

You have 2 free member-only stories left this month. [Sign up for Medium and get an extra one](#)

Visualizing Activation Heatmaps using TensorFlow

Areeb Gani

Follow

Jan 24, 2020 · 5 min read

★

It can be beneficial to visualize what the Convolutional Neural Network values when it does a prediction, as it allows us to see whether our model is on track, as well as what features it finds important. For example, in determining whether an image is human or not, our model may find that facial features are determining factors.

To visualize the heatmap, we will use a technique called Grad-CAM (Gradient Class Activation Map). The idea behind it is quite simple; to find the importance of a certain class in our model, we simply take its gradient with respect to the final convolutional layer and then weigh it against the output of this layer.

Francois Chollet, the author of Deep Learning with Python and the creator of Keras, says, “one way to understand this trick is that we are weighting a spatial map of how intensely the input image activates different channels by how important each channel is with regard to the class, resulting in a spatial map of how intensely the input image activates the class.”

This is the layout of using Grad-CAM:

```
1) Compute the model output and last convolutional layer output for the image.

2) Find the index of the winning class in the model output.

3) Compute the gradient of the winning class with respect to the last convolutional layer.

3) Average this, then weigh it with the last convolutional layer (multiply them).

4) Normalize between 0 and 1 for visualization

5) Convert to RGB and layer it over the original image.
```

Let's start by importing what we need.

```
1 import tensorflow as tf
2 import tensorflow.keras.backend as K
3 from tensorflow.keras.applications.inception_v3 import InceptionV3
4 from tensorflow.keras.preprocessing import image
5 from tensorflow.keras.applications.inception_v3 import preprocess_input, decode_predictions
6 import numpy as np
7 import os
8 import matplotlib.pyplot as plt
9 import cv2
10 from google.colab.patches import cv2_imshow # cv2.imshow does not work on Google Colab
import sys
sys.path.append('..')
import imports.py hosted with ❤ by GitHub view raw
```

Now let's load the model. Since the goal of this tutorial is how to generate an activation heatmap, we will just use the Inception V3 model, which is already pretrained. It is trained to classify many different classes.

This model takes in a 299x299 image. According to [Sik-Ho Tsang](#), at “42 layers deep, the computation cost is only about 2.5 higher than that of GoogLeNet and much more efficient than that of VGGNet.” It is a very deep network, which is why it is provided as a pretrained model in the Keras library. The following will print out the architecture of the model — although there are many computations, we are only looking for the final convolutional layer, which lies near the end of the list.

```
1 model = InceptionV3(weights='imagenet')
2
3 model.summary()
modelLoad.py hosted with ❤ by GitHub view raw
```

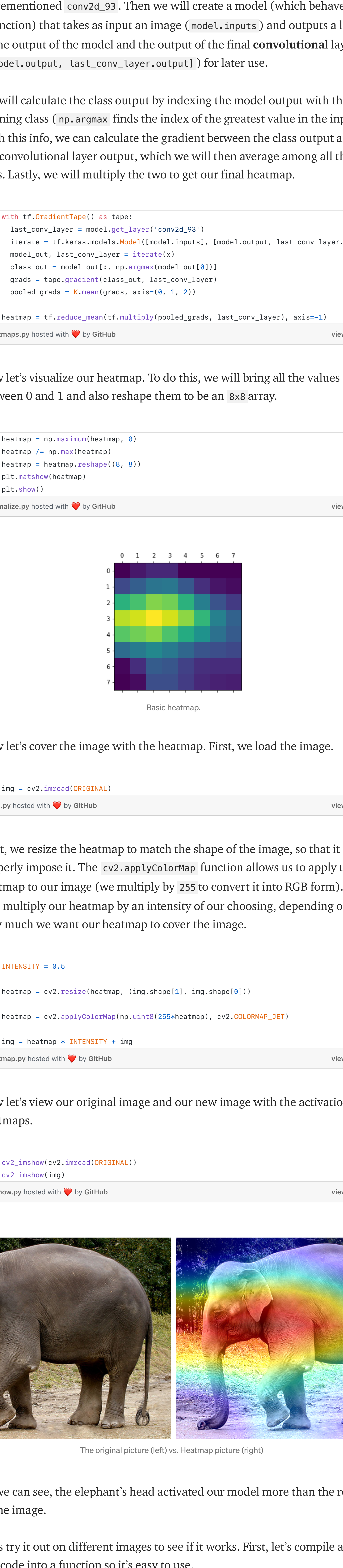
As we can see, the final convolutional layer is `conv2d_93` for this model. Now let's load some images to test and see what it looks like.

