Generative Classifiers as a Basis for Trustworthy Image Classification – Appendix –

Contents

A. Methods – Additional Materials

A.1. Out-of-Distribution Detection

Originally, ony a single threshhold on the learned likelihood was used to detect OoD inputs, which fails in several cases where the likelihoods of the inputs are unnaturally high [34]. As a way to correct for this, the typicality test [36] uses both an upper and a lower threshold, centered symmetrically around the mean log-likelihood of the training data (Fig. 8, middle). For our ImageNet models, we observe that the distribution of log-likelihood values in the training set is highly asymmetrical (see Fig. 8). Therefore, we introduce our third possibility, a two-tailed quantile test. Instead of the thresholds being symmetric around the mean, they are chosen so that an equal mass of the log-likelihood histogram lies above the upper and below the lower threshold (Fig. 8, middle). In practice, we only measure minor differences in performance between the single-sample typicality test and the two-tailed quantile test.

All three tests can also be seen as hypothesis tests, with the null hypothesis being that the input is in-distribution. The p-value for the hypothesis test is the fraction of training samples with scores in the OoD-zone, which also equals the false positive rate. To evaluate the OoD detection capabilities, we do not use a single threshold value, but want a measure that is independent of it. This is because the acceptable false negative/false positive trade-off depends on the context/application that the model is used in. By varying the p-value of the test, we produce a receiver operating characteristic (ROC) curve. The area under this curve (ROC-AUC), in percent, serves as a scalar measurement of the OoD detection capabilities. An ROC-AUC of 100% means that the OoD samples are perfectly separated from the in-distribution samples and can always be identified cor-

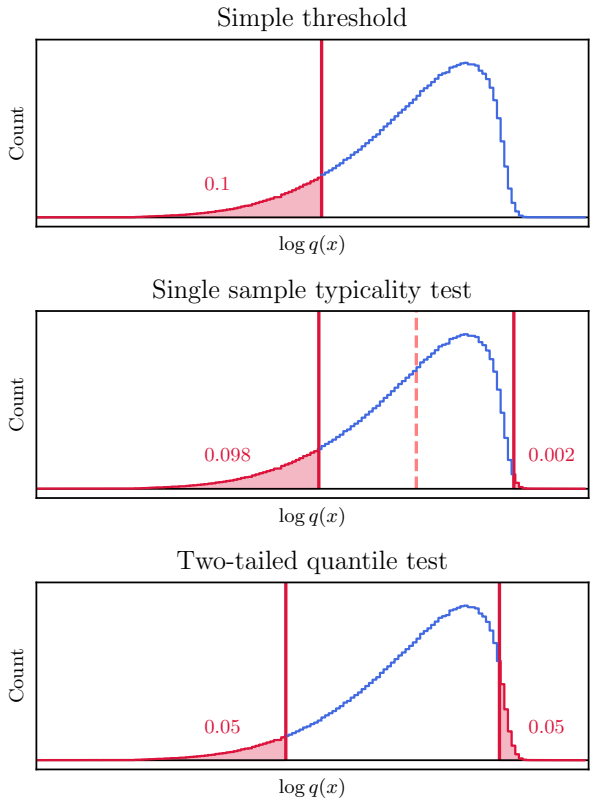


Figure 8: Illustration of three different OoD tests based on the estimated likelihood. The curve shows the distribution of likelihood scores in the training set. The blue part counts as in-distribution, and the red part as OoD. The threshold is chosen such that the red area (false positive rate, p-value), is 0.1 in all three cases, for illustration. In practice, this would be chosen much lower, e.g. 0.001. The small red numbers indicate the fraction of training samples above and below each threshold.

rectly. A value of 50% indicates that the test performs exactly as well as randomly deciding. Below 50%, worse than random performance, the OoD data appears to be more in-distribution as a significant fraction of the training data itself.

B. Experiments – Additional Materials

B.1. Network Architecture

In the following, we outline the design choices and training procedure used for training the INN model as a GC on the ImageNet dataset. It has been noted in the past that there are strong parallels between ResNet residual blocks [21] and INN affine coupling blocks [13], described further below. In fact, under some additional constraints, standard ResNet residual blocks can also be numerically inverted [4]. Therefore, a standard ResNet is not only the most fitting comparison to our GC, but also informs many of our design choices. The argument is, that ResNets contain many carefully tested design choices, leading to their excellent discriminative performance. Adopting these choices where possible saves us from performing an infeasible number of ablations and comparisons ourselves, and still achieve relatively good performance empirically.

Affine coupling operation. As a basic building block of our network, we use the affine coupling block shown in Fig. 9. Such blocks were fist introduced in [13], and are exactly and cheaply invertible, as well as having a tractable Jacobian determinant. The incoming features are first split in two halves, say u_1 and u_2, along the channel dimension. The first half u_1 is not changed, and passed straight through. A subnetwork, similar to the residual subnetwork of a ResNet then predicts affine coefficients s, t from u_1, which are used to perform an affine transformation on the other half of the features u_2. This gives us outputs v_1, v_2:

$$v_2 = s(u_1) \odot u_2 + t(u_1) \quad \text{and} \quad v_1 = u_1 \tag{13}$$

To invert this operation given only v_1, v_2, note that $u_1 = v_1$ is trivially available, so the same coefficients s, t can be re-computed for the inverse. With these, the affine transformation itself can be analytically inverted, to get back $u_2 = (v_2 - t(v_1)) \oslash s(v_1)$. To guarantee invertibility, we restrict $s(\cdot) > 0$. In theory, $s(\cdot) \neq 0$ suffices, but this complicates the situation and does not improve expressive power: mirroring an output dimension is irrelevant for the network and the loss. We ensure $s(\cdot) > 0$ by using $\exp(\alpha \tanh(\cdot))$ activation on the s-outputs of the subnetwork, as previously in [13], where α is a fixed hyperparameter. In principle, $\exp$ alone would be enough, but this leads to instabilities during training, as it can become infinitely large. Importantly to note, the subnetwork itself never has to be inverted, and is always computed forward. Therefore, it can contain the usual operations such as convolutions or batch normalization.

To compare, in a standard ResNet block, a copy operation is used instead of the split, and a simple addition is performed in place of the affine transformation. Apart from this, the structure is very similar.

Complete coupling blocks. The expressive power of the affine coupling above is insufficient: half the data is not touched at all, and the remaining varibles can only be scaled up/down by a factor of at most $\exp(\pm\alpha)$. We add two more invertible operations to solve these problems: We first perform a global channel-wise affine transformation to all variables with scaling s_{global} and bias t_{global}. This technique was already proposed in [13] and refined in [28] as 'ActNorm'. Note that in feed-forward networks, this is also often done as part of the batch normalization layers. Again, s_{global} must be positive, and we achieve the best results choosing $s_{\text{global}} = s_0 \text{softplus}(\gamma) = s_0 \log(1+e^{\gamma})$. Here, γ and t_{global} are learned directly as free parameters, and s_0 is a scalar hyperparameter which we fix to 0.1, while γ is initialized to 10.

Secondly, we want to use a different split in the next block, and therefore have to apply some invertible operation that mixes the channels. So far, there is no 'default' approach to this in the INN literature. Various methods exist, such as simply swapping the two halves [12], learned householder reflections [50], fixed permutations [1], and learning unconstrained mixing matrices [28], among others. While it is desirable to use a learned mixing operations, we do not find any benefits in practice. The method used for [28] has no guaranteed invertibility, and the training can simply crash when the matrix becomes singular. The householder matrices from [50] quickly become computationally expensive with many reflections, and in our case bring no empirical benefit over fixed (not learned) mixing. Instead, we use a random orthogonal matrix from the $O(N)$ Haar distribution after each coupling block, that stays fixed during training. This encourages more mixing than a simple hard permutation, and empirically gives the best results with our architecture.

With an orthogonal mixing matrix, the overall log-Jacobian-determinant of one coupling block can be shown to be

$$\log \left| \det(J) \right| = \sum \log s(u_1) + \sum \log s_{\text{global}}. \tag{14}$$

Due to the chain rule, and product decomposition of the determinant, the sum of the log-Jac-det of each coupling block will give the log-Jac-det of the entire network. An illustration of a coupling block is given in Fig. 9, left.

Subnetworks. We adopt the ResNet design choices for building the affine subnetworks, with one modification: we

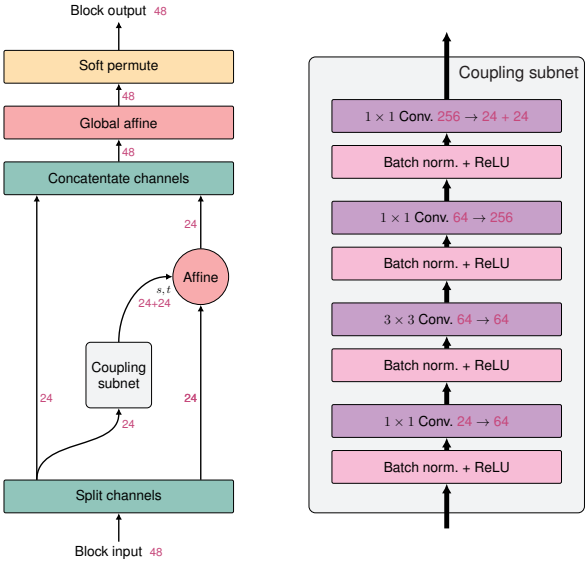


Figure 9: Illustration of the coupling blocks used, as well as the structure of the subnetworks used to predict the affine components. The purple numbers indicate the number of feature channels, given as an example for the fist resolution level (see Table 1, *Conv_2_x*).

