Decomposing the deep: finding class-specific filters in deep CNNs

Akshay Badola¹ • Cherian Roy¹ • Vineet Padmanabhan¹ • Rajendra Prasad Lal¹

Received: 30 May 2022 / Accepted: 24 February 2023 / Published online: 15 March 2023 © The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2023

Abstract
Interpretability of Deep Neural Networks has become a major area of exploration. Although these networks have achieved state-of-the-art results in many tasks, it is extremely difficult to interpret and explain their decisions. In this work, we analyze the final and penultimate layers of Deep Convolutional Networks for image classification with respect to $\ell_1$ norm and develop an algorithm for identifying subsets of features that contribute most toward the network’s decision for each class. We also develop a novel decomposed softmax to efficiently re-train the network such that the class-specific decomposition is preserved. We provide a comparison with other methods for identifying class-specific filters and show, using Pairwise Mutual Information Score, that our technique provides better decomposition. The resulting decomposed final layer provides a low-dimensional embedding (decreased by around a factor of 10) per class, which is far more interpretable. Such a low-dimensional class-specific embedding makes diagnosing issues with misclassifications of a certain class in the data easier as fewer weights contribute to the decision for the data points of that class. It also enables the network toward easier diagnostics and pruning as an entire part of the final and pre-final layer can be excluded to remove predictions for data points belonging to a particular label. The resulting layer also achieves a modest computational cost gain as compared to the final layer of the full network. Our algorithm is unsupervised in nature and can be applied to any CNN.

Keywords Deep learning • Convolutional neural networks • Interpretable CNNs • Disentanglement

1 Introduction

Deep Neural Networks have been a paradigm shift in machine learning, but they have remained primarily black boxes. Large networks can contain millions [1, 2] to billions [3, 4] of parameters. Such models are highly over-parameterized and in fact, overparametrization is essential to their generalization ability [5–7]. In such cases, it becomes extremely difficult to attribute the prediction of any network to its parameters which leads to poor understanding and interpretability of the network. The lack of interpretability of these models makes it very difficult for humans to trust them in critical situations, like medical diagnosis or autonomous vehicles. It also becomes very hard to diagnose and correct the models themselves.

The final layers¹ of a CNN hold particular interest for interpretability, as detectors that appear there are more aligned with semantic concepts. The final layer of the network projects the features via softmax onto the decision simplex and its properties are critical to the label predicted by the network.

While there has been recent work in interpreting and disentangling neural network features, in the current literature, we have not come across an efficient, unsupervised method to identify and rank the features in the final and penultimate layers in a class-specific manner. In this work, we focus on extracting such features from the final and

¹ When we say final layer(s) we mean the layers closer to the classifying layer. Following previous works we also mention them as top layer(s) and the layers closer to input as bottom layer(s).
penultimate layers of some popular CNN architectures, namely Resnet [2], Densenet [8], and Efficientnets [9]. As decomposing the network leads to a drop in accuracy, we also develop a novel decomposed softmax to retrain the network while preserving the class-specific subspace separation in the final layers. We also provide the implementation of our experiments.\(^2\)

2 Neural network interpretability

Zhang et al. [10] define interpretability as an “ability to provide explanations in understandable terms to a human”. We, however, leave it to humans to develop the terms necessary for communication and instead, define interpretability as a representation of a model amenable to inspection, adaptation, and attribution.

An interpretable model, and specifically an interpretable Deep Neural model in our case, should be easy to inspect, easy to adapt, and easy to attribute. In particular, it should facilitate:

1. Input inspection and attribution: An interpretable model should allow correspondence between parts of an input to the labels. The correspondence may be simplified for this purpose. For example, in an image, parts of an object which contribute more toward the classification could be filtered for inspection and attribution.
2. Feature attribution: An interpretable model should allow correspondence between intermediate representations of an input to the labels.
3. Parameter attribution: An interpretable model should allow correspondence between the parameters of the model and the input-label decision-making process. That is, it should be evident that which parameters act on which parts of inputs to produce which labels. Again, the number of such parameters, and inputs-parts that would be attributed, could be reduced for simplification.

For any model, interpretation should result in some attribution from the parameters of the model toward the predictions made by it. This is separate from distilling a network into a more explainable model like a decision tree [11], which aims more toward converting the model to a simpler one; while allowing for significant accuracy loss. Model interpretation (and consequent simplification), however, does not attempt to significantly alter the model and largely attempts to maintain the model’s accuracy.

In the context of CNNs, we can look to interpret them based on parts of images and the filter banks acting on them, as deep CNNs contain banks of convolutional filters stacked one after the other. These filters, which provide a locally linear weighted average of the image signal, can be considered for interpretability. As, instead of individual weights, they act in tandem on the input and intermediate representations. We also note that while our method has been developed for CNNs, it can also be applied, in theory, to any network that uses softmax for classification. We leave that to future work however.

3 Related work

There is sufficient prior work on inspecting and transferring the filters in a Deep CNN. A detailed survey is given in [10]. Earlier works like [12–14] analyzed the filters in intermediate layers and found that the features from pre-trained neural networks can be used for other tasks.

Later work has moved more toward class attribution and aligning features to semantic concepts. Gonzalez-Garcia et al. [15] and Bau et al. [16] analyze the activation maps of CNNs and discover that the final layers of the networks align closer to semantic concepts than the early layers which were more attuned to detecting texture and color.

