A. SAGA-based solver for generalized linear models

In this section, we describe in further detail our solver for learning regularized GLMs in relation to existing work. Note that many of the components underlying our solver have been separately studied in prior work. However, we are the first to effectively combine them in a way that allows for GPU-accelerated fitting of GLMs at ImageNet-scale. The key algorithmic primitives we leverage to this end are variance reduced optimization methods and path algorithms for GLMs.

Specifically, our solver uses a mini-batch derivative of the SAGA algorithm (Gazagnadou et al., 2019), which belongs to a class of a variance reduced proximal gradient methods. These approaches have several benefits: a) they are easily parallelizable via GPU, b) they enjoy faster convergence rates than stochastic gradient methods, and c) they require minimal tuning and can converge with a fixed learning rate.

Algorithm 1 provides a step-by-step description of our solver. Here, the proximal operator for elastic net regularization is

$$\text{Prox}_{\lambda_1,\lambda_2}(\beta) = \begin{cases} \frac{\beta-\lambda_1}{1+\lambda_2} & \text{if } \beta > \lambda_1 \\ \frac{\beta+\lambda_1}{1+\lambda_2} & \text{if } \beta < \lambda_1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Table for storing gradients Note that the SAGA algorithm requires saving the gradients of the model for each individual example. For ImageNet-sized problems, this requires a prohibitive amount of memory, as both the number of examples ($>$1 million) and the size of the gradient (of the linear model) are large.

It turns out that for linear models with k outputs, it is actually possible to store all of the necessary gradient information for a single example in a vector of size k—as demonstrated by Defazio et al. (2014). The key idea behind this approach is that rather than storing the full gradient step $(x_i^T\beta + \beta_0 - y_i)x_i$, we can instead just store the scalar $a_i = (x_i^T\beta + \beta_0 - y_i)$ per output (i.e., a vector of length k in the case of multiple outputs). Thus, for a dataset with n examples, this reduces the memory requirements of the gradient table to $O(nk)$. For ImageNet, we find that the entire table easily fits within GPU memory limits.

There is one caveat here: in order to use this memory trick, it is necessary to incorporate the ℓ_2 regularization from the elastic net into the proximal operator. This is precisely why we use the proximal operator of the elastic net, rather than of the ℓ_1 regularization. Unfortunately, this means that the smooth part of the objective (i.e. the part not used in the proximal operator) is no longer guaranteed to be strongly convex, and so the theoretical analysis of Gazagnadou et al. (2019) no longer strictly applies. Nonetheless, we find that these variance reduced methods can still provide strong practical convergence rates in this setting without requiring much tuning of batch sizes or learning rates.

Stopping criterion We implement two simple stopping criteria, which both take in a tolerance level ε_{tol}. The first is a gradient-based stopping criteria, which terminates when:

$$\sqrt{\|\beta^{i+1} - \beta^i\|_2^2 + \|\beta_0^{i+1} - \beta_0^i\|_2^2} \leq \varepsilon_{\text{tol}}$$

Intuitively, this stops when the change in the estimated coefficients is small. Our second stopping criteria is more conservative and uses a longer search horizon, and stops when the training loss has not improved by more than ε_{tol} for more than T epochs for some T, which we call the lookbehind stopping criteria.

In practice, we find that the gradient-based stopping criteria with $\varepsilon_{\text{tol}} = 10^{-4}$ is sufficient for most cases (i.e. the solver has converged sufficiently such that the number of non-zero entries will no longer change). For significantly larger problems such as ImageNet, where individual batch sizes can have much larger variability in progressing the training objective, we find that the lookbehind stopping criteria is sufficient with $\varepsilon_{\text{tol}} = 10^{-4}$ and $T = 5$.

Relation of the solver to existing work We now discuss how our solver borrows and differs from existing work. First, note that the original SAGA algorithm (Defazio et al., 2014) analyzes the regularized form but updates its gradient estimate with one sample at a time, which is not amenable to GPU parallelism. On the other hand, Gazagnadou et al. (2019) analyze a minibatch variant of SAGA but without regularization. In our solver, we use a straightforward adaptation of minibatch SAGA to its regularized equivalent by including a proximal step for the elastic net regularization after the gradient step.

To compute the regularization paths, we closely follow the framework of Friedman et al. (2010). Specifically, we compute solutions for a decreasing sequence of regularization, using the solution of the previous regularization as a warm start for

Algorithm 1 GPU-accelerated solver for the elastic net for a step size γ and regularization parameters λ, α

Initialize table of scalars $a'_i = 0$ for $i \in [n]$
Initialize average gradient of table $g_{avg} = 0$ and $g_{0avg} = 0$
for minibatch $B \subset [n]$ **do**
  for $i \in B$ **do**
    $a_i = x_i^T \beta + \beta_0 - y_i$
    $g_i = a_i \cdot x_i$ *// calculate new gradient information*
    $g'_i = a'_i \cdot x_i$ *// calculate stored gradient information*
  end for
  $g = \frac{1}{|B|} \sum_{i \in B} g_i$
  $g' = \frac{1}{|B|} \sum_{i \in B} g'_i$
  $g_0 = \frac{1}{|B|} \sum_{i \in B} a_i$
  $g'_0 = \frac{1}{|B|} \sum_{i \in B} a'_i$
  $\beta = \beta - \gamma(g - g' + g_{avg})$
  $\beta_0 = \beta_0 - \gamma(g_0 - g'_0 + g_{0avg})$
  $\beta = \text{Prox}_{\gamma\lambda\alpha, \gamma\lambda(1-\alpha)}(\beta)$
  for $i \in B$ **do**
    $a'_i = a_i$ *// update table*
    $g_{avg} = g_{avg} + \frac{|B|}{n}(g - g')$ *// update average*
    $g_{0avg} = g_{0avg} + \frac{|B|}{n}(g_0 - g'_0)$
  end for
end for

the next. The maximum regularization value which fits only the bias term is calculated as the fixed point of the coordinate descent iteration as

$$\lambda_{max} = \max_j \frac{1}{N\alpha} \left| \sum_{i=1}^{n} x_{ij} y_i \right| \tag{5}$$

and scheduled down to $\lambda_{min} = \varepsilon \lambda_{max}$ over a sequence of K values on a log scale, as done by Friedman et al. (2010). Typical suggested values are to take $K = 100$ and $\varepsilon = 0.001$, which are what we use in all of our experiments. For extensions to logistic and multinomial regression, we refer the reader to Friedman et al. (2010), and note that our approach is the same but substituting our SAGA-based solver in liue of the coordinate descent-based solver.

A.1. Timing Experiments

In this section, we discuss how the runtime of our solver scales with the problem size. To be able to compare our solver with existing approaches, the experiments performed here are at a smaller scale than those in the main body of the paper.

Problem setting & hyperparameters. The problem we examine is that of fitting a linear decision layer for the CIFAR-10 dataset using the deep feature representation of an ImageNet-trained ResNet-50 (2048-dimensional features). We then vary the number of training examples (from 1k to 50k) and fit an elastic net regularized GLM using various methods. We compare `glmnet` (state-of-the-art, coordinate descent-based solver) on a 9th generation Intel Core i7 with 6 cores clocked at 2.6Ghz, and our approach `glm-saga` using a GeForce GTX 1080ti. We note that in these small-scale experiments, the graphics card remains at around 10-20% utilization, indicating that the problem size is too small to fully utilize the GPU.

We fix $\alpha = 0.99$, $\varepsilon = 10^{-4}$, set aside 10% of the training data for validation, and calculate regularization paths for $k = 100$ different values, which are the defaults for `glmnet`. For our approach, we additionally use a mini-batch size of 512, a learning rate of 0.1, and a tolerance level of 10^{-4} for the gradient-based stopping criteria.

Improvements in scalability As expected, on smaller problem instances with a couple thousand examples, `glmnet` is faster than our solver—cf. Table 36. This is largely due to the increased base running time of our solver—a consequence of gradient based methods requiring some time to converge. However, as the problem size grows, the runtime of `glmnet` increases rapidly, and exceeds the running time of `glm-saga` at 3,000 datapoints. For example, it takes almost 40 minutes

Table 36: Runtime in minutes for `glmnet` and `glm-saga` for fitting a sparse decision layer on the CIFAR-10 dataset using deep representations (2048D) for a pre-trained ResNet-50. Here, we assess how the runtime of different solvers scales as a function of training data points.

	Number of examples					
Solver	1k	2k	3k	4k	5k	50k
`glmnet`	2	7	25	39	58	776
`glm-saga`	9	13	17	19	22	33

to fit 4,000 data points with `glmnet`, an increase of 20x the running time for 4x the data relative to the running time for 1,000 data points. In contrast, our solver only needs 19 minutes to fit 4,000 datapoints, an increase of 2x the running time for 4x the data. Consequently, while `glmnet` takes a considerable amount of time to fit the full CIFAR10 problem size (50,000 datapoints)—nearly 13 hours—our solver can do the same in only 33 minutes. Notably, our solver can fit the regularization paths of the decision layer for the full ImageNet dataset (1 million examples with 2048 features) in approximately 6 hours.

Backpropagation libraries One more alternative to fitting linear models at scale is to use a standard autodifferentiation library such as PyTorch or Tensorflow. However, typical optimizers used in these libraries do not handle non-smooth regularizers well (i.e., the ℓ_1 penalty of the elastic net). In practice, these types of approaches must gradually schedule learning rates down to zero in order to converge, and take too long to compute regularization paths. For example, the fixed-feature transfer experiments from Salman et al. (2020) takes approximately 4 hours to fit the same CIFAR10 timing experiment for a single regularization value. In contrast, the SAGA-based optimizers enables a flexible range of learning rates that can converge rapidly without needing to tune or decay the learning rate over time.

A.2. Elastic net, ℓ_1, and ℓ_2 regularization

The elastic net is known to combine the benefits of both ℓ_1 and ℓ_2 regularization for linear models. The ℓ_1 regularization, often seen in the LASSO, primarily provides sparsity in the solution. The ℓ_2 regularization, often seen as ridge regression, brings improved performance, a unique solution via strong convexity, and a grouping effect of similar neurons. Due to this last property of ℓ_2 regularization, highly correlated features will become non-zero at the same time over the regularization path. The elastic net combines all of these strengths, and we refer the reader to Tibshirani & Wasserman (2017) for further discussion on the interaction between elastic net, ℓ_1, and ℓ_2.

A.3. Feature ordering

In the main body of the paper, we utilized regularization paths obtained via elastic net to obtain a sparse decision layer over deep features. We now discuss an additional use case of regularization paths—as a means to assess relative (deep) feature importance within the decision layer of a standard deep network. Such an ordering could, for instance, provide an alternative criteria for feature selection in "feature-highlighting" explanations (Barocas et al., 2020).

The underlying mechanism that allows us to do this is the ℓ_1 regularization in the elastic net, which imposes sparsity properties on the coefficients of the resulting linear model (Tibshirani, 1994). Specifically, the coefficients for each feature become non-zero at discrete points in the regularization path, as λ tends to zero. Informally, one can view features that are assigned non-zero coefficients earlier as being more useful from an accuracy standpoint, given the sparsity regularization.