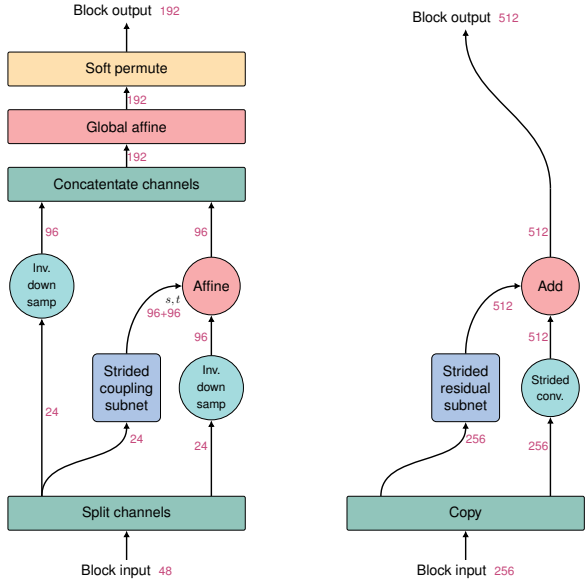


Figure 10: Illustration of our downsampling coupling blocks (*left*), compared to the standard ResNet downsampling blocks (*right*). The invertible downsampling operation (blue circles) reorders inputs in a checkerboard pattern as in [27].

add an additional 1x1 projection layer as the final output. This is motivated by the fact that the INN has less feature maps than the ResNet for all but the last resolution level. Therefore, the expressive power would be limited by only having this few output channels for the final convolution. The subnetwork design is shown in Fig. 9, right.

Downsampling blocks. In the past, various invertible downsampling operations have been used, e.g. [13, 27, 2]. Notably, none of these have a learnable component, such as strided convolutions. Instead, we introduce a *downsampling coupling block*, as a natural extension of the downsampling residual blocks present at the end of each ResNet section. Shown in more detail in Fig. 10, we use two of the invertible re-ordering and re-shaping operations from [27], but nested within a single coupling block. This way, the subnetwork can make use of a strided 3×3 convolution as a learned component to the downsampling. Note that we did not perform rigorous ablations of this introduction, and chose it mainly for better conformity to standard ResNets.

Network layout. The overall network layout is the same as for the standard ResNet-50, which offers a good trade-off between performance and model complexity. The input images are immediately downsampled twice, once using a downsampling coupling block with a 7×7 convolution, then with a Haar wavelet transform as in [2]. The ResNet analogue is the so-called *entry flow*, which also uses a strided 7×7 convolution and a max-pooling operation. A series of coupling blocks follow this, with downsampling blocks distributed throughout, chosen in the same way as for the ResNet-50, detailed in Tab. 1. The output of the INN consists of 3072 two dimensional feature maps at a resolution of 7×7 (compared to 2048 feature maps for the ResNet).

Layer	**Blocks**	**Im. size**	**Channels**		**R.F.**	
			INN	ResNet	INN	ResNet
Input		224	3	3		
Entry flow	1	112	12	64	8	6
Pool (Haar/max)		56	48	64	10	10
Conv_2_x	3	56	48	256	34	34
Conv_3_x	4	28	192	512	106	90
Conv_4_x	6	14	768	1024	314	266
Conv_5_x	3	7	3072	2048	538	426
Pool (DCT/avg.)		1	150 528	2048	∞	∞

Table 4: For each of the resolution levels in the INN and ResNet-50, the number of coupling/residual blocks and spatial size is given, along with the number of feature channels and the maximum possible receptive field (R.F.).

In the ResNet, the output feature maps are passed through a global mean pooling operation. As explained in [26], a discrete cosine transform (DCT) presents the best invertible alternative to this: From our 3072 feature maps, the DCT also produces mean pooled outputs, along with 48 other outputs per feature map, that encode the remaining in-

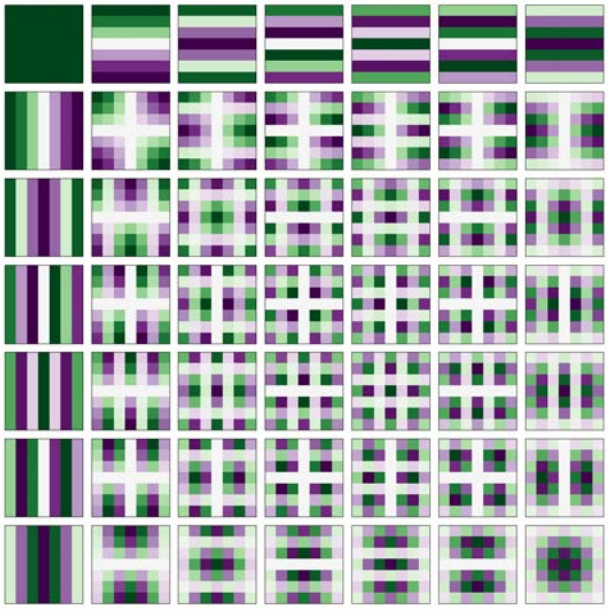


Figure 11: Each 7×7 feature map is transformed to 49 orthogonal output features, using the DCT coefficients shown above. Green > 0, Purple < 0, white $= 0$. The top left most output feature is equal to the mean pooling operation.

formation. The DCT coefficients are visualized in Fig. 11. As a final step, the ResNet performs a linear projection to the 1000 logits. The analogous operation for the INN is taking the distance of the output z to each of the 1000 cluster centers.

Low-rank μ_y. If each entry of μ_y is learned independently, the total number of parameters for D-dimensional latent space and M classes will be $DM \approx 150\,\text{Mil}$ for ImageNet. This is completely impractical, as μ_y alone would make up the majority of network parameters, which will only lead to overfitting. We solve this by dividing up μ_y into two parts, corresponding to the mean-pooled and the higher-order DCT variables: $\mu_y = [\mu_{\text{mean},y},\ \mu_{\text{rest},y}]$. We freely learn all approx. $3\,\text{Mil}$ parameters of $\mu_{\text{mean},y}$, and choose a low-rank representation for the remaining $\mu_{\text{rest},y}$, using K prototype vectors μ_k:

$$\mu_{\text{rest},y} = \sum_{k=1}^{K} \alpha_{yk} \mu_k \tag{15}$$

Both μ_k and α_{yk} are learned. This reduces the number of parameters to $D_{\text{mean}}M + K(D_{\text{rest}} + M)$. Choosing $K = 128$ empirically gives the best validation performance, and results in approx. $19\,\text{Mil}$ parameters, almost a factor of 10 less than the full DM. However, it is important to note that this is still much more than the fully connected layer

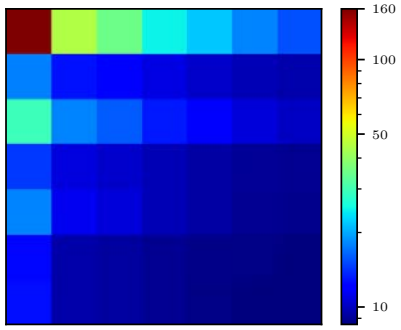


Figure 12: For the 49 DCT components images shown in Fig. 11, the mean spread of the corresponding entries of μ_y across classes is shown. Intuitively, this is how much each DCT component contributes to classification. A value of 0 means that these dimensions do not affect the classification at all. The mean pooled component has by far the largest influence, and the contribution of the high order components (bottom right) is negligible. Due to the random horizontal flip augmentation, the horizontally anti-symmetric components hardly contribute (alternating rows).

of a standard ResNet, with approx. $2\,\text{Mil}$ parameters. This indicates it might be possible to find an even more efficient representation of μ_y without sacrificing performance. The influence of each component of the low-rank μ_y is shown in Fig. 12. While $\mu_{\text{mean},y}$ contributes by far the most, training without $\mu_{\text{rest},y}$ entirely (setting it to zero), degrades the validation top-1 prediction performance by several percentage points.

Data augmentation and training. As data augmentation, we perform the usual random crops and horizontal flips, with two additions: Firstly, as is standard practice with normalizing flows specifically, we add uniform noise with amplitude $1/255$ to the images, to remove the quantization. This is necessary when training with the Jacobian, as the quantization otherwise leads to problems. Secondly, we use label smoothing [46] with $\alpha = 0.05$. This is necessary to prevent the mixture centroids from drifting further and further apart: training with perfectly hard labels makes the implicit assumption that all class components are infinitely separated.

