Making Computer Vision Models Robust and Adaptive

Présentée le 4 décembre 2023
Faculté informatique et communications
Laboratoire d'intelligence et d'apprentissage visuels
Programme doctoral en informatique et communications
pour l'obtention du grade de Docteur ès Sciences
par
Shuqing Teresa YEO

Acceptée sur proposition du jury
Prof. A. M. Alahi, président du jury
Prof. A. Roshan Zamir, Prof. P. Dillenbourg, directeurs de thèse
Prof. M. Cord, rapporteur
Prof. P. W. Koh, rapporteur
Dr M. Salzmann, rapporteur
Acknowledgements

This thesis is the result of the help and support of my advisors, collaborators, lab members, friends, and family. To all of whom, I am immensely grateful.

I would first like to thank my advisor Amir Zamir. Amir took a chance on me when I was a lost and clueless PhD student. He taught me how to do research, how to present (with very, very long comments on every paper, and slide deck, including comments on the redundancy of the word very) and created a wonderfully collaborative and supportive lab environment. Amir would hold us to standards higher than ourselves, and it has greatly influenced the way I think about research directions and approach problems. I am also thankful to my co-advisor Pierre Dillenbourg. Pierre offered me a lifeline when I needed one, was a listening ear all these years, and saw me through many ups and downs. Both Amir and Pierre saved my PhD and I would not be here without them.

I am honoured to have on my jury, Alexandre Alahi, Mathieu Cord, Pang Wei Koh, and Mathieu Salzmann. I am grateful for their time, discussions, and feedback.

To the members of VILAB for teaching me invaluable life-long lessons, especially all things food related like how not to eat dumplings: to Oguzhan, the deputy boss of the lab, Roman, who has so many cool cameras, Andrei, the best kimchi maker in the world. To other long-term residents: Ainaz, David, Marius, Jiawei, and the new students: Alvin, Jason, Kunal, Rishu, thank you for making the lab lively. To those I had the pleasure of working with: Zahra, Alex, Ruchira, Harold, Pooya, thank you for making deadlines more enjoyable. Finally, to Stephanie, for making administrative work seem easy.

I am also grateful to have been adopted by the (then) giant and incredibly social CHILI family: Alexis, Arzu, Ayberk, Elmira, Hala, Jauwairia, Jenny, Kevin, Louis, Ramtin, Sina, Stian, Thibault, Łukasz, Utku, Wafa. Special thanks to Kevin and Ramtin, who kept me going with their wise words and lame jokes respectively. Hala, for the delicious Lebanese desserts and for being contagiously cheerful. During my stint at LIONS, I was fortunate to have worked closely with Kamal. Kamal would patiently distill complicated concepts for me while sharing happy memories from his PhD days. And to Chen, for company amidst the misery. I am also thankful for the people who were always up for lunch, dinner, coffee, drinks, bouldering, or on rare occasions, hikes for making my time here eventful. And very much less grateful that almost everyone graduated and left so early.
Acknowledgements

Finally, to those most affected by my constant stress and lousy schedule. They were an unwavering source of support when the going gets tough and the most ardent cheerleaders when the tough gets going. To my parents, who are always worried that I am not eating enough, and to Federico, for making sure that I am. I could not have done it without you. This thesis is for you.

Teresa
Abstract

Visual perception is indispensable for many real-world applications. However, perception models deployed in the real world will encounter numerous and unpredictable distribution shifts, for example, changes in geographic locations, motion blur, and adverse weather conditions, among many others. Thus, to be useful in the real world, these models need to generalize to the complex distribution shifts that can occur. This thesis focuses on three directions aimed at achieving this goal.

For the first direction, we introduce two robustness mechanisms. They are training-time mechanisms as inductive biases are incorporated at training-time and at test-time, the weights of the models are frozen. The first robustness mechanism we introduce ensembles predictions from a diverse set of cues. As each cue responds differently to a distribution shift, we adopt a principled way of merging these predictions and show that it can result in a final robust prediction. The second mechanism is motivated by the rigidity and biases of existing datasets. Examples of dataset biases include containing mostly scenes from developed countries, professional photographs, and so on. Here, we aim to control pre-trained generative models to generate targeted training data to account for these biases, that we can use to fine-tune our models.

Training-time robustness mechanisms attempt to anticipate the shifts that can occur. However, distribution shifts can be unpredictable and models may return unreliable predictions if this shift was not accounted for at training time. Thus, for our second direction, we propose to incorporate test-time adaptation mechanisms so that models can adapt to shifts as they occur. To do so we create a closed-loop system that learns to use feedback signals computed from the environment. We show that this system is able to adapt efficiently at test time.

For the last direction, we introduce a benchmark for testing models on realistic shifts. These shifts are attained from a set of image transformations that take the geometry of the scene into account. Thus, they are more likely to occur in the real world. We show that they can expose the vulnerabilities of existing models.

Keywords: distribution shifts, robustness, adaptation, benchmarks
Résumé

La perception visuelle est indispensable pour de nombreuses applications du monde réel. Cependant, les modèles de perception déployés dans le monde réel seront confrontés à de nombreux changements de répartition imprévisibles, par exemple des changements de localisation géographique, du flou de mouvement et des conditions météorologiques défavorables, entre autres. Ainsi, pour être utiles dans le monde réel, ces modèles doivent être généralisés aux changements de distribution complexes qui peuvent survenir. Cette thèse se concentre sur trois directions visant à atteindre cet objectif.

Pour la première direction, nous introduisons deux mécanismes de robustesse. Ce sont des mécanismes de temps de formation car des biais inductifs sont incorporés au moment de la formation et au moment du test, les poids des modèles sont gelés. Le premier mécanisme de robustesse consiste à introduire des prédictions d’ensembles à partir d’un ensemble diversifié d’indices. Ceci est basé sur l’idée que les prédictions faites via différents signaux réagissent différemment à un changement de distribution, on devrait donc pouvoir les fusionner en une seule prédiction finale robuste. Le deuxième mécanisme est motivé par la rigidité et les biais des ensembles de données existants. Des exemples de biais dans les ensembles de données incluent le fait qu’ils contiennent principalement des scènes provenant de pays développés, des photographies professionnelles, etc. Ici, nous visons à contrôler des modèles génératifs pré-entraînés pour générer des données d’entraînement ciblées pour tenir compte de ces biais, que nous pouvons utiliser pour affiner nos modèles.

Les mécanismes de robustesse du temps de formation tentent d’anticiper les changements qui peuvent survenir. Cependant, les changements de distribution peuvent être imprévisibles et les modèles peuvent renvoyer des prédictions peu fiables si ce changement n’a pas été pris en compte au moment de la formation. Ainsi, pour notre deuxième direction, nous proposons d’incorporer des mécanismes d’adaptation au temps de test afin que les modèles puissent s’adapter aux changements au fur et à mesure qu’ils se produisent. Pour ce faire, nous créons un système en boucle fermée qui apprend à utiliser les signaux de rétroaction calculés à partir de l’environnement. Nous montrons que ce système est capable de s’adapter efficacement au moment du test.

Pour la dernière direction, nous introduisons un benchmark pour tester des modèles sur des changements réalistes. Ces décalages sont obtenus à partir d’un ensemble de
Résumé

transformations d’image qui prennent en compte la géométrie de la scène. Ils sont donc plus susceptibles de se produire dans le monde réel. Nous montrons qu’ils peuvent exposer les vulnérabilités des modèles existants.

Mots-clés : décalages de distribution, robustesse, adaptation, benchmarks
Contents

2.4.2 Fine-tuning on generated data .......................... 31
2.5 Discussion on target domain informed vs uninformed generation .... 32
2.6 Conclusion .............................................. 33

II Adaptation Mechanisms ........................................ 35

3 Fast adaptation using test-time feedback ........................ 37
  3.1 Introduction ............................................. 37
  3.2 Related Work ........................................... 37
  3.3 Method .................................................. 40
    3.3.1 How to adapt at test-time? ......................... 40
    3.3.2 Which test-time adaptation signals to use? .... 42
  3.4 Experiments ............................................ 43
    3.4.1 Experimental Setup ................................. 43
    3.4.2 Adaptation with RNA vs TTO .................... 46
    3.4.3 Experiments using Various Target Tasks ........ 48
    3.4.4 Ablations and additional results ................. 51
  3.5 Discussions of RNA compared to other approaches .......... 52
  3.6 Conclusion and Limitations ................................ 55

III Benchmarks .................................................. 57

4 3D Common Corruptions ......................................... 59
  4.1 Introduction ............................................. 59
  4.2 Related Work ........................................... 60
  4.3 Generating 3D Common Corruptions ....................... 63
    4.3.1 Corruption Types .................................. 63
    4.3.2 Starter 3D Common Corruptions Dataset .......... 65
    4.3.3 Applying 3DCC to standard vision datasets ...... 66
  4.4 3D Data Augmentation ................................... 66
  4.5 Experiments ............................................. 66
    4.5.1 Preliminaries ...................................... 67
    4.5.2 3D Common Corruptions Benchmark ............... 68
    4.5.3 3D data augmentation to improve robustness ...... 74
  4.6 Conclusion and Limitations ................................ 75

Conclusion ..................................................... 77
  Limitations and Future Work ................................... 77

A Appendix ..................................................... 81
  A.1 Ensembling diverse predictions ......................... 82
## Contents

A.1.1 Quantitative Results ........................................ 82  
A.1.2 Qualitative Results ......................................... 86  
A.1.3 Further Method Details ...................................... 86  
A.1.4 Middle domain definitions .................................. 87  
A.1.5 Visualizations of Common Corruption on Taskonomy data . 88  
A.2 Fast adaptation using test-time feedback .................... 100  
  A.2.1 Monocular Depth ........................................... 100  
  A.2.2 Optical Flow Experiments ................................ 103  
  A.2.3 Dense 3D Reconstruction .................................. 104  
  A.2.4 Semantic Segmentation .................................... 104  
  A.2.5 Image classification ....................................... 106  
A.3 3D Common Corruptions ....................................... 127  
  A.3.1 Quantitative Results ...................................... 127  
  A.3.2 Qualitative Results ....................................... 128  
  A.3.3 Further method details .................................... 129  
  A.3.4 Visualizing Corruptions ................................... 130  

Bibliography ......................................................... 165  
B Curriculum Vitae .................................................. 167
Introduction

Visual perception models have been adopted in many real-world applications. These applications range from safety-critical ones like autonomous driving, and diagnosing diseases, to public policy and hiring decisions. However, when faced with distribution shifts from its training data, the predictions of these models are unreliable and can exhibit unintended biases. As the real world consists of rich, complex, and unpredictable distribution shifts, this is a challenge that needs to be solved before deploying these models in the real world.

This thesis presents three directions for improving the predictions of models under distribution shifts. The first involves robustness mechanisms. These mechanisms incorporate inductive biases into the model during training-time, in anticipation of the shifts that can occur at test time. The second direction studies adaptation mechanisms, which allows the model to adapt to shifts as they occur. Finally, for the last direction, we introduce a benchmark that can be used to evaluate models. See Figure 1 for a schematic overview of these directions.

Types of distribution shifts

One way to categorize distribution shifts is by how the distributions differ at training and test-time. Let us assume that the input data can be decomposed into a set of attributes. Thus, distribution shifts arise when the distribution over these attributes is not the same at training and test-time [1]. This gives rise to several types of shifts that commonly occur in the real world. 1. Unseen shifts: This occurs when certain attributes of the data are not seen during training but appear at test time e.g. noise or blur. 2. Long-tails: Some examples of this are when data from certain regions or demographics are under-sampled. 3. Spurious correlations: This occurs when the attributes that are correlated at training time are not correlated at test time e.g., an object that occurs with a certain background during training time may appear with a different background at test time. A model that learns to make predictions based on the background would perform poorly at test time.

Another way to categorize distribution shifts is by how they are generated. Popular
Reasons for lack of robustness

I. Robustness mechanisms are training-time mechanisms as they incorporate inductive biases into the model at training time and at test time the weights of the model are frozen. Ensemble methods combine the predictions from several models to get a final strong prediction. We propose to apply transformations to the input $x$ to get a set of cues, $\hat{x}$s. We then train a model to predict the target task from each of these cues. As each cue responds differently to distribution shifts, we adopt a principled way of merging these predictions and show that it can result in a final robust prediction. In Chapter 4, we introduce a set of realistic transformations i.e., $x \rightarrow \hat{x}$, and show that augmenting the training data with these transformations improves robustness. However, these augmentations are generated in an open-loop manner, i.e., without feedback from the model. Thus, in Chapter 2, instead of applying transformations to the input, we learn to control a generative model to generate training data for a given model by using its loss as a feedback signal. II. Adaptation mechanisms are test-time mechanisms as they aim to adapt to distribution shifts as they occur. To do so, they compute an adaptation signal from the environment, $e$, at test time and use it to update the model. Thus, creating a closed-loop system with the environment. In Chapter 3, we propose an efficient way of performing test-time adaptation. III. Benchmarks allow us to evaluate our models before deploying them in the real world. In Chapter 4, we introduce a set of realistic corruptions, informed by the geometry of the scene. We show that they can be computed efficiently and can expose the vulnerabilities of existing networks.

Examples include low-level shifts like Gaussian blur, speckle noise [2] etc., and viewpoint changes that can result in occlusions [3, 4]. There can also be changes in geographic location [5, 6], time [7, 8] and so on. One can also optimize for shifts to fool a given model, creating an adversarial attack. The adversarial attack can be optimized to be imperceptible [9, 10, 11], over spatial transformation like rotation and translation parameters [12], or even over how the training and test data is split [13]. In this thesis, we focus mostly on unseen shifts, and in particular low-level and optimized ones.

Reasons for lack of robustness

Our training datasets tend to exhibit biases. Some examples include photographer’s bias, i.e., where photographers tend to take pictures of objects in similar ways, and as a result, most objects tend to be centered or taken in their frontal view [14, 15, 16].
example is location bias, when there is more data from certain parts of the world or ethnicity [17] compared to others e.g., satellite images from the Americas compared to that from Africa [18]. Thus, training our models on data with these biases can result in poor performance at test-time, when it is given a e.g., different viewpoint of an object or images from different countries. Furthermore, learning with stochastic gradient descent can return models that prefer certain attributes over others. [19] showed that networks exhibit simplicity bias, i.e., when there exist several attributes of the input data that are predictive of the target task, networks prefer to use the “simpler” attributes. These “simpler” attributes are e.g., background or texture, as opposed to semantic features of the object. This causes the model to be sensitive to perturbations in the input or style changes.

There are many other hypotheses for why our models are not robust, from learning from unnatural static I.I.D. data, typically of a single modality, being disconnected from the environment i.e., lack of interaction, and so on. A discussion of this can be found in the conclusion section, under future work. Thus, the above paragraph presents a narrow view of the possible reasons, as it assumes that we are constrained to training black box neural networks on I.I.D. inputs.

Making models robust

Robustness mechanisms anticipate the distribution shift that can occur and incorporate inductive biases into the model to help it generalize. They are training-time mechanisms as these inductive biases are incorporated at training-time and at test-time the weights of the models are frozen. There are several ways of making models more robust e.g., data augmentations, architecture changes, and so on. We will discuss the ones relevant to this thesis.

Ensemble methods combine the predictions of several models to attain a strong final prediction. Ensembles rely on the diversity of individual models to de-correlate errors. Sources of diversity include using different initializations [20], hyperparameters [21] or network architectures [22] for the ensemble components, or training the ensemble with additional loss terms [23, 24, 25, 26]. Other works average the weights of several models, to return one final model [27, 28, 29]. This has the benefit of having no additional cost at inference time. In Chapter 1, we propose to make our predictions from a diverse set of cues. This is based on the idea that predictions made via different cues respond differently to distribution shifts, hence one should be able to merge them into one robust final prediction. We perform the merging in a straightforward but principled manner based on the uncertainty associated with each prediction.

Learning with data from generative models. Sampling data from generative models such as generative adversarial networks (GANs) or diffusion models allows us to vastly expand the original training data. Recent works have made use of the latter to either
Making models adaptive

replace or expand the training data for image recognition tasks [30, 31, 32]. These works used pre-defined prompts or a language model to augment the pre-defined prompts, to generate data. Other works learn to generate training data for a given model by controlling the latent space of GANs [33, 34] or variational autoencoders (VAEs) [35]. We show in Chapter 2 that we can generate data for a given model but in contrast to these works, we employ diffusion models as the data generator.

Robustness benchmarks allow us to evaluate our models before deploying them in the real world. The WILDS benchmark [18] is a combination of 10 datasets that represents shifts that can arise in the real world e.g., shifts across time, location in satellite images, or hospitals. In contrast, the Common Corruptions benchmark [2] introduced synthetic corruptions on real images that expose the sensitivities of image recognition models. These corruptions are cheap to apply, however, they do not take into account the geometry of the scene, i.e., the corruptions are applied uniformly over the image. In Chapter 4, we propose 3D Common Corruptions, that modify real images using 3D information to generate realistic corruptions. Thus, leading to corruptions that are more likely to occur in the real world.

Data-augmentations. Increasing the diversity of the training data via data augmentation is a popular way of improving the model’s robustness. Methods such as AugMix [36], AutoAugment [37], RandAugment [38] combine standard augmentations like color jitter, rotation, shearing. Methods like CutMix [39], Mixup [40] introduce new augmentations by replacing a portion of an image with a crop of another image or by linearly combining two images. Other works train with low-level shifts like noise or blur [41, 42], or adversarial attacks [11]. In Chapter 4, we introduce a set of realistic corruptions that incorporates the geometry of the scene into the transformations and show that training with these realistic corruptions results in more robust predictions.

Domain Generalization assume access to datasets from multiple domains during training [43, 44]. Some methods involve meta-learning [45], matching the features across the different domains [46, 47, 48] or enforcing invariance in the gradients [49]. Invariant risk minimization [50] learns features such that the optimal linear classifier on top of that representation matches across domains. Group distributionally robust optimization [51] minimizes the worst-case loss over domains. However, [52] showed that empirical risk minimization, when well implemented, achieves state-of-the-art performance across datasets. In this thesis, we assume access to only one domain during training, although extending this to multiple domains would be interesting for future work.

Making models adaptive

Distribution shifts that can occur in the real world are numerous and unpredictable. Training-time robustness mechanisms that attempt to take anticipatory measures (e.g., data augmentation or architecture changes) have inherent limitations. This is because the
learned model is frozen at test-time, thus upon encountering an out-of-distribution input, its predictions may collapse. This is the main motivation behind test-time adaptation methods, which instead aim to adapt to such shifts as they occur. These methods choose adaptation over anticipation.

Adaptive methods create a closed loop with the environment and use an adaptation signal at test time. The adaptation signal is a quantity that can be computed at test time from the environment. Optimizing a self-supervised signal e.g. entropy [53], mutual information [54] at test-time is a popular way of performing adaptation. While this can successfully adapt a network, it is inefficient as it does not make use of the learnable regularities in the adaptation process, and consequently, is unconducive for real-world applications. It also results in a rigid framework as the update mechanism is fixed to be the same as the training process of neural networks, i.e., using stochastic gradient descent. In Chapter 3, we show that this process can be effectively amortized using a controller network, which yields orders of magnitude faster results. In addition, it provides flexibility advantages as the controller is implemented using a neural network and can be engineered to include arbitrary inductive biases and desired features.
Robustness Mechanisms Part I
1 Ensembling diverse predictions

1.1 Introduction

We begin the first part on robustness mechanisms by demonstrating a way to use ensembling techniques to improve predictions. Suppose we want to learn a mapping from an input domain, e.g. RGB images, to a target domain, e.g. surface normals (see Fig. 1.1). A common approach is to learn this mapping with a direct path, i.e. \( \text{RGB} \rightarrow \text{surface normals} \). Since this path directly operates on the input domain, it is prone to being affected by any slight alterations in the RGB image, e.g. brightness changes. An alternative can be to go through a middle domain\(^1\) that is invariant to that change. For example, the surface normals predicted via the \( \text{RGB} \rightarrow 2D \text{ edges} \rightarrow \text{surface normals} \) path will be resilient to brightness distortions in the input as the 2D edges domain abstracts that away. However, one does not know which middle-domain to use ahead of time as the distortions that a model may encounter are broad and apriori unknown, and some middle domains can be too lossy for certain downstream predictions. These issues can be mitigated by employing an ensembling approach where predictions made via a diverse set of middle domains are merged into one strong prediction on-the-fly.

This paper presents a general approach for the aforementioned process. We first use a set of \( K \) middle domains from which we learn to predict the final domain (Fig. 1.1).

Prior knowledge of the relationship between middle domains is not needed as their

---

\(^1\)or equivalently “middle task”, as most vision tasks can be viewed as mapping an input onto some other domain.
Chapter 1. Ensembling diverse predictions

Figure 1.1: **An overview of the proposed method for creating a robust and diverse ensemble of predictions.** A set of $K$ networks predict a target domain (here surface normals) given an input image that has undergone an unknown distribution shift (here JPEG compression degradation), via $K$ middle domains (e.g. 2D texture edges, low-pass filtering, greyscale image, emboss filtering, etc). The predictions from the $K$ paths are then merged into one final strong prediction using weights that are based on the uncertainty associated with each prediction. This method is shown to be significantly robust against adversarial and non-adversarial distribution shifts for several tasks. In the figure above, solid (-) and dashed (dash-dot) arrows represent learned and analytical functions, respectively.

contribution to the final prediction is guided by their predicted uncertainties in a fully computational manner independent of the definition of the middle domain. In other words, no manual modification or re-design is needed upon a change in these domains. Moreover, the middle domains we adopt can all be *programmatically extracted*. Thus, this framework does not require any additional supervision/labeling than what a dataset already comes with. The proposed method would be equally applicable if the middle domains were also obtained using a learning based approach, e.g. predicting surface normals from the output of another network such as a depth estimator. We show in Sec. 4.5 that the method performs well insensitive to the choice of middle domains and it generalizes to completely novel non-adversarial and adversarial corruptions.

1.2 Related Work

This work has connections to a number of topics, including ensembling, uncertainty estimation and calibration, inductive bias learning [55], or works in neuroscience that suggest the brain uses multiple, sometimes partially redundant, cues to perceive [56, 57]. We give an overview of some of them within the constraints of space.

**Ensembling** allows us to resolve the bias-variance trade-off which states that errors
in a models’ prediction can be decomposed into bias, variance, and an irreducible data-dependent noise term \[58, 59\]. This is done by combining multiple models with low bias and high variance, e.g. bagging [60], or with high bias and low variance, e.g. boosting [60], to have predictions with both low bias and variance.

A primary challenge for ensemble methods is to ensure diversity. Sources of diversity include using different initializations [20], hyperparameters [21] or network architectures [22] for the ensemble components, or training the ensemble with additional loss terms [23, 24, 25]. However, under distribution shifts, reduction in performance can stem from an increase in the bias, rather than the variance term [61]. Our set of middle domains yields a more diverse ensemble by design and promotes invariance to different distortions to keep bias low (see Fig. 1.1).

**Estimating uncertainty:** Uncertainty in a model’s prediction can be decomposed into two sources [62, 63]. Epistemic uncertainty accounts for uncertainty in the model parameters, while aleatoric uncertainty stems from the noise inherent in the data. There are many proposed methods to estimate the former, such as using dropout [64, 65], stochastic variational inference methods [66, 67, 68, 69, 70, 71], ensembling [20], and consistency energy [72] where a single uncalibrated uncertainty estimate is extracted from consistency of different paths. Most of the existing methods in this area solely estimate uncertainty without using it towards improving the predictions. In contrast, our formulation estimates a calibrated uncertainty for each path and uses it to produce a stronger prediction.

**Improving calibration with auxiliary datasets:** Neural networks tend to produce outputs that are miscalibrated, i.e. their estimated uncertainty does not reflect the true likelihood of being correct [73, 74]. In particular, their predictions tend to be overconfident for unfamiliar examples. This is usually handled by a calibration step. Similar to [75, 76, 77, 78], we use a separate dataset from the one at test time to train the model to output high sigmas (uncertainties) for unfamiliar cases. Previous papers focus on generalizing uncertainties for classification; in Section 1.3.1, we show this can be extended to dense regression problems.

**Enforcing consistency constraints** in the context of cross-task predictions involves ensuring that the output predictions remain the same regardless of the intermediate domain [72, 79, 80, 81]. Particularly in contrast to [72] which uses (non-probabilistic) training-time consistency constraints to improve a network’s prediction and does not have any consolidation mechanism, our goal is to robustify the final prediction by merging the output of multiple prediction paths at the test time. Our formulation and the training-time consistency constraints are complimentary.

**Robustness via data augmentation:** One approach to addressing robustness involves using data augmentation during training [11, 40, 36, 82, 83]. Such methods usually
Chapter 1. Ensembling diverse predictions

involve training with a set of corruptions to generalize to unseen ones [41]. However, performance gains can be non-uniform, e.g. Gaussian noise augmentation improves performance on other noise corruptions (e.g. impulse, shot noise) but hurts performance on fog and contrast [84]. Instead, our main mechanism uses a large set of middle domains (not corruptions) to be resistant to a wide range of diverse unseen corruptions. We do not use any corruptions during training, except to calibrate uncertainty.