Network Dissection [16] attempts to associate the activation maps of CNNs with semantic concepts. Bach et al. [17] introduce the concept of Relevance Propagation and visualize the contributions of individual pixels in an image. Hendricks et al. [18] use a Reinforcement Learning-based loss function to provide natural language explanations of the model’s predictions. Guillaume and Bengio [19] integrate separate linear classifiers at each layer to understand the model’s decision-making process. Zhang et al. [20] build templates specific to parts and incorporate that during the training of the CNN to generate interpretable feature maps. Liang et al. [21] use sparsity to find out the important filters in a CNN, though, instead of going toward a parameter-level sparsity like dropout or quantization [22], they look at one Convolutional Filter as a single parameter. Some other approaches like [23] and [24] attempt to estimate the contributions of the filters via modeling them as a graphical model. The concept of Class Activation Maps is described in [25, 26], who try to identify discriminative regions of CNNs, while [27] and [28] aim for linguistic descriptions of model explanations.

A parallel line of investigation has studied feature importance in CNNs and sparse CNNs. Liu et al. [29] use sparse low-rank decompositions in pre-trained CNNs. Li et al. [30] use \(\ell_1\) norm of filters to rank the convolutional filters as a whole and prune the less important ones. Kumar et al. [31] also use \(\ell_1\) norm with a capped \(\ell_1\) norm to formulate classification with CNNs as a lasso-like problem. Lin et al. [32] introduce a structured sparsity.
regularization. Li et al. \cite{33} introduce a kernel sparsity and entropy (KSE) measure, which quantifies both sparsity and diversity of the convolution kernels. Yin et al. \cite{34} perform decomposition from a compression perspective. Li et al. \cite{35} instead perform random channel pruning.

The notion of class specificity is explored in Wang et al. \cite{36}. They try to increase accuracy with filters that capture class-specific discriminative patches. These class-specific filters are closely related to earlier work by Jiang et al. \cite{37}, who introduce the concept of Label Consistent Neural Networks to learn features that they claim alleviate gradient vanishing and leads to faster convergence. Liang et al. \cite{21} try to use sparsity to find out the important filters which they claim are also class specific.

Our work takes inspiration from \cite{30,36} and \cite{21}. Like \cite{30} and \cite{21} we used $\ell_1$ norm to identify the filters which contribute more to a network’s decision making, but instead of a whole network, we focus on class-specific features and only on a final layer. \cite{30} do not identify class-specific filters. \cite{21} do identify class-specific filters but they use an $\ell_1$ norm-based importance matrix, which they train in a supervised manner. \cite{36} attempt to increase the accuracy by identifying class-specific discriminative patches. We combine the ideas of class specificity from \cite{21,36} and $\ell_1$ norm-based filter importance \cite{30} to arrive at an unsupervised method for identifying the $k$ most influential features based on an $\ell_1$ norm importance criterion in the final and penultimate layers of a CNN and demonstrate its efficacy with experiments on various CNNs. Like \cite{36} we re-train the network to recover the original accuracy. For re-training we develop a novel decomposed softmax which instead of acting on the individual output nodes, acts on the sum of class-specific nodes. This also provides a modest computational gain while doing inference.

### 3.1 Our contributions

Our contributions are following:

1. We show that only a few filters per class are needed to make a decision for a deep CNN.
2. We provide an algorithm to obtain those filters from any pre-trained network with a single fully connected layer.
3. We demonstrate the relation between depth and filter disentanglement in CNNs and show that deeper networks lead to lower dimensional representations in the final layer.
4. We show how to retrain the decomposed network to recover the original accuracy with a decomposed softmax.
5. We show that these filters have a greater correspondence with objects within the image and are critical to classification for that class.

The rest of the paper is organized as follows: We discuss Notation and CNNs in Sect. 4 and then move on to the concept of class-specific features. We provide an overview of the analysis of techniques on final layer of CNNs and class-specific features therein, in Sect. 5. We describe the experimental details and results in Sect. 6. We discuss implications and future scope in Sect. 7.

### 4 Key concepts

We discuss CNNs, filters, and layers in the next sections and establish notation which will aid us in our describing our methods.

#### 4.1 Notation

Let $\{(X, Y)\}$ be the set of data tuples. An instance of the data $(X, y) \in \{(X, Y)\}$ is a tuple of (image, label) with $X \in \mathbb{R}^{c \times h \times w}$ where $c, h, w$ are the channels, height, and width of the image, respectively. $y$ is an integer response variable representing one of the $n$ classes, $y \in \mathbb{Z}^+, 0 \leq y < n$. $y$ can also be represented as an index vector indexing the $i$th class, $y \in \{0, 1\}^n : y = i \equiv \{0, 0, ..., 1, ..., 0\}$.

Formally a Deep Neural Network is a set of weights that act as a sequence of operators on input $X$, such that, $y = A W^d (...) A W^1 (A W^0 (X))$ where $W^d$ is a weight tensor at depth $d$ and $A$ is an element-wise function also known as an activation function. While the activation function $A$ need not be same for each weight $W^d$, for CNNs that we consider only RELU max(0,$x$) is used. An exception is the weights at the end of the network where a softmax $\sigma(x) = \exp(x_i) / \sum \exp(x_j)$ is used.

A layer of a network is a single weight+activation operation $O^d = A(W^d (I^d))$ where $I^d, W^d, O^d$ are input, weights and output at depth $d$. We will denote the layer at depth $d$ as $L^d$.

#### 4.1.1 CNNs

A Convolutional Network consists of filter banks of convolution (or cross-correlation) filters which are square matrices of odd rank acting on the input with an element-wise activation function on the output. Such an output $O^d$ at layer $d$ is called a feature map at $d$th layer. Filters are the fundamental unit for a CNN, and it is convenient to represent and analyze a CNN as operations on the input by the
filters. For our purposes, we will denote the $j$th filter for layer $i$ as $L^i_j$.

Modern Deep CNNs rely heavily on RELU and Batch Normalization [38] in the intermediate layers. Batch Normalization and RELU are performed after the Convolution operator and the entire unit can be considered as a single operator. A convolution operation of a filter $w$ of size $k \times k$ on an input image matrix $I$ of size $h \times w$ is defined as:

$$\text{Conv}_{w,k,k}: I \rightarrow O \quad \text{where} \quad O_{i,j} = \sum_{l=-k'}^{k'} I_{i-l-k,j-k} w_{k-l,k-1}$$

$k'$ is $\lfloor k/2 \rfloor$.

