Week 4 Assignment: Saliency Maps

Welcome to the final programming exercise of this course! For this week, your task is to adapt the Cats vs Dogs Class Activation Map ungraded lab (the second ungraded lab of this week) and make it generate saliency maps instead.

As discussed in the lectures, a saliency map shows the pixels which greatly impacts the classification of an image.

- This is done by getting the gradient of the loss with respect to changes in the pixel values, then plotting the results.
- From there, you can see if your model is looking at the correct features when classifying an image.
    - For example, if you're building a dog breed classifier, you should be wary if your saliency map shows strong pixels outside the dog itself (e.g. sky, grass, dog house, etc...).

In this assignment you will be given prompts but less starter code to fill in in.

- It's good practice for you to try and write as much of this code as you can from memory and from searching the web.
- **Whenever you feel stuck**, please refer back to the labs of this week to see how to write the code. In particular, look at:
    - **Ungraded Lab 2: Cats vs Dogs CAM**
    - **Ungraded Lab 3: Saliency**

Download test files and weights

Let's begin by first downloading files we will be using for this lab.

```
# Download the same test files from the Cats vs Dogs ungraded lab
!wget -O cat1.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/cat1
!wget -O cat2.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/cat2
!wget -O catanddog.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages
!wget -O dog1.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/dog1
!wget -O dog2.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/dog2

# Download prepared weights
!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1kipXTxesGJKGY1B8uSPRvx
!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1oiV6tjy5k7h9OHGTQaf0Oh
```

```
2021-05-23 18:23:53 (108 MB/s) - ‘catanddog.jpg’ saved [561943/561943]

--2021-05-23 18:23:53--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/ML
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.20.128, 74.125.142.128,
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.20.128|:443... connect
HTTP request sent, awaiting response... 200 OK
Length: 338769 (331K) [image/jpeg]
Saving to: ‘dog1.jpg’

dog1.jpg            100%[===================>] 330.83K  --.-KB/s    in 0.003s

2021-05-23 18:23:53 (114 MB/s) - ‘dog1.jpg’ saved [338769/338769]
```

```
--2021-05-23 18:23:53--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/ML
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.135.128, 74.125.20.128,
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.135.128|:443... connec
HTTP request sent, awaiting response... 200 OK
Length: 494803 (483K) [image/jpeg]
Saving to: ‘dog2.jpg’

dog2.jpg            100%[===================>] 483.21K  --.-KB/s    in 0.005s

2021-05-23 18:23:53 (98.8 MB/s) - ‘dog2.jpg’ saved [494803/494803]

--2021-05-23 18:23:53--  https://docs.google.com/uc?export=download&id=1kipXTxesGJKGY1B8uSP
Resolving docs.google.com (docs.google.com)... 74.125.195.138, 74.125.195.101, 74.125.195.1
Connecting to docs.google.com (docs.google.com)|74.125.195.138|:443... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://doc-0o-6k-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksu
Warning: wildcards not supported in HTTP.
--2021-05-23 18:23:54--  https://doc-0o-6k-docs.googleusercontent.com/docs/securesc/ha0ro93
Resolving doc-0o-6k-docs.googleusercontent.com (doc-0o-6k-docs.googleusercontent.com)... 74
Connecting to doc-0o-6k-docs.googleusercontent.com (doc-0o-6k-docs.googleusercontent.com)|7
HTTP request sent, awaiting response... 200 OK
Length: 414488 (405K) [application/octet-stream]
Saving to: ‘0_epochs.h5’

0_epochs.h5         100%[===================>] 404.77K  --.-KB/s    in 0.003s

2021-05-23 18:23:54 (141 MB/s) - ‘0_epochs.h5’ saved [414488/414488]

--2021-05-23 18:23:54--  https://docs.google.com/uc?export=download&id=1oiV6tjy5k7h9OHGTQaf
Resolving docs.google.com (docs.google.com)... 74.125.195.138, 74.125.195.101, 74.125.195.1
Connecting to docs.google.com (docs.google.com)|74.125.195.138|:443... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://doc-08-6k-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksu
Warning: wildcards not supported in HTTP.
--2021-05-23 18:23:55--  https://doc-08-6k-docs.googleusercontent.com/docs/securesc/ha0ro93
Resolving doc-08-6k-docs.googleusercontent.com (doc-08-6k-docs.googleusercontent.com)... 74
Connecting to doc-08-6k-docs.googleusercontent.com (doc-08-6k-docs.googleusercontent.com)|7
HTTP request sent, awaiting response... 200 OK
Length: 414488 (405K) [application/octet-stream]
Saving to: ‘15_epochs.h5’

15_epochs.h5        100%[===================>] 404.77K  --.-KB/s    in 0.003s

2021-05-23 18:23:56 (129 MB/s) - ‘15_epochs.h5’ saved [414488/414488]
```