Consequently, the order in which (deep) features are incorporated into the sparse decision layers, within the regularization path, may shed light on their relative utility within the *standard* deep network. In Figures 12- 15, we illustrate regularization paths along with the derived feature ordering for standard and robust ResNet-50 classifiers trained on ImageNet and Places-10 datasets. For all the models, it appears that features that are incorporated earlier into the regularization path (for a class) are actually more semantically aligned with the corresponding object category.

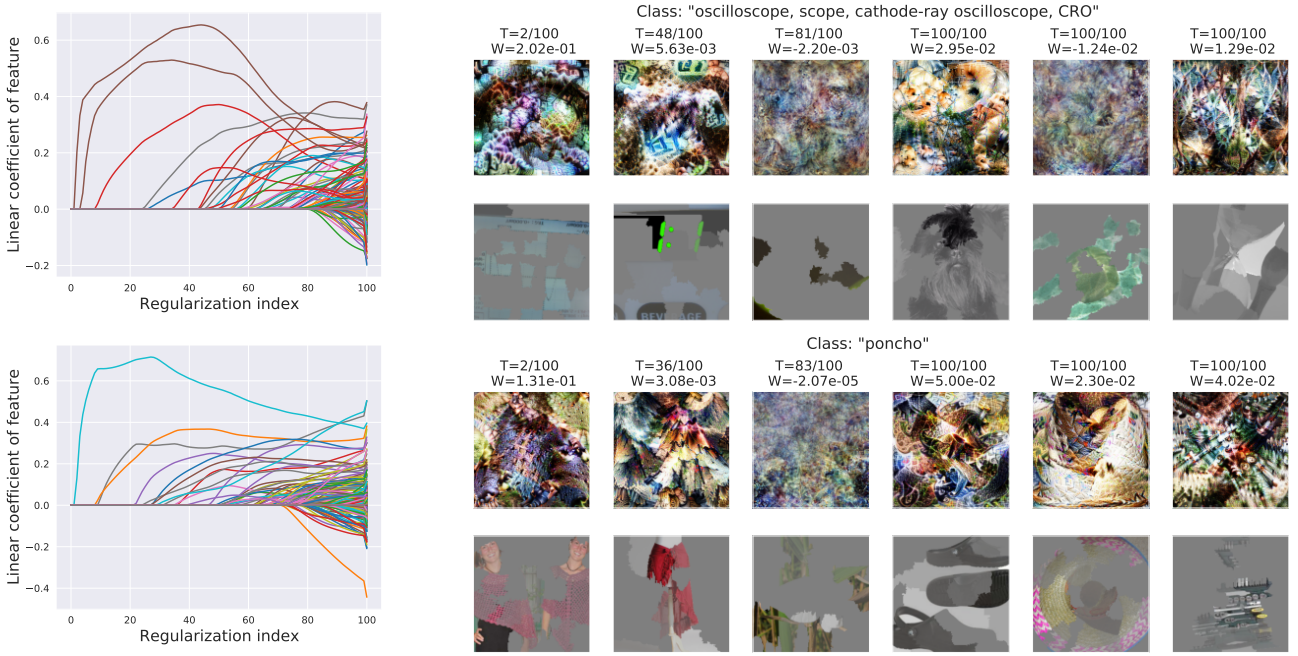


Figure 12: Sample regularization paths (*left*) and feature ordering (*right*) for sparse decision layers trained on deep features of a ResNet-50 classifier for two ImageNet classes. Regularization paths highlight when different deep features are incorporated into the decision layer as the sparsity regularization is reduced. Sample features (as feature visualizations and LIME superpixels) included into the decision layer at increasing regularization indices (T) are shown on the right.

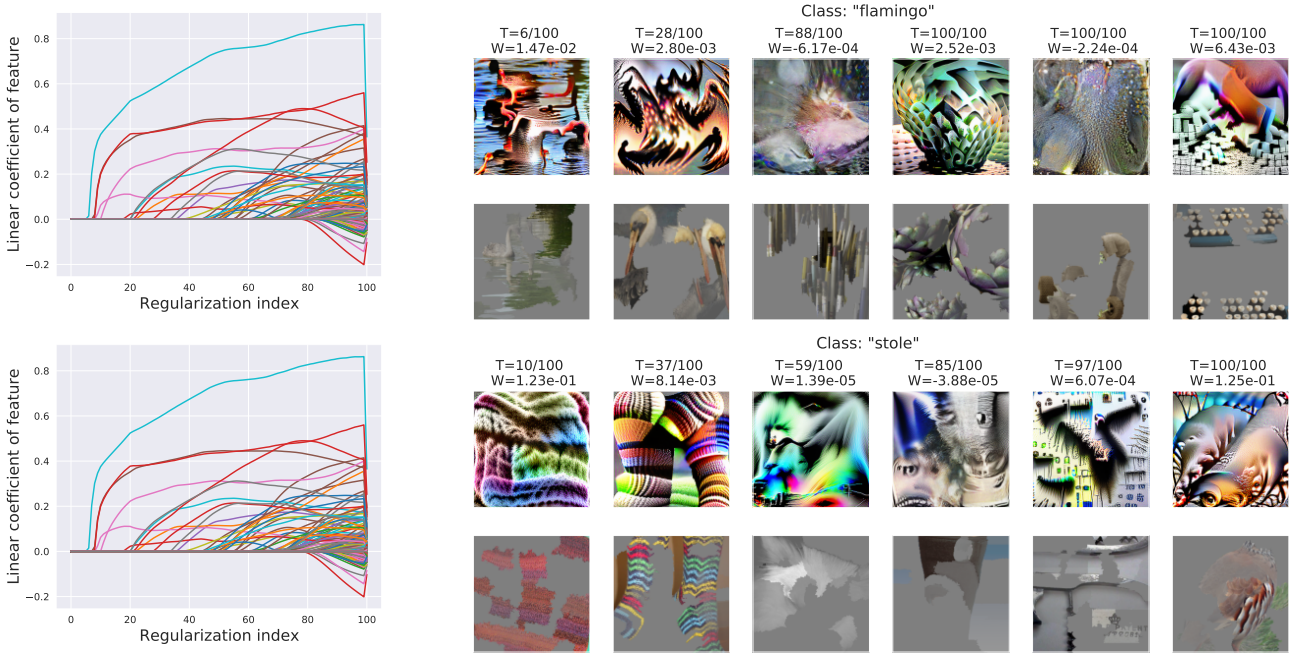


Figure 13: Sample regularization paths (*left*) and feature ordering (*right*) for sparse decision layers trained on deep features of a robust ResNet-50 classifier for two ImageNet classes. Regularization paths highlight when different deep features are incorporated into the decision layer as the sparsity regularization is reduced. Sample features (as feature visualizations and LIME superpixels) included into the decision layer at increasing regularization indices (T) are shown on the right.

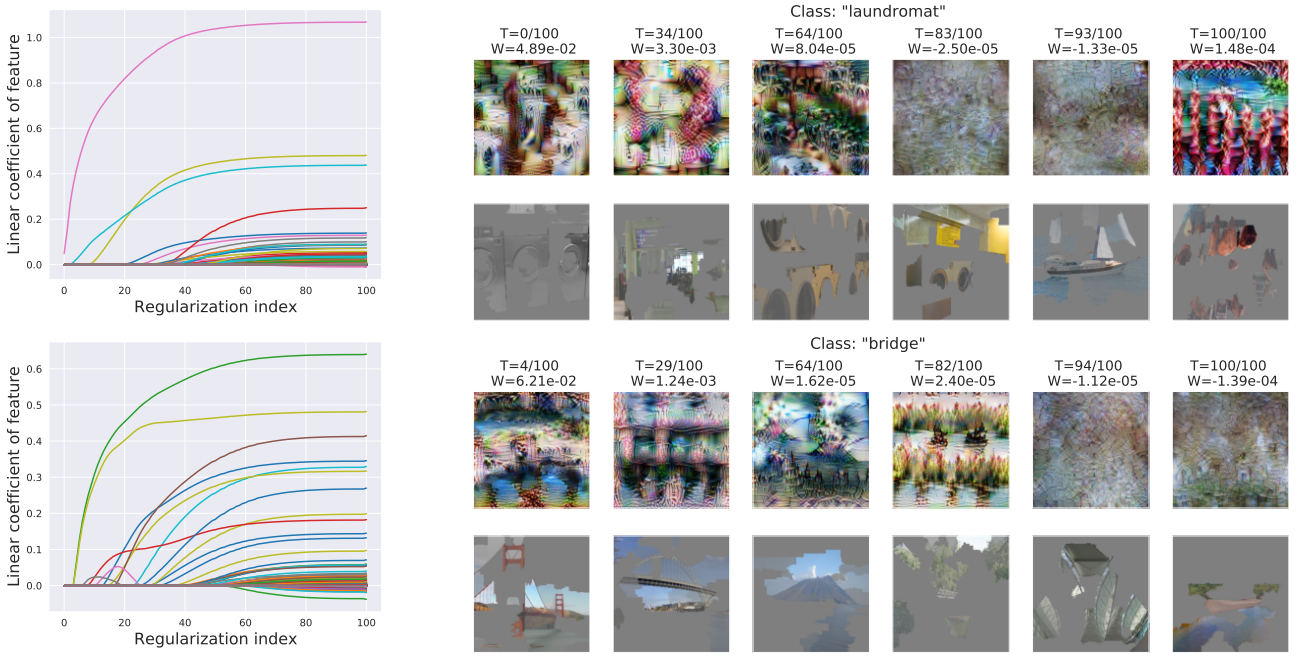


Figure 14: Sample regularization paths (*left*) and feature ordering (*right*) for sparse decision layers trained on deep features of a ResNet-50 classifier for two Places-10 classes. Regularization paths highlight when different deep features are incorporated into the decision layer as the sparsity regularization is reduced. Sample features (as feature visualizations and LIME superpixels) included into the decision layer at increasing regularization indices (T) are shown on the right.