Let $L^d$ be the input at the final layer of the CNN with total depth $d$. The output at the final layer is then $O^d = \sigma(W^d)$, where $\sigma$ is the softmax operator. $\sum_j O^d_{i,j} = 1$ because of softmax, and thus it can be interpreted as a probability distribution over $y$. The probability of each class $y_i$ is given by $O^d_{i,j}$ and the most probable label $y$ is given by $\hat{y} = \text{argmax}(O^d)$. See Fig. 1 for an illustration.

For further details, we refer the reader to [1, 39] and [2] for internals of CNNs.

### 4.2 Filter disentanglement and label consistency

As mentioned earlier, the sheer number of features and layers in CNNs makes them very hard to interpret. One approach toward their interpretation is Filter Disentanglement. Filter Disentanglement refers to the fact that the filters in a CNN should represent separate concepts. As concepts in a network are hard to identify, we can instead use the attribution of a filter toward the prediction of a label as a surrogate measure of disentanglement. Ideally every filter should be responsible for the detection of a particular pattern in the image, but that is difficult in practice. In particular, at the bottom layers the filters are highly entangled and learn very generic features [16]. As we move up the network, at the top layers, they tend to be less entangled but not entirely so.

As in Sect. 4.1.1, let the set of filters for a CNN at depth $d$ constitute the layer $L^d$. Now consider a single label $y \in \mathcal{Y}$. Let $\exists L^d_{J_y} \subset L^d$ where $J_y$ is an index set such that, $L^d_{J_y}$ alone is responsible for the prediction of label $y$.

For such a set $J_y$, we define the influence relation $\prec$ for the probability $P(y^d)$ of a label $y$ at depth $d$ of a CNN as: $P(y^d) \prec L^d_{J_y}$. We say that the features $L^d_{J_y}$ are influential at depth $d$ for the prediction of label $y$.

We will focus only on the final layer so we can remove the superscript $d$ without ambiguity. To enforce disentanglement and considering only the final layer we should ideally have,

1. $|L_{J_y}| \ll |L|, \forall y$ and,
2. $L_{J_{y_1}} \cap L_{J_{y_2}} = \emptyset, \forall y_1, y_2 \in \mathcal{Y}$

That is, the filters at depth $d$ for each class $y$ should be disjoint. However, as we mentioned above, that is not practically feasible. Instead, we can hope to find:

1. $|L_{J_y}| \ll |L|, \forall y$
2. $\arg\max_{J_{y_1} \cap J_{y_2}} (L_{J_{y_1}} \cap L_{J_{y_2}}), \forall y_1, y_2 \in \mathcal{Y}$

Or we should seek index sets $J_y$ such that there is minimal overlap between the two classes. These class-specific filters are closely related to [37], except that [37] associate a neuron (or weight) with a label while we associate filters. The class-specific filters discussed in Liang et al. [21] are more similar to these. However, unlike [21] and [36] our method does not require supervision and can be used for any pre-trained network. And, although we use $\ell_1$ norm to
identify filters, our method identifies class-specific filters and decomposes the final layer while the method in [30] does not.

5 Influential features

The penultimate and final layers are of particular interest in feature attribution in CNNs as they contain the final representation of the image and directly lead to the prediction of the label, while the lower layers learn the filters which respond to textures and lower-level features [16]. Here we describe the various factors in determining the influential features in the final layer.

5.1 $\ell_1$ Norm for feature importance

Previously [30] and [21] have both used $\ell_1$ norm for estimating filter importance. However, earlier works such as the two above, do not explicitly discover a lower dimensional decision surface. Following [30] and [21] we look at the $\ell_1$ norm of the resulting features per class as the primary differentiating factor.

One approach would be to check the $\ell_1$ norm of weights in the final and pre-final layers, which is similar to [21]. However, we can note that the $\ell_1$ norm of the weights will not vary for each class, so it is much harder to identify class-specific features with that. In our initial experiments also, this hypothesis was verified.

Another key observation we make is that all the features in the entire CNN are $\geq 0$ because of RELU. In effect, the feature vectors/tensors are positive semidefinite, which means that the $\ell_1$ norm of each feature directly contributes to the classification output of the final layer. The weights, on the other hand are roughly $50\% \geq 0$ and $50\% \leq 0$ and therefore, it is easier to quantify feature importance than weight importance with $\ell_1$ norm.

5.2 Separating outputs for each class

A second insight we had is that the final layer output for each class is the result of a dot product between features for that class and weights for that class. Therefore the final layer can be thought of as $c$ separate dot products, where $c = |Y|$ as earlier is the number of classes. Hence, selecting certain features for each class will not affect the output for the other.

Formally, $P(y|X) = O^d = \sigma(WIt^d)$ as in Sect. 4.2, where $O^d \in \mathbb{R}^c$ and $\sum_i O^d_i = 1$. If we omit $d$ and $X$ for simplicity, then probability for $ith$ class, $P(y = i)$ is the value of $ith$ component of the output from the final layer, which is $O_i$.

Now, the final layer is a single matrix, so $O_i$ is simply the dot product of $ith$ row of the weight matrix $W$ with the input feature vector $I$. If $m$ is the width of the final layer, then $O_i = W_i \cdot I$, where $W \in \mathbb{R}^{m \times n}$ and $W, I \in \mathbb{R}^m$.

Now, let $w_{ki}$ be the $k$ dimensional subspace of $W_i$ for the index $i$, that is, $w_{ki} \subseteq W_i$. $w_{ki} \in \mathbb{R}^k, k \ll m$. Then we define the probability for label $i$ at final layer with reduced dimension $k$ as $P(y^k = i) = \sum w_{ki}I_k$.

