Computing Receptive Fields of Convolutional Neural Networks

Mathematical derivations and open-source library to compute receptive fields of convnets, enabling the mapping of extracted features to input signals.

AUTHORS

André Araujo
Wade Norris
Jack Sim

AFFILIATIONS

Google Research
Perception Labs
Google Research

PUBLISHED

Nov. 4, 2019

DOI

10.23915/distill.00021

Contents

While deep neural networks have overwhelmingly established state-of-the-art results in many artificial intelligence problems, they can still be difficult to develop and debug. Recent research on deep learning understanding has focused on feature visualization [1, 2], theoretical guarantees [3, 4], model interpretability [5, 6], and generalization [7, 8].

In this work, we analyze deep neural networks from a complementary perspective, focusing on convolutional models. We are interested in understanding the extent to which input signals may affect output features, and mapping features at any part of the network to the region in the input that produces them. The key parameter to associate an output feature to an input region is the *receptive field* of the convolutional network, which is defined as the size of the region in the input that produces the feature.

As our first contribution, we present a mathematical derivation and an efficient algorithm to compute receptive fields of modern convolutional neural networks. Previous work [9, 10] discussed receptive field computation for simple convolutional networks where there is a single path from the input to the output, providing recurrence equations that apply to this case. In this work, we revisit these derivations to obtain a closed-form expression for receptive field computation in the single-path case. Furthermore, we extend receptive field computation to modern convolutional networks where there may be multiple paths from the input to the output. To the best of our knowledge, this is the first exposition of receptive field computation for such recent convolutional architectures.

Today, receptive field computations are needed in a variety of applications. For example, for the computer vision task of object detection, it is important to represent objects at multiple scales in order to recognize small and large instances; understanding a convolutional feature's span is often required for that goal (e.g., if the receptive field of the network is small, it may not be able to recognize large objects). However, these computations are often done by hand, which is both tedious and error-prone. This is because there are no libraries to compute these parameters automatically. As our second contribution, we fill the void by introducing an open-source library which handily performs the computations described here. The library is integrated into the Tensorflow codebase and can be easily employed to analyze a variety of models, as presented in this article.

We expect these derivations and open-source code to improve the understanding of complex deep learning models, leading to more productive machine learning research.

Overview of the article

We consider fully-convolutional neural networks, and derive their receptive field size and receptive field locations for output features with respect to the input signal. While the derivations presented here are general enough for any type of signal used at the input of convolutional neural networks, we use images as a running example, referring to modern computer vision architectures when appropriate.

First, we derive closed-form expressions when the network has a single path from input to output (as in AlexNet [11] or VGG [12]). Then, we discuss the more general case of arbitrary computation graphs with multiple paths from the input to the output (as in ResNet [13] or Inception [14]). We consider potential alignment issues that arise in this context, and explain an algorithm to compute the receptive field size and locations.

Finally, we analyze the receptive fields of modern convolutional neural networks, showcasing results obtained using our open-source library.

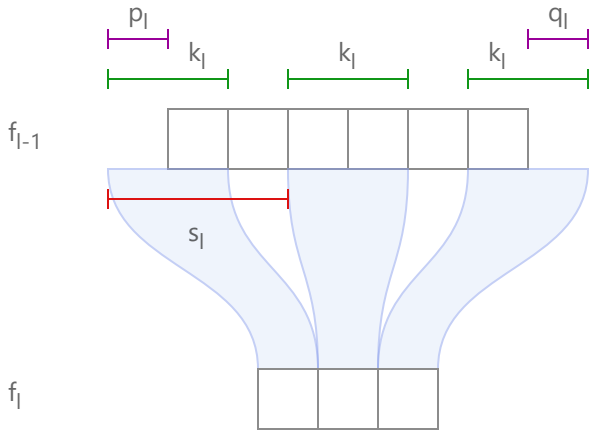
Problem setup

Consider a fully-convolutional network (FCN) with L layers, $l = 1, 2, \ldots, L$. Define feature map

$f_l \in R^{h_l \times w_l \times d_l}$ to denote the output of the l-th layer, with height h_l, width w_l and depth d_l. We denote the input image by f_0. The final output feature map corresponds to f_L.
To simplify the presentation, the derivations presented in this document consider 1-dimensional input signals and feature maps. For higher-dimensional signals (e.g., 2D images), the derivations can be applied to each dimension independently. Similarly, the figures depict 1-dimensional depth, since this does not affect the receptive field computation.

Each layer l's spatial configuration is parameterized by 4 variables, as illustrated in the following figure:

