Identifying Class Specific Filters with L1 Norm Frequency Histograms in Deep CNNs

Akshay Badola¹, Cherian Roy², Vineet Padmanabhan³, and Rajendra Lal⁴

¹,²,³,⁴ School of Computer and Information Sciences, University of Hyderabad, Prof C R Rao Road, Gachibowli, Hyderabad, 500019, Telangana, India
¹badola@uohyd.ac.in
²cherianthakidiyel@gmail.com
³vineetnair@uohyd.ac.in
⁴rplcs@uohyd.ernet.in

Abstract

Interpretability of Deep Neural Networks has become a major area of exploration. Although these networks have achieved state of the art accuracy 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 and provide an efficient method for identifying subsets of features that contribute most towards the network’s decision for a class. We demonstrate that the number of such features per class is much lower in comparison to the dimension of the final layer and therefore the decision surface of Deep CNNs lies on a low dimensional manifold and is proportional to the network depth. Our methods allow to decompose the final layer into separate subspaces which is far more interpretable and has a lower computational cost as compared to the final layer of the full network. We also release our code at https://github.com/akshaybadola/cnn-class-specific-filters-with-histogram

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 overparameterized and in fact, overparametrization is essential to their generalization ability [5], [6], [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 those models in critical situations, like medical diagnosis or self-driving vehicles. It also becomes very hard to diagnose and correct the models themselves.

Final layers\(^1\) 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 final 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 method to identify and rank the features in the final and penultimate layers and consequently decompose the layer. In this work we focus on the final and penultimate layers of some popular CNN architectures, namely Resnet [2], Densenet [8], Efficientnets [9]. We illustrate some properties of those layers and consequent features and provide an efficient method to decompose the final layer which results in interpretability via disentanglement and reduces the computational complexity of the final layer.

Related Work

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 for 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 the purpose. For example, in an image, parts of an object which contribute more towards 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 parameters of the model and the input-label decision making process. That is, it should be evident that \textit{which} parameters act on \textit{which} parts of inputs to produce \textit{which} labels. Again, the number of such

\(^1\)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).
parameters and inputs-parts, that would be attributed could be filtered for simplification.

For any model, interpretation should result in some attribution from the parameters of the model towards the predictions made by it. This is separate than distilling a network into a more explainable model like a decision tree [11], which aims more towards 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 accuracy.

In the context of CNNs, we can look to interpret them on the basis of 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.

**CNN Interpretability**

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], [13], [14], analyzed the filters in intermediate layers and found that the features from pretrained neural networks can be used for other tasks.

Later work moved more towards class attribution and aligning features to semantic concepts. Gonazelz-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. Guillame 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 towards 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], which 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 pretrained 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. [33] introduce a kernel sparsity and entropy (KSE) measure which quantifies both sparsity and diversity of the convolution kernels.

The notion of class specificity is explored in Wang et. al. [34]. They try to increase accuracy with filters that capture class-specific discriminative patches. These class-specific filters are closely related to earlier work in Jiang et. al. [35], who introduce the concept of Label Consistent Neural Networks to learn features which they claim alleviate gradient vanishing and leads to faster convergence. [21] try to use sparsity to find out the important filters which they claim are also class specific.

Our Contributions

Our work takes inspiration from [30], [34] and [21]. We combine the ideas of class specificity [34], [21] and $\ell_1$ norm based filter importance [30] to arrive at a method for identifying the $k$ most influential features based on an $\ell_1$ norm importance metric in the final and penultimate layers of a CNN and demonstrate its efficacy with experiments on various CNNs.

In particular:

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.

The rest of the paper is organized as follows: We discuss CNNs and the concept class specific features in Section . We provide an overview of analysis of techniques on final layer and class specific features therein, in Section . We describe the experimental details and results in Section . We discuss implications and future scope in Section .

Background

We discuss CNNs, filters and layers in the next sections and establish notation which will aid us in our describing our methods.
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 which act as a sequence of operators on input \( X \), such that,

\[
y = A(W^d(...A(W^0(X))))
\]

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 \( L^d \).

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 due to 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 [36] 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:

\[
\text{Conv}_{w_{k \times k}} : I \to O \text{ where } O_{i,j} = \sum_{l=-k'}^{k'} I_{i-k-j-k} w_{k-l,k-l}
\]

and \( k' = \lfloor k/2 \rfloor \)

Let \( I^d \) be the input at the final layer of the CNN with total depth \( d \). The output at the final layer is then \( O^d = \sigma(WI^d) \), where \( \sigma \) is the softmax operator. \( \sum O^d_i = 1 \) 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 \( O^d_i \) and the most probable label \( \hat{y} \) is given by \( \hat{y} = \arg\max(O^d) \). See Fig 1 for an illustration.
Figure 1: CNN with Fully Connected Final Layer. An image is input at the head of the network where it travels sequential through the convolutional layers (denoted by CONV). At the final layer (FC layer) the input coming from the final convolutional layer is flatted to a vector. The predictions are done by applying softmax $\sigma$ elementwise to the outputs.

