



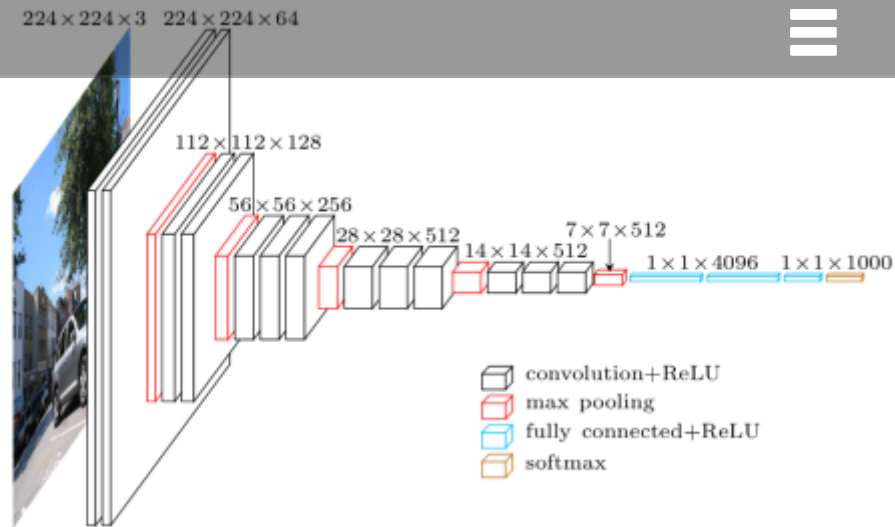
Alexis Cook

9 Apr 2017 • on [keras localization](#)

Global Average Pooling Layers for Object Localization

For image classification tasks, a common choice for convolutional neural network (CNN) architecture is repeated blocks of convolution and max pooling layers, followed by two or more densely connected layers. The final dense layer has a softmax activation function and a node for each potential object category.

As an example, consider the VGG-16 model architecture, depicted in the figure below.



We can summarize the layers of the VGG-16 model by executing the following line of code in the terminal:

```
python -c 'from keras.applications.vgg16 import VGG16; VGG16().summary()'
```

Your output should appear as follows:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]