The training scheme is the same as for the standard ResNet [21]: we use the SGD optimizer with a momentum of 0.9 and the weight decay set to 0.0001. We set the initial learning rate slightly lower to 0.07 compared to 0.1 for the original ResNet. We also perform two subsequent cooling steps whenever the loss plateaus, decreasing the learning rate by a factor of 10 each time. The batch size is 64 per

	ResNet	INN
Network parameters (M)	23.5	55.4
All parameters (M)	25.6	77.5
FLOPs (G)	4.07	9.08

Table 5: Number of parameters and computational cost for each model. *'Network parameters'* only counts the coupling/residual blocks. *'All parameters'* additionally includes the fully connected output layer of the ResNet, and the parametrization of μ_y for the INN. The (M) and (G) indicates Mega and Giga respectively. For FLOPs, the fused multiply-add instruction (FMA) is counted as a single FLOP, as it is commonly a single instruction in modern computing architectures.

GPU, training on 6 GPUs.

The constraint of invertibility is associated with an extra cost of parameters and computation cost compared to a purely feed-forward network. Table 5 summarizes this in comparison to a standard ResNet-50. Both in terms of network parameters, as well FLOPs needed for one forward pass of the network, the cost of the INN is about twice as high as the ResNet. We are optimistic this overhead can be reduced in the future with more efficient INN architectures.

B.2. Receptive Field

While the maximum possible receptive field (RF) of the INN and a standard trained ResNet are roughly comparable (see Table 1), we see large differences in the effective RF. For the effective RF, we pick a feature space column u, before the DCT pooling operation. Meaning, from the $H \times W \times 3072$ feature space, u will be the $1 \times 1 \times 3072$ column. We choose a column from the center to avoid interactions with the edges. We call the individual features u_l ($l = 1 \dots 3072$). We now measure the gradient w.r.t. each channel of each image pixel x_{ijk}, for real input images. The pixel position is ij, and the color channel is k. We define the 'sensitivity' of the model at each position as the L_1 norm of the gradient of the features w.r.t. that input position, averaged over images from the test set:

$$\text{Sensitivity}(i,j) = \mathbb{E}_{x \in \text{test}} \left[\sum_{k=1}^{3} \sum_{l=1}^{3072} \left| \frac{\partial u_l}{x_{ijk}} \right| \right] \qquad (16)$$

There are other definitions that would be equally sensible (squared gradients, frobenius norm, etc.), but the results always show the same behaviour.

The cross-sectional shape of this represents the effective RF, and is shown in Fig. 13. We observe that for low β, the effective RF is very narrow. In fact it is almost as narrow as it could possibly be: for $\beta \leq 4$, the FWHM of the sensitivity is only 64 pixels. This is the same we would get from only

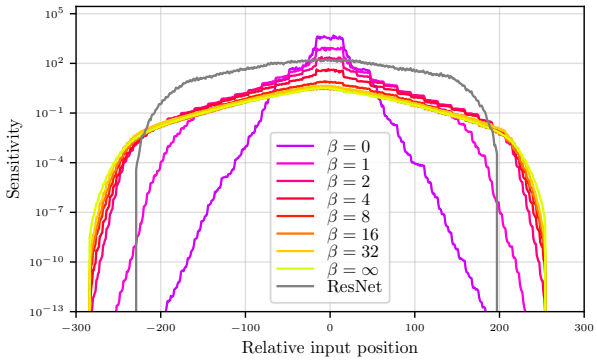


Figure 13: Effective receptive field for each value of β, just before the final pooling operation. Note the logarithmic sensitivity axis.

the downsampling steps, without any spatial convolutions (with 6 downsamplings, $2^6 = 64$). This could indicate that for the likelihood estimation, local details and structures are more important than any long-range features. For higher values of β, the response more closely matches that of a standard trained ResNet (1.25 times wider in line with the 1.25 times larger maximum possible RF).

B.3. Calibration Error

The calibration of a model measures the truthfulness of the predictive posteriors. In short, if we consider predictions where the model is e.g. 80% confident in a class, we would expect the prediction to be correct 80% of the time. If it were correct more often, it would be underconfident, and vice versa, more commonly, if it were correct in much fewer than 80% of cases, it would be overconfident. Plotting the fraction of correct predictions R over the binned confidence C of predictions gives the so-called calibration curve $R(C)$. For a perfectly calibrated model, the curve will follow the diagonal, but usually the behaviour deviates.

To quantitatively measure the deviations, we compute the expected- (ECE), the max- (MCE) and the overconfidence calibration error (OCE). More details on the computation of these measures can be found e.g. in Appendix D of [3]. The ECE measures the expected distance from the diagonal, weighted by the bin count $n(C)$ at any confidence:

$$\text{ECE} = \frac{1}{n_{\text{tot}}} \sum_C n(C) |C - R(C)| \qquad (17)$$

But for tasks with more than ~ 10 classes, the ECE is almost completely dominated by the 'negative' predictions: for any ImageNet prediction, typically only a few classes have a meaningful confidence, while e.g. 990 of the 1000 classes will have confidences $< 0.1\%$. So the lower end of the curve is weighted ~ 100 times stronger than the rest

	IB-INN, $\beta =$							RN
	1.0	2.0	4.0	8.0	16.0	32.0	∞	
ECE (%)	0.16	0.16	0.16	0.17	0.16	0.17	0.17	0.17
MCE (%)	5.54	3.13	5.47	4.57	5.50	5.28	5.10	7.72
OCE	3.87	4.13	4.31	4.73	4.15	4.94	5.12	6.75

Table 6: Calibration Errors for different values for β and for the ResNet. Expected Calibration Error (ECE), Max Calibration Error (MCE), Overconfidence Calibration Error (OCE) (see text for definitions).

of the curve, severely shifting the ECE statistic towards the very low confidence regime. The MCE measures the maximum distance from the diagonal:

$$\mathrm{MCE} = \max_C |C - R(C)| \tag{18}$$

The MCE is not affected by the same phenomenon as the ECE, but in return is subject to random fluctuations of sparsely populated regions on the curve; it only takes a single bin into account. Finally, the OCE measures the normalized fraction of wrong predictions that are highly confident with $C \geq C_{\mathrm{crit}}$, where we use $C_{\mathrm{crit}} = 99.7\%$.

$$\mathrm{OCE} = \frac{1}{1 - C_{\mathrm{crit}}} \sum_{C \geq C_{\mathrm{crit}}} |1 - R(C)| \tag{19}$$

For instance, an OCE of 3.5 would mean that in these high-confidence cases, the model is wrong 3.5 times more often than allowed, the error rate should be $\leq 1 - C_{\mathrm{crit}} = 0.3\%$ in these cases. This measures more directly the cases we may be interested in: we want to be able to trust the decisions if they are very confident. The OCE is less noisy than MCE in our case, as it takes more samples into account.

We report the result in Table 6, and show curves in Fig. 14. In short, we confirm previous observations e.g. in [3]: the GC models are better calibrated than DCs. The OCE shows the clearest trend of increasing overconfidence with β. Even from the $\beta = \infty$ model to the standard ResNet, there is a significant jump in the calibration error, also seen clearly in the full calibration curves. As the loss function for training at $\beta = \infty$ is essentially the same as a standard ResNet, this must be due to the construction of the model. Explained further in 4.2 ('Class similarities'), our conjecture is that it is due to the latent space structure specifically.

C. Explainability – Additional Materials

C.1. 2D Decision Space

In the following, we show another possibility to visualize the decision space for a smaller set of classes. In our case, we select 10 labels from all ImageNet classes. Starting from the full model, the μ_y of the selected classes are

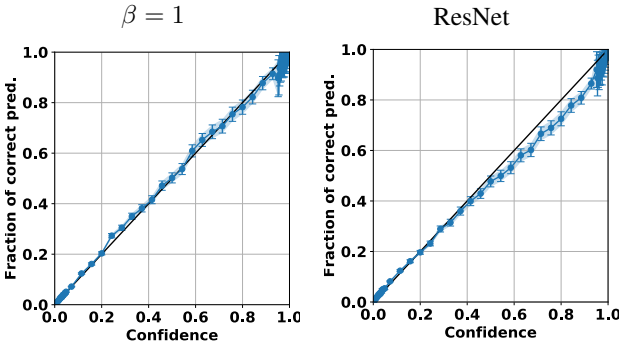


Figure 14: Calibration curves for the model with $\beta = 1$, and a standard ResNet-50, for reference. Deviations below diagonal = overconfidence, above = underconfidence. The error bars are the Poisson errors computed from the bin count.

constrained to a plane and fine tuned, reaching 90% accuracy for this simplified 10-class case. This allows us to show the entire decision space in a single 2D plot. The decision boundaries between all classes form a Voronoi tessellation of the decision space. All latent vectors inside the Voronoi cell of a certain class will have the highest probability under that class. In the case where the μ_y are not constrained to a plane and all 1000 classes are used, the behaviour is the same, with high-dimensional polygons for each class, but this can not be readily visualized.