Import the required packages

Please import:

- Tensorflow
- Tensorflow Datasets
- Numpy
- Matplotlib's PyPlot
- Keras plot_model utility
- Keras Models API classes you will be using
- Keras layers you will be using
- OpenCV (cv2)

```
# YOUR CODE HERE
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

import keras
from keras.utils import plot_model
from keras.models import Sequential,Model
from keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,GlobalAveragePooling2D

import cv2
```

Download and prepare the dataset.

Load Cats vs Dogs

- Required: Use Tensorflow Datasets to fetch the `cats_vs_dogs` dataset.
  - Use the first 80% of the *train* split of the said dataset to create your training set.
  - Set the `as_supervised` flag to create `(image, label)` pairs.
- Optional: You can create validation and test sets from the remaining 20% of the *train* split of `cats_vs_dogs` (i.e. you already used 80% for the train set). This is if you intend to train the model beyond what is required for submission.

```
# Load the data and create the train set (optional: val and test sets)
train_data = tfds.load('cats_vs_dogs', split='train[:80%]', as_supervised=True)
validation_data = tfds.load('cats_vs_dogs', split='train[80%:90%]', as_supervised=True)
test_data = tfds.load('cats_vs_dogs', split='train[-10%:]', as_supervised=True)
# YOUR CODE HERE
```

```
Downloading and preparing dataset cats_vs_dogs/4.0.0 (download: 786.68 MiB, generated: Unknown
Dl Completed...: 100%  1/1 [00:11<00:00, 11.77s/ url]
Dl Size...: 100%  786/786 [00:11<00:00, 66.97 MiB/s]
```

```
WARNING:absl:1738 images were corrupted and were skipped
Shuffling and writing examples to /root/tensorflow_datasets/cats_vs_dogs/4.0.0.incompleteJT7TT
100%  23149/23262 [00:03<00:00, 8250.52 examples/s]
Dataset cats_vs_dogs downloaded and prepared to /root/tensorflow_datasets/cats_vs_dogs/4.0.0.
```

Create preprocessing function

Define a function that takes in an image and label. This will:

- cast the image to float32
- normalize the pixel values to [0, 1]

- resize the image to 300 x 300

```
def augmentimages(image, label):
  # YOUR CODE HERE
  # cast to float
  image = tf.cast(image, tf.float32)
  # normalize the pixel values
  image = (image/255)
  # resize to 300 x 300
  image = tf.image.resize(image,(300,300))
  return image, label
```

Preprocess the training set

Use the `map()` and pass in the method that you just defined to preprocess the training set.

```
augmented_training_data = train_data.map(augmentimages) # YOUR CODE HERE
```

Create batches of the training set.

This is already provided for you. Normally, you will want to shuffle the training set. But for predictability in the grading, we will simply create the batches.

```
 # Shuffle the data if you're working on your own personal project
 train_batches = augmented_training_data.shuffle(1024).batch(32)
```

```
train_batches = augmented_training_data.batch(32)
```

Build the Cats vs Dogs classifier

You'll define a model that is nearly the same as the one in the Cats vs. Dogs CAM lab.

- Please preserve the architecture of the model in the Cats vs Dogs CAM lab (this week's second lab) except for the final `Dense` layer.
- You should modify the Cats vs Dogs model at the last dense layer to output 2 neurons instead of 1.
    - This is because you will adapt the `do_salience()` function from the lab and that works with one-hot encoded labels.
    - You can do this by changing the `units` argument of the output Dense layer from 1 to 2, with one for each of the classes (i.e. cats and dogs).
    - You should choose an activation that outputs a probability for each of the 2 classes (i.e. categories), where the sum of the probabilities adds up to 1.

```
# YOUR CODE HERE
model = Sequential()
model.add(Conv2D(16,input_shape=(300,300,3),kernel_size=(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same'))
model.add(GlobalAveragePooling2D())
model.add(Dense(2,activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 300, 300, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 150, 150, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 150, 150, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 37, 37, 128)       73856
_________________________________________________________________
global_average_pooling2d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 2)                 258
=================================================================
Total params: 97,698
Trainable params: 97,698
Non-trainable params: 0
_________________________________________________________________
```