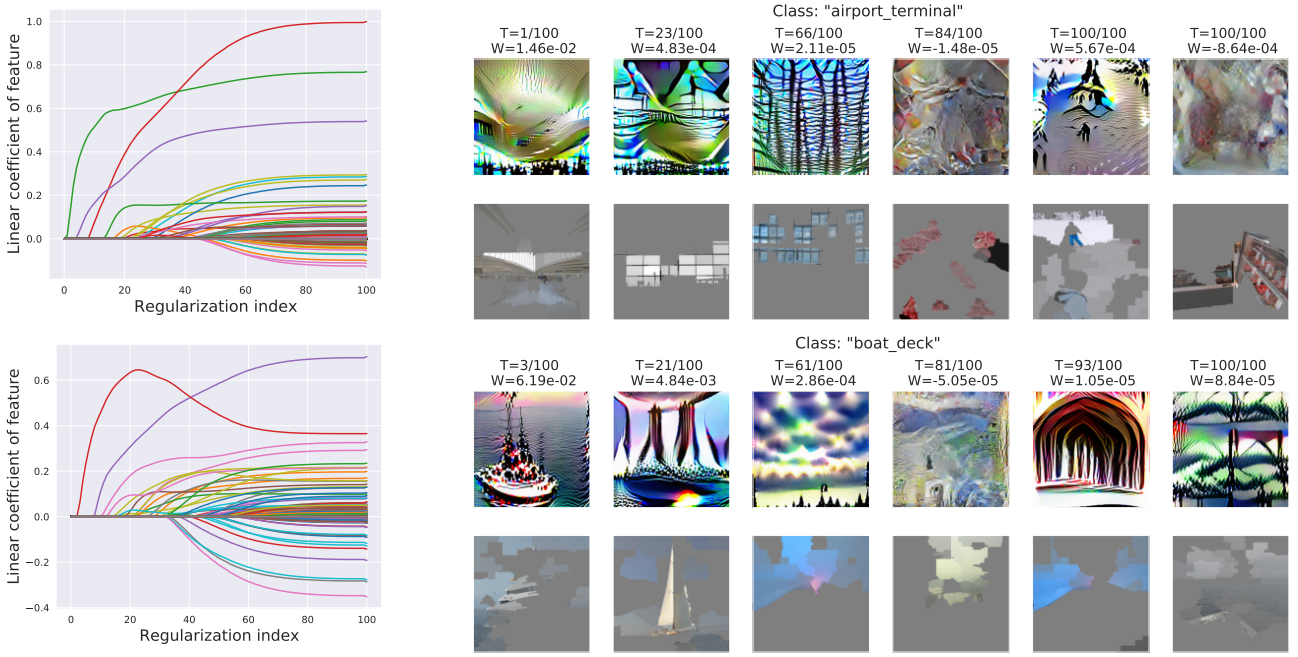


Figure 15: Sample regularization paths (*left*) and feature ordering (*right*) for sparse decision layers trained on deep features of a robust ResNet-50 classifier for two Places-10 classes. Regularization paths highlight when different deep features are incorporated into the decision layer as the sparsity regularization is reduced. Sample features (as feature visualizations and LIME superpixels) included into the decision layer at increasing regularization indices (T) are shown on the right.

B. Feature interpretations

We now discuss in depth our procedure for generating feature interpretations for deep features in the vision and language settings.

B.1. Feature visualization

Feature visualization is a popular approach to interpret individual neurons within a deep network. Here, the objective is to synthesize inputs (via optimization in pixel space) that highly activate the neuron of interest. Unfortunately, for standard networks trained via empirical risk minimization, it is well-known that vanilla feature visualization—using just gradient descent in input space—fails to produce semantically-meaningful interpretations. In fact, these visualizations frequently suffer from artifacts and high frequency patterns (Olah et al., 2017). One cause for this could be the reliance of standard models on input features that are imperceptible or unintuitive, as has been noted in recent studies (Ilyas et al., 2019).

To mitigate this challenge, there has been a long line of work on defining modified optimization objectives to produce more meaningful feature visualizations (Olah et al., 2017). In this work, we use the Tensorflow-based Lucid library[6] to produce feature visualizations for standard models. Therein, the optimization objective contains additional regularizations to penalize high-frequency changes in pixel space and to encourage transformation robustness. Further, gradient descent is performed in the Fourier basis to further discourage high-frequency input patterns. We defer the reader to Olah et al. (2017) for a more complete presentation.

In contrast, a different line of work (Tsipras et al., 2019; Engstrom et al., 2019) has shown that robust (adversarially-trained) models tend to have better feature representations than their standard counterparts. Thus, for robust models, gradient descent in pixel space is already sufficient to find semantically-meaningful feature visualizations.

B.2. LIME

Image superpixels. Traditionally, LIME is used to obtain instance-specific explanations—i.e., to identify the superpixels in a given test image that are most responsible for the model's prediction. However, in our setting, we would like to obtain a global understanding of deep features, independent of specific test examples. Thus, we use the following two step-procedure to obtain LIME-based feature interpretations:

1. Rank test set images based on how strongly they activate the feature of interest. Then select the top-k (or conversely bottom-k) images as the most prototypical examples for positive (negative) activation of the feature.
2. Run LIME on each of these examples to identify relevant superpixels. At a high level, this involves performing linear regression to map image superpixels to the (normalized) activation of the deep feature (rather than the probability of a specific class as is typical).

Due to space constraints, we use $k = 1$ in all our figures. However, in our analysis, we found the superpixels identified with $k = 1$ to be representative of those obtained with higher values.

Word clouds for language models For language models, off-the-shelf neuron interpretability tools are somewhat more limited than their vision counterparts. Of the tools listed above, only LIME is used in the language domain to produce sentence-specific explanations. Similar to our methodology for vision models, we apply LIME to a given deep feature representation rather than the output neuron. However, rather than selecting prototypical images, we instead aggregate LIME explanations over the entire validation set.

Specifically, for a given feature, we average the LIME weighting for each word over all of the sentences that the word appears in. This allows us to identify words that strongly activate/deactivate the given feature globally over the entire validation set, which we then visualize using word clouds. In practice, since a word cloud has limited space, we provide the top 30 most highly weighted words to the word cloud generator. The exact procedure is shown in Algorithm 2, and we use the word cloud generator from `https://github.com/amueller/word_cloud`.

[6] `https://github.com/tensorflow/lucid`

Algorithm 2 Word cloud feature visualization for language models for a vocabulary V, a corpus w_{ij} for $i, j \in [m] \times [n]$ of m sentences with n words

for $i = 1 \dots m$ **do**
  $\beta_i = \texttt{LIME}(w_i)$ *// generate LIME explanation for each sentence*
end for
for $w \in V$ **do**
  $K_w = \sum_{ij:w=w_{ij}} 1$ *// count number of occurances of word*
  $\hat{\beta}_w = \frac{1}{K_w} \sum_{ij:w=w_{ij}} \beta_{ij}$ *// calculate average LIME explanation of word*
end for
return $\texttt{Wordcloud}(\beta, V)$ *// generate word cloud for vocabulary V weighted by β*

C. Datasets and Models

C.1. Datasets

We perform our experiments on the following widely-used vision and language datasets.

- ImageNet-1k (Deng et al., 2009; Russakovsky et al., 2015).
- Places-10: A subset of Places365 (Zhou et al., 2017) containing the classes "airport terminal", "boat deck", "bridge", "butcher's shop", "church-outdoor", "hotel room", "laundromat", "river", "ski slope" and "volcano".
- Stanford Sentiment Treebank (SST) (Socher et al., 2013) with labels for "positive" and "negative" sentiment.
- Toxic Comments (Wulczyn et al., 2017) with labels for "toxic", "severe toxic", "obscene", "'threat", "insult", and 'identity hate".

Balancing the comment classification task. The toxic comments classification task has a highly unbalanced test set, and is largely skewed towards non-toxic comments. Consequently, the baseline accuracy for simply predicting the non-toxic label is often upwards of 90% on the unbalanced test set. To get a more interpretable and usable performance metric, we instead randomly subsample the test set to be balanced with 50% each of toxic and non-toxic comments from the corresponding toxicity category. Thus, the baseline accuracy for random chance for toxic comment classification in our experiments is 50%.

C.2. Models

We consider ResNet-50 (He et al., 2016) classifiers and BERT (Devlin et al., 2018) models for vision and language tasks respectively. In the vision setting, we consider both standard and robust models (Madry et al., 2018).

Vision. All the models are trained for 90 epochs, weight decay 1e-4 and momentum 0.9. We used a batch size of 512 for ImageNet and 128 for Places-10. The initial learning rate is 0.1 and is dropped by a factor of 10 every 30 epochs. The robust models were obtained using adversarial training with a ℓ_2 PGD adversary (Madry et al., 2018) with $\varepsilon = 3$, 3 attack steps and attack step size of $\frac{2 \times \varepsilon}{3}$.

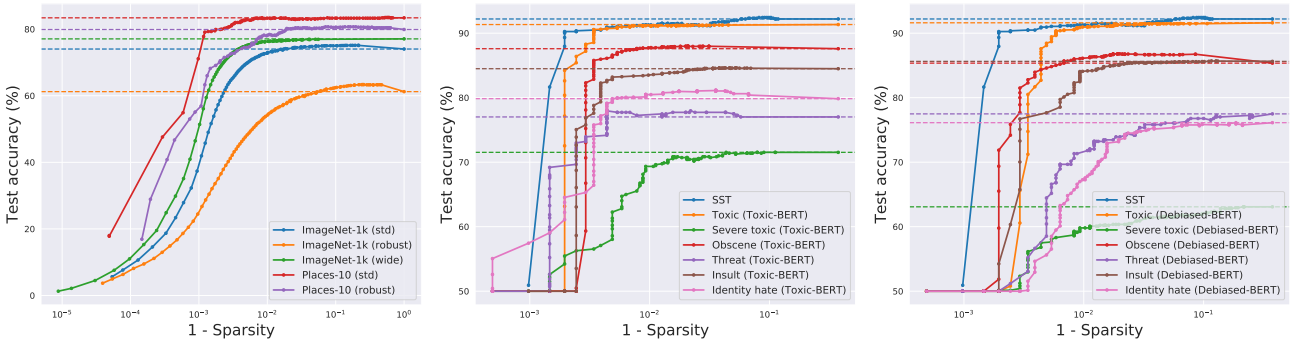
Language. The language models are all pretrained and available from the HuggingFace library, and use the standard BERT base architecture. Specifically, the sentiment classification model is from `https://huggingface.co/barissayil/bert-sentiment-analysis-sst` and the toxic comment models (both Toxic-BERT and Debiased-BERT) come from `https://huggingface.co/unitary/toxic-bert` (Hanu & Unitary team, 2020).[7]

[7]The authors informed us that the models stored on HuggingFace do not properly output multi-label outputs, detailed at `https://github.com/unitaryai/detoxify/issues/15`. However, we only used these models to produce deep representations. We then fed these representations into the sparse linear layers from our framework, and so this issue did not affect our experiments.

D. Evaluating sparse decision layers

D.1. Trade-offs for all datasets

In Figure 16, we present an extended version of Figure 4a—including all the tasks and models we consider in both the vision and language setting. Each point on the curve corresponds to single linear classifier from the regularization path in Equation (3). Note that we include the (same) SST curve in both language plots for the Toxic and Debiased BERT models.



(a) (b) (c)

Figure 16: Sparsity vs. accuracy trade-offs of models with sparse decision layers for (a) vision and (b,c) language tasks.

D.1.1. Selecting a single sparse model

As discussed in Section 2.1, the elastic net yields a sequence of linear models—with varying accuracy and sparsity—also known as the regularization path. In practice, performance of these models on a hold-out validation set can be used to guide model selection based on application-specific criteria. In our experiments, we set aside 10% of the train set for this purpose.

Our model selection thresholds. For both vision and NLP tasks, we use the validation set to identify the sparsest decision layer, whose accuracy is no more than 5% lower on the validation set, compared to the best performing decision layer. As discussed in the paper, these thresholds are meant to be illustrative and can be varied depending on the specific application. We now visualize the per-class distribution of deep features for the sparse decision layers selected in Table 4b. (We omit the NLP tasks as they entail only two classes.)

