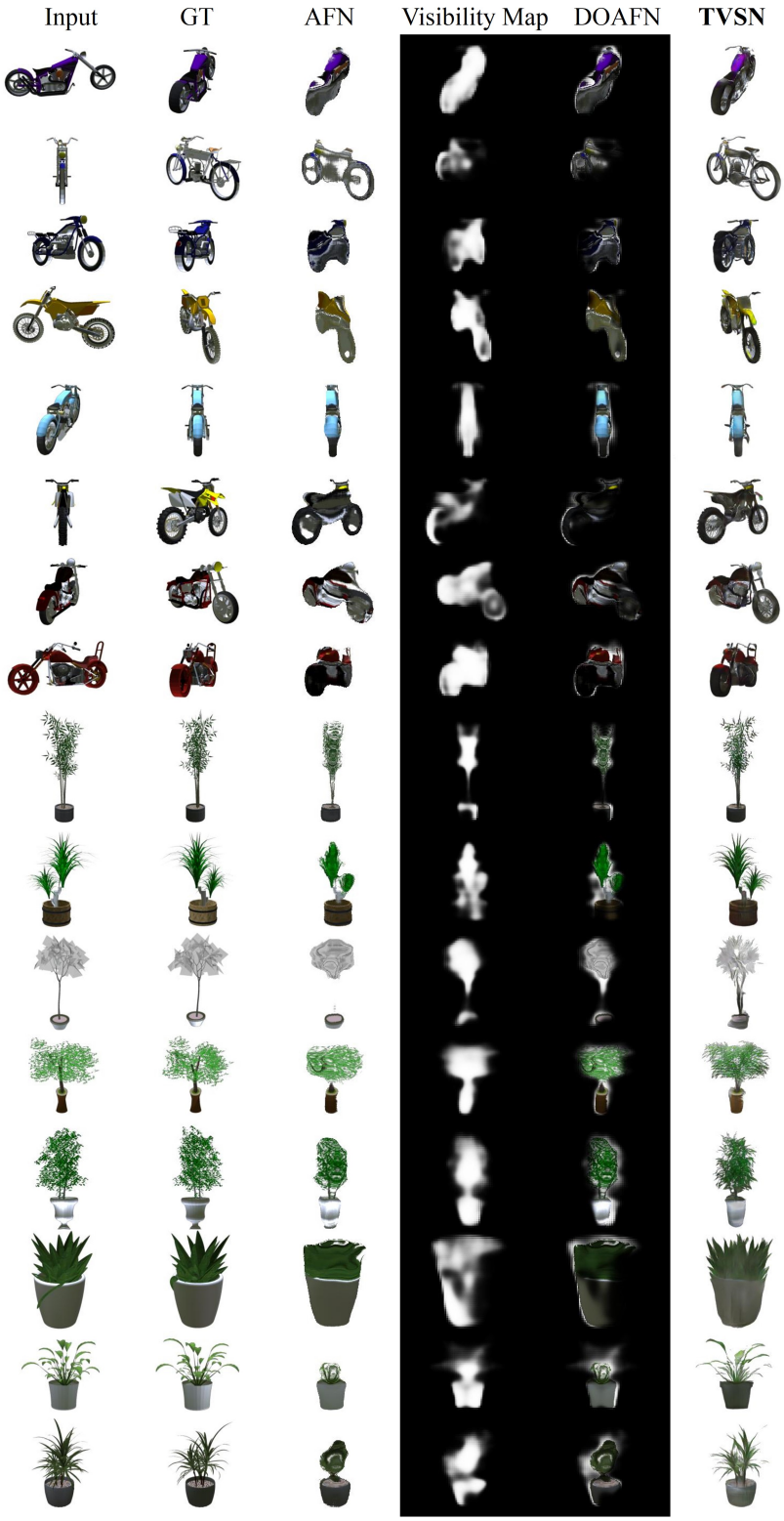


Figure 6.7: Test results of motorcycle and flowerpot categories

REFERENCES

openmvs: open multi-view stereo reconstruction library. https://github.com/cdcseacave/openMVS. Accessed: 2016-11-14.

Pytorch, http://www.pytorch.org.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. (2018a). Continuous adaptation via meta-learning in nonstationary and competitive environments. In *ICLR*.

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. (2018b). Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments. In *International Conference on Learning Representations (ICLR)*.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276.