Recall that the predicted label with the full width is $\hat{y} = \text{argmax}(y)$. We define a prediction with reduced dimension $k$ as $\hat{y}_k = \text{argmax}(y_k)$. The goal then is to find such $w_{ki}$ for each class $i$, such that the difference in the predictions $d(\hat{y}_k, \hat{y})$ is minimized, where $d$ is some metric.

That is, $w_k$ are the truncated weights which minimize the difference between the predicted labels at width $k$ and predicted labels at full width $n$ for each class.

5.3 Finding the influential features

We note that finding the class-specific influential features $w_{ki}$ is non-trivial, as the exact subset of the weights cannot be known easily. An exhaustive search for this task would be of exponential complexity. However, as we mention in Sect. 5.1 $\ell_1$ norm of the features can help us in guiding toward the correct set of filters.

In our experiments, we found that although selecting weights by top $\ell_1$ norm would result in weights attribution, it would not result in class-specific features attribution. Instead, searching for topk features per class with $\ell_1$ norm gave us better results. Even combining topk filters per class with topk weights led to poorer results than with topk features per class.

A problem though is that the features per instance vary across a single class and the label of the instance cannot be known in advance. However, a topk selection from histogram of topk features for all instances in a given class had mass concentrated around a few points, which corresponded to such influential features. We provide the details of the algorithm in the next section.

5.4 Algorithm

Here we discuss the algorithm to obtain the $k$ most influential features for each class from a pre-trained CNN. For the algorithm 1 below, $I$ are the set of features at the final layer for all classes, where we omit the layer superscript used earlier for simplicity. Denote the set of features for label $y$ by $I_y$. We want to obtain the mapping $\mathbb{I} : y \rightarrow I_y$, such that, $\mathbb{I}(y)$ gives the most influential features for class $y$. The parameters for the algorithm are $k_1, k_2 \in \mathbb{Z}^+$ which...
define the initial and subsequent feature selection, which we describe below.

We proceed by noting top $k$ features by $\ell_1$ norm of each data instance for each class at the pre-final layer and select the indices in a set $I_{y}^{topk}$ for class $y$. We set $k \ll m$ where $m$ is the dimension of the features at the pre-final layer (and hence, also the dimension of the final layer). E.g., if for a 64-dimensional feature vector, let the top five components sorted by $\ell_1$ norm occur at indices $\langle 3, 5, 14, 18, 28 \rangle$. Then the index set for $k = 5$ for the entire class is aggregation $\bigcup I_{y}^{topk}$ of all such sets, where $\bigcup$ is an aggregation operator, e.g., $\bigcup \langle 3, 5 \rangle < \bigcup \langle 3, 14 \rangle = \langle 3, 5, 14 \rangle$.

The set $\bigcup I_{y}^{topk}$ then denotes all the occurrences of a particular dimension (or index) of a feature in top $k$ $\ell_1$ norm set, for a class in the pre-final layer. The frequency distribution for one such class for ResNet20 is given in Fig. 2. From that histogram, we then select the top $k_2$ most frequent indices.

We also experimented with $k_1$, which covers a certain percentage (say 90) of the contribution of the filters. However, we found that a value of 5 for CIFAR-10 [40] gave us 90 coverage, which we then chose to set for all our experiments as that is faster to implement.

For Imagenet [41], the number of filters that contributed over 90 norm was very large, and we restricted ourselves to top-50. Even though the initial accuracy after decomposition is lower, we were able to retrain the network back close within 1 of the original accuracy.

### Algorithm 1 Extract Influential Features

1. **Input:** $\mathcal{I}$, $\mathcal{I}_y$, $k_1$, $k_2$
2. **Output:** Mapping $\mathcal{I}_y$ of Influential Features
3. **procedure** `GET_INDICES($\mathcal{I}$, $y$)`
   4. **for** $I_y \leftarrow \mathcal{I}$ **do**
   5. **for** $I_y \leftarrow \mathcal{I}_y$ **do**
   6. $I_{y}^{topk_1} = I_{y}^{topk_1} \bigcup \text{sorted}_k(y)$ w.r.t. $\ell_1$ norm
   7. **end for**
   8. $I_y \leftarrow \topk_2 (\text{HIST}(I_{y}^{topk_1}))$
   9. $\mathcal{I}_y = \mathcal{I} \bigcup (y, I_y)$
   10. **end for**
11. **end procedure**

HIST in algorithm 1 refers to the histogram of frequencies of each index.

### 6 Experiments and results

We conducted some initial experiments on Class-Specific Gates (CSG) [21] and we discuss them in 6.1 as to illustrate our argument. We then analyze and discuss the efficacy of our proposed influential features in 6.2. As mentioned earlier, we take inspiration from [30] and [21] and combine class specificity with $\ell_1$ norm importance. We note that while [21] is a supervised method, their approach is closest to ours in principle. [30] is unsupervised, but their aim is to induce sparsity and not interpretability. Also, their approach does not lead to explicit class-specific disentanglement. We demonstrate that our unsupervised approach gives: (a) as good if not better results than [21] (b) leads to
explicit filter disentanglement (c) reduces computational cost in the final layer.

6.1 Class-specific filters

We first discuss experiments with class-specific gates (CSG). Liang et al. [21] proposed to learn a matrix CSG $\in \mathbb{R}^{m \times n}$ where has $m$ is the dimension of final layer and $c = |\mathcal{X}|$ is the number of classes. See [21] for details of the training procedure. One important issue to consider with [21] is that the label CSG matrix would not be available when performing inference as the labels for new data are not known. Hence, it can only be used to interpret a CNN on a given dataset and cannot be used for any new data.

While [21] have released part of the code for their experiments, they did not release the CSG learning code, especially CSG for CIFAR-10 and Imagenet.\(^3\) We implement CSG and summarize the results in Table 1.