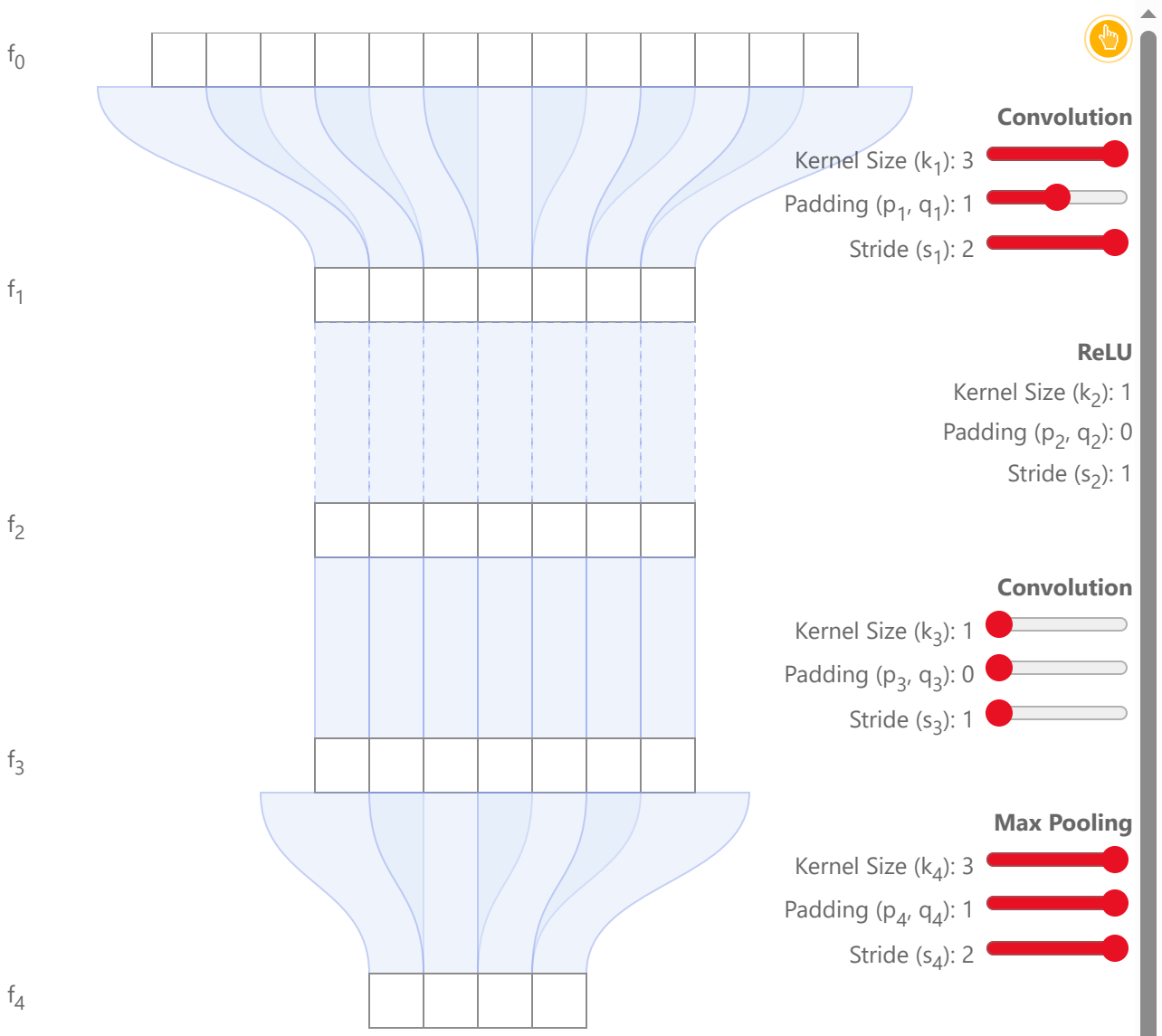
- k_l: kernel size (positive integer)
- s_l: stride (positive integer)
- p_l: padding applied to the left side of the input feature map (non-negative integer) [1]
- q_l: padding applied to the right side of the input feature map (non-negative integer)



Kernel Size (k_l): 2
Left Padding (p_l): 1
Right Padding (q_l): 1
Stride (s_l): 3

We consider layers whose output features depend locally on input features: e.g., convolution, pooling, or elementwise operations such as non-linearities, addition and filter concatenation. These are commonly used in state-of-the-art networks. We define elementwise operations to have a "kernel size" of 1, since each output feature depends on a single location of the input feature maps.

Our notation is further illustrated with the simple network below. In this case, $L = 4$ and the model consists of a convolution, followed by ReLU, a second convolution and max-pooling. [2]



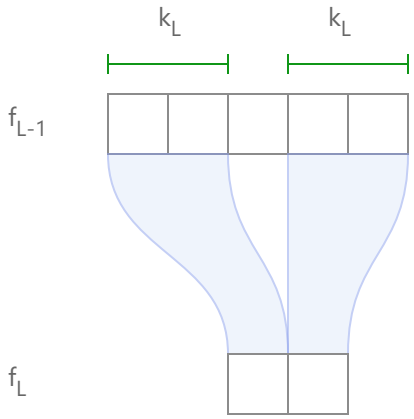
Single-path networks

In this section, we compute recurrence and closed-form expressions for fully-convolutional networks with a single path from input to output (e.g., AlexNet [11] or VGG [12]).

Computing receptive field size

Define r_l as the receptive field size of the final output feature map f_L, with respect to feature map f_l. In other words, r_l corresponds to the number of features in feature map f_l which contribute to generate one feature in f_L. Note that $r_L = 1$.

As a simple example, consider layer L, which takes features f_{L-1} as input, and generates f_L as output. Here is an illustration:

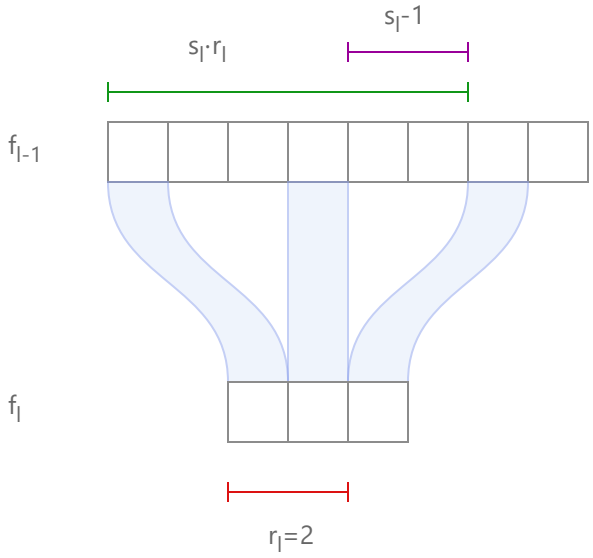


Kernel Size (k_L): 2
Padding (p_L, q_L): 0
Stride (s_L): 3

It is easy to see that k_L features from f_{L-1} can influence one feature from f_L, since each feature from f_L is directly connected to k_L features from f_{L-1}. So, $r_{L-1} = k_L$.