C.2. Class Similarity Matrix

For the pairwise predictive uncertainty, we only consider two classes, $y \in \{1, 2\}$. We denote the distance of the class centers as $\Delta\mu = \|\mu_1 - \mu_2\|$. We assume $y = 1$ is the top prediction. This is just for simplification, as 1 and 2 can be swapped in the derivation if $y = 2$ is the top prediction. The prediction confidence c for any latent vector z is then between 0.5 and 1.0, computed as

$$c(z) = \frac{q(z \mid y{=}1)}{q(z \mid y{=}1) + q(z \mid y{=}2)} \tag{20}$$

The model's latent density is

$$q(Z) = \frac{1}{2}\mathcal{N}(\mu_1; 1) + \frac{1}{2}\mathcal{N}(\mu_2; 1) \tag{21}$$

This allows us to explicitly work out how the confidences will be distributed through the change-of-variables formula. Note that z can be expressed in cylindrical coordinates oriented along the line connecting μ_1 and μ_2. All the radial parts integrate out, only the position along this line is relevant. After some substitutions and simplifications, we ob-

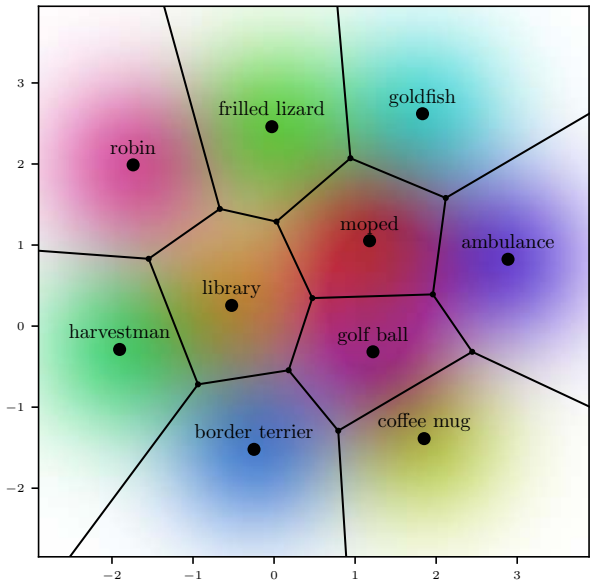


Figure 15: Latent space of a model with only ten classes, where the μ_y (black points) are constrained to a plane. The black lines are the decision boundaries, e.g. all points inside the 'moped'-polygon will be classified as a moped. The background is colored according to the probability density of each mixture component.

tain

$$p(c) = \frac{1}{A} \underbrace{\left(c - c^2\right)^{-3/2} \exp\left(-\frac{1}{2\Delta\mu^2} \log^2\left(\frac{1}{c} - 1\right)\right)}_{:=\rho(c)} \tag{22}$$

A is the normalization constant and has no closed form:

$$A = \int_{1/2}^{1} \rho(c) \mathrm{d}c \tag{23}$$

And we simply call the unnormalized density $\rho(c)$. Finally, the expected confidence $\overline{C}$ can be readily computed as

$$\overline{C} = \frac{\int c\rho(c)\mathrm{d}c}{\int \rho(c)\mathrm{d}c} \tag{24}$$

The expected *uncertainty* as opposed to the confidence is simply $1 - \overline{C}$.

C.3. Saliency Heatmaps

As outlined in the paper, to derive the saliency and posterior heatmaps, we start with the following definition:

$$\begin{aligned} w^{(y)} &= \mathrm{DCT}^{-1}(z) - \mathrm{DCT}^{-1}(\mu_y) \\ &= \mathrm{DCT}^{-1}\big(z - \mu_y\big). \end{aligned} \tag{25}$$

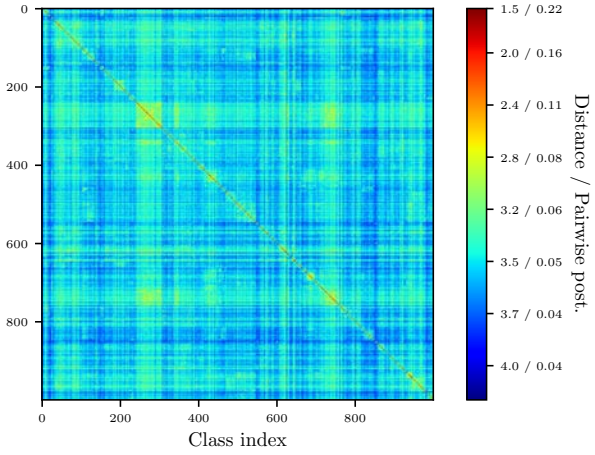


Figure 16: Similarity matrix between all 1000 classes. The two large clusters around class index 250 and 750 are dogs. The colormap indicates the pairwise distance of the μ_y as well as the expected pairwise posterior, meaning e.g. the binary decision between a tabby cat and a tiger cat is associated with 20% expected uncertainty, by construction (see text).

Because the DCT operation is linear and orthogonal, it conserves distances, and we can write

$$q(z|y) \propto \exp\left(-\frac{1}{2}\|z - \mu_y\|^2\right) = \exp\left(-\frac{1}{2}\|w^{(y)}\|^2\right) \tag{26}$$

We can consider the spatial structure present in $w^{(y)}$: It will have three indices, k, l for the spatial position and m for the feature channels: $w^{(y)}_{klm}$. We can simply factorize over the spatial dimensions. For the log-probability, we get

$$\log q(z|y) = \sum_{k,l} -\frac{1}{2}\|w^{(y)}_{kl,:}\|^2 + const. \tag{27}$$

$$:= \sum \log q(w_{kl}|y) \tag{28}$$

We will ignore the $const = -\dim(z)\log(2\pi)/2$ for convenience. This spatial decomposition of $\log p(z|y)$ allows us to make various heatmap visualziations in a principled way.

First, we consider $-\log p(w_{kl}|y)$. Considering the sum over pixels, this looks like a point-wise entropy. The common interpretation from information theory is, that this is a measure how much information is contained in each part of the image. The values in each pixel sum to $-\log p(z|y)$, which is then the overall entropy of the latent vector for this image. To remove the class dependence, we plot the 'saliency heatmap':

$$Q_{\mathrm{Saliency}}(k,l) = -\log\left(\sum_y q(w_{kl}|y)p(y)\right) \tag{29}$$

Some examples for this are shown in Fig. 17.

C.4. Posterior Heatmaps

We can now consider the class prediction:

$$q(y|x) = \frac{q(z|y)}{\sum_{y'} q(z|y')} =: \frac{q(z|z)}{S(z)}, \tag{30}$$

where $p(y) = 1/M$ and the Jacobian $|\det J|$ both cancel out. We therefore plot for any class the following 'class posterior heatmap':

$$Q_{\text{Class}}(k,l,y) = \log p(w_{kl}|y) - S_{kl} \quad \text{s.t.} \quad \sum_{kl} S_{kl} = S \tag{31}$$

The $-S_{kl}$ term means a fixed 'image' is subtracted from each heatmap, representing the denominator, which is constant for all classes. There is some freedom to choose S_{kl}, as long as it sums to S. When distributing it evenly over space, the differences in the heatmaps between classes are hard to see by eye, compared to the common differences within the heatmaps shared across classes, which are larger by magnitude. Heuristically, we instead find the best contrast when we choose the relative weight of each S_{kl} in the following way:

$$S_{kl} = S\,\frac{r_{kl} + 0.03}{\sum_{kl}(r_{kl} + 0.03)} \tag{32}$$

where r_{kl} is the same as $\log p(w_{kl})$ but normalized to the $[0,1]$-range over each image. Additional examples are shown in Fig. 17.

Comparing to Eq. 30, we see that summing Q_{Class} over feature-space pixels gives exactly the log-prediction $\log q_\theta(y|x)$. So Q_{Class} represents a spatial decomposition of the actual predictive output:

$$q(y|x) = \exp\left(\sum_{kl} Q_{\text{Class}}(k,l,y)\right) \tag{33}$$

D. Robustness – Additional Materials

D.1. Corrupted Images Examples

Examples of the different corruptions and the severity levels are shown in Fig. 18.

D.2. Adversarial Attack Objectives

As explained in the paper, we performed the well established 'Carlini-Wagner' white-box targeted attack method introduced in [8]. Here, the attacked image is parametrized as $x_{\text{adv}} = \frac{1}{2}(\tanh(w) + 1)$, to ensure the image values are between 0 and 1. The attack then consists of optimizing w directly to minimize the following objective:

$$\mathcal{L}_{\text{CW}}(w,x) = \|x_{\text{adv}}(w) - x\|^2 + c\max\Big(\max(\{l_y : y \neq t\}) - l_t, -\kappa\Big) \tag{34}$$

The original image is x, and the logits output by the model for each class y are l_y. The target class, that the attacked image is supposed to be classified as, is $t := y_{\text{target}}$. The logits are recomputed by the model on each iteration using the updated $x_{\text{adv}}(w)$, which they depend on: $l_y = l_y(x_{\text{adv}}(w))$. The gradients are propagated through the model. We call the max-term $\mathcal{L}^{(\kappa)}_{\text{class}}(y_{\text{target}})$ in the paper.

In other words, the attack objective simultaneously attempts to make x_{adv} and x the same, and to maximize the difference between the logit of the target class, and the currently next highest predicted class. Once the distance is larger than the hyperparameter κ in favour of the target class, this loss term does not contribute anymore. Adjusting κ therefore has a direct influence on the confidence of the (wrong) predictive posterior. From an attackers point of view it is optimal to fool a classifier to make certain but wrong predictions by setting a high value for κ, while finding a w so that x_{adv} is as close as possible to the original image x. Ideally the differences between x_{adv} and x remain imperceptible to the human eye. From the victim networks point of view, the targeted wrong prediction should be as uncertain as possible, and the difference between x_{adv} and x as large as possible.