Adversarial attacks add imperceptible worst case shifts to the input to fool a model [9, 10, 11]. In contrast to [23, 24, 25], which are ensemble based adversarial robustness methods with an additional loss term to promote diversity, the diversity of our ensembles is a natural consequence of using different middle domains. While our focus is not limited to robustness against adversarial attacks, it yields supportive evaluations against them as well (Sec. 1.4.1).

Figure 1.2: Addressing overconfident inaccurate predictions under high distortions. Left: Qualitative prediction results of image (re)shading and their corresponding uncertainty estimates (i.e. sigma) under two intensities of speckle noise distortion. This is shown for a single UNet model before and after sigma training (ST) as well as for deep ensembles (standard deviation of predictions in the ensemble). Darker denotes lower uncertainty/sigma. ST was done using Gaussian noise and Gaussian blur distortions. Using other distortions yields similar performance (see appendix). Right: Scatter plot of $\ell_1$ error versus average sigma. Each point is computed from an average over 16k test images for one of the unseen distortions and one of 5 levels of shift intensity. Notice (qualitatively and quantitatively) that when the models without ST produce poor results, their uncertainty does not correspondingly increase. Our ST helps the model to have a stronger correlation between its uncertainty estimates and error when tested on unseen distortions. This indicates that sigma after ST can be an effective signal for merging multiple predictions. Note that the predicted mean ("Prediction") does not change with ST.

1.3 Method

We explain the technical details of our method below.

Notations: Define $X$ as the RGB domain, $Y = \{Y_j\}_{j=1}^K$ as the $K$ intermediate domains, $Z$ as the desired prediction domain. A single datapoint $n$ from these $K$ domains is denoted as $(x_n, y_{1,n}, \ldots, y_{j,n}, \ldots, y_{K,n}, z_n)$. $\mathcal{F}_{XY}$ is the set of functions that maps the
1.3 Method

RGB images to their intermediate domains, \( \mathcal{F}_{XY} = \{ f_j : X \rightarrow Y_j \}_{j=1}^K \), and \( \mathcal{F}_{YZ} \) is the set of functions mapping from the intermediate to the target prediction domain, \( \mathcal{F}_{YZ} = \{ g_j : Y_j \rightarrow Z \}_{j=1}^K \). Given \( K \) predictions of domain \( Z \), they are merged using the function \( m \) to get a final single prediction, \( m : \{ g_j(Y_j) \}_{j=1}^K \rightarrow Z \).

1.3.1 Estimating Per-Path Predictions and Uncertainty

We learn the mappings \( g_j \) using a neural network. We model the noise in the predictions made by \( g_j \) with a Laplace distribution. Thus, for an input sample \( y_{j,n} \), the network outputs two sets of parameters \([\hat{z}_{j,n}, \hat{s}_{j,n}] = g_j(y_{j,n})\) where we set \( \hat{s}_{j,n} = \log \hat{b}_{j,n} \) for numerical stability and \( \hat{b}_{j,n} \) is the scale parameter of the Laplace distribution. We remove the dependence on \( j \) for brevity. This leads to the following negative log-likelihood (NLL) loss for \( g \):

\[
L_{g,NLL} = \frac{1}{N} \sum_{n=1}^{N} \exp\left( -\hat{s}_n \right) \| \hat{z}_n - z_n \|_1 + \hat{s}_n, \tag{1.1}
\]

where \( N \) is the number of samples, and \( z_n \) is the label for the \( n \)th sample. This results in an \( \ell_1 \)-norm loss on the errors as opposed to an \( \ell_2 \)-norm loss with a Gaussian distribution, which has been shown to improve prediction quality [63, 72]. Finally, the sigma is given by \( \sqrt{2} \exp(\hat{s}_n) \) and it captures the uncertainty in predictions.

**Calibration via Sigma training (ST):** Uncertainty estimates under distribution shifts are poorly calibrated [85], i.e. there is a tendency to output a poor prediction with high confidence. This can be seen in Fig. 1.2, “Before sigma training” columns. With a higher noise distortion, the prediction clearly degraded, but the uncertainty estimate did not increase correspondingly. This issue persists even with methods that estimate epistemic uncertainty (Fig. 1.2, “Deep Ensembles” columns) which are meant to detect these shifts.

To mitigate this, we adopt a two-stage training setup where the network trained on in-distribution data is further trained to output high uncertainty outside the training distribution. We denote this step as sigma training (ST). Here, \( \hat{s}_n \) is trained to learn its maximum likelihood estimator, with the loss denoted as sigma calibration (SC). As the goal of this step is to maximize the likelihood by correcting the sigma \( \hat{s}_n \), and not the mean \( \hat{z}_n \), under a distortion (\( \text{dist} \)), we add a loss term to ensure that \( \hat{z}_n \) does not deviate from its predictions at the start of ST, which we define as \( \hat{z}_{n0} \). We denote this loss as mean grounding (MG). Finally, we include the original NLL from Eq. 1.1 on undistorted data (\( \text{undist} \)) to prevent forgetting. This results in the following loss formulation:

\[
L_{g,ST} = L_{g,NLL} + \alpha_1 L_{g,MG} + \alpha_2 L_{g,SC}, \tag{1.2}
\]

where \( \alpha_1, \alpha_2 \) controls the weighting between the loss terms. For a given \( \hat{z}_{n0} \), the MG loss
Chapter 1. Ensembling diverse predictions

is defined as the $\ell_1$-norm distance between the current prediction and the one at the start of sigma training, i.e. $L_{\text{dist}}^{g,MG} = \| \hat{z}_n^0 - \hat{z}_n \|_1$. The SC loss guides the scale parameter towards its maximum likelihood estimate, i.e. $L_{\text{dist}}^{g,SC} = \| \exp(\hat{s}_n) - \arg\min_{\hat{s}_n} L_{\text{dist}}^{g,NLL}_j \|_1 = \| \exp(\hat{s}_n) - |z_n - \hat{z}_n^0| \|_1$.

Figure 1.3: **How does the method work?** Each network in each path receives different cues for making a prediction, due to going through different middle domains. **Left:** Given a distorted pixelated query, each path (columns) is affected differently by the distortion, which is reflected in its prediction, uncertainty, and weights (lighter means higher weights/uncertainty). The inverse variance merging uses the weights to assemble a final prediction that is better than each of the individual predictions. (The uncertainties of surface normals look colorful as surface normals domain includes 3 channels, thus there are 3 uncertainty channels.) **Right:** Similarly, for a query with glass blur distortion, the method successfully disregards the degraded predictions and assembles an accurate final prediction. Note that the proposed method (inverse var. merging) obtains significantly better results than learning from the RGB directly (leftmost column of each example) which is the most common approach. The quality of the final prediction depends on the following elements: 1. For each pixel, at least one middle domain is robust against the encountered distortion, and 2. The uncertainty estimates are well correlated with error, allowing the merger to select regions from the best performing paths. Uniform merging does not take into account the uncertainties and consequently lead to worse predictions. The elliptical markers denote sample regions where the merged result is better than all individual predictions.

Following ST, the network outputs sigmas that are highly correlated with error (Fig. 1.2, rightmost plot). Given multiple predictions of the same target domain and their sigma estimates, this allows us to use the latter as a signal for merging to get a single strong prediction (Sec. 1.3.2).

As the objective of ST is to expose the network to inputs with high distortions as opposed to updating the final predicted mean, *any* corruption with high intensity will suffice. The distortions used for ST are not the same distortions as the ones at test time. Please see the appendix Section 2.5 for a detailed study. Furthermore, the experiments (Fig. 1.2, Fig. 1.6, Table 1.1) indicate that sigma clearly generalizes to unseen distortions.
1.3 Method

1.3.2 Merging Predictions

After obtaining the set of mappings $\mathcal{F}_{XY}$ and $\mathcal{F}_{YZ}$ with the method described above, it remains to merge the predictions coming from multiple paths using a merging function $m$. We employ an analytical approach given by $m(\{g_j(y_{j,n})\}_{j=1}^K) = C \sum_{j=1}^K \exp(-2\hat{s}_{j,n})\hat{z}_{j,n}$ where $C$ is a normalizing constant defined as $C = \left(\sum_{j=1}^K \exp(-2\hat{s}_{j,n})\right)^{-1}$. This performs a straightforward weighting of each pixel in each path by the inverse of its variance [86] which can be done with negligible computational cost. We denoted this as Inverse variance merging and will show in Section 1.4.1 that it performs better than other analytical and learning based variants of our method. Algorithm 1 summarizes our training procedure.

**Algorithm 1** Summary of the training procedure of our method

**Require:** Define $f_j \in \mathcal{F}_{XY}$ and $g_j \in \mathcal{F}_{YZ}$ $\forall j$.

1. **for** $j = 1 : K$ **do**
2. Train $g_j$ using NLL loss in Eq. 1.1.
3. (Optional) Train $g_j$ using consistency constraints [72]. (Sec. 1.4.1)
4. Perform sigma training over $g_j$ with Eq. 1.2.
5. **end for**
6. Merge the $K$ predictions from the $\mathcal{F}_{YZ}$ networks using Inv. var. merging (Sec. 1.3.2).

A working example. Figure 1.3 illustrates our method with an example. For a given image, each path’s prediction, uncertainty, and corresponding weights are shown. For the distorted (pixelated) query in the left, each path reacted differently to the distortion, and the final prediction is obtained by combining individual predictions based on their uncertainties. Similar observations can be made for the glass blurred image in the right, where the method learned weights in a way such that the degraded paths are not
used in the final prediction. We also show the final prediction from a uniform average of each path. While it is better than simply using the direct path ($X \rightarrow Z_{\text{normal}}$ or $X \rightarrow Z_{\text{reshade}}$), using the uncertainty estimates as weights results in a notably more accurate prediction.

There are two key elements to the effectiveness of our method. I. With a diverse set of middle domains, it is more likely that one of them will be less affected by distortions and returns an accurate prediction. II. The error of the prediction correlates well with its corresponding uncertainty estimates, i.e. the uncertainty is low in the region of the image where the prediction is accurate. This allows us to use these uncertainty estimates as a signal to have a final prediction with parts of the image taken from different paths.

1.4 Experiments

We demonstrate that the proposed approach leads to robustness against different distribution shifts, over different datasets, and different prediction tasks. For pixel-wise prediction tasks, we train on the Taskonomy dataset [88]. To evaluate the robustness under corruptions, we report performance under Common Corruptions [2] and adversarial perturbations [9, 10, 11]. To evaluate against dataset shifts, we report on Replica [87] and Habitat [89] datasets. For classification, we train on ImageNet [90], CIFAR [91], and evaluate on ImageNet-C and CIFAR-C [2]. Please see the appendix for more extensive qualitative results.

1.4.1 Evaluations on Pixel-Wise Prediction Tasks

Training dataset: We use Taskonomy [88] as our training dataset which includes 4 million real images of indoor scenes with multiple annotations for each image. We report results for surface normals, depth (zbuffer), and reshading prediction, as popular target domains.

Middle Domains: From the RGB images we extract 2D edges, Laplace edges, greyscale, embossed, low-pass filtered, sharpened, and wavelet images as the middle domains (detailed definitions can be found in the appendix ). These middle domains are commonly used for low-level image processing tasks with negligible computation cost [92, 93] and do not need any supervision. The performance was not sensitive to the choice of middle domains as the method consistently outperforms baselines and improves with more middle domains (Sec. 1.4.1, Fig. 1.7).

Evaluation datasets: Our goal is to have test data that has a distribution shift from the training data to evaluate the robustness of our method. All the results are reported on the test set of the following datasets:

Taskonomy with Common Corruptions [2]: We apply the Common Corruptions on the test set of Taskonomy. They include all corruptions except outdoor corruptions
1.4 Experiments

Figure 1.5: Qualitative results under distribution shifts for reshading, surface normals, and depth predictions. Each row shows the predictions from a query image from the Taskonomy test set under increasing speckle noise. Our method degrades less than the other baselines, demonstrating the effectiveness of using different cues to obtain a robust prediction. Notable improvements in the accuracy can be seen especially in fine-grained regions.

(snow, frost, fog) and the ones that change the geometry of the scene (elastic transform, motion, and zoom blur). We exclude Gaussian noise and blur from evaluations as they were used for ST, to keep training and testing fully separate. Visualizations of a subset of distortions are shown in Figure 1.4 and for all severities in the appendix.

Taskonomy with Adversarial corruptions [9, 10, 11]: We generate adversarial examples using Iterative-Fast Gradient Sign Method (I-FGSM) [10].

Other datasets: Replica [87] consists of 1227 images from high quality 3D reconstructions of indoor scenes. Similar to Taskonomy, we also apply common corruptions on these images. Habitat [89] consists of 1116 images from mesh renderings with a substantial shift from Taskonomy. We test on both datasets without fine-tuning (see appendix).

Training details: All networks for our method and baselines use the same UNet backbone architecture [94] and were trained with AMSGrad [95]. We used a learning rate of $5 \times 10^{-4}$, weight decay of $2 \times 10^{-6}$, and batch size of 64. The upsampling blocks of all networks resize the activation maps using bilinear interpolation.

We also augment the network training with “cross-task consistency constraints” (X-TC) [72] for generally better results, but this is not a fundamental requirement (ablation results provided in Sec. 1.4.1). We follow [72] and apply non-probabilistic perceptual losses on the predicted mean.

Baselines: We evaluate the following baselines. They are trained with NLL loss (Eq. 1.1), i.e. the models output both mean and sigma.

Baseline UNet: It is a single network that maps from RGB to the target domain without going through a middle domain (i.e. direct). This is the main baseline.

Multi-domain baseline: It is a network model with RGB image and all middle domains as inputs. Since this model is not forced to use different middle domains as opposed to
Chapter 1. Ensembling diverse predictions

the proposed method, it reveals if learning from middle domains needs to be explicit and distributed.

*Multi-task baseline:* It is a single model that maps from RGB to depth, reshading, and normals. This is to reveal if learning additional tasks improved robustness.

*Data augmentation baselines:* We consider a baseline UNet adversarially trained to defend against I-FGSM attacks with $\epsilon = (0,16]$. This baseline shows how well adversarial robustness translates to non-adversarial distortions. We also include style augmentation [96] as another baseline, which has been shown to reduce the texture bias that are less robust than shape cues.

*Blind guess* is a single prediction that captures the overall statistics of the domain, i.e. it returns the best guess of what the prediction should be independent of the input. Hence, it shows what can be learned from general dataset regularities (further details are in the appendix).

*Deep ensembles* [20] creates an ensemble by training the same exact networks with different initializations. Although there are recent papers proposing new ways to enforce diversity in ensembles, their improvement in performance against deep ensembles has not been found significant under non-adversarial shifts [97, 21]. Thus, deep ensembles remains the most relevant ensemble baseline. We use the same number of paths, i.e. ensemble components, as in our method. The predictions from each path are weighted equally to attain the final prediction. This baseline reveals if learning from different cues yields diverse predictions that results in a stronger final estimator.

**Cross-domain ensemble setups evaluated:** We evaluate several variants of our merging method. In all variants, different paths goes through different middle domain to produce a prediction along with one path being the direct prediction. They are then merged into the final prediction. We show the proposed analytical merging is superior to others.

*Inverse variance merging:* Each path’s prediction is weighted inversely proportional to its variance, as proposed in Section 1.3.2.

*Uniform merging:* A simplified merging where each path is weighted equally, i.e. uncertainty is not used.

*Network merging:* A neural network is used to merge the predictions. Specifically, we consider a stacking model [98] that learns the final predictions given the outputs from each path and models the final output as a mixture of Laplacians. It has the advantage that the loss is over the entire image, thus, taking into account its spatial structure (see the appendix for details).
1.4 Experiments

Figure 1.6: Quantitative robustness evaluations using Common Corruption distortions applied on Taskonomy test set: Average $\ell_1$ errors over 11 unseen distortions. Our main method inv. var. merging, and frequently its simplified variant uniform merging and network merging, are more robust against shifts compared to the baselines. Error bars indicate one ‘standard error’ from the mean (via bootstrapping). Plots for additional perceptual error metrics and individual distortions are provided in the appendix.

Table 1.1: Robustness against adversarial corruptions. $\ell_1$ errors for surface normals, reshape, and depth under adversarial attacks are reported. (Lower is better. Errors are multiplied by 100 for readability.) The proposed method significantly improves robustness against I-FGSM [10] based attacks without adversarial training, compared to the baselines. The last row shows the error for a model that has undergone adversarial training [11] with the same attacks as those evaluated at test time, hence it gives a lower bound on the error (see supplementary for additional details).

Robustness to Common Corruptions

Figures 1.4 and 1.5 show the qualitative results of our method against the baselines. Performance under various distortions is demonstrated in Figure 1.4 for the surface normals predictions of a sample image from Replica dataset. The proposed method consistently outperforms the baselines and provides more accurate predictions especially in fine-grained regions. This is further supported by quantitative results in Figure 1.6 where the $\ell_1$ error over these distortions are notably lower for the proposed method compared to the baselines in all three target domains and shift intensities.

Among the evaluated baselines the data augmentation methods are the most competitive, e.g. adversarial robustness partially transferred to non-adversarial distortions, though inverse variance merging performs notably better.

We also observe inverse variance merging does much better than uniform merging and also better or comparable to network merging (Fig. 1.6) despite being simpler, more lightweight, and interpretable. Moreover, it does not demand fixing the number of paths.
Chapter 1. Ensembling diverse predictions

Figure 1.7: Effect of sigma and/or consistency training. (Top row) The plots show the relative change in $\ell_1$ error compared to deep ensembles (i.e. negative means outperforming deep ensembles). The proposed method outperforms deep ensembles under distribution shifts even without ST and X-TC (consistency).

Robustness as a function of number of paths. (Bottom row) The plots show the average $\ell_1$ error as we increase the number of paths (or ensemble components in the case of deep ensembles). The proposed method (inv. var. merging) and its simplified variant (uniform merging) consistently outperforms deep ensembles which plateaus much faster.

Robustness to Adversarial Attacks

We demonstrate the effectiveness of the proposed method under adversarial attacks. The attacks are generated by I-FGSM. Following [10], we use attack strengths $\epsilon = [2, 4, 8, 16]$, with the number of iterations given by $N = \min(4 + \epsilon, 1.25\epsilon)$. The results are shown in Table 1.1. Neither our method nor the baselines utilize explicit adversarial defense mechanisms – while deep ensembles perform nearly as poorly as baseline UNet, the proposed method performs significantly better. This indicates that using middle domains promotes ensemble diversity in a way that makes it more challenging to create one attack that fools all paths simultaneously, hence this approach can be a promising remedy for adversarial attacks as well. Moreover, the proposed method also outperforms Uniform merging (see the appendix for the results) which does not use uncertainty estimates during merging. This indicates that the additional uncertainty output did not create an additional avenue for attack that I-FGSM could exploit.

Note that we do not obfuscate gradients by e.g. intentionally making certain operations non-differentiable, or using stochastic transforms [99]. The analytical operations to obtain the middle domains are deterministic and differentiable.
1.4 Experiments

Figure 1.8: Importance of each middle domain for different distortions. The chart shows the order of the best performing paths for surface normal prediction for different distortions, with 8 denoting the most important and 1 the least important path. The plot shows, for instance, “noise” distortions benefited most from the low-pass middle domain while “contrast” distortion benefited most from the sharpened middle domain.

Additional Ablation Studies

Contribution of ST/X-TC: To quantify the contribution of each stage of training to the overall robustness of our setup, we study the performance of our proposed method without sigma training (ST) and/or cross-task consistency constraints (X-TC) in the first row of Figure 1.7 (and appendix). Our method, with or without ST or X-TC constraints, still outperformed deep ensembles as almost all bars are below the 0 line.

In Section 2.3 of the appendix, we compare the effect of equipping deep ensembles with ST and X-TC, and perform uniform and inverse variance merging. Thus, the only difference with our method is the use of middle domains. Our method still outperforms.

Robustness vs number of employed paths: In Figure 1.7, we investigate performance as a function of number of paths. Each point shows the average $\ell_1$ error of all possible combinations for a given number of paths. Although all methods improve as more paths are added, our proposed methods has a much steeper downward trend than deep ensembles and our uniform merging variant, indicating that performance gap increases with more paths.

Sensitivity to choice of middle domains: Figure 1.7 also shows that the performance of our method is not sensitive to a particular set of middle domains. For a fixed number of paths $n$, our method outperforms deep ensembles for all possible combinations of $n$ paths on average.

Path importance: We show the importance of each middle domain for the final prediction under each distortion in Fig. 1.8. For number of paths $n = 1, \ldots, 8$, we compute the set of best performing paths, i.e. the set of $n$ paths with the lowest $\ell_1$ error, denoted by $P_n = \{p_{(i)}\}_{i=1}^n$. The $n^{th}$ best performing path is given by $P_n \setminus P_{n-1}$. The plot shows different paths indeed react differently to a given corruption, e.g. noise distortions substantially benefited from low-pass, while contrast distortion did not – thus the benefit
Chapter 1. Ensembling diverse predictions

is not attributed to one or few middle domains under all distortions.

1.4.2 Robustness of Sigmas to Distribution Shifts

We have showed that our method returns predictions that are robust under a wide range of distribution shifts. Are our predicted uncertainties also able to generalize under distribution shifts, i.e. do we get high uncertainties when predictions get worse? To investigate this, we consider epistemic uncertainty which is used to capture the model’s uncertainty and is an indicator of distribution shifts [63]. The left plot of Figure 1.9 shows a scatter plot of average epistemic uncertainties against error and the right shows the average error for all epistemic sigma values less than a threshold \( \tau \). The predicted uncertainties from deep ensembles initially increase with error but do not increase past 0.15 sigma despite an increase in error, thus is overconfident, while our method shows an increasing trend.

![Figure 1.9: Generalization of predicted uncertainties.](image)

Performance on undistorted data

In order to demonstrate that the robustness of our method on out-of-distribution data did not come at the cost of degraded performance on in-distribution data, we provide quantitative evaluations on the undistorted Taskonomy and Replica datasets in appendix. The results show the performance of our method, when tested on undistorted data, is indeed comparable to or better than the methods that are trained to perform well only on undistorted data.

1.4.3 Evaluation on Classification Tasks

The benefits of the proposed method is not limited to regression or dense pixel-wise tasks. We performed an experiment on ImageNet-C to evaluate the robustness against Common Corruptions (Table 1.2). Our method and deep ensembles both use 8 paths with identical ResNet-50 [100] network architecture. In addition, in this experiment our method does a simple averaging of the output probabilities from each path, similar to deep ensembles, and no ST or X-TC training was involved. The superior results show the
1.5 Conclusion and Discussion

Table 1.2: **Robustness on ImageNet-C**. Error on clean and distorted data (mean Corruption Error, mCE). Following [2], the mCE is relative to AlexNet [101]. All methods are trained on clean ImageNet training data. Our method performs noticeably better under distortions compared to deep ensembles and a single model baseline ResNet. See the appendix for a detailed breakdown and additional results on CIFAR.

<table>
<thead>
<tr>
<th>Method</th>
<th>Clean error</th>
<th>mCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline ResNet-50</td>
<td>24.37</td>
<td>76.21</td>
</tr>
<tr>
<td>Deep ensembles</td>
<td>21.50</td>
<td>70.43</td>
</tr>
<tr>
<td>Ours</td>
<td>21.61</td>
<td>67.85</td>
</tr>
</tbody>
</table>

basic value in using a diverse set of middle domains. Similar conclusions were obtained for CIFAR-10-C and CIFAR-100-C datasets (full results in the appendix).

**1.5 Conclusion and Discussion**

We presented a general framework for making robust predictions based on creating a diverse ensemble of various middle domains. Experiments demonstrated that this approach indeed leads to more robust predictions compared to several baselines.

We also showed that our method is not sensitive to the choice of middle domains (Sec. 1.4.1) or the corruptions used for ST (supplementary). Furthermore, even after equipping deep ensembles with ST and consistency training (Sec. 1.4.1, supplementary), our method still outperforms, confirming the effectiveness of using middle domains.

Below we briefly discuss some of the limitations:

*Uncertainty under distribution shift*: Our method relies on having reasonable uncertainty estimates (i.e. sigma) in presence of distribution shifts. While we observed sigma training to be helpful for this purpose, and also, uniform merging which does not rely on uncertainty estimates to still outperform the baselines, our method will benefit from better uncertainty estimation techniques.

*Choice of middle domains*: We adopted a fixed set of middle domains, and, as discussed in Sec. 1.4.1, the final performance was not sensitive to the adopted dictionary. However, learning or computationally selecting such middle domains with the objective of downstream robustness could be a worthwhile future direction.

*Multi-modal distributions*: We modeled our individual path outputs with single-modal distributions for convenience and considered multi-modal distributions only at merging step. Allowing for multi-modality in each path’s output may further help with ambiguous data points.
Chapter 1. Ensembling diverse predictions

Computational cost: While the computational complexity of our method and deep ensembles [20] are virtually the same, the methods based on ensembling generally increase the computational complexity as they involve turning one estimator into multiple. Investigating if the models in the ensemble can be compressed would be worthwhile – especially for our method since the diversity in the ensemble is by structure and owed to adopting different middle-domains, rather than stochasticities that often assume independence among models.

