Austin Vornhagen

10/11/2021

CSCI 8110

Assignment 2

**Runnable codes (Keras and Tensorflow backend functions without other libraries):**

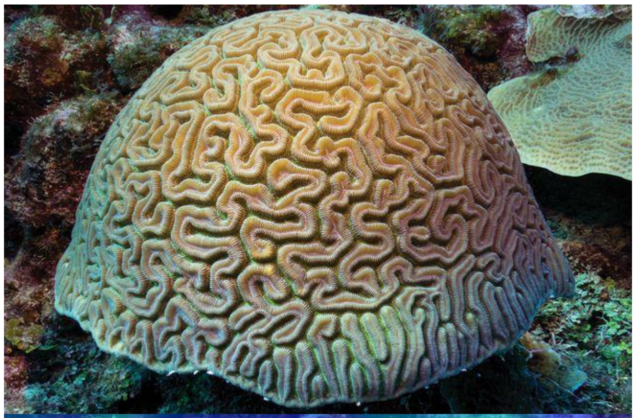
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
from keras.applications.xception import decode_predictions
from IPython.display import Image, display
import matplotlib.cm as cm


def get_img_array(img_path, size):
    # Load image
    img = image.load_img(img_path, target_size=size)

    # Send image to array
    array = image.img_to_array(img)

    # Add dimension to the array
    array = np.expand_dims(array, axis=0)
    return array


def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_inde
x=None):
    # First, we create a model that maps the input image to the activation
s
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, mod
el.output]
    )

    # Then, we compute the gradient of the top predicted class for our inp
ut image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradi
ent
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted
class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap betwee
n 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()


def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=
0.9):
    # Load the original image
    img = image.load_img(img_path)
    img = image.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]
    # Create an image with RGB colorized heatmap
    jet_heatmap = image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = image.img_to_array(jet_heatmap)

    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = image.array_to_img(superimposed_img)

    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
    display(Image(cam_path))


# Set image size, last convolutional layer, and image titles
img_size = (224, 224)
last_conv_layer_name = "block5_conv3"
image_titles = ['Brain Coral', 'Jellyfish', 'Killer Whale', 'Lion', 'Sea A
nemone']

# The local path to the target images
img_path1 = "/content/drive/MyDrive/brain coral.jpg"
img_path2 = "/content/drive/MyDrive/jellyfish.jpg"
img_path3 = "/content/drive/MyDrive/killer whale.jpg"
img_path4 = "/content/drive/MyDrive/lion.jpg"
img_path5 = "/content/drive/MyDrive/sea anemone.jpg"


# Prepare images
img_array1 = preprocess_input(get_img_array(img_path1, size=img_size))
img_array2 = preprocess_input(get_img_array(img_path2, size=img_size))
img_array3 = preprocess_input(get_img_array(img_path3, size=img_size))
img_array4 = preprocess_input(get_img_array(img_path4, size=img_size))
img_array5 = preprocess_input(get_img_array(img_path5, size=img_size))

# Make model
model = VGG16(weights='imagenet', include_top=True)

# Remove last layer's softmax
model.layers[-1].activation = None

# Perform predictions
preds1 = model.predict(img_array1)
preds2 = model.predict(img_array2)
```

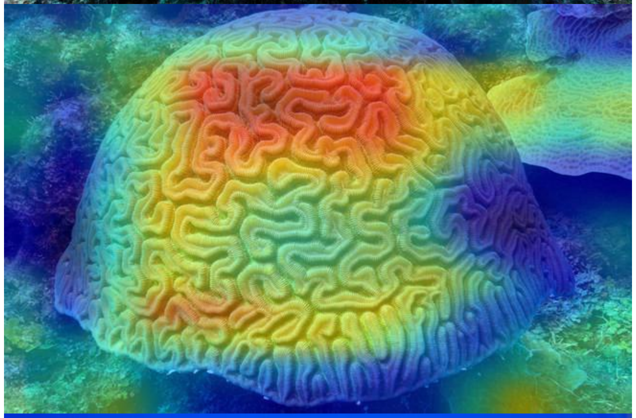
```
preds3 = model.predict(img_array3)
preds4 = model.predict(img_array4)
preds5 = model.predict(img_array5)

# Print what the top predicted class is
print("Predicted:", decode_predictions(preds1, top=1)[0])
print("Predicted:", decode_predictions(preds2, top=1)[0])
print("Predicted:", decode_predictions(preds3, top=1)[0])
print("Predicted:", decode_predictions(preds4, top=1)[0])
print("Predicted:", decode_predictions(preds5, top=1)[0])