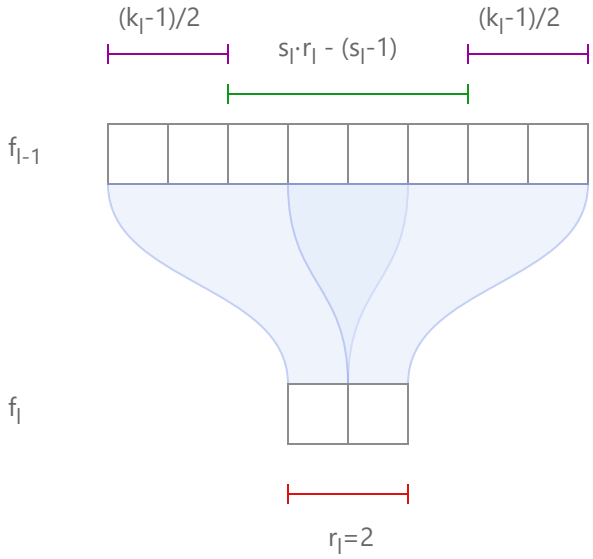
Now, consider the more general case where we know r_l and want to compute r_{l-1}. Each feature f_l is connected to k_l features from f_{l-1}.

First, consider the situation where $k_l = 1$: in this case, the r_l features in f_l will cover $r_{l-1} = s_l \cdot r_l - (s_l - 1)$ features in in f_{l-1}. This is illustrated in the figure below, where $r_l = 2$ (highlighted in red). The first term $s_l \cdot r_l$ (green) covers the entire region where the features come from, but it will cover $s_l - 1$ too many features (purple), which is why it needs to be deducted. [3]



Kernel Size (k_l): 1
Padding (p_l, q_l): 0
Stride (s_l): 3

For the case where $k_l > 1$, we just need to add $k_l - 1$ features, which will cover those from the left and the right of the region. For example, if we use a kernel size of 5 ($k_l = 5$), there would be 2 extra features used on each side, adding 4 in total. If k_l is even, this works as well, since the left and right padding will add to $k_l - 1$. [4]



Kernel Size (k_l): 5
Padding (p_l, q_l): 0
Stride (s_l): 3

So, we obtain the general recurrence equation (which is first-order, non-homogeneous, with variable coefficients):

$$r_{l-1} = s_l \cdot r_l + (k_l - s_l) \tag{1}$$

This equation can be used in a recursive algorithm to compute the receptive field size of the network, r_0. However, we can do even better: we can solve the recurrence equation and obtain a solution in terms of the k_l's and s_l's:

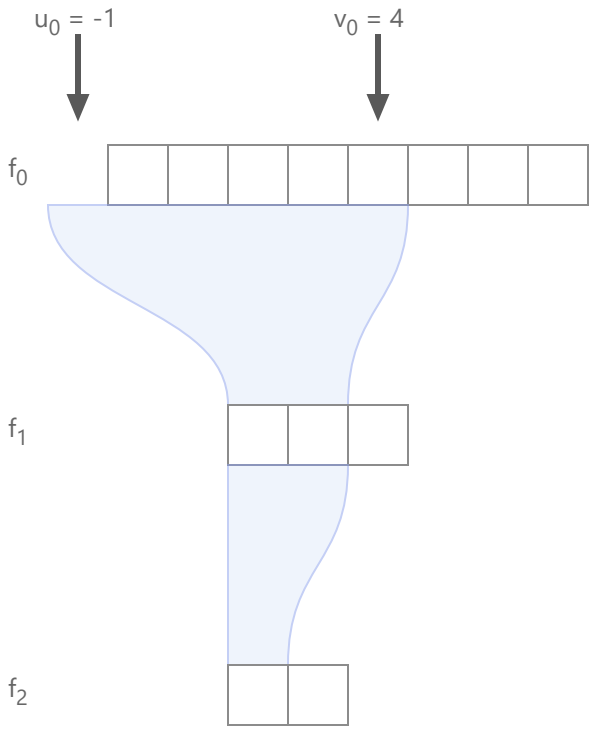
$$r_0 = \sum_{l=1}^{L} \left((k_l - 1) \prod_{i=1}^{l-1} s_i \right) + 1 \tag{2}$$

This expression makes intuitive sense, which can be seen by considering some special cases. For example, if all kernels are of size 1, naturally the receptive field is also of size 1. If all strides are 1, then the receptive field will simply be the sum of $(k_l - 1)$ over all layers, plus 1, which is simple to see. If the stride is greater than 1 for a particular layer, the region increases proportionally for all layers below that one. Finally, note that padding does not need to be taken into account for this derivation.

Computing receptive field region in input image

While it is important to know the size of the region which generates one feature in the output feature map, in many cases it is also critical to precisely localize the region which generated a feature. For example, given feature $f_L(i, j)$, what is the region in the input image which generated it? This is addressed in this section.

Let's denote u_l and v_l the left-most and right-most coordinates (in f_l) of the region which is used to compute the desired feature in f_L. In these derivations, the coordinates are zero-indexed (i.e., the first feature in each map is at coordinate 0). Note that $u_L = v_L$ corresponds to the location of the desired feature in f_L. The figure below illustrates a simple 2-layer network, where we highlight the region in f_0 which is used to compute the first feature from f_2. Note that in this case the region includes some padding. In this example, $u_2 = v_2 = 0$, $u_1 = 0, v_1 = 1$, and $u_0 = -1, v_0 = 4$.



Kernel Size (k_1): 3
Padding (p_1, q_1): 1
Stride (s_1): 3

Kernel Size (k_2): 2
Padding (p_2, q_2): 0
Stride (s_2): 1

We'll start by asking the following question: given u_l, v_l, can we compute u_{l-1}, v_{l-1}?

Start with a simple case: let's say $u_l = 0$ (this corresponds to the first position in f_l). In this case, the left-most feature u_{l-1} will clearly be located at $-p_l$, since the first feature will be generated by placing the left end of the kernel over that position. If $u_l = 1$, we're interested in the second feature, whose left-most position u_{l-1} is $-p_l + s_l$; for $u_l = 2$, $u_{l-1} = -p_l + 2 \cdot s_l$; and so on. In general:

$$u_{l-1} = -p_l + u_l \cdot s_l \tag{3}$$
$$v_{l-1} = -p_l + v_l \cdot s_l + k_l - 1 \tag{4}$$

where the computation of v_l differs only by adding $k_l - 1$, which is needed since in this case we want to find the right-most position.