This chapter is based on the paper: T. Yeo*, O. F. Kar*, A. Zamir, *Robustness via Cross-Domain Ensembles*, ICCV 2021. I was the main contributor of this work.
2 Controlled Training Data Generation

2.1 Introduction

In this chapter, we present another robustness mechanism based on generating targeted training data. Large datasets have been a driving force in the progress of deep learning. It has also been shown that training on large and diverse datasets tends to improve robustness [102, 103]. However, many datasets exhibit biases, e.g., they are collected from certain geographic regions or with images that tend to be unoccluded and front facing [14, 15, 16]. Neural networks trained on these datasets can return unreliable predictions when tested on inputs with distribution shifts e.g., different geographic locations or unusual viewpoints or occlusions. This chapter presents a method to control generative models to mitigate the bias of training datasets.

An example of generative models is text-to-image models that generate an image conditioned on a given text prompt. These models have displayed remarkable generation and compositional capabilities [104, 105]. Several works have adopted these models to generate training data [30, 31, 32]. However, most of these works make use of pre-defined prompt templates. Thus, they are open-loop systems that generate data without any feedback and to some extent require anticipating the kinds of shifts that will be seen at test-time when defining the prompts.

Early works have attempted to control GANs or VAEs to generate targeted training data [33, 34, 35]. These methods create a closed-loop system by making use of feedback from the model that it aims to improve. Our work falls into this category, with the difference that we employ diffusion models for generation. We aim to learn a suitable prompt to generate images using feedback from the loss of a given pre-trained model. To do so, we borrow the idea of textual inversion [106] that aims to learn a word embedding, with placeholder string $S^*$, that represents a concept e.g., object, individual, style. While
textual inversion learns $S^*$ from a set of images with that concept, our work aims to find $S^*$ that results in generations that **maximize the loss of a given pre-trained model** $f$. See the left of Fig. 2.1 for how our method works and the right for some examples of $S^*$ found by our method.

Figure 2.1: **Controlling generative models to generate targeted training data.**

Left: An overview of how we generate training data. $g$ is a generative model e.g., ControlNet [105] that takes in text prompts and additional conditioning (e.g. depth, segmentation, edge maps), to generate an RGB image. The additional conditioning in this example is depth maps. $f$ is a pre-trained model, in this case, trained to perform depth estimation. We maximize the loss of $f$ on the generated data with respect to a word embedding that has the corresponding string $S^*$. This results in $S^*$ representing a style that is distinctly different from the original training data (see examples on the right). As $g$ takes as input both the text prompt and depth map (as additional conditioning) and as this conditioning is the same as the target task, as long as the generations remain faithful to the condition, **this results in paired data** (RGB image and its corresponding label) that we can use to finetune $f$. See Sec. 2.3.2 for more details.

Right: We show the results from optimizing for several $S^*$. Each $S^*$ represents a distinct concept, and results in generations different from the original training data, shown in the first column. These $S^*$s **can be applied to any image in the training data**, even those that it was not optimized on. $S^*_1$ seems to represent a festive style, $S^*_2$ snow, $S^*_3$ transforms the scene into a wooden cabin, and so on. See Sec. 2.4 for more results.

### 2.2 Related Work

This work focuses on improving model robustness by training on generated data. We give an overview of the relevant topics.

**Closed-loop data generation** guides the generation process via the loss of a given neural network. [33, 35] control the latent space of GANs or VAEs to generate data that maximizes the loss of the network on the generated data. Similarly, [3] controls a NeRF to generate unusual viewpoints. [107] uses an SVM to first identify the failure modes of a given model, then uses this information to generate training data with a diffusion
model. Our method also employs a diffusion model but controls the generation in the prompt space.

**Open-loop data generation** uses pre-defined perturbations or prompts to generate data. Recent works that adopted diffusion models to generate training data use pre-defined prompt templates [32, 31, 30] or use a language model to generative variations of a given prompt [31]. These methods, to some extent, require anticipating the kind of data that will be seen at test-time when defining the prompts. In contrast, our method learns the prompt to generate data. Thus, it does not anticipate the data that will be seen at test-time, and the data generation and training can be done end-to-end.

**Data augmentation.** Data augmentation techniques apply transformations to a given RGB image to increase the diversity of training datasets to improve generalization performance. Traditional augmentations apply transformations such as color jitter, random crop, flipping, and so on to the given image. RandAugment [38], AugMix [36] are ways of combining these augmentations. Mixup [40] and CutMix [39] are proposed augmentations that transform a given image using another image in the dataset. Our method results in images that are more diverse than standard augmentations and more realistic than mixing multiple images. We can also have closed-loop augmentations, i.e., transformations that are guided by the loss of a network. AutoAugment [37] learns the optimal mix of augmentations while adversarial training [11] learns the perturbations to be added to the input.

**Controlling diffusion models.** Methods like ControlNet [105] and T2I-Adapter [108] adapt a pre-trained diffusion model to allow for additional conditioning e.g., edge, segmentation, and depth maps. We employ these models for generation as it allows us to generate paired data for different tasks, given the labels from an existing dataset. Editing methods aim to modify a given image, either via the prompt [109], masks [110], instructions [111] or inversion of the latent space [112, 113]. In contrast, personalization methods aim to adapt models to a given concept e.g., an object, individual, or style. Popular examples include textual inversion [106] and DreamBooth [114], which aim to find a token to represent a concept given several images of that concept. The former freezes the diffusion model, while the latter finetunes it. Extensions of these works learn to represent multiple concepts [115, 116]. While our goal is not personalized generation, we borrow the idea of optimizing for a token to represent a concept.

### 2.3 Method

**Notations.** Let $\mathcal{X}$ be the input image domain, and $\mathcal{Y}$ the target domain. $x$ and $y$ are samples from these domains. We use $f : \mathcal{X} \rightarrow \mathcal{Y}$ to denote the pre-trained model. $\mathcal{L}(f(x), y)$ is the training loss and $\mathcal{D}$ the training dataset for $f$. Let $g$ denote the generative model that takes in text prompt $p$ and optionally, conditioning $z$, i.e., $g : \mathcal{P} \times \mathcal{Z} \rightarrow \mathcal{X}$. 
Chapter 2. Controlled Training Data Generation

2.3.1 Preliminaries on diffusion models and textual inversion

**Diffusion models** [117] learn the data distribution by iteratively denoising a normally distributed variable. These models either operate on the image [118, 119, 120] or to reduce computation cost, the latent space [104]. Diffusion models can also perform generation conditioned on inputs such as text prompts, \( p \), i.e., text-to-image (T2I) generation. One example of a T2I latent diffusion model is Stable Diffusion [104]. It consists of an autoencoder that is used to map images into a latent code (\( z = \mathcal{E}(x) \)), and back to the original image (\( \tilde{x} = \mathcal{D}(z) \)) and a diffusion model that is trained to perform denoising in the latent space. The diffusion model is trained with the following loss:

\[
L_{\text{LDM}} := \mathbb{E}_{x \sim \mathcal{N}(0,1), t} \left[ \left\| \epsilon - \epsilon_\theta(z_t, t, p) \right\|_2^2 \right],
\]

where \( z_t \) is the latent noised to time \( t \), \( \epsilon \) is the unscaled noised sample, \( \epsilon_\theta \) is the denoising network.

**ControlNet** [105] adds additional layers to the Stable Diffusion model to allow for additional conditioning, in addition to text prompts. This conditioning can be depth, edge, segmentation maps, etc. We employ ControlNet for generation as it allows us to condition on the **labels of existing datasets** e.g., depth or segmentation maps, to generate RGB images. This results in paired data i.e., RGB images with their corresponding labels, that can be used for training. See Sec. 2.3.2 for further discussion.

**Textual inversion** [106] aims to learn a concept e.g. style, object, represented by a placeholder word \( S^* \), given a set of images. Each word (or sub-word) in the prompt is converted to a token, after which, an embedding vector, \( v \). More concretely, textual inversion aims to learn an embedding vector, \( v^* \), by optimizing \( L_{\text{LDM}} \) while keeping \( \epsilon_\theta \) fixed. It learns an embedding \( v^* \) to reconstruct the given set of images. The optimization objective is given by,

\[
v^* = \arg\min_v \mathbb{E}_{x \sim \mathcal{N}(0,1), t} \left[ \left\| \epsilon - \epsilon_\theta(z_t, t, p) \right\|_2^2 \right].
\]

2.3.2 Optimizing for a \( S^* \) that maximizes loss

Our goal is to generate training data that reflects the **failure modes** of a given pre-trained model, \( f \). To do this, we **optimize for inputs that maximize the loss of \( f \)**. There are several ways of attaining these inputs. We can e.g., optimize for an imperceptible perturbation, constrained by an \( \ell_p \) norm, to be added to the input image [9], select pre-defined groups in the training data [51] or optimize the latent space of generative models [33, 35]. To generate data, we adopt T2I diffusion models [104, 105] as they exhibit powerful generative capabilities. We then perform this optimization in the space of text prompts as it can potentially result in more interpretable and realistic generations.

Similar to textual inversion [106], we aim to **learn a \( v^* \) that represents a concept**. However, instead of the objective of reconstructing a set of images, we aim to fool a given pre-trained model, \( f \). As described above, ControlNet is able to take domains such as depth as conditioning, thus, we can **generate data conditioned on the depth labels from the original training dataset \( D \)**. More concretely, we use the label \( y \), as conditioning while optimizing for \( v^* \) to generate an RGB image \( x \). The optimization objective is given by,
2.4 Experiments

\[ v^* = \arg \max_v \mathbb{E}_{x \sim g(y, p(v))} \mathcal{L}(f(x), y). \]  
\( g \) in this case is ControlNet and encapsulates the autoencoder, \( D \odot E \), text encoder, and conditional denoising diffusion model, \( \epsilon_\theta \).

Performing the above optimization gives us a \( v^* \), which corresponds to a placeholder word \( S^* \), which we can then use to generate data with a prompt like “\( S^* \)” and condition \( y \). Furthermore, we observe the following: 1. the resulting generations tend to follow the conditioning well (see Sec. 2.3.3 for more details), 2. \( S^* \) can be applied on any image in the dataset, even those that were not optimized over. Thus, generating data with \( S^* \) and labels \( y \) allows us to attain paired data i.e., \( (g(y, p), y) \) for dense tasks e.g., depth, semantic segmentation. These generated data are also distinct from the training data and result in a much higher loss of \( f \) (see Fig. 2.3). Thus, they represent the failure modes of \( f \) and we finetune \( f \) on this generated data to improve its robustness (see Sec. 2.4.2 for results).

2.3.3 Constraining the optimization of \( S^* \)

The above formulation results in the generation of diverse data that is distinctly different from the original training data (see Fig. 2.1, left, Fig. 2.3, 5th column). While most generations are faithful to the conditioning, it is possible that the optimization returns a degenerate solution i.e., a blank image as that would successfully maximize the loss. Thus, we also consider several modifications to the above formulation.

Image to image generation [121]. Instead of generating data from randomly sampled noise, this technique encodes a given image into a latent representation. Denoising is then done on this latent representation and results in generations that are visually closer to the given image. While this technique results in generations that maximize the loss of \( f \) and is less likely to return a collapsed generation, it comes at the expense of lower diversity in the generations (see Fig. 2.2). Thus, the following experiments did not adopt this technique. Balancing the diversity of the generations with its faithfulness is left to future work.

Optimizing for tokens [122]. Instead of optimizing for an embedding \( v^* \), we can optimize for tokens from the existing vocabulary, similar to [122]. This constrains the possible generations to make use of the limited number of existing tokens, and can potentially result in generations that are more interpretable. However, optimizing for up to 128 tokens at the same time did not seem to result in a significant decrease in the loss.

2.4 Experiments

In this section, we will demonstrate that our proposed method is able to generate training data that can be used to improve the robustness of a pre-trained model.
2.4.1 Experimental setup

We consider the depth prediction task, with the model $f$ trained on the Taskonomy dataset [88, 72]. As mentioned in Sec. 2.3.2, we optimize for $n$ embeddings, $\{v^*_i\}_{i=1}^n$. For a datapoint, $(x, y)$, in the Taskonomy dataset, we randomly sample an embedding vector, $v^*_i$, with the corresponding placeholder word $S^*_i$, and generate an RGB image with the prompt “$S^*_i$” and the conditioning $y$. We then finetune $f$ on the generated data, for a range of dataset sizes, 15k, 30k, and 60k datapoints.

**Baselines.** We evaluate the following baselines. They all involve fine-tune the same model $f$ but on different datasets:

- **Control:** This fine-tunes $f$ on the original training data. This baseline is to ensure that the difference in performance is due to the generated data, rather than e.g., longer training or a change in optimization hyperparameters.
- **Prompt “room”:** As the Taskonomy dataset consists of indoor images from mostly residential buildings, we generate data with the prompt “room”.
- **Prompt “”:** Here we do not constrain the generations to look like rooms. Thus, we can potentially get generations that although still resemble indoor scenes but are not living spaces e.g., an indoor pool, cafe, etc.
- **Sampled from $\mathcal{N}(\mu_{emb}, \sigma_{emb})$:** In our proposed method, we optimize for $n$ embedding vectors. Here, we fit a distribution to the embeddings in the vocabulary and sample $n$ embeddings to be used in the data generation. Thus, this baseline controls for the number of embedding vectors used in the data generation and can potentially result in more diverse data.

For all generations, we use ControlNet version 1.0 with depth conditioning [105]. The text prompt always includes “photorealistic, highly detailed”. The proposed method and
2.4 Experiments

baselines are fine-tuned on the same datapoints i.e., all generations use the same set of labels, for conditioning and the control baseline uses the same datapoints from the original dataset.

Figure 2.3: A comparison of generations with different prompts. **Left:** A comparison of the generated data for the baselines and our proposed method. Although the baselines (columns 2-4) result in generations that are different from the original data (first column), they tend to be similar to each other. Optimizing for word embeddings results in styles that are clearly distinct from the original data. Furthermore, as we optimized for several different word embeddings, we were able to generate different styles for each image, as seen in the different rows. Note that sampling different embeddings (second last column) does not result in generations that are as diverse. **Right:** The $\ell_1$ error of $f$ on the different generated data. Evaluating on data generated from optimized word embedding results in the highest loss.

2.4.2 Fine-tuning on generated data

**Comparing the generated images with different prompts.** Figure 2.3 shows the results of the generations for the baseline and proposed method. The generations from the baselines (columns 2-4), although they result in styles different from that of the original training data, tend to have similar styles, while those from our proposed method (last column) exhibit more distinct and diverse styles. Furthermore, the generations seem to keep the geometry of the scene. Thus, it allows us to use the generated RGB images and depth conditioning as training data. The loss of $f$ on the generated data is shown on the right of Fig. 2.3. The generated data for the baselines results in a higher loss compared to that on the original training data and optimizing for prompts results in the highest loss.

**Performance on OOD data.** We evaluate our method and the baselines i.e., after finetuning on their respective generated datasets, qualitatively on random query images (Fig. 2.4, left) and quantitively on the Taskonomy dataset under Common Corruptions and Replica dataset (Fig. 2.4, right). Our proposed method, shown in the last row, returns more accurate predictions in general and even on outdoor scenes. Furthermore, under common corruptions and cross-dataset shifts, when fine-tuned on dataset sizes 30k
Chapter 2. Controlled Training Data Generation

Figure 2.4: **Results under distribution shifts.** *Left:* Qualitative results of the models fine-tuned on the different generated data. The input images are query images from the internet and the following rows show the predictions from the different fine-tuned models. Our proposed method, shown in the last row, results in more robust predictions, even on outdoor scenes. *Right:* $\ell_1$ error relative to the control baseline (multiplied by 100 for readability), lower is better. The different models fine-tuned on different datasets and on different dataset sizes, are evaluated on Taskonomy under Common Corruptions (averaged over all corruptions and severity levels) and the Replica dataset. In both cases, finetuning on generated data from optimized prompts tends to result in better performance.

or larger, our method shows clear improvements.

2.5 Discussion on target domain informed vs uninformed generation

The proposed framework generates data that is *uninformed about the target domain* i.e., the training objective is to generate data that maximizes the loss of $f$ and does not make use of any knowledge of the target domain e.g., unlabelled data, domain description. How well would this framework do if we are interested in a certain target domain?

We experimented with the iWildCam dataset [123] from the WILDS benchmark [18]. The dataset consists of camera trap images and the task is multi-species classification. As the images are taken in the wild, the animals can be highly occluded and have unusual poses or perspectives. Thus, as it may be challenging to generate animals with unusual poses, we generate data using an in-painting diffusion model\(^1\) [104] to keep the region of the image with the animal. Thus, we also make use of the bounding boxes of the animals from [124] for the generation. We run the optimization process described in 2.3.2 to get several $S^*$. The generations from these $S^*$ are shown in Fig. 2.5. We use the ERM model

\(^1\)https://huggingface.co/runwayml/stable-diffusion-inpainting
2.6 Conclusion

We presented a way of controlling diffusion models to generate targeted training data. This results in generations that are diverse and distinctly different from the original training data (see Fig. 2.1, 2.3). Fine-tuning on this data results in improved robustness compared to the baselines.

Below we briefly discuss some of the limitations:

*Computational cost:* Performing the optimization described in Sec. 2.3.2 is expensive, compared to e.g., generating adversarial attacks by optimizing for additive perturbations. The optimization of each $S^*$ takes about an hour on 3 V100s. This is partly due to the nature of diffusion models i.e., they require several denoising iterations to generate an image. Learning to predict an $S^*$ given the state of the model $f$ is an interesting future direction.
Chapter 2. Controlled Training Data Generation

*Fixed conditioning:* In this work we assume access to labels, that can be used as conditioning inputs to ControlNet to generate new RGB images. However, there can also be shifts in the labels when e.g., going from indoor to outdoor scenes or changes in viewpoint or field of view. Learning a generative model to predict the labels/conditioning from e.g. noise or meta-data like camera parameters or proportion of each class in a scene [126] would allow us to control both the label and RGB input domain.

*Curated demonstrations:* This framework can potentially address other types of distribution shifts, e.g., datasets with spurious correlations, and reveal the biases of different models and training data. Attaining such demonstrations is a worthwhile future direction.

This chapter is based on a soon-to-be-published work: T. Yeo, A. Atanov, A. Alekseev, H. Benoit, R. Ray, P. Esmaeil, A. Zamir, *Controlled Training Data Generation with Diffusion Models for Robustness.* I was the main contributor of this work.
Adaptation Mechanisms

Part II
3 Fast adaptation using test-time feedback

3.1 Introduction

The previous chapters presented training-time mechanisms. These mechanisms attempt to anticipate the distribution shifts that can occur at test-time. As distribution shifts that occur in the real world are numerous and unpredictable and the models are frozen at test-time, training-time mechanisms have inherent limitations. This is the main motivation behind test-time adaptation methods, which instead aim to adapt to such shifts as they occur. In other words, these methods choose adaptation over anticipation. In this chapter, we propose a test-time adaptation framework that aims to perform efficient adaptation of a main network using a feedback signal.

One can consider performing test-time training for this purpose, similar to previous works [53, 127, 128]. While this can successfully adapt a network, it is unnecessarily inefficient as it does not make use of the learnable regularities in the adaptation process, and consequently, is unconducive for real-world applications. It also results in a rigid framework as the update mechanism is fixed to be the same as the training process of neural networks (SGD). We show this process can be effectively amortized using a learning-based feed-forward controller network, which yields orders of magnitude faster results (See Fig. 3.1, Sec. 3.4.3). In addition, it provides flexibility advantages as the controller is implemented using a neural network and can be engineered to include arbitrary inductive biases and desired features.

3.2 Related Work

Our work focuses on how to adapt a neural network in an efficient way at test-time on a range tasks and adaptation signals. We give an overview of relevant topics.
Chapter 3. Fast adaptation using test-time feedback

Figure 3.1: Adaptive vs non-adaptive neural network pipelines. **Top:** In order to be robust, non-adaptive methods include training-time interventions that *anticipate and counter* the distribution shifts that will occur at test-time (e.g., via data augmentation). The learned model, \( f_\theta \), is frozen at test-time, thus upon encountering an out-of-distribution input, its predictions may collapse. **Bottom:** Adaptive methods create a *closed loop* and use an *adaptation signal* at test-time. The adaptation signal is a quantity that can be computed at test-time from the environment. \( h_\phi \) acts as a “controller” by taking in an error feedback, computed from the adaptation signal and model predictions, to adapt \( f_\theta \) accordingly. It can be implemented as a (i) standard optimizer (e.g., using SGD) or (ii) neural network. The former is equivalent to test-time optimization (TTO), while the latter aims to *amortize* the optimization process, by training a controller network to adapt \( f_\theta \) – thus, it can be more efficient and powerful. In this chapter, we study the latter approach and show its efficiency and flexibility.

**Robustness methods** *anticipate* the distribution shift that can occur and incorporate inductive biases into the model to help it generalize. Popular methods include data augmentation [11, 40, 129, 36, 39, 42, 7, 4], self-/pre-training [130, 131, 132, 133, 134, 135, 136], architectural changes [137, 138, 139, 140, 141] or ensembling [20, 85, 142, 26, 143, 144]. We focus on adaptation mechanisms and identifying practical adaptation signals that can be used at *test-time*.

**Conditioning methods** use *auxiliary inputs* to adapt a model. Some examples include using HyperNetworks [145, 146] or cross-attention [104, 147]. A popular method that has been adopted in different problem settings, e.g., style transfer [148, 149, 150], few-shot learning [151, 152, 153, 154, 155], is performing feature-wise modulation [156, 157].
3.2 Related Work

involves training a model to use the auxiliary information to predict affine transformation parameters that will be applied to the features of the target model. Our formulation can be viewed to be a particular form of conditioning, and we show it results in a framework that is expressive, efficient, and generalizable.

**Amortized optimization methods** make use of learning to improve (e.g., speed-up) the solution of optimization problems, particularly for settings that require repeatedly solving similar instances of the same underlying problem [158, 159, 160, 161, 162, 163, 164, 165, 166]. Fully amortized optimization methods model the shared structure between past instances of solved problems to regress the solution to a new problem [145, 167, 168]. As adapting to distribution shifts can be cast as solving an optimization problem at test-time, our method can be interpreted as an amortized solution.

**Test-time adaptation methods for geometric tasks.** Many existing frameworks, especially in geometric tasks such as aligning a 3D object model with an image of it, in effect instantiate a task-specific case of closed-loop optimization for each image [164, 161, 169]. Common sources of their adaptation quantity include sensor data [170, 171, 172, 173, 174, 175], structure from motion (SFM) [176, 177], motion [178], and photometric and multi-view consistency constraints (MVC) [179, 180]. Many of the latter methods often focus on depth prediction and they introduce losses that are task-specific, e.g., [176] optimize a photometric consistency loss. We differ by aiming to investigate a more general framework for test-time adaptation that can be applied to several tasks. For MVC, while we adopt the same losses as [179], we show under collapsed predictions, optimizing only MVC constraints is not sufficient for recovering predictions; depth predictions need to be adapted and this can be done efficiently using our proposed framework (see Sec. 3.4.3).

**Test-time adaptation methods for semantic tasks.** Most of these works involve optimizing a self-supervised objective at test-time [181, 53, 182, 128, 183, 127, 184, 54, 185]. They differ in the choice of self-supervised objectives, e.g., prediction entropy [53], mutual information [54], and parameters optimized [184]. However, as we will discuss in Sec. 3.3.2, and as shown by [184, 128, 186], existing methods can fail silently, i.e. successful optimization of the adaptation signal loss does not necessarily result in better performance on the target task. We aim to have a more general method and also show that using proper adaptation signals results in improved performance.

**Weak supervision for semantic tasks** uses imperfect, e.g., sparse and noisy supervision, for learning. In the case of semantic segmentation, examples include scribbles [187] and sparse annotations [188, 189, 190, 191, 192]. For classification, coarse labels are employed in different works [193, 194]. We aim to have a more general method and adopt these as test-time adaptation signals. Further, we show that self-supervised vision backbones, e.g., DINO [195], can also be used to generate such signals and are useful for adaptation (See Sec. 3.3.2).
Multi-modal frameworks are models that can use the information from multiple sources, e.g., RGB image, text, audio, etc., [196, 197, 198, 199, 200, 201, 133, 202, 203, 204]. Schematically, our method has similarities to multi-modal learning (as many amortized optimization methods do) since it simultaneously uses an input RGB image and an adaptation signal. The main distinction is that our method implements a particular process toward adapting a network to a shift using an adaptation signal from the environment – as opposed to a generic multi-modal learning.

3.3 Method

In Fig. 3.1, we schematically compared methods that incorporate robustness mechanisms at training-time (thus anticipating the distribution shift) with those that adapt to shifts at test-time. Our focus is on the latter. In this section, we first discuss the benefits and downsides of common adaptation methods (Sec. A.1.3). We then propose an adaptation method that is fast and can be applied to several tasks (Sec. 3.3.1). To adapt, one also needs to be able to compute an adaptation signal, or proxy, at the test-time. In the second part of the section, we implement a number of practical adaptation signals for a number of tasks (Sec. 3.3.2).

3.3.1 How to adapt at test-time?

An adaptive system is one that can respond to changes in its environment. More concretely, it is a system that can acquire information to characterize such changes, e.g., via an adaptation signal that provides an error feedback, and make modifications that would result in a reduction of this error (see Fig. 3.1). The methods for performing the adaptation of the system range from gradient-based updates, e.g., just using SGD to fine-tune the parameters [181, 53, 128], – to the more efficient semi-amortized [153, 205] and amortized approaches [206, 151, 152] (see Fig. 6 of [152] for an overview). As amortization methods train a controller network to substitute the explicit optimization process, they only require a forward pass at test-time. Thus, they are computationally efficient. Gradient-based approaches, e.g., TTO, can be powerful adaptation methods when the test-time signal is robust and well-suited for the task (see Fig. 3.4). However, they are inefficient and also have the risk of overfitting and the need for carefully tuned optimization hyperparameters [184]. In this work, we focus on an amortization-based approach, RNA.