Filter Disentanglement and Label Consistency

As mentioned earlier, the sheer number of features and the layers in CNNs makes them very hard to interpret. One approach towards 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 towards the prediction of a label as a surrogate measure of disentanglement. Ideally every filter should be responsible for detection of a particular pattern in the image, however 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 mentioned in Section, the set of filters for a CNN at depth $d$ constitutes the layer $L^d$. Now consider a single label $y \in \mathcal{Y}$. Let $\exists J_y \subseteq L^d$ 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. $L_{J_{y_1}} \neq L_{J_{y_2}}$, $\forall y_1, y_2 \in \mathcal{Y}$

6
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. $\arg\min_{J_{y_1}, J_{y_2}} (L_{J_{y_1}} \cap L_{J_{y_2}}), \forall y_1, y_2 \in \mathcal{Y}$

That is, we should seek index sets $J_y$ such that there is minimal overlap between
two classes.

These class-specific filters are closely related to [35], except that [35] 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 [34] our method does not require supervision and can
be used for any pretrained 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.

**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.

**$\ell_1$ norm for Feature Importance**

The importance of $\ell_1$ norm for the filters becomes evident considering [30] and
[21] which both use $\ell_1$ 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 $50\% \leq 0$. 
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, \( P(y|X) = O^d = \sigma(W^d) \) as in Section, where \( O^d \in \mathbb{R}^n \) and \( \sum_i O_i^d = 1 \). If we omit \( d \) and \( X \) for simplicity, then probability for \( i^{th} \) class, \( P(y = i) \) is the value of \( i^{th} \) 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 \( i^{th} \) 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 \( W_i \in \mathbb{R}^{m \times n} \) and \( W_i, I \in \mathbb{R}^m \).

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

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 \( w_{k,i} \) 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.

Finding the Influential Features

We note that finding the class specific influential features \( w_{k,i} \) is non-trivial as the exact subset of the weights cannot be known easily and an exhaustive search is of exponential complexity. However, as we mention in Section \( \ell_1 \) norm of the features can help us in guiding towards 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 top \( k \) features per class with \( \ell_1 \) norm gave us better results. Even combining top \( k \) filters per class with top \( k \) weights led to poorer results than with top \( k \) 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 top \( k \) selection from histogram of top \( k \) features for all instances in a given class had mass concentrated around a few points, which corresponded to such influential features. We give the details of the algorithm in the next section.
Algorithm

Here we discuss the algorithm to obtain the $k$ most influential features for each class from a pretrained CNN. For the algorithm 1 below, $\mathcal{I}$ are the set of features at final layer for all classes, where we omit the layer superscript used earlier for simplicity. Denote the set of features for label $y$ by $I_y$. Then we want to obtain the mapping $I : y \rightarrow I_y$, such that, $I(y)$ 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 $k_1$ features by $\ell_1$ norm of each data instance for each class at the pre-final layer and select the indices in a set $I_y^{topk_1}$ for class $y$. We set $k_1 \ll m$ 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 5 components sorted by $\ell_1$ norm occur at indices $< 3, 5, 14, 18, 28 >$. Then the index set for $k_1 = 5$ for the entire class is aggregation $\bigcup I_y^{top5}$ of all such sets, where $\bigcup$ is an aggregation operator, e.g. $\bigcup < 3, 5 > < 3, 14 > = < 3, 3, 5, 14 >$

The set $\bigcup I_y^{top}$ then denotes all the occurrences of a particular dimension (or index) of a feature in $top \ell_1$ norm set, for a class in 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 contribution of the filters. However, we found that a value of 5 for CIFAR-10 and 50 for Imagenet datasets gave us 90% coverage, which we then chose to set for all our experiments as that is faster to implement.

Algorithm 1 Extract Influential Features

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

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

Experiments and Results

We conducted some initial experiments on Class Specific Gates (CSG) [21] and we discuss them in as to illustrate our argument. We then analyze and discuss
the efficacy of our proposed influential features in . 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 doesn’t lead to explicit class specific disentanglement. We demonstrate that our unsupervised approach gives as a) good if not better results than [21] b) leads to explicit filter disentanglement c) reduces computational cost in the final layer.