For GCs, there are not logits per se. Instead, we use the conditional log-likelihoods $l_y = \log p(x|y)$, to get the same behaviour. We performed all adversarial attacks on the same randomly chosen 200 test images, paired with the fixed random target class each. To perform the attack, we use the Adam optimizer with its initial learning rate set to 0.01, as in [8]. We performed the attacks with three different values for κ: 0.01, 1.0, ∞. The parameter c was fixed and set to 10, which is the lowest possible value for achieving a 100% attack success rate on all our tested models. We assume the attack converged whenever $\mathcal{L}_{CW}$ stops improving for 20 consecutive gradient steps.

As illustrated previously in [7], any adversarial attack defense- or detection mechanism can itself become target of a modified attack, fooling the classification and the detection at the same time.

In line with this work, we construct a modified attack loss to achieve fooling the two-tailed quantile test we utilized for detecting attacks. As stated in the main part of this

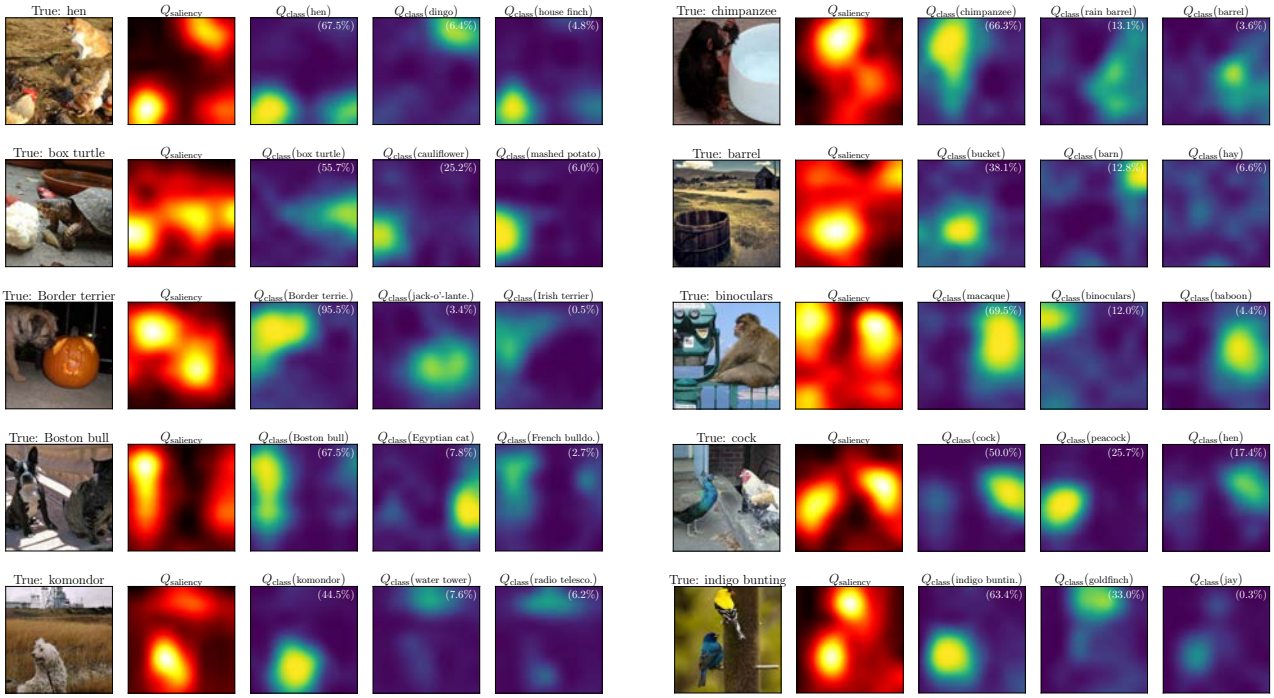


Figure 17: Additional examples for saliency maps and posterior heatmaps for the top three classes. The white inset numbers indicate the confidence in that class, which is equal to the exponential of the sum over the posterior heatmap (see text).

work we denote it as $\mathcal{L}_{CWD}(w, x)$:

$$\mathcal{L}_{\text{CWD}}(w,x) = \mathcal{L}_{\text{CW}}(w,x) + d \cdot \underbrace{\Big(\underset{x' \sim X_{\text{train}}}{\text{median}} \big(\log q(x') \big) - \log q(x_{\text{adv}}(w)) \Big)^2}_{\mathcal{L}_{\text{detect}}} \tag{35}$$

In the added $\mathcal{L}_{\text{detect}}$ term, $\underset{x' \sim X_{\text{train}}}{\text{median}} \big(\log q(x') \big)$ stands for the median estimated probability density (PD) of the training set and $\log q(x_{\text{adv}}(w))$ for the estimated PD of the perturbed image. Intuitively, we are now forcing x_{adv} to move to the center of the distribution of PD values of the training data. If it reaches the median exactly, the ROC-AUC detection score will be 0% (also see Sec. A.1).

D.3. Adversarial Trajectories

We find that the attack consists of two distinct stages. First, the attack attempts to cross into the area belonging to the target class, leaving a certain margin specified implicitly by κ. Second, the attack minimizes the magnitude of the adversarial perturbation, while staying inside this region (sometimes stepping outside the region for a single iteration). We can visualize the attack's trajectory and its effect on the decicison explicitly, using the 2D decision model from Sec. C.1, see Fig. 19. We also perform the same visualization for the full model with 1000 classes, shown in Fig. 20. We observe the same behaviour, although the decision boundaries can no longer be visualized. For the 2D figure, we consciously chose a target class located at the 'edge' of the latent space, not circled by other classes on all sides. This is because for the 1000 class case in higher dimensions, all classes are essentially guaranteed to be such 'edge' classes.

An important lesson to take from this is that the area of maximal confidence of the attack is not necessarily closest to μ_{target}. Instead, the confidence depends on the *difference* of the squared distance to the other classes (see Eq. 34). Especially for high κ, sufficient confidence is only achieved far outside of the original distribution, which is what leads to the almost perfect detection score for $\kappa = \infty$ reported in Sec. D.4, Table 9 under the OoD column. This result is also also visually illustrated in Fig. 7 (second column, third row) in the main paper.

D.4. Adversarial Attacks – Full Results

All results concerning adversarial attacks are summarized in Fig. 23, and Table 9, corresponding to Fig. 7 in the main paper.

In our evaluation, we observe the GCs to be measurably more robust compared to the DC in terms of neces-

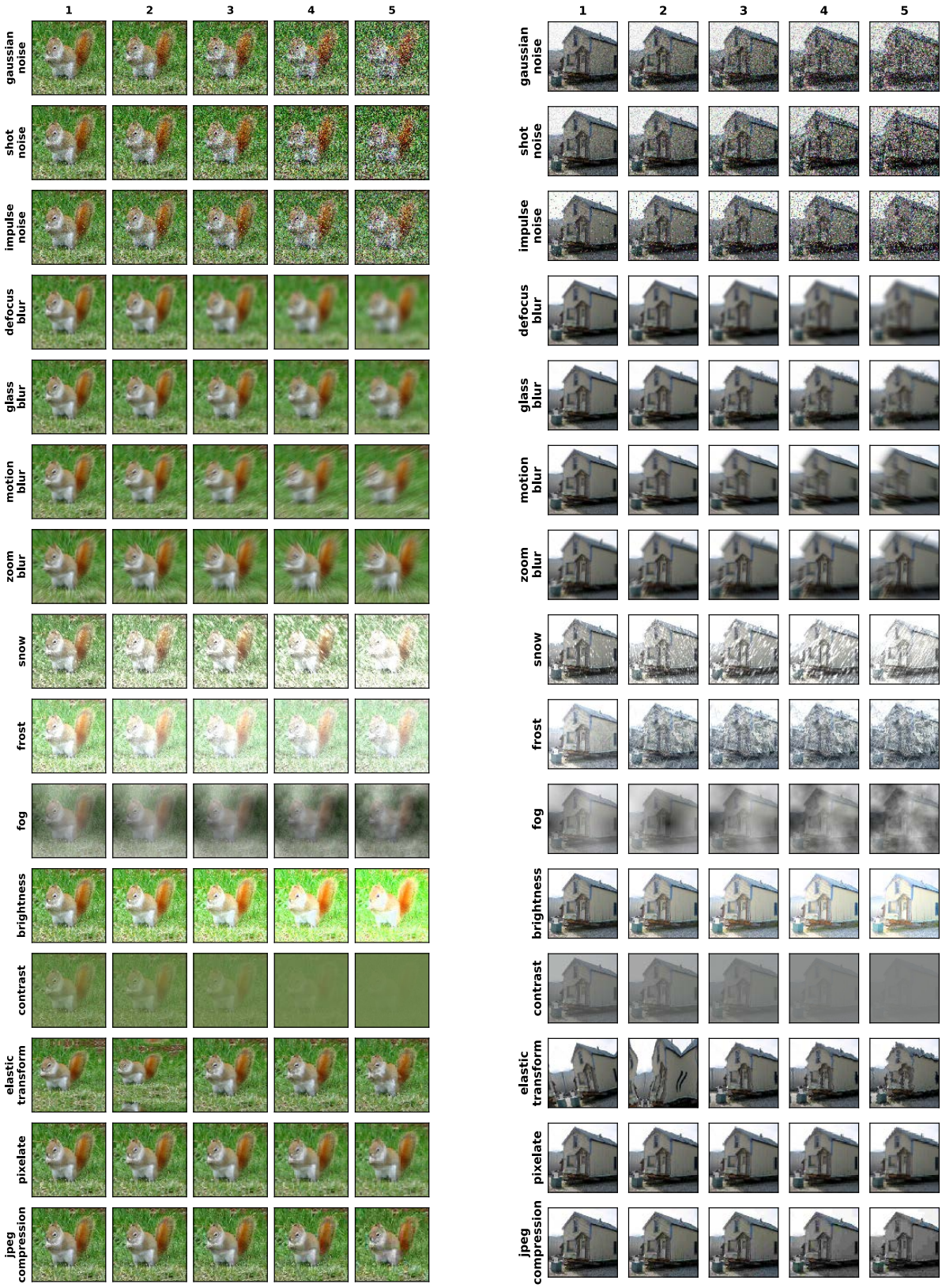
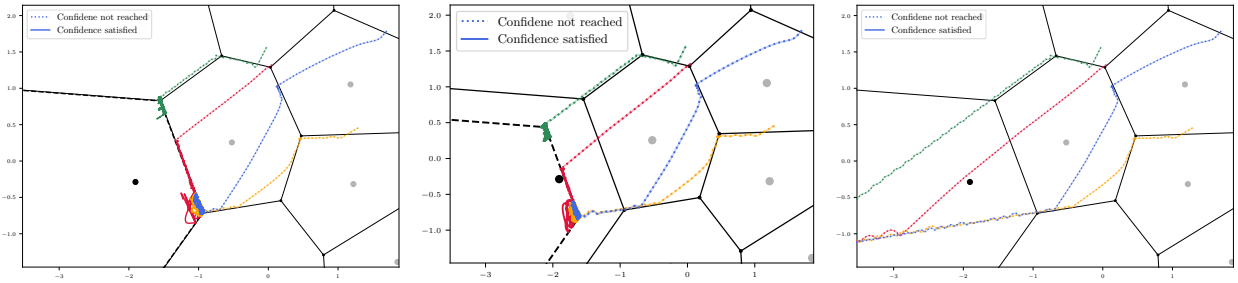