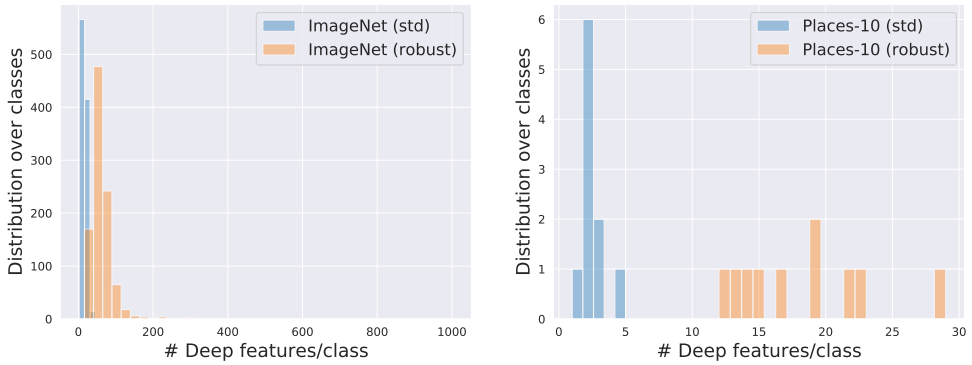


Figure 17: Distribution of the number of deep features used per class by sparse decision layers of vision models. Note that a standard (dense) decision layer uses all 2048 deep features to predict every class.

Table 37: Extended version of Table 4b: Comparison of the accuracy of dense/sparse decision layers when they are constrained to utilize only the top-k deep features (based on weight magnitude). We also show overall model accuracy, and the accuracy gained by using the remaining deep features.

Dataset/Model	k	Dense			Sparse		
		All	Top-k	Rest	All	Top-k	Rest
ImageNet (std)	10	74.03	58.46	55.22	72.24	69.78	10.84
ImageNet (wide, std)		77.07	72.42	48.75	73.48	73.45	0.91
ImageNet (robust)		61.23	28.99	34.65	59.99	45.82	19.83
Places-10 (std)	10	83.30	83.60	81.20	77.40	77.40	10.00
Places-10 (robust)		80.20	76.10	76.40	77.80	76.60	40.20
SST	5	91.51	53.21	91.17	90.71	90.48	50.92
Toxic-BERT (toxic)	5	83.33	55.35	57.87	82.47	82.33	50.00
Toxic-BERT (severe toxic)		71.53	50.00	50.14	67.57	50.00	50.00
Toxic-BERT (obscene)		80.41	50.03	50.00	77.32	72.39	50.00
Toxic-BERT (threat)		77.01	50.00	50.00	76.30	74.17	50.00
Toxic-BERT (insult)		72.72	50.00	50.00	77.14	75.80	50.00
Toxic-BERT (identity hate)		79.85	57.87	50.00	74.93	71.49	50.00
Debiased-BERT (toxic)	5	91.61	50.00	83.26	87.59	78.58	50.00
Debiased-BERT (severe toxic)		63.08	50.00	50.00	55.86	53.81	50.00
Debiased-BERT (obscene)		85.36	50.00	58.36	81.50	81.17	50.00
Debiased-BERT (threat)		77.49	50.00	50.00	68.96	50.00	50.00
Debiased-BERT (insult)		85.63	50.00	59.95	79.28	71.48	50.00
Debiased-BERT (identity hate)		76.12	50.00	50.84	71.98	50.00	50.00

D.2. Feature highlighting

In Table 37 we show an extended version of Table 4b, which now contains an additional wide ImageNet representation as well as 3 additional toxic comment categories for each toxic comment classifier. The overall test accuracy of a subset of these models (before sparsification) is under 'Dense $\rightarrow$ All' in Figure 4b.

D.3. Additional comparisons of features

In Figure 16, we visualize additional deep features used by BERT models with sparse decision layers for the SST sentiment analysis task. Figures 17- 22 show feature interpretations of deep features used by ResNet-50 classifiers with sparse decision layers trained on ImageNet and Places-10. Due to space constraints, we limit the feature interpretations for vision models to (at most) five randomly-chosen deep features used by the dense/sparse decision layer in Figure 3 and Figures 17- 22. To allow for a fair comparison between the two decision layers, we sample these features as follows. Given a target class, we first determine the number of deep features (k) used by the sparse decision layer to recognize objects of that class. Then, for both decision layers, we randomly sample five deep features from the top-k highest weighted ones (for that class).

D.3.1. LANGUAGE MODELS

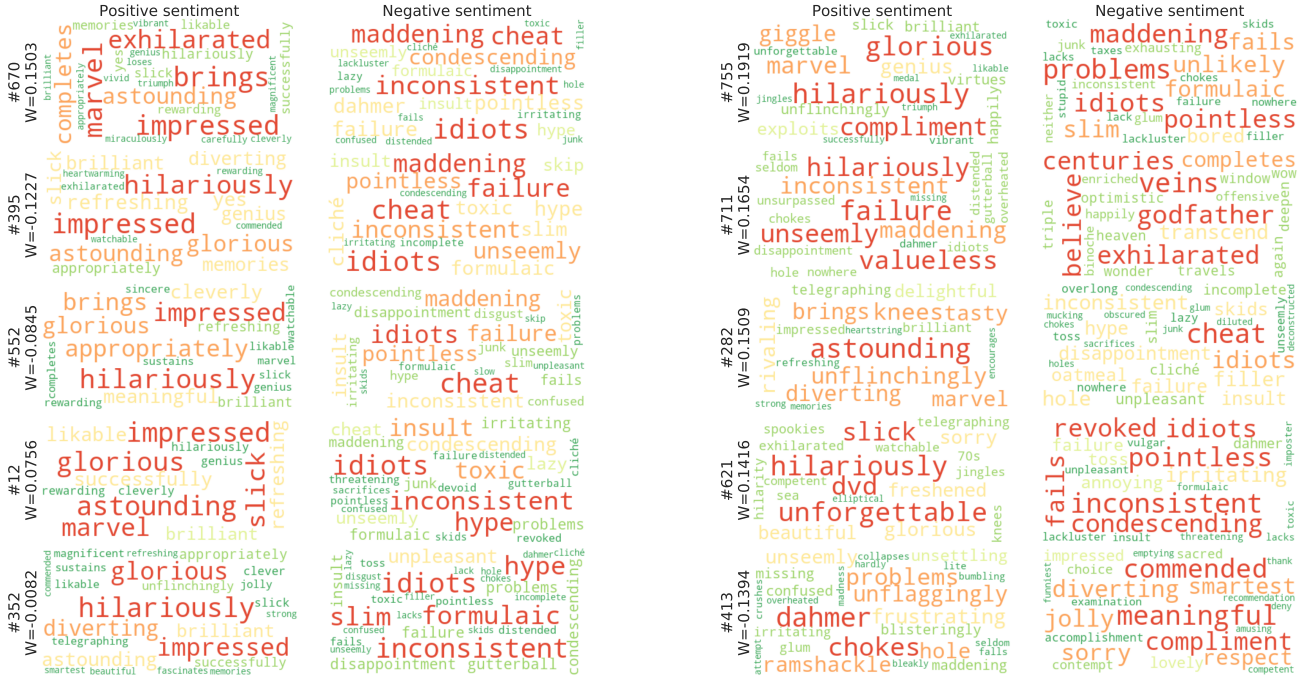
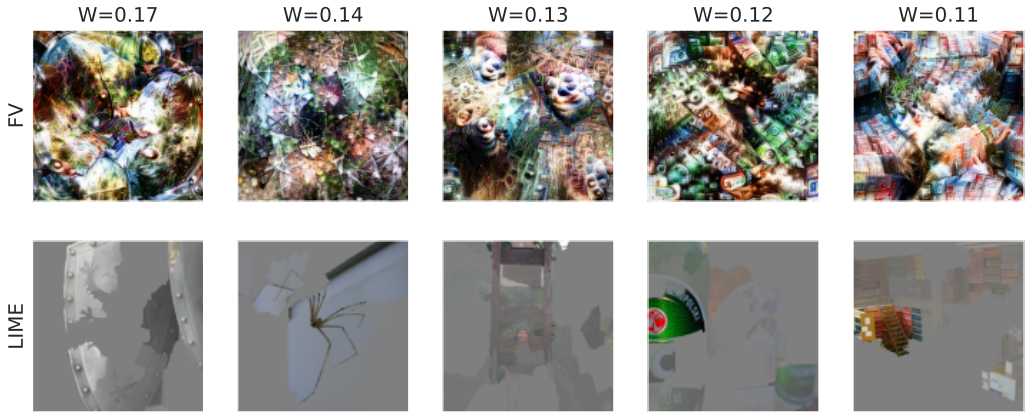


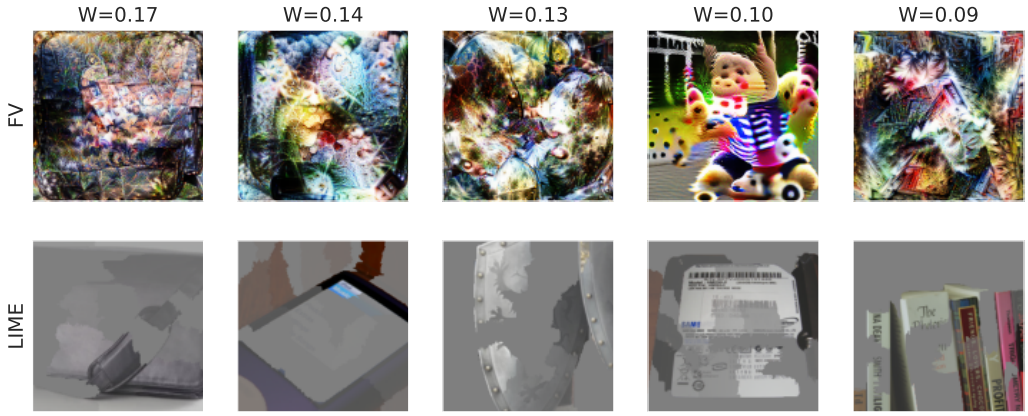
Figure 16: Additional SST word clouds visualizing the positive and negative activations for the top 5 features of the (a) sparse decision layer, (b) dense decision layer, and (c) additional randomly-selected features (positive or negative weighting is according to the dense decision layer). While the sparse model focuses on features that have clear positive and negative semantic meaning in their word clouds, the dense model and the other randomly-selected features are noticeably more mixed in sentiment.

D.3.2. VISION MODELS

(a) Class samples



(b) Dense



(c) Sparse

Figure 17: Deep features used by a standard ($\varepsilon = 0$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen ImageNet class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.