Expected Output:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 300, 300, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 150, 150, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 150, 150, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 37, 37, 128)       73856
_________________________________________________________________
```

```
global_average_pooling2d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 2)                 258
=================================================================
Total params: 97,698
Trainable params: 97,698
Non-trainable params: 0
_________________________________________________________________
```

Create a function to generate the saliency map

Complete the `do_salience()` function below to save the **normalized_tensor** image.

- The major steps are listed as comments below.
  - Each section may involve multiple lines of code.
- Try your best to write the code from memory or by performing web searches.
  - Whenever you get stuck, you can review the "saliency" lab (the third lab of this week) to help remind you of what code to write

```
#tf.one_hot([0] * 300, 2)


def do_salience(image, model, label, prefix):
  '''
  Generates the saliency map of a given image.

  Args:
    image (file) -- picture that the model will classify
    model (keras Model) -- your cats and dogs classifier
    label (int) -- ground truth label of the image
    prefix (string) -- prefix to add to the filename of the saliency map
  '''

  # Read the image and convert channel order from BGR to RGB
  # YOUR CODE HERE
  img = cv2.cvtColor(cv2.imread(image), cv2.COLOR_BGR2RGB)

  # Resize the image to 300 x 300 and normalize pixel values to the range [0, 1]
  # YOUR CODE HERE
  img = cv2.resize(img, (300, 300)) / 255.0


  # Add an additional dimension (for the batch), and save this in a new variable
  # YOUR CODE HERE
  img = np.expand_dims(img, axis=0)

  # Declare the number of classes
  # YOUR CODE HERE
  num_classes = 2


  # Define the expected output array by one-hot encoding the label
  # The length of the array is equal to the number of classes
```

```
# YOUR CODE HERE
expected_output = tf.one_hot([0] * img.shape[0], num_classes)

# Witin the GradientTape block:
# Cast the image as a tf.float32
# Use the tape to watch the float32 image
# Get the model's prediction by passing in the float32 image
# Compute an appropriate loss
# between the expected output and model predictions.
# you may want to print the predictions to see if the probabilities adds up to 1
# YOUR CODE HERE
with tf.GradientTape() as tape:
  # cast image to float
  inputs = tf.cast(img, tf.float32)

  # watch the input pixels
  tape.watch(inputs)

  # generate the predictions
  predictions = model(inputs)
  print(predictions)

  # get the loss
  loss = tf.keras.losses.categorical_crossentropy(
      expected_output, predictions
  )


# get the gradients of the loss with respect to the model's input image
# YOUR CODE HERE
gradients = tape.gradient(loss, inputs)

# generate the grayscale tensor
# YOUR CODE HERE
grayscale_tensor = tf.reduce_sum(tf.abs(gradients), axis=-1)

# normalize the pixel values to be in the range [0, 255].
# the max value in the grayscale tensor will be pushed to 255.
# the min value will be pushed to 0.
# Use the formula: 255 * (x - min) / (max - min)
# Use tf.reduce_max, tf.reduce_min
# Cast the tensor as a tf.uint8
# YOUR CODE HERE
normalized_tensor = tf.cast(
  255
  * (grayscale_tensor - tf.reduce_min(grayscale_tensor))
  / (tf.reduce_max(grayscale_tensor) - tf.reduce_min(grayscale_tensor)),
  tf.uint8,
)


# Remove dimensions that are size 1
# YOUR CODE HERE
normalized_tensor = tf.squeeze(normalized_tensor)

# plot the normalized tensor
# Set the figure size to 8 by 8
# do not display the axis
# use the 'gray' colormap
```

```
  # This code is provided for you.
  plt.figure(figsize=(8, 8))
  plt.axis('off')
  plt.imshow(normalized_tensor, cmap='gray')
  plt.show()

  # optional: superimpose the saliency map with the original image, then display it.
  # we encourage you to do this to visualize your results better
  # YOUR CODE HERE
  gradient_color = cv2.applyColorMap(normalized_tensor.numpy(), cv2.COLORMAP_HOT)
  gradient_color = gradient_color / 255.0
  #print(img.shape, gradient_color.shape)
  super_imposed = cv2.addWeighted(img[0], 0.5, gradient_color, 0.5, 0.0)
  plt.figure(figsize=(8, 8))
  plt.imshow(super_imposed)
  plt.axis('off')
  plt.show()

