A. Mathematical Perspective: Support Functions and Imaginative Polytopes

A.1. Preliminaries

A *polytope* $P \subseteq \mathbb{R}^d$ is the convex hull of finitely many vertices in $\mathbb{R}^d$, or by the Minkowski–Weyl Theorem any finite intersection of finitely many closed halfspaces [App10]. For a polytope $P \subseteq \mathbb{R}^d$, the *support function* $f_P : \mathbb{R}^d \to \mathbb{R}$ is

$$f_P(x) \coloneqq \max_{p \in P} \langle p, x \rangle_{L^2}. \tag{12}$$

We call the set of points in P that cannot be represented as a nontrivial convex combination of two distinct points in P the vertices of P and denote them by $\mathrm{vert}(P)$. It is easy to see that

$$f_P(x) \coloneqq \max_{p \in \mathrm{vert}(P)} \langle p, x \rangle_{L^2}. \tag{13}$$

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *piecewise linear* (PWL) if there exists a finite polyhedral partition of $\mathbb{R}^n$ such that f is affine on each piece. It is *continuous piecewise linear* (CPWL) if it is also continuous. Note that a function is CPWL exactly if it can be represented by a ReLU network [App7].

Given the representation Eq. (13) it is easy to see that the vertices of P correspond one by one to all attainable gradients of f_P. Further, we observe from Eq. (13) that support functions are convex, continuous, positively homogeneous ($f_P(\lambda c) = \lambda f_P(c)$ for $\lambda \geq 0$), and CPWL. Indeed, the mapping

$$\begin{aligned} \Phi : \ \mathsf{Polytopes}_d &\longrightarrow \mathsf{CPWL}_d^+, \\ P &\longmapsto f_P, \end{aligned} \tag{14}$$

is a bijection between d-dimensional polytopes and convex, positively homogeneous CPWL functions.

Tropical geometry refers to P as the *Newton polytope* of f_P; the correspondence and its learning-theoretic consequences have been exploited in analyses of neural networks and their expressivity [App9], [App5], [App4].

We call a function $f : \mathbb{R}^n \to \mathbb{R}$ *monotone* in case for any $x_1, x_2 \in \mathbb{R}^d$ with $x_1 \leq x_2$ (understood elementwise) one has $f(x_1) \leq f(x_2)$.

As shown in Appendix F.1, the split-streams g and h defined in Sec. 3 are positively homogeneous monotone convex CPWL functions and therefore possess corresponding Newton polytopes P_g and P_h by the bijection in Eq. (14). It is easy to see that by the monotonicity of g and h all points in P_g and P_h have non-negative coordinates as the gradients of g and h are non-negative.

Theorem 2. *Let $g : \mathbb{R}^d \longrightarrow \mathbb{R}$ be a monotone and CPWL function. Then $\nabla g \geq 0$,* i.e. *all its attainable gradients are non-negative.*

Proof of Theorem 2. Assume for a contradiction that there is $x \in \mathbb{R}^d$ and $j \in [d]$ such that the jth component of the gradient at x is negative, *i.e.* $\left(\frac{\partial g}{\partial x}\right)_j < 0$. As g is piecewise linear, there is $\epsilon > 0$ such that g is linear in the ϵ-ball around x. However for $\tilde{x} \coloneqq x - \frac{\epsilon}{2} e_j$ one finds $\tilde{x} \leq x$ but $f(\tilde{x}) > f(x)$. This contradiction concludes the proof of Theorem 2. □

A.2. Imaginative Polytopes

In the ablation study Appendix C we observe that the gradients of the split-pair show greater robustness than the gradients of the original model, *i.e.* small input perturbation results in less perturbed gradients. We want to argue that indeed the non-negativity of the split-stream gradient could be one important reason for this behavior. Indeed, as the split-stream outputs are just the maximum scalar products of the input with all attainable gradients respectively, see Eq. (13), the non-negativity of the gradients results in more stable gradients due to no cancellation in the scalar product. Further, any jump of the gradients in the original network does not force both gradients of g and h to jump, but only one of them.

Secondly, we observe dramatically better localization properties of the split-pair gradients compared to the original gradients. Again, we find it intuitive that positive gradients show better focus on salient image regions as no cancellation can happen.

Because the vertices of P_g and P_h empirically align closely with salient image structures we term them *Imaginative Polytopes*: intrinsic geometric objects induced by ReLU networks that summarize the model's learned visual features and thereby help to explain how common vision models envision and "imagine" the world. Their vertices correspond to distinct activation patterns and represent the fundamental "prototypes" that the network has learned to distinguish different input features. We visualize

the g-stream gradient of VGG16 with respect to the "dragonfly" class output logit for four ImageNet test images in Fig. 5, where we applied backward pass shifting with $\alpha = 0.25$. We remark that the h-stream gradient looks very similar, as it only differs to the g-stream gradient in the gradient of the original VGG model – for this reason we linked the g-stream gradient visualizations also with corresponding h-stream gradients.

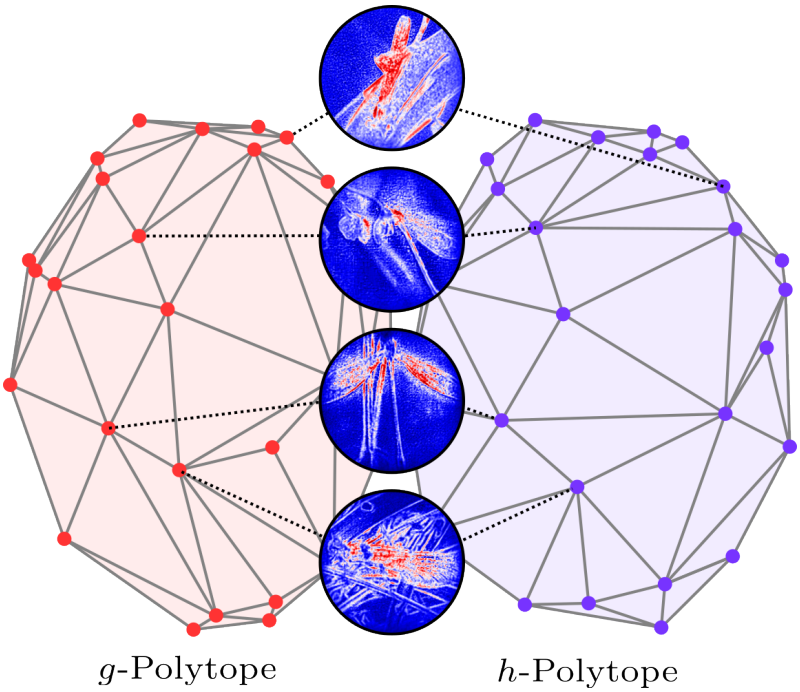

Figure 5. Concept visualization of the imaginative polytopes corresponding to VGG's dragonfly class output neuron. The polytope's vertices represent the g-stream and h-stream gradients respectively, revealing learned visual prototypes.

B. Saliency Analysis

This appendix provides comprehensive evaluation results for all XAI methods tested in the experimental setup described in Sec. 5.2.

B.1. VGG16 Qualitative Saliency Map Comparison

Complementing Fig. 4, we give more saliency examples in Fig. 6. Again, we observe more focus on the whole classified objects by SplitCAM in layer 26 compared to LayerCAM in the same layer. SplitGrad in layer 0 detects too many edges of the original image to be a useful explanation of the model's decision. Especially in images with many image details in the background, *e.g*. the rubble in the wood rabbit image, it fails to focus on the important image parts. This aligns with the observed performance peaks of SplitGrad and SplitCAM in deeper layers, presented in Fig. 7 in the layer ablation studies.

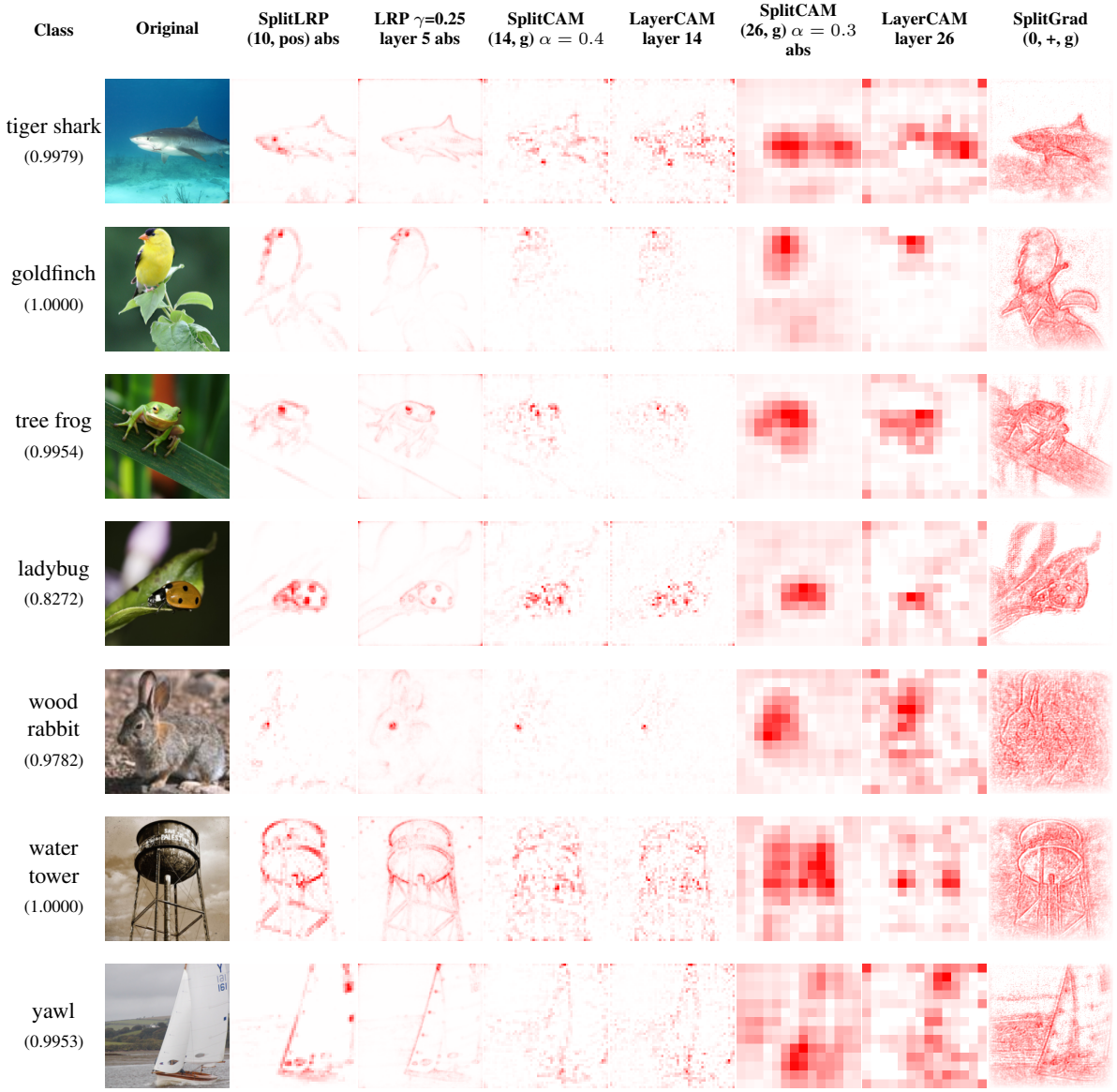

Figure 6. Saliency map comparison for VGG16 on images from the custom ImageNet-S test subset. Each row shows the predicted class (with confidence), original image, and saliency maps from different methods. Methods include SplitLRP, classical LRP, SplitCAM variants, LayerCAM, and SplitGrad.

B.2. Quantitative Analysis on VGG16

Table 4 presents detailed quantitative metrics analysis across all split configurations, baselines, and classical methods for VGG16. We observe a slight boost in performance between SplitCAM with $\alpha = 0.4$ compared to $\alpha = 0.5$ and compared to the classical LayerCAM respectively, indicating that the LayerCAM formulas indeed yield more powerful explanations on the split-pair compared to the original model. This can be seen even more drastically in ablation studies Appendix C.

Table 4. Comprehensive XAI Evaluation Results (VGG16). The abbreviations are similar to Tab. 1. For each method configuration we highlight the best metrics values across the considered layer configurations in bold.