Class Specific Filters

We first discuss experiments with class specific gates (CSG). Liang et. al. [21] had proposed to learn a matrix \( CSG \in \mathbb{R}^{m \times n} \) where \( m \) is the dimension of final layer and \( c = |\mathcal{Y}| \) 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 doing inference as the labels for new data is 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\(^2\). We implement CSG and summarize the results in Table 1.

STD output in the Table 1 is \( O_{STD}^i = O_i^d = \sigma(W d^i) \), which is the same as CNN output without a CSG matrix. Output using CSG matrix is \( O_{CSG}^i = \sigma(W(I^d \odot CSG_i)) \), 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.

\(^2\)See https://github.com/hyliang96/CSGCNN
| Dataset   | Model | Training | Accuracy |
|----------|-------|----------|----------|
| CIFAR-10 | Resnet20 | CSG      | 0.8809   |
|          |       | STD      | 0.8809   |
|          | Resnet34 | CSG      | 0.8407   |
|          |       | STD      | 0.8407   |
| Tiny Imagenet 200 | Resnet18 | CSG      | 0.3633   |
|          |       | STD      | 0.3641   |
|          | Resnet34 | CSG      | 0.3794   |
|          |       | STD      | 0.3852   |

Table 1: CSG Experiments Conducted by us

**Influential Features**

We then performed experiments to obtain influential filters according to Algorithm 1 for datasets CIFAR-10 and Imagenet, and families of models a) Resnet b) Densenet c) Efficientnet to demonstrate that our method can work for any CNN.

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 [37] and aim to reduce the computational cost while preserving accuracy.

CIFAR-10 consists of 50,000 training images for 10 classes, i.e., \( c = |\mathcal{Y}| = 10 \), of size 32 × 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 × 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 \( \bigcup f_{topk} \) only from the training set. The results are then calculated on the validation set.

**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 pretrained model had accuracy 91.17. We measure the efficacy of the resulting decomposed final layer as the ratio \( r_A = \frac{A_d}{A_f} \), 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 3 filters per class. This confirms that the decision surface lies on a much lower dimensional subspace than the dimension of the final layer.

11
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 initial filter selection $k_1$ is very large, then equivalent $k_2$ setting leads to lower performance as compared to a lower $k_1$.

**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.

![Figure 3: Effect of $k_1$ and $k_2$ on Accuracies for Resnet20 and CIFAR-10](image.png)

| Final Dim n | Resnet50 | Resnet101 | Resnet152 |
|-------------|----------|-----------|-----------|
| $k_2/c$     | 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 2: Effect of depth on Resnet variants
We also evaluate on Wide Resnets [38] 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 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 EfficientNet deviates from this behaviour which we discuss later. Densenets [8] have different $n$ and hence $k/n$ differs for them. For Densenets we compare Densenet121 with Densenet169 and Densenet161 with Densenet201, as the two pairs have closer $n$ and hence $k/n$.

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.

### Table 3: Effect of depth on Wide Resnet

| Final Dim $n$ | Wide Resnet 50 | Wide Resnet 101 |
|--------------|----------------|-----------------|
| $k_2/c$      | 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

| Final Dim $n$ | Densenet121 | Densenet169 |
|--------------|-------------|-------------|
| $k_3/c$      | 0.05        |             |
| $k_3/n$      | 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 121 and 169

| Final Dim $n$ | Densenet161 | Densenet201 |
|--------------|-------------|-------------|
| $k_3/c$      | 0.05        |             |
| $k_3/n$      | 0.0226449   | 0.0260417   |
| $A_d$        | 0.6766      | 0.68214     |
| $A_f$        | 0.75268     | 0.7455      |
| $r_A$        | 0.898921    | 0.91501     |
|                   | 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        |

Table 6: Effect of depth on Efficientnet variants

Figure 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.
variants. The final layer dimension $n$ also differs for Densenets 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 disentanglement. See Figure 4 for a summary of $r_A$ on all models.

Only with Efficientnets (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 $r_A$. efficientnet_b3 is a curiosity as that model is both deeper and wider.

Predicting 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 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 accuracy of 91.51, which is very close to the original accuracy of 91.71. An illustration of how efficient prediction with influential features works is given in Figure 5.

The computational complexity for the final layer is $d \times n$ 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 $64 \times 10$ which drops to $5 \times 10$ with influential features.

Discussion and Future Work

We have described here a method of identifying the $k$ most influential features for the final layers of a CNN. The method allows us greater interpretability into the CNN and also to decompose the final layer into separate class specific final layer. We have also shown that deeper networks have inherently more disentangled representations in the final layers. We also provide the implementation for the experiments and the decomposed final layer computations.