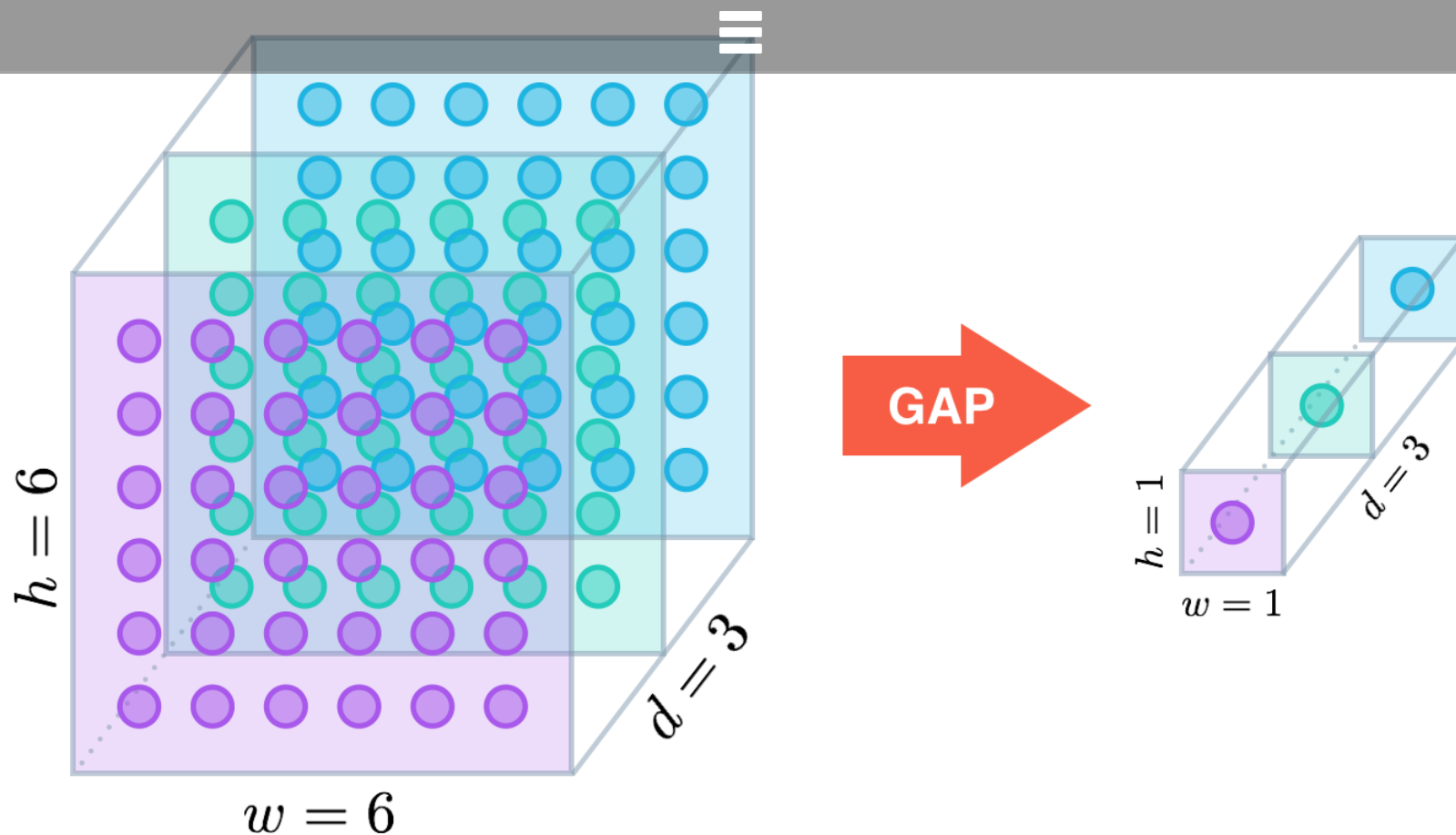
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
flatten (Flatten)	(None, 25088)	0	block5_pool[0][0]
fc1 (Dense)	(None, 4096)	102764544	flatten[0][0]
fc2 (Dense)	(None, 4096)	16781312	fc1[0][0]
predictions (Dense)	(None, 1000)	4097000	fc2[0][0]
=====			
Total params: 138357544			

You will notice five blocks of (two to three) convolutional layers followed by a max pooling layer. The final max pooling layer is then flattened and followed by three densely connected layers. Notice that most of the parameters in the model belong to the fully connected layers!

As you can probably imagine, an architecture like this has the risk of overfitting to the training dataset. In practice, dropout layers are used to avoid overfitting.

Global Average Pooling

In the last few years, experts have turned to global average pooling (GAP) layers to minimize overfitting by reducing the total number of parameters in the model. Similar to max pooling layers, GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor. However, GAP layers perform a more extreme type of dimensionality reduction, where a tensor with dimensions $h \times w \times d$ is reduced in size to have dimensions $1 \times 1 \times d$. GAP layers reduce each $h \times w$ feature map to a single number by simply taking the average of all hw values.



The [first paper](#) to propose GAP layers designed an architecture where the final max pooling layer contained one activation map for each image category in the dataset. The max pooling layer was then fed to a GAP layer, which yielded a vector with a single entry for each possible object in the classification task. The authors then applied a softmax activation function to yield the predicted probability of each class. If you peek at the [original paper](#), I especially recommend checking out Section 3.2, titled "Global Average Pooling".

as a detector for a different pattern in the image, localized in space. To get the class activation map corresponding to an image, we need only to transform these detected patterns to detected objects.

This transformation is done by noticing each node in the GAP layer corresponds to a different activation map, and that the weights connecting the GAP layer to the final dense layer encode each activation map's contribution to the predicted object class. To obtain the class activation map, we sum the contributions of each of the detected patterns in the activation maps, where detected patterns that are more important to the predicted object class are given more weight.