Method	Layer	Abs	Select. ↓	Attr. Loc. ↑	Point. ↑	Pixel Fl. AUC@5 ↑	Pixel Fl. AUC@20 ↑	Max Sens. ↓
SplitCAM Methods								
SplitCAM (sc, α 0.4)	(12, g)	n	4.720	**0.651**	**0.912**	0.213	1.794	**0.389**
SplitCAM (sc, α 0.4)	(14, g)	n	**2.727**	0.563	0.836	**0.300**	**2.404**	1.012
SplitCAM (sc, α 0.4, wta)	(14, g)	n	**2.761**	0.543	0.827	**0.303**	**2.414**	1.056
SplitCAM (sc, α 0.4, wta)	(26, +, g)	y	4.711	**0.635**	**0.938**	0.165	1.590	**0.456**
SplitCAM (sc, α 0.5, wta)	(14, g)	n	**2.766**	0.536	0.823	**0.303**	**2.416**	1.056
SplitCAM (sc, α 0.5, wta)	(26, +, g)	n	4.608	**0.607**	**0.903**	0.185	1.684	**0.566**
SplitGrad Methods								
SplitGrad (α 0.4)	(12, +, g)	n	5.633	**0.613**	**0.873**	0.148	1.461	**0.364**
SplitGrad (α 0.4)	(14, +, g)	y	**3.429**	0.533	0.820	**0.235**	**2.119**	0.851
SplitGrad (α 0.4, wta)	(12, +, g)	n	4.735	**0.635**	**0.883**	0.186	1.756	0.651
SplitGrad (α 0.4, wta)	(14, +, g)	y	**3.287**	0.540	0.818	**0.249**	**2.186**	0.857
SplitGrad (α 0.5, wta)	(12, +, g)	y	**3.379**	**0.551**	0.859	**0.261**	**2.220**	0.869
SplitGrad (α 0.5, wta)	(26, +, g)	y	4.948	0.543	**0.899**	0.162	1.527	**0.617**
SplitLRP Methods								
SplitLRP (sc)	(7, pos)	n	**3.062**	0.092	**0.857**	**0.296**	**2.290**	0.865
SplitLRP (sc)	(19, pos)	n	4.393	**0.600**	0.823	0.166	1.656	**0.291**
SplitLRP (sc, wta)	(7, pos)	n	**3.235**	-0.036	0.836	**0.296**	**2.291**	0.866
SplitLRP (sc, wta)	(26, pos)	y	5.168	**0.588**	**0.871**	0.151	1.473	**0.282**
Classical Methods (No Layer Optimization)								
Deconvolution	0	y	5.435	0.468	0.629	0.204	1.715	0.956
DeepLift	0	y	4.447	0.551	0.726	0.281	1.945	1.053
Feature Ablation	0	y	6.029	0.479	0.624	0.123	1.250	0.859
Gradient SHAP	0	y	4.808	0.552	0.783	0.261	1.862	1.142
Integrated Gradients	0	y	4.838	0.554	0.797	0.262	1.851	1.069
Occlusion	0	y	5.214	0.499	0.716	0.130	1.217	0.699
GradCAM++	30	n	5.955	0.564	0.827	0.130	1.245	0.401
Classical Methods (Layer Optimization)								
Guided Backprop	2	n	**3.276**	**0.632**	0.883	**0.281**	**2.302**	0.571
Guided Backprop	5	n	3.448	0.614	**0.887**	0.240	2.091	**0.375**
Guided GradCAM	0	n	**4.301**	**-0.053**	**0.857**	0.161	**1.459**	1.047
Guided GradCAM	26	n	5.130	-2.635	0.843	**0.179**	1.415	**0.947**
GradCAM	0	n	6.006	**0.356**	**0.638**	0.060	0.899	1.123

Continued on next page

Table 4 – continued from previous page

Method	Layer	Abs	Select. ↓	Attr. Loc. ↑	Point. ↑	Pixel Fl. AUC@5 ↑	Pixel Fl. AUC@20 ↑	Max Sens. ↓
GradCAM	14	n	**5.555**	0.306	0.580	**0.136**	**1.525**	**0.903**
LayerCAM	14	-	2.792	0.576	0.798	**0.317**	**2.474**	**2.656**
LayerCAM	2	-	**3.128**	**0.580**	**0.855**	0.300	2.191	5.508
LRP $\gamma = 0.25$	0	n	**3.189**	**0.615**	**0.680**	0.218	1.918	0.519
LRP $\gamma = 0.25$	14	n	3.593	0.577	0.611	**0.279**	**2.255**	**0.491**
Baseline Methods								
Baseline: Binary Mask	0	y	6.011	1.000	1.000	0.091	1.054	–
Baseline: Original Image	0	y	8.770	-1.699	0.572	0.061	0.619	–
Random Baseline	0	y	3.570	0.425	0.458	0.203	1.585	–

B.3. Quantitative Analysis on ResNet18

Table 5 presents detailed quantitative metrics across all split configurations, baselines, and classical methods for ResNet18.

Table 5. Comprehensive XAI Evaluation Results (ResNet50). The abbreviations are similar to Tab. 2. For each method configuration we highlight the best metrics values across the considered layer configurations in bold.

Method	Layer	Abs	Select. ↓	Attr. Loc. ↑	Point. ↑	Pixel Fl. AUC@5 ↑	Pixel Fl. AUC@20 ↑	Max Sens. ↓
SplitCAM Methods								
SplitCAM (sc, α 0.3, wta)	(1.0.2, g)	n	**3.503**	0.487	0.747	**0.168**	**1.395**	1.277
SplitCAM (sc, α 0.3, wta)	(4.1.2, g)	n	5.654	**0.673**	**0.972**	0.081	0.880	**0.233**
SplitCAM (sc, α 0.5, wta)	(2.0.1, g)	n	**3.682**	-1.370	0.887	**0.158**	**1.447**	1.078
SplitCAM (sc, α 0.5, wta)	(4.1.1, +, g)	y	6.526	**0.611**	**0.956**	0.066	0.737	**0.187**
SplitGrad Methods								
SplitGrad (α 0.3, wta)	(1.0.2, g)	y	**4.449**	0.513	0.777	**0.149**	**1.266**	0.813
SplitGrad (α 0.3, wta)	(4.1.2, g)	y	5.663	**0.649**	**0.972**	0.082	0.882	**0.219**
SplitGrad (α 0.5, wta)	(3.1.1, g)	y	**3.588**	0.559	0.880	**0.130**	**1.366**	0.859
SplitGrad (α 0.5, wta)	(4.1.2, g)	y	5.690	**0.654**	**0.972**	0.083	0.886	**0.230**
SplitLRP Methods								
SplitLRP (sc, wta)	(conv1, pos)	y	**6.313**	**0.506**	**0.587**	**0.056**	**0.697**	0.755
SplitLRP (sc, wta)	(1.0.2, pos)	y	6.785	0.483	0.567	0.050	0.686	**0.696**
Classical Methods (No Layer Optimization)								
Deconvolution	conv1	y	5.474	0.478	0.666	0.151	1.177	0.618
Gradient SHAP	conv1	y	5.269	0.527	0.779	0.166	1.227	1.074
GradCAM++	4.1.2	n	6.054	0.556	0.945	0.077	0.838	0.140
Integrated Gradients	conv1	y	5.265	0.528	0.777	0.167	1.226	0.994
Feature Ablation	conv1	n	4.655	0.136	0.684	0.188	1.388	1.077
Occlusion	conv1	n	5.300	0.179	0.726	0.130	1.030	0.785
Deconvolution	conv1	n	3.934	0.484	0.666	0.147	1.203	0.751
Classical Methods (Layer Optimization)								

Continued on next page

Table 5 – continued from previous page

Method	Layer	Abs	Select. ↓	Attr. Loc. ↑	Point. ↑	Pixel Fl. AUC@5 ↑	Pixel Fl. AUC@20 ↑	Max Sens. ↓
GradCAM	2.0.1	y	**4.003**	0.450	0.678	0.052	**1.024**	0.738
GradCAM	4.1.2	n	6.070	**0.554**	**0.945**	**0.076**	0.829	**0.131**
Guided GradCAM	4.0.2	n	4.929	**0.742**	0.866	0.119	1.008	0.687
Guided GradCAM	4.1.2	n	**3.843**	0.717	**0.883**	**0.129**	**1.096**	**0.612**
LayerCAM	2.0.2	n	**3.690**	0.549	0.865	**0.144**	**1.428**	0.913
LayerCAM	4.1.1	n	5.661	**0.664**	**0.971**	0.083	0.890	**0.227**
LRP $\gamma = 0.25$	conv1	y	**3.550**	**0.656**	0.852	**0.213**	**1.737**	0.521
LRP $\gamma = 0.25$	4.1.1	n	5.649	0.630	**0.965**	0.081	0.877	**0.200**
Guided Backprop	conv1	n	**3.517**	**0.621**	0.867	**0.212**	**1.757**	0.459
Guided Backprop	3.1.2	n	3.874	0.582	**0.938**	0.121	1.307	**0.213**
Baseline Methods								
Baseline: Original Image	conv1	y	8.351	-1.699	0.572	0.049	0.435	–
Baseline: Binary Mask	conv1	y	5.964	1.000	1.000	0.067	0.739	–
Random Baseline	-	y	4.071	0.425	0.440	0.147	1.067	–

B.4. Randomization Sanity Check

In this subsection, we apply the model randomization sanity check introduced by [App1]. The test evaluates whether a saliency method truly depends on the model's learned parameters or whether it merely reflects input-dependent patterns that remain even when the model is corrupted. For each attribution method, we compute saliency maps on the trained VGG16 model and compare them to saliency maps obtained after randomly reinitializing the weights of the last three feed-forward layers using Gaussian noise. A method that is genuinely sensitive to the model's trained parameters should show a substantial drop in similarity between the two saliency maps, as proposed by the original paper. While [App2] discuss important shortcomings of top-down randomization-based sanity checks, we apply this test as a complementary diagnostic tool to assess the relative sensitivity of split-based methods to learned model parameters.

We quantify similarity using three metrics: **(i) HOG similarity**, which measures agreement of gradient-orientation structure via the Pearson correlation of the two Histogram of Oriented Gradients; **(ii) Spearman rank correlation (signed)**, which captures preservation of the pixel-wise ordering and direction of attributions; and **(iii) SSIM**, the Structural Similarity Index, assessing perceptual similarity based on luminance, contrast, and structural information. Lower similarity values indicate stronger dependence on learned model parameters.

Analysis. The results in Tab. 6 highlight that the backward-pass stabilization used in SplitCAM and SplitGrad effectively mitigates the gradient explosion issues caused by the non-negative split-stream weights. In contrast, SplitLRP remains fairly insensitive to model randomization: across HOG and Spearman similarity metrics it shows values close to one (HOG ≈ 0.99, Spearman ≈ 1.00). The change of attribution maps when the final VGG16 layers are reinitialized is only measured by the SSIM metric (SSIM ≈ 0.45–0.56). By contrast, our SplitCAM and SplitGrad variants show substantially lower similarities than all of the considered baselines, particularly in Spearman and SSIM, demonstrating their exceptionally strong dependence on the learned weights and the model's actual decision process.

We attribute the insensitivity of SplitLRP to the last three VGG16 layer weights to the gradient explosion inherent to the split-stream due to its non-negative weight matrices. Since we apply the ϵ-LRP rule, which is closely related to the vanilla LRP rule and effectively reduces to a scaled version of Input $\times$ Gradient on the split-stream, the resulting SplitLRP attributions suffer from a diminishing-gradient effect. This makes changes in the final-layer weights largely insignificant for the propagated relevance. Consequently, the deeper the considered split layer, the more stable the SplitLRP maps become under randomization. Indeed, the layer-26 SplitLRP configuration performs 'best' (i.e., shows the lowest similarity to the original maps) across all three metrics and SplitLRP configurations, confirming its stronger sensitivity to the learned parameters compared to early-layer SplitLRP attribution maps. Here, the scaling in the backward pass of SplitGrad and SplitCAM described and analyzed in Appendix F.3 proves powerful.

Table 6. Randomization sanity check: Mean values across the custom test subset of ImageNet-S. The abbreviations are similar to Tab. 1.