# Generate class activation heatmap
heatmap1 = make_gradcam_heatmap(img_array1, model, last_conv_layer_name)

# Display original image and Image with heatmap
display(Image(img_path1))
save_and_display_gradcam(img_path1, heatmap1)

# Generate class activation heatmap
heatmap2 = make_gradcam_heatmap(img_array2, model, last_conv_layer_name)

# Display original image and Image with heatmap
display(Image(img_path2))
save_and_display_gradcam(img_path2, heatmap2)

# Generate class activation heatmap
heatmap3 = make_gradcam_heatmap(img_array3, model, last_conv_layer_name)

# Display original image and Image with heatmap
display(Image(img_path3))
save_and_display_gradcam(img_path3, heatmap3)

# Generate class activation heatmap
heatmap4 = make_gradcam_heatmap(img_array4, model, last_conv_layer_name)

# Display original image and Image with heatmap
display(Image(img_path4))
save_and_display_gradcam(img_path4, heatmap4)

# Generate class activation heatmap
heatmap5 = make_gradcam_heatmap(img_array5, model, last_conv_layer_name)

# Display original image and Image with heatmap
display(Image(img_path5))
```

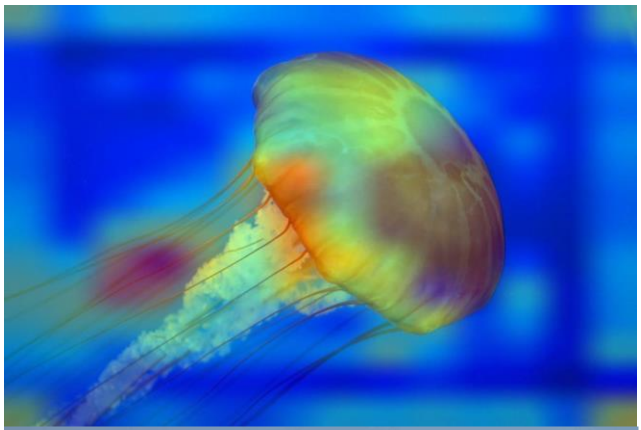
```
save_and_display_gradcam(img_path5, heatmap5)
```

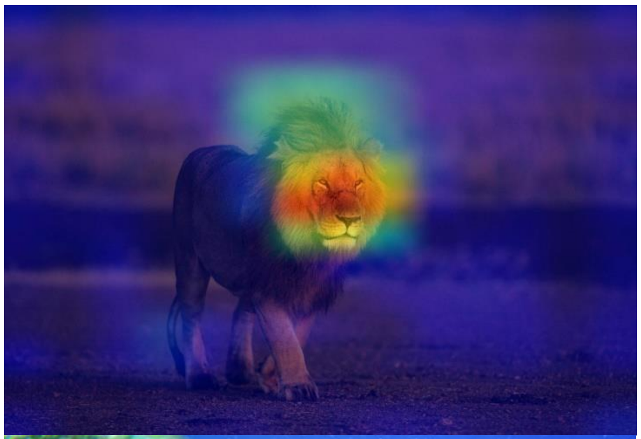
**Plot example input images and their corresponding outputs (Keras and Tensorflow backend functions without other libraries):**

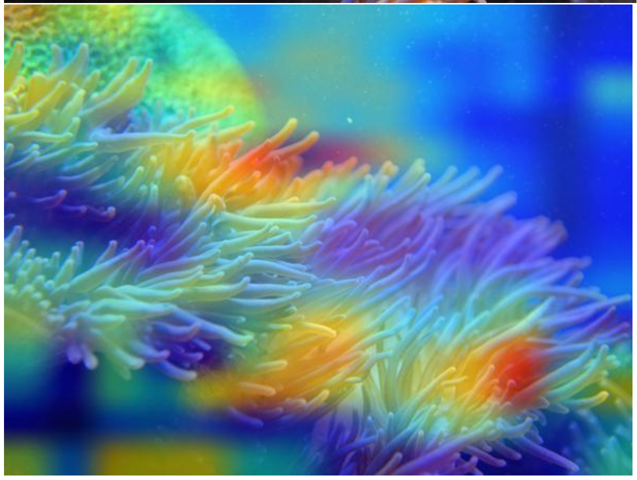
I displayed the input image that was used followed by the corresponding output with the heatmap. To find the input images, I went to the ImageNet 1000 classes text form, picked 5 labels, and searched google images for images that matched those labels. We wanted images that match up with the ImageNet classes so that we know the model should be able to predict what is contained in the image.







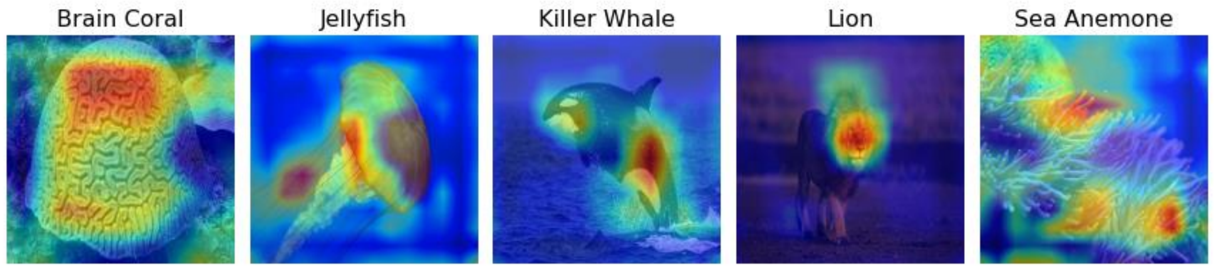




**Runnable codes (tf-Keras-vis):**

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm
from keras.applications.xception import decode_predictions

!pip install tf_keras_vis
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear
from tf_keras_vis.utils.scores import CategoricalScore
from tf_keras_vis.gradcam import Gradcam

# When the softmax activation function is applied to the last layer of mod
el, it may obstruct generating the attention images, so you should replace