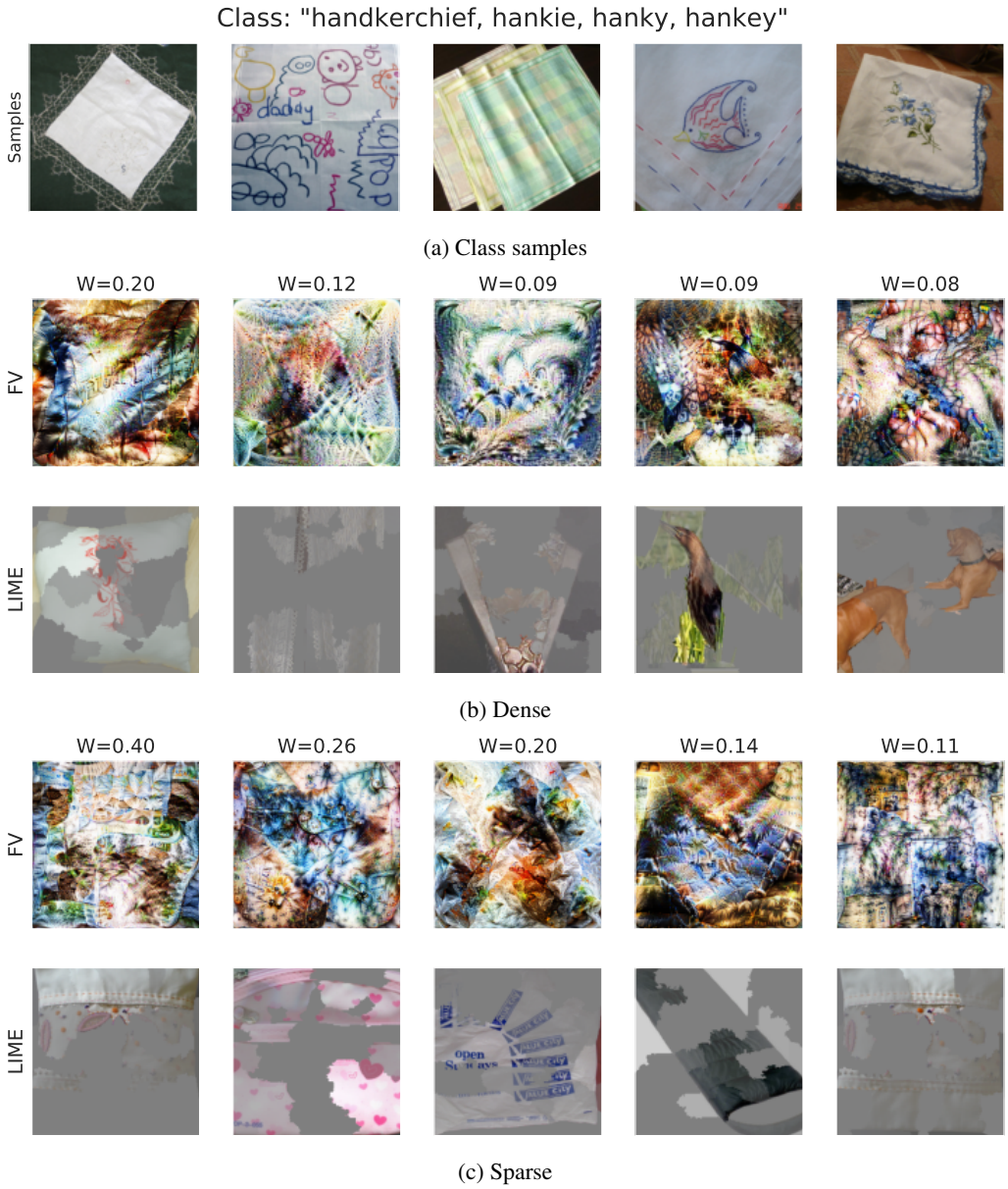
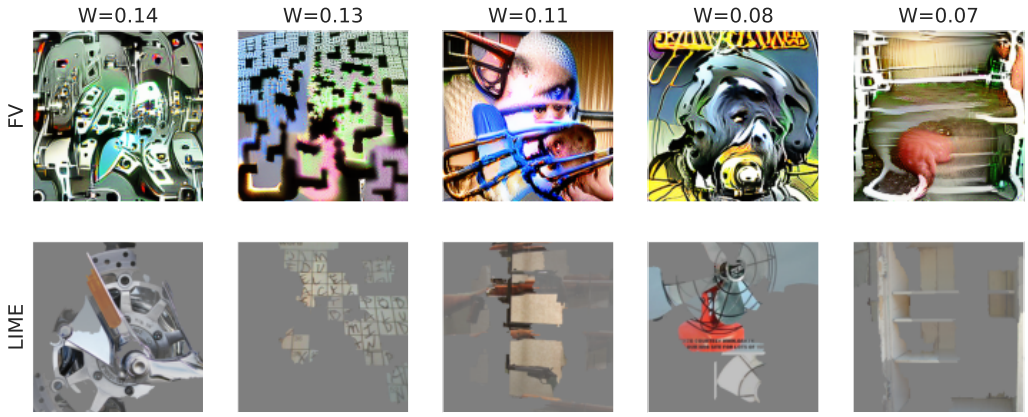
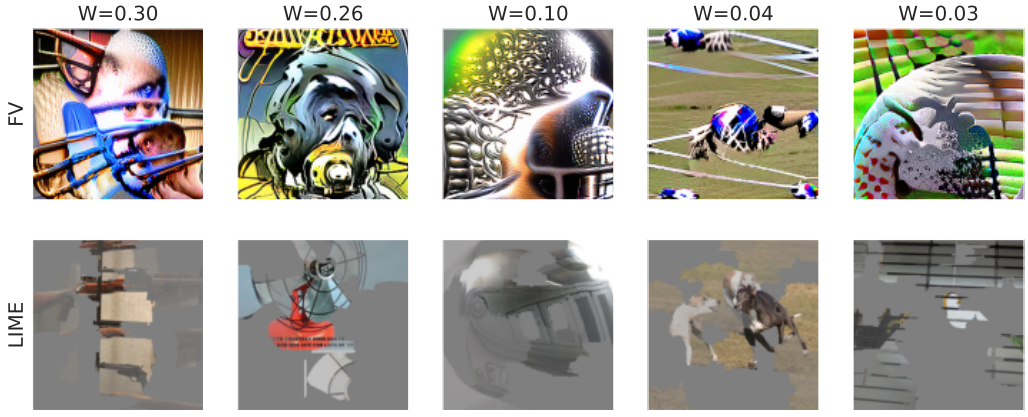


Figure 18: Deep features used by a standard ($\varepsilon = 0$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen ImageNet class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.

(a) Class samples



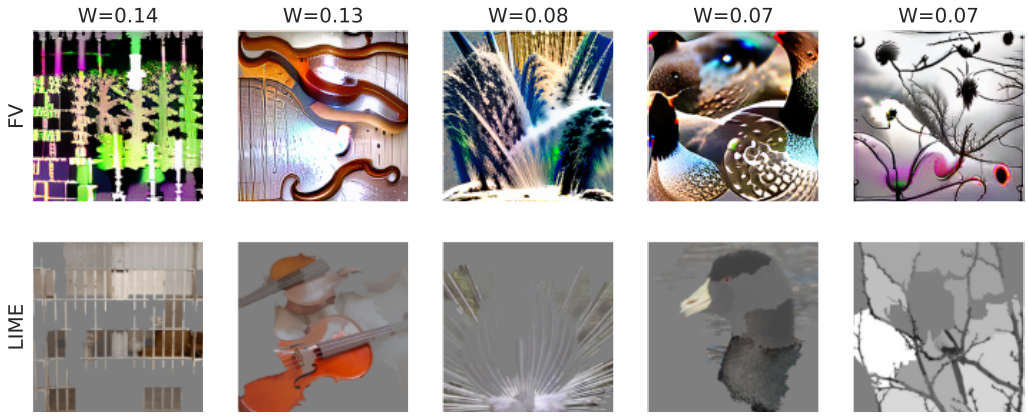
(b) Dense



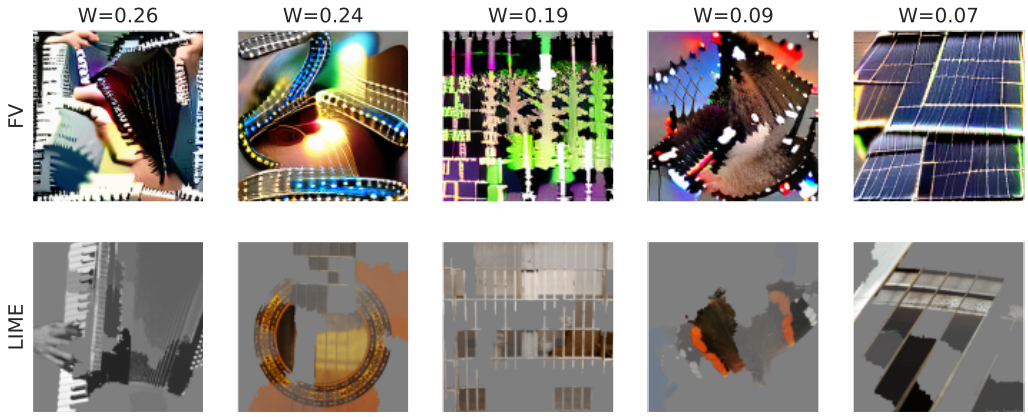
(c) Sparse

Figure 19: Deep features used by a adversarially-trained ($\varepsilon = 3$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen ImageNet class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.

(a) Class samples



(b) Dense



(c) Sparse

Figure 20: Deep features used by a adversarially-trained ($\varepsilon = 3$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen ImageNet class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.

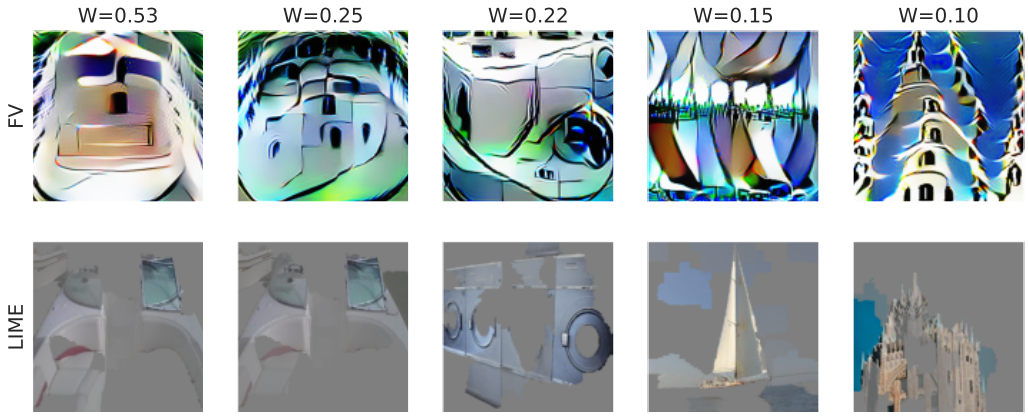
(a) Class samples

(b) Dense

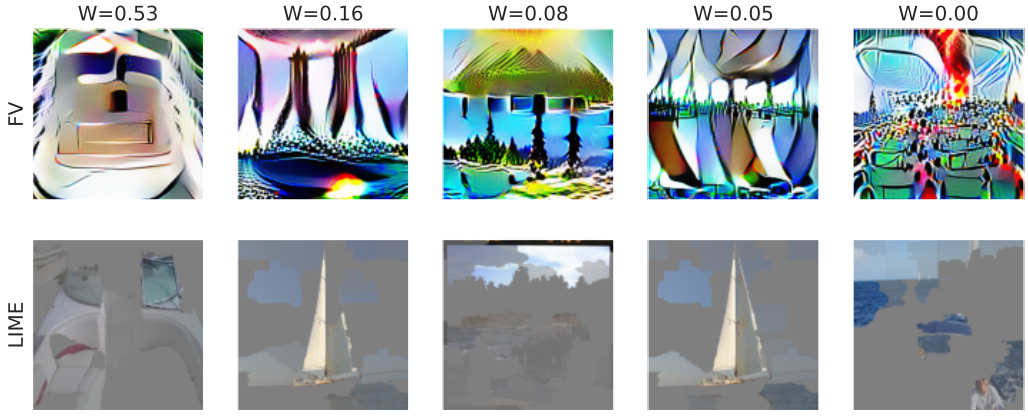
(c) Sparse

Figure 21: Deep features used by a standard ($\varepsilon = 0$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen Places-10 class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.