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and Freitas, N. d. (2016). Learning to learn by gradient descent by gradient descent. In *NeurIPS*.

Antoniou, A., Edwards, H., and Storkey, A. (2018). How to train your MAML. *arXiv:1810.09502*.

Babenko, B., Yang, M.-H., and Belongie, S. (2010). Robust object tracking with online multiple instance learning. *TPAMI*.

Bai, Q., Wu, Z., Sclaroff, S., Betke, M., and Monnier, C. (2013). Randomized ensemble tracking. In *ICCV*.

Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. B. (2009). Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Computer Graphics (TOG)*.

Bertinetto, L., Henriques, J. F., Valmadre, J., Torr, P. H. S., and Vedaldi, A. (2016a). Learning feed-forward one-shot learners. In *NeurIPS*.

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. (2016b). Fully-convolutional siamese networks for object tracking. *ECCV Workshop*.

Bolme, D. S., Beveridge, J. R., Draper, B. A., and Lui, Y. M. (2010). Visual object tracking using adaptive correlation filters. In *CVPR*.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012*.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2017). Learning to learn without gradient descent by gradient descent. In *ICML*.

Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. (2016). 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *ECCV*.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2018). AutoAugment: Learning Augmentation Policies from Data. *arXiv:1805.09501*.

Danelljan, M., Bhat, G., Shahbaz Khan, F., and Felsberg, M. (2017). ECO: Efficient convolution operators for tracking. In *CVPR*.

Danelljan, M., Hager, G., Khan, F. S., and Felsberg, M. (2014). Accurate scale estimation for robust visual tracking. In *BMVC*.

Danelljan, M., Hager, G., Khan, F. S., and Felsberg, M. (2015). Learning spatially regularized correlation filters for visual tracking. In *ICCV*.

Danelljan, M., Robinson, A., Shahbaz Khan, F., and Felsberg, M. (2016). Beyond Correlation Filters: Learning continuous convolution operators for visual tracking. In *ECCV*.

DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv:1708.04552*.

Dosovitskiy, A. and Brox, T. (2016). Generating images with perceptual similarity metrics based on deep networks. In *NIPS*.

Dosovitskiy, A., Fischery, P., Ilg, E., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., Brox, T., et al. (2015a). Flownet: Learning optical flow with convolutional networks. In *ICCV*.

Dosovitskiy, A., Springenberg, J. T., and Brox, T. (2015b). Learning to generate chairs with convolutional neural networks. In *CVPR*.

Du, Y., Czarnecki, W. M., Jayakumar, S. M., Pascanu, R., and Lakshminarayanan, B. (2018). Adapting Auxiliary Losses Using Gradient Similarity. *arXiv:1812.02224*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. *28th Annual Conference on Computer Graphics and Interactive Techniques*.

Esteban Real, Alok Aggarwal, Y. H. Q. V. L. (2019). Regularized evolution for image classifier architecture search. In *AAAI*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Finn, C. and Levine, S. (2018). Meta-Learning and Universality: Deep Representations and Gradient Descent Can Approximate Any Learning Algorithm. In *International Conference on Learning Representations (ICLR)*.

Fischer, P., Dosovitskiy, A., Ilg, E., Hausser, P., Hazrbas, C., and Golkov, V. (2015). FlowNet: Learning optical flow with convolutional networks. In *CVPR*.

Flynn, J., Neulander, I., Philbin, J., and Snavely, N. (2016). Deepstereo: Learning to predict new views from the world's imagery. In *CVPR*.

Furukawa, Y. (2015). Multi-view stereo: A tutorial. *Foundations and Trends in Computer Graphics and Vision*.

Galoogahi, H. K., Sim, T., and Lucey, S. (2015). Correlation filters with limited boundaries. In *CVPR*.

Gan, Q., Guo, Q., Zhang, Z., and Cho, K. (2015). First step toward model-free, anonymous object tracking with recurrent neural networks. *arXiv:1511.06425*.

Garcia, V. and Bruna, J. (2018a). Few-Shot Learning with Graph Neural Networks. In *International Conference on Learning Representations (ICLR)*.

Garcia, V. and Bruna, J. (2018b). Few-Shot Learning with Graph Neural Networks. In *International Conference on Learning Representations (ICLR)*.

Garg, R., BG, V. K., Carneiro, G., and Reid, I. (2016). Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *ECCV*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *NIPS*.