  # save the normalized tensor image to a file. this is already provided for you.
  salient_image_name = prefix + image
  normalized_tensor = tf.expand_dims(normalized_tensor, -1)
  normalized_tensor = tf.io.encode_jpeg(normalized_tensor, quality=100, format='grayscale')
  writer = tf.io.write_file(salient_image_name, normalized_tensor)
```

Generate saliency maps with untrained model

As a sanity check, you will load initialized (i.e. untrained) weights and use the function you just implemented.

- This will check if you built the model correctly and are able to create a saliency map.

If an error pops up when loading the weights or the function does not run, please check your implementation for bugs.

- You can check the ungraded labs of this week.

Please apply your `do_salience()` function on the following image files:

- `cat1.jpg`
- `cat2.jpg`
- `catanddog.jpg`
- `dog1.jpg`
- `dog2.jpg`

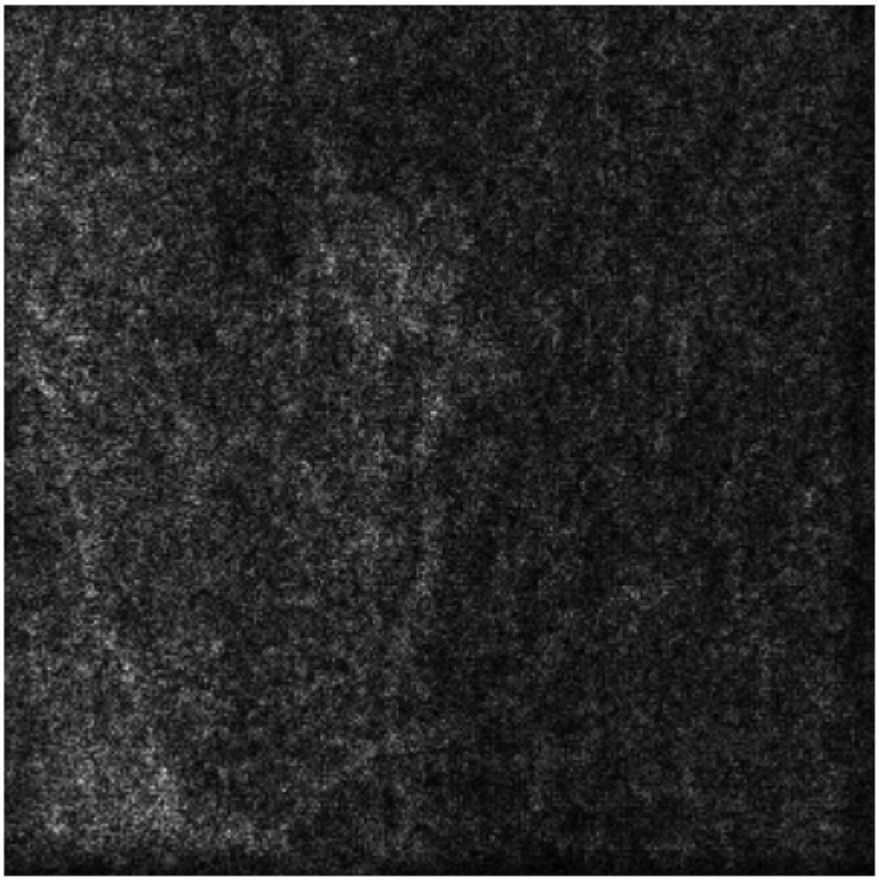
Cats will have the label `0` while dogs will have the label `1`.

- For the catanddog, please use `0`.
- For the prefix of the salience images that will be generated, please use the prefix `epoch0_salient`.

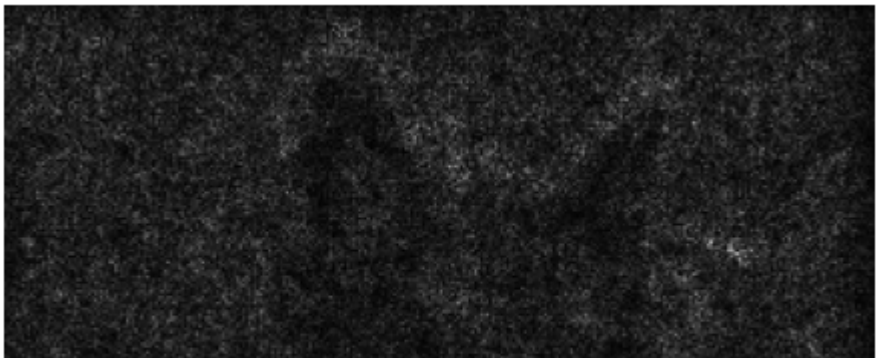
```
# load initial weights
model.load_weights('0_epochs.h5')