(a) Class samples



(b) Dense



(c) Sparse

Figure 22: Deep features used by a adversarially-trained ($\varepsilon = 3$) ResNet-50 with dense (*middle*) and sparse decision layers (*bottom*) for a randomly-chosen Places-10 class. For each (deep) feature, we show its corresponding linear coefficient in the decision layer (W), along with feature interpretations in the form of feature visualizations (FV) and LIME superpixels.

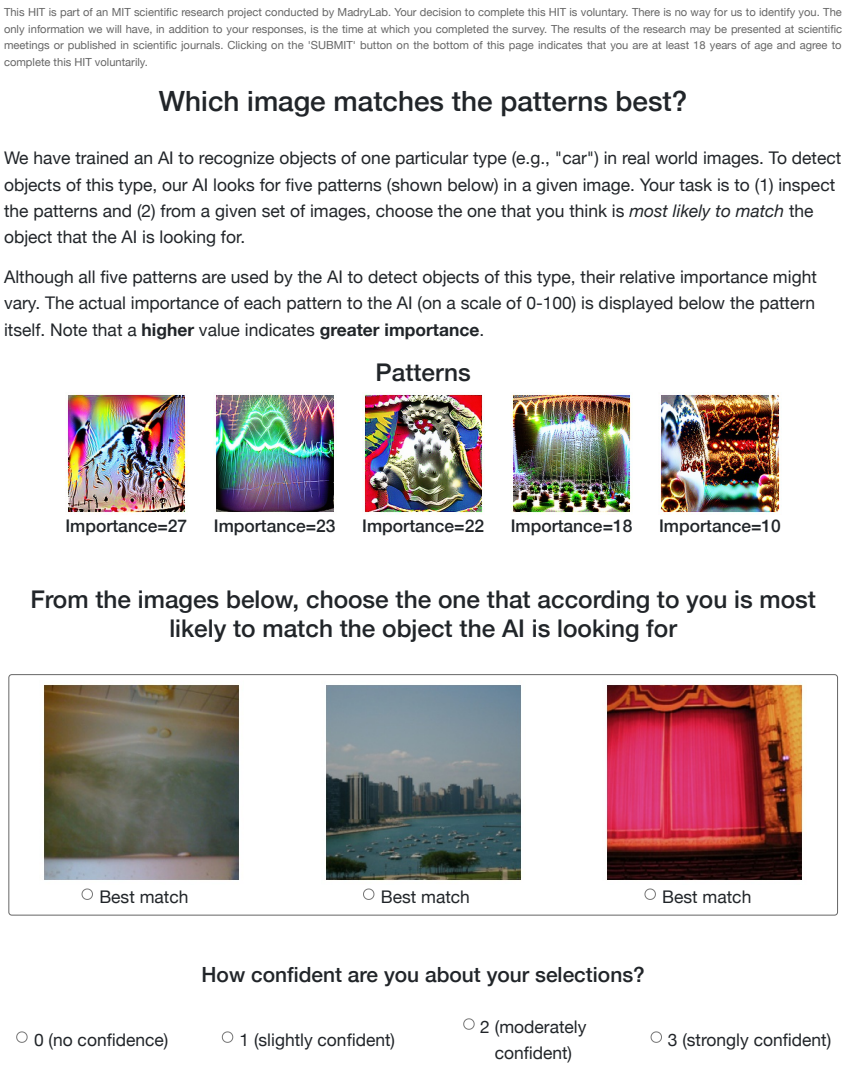


Figure 23: Sample MTurk task to assess how amenable models with dense/sparse decision layers are to human understanding.

D.4. Human evaluation

We now detail the setup of our MTurk study from Section 3.3. For our analysis, we use a ResNet-50 that has been adversarially-trained ($\varepsilon = 3$) on the ImageNet dataset. To obtain a sparse decision layer, we then train a sequence of GLMs via elastic net (cf. Section 2.1) on the deep representation of this network. Based on a validation set, we choose a single sparse decision layer—with 57.65% test accuracy and 39.18 deep features/class on average.

Task setup Recall that our objective is to assess how effectively annotators are able to simulate the predictions of a model when they are exposed to its (dense or sparse) decision layer. To this end, we first randomly select 100 ImageNet classes. Then, for each such 'target class' and decision layer (dense/sparse) pair, we created a task by:

1. **Selecting deep features:** We randomly-select *five* deep features utilized by the decision layer to recognize objects of the target class. To make the comparison more fair, we restrict our attention to deep features that are assigned significant weight (>5% of the maximum) by the corresponding model. We then present these deep features to annotators via feature visualizations. Also shown alongside are the (normalized and rescaled) linear coefficients for each deep feature.

2. **Selecting test inputs:** We rank all the test set ImageNet images based on the probability assigned by the corresponding

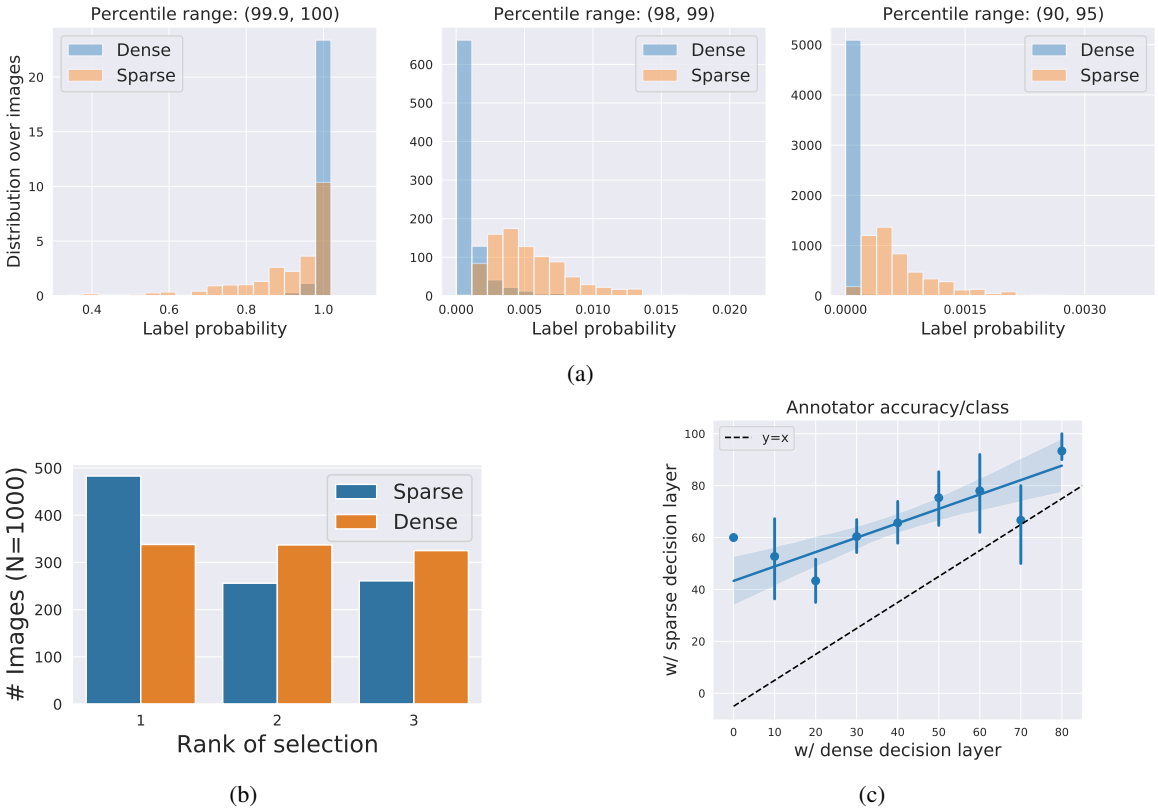


Figure 24: (a) Distribution of the target label probability assigned by the model (with a dense/sparse decision layer) to image candidates used in our MTurk Study. (b) Distribution of images selected by annotators in terms of the ranking of their target class probability. Here, a rank of k implies that the selected image has the k-th highest probability of the target class (out of the 3 images) according to the model. (c) Per-class accuracy of annotators in simulating the predictions of models with dense/sparse decision layer.

model (i.e., the ResNet-50 with a dense/sparse decision layer) to the target class. We then randomly select three images, such that they lie in the following percentile ranges in terms of target class probability: (90, 95), (98, 99) and (99.99, 100). Note that since ImageNet has 1000 diverse object categories, the target class probability of a randomly sampled image from the dataset is likely to be extremely small. Thus, fixing the percentiles as described above allows us to pick image candidates that are: (i) somewhat relevant to the target class; and (ii) of comparable difficulty for both types of decision layers. In Figure 24a, we present the target probability distribution as per the model for image candidates selected in this manner.

Finally, annotators are presented with the deep features chosen above—describing them as patterns used by an AI model to recognize objects of a certain (unspecified) type. They are then asked to pick one of the image candidates (randomly-permuted) that best matches the patterns. Annotators are also asked to mark their confidence on a likert scale. A sample task is shown in Figure 23.

For each target label-decision layer pair, we obtain 10 tasks by repeating the random selection process above. This results in a total of 2000 tasks (100 classes x 2 models x 10 tasks/(class, model)). Each task is presented to 5 annotators, compensated at $0.04 per task.

Quality control For each task, we aggregated results over all the annotators. While doing so, we eliminated individual instances where a particular annotator made no selections. We also completely eliminated instances corresponding to annotators who consistently (>80% of the times) left the tasks blank. Finally, while reporting our results, we only keep tasks for which we have selections from at least two (of five) annotators. We determine the final selection based on a majority vote over annotators, weighted by their confidence.

Results In Table 25, we report annotator accuracy—in terms of their ability to correctly identify the image with the highest target class probability as per the model. We also present a break down of the overall accuracy depending on whether or not the "correct image" is from the target class. We find that sparsity significantly boosts annotators' ability to intuit (simulate) the model—by nearly 30%. In fact, their performance on models with dense decision layers is close to chance (33%). Note also that for models with sparse decision layers, annotators are able to correctly simulate the predictions *even* when the correct image belongs to a different class.

Accuracy (%)	Dense	Sparse
Overall	35.61 ± 3.09	$\mathbf{63.02 \pm 3.02}$
From target class	44.02 ± 5.02	$\mathbf{72.22 \pm 4.74}$
From another class	30.64 ± 3.65	$\mathbf{57.33 \pm 4.00}$

Table 25: Accuracy of annotators at simulating the model given explanations from the dense and sparse classifiers.

In Figure 24b, we visualize how the image selected by annotators ranks in terms of the model's target class probability, over all tasks. Note that a rank of one implies that the annotators correctly selected the image which the model predicts as having highest target class probability. This figure largely corroborates the findings in 25—in particular, highlighting that for the standard (dense) decision layer, annotator selections are near-random. In Figure 24c, we visualize annotator accuracy—aggregated per (the 10 tasks for a) target class—for models with dense and sparse decision layers.

E. Model biases and spurious correlations

E.1. Toxic comments

In this section, we visualize the word clouds for the toxic comment classifiers which reveal the biases that the model has learned from the data. Note that these figures are heavily redacted due to the nature of these comments.

