A APPENDIX

A.1 DETAILS OF THE EXPERIMENTAL SETUP

Data Sets. We use six data sets: The SVHN data set (Netzer et al., 2011) contains images of street view housing numbers, which are transformed to gray-scale during pre-processing. The MNIST data set (LeCun & Cortes, 2010) consists of gray-scale images of handwritten digits. While the CIFAR10 data set (Krizhevsky et al., 2009) contains $3 \times 32 \times 32$ (low-resolution) images, STL10 (Adam Coates, 2011) contains 3×96 (high-resolution) images of objects. Both data sets have 9 overlapping classes (airplane, bird, car, cat, deer, dog, horse, ship, truck) and one different class (frog in CIFAR10, monkey in STL10). The FashionMNIST/FMNIST data set (Xiao et al., 2017) contains gray-scale images of clothes and the KMNIST (Clanuwat et al., 2018) data set consists of gray-scale images of Japanese characters. Each training set is split into in training data (90%) and validation data (10%).

Transfer Learning Tasks Based on the six data sets discussed above we create four transfer learning tasks of different relatedness and difficulty. Usually a transfer learning task is simpler if the source domain is more complex or general compared to the target domain than the other way round. We consider the following transfer tasks (source domain $\rightarrow$ target domain): SVHN $\rightarrow$ MNIST (highly related), CIFAR10 $\rightarrow$ STL10 (related and difficult because of the transfer from a simple/low-resolution to a complex/high-resolution domain), MNIST $\rightarrow$ KMNIST (related) and FMNIST $\rightarrow$ KMNIST (distant).

Models. All models are based on the WideResNet-32-10 architecture (Zagoruyko & Komodakis, 2017) and can be decomposed in a feature extractor and a classifier. The feature extractor consists of 32 convolutional layers and a widening factor of 10. The classifiers consist of one or four fully connected layers with GroupSort activation functions (Anil et al., 2019). Models are implemented in Pytorch and optimized using Adam optimizer and a learning rate of 0.0001. Training is done on GPUs (1 TB SSD) with early stopping by evaluating the validation set loss using a frequency of 2 and a patience of 10 epochs. Weight normalization is implemented such that each row $\boldsymbol{W}_j^{(i)} \in \mathbb{R}^n$ of the weight matrix $\boldsymbol{W}^{(i)}$ is split into directions $\boldsymbol{V}_j^{(i)} \in \mathbb{R}^n$ and magnitude $g_j^{(i)} \in \mathbb{R}$, i.e. $\boldsymbol{W}_j^{(i)} = g_j^{(i)} \boldsymbol{V}_j^{(i)} / ||\boldsymbol{V}_j^{(i)}||$. During training, the parameters are updated and the magnitudes are projected back onto the allowed set restricted by the norm thresholds k_i, i.e. $g_j^{(i)} = \min(g_j^{(i)}, k_i)$. Since $L_2(\boldsymbol{x}) \subseteq L_\infty(\boldsymbol{x})$, this enforces the desired Lipschitz constant w.r.t. L_∞-norm on the weight matrix. We use the L_2 norm of each row, because an update does not clip the largest value to the Lipschitz constant k (and might result in a matrix with $\boldsymbol{W}_{j,k}^{(i)} = k$ for many entries), but preserves the direction of an updated w.r.t. the row during training.

Attacks. We use two different attack types: Noise attacks and Project Gradient Descent (PGD) attacks with attack radii of 0.1. The perturbation is bounded by the L_2-norm and applied to the input after data normalization. For adversarial training we use 10-step PGD attacks, while robustness analysis uses 50-step PGD attacks.

Randomized Smoothing. Randomized smoothing techniques draw samples $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{x}, \sigma)$ from the close neighborhood of input $\boldsymbol{x}$, propagate them through the neural network and aggregate the outputs to obtain a smooth prediction. We use $\sigma = 0.1$, draw 500 samples for each input and bind the probability of returning an incorrect answer/prediction by $\alpha = 10^{-4}$. Since median smoothing for multi-dimensional problems must be adapted to the number of dimensions, we scale such that the overall α remains 10^{-4} as described in Kumar & Goldstein (2021). Smoothing experiments are performed on a balanced, randomly chosen subset of the test set, which consists of 1000 instances.

Adversarial Feature and Input search. Since computing adversarial features $\boldsymbol{z}''$ for a one-layer classifier is a linear optimization problem, we solve it be using Gurobi. For the input search we use project gradient descent (with learning rate 0.05, regularization of 0.1 and at most 1000 steps) and minimize the L_1-distance between adversarial features $\boldsymbol{z}'$ and features $\boldsymbol{z}'' = f(\boldsymbol{x}'')$ corresponding to input $\boldsymbol{x}'$. The adversarial feature search and the input search are performed on a balanced, randomly chosen subset of the test set, which consists of 1000 instances.