 the function to a linear activation function.
replace2linear = ReplaceToLinear()


# 1 is the imagenet index corresponding to brain coral, 107 to jellyfish,
148 to killer whale, 291 to lion, and 108 to sea anemone
score = CategoricalScore([109, 107, 148, 291, 108])

# Create the model
model = VGG16(weights='imagenet', include_top=True)


# Image titles
image_titles = ['Brain Coral', 'Jellyfish', 'Killer Whale', 'Lion', 'Sea A
nemone']

# Load images and Convert them to a Numpy array
img1 = image.load_img('/content/drive/MyDrive/brain coral.jpg', target_siz
e=(224, 224))
img2 = image.load_img('/content/drive/MyDrive/jellyfish.jpg', target_size=
(224, 224))
img3 = image.load_img('/content/drive/MyDrive/killer whale.jpg', target_si
ze=(224, 224))
img4 = image.load_img('/content/drive/MyDrive/lion.jpg', target_size=(224,
 224))
img5 = image.load_img('/content/drive/MyDrive/sea anemone.jpg', target_siz
e=(224, 224))
images = np.asarray([np.array(img1), np.array(img2), np.array(img3), np.ar
ray(img4), np.array(img5)])

# Preparing input data for VGG16
x = preprocess_input(images)

# Rendering
f, ax = plt.subplots(nrows=1, ncols=5, figsize=(12, 4))
for i, title in enumerate(image_titles):
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(images[i])
    ax[i].axis('off')
plt.tight_layout()
plt.show()


# Perform predictions
features = model.predict(x)

# Print what the top predicted class is
print("Predicted:", decode_predictions(features, top=1)[0])
print("Predicted:", decode_predictions(features, top=1)[1])
print("Predicted:", decode_predictions(features, top=1)[2])
print("Predicted:", decode_predictions(features, top=1)[3])
print("Predicted:", decode_predictions(features, top=1)[4])


# Create Gradcam object
gradcam = Gradcam(model,
                  model_modifier=replace2linear,
                  clone=True)

# Generate heatmap with GradCAM
cam = gradcam(score,
              x,
              penultimate_layer=-1)

# Render
f, ax = plt.subplots(nrows=1, ncols=5, figsize=(12, 4))
for i, title in enumerate(image_titles):
    heatmap = np.uint8(cm.jet(cam[i])[..., :3] * 255)
    ax[i].set_title(title, fontsize=16)
    ax[i].imshow(images[i])
    ax[i].imshow(heatmap, cmap='jet', alpha=0.5) # overlay
    ax[i].axis('off')
plt.tight_layout()
plt.show()
```