The proposed approach is novel and there is sufficient scope to extend it for the preceding layers in the CNN. We have also seen that the accuracy can also be recovered when trained for very few epochs for CIFAR-10. This raises the possibility of further hierarchical decomposition of the CNN as a tree like structure which can be explored in future work.
Models like EfficientNet [9] however, 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.

References

[1] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf, 2019. Accessed: 2021-06-25.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. Henighan, R. Child, A. Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish,
Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In NeurIPS, 2020.

[5] A. Choromańska, Mikael Henaff, Michaël Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In AISTATS, volume 38, pages 192–204, nov 2015.

[6] S. Du, Xiyu Zhai, B. Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In ICLR, 2019.

[7] S. Du, J. Lee, H. Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In ICML, nov 2019.

[8] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2261–2269, aug 2017.

[9] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In ICML, 2019.

[10] Y. Zhang, P. Tiño, A. Leonardis, and K. Tang. A survey on neural network interpretability. IEEE Transactions on Emerging Topics in Computational Intelligence, 2020.

[11] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In CEx@AI*IA, 2017.

[12] Matthew D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In ECCV, 2014.

[13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In ICLR, 2014.

[14] K. Simonyan, A. Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In ICLR, 2014.

[15] Abel Gonzalez-Garcia, Davide Modolo, and V. Ferrari. Do semantic parts emerge in convolutional neural networks? International Journal of Computer Vision, 126:476–494, jul 2017.

[16] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection - quantifying interpretability of deep visual representations. In CVPR, pages 3319–3327, 2017.

[17] S. Bach, Alexander Binder, Grégoire Montavon, F. Klauschen, K. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. In PloS one, 2015.
[18] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, B. Schiele, and Trevor Darrell. Generating visual explanations. In ECCV, 2016.

[19] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In ICLR, 2017.

[20] Quanshi Zhang, Y. Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8827–8836, oct 2018.

[21] Haoyu Liang, Zhihao Ouyang, Yuyuan Zeng, Hang Su, Zihao He, Shu-Tao Xia, Jun Zhu, and Bo Zhang. Training interpretable convolutional neural networks by differentiating class-specific filters. In ECCV, pages 622–638. Springer International Publishing, jul 2020.

[22] J. Yang, X. Shen, Jun Xing, X. Tian, H. Li, B. Deng, Jianqiang Huang, and X. Hua. Quantization networks. In 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR), 2019.

[23] Q. Zhang, Ruiming Cao, Feng Shi, Y. Wu, and S. Zhu. Interpreting CNN knowledge via an explanatory graph. In AAAI, 2018.

[24] Quanshi Zhang, Yu Yang, Y. Wu, and Song-Chun Zhu. Interpreting CNNs via decision trees. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 6254–6263, feb 2019.

[25] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In 2016 IEEE conference on computer vision and pattern recognition (CVPR). IEEE, jun 2016.

[26] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In 2017 IEEE international conference on computer vision (ICCV), 2017.

[27] Sandareka Wickramanayake, W. Hsu, and M. Lee. Flex: Faithful linguistic explanations for neural net based model decisions. In AAAI, pages 2539–2546, jul 2019.

[28] Sandareka Wickramanayake, W. Hsu, and M. Lee. Comprehensible convolutional neural networks via guided concept learning. 2021.

[29] Baoyuan Liu, M. Wang, H. Foroosh, M. Tappen, and Marianna Pensky. Sparse convolutional neural networks. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 806–814, jun 2015.

[30] Hao Li, Asim Kadav, Igor Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient ConvNets. In ICLR, 2017.
[31] Aakash Kumar, Ali Muhammad Shaikh, Yun Li, Hazrat Bilal, and B. Yin. Pruning filters with L1-norm and capped L1-norm for CNN compression. *Applied Intelligence*, 51:1152–1160, sep 2021.

[32] Shaohui Lin, R. Ji, Yuchao Li, Cheng Deng, and Xuelong Li. Toward compact ConvNets via structure-sparsity regularized filter pruning. In *IEEE transactions on neural networks and learning systems*, 2020.

[33] Yuchao Li, Shaohui Lin, Baochang Zhang, Jianzhuang Liu, D. Doermann, Yongjian Wu, Feiyue Huang, and R. Ji. Exploiting kernel sparsity and entropy for interpretable CNN compression. In 2019 *IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2019.

[34] Yaming Wang, Vlad I. Morariu, and L. Davis. Learning a discriminative filter bank within a CNN for fine-grained recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4148–4157, nov 2018.

[35] Zhuolin Jiang, Yaming Wang, L. Davis, Walter Andrews, and Viktor Rozgic. Learning discriminative features via label consistent neural network. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 207–216, feb 2017.

[36] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[37] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, nov 2017.

[38] Sergey Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.