STD output in Table 1 is $O_{STD}^c = \sigma(W^c)$, which is the same as CNN output without a CSG matrix. Output using CSG matrix is $O_{CSG}^c = \sigma(W(F \odot \text{CSG}))$, where $\odot$ denotes the Hadamard or element-wise product. We note that the CSG accuracy is similar to the STD accuracy, CSG and STD are CSG and STD output paths, respectively.

6.2 Influential features

We have performed experiments to obtain influential filters according to Algorithm 1 on families of models a) Resnet b) Densenet c) Efficientnet to demonstrate that our method can work for any CNN. We have used standard datasets CIFAR-10 [40] and Imagenet [41] in our experiments.

Resnet [2] are the most popular variants of CNNs because of their lower computational cost and good generalization in spite of it. Densenets [8] are another popular architecture which connect each layer to every other layer. Efficientnets [9] are recent models which are developed with Neural Architecture Search [42] and aim to reduce the computational cost while preserving accuracy. The reader is referred to the respective papers for the details of the models.

Resnet family of models is our primary focus for retraining, visualization, and criticality experiments. We calculate and compare influential features for all the other models.

CIFAR-10 consists of 50,000 training images for ten classes, i.e., $c = |\mathcal{X}| = 10$, of size $32 \times 32$ and 10,000 validation images. Imagenet has 1,000,000 images of varying sizes with $c = 1000$ but while training they are resized to $224 \times 224$. The validation split has 50,000 images for Imagenet. For both the datasets and all the models we determine $k_1, k_2$ and $|\mathcal{X}|^{\text{top}}$ only from the training set. The results are then calculated on the validation set.

6.2.1 Effect of $k_1$ and $k_2$

We conduct a detailed study on Resnet20 model and CIFAR-10 dataset to analyze the effect of values of $k_1$ and $k_2$ on the resulting model. For the Resnet20 model, our pre-trained model had an accuracy of 91.17. We measure the efficacy of the resulting decomposed final layer as the ratio $r_A = \frac{A_f}{A_d}$, where $A_d, A_f$ are the decomposed and full-width accuracies, respectively. See Fig 3 for the results.

We observe overall, that reducing the dimensionality by lowering the number of input features to the final layer also results in a reduction of accuracy. This is to be expected as we are losing information at the last layer. We note that even for $k_2 = 3$, we get 84.16 accuracy which is pretty good for only three filters per class. This confirms that the decision surface lies on a much lower dimensional subspace than the dimension of the final layer.

Another thing we can note is that for a given value of $k_1$ the best results are obtained by setting $k_2 = k_1$. Also, if the initial filter selection $k_1$ is very large, then the equivalent $k_2$ setting leads to lower performance compared to a lower $k_1$.

6.2.2 Effect of depth

Here we discuss the results of experiments on various other models including variants of Resnet. For all the following experiments, we use Imagenet Dataset and set $k_1 = k_2 = 50$. Two important metrics in experiments with depth are $k_2/n$ and number of filters per class $k_2/c$. For all variants of Resnet $k_2/n, k_2/c$ and dimension of final layer $n$ remain the same. Table 2 summarizes the results for Resnet models.

We can see that the relative accuracy $r_A$ drops for Imagenet in these variants compared to Resnet20, but the number of filters per class $k_2/c$ is also lower at 0.05 as compared to 0.5 for Resnet20 which is an order of magnitude. $k_2/n$ is also lower at 0.024 as compared to 0.078. This is due to the much higher number of classes in the Imagenet dataset. Apart from that the effect of $r_A$ on depth is clear as it increases monotonically with increasing depth.

We also evaluate on Wide Resnets [43] which contain a greater number of convolution filters per layer as compared to standard Resnet models. We get a much higher relative accuracy $r_A$ on these as compared to standard Resnets for the same number of layers (see Table 3). We suspect that the greater number of convolutional filters help in

\(^3\) See https://github.com/hyliang96/CSCGCNN.
disentanglement as there are more filters per class in the previous layers.

For all the models we can see that across a class of models $r_A$ increases with depth. Only EfficientNets deviate from this behavior which we discuss later. Densnets have different $n$ and hence $k_2/n$ differs for them. For Densnets we compare Densenet121 with Densenet169 and Densenet161 with Densenet201, as the two pairs have closer $n$ and hence $k_2/n$.

Coming to Densnets, we can see a similar trend of increasing $r_A$ with depth (Tables 4, 5). The $r_A$ here is comparable to that of Resnets but not Wide Resnet variants. The final layer dimension $n$ also differs for Densnets, so we compare models with a similar $n$. We can see that $r_A$ tends to increase with depth which shows the effect of depth on filter disentaglement. See Fig. 4 for a summary of $r_A$ on all models.

Only with Effcientnets (Table 6), which are NAS-based models, do we see a deviation from this pattern as their model architecture differs from human designed networks. With Efficientnets, as the number of layers increases from 82 to 116 we see a corresponding jump in $r_A$ except Efficientnet_b2 which has same number of layers as Efficientnet_b1 but has a wider final layer, we see a drop in $r_A$.  