The following code downloads multiple images which will be used to demonstrate the Grad-CAM process.

```
1 wget https://indiasendangered.com/wp-content/uploads/2011/09/elephant.jpg
2 wget https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/122341633/11122341633.jpg
3 wget https://icatcare.org/app/uploads/2018/07/Thinking-of-getting-a-cat.png
downloadFiles.py hosted with ❤ by GitHub view raw
```

Let's visualize the image.

```
1 ORIGINAL = 'elephant.jpg'
2
3 DIM = 299
4
5 img = image.load_img(ORIGINAL, target_size=(DIM, DIM))
6
7 cv2_imshow(cv2.imread(ORIGINAL)) # Visualize image
vizImage.py hosted with ❤ by GitHub view raw
```



Indian elephant.

Now let's preprocess the input to feed into our model. We will need to add a dimension to our image and preprocess it as well, using the `preprocess_input` function provided by `tf.keras`.

```
1 x = image.img_to_array(img)
2 x = np.expand_dims(x, axis=0)
3 x = preprocess_input(x)
4
5 preds = model.predict(x)
6 print(decode_predictions(preds))
decodePredictions.py hosted with ❤ by GitHub view raw
```

As we can see, the above picture was predicted as being `Indian_elephant` with a probability of `.962`.

...

Grad-CAM

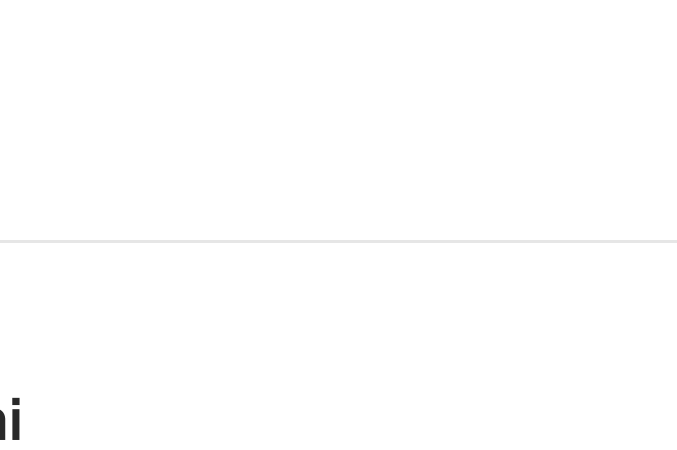
Now we can start the Grad-CAM process. To start, we will need to define a `tf.GradientTape`, so TensorFlow can calculate the gradients (this is a new feature in TF 2). Next, we will get the final convolutional layer, which is the aforementioned `conv2d_93`. Then we will create a model (which behaves as a function) that takes as input an image (`model.inputs`) and outputs a list of the output of the model and the output of the final convolutional layer (`model.output`, `last_conv_layer.output`) for later use.

We will calculate the class output by indexing the model output with the winning class (`np.argmax` finds the index of the greatest value in the input). With this info, we can calculate the gradient between the class output and the convolutional layer output, which we will then average among all the axes. Lastly, we will multiply the two to get our final heatmap.

```
1 with tf.GradientTape() as tape:
2     last_conv_layer = model.get_layer('conv2d_93')
3     iterate = tf.keras.models.Model([model.inputs], [model.output, last_conv_layer.output])
4     model_out, last_conv_layer = iterate(x)
5     class_out = model_out[:, np.argmax(model_out[0])]
6     grads = tape.gradient(class_out, last_conv_layer)
7     pooled_grads = K.mean(grads, axis=(0, 1, 2))
8
9     heatmap = tf.reduce_mean(tf.multiply(pooled_grads, last_conv_layer), axis=-1)
heatmaps.py hosted with ❤ by GitHub view raw
```

Now let's visualize our heatmap. To do this, we will bring all the values between 0 and 1 and also reshape them to be an `8x8` array.

```
1 heatmap = np.maximum(heatmap, 0)
2 heatmap /= np.max(heatmap)
3 heatmap = heatmap.reshape((8, 8))
4 plt.matshow(heatmap)
5 plt.show()
normalize.py hosted with ❤ by GitHub view raw
```



Basic heatmap.

