# Convolutional Neural Networks for Discriminative Localization

## 1.Introduction

Convolutional neural networks (CNNs) are the current state-of-the-art model architecture for image classification tasks. In machine learning, a convolutional neural network (CNN) is a class of deep feed-forward artificial neural networks that have successfully been applied to analyzing visual imagery. CNNs apply a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model can then use for classification.

Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. Figure 1 illustrates a general flow of CNNs to recognize images. CNNs are inputting the 2D structure of training images and output the categorical prediction in the output layer.

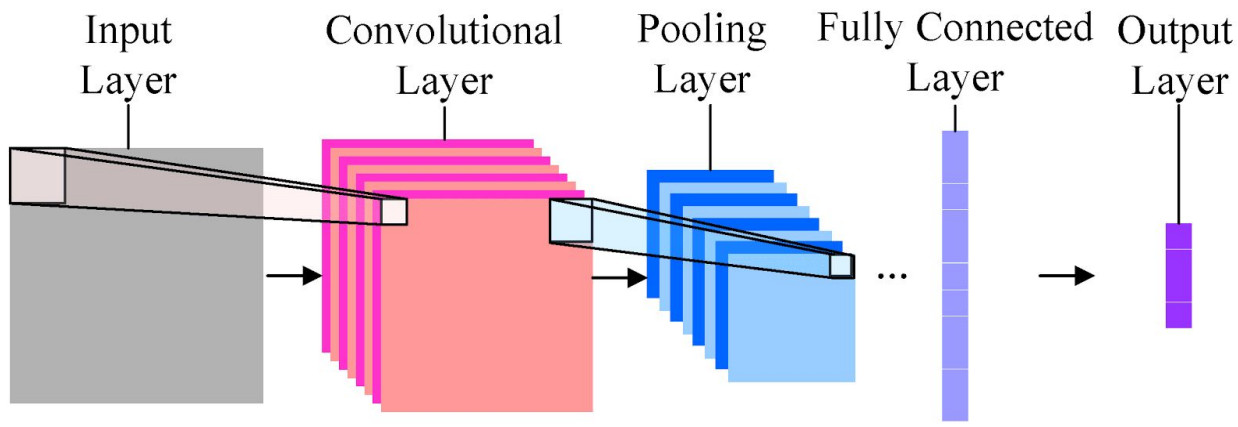



Figure 1. Common CNNs Architecture

Recent studies have shown that being trained on image-level labels, CNNs have remarkable ability to classify images and localize objects [1]. Conventional convolutional neural networks perform convolution in the lower layers of the network. For classification, the feature maps of the last convolutional layer are vectorized and fed into fully connected layers (FCs) followed by a softmax logistic regression layer. However, the fully connected layers are prone to overfitting, hampering the generalization ability of overall network. Furthermore, the remarkable talent to localize objects in the convolutional layers is lost when fully connected layers are used for classification.

Dropout is proposed as a regularizer to improve the generalization ability and prevent overfitting. Another strategy called global average pooling (GAP) to replace the traditional FCs in CNN is proposed to generate feature map for each corresponding category of the classification task in the last network layer [7].

## 1.2 Global Average Pooling (GAP)

Recent studies have proved that global average pooling minimizes the risk of overfitting by reducing the total number of parameters in the model. GAP layers perform an extreme type of dimensionality reduction, where a tensor with dimensions $\mathbf{h \times w \times d}$ is reduced in size to have dimensions $\mathbf{1 \times 1 \times d}$ [5].

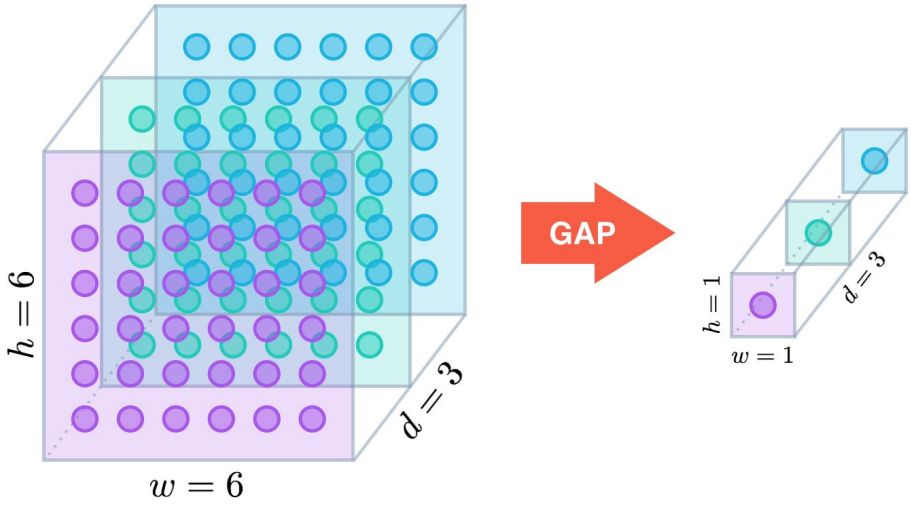



Figure 2. GAP layers reduce each $\mathbf{h \times w}$ feature map to a single number by simply taking the average of all $\mathbf{hw}$ values.

