# Ungraded Lab: Cats vs. Dogs Class Activation Maps

You will again practice with CAMs in this lab and this time there will only be two classes: Cats and Dogs. You will be revisiting this exercise in this week's programming assignment so it's best if you become familiar with the steps discussed here, particularly in preprocessing the image and building the model.

## Imports

```
[3] import tensorflow_datasets as tfds
    import tensorflow as tf

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

    import numpy as np
    import matplotlib.pyplot as plt
    import scipy as sp
    import cv2
```

## Download and Prepare the Dataset

We will use the Cats vs Dogs dataset and we can load it via Tensorflow Datasets. The images are labeled 0 for cats and 1 for dogs.

```
[4] 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)
```

```
Downloading and preparing dataset cats_vs_dogs/4.0.0 (download: 786.68 MiB, generated: Unknown size, total: 786.68 MiB) to /root/tensorflow_datasets/cats_vs_dogs/4.0.0...
Dl Completed...: 100% 1/1 [00:11<00:00, 11.13s/ url]
Dl Size...: 100% 786/786 [00:11<00:00, 70.83 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.incompleteFF53R7/cats_vs_dogs-train.tfrecord
98% 22822/23262 [00:08<00:00, 8670.26 examples/s]
Dataset cats_vs_dogs downloaded and prepared to /root/tensorflow_datasets/cats_vs_dogs/4.0.0. Subsequent calls will reuse this data.
```

The cell below will preprocess the images and create batches before feeding it to our model.

```
[5] def augment_images(image, label):

      # 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

    # use the utility function above to preprocess the images
    augmented_training_data = train_data.map(augment_images)

    # shuffle and create batches before training
    train_batches = augmented_training_data.shuffle(1024).batch(32)
```

## Build the classifier

This will look familiar to you because it is almost identical to the previous model we built. The key difference is the output is just one unit that is sigmoid activated. This is because we're only dealing with two classes.

```
[6] 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(1,activation='sigmoid'))

    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, 1)                 129
=================================================================
Total params: 97,569
Trainable params: 97,569
Non-trainable params: 0
_________________________________________________________________
```

The loss can be adjusted from last time to deal with just two classes. For that, we pick `binary_crossentropy`.

```
[7]  # Training will take around 30 minutes to complete using a GPU. Time for a break!

     model.compile(loss='binary_crossentropy',metrics=['accuracy'],optimizer=tf.keras.optimizers.RMSprop(lr=0.001))
     model.fit(train_batches,epochs=25)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
Epoch 1/25
582/582 [==============================] - 95s 104ms/step - loss: 0.6686 - accuracy: 0.5885
Epoch 2/25
582/582 [==============================] - 61s 102ms/step - loss: 0.6316 - accuracy: 0.6405
Epoch 3/25
582/582 [==============================] - 61s 102ms/step - loss: 0.6003 - accuracy: 0.6753
Epoch 4/25
582/582 [==============================] - 62s 104ms/step - loss: 0.5844 - accuracy: 0.6929
Epoch 5/25
582/582 [==============================] - 61s 102ms/step - loss: 0.5705 - accuracy: 0.7080
Epoch 6/25
582/582 [==============================] - 61s 103ms/step - loss: 0.5550 - accuracy: 0.7220
Epoch 7/25
582/582 [==============================] - 61s 103ms/step - loss: 0.5507 - accuracy: 0.7299
Epoch 8/25
582/582 [==============================] - 61s 103ms/step - loss: 0.5284 - accuracy: 0.7418
Epoch 9/25
582/582 [==============================] - 61s 102ms/step - loss: 0.5179 - accuracy: 0.7517
Epoch 10/25
582/582 [==============================] - 61s 101ms/step - loss: 0.5060 - accuracy: 0.7577
Epoch 11/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4960 - accuracy: 0.7644
Epoch 12/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4844 - accuracy: 0.7731
Epoch 13/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4771 - accuracy: 0.7787
Epoch 14/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4665 - accuracy: 0.7856
Epoch 15/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4580 - accuracy: 0.7920
Epoch 16/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4506 - accuracy: 0.7932
Epoch 17/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4420 - accuracy: 0.8021
Epoch 18/25
582/582 [==============================] - 62s 103ms/step - loss: 0.4339 - accuracy: 0.8092
Epoch 19/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4268 - accuracy: 0.8112
Epoch 20/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4137 - accuracy: 0.8177
Epoch 21/25
582/582 [==============================] - 61s 102ms/step - loss: 0.4075 - accuracy: 0.8215
Epoch 22/25
582/582 [==============================] - 61s 102ms/step - loss: 0.3967 - accuracy: 0.8290
Epoch 23/25
582/582 [==============================] - 61s 103ms/step - loss: 0.3911 - accuracy: 0.8294
Epoch 24/25
582/582 [==============================] - 61s 102ms/step - loss: 0.3813 - accuracy: 0.8337
Epoch 25/25
582/582 [==============================] - 61s 101ms/step - loss: 0.3738 - accuracy: 0.8389
<tensorflow.python.keras.callbacks.History at 0x7eff54276d10>
```