Gordon, D., Farhadi, A., and Fox, D. (2017). Re3: Real-time recurrent regression networks for object tracking. *arXiv:1705.06368*.

Grabner, H., Leistner, C., and Bischof, H. (2008). Semi-supervised on-line boosting for robust tracking. In *ECCV*.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. (2018). Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. In *International Conference on Learning Representations (ICLR)*.

Grosse, R. and Martens, J. (2016). A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning (ICML)*.

Hare, S., Golodetz, S., Saffari, A., Vineet, V., Cheng, M.-M., Hicks, S. L., and Torr, P. H. S. (2015). Struck: Structured output tracking with kernels. *TPAMI*.

He, K., Wang, Y., and Hopcroft, J. (2016a). A powerful generative model using random weights for the deep image representation. In *NIPS*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778.

Held, D., Thrun, S., and Savarese, S. (2016). Learning to track at 100 fps with deep regression networks. In *ECCV*.

Henriques, J. F., Caseiro, R., Martins, P., and Batista, J. (2015). High-speed tracking with kernelized correlation filters. *TPAMI*.

Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In *International Conference on Artificial Neural Networks*.

Hochreiter, S., Younger, A. S., and Conwell, P. R. (2001). Learning to learn using gradient descents. *ICANN*.

Hoiem, D., Efros, A. A., and Hebert, M. (2005). Automatic photo pop-up. *ACM Transactions on Computer Graphics (TOG)*.

Hong, Z., Chen, Z., Wang, C., Mei, X., Prokhorov, D., and Tao, D. (2015). Multi-store tracker (muster): a cognitive psychology inspired approach to object tracking. In *CVPR*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial transformer networks. In *NIPS*.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2016). Spatial transformer networks. In *NeurIPS*.

Jasper Snoek, Oren Rippel, K. S. R. K. N. S. N. S. M. M. A. P. P.-R. P. A. (2015). Scalable bayesian optimization using deep neural networks. In *ICML*.

Ji, D., Kwon, J., McFarland, M., and Savarese, S. (2017). Deep view morphing. In *CVPR*.

Johnson, J., Alahi, A., and Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In *ECCV*.

Kahou, S. E., Michalski, V., and Memisevic, R. (2017). RATM: Recurrent attentive tracking model. *CVPR Workshop*.

Kalal, Z., Mikolajczyk, K., and Matas, J. (2010). Tracking-learning-detection. *TPAMI*.

Kang, S. B. and Shum, H.-Y. (2000). A review of image-based rendering techniques.

Kholgade, N., Simon, T., Efros, A., and Sheikh, Y. (2014). 3d object manipulation in a single photograph using stock 3d models. *ACM Transactions on Computer Graphics (TOG)*.

Kim, T., Yoon, J., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian Model-Agnostic Meta-Learning. In *Neural Information Processing Systems (NeurIPS)*.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Kolda, T. G. and Bader, B. W. (2009). Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500.

Kossaif, J., Lipton, Z., Khanna, A., Furlanello, T., and Anandkumar, A. (2018). Tensor Regression Networks. *arXiv:1707.08308*.

Kristan, M., Leonardis, A., Matas, J., Felsberg, M., and et al (2015). The visual object tracking vot2015 challenge results.

Kristan, M., Leonardis, A., Matas, J., Felsberg, M., and et al (2016). The visual object tracking vot2016 challenge results. In *ECCV Workshop*.

Kulkarni, T. D., Whitney, W. F., Kohli, P., and Tenenbaum, J. B. (2015). Deep convolutional inverse graphics network. In *NIPS*.

Kwak, H. and Zhang, B.-T. (2016). Generating images part by part with composite generative adversarial networks. *arXiv:1607.05387*.

Lamb, A., Dumoulin, V., and Courville, A. (2016). Discriminative regularization for generative models. *arXiv:1602.03220*.

Larsen, A. B. L., Snderby, S. K., Larochelle, H., and OleWinther (2016). Autoencoding beyond pixels using a learned similarity metric. In *ICML*.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-Learning with Differentiable Convex Optimization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Li, H., Li, Y., and Porikli, F. (2014). Deeptrack: Learning discriminative feature representations by convolutional neural networks for visual tracking. In *BMVC*.

Li, K. and Malik, J. (2017). Learning to optimize. In *ICLR*.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-SGD: Learning to learn quickly for few shot learning. *arXiv:1707.09835*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *ECCV*.