In this work, we detect dog from the cat images and outputs the prediction of a dog along with a heat map localizing the areas of the image most indicative of a dog. Our CNN is consist of four blocks of convolution and pooling layers, followed by one densely layer. The final densely layer has a sigmoid activation function and a node for each potential object category[2]. We train our predefined model in our Dogs vs. Cats dataset and product a heatmap image when input a customer's image into the trained model.

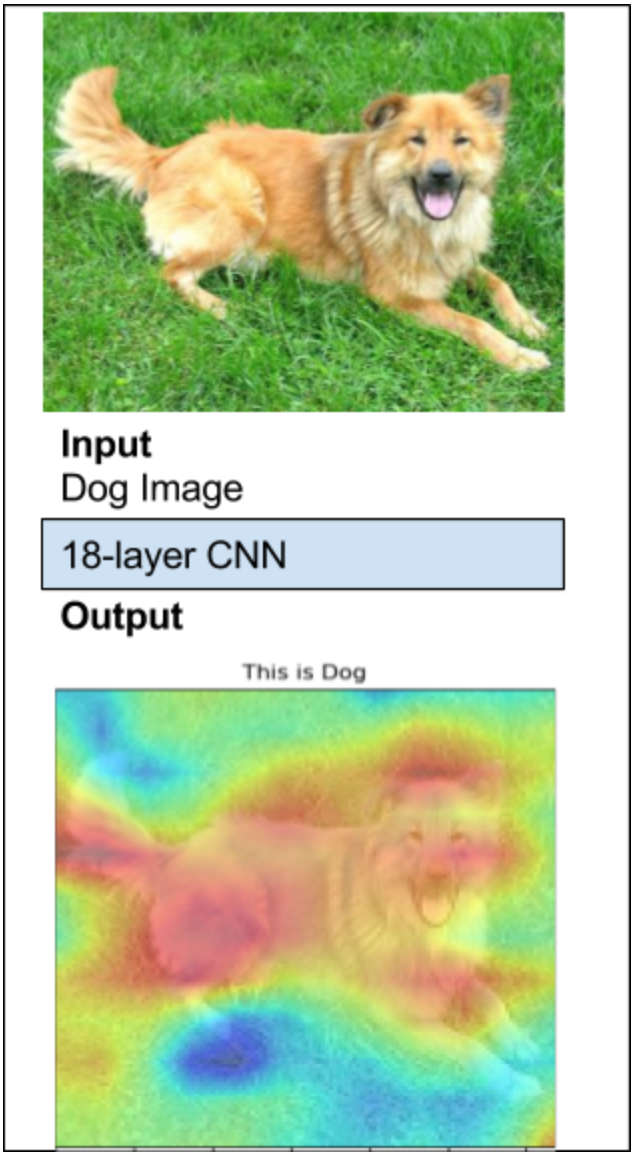



Figure 3. On this example, the model correctly detects dog and also localizes areas in the image most indicative of the dog.

In Section 2, we describe our datasets and measurements. In Section 3 we briefly explain our model architecture and the method to get class activation map (CAP). In Section 4 we present some examples of our model's output and the potential way that improve the accuracy or decrease the losses of negative cases.

# 2. Dataset

In this study, we use the Dogs vs. Cats Datasets released by the Kaggle Competition in 2014 which the dataset have two parts, train file, and test file. The training file contains 25K labeled images containing 12.5K cats' photos and 12.5K dogs' pictures. For the recognition task, we

randomly split the training dataset into training (10K cats, 10K dogs), validation (2.5K cats, 2.5K dogs).

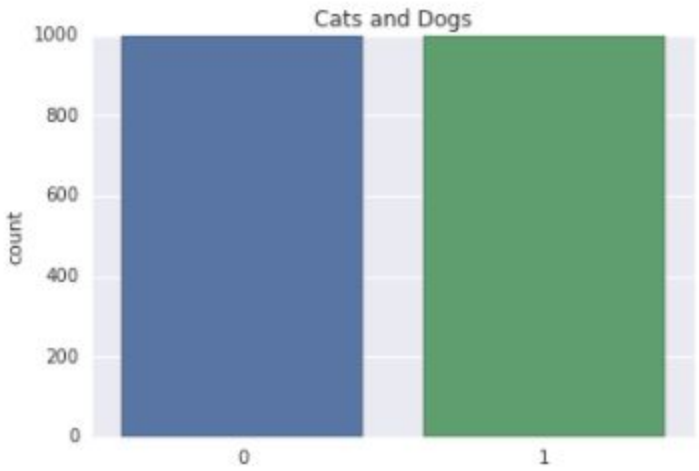



Figure 4 labels the dog as 1, the cat as 0

Figure 5 A quick side-by-side comparison of the animals.