Figure 18: We show the different corruption types (rows) with their severity levels from 1-5 (columns) applied to two sample images (Samples belong to the ImageNet classes 'fox squirrel' and 'mobile home').



$\kappa = 0.01$ $\kappa = 1.0$ $\kappa = \infty$

Figure 19: Additional examples for Fig. 6 with different settings for κ.

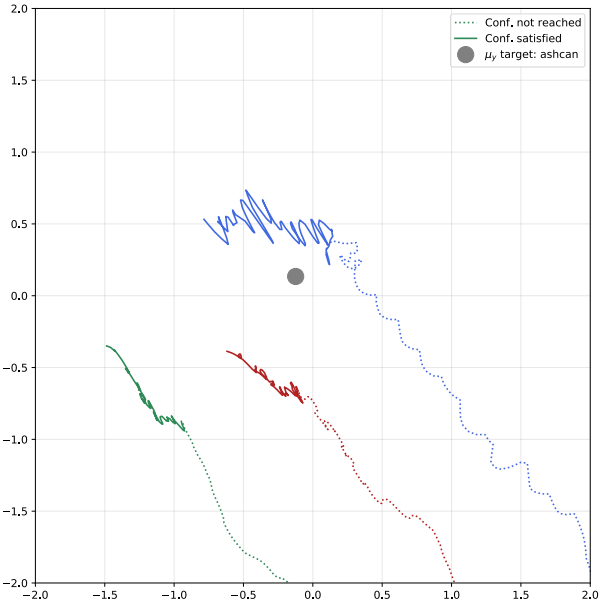


Figure 20: Visualization of the adversarial trajectories for the full model, $\kappa = 1$. The trajectory is projected to 2D by fitting a plane through the five classes that the trajectory passes closest to.

sary adversarial perturbation in order to successfully fool the model. For achieving a successful attack, the adversarial noise generally needs to be amplified for models with better generative modeling capabilities (smaller values for β), as is the case for higher values for κ (forcing highly confident but wrong predictions). We would expect the trend to continue for $\beta < 1$, for the adversarial perturbations to be even larger, but at that point the task performance may not be satisfactory anymore. We show this qualitatively in Fig. 21. The gap to the ResNet (roughly factor 2) is consistent to what was observed for a simplified version of CIFAR10 in [33]. In terms of κ, the adversarial perturbations increase a lot for $\kappa = \infty$ (forcing confident fooled predictions), but the increase is homogeneous across models including the ResNet. Furthermore, as can be seen in Figure 30, optimizing for highest possible confidence results in adversarial noise that is clearly visible to the human eye. For $\kappa = 0.01$ and $\kappa = 1$ on the other hand, the applied noise is a lot harder to perceive by humans (See Fig. 24 and Fig. 27). We make a second important observation: For most models, the predictive confidence is similar to the ResNet. However, $\beta = 1$ and $\beta = 2$ are 100% confident in their (wrong) prediction, even for low values of κ. During the attack, this occurs while the fooling part of the loss is already satisfied and has no effect. The phenomenon is purely due to the attack reducing the amplitude of the perturbation. Evidently, by reducing the attack amplitude, the image moves into an even more confident region of latent space. So in the sense of predictive uncertainty on adversarial examples, GCs actually seem to be more vulnerable to adversarial attacks.

Regarding the adversarial perturbation needed in order to successfully fool the model while simultaneously trying to fool the attack detection mechanism we make three main observations: the adversarial noise is increased when putting a higher focus on fooling the attack detection mechanism. This can also be clearly seen in Fig. 22 and in the quantitative comparison in Fig. 23, first column when comparing the three bars per β. Second, as shown in the second column of Fig. 23, we observe the attack detection capabilities generally to decrease. For the good detection models such as $\beta = 1$, the score stays reasonably high, while the weaker models have a detection score significantly worse than random. Lastly, the predictive uncertainty is not affected by the detection attack at all (Fig. 23, third column).

Inspecting the perturbed images also provides some clues as to how the attack fools the detection mechanism: They show uniformly decreased contrast. As shown in [34], such low-contrast images have unnaturally high estimated likelihoods. In our case, this seems to compensate for the lower estimated likelihood caused by the noise-like adversarial perturbations, to make the image appear 'typical' overall.

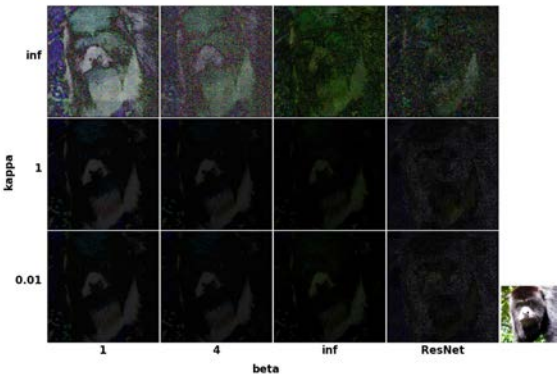


Figure 21: Qualitative results demonstrating the influence of κ (controlling the classifiers final confidence on targeted classes) and β (controlling the generative modeling capability of the classifier) on adversarial attack robustness. The discriminative classifier ResNet is added for reference. The figure, showing the per-pixel errors in RGB space, gives the absolute difference between the original (bottom right corner) and the adversarially perturbed image, amplified by a scaling factor for visibility. For adversarial attacks to achieve highly confident posteriors (high value for κ) the noise has to be amplified. In order to successfully trick a classifier with better generative modeling capabilities (low value for beta) the noise added by the attack has to be even larger.

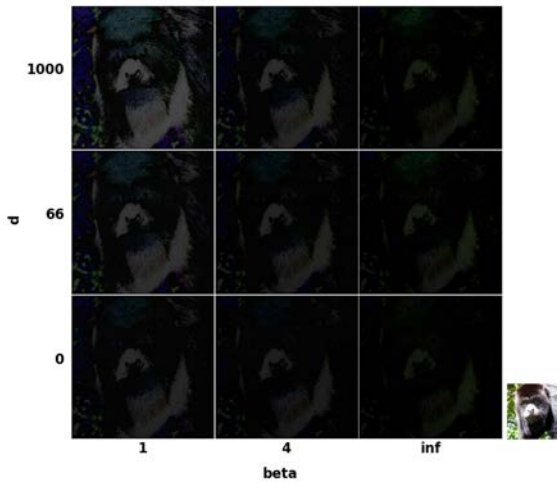


Figure 22: Qualitative results demonstrating the influence of d (controlling the strength put on fooling the attack detection mechanism) and β on adversarial attack robustness for κ fixed to 1. The more weight is put on fooling the attack detection mechanism (higher values for d), the more noise must be added to the input image by the adversarial attack. In order to fool the generally stronger detection mechanism of classifiers with higher generative modeling capabilities, the noise must be even higher.