Lopez-Paz, D. and Ranzato, M. (2017). Gradient Episodic Memory for Continual Learning. In *Neural Information Processing Systems (NeurIPS)*.

Luca Franceschi, Paolo Frasconi, S. S. R. G. M. P. (2018). Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *International Conference on Machine Learning (ICML)*.

Ma, C., Huang, J.-B., Yang, X., and Yang, M.-H. (2015a). Hierarchical convolutional features for visual tracking. In *ICCV*.

Ma, C., Yang, X., Zhang, C., and Yang, M.-H. (2015b). Long-term correlation tracking. In *CVPR*.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Gradient-based hyperparameter optimization through reversible learning. In *ICML*.

Martens, J. and Grosse, R. (2015). Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *International Conference on Machine Learning (ICML)*.

Mathieu, M., Couprie, C., and LeCun, Y. (2016). Deep multi-scale video prediction beyond mean square error. In *ICLR*.

Max Jaderberg, Valentin Dalibard, S. O. W. M. C. J. D. A. R. O. V. T. G.-I. D. K. S. C. F. K. K. (2017). Population based training of neural networks. *arXiv:1711.09846*.

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, C. D., and Sohl-Dickstein, J. (2018). Learned optimizers that outperform sgd on wall-clock and test loss. *arXiv:1810.10180*.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2017). Unrolled generative adversarial networks. In *ICLR*.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018a). A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations (ICLR)*.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018b). A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations (ICLR)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Muandet, K., Fukumizu, K., Dinuzzo, F., and Schlkopf, B. (2012). Learning from Distributions via Support Measure Machines. In *Neural Information Processing Systems (NeurIPS)*.

Mueller, M., Smith, N., and Ghanem, B. (2017). Context-aware correlation filter tracking. In *CVPR*.

Munkhdalai, T., Yuan, X., Mehri, S., and Trischler, A. (2018). Rapid Adaptation with Conditionally Shifted Neurons. In *International Conference on Machine Learning (ICML)*.

Nam, H. and Han, B. (2016). Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*.

Newell, A., Yang, K., and Deng, J. (2016). Stacked hourglass networks for human pose estimation. In *ECCV*.

Nichol, A., Achiam, J., and Schulman, J. (2018). On First-Order Meta-Learning Algorithms. *arXiv:1803.02999*.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer Science & Business Media.

Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation.

Olga Wichrowska, Niru Maheswaranathan, M. W. H. S. G. C. M. D. N. d. F. J. S.-D. (2017). Learned optimizers that scale and generalize. In *ICML*.

Ollivier, Y. (2017). Online Natural Gradient as a Kalman Filter. *arXiv:1703.00209*.

Oreshkin, B. N., Rodriguez, P., and Lacoste, A. (2018). TADAM: Task dependent adaptive metric for improved few-shot learning. In *Neural Information Processing Systems (NeurIPS)*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Pathak, D., Krähenbühl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting deepak. In *CVPR*.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160.

Qiao, S., Liu, C., Shen, W., and Yuille, A. (2018). Few-Shot Image Recognition by Predicting Parameters from Activations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *ICLR*.

Rematas, K., Nguyen, C., Ritschel, T., Fritz, M., and Tuytelaars, T. (2016). Novel views of objects from a single image. *arXiv:1602.00328*.

Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J. B., Larochelle, H., and Zemel, R. S. (2018). Meta-Learning for Semi-Supervised Few-Shot Classification. In *International Conference on Learning Representations (ICLR)*.

Ren, S., He, K., Girshick, R., and Sun, J. (2016). Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., and Tesauro, G. (2019). Learning to Learn without Forgetting By Maximizing Transfer and Minimizing Interference. In *International Conference on Learning Representations (ICLR)*.

Rothfuss, J., Lee, D., Clavera, I., Asfour, T., and Abbeel, P. (2019). Promp: Proximal meta-policy search. In *International Conference on Learning Representations (ICLR)*.

Roux, N. L., Manzagol, P.-A., and Bengio, Y. (2008). Topmoumoute online natural gradient algorithm. In *Neural Information Processing Systems (NeurIPS)*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *IJCV*.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2019). Meta-Learning with Latent Embedding Optimization. In *International Conference on Learning Representations (ICLR)*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and chen, X. (2016). Improved techniques for training gans. In *NIPS*.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In *ICML*.

Schmidhuber, J. (1987). Evolutionary principles in self-referential learning. *Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*.