How the Code Operates

Let's examine the ResNet-50 architecture by executing the following line of code in the terminal:

```
python -c 'from keras.applications.resnet50 import ResNet50; ResNet50().summary()'
```

The final few lines of output should appear as follows (*Notice that unlike the VGG-16 model, the majority of the trainable parameters are not located in the fully connected layers at the top of the network!*):

The **ResNet-50 model** takes a less extreme approach; instead of getting rid of dense layers altogether, the GAP layer is followed by one densely connected layer with a softmax activation function that yields the predicted object classes.

Object Localization

In mid-2016, **researchers at MIT** demonstrated that CNNs with GAP layers (a.k.a. GAP-CNNs) that have been trained for a classification task can also be used for **object localization**. That is, a GAP-CNN not only tells us *what* object is contained in the image - it also tells us *where* the object is in the image, and through no additional work on our part! The localization is expressed as a heat map (referred to as a class activation map), where the color-coding scheme identifies regions that are relatively important for the GAP-CNN to perform the object identification task. Please check out the YouTube video below for an *awesome* demo!



In the **repository**, I have explored the localization ability of the pre-trained ResNet-50 model, using the technique from **this paper**. The main idea is that each of the activation maps in the final layer preceding the GAP layer acts

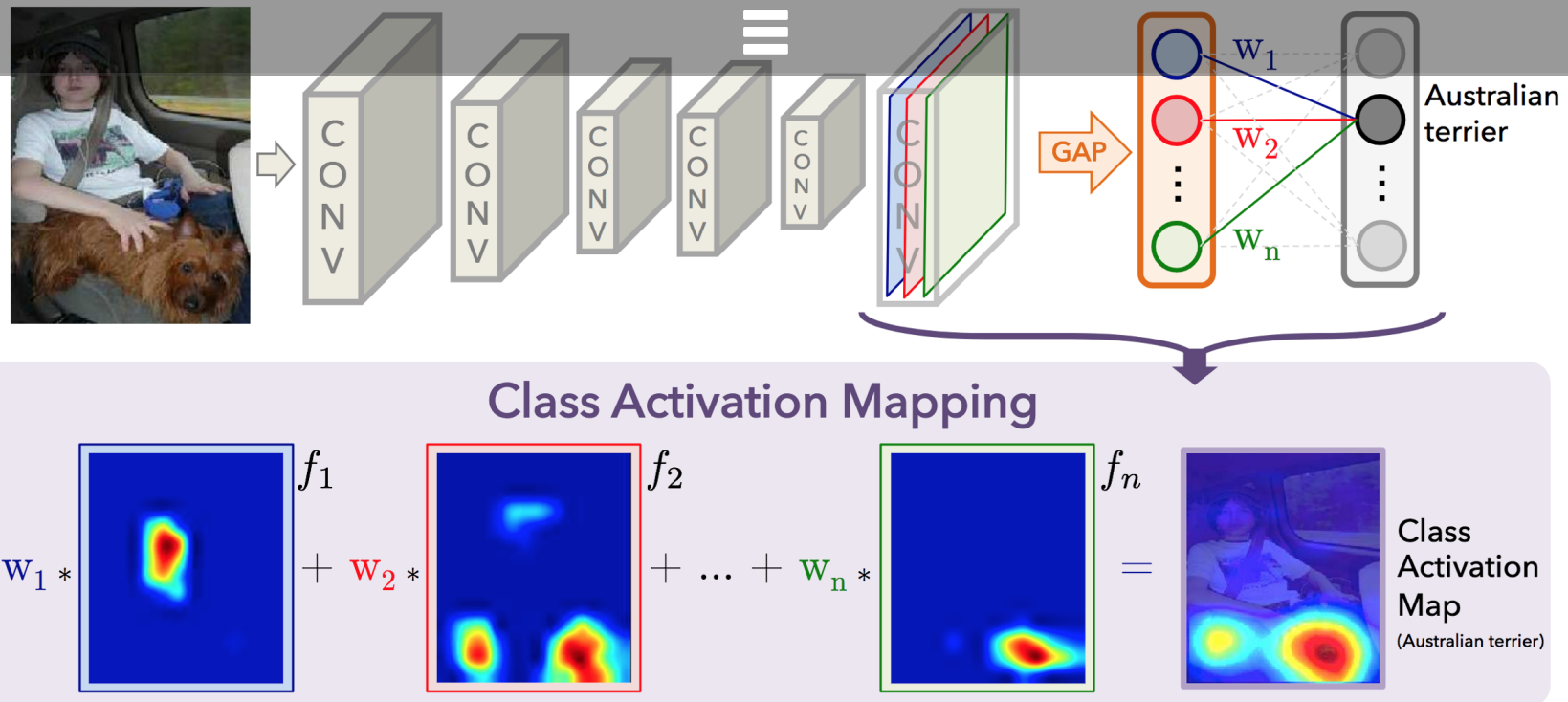
res5c_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_48[0][0]
bn5c_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5c_branch2c[0][0]
add_16 (Add)	(None, 7, 7, 2048)	0	bn5c_branch2c[0][0] activation_46[0][0]
activation_49 (Activation)	(None, 7, 7, 2048)	0	add_16[0][0]
avg_pool (AveragePooling2D)	(None, 1, 1, 2048)	0	activation_49[0][0]
flatten_1 (Flatten)	(None, 2048)	0	avg_pool[0][0]
fc1000 (Dense)	(None, 1000)	2049000	flatten_1[0][0]
=====			
Total params: 25,636,712			
Trainable params: 25,583,592			
Non-trainable params: 53,120			