## Building the CAM model

You will follow the same steps as before in generating the class activation maps.

```
[8]  gap_weights = model.layers[-1].get_weights()[0]
     gap_weights.shape

     cam_model  = Model(inputs=model.input,outputs=(model.layers[-3].output,model.layers[-1].output))
     cam_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_input (InputLayer)    [(None, 300, 300, 3)]     0
_________________________________________________________________
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, 1)                 129
=================================================================
```

```
Total params: 97,569
Trainable params: 97,569
Non-trainable params: 0
_________________________________________________________________
```

```
[9] def show_cam(image_value, features, results):
      '''
      Displays the class activation map of an image

      Args:
        image_value (tensor) -- preprocessed input image with size 300 x 300
        features (array) -- features of the image, shape (1, 37, 37, 128)
        results (array) -- output of the sigmoid layer
      '''

      # there is only one image in the batch so we index at `0`
      features_for_img = features[0]
      prediction = results[0]

      # there is only one unit in the output so we get the weights connected to it
      class_activation_weights = gap_weights[:,0]

      # upsample to the image size
      class_activation_features = sp.ndimage.zoom(features_for_img, (300/37, 300/37, 1), order=2)

      # compute the intensity of each feature in the CAM
      cam_output  = np.dot(class_activation_features,class_activation_weights)

      # visualize the results
      print(f'sigmoid output: {results}')
      print(f"prediction: {'dog' if round(results[0][0]) else 'cat'}")
      plt.figure(figsize=(8,8))
      plt.imshow(cam_output, cmap='jet', alpha=0.5)
      plt.imshow(tf.squeeze(image_value), alpha=0.5)
      plt.show()
```

## Testing the Model

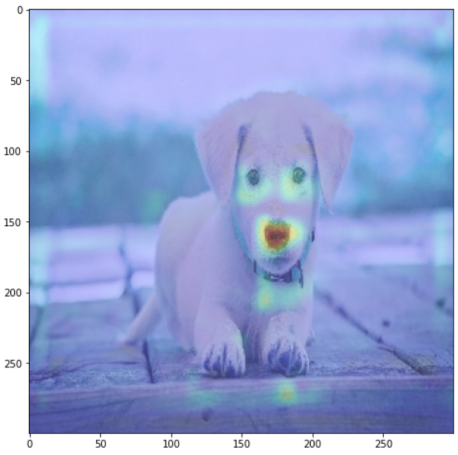
Let's download a few images and see how the class activation maps look like.

```
[10] !wget -O cat1.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/cat1.jpg
     !wget -O cat2.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/cat2.jpg
     !wget -O catanddog.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/catanddog.jpg
     !wget -O dog1.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/dog1.jpg
     !wget -O dog2.jpg https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/dog2.jpg

     --2021-08-10 03:41:34--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/cat1.jpg
     Resolving storage.googleapis.com (storage.googleapis.com)... 142.251.2.128, 74.125.137.128, 142.250.101.128, ...
     Connecting to storage.googleapis.com (storage.googleapis.com)|142.251.2.128|:443... connected.
     HTTP request sent, awaiting response... 200 OK
     Length: 414826 (405K) [image/jpeg]
     Saving to: ‘cat1.jpg’

     cat1.jpg            100%[===================>] 405.10K  --.-KB/s    in 0.006s

     2021-08-10 03:41:34 (67.0 MB/s) - ‘cat1.jpg’ saved [414826/414826]

     --2021-08-10 03:41:34--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/cat2.jpg
     Resolving storage.googleapis.com (storage.googleapis.com)... 142.251.2.128, 74.125.137.128, 142.250.101.128, ...
     Connecting to storage.googleapis.com (storage.googleapis.com)|142.251.2.128|:443... connected.
     HTTP request sent, awaiting response... 200 OK
     Length: 599639 (586K) [image/jpeg]
     Saving to: ‘cat2.jpg’

     cat2.jpg            100%[===================>] 585.58K  --.-KB/s    in 0.005s

     2021-08-10 03:41:34 (123 MB/s) - ‘cat2.jpg’ saved [599639/599639]

     --2021-08-10 03:41:34--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/catanddog.jpg
     Resolving storage.googleapis.com (storage.googleapis.com)... 142.251.2.128, 74.125.137.128, 142.250.101.128, ...
     Connecting to storage.googleapis.com (storage.googleapis.com)|142.251.2.128|:443... connected.
     HTTP request sent, awaiting response... 200 OK
     Length: 561943 (549K) [image/jpeg]
     Saving to: ‘catanddog.jpg’

     catanddog.jpg       100%[===================>] 548.77K  --.-KB/s    in 0.004s

     2021-08-10 03:41:34 (146 MB/s) - ‘catanddog.jpg’ saved [561943/561943]

     --2021-08-10 03:41:34--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/dog1.jpg
     Resolving storage.googleapis.com (storage.googleapis.com)... 142.250.101.128, 142.250.141.128, 142.251.2.128, ...
     Connecting to storage.googleapis.com (storage.googleapis.com)|142.250.101.128|:443... connected.
     HTTP request sent, awaiting response... 200 OK
     Length: 338769 (331K) [image/jpeg]
     Saving to: ‘dog1.jpg’

     dog1.jpg            100%[===================>] 330.83K  --.-KB/s    in 0.002s

     2021-08-10 03:41:35 (144 MB/s) - ‘dog1.jpg’ saved [338769/338769]

     --2021-08-10 03:41:35--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/MLColabImages/dog2.jpg
     Resolving storage.googleapis.com (storage.googleapis.com)... 142.251.2.128, 74.125.137.128, 142.250.101.128, ...
     Connecting to storage.googleapis.com (storage.googleapis.com)|142.251.2.128|:443... connected.
     HTTP request sent, awaiting response... 200 OK
     Length: 494803 (483K) [image/jpeg]
     Saving to: ‘dog2.jpg’

     dog2.jpg            100%[===================>] 483.21K  --.-KB/s    in 0.003s

     2021-08-10 03:41:35 (147 MB/s) - ‘dog2.jpg’ saved [494803/494803]
```