Notation. We use $X$ to denote the input image domain, and $Y$ to denote the target domain for a given task. We use $f_{\theta} : X \rightarrow Y$ to denote the model to be adapted, where $\theta$ denotes the model parameters. We denote the model before and after adaptation as $f_{\theta}$ and $\hat{f}_{\theta}$ respectively, where the latter is used to get the final predictions after adaptation. $L$ and $D$ are the original training loss and training dataset of $f_{\theta}$, e.g., for classification, $L$ will be the cross-entropy loss and $D$ the ImageNet training data. As shown in Fig. 3.1, $h_{\phi}$ is a controller for $f_{\theta}$. It can be an optimization algorithm, e.g., SGD, or a neural network.
3.3 Method

Rapid Network Adaptation (RNA)

For adaptation, we choose to use a neural network for $h_\phi$. The adaptation signal and model predictions are passed as inputs to $h_\phi$ and it is trained to regress the parameters $\hat{\theta}(\phi) = h_\phi(f_\theta(B), z)$. This corresponds to an objective-based amortization of the TTO process [166]. Using both the adaptation signal $z$ and model prediction $f_\theta(B)$ informs the controller network about the potential errors of the model. The training objective for $h_\phi$ is $\min_{\phi} \mathbb{E}_D [L(f_{\hat{\theta}(\phi)}(B), y)]$, where $(B, y) \sim D$ is a training batch sampled from $D$. Note that the original weights of $f$ are frozen and $h_\phi$ is a small network, having only 5-20% of the number of parameters of $f$, depending on the task. We call this method as rapid network adaptation (RNA) and experiment with different variants of it in Sec. 3.4.

There exist many options for implementing the amortization process, e.g., $h_\phi$ can be trained to update the input image or the weights of $f_\theta$. We choose to modulate the features of $f_\theta$ as it has been shown to work well in different domains [156] and gave the best results. To do this, we insert $k$ Feature-wise Linear Modulation (FiLM) layers [157] into $f_\theta$. Each FiLM layer performs: FiLM($x_i; \gamma_i, \beta_i$) = $\gamma_i \odot x_i + \beta_i$, where $x_i$ is the activation of layer $i$. $h_\phi$ is a network that takes as input the adaptation signal $z$ and model predictions and outputs the coefficients $\{\gamma_i, \beta_i\}$ of all $k$ FiLM layers. $h_\phi$ is trained on the same dataset $D$ as $f_\theta$, therefore, unlike TTO, it is never exposed to distribution shifts during training. Moreover, it is able to generalize to unseen shifts (see Sec. 3.4.3). See the supplementary for the full details as well as the other RNA implementations we investigated.
3.3.2 Which test-time adaptation signals to use?

Independent of the RNA method and while developing adaptation signals is not the main focus of this study, we need to choose some for experimentation. Existing test-time adaptation signals, or proxies, in the literature include prediction entropy [53], spatial autoencoding [128] and self-supervised tasks like rotation prediction [181], contrastive [182] or clustering [184] objectives. The more aligned the adaptation signal is to target task, the better the performance on the target task [181, 182]. More importantly, a poor signal can cause the adaptation to fail silently [184, 128]. Figure 3.3 shows how the original loss on the target task changes as different proxy losses from the literature, i.e. entropy [53], consistency between different middle domains [142, 72] are minimized. In all cases, the proxy loss decreases, however, the improvement in the target loss varies. Thus, successful optimization of existing proxy losses does not necessarily lead to better performance on the target task. In this chapter, we focus on adopting a few practical and real-world signals for our study. Furthermore, RNA turns out to be less susceptible to a poor adaptation signal vs TTO (see sup. mat. Tab. 1). This is because RNA is a neural network trained to use these signals to improve the target task, as opposed to being fixed at being SGD as TTO is.

Employed test-time adaptation signals

We develop test-time adaptation signals for several geometric and semantic tasks as shown in Fig. 3.4. Our focus is not on providing an extensive list of adaptation signals, but rather on using practical ones for experimenting with RNA as well as demonstrating the benefits of using signals that are rooted in the known structure of the world and the task in hand. For example, geometric computer vision tasks naturally follow the multi-view geometry constraints, thus making that a proper candidate for approximating the test-time error, and consequently, an informative adaptation signal.

Geometric Tasks. The field of multi-view geometry and its theorems, rooted in the 3D structure of the world, provide a rich source of adaptation signals. We demonstrate our results on the following target tasks: monocular depth estimation, optical-flow estimation, and 3D reconstruction. For all, we first run a standard structure-from-motion (SFM) pipeline [207]. Then, for depth estimation, we employ the z-coordinates of resulting sparse 3D keypoints from each image as the adaptation signal. For optical flow, we perform keypoint matching across images (which returns sparse optical flow). Lastly, for 3D reconstruction, in addition to the previous two signals, we employ consistency between depth and optical flow predictions as another signal.

Semantic Tasks. For semantic segmentation, we first experiment with using a low number of click annotations for each class, similar to the works on active annotation tools [192, 190, 189]. Likewise, for classification, we use the hierarchical structure of semantic classes, and use coarse labels generated from the WordNet tree [208], similar to [209]. Although these signals (click annotations and coarse labels) are significantly
3.4 Experiments

We demonstrate that our approach consistently outperforms the baselines for adaptation to different distribution shifts (2D and 3D Common Corruptions [2, 4], cross-datasets), over different tasks (monocular depth, image classification, semantic segmentation, optical flow) and datasets (Taskonomy [88], Replica [87], ImageNet [210], COCO [211], ScanNet [212], Hypersim [213]).

3.4.1 Experimental Setup

We describe our experimental setup, i.e. the different adaptation signals, adaptation mechanisms, datasets and baselines, for different tasks. Please see Tab. 3.1 for a summary.

Baselines. We evaluate the following baselines:

*Pre-Adaptation Baseline:* The network \( f \) that maps from RGB to the target task, e.g., weaker versions of the actual ground truth, thus being cheaper to obtain, it may not be realistic to assume access to them at test-time for certain applications, e.g., real-time ones. Thus, we also show how these can be obtained via \( k \)-NN retrieval from the training dataset and patch matching using spatial features obtained from a pre-trained self-supervised vision backbone [195] (see Fig. 3.4).
Figure 3.4: **Examples of employed test-time adaptation signals.** We use a range of adaptation signals in our experiments. These are practical to obtain and yield better performance compared to other proxies. In the left plot, for depth and optical flow estimation, we use sparse depth and optical flow via SFM. In the middle, for classification, for each test image, we perform $k$-NN retrieval to get $k$ training images. Each of these retrieved images has a one hot label associated with it, thus, combining them gives us a coarse label that we use as our adaptation signal. Finally, for semantic segmentation, after performing $k$-NN as we did for classification, we get a pseudo-labelled segmentation mask for each of these images. The features for each patch in the test image and the retrieved images are matched. The top matches are used as sparse supervision. See Sec. 3.4.1 for more details.

<table>
<thead>
<tr>
<th>Task</th>
<th>Adaptation signal</th>
<th>Adapted model</th>
<th>Training data</th>
<th>OOD evaluation data</th>
<th>Baselines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>SFM, masked GT</td>
<td>UNet [94], DPT [114]</td>
<td>TestImage</td>
<td>SFM, Replica, Replica-CC, ScanNet</td>
<td>Pre-adaptation, densification, masked GT, Taskonomy-CC, 3DCC, Hypersim, TENT, TTO, TTO-edges, TTO</td>
</tr>
<tr>
<td>Semantic</td>
<td>Click annotations, patch matching</td>
<td>FCN [104]</td>
<td>COCO (20 classes from Pascal VOC)</td>
<td>Pre-adaptation, Patch matching, ImageNet-C</td>
<td>Pre-adaptation, TDO</td>
</tr>
<tr>
<td>Segmentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td>DenseNets, DenseNets-400 [110], DenseNets-400</td>
<td>COCO [104]</td>
<td>ImageNet, ImageNet-3DCC, ImageNet-V2</td>
<td>Pre-adaptation, DINO, TDO, TGO, TTO-edges, TTO</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: **Overview of the experiments for different target tasks, adaptation methods, and adaptation signals.** For each task, we list the adaptation signal (Sec. 3.3.2) that we use for adaptation. We also list the models that we adapt, and the out-of-distribution (OOD) data used for evaluations and the relevant baselines. When there are different options for adaptation signal, e.g., in the case of depth, the signal is denoted in italics followed by the corresponding OOD dataset. The weights for the semantic segmentation, classification and optical flow models were taken from PyTorch [217].

depth estimation, with no test-time adaptation. We denote this as Baseline for brevity. **Densification:** A network that maps from the given adaptation signal for the target task to the target task, e.g., sparse depth from SFM to dense depth. This is a control baseline and shows what can be learned from the test-time supervision alone, without employing input image information or a designed adaptation architecture. See Sec. 3.4.3 for a variant which includes the image. **TTO (episodic):** We adapt the Baseline model to each episode by optimizing the proxy loss (see Tab. 3.1 for the adaptation signal used for each task.) at test-time. Its weights
3.4 Experiments

Figure 3.5: **RNA can achieve similar performance as TTO in a much shorter time.** We compare how the $\ell_1$ errors of the adaptation mechanisms decrease over wall-clock time (s). The errors are averaged over all episodes (and all corruptions for Replica-CC). RNA only requires a forward pass at test-time, while TTO requires multiple forward and backward passes. On ScanNet and Replica-CC, RNA takes 0.01s, while TTO takes 3s to achieve similar performance. Furthermore, RNA is *not trained with test-time shifts* unlike TTO, thus, it learned to use the additional supervision to adapt to unseen shifts.

are reset to the Baseline model’s after optimizing each batch, similar to [53, 127].

*TTO (online):* We continually adapt to a distribution shift defined by a corruption and severity. Test data is assumed to arrive in a stream, and each data point has the same distribution shift, e.g., noise with a fixed standard deviation [53, 181]. The difference with TTO (episodic) is that the model weights are not reset after each iteration. We denote this as TTO for brevity.

*TTO with Entropy supervision (TENT [53]):* We adapt the Baseline model trained with log-likelihood loss by optimizing the entropy of the predictions. This is to reveal the effectiveness of entropy as a signal as proposed in [53].

*TTO with Sobel Edges supervision (TTO-Edges):* We adapt the Baseline model trained with an additional decoder that predicts a self-supervised task, similar to [181]. We choose to predict Sobel edges as it has been shown to be robust to certain shifts [142]. We optimize the error of the edges predicted by the model and edges extracted from the RGB image to reveal the value of edge error as a supervision.

**RNA configurations.** At test-time, we first get the predictions of the Baseline model and compute an adaptation signal. The predictions and adaptation signal are then passed to $h_\phi$ which adapt $f_\theta$ to $f_\hat{\theta}$. The test images are then passed to $f_\hat{\theta}$ to get the final predictions. We evaluate following variants of RNA.

*RNA (frozen $f$):* Baseline model weights, $f_\theta$, are frozen when training $h_\phi$. We call this variant RNA for brevity.

*RNA (jointly trained $f$):* In contrast to frozen $f$ variant, here we train RNA jointly with the Baseline network. This variant requires longer training.

**Adaptation signal.** As described in Sec. 3.3.2, we compute a broad range of test-time signals from the following processes. Each case describes a process applied on query image(s) in order to extract a test-time quantity.
Chapter 3. Fast adaptation using test-time feedback

Structure-from-motion (SFM): Given a batch of query images, we use COLMAP [207] to run SFM, which returns sparse depth. The percentage of valid pixels, i.e. depth measurements, is about 0.16% on Replica-CC and 0.18% on Replica. For ScanNet we use the pre-computed sparse depth from [218], which has about 0.04% valid pixels. As running SFM on corrupted images results in noisy sparse depth, we train $h_\phi$ to be invariant to noise [174, 218].

Masked ground truth (GT): We apply a random mask to the GT depth of the test image. We fixed the valid pixels to 0.05% of all pixels, i.e. similar sparsity as SFM (see the supplementary for other values). This a control proxy as it enables a better evaluation of the adaptation methods without conflating with the shortcomings of adaptation signals. It is also a scalable way of simulating sparse depth from real-world sensors, e.g., LiDAR, as also done in [170, 219, 220].

Click annotations: We generate click annotations over random pixels for each class in a given image using GT – simulating an active annotation pipeline. The number of pixels ranges from 3 to 25, i.e. roughly 0.01% of the total pixels, similar to [188, 189, 190, 191, 192].

Patch matching: To not use GT click annotations, for each test image, we first retrieve its $k$-NN images from the original clean training dataset using DINO features [195]. We then get segmentation masks on these $k$ images. If the training dataset has labels for segmentation we use them directly, otherwise we obtain them from a pretrained network. For each of the $k$ training images and test image, we extract non-overlapping patches. The features for each patch that lie inside the segmentation masks of the $k$ training images are matched to the features of every patch in the test image. These matches are then filtered and used as sparse segmentation annotations. See Fig. 3.4 for illustration.

Coarse labels (WordNet): We generate 45 coarse labels from the 1000-way ImageNet labels, i.e. making the labels 22x coarser, using the WordNet tree [208], similar to [194]. See supplementary for more details on the construction and results for other coarse label sets.

Coarse labels (DINO $k$-NN): For each test image, we retrieve the $k$-NN images from the training dataset using DINO features [195]. Each of these $k$ training images is associated with an ImageNet class, thus, combining $k$ one-hot labels gives us a coarse label.

Keypoint matching: We perform keypoint matching across images to get sparse optical flow.

3.4.2 Adaptation with RNA vs TTO

Here we summarize our observations from adapting with RNA vs TTO. As described earlier, TTO represents the approach of closed-loop adaptation using the adaptation signal but without benefiting from any amortization (the adaptation process is fixed to be standard SGD). These observations hold across different tasks (see Sec. 3.4.3 for results).
3.4 Experiments

Figure 3.6: **Qualitative results of RNA vs the baselines** for semantic segmentation on random query images on COCO-CC (left) and depth on images from ScanNet, Taskonomy-3DCC and Replica-CC (right). For semantic segmentation, we use 15 pixel annotations per class. For Taskonomy-3DCC, we use sparse depth with 0.05% valid pixels (30 pixels per image). See Fig. 3.7 for results on different adaptation signal levels. For ScanNet and Replica-CC, the adaptation signal is sparse depth measurements from SFM [207] with similar sparsity ratios to Taskonomy-3DCC. The predictions with proposed adaptation signals are shown in the last two rows. They are noticeably more accurate compared to the baselines. Comparing TTO and RNA, RNA’s predictions are more accurate for segmentation, and sharper than TTO for depth (see the ellipses) while being significantly faster. See supplementary for more results.

**RNA is efficient.** Being able to adapt efficiently at test-time is crucial for many real-world problems. As RNA only requires a forward pass at test-time, it is orders of magnitude faster than TTO and is able to attain comparable performance to TTO. In Fig. 3.5, we compare the runtime of adaption with RNA and TTO for depth prediction. On average, for a given episode, RNA obtains similar performance as TTO in 0.01s, compared to TTO’s 3-5s. Similarly, for dense 3D reconstruction, RNA is able to adapt in 0.008s compared to TTO’s 66s (see Fig. 3.9). This suggests a successful amortization of the adaptation optimization by RNA.

Furthermore, RNA’s training is also efficient as it only requires training a small model, i.e. 5-20% of the Baseline model’s parameters, depending on the task. Thus, RNA has a fixed overhead, and small added cost at test-time.

**RNA’s predictions are sharper than TTO for dense prediction tasks.** From the last two rows of Fig. 3.6, it can be seen that RNA retains fine-grained details. This is a noteworthy point and can be attributed to the fact that *RNA benefits from a neural network, thus its inductive biases can be beneficial (and further engineered) for such advantages.* This is a general feature that RNA, and more broadly using a learning-based function to amortize adaptation optimization, brings – in contrast to limiting the
Chapter 3. Fast adaptation using test-time feedback

<table>
<thead>
<tr>
<th>Adaptation Signal</th>
<th>SFM</th>
<th>Sparse GT</th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Replica</td>
<td>ScanNet</td>
<td>Taskonomy</td>
</tr>
<tr>
<td>Shift</td>
<td>CDS</td>
<td>CC</td>
<td>CDS</td>
</tr>
<tr>
<td>Pre-adaptation Baseline</td>
<td>1.75</td>
<td>6.08</td>
<td>3.30</td>
</tr>
<tr>
<td>Densification</td>
<td>2.50</td>
<td>4.19</td>
<td>2.35</td>
</tr>
<tr>
<td>TENT [53]</td>
<td>2.03</td>
<td>6.09</td>
<td>4.03</td>
</tr>
<tr>
<td>TTO-Edges [181]</td>
<td>1.73</td>
<td>6.14</td>
<td>3.28</td>
</tr>
<tr>
<td>RNA (frozen $f$)</td>
<td>1.72</td>
<td>4.26</td>
<td>1.77</td>
</tr>
<tr>
<td>RNA (jointly trained $f$)</td>
<td>1.66</td>
<td>3.41</td>
<td>1.74</td>
</tr>
<tr>
<td>TTO (Episodic)</td>
<td>1.72</td>
<td>3.31</td>
<td>1.85</td>
</tr>
<tr>
<td>TTO (Online)</td>
<td>1.62</td>
<td>3.16</td>
<td>1.76</td>
</tr>
</tbody>
</table>

Table 3.2: Quantitative adaptation results on depth estimation. $\ell_1$ errors on the depth prediction task. (Lower is better. Multiplied by 100 for readability. The best models within 0.0003 error are shown in bold.) We generate distribution shifts by applying Common Corruptions (CC), 3D Common Corruptions (3DCC) and from performing cross-dataset evaluations (CDS). The results from CC and 3DCC are averaged over all distortions and severity levels on Taskonomy and 3 severity levels on Replica data. The adaptation signal from Taskonomy is masked GT (fixed at 0.05% valid pixels) while that from Replica and ScanNet is sparse depth from SFM. RNA and TTO notably outperform the baselines. **RNA successfully matches the performance of TTO while being around 10 times faster.** See supplementary for the losses for different corruption types, sparsity levels, and the results of applying RNA to other adaptation signals.

RNA generalizes to unseen shifts. RNA performs better than TTO for low severities (see supplementary for more details). However, as it was not exposed to any corruptions, the performance gap against TTO narrows at high severities as expected, which is exposed to corruptions at test-time.

We hypothesize that the generalization property of RNA is due to the following reasons. **1.** Even though $f_\theta$ was trained to convergence, it does not achieve exactly 0 error. Thus, when $h_\phi$ is trained with a frozen $f_\theta$ with the training data, it can still learn to correct the errors of $f_\theta$, thus, adapting $f_\theta$. **2.** Adaptation signals, by definition, are expected to be relatively robust to distribution shifts. Even though the RGB image has been corrupted or from a new domain, the adaptation signal is more aligned with the target task. Thus, the input to $h_\phi$ does not undergo a significant shift and is able to adapt $f_\theta$.

3.4.3 Experiments using Various Target Tasks

In this section, we provide a more comprehensive set of evaluations covering various target tasks and adaptation signals. In all cases, RNA is a fixed general framework without being engineered for each task and shows supportive results.

Depth. We demonstrate the results quantitatively in Tab. 3.2 and Fig. 3.5 and qualita-
3.4 Experiments

Figure 3.7: **Qualitative adaptation results on semantic segmentation** on random query images on COCO-CC. RNA notably improves the prediction quality using error feedback from as few as 3 random pixels.

Figure 3.8: **Quantitative adaptation results on semantic segmentation.** Each point shows the mean IOU over 15 corruptions and 5 severities. RNA significantly improves over baselines. Black dashed line shows the mean IOU of baseline model for *clean* validation images, and is provided as a reference. Numbers in the legend denote averages over all supervision pixel counts. See supplementary for a breakdown.

Intuitively in Fig. 3.6. In Tab. 3.2, we compare RNA against all baselines, and over several distribution shifts and different adaptation signals. Our RNA variants outperform the baselines overall. TTO (online) has a better performance than TTO (episodic) as it assumes a smoothly changing distribution shift, and it continuously updates the model weights. RNA (jointly trained $f$) has a better performance among RNA variants. This is reasonable as the target model is not frozen, thus, is less restrictive.

As another baseline, we trained a single model that takes as input a *concatenation of the RGB image and sparse supervision*, i.e. multi-modal input. However, its average performance on Taskonomy-CC was 42.5% worse than RNA’s (see sup. mat. Sec. 3.2). Among the baselines that do not adapt, densification is the strongest under distribution shift due to corruptions. This is expected as it does not take the RGB image as input, thus, is not affected by the underlying distribution shift. However, as seen from the qualitative results in Figs. 3.6, 3.7, it is unable to predict fine-grained details, unlike RNA. We also show that the gap between RNA and densification widens with sparser supervision (see sup. mat. Fig. 1), which confirms that RNA is making use of the error feedback signal, to adapt $f$.

**Dense 3D Reconstruction.** Here, we combine multiple adaptation signals from
Chapter 3. Fast adaptation using test-time feedback

Figure 3.9: Adaptation results for 3D reconstruction. Using appropriate adaptation signals from multi-view geometry can recover accurate 3D reconstructions. We report the average $\ell_1$ error between ground truth 3D coordinates and the estimated ones. The titles above each column refers to the depth model used to get the reconstruction. TTO+MVC corresponds to the predictions after multi-view consistency optimization. It can be seen that RNA and TTO improve the reconstructions over the baselines with RNA being significantly faster. See supplementary for more results.

multi-view geometry. First, we adapt the weights of the depth and optical flow models independently. The results from this adaptation can be found in the previous paragraph (for depth) and supplementary (for optical flow). Next, the two models are adapted to make their predictions consistent with each other. This is achieved using the same process as [179], i.e. multi-view-consistency (MVC). See supplementary for details.

Figure 3.9 shows the point cloud visualizations on Replica with Gaussian Noise corruption. This results in collapsed depth predictions, thus, reconstructions are unusable (Baseline column) and performing MVC is not helpful (Baseline+MVC). Adapting the depth predictions using TTO and MVC improves the reconstruction notably while RNA achieves a similar performance significantly faster.

Semantic Segmentation. We experiment with click annotations and DINO patch matching as adaptation signals.

Click annotations: In Fig. 3.8, we show how the IoU changes with the adaptation signal level on COCO-CC. As the Baseline and TENT do not make use of this signal, their IoU is a straight line. RNA clearly outperforms the baselines for all levels of adaptation signal. Figure 3.7 shows the qualitative results with increasing supervision, and Fig. 3.6 (left) a comparison against all baselines, demonstrating higher quality predictions with RNA.

DINO patch matching: We perform patch matching on DINO features (described in Sec. 3.4.1) to get the adaptation signal. As the patch matching process can be computationally expensive, we demonstrate our results on all cat classes in ImageNet and over
3.4 Experiments

### Adaptation Signal Dataset

<table>
<thead>
<tr>
<th>Adaptation Signal</th>
<th>Dataset</th>
<th>Clean</th>
<th>IN-C</th>
<th>IN-3DCC</th>
<th>IN-V2</th>
<th>Rel. Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Pre-adaptation Baseline</td>
<td>23.85</td>
<td>61.66</td>
<td>54.97</td>
<td>37.15</td>
<td>1.00</td>
</tr>
<tr>
<td>Entropy</td>
<td>TENT</td>
<td>24.67</td>
<td>46.19</td>
<td>47.13</td>
<td>37.07</td>
<td>5.51</td>
</tr>
<tr>
<td>Coarse labels</td>
<td>Densification</td>
<td>95.50</td>
<td>95.50</td>
<td>95.50</td>
<td>95.50</td>
<td>-</td>
</tr>
<tr>
<td>(wordnet)</td>
<td>TTO (Online)</td>
<td>4.72</td>
<td>40.62</td>
<td>42.90</td>
<td>36.77</td>
<td>5.72</td>
</tr>
<tr>
<td></td>
<td>RNA (frozen f)</td>
<td>16.72</td>
<td>41.21</td>
<td>40.37</td>
<td>25.53</td>
<td>1.39</td>
</tr>
<tr>
<td>Coarse labels</td>
<td>DINO (k-NN)</td>
<td>25.56</td>
<td>52.64</td>
<td>48.24</td>
<td>37.39</td>
<td>-</td>
</tr>
<tr>
<td>(DINO)</td>
<td>TTO (Online)</td>
<td>24.59</td>
<td>51.59</td>
<td>49.18</td>
<td>36.96</td>
<td>5.72</td>
</tr>
<tr>
<td></td>
<td>RNA (frozen f)</td>
<td>24.36</td>
<td>54.86</td>
<td>52.29</td>
<td>36.88</td>
<td>1.39</td>
</tr>
</tbody>
</table>

Table 3.3: Quantitative adaptation results on on ImageNet (IN) classification task. We evaluate on the clean validation set, ImageNet-{C,3DCC,V2}. We report average error (%) for 1000-way classification task over all corruptions and severities. For the coarse labels with WordNet supervision, we use 45-coarse labels. For DINO k-NN, we set $k = 20$.

one noise, blur and digital corruption for 3 levels of severity. We used the predictions of a pre-trained FCN on the clean images as pseudolabels to compute IoU. The mean IoU averaged over these corruptions and severities is 48.98 for the baseline model, 53.45 for TTO. RNA obtains a better IOU of 58.04, thus it can make use of the sparse annotations from DINO patch matching.