Note that these expressions are very similar to the recursion derived for the receptive field size (1). Again, we could implement a recursion over the network to obtain u_l, v_l for each layer; but we can also solve for u_0, v_0 and obtain closed-form expressions in terms of the network parameters:

$$u_0 = u_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} p_l \prod_{i=1}^{l-1} s_i \tag{5}$$

This gives us the left-most feature position in the input image as a function of the padding (p_l) and stride (s_l) applied in each layer of the network, and of the feature location in the output feature map (u_L).

And for the right-most feature location v_0:

$$v_0 = v_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} (1 + p_l - k_l) \prod_{i=1}^{l-1} s_i \tag{6}$$

Note that, different from (5), this expression also depends on the kernel sizes (k_l) of each layer.

Relation between receptive field size and region. You may be wondering that the receptive field size r_0 must be directly related to u_0 and v_0. Indeed, this is the case; it is easy to show that $r_0 = v_0 - u_0 + 1$, which we leave as a follow-up exercise for the curious reader. To emphasize, this means that we can rewrite (6) as:

$$v_0 = u_0 + r_0 - 1 \tag{7}$$

Effective stride and effective padding. To compute u_0 and v_0 in practice, it is convenient to define two other variables, which depend only on the paddings and strides of the different layers:

- *effective stride* $S_l = \prod_{i=l+1}^{L} s_i$: the stride between a given feature map f_l and the output feature map f_L
- *effective padding* $P_l = \sum_{m=l+1}^{L} p_m \prod_{i=l+1}^{m-1} s_i$: the padding between a given feature map f_l and the output feature map f_L

With these definitions, we can rewrite (5) as:

$$u_0 = -P_0 + u_L \cdot S_0 \tag{8}$$

Note the resemblance between (8) and (3). By using S_l and P_l, one can compute the locations u_l, v_l for feature map f_l given the location at the output feature map u_L. When one is interested in computing feature locations for a given network, it is handy to pre-compute three variables: P_0, S_0, r_0. Using these three, one can obtain u_0 using (8) and v_0 using (7). This allows us to obtain the mapping from any output feature location to the input region which influences it.

It is also possible to derive recurrence equations for the effective stride and effective padding. It is straightforward to show that:

$$S_{l-1} = s_l \cdot S_l \tag{9}$$
$$P_{l-1} = s_l \cdot P_l + p_l \tag{10}$$

These expressions will be handy when deriving an algorithm to solve the case for arbitrary computation graphs, presented in the next section.

Center of receptive field region. It is also interesting to derive an expression for the center of the receptive field region which influences a particular output feature. This can be used as the location of the feature in the input image (as done for recent deep learning-based local features [15], for example).

We define the center of the receptive field region for each layer l as $c_l = \frac{u_l + v_l}{2}$. Given the above expressions for u_0, v_0, r_0, it is straightforward to derive c_0 (remember that $u_L = v_L$):

$$\begin{aligned} c_0 &= u_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} \left(p_l - \frac{k_l - 1}{2} \right) \prod_{i=1}^{l-1} s_i \\ &= u_L \cdot S_0 - \sum_{l=1}^{L} \left(p_l - \frac{k_l - 1}{2} \right) \prod_{i=1}^{l-1} s_i \\ &= -P_0 + u_L \cdot S_0 + \left(\frac{r_0 - 1}{2} \right) \end{aligned} \tag{11}$$

This expression can be compared to (8) to observe that the center is shifted from the left-most pixel by $\frac{r_0 - 1}{2}$, which makes sense. Note that the receptive field centers for the different output features are spaced by the effective stride S_0, as expected. Also, it is interesting to note that if $p_l = \frac{k_l - 1}{2}$ for all l, the centers of the receptive field regions for the output features will be aligned to the first image pixel and located at $0, S_0, 2S_0, 3S_0, \ldots$ (note that in this case all k_l's must be odd).

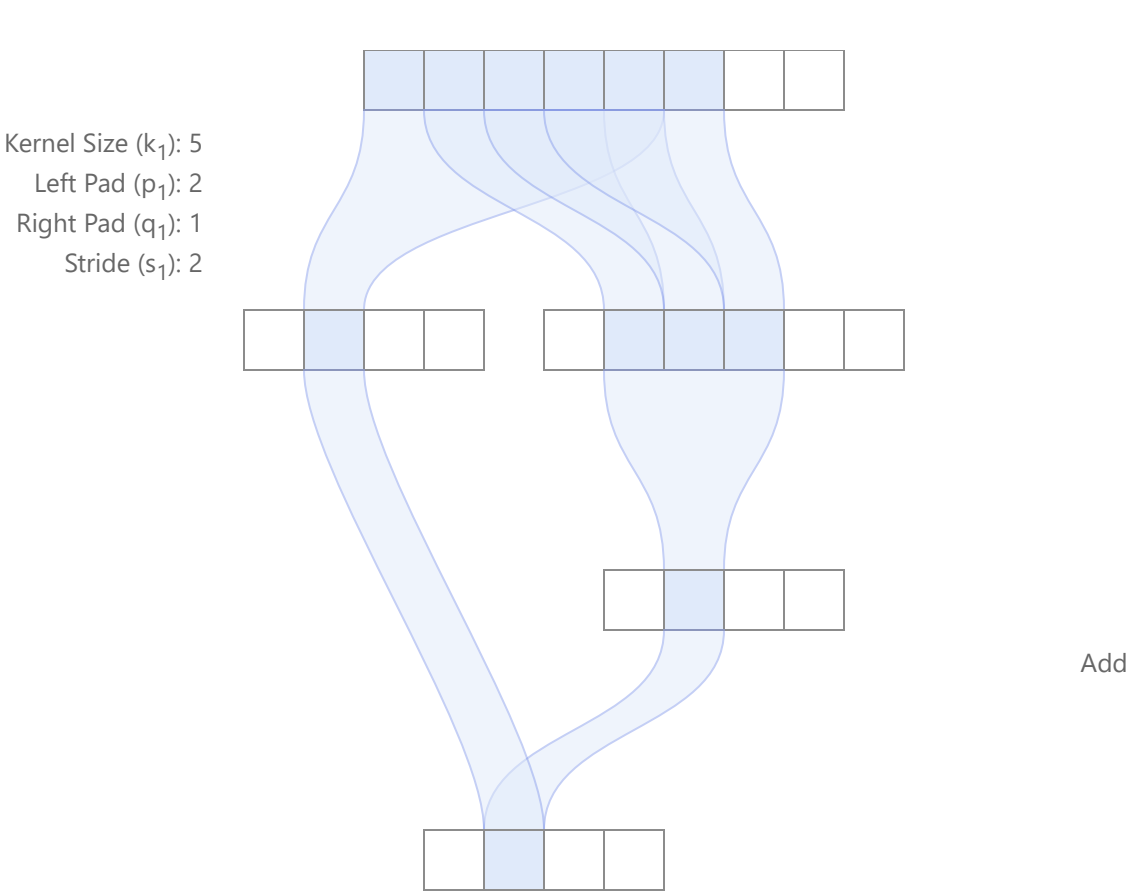
Other network operations. The derivations provided in this section cover most basic operations at the core of convolutional neural networks. A curious reader may be wondering about other commonly-used operations, such as dilation, upsampling, etc. You can find a discussion on these in the appendix.