The `Activation`, `AveragePooling2D`, and `Dense` layers towards the end of the network are of the most interest to us. Note that the `AveragePooling2D` layer is in fact a GAP layer!

We'll begin with the `Activation` layer. This layer contains 2048 activation maps, each with dimensions 7×7 . Let f_k represent the k -th activation map, where $k \in \{1, \dots, 2048\}$.

The following `AveragePooling2D` GAP layer reduces the size of the preceding layer to $(1, 1, 2048)$ by taking the average of each feature map. The next `Flatten` layer merely flattens the input, without resulting in any change to the information contained in the previous GAP layer.

The object category predicted by ResNet-50 corresponds to a single node in the final `Dense` layer; and, this single node is connected to every node in the preceding `Flatten` layer. Let w_k represent the weight connecting the k -th node in the `Flatten` layer to the output node corresponding to the predicted image category.



Then, in order to obtain the class activation map, we need only compute the sum

$$w_1 \cdot f_1 + w_2 \cdot f_2 + \dots + w_{2048} \cdot f_{2048}.$$

You can plot these class activation maps for any image of your choosing, to explore the localization ability of ResNet-50. Note that in order to permit comparison to the original image, **bilinear upsampling** is used to resize each activation map to **224 × 224**. (This results in a class activation map with size **224 × 224**.)



If you'd like to use this code to do your own object localization, you need only download the [repository](#).

31 Comments

Alexis Cook

 Disqus' Privacy Policy

 Washington Pago... ▾

 Recommend 31

 Tweet

 Share

Sort by Best ▾



Join the discussion...



Rahul Deora • 2 years ago

This is a pretty incomplete post, not explaining why GAP reduces overfitting. Imagine I show my network images of dogs all in the right hand side. So now my final activation map before flatten layers will only have high activations on right side and the connections to flatten will be trained to only have high weights in that

region. So if my dog is now on left, that region does **not** have high weights to fc layer and is not detected.

By doing a global average we make the convolution invariant to where the object of interest is and this acts as a data augmentation of moving the object around to different regions. This prevents overfitting of the fc layers.

23 ^ | v • Reply • Share ›



Florian • 2 years ago

Nice explanation!