Method	HOG ↑	Spearman ↑	SSIM ↑
Baselines			
Deconvolution	0.985	0.999	0.481
DeepLift	0.365	0.274	0.716
GradientShap	0.286	0.081	0.571
input_x_gradient	0.313	0.082	0.621
integrated_gradients	0.303	0.102	0.584
layer_cam_features_14_relu	0.688	0.123	0.417
layer_cam_features_2_relu	0.542	0.114	0.475
saliency	0.188	0.416	0.353
SplitCAM			
SplitCAM (sc, α 0.4) (12, g)	0.736	0.729	0.202
SplitCAM (sc, α 0.4) (14, g)	0.724	0.194	0.174
SplitCAM (sc, α 0.4, wta) (14, g)	0.724	0.389	0.303
SplitCAM (sc, α 0.4, wta) (26, +, g) abs	0.728	0.342	0.319
SplitCAM (sc, α 0.5, wta) (14, g)	0.728	0.096	0.292
SplitCAM (sc, α 0.5, wta) (26, +, g)	0.804	0.186	0.148
SplitGrad			
SplitGrad (α 0.4) (12, +, g)	0.677	0.711	0.174
SplitGrad (α 0.4) (14, +, g) abs	0.704	0.197	0.091
SplitGrad (α 0.4, wta) (12, +, g)	0.706	0.454	0.174
SplitGrad (α 0.4, wta) (14, +, g) abs	0.721	0.211	0.199
SplitGrad (α 0.5, wta) (12, +, g) abs	0.725	0.238	0.206
SplitGrad (α 0.5, wta) (26, +, g) abs	0.795	0.149	0.306
SplitLRP			
SplitLRP (7, pos)	0.997	1.000	0.471
SplitLRP (19, pos)	0.998	1.000	0.562
SplitLRP (wta) (7, pos)	0.998	1.000	0.456
SplitLRP (wta) (26, pos) abs	0.994	0.999	0.446

B.5. Computational Efficiency Analysis

To assess the computational overhead of our split-based methods, we measure mean computation times across the custom 50 image validation subset of ImageNet-S for both ResNet18 and VGG16 architectures. The model splitting operation itself is performed once per model. However, we averaged the running time over 20 split procedures for each of the models. It requires only 0.033 seconds for ResNet18 and 0.030 seconds for VGG16, representing negligible overhead. Table 7 presents the mean saliency computation times, revealing that SplitCAM and SplitGrad achieve competitive performance (0.33–0.44 seconds per image) compared to classical CAM-based methods, while being significantly faster than perturbation-based approaches such as Feature Ablation and Occlusion. Notably, the shifting parameter α used in SplitCAM and SplitGrad does not affect runtime, as it only modifies the forward pass computations without introducing additional computational complexity. For comparability in this test we compute the attribution maps with respect to the first model layer for all methods.

We note that we manually implemented the split-pair forward and backward passes in PyTorch without using its hooking system. Further, we build the split-pair as an independent module and duplicate all weight matrices when splitting into their positive and negative part. As a result, our current implementation is not fully optimized; both running time and memory efficiency can be improved substantially. In principle, the split stream can operate directly on the original model weights, making weight duplication unnecessary. Moreover, the forward and backward computations for each layer can be parallelized, so the overall overhead relative to the standard forward and backward pass of the model is expected to be modest.

Table 7. Mean Computation Time Comparison (seconds)

Method	**ResNet18**	**VGG16**
Model Splitting Time	0.0331	0.0295
Split Methods		
SplitCAM	0.4368	0.3348
SplitGrad	0.4284	0.3309
SplitLRP	0.5711	0.3204
Classical Methods		
GradCAM	0.0216	0.0232
GradCAM++	0.0157	0.0155
LayerCAM	0.0149	0.0124
Guided GradCAM	0.3102	0.2808
LRP $\gamma = 0.25$	0.0599	0.0507
DeepLift	0.0203	0.0210
Integrated Gradients	0.0070	0.0121
Gradient SHAP	0.0084	0.0082
Guided Backprop	0.0253	0.0258
Deconvolution	0.0087	0.0074
Saliency	0.0060	0.0047
Input $\times$ Gradient	0.0055	0.0046
Feature Ablation	231.6631	192.2602
Occlusion	238.2302	201.0051

C. Ablation Studies

In this appendix, we present ablation studies examining the effect of the proposed stabilization techniques for the forward and backward pass, as well as the influence of the selected max-pooling implementation, see Sec. 3.4.

C.1. Backward Pass Stabilization Ablation

Method	Metric	α 0.50	α 0.45	α 0.40	α 0.35	α 0.30	α 0.00
SplitGrad (wta) (26, g), abs	Point. ↑	0.8975	0.9064	0.9470	**0.9558**	0.8887	0.4011
	Attr. Loc. ↑	0.5434	0.5476	0.6051	**0.6169**	0.5346	0.4486
	Max Sens. ↓	0.6160	0.6071	0.4399	0.2382	0.3344	**0.0283**
	Pixel Fl. @20 ↑	1.5270	1.5337	**1.5808**	1.4476	1.3171	1.2471
SplitGrad (26, g), abs	Point. ↑	0.8975	0.9064	**0.9470**	0.9099	0.2827	0.6413
	Attr. Loc. ↑	0.5434	0.5476	**0.6037**	0.5216	0.4245	0.4563
	Max Sens. ↓	0.6171	0.6102	0.4426	0.3864	0.0622	**0.0079**
	Pixel Fl. @20 ↑	1.5272	1.5333	**1.5876**	1.3702	0.2431	0.5755
SplitGrad (wta) (14, g), abs	Point. ↑	0.8198	0.8198	0.8198	**0.8322**	0.8127	0.7898
	Attr. Loc. ↑	**0.5403**	0.5403	0.5402	0.5006	0.4936	0.4857
	Max Sens. ↓	0.8579	0.8597	0.8584	0.0875	0.0677	**0.0481**
	Pixel Fl. @20 ↑	2.1887	**2.1894**	2.1869	1.8740	1.8304	1.7684
SplitGrad (14, g), abs	Point. ↑	0.8198	**0.8198**	0.8180	0.7951	0.8004	0.7968
	Attr. Loc. ↑	0.5403	**0.5403**	0.5330	0.5054	0.4926	0.4847
	Max Sens. ↓	0.8618	0.8584	0.8478	0.1103	0.0662	**0.0301**
	Pixel Fl. @20 ↑	**2.1894**	2.1891	2.1185	1.3425	1.2959	1.1927
SplitCAM (sc, wta) (26, +, g), abs	Point. ↑	0.8675	0.8675	**0.9382**	0.9329	0.6131	0.8551
	Attr. Loc. ↑	0.5501	0.5535	0.6347	**0.6627**	0.4033	0.5133
	Max Sens. ↓	0.6906	0.6820	0.4573	0.2645	0.2511	**0.1196**
	Pixel Fl. @20 ↑	1.6067	**1.6085**	1.5849	1.4989	0.8534	1.5071
SplitCAM (sc) (26, +, g), abs	Point. ↑	0.8940	0.8905	**0.9346**	0.6466	0.8251	0.8604
	Attr. Loc. ↑	0.5595	0.5636	**0.6363**	0.4015	0.4977	0.5114
	Max Sens. ↓	0.6738	0.6640	0.4520	0.3729	0.1102	**0.1062**
	Pixel Fl. @20 ↑	1.5902	**1.5978**	1.5906	0.9076	1.2784	1.3605
SplitCAM (sc, wta) (14, g)	Point. ↑	0.8251	0.8251	**0.8269**	0.7880	0.7809	0.7703
	Attr. Loc. ↑	0.5363	0.5363	**0.5435**	0.5338	0.5286	0.5225
	Max Sens. ↓	1.0566	1.0569	1.0489	**0.1620**	0.1625	0.1630
	Pixel Fl. @20 ↑	2.4128	2.4130	**2.4135**	1.9014	1.8613	1.8731
SplitCAM (sc) (14, g)	Point. ↑	0.8251	0.8251	**0.8428**	0.7756	0.7686	0.7756
	Attr. Loc. ↑	0.5363	0.5363	**0.5626**	0.5363	0.5271	0.5208
	Max Sens. ↓	1.0552	1.0593	1.0136	0.1629	0.1585	**0.1555**
	Pixel Fl. @20 ↑	2.4171	**2.4172**	2.4065	1.5032	1.5082	1.5139

Table 8. Alpha ablation as performed in Sec. 5.3.3 across four SplitGrad and SplitCAM configuration of layers 14 and 26 respectively. The notation is similar to Tab. 1.

In this subsection, we present the full ablation study referenced in Sec. 5.3.3. Table 8 reports the α–ablation results for SplitCAM and SplitGrad across selected configurations.

Overall, we observe that performance generally improves as α decreases below 0.5, while all metrics—except Maximum Sensitivity—exhibit a sharp performance drop at $\alpha = 0.0$. This highlights that the stabilization mechanism in the backward pass plays a crucial role in the effectiveness of both methods. While the improvements in the Pixel Flipping metric are modest, the gains observed in the two localization metrics and the robustness metric are substantial.

The table further provides pairwise comparisons between configurations using max-pooling in convex mode versus the "winner-takes-all" mode, see Sec. 3.4. For α values close to 0.5, the differences across the SplitCAM and SplitGrad variants are minimal, indicating that the impact of max-pooling on SplitCAM's forward pass is minor in this regime. We emphasize that, since split-pair gradients coincide with the scaled version of the original gradients for $\alpha = 0.5$, the choice of max-pooling operation has no influence on the gradients in this case. The impact becomes most apparent for $\alpha \in \{0.30, 0.35\}$, where the "winner-takes-all" mode shows mostly stronger performance than the monotone mode across all methods and metrics.

C.2. Forward Pass Stabilization Ablation

In this subsection we want to investigate the influence of the forward stabilization procedure on SplitLRP's performance of layer configurations selected by the evaluation framework Sec. 5.2 as well as performance of SplitCAM on manually selected layer configurations. As a baseline, we introduce a shifting mode that effectively shifts the activations of positive neurons to half of the respective original activation and the activations of negative neurons to the negative half of it. We also tested a SplitCAM variant with a forward pass without stabilization, where we observe an activation explosion in late VGG16 layers in the order of 10^{18} resulting in numerical errors in the order of 10^9 despite trying to correct the activations in every layer such that their difference equals the original activation. For SplitLRP, the absence of the forward stabilization produced nearly entirely white saliency maps with only small, random, and semantically meaningless artifacts. Since these results were clearly unusable, we did not compute quantitative metrics for this setting.

C.2.1. SplitCAM

Table 10 summarizes the performance of all SplitCAM configurations—corresponding to those in Tab. 11—*without* forward-pass stabilization. Table 11 presents the results *with* the shifting-based stabilization procedure.

C.2.2. SplitLRP

Table 9 directly compares SplitLRP using shifting versus scaling stabilization in the forward pass. For both layer-7 configurations, replacing scaling with shifting leads to substantial performance drops across all metrics. For the later-layer configurations, the picture is mixed: while shifting yields slightly higher Attribution Localization scores, the scaling variants achieve markedly better Pointing Game performance and significantly improved robustness (Maximum Sensitivity).
These findings highlight the clear advantage of using scaled split-pair activations – and not the original activations – for LRP within the DC-decomposition framework.

Table 9. Scale vs Shift Comparison for SplitLRP Methods on VGG16.