Arbitrary computation graphs

Most state-of-the-art convolutional neural networks today (e.g., ResNet [13] or Inception [14]) rely on models where each layer may have more than one input, which means that there might be several different paths from the input image to the final output feature map. These architectures are usually represented using directed acyclic computation graphs, where the set of nodes $\mathcal{L}$ represents the layers and the set of edges $\mathcal{E}$ encodes the connections between them (the feature maps flow through the edges).

The computation presented in the previous section can be used for each of the possible paths from input to output independently. The situation becomes trickier when one wants to take into account all different paths to find the receptive field size of the network and the receptive field regions which correspond to each of the output features.

Alignment issues. The first potential issue is that one output feature may be computed using misaligned regions of the input image, depending on the path from input to output. Also, the relative position between the image regions used for the computation of each output feature may vary. As a consequence, **the receptive field size may not be shift-invariant** . This is illustrated in the figure below with a toy example, in which case the centers of the regions used in the input image are different for the two paths from input to output.

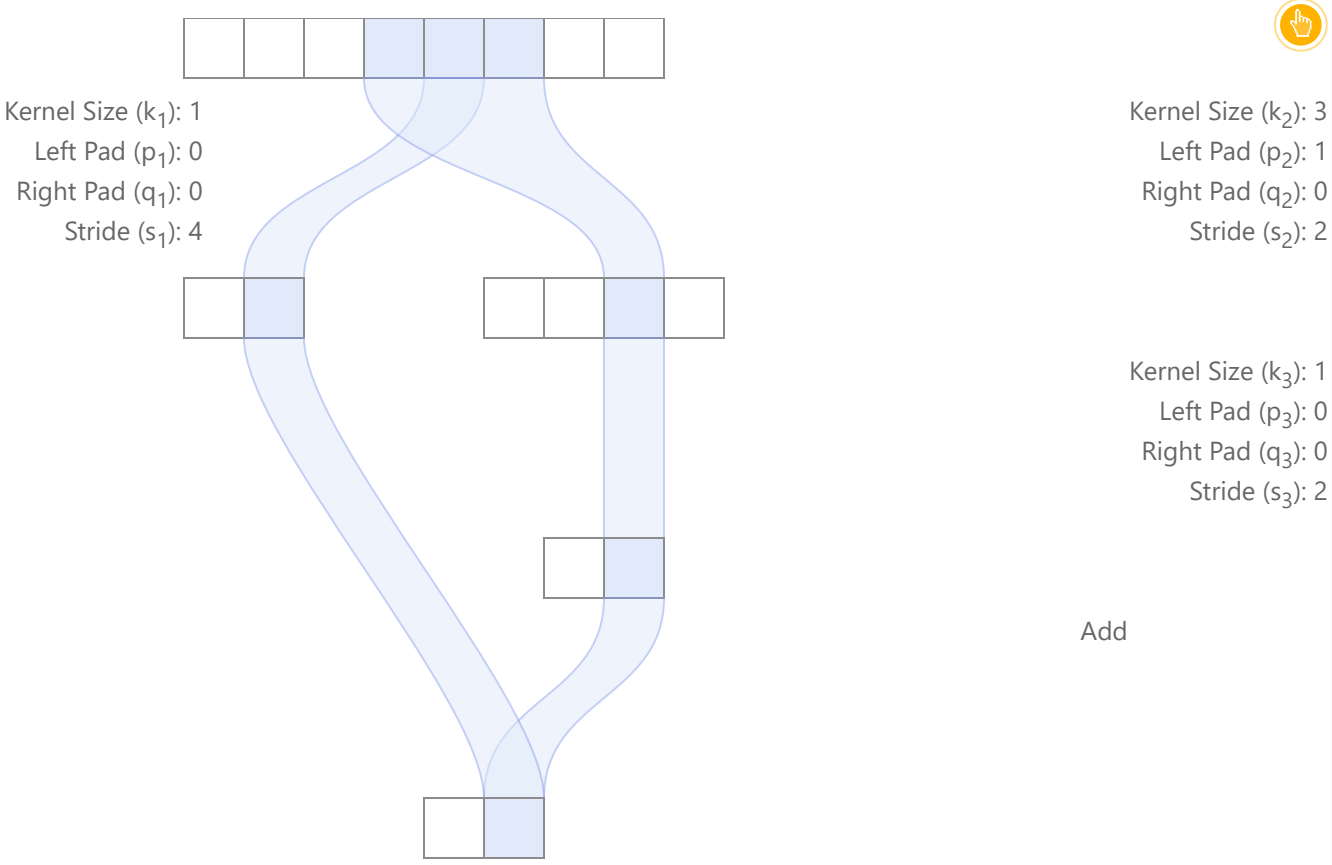


In this example, padding is used only for the left branch. The first three layers are convolutional, while the last layer performs a simple addition. The relative position between the receptive field regions of the left and right paths is inconsistent for different output features, which leads to a lack of alignment (this can be seen by hovering over the different output features). Also, note that the receptive field size for each output feature may be different. For the second feature from the left, 6 input samples are used, while only 5 are used for the third feature. This means that the receptive field size may not be shift-invariant when the network is not aligned.