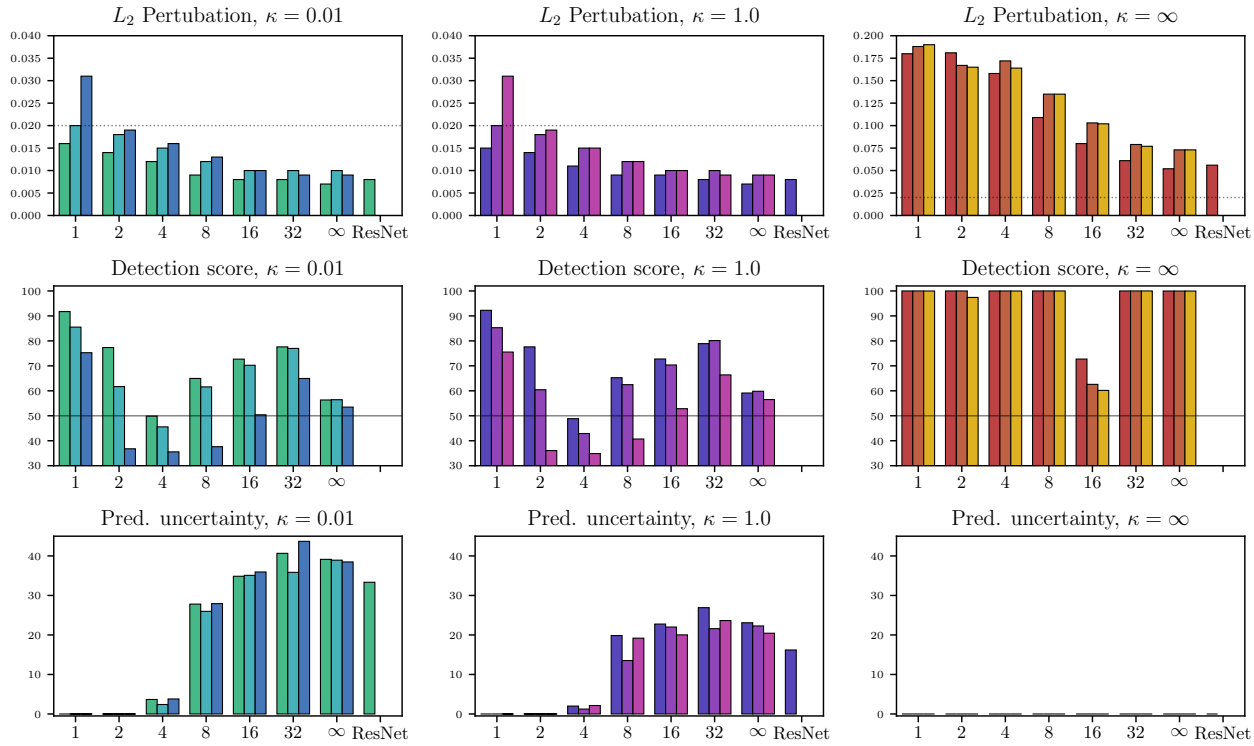


Figure 23: Behaviour of GCs under adversarial attacks. The three rows of plots give the mean perturbation, detection score, and uncertainty of the wrong prediction ($1 - \text{confidence}$). The three columns of plots correspond to adversarial attacks with $\kappa = 0.01$ (targeted prediction with any confidence is enough), $\kappa = 1$ (targeted prediction should have high confidence), and $\kappa = \infty$ (targeted prediction should be as confident as possible). The three bars for each β correspond to: standard adversarial attack ($d = 0$), as well as $d = 66$ and $d = 1000$, i.e. the detection mechanism is fooled at the same time as the prediction. The dotted line in the top row roughly indicates the level at which attacks are clearly visible by eye. Note that this is subjective and only a rough indication. The line in the second row indicates random performance, i.e. the OoD detection does nothing useful.

β	Confidence			Corruption			Success			OoD 1t-tt p(x)	OoD p(x)			Entropy X	Entropy $X_{Corr.}$			p(x) Val
	d=0	d=66	d=1000	d=0	d=66	d=1000	d=0	d=66	d=1000	d=0	d=0	d=66	d=1000		d=0	d=66	d=1000	
0	89.94	74.5	93.99	0.033	0.057	0.123	90	74	93	99.91	99.77	99.15	98.79	1.99	0	0	0	48.67
1	100	99.99	99.99	0.016	0.020	0.031	100	100	100	93.86	91.74	85.52	75.26	1.92	0	0	0	47.99
2	99.99	99.99	99.99	0.014	0.018	0.019	100	100	100	79.68	77.34	61.69	36.78	1.63	0	0	0	48.39
4	96.33	97.6	96.2	0.012	0.015	0.016	100	100	100	48.17	49.88	45.56	35.55	1.46	0.23	0.17	0.26	49.13
8	72.19	74.02	72.07	0.009	0.012	0.013	100	100	99	63.83	64.96	61.57	37.62	1.36	1.64	1.55	1.73	51.35
16	65.14	64.91	64.06	0.008	0.010	0.010	100	99	100	70.78	72.69	70.23	50.40	1.30	2.04	2.07	2.22	51.82
32	59.34	64.15	56.3	0.008	0.010	0.009	100	98	98	76.24	77.60	76.98	64.94	1.28	2.39	2.07	2.58	53.23
infinity	60.86	61.07	61.52	0.007	0.010	0.009	100	98	98	56.41	56.33	56.44	53.48	1.09	2.23	2.17	2.20	52.12
RN	66.66	-	-	0.008	-	-	100	-	-	-				1.02	1.84	-	-	-

Table 7: $\kappa = 0.01$

β	Confidence			Corruption			Success			OoD 1t-tt p(x)	OoD p(x)			Entropy X	Entropy $X_{Corr.}$			p(x) Val
	d=0	d=66	d=1000	d=0	d=66	d=1000	d=0	d=66	d=1000	d=0	d=0	d=66	d=1000		d=0	d=66	d=1000	
0	91.50	77.5	94.5	0.032	0.056	0.122	91	77	94	99.50	99.38	99.35	98.89	1.99	0	0	0	48.67
1	100	100	99.99	0.015	0.020	0.031	100	100	100	94.26	92.25	85.28	75.53	1.92	0	0	0	47.99
2	99.99	99.99	99.99	0.014	0.018	0.019	100	100	100	80.22	77.60	60.43	36.07	1.63	0	0	0	48.39
4	98.0	98.76	97.86	0.011	0.015	0.015	100	100	100	48.98	48.85	42.89	34.87	1.46	0.11	0.09	0.15	49.13
8	80.15	86.47	80.81	0.009	0.012	0.012	100	100	100	63.98	65.25	62.47	40.70	1.36	1.26	0.86	1.26	51.35
16	77.24	77.99	79.98	0.009	0.010	0.010	100	100	100	70.99	72.75	70.35	52.84	1.30	1.46	1.37	1.30	51.82
32	73.09	78.41	76.35	0.008	0.010	0.009	100	100	100	77.79	78.90	80.13	66.38	1.28	1.68	1.34	1.54	53.23
infinity	76.91	77.72	79.56	0.007	0.009	0.009	100	100	100	59.09	59.12	59.82	56.50	1.09	1.44	1.38	1.28	52.12
RN	83.8	-	-	0.08	-	-	100	-	-	-				1.00	1.15	-	-	-

Table 8: $\kappa = 1$

β	Confidence			Corruption			Success			OoD 1t-tt p(x)	OoD p(x)			Entropy X	Entropy $X_{Corr.}$			p(x) Val
	d=0	d=66	d=1000	d=0	d=66	d=1000	d=0	d=66	d=1000	d=0	d=0	d=66	d=1000		d=0	d=66	d=1000	
0	100	100	100	0.231	0.237	0.237	100	100	100	100	100	100	100	1.99	0	0	0	48.67
1	100	100	100	0.180	0.188	0.190	100	100	100	100	100	100	99.98	1.92	0	0	0	47.99
2	100	100	100	0.181	0.167	0.165	100	100	100	100	100	100	97.41	1.63	0	0	0	48.39
4	100	100	100	0.158	0.172	0.164	100	100	100	100	100	100	99.98	1.46	0	0	0	49.13
8	100	100	100	0.109	0.135	0.135	100	100	100	100	100	100	100	1.36	0	0	0	51.35
16	100	100	100	0.080	0.103	0.102	100	100	100	73.05	72.72	62.59	60.17	1.30	0	0	0	51.82
32	100	100	100	0.061	0.079	0.077	100	100	100	99.97	100	100	100	1.28	0	0	0	53.23
infinity	100	100	100	0.052	0.073	0.073	100	100	100	99.98	99.98	99.97	99.97	1.09	0	0	0	52.12
RN	100	-	-	0.056	-	-	100	-	-	-				1.04	0	-	-	-

Table 9: $\kappa = \infty$

Table 10: Table 7,8 and 9 show the quantitative results of our adversarial attack experiments. Each table was obtained by performing the attack with a different value for $\kappa \in 0.01, 1, \inf$. A high value for κ aims a more certain posterior for targeted classes. The cell background colors green, orange and red stand for different values for d to ease the comparison across models and tables for a human reader. The variable d quantifies the strength on fooling the intrinsic attack detection mechanism of our learned classifiers. Note, that the ResNet does not model the data likelihood and therefore has not this capability. We report the maximum class probability (Confidence), the pixel-wise l^2-distance between the original and the adversarially perturbed image averaged over all pixels (Corruption), the success rate of the attack (Success), the one (1t-tt) and two-tailed ($p(x)$) typicality test OoD detection scores, as well as the posterior predictive uncertainty for the original (X) and the corrupted validation data $x_{corr.}$. Furthermore, we report the likelihood of the original validation data ($p(X)$ Val).