Method	**Type**	**Select.** ↓	**Attr. Loc.** ↑	**Point.** ↑	**Pixel Fl. AUC@5** ↑	**Pixel Fl. AUC@20** ↑	**Max Sens.** ↓
SplitLRP (wta), 7	sc	**3.235**	**-0.036**	**0.836**	**0.296**	**2.291**	**0.866**
	sh	3.619	-0.610	0.698	0.260	2.070	0.914
SplitLRP (wta), 26, abs	sc	**5.168**	0.588	0.871	0.151	1.473	**0.282**
	sh	5.168	**0.628**	**0.877**	**0.161**	**1.478**	0.468
SplitLRP, 7	sc	**3.235**	**-0.036**	**0.836**	**0.296**	**2.291**	**0.866**
	sh	3.619	-0.695	0.698	0.260	2.070	0.914
SplitLRP, 19	sc	5.168	0.588	**0.871**	0.151	1.473	**0.282**
	sh	**4.955**	**0.629**	0.853	**0.197**	**1.692**	0.507

C.3. Layer Dependence Analysis

Figure 7 shows the layer dependence of different attribution methods. Each method is normalized independently to visualize the relative performance across different layers. Unlike the ablations of the previous section, here we only evaluated across the custom validation subset of ImageNet-S.
Overall, the plots reveal a strong layer dependence of all considered methods and metrics. Surprisingly, the γ-LRP does not show the strongest performance in the first layer. There are clear patterns for each of the metrics, valid across different methods. Pixel Flipping performs exceptionally well in layer 14 across almost all methods, except SplitCAM and SplitGrad with $\alpha = 0.3$. Maximum sensitivity performs best in later layers, while exploding in the last layers for LRP and SplitGrad with $\alpha = 0.3$.
Figure 7 resembles a characteristic fingerprint of a trained classification model. It reveals which layers localize most accurately, which exhibit the highest robustness to small input perturbations, and which activation or gradient pathways most strongly influence the Pixel Flipping behavior. These layer-wise patterns may provide insights into how evidence is processed throughout the network and could potentially guide future diagnostics or architectural analysis. However, a deeper investigation of these research questions lies beyond the scope of this work.

Method	Metric	α 0.50	α 0.45	α 0.40	α 0.35	α 0.30	α 0.00
SplitCAM (sc, wta) (26, +, g), abs	Point. ↑	0.7686	0.7686	0.8632	**0.8712**	0.3441	0.5453
	Attr. Loc. ↑	0.5374	0.5412	0.5943	**0.6118**	0.3595	0.4569
	Max Sens. ↓	0.6578	0.6507	0.4841	0.2873	0.1918	**0.0135**
	Pixel Fl. @20 ↑	1.2744	1.2859	**1.3209**	1.1792	0.4430	0.6527
SplitCAM (sc) (26, +, g), abs	Point. ↑	0.7565	0.7565	**0.8451**	0.3783	0.4507	0.4970
	Attr. Loc. ↑	0.5346	0.5384	**0.5897**	0.3663	0.4420	0.4545
	Max Sens. ↓	0.6604	0.6512	0.4985	0.3077	0.0279	**0.0043**
	Pixel Fl. @20 ↑	1.2680	1.2856	**1.3295**	0.5912	0.4906	0.5563
SplitCAM (sc, wta) (14, g)	Point. ↑	**0.8491**	0.8491	0.8491	0.7867	0.7847	0.7686
	Attr. Loc. ↑	0.5324	0.5321	**0.5406**	0.5306	0.5261	0.5200
	Max Sens. ↓	1.0834	1.0814	1.0885	**0.2271**	0.2309	0.2384
	Pixel Fl. @20 ↑	2.4454	2.4482	**2.4486**	1.9588	1.9193	1.9349
SplitCAM (sc) (14, g)	Point. ↑	0.7746	0.7767	**0.7847**	0.5855	0.5594	0.5573
	Attr. Loc. ↑	0.4003	0.4144	**0.5517**	0.5300	0.5209	0.5137
	Max Sens. ↓	1.5733	1.5627	1.5443	**1.2661**	1.3939	1.4272
	Pixel Fl. @20 ↑	2.1298	**2.1384**	2.1371	1.6669	1.6793	1.6740

Table 10. Alpha ablation for scale normalization across four SplitCAM configurations of layers 14 and 26. Here, in the forward pass no stabilization is applied. Notation and structure is similar to Tab. 8. Green cells indicate better performance of the method compared to the default variant with scale stabilization in the forward pass.

Method	Metric	α 0.50	α 0.45	α 0.40	α 0.35	α 0.30	α 0.00
SplitCAM (sc, wta) (26, +, g), abs	Point. ↑	0.8330	0.8149	0.7968	**0.9235**	0.6036	0.8592
	Attr. Loc. ↑	0.5242	0.5203	0.5147	**0.6549**	0.3968	0.5093
	Max Sens. ↓	0.6180	0.6182	0.6509	0.3104	0.2791	**0.1665**
	Pixel Fl. @20 ↑	**1.6804**	1.6646	1.5065	1.4962	0.8896	1.5405
SplitCAM (sc) (26, +, g), abs	Point. ↑	0.8330	0.8149	0.7988	0.5654	0.8431	**0.8652**
	Attr. Loc. ↑	**0.5242**	0.5203	0.5076	0.3736	0.4949	0.5094
	Max Sens. ↓	0.6129	0.6168	0.6494	0.3820	0.1686	**0.1664**
	Pixel Fl. @20 ↑	**1.6804**	1.6646	1.5105	0.8714	1.4289	1.4765
SplitCAM (sc, wta) (14, g)	Point. ↑	**0.8310**	0.8310	0.8310	0.7887	0.7827	0.7726
	Attr. Loc. ↑	0.5312	0.5312	**0.5376**	0.5315	0.5264	0.5203
	Max Sens. ↓	1.0562	1.0521	1.0649	**0.1633**	0.1641	0.1642
	Pixel Fl. @20 ↑	**2.4412**	2.4412	2.4400	1.9306	1.8953	1.9059
SplitCAM (sc) (14, g)	Point. ↑	0.8310	0.8310	**0.8431**	0.7746	0.7706	0.7807
	Attr. Loc. ↑	0.5311	0.5312	**0.5580**	0.5337	0.5245	0.5183
	Max Sens. ↓	1.0554	1.0568	1.0198	0.1641	0.1587	**0.1570**
	Pixel Fl. @20 ↑	**2.4412**	2.4411	2.4318	1.5187	1.5222	1.5329

Table 11. Alpha ablation for scaling normalization across four SplitCAM configurations of layers 14 and 26. Here, in the forward pass we apply shift normalization. Notation and structure is similar to Tab. 8. Green cells indicate better performance of the method compared to the default variant with scale stabilization in the forward pass.

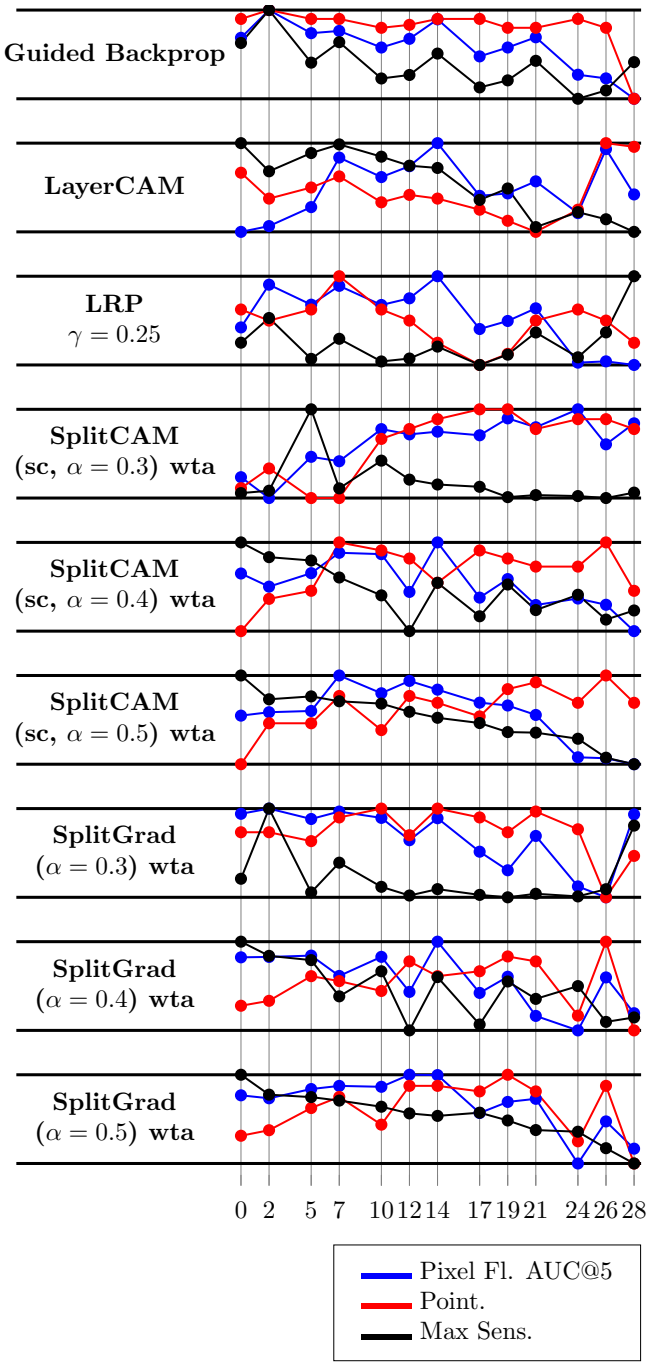

Figure 7. Layer dependence for different attribution methods. Each method is normalized independently to show relative performance across layers.

D. Difference of Networks

In this appendix, we empirically investigate two questions using MNIST as a conceptual testbed. We deliberately choose a simple image classification task to clearly visualize the mechanics of difference-of-convex models and the separation of factual vs. counterfactual evidence, before the complexity of larger datasets obscures these structural insights.

(A) Are the faithful saliency maps observed in Sec. 5 a general property of monotone neural networks?

(B) Can we directly train a *difference-of-two monotone neural networks*, rather than obtaining such a decomposition by splitting a general architecture?

Extending DIC/DM training to more complex datasets such as CIFAR-10 or ImageNet is a natural and promising direction for future work.

We introduced the two resulting architectures as DMs and DICs as well as their inverted-input variations DMINs and DICINs in Sec. 3.5. [App8] proposed an architecture similar to DICs as Convex Difference Neural Network (*CDiNN*) in the context of optimal control. Our experiments provide evidence that montone neural networks are inherently self-explanatory: ICNNs consistently focus on the most informative present and missing visual features in the image classification task. We remark that here we view ICNNs as monotone networks on an affine transformation of the input embodied by the first network layer. The gradients of the DICIN models clearly separate their concerns: The gradients of the g-model provide factual, while the gradients of the h-model provide counterfactual explanations. Moreover, all proposed network architectures achieve excellent test accuracy on MNIST.

D.1. Experimental Setup

We evaluate two representative architectures – as rank-2 Maxout networks consistently outperformed standard ReLU networks in the difference setting, we settled with the Maxout variants of two classical neural network architectures. We remark that this choice aligns with the fact that the split-streams in Sec. 3 effectively are rank-2 Maxout networks.

`cnn_maxout`. A convolutional neural network with Maxout activations. It consists of four convolutional layers with 32, 64, 128, and 128 channels, each using 3×3 kernels with padding 1 and batch normalization after every layer. After flattening the network applies three fully connected layers with dimensions $6272 \to 1024 \to 256 \to 10$, applying rank-2 Maxout activations after each hidden layer. In total, it contains approximately 1.2M parameters, including the Maxout expansion.

`mlp_maxout`. A wide multilayer perceptron that also employs Maxout units. It comprises four fully connected layers with dimensions $784 \to 800 \to 800 \to 800 \to 10$, using rank-2 Maxout activations after each hidden layer and batch normalization after the first three layers. This network contains approximately 4.2M parameters, including the Maxout expansion.

Restricting the weights.

To enforce non-negativity we replace weight matrices W with trainable parameters $\tilde{W}$ and apply a positivity transform p such that the effective weights are $p(\tilde{W})$. We explored several choices for p (*e.g. softplus*, exponential, square) and found that $p =$ ReLU yielded the best test accuracy when all $\tilde{W}$ were initialized by sampling from a Gaussian distribution and taking the absolute value. As a technical note, we also apply p to the scaling and shifting parameters of all *batch normalization* layers, as well as to all biases.

Because non-negative weights can cause exploding activations, we use a small initialization scale for all base model weights and apply a weight decay of 10^{-3} in all experiments.

D.2. Results

Table 12. Test accuracy across restriction modes (%). Values show median with [min, max] range over 10 runs.