In Figure 26, we visualize the top five features for the sparse (Figure 26a) and dense (Figure 26b) decision layers of Toxic-Bert. We note that more of the words in the sparse decision layer refer to identity groups, whereas this is less clear in the dense decision layer. Even if we expand our interpretation to the top 10 neurons with the largest weight, only 7.5% of the words refer to identity groups for the model with a dense decision layer.

In Figure 27, we perform a similar visualization as for the Toxic-BERT model, but for the Debiased-BERT model. The word clouds for the sparse decision layer (Figure 27a) provide evidence that the Debiased-BERT model no longer uses identity words as prevalently for identifying toxic comments. However, it is especially clear from the word clouds for the sparse decision layer that a significant fraction of the non-toxic word clouds contain identity words. This suggests that the model now uses these identity words as strong evidence for non-toxicity, which can be also reflected to a lesser degree in the wordclouds for the dense decision layer (Figure 27b).

E.2. ImageNet

E.2.1. Human study

We now detail the setup of our MTurk study from Section 4.1. For our analysis, we use a standard ResNet-50 trained on the ImageNet dataset—with the default (dense) decision layer, as well as its sparse counterpart from Figure 4b.

Task setup. This task is designed to semi-automatically identify learned correlations in classifiers with dense/sparse decision layers. To this end, we randomly-select 1000 class pairs from each model, such that the classes share a common deep feature in the decision layer. We only consider features to which the model assigns a substantial weight for both classes (>5% maximum weight). Then, for each class (from the pair), we select the three images that maximally activate the deep feature of interest. Doing so allows us to identify the most prototypical images from each class for the given deep feature.

We then present annotators on MTurk with the six chosen images, grouped by class along with the label. We ask them: (a) whether the images share a common pattern; (b) how confident they are about this selection on a likert scale; (c) to provide a short free text description of the pattern; and (d) for each class, to determine if the pattern is part of the class object or the surrounding. A sample task is shown in Figure 28. Each task was presented to 5 annotators, compensated at $0.07 per task.

(a) (b)

Figure 26: Word cloud visualizations of the top 5 deep features in Toxic-BERT for the (a) sparse decision layer and (b) dense decision layer

(a) (b)

Figure 27: Word cloud visualizations of the top 5 deep features in Debiased-BERT for the (a) sparse decision layer and (b) dense decision layer

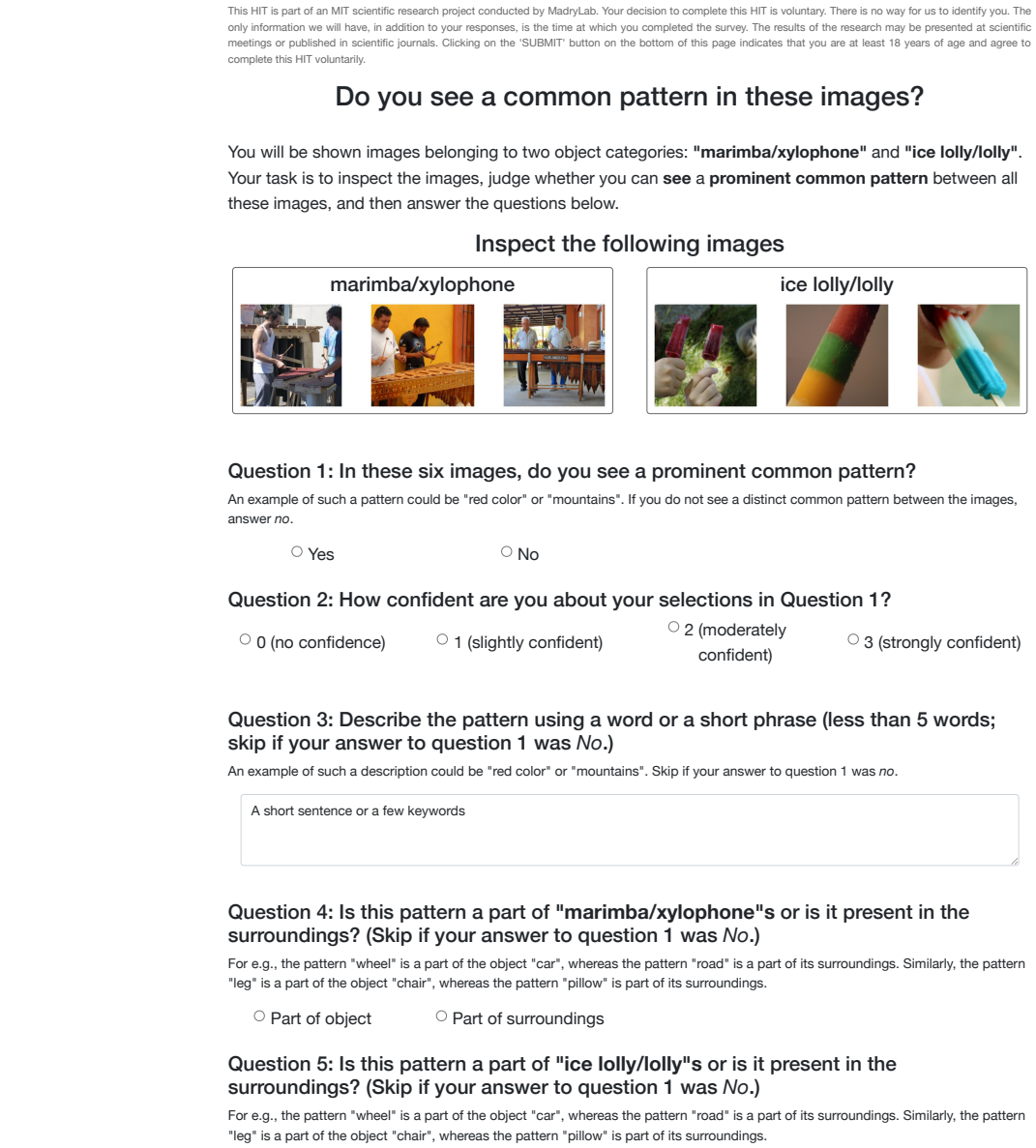


Figure 28: Sample MTurk task to diagnose (spurious) correlations in deep networks via their dense/sparse decision layers.

Quality control For each task, we aggregated results over all the annotators. While doing so, we eliminated individual instances where a particular annotator made no selections. We also completely eliminated instances corresponding to annotators who consistently (>80% of the time) left the task blank. Finally, while reporting our results, we only keep tasks for which we have selections from at least three (of five) annotators. We determine the final selection based on a majority vote over annotators, weighted by their confidence.

E.2.2. Additional visualizations of spurious correlations

In Figure 29, we provide additional examples of correlations detected using our MTurk study. Then in Figure 30, we summarize annotator-provided descriptions for all the patterns identified in ImageNet classifiers with sparse decision layers via a word cloud. This visualization sheds light into the nature of correlations extracted by ImageNet classifiers from their training data—for instance, we see that the patterns most frequently identified by annotators relate to object color and shape.

Figure 29: Additional examples of correlations in ImageNet models detected using our MTurk study. Each row contains protypical images from a pair of classes, along with the annotator-provided descriptions for the shared deep feature that these images strongly activate. For each class, we also display if annotators marked the feature to be a "spurious correlation".

Figure 30: Word cloud visualization of descriptions provided by annotators for patterns learned by "shared deep features" in standard ImageNet-trained ResNet-50 classifiers with sparse decision layers.

Algorithm 3 Counterfactual generation for a sentence of n words $x = (x_1, \dots x_n)$, a deep encoder $h : \mathbb{R}^n \to \mathbb{R}^m$, and a linear decision layer with coefficients (w, b).

$z = h(x)$ *// calculate deep features*
$y = \arg\max_y w_y z + b_y$ *// calculate prediction*
$Z^+, Z^- = \emptyset, \emptyset$ *// initialize candidate word substitutions*
for $i = 1 \dots m$ **do**
  for $j = 1 \dots n$ **do**
    if $x_j \in \text{WordCloud}^+(z_i) \wedge w_{yi} > 0$ **then**
      $Z^+ = Z^+ \cup \{(x_j, z_i)\}$ *// candidate word substitution with positive weight and positive activation*
    else if $x_j \in \text{WordCloud}^-(z_i) \wedge w_{yi} < 0$ **then**
      $Z^- = Z^- \cup \{(x_j, z_i)\}$ *// candidate word substitution with negative weight and negative activation*
    end if
  end for
end for
if $|Z^+ \cup Z^-| = 0$ **then**
  return-1 *// No overlapping words found for counterfactual generation*
end if
Randomly select $(x_j, z_i) \in Z^+ \cup Z^-$ *// select a random word to substitute and its corresponding feature*
if $(x_j, z_i) \in Z^+$ **then**
  Randomly select $\hat{x}_j \in \text{WordCloud}^-(z_i)$ *// if positive, select a random negative word*
else if $(x_j, z_i) \in Z^-$ **then**
  Randomly select $\hat{x}_j \in \text{WordCloud}^+(z_i)$ *// if negative, select a random positive word*
end if
$\hat{x} = (x_1, \dots, x_{j-1}, \hat{x}_j, x_{j+1}, \dots, x_n)$ *// perform word substitution*
return $\hat{x}$ *// return generated counterfactual*

F. Counterfactual experiments

F.1. Language counterfactuals

We describe in detail how to generate counterfactuals from the word cloud interpretations and the linear decision layer. The complete algorithm can be found in Algorithm 3, which we describe next.

Let $x = (x_1, \dots, x_n)$ be a sentence with n words, $z = f(s) \in \mathbb{R}^m$ be the deep encoding of x, and $y = \arg\max_y w_y z + b_y \in [k]$ be the model's prediction of x for a given decision layer with coefficients (w, b). Our goal is to generate a counterfactual that can flip the model's prediction y to some other class. Furthermore, let $\text{WordCloud}^+(z_i)$ and $\text{WordCloud}^-(z_i)$ be the LIME-based word clouds representing the positive and negative activations of ith deep feature, z_i. Then, counterfactual generation in the language setting involves the following steps:

1. Find all deep features which use words in x as evidence for the predicted label y (according to the word clouds). Specifically, calculate $Z = Z^- \cup Z^+$ where

$$Z^+ = \{(x_j, z_i) : \exists j \text{ s.t. } x_j \in \text{WordCloud}^+(z_i) \wedge w_{yi} > 0\} \tag{6}$$

$$Z^- = \{(x_j, z_i) : \exists j \text{ s.t. } x_j \in \text{WordCloud}^-(z_i) \wedge w_{yi} < 0\} \tag{7}$$

2. Randomly select a deep feature (and its word) $(x_j, z_i) \in Z$
3. If $z_i \in Z^+$, randomly select a word $\hat{x} \in \text{WordCloud}^-(z_i)$. Otherwise, if $z_i \in Z^-$, randomly select a word $\hat{x}_j \in \text{WordCloud}^+(z_i)$.
4. Perform the word substitution $x_j \to \hat{x}_j$ to get the counterfactual sentence, $\hat{x} = (x_0, \dots, x_{j-1}, \hat{x}_j, x_{j+1}, \dots, x_n)$.

Note that it is possible for there to be no features that use words in a given sentence as evidence for its prediction, which results in no candidate word substitutions (i.e. $\|Z\| = 0$). Consequently, it is possible for a sentence to have a counterfactual

generated from the dense decision layer but not in the sparse decision layer (or vice versa). For our sentiment counterfactual experiments, we restrict our analysis to sentences which have counterfactuals in both the sparse and dense decision layers. However, we found that similar results hold if one considers all possible counterfactuals for each individual model instead.