Figure 24: $\kappa = 0.01$, $d = 0$

Figure 25: $\kappa = 0.01$, $d = 66$

Figure 26: $\kappa = 0.01$, $d = 1000$

Figure 27: $\kappa = 1$, $d = 0$

Figure 28: $\kappa = 1$, $d = 66$

Figure 29: $\kappa = 1$, $d = 1000$

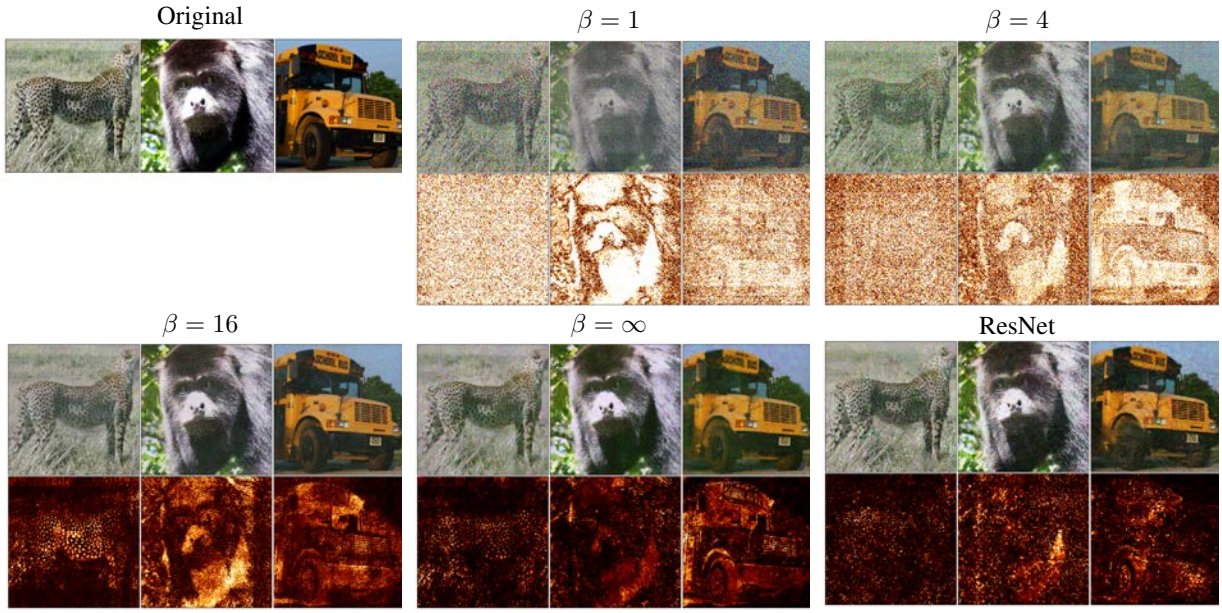


Figure 30: $\kappa = \infty$, $d = 0$

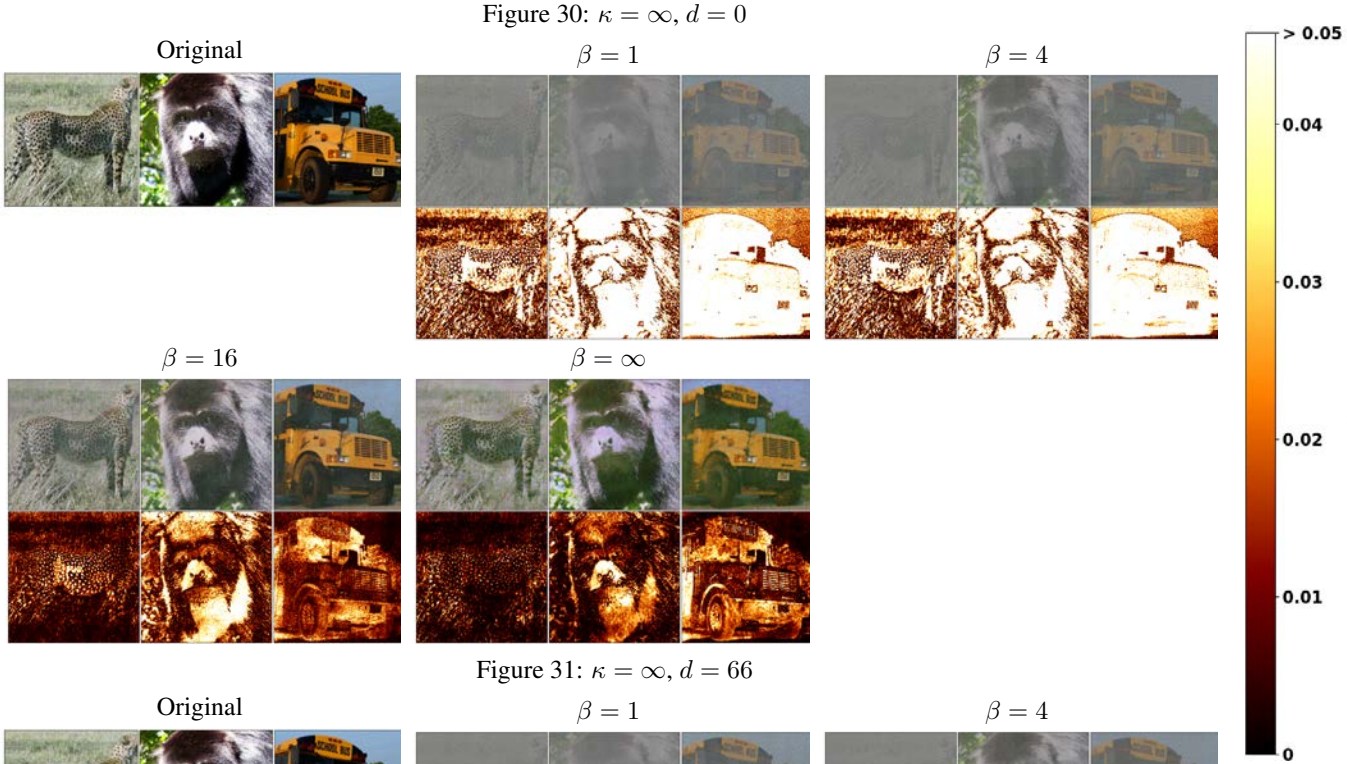


Figure 31: $\kappa = \infty$, $d = 66$

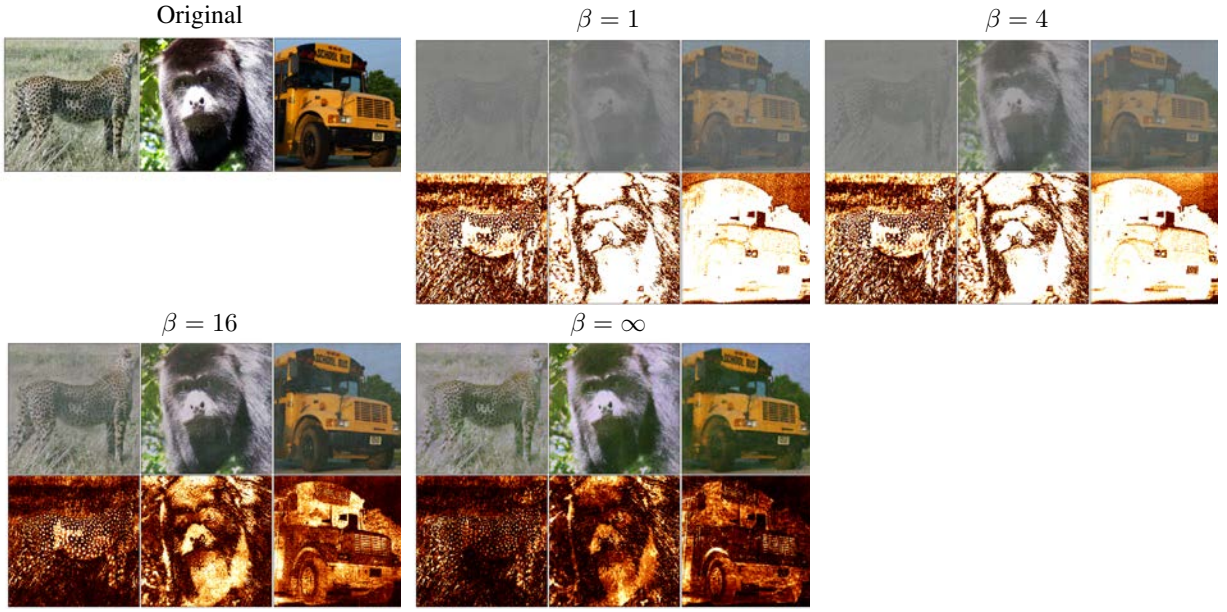


Figure 32: $\kappa = \infty$, $d = 1000$