Model	IC	Mon	None	DIC	DM	Diff_None	DICIN	DMIN
`mlp_maxout`	98.36 [98.07, 98.69]	98.30 [97.64, 98.49]	98.39 [98.07, 98.63]	98.56 [98.33, 98.96]	98.55 [98.23, 98.71]	98.47 [98.19, 98.72]	98.58 [98.41, 98.84]	98.54 [98.36, 98.65]
`cnn_maxout`	99.16 [98.86, 99.27]	98.50 [98.19, 98.79]	99.19 [98.61, 99.44]	97.94 [97.81, 98.04]	98.06 [97.99, 98.19]	97.94 [97.72, 98.45]	98.17 [97.96, 98.34]	98.28 [98.17, 98.54]

Table 12 reports the median test accuracies of all considered architectures across 10 trainings with different random seeds. As a baseline we also trained and evaluated the difference of two unrestricted models, denoted by Diff_None as well as an

unrestricted model, denoted by None. The highest test performances are observed from `cnn_maxout` None and IC models, reaching a median accuracy of 99.19% and 99.16% respectively. We remark that all our models achieve strong test accuracies. For the `mlp_maxout` the split variants DIC and DM outperform their respective single model counterparts IC and Mon. We assume that the strong inductive biases underlying a single `cnn_maxout` model that are favorable for the MNIST task are a reason why this does not translate to the `cnn_maxout` base model. For both basemodels the inverted-input mode boosts the difference-of-two architecture.

In case of `mlp_maxout` the IC and Mon are competitive with the unrestricted base model, while the test results are more seed-dependent for the restricted models and show higher variance. All the difference-of-two variants outperform the base model.

In case of `cnn_maxout`, the IC variant achieves a surprisingly high median test accuracy of 99.16% comparable to the performance of the unrestricted baseline, even showing better minimal test accuracy than the baseline. All the difference-of-two variants achieve strong, but unambiguously worse test accuracies compared to the single-model variants.

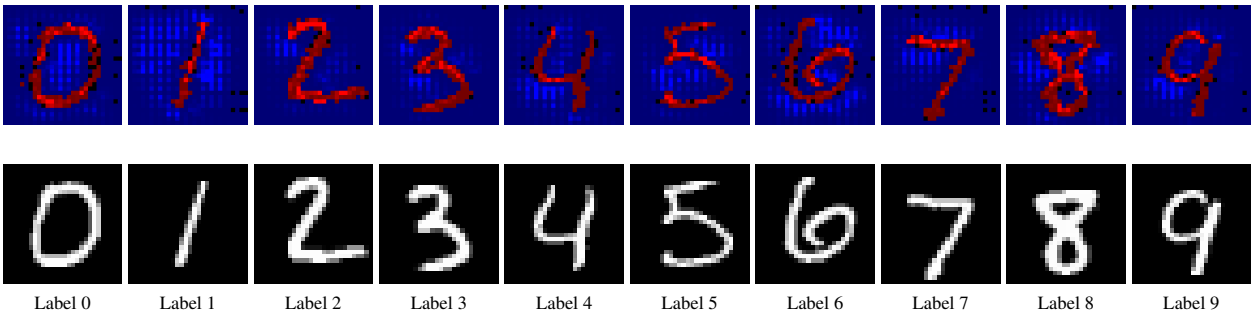

Figure 8. Input gradient visualizations of the `cnn_maxout` IC model of ten digits with respect to their ground truth label. Top row: saliency maps. Bottom row: original images for each label. We colored positive and negative split-stream gradient values above threshold 0.2 red and blue respectively, where we colorized the ranking of the absolute values with an exponential decay.

Fig. 8 visualizes input gradients of the `cnn_maxout` DIC model on representative test samples. The gradients are mostly positive on the strokes and negative on the background, indicating that adding white pixels in certain regions and deleting stroke pixels would decrease the predicted confidence. We observe that the hole of the "0", the gaps between the arches of "3" (distinguishing it from "8" and "9"), the area between the belly of "6" and the arch of "8", and the vertical line and lower curve of "7" and "8" respectively, are all critical for classification. It is remarkable that the positive gradient aligns precisely with the digit's strokes.

However, the gradients of the `cnn_maxout` IC do not exhibit class-dependent directional variation, in contrast to those of the `cnn_maxout` DIC model. Figure 3 shows overlays of g- and h-network gradients (red and blue, respectively) for all classes on two test samples with correctly predicted ground-truth label "3". Since the h-model receives $1 - x$ as input, a high gradient value (bright blue pixel) indicates that adding a white pixel at that location would increase the corresponding class logit.

Class "1" shows no g-gradient activity above the threshold, while the g-gradient for the ground-truth class "3" is strongest and follows the digit strokes most accurately. For classes "6", "8", and "9", the h-gradients are dense between the arches of the "3", indicating that those missing pixels are discriminative for these classes. In several cases, red gradients also align with the strokes of competing classes within the original digit structure.

D.3. Conclusion

Considering our results with small models on MNIST, we can answer both questions raised at the beginning of this appendix affirmatively. Concerning question (B), we observed strong performance of our proposed DM and DIC models, in case of the small MLP even outperforming all single-model variants. Concerning question (A), we even report better interpretability of the monotone and input-convex models in their single-model and difference-of-two setups compared to the DC decompositions of pretrained network architectures. Indeed, we were not able to rigorously prove the disentanglement of concerns to factual and counterfactual features in the split-pairs of the pretrained VGG16 and ResNet18 models. It is future work to translate our ideas from the proof of concept presented in this appendix to larger network architectures.

E. Generalization of the Split to Multiplicative Maxout Networks

In this chapter we lay the theoretical groundwork to split neural networks that use gating, SE-blocks and the attention mechanism into two convex neural networks. For full generality let us switch to an abstract neural network model, containing so-called addition, multiplication and input neurons. Furthermore, we consider rank-k Maxout networks, as they are a generalization of ReLU networks. For notational convenience we do not use bias, as it can be simply simulated by an additional input neuron of constant activation equal to 1. The neurons and their connections, so-called arcs, form a directed, weighted and acyclic graph—we choose to not model neural network layers for conceptual simplicity.

For an addition or multiplication neuron v we denote the set of its input neurons by $\delta^{\text{in}}v$ and its incoming weights by $w^{uv} \in \mathbb{N}$ $(u \in \delta^{\text{in}}v)$. The weights of incoming archs are $\mathbb{R}^k$-vectors in case of addition neurons and natural numbers in case of multiplication neurons. Any neuron v has a one-dimensional activation $a^v \in \mathbb{R}$. In case of input neurons we consider it constant. In case of a multiplication neuron it calculates as

$$a^v = \max_{i\in[k]} z_i^v, \text{ where the preactivations are given by} \tag{15}$$

$$z_i^v = \sum_{u\in\delta^{\text{in}}v} w_i^{uv} \cdot a^u. \tag{16}$$

The activation of a multiplication neuron calculates as

$$a^v = \prod_{u\in\delta^{\text{in}}v} (a^u)^{w^{uv}}. \tag{17}$$

We remark that we restrict input weights of multiplication neurons to natural numbers, as we do not want to handle division by zero or complex outputs. We do not use an activation function for the multiplication nodes, as one can combine multiplication neurons with addition neurons and obtain the same behaviour.

We call a network made of such multiplication and addition neurons a rank-k Maxout multiplication network. We call such a multiplication network *monotone* in case all its weights are non-negative and *input-convex* in case all weights are non-negative, except weights of arcs incident to input neurons. The following Theorem is reminiscent of the maximum representation of the support function in Eq. (13). Recall that a posynomial is a polynomial with only positive coefficients.

Theorem 3. *An input-convex rank-k Maxout multiplication network represents the maximum over a finite set of natural-exponent polynomials in the input variables. A monotone rank-k Maxout multiplication network represents the maximum over a finite set of natural-exponent posynomials in the input variables.*

Proof of Theorem 3. We only show the claim about the monotone network, as the claim about an input-convex model follows analogously. Let N be a monotone rank-k Maxout multiplication network. It is an acyclic graph of input, addition and multiplication neurons. We prove the stronger statement that the activation a^v of each neuron is the maximum of posynomials. The base cases are trivial as the activation a^v of a single input neuron v is the projection of the activations of all network input neurons on exactly that network input neuron's activation.

Observe that the set of functions represented by the maximum of posynomials is closed under scaling with positive constants, addition, taking the maximum and multiplication. This observation yields the induction steps and therefore closes the proof of Theorem 3. □

For convenience let us define the simple addition neuron that just sums weighted incoming inputs without applying an activation function. It is equivalent to a rank-k Maxout neuron with k-times copied weights. The proof of the following Theorem follows and extends the idea of Theorem in [App6]. Weight matrices are split similar to the approach for ReLU networks in Sec. 3.1 into positive and negative parts while the Maxout units are decomposed using the idea introduced for Maxpooling in Sec. 3.4. Namely, for $a^+,\ a^- \in \mathbb{R}^k$ we observe

$$\max_{j\in[k]}(a_j^+ - a_j^-) = \max_{j\in[k]}\Big\{a_j^+ + \sum_{i\neq j} a_i^-\Big\} - \sum_i a_i^-. \tag{18}$$

We split multiplication neurons by multiplying out the differences of the activations of positive and negative input neurons respectively.

Theorem 4. *Let $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ be represented by a rank-k Maxout multiplication network. Then it can be written as the difference $f = g - h$ of two convex functions g and h that are both representable with an input-convex rank-k Maxout multiplication network. In case that the input count to each multiplication neuron is bounded, the neuron counts of the monotone and input-convex networks are linear in the neuron count of the original network.*

Proof of Theorem 4. Let N_0 be the Maxout multiplication network representing f and let

$$v_1, v_2, \ldots, v_s$$

be a topological order of the neurons of N_0.
We construct a sequence of equivalent neural networks $N_1, \ldots, N_{s-1}$ and the final decomposition network N_s, where in each step we split a neuron into multiple neurons including v^+, v^- such that all new neurons have only positive incoming weights. Notice that in the intermediate networks we also make use of helping labels on input weights to multiplication neurons, changing their functionality. After the algorithm, in N_s all neurons are going to be split, there are no helping labels and no negative weights left. Setting the network outputs to $g = a^{(v_s^+)}$ and $h = a^{(v_s^-)}$ we obtain the final decomposition.
case $v \coloneqq v_p$ is an input neuron. We want to ensure that all the inputs to addition and multiplication neurons are positive. For this reason, we add addition neurons v^+ and v^- and connect them only to v, where we set all weight entries to zero, except one weight entry that is set to 1 in case of v^+ and -1 in case of v^-. We note, that this simulates two ReLU activations – v^+ transmits positive inputs and v^- transmits negative inputs.
Then, we connect v^+ to all outputs of v with the original weights and v^- to all outputs of v with signs flipped for weights that go to addition neurons. Lastly, the output weights of v^+ and v^- get a unique label, changing how multiplication nodes process their input. Namely, they consider $(a^{v^+} - a^{v^-})$ as a single input.
case v is an addition neuron. Let us consider the incoming weights w_i^{uv} of $v \coloneqq v_p$. We split them into a positive and negative part, that is,

$$w_i^{uv} = a_i^{uv} - b_i^{uv}, \quad \text{with } a_i^{uv} = \max\{w_i^{uv}, 0\}, \quad b_i^{uv} = \max\{-w_i^{uv}, 0\}$$

Observe that

$$z^v = \max_{i=1,\ldots,k} \left\{ \sum_{u \in \delta^{\text{in}} v} \left(a_i^{uv} + \sum_{j \neq i} b_j^{uv} \right) z_u \right\} - \sum_{u \in \delta^{\text{in}} v} \sum_j b_j^{uv} z_u. \tag{19}$$

Let us split v into v^+ and v^- where v^+ has input weights $w_i^{uv^+} \coloneqq a_i^{uv} + \sum_{j \neq i} b_j^{uv}$ and the same output weights as v. v^- is a simple addition neuron and has input weights $w^{uv^-} \coloneqq \sum_{j=1}^k b_j^{uv}$. Furthermore, it has the same output weights as v, only with a flipped sign towards addition neurons. Again, the output weights of v^+ and v^- get a unique label, changing how multiplication nodes process their input.
case $v \coloneqq v_p$ is a multiplication neuron. We recall the following extension of the binomial Theorem.

$$a^v = \prod_{u \in \delta^{\text{in}} v} (a^{u^+} - a^{u^-})^{w^{uv}} = \sum_{\substack{q \in \mathbb{N}^{\delta^{\text{in}} v} \\ \forall u \in \delta^{\text{in}} v:\ 0 \leq q_u \leq w_{uv}}} \sigma_q \prod_{u \in \delta^{\text{in}} v} \binom{w_{uv}}{q_u} (a^{u^+})^{(w_{uv} - q_u)} \cdot (a^{u^-})^{q_u}, \tag{20}$$

where $\sigma_q \coloneqq 2 \cdot \mathbf{1}\{\sum_{u \in \delta^{\text{in}} v} q_u \text{ is even}\} - 1$.
Let us introduce one multiplication neuron for each of the summands. To avoid exponential neuron explosion, we introduce two simple addition neurons v^+ and v^- that have multiplication neurons as input with weight 1, one for all the positive and one for all the negative summands, respectively. Now, similarly as before, we connect v^+ to all outputs of v with the original weights and v^- to all outputs of v with sign flipped weights towards addition neurons. Moreover, we label the outputs of v^+ and v^- uniquely, so they are correctly processed by multiplicative neurons.