---

### Table 1 CSG experiments conducted by us

| Dataset   | Model   | Training | Accuracy |
|-----------|---------|----------|----------|
| Resnet20  | CSG     | 0.8809   |
| STD       | 0.8809  |
| CIFAR-10  | Resnet34| CSG      | 0.8407   |
| STD       | 0.8407  |
| Tiny      | Resnet18| CSG      | 0.3633   |
| STD       | 0.3641  |
| Imagenet  | Resnet34| CSG      | 0.3794   |
| STD       | 0.3852  |

### Table 2 Effect of depth on resnet variants

|                    | Resnet50 | Resnet101 | Resnet152 |
|--------------------|----------|-----------|-----------|
| Final dim $n$      | 2048     |           |           |
| $k_2/c$            | 0.05     |           |           |
| $k_2/n$            | 0.0244141|           |           |
| $A_d$              | 0.64792  | 0.68164   | 0.69346   |
| $A_f$              | 0.74548  | 0.75986   | 0.77014   |
| $r_A$              | 0.869131 | 0.89706   | 0.900434  |

### Table 3 Effect of depth on wide resnet

|                           | Wide Resnet 50 | Wide Resnet 101 |
|---------------------------|----------------|-----------------|
| Final dim $n$             | 2048           |                 |
| $k_2/c$                   | 0.05           |                 |
| $k_2/n$                   | 0.0244141      |                 |
| $A_d$                     | 0.75008        | 0.75988         |
| $A_f$                     | 0.77256        | 0.77908         |
| $r_A$                     | 0.970902       | 0.975356        |

### Table 4 Effect of depth on densenet 121 and 169

|                   | Densenet121   | Densenet169     |
|--------------------|---------------|-----------------|
| Final dim $n$      | 2048          |                 |
| $k_2/c$            | 0.05          |                 |
| $k_2/n$            | 0.0488281     | 0.0300481       |
| $A_d$              | 0.6372        | 0.66328         |
| $A_f$              | 0.71956       | 0.73754         |
| $r_A$              | 0.885541      | 0.899314        |

### Table 5 Effect of depth on densenet 161 and 201

|                 | Densenet161   | Densenet201     |
|-----------------|---------------|-----------------|
| Final dim $n$   | 2048          |                 |
| $k_2/c$         | 0.05          |                 |
| $k_2/n$         | 0.0226449     | 0.0260417       |
| $A_d$           | 0.6766        | 0.68214         |
| $A_f$           | 0.75268       | 0.7455          |
| $r_A$           | 0.898921      | 0.91501         |

---

Fig. 3  Effect of $k_1$ and $k_2$ on Accuracies for Resnet20 and CIFAR-10
EfficientNet_b3 is a curiosity as that model is both deeper and wider.

### 6.2.3 Decomposition and prediction with influential features

The computational cost of predicting with influential features is also less, as only a few filters are needed for predicting a class. Because of the identification of explicit class-specific filters we can decompose the final layer into class-specific subspaces. The network can then be retrained to recover the original accuracy. For Resnet20 and CIFAR-10, the accuracy drops to 88.59 with influential features with $k_1 = k_2 = 5$. After that we can decompose the final layer so that each class has a separate subspace of features. After training for one epoch, we get an accuracy of 88.29%.

![Diagram](image1.png)

**Fig. 4** Effect of number of layers (depth) of a network with Relative Accuracy $r_A$. We can see that $r_A$ increases with depth for a class of models.

| Efficientnet_b0 | Efficientnet_b1 | Efficientnet_b2 | Efficientnet_b3 |
|-----------------|-----------------|-----------------|-----------------|
| Final dim $n$   | 1280            | 1280            | 1408            | 1536            |
| Num layers      | 82              | 116             | 116             | 131             |
| $k_2/c$         | 0.05            | 0.05            | 0.05            | 0.05            |
| $k_2/n$         | 0.0390625       | 0.0390625       | 0.0355114       | 0.0325521       |
| $A_d$           | 0.64838         | 0.7046          | 0.6778          | 0.64842         |
| $A_f$           | 0.7609          | 0.76392         | 0.76762         | 0.76928         |
| $r_A$           | 0.852122        | 0.922348        | 0.882989        | 0.842892        |

**Fig. 5** Prediction with influential features for class $y_3, y_4$. Only 4 and 3 input features are required, respectively. The final layer can be decomposed as shown, and the probability for label can be $\sigma(I_{inf} \odot W_{inf})$.

EfficientNet_b3 is a curiosity as that model is both deeper and wider.

### Table 6 Effect of depth on efficientnet variants

Output for a fully connected label $y_3$. Sparse outputs from Influential Inputs for labels $y_1, y_4$.

Separable outputs from Influential Inputs for labels $y_1, y_4$. Unused weights are not shown.
of 91.51. An illustration of how efficient prediction with influential features works is given in Fig. 5.

For Resnet50 with Imagenet, after decomposition, the accuracy is 64.7. Unlike CIFAR-10, it requires around 10 epochs to get close to the original accuracy of the pre-trained Resnet50, which we get at 73.91.

The computational complexity for the final layer is \( \frac{d}{C^2} n \), where \( d \) is the dimension of the final layer and \( n \) is the number of classes. After using only influential features, it drops significantly. For example, for Resnet20 and CIFAR-10, the computational complexity for the final layer is \( \frac{64}{C^2} 10 \), which drops to \( \frac{5}{C^2} 10 \) with influential features.

A detailed schematic of training and inference with influential features is given in 6.

### 6.2.4 Decomposed softmax

In a standard CNN, for a given data input \( x \in \mathcal{X} \), the prediction is given as \( \sigma(W^d) \) where \( I^d \) is the input at depth \( d \) and \( W \in \mathbb{R}^{m \times n} \). For the input, the output is a floating point representation normalized by the softmax. The probability for each \( y_i \) is:

\[
P(y_i|x) \sim \sum_j W_{ij} x_j \equiv <W_i x>,
\]

and the predicted class is:

\[
P(\hat{y} = y_i|x) \sim \arg\max_i <W_i x>
\]

For predicting with influential features the probabilities are computed as:

\[
P(\hat{y} = y_i|x) \sim \arg\max_i <w_{inf} x>
\]

where

\[
w_{inf} \in \mathbb{R}^k, \; k < < m
\]

The output thus can be thought of as a decomposed softmax. An illustration of this is given in Fig 5.

### 6.2.5 Visualizing influential features

Here we visualize the features from retrained model Resnet50 after decomposition and compare it with the standard Resnet50 trained on Imagenet.

