GradCAM: Continuation (Part 2) - Lecture Notebook

In the previous lecture notebook (GradCAM Part 1) we explored what Grad-Cam is and why it is useful. We also looked at how we can compute the activations of a particular layer using Keras API. In this notebook we will check the other element that Grad-CAM requires, the gradients of the model's output with respect to our desired layer's output. This is the "Grad" portion of Grad-CAM.

Let's dive into it!

In [1]:
```
import keras
from keras import backend as K
from util import *
```

```
Using TensorFlow backend.
```

The `load_C3M3_model()` function has been taken care of and as last time, its internals are out of the scope of this notebook.

In [2]:
```
# Load the model we used last time
model = load_C3M3_model()
```

```
Got loss weights
Loaded DenseNet
Added layers
Compiled Model
Loaded Weights
```

Kindly recall from the previous notebook (GradCAM Part 1) that our model has 428 layers.

We are now interested in getting the gradients when the model outputs a specific class. For this we will use Keras backend's `gradients(..)` function. This function requires two arguments:

- Loss (scalar tensor)
- List of variables

Since we want the gradients with respect to the output, we can use our model's output tensor:

In [3]:
```
# Save model's output in a variable
y = model.output

# Print model's output
y
```

Out[3]:
```
<tf.Tensor 'dense_1/Sigmoid:0' shape=(?, 14) dtype=float32>
```

However this is not a scalar (aka rank-0) tensor because it has axes. To transform this tensor into a scalar we can slice it like this:

In [4]:
```
y = y[0]
y
```

Out[4]:
```
<tf.Tensor 'strided_slice:0' shape=(14,) dtype=float32>
```

It is still *not* a scalar tensor so we will have to slice it again:

In [5]:
```
y = y[0]
y
```

Out[5]:
```
<tf.Tensor 'strided_slice_1:0' shape=() dtype=float32>
```

Now it is a scalar tensor!

The above slicing could be done in a single statement like this:

```
y = y[0,0]
```

But the explicit version of it was shown for visibility purposes.

The first argument required by `gradients(..)` function is the loss, which we will like to get the gradient of, and the second is a list of parameters to compute the gradient with respect to. Since we are interested in getting the gradient of the output of the model with respect to the output of the last convolutional layer we need to specify the layer as we did in the previous notebook:

In [6]:
```
# Save the desired layer in a variable
layer = model.get_layer("conv5_block16_concat")

# Compute gradient of model's output with respect to last conv layer's output
gradients = K.gradients(y, layer.output)

# Print gradients list
gradients
```

Out[6]:
```
[<tf.Tensor 'gradients/AddN:0' shape=(?, ?, ?, 1024) dtype=float32>]
```

Notice that the gradients function returns a list of placeholder tensors. To get the actual placeholder we will get the first element of this list:

In [7]:
```
# Get first (and only) element in the list
gradients = gradients[0]

# Print tensor placeholder
gradients
```

Out[7]:
```
<tf.Tensor 'gradients/AddN:0' shape=(?, ?, ?, 1024) dtype=float32>
```

As with the activations of the last convolutional layer in the previous notebook, we still need a function that uses this placeholder to compute the actual values for an input image. This can be done in the same manner as before. Remember this **function expects its arguments as lists or tuples**:

In [8]:
```
# Instantiate the function to compute the gradients
gradients_function = K.function([model.input], [gradients])

# Print the gradients function
gradients_function
```

Out[8]:
```
<keras.backend.tensorflow_backend.Function at 0x7f84e625bb00>
```

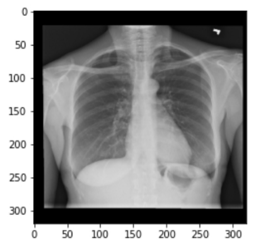
Now that we have the function for computing the gradients, let's test it out on a particular image. Don't worry about the code to load the image, this has been taken care of for you, you should only care that an image ready to be processed will be saved in the x variable:

In [9]:
```python
# Load dataframe that contains information about the dataset of images
df = pd.read_csv("nih_new/train-small.csv")

# Path to the actual image
im_path = 'nih_new/images-small/00000599_000.png'

# Load the image and save it to a variable
x = load_image(im_path, df, preprocess=False)

# Display the image
plt.imshow(x, cmap = 'gray')
plt.show()
```



We should normalize this image before going forward, this has also been taken care of:

In [10]:
```python
# Calculate mean and standard deviation of a batch of images
mean, std = get_mean_std_per_batch(df)

# Normalize image
x = load_image_normalize(im_path, mean, std)
```

Now we have everything we need to compute the actual values of the gradients. In this case we should also **provide the input as a list or tuple**:

In [11]:
```python
# Run the function on the image and save it in a variable
actual_gradients = gradients_function([x])
```

An important intermediary step is to trim the batch dimension which can be done like this:

In [12]:
```python
# Remove batch dimension
actual_gradients = actual_gradients[0][0, :]
```

In [13]:
```python
# Print shape of the gradients array
print(f"Gradients of model's output with respect to output of last convolutional layer have shape: {actual_gradients.shape}")

# Print gradients array
actual_gradients
```

```
  Gradients of model's output with respect to output of last convolutional layer have shape: (10, 10, 1024)
```

Out[13]:
```
array([[[-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        ...,
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05]],

       [[-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        ...,
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05]],

       [[-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        ...,
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05]],

       ...,

       [[-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        ...,
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05]],
       [[-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        ...,
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05]],

       [[-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        ...,
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05],
        [-1.6725008e-09,  3.3696794e-09,  3.9488251e-07, ...,
          1.1026199e-04, -7.3800140e-05,  7.6895798e-05]]], dtype=float32)
```

Looks like everything worked out nicely! You will still have to wait for the assignment to see how these elements are used by Grad-CAM to get visual interpretations. Before you go you should know that there is a shortcut for these calculations by getting both elements from a single Keras function:

In [14]:
```
# Save multi-input Keras function in a variable
activations_and_gradients_function = K.function([model.input], [layer.output, gradients])

# Run the function on our image
act_x, grad_x = activations_and_gradients_function([x])

# Remove batch dimension for both arrays
act_x = act_x[0, :]
grad_x = grad_x[0, :]
```

In [15]:
```
# Print actual activations
print(act_x)

# Print actual gradients
print(grad_x)
```

```
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]]

 [[-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]
  ...
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]
  [-1.6725008e-09  3.3696794e-09  3.9488251e-07 ...  1.1026199e-04
   -7.3800140e-05  7.6895798e-05]]]
```

Congratulations on finishing this lecture notebook! Hopefully you will now have a better understanding of how to leverage Keras's API power for computing gradients. Keep it up!