Claim 1 (Correctness)**.** *For any $0 \leq j < s$, N_j represents f and in N_s the positive and negative output neuron represent f by their difference, namely $f = g - h$.*

Proof of Claim 1. We need to show that our input, addition and multiplication neuron transformations are not changing the function represented by the network. Then it is also easy to see that the claim $f = g - h$ holds. Indeed, the split of the input neurons into positive and negative input combined with the sign flip of weights towards addition neurons as well as the labels

towards multiplication neurons does not change the function represented by the network. The same holds for addition neurons, where we use Eq. (19). Lastly, using Eq. (20) it is easy to see that also the multiplication neuron transformation does not change the function represented by the network. □

Let g' and h' be the functions represented by positive and negative output neuron respectively, where we consider the positive and negative neurons V'_{in} created when splitting the network input neurons as the new input neurons. Let us call these networks $N_{g'}$ and $N_{h'}$.
As by Theorem 3 g' and h' are maxima of posynomials and therefore monotone and convex on positive input, and as the activations of V'_{in} in N_s are convex in the network input, we conclude, that g and h are convex.
Further remark, that indeed the neural networks representing g and h are input-convex rank-k Maxout multiplication networks. Now, together with Claim 1 this concludes the proof of Theorem 4. □

Finally, let us remark that by not splitting the input neurons as done in the proof of Theorem 4, one obtains two monotone networks in the split. However, these networks represent functions only monotone on positive input, as posynomials are only monotone on positive input. As a fix, one could simply manually add a big constant C to all network inputs and circumvent by shifting the bias of the original model. This way, the split networks represent monotone and convex functions on $\mathbb{R}^d_{\geq -C}$.

F. Further Proofs

In this appendix we rigorously define and analyze the shifting procedures proposed in Sec. 3 and provide the proofs to Theorems referred to in the main paper. First, in Appendix F.1 we give proofs about monotonicity and convexity of weight-restricted neural networks and about the correctness of the proposed DC decomposition algorithm for ReLU networks. Secondly, in Appendix F.2 we provide some intuition for the shift procedure in the forward pass and use these insights to illuminate the relationship between the different local sensitivities defined in Sec. 3.3. Lastly, in Appendix F.3 we rigorously define the shifting procedure in the backward pass and develop closed formulas for the stabilized gradients of the split-pair. The latter demonstrate how our shifting procedure effectively keeps the absolute values of the stabilized split-pair gradients within a numerically stable range and how it can potentially be used to emphasize the splitting information of different layers in the calculated gradients.

F.1. Proof of the Fundamental Theorems

In this section we give the simple proofs of Theorems referred to in Sec. 3 for completeness. The following Theorem is already discussed in [App3].

Theorem 5. *An input-convex neural network represents a convex function. A monotone neural network represents a monotone function.*

Proof of Theorem 5. Let N_{mon} and N_{ic} be a monotone and input-convex neural network respectively. Let us denote their activation function by act_{mon} and act_{ic}. For convenience we denote both their weights by $W^{(1)}, \dots, W^{(L)}$. We want to prove the monotonicity and convexity of $f^{(l)}_{\text{mon}}$ and $f^{(l)}_{\text{ic}}$ by induction on $l = L, \dots, 0$.
base $l = L$. Since $f^{(l)}_{\text{mon}}$ and $f^{(l)}_{\text{ic}}$ are the activation functions themselves, the induction base case is trivial.
step Let $0 \leq l < L$.

$$f^{(l)}_{\text{mon}}(x) = f^{(l+1)}_{\text{mon}}(x) \left(\text{act}_{\text{mon}} \left(W^{(l+1)} x\right)\right), \tag{21}$$

$$f^{(l)}_{\text{ic}}(x) = f^{(l+1)}_{\text{ic}}(x) \left(\text{act}_{\text{ic}} \left(W^{(l+1)} x\right)\right). \tag{22}$$

It is easy to see that $f^{(l)}_{\text{mon}}$ is monotone as the composition of monotone functions. Similarly, it is easy to see that $f^{(l)}_{\text{ic}}(x)$ is convex as the composition of a linear, and two convex and monotone functions. We remark that the monotonicity of act_{ic} and $f^{(l+1)}_{\text{ic}}$ is necessary - the composition of two convex functions is not convex in general. However, as we can see by the inductive argument, we do not require $f^{(0)}_{\text{ic}}$ to be monotone, so we can allow negative weights in the first layer for the argument to run through. □

We remark that monotonicity of the activation function for the input-convex neural networks is indeed necessary for its convexity.

Remark 1 (Bias)**.** *It is easy to see that we can add bias – that is not restricted to be non-negative – to a monotone or input-convex neural network without breaking the properties guaranteed by Theorem 5. For this insight let us briefly view bias as a shift of the activation function. Indeed, shifting the activation function neither breaks its convexity, its monotonicity nor its CPWL property.*

Now let us prove the correctness of the DC decomposition as described in Sec. 3.

Proof of Theorem 1. We prove the Theorem by induction on $l = L, \dots, 0$.
base $l = L$. Observe $g^{(L)}(x^+, x^-) = \max\{x^+, x^-\}$ and $h^{(L)}(x^+, x^-) = x^-$. We can verify correctness by

$$f^{(L)}\left(x^+ - x^-\right) = \text{ReLU}\left(x^+ - x^-\right) = \max\left\{x^+, x^-\right\} - x^- = g^{(L)}\left(x^+\right) - h^{(L)}\left(x^-\right). \tag{23}$$

step Let $0 \leq l < L$. Observe that $g^{(l)}(x^+, x^-) = g^{(l+1)}\left(a^{(l+1,+)}, a^{(l+1,-)}\right)$, where

$$a^{(l+1,+)} = \max\left\{W^{(l+1,+)}x^+ + W^{(l+1,-)}x^-, W^{(l+1,+)}x^- + W^{(l+1,-)}x^+\right\}, \text{ and} \tag{24}$$

$$a^{(l+1,-)} = W^{(l+1,+)}x^- + W^{(l+1,-)}x^+. \tag{25}$$

Using the induction hypothesis we obtain

$$f^{(l)}\left(x^+ - x^-\right) = f^{(l+1)}\left(\text{ReLU}\left(\left(W^{(l+1,+)} - W^{(l+1,-)}\right)\left(x^+ - x^-\right)\right)\right) \tag{26}$$

$$= f^{(l+1)}\left(\max\left\{W^{(l+1,+)}x^+ + W^{(l+1,-)}x^-, W^{(l+1,-)}x^- + W^{(l+1,+)}x^-\right\} - \left(W^{(l+1,-)}x^+ + W^{(l+1,+)}x^-\right)\right) \tag{27}$$

$$= g^{(l+1)}\left(a^{(l+1,+)}, a^{(l+1,-)}\right) - h^{(l+1)}\left(a^{(l+1,+)}, a^{(l+1,-)}\right) \tag{28}$$

$$= g^{(l)}\left(x^+, x^-\right) - h^{(l)}\left(x^+, x^-\right). \tag{29}$$

This completes the proof of Theorem 1. □

F.2. Shifting in the forward pass

Shifting the activation of the positive and negative neurons in layer l in the forward pass by the same amount c, shifts both the output of the positive and the negative stream by the same amount, which is linear in c. For notational convenience, let us define the product of the absolute weight matrices of layer $L, \dots, l+1$ by

$$W_{\text{abs}}^{[l,\rightarrow]} := \prod_{j=L,\dots,l+1} \left|W^{(j)}\right|. \tag{30}$$

Theorem 6. *For any $0 \leq l \leq L$ and any $c,\ a^{(+)},\ a^{(-)} \in \mathbb{R}^{d^{(l)}}$ we have*

$$g^{(l)}\left(\begin{bmatrix} a^{(+)} + c \\ a^{(-)} + c \end{bmatrix}\right) = g^{(l)}\left(\begin{bmatrix} a^{(+)} \\ a^{(-)} \end{bmatrix}\right) + W_{abs}^{[l,\rightarrow]} \cdot c \tag{31}$$

$$h^{(l)}\left(\begin{bmatrix} a^{(+)} + c \\ a^{(-)} + c \end{bmatrix}\right) = h^{(l)}\left(\begin{bmatrix} a^{(+)} \\ a^{(-)} \end{bmatrix}\right) + W_{abs}^{[l,\rightarrow]} \cdot c. \tag{32}$$

Proof of Theorem 6. We show the claim for $g^{(l)}$ by induction on l. It follows analogously for $h^{(l)}$.
The base case $l = L$ is trivial, as $W_{\text{abs}}^{[L,\rightarrow]}$ is an empty product and therefore the identity. For the step, let $0 \leq l < L$. As the positive and negative neurons in layer $l+1$ are connected to the previous layer with the same weights, only that $W^{(l+1,+)}$ and $W^{(l+1,-)}$ are flipped, uniformly shifting the positive and negative activations of layer l results in a uniform shift in layer $l+1$, see Fig. 2 for visualization.

$$a^{(l+1,+)} = \max\left\{W^{(l+1,+)}a^{(+)} + W^{(l+1,-)}a^{(-)},\ W^{(l+1,-)}a^{(+)} + W^{(l+1,+)}a^{(-)}\right\} + \left|W^{(l+1)}\right| c$$

and

$$a^{(l+1,-)} = W^{(l+1,-)}a^{(+)} + W^{(l+1,+)}a^{(-)} + \left|W^{(l+1)}\right| \cdot c,$$

using the induction hypothesis we obtain

$$g^{(l)}\left(\begin{bmatrix} a^{(+)} + c \\ a^{(-)} + c \end{bmatrix}\right) = g^{(l+1)}\left(\begin{bmatrix} a^{(l+1,+)} \\ a^{(l+1,-)} \end{bmatrix}\right) = g^{(l)}\left(\begin{bmatrix} a^{(+)} \\ a^{(-)} \end{bmatrix}\right) + W_{\text{abs}}^{[l,\rightarrow]} c, \tag{33}$$

which completes the proof of Theorem 6. □

This observation has consequences on the connection of the local sensitivities $\delta^{(l,+,g)}$ and $\delta^{(l,-,g)}$ as well as $\delta^{(l,+,h)}$ and $\delta^{(l,-,h)}$.