For a finer detection, one can introduce an upsampling path in the architecture of the network, before computing the CAMs. Check this paper (GP-Unet) for details:

<https://arxiv.org/abs/1705....>

1 ^ | v • Reply • Share ›



Evgeny Demidov • 7 months ago

Thank you for your post. Look at "CNN Heat Maps CAM" interactive demo

<https://www.ibiblio.org/e-n...>

^ | v • Reply • Share ›



dinial utami • 10 months ago

so the GAP (global average pooling) is average pooling? why must give global?

^ | v • Reply • Share ›



montaser Ali Said • a year ago

thanks Alexis so much , very good post .

^ | v • Reply • Share ›



J Kent • a year ago

The post is fine. Its intended that readers do some of the work.

If we agree to disagree, let's look beyond the post.

Its a very interesting development, that we can visualize activations in this way to

give a sense of what is being processed. And the video showing where the NN is looking is very cool.

One important point being, this information was there all along, and nobody picked-up on the significance.

This paper in section 3.2 (which is linked above) provides a reasonable explanation of the role of GAP <https://arxiv.org/pdf/1312.....> .

Its a filter. All of these steps represent filters.

Thank you to the original poster for pointing out this observation.

Please everyone take care !

^ | v • Reply • Share ›



Alexandre Huat • a year ago

Crystal clear. Thanks.

^ | v • Reply • Share ›



Hichem Bro • a year ago

Can I use the GAP illustration for my project?

^ | v • Reply • Share ›



Darshan Iyer • a year ago

Wonderful explanation Alexis. Very clear.

^ | v • Reply • Share ›



Mahmood • 2 years ago

Hi Alex

I am using ChestXray to test Resnet50 , this dataset explored here

<https://stanfordmlgroup.git...>

I am trying to get simliar results and I followed what is explained in this paper but I could not get such results .

The dataset is very imbalanced . do you have any idea how I can improve AUC .

  • Reply • Share ›**Sathiya Chakravarthy** • 2 years ago

So I am running a neural network using global max pooling after several convolution layers. Do I need a fully connected layer after the Global pooling or should my global max pooling be connected directly to softmax classifier?

  • Reply • Share ›**Michał Woś** → Sathiya Chakravarthy • 2 years ago • edited



If a number of your classes (let's say n) is different than a depth (let's say d) of your last conv layer, then you need FC layer in the middle. Because output of GAP gives you a tensor of shape $1 \times 1 \times d$ and if you flatten that, you've got tensor with d elements, whereas you expect n -number input to softmax. So those d elements have to be combined e.g. by FC layer of size n .

If $d=n$, then flattening of GAP output is enough to match shapes..

  • Reply • Share ›**Bruce Dai** → Michał Woś • 2 years ago

Exactly, like in the Network In Network, so we can directly use the single feature map before the GAP which corresponds to the category as the class activation map, right?

Which architecture do you think is better? With a FC, and get a weighted average of previous feature maps as the class activation map. Or no FC, simply use one feature map for each category.

6   • Reply • Share ›

**Michał Woś** → Bruce Dai • 2 years ago

I believe both gonna work well for classification, however the only one I tested in the past is the second one - one feature map for each category and then GAP. Both approaches have the same number of parameters (assuming using bottleneck conv with kernel 1×1 without bias for the second approach

you mentioned). The difference is in required RAM during evaluation though - solution with conv needs more.

^ | v • Reply • Share ›



weakish • 2 years ago

Hi, I'd like to translate this post to Chinese. Can you give me the permission? The translated text will be published at ai.jqr.com and related Chinese social network accounts. Thanks.

^ | v • Reply • Share ›



Alexis Cook Mod → weakish • 2 years ago

Yes, you have permission. Please provide a link here when it's ready - I'd love to see!

^ | v • Reply • Share ›



weakish → Alexis Cook • 2 years ago

I finished the translation last Friday, and the translated text is published at <https://www.jqr.com/article...> (Credit is given in the beginning (in translator's note) and there is a backlink at the end of the translated text.)