**Image Classification.** We experiment with coarse labels from WordNet and DINO k-NN as adaptation signals.

*Coarse labels (WordNet):* Table 3.3 shows the results from using 45-coarse labels on ImageNet-{C,3DCC,V2}. This corresponds to 22x coarser supervision compared to the 1000 classes that we are evaluating on. TENT seems to have notable improvements in performance under corruptions for classification, unlike for semantic segmentation and depth. We show that using coarse supervision results in even better performance, about a further 5 pp reduction in error. Furthermore, on uncorrupted data, i.e. clean, and ImageNet-V2 [221], RNA gives roughly 10 pp improvement in performance compared to TTO. Thus, coarse supervision provides a useful signal for adaptation while requiring much less effort than full annotation [193]. See supplementary for results on other coarse sets.

*Coarse labels (DINO k-NN):* We also show results from using coarse sets generated from DINO k-NN retrieval. This is shown in the last 3 rows of Tab. 3.3. Both RNA and TTO use this coarse information to outperform the non-adaptive baselines. However, they do not always outperform TENT, which could be due to the noise in retrieval.

#### 3.4.4 Ablations and additional results

**Adaptation on other architectures.** Table 3.4 shows the results of incorporating RNA to different architectures, namely the dense prediction transformer (DPT) [214] for depth and ConvNext [141] for image classification. In both cases, RNA is able to improve on the error and runtime of TTO. Thus, RNA can be applied to a range of
Chapter 3. Fast adaptation using test-time feedback

<table>
<thead>
<tr>
<th>Task (Arch.)</th>
<th>Depth (DPT [214])</th>
<th>Classification (ConvNext [141])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>Clean CC Rel. Runtime</td>
<td>Clean IN-C Rel. Runtime</td>
</tr>
<tr>
<td>Pre-adaptation Baseline</td>
<td>2.23 3.76 1.00</td>
<td>18.13 42.95 1.00</td>
</tr>
<tr>
<td>TTO (Online)</td>
<td>1.82 2.61 13.85</td>
<td>17.83 41.44 11.04</td>
</tr>
<tr>
<td>RNA (frozen f)</td>
<td>1.13 1.56 1.01</td>
<td>14.32 38.04 1.07</td>
</tr>
</tbody>
</table>

Table 3.4: RNA works across different architectures. Quantitative adaptation results on depth estimation and image classification on Taskonomy and ImageNet datasets, respectively. (Lower is better. \( \ell_1 \) errors for depth estimation are multiplied by 100 for readability.)

architectures.

**Controlling for number of parameters.** We ran a control experiment where all methods have the same architecture, thus, same number of parameters. The results are in supplementary Table 2. RNA still returns the best performance. Thus, its improvement over the baselines is not due to a different architecture or number of parameters but due to its test-time adaptation mechanism.

**Different implementations of RNA.** We experiment with different controller architectures e.g., HyperNetworks [145], other FiLM variants, or adapting the input instead of the model parameters. See supplementary Sec. 2.2 for the details and a conceptual discussion on the trade-offs of the choices of implementing this closed-loop “control” system, namely those that make stronger model-based assumptions.

3.5 Discussions of RNA compared to other approaches

There are many methods that aim to handle distribution shifts. Figure 3.10 gives an overview of how these methods can be characterized. Open-loop systems predict \( y \) by only using their inputs *without receiving feedback*. Training-time robustness methods, image modifications, and multi-modal methods fall into this category. These methods assume the learned model is frozen at test-time. Thus, they aim to incorporate inductive biases at training time that could be useful against distribution shifts at test-time. The closed-loop systems, on the other hand, are *adaptive* as they make use of an error feedback signal that can be computed at test-time from the model predictions and an adaptation signal.

The closed-loop systems can be instantiated as *model-based* or *model-free* adaptation methods. The former performs adaptation by estimating the parameters of the distribution shift using the feedback signal. For this purpose, different forms of feedback can be useful, e.g. an error feedback and the feedback from the input image itself can lead to successful estimation of the shift parameters. This is a form of inductive bias that can help the method generalize for similar shifts. Furthermore, explicitly modelling the distribution shift parameters results in an interpretable system that will not fail
### 3.5 Discussions of RNA compared to other approaches

<table>
<thead>
<tr>
<th>Open-Loop</th>
<th>Closed-Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robustness methods</td>
<td>Model-based adaptation</td>
</tr>
<tr>
<td>$x \rightarrow y$</td>
<td>$x \rightarrow y$</td>
</tr>
<tr>
<td>Image modification (Denoising)</td>
<td>$e \leftarrow z$</td>
</tr>
<tr>
<td>$x \rightarrow \hat{x} \rightarrow y$</td>
<td>$x \rightarrow y$</td>
</tr>
<tr>
<td>Multi-modal methods</td>
<td>Model-free adaptation</td>
</tr>
<tr>
<td>$x \rightarrow z \rightarrow y$</td>
<td>$x \rightarrow z$</td>
</tr>
</tbody>
</table>

Figure 3.10: **An overview of methods that aim to handle distribution shifts.** On the left we have open-loop systems. They predict $y$ by only using their inputs without receiving feedback. The first and popular example of open-loop systems is training-time robustness methods (data augmentation, architectural changes, etc.). The next example is the methods that modify the input $x$, e.g. denoising or style changes, independent of $y$. Furthermore, there are multi-modal methods that use additional input $z$. As the learned model is frozen at test-time, these methods need to anticipate the distribution shift by incorporating inductive biases at training time (See also Fig. 3.1). In contrast, closed-loop systems on the right make use of its current output, $y$, and an adaptation signal, $z$, to form an error feedback signal that can be used to update its predictions. Thus, they adapt to the shifts as they occur. We can then group closed-loop systems into model-based and model-free methods. The former performs adaptation by estimating the parameters of the distribution shift, $e$, while the latter performs adaptation without explicitly predicting $e$. Furthermore, this adaptation can be performed via running an optimization, i.e. test-time optimization (TTO) via SGD, or via amortization, i.e., training a side-network to predict TTO updates that minimizes the error feedback. Our proposed method, RNA, belongs to the model-free adaptation approaches that makes use of amortization for efficiency.

Silently. In [222], to learn a policy for legged robots, they defined a model of the environment e.g., ground friction, center of mass, etc., and showed that it generalized well. However, our experiments with a model-based approach, where we defined the parameters of the environment as the possible distribution shifts can occur e.g., intensity of a noise or blur corruption, did not show promising results. It can because our model is not an accurate reflection of the real world, and/or that such a model is not needed for static vision tasks.

In contrast, model-free methods do not estimate these parameters and learn to adapt based only on the error feedback signal. Our proposed method RNA belongs to model-free approaches, and as we have shown in this chapter, it generalizes well to a diverse set of unseen distribution shifts. Note that model-free adaptation methods, including RNA, has schematic similarities to multi-modal learning approaches as they simultaneously use...
Chapter 3. Fast adaptation using test-time feedback

an RGB input image and an adaptation signal. The main distinction is that our method implements a particular process toward adapting a network to a shift using an adaptation signal from the environment – as opposed to a generic multi-modal learning.

Closed-loop systems aim to minimize an error feedback signal. There are different ways to implement this. Running an optimization, e.g. SGD, at test-time is a popular choice, as done in several other works [181, 53, 128, 54]. Since this may be unnecessarily expensive, another way is to amortize this optimization process [166]. This results in our proposed method, RNA.

RNA vs Training-time Robustness Methods. As discussed before, training-time robustness methods, e.g. data augmentation, aim to anticipate distribution shifts by building invariances at training-time. On the other hand, RNA performs adaptation at test-time using error feedback. Furthermore, as RNA makes use of an error signal, it is able to handle cases where there are multiple possible predictions for a given input, e.g., scale ambiguity for monocular depth estimation, while the robustness methods may not.

RNA vs Denoising. The denoising methods, and in general the methods performing modification in the input image, e.g., domain adaptation methods that aims to map an image in the target domain to the style of the source domain [80], are concerned with reconstructing plausible images without taking the downstream prediction \( x \to y \) into account (shown as gray in Fig. 3.10). Moreover, it has been shown that imperceptible artifacts in the denoised & modified image could result in degraded predictions [2, 142, 183]. In contrast, RNA performs updates with the goal of reducing the error of the target task.

RNA vs Model-based Adaptation. As explained before, model-based approaches require building a well-defined model of the distribution shifts that can be faced at test-time. While this can effectively work for the modeled shifts, the performance can quickly deteriorate for the ones that are outside the scope of the model. Thus, the effectiveness for a more general and practical adaptation setting is limited. In contrast, RNA adopts a model-free adaptation approach that does not require a tedious modelling of distribution shifts at the expense of lack of interpretability. As our experiments on a diverse set of tasks and distribution shifts show, RNA is able to generalize and outperforms the baselines.

RNA vs TTO. RNA aims to amortize the optimization process of TTO. Thus, it aims to simulate TTO by learning to reduce errors in the predictions using the error feedback signal. In Sec. 3.4, we show that RNA is able to adapt orders of magnitude faster than running test-time optimization (TTO). See Sec. 3.4.2 for detailed discussions and results.

Using different forms of feedback signal as input to RNA. In the case where
only the adaptation signal, $z$, is passed as input, it is possible that the side-network is implicitly modelling an error feedback signal. This is because it is trained alongside the main model ($x \rightarrow y$), thus, it sees and learns to correct the main model’s errors during training. We found that having an error feedback signal as input results in better performance on average, thus, we adopted this as our main method.

3.6 Conclusion and Limitations

We presented RNA, a method for efficient adaptation of neural networks at test-time using a closed-loop formulation. It involves training a side network to use a test-time adaptation signal to adapt a main network. This network acts akin to a “controller” and adapts the main network based on the adaptation signal. We showed that this general and flexible framework can generalize to unseen shifts, and as it only requires a forward pass at test-time, it is orders of magnitude faster than TTO. We evaluated this approach using a diverse set of adaptation signals and target tasks. We briefly discuss the limitations and potential future works:

Different implementations of RNA and amortization methods. While we experimented with several RNA variants (see supplementary for details), further investigations using other techniques, e.g. cross-attention [147] or building in a more explicit “model” of the shifts and environment, could be worthwhile. In general, as the role of the controller network is to amortize the training optimization of the main network, the amortized optimization literature [166] is an apt resource to consult for this purpose.

Hybrid mechanism for activating TTO in RNA. TTO constantly adapts a model to a distribution shift, hence, in theory, it can adapt to any shift despite being comparatively inefficient. To have the best of both worlds, investigating mechanisms for selectively activating TTO within RNA when needed can be useful.

Finding adaptation signals for a given task. While the focus of this study was not on developing new adaption signals, we demonstrated useful ones for several core vision tasks, but there are many more. Finding these signals requires either knowledge of the target task so a meaningful signal can be accordingly engineered or core theoretical works on understanding how a proxy and target objectives can be “aligned” for training.

This chapter is based on the paper: T. Yeo, O. F. Kar, Z. Sodagar, A. Zamir, Fast Adaptation of Neural Networks using Test-Time Feedback, ICCV 2023. I was the main contributor of this work.
Benchmarks Part III
4 3D Common Corruptions

4.1 Introduction

This chapter presents a set of distribution shifts to test models’ robustness before deploying these models in the real world. In contrast to previously proposed shifts which perform uniform 2D modifications over the image, such as Common Corruptions (2DCC) [2], our shifts incorporate 3D information to generate corruptions that are consistent with the scene geometry. This leads to shifts that are more likely to occur in the real world (See Fig. 4.1). The resulting set includes 20 corruptions, each representing a distribution shift from training data, which we denote as 3D Common Corruptions (3DCC). 3DCC addresses several aspects of the real world, such as camera motion, weather, occlusions, depth of field, and lighting. Figure 4.2 provides an overview of all corruptions. As shown in Fig. 4.1, the corruptions in 3DCC are more diverse and realistic compared to 2D-only approaches.

We show in Sec. 4.5 that the performance of the methods aiming to improve robustness, including those with diverse data augmentation, reduce drastically under 3DCC. Furthermore, we observe that the robustness issues exposed by 3DCC well correlate with corruptions generated via photorealistic synthesis. Thus, 3DCC can serve as a challenging testbed for real-world corruptions, especially those that depend on scene geometry.

Motivated by this, our framework also introduces new 3D data augmentations. They take the scene geometry into account, as opposed to 2D augmentations, thus enabling models to build invariances against more realistic corruptions. We show in Sec. 4.5.3 that they significantly boost model robustness against such corruptions, including the ones that cannot be addressed by the 2D augmentations.

The proposed corruptions are generated programmatically with exposed parameters, enabling fine-grained analysis of robustness, e.g. by continuously increasing the 3D
Chapter 4. 3D Common Corruptions

Figure 4.1: Using 3D information to generate real-world corruptions. The top row shows sample 2D corruptions applied uniformly over the image, e.g. as in Common Corruptions [2], disregarding 3D information. This leads to corruptions that are unlikely to happen in the real world, e.g. having the same motion blur over the entire image irrespective of the distance to camera (top left). Middle row shows their 3D counterparts from 3D Common Corruptions (3DCC). The circled regions highlight the effect of incorporating 3D information. More specifically, in 3DCC, 1. motion blur has a motion parallax effect where objects further away from the camera seem to move less, 2. defocus blur has a depth of field effect, akin to a large aperture effect in real cameras, where certain regions of the image can be selected to be in focus, 3. lighting takes the scene geometry into account when illuminating the scene and casts shadows on objects, 4. fog gets denser further away from the camera, 5. occlusions of a target object, e.g. fridge (blue mask), are created by changing the camera’s viewpoint and having its view naturally obscured by another object, e.g. the plant (red mask). This is in contrast to its 2D counterpart that randomly discards patches [223].

motion blur. They are efficient to compute and can be computed on-the-fly during training as data augmentation with a small increase in computational cost. They are also extendable, i.e. they can be applied to standard vision datasets, e.g. ImageNet [210], that do not come with 3D labels.

4.2 Related Work

This chapter presents a data-focused approach [224, 225] to robustness. We give an overview of some of the related topics within the constraints of space.

Robustness benchmarks based on corruptions: Several studies have proposed robustness benchmarks to understand the vulnerability of models to corruptions. A popular benchmark, Common Corruptions (2DCC) [2], generates synthetic corruptions on real images that expose sensitivities of image recognition models. It led to a series of
4.2 Related Work

Figure 4.2: The new corruptions. We propose a diverse set of new corruption operations ranging from defocusing (near/far focus) to lighting changes and 3D-semantic ones, e.g. object occlusion. These corruptions are all automatically generated, efficient to compute, and can be applied to most datasets (Sec. 4.3.3). We show that they expose vulnerabilities in models (Sec. 4.5.2) and are a good approximation of realistic corruptions (Sec. 4.5.2). A subset of the corruptions marked in the last column are novel and commonly faced in the real world, but are not 3D based. We include them in our benchmark. For occlusion and scale corruptions, the blue and red masks denote the amodal visible and occluded parts of an object, e.g. the fridge.

works either creating new corruptions or applying similar corruptions on other datasets for different tasks [226, 227, 228, 229, 230, 231]. In contrast to these works, 3DCC modifies real images using 3D information to generate realistic corruptions. The resulting images are both perceptually different and expose different failure modes in model predictions compared to their 2D counterparts (See Fig. 4.1 and 4.8). Other works create and capture the corruptions in the real world, e.g. ObjectNet [232]. Although realistic, it requires significant manual effort and is not extendable. A more scalable approach is to use computer graphics based 3D simulators to generate corrupted data [233] which can lead to generalization concerns. 3DCC aims to generate corruptions as close to the real world as possible while staying scalable.

Robustness analysis works use existing benchmarks to probe the robustness of different methods, e.g. data augmentation or self-supervised training, under several distribution shifts. Recent works investigated the relation between synthetic and natural distribution shifts [102, 7, 103, 234] and effectiveness of architectural advancements [138, 139, 235]. We select several popular methods to show that 3DCC can serve as a challenging benchmark (Fig. 4.6 and 4.7).

Improving robustness: Numerous methods have been proposed to improve model robustness such as data augmentation with corrupted data [84, 11, 129, 236], texture
Figure 4.3: **Left:** We show the inputs needed to create each of our corruptions, e.g., the 3D information such as depth, and RGB image. These corruptions have also been grouped (in solid colored lines) according to their corruption types. For example, to create the distortions in the dashed box in the right, one only needs the RGB image and its corresponding depth. For the ones in the left dashed box, 3D mesh is required. Note that one can create view changes corruptions also from panoramic images if available, without a mesh. **Right:** As an example, we show an overview of generating depth of field effect efficiently. The scene is first split into multiple layers by discretizing scene depth. Next, a region is chosen to be kept in focus (here it is the region closest to the camera). We then compute the corresponding blur levels for each layer according to their distance from the focus region, using a pinhole camera model. The final refocused image is obtained by compositing blurred image layers.

Changes [96, 7], image compositions [40, 39] and transformations [36, 42]. While these methods can generalize to some unseen examples, performance gains are non-uniform [41, 84]. Other methods include self-training [131], pre-training [130, 237], architectural changes [138, 139], and diverse ensembling [23, 24, 25, 142]. Here we instead adopt a data-focused approach to robustness by i. providing a large set of realistic distribution shifts and ii. introducing new 3D data augmentation that improves robustness against real-world corruptions (Sec. 4.5.3).

**Photorealistic image synthesis** involves techniques to generate realistic images. Some of these techniques have been recently used to create corruption data. These techniques are generally specific to a single real-world corruption. Examples include adverse weather conditions [238, 239, 240, 241, 242], motion blur [243, 244], depth of field [245, 246, 247, 132, 248], lighting [249, 250], and noise [251, 252]. They may be used for purely artistic purposes or to create training data. Some of our 3D transformations are instantiations of these methods, with the downstream goal of testing and improving model robustness in a unified framework with a wide set of corruptions.

**Image restoration** aims to undo the corruption in the image using classical signal processing techniques [253, 254, 255, 256] or learning-based approaches [257, 258, 259, 260, 261, 262, 263]. We differ from these works by generating corrupted data, rather than removing it, to use them for benchmarking or data augmentation. Thus, in the latter, we train with these corrupted data to encourage the model to be invariant to corruptions, as opposed to training the model to remove the corruptions as a pre-processing step.
**4.3 Generating 3D Common Corruptions**

**Adversarial corruptions** add imperceptible *worst-case* shifts to the input to fool a model [9, 10, 11, 264]. Most of the failure cases of models in the real world are not the result of adversarial corruptions but rather *naturally occurring distribution shifts*. Thus, our focus in this paper is to generate corruptions that are likely to occur in the real world.

**4.3 Generating 3D Common Corruptions**

**4.3.1 Corruption Types**

We define different corruption types, namely *depth of field*, *camera motion*, *lighting*, *video*, *weather*, *view changes*, *semantics*, and *noise*, resulting in 20 corruptions in 3DCC. Most of the corruptions require an RGB image and scene depth, while some needs 3D mesh (See Fig. 4.3). We use a set of methods leveraging 3D synthesis techniques or image formation models to generate different corruption types, as explained in more detail below. Further details are provided in the appendix.

**Depth of field** corruptions create refocused images. They keep a part of the image in focus while blurring the rest. We consider a layered approach [132, 248] that splits the scene into multiple layers. For each layer, the corresponding blur level is computed using the pinhole camera model. The blurred layers are then composited with alpha blending. Figure 4.3 (right) shows an overview of the process. We generate *near focus* and *far focus* corruptions by randomly changing the focus region to the near or far part of the scene.

**Camera motion** creates blurry images due to camera movement during exposure. To generate this effect, we first transform the input image into a point cloud using the depth information. Then, we define a trajectory (camera motion) and render novel views along this trajectory. As the point cloud was generated from a single RGB image, it has incomplete information about the scene when the camera moves. Thus, the rendered views will have disocclusion artifacts. To alleviate this, we apply an inpainting method from [244]. The generated views are then combined to obtain parallax-consistent motion blur. We define *XY-motion blur* and *Z-motion blur* when the main camera motion is along the image XY-plane or Z-axis, respectively.

**Lighting** corruptions change scene illumination by adding new light sources and modifying the original illumination. We use Blender [265] to place these new light sources and compute the corresponding illumination for a given viewpoint in the 3D mesh. For the *flash* corruption, a light source is placed at the camera’s location, while for *shadow* corruption, it is placed at random diverse locations outside the camera frustum. Likewise, for *multi-illumination* corruption, we compute the illumination from a set of random light sources with different locations and luminosities.
Chapter 4. 3D Common Corruptions

**Video** corruptions arise during the processing and streaming of videos. Using the scene 3D, we create a video using multiple frames from a single image by defining a trajectory, similar to motion blur. Inspired by [229], we generate average bit rate (ABR) and constant rate factor (CRF) as H.265 codec compression artifacts, and bit error to capture corruptions induced by imperfect video transmission channel. After applying the corruptions over the video, we pick a single frame as the final corrupted image.

**Weather** corruptions degrade visibility by obscuring parts of the scene due to disturbances in the medium. We define a single corruption and denote it as fog 3D to differentiate it from the fog corruption in 2DCC. We use the standard optical model for fog [238, 239, 240]:

\[
I(x) = R(x)t(x) + A(1 - t(x)),
\]

where \( I(x) \) is the resulting foggy image at pixel \( x \), \( R(x) \) is the clean image, \( A \) is atmospheric light, and \( t(x) \) is the transmission function describing the amount of light that reaches the camera. When the medium is homogeneous, the transmission depends on the distance from the camera, \( t(x) = \exp(-\beta d(x)) \) where \( d(x) \) is the scene depth and \( \beta \) is the attenuation coefficient controlling the fog thickness.

**View changes** are due to variations in the camera extrinsics and focal length. Our framework enables rendering RGB images conditioned on several changes, such as field of view, camera roll and camera pitch, using Blender. This enables us to analyze the sensitivity of models to various view changes in a controlled manner. We also generate images with view jitter that can be used to analyze if models predictions flicker with slight changes in viewpoint.

**Semantics:** In addition to view changes, we also render images by selecting an object in the scene and changing its occlusion level and scale. In occlusion corruption, we generate views of an object occluded by other objects. This is in contrast to random 2D masking of pixels to create an unnatural occlusion effect that is irrespective of image content, e.g. as in [223, 235] (See Fig. 4.1). Occlusion rate can be controlled to probe model robustness against occlusion changes. Similarly, in scale corruption, we render views of an object with varying distances from the camera location. Note that the corruptions require a mesh with semantic annotations, and are generated automatically, similar to [266]. This is in contrast to [232] which requires tedious manual effort. The objects can be selected by randomly picking a point in the scene or using the semantic annotations.

**Noise** corruptions arise from imperfect camera sensors. We introduce new noise corruptions that do not exist in the previous 2DCC benchmark. For low-light noise, we decreased the pixel intensities and added Poisson-Gaussian distributed noise to reflect the low-light imaging setting [251]. ISO noise also follows a Poisson-Gaussian distribution, with a fixed photon noise (modeled by a Poisson), and varying electronic noise (modeled by a Gaussian). We also included color quantization as another corruption that reduces the bit depth of the RGB image. Only this subset of our corruptions is not based on 3D
4.3 Generating 3D Common Corruptions

Figure 4.4: Visualizations of 3DCC with increasing shift intensities. Top: Increasing the shift intensity results in larger blur, less illumination, and denser fog. Bottom: The object becomes more occluded or shrinks in size using calculated viewpoint changes. The blue mask denotes the amodal visible parts of the fridge/couch, and the red mask is the occluded part. The leftmost column shows the clean images. Visuals for all corruptions for all shift intensities are shown in the appendix.

4.3.2 Starter 3D Common Corruptions Dataset

We release the full open source code of our pipeline, which enables using the implemented corruptions on any dataset. As a starter dataset, we applied the corruptions on 16k Taskonomy [88] test images. For all the corruptions except the ones in view changes and semantics which change the scene, we follow the protocol in 2DCC and define 5 shift intensities, resulting in approximately 1 million corrupted images (16k×14 × 5).

Directly applying the methods to generate corruptions results in uncalibrated shift intensities with respect to 2DCC. Thus, to enable aligned comparison with 2DCC on a more uniform intensity change, we perform a calibration step. For the corruptions with a direct counterpart in 2DCC, e.g. motion blur, we set the corruption level in 3DCC such that for each shift intensity in 2DCC, the average SSIM [267] values over all images is the same in both benchmarks. For the corruptions that do not have a counterpart in 2DCC, we adjust the distortion parameters to increase shift intensity while staying in a similar SSIM range as the others. For view changes and semantics, we render 32k images with smoothly changing parameters, e.g. roll angle, using the Replica [87] dataset.
Chapter 4. 3D Common Corruptions

Figure 4.4 shows example corruptions with different shift intensities.

4.3.3 Applying 3DCC to standard vision datasets

While we employed datasets with full scene geometry information such as Taskonomy [88], 3DCC can also be applied to standard datasets without 3D information. We exemplify this on ImageNet [210] and COCO [211] validation sets by leveraging depth predictions from the MiDaS [268] model, a state-of-the-art depth estimator. Figure 4.5 shows example images with near focus, far focus, and fog 3D corruptions. Generated images are physically plausible, demonstrating that 3DCC can be used for other datasets by the community to generate a diverse set of image corruptions. In Sec. 4.5.2, we quantitatively demonstrate the effectiveness of using predicted depth to generate 3DCC.