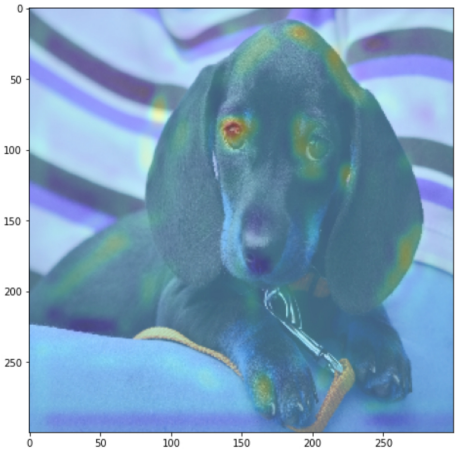
```
[11] # utility function to preprocess an image and show the CAM
     def convert_and_classify(image):

       # load the image
       img = cv2.imread(image)

       # preprocess the image before feeding it to the model
```

```
    img = cv2.resize(img, (300,300)) / 255.0

    # add a batch dimension because the model expects it
    tensor_image = np.expand_dims(img, axis=0)

    # get the features and prediction
    features,results = cam_model.predict(tensor_image)

    # generate the CAM
    show_cam(tensor_image, features, results)

convert_and_classify('cat1.jpg')
convert_and_classify('cat2.jpg')
convert_and_classify('catanddog.jpg')
convert_and_classify('dog1.jpg')
convert_and_classify('dog2.jpg')
```

```
prediction: cat
```



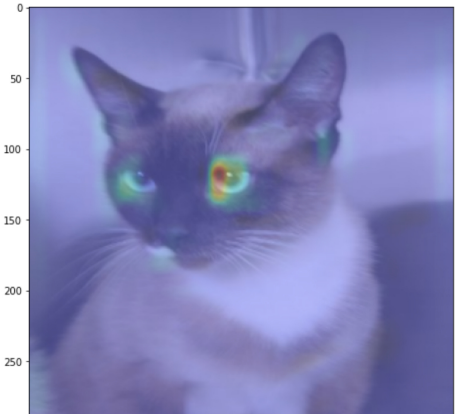

```
sigmoid output: [[0.5589401]]
prediction: dog
```




Let's also try it with some of the test images before we make some observations.

```
[12] # preprocess the test images
     augmented_test_data = test_data.map(augment_images)
     test_batches = augmented_test_data.batch(1)


     for img, lbl in test_batches.take(5):
       print(f"ground truth: {'dog' if lbl else 'cat'}")
       features,results = cam_model.predict(img)
       show_cam(img, features, results)
```

```
ground truth: cat
sigmoid output: [[0.43890488]]
prediction: cat
```




If your training reached 80% accuracy, you may notice from the images above that the presence of eyes and nose play a big part in determining a dog, while whiskers and a colar mostly point to a cat. Some can be misclassified based on the presence or absence of these features. This tells us that the model is not yet performing optimally and we need to tweak our process (e.g. add more data, train longer, use a different model, etc).

✓ 11s completed at 10:41 PM