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 resepct 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.

```python
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input, decode_predi
import numpy as np
import os
import matplotlib.pyplot as plt
import cv2
from google.colab.patches import cv2_imshow # cv2.imshow does not work on Google Colal
```

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.

```python
model = InceptionV3(weights='imagenet')

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.

```
!wget https://indiasendangered.com/wp-content/uploads/2011/09/elephant.jpg
!wget https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/12234
!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.

```python
ORIGINAL = 'elephant.jpg'

DIM = 299

img = image.load_img(ORIGINAL, target_size=(DIM, DIM))

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`.

```python
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
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.

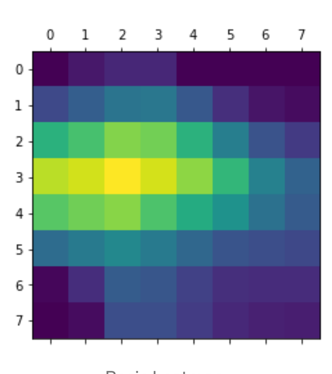
```python
with tf.GradientTape() as tape:
  last_conv_layer = model.get_layer('conv2d_93')
  iterate = tf.keras.models.Model([model.inputs], [model.output, last_conv_layer.outpu
  model_out, last_conv_layer = iterate(x)
  class_out = model_out[:, np.argmax(model_out[0])]
  grads = tape.gradient(class_out, last_conv_layer)
  pooled_grads = K.mean(grads, axis=(0, 1, 2))

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.

```python
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
heatmap = heatmap.reshape((8, 8))
plt.matshow(heatmap)
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.

```python
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.

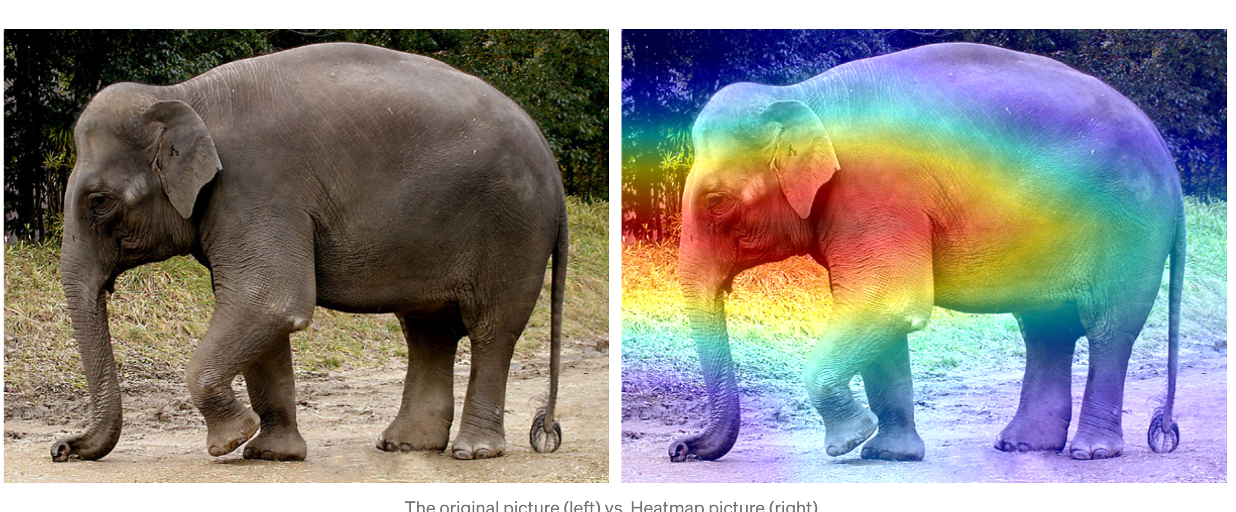
```python
INTENSITY = 0.5

heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

heatmap = cv2.applyColorMap(np.uint8(255*heatmap), cv2.COLORMAP_JET)

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.

```python
cv2_imshow(cv2.imread(ORIGINAL))
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.

```python
def gradCAM(orig, intensity=0.5, res=250):
  img = image.load_img(orig, target_size=(DIM, DIM))

  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)

  preds = model.predict(x)
  print(decode_predictions(preds)[0][0][1]) # prints the class of image

  with tf.GradientTape() as tape:
    last_conv_layer = model.get_layer('conv2d_93')
    iterate = tf.keras.models.Model([model.inputs], [model.output, last_conv_layer.ou
    model_out, last_conv_layer = iterate(x)
    class_out = model_out[:, np.argmax(model_out[0])]
    grads = tape.gradient(class_out, last_conv_layer)
    pooled_grads = K.mean(grads, axis=(0, 1, 2))

  heatmap = tf.reduce_mean(tf.multiply(pooled_grads, last_conv_layer), axis=-1)
  heatmap = np.maximum(heatmap, 0)
  heatmap /= np.max(heatmap)
  heatmap = heatmap.reshape((8, 8))

  img = cv2.imread(orig)

  heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

  heatmap = cv2.applyColorMap(np.uint8(255*heatmap), cv2.COLORMAP_JET)

  img = heatmap * intensity + img

  cv2_imshow(cv2.resize(cv2.imread(orig), (res, res)))
  cv2_imshow(cv2.resize(img, (res, res)))

gradCAM("Chinook-On-White-03.jpg")
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-tf2-0-tf-keras

WRITTEN BY
Areeb Gani
Follow

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