# generate the saliency maps for the 5 test images
# YOUR CODE HERE
images = ['cat1.jpg', 'cat2.jpg', 'catanddog.jpg', 'dog1.jpg', 'dog2.jpg']
labels = [0, 0, 0, 1, 1]
for image, label in zip(images, labels):
```

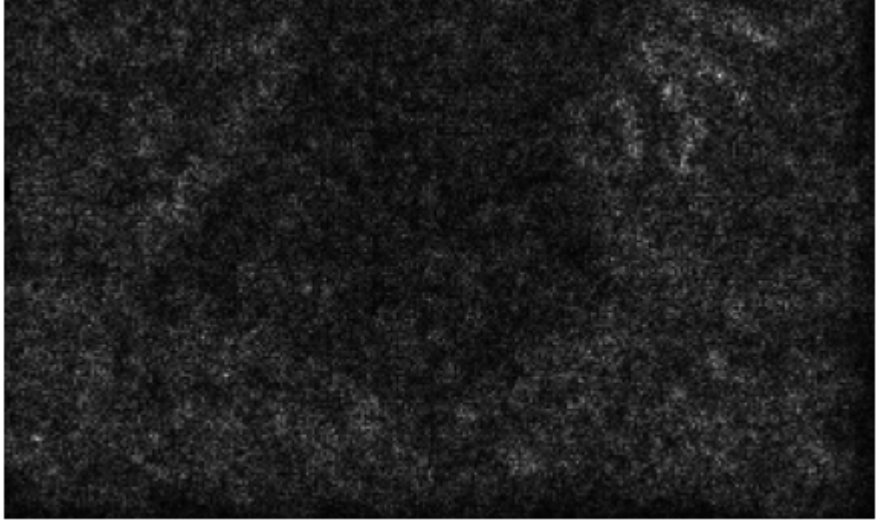
```
do_salience(image, model, label, prefix='epoch0_salient')
```

```
tf.Tensor([[0.49321154 0.5067885 ]], shape=(1, 2), dtype=float32)
```

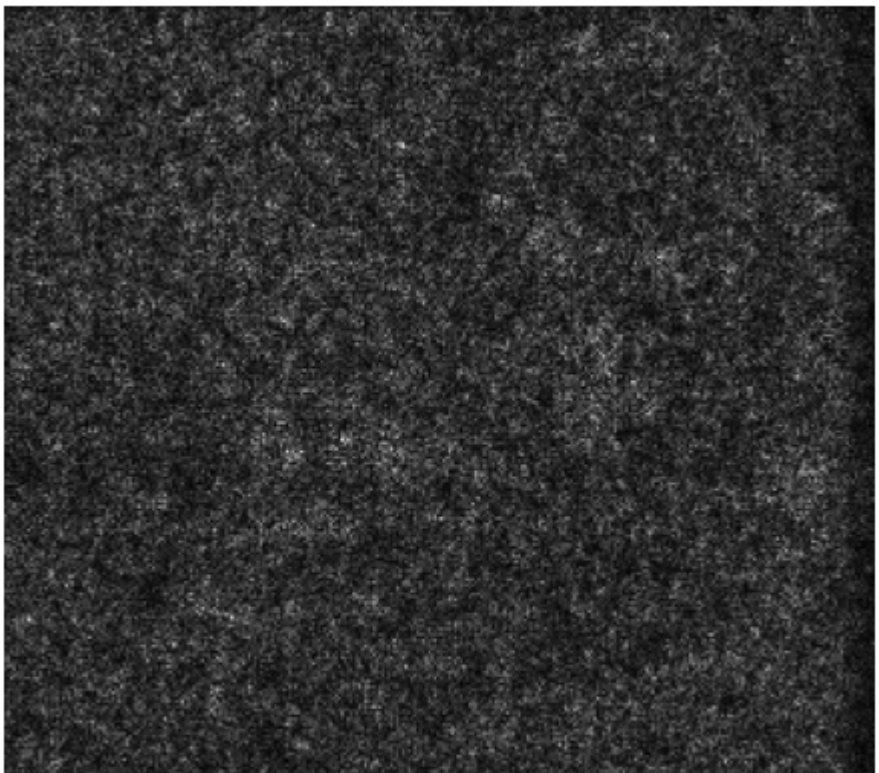