## 2.2 Metrics

Accuracy is a common metric for binary classifiers and it equally considers both true positive (TP) and true negatives (TN).

$$\text{Accuracy} = \frac{TP + TN}{dataset\ size}$$

The dog detection task is a binary classification problem, where the input is a labeled image and the output is a binary label $\mathcal{Y} \subset \{0,1\}$ indicating the absence or presence of dog respectively. We use *loss* value to evaluate the performance of our model during the training steps. For a single example in the training set, we optimize the weighted binary cross entropy loss

$$L(X, y) = -w_+ * y * log(Y = 1|X) - w_- * (1 - y) * log(Y = 0|X)$$

Where $p(Y = i|X)$ is the probability that the network assigns to the label ***i***, $w_+ = |N|/(|P| + |N|)$, and $w_- = |P|/(|P| + |N|)$ with $|P|$ and $|N|$ the number of positive cases and negative cases of dogs in the training set respectively.

# 3. Methodology

CNNs contain three major components: Convolutional layers, Pooling layers, and Dense (fully connected) layers. Convolutional layer performs a set of mathematical convolution to produce values in the output feature map and it typically applies a RELU activation function to the output to introduce nonlinearities into the model. The pooling layer down-samples the image data extracted by the convolutional layer to reduce the dimensionality of the feature map so as to decrease processing time. Dense layers perform classification on the features extracted by the convolutional layers and down-sampled by the pooling layers [6].

We use *class activation map* to refer to the weighted activation maps generated for each image. Given a simple connectivity structure, we can identify the importance of the image regions by projecting back the weights of the output layer on to the convolutional feature maps, a technique we call class activation mapping [4]. A class activation map (CAM) for a particular category indicates the discriminative image regions used by the CNN to identify that category.

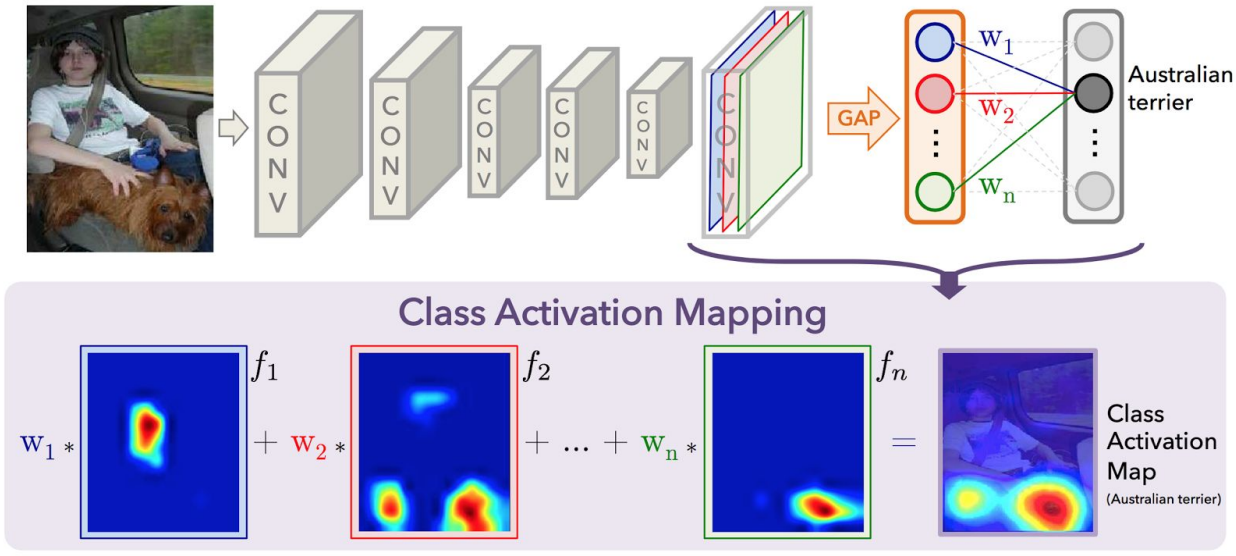



Figure 6. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the specific discriminative regions.