For many computer vision tasks, it is highly desirable that output features be aligned: "image-to-image translation" tasks (e.g., semantic segmentation, edge detection, surface normal estimation, colorization, etc), local feature matching and retrieval, among others.

When the network is aligned, all different paths lead to output features being centered consistently in the same locations. All different paths must have the same effective stride. It is easy to see that the receptive field size will be the largest receptive field among all possible paths. Also, the effective padding of the network corresponds to the effective padding for the path with largest receptive field size, such that one can apply (8), (11) to localize the region which generated an output feature.

The figure below gives one simple example of an aligned network. In this case, the two different paths lead to the features being centered at the same locations. The receptive field size is 3, the effective stride is 4 and the effective padding is 1.



Alignment criteria . More precisely, for a network to be aligned at every layer, we need every possible pair of paths i and j to have $c_l^{(i)} = c_l^{(j)}$ for any layer l and output feature u_L. For this to happen, we can see from (11) that two conditions must be satisfied:

$$S_l^{(i)} = S_l^{(j)} \tag{12}$$

$$-P_l^{(i)} + \left(\frac{r_l^{(i)} - 1}{2}\right) = -P_l^{(j)} + \left(\frac{r_l^{(j)} - 1}{2}\right) \tag{13}$$

for all i, j, l.

Algorithm for computing receptive field parameters: sketch. It is straightforward to develop an efficient algorithm that computes the receptive field size and associated parameters for such computation graphs. Naturally, a brute-force approach is to use the expressions presented above to compute the receptive field parameters for each route from the input to output independently, coupled with some bookkeeping in order to compute the parameters for the entire network. This method has a worst-case complexity of $\mathcal{O}\left(|\mathcal{E}| \times |\mathcal{L}|\right)$.

But we can do better. Start by topologically sorting the computation graph. The sorted representation arranges the layers in order of dependence: each layer's output only depends on layers that appear before it. By visiting layers in reverse topological order, we ensure that all paths from a given layer l to the output layer L have been taken into account when l is visited. Once the input layer $l = 0$ is reached, all paths have been considered and the receptive field parameters of the entire model are obtained. The complexity of this algorithm is $\mathcal{O}\left(|\mathcal{E}| + |\mathcal{L}|\right)$, which is much better than the brute-force alternative.

As each layer is visited, some bookkeeping must be done in order to keep track of the network's receptive field parameters. In particular, note that there might be several different paths from layer l to the output layer L. In order to handle this situation, we keep track of the parameters for l and update them if a new path with larger receptive field is found, using expressions (1), (9) and (10). Similarly, as the graph is traversed, it is important to check that the network is aligned. This can be done by making sure that the receptive field parameters of different paths satisfy (12) and (13).

Discussion: receptive fields of modern networks

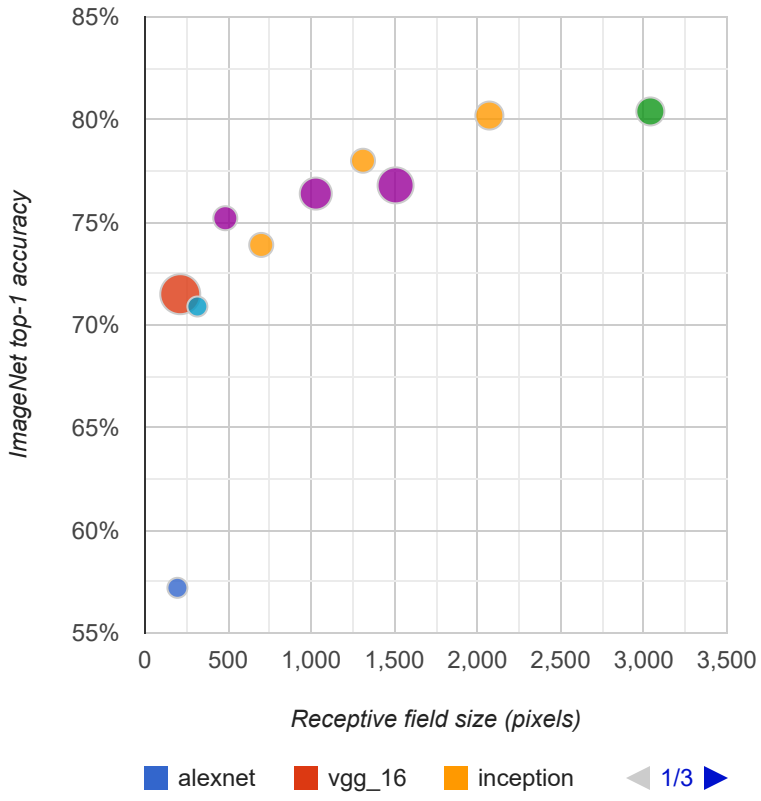
In this section, we present the receptive field parameters of modern convolutional networks [5], which were computed using the new open-source library (script here). The pre-computed parameters for AlexNet [11], VGG [12], ResNet [13], Inception [14] and MobileNet [16] are presented in the table below. For a more comprehensive list, including intermediate network end-points, see this table.

ConvNet Model	Receptive Field (r)	Effective Stride (S)	Effective Padding (P)	Model Year
alexnet_v2	195	32	64	2014
vgg_16	212	32	90	2014
mobilenet_v1	315	32	126	2017
mobilenet_v1_075	315	32	126	2017
resnet_v1_50	483	32	239	2015
inception_v2	699	32	318	2015
resnet_v1_101	1027	32	511	2015
inception_v3	1311	32	618	2015
resnet_v1_152	1507	32	751	2015
resnet_v1_200	1763	32	879	2015
inception_v4	2071	32	998	2016

ConvNet Model	Receptive Field (r)	Effective Stride (S)	Effective Padding (P)	Model Year
inception_resnet_v2	3039	32	1482	2016

As models evolved, from AlexNet, to VGG, to ResNet and Inception, the receptive fields increased (which is a natural consequence of the increased number of layers). In the most recent networks, the receptive field usually covers the entire input image: this means that the context used by each feature in the final output feature map includes all of the input pixels.