```
tf.Tensor([[0.49322364 0.5067764 ]], shape=(1, 2), dtype=float32)
```

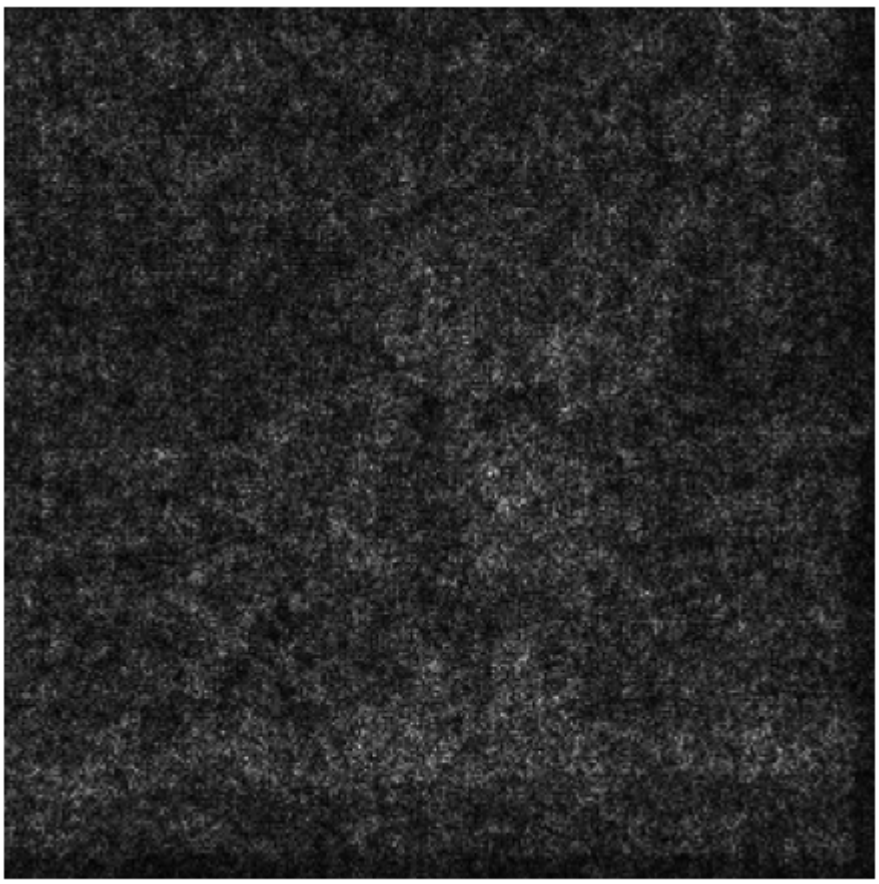


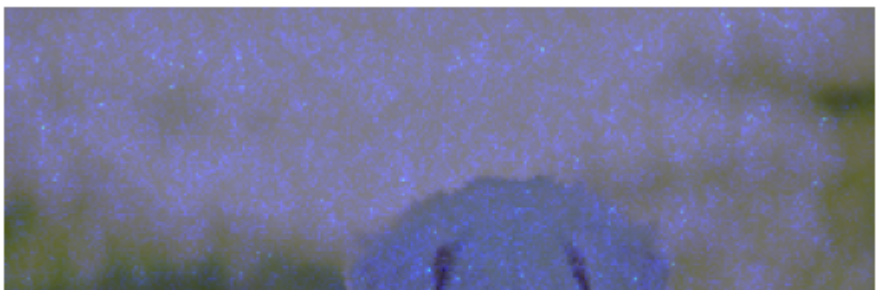


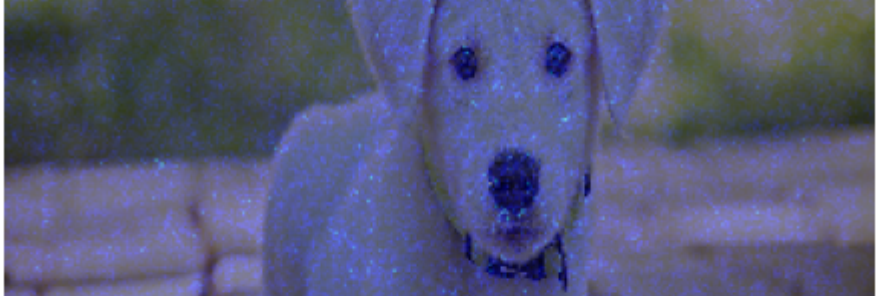
```
tf.Tensor([[0.4922674 0.5077326]], shape=(1, 2), dtype=float32)
```