Now let's cover the image with the heatmap. First, we load the image.

```
1 img = cv2.imread(ORIGINAL)
load.py hosted with ❤ by GitHub view raw
```

Next, we resize the heatmap to match the shape of the image, so that it can properly impose it. The `cv2.applyColorMap` function allows us to apply the heatmap to our image (we multiply by 255 to convert it into RGB form). We also multiply our heatmap by an intensity of our choosing, depending on how much we want our heatmap to cover the image.

```
1 INTENSITY = 0.5
2
3 heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
4
5 heatmap = cv2.applyColorMap(np.uint8(255*heatmap), cv2.COLORMAP_JET)
6
7 img = heatmap * INTENSITY + img
heatmap.py hosted with ❤ by GitHub view raw
```

Now let's view our original image and our new image with the activation heatmaps.

```
1 cv2_imshow(cv2.imread(ORIGINAL))
2 cv2_imshow(img)
imshow.py hosted with ❤ by GitHub view raw
```


The original picture (left) vs. Heatmap picture (right)

As we can see, the elephant's head activated our model more than the rest of the image.

Let's try it out on different images to see if it works. First, let's compile all of our code into a function so it's easy to use.

```
1 def gradCAM(orig, intensity=0.5, res=256):
2     img = image.load_img(orig, target_size=(DIM, DIM))
3
4     x = image.img_to_array(img)
5     x = np.expand_dims(x, axis=0)
6     x = preprocess_input(x)
7
8     preds = model.predict(x)
9     print(decode_predictions(preds)[0][0][1]) # prints the class of image
10
11     with tf.GradientTape() as tape:
12         last_conv_layer = model.get_layer('conv2d_93')
13         iterate = tf.keras.models.Model([model.inputs], [model.output, last_conv_layer.output])
14         model_out, last_conv_layer = iterate(x)
15         class_out = model_out[:, np.argmax(model_out[0])]
16         grads = tape.gradient(class_out, last_conv_layer)
17         pooled_grads = K.mean(grads, axis=(0, 1, 2))
18
19         heatmap = tf.reduce_mean(tf.multiply(pooled_grads, last_conv_layer), axis=-1)
20         heatmap = np.maximum(heatmap, 0)
21         heatmap /= np.max(heatmap)
22         heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
23
24         img = cv2.imread(orig)
25
26         heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
27
28         heatmap = cv2.applyColorMap(np.uint8(255*heatmap), cv2.COLORMAP_JET)
29
30         img = heatmap * intensity + img
31
32     cv2_imshow(cv2.resize(cv2.imread(orig), (res, res)))
33     cv2_imshow(cv2.resize(heatmap, (res, res)))
34
35     gradCAM("Chinook-On-White-03.jpg")
36     gradCAM("Thinking-of-getting-a-cat.png")
gradCAM.py hosted with ❤ by GitHub view raw
```


Class: Labrador retriever

Class: tabby

As is evident, our Grad-CAM function can accurately and precisely show us the activation heatmap of the model, telling us what the neural network “sees” and what it values when making its prediction. This could not only improve model explainability but accuracy as well.

...

References

<https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.4-visualizing-what-convnets-learn.ipynb>

<https://stackoverflow.com/questions/58322147/how-to-generate-cnn-heatmaps-using-built-in-keras-in-tf-2-0-tf-keras>

Sign up for Analytics Vidhya News Bytes

By Analytics Vidhya

Latest news from Analytics Vidhya on our Hackathons and some of our best articles! [Take a look.](#)

✉

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

TensorFlow

Heatmap

Keras

Convolutional Network

👁 122

💬 2

WRITTEN BY

Areeb Gani

Follow

Analytics Vidhya

Analytics Vidhya is a community of Analytics and Data Science professionals. We are building the next-gen data science ecosystem <https://www.analyticsvidhya.com>

Follow

Automated subject indexing using multiple algorithms

venkatesh a

Visualising Neurons in a Neural Network

Aman Priyanshu

Version Control for ML Models

Datron in Datatron

Exploring the realms of Natural Language Processing through TEXT-BLOB

Shounakk

Naive Bayes in Machine Learning

Hrishav kumar

Codeless Deep Learning Pipelines with Ludwig and CometML

Cecelia Shao

Why Automated Feature Engineering Will Change the Way You Do Machine Learning

Will Koehrsen in Towards Data Science

[Archived Post] Focal Loss for Dense Object Detection

Jae Duk Seo

Learn more.

Medium is an open platform where 170 million readers come to find insightful and dynamic thinking. Here, expert and undiscovered voices alike dive into the heart of any topic and bring new ideas to the surface. [Learn more](#)

Make Medium yours.

Follow the writers, publications, and topics that matter to you, and you'll see them on your homepage and in your inbox. [Explore](#)

Write a story on Medium.

If you have a story to tell, knowledge to share, or a perspective to offer — welcome home. It's easy and free to post your thinking on any topic. [Start a blog](#)

Medium

About Write Help Legal