For each image, we subtract the mean of the influential filters at the final convolutional layer after average pooling at each dimension and upsample the activation to \( 224 \times 224 \), which is the standard size of the input image to Resnet models. For comparison, we select \( k \) filters from other filters for the same class. The results are in Table 7, where standard Resnet50 features trained on Imagenet are displayed alongside influential features and non-influential features on the decomposed Resnet50.

We can see that the influential features have better overlap with the regions for the target class of the objects in the scene, while the non-influential features seem to focus more on the environment. This lends credence to our
Table 7  Comparison of visualization of standard Resnet and influential and non-influential features

| Category           | Image | Standard Resnet Features | Non-Influential Features | Influential Features |
|--------------------|-------|---------------------------|--------------------------|---------------------|
| Abacus             | ![Image](Abacus) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Seat belt          | ![Image](Seat belt) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Squirrel monkey    | ![Image](Squirrel monkey) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Wreck              | ![Image](Wreck) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Madagascar cat     | ![Image](Madagascar cat) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Standard poodle    | ![Image](Standard poodle) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Welsh springer spaniel | ![Image](Welsh springer spaniel) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Frilled lizard     | ![Image](Frilled lizard) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
| Thatch             | ![Image](Thatch) | ![Image](Standard) | ![Image](Non-Influential) | ![Image](Influential) |
hypothesis of these features being critical to the object classification.

6.2.6 Quantifying decomposition

We use Mutual Information Score (MI) between two sets of class influential features to measure the extent of decomposition affected with influential features. A lower MI indicates more independence between two sets of features.

\[
\text{MI}(U, V) = \sum_{i=1}^{\left|U\right|} \sum_{j=1}^{\left|V\right|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|}
\]

We take pairwise sets of indices of influential features for all classes for CIFAR-10 and calculate the Mutual Information between each. We repeat the same for CSG for comparison. The result displayed in Table 8 is the mean for all classes at varying \(k\).

6.2.7 Essentiality of influential features

Here we perform an ablation study with and without the influential features to demonstrate their essential nature. For the study, we replace the values at the pre-final layer of influential features and non-influential features respectively with Gaussian Noise. [21] zero out the values of the corresponding features, but that would create a more drastic perturbation which is sure to affect the result negatively. We feel introducing Gaussian Noise is instead a better way to check the features’ essential nature. The results are in Table 9.

For comparison, we conduct the same evaluation for [21]. The evaluation is conducted by taking the top-5 features for both influential features and CSG for CIFAR-10 and top-50 features for Imagenet. Resnet20 was trained and evaluated on CIFAR-10 while Resnet50 was evaluated on Imagenet. We had recreated the CIFAR-10 experiments from [21] while experiments for Imagenet were not available.

For Resnet20 and CIFAR-10, the accuracy by replacing influential features with noise drops from 91.17, while replacing non-influential features results in an accuracy drop of 8.97. For Resnet50 and Imagenet, the results are even more pronounced with accuracy drop with noise for influential features resulting in a drop of 52.05 drop for corresponding non-influential features.

This shows that influential features are essential to the correct classification for the given class. We can explain the difference in results between CIFAR-10 and Imagenet due to the lower resolution of CIFAR-10 images, where object features in the images themselves are not so pronounced.

For CSG, on the other hand, we can see that adding Gaussian Noise leads to a drop in accuracy for both the class-specific features and other features. This demonstrates the superior class selectivity of our method.

7 Discussion and future work

We have described here a novel method of identifying the \(k\) most influential class-specific features for the final layers of a CNN. We have also given a decomposed softmax to retrain the network after decomposition. The final layer feature contribution makes the network more interpretable per class. We have also shown that these features are essential for the correct classification of the given labels.

We have also shown that deeper networks and wider networks at the same depth have inherently more disentangled representations in the final layers. Finally, we also provide the implementation for the experiments and the decomposed final layer computations.

For future work, the focus of the investigation can shift to the inner layers of the CNN and the possibility of further hierarchical decomposition of the CNN as a tree-like structure. The proposed method can also be checked on language models and, if it works, can be used for

| \(k\) | Ours | CSG |
|------|------|------|
| 5    | 0.0345378 | 0.0375091 |
| 10   | 0.0570505 | 0.0540359 |
| 20   | 0.0708032 | 0.0817442 |

The bold indicates the lower mutual information score

| Model | Original accuracy | Accuracy with influential features replaced with Gaussian noise | Accuracy with non-influential features replaced with Gaussian noise |
|-------|------------------|---------------------------------------------------------------|---------------------------------------------------------------|
| Resnet20 (Ours) | 91.17 | 47.52 | 84.98 |
| Resnet20 (CSG) | 23.63 | 8.32 | |
| Resnet50 | 74.55 | 12.52 | 74.40 |
| Resnet50 (CSG) | NA | NA | |
interpretable visual semantic alignment for joint language and vision tasks.

Another aspect that can be explored in the future is the overlap between influential features and the object itself in the image. These features may be used for Semantic Segmentation of objects without training segmentation annotations. That might require a reformulation of the task with additional constraints added to classification.

Models like EfficientNet [9] give us different results as they are derived from Neural Architecture Search (NAS), which leads to more complex models, and we can hypothesize that their representations are more entangled. Other models, on the other hand, are designed by human intuition and are simpler. However, that would require exploration of NAS-based models which we also leave to future work.

**Supplementary Information** The online version supplementary material available at https://doi.org/10.1007/s00521-023-08441-z.

**Acknowledgements** We wish to thank the MEITY, Govt. of India, which funded part of this research with Visvesvaraya PhD Fellowship award number MEITY-PHD-1035.

**Data availability** All the data associated with the experiments in this paper are available as electronic supplementary material.

**Code availability** All the code associated with the experiments in this paper are available at an online code repository.