We can also relate the growth in receptive fields to increased classification accuracy. The figure below plots ImageNet top-1 accuracy as a function of the network's receptive field size, for the same networks listed above. The circle size for each data point is proportional to the number of floating-point operations (FLOPs) for each architecture.



We observe a logarithmic relationship between classification accuracy and receptive field size, which suggests that large receptive fields are necessary for high-level recognition tasks, but with diminishing rewards. For example, note how MobileNets achieve high recognition performance even if using a very compact architecture: with depth-wise convolutions, the receptive field is increased with a small compute footprint. In comparison, VGG-16 requires 27X more FLOPs than MobileNets, but produces a smaller receptive field size; even if much more complex, VGG's accuracy is only slightly better than MobileNet's. This suggests that networks which can efficiently generate large receptive fields may enjoy enhanced recognition performance.

Let us emphasize, though, that the receptive field size is not the only factor contributing to the improved performance mentioned above. Other factors play a very important role: network depth (i.e., number of layers) and width (i.e., number of filters per layer), residual connections, batch normalization, to name only a few. In other words, while we conjecture that a large receptive field is necessary, by no means it is sufficient. [6]

Finally, note that a given feature is not equally impacted by all input pixels within its receptive field region: the input pixels near the center of the receptive field have more "paths" to influence the feature, and consequently carry more weight. The relative importance of each input pixel defines the *effective receptive field* of the feature. Recent work [17] provides a mathematical formulation and a procedure to measure effective receptive fields, experimentally observing a Gaussian shape, with the peak at the receptive field center. Better understanding the relative importance of input pixels in convolutional neural networks is an active research topic.

Solving recurrence equations: receptive field size

The first trick to solve (1) is to multiply it by $\prod_{i=1}^{l-1} s_i$:

$$\begin{aligned} r_{l-1} \prod_{i=1}^{l-1} s_i &= s_l \cdot r_l \prod_{i=1}^{l-1} s_i + (k_l - s_l) \prod_{i=1}^{l-1} s_i \\ &= r_l \prod_{i=1}^{l} s_i + k_l \prod_{i=1}^{l-1} s_i - \prod_{i=1}^{l} s_i \end{aligned} \tag{14}$$

Then, define $A_l = r_l \prod_{i=1}^{l} s_i$, and note that $\prod_{i=1}^{0} s_i = 1$ (since 1 is the neutral element for multiplication), so $A_0 = r_0$. Using this definition, (14) can be rewritten as:

$$A_l - A_{l-1} = \prod_{i=1}^{l} s_i - k_l \prod_{i=1}^{l-1} s_i \tag{15}$$

Now, sum it from $l = 1$ to $l = L$:

$$\sum_{l=1}^{L}(A_l - A_{l-1}) = A_L - A_0 = \sum_{l=1}^{L}\left(\prod_{i=1}^{l} s_i - k_l \prod_{i=1}^{l-1} s_i\right) \tag{16}$$

Note that $A_0 = r_0$ and $A_L = r_L \prod_{i=1}^{L} s_i = \prod_{i=1}^{L} s_i$. Thus, we can compute:

$$\begin{aligned} r_0 &= \prod_{i=1}^{L} s_i + \sum_{l=1}^{L}\left(k_l \prod_{i=1}^{l-1} s_i - \prod_{i=1}^{l} s_i\right) \\ &= \sum_{l=1}^{L} k_l \prod_{i=1}^{l-1} s_i - \sum_{l=1}^{L-1} \prod_{i=1}^{l} s_i \\ &= \sum_{l=1}^{L} k_l \prod_{i=1}^{l-1} s_i - \sum_{l=1}^{L} \prod_{i=1}^{l-1} s_i + 1 \end{aligned} \tag{17}$$

where the last step is done by a change of variables for the right term.

Finally, rewriting (17), we obtain the expression for the receptive field size r_0 of an FCN at the input image, given the parameters of each layer:

$$r_0 = \sum_{l=1}^{L}\left((k_l - 1)\prod_{i=1}^{l-1} s_i\right) + 1 \tag{18}$$

Navigate back to the main text

Solving recurrence equations: receptive field region

The derivations are similar to the one we use to solve (1). Let's consider the computation of u_0. First, multiply (3) by $\prod_{i=1}^{l-1} s_i$.

$$\begin{aligned} u_{l-1}\prod_{i=1}^{l-1} s_i &= u_l \cdot s_l \prod_{i=1}^{l-1} s_i - p_l \prod_{i=1}^{l-1} s_i \\ &= u_l \prod_{i=1}^{l} s_i - p_l \prod_{i=1}^{l-1} s_i \end{aligned} \tag{19}$$

Then, define $B_l = u_l \prod_{i=1}^{l} s_i$, and rewrite (19) as:

$$B_l - B_{l-1} = p_l \prod_{i=1}^{l-1} s_i \tag{20}$$

And sum it from $l = 1$ to $l = L$:

$$\sum_{l=1}^{L}(B_l - B_{l-1}) = B_L - B_0 = \sum_{l=1}^{L} p_l \prod_{i=1}^{l-1} s_i \tag{21}$$

Note that $B_0 = u_0$ and $B_L = u_L \prod_{i=1}^{L} s_i$. Thus, we can compute:

$$u_0 = u_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} p_l \prod_{i=1}^{l-1} s_i \tag{22}$$

Navigate back to the main text

Other network operations

Dilated (atrous) convolution. Dilations introduce "holes" in a convolutional kernel. While the number of weights in the kernel is unchanged, they are no longer applied to spatially adjacent samples. Dilating a kernel by a factor of α introduces striding of α between the samples used when computing the convolution. This means that the spatial span of the kernel ($k > 0$) is increased to $\alpha(k-1)+1$. The above derivations can be reused by simply replacing the kernel size k by $\alpha(k-1)+1$ for all layers using dilations.