4.4 3D Data Augmentation

While benchmarking uses corrupted images as test data, one can also use them as augmentations of training data to build invariances towards these corruptions. This is the case for us since, unlike 2DCC, 3DCC is designed to capture corruptions that are more likely to appear in the real world, hence it has a sensible augmentation value as well. Thus, in addition to benchmarking robustness using 3DCC, our framework can also be viewed as new data augmentation strategies that take the 3D scene geometry into account. We augment with the following corruption types in our experiments: depth of field, camera motion, and lighting. The augmentations can be efficiently generated on-the-fly during training using parallel implementations. For example, the depth of field augmentations take 0.87 seconds (wall clock time) on a single V100 GPU for a batch size of 128 images with $224 \times 224$ resolution. For comparison, applying 2D defocus blur requires 0.54 seconds, on average. It is also possible to precompute certain selected parts of the augmentation process, e.g., the illuminations for lighting augmentations, to increase efficiency. We incorporated these mechanisms in our implementation. We show in Sec. 4.5.3 that these augmentations can significantly improve robustness against real-world distortions.

4.5 Experiments

We perform evaluations to demonstrate that 3DCC can expose vulnerabilities in models (Sec. 4.5.2) that are not captured by 2DCC (Sec. 4.5.2). The generated corruptions are similar to expensive realistic synthetic ones (Sec. 4.5.2) and are applicable to datasets without 3D information (Sec. 4.5.2) and for semantic tasks (Sec. 4.5.2). Finally, the proposed 3D data augmentation improves robustness qualitatively and quantitatively (Sec. 4.5.3).
Figure 4.5: 3DCC can be applied to most datasets, even those that do not come with 3D information. Several query images from the ImageNet [210] and COCO [211] dataset are shown with near focus, far focus and fog 3D corruptions applied. Notice how the objects in the circled regions go from sharp to blurry depending on the focus region and scene geometry. To get the depth information needed to create these corruptions, predictions from MiDaS [268] model is used. This gives a good enough approximation to generate realistic corruptions (as we will quantify in Sec. 4.5.2).

4.5.1 Preliminaries

**Evaluation Tasks:** 3DCC can be applied to any dataset, irrespective of the target task, e.g. dense regression or low-dimensional classification. Here we mainly experiment with surface normals and depth estimation as target tasks widely employed by the community. We note that the robustness of models solving such tasks is underexplored compared to classification tasks (See Sec. 4.5.2 for results on panoptic segmentation and object recognition). To evaluate robustness, we compute the $\ell_1$ error between predicted and ground truth images.

**Training Details:** We train UNet [94] and DPT [214] models on Taskonomy [88] using learning rate $5 \times 10^{-4}$ and weight decay $2 \times 10^{-6}$. We optimize the likelihood loss with Laplacian prior using AMSGrad [95], following [142]. Unless specified, all the models use the same UNet backbone (e.g. Fig. 4.6). We also experiment with DPT models trained on Omnidata [132] that mixes a diverse set of training datasets. Following [132], we train with learning rate $1 \times 10^{-5}$, weight decay $2 \times 10^{-6}$ with angular & $\ell_1$ losses.

**Robustness mechanisms evaluated:** We evaluate several popular data augmentation strategies: DeepAugment [7], style augmentation [96], and adversarial training [10]. We also include Cross-Domain Ensembles (X-DE) [142] that has been recently shown to improve robustness to corruptions by creating diverse ensemble components via input transformations. We refer to the appendix for training details. Finally, we train a model with augmentation with corruptions from 2DCC [2] (2DCC augmentation), and another model with 3D data augmentation on top of that (2DCC + 3D augmentation).
4.5.2 3D Common Corruptions Benchmark

3DCC can expose vulnerabilities

We perform a benchmarking of the existing models against 3DCC to understand their vulnerabilities. However, we note that our main contribution is not the performed analyses but the benchmark itself. The state-of-the-art models may change over time and 3DCC aims to identify the robustness trends, similar to other benchmarks.

Effect of robustness mechanisms: Figure 4.6 shows the average performance of different robustness mechanisms on 3DCC for surface normals and depth estimation tasks. These mechanisms improved the performance over the baseline but are still far from the performance on clean data. This suggests that 3DCC exposes robustness issues and can serve as a challenging testbed for models. The 2DCC augmentation model returns slightly lower $\ell_1$ error, indicating that diverse 2D data augmentation only partially helps against 3D corruptions.

Effect of dataset and architecture: We provide a detailed breakdown of performance against 3DCC in Fig. 4.7. We first observe that baseline U-Net and DPT models trained on Taskonomy have similar performance, especially on the view change corruptions. By training with larger and more diverse data with Omnidata, the DPT performance improves. Similar observations were made on vision transformers for classification [269, 138]. This improvement is notable with view change corruptions, while for the other corruptions, there is a decrease in error from 0.069 to 0.061. This suggests that combining architectural advancements with diverse and large training data can play an important role in robustness against 3DCC. Furthermore, when combined with 3D augmentations, they improve
robustness to real-world corruptions (Sec. 4.5.3).

Figure 4.7: Detailed breakdown of performance on 3DCC. The benchmark can expose trends and models’ sensitivity to a wide range of corruptions. We show this by training models on either Taskonomy [88] or Omnidata [132] and with either a UNet [94] or DPT [214] architecture. The average $\ell_1$ error over all shift intensities for each corruption is shown (lower is better). **Top:** We observe that Taskonomy models are more susceptible to changes in field of view, camera roll, and pitch compared to Omnidata trained model, which is consistent with their methods. **Bottom:** The numbers in the legend are the average performance over all the corruptions. We can see that all the models are sensitive to 3D corruptions, e.g. *z-motion blur* and *shadow*. Overall, training with large diverse data, e.g. Omnidata, and using DPT is observed to notably improve performance.

Redundancy of corruptions in 3DCC and 2DCC

In Fig. 4.1, a qualitative comparison was made between 3DCC and 2DCC. The former generates more realistic corruptions while the latter does not take scene 3D into account and applies uniform modifications over the image. In Fig. 4.8, we aim to quantify the similarity between 3DCC and 2DCC. On the left of Fig. 4.8, we compute the correlations of $\ell_1$ errors between clean and corrupted predictions made by the baseline model for a subset of corruptions (full set is in the appendix). 3DCC incurs less correlations both intra-benchmark as well as against 2DCC (Mean correlations are 0.32 for 2DCC-2DCC, 0.28 for 3DCC-3DCC, and 0.30 for 2DCC-3DCC). Similar conclusions are obtained for depth estimation (in the appendix). In the right, we provide the same analysis on the
Chapter 4. 3D Common Corruptions

Figure 4.8: Redundancy among corruptions. We quantified the pairwise similarity of a subset of corruptions from 2DCC and 3DCC by computing their correlations in the $\ell_1$ errors of the surface normals predictions (left) and RGB images (right). 3DCC incurs less correlations both intra-benchmark as well as against 2DCC. Thus, 3DCC has a diverse set of corruptions and these corruptions do not have a significant overlap with 2DCC. Using depth as target task yields similar conclusions (full affinity matrices are provided in the appendix).

RGB domain by computing the $\ell_1$ error between clean and corrupted images, again suggesting that 3DCC yields lower correlations.

Figure 4.9: Visual comparisons of 3DCC and expensive After Effects (AE) generated depth of field effect on query images from Hypersim. 3DCC generated corruptions are visually similar to those from AE.

Soundness: 3DCC vs Expensive Synthesis

3DCC aims to expose a model’s vulnerabilities to certain real-world corruptions. This requires the corruptions generated by 3DCC to be similar to real corrupted data. As generating such labeled data is expensive and scarcely available, as a proxy evaluation, we instead compare the realism of 3DCC to synthesis made by Adobe After Effects (AE) which is a commercial product to generate high-quality photorealistic data and often relies on expensive and manual processes. To achieve this, we use the Hypersim [213] dataset that comes with high-resolution z-depth labels. We then generated 200 images that are near- and far-focused using 3DCC and AE. Figure 4.9 shows sample generated images from both approaches that are perceptually similar. Next, we computed the prediction errors of a baseline normal model when the input is from 3DCC or AE. The scatter plot of $\ell_1$ errors are given in Fig. 4.10 and demonstrates a strong correlation, 0.80, between the two approaches. For calibration and control, we also provide the scatter
4.5 Experiments

plots for some corruptions from 2DCC to show the significance of correlations. They have significantly lower correlations with AE, indicating the depth of field effect created via 3DCC matches AE generated data reasonably well.

Effectiveness of applying 3DCC to other datasets

We showed qualitatively in Fig. 4.5 that 3DCC can be applied to standard vision datasets like ImageNet [210] and COCO [211] by leveraging predicted depth from a state-of-the-art model from MiDaS [268]. Here, we quantitatively show the impact of using predicted depth instead of ground truth. For this, we use the Replica [87] dataset that comes with ground truth depth labels. We then generated 1280 corrupted images using ground truth depth and predicted depth from MiDaS [268] without fine-tuning on Replica. Figure 4.11 shows the trends on three corruptions from 3DCC generated using ground truth and predicted depth. The trends are similar and the correlation of errors is strong (0.79). This suggests that the predicted depth can be effectively used to apply 3DCC to other datasets, and the performance is expected to improve with better depth predictions.

Figure 4.10: Corruptions of 3DCC are similar to expensive realistic synthetic ones while being cheaper to generate. Scatter plots of $\ell_1$ errors from the baseline model predictions on 3DCC against those created by Adobe After Effects (AE). The correlation between 3DCC near (far) focus and those from AE near (far) focus is the strongest (numbers are in the legend of left column). We also added the most similar corruption from 2DCC (defocus blur) for comparison, yielding weaker correlations (middle). Shot noise (right) is a control baseline, i.e. a randomly selected corruption, to calibrate the significance of the correlation measure.

3DCC evaluations on semantic tasks

The previous benchmarking results were focusing on surface normals and depth estimation tasks. Here we perform a benchmarking on panoptic segmentation and object recognition tasks as additional illustrative 3DCC evaluations. In particular for panoptic segmentation, we use semantic corruptions from Sec. 4.3.1, and for object classification, we introduce
Chapter 4. 3D Common Corruptions

Figure 4.11: **Effectiveness of applying 3DCC without ground truth depth.** Three corruptions from 3DCC are generated using depth predictions from MiDaS [268] model on unseen Replica data. Scatter plots show the \( \ell_1 \) errors from the baseline model when corruptions are generated using the predicted depth (x-axis) or the ground truth (y-axis). The trends are similar between two corrupted data results, suggesting the predicted depth is an effective approximation to generate 3DCC. See the appendix for more tests including control baselines.

![Scatter plots](image)

Figure 4.12: **Qualitative results of learning with 3D data augmentation** on random queries from OASIS [270], AE (Sec. 4.5.2), manually collected DSLR data, and in-the-wild YouTube videos for surface normals. The ground truth is gray when it is not available, e.g. for YouTube. The predictions in the last two rows are from the O+DPT+2DCC+3D (Ours) model. It is further trained with cross-task consistency (X-TC) constraints [72] (Ours+X-TC). They are noticeably sharper and more accurate.

![Qualitative results](image)

*ImageNet-3DCC* by applying corruptions from 3DCC to ImageNet validation set, similar to 2DCC [2].

**Semantic corruptions:** We evaluate the robustness of two panoptic segmentation models from [132] against *occlusion corruption* of 3DCC. The models are trained on Omnidata [132] and Taskonomy [88] datasets with a Detectron [271] backbone. See the
4.5 Experiments

appendix for details.

Figure 4.13: **Robustness against occlusion corruption of 3DCC.** The plot shows the intersection over union (IoU) scores of Detectron models [271] for different objects over a range of occlusion ratios. The models are trained on Taskonomy [88] or Omnidata [132] datasets. The occlusion ratio is defined as the number of occluded pixels divided by the sum of occluded and visible pixels of the object. This is computed over the test scenes of Replica [87]. The plots expose the occlusion handling capabilities of the models and show that the Omnidata trained model is generally more robust than the Taskonomy one. The degradation in model predictions is class-specific and becomes more severe with higher occlusion ratios.

![Occlusion Robustness Plot]

Figure 4.14: **Robustness on ImageNet-3DCC and ImageNet-2DCC.** Errors on ImageNet validation images corrupted by 3DCC and 2DCC are computed for the models in robustness leaderboards [2, 264]. Following [2], we compute the mean corruption error (mCE) relative to AlexNet [101]. The performance degrades significantly against ImageNet-3DCC, thus it can serve as a challenging benchmark. As expected, the general trends are similar between the two benchmarks as 2D and 3D corruptions are not completely disjoint. A similar observation was also made in [230] even when the corruptions are designed to be dissimilar to 2DCC. Still, there are notable differences that can be informative during model development by exposing trends and vulnerabilities that are not captured by 2DCC, e.g. ANT [41] has better mCE on 2DCC compared to AugMix [36], while they perform similarly on 3DCC. Likewise, combining DeepAugment [7] with AugMix improved the performance on 2DCC significantly more than 3DCC. See the appendix for more results.

Figure 4.13 quantifies the effect of occlusion on the predictions of models, i.e. how the models’ intersection over union (IoU) scores change with increasing occlusion, for selected objects. This is computed on the test scenes from Replica [87]. The Omnidata trained
model is generally more robust than the Taskonomy one, though we see a decrease in IoU in both models as occlusion increases. The trends are class-specific possibly due to shape of the objects and their scene context, e.g. fridge predictions remain unchanged up until 0.50 occlusion ratio, while couch predictions degrade more linearly for Omnidata model. This evaluation showcases one use of semantic corruptions in 3DCC, which are notably harder to accomplish using other benchmarks that do not operate based on 3D scans.

**ImageNet-3DCC:** We compare performances of the robust ImageNet models [96, 41, 36, 7] from RobustBench [264] and ImageNet-2DCC [2] (i.e. ImageNet-C) leaderboards in Fig. A.39. Following 2DCC, we compute mean corruption error (mCE) by dividing the models errors by AlexNet [101] errors and averaging over corruptions. The performance of models degrades significantly, including those with diverse augmentations. Thus, ImageNet-3DCC can serve as a challenging benchmark for object recognition task. As expected, while the general trends are similar between the two benchmarks as 2D and 3D corruptions are not completely disjoint [230], 3DCC exposes vulnerabilities that are not captured by 2DCC, which can be informative during model development. See the appendix for further results.

### 4.5.3 3D data augmentation to improve robustness

We demonstrate the effectiveness of the proposed augmentations qualitatively and quantitatively. We evaluate UNet and DPT models trained on Taskonomy (T+UNet, T+DPT) and DPT trained on Omnidata (O+DPT) to see the effect of training dataset and model architecture. The training procedure is as described in Sec. 4.5.1. For the other models, we initialize from O+DPT model and train with 2DCC augmentations (O+DPT+2DCC) and 3D augmentations on top of that (O+DPT+2DCC+3D), i.e. our proposed model. We also further trained the proposed model using cross-task consistency (X-TC) constraints from [72], denoted as (Ours+X-TC) in the results. Lastly, we evaluated a model trained on OASIS training data from [270] (OASIS).

**Qualitative evaluations:** We consider i. OASIS validation images [270], ii. AE corrupted data from Sec. 4.5.2, iii. manually collected DSLR data, and iv. in-the-wild YouTube videos. Figure 4.12 shows that predictions made by the proposed models are significantly more robust compared to baselines.

**Quantitative evaluations:** In Table 4.1, we compute errors made by the models on 2DCC, 3DCC, AE, and OASIS validation set (no fine-tuning). Again, the proposed models yield lower errors across datasets showing the effectiveness of augmentations. Note that robustness against corrupted data is improved without sacrificing performance on in-the-wild clean data, i.e. OASIS.
4.6 Conclusion and Limitations

We introduce a framework to test and improve model robustness against real-world distribution shifts, particularly those centered around 3D. Experiments demonstrate that the proposed 3D Common Corruptions is a challenging benchmark that exposes model vulnerabilities under real-world plausible corruptions. Furthermore, the proposed data augmentation leads to more robust predictions compared to baselines. We believe this direction opens up a promising direction in robustness research by showing the usefulness of 3D corruptions in benchmarking and training. Below we briefly discuss some of the limitations:

3D quality: 3DCC is upper-bounded by the quality of 3D data. The current 3DCC is an imperfect but useful *approximation* of real-world 3D corruptions, as we showed. The fidelity is expected to improve with higher resolution sensory data and better depth prediction models.

Non-exhaustive set: Our set of 3D corruptions and augmentations are not *exhaustive*. They instead serve as a starter set for researchers to experiment with. The framework can be employed to generate more domain-specific distribution shifts with minimal manual effort.

Large-scale evaluation: While we evaluate some recent robustness approaches in our analyses, our main goal was to show that 3DCC successfully exposes vulnerabilities. Thus, performing a comprehensive robustness analysis is beyond the scope of this work. We encourage researchers to test their models against our corruptions.

Balancing the benchmark: We did not explicitly balance the corruption types in our benchmark, e.g. having the same number of noise and blur distortions. Our work can further benefit from weighting strategies trying to calibrate average performance on corruption benchmarks, such as [272].

Use cases of augmentations: While we focus on robustness, investigating their usefulness
Chapter 4. 3D Common Corruptions

on other applications, e.g. self-supervised learning, could be worthwhile.

Evaluation tasks: We experiment with dense regression tasks. However, 3DCC can be applied to different tasks, including classification and other semantic ones. Investigating failure cases of semantic models against, e.g. on smoothly changing occlusion rates for several objects, using our framework could provide useful insights.

This chapter is based on the paper: O. F. Kar, T. Yeo, A. Atanov, A. Zamir, 3D Common Corruptions, CVPR 2022. I contributed to generating and evaluating on the semantic corruptions, several other qualitative evaluations and quantitative evaluations on OASIS.
Conclusion

This thesis touched upon three directions for making models useful in the real world with a focus on improving robustness. For the first direction, we discussed robustness mechanisms. There, we demonstrated how ensembling predictions from diverse cues can be used to improve the final prediction. We also showed how generative models can be used to generate targeted training data. For the second direction on adaptation mechanisms, we presented an approach for efficient adaptation at test-time. Finally, for the last direction, we proposed realistic corruptions that can be used as an evaluation benchmark or as augmentations. However, this thesis has by no means fully addressed its goal of making models more robust and adaptive. We will now discuss its limitations and future work.

Limitations and Future Work

Focus on low level shifts like common corruptions. In all chapters, the bulk of our evaluations were on (3D) Common Corruptions, followed by several cross-dataset evaluations and unlabelled random images or videos from the internet. While many works focus on robustness to corruptions, distribution shifts that can be encountered in the real world are rich, complex, and not limited to these shifts. At the high level, there can be unseen shifts, e.g., new geographic locations, spurious correlations due to e.g., objects co-occurring with certain backgrounds in the training data, or underrepresented subpopulations. Furthermore, there have been massive efforts in the last few years in putting together datasets with different kinds of shifts into an easy-to-use evaluation framework e.g., WILDS [18, 273], Shift happens benchmark\(^1\). The methods proposed in this thesis may be able to handle a wider range of shifts than what we have considered, performing such evaluations would be interesting future work.

Benefits of scale. In Chapter 4, we saw that training a transformer model on a mixture of datasets with heavy data augmentations resulted in significantly improved predictions, and at times outperforming the predictions of the proposed methods in the other chapters. This is in line with other recent works on classification that show that large pre-training datasets and models improve robustness [102, 103]. This begs the question, how far

\(^{1}\)https://github.com/shift-happens-benchmark/icml-2022
Limitations and Future Work

can scaling up take us and when do we need carefully curated inductive biases e.g., ensembling from diverse cues with explicit merging mechanisms? In this thesis, we have implicitly assumed that the benefits of the latter, i.e., a more interpretable framework that is less likely to fail silently, outweigh the benefits of a potentially more accurate system that is less interpretable and may fail silently. However, which to choose likely depends on context and application.

Efficient and continuous adaptation via universal representations. In Chapter 3, we introduced a method for efficient adaptation. This is a promising direction, as it is based on the premise that we are not able to anticipate all possible shifts, thus, one should learn to adapt at test-time. Furthermore, there can also be shifts in the target task. Our current models require substantial compute and data to adapt to new domains and tasks. To do well on a single task/domain, the system learns to be invariant to attributes that are irrelevant to the task/domain. However, on unseen tasks/domains, the seen task’s/domain’s irrelevant attribute may be necessary for the new unseen task.

Thus, decomposing the input into different attributes can allow for more efficient adaptation to new tasks/domains. Having decomposed the input, a learnable function is used to select relevant attributes, depending on their relevance to the target task. Independent modules can be used to process the different attributes and later aggregated to make a final prediction. Thus, this results in a system that is potentially more robust as the failure of one module is independent of the others. It can also be more efficient as adapting to new tasks/domains only requires learning which attributes to use as opposed to fine-tuning the entire network.

Language as an inductive bias. Language encodes some information about how we see and understand the world, it may provide useful inductive biases that can make our models more aligned with how we make decisions or perceive the world. In Chapter 2, we made use of text-to-image models to generate training data. Controlling the generation in the text space can allow us to generate data that is potentially more interpretable. Another example is if performing the decomposition described above using a visual-language model to attain more interpretable attributes would result in a model that is more robust.

Active perception involves interaction and exploration to develop a better understanding of the world. It allows for active data collection to e.g., minimize the uncertainty of our predictions. More concretely, the input to the network can be heavily occluded or corrupted that none of our advances with robustness methods on static images can work. However, being able to change the camera’s parameters e.g., viewpoint, or zoom can allow us to get a better view of the object and, thus, make a better prediction. Furthermore, being able to interact with the environment allows for the learning of causal representations. Such representations encode the data generation process or the relationship between events. Thus, predictions based on these representations can be
Limitations and Future Work

potentially more robust.

**Learning with richer inputs.** This thesis mostly focused on learning from static I.I.D. images, however, this is far removed from how we learn. We benefit from a stream of rich, continuous inputs of different modalities [274]. Existing works have shown that this can also be beneficial for neural networks [275, 276]. We are interested in enriching our data by combining different modalities, including non-visual ones e.g., audio, and haptics feedback. Each modality offers different benefits, e.g., audio feedback gives us information about objects or events that are not in our line of sight, and haptic feedback gives us a description of the object in terms of its shape, weight, or material. Thus, the network can learn to integrate these modalities, and in the event that any input modality is corrupted, it is able to use the others to return an accurate prediction.

The distribution shifts that neural networks will encounter in the real world can be rich and complex. Despite the advancements in the community, neural networks remain brittle or only work well under certain shifts. This thesis presented several directions toward the goal of having models that work well in the real world. We hope that it can inspire people to think about the many possible directions for tackling this problem.
A Appendix
Appendix A. Appendix

A.1 Ensembling diverse predictions

A.1.1 Quantitative Results

Performance on Distorted Data

Performance under common corruptions and dataset shift: Figure A.7 shows the $\ell_1$ errors of our method and several baselines for each distortion, for normal, reshading and depth targets. The proposed inverse variance and network merging approaches consistently outperform baselines. In Table A.1 we report results on Replica with common corruptions (Replica-C) and Habitat. They show supportive trends for our method, similar to Taskonomy results in Fig. 6 of main paper.

<table>
<thead>
<tr>
<th>Method</th>
<th>Replica-C</th>
<th>Habitat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normals</td>
<td>Reshading</td>
</tr>
<tr>
<td>Blind guess</td>
<td>15.0</td>
<td>17.0</td>
</tr>
<tr>
<td>Baseline</td>
<td>6.2</td>
<td>12.1</td>
</tr>
<tr>
<td>Deep ensembles</td>
<td>6.0</td>
<td>11.6</td>
</tr>
<tr>
<td>Ours</td>
<td>5.1</td>
<td>10.1</td>
</tr>
</tbody>
</table>

Table A.1: Robustness on common corruptions and dataset shift. The table provides quantitative evaluation on Replica-C and Habitat. $\ell_1$ losses are reported, multiplied by 100 for readability. Our proposed method outperforms the baselines.

Perceptual losses: Similar to the evaluation setup by [72], besides the direct metric, we also report perceptual errors in Figure A.5 for a more complete evaluation. Such metrics evaluate the same prediction in a different representation space (e.g. depth $\rightarrow$ normal) to give a non-uniform weighting to pixels depending on their properties. The proposed method yields notably lower perceptual errors for all target domains compared to baselines.

Additional ablation results: Figure A.6 demonstrates the performance when the proposed method does not use sigma training and training with consistency constraints. This evaluation is performed for both target and perceptual domains. We also provide the results when deep ensembles are trained with consistency constraints. Our method significantly outperforms the baselines in all evaluations. In Figure A.8 we further demonstrate the performance for each distortion.

Performance on Clean Data

In Table A.4, we report quantitative evaluations on the undistorted Taskonomy [88] and Replica [87] datasets for the target domains depth, reshading, and surface normals. Our method variants yield similar performance on in-distribution data, while also being more robust against distribution shifts as shown in the main paper. This demonstrates the robustness on out-of-distribution data did not come at the cost of degraded performance for in-distribution data. We do not expect robust methods (ours and deep ensembles) to
yield a better performance on in-distribution data as well, since they are not designed to do so and do not have a reason for that. However, we observe that they occasionally fit better to even the in-distribution data, as shown in Table A.4.