Theorem 7.

$$\delta^{(l,+,g)} = W_{abs}^{[l,\rightarrow]} - \delta^{(l,-,g)}, \tag{34}$$

$$\delta^{(l,+,h)} = W_{abs}^{[l,\rightarrow]} - \delta^{(l,-,h)} \tag{35}$$

Proof of Theorem 7. For $j \in [d^{(l)}]$ Theorem 6 yields

$$\delta_j^{(l,+,g)} = \lim_{\epsilon \to 0} \frac{g^{(l)}(a^{(l,+)} + \epsilon \cdot e_j, a^{(l,-)})}{\epsilon} = \lim_{\epsilon \to 0} \frac{g^{(l)}(a^{(l,+)}, a^{(l,-)} - \epsilon \cdot e_j) + \epsilon W_{\text{abs}}^{[l,\rightarrow]} e_j}{\epsilon} = W_{\text{abs}}^{[l,\rightarrow]} e_j - \delta_j^{(l,-,g)} \tag{36}$$

The claim about $\delta^{(l,+,h)}$ follows analogously, closing the proof of Theorem 7. □

F.3. Stabilizing the split-pair backward pass

F.3.1. The algorithm

In order to stay in numerically stable regimes we shift the local sensitivities during the backward pass in each layer uniformly by $\text{shift}_\delta^{(l,g)}$ and $\text{shift}_\delta^{(l,h)}$. We compute these modified saliency tensors $\delta_{\text{shift}}^{(l,+,g)}, \delta_{\text{shift}}^{(l,-,g)}, \delta_{\text{shift}}^{(l,+,h)}, \delta_{\text{shift}}^{(l,-,h)}$ in an iterative way: using the regular gradient formulas, and the already modified local sensitivities inflowing from subsequent network layers, we compute intermediate local sensitivities $\delta_{\text{intermediate}}^{(l,+,g)}$, $\delta_{\text{intermediate}}^{(l,-,g)}$, $\delta_{\text{intermediate}}^{(l,+,h)}$, $\delta_{\text{intermediate}}^{(l,-,h)}$. In a second step, we calculate the modifed version by

$$\delta_{\text{shift}}^{(l,+,g)} := \delta_{\text{intermediate}}^{(l,+,g)} - \text{shift}_\delta^{(l,g)} \tag{37}$$

$$\delta_{\text{shift}}^{(l,-,g)} := \delta_{\text{intermediate}}^{(l,-,g)} - \text{shift}_\delta^{(l,g)} \tag{38}$$

$$\delta_{\text{shift}}^{(l,+,h)} := \delta_{\text{intermediate}}^{(l,+,h)} - \text{shift}_\delta^{(l,h)} \tag{39}$$

$$\delta_{\text{shift}}^{(l,-,h)} := \delta_{\text{intermediate}}^{(l,-,h)} - \text{shift}_\delta^{(l,h)} \tag{40}$$

For the shifting we use

$$\text{shift}_\delta^{(l,g)} := \alpha^{(l)} \cdot \left(\delta_{\text{intermediate}}^{(l,+,g)} + \delta_{\text{intermediate}}^{(l,-,g)}\right), \tag{41}$$

$$\text{shift}_\delta^{(l,h)} := \alpha^{(l)} \cdot \left(\delta_{\text{intermediate}}^{(l,+,h)} + \delta_{\text{intermediate}}^{(l,-,h)}\right). \tag{42}$$

It turns out, that indeed these shifts are just α-multiples of the matrices $W_{\text{abs}}^{[l,\to]}$ defined in Eq. (30). We provide the closed formulas of the stabilized split-pair gradients at the end of the next section in Theorem 9.

F.3.2. Closed formulas for the stabilized split-pair gradients

In standard backpropagation for $0 \leq l < L$ the chain rule yields

$$\delta^{(l)} := \frac{\partial f^{(l)}}{\partial a^{(l)}} = \frac{\partial f^{(l)}}{\partial a^{(l+1)}} \frac{\partial a^{(l+1)}}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} = \delta^{(l+1)} \text{diag}\left(\mathbb{1}\left\{z^{(l+1)} \geq 0\right\}\right) W^{(l+1)}. \tag{43}$$

In order to define the derivatives of g and h with respect to layer activations of the original network, we need the following helper function.

$$\tilde{g}^{(l)} : \mathbb{R}^{d^{(l)}} \longrightarrow \mathbb{R}, \tag{44}$$

$$a^{(l)} \longmapsto g^{(l)}\left(\begin{bmatrix} \frac{a^{(l)}}{2} \\ -\frac{a^{(l)}}{2} \end{bmatrix}\right). \tag{45}$$

Defining $\tilde{h}^{(l)}$ similarily we now can rigorously define the split-streams gradients with respect to a unified input vector.

$$\delta^{(l,g)} := \frac{\partial \tilde{g}^{(l)}}{\partial a^{(l)}}, \qquad \delta^{(l,h)} := \frac{\partial \tilde{h}^{(l)}}{\partial a^{(l)}}, \tag{46}$$

To calculate $\delta^{(l,g)}$ directly, we use the chain rule:

$$\delta^{(l,g)} = \frac{\partial \tilde{g}^{(l)}}{\partial a^{(l)}} = \frac{\partial g^{(l)}}{\partial a^{(l,+)}} \frac{\partial a^{(l,+)}}{\partial a^{(l)}} + \frac{\partial g^{(l)}}{\partial a^{(l,-)}} \frac{\partial a^{(l,-)}}{\partial a^{(l)}} = \frac{\delta^{(l,+,g)} - \delta^{(l,-,g)}}{2}. \tag{47}$$

Similarly, we obtain

$$\delta^{(l,h)} = \frac{\delta^{(l,+,h)} - \delta^{(l,-,h)}}{2}. \tag{48}$$

We want to find a closed formula for all the derivatives. For this purpose let us define

$$m^{(l,+)} := \mathbb{1}\left\{ z^{(l,+)} \geq z^{(l,-)} \right\} = \mathbb{1}\left\{ z^{(l)} \geq 0 \right\}, \qquad M^{(l,+)} := \text{diag}\left(m^{(l,+)}\right) \tag{49}$$

$$m^{(l,-)} := \mathbb{1}\left\{ z^{(l,-)} > z^{(l,+)} \right\} = \mathbb{1}\left\{ z^{(l)} < 0 \right\}, \qquad M^{(l,-)} := \text{diag}\left(m^{(l,-)}\right) \tag{50}$$

understood elementwise. For ease of notation we define

$$W^{(l,x)} := M^{(l,+)}W^{(l)} \tag{51}$$

It is easy to see from Eq. (43) that $\delta^{(l)} = \prod_{j=L,\dots,l+1} W^{(j,x)}$ and analogously for any $0 \leq l_1 < l_2 \leq L$

$$\frac{\partial z^{(l_2)}}{\partial a^{(l_1)}} = \prod_{j=l_2,\dots,(l_1+1)} W^{(j,x)} \tag{52}$$

This formula also holds for $l_1 = l_2$ in case we interpret $\frac{\partial z^{(l)}}{\partial a^{(l)}}$ as the derivative of a network where input and output layer coincide; in this case the derivative is the identity.

Before looking at the shifted gradients let us develop some understanding of the plain gradients of the split-streams. In a first step the proof of the following Theorem calculates recursive formulas for those gradients and in a second step unfolds them into large summation formulas.

Theorem 8.

$$\delta^{(l,g)} = \frac{1}{2} \cdot \left[\delta^{(l)} - \sum_{l \leq j < L} W_{abs}^{[j+1,\rightarrow]} \cdot \left(M^{(j+1,-)} W^{(j+1)} \right) \cdot \prod_{i=(j-1),\dots,l} W^{(i+1,x)} \right], \tag{53}$$

$$\delta^{(l,h)} = -\frac{1}{2} \cdot \left[\delta^{(l)} + \sum_{l \leq j < L} W_{abs}^{[j+1,\rightarrow]} \cdot \left(M^{(j+1,-)} W^{(j+1)} \right) \cdot \prod_{i=(j-1),\dots,l} W^{(i+1,x)} \right]. \tag{54}$$

Proof of Theorem 8. First, let us find recursive formulations of $\delta^{(l,g)}$ and $\delta^{(l,h)}$. Let us calculate

$$\frac{\partial a^{(l+1,+)}}{\partial a^{(l,+)}} = \frac{\partial a^{(l+1,+)}}{\partial z^{(l+1,+)}} \frac{\partial z^{(l+1,+)}}{\partial a^{(l,+)}} + \frac{\partial a^{(l+1,+)}}{\partial z^{(l+1,-)}} \frac{\partial z^{(l+1,-)}}{\partial a^{(l,+)}} = M^{(l+1,+)}W^{(l+1,+)} + M^{(l+1,-)}W^{(l+1,-)}, \tag{55}$$

$$\frac{\partial a^{(l+1,-)}}{\partial a^{(l,+)}} = \frac{\partial a^{(l+1,-)}}{\partial z^{(l+1,+)}} \frac{\partial z^{(l+1,+)}}{\partial a^{(l,+)}} + \frac{\partial a^{(l+1,-)}}{\partial z^{(l+1,-)}} \frac{\partial z^{(l+1,-)}}{\partial a^{(l,+)}} = 0 \cdot W^{(l+1,+)} + W^{(l+1,-)}, \tag{56}$$

$$\frac{\partial a^{(l+1,+)}}{\partial a^{(l,-)}} = \frac{\partial a^{(l+1,+)}}{\partial z^{(l+1,+)}} \frac{\partial z^{(l+1,+)}}{\partial a^{(l,-)}} + \frac{\partial a^{(l+1,+)}}{\partial z^{(l+1,-)}} \frac{\partial z^{(l+1,-)}}{\partial a^{(l,-)}} = M^{(l+1,+)}W^{(l+1,-)} + M^{(l+1,-)}W^{(l+1,+)}, \tag{57}$$

$$\frac{\partial a^{(l+1,-)}}{\partial a^{(l,-)}} = \frac{\partial a^{(l+1,-)}}{\partial z^{(l+1,+)}} \frac{\partial z^{(l+1,+)}}{\partial a^{(l,-)}} + \frac{\partial a^{(l+1,-)}}{\partial z^{(l+1,-)}} \frac{\partial z^{(l+1,-)}}{\partial a^{(l,-)}} = 0 \cdot W^{(l+1,-)} + W^{(l+1,+)}. \tag{58}$$

Again the chain rule yields the iterative formulas

$$\begin{aligned} \delta^{(l,+,g)} &= \frac{\partial g^{(l+1)}}{\partial a^{(l+1,+)}} \frac{\partial a^{(l+1,+)}}{\partial a^{(l,+)}} + \frac{\partial g^{(l+1)}}{\partial a^{(l+1,-)}} \frac{\partial a^{(l+1,-)}}{\partial a^{(l,+)}} \\ &= \delta^{(l+1,+,g)} \left(M^{(l+1,+)}W^{(l+1,+)} + M^{(l+1,-)}W^{(l+1,-)} \right) + \delta^{(l+1,-,g)} W^{(l+1,-)} \end{aligned} \tag{59}$$

and further

$$\begin{aligned} \delta^{(l,-,g)} &= \frac{\partial g^{(l+1)}}{\partial a^{(l+1,+)}} \frac{\partial a^{(l+1,+)}}{\partial a^{(l,-)}} + \frac{\partial \hat{g}^{(l+1)}}{\partial a^{(l+1,-)}} \frac{\partial a^{(l+1,-)}}{\partial a^{(l,-)}} \\ &= \delta^{(l+1,+,g)} \left(M^{(l+1,+)}W^{(l+1,-)} + M^{(l+1,-)}W^{(l+1,+)} \right) + \delta^{(l+1,-,g)} W^{(l+1,+)}. \end{aligned} \tag{60}$$

With this, we obtain

$$\begin{aligned} 2 \cdot \delta^{(l,g)} &= \delta^{(l,+,g)} - \delta^{(l,-,g)} \\ &= \delta^{(l+1,+,g)} \left(M^{(l+1,+)} \cdot W^{(l+1)} - M^{(l+1,-)} \cdot W^{(l+1)} \right) - \delta^{(l+1,-,g)} W^{(l+1)} \\ &= \left(\delta^{(l+1,+,g)} - \delta^{(l+1,-,g)} \right) \cdot M^{(l+1,+)} W^{(l+1)} - \left(\delta^{(l+1,+,g)} + \delta^{(l+1,-,g)} \right) M^{(l+1,-)} W^{(l+1)} \\ &= 2 \cdot \delta^{(l+1,g)} \cdot W^{(l+1,x)} - W_{\text{abs}}^{[l+1,\rightarrow]} M^{(l+1,-)} W^{(l+1)}, \end{aligned} \tag{61}$$

where we have used in the penultimate step that $M^{(l+1,+)} + M^{(l+1,-)} = \mathbb{1}$. Similarly, we obtain

$$2 \cdot \delta^{(l,h)} = 2 \cdot \delta^{(l+1,h)} \cdot W^{(l+1,x)} - W_{\text{abs}}^{[l+1,\rightarrow]} M^{(l+1,-)} W^{(l+1)}. \tag{62}$$

Let us sanity check the recursive formulas Eq. (61) and Eq. (62) by subtracting them. We obtain $\delta^{(l,g)} - \delta^{(l,h)} = \left(\delta^{(l+1,g)} - \delta^{(l+1,h)}\right) W^{(l+1,x)}$, which turns out to be just Eq. (43).