^ | v • Reply • Share ›



Ghulam Rasool • 2 years ago

Thanks

^ | v • Reply • Share ›



JMarc • 2 years ago

Hi Alexis, thank you for the post. A really clear and concise explanation of the working of the CAM. Thank you for sharing! I find it surprising that despite the significant upsampling (*32), the CAM has somehow good resolution: i.e follows pretty accurately the head of the dogs. What's also surprising is that the body of the dogs is not really used by the model to predict the breed.

^ | v • Reply • Share ›



Nick Nobutaka Toyoda Kim • 2 years ago • edited



Hi, I'm a Udacity deep learning student and you're my favorite lecturer! But I'm wondering if you can give us a code example using GAP. I'm stuck on that in the dog project as I cannot figure out what the parameters need to be. Thanks!

^ | v • Reply • Share ›



Alexis Cook Mod → Nick Nobutaka Toyoda Kim • 2 years ago

Hi, Nick! I'm glad you're enjoying the lesson! Here's a code example: <https://github.com/udacity/...> The last video in the CNN lesson (titled *Transfer Learning in Keras*) walks through the code in the notebook.

^ | v • Reply • Share ›



Nick Nobutaka Toyoda Kim → Alexis Cook • 2 years ago

Thanks Alexis! I've really gotta go back through all the videos again.

^ | v • Reply • Share ›



peerajak • 3 years ago

Hi, I'm student at Udacity deep learning nanodegree. I don't understand the global average pooling. Global average pooling reduce f0-f2047 from 27x27 to 1x1. So the result is a vector of 1x1x2047. Suppose the output node is 12 class softmax classifier, we will have dense layer of weight size 2047x12.

My question is, if the above understanding is correct, where is "This sum is the 224x224 class activation map.", and what is w_0 - w_2047 in your last paragraph.

Thank you very much.

^ | v • Reply • Share ›



Alexis Cook Mod → peerajak • 3 years ago • edited

Hi Peerajak, thanks for your message!

The final layer that precedes the GAP layer has shape 7x7x2048.

You can use bilinear upsampling to increase the size of this layer to 224x224x2048. This is done in line 39 of the `ResNet_CAM.py` file in the repository.

So, now we can think of each of this upsampled layer as composed of 2048 upsampled *activation maps*, each with size 224x224. These are denoted $f_0, f_1, \dots, f_{2047}$ above.

The *class activation map* has size 224x224 and is exactly equal to $w_0 * f_0 + w_1 * f_1 + \dots + w_{2047} * f_{2047}$.

... where I still have to tell you what the w_0, \dots, w_{2047} are! Towards that end ...

Remember the final convolutional layer of ResNet-50? It was followed (a few layers later ...) by a global average pooling layer, and that global average pooling layer had size 1x1x2048. This was then followed by a flatten layer, which had size 2048. Then there was a dense layer (with 1000 nodes, one for each category of imagenet).

The $w_0, w_1, \dots, w_{2047}$ potentially vary with the image that you supply to the code, and they depend on the ImageNet category that ResNet predicts for the image!

Here's how the $w_0, w_1, \dots, w_{2047}$ are determined:

Step 1) Pass the image through ResNet, and see what ImageNet category is predicted. This ImageNet category corresponds to one node in the final/output dense layer.

Step 2) That one node in the final layer (from Step 1) is connected to ALL of the 2048 nodes in the previous flatten layer. There are exactly 2048 weights corresponding to those connections (plus 1, if you include the bias, but we will ignore that). Those 2048 weights correspond to the $w_0, w_1, \dots, w_{2047}$.

Let me know if anything is unclear, and thanks for reaching out!

(Regarding what you said above, it is conceptually correct, but just off by a number or two ... so, global average pooling reduces f_0 - f_{2047} to $1 \times 1 \times 2048$ [since we're indexing from zero]. Then, if the output layer is for 12 classes, and you include the bias, there are $(2048 + 1) \times 12$ weights. If you ignore the bias, there are 2048×12 weights.)

