Rapid Network Adaptation: Learning to Adapt Neural Networks Using Test-Time Feedback

1. Overview and Discussion

1.1. Outline

We provide further discussions, details and evaluations in the supplementary material, as outlined below.

- **Overview video** providing a short summary of the paper and qualitative results (on the project page)
- A discussion on handling distribution shifts using different methods and the positioning of RNA (Sec. 1.2).
- Additional details and results for monocular depth estimation (Sec. 2)
- Additional details and results for optical flow estimation (Sec. 3)
- Additional details and results for dense 3D reconstruction (Sec. 4)
- Additional details and results for semantic segmentation (Sec. 5)
- Additional details and results for image classification (Sec. 6)
- Our development [code](#) with documentation (on the project page)

1.2. A discussion on handling distribution shifts using different methods

There are many methods that aim to handle distribution shifts. Figure 1 gives an overview of how these methods can be characterized. Open-loop systems 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. 1 of the main paper). 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.

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 dis-
tribution shift parameters results in an interpretable system that will not fail silently. However, as it requires creating a model of distribution shifts, it is less likely to adapt to shifts that were not modelled. 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 the paper, it generalizes well to a diverse set of unseen distribution shifts. Note that model-free adaptation methods, including RNA, has schematic similarities to multimodal learning approaches as they simultaneously use 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 multimodal 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 [14, 16, 3, 8]. Since this may be unnecessarily expensive, another way is to amortize this optimization process [1]. Note that we do not need to regress the gradients of the optimization process. Instead, we simulate TTO by training RNA to reduce errors in the predictions using the error feedback signal. RNA achieves this well and performs adaptation orders of magnitude faster than running test-time optimization (TTO).

Below we provide further clarifications and discussions on RNA vs other approaches.

**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 [18], are concerned with reconstructing plausible images without taking the downstream prediction \( x \rightarrow y \) into account (shown as gray in Fig. 1). Moreover, it has been shown that imperceptible artifacts in the denoised & modified image could result in degraded predictions [6, 17, 4]. 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.** See Sec. 4.2 in the main paper for an extensive discussion.

**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 modellling 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.

### 2. Monocular Depth

#### 2.1. Training Details

All networks for our method and baselines use the same UNet architecture [12] and were trained with AMSGrad [11], 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 \( f_{\theta, \phi} \). Each FiLM layer performs the following operation

\[
\text{FiLM}(x; \gamma_i, \beta_i) = \gamma_i \odot x_i + \beta_i,
\]

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:

\[
\min_{\phi} \mathcal{L} = \sum_{n=1}^{N} \left\| f_{\theta}(x_n; h_\phi(y_n \odot \bar{m}_n)) - y_n \right\|_1
\]  

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 [2] to extract keypoints locations to generate \( \hat{m} \). This creates a sparse binary mask that we apply on the GT. We then add Gaussian noise to \( y_n \odot \hat{m}_n \) to account for potential outliers. At test-time, RNA only requires a forward pass through \( h_\phi \) and \( f_{\theta} \).
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:

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

(2)

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., $\hat{\theta} \subseteq \theta$.

2.2. 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 [17] with calibrated uncertainty estimates as it was shown to predict uncertainties that are better correlated with error. The results are shown in Table 1. 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>Sobel edge</th>
<th>Entropy</th>
<th>Entropy (calibrated)</th>
<th>Sparse GT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNA (GN)</td>
<td>0.29</td>
<td>0.07</td>
<td>0.12</td>
<td>4.06</td>
</tr>
<tr>
<td>TTO (GN)</td>
<td>0.02</td>
<td>-0.11</td>
<td>0.04</td>
<td>0.55</td>
</tr>
<tr>
<td>TTO (F)</td>
<td>0.06</td>
<td>-1.21</td>
<td>0.06</td>
<td>2.76</td>
</tr>
</tbody>
</table>

Table 1: 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 2 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>Method</th>
<th>Pre-adaptation</th>
<th>Densification</th>
<th>TTO</th>
<th>RNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>Baseline</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift</td>
<td>Baseline</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: 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.

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. 2).

Results for different levels of GT supervision. In Fig. 2, 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 $h_0$. 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 [6]. Training with validation corruptions has also been done in [13]. 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 \( h^1_{\phi} \) to take in environmental parameters and update \( x \). This is akin to a controller. Next, we train \( h^2_{\phi} \) to predict these environment parameters from the adaptation signal and predictions, akin to a sensor. Finally, we finetune both \( h^1_{\phi} \) and \( h^2_{\phi} \). Thus, \( h_{\phi} = h^2_{\phi} \circ h^1_{\phi} \). 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. 4. Although the performance of both version of RNA are similar, we believe this is an interesting future direction.

### Qualitative results.
We provide more qualitative results in Fig. 9 where our method outperforms the baselines.

### 3. Optical Flow Experiments

To predict optical flow, we use a pre-trained RAFT model from [15]. 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 3 shows the results. Adaptation with TTO results in an 8.5% improvement over the baseline.

### 4. Dense 3D Reconstruction

**Test-Time Optimization details.** As mentioned in the main paper, we achieve multi-view consistency using the same process as [9], 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. 4.

### 5. Semantic Segmentation

#### 5.1. 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 [16], 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 [16] 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 5.2 for all results.

**RNA.** As an encoder we used a small CNN with 3 down-sampling 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.

#### 5.2. Additional Results

**Quantitative Results.** We provide the results for mean IOU vs number of pixels for different severities in Fig. 10. Figures 11, 12, 13, 14, 15, 16, 17 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 18, 19, 20.
6. Image classification

6.1. Generating Coarse Label Sets

One method to generate coarse label sets using WordNet tree [10] is proposed in [7]. This method and other cluster-
ing 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 5 and Table 9 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 5, 6, 7, 8 for the coarse labels and their IDs. The resulting sets are more balanced than the 127-label set provided in [7]. See Figures 5, 6, 7, 8 and Table 9 for more details about the statistics of the coarse label sets.