F.2. ImageNet counterfactuals

In Figure 31, we illustrate our pipeline for counterfactual image generation. Our starting point is a particular spurious correlation (between a data pattern and a target class) identified via the MTurk study in Section 4.1. We then select images from other ImageNet classes to add the spurious pattern to, and annotate the relevant region where it should be added. We obtain the spurious patterns by automatically scraping search engines. Finally, we combine the original images with the retrieved spurious pattern, using the mask as the weighting, to obtain the desired counterfactual images. These images are then supplied to the model, to test whether the addition of the spurious input pattern indeed fools the model into perceiving the counterfactuals as belonging to the target class.

G. Validating ImageNet misclassifications

G.1. Human study

We now detail the setup of our MTurk study from Section 3.3. For our analysis, we use a ResNet-50 that has been adversarially-trained ($\varepsilon = 3$) on the ImageNet dataset. To obtain a sparse decision layer, we then train a sequence of GLMs via elastic net (cf. Section 2.1) on the deep representation of this network. Based on a validation set, we choose a single sparse decision layer—with 57.65% test accuracy and 39.18 deep features/class on average.

Task setup. In this task, our goal is to understand if annotators can identify data patterns that are responsible for misclassifications. To this end, we start by identifying deep features that are strongly activated for misclassified inputs.

For any misclassified input x with ground truth label l and predicted class p, we can compute for every deep feature $f_i(x)$:

$$\gamma_i = W[p, i] \cdot f_i(x) - W[l, i] \cdot f_i(x) \tag{8}$$

where W is the weight matrix of the decision layer. Intuitively, this score measures the extent to which a deep feature contributes to the predicted class, relative to its contribution to the ground truth class. Then, sorting deep features based on decreasing/increasing values of this score, gives us a measure of how important each of them are for the predicted/ground truth label. Let us denote f_p as the deep feature with the highest score γ_i and f_l as the one with the lowest.

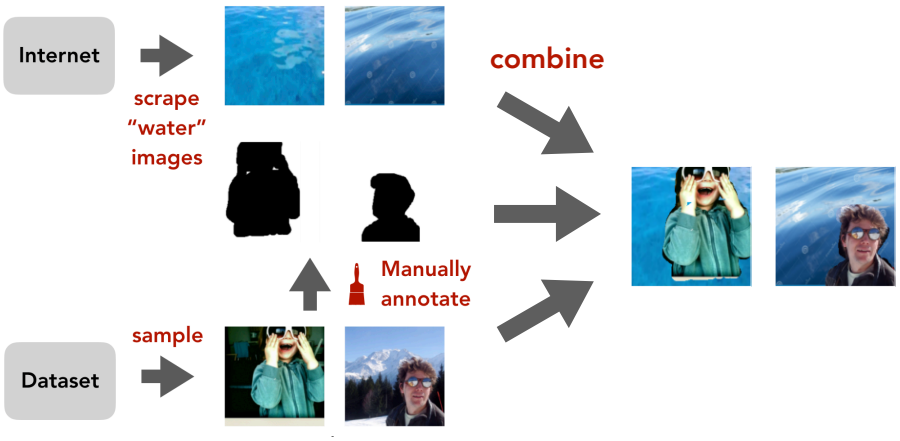


Figure 31: Image counterfactual generation process. We start with a correlation identified during our MTurk study in Section 4.1—for example, the model associates "water" with the class "snorkel". To generate the counterfactuals shown in Figure 9, we first select images from other ImageNet classes. We then manually annotate regions in these images to replaces with "water" bacgrounds obtained via automated image search on the Internet. Finally, we additively combine the "water" backgrounds and the original images, weighted by the mask, to obtain the resulting counterfactual inputs.

We find that for the robust ResNet-50 model with a sparse decision layer, the single top deep feature based on this score (f_p) alone is responsible for 26% of the misclassifications (5673 examples in all). That is, for each of these examples, simply turning $f_p = 0$ flips the model's prediction from p to l. We henceforth refer to these deep features (one per misclassified input) as "problematic" features.

For our task, we randomly subsample 1330 of the aforementioned 5673 misclassified inputs. We then construct MTurk tasks, wherein annotators are presented with one such input (without any information about the ground truth or predicted labels), along with the feature visualizations for two deep features. These two features are either (with equal probability):

- (f_l, f_p): The deep features which (relatively) contribute most to the ground truth and predicted class respectively.
- (f_l, f_r): The deep feature which (relatively) contributes most to the ground truth class, along with a randomly-chosen one (out of the 2048 possible deep features). This is meant to serve as a control.

Annotators are then asked: (a) to select all the patterns (i.e., feature visualization of a deep feature) that match the image; (b) to select the one that best matches the image (if they selected both in (a)); (c) to mark their confidence on a likert scale. A sample task is shown in Figure 32. Each task was presented to 5 annotators, compensated at $0.03 per task.

Note that, in the case where the ground truth label for each image is actually pertinent to it and that model relies on semantically-meaningful deep features for every class, we would expect annotators to select f_l to match the image 100% of the time. On the other hand, we would expect that annotators rarely select f_r to match the image.

Quality control For each task, we aggregated results over all the annotators. While doing so, we eliminated individual instances where a particular annotator made no selections. We also completely eliminated instances corresponding to annotators who consistently (>80% of the times) left the task blank. Finally, while reporting our results, we only keep tasks for which we have selections from at least two (of five) annotators. We determine the final selection based on a majority vote over annotators, weighted by their confidence.

G.2. Additional error visualizations

In Figure 33, we present additional examples of misclassifications for which annotators the top deep feature used by the sparse decision layer to detect the predicted class to be a better match for the image than the corresponding top feature for the ground truth class.

Identify the patterns that match the given image

Please inspect the image and patterns below, and answer the following questions.

Task 1: Select all the patterns that match the image shown on the left.

Please select at least one pattern. Select both patterns only in cases where you strongly believe that they are both visually similar to the image.

○ Matches image

○ Matches image

Task 2: Which of the two patterns matches the given image *better* according to you? (Answer only if you selected both patterns in Task 1)

○ Best match

○ Best match

Task 3: How confident are you about your selections?

○ 0 (no confidence) ○ 1 (slightly confident) ○ 2 (moderately confident) ○ 3 (strongly confident)

Figure 32: Sample MTurk task to identify input patterns responsible for the misclassifications in deep networks with the help of their (sparse) decision layers.

Figure 33: Additional examples of misclassified ImageNet images for which annotators deem the top activated feature for the predicted class (*rightmost*) as a better match than the top activated feature for the ground truth class (*middle*).

G.3. Model confusion

In Figure 34, we visualize the correlation between model confusion within a pair of classes, and the number of shared features between them in the sparse decision layer. Model confusion within a class pair (i, j) is measured as $\max(C_{(i,j)}, C_{(j,i)})$, where C is the overall confusion matrix. We find that for models with sparse decision layers, the feature overlap between two classes, is significantly correlated with model errors within that class pair. One can thus inspect the corresponding shared features—cf. Figure 35 for an example—to better understand the underlying causes for inter-class model confusion.

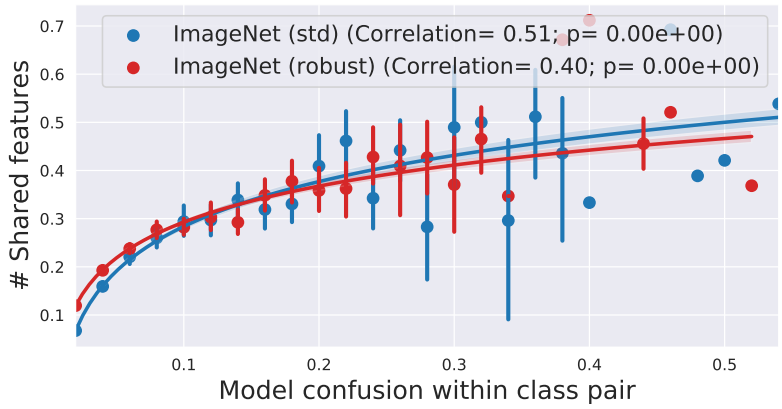


Figure 34: Correlation between the number of features shared in the sparse decision layer of a model for two classes, and model confusion between them. Model confusion within a class pair is measured as the maximum of the corresponding entries ($C_{(i,j)}$, $C_{(j,i)}$) of the overall confusion matrix.

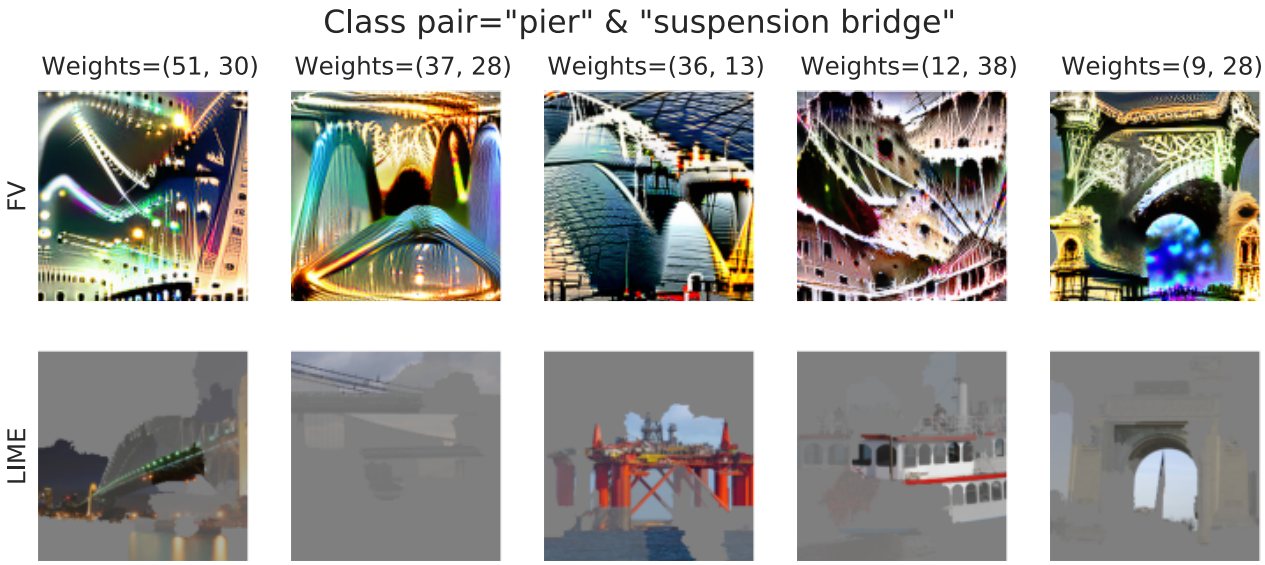


Figure 35: Sample visualization of confusing features: Five of the deep features used by a robust ($\varepsilon = 3$) ImageNet-trained ResNet-50 with sparse decision layer to identify objects of classes “pier” and “suspension bridge” which are frequently confused by the model ($C_{i,j}$ and $C_{j,i}$ are 16% and 24% respectively). Each of these deep features is interpreted using feature visualizations (FV) and LIME superpixels; shown alongside their linear coefficients (W).