**Declarations**

**Competing interests** The authors declare no competing interests.

**References**

1. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9

2. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778

3. Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I (2019) Language models are unsupervised multitask learners. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf. Accessed: 2021-06-25

4. Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler DM, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D (2020) Language models are few-shot learners. In: Advances in Neural Information Processing Systems

5. Choromanska A, Henaff M, Mathieu M, Arous GB, LeCun Y (2014) The loss surfaces of multilayer networks. In: Proceedings of the International Conference on Artificial Intelligence and Statistics vol. 38, pp. 192–204

6. Du S, Zhai X, Póczos B, Singh A (2019) Gradient descent provably optimizes over - parameterized neural networks. In: Proceedings of the International Conference on Learning Representations

7. Du S, Lee J, Li H, Wang L, Zhai X (2019) Gradient descent finds global minima of deep neural networks. In: Proceedings of the International conference on machine learning

8. Huang G, Liu Z, Weinberger KQ (2017) Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2261–2269

9. Tan M, Le QV (2019) Efficientnet: rethinking model scaling for convolutional neural networks. In: Proceedings of the International Conference on Machine Learning

10. Zhang Y, Tiño P, Leonardis A, Tang K (2020) A survey on neural network interpretability. IEEE Trans Emerg Topics Comput Intell 5(5):726–742

11. Frosst N, Hinton GE (2017) Distilling a neural network into a soft decision tree. In: Cex@ai*ia

12. Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: Proceedings of the European Conference on Computer Vision

13. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R (2014) Intriguing properties of neural networks. In: Proceedings of the International Conference on Learning representations

14. Simonyan K, Vedaldi A, Zisserman A (2014) Deep inside convolutional networks: visualising image classification models and saliency maps. In: Proceedings of the International Conference on Learning representations

15. Gonzalez-Garcia A, Modolo D, Ferrari V (2017) Do semantic parts emerge in convolutional neural networks? Int J Comput Vis 126:476–494

16. Bau D, Zhou B, Khosla A, Oliva A, Torralba A (2017) Network dissection - quantifying interpretability of deep visual representations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3319–3327

17. Bach S, Binder A, Montavon G, Klauschen F, Müller K, Samek W (2015) On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. In: Plos One

18. Hendricks LA, Akata Z, Rohrbach M, Donahue J, Schiele B, Darrell T (2016) Generating visual explanations. In: Proceedings of the European Conference on Computer Vision

19. Alain G, Bengio Y (2017) Understanding intermediate layers using linear classifier probes. In: Proceedings of the International Conference on Learning Representations

20. Zhang Q, Wu Y, Zhu S-C (2018) Interpretable convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8827–8836

21. Liang H, Ouyang Z, Zeng Y, Su H, He Z, Xia S, Zhu J, Zhang B (2020) Training interpretable convolutional neural networks by differentiating class-specific filters. In: European Conference on Computer Vision, pp. 622–638. Springer, Cham

22. Yang J, Shen X, Xing J, Tian X, Li H, Deng B, Huang J, Hua X (2019) Quantization networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

23. Zhang Q, Cao R, Shi F, Wu Y, Zhu S (2018) Interpreting CNN knowledge via an explanatory graph. In: Proceedings of the AAAI Conference on Artificial Intelligence

24. Zhang Q, Yang Y, Wu Y, Zhu S-C (2019) Interpreting CNNs via decision trees. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6254–6263

25. Zhou B, Khosla A, Lapedriza A, Oliva A, Torralba A (2016) Learning deep features for discriminative localization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
26. Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D (2017) Grad-CAM: visual explanations from deep networks via gradient-based localization. In: Proceedings of the IEEE International Conference on Computer Vision
27. Wickramanayake S, Hsu W, Lee M (2019) FLEX: faithful linguistic explanations for neural net based model decisions. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 2539–2546
28. Wickramanayake S, Hsu W, Lee M (2021) Comprehensible convolutional neural networks via guided concept learning. In: Proceedings of IEEE International Joint Conference on Neural Network
29. Liu B, Wang M, Foroosh H, Tappen M, Pensky M (2015) Sparse convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 806–814
30. Li H, Kadav A, Durandovic I, Samet H, Graf H (2017) Pruning filters for efficient ConvNets. In: Proceedings of the International Conference on Learning Representations
31. Kumar A, Shaikh AM, Li Y, Bilal H, Yin B (2021) Pruning filters with L1-norm and capped L1-norm for CNN compression. Appl Intell 51:1152–1160
32. Lin S, Ji R, Li Y, Deng C, Li X (2020) Toward compact ConvNets via structure-sparsity regularized filter pruning. In: IEEE Trans Neural Netw Learn Syst
33. Li Y, Lin S, Zhang B, Liu J, Doerrmann D, Wu Y, Huang F, Ji R (2019) Exploiting kernel sparsity and entropy for interpretable CNN compression. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
34. Yin M, Sui Y, Yang W, Zang X, Gong Y, Yuan B (2022) HODEC: towards efficient high-order DEcomposed convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
35. Li Y, Adamczewski K, Li W, Gu S, Timofte R, Gool L (2022) Revisiting random channel pruning for neural network compression. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition
36. Wang Y, Morariu VL, Davis L (2018) Learning a discriminative filter bank within a CNN for fine-grained recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4148–4157
37. Jiang Z, Wang Y, Davis L, Andrews W, Rozgic V (2017) Learning discriminative features via label consistent neural network. In: Proc. of the IEEE Winter Conference on Applications of Computer Vision, pp. 207–216
38. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the International Conference on Machine learning
39. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Adv Neural Inf Process Syst, pp. 1097–1105
40. Krizhevsky A (2009) Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
41. Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255
42. Zoph B, Le QV (2017) Neural architecture search with reinforcement learning. In: Proceedings of the International Conference on Learning Representations
43. Zagoruyko S, Komodakis N (2016) Wide residual networks. In: Proceedings of the British Machine Vision Conference

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.