6.1.1 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 [16]. 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:
\[
\min \mathcal{L} = -\log \sum_c p_c + w_c \sum_c -p_c \log p_c
\]  
(3)

where \( p_c \) is the probability of class \( c \) and \( S \) is the set of classes that are present in the coarse label.

**TENT.** We used the optimizer and parameters that were reported in [16] 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 the output size 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.

### 6.2. Additional Results

In Figures 21, 22, 23 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 [7] in Fig. 24. Note that this coarse set has imbalances as explained in 6.1, yet our methods still can benefit from it.

### Table 5: The classes used in the 127-coarse label set from [7]. See Figure 5 for more details.

<table>
<thead>
<tr>
<th>Class</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bird</td>
<td>n02000954 wading</td>
</tr>
<tr>
<td>mammal</td>
<td>n01726692 snake</td>
</tr>
<tr>
<td>bird</td>
<td>n01674464 lizard</td>
</tr>
<tr>
<td>bird</td>
<td>n01503061 bird</td>
</tr>
<tr>
<td>snail</td>
<td>n02606384 damselfish</td>
</tr>
<tr>
<td>fish</td>
<td>n02605316 butterfly fish</td>
</tr>
<tr>
<td>n02000215 butterfly fish</td>
<td></td>
</tr>
<tr>
<td>n02085920 damselfish</td>
<td></td>
</tr>
<tr>
<td>n02159955 insect</td>
<td></td>
</tr>
<tr>
<td>n02120997 feline</td>
<td></td>
</tr>
<tr>
<td>n02104523 shepherd dog</td>
<td></td>
</tr>
<tr>
<td>n02098550 sporting dog</td>
<td></td>
</tr>
<tr>
<td>n02087551 hunting dog</td>
<td></td>
</tr>
<tr>
<td>n02159955 insect</td>
<td></td>
</tr>
</tbody>
</table>

### Table 6: The classes used in the 85-coarse label set. See Figure 6 for more details.

<table>
<thead>
<tr>
<th>Class</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>bird</td>
<td>n02000954 wading</td>
</tr>
<tr>
<td>mammal</td>
<td>n01726692 snake</td>
</tr>
<tr>
<td>bird</td>
<td>n01674464 lizard</td>
</tr>
<tr>
<td>bird</td>
<td>n01503061 bird</td>
</tr>
<tr>
<td>snail</td>
<td>n02606384 damselfish</td>
</tr>
<tr>
<td>fish</td>
<td>n02605316 butterfly fish</td>
</tr>
<tr>
<td>n02000215 butterfly fish</td>
<td></td>
</tr>
<tr>
<td>n02085920 damselfish</td>
<td></td>
</tr>
<tr>
<td>n02159955 insect</td>
<td></td>
</tr>
<tr>
<td>n02120997 feline</td>
<td></td>
</tr>
<tr>
<td>n02104523 shepherd dog</td>
<td></td>
</tr>
<tr>
<td>n02098550 sporting dog</td>
<td></td>
</tr>
<tr>
<td>n02087551 hunting dog</td>
<td></td>
</tr>
<tr>
<td>n02159955 insect</td>
<td></td>
</tr>
</tbody>
</table>

### References


Table 7: The classes used in the 45-coarse label set. See Figure 7 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>44.0</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 8: The classes used in the 26-coarse label set. See Figure 8 for more details.

Table 9: The statistics of coarse label sets. See Figures 5, 6, 7, 8 for more details.


7 Minyoung Huh, Pulkit Agrawal, and Alexei A. Efros. What makes imagenet good for transfer learning?, 2016. 5, 6, 7, 8, 24

8 Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In International Conference on Machine Learning, pages 6028–6039. PMLR, 2020. 2


13 Evgenia Rusak, Lukas Schott, Roland S Zimmermann, Julian Bitterwolf, Oliver Bringmann, Matthias Bethge, and...
Figure 7: Distribution of 45-coarse label set. None of ImageNet classes are using coarse labels with coarsity level of 5 or less, 44 classes are using coarse labels with coarsity level of 10 or less, and 39 classes are using the coarse labels with coarsity level of 200 or more.

Figure 8: 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.


[17] Teresa Yeo, O’guzhan Fatih Kar, and Amir Zamir. Robustness via cross-domain ensembles. In *Proceedings of
Figure 9: Supplementary results for monocular depth estimation. Qualitative comparison of our method vs baselines on query images from ScanNet, Replica, and Taskonomy datasets.

Figure 10: 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.
Figure 11: 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 12: 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 13: 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 14: 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.
Figure 15: 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 16: 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.
Figure 17: 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 18: Supplementary results for semantic segmentation. Qualitative comparison of our method vs baselines for defocus blur and glass blur corruptions applied to COCO validation images.
Figure 19: Supplementary results for semantic segmentation. Qualitative comparison of our method vs baselines for JPEG compression and motion blur corruptions applied to COCO validation images.
Figure 20: Supplementary results for semantic segmentation. Qualitative comparison of our method vs baselines for shot noise corruption applied to COCO validation images.
Figure 21: Supplementary results for ImageNet classification. Error for individual corruptions from ImageNet-C and ImageNet-3DCC. TTO and RNA use 26-way coarse label supervision. Numbers in the legend denote the average over the corruptions.
Figure 22: 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.
Figure 23: 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.
Figure 24: 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 [7]. Numbers in the legend denote the average over the corruptions.