**Plot example input images and their corresponding outputs (tf-Keras-vis):**



**Discussion of thinking and discovery:**

I am using Google Colab for this assignment. I am still learning the syntax for Keras and Tensorflow. Shortly after starting this assignment, I didn't know how to get images into the code in python. The way I solved this problem was by placing all the images I was going to use on my personal Google Drive, and using the Google Colab functionality to connect to Google Drive. From there I could find the files in the "Files" pane on the side of my screen and then get the path that my code would use to find the image.

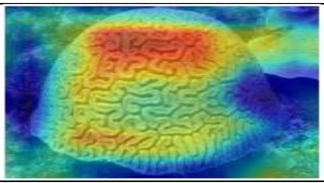
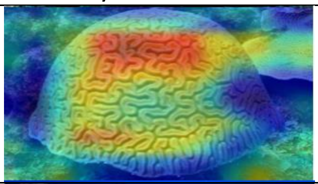
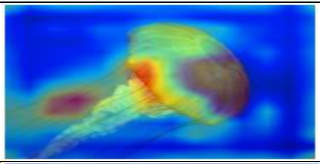
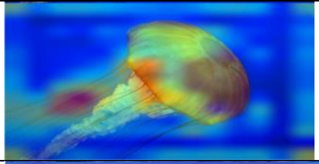
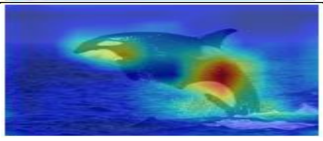
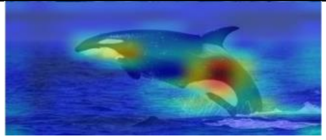
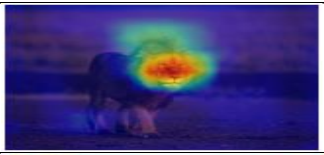
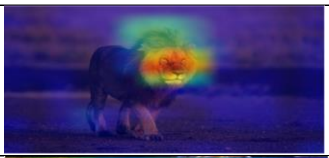
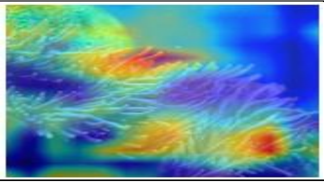
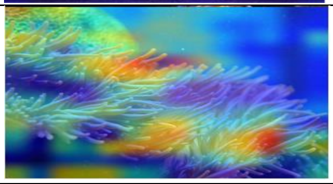
I learned that when running model.summary(), I had gaps in my understanding of the output. In the implementation with Keras and Tensorflow only, I needed to get the name of the last convolutional layer. I didn't have a full understanding about what a MaxPooling2D layer or Flatten layer were, and if they were convolutional or not. A MaxPooling2D layer conducts a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. A Flatten layer convers the data in a 1-dimensional array. This meant the last convolutional layer was block5_conv3.

I tried to download the imagenet dataset but it was 1.1 terabytes of information so I just got input images from the internet that I thought matched at least 1 class in the 1000 ImageNet classes of human readable labels.

For the implementation using Keras and Tensorflow only, I referenced keras.io for Grad-CAM class activation visualization syntax, the VGG16 syntax, and the syntax for the display function. For the tf-Keras-vis implementation, I used tf-Keras-vis github repository https://github.com/keisen/tf-keras-vis as a reference for the Grad-CAM syntax, and the syntax for the display.

The heat map, which is the program output, is supposed to show which parts of an image have the most impact on the classification score. Red pixels are going to have the most impact on the classification score and blue pixels are going to have the least impact on the classification score. For example, if we were to remove the red pixels, we would lose a lot of conviction in our prediction of which class the image belongs to. If we were to remove the blue pixels, the difference in the conviction of our prediction about which class the image belongs to would be small.

I compared how the output images look when using tf-Keras-vis library vs using Keras and Tensorflow. There are differences between the output images but they are pretty insignificant. Both implementations have about the same locations for high impact red pixels and low impact blue pixels.

| Image | tf-Keras-vis | K&T only | Observation |
|---|---|---|---|
| Brain Coral |  |  | Areas of interest are the same. Small differences on the edges of those areas. |
| Jellyfish |  |  | The red pixels look sharper on the tf-Keras-vis. Areas of interest are the same. |
| Killer Whale |  |  | Areas of interest are the same. Small differences with the shading of the heatmap. |
| Lion |  |  | K&T red pixels appears wider than tf-Keras-vis red pixels. Area of interest is the same. |
| Sea Anemone |  |  | Areas of interest are the same. |

When you increase the alpha number of the heat map the opacity of the heatmap seems to increase. I used 0.9 for the Keras and Tensorflow only implementation, and I used 0.5 for the tf-Keras-vis implementation.

Predictions and scores for the input images:

| Image | tf-Keras-vis prediction | tf-Keras-vis score | K&T only prediction | K&T only score |
|---|---|---|---|---|
| Brain Coral | `brain_coral` | `0.9990822` | `brain_coral` | `38.013634` |
| Jellyfish | `jellyfish` | `0.99999` | `jellyfish` | `24.701704` |
| Killer Whale | `killer_whale` | `0.9980908` | `killer_whale` | `25.637856` |
| Lion | `lion` | `0.9796825` | `lion` | `16.420755` |
| Sea Anemone | `sea_anemone` | `0.4919445` | `sea_anemone` | `17.3063` |

From this we can see that for both implementations, the prediction was correct 10/10 times. We can see in the tf-Keras-vis implementation the sea anemone received the lowest score and the jellyfish received the highest score. In the K&T only implementation the lion received the lowest score and the brain coral received the highest score.

The last thing I learned was Grad-CAM was made to be able to assign a category or class to an image (called class-discriminative) and capture fine-grained details of an image (meaning it is high-resolution). The reason we need explainable deep learning is because interpretability matters. This lets humans build trust in intelligent systems and helps the people using these implementations debug more complex models that are not performing as expected.