Schmidhuber, J. (1992). Learning to control fast-weight memories: an alternative to dynamic recurrent networks. *Neural Computation*.

Schulman, J., Levine, S., Moritz, P., and Michael I. Jordan, P. A. (2015). Trust region policy optimization. In *International Conference on Machine Learning (ICML)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv:1707.06347*.

Schwartz, E., Karlinsky, L., Shtok, J., Harary, S., Marder, M., Feris, R., Kumar, A., Giryes, R., and Bronstein, A. M. (2018). Delta-encoder: an effective sample synthesis method for few-shot object recognition. In *Neural Information Processing Systems (NeurIPS)*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Snell, J., Swersky, K., and Zemel, R. S. (2017). Prototypical Networks for Few-shot Learning. In *Neural Information Processing Systems (NeurIPS)*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *NeurIPS*.

Song, Y., Ma, C., Gong, L., Zhang, J., Lau, R., and Yang, M.-H. (2017). CREST: Convolutional residual learning for visual tracking. In *ICCV*.

Su, H., Qi, C. R., Li, Y., and Guibas, L. J. (2015). Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *ICCV*.

Sung, F., Yang, Y., Zhang, L., Xiang, T., and Philip H.S. Torr, T. M. H. (2018). Learning to Compare: Relation Network for Few-Shot Learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tao, R., Gavves, E., and Smeulders, A. W. M. (2016). Siamese instance search for trackin. In *CVPR*.

Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2016). Multi-view 3d models from single images with a convolutional network. In *ECCV*.

Thrun, S. and Pratt, L. (1998). Learning to learn: introduction and overview. *Springer*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*.

Ulyanov, D., Lebedev, V., Vedaldi, A., and Lempitsky, V. (2016). Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*.

Ustyuzhaninov, I., Brendel, W., Gatys, L., and Bethge, M. (2016). Texture synthesis using shallow convolutional networks with random filters. *arXiv:1606.00021*.

Valmadre, J., Bertinetto, L., Henriques, J. F., Vedaldi, A., and Torr, P. H. S. (2017). End-to-end representation learning for correlation filter based tracking. In *CVPR*.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *arXiv:1609.03499*.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. In *ICML*.

Varley, J., DeChant, C., Richardson, A., Nair, A., Ruales, J., and Allen, P. (2016). Shape completion enabled robotic grasping. *arXiv:1609.08546*.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching Networks for One Shot Learning. In *Neural Information Processing Systems (NeurIPS)*.

Wang, Y.-X., Girshick, R., Hebert, M., and Hariharan, B. (2018). Low-Shot Learning from Imaginary Data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wang, Y.-X. and Hebert, M. (2016). Learning to Learn: Model regression networks for easy small sample learning. In *ECCV*.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

Wexler, Y., Shechtman, E., and Irani, M. (2007). Space-time completion of video. *TPAMI*.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., de Freitas, N., and Sohl-Dickstein, J. (2017a). Learned optimizers that scale and generalize. In *ICML*.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., de Freitas, N., and Sohl-Dickstein, J. (2017b). Learned Optimizers that Scale and Generalize. In *International Conference on Machine Learning (ICML)*.

Wu, J., Zhang, C., Xue, T., Freeman, W. T., and Tenenbaum, J. B. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*.

Yang, J., Reed, S., Yang, M.-H., and Lee, H. (2015). Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *NIPS*.

Yang, T. and Chan, A. B. (2017). Recurrent filter learning for visual tracking. In *ICCV*.

Yuhuai Wu, Mengye Ren, R. L. R. G. (2018). Understanding short-horizon bias in stochastic meta-optimization. *arXiv:1803.02021*.

Yumer, M. E. and Mitra, N. J. (2016). Learning semantic deformation flows with 3d convolutional networks. In *ECCV*.

Zagoruyko, S. and Komodakis, N. (2016). Wide Residual Networks. In *The British Machine Vision Conference (BMVC)*.

Zhang, K., Zhang, L., Liu, Q., and andMing Hsuan Yang, D. Z. (2014). Fast visual tracking via dense spatio-temporal context learning. In *ECCV*.

Zhou, T., Tulsiani, S., Sun, W., Malik, J., and Efros, A. A. (2016). View synthesis by appearance flow. In *ECCV*.

Zoph, B. and Le, Q. V. (2017). Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning Transferable Architectures for Scalable Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.