## 3.2 Data Preprocessing

Before inputting the images into the network, each of the images gets a label. We randomly shuffle the list of images and resize the images to 128x128x3 by interpolation. We also augment the training data with random horizontal flipping and convert the integer labels to category labels that are used by the final classification.

## 3.3 Model and CAM

In this work, our model is $18$-layer convolutional network trained on the Dogs vs. Cats dataset. We replace the final max pooling layer with average pooling layer for visualizing the class activation mapping and add one dense layer that has a single output, after which we apply a sigmoid nonlinearity[3]. We use dense connections and batch normalization to make the optimization of our CNNs tractable.

The input layer to the first convolutional layer is an $\mathbf{m} \times \mathbf{m} \times \mathbf{r}$ image where m is the height and width of the image and r is the number of channels (in this work, $\mathrm{m} = 128, \mathrm{r} = 3$ ), e.g. an RGB image has $\mathrm{r} = 3$. The next is a stack of convolutional models that perform feature extraction. The first three models consist of a convolutional layer followed by a max pooling layer. The last convolutional model before the final dense layer is composed of a convolutional layer following an average pooling layer.

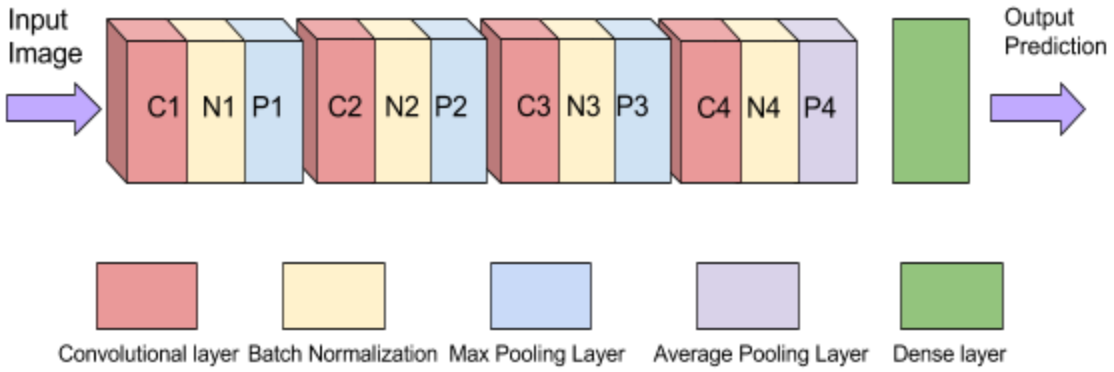



Figure 7. the simple model architecture

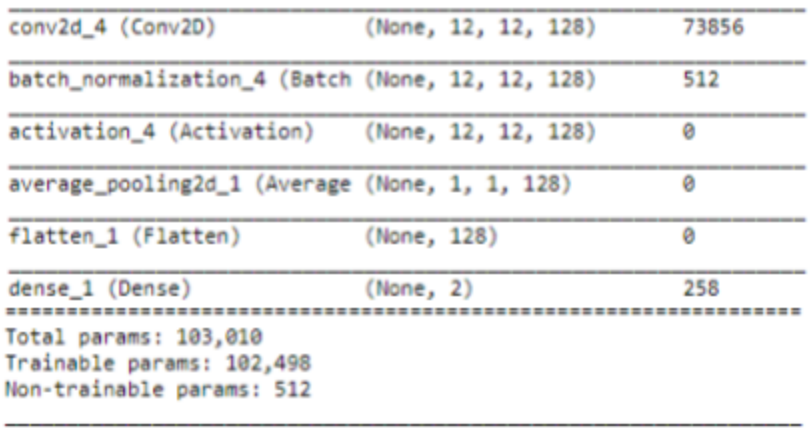



Figure 8 shows the final few lines of model summary. We will focus on the **activation_4** layer, **average_pooling_1** layer, and **dense_1** layer. In fact,the Average Pooling layer is actually a global average pooling layer.

We could find the output of the *activation_4* layer is $12\text{x}12\text{x}128$, this layer contains $128$ activation maps, each map with dimensions $12\text{x}12$ (or it look like sliced bread with different flavors, but these pieces are the same size).The following *average_pooling_2d_1* GAP layer reduces the size of the preceding layer to $(1,1,128)$ by taking the average of each feature map. The next *flatten_1* layer merely flattens the previous output, without any change to the information contained in the previous GAP layer. The object classification predicted by our model corresponds to a single node in the final *dense_1* layer. Every single node is connected to every node in the former Flatten layer.

For a given image, We suppose $f_k(x, y)$ represent the activation of unit $k$ in the last convolutional layer at spatial location $(x, y)$ (from the prior illustration, $x \subset \{x_1, x_2, \ldots x_{12}\}$, $y \subset \{y_1, y_2, \ldots y_{12}\