```
tf.Tensor([[0.49055642 0.5094436 ]], shape=(1, 2), dtype=float32)
```



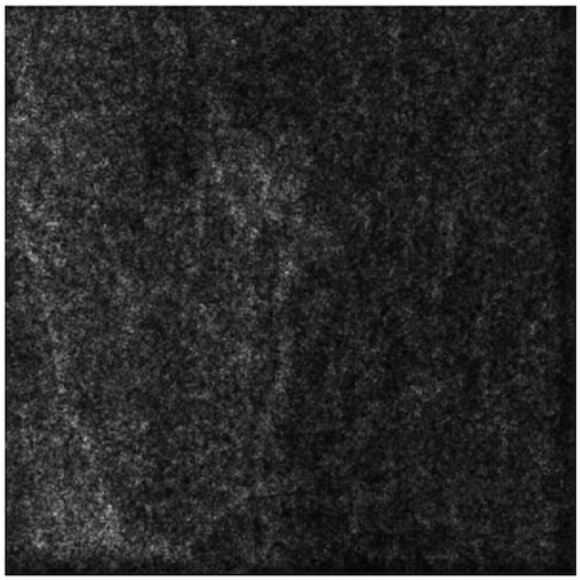




With untrained weights, you will see something like this in the output.

- You will see strong pixels outside the cat that the model uses that when classifying the image.
- After training that these will slowly start to localize to features inside the pet.

```
tf.Tensor([[0.4932115 0.5067885]], shape=(1, 2), dtype=float32)
```



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

Configure the model for training

Use `model.compile()` to define the loss, metrics and optimizer.

- Choose a loss function for the model to use when training.
    - For `model.compile()` the ground truth labels from the training set are passed to the model as **integers** (i.e. 0 or 1) as opposed to one-hot encoded vectors.
    - The model predictions are class probabilities.
    - You can browse the tf.keras.losses and determine which one is best used for this case.

    - Remember that you can pass the function as a string (e.g. `loss = 'loss_function_a'`).
- For metrics, you can measure `accuracy`.
- For the optimizer, please use RMSProp.
    - Please use the default learning rate of `0.001`.

```
# YOUR CODE HERE
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001), loss='sparse_categorical_c
```

Train your model

Please pass in the training batches and train your model for just **3** epochs.

- **Note:** Please do not exceed 3 epochs because the grader will expect 3 epochs when grading your output.
    - After submitting your zipped folder for grading, feel free to continue training to improve your model.

We have loaded pre-trained weights for 15 epochs so you can get a better output when you visualize the saliency maps.

```
# load pre-trained weights
model.load_weights('15_epochs.h5')

# train the model for just 3 epochs
# YOUR CODE HERE
model.fit(train_batches, epochs=3) #, validation_data=validation_data.batch(32))
```

```
Epoch 1/3
582/582 [==============================] - 43s 73ms/step - loss: 0.4544 - accuracy: 0.7944
Epoch 2/3
582/582 [==============================] - 43s 73ms/step - loss: 0.4355 - accuracy: 0.8065
Epoch 3/3
582/582 [==============================] - 43s 73ms/step - loss: 0.4265 - accuracy: 0.8136
<tensorflow.python.keras.callbacks.History at 0x7f4d701aca90>
```

Generate saliency maps at 18 epochs

You will now use your `do_salience()` function again on the same test images. Please use the same parameters as before but this time, use the prefix `salient`.

```
# YOUR CODE HERE
images = ['cat1.jpg', 'cat2.jpg', 'catanddog.jpg', 'dog1.jpg', 'dog2.jpg']
labels = [0, 0, 0, 1, 1]
for image, label in zip(images, labels):
  do_salience(image, model, label, prefix='salient')
```