Finally, it is easy to obtain the closed formulas by the recursion. The induction base $l = L$ is correct, as $\delta^{(L,g)} = \frac{\mathbb{1}}{2} = \frac{\delta^{(L)}}{2}$ and $\delta^{(L,h)} = -\frac{\mathbb{1}}{2} = -\frac{\delta^{(L)}}{2}$. For the step, we plug in the induction hypothesis.

$$\begin{aligned}
\delta^{(l,g)} &= \delta^{(l+1,g)} \cdot W^{(l+1,x)} - \frac{1}{2} \cdot W_{\text{abs}}^{[l+1,\rightarrow]} \cdot \left(M^{(l+1,-)} W^{(l+1)}\right) && (63)\\
&= \frac{1}{2} \cdot \left[\delta^{(l+1)} - \sum_{l+1 \leq j < L} W_{\text{abs}}^{[j+1,\rightarrow]} \cdot \left(M^{(j+1,-)} W^{(j+1)}\right) \cdot \prod_{i=(j-1),\ldots,l+1} W^{(i+1,x)}\right] \cdot W^{(l+1,x)} && (64)\\
&\quad - \frac{1}{2} \cdot W_{\text{abs}}^{[l+1,\rightarrow]} \cdot \left(M^{(l+1,-)} W^{(l+1)}\right) && (65)\\
&= \frac{1}{2} \cdot \left[\delta^{(l)} - \sum_{l+1 \leq j < L} W_{\text{abs}}^{[j+1,\rightarrow]} \cdot \left(M^{(j+1,-)} W^{(j+1)}\right) \cdot \prod_{i=(j-1),\ldots,l} W^{(i+1),x}\right] && (66)\\
&\quad - \frac{1}{2} \cdot W_{\text{abs}}^{[l+1,\rightarrow]} \cdot \left(M^{(l+1,-)} W^{(l+1)}\right) && (67)\\
&= \frac{1}{2} \cdot \left[\delta^{(l)} - \sum_{l \leq j < L} W_{\text{abs}}^{[j+1,\rightarrow]} \cdot \left(M^{(j+1,-)} W^{(j+1)}\right) \cdot \prod_{i=(j-1),\ldots,l} W^{(i+1,x)}\right] && (68)
\end{aligned}$$

One can prove the claim about $\delta^{(l,h)}$ analogously, where the only difference is a sign flip in the induction base case. This concludes the proof of Theorem 8. □

Finally, we want to give a mathematical interpretation of what the shifted gradients actually calculate. Let us introduce the notation

$$\delta_{\text{shift}}^{g} := \frac{\delta_{\text{shift}}^{(0,+,g)} - \delta_{\text{shift}}^{(0,-,g)}}{2}, \tag{69}$$

$$\delta_{\text{shift}}^{h} := \frac{\delta_{\text{shift}}^{(0,+,h)} - \delta_{\text{shift}}^{(0,-,h)}}{2}. \tag{70}$$

The closed formulas for $\delta_{\text{shift}}^{g}$ and $\delta_{\text{shift}}^{h}$ have an interesting interpretation using the following network. Let $Q[l]$ be the function represented by the original network where all ReLU activations from the l-th layer onwards are replaced with a $\min\{x, 0\}$ and the weights in layers $l+1, \ldots, L$ are replaced by their absolute values. Indeed, we observe that the summands arising in the formulas of Theorem 8 turn out to be just the respective gradients of $Q[l]$.

$$\frac{\partial Q[l]}{\partial x} = W_{\text{abs}}^{[l,\rightarrow]} \cdot \left(M^{(l,-)} W^{(l)}\right) \cdot \prod_{i=(l-1),\ldots,1} W^{(i,x)}, \tag{71}$$

where the first product term corresponds to the derivative of the last layers with only positive weights and $\min\{x, 0\}$-activation. Note that since the output of the l-th layer is negative and all the following weights are positive, all upcoming activations are negative. This means that the minimum activation functions let all activations pass unchanged.

Theorem 9 (Closed formulas for shifted saliency maps)**.** *With* $\overline{\alpha}_j := \prod_{j \leq i \leq L} \left(1 - 2\alpha^{(i)}\right)$ *we obtain*

$$\delta_{\textit{shift}}^{g} = \frac{1}{2} \left[\delta - \sum_{0 \leq j < L} \overline{\alpha}_j \frac{\partial Q[j+1]}{\partial x}\right], \tag{72}$$

$$\delta_{\textit{shift}}^{h} = -\frac{1}{2} \left[\delta + \sum_{0 \leq j < L} \overline{\alpha}_j \frac{\partial Q[j+1]}{\partial x}\right]. \tag{73}$$

In case, we do not shift, this simply means $\overline{\alpha}_j = 1$. Observe that, while not depending on shifts in the forward pass the stable gradients with respect to g and h do depend on the shifts in the backward pass and also on how we split the input x to the positive and negative stream inputs $z^{(0,+)}$ and $z^{(0,-)}$. In our definition, we choose to split them half-half, resulting in a half-half distribution of δ among δ^g and δ^h.

Remark 2 (Shifting)**.** *As shown in the following proof, the gradient $\frac{\partial Q[j]}{\partial x}$ in Eq. (72) captures the contribution of the split information flow in layer j. We observe that $\frac{\partial Q[j]}{\partial x}$ tends to grow rapidly as j approaches 0. Consequently, the shifting mechanism provides an effective lever to prevent gradient explosion and to regulate the influence of split related gradient information originating from the different layers.*

Proof of Theorem 9. As in Theorem 8, we want to prove the following claim by induction on l.

$$\delta^{(l,g)} = \frac{1}{2}\left[\delta^{(l)} - \sum_{l\leq j<L}\prod_{j\leq i\leq L}\left(1-2\alpha^{(i)}\right) W_{\text{abs}}^{[j+1,\rightarrow]}\cdot\left(M^{(j+1,-)}W^{(j+1)}\right)\cdot\prod_{i=(j-1),\ldots,l} W^{(i+1,x)}\right], \tag{74}$$

$$\delta^{(l,h)} = -\frac{1}{2}\left[\delta^{(l)} + \sum_{l\leq j<L}\prod_{j\leq i\leq L}\left(1-2\alpha^{(i)}\right) W_{\text{abs}}^{[j+1,\rightarrow]}\cdot\left(M^{(j+1,-)}W^{(j+1)}\right)\cdot\prod_{i=(j-1),\ldots,l} W^{(i+1,x)}\right]. \tag{75}$$

This in the case $l = 0$, together with Eq. (71) yields the formula stated in the Theorem. We defined

$$\delta_{\text{intermediate}}^{(l,+,g)} = \delta_{\text{shift}}^{(l+1,+,g)}\left(M^{(l+1,+)}W^{(l+1,+)} + M^{(l+1,-)}W^{(l+1,-)}\right) + \delta_{\text{shift}}^{(l+1,-,g)}W^{(l+1,-)} \tag{76}$$

$$\delta_{\text{intermediate}}^{(l,-,g)} = \delta_{\text{shift}}^{(l+1,+,g)}\left(M^{(l+1,+)}W^{(l+1,-)} + M^{(l+1,-)}W^{(l+1,+)}\right) + \delta_{\text{shift}}^{(l+1,-,g)}W^{(l+1,+)} \tag{77}$$

Similarly to the calculation in Eq. (61) we obtain

$$\begin{aligned} 2\cdot\delta_{\text{shift}}^{(l,g)} &\coloneqq \delta_{\text{shift}}^{(l,+,g)} - \delta_{\text{shift}}^{(l,-,g)} \\ &= \delta_{\text{intermediate}}^{(l,+,g)} - \delta_{\text{intermediate}}^{(l,-,g)} \\ &= \left(\delta_{\text{shift}}^{(l+1,+,g)} - \delta_{\text{shift}}^{(l+1,-,g)}\right)\cdot W^{(l+1,x)} - \left(\delta_{\text{shift}}^{(l+1,+,g)} + \delta_{\text{shift}}^{(l+1,-,g)}\right)M^{(l+1,-)}W^{(l+1)} \end{aligned} \tag{78}$$

To simplify the expression we use the following Claim.

Claim 2.

$$\delta_{\textit{shift}}^{(l,+,g)} + \delta_{\textit{shift}}^{(l,-,g)} = \prod_{l\leq j\leq L}\left(1-2\alpha^{(j)}\right) W_{\textit{abs}}^{[l,\rightarrow]}. \tag{79}$$

Proof of Claim 2. The induction base solves

$$\begin{aligned} \delta_{\text{shift}}^{(L,+,g)} + \delta_{\text{shift}}^{(L,-,g)} &= \left(1-2\alpha^{(L)}\right)\left(\delta_{\text{intermediate}}^{(L,+,g)} + \delta_{\text{intermediate}}^{(L,-,g)}\right) \\ &= \left(1-2\alpha^{(L)}\right)\left(\delta^{(L,+,g)} + \delta^{(L,-,g)}\right) \\ &= \left(1-2\alpha^{(L)}\right)\mathbb{1} \\ &= \left(1-2\alpha^{(L)}\right) W_{\text{abs}}^{[L,\rightarrow]}. \end{aligned} \tag{80}$$

For $0 \leq l < L$ adding Eq. (76) and Eq. (77) yields

$$\begin{aligned} \delta_{\text{intermediate}}^{(l,+,g)} + \delta_{\text{intermediate}}^{(l,-,g)} &= \delta_{\text{shift}}^{(l+1,+,g)}\left(M^{(l+1,+)} + M^{(l+1,-)}\right)\left|W^{(l+1)}\right| + \delta_{\text{shift}}^{(l+1,-,g)}\left|W^{(l+1)}\right| \\ &= \left(\delta_{\text{shift}}^{(l+1,+,g)} + \delta_{\text{shift}}^{(l+1,-,g)}\right)\left|W^{(l+1)}\right| \end{aligned} \tag{81}$$

Now using Eq. (81), induction yields

$$\begin{aligned}
\delta_{\text{shift}}^{(l,+,g)} + \delta_{\text{shift}}^{(l,-,g)} &= \left(1 - 2\alpha^{(l)}\right)\left(\delta_{\text{intermediate}}^{(l,+,g)} + \delta_{\text{intermediate}}^{(l,-,g)}\right) \\
&= \left(1 - 2\alpha^{(l)}\right)\left(\delta_{\text{shift}}^{(l+1,+,g)} + \delta_{\text{shift}}^{(l+1,-,g)}\right)\left|W^{(l+1)}\right| \\
&= \left(1 - 2\alpha^{(l)}\right)\prod_{l+1\leq j\leq L}\left(1 - 2\alpha^{(j)}\right) W_{\text{abs}}^{[l+1,\to]}\left|W^{(l+1)}\right| \\
&= \prod_{l\leq j\leq L}\left(1 - 2\alpha^{(j)}\right) W_{\text{abs}}^{[l,\to]}. \qquad (82)
\end{aligned}$$

which concludes the proof of Claim 2. □

Using the Claim and Eq. (78) we obtain the recursion

$$2 \cdot \delta_{\text{shift}}^{(l,g)} = 2 \cdot \delta_{\text{shift}}^{(l+1,g)} - \prod_{l+1\leq j\leq L}\left(1 - 2\alpha^{(j)}\right) W_{\text{abs}}^{[l+1,\to]} M^{(l+1,-)} W^{(l+1)} \qquad (83)$$

Unfolding this recursion one by one as in the inductive proof of Theorem 8, this closes the proof of Theorem 9. □

References (Appendix)

[App1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018.

[App2] Alexander Binder, Leander Weber, Sebastian Lapuschkin, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Shortcomings of top-down randomization-based sanity checks for evaluations of deep neural network explanations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16143–16152, June 2023.

[App3] Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

[App4] Marie-Charlotte Brandenburg, Georg Loho, and Guido Montufar. The real tropical geometry of neural networks for binary classification. *Transactions on Machine Learning Research*, 2024.

[App5] Christian Alexander Haase, Christoph Hertrich, and Georg Loho. Lower bounds on the depth of integral ReLU neural networks via lattice polytopes. In *The Eleventh International Conference on Learning Representations*, 2023.

[App6] Christoph Hertrich and Georg Loho. Neural networks and (virtual) extended formulations, 2025.

[App7] Anirbit Mukherjee, Amitabh Basu, Raman Arora, and Poorya Mianjy. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, 2018.

[App8] Parameswaran Sankaranarayanan and Raghunathan Rengaswamy. Cdinn – convex difference neural networks. *Neurocomputing*, 495(C):153–168, 2022.

[App9] Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5824–5832. PMLR, 2018.

[App10] Günter M. Ziegler. *Lectures on Polytopes*. Springer, New York, 1995.