1 ^ | v • Reply • Share ›



Quang Vu ➔ Alexis Cook • 2 years ago

Hi Alexis, thank you for writing this blog post!

The unclear point for me is how "upsampling" work. From the SciPy link you gave, it seems to zoom the output from the feature layer $(7, 7, 2048)$ to match the original image $(224, 224, 2048)$, before passing to the final dense layer. In your notation, the $(7, 7, 2048)$ output is called "k" in $\{1 \dots 2048\}$

I am trying to find a correct understanding of that "k" output. Is each k-output a piece of the original image, or is it a simplified version of the image where all the patterns remains the same location in the original image?

In the former case, all the geo location mapping is lost during training, isn't it?

In the latter case, the training would not be transformation invariant, would it? For example, if we rotate the image, we will have a rotated plot. Everything would seem to be location-related, and not connectivity-related.

^ | v • Reply • Share ›



Alexis Cook Mod ➔ Quang Vu • 2 years ago

Hello! The "k-output" (or the feature map f_k above) is (not a piece of the original image and instead best thought of as) a simplified version of the image, where the discovered patterns remain in the same (relative) locations as the original image (i.e. with the letter 'a' in the original image).

image (... so, it's the latter case!).



As for your question about invariance, remember that in the case of this network:

- (1) all of the "k-outputs" / feature maps are passed to a global average pooling layer,
- (2) which produces a *single node for each feature map*. The resultant collection of nodes ...
- (3) is then used as input to a fully connected layer (which corresponds to the output layer of the CNN).

If we rotate the input image, as you said, we end up with rotated "k-outputs"/feature maps. But notice that:

- (1) the *values* of all of the entries in the feature maps should be roughly the same (... they are just in different locations),
 - (2) so if I pass these rotated feature maps into a global average pooling layer, I should get the same values as I got [in (2) above] when I used the original [not rotated] input image,
- so I should get the same predicted class.

In this way, the GAP layer should *help* with statistical invariance (i.e., because of the GAP layer, the network is more likely to be rotation/translation/etc invariant), and the fact that a rotated input image leads to rotated feature maps is less likely to be an issue for the network :). I hope I've answered your question! If not, please let me know!

^ | v • Reply • Share ›



peerajak ➔ Alexis Cook • 3 years ago

Thank you. I think I understand now. My understanding is that

- > A set of w_0 - w_{2047} for each class of one-hot encoded output
- > suppose there are 12 classes, we have 12 of the said sets.
- > Suppose we wish to find the *class activation map* we have

weighted linear combination of *activation maps*, which is

$$\text{class_activation_map} = w_0(f_0) + w_1(f_1) + \dots + w_{2047}(f_{2047})$$

where w_i is a scalar, and f_i is a matrix of size 224×224

Since this is a scalar multiplication of a matrix $w_i(f_i)$ is of size 224×224 , and so the same for class_activation_map.

There are 12 class_activation_maps for 12 classes.

Am I correct now?

^ | v • Reply • Share ›



Alexis Cook Mod → peerajak • 3 years ago

Yes, that is nearly perfect. I'd only change the very last sentence.

This is correct: "suppose there are 12 classes, we have 12 of the said sets ($w_0 - w_{2047}$).". However, remember that $f_0 - f_{2047}$ varies with each image input, so the class activation map ($= w_0(f_0) + w_1(f_1) + \dots + w_{2047}(f_{2047})$) will be different for each image.

The class activation map is exactly what is plotted in the dog image examples.

^ | v • Reply • Share ›



peerajak → Alexis Cook • 3 years ago

thank you.

^ | v • Reply • Share ›



ammara habib • 2 years ago

Nice explanation!

I have a question that how flatten layer works? What is the equation behind the flatten layer?

1 • Reply • Share ›



Ricardo Cruz → ammara habib • a month ago

The flatten layer merely reshapes the tensor. Instead of having a matrix

comments powered by Disqus

© 2017 Alexis Cook. All rights reserved.