Upsampling. Upsampling is frequently done using interpolation (e.g., bilinear, bicubic or nearest-neighbor methods), resulting in an equal or larger receptive field — since it relies on one or multiple features from the input. Upsampling layers generally produce output features which depend locally on input features, and for receptive field computation purposes can be considered to have a kernel size equal to the number of input features involved in the computation of an output feature.

Separable convolutions. Convolutions may be separable in terms of spatial or channel dimensions. The receptive field properties of the separable convolution are identical to its corresponding equivalent non-separable convolution. For example, a 3×3 depth-wise separable convolution has a kernel size of 3 for receptive field computation purposes.

Batch normalization. At inference time, batch normalization consists of feature-wise operations which do not alter the receptive field of the network. During training, however, batch normalization parameters are computed based on all activations from a specific layer, which means that its receptive field is the whole input image.

Navigate back to the main text

Acknowledgments

We would like to thank Yuning Chai and George Papandreou for their careful review of early drafts of this manuscript. Regarding the open-source library, we thank Mark Sandler for helping with the starter code, Liang-Chieh Chen and Iaroslav Tymchenko for careful code review, and Till Hoffman for improving upon the original code release. Thanks also to Mark Sandler for assistance with model profiling.

Footnotes

1. A more general definition of padding may also be considered: negative padding, interpreted as cropping, can be used in our derivations without any changes. In order to make the article more concise, our presentation focuses solely on non-negative padding. [↩]

2. We adopt the convention where the first output feature for each layer is computed by placing the kernel on the left-most position of the input, including padding. This convention is adopted by all major deep learning libraries. [↩]

3. As in the illustration below, note that, in some cases, the receptive field region may contain "holes", i.e., some of the input features may be unused for a given layer. [↩]

4. Due to border effects, note that the size of the region in the original image which is used to compute each output feature may be different. This happens if padding is used, in which case the receptive field for border features includes the padded region. Later in the article, we discuss how to compute the receptive field region for each feature, which can be used to determine exactly which image pixels are used for each output feature. [↩]

5. The models used for receptive field computations, as well as the accuracy reported on ImageNet experiments, are drawn from the TF-Slim image classification model library. [↩]

6. Additional experimentation is needed to confirm this hypothesis: for example, researchers may experimentally investigate how classification accuracy changes as kernel sizes and strides vary for different architectures. This may indicate if, at least for those architectures, a large receptive field is necessary. [↩]

References

1. **Visualizing higher-layer features of a deep network** [PDF]
   Erhan, D., Bengio, Y., Courville, A. and Vincent, P., 2009. University of Montreal, Vol 1341, pp. 3.

2. **Visualizing and Understanding Convolutional Networks** [PDF]
   Zeiler, M.D. and Fergus, R., 2014. Proc. ECCV.

3. **Global Optimality in Neural Network Training** [PDF]
   Haeffele, B. and Vidal, R., 2017. Proc. CVPR.

4. **On the Global Convergence of Gradient Descent for Over-parameterized Models using Optimal Transport** [PDF]
   Chizat, L. and Bach, F., 2018. Proc. NIPS.

5. **Object Detectors Emerge in Deep Scene CNNs** [PDF]
   B. Zhou, A.K. and Torralba, A., 2015. Proc. ICLR.

6. **Interpreting Deep Visual Representations via Network Dissection** [PDF]
   Zhou, B., Bau, D., Oliva, A. and Torralba, A., 2018. IEEE Transactions on Pattern Analysis and Machine Intelligence.

7. **In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning** [PDF]
   Neyshabur, B., Tomioka, R. and Srebro, N., 2014.

8. **Understanding Deep Learning Requires Rethinking Generalization** [PDF]
   Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O., 2017. Proc. ICLR.

9. **A Guide to Receptive Field Arithmetic for Convolutional Neural Networks** [link]
   Dang-Ha, T., 2017.

10. **What are the Receptive, Effective Receptive, and Projective Fields of Neurons in Convolutional Neural Networks?** [PDF]
    Le, H. and Borji, A., 2017.

11. **ImageNet Classification with Deep Convolutional Neural Networks** [PDF]
    Krizhevsky, A., Sutskever, I. and Hinton, G., 2012. Proc. NIPS.

12. **Very Deep Convolutional Networks for Large-Scale Image Recognition** [PDF]
    Simonyan, K. and Zisserman, A., 2015. Proc. ICLR.

13. **Deep Residual Learning for Image Recognition** [PDF]
    He, K., Zhang, X., Ren, S. and Sun, J., 2016. Proc. CVPR.

14. **Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning** [PDF]
    Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A., 2016.

15. **Large-Scale Image Retrieval with Attentive Deep Local Features** [PDF]
    Noh, H., Araujo, A., Sim, J., Weyand, T. and Han, B., 2017. Proc. ICCV.

16. **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications** [PDF]
    Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017.

17. **Understanding the Effective Receptive Field in Deep Convolutional Neural Networks** [PDF]
    Luo, W., Li, Y., Urtasun, R. and Zemel, R., 2016. Proc. NIPS.

Updates and

Corrections

If you see mistakes or want to suggest changes, please create an issue on GitHub.

Reuse

Citation

For attribution in academic contexts, please cite this work as

```
Araujo, et al., "Computing Receptive Fields of Convolutional Neural Networks", Distill, 2019.
```

BibTeX citation

```
@article{araujo2019computing,
  author = {Araujo, André and Norris, Wade and Sim, Jack},
  title = {Computing Receptive Fields of Convolutional Neural Networks},
  journal = {Distill},
  year = {2019},
  note = {https://distill.pub/2019/computing-receptive-fields},
  doi = {10.23915/distill.00021}
}
```

Distill is dedicated to clear explanations of machine learning

About Submit Prize Archive RSS GitHub Twitter ISSN 2476-0757