<table>
<thead>
<tr>
<th>Noise Blur Weather Digital</th>
<th>Clean error</th>
<th>Avg.</th>
<th>Gauss.</th>
<th>Shot</th>
<th>Impulse</th>
<th>Defocus</th>
<th>Gauss</th>
<th>Motion</th>
<th>Zoom</th>
<th>Studio</th>
<th>Frost</th>
<th>Bright</th>
<th>Contrast</th>
<th>Elastic</th>
<th>Pixel</th>
<th>JPEG</th>
<th>IPEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep ensembles</td>
<td>21.5</td>
<td>32.2</td>
<td>67.9</td>
<td>68.6</td>
<td>71.2</td>
<td>71.7</td>
<td>82.1</td>
<td>75.8</td>
<td>77.3</td>
<td>69.1</td>
<td>67.2</td>
<td>59.1</td>
<td>51.3</td>
<td>55.8</td>
<td>82.1</td>
<td>54.4</td>
<td>65.9</td>
</tr>
<tr>
<td>Ours</td>
<td>20.3</td>
<td>31.8</td>
<td>69.8</td>
<td>68.5</td>
<td>65.9</td>
<td>49.6</td>
<td>50.0</td>
<td>45.8</td>
<td>41.4</td>
<td>41.8</td>
<td>42.7</td>
<td>39.0</td>
<td>46.6</td>
<td>46.0</td>
<td>32.7</td>
<td>39.0</td>
<td>36.4</td>
</tr>
<tr>
<td>Deep ensembles</td>
<td>18.3</td>
<td>27.4</td>
<td>55.7</td>
<td>58.4</td>
<td>58.5</td>
<td>9.9</td>
<td>26.0</td>
<td>28.3</td>
<td>25.5</td>
<td>24.8</td>
<td>27.1</td>
<td>18.3</td>
<td>20.2</td>
<td>20.2</td>
<td>22.8</td>
<td>26.2</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>16.5</td>
<td>24.3</td>
<td>46.5</td>
<td>36.1</td>
<td>33.0</td>
<td>18.4</td>
<td>22.7</td>
<td>23.7</td>
<td>23.3</td>
<td>17.3</td>
<td>18.0</td>
<td>15.0</td>
<td>28.0</td>
<td>28.0</td>
<td>28.1</td>
<td>26.8</td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Robustness on ImageNet-C, CIFAR-100-C, and CIFAR-10-C. Error on clean and distorted data. For all metrics, lower is better. All methods are trained only on clean ImageNet, CIFAR-100, and CIFAR-10 training data, respectively. Our method performs noticeably better compared to deep ensembles and a single model baseline ResNet.

**Additional ablations with deep ensembles**

The key role of middle domains can be further seen from Figure A.2. It shows the error against distribution shift when deep ensembles is trained with consistency constraints (cons) and sigma training (ST) for Reshade task. Merging is done with both the proposed uncertainty based weighting (inv. var. merging) and the average of the predictions (uniform merging). Thus, the only remaining difference between deep ensembles and ours is in utilizing middle domains. In both cases, our method clearly outperforms.

**Path Importance**

Figure A.9 shows the performance for individual distortions as a function of number of paths. The proposed method consistently outperforms deep ensembles and the performance improves with more paths. In Table A.3 and Figure A.4, we further investigate the importance of each employed path and best performing paths for the depth, reshading, and normals prediction.

**Effect of Employed Distortions for Sigma Training**

Here we investigate the effect of using different distortions for the sigma training (ST) step. For this we consider four paths: rgb, greyscale, sharpened, 2D edges. We combine the predictions from these paths using the inverse variance merging with three different distortion sets for the ST stage: ST #1 uses Gaussian blur and Gaussian noise similar to the results in the main paper while ST #2 uses speckle noise and pixelate distortions and ST #3 uses pixelate and Gaussian blur distortions. Figure A.1 shows the average $\ell_1$ error evaluated on nine unseen distortions for these variants. As can be seen, all variants perform similarly, suggesting that the ST step does not require handpicking the training distortions, and can be performed successfully with any distortion with sufficiently high intensity.
Appendix A. Appendix

Figure A.1: **Sigma training does not require handpicking the training distortions.** We consider inverse variance merging with four paths (direct, grey, sharpened, edge) for reshading prediction. ST #1 is trained with Gaussian blur and Gaussian noise, while ST #2 is trained with speckle noise and pixelate distortions, and ST #3 is trained with pixelate and Gaussian blur distortions (no ST shows the results when there is no sigma training). The evaluations are performed over nine unseen distortions (shot noise, impulse noise, defocus blur, glass blur, contrast, brightness, saturate, jpeg compression, spatter) for these variants. As can be seen, sigma training performs similarly, suggesting that any distortion with sufficiently high intensity works.

Figure A.2: **Deep ensembles trained with consistency constraints and sigma training (ST).** The only remaining difference between deep ensembles and ours is in utilizing middle domains. Our method outperforms for both inv. var. merging (left) and uniform merging (right).

**Performance on Imagenet-C, CIFAR-100-C, and CIFAR-10-C**

The absolute errors on ImageNet-C, CIFAR-100-C, and CIFAR-10-C are given in Table A.2. For deep ensembles and our method, we use the same backbone as the baseline ResNet model. Our method significantly outperforms the baselines for all three classification tasks.
Figure A.3: Statistical informed guesses ("Blind Guess") on the Taskonomy dataset for depth, normal, and reshading tasks. Top row shows the predicted mean and the bottom row shows the predicted standard deviation. The predictions minimize the NLL loss on the training dataset.

Table A.4: Robustness on out-of-distribution data did not come at the cost of degrading performance on in-distribution data. The table provides quantitative evaluation on undistorted Taskonomy and Replica test sets. Results are reported for depth, reshading, and normal using direct and perceptual error metrics. The perceptual metrics evaluate the target prediction in another domain, similar to [72]. $\ell_1$ losses are reported, multiplied by 100 for readability. Our proposed method outperforms on the perceptual metrics while being comparable in the direct metrics, showing that the performance does not decrease on undistorted data while being robust against distribution shifts.

Figure A.4: Importance of each middle domain for different distortion. The order of the best performing paths for depth, reshade, and surface normals predictions for each of the distortions, with 8 denoting the most important and 1 the least important one, is shown. This is an extension of Fig. 7b in the main paper with reshade and depth target domains.
Appendix A. Appendix

A.1.2 Qualitative Results

Video Evaluation

We perform video evaluations on distorted and undistorted video clips to further evaluate the performance of our method against several baselines. We strongly recommend watching them.

Distorted Images and External Queries

We provide qualitative results in Figure A.10 on out of distribution data that is markedly different from those seen during training to demonstrate the generalizability of our method. Furthermore, in Figures A.11, A.12, A.13, A.14 we provide additional prediction results for distorted Replica test images for normals, reshading, and depth prediction.

A.1.3 Further Method Details

Formulation for Learning Based Merging

Multi-modal predictions: For ill-posed problems such as ours, adopting mixture models allows us to capture inherent ambiguity in the data by assuming that there are several possible distributions that could have generated the observed data (e.g. there can be several depth estimates that corresponds to a given RGB image). Given $K$ random variables, their mixing weights $\{\hat{w}_i\}_{i=1}^K$ reflect the uncertainty over which of the $K$ variables generated the observed data. From Sec. 3.1 in the main paper, we estimate the parameters of these $K$ distributions, $\{\hat{z}_i, \hat{s}_i\}_{i=1}^K$, then function $m$ learns to output the mixing weights given these set of parameters. Our final distribution is a mixture of Laplacians,

$$h_{Mix}(z) = \sum_i \hat{w}_i h(z | \hat{z}_i, \hat{s}_i) \quad (A.1)$$

where $h(z | \hat{z}_i, \hat{s}_i)$ is the probability density function of a Laplace distribution with mean $\hat{z}_i$ and scale $\exp(\hat{s}_i)$. The final loss is the NLL of the multi-modal prediction

$$\mathcal{L}_m = \frac{1}{N} \sum_{n=1}^N - \log h_{Mix}(z_n).$$

Mixture-weights: The final prediction is computed as the weighted average of mixture components’ means using the weights $\hat{w}_i$ as in Eq. A.1. We approximate the weights for a path to be proportional to its mixture probability distribution function (p.d.f.) evaluated at its estimate $\hat{z}_i$, i.e. $\hat{w}_i \propto h_{Mix}(\hat{z}_i)$. To enforce that $h_{Mix}$ is a p.d.f, we require $\hat{w}_i \geq 0$ and $\sum_i \hat{w}_i = 1$.

Learning based merging: We consider a stacking model [98] that obtains the final predictions given the outputs from each path using the predicted mixture weights $\hat{w}_i$. It has the advantage that the loss is over the entire image, thus, taking into account its spatial structure. This option is denoted as Network merging.
A.1 Ensembling diverse predictions

Training with Cross-Task Consistency Loss

The proposed ensembling method can be further augmented with “cross-task consistency constraints” [72], to ensure that predictions from the different paths are in cross-task agreement. While this is not a fundamental step for the method, it yields better accuracy especially in fine-grained regions. Following [72], we consider a set of perceptual loss networks on the outputs of $g$. This corresponds to minimizing an $\ell_1$ error between the predictions obtained by the model and those from the ground truth in the perceptual domain.

Adversarial Attack/Defence Setup

To generate the I-FGSM attack [10], we apply the following:

$$X_0 = X$$
$$X_{n+1} = \text{Clip}_{X,\epsilon}\{X_n + \alpha \text{sign}(\nabla J(X_n, y))\}$$

(A.2) \hspace{1cm} (A.3)

Similar to [10], we set $\alpha = 1$ in our experiments and the number of iterations given by $N = \min(4 + \epsilon, 1.25\epsilon)$. For adversarial training, we finetune the Baseline UNets with an equal number of adversarial and clean examples in the training data. The former were generated from a I-FSGM attack with $\epsilon = (0 - 16]$.

Table A.5 shows that the proposed method outperforms the baselines against corruptions generated with I-FSGM.

Blind Guess Baseline

This baseline is computed using the following formula:

$$g^* = \min_g \mathcal{L}_{NLL}$$

(A.4)

The resulting “blind guess” minimizes expected NLL loss on the training dataset. Hence it is a statistically informed guess which does not look at the input for predicting the label. A visualization of these guesses for depth, normal, and reshading are provided in Figure A.3.

A.1.4 Middle domain definitions

Here we define the middle domains. They can be extracted with standard libraries such as OpenCV [93], Scikit-Image [277], or implemented in Pytorch [217]. Our Pytorch implementation for them can be found here.

2D edges are from the output of a Canny edge detector without non-maximum suppression.
Appendix A. Appendix

<table>
<thead>
<tr>
<th>Methods</th>
<th>Normal</th>
<th>Reshade</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Baseline UNet</td>
<td>8.23</td>
<td>11.53</td>
<td>14.39</td>
</tr>
<tr>
<td>Deep ensembles</td>
<td>7.49</td>
<td>11.13</td>
<td>13.36</td>
</tr>
<tr>
<td>Uniform merging</td>
<td>7.77</td>
<td>9.48</td>
<td>11.00</td>
</tr>
<tr>
<td>Inv. var. merging</td>
<td>7.60</td>
<td>8.89</td>
<td>10.40</td>
</tr>
<tr>
<td>Adv. T. (lower bound error)</td>
<td>5.50</td>
<td>6.76</td>
<td>8.36</td>
</tr>
<tr>
<td>Greyscale images are computed by taking the average over the R, G, and B channels of the input image.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embossed: We perform the filtering using four emboss kernels with different directions and concatenate the embossed outputs.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Laplace edges: Similar to 2D edges, we perform the filtering using a Laplacian filter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low-pass filtered: We perform a low pass filtering with a Gaussian filter.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sharpened: We sharpen the image by subtracting its low-pass filtered version and then adding the original image.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wavelet: We use the Daubechies-1 with 3 levels of decomposition to obtain wavelet coefficients. We then perform bilinear upsampling on the low resolution coefficients to have the same size as the original image.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A.1.5 Visualizations of Common Corruption on Taskonomy data

Common Corruptions [2] used during evaluation are shown for a sample Taskonomy image in Figure A.15.
A.1 Ensembling diverse predictions

Figure A.5: **Average $\ell_1$ loss over 11 unseen distortions in perceptual domains.** Similar to Fig. 6 in the main paper, this shows the performance for unseen distortions where $\ell_1$ losses are computed in *perceptual* domains instead, e.g. for the top left plot, loss in the reshape inferred out of the predicted depth is computed. The proposed ensembling approaches are more robust against shift with increasing intensities in perceptual domains as well.

Figure A.6: **Average $\ell_1$ loss over 11 unseen distortions in target and perceptual domains.** This shows the performance in target and perceptual domains when the proposed method does not use sigma training or training with consistency constraints.
Figure A.7: $\ell_1$ error for each distortion for our method against several baselines for normal and reshaed task. This figure is an extension of Fig. 6 in the main paper for individual distortions.
A.1 Ensembling diverse predictions

Figure A.7: $\ell_1$ error for each distortion for our method against several baselines for depth task. This figure is an extension of Fig. 6 in the main paper for individual distortions.

Figure A.8: $\ell_1$ error for each distortion for our method without sigma training and consistency training stages. This figure is an extension of Fig. 7a (top) in the main paper for individual distortions.
Figure A.8: $\ell_1$ error for each distortion for our method without sigma training and consistency training stages for reshape and depth task. This figure is an extension of Fig. 7a (top) in the main paper for individual distortions.
A.1 Ensembling diverse predictions

Figure A.9: **Robustness as a function of number of paths** for normal and reshape tasks. This is an extension of Fig. 7a (bottom) in the main paper and shows the $\ell_1$ error for each unseen distortion as we increase the number of paths. For deep ensembles, this corresponds to increasing the number of ensemble components. The proposed method (inv. var. merging) and its simpler variants (uniform merging) consistently outperforms deep ensembles.
Figure A.9: Robustness as a function of number of paths for depth task. This is an extension of Fig. 7a (bottom) in the main paper and shows the $\ell_1$ error for each unseen distortion as we increase the number of paths. For deep ensembles, this corresponds to increasing the number of ensemble components. The proposed method (inv. var. merging) and its simpler variants (uniform merging) consistently outperforms deep ensembles.

Figure A.10: Qualitative results on undistorted external query images for the proposed method and baselines trained with (Cons) and without (No Cons) consistency constraints.
Figure A.11: Qualitative results on Replica images with pixelate distortion (shift intensity 3) for the proposed method and baselines trained with (Cons) and without (No Cons) consistency constraints.
Figure A.12: Qualitative results on Replica images with shot noise distortion (shift intensity 3) for the proposed method and baselines trained with (Cons) and without (No Cons) consistency constraints.
Figure A.13: Qualitative results on Replica images with defocus blur distortion (shift intensity 3) for the proposed method and baselines trained with (Cons) and without (No Cons) consistency constraints.
Figure A.14: Qualitative results on Replica images with JPEG distortion (shift intensity 3) for the proposed method and baselines trained with (Cons) and without (No Cons) consistency constraints.
Figure A.15: **Visuals of common corruptions** used for a single image sample with increasing severities from left to right.
Appendix A. Appendix

A.2 Fast adaptation using test-time feedback

A.2.1 Monocular Depth

Training Details

All networks for our method and baselines use the same UNet architecture [94] and were trained with AMSGrad [95], with a learning rate of $5 \times 10^{-4}$, weight decay $2 \times 10^{-6}$ and batch size of 64.

Rapid Network Adaptation (RNA). We aim to fuse the additional information provided by $g$ at test-time into $f_\theta$ via a small network, $h_\phi$. We insert $k$ Feature-wise Linear Modulation (FiLM) layers into the network $f_\theta$, we denote this network as $f_{\hat{\theta}}$. Each FiLM layer performs the following operation:

$$\text{FiLM}(x_i; \gamma_i, \beta_i) = \gamma_i \odot x_i + \beta_i,$$

where $x_i$ is the activation of layer $i$. For depth estimation, $h_\phi$ only predicts $\{\beta_i\}_{i=1}^k$, thus, $\gamma_i = 1, i = 1 \ldots k$ as it gave similar results with fewer parameters. $h_\phi$ consists of an encoder with 3 downsampling blocks and $k$ convolution layers (one for each FiLM layer) to predict the $\beta_i$ parameters. $h_\phi$ was trained by randomly masking the GT, with sparsity ratio ranging from 0% to 0.25%. $h_\phi$ is trained using the following loss function:

$$\min_{\phi} \mathcal{L} = \sum_{n=1}^N \|f_{\hat{\theta}}(x_n; h_\phi(y_n \odot \bar{m}_n)) - y_n\|_1.$$  \hfill (A.5)

where $\bar{m}_n$ is the mask that simulates the sparsity pattern that will be encountered at test-time. If $g$ extracts sparse depth via SFM, we use the model from [278] to extract keypoints locations to generate $\bar{m}$. This creates a sparse binary mask that we apply on the GT. We then add Gaussian noise to $y_n \odot \bar{m}_n$ to account for potential outliers. At test-time, RNA only requires a forward pass through $h_\phi$ and $f_{\hat{\theta}}$.

Test-Time Optimization (TTO) details. We use the same optimization parameters as in training time. Optimization is done for 10 iterations for each batch. With the exception of the TENT baseline, all parameters of the model were updated. For TENT, only the GroupNorm parameters were updated as it results in more stable optimization. Test-Time Optimization minimizes the following loss function:

$$\min_{\hat{\theta}} \mathcal{L} = \sum_{n=1}^N \|f_{\hat{\theta}}(x_n) \odot m_n - z_n\|_1.$$  \hfill (A.6)

where $N$ is the number of datapoints, $\odot$ is the element-wise product and $\hat{\theta}$ is the subset of parameters of $f_\theta$ that is updated i.e., $\hat{\theta} \subseteq \theta$. 

100
A.2 Fast adaptation using test-time feedback

Additional Results

RNA with existing supervision signals. We show that one can train a controller network to use existing adaptation signals, e.g. prediction entropy and Sobel edges to get better performance at test time. As described in Sec. 4.1 in the main paper, to use supervision from entropy, we train a Baseline UNet model with NLL loss. The model outputs the prediction and an uncertainty estimate of the predictions. We also show results with the model from [142] with calibrated uncertainty estimates as it was shown to predict uncertainties that are better correlated with error. The results are shown in Table A.6. For all cases, RNA improves on the performance over the baseline, even if the adaptation signal is poor. This is in contrast to TTO, where the performance can be worse after adaptation, e.g. after optimizing entropy.

<table>
<thead>
<tr>
<th>Method</th>
<th>Supervision</th>
<th>Sobel edge</th>
<th>Entropy</th>
<th>Entropy (calibrated)</th>
<th>Sparse GT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNA</td>
<td></td>
<td>0.29</td>
<td>0.07</td>
<td>0.12</td>
<td>4.06</td>
</tr>
<tr>
<td>TTO (GN)</td>
<td></td>
<td>-0.02</td>
<td>-0.11</td>
<td>0.04</td>
<td>0.55</td>
</tr>
<tr>
<td>TTO (F)</td>
<td></td>
<td>0.06</td>
<td>-1.21</td>
<td>0.06</td>
<td>2.76</td>
</tr>
</tbody>
</table>

Table A.6: RNA can be used with existing supervision signals. $\ell_1$ errors on the depth prediction task. The numbers are relative to the baseline error (i.e. the difference between that method’s $\ell_1$ error and that of the pre-adaptation baseline’s). F denotes TTO by optimizing all parameters and GN denotes TTO by optimizing only group normalization parameters.

Controlling for different number of parameters. Table A.7 shows the results on Common Corruptions applied to Taskonomy test set. All methods have the same architecture, thus, same number of parameters. RNA still outperforms, thus, its performance is not due to extra parameters or architecture.

<table>
<thead>
<tr>
<th>Shift</th>
<th>Pre-adaptation</th>
<th>Densification</th>
<th>TTO</th>
<th>RNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>0.045</td>
<td>0.023</td>
<td>0.019</td>
<td>0.018</td>
</tr>
</tbody>
</table>

Table A.7: Controlling for number of parameters. $\ell_1$ errors (multiplied by 100 for readability) on the depth estimation task, evaluated on the Taskonomy test set under a subset of common corruptions. Each method is using the same architecture and number of parameters. The adaptation signal here is masked GT, fixed at 0.05% of valid pixels.

Implementations of RNA with different architectures. We experimented with several versions of RNA. Instead of adding FiLM layers to adapt $f_\theta$ (denoted as FiLM-$f$), we also added FiLM layers to a UNet model that is trained to update the input image $x$ (denoted as FiLM-$x$). For FiLM-$x$, only $x$ is updated and there is no adaptation on $f_\theta$. Lastly, as Hypernetworks [145] have been shown to be expressive and suitable for adaptation, we trained a HyperNetwork, in this case an MLP, to predict the weights of a 3-layer convolutional network that updates $x$ (denoted as HyperNetwork-$x$). In all cases, RNA takes in the sparse error, i.e., the difference between the predictions and the
adaption signal, as opposed to separate inputs as done in the main paper. The results of adaptation with these variants of RNA are shown in Table A.8. The FiLM-f variant performed best, thus, we adopted it as our main architecture.

<table>
<thead>
<tr>
<th>Method</th>
<th>Shift</th>
<th>Taskonomy-CC</th>
<th>Taskonomy-3DCC</th>
<th>HyperSim</th>
<th>BlendedMVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-adaptation</td>
<td>0.027</td>
<td>0.057</td>
<td>0.048</td>
<td>0.336</td>
<td>3.450</td>
</tr>
<tr>
<td>RNA (HyperNetwork-x)</td>
<td>0.019</td>
<td>0.041</td>
<td>0.033</td>
<td>0.277</td>
<td>2.567</td>
</tr>
<tr>
<td>RNA (FiLM-x)</td>
<td>0.019</td>
<td>0.039</td>
<td>0.033</td>
<td>0.279</td>
<td>2.636</td>
</tr>
<tr>
<td>RNA (FiLM-f)</td>
<td>0.013</td>
<td>0.024</td>
<td>0.020</td>
<td>0.198</td>
<td>2.310</td>
</tr>
</tbody>
</table>

Table A.8: RNA with different architectures. \( \ell_1 \) errors on the depth estimation task under distribution shifts are reported. The adaptation signal here is masked GT, fixed at 0.05% of valid pixels.

**RNA performs better than training a single model that takes as input the RGB image and sparse supervision concatenated.** We call this model Multi-domain. For 0.05% GT supervision, RNA has a much better performance, with an average loss over all distortions and severities of 0.0179 while the multi-domain model has an average loss of 0.0255 (see Fig. A.16).

**Results for different levels of GT supervision.** In Fig. A.16, we show how the error changes with increasing GT supervision for our proposed methods and the baselines. Note that for the two RNA variants (frozen \( f \) and jointly trained \( f \)), and multi-domain model, we also included the case where only the supervision signal \( z \) in passed as input to \( h_\phi \). These methods have been post-fixed with ‘\( z \) input only’ in the legend.

**Model-based RNA.** During training of RNA, we applied several augmentations on both \( x \) (speckle noise, Gaussian noise, spatter) and \( y \) (affine transformation). The augmentations on \( x \) are those from the validation set of Common Corruptions [2]. Training with validation corruptions has also been done in [41]. The motivation for \( y \) augmentation is that there is a scale ambiguity for monocular depth estimation. Thus, we can change the scale of the ground truth depth label, while keeping \( x \) the same, this trains RNA to predict depth values in different ranges. We call the parameters of these augmentations e.g., the kernel size for Gaussian blur, environmental parameters.

We trained RNA to predict environmental parameters, and to use these parameters for adaptation. Thus, compared to the variants of RNA mentioned before, we are now forcing RNA to learn a model of the environment. Training is done in several stages. First, we trained \( h_\phi^1 \) to take in environmental parameters and update \( x \). This is akin to a controller. Next, we train \( h_\phi^2 \) to predict these environment parameters from the adaptation signal and predictions, akin to a sensor. Finally, we finetune both \( h_\phi^1 \) and \( h_\phi^2 \). Thus, \( h_\phi = h_\phi^2 \circ h_\phi^1 \). This model is denoted as (model-based). We also trained a version of this with a single training stage, i.e., \( h_\phi \) is not forced to learn to predict the environment parameters (denoted as model-free). The results are shown in Tab. A.9. Although the performance of both version of RNA are similar, we believe this is an interesting future direction.
A.2 Fast adaptation using test-time feedback

<table>
<thead>
<tr>
<th>Method\Shift</th>
<th>None</th>
<th>Taskonomy-CC</th>
<th>Taskonomy-3DCC</th>
<th>Hypersim</th>
<th>BlendedMVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-adaptation</td>
<td>0.025</td>
<td>0.053</td>
<td>0.045</td>
<td>0.336</td>
<td>3.450</td>
</tr>
<tr>
<td>RNA (model-based)</td>
<td>0.020</td>
<td>0.042</td>
<td>0.035</td>
<td>0.212</td>
<td>2.281</td>
</tr>
<tr>
<td>RNA (model-free)</td>
<td>0.020</td>
<td>0.041</td>
<td>0.033</td>
<td>0.215</td>
<td>2.326</td>
</tr>
</tbody>
</table>

Table A.9: **Comparison of model-based and model-free versions of RNA.** $\ell_1$ errors on the depth estimation task, under distribution shifts. The adaptation signal here is masked GT, fixed at 0.05% of valid pixels.

Figure A.16: $\ell_1$ error vs different levels of GT supervision. These results are on Taskonomy-CC data, as described in Sec. 4.3 in the main paper. Each curve shows an average over all the distortions listed in the title.

**Qualitative results.** We provide more qualitative results in Fig. A.23 where our method outperforms the baselines.

A.2.2 Optical Flow Experiments

To predict optical flow, we use a pre-trained RAFT model from [215]. We use sparse optical flow as our supervision signal, attained from keypoint matching between images. We perform TTO to adapt the model. We use the same episodes from Replica+CC as described in Sec. 4.3 in the main paper. TTO was done for 10 iterations for each episode, all parameters of the RAFT model were updated. Figure A.17 shows the results.
Figure A.17: **Quantitative results for optical flow.** These results were attained from Replica+CC data, as described in Sec 4.3 in the main paper.

Adaptation with TTO results in an 8.5% improvement over the baseline.

### A.2.3 Dense 3D Reconstruction

**Test-Time Optimization details.** As mentioned in the main paper, we achieve multi-view consistency using the same process as [179], where: 1. every pixel is backprojected into 3D world-coordinates using the estimated depth and camera poses, 2. optical-flow predictions are employed to establish dense correspondences across pixels, 3. weights of the depth model are optimized to minimize the discrepancy between the estimated 3D world coordinates of corresponding pixels.

We provide additional qualitative results and the corresponding error images in Fig. A.18.

### A.2.4 Semantic Segmentation

**Training Details**

**TTO.** We optimize by SGD with 0.0001 learning rate, 0.9 momentum, batch size 2 for 10 iterations per batch.

**TENT.** Following [53], we optimize by SGD with 0.0001 learning rate, 0.9 momentum, batch size 1. As TENT is unstable for online and multi-iteration optimization, we restart the model after each batch, i.e. episodic optimization, and run 1 iteration per batch.

**TENT (all).** In contrast to [53] which only updates batch normalization parameters at test-time, we also included a TENT baseline that optimizes all parameters. We optimize by SGD with 0.00001 learning rate, 0.9 momentum, batch size 1. We run the optimization for each image for 10 iterations to be comparable to the TTO model. Note that we reduced learning rate by a factor of 10 as TENT was unstable. Since TENT and TENT (all) models perform similarly, we only show the TENT results in the main paper. See A.2.4 for all results.
A.2 Fast adaptation using test-time feedback

Figure A.18: Extension of Fig. 4 from main paper to show more examples and error maps.

**RNA.** As an encoder we used a small CNN with 3 downsampling blocks. We trained the FiLM generator with frozen segmentation model on the clean COCO training dataset with cross entropy losses. During the training we sparsify the target segmentation mask uniformly in [0,30] pixels to generate sparse ground truth inputs to the encoder. We optimize by Adam with 0.0001 learning rate and 0.0001 weight decay. We select the model with highest mean IOU on the clean validation set. The model is trained with two forward passes. During the first pass the input to FiLM encoder is sparse ground truth and zeros as prediction. After this, in the second pass the encoder takes the sparse ground truth and the prediction from the previous pass as input. We compute the cross-entropy loss for the prediction of second pass.

**Densification.** It uses the same FCN-ResNet50 model as other baselines and is trained with the same setup as RNA.

**Additional Results**
Appendix A. Appendix

Quantitative Results. We provide the results for mean IOU vs number of pixels for different severities in Fig. A.24. Figures A.25, A.26, A.27, A.28, A.29, A.30, A.31 give a breakdown of the performance for our methods and baselines for each corruption, severity level, and number of pixel annotations.

Qualitative Results. We include additional qualitative comparisons between our methods and baselines in Figures A.32, A.33, A.34.

A.2.5 Image classification

Generating Coarse Label Sets

One method to generate coarse label sets using WordNet tree [208] is proposed in [194]. This method and other clustering methods create imbalanced coarse labels and too many ImageNet classes are assigned to coarse labels that are either too coarse or too fine-grained (See Figure A.19 and Table A.14 for the statistics). As we aim to use coarse supervision to adapt models at test-time, we focus on generating more balanced coarse labels, as explained below.

To generate balanced coarse label sets, instead of going from top to bottom or bottom to top for a fixed depth in WordNet tree, we follow a different approach. For each ImageNet class we go up until we get to a hypernym that has a certain number of hyponyms that are ImageNet classes. That number determines the coarsity level of the coarse label. To achieve this, we use a priority-based selection criteria where we define certain ranges to reach a given coarsity level. Using this approach, we created three different coarse sets with 26, 45, and 85 coarse labels. See Tables A.10, A.11, A.12, A.13 for the coarse labels and their IDs. The resulting sets are more balanced than the 127-label set provided in [194]. See Figures A.19, A.20, A.21, A.22 and Table A.14 for more details about the statistics of the coarse label sets.

Training Details

The baseline model used for classification is ResNet50 trained on ImageNet.

TTO. We optimize by SGD with 0.00025 learning rate, 0.9 momentum and batch size 64, following [53]. The following loss function is used for TTO, which is a linear combination of cross-entropy loss on the summation of probabilities of all of the classes in the coarse label set, and the entropy of the predictions:

\[
\min_{\phi} \mathcal{L} = -\log \sum_{c \in \mathcal{P}} p_c + w_c \sum_{c} -p_c \log p_c
\]

where \( p_c \) is the probability of class \( c \) and \( \mathcal{P} \) is the set of classes that are present in the coarse label.
A.2 Fast adaptation using test-time feedback

**TENT.** We used the optimizer and parameters that were reported in [53] to adapt TENT. For both TTO and TENT, we optimize the transformation parameters of the normalization layers and estimate the normalization statistics from the current batch. Note that for each batch we re-evaluate after the updates (in our experiments we run 1 iteration per batch) to get the final predictions.

**RNA.** We optimize by Adam with 0.0001 learning rate, 0.0001 weight decay, batch size 64 and for about 50 epochs. The FiLM generator we used has an encoder-decoder structure. The encoder has one hidden layer with 128 nodes and 64 nodes for the output layer. The decoder for each FiLM layer has one hidden layer with 64 nodes and the output size is equal to the number of FiLM layer parameters. The FiLM layers are inserted between normalization layers and ReLUs. During training all of the model parameters are fixed and only the FiLM generator parameters are being trained, and cross-entropy loss is minimized.

**Additional Results**

In Figures A.35, A.36, A.37 we provide a breakdown of performance against individual corruptions from ImageNet-C and ImageNet-3DCC using 26-, 45-, and 85-way coarse labels. We also included the results when we used 127-way coarse labels from [194] in Fig. A.38. Note that this coarse set has imbalances as explained in A.2.5, yet our methods can still benefit from it.
A.2 Fast adaptation using test-time feedback

Table A.13: The classes used in the 26-coarse label set. See Figure A.22 for more details.

<table>
<thead>
<tr>
<th>Coarse Label Set</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Median</th>
<th>Mode</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>127-way</td>
<td>1</td>
<td>218</td>
<td>88.40</td>
<td>59.0</td>
<td>218</td>
<td>79.96</td>
</tr>
<tr>
<td>85-way</td>
<td>6</td>
<td>522</td>
<td>44.05</td>
<td>24.0</td>
<td>26</td>
<td>59.10</td>
</tr>
<tr>
<td>45-way</td>
<td>7</td>
<td>522</td>
<td>59.33</td>
<td>50.0</td>
<td>67</td>
<td>60.27</td>
</tr>
<tr>
<td>26-way</td>
<td>7</td>
<td>522</td>
<td>105.86</td>
<td>67.0</td>
<td>158</td>
<td>84.77</td>
</tr>
</tbody>
</table>

Table A.14: The statistics of coarse label sets. See Figures A.19, A.20, A.21, A.22 for more details.
Figure A.20: Distribution of 85-coarse label set. None of ImageNet classes are using coarse labels with coarsity level of 5 or less, 114 classes are using coarse labels with coarsity level of 10 or less, and 28 classes are using the coarse labels with coarsity level of 200 or more.
A.2 Fast adaptation using test-time feedback

Figure A.22: **Distribution of 26-coarse label set.** None of ImageNet classes are using coarse labels with coarsity level of 5 or less, 38 classes are using coarse labels with coarsity level of 10 or less, and 113 classes are using the coarse labels with coarsity level of 200 or more.
Figure A.23: **Supplementary results for monocular depth estimation.** Qualitative comparison of our method vs baselines on query images from ScanNet, Replica, and Taskonomy datasets.

Figure A.24: **Supplementary results for semantic segmentation.** Mean IOU vs number of pixels for different severities. Numbers in the legend denote the average over all pixel levels.
A.2 Fast adaptation using test-time feedback

Figure A.25: **Supplementary results for semantic segmentation.** Mean IOU vs individual corruptions for different severities when the number of pixels is 3. Numbers in the legend denote the average over the corruptions.
Figure A.26: **Supplementary results for semantic segmentation.** Mean IOU vs individual corruptions for different severities when the number of pixels is 4. Numbers in the legend denote the average over the corruptions.
Figure A.27: **Supplementary results for semantic segmentation.** Mean IOU vs individual corruptions for different severities when the number of pixels is 5. Numbers in the legend denote the average over the corruptions.
Figure A.28: **Supplementary results for semantic segmentation.** Mean IOU vs individual corruptions for different severities when the number of pixels is 10. Numbers in the legend denote the average over the corruptions.
A.2 Fast adaptation using test-time feedback

Figure A.29: **Supplementary results for semantic segmentation.** Mean IOU vs individual corruptions for different severities when the number of pixels is 15. Numbers in the legend denote the average over the corruptions.
Figure A.30: **Supplementary results for semantic segmentation.** Mean IOU vs individual corruptions for different severities when the number of pixels is 20. Numbers in the legend denote the average over the corruptions.
A.2 Fast adaptation using test-time feedback

Figure A.31: **Supplementary results for semantic segmentation.** Mean IOU vs individual corruptions for different severities when the number of pixels is 25. Numbers in the legend denote the average over the corruptions.
Figure A.32: **Supplementary results for semantic segmentation.** Qualitative comparison of our method vs baselines for defocus blur and glass blur corruptions applied to COCO validation images.
### A.2 Fast adaptation using test-time feedback

Figure A.33: **Supplementary results for semantic segmentation.** Qualitative comparison of our method vs baselines for JPEG compression and motion blur corruptions applied to COCO validation images.

<table>
<thead>
<tr>
<th>Input Image</th>
<th>Ground Truth</th>
<th>Baseline</th>
<th>TENT</th>
<th>Densification</th>
<th>TTO</th>
<th>RNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG Compr.</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
<td><img src="image5" alt="Image" /></td>
<td><img src="image6" alt="Image" /></td>
</tr>
<tr>
<td>Motion Blur</td>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
<td><img src="image9" alt="Image" /></td>
<td><img src="image10" alt="Image" /></td>
<td><img src="image11" alt="Image" /></td>
<td><img src="image12" alt="Image" /></td>
</tr>
</tbody>
</table>
Figure A.34: **Supplementary results for semantic segmentation.** Qualitative comparison of our method vs baselines for shot noise corruption applied to COCO validation images.
A.2 Fast adaptation using test-time feedback

Figure A.35: Supplementary results for ImageNet classification. Error for individual corruptions from ImageNet-C and ImageNet-3DCC. TTO and RNA uses 26-way coarse label supervision. Numbers in the legend denote the average over the corruptions.
Figure A.36: **Supplementary results for ImageNet classification.** Error for individual corruptions from ImageNet-C and ImageNet-3DCC. TTO and RNA uses 45-way coarse label supervision. Numbers in the legend denote the average over the corruptions.
A.2 Fast adaptation using test-time feedback

Figure A.37: **Supplementary results for ImageNet classification.** Error for individual corruptions from ImageNet-C and ImageNet-3DCC. TTO and RNA uses 85-way coarse label supervision. Numbers in the legend denote the average over the corruptions.
Appendix A. Appendix

Figure A.38: Supplementary results for ImageNet classification. Error for individual corruptions from ImageNet-C and ImageNet-3DCC. TTO and RNA uses 127-way coarse label supervision from [194]. Numbers in the legend denote the average over the corruptions.
A.3 3D Common Corruptions

A.3.1 Quantitative Results

Robustness mechanisms against 3DCC and 2DCC

Figure A.43 shows $\ell_1$ errors of robustness mechanisms against individual corruptions in 3DCC for surface normals estimation. Figure A.44 shows the same result for depth estimation. For both tasks, 3DCC leads to significantly degraded predictions for models trained with robustness mechanisms. For completeness, we also provide performances of these models against corruptions in 2DCC, in Figures A.45 and A.46.

Redundancy between 3DCC and 2DCC

We provide in Figures A.47 and A.48 the full affinity matrices between 2DCC and 3DCC by computing the correlations of $\ell_1$ errors made in the surface normals and depth estimation tasks, respectively. Figure A.49 shows the same result by computing $\ell_1$ errors in the RGB domain. As can be seen, 3DCC yields lower correlations both intra-benchmark and against 2DCC.

Effectiveness of predicted depth to generate 3DCC

We compare the effectiveness of using MiDaS [268] depth estimation model to generate 3DCC against two control baselines in Fig A.42. The first one is incorrect instance depth where we randomly swap depth predictions for a given RGB image with another depth prediction. The second one is blind guess depth which minimizes the expected likelihood loss in the training dataset, hence it is a statistically informed guess reflecting the dataset regularities [142, 72] (See Fig. A.41 for its visualization). As shown in Fig. A.42, the predicted depth yields higher correlation with the ground truth compared to control baselines, showing that it can be used to generate 3DCC for datasets without 3D information.

Comparing robust ImageNet models on 2DCC and 3DCC benchmarks

We compare performances of the robust ImageNet models from RobustBench and ImageNet-C leaderboards below (See the links for full model names) in Figure A.39. A quick look at the scatter plot (top) shows that the general trends between 2DCC and 3DCC are similar, an observation also made in [230] even when the corruptions are designed to be dissimilar to 2DCC. Hence, this is also expected for our case as 2D and 3D corruptions are not completely disjoint (expected). But, we observe notable differences between the two benchmarks in local trends, e.g. in the ellipsoid regions certain robustness mechanisms improved performance on 2DCC while being ineffective against 3DCC.

The bar plot (bottom) comparing a subset of corruptions that exists in both benchmarks
Appendix A. Appendix

Figure A.39: Comparing ImageNet models on 2DCC and 3DCC benchmarks (i.e. ImageNet-C vs ImageNet-3DCC). **Top:** Comparison of mean corruption errors (MCEs) on 2DCC and 3DCC. **Bottom:** Comparison of MCEs for a subset of corruptions that exists in both benchmarks (e.g. 2D defocus blur vs its 3D version). See Sec. A.3.1 for details.

(e.g. 2D defocus blur vs its 3D version), further reflects the differences where **1.** all models have consistently higher errors on 3D corruptions compared to their 2DCC counterparts and **2.** certain models, e.g. AugMix and AugMix+DA, face a larger drop in performance on 3DCC compared to the other models, indicating that AugMix may be biased towards 2DCC. Thus, 3DCC evaluations can be informative during model development as they **expose nonlinear trends and vulnerabilities that are not captured by 2DCC** (also discussed in Sec. 5.2.2).

**Performance of 3D data augmentation**

We show in Fig. A.50 the performance of 3D data augmentation and baselines on individual corruptions from 3DCC for surface normals estimation task. Similarly, Fig. A.51 shows the performance on 2DCC and Table A.15 provides full performance metrics on OASIS [270] benchmark. As can be seen from the results, 3D data augmentation notably boosts robustness.

**A.3.2 Qualitative Results**

**Video evaluations**

We perform evaluations on clips from manually collected DSLR data, YouTube videos, Adobe After Affects (AE) generated corrupted data (Sec.5.2.3 in the main paper), and sample queries from OASIS [270]. They suggest the proposed 3D data augmentation yields notably sharper and more accurate predictions with less flickering, compared to
A.3 3D Common Corruptions

baselines. We recommend watching the clips on the project page.

Additional queries

In addition to video evaluations in Sec. A.3.2 we provided additional results on OASIS and AE datasets in Figures A.52 and A.53, again suggesting 3D data augmentation is beneficial for improving robustness.

A.3.3 Further method details

Data augmentation mechanisms

Below we provide additional details about the training procedures of data augmentation models. All models were finetuned from the Baseline UNet (T+UNet) with an equal number of clean and augmented images.

Adversarial training: The adversarial examples are generated from an I-FSGM attack with $\epsilon = (0 - 16)$. To generate the I-FGSM attack [10], we apply the following:

$$X_0 = X,$$

$$X_{n+1} = \text{Clip}_{X, \epsilon}\{X_n + \alpha \text{sign}(\nabla J(X_n, y))\}$$

where $J$ is the loss function. Similar to [10], we set $\alpha = 1$ in our experiments and the number of iterations given by $N = \min(4 + \epsilon, 1.25\epsilon)$.

Style [96]: We applied the AdaIN style transfer [150]. The stylization coefficient is randomly selected from the range $[0.1, 0.5]$.

DeepAugment [7]: We use the same perturbations as [7], with the exception of the ones that change the scene geometry, e.g. rotation, flipping.

Implementing corruptions

We release the full open source code of our pipeline, which enables using the implemented corruptions on any dataset. Below, we provide further details about implementing corruptions.

Depth of field: We divide the scene into two regions using hyperfocal distance [279] which is the focus distance yielding the largest depth of field. We then define near focus region as parts of the scene closer to camera than hyperfocal distance (and vice versa for far focus). After picking the focus region, we perform the blurring (See Fig. 3 right of main paper for an illustration.)

Video: We use ffmpeg scripts to generate video-based corruptions.

Semantics: We use Replica[87] dataset which comes with 3D mesh annotations. To
Appendix A. Appendix

<table>
<thead>
<tr>
<th></th>
<th>Angular error°</th>
<th>% within t°</th>
<th>Relative Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
<td>11.25°</td>
</tr>
<tr>
<td>T+UNet</td>
<td>30.49</td>
<td>22.93</td>
<td>23.18</td>
</tr>
<tr>
<td>T+DPT</td>
<td>32.13</td>
<td>25.68</td>
<td>18.82</td>
</tr>
<tr>
<td>OASIS</td>
<td>24.63</td>
<td>19.06</td>
<td>30.10</td>
</tr>
<tr>
<td>O+DPT</td>
<td>24.42</td>
<td>18.46</td>
<td>28.82</td>
</tr>
<tr>
<td>O+DPT+2DCC</td>
<td>23.67</td>
<td>17.75</td>
<td>30.24</td>
</tr>
<tr>
<td>O+DPT+2DCC+3D (Ours)</td>
<td>24.65</td>
<td>18.53</td>
<td>28.89</td>
</tr>
<tr>
<td>Ours (+X-TC [72])</td>
<td>23.89</td>
<td>18.34</td>
<td>28.66</td>
</tr>
</tbody>
</table>

Table A.15: Evaluations on OASIS. Similar to Table 1 in the main paper, but results for more metrics are shown.

Figure A.40: Visualizations of view change corruptions from 3DCC for different sampled angles.

obtain occlusion masks, while it is possible to perform ray-tracing, it could be expensive and time-consuming. Thus, we also investigated an alternative approach and modified the mesh by removing all the objects except the target one and rendering semantic masks. Performing a second rendering when all objects are in the mesh, i.e. original state, yields the semantic masks with occlusions (e.g. blue masks in Fig 1 and Fig. 4 in the main paper). The difference between the two masks correspond to occlusion mask (e.g. red masks in Fig. 1 and Fig. 4 in the main paper).

A.3.4 Visualizing Corruptions

We show visualizations of corruptions from 3DCC and 2DCC for 5 shift intensities in Figures A.54 and A.55, respectively. Furthermore, we also show samples from view changes corruptions from 3DCC in Fig. A.40.
A.3 3D Common Corruptions

Figure A.41: Statistically informed blind guess depth prediction of Taskonomy dataset [142, 72].

Figure A.42: Effectiveness of applying 3DCC without ground truth depth. Similar to Fig. 12 in the main paper, but control baselines are also provided, namely incorrect instance depth and blind guess depth. The predicted depth from MiDaS [268] model yields the strongest correlations with the ground truth depth.

Figure A.43: Average $\ell_1$ losses of robustness mechanisms against corruptions in 3DCC for surface normals estimation. Similar to Fig. 6 in the main paper, but demonstrates the performance for individual corruptions in 3DCC.
Appendix A. Appendix

Figure A.44: **Average $\ell_1$ losses of robustness mechanisms against corruptions in 3DCC for depth estimation.** Similar to Fig. 6 in the main paper, but demonstrates the performance for individual corruptions in 3DCC.

Figure A.45: **Average $\ell_1$ losses of robustness mechanisms against corruptions in 2DCC for surface normals estimation.** Similar to Fig. 6 in the main paper, but demonstrates the performance for individual corruptions in 2DCC.
Figure A.46: **Average $\ell_1$ losses of robustness mechanisms against corruptions in 2DCC for depth estimation.** Similar to Fig. 6 in the main paper, but demonstrates the performance for individual corruptions in 2DCC.
Figure A.47: Redundancy among corruptions in 2DCC and 3DCC in the $\ell_1$ errors of surface normals prediction. Similar to Fig. 9 in the main paper, but are shown for all corruptions.
Figure A.48: Redundancy among corruptions in 2DCC and 3DCC in the $\ell_1$ errors of depth prediction. Similar to Fig. 9 in the main paper, but are shown for all corruptions.
Figure A.49: Redundancy among corruptions in 2DCC and 3DCC in the $\ell_1$ errors of RGB images. Similar to Fig. 9 in the main paper, but are shown for all corruptions.
A.3 3D Common Corruptions

Figure A.50: Average $\ell_1$ losses of 3D data augmentation and baselines against corruptions in 3DCC for surface normals estimation. Similar to Tab. 1 in the main paper, but demonstrates the performance for individual corruptions in 3DCC.

Figure A.51: Average $\ell_1$ losses of 3D data augmentation and baselines against corruptions in 2DCC for surface normals estimation. Similar to Tab. 1 in the main paper, but demonstrates the performance for individual corruptions in 2DCC.
Figure A.52: **Qualitative results on corruptions generated with After Effects.** An extension of the qualitative results in Fig. 8 of the main paper showing the benefits of 3D augmentation. See Sec. 5.2.3 of the main paper for details on how the corruptions were generated.
A.3 3D Common Corruptions

Figure A.53: Qualitative results on OASIS [270]. An extension of the qualitative results in Fig. 8 of the main paper showing the benefits of 3D augmentation.

Figure A.54: Visualizations of corruptions from 3DCC for 5 shift intensities.
Figure A.55: Visualizations of corruptions from 2DCC for 5 shift intensities.
Bibliography


PMLR, 2013.


Bibliography


Bibliography


Bibliography


[255] M. Elad and M. Aharon, “Image denoising via sparse and redundant representations over learned dictionaries,” *IEEE Transactions on Image processing*, vol. 15, no. 12,


Email: teresa.ysq@gmail.com  Website: aserety.github.io

EDUCATION

École Polytechnique Fédérale de Lausanne  Lausanne, CH
PhD in Computer Science  Sept 2017 - Aug 2023
Advisors: Amir Zamir, Pierre Dillenbourg
University of Cambridge  Cambridge, UK
MPhil in Machine Learning and Machine Intelligence  Sept 2015 - Aug 2016

EXPERIENCE

Teaching Assistant, EPFL  Sept 2017 - Aug 2022
Fall 2021: CS503 Visual Intelligence: Machines and Minds
Spring 2018, 2019, 2020: EE559 Deep Learning
Fall 2020: MATH111 Linear Algebra
Fall 2019: CS433 Machine Learning

PAPERS

Training Data Generation with Diffusion Models for Robustness  In Progress
T. Yeo, A. Atanov, A. Alekseev, H. Benoit, R. Ray, P. Esmaeil, A. Zamir
4M: Massively Multimodal Masked Modelling  NeurIPS’23 (Spotlight)
D. Mizrahi, R. Bachmann, O. F. Kar, T. Yeo, M. Gao, A. Dehghan, A. Zamir
Fast Adaptation of Neural Networks Using Test-Time Feedback  ICCV’23
T. Yeo, O. F. Kar, Z. Sodagar, A. Zamir
Task Discovery: Finding the Tasks that Neural Networks Generalize on  NeurIPS’22
A. Atanov, A. Filatov, T. Yeo, A. Sakhmshetty, A. Zamir
3D Common Corruptions and Data Augmentation  CVPR’22 (Oral)
O. F. Kar, T. Yeo, A. Atanov, A. Zamir
Robustness via Cross-domain Ensembles  ICCV’21 (Oral)
T. Yeo*, O. F. Kar*, A. Sax, A. Zamir
Robust Learning Through Cross-Task Consistency  Arxiv’20
A. Zamir*, A. Sax*, T. Yeo, O. F. Kar, N. Cheerla, R. Suri, Z. Cao,
J. Malik, L. Guibas
Iterative Classroom Teaching  AAAI’19 (Oral)
T. Yeo, P. Kamalaruban, A. Singla, A. Merchant, T. Asselborn, L. Faucon,
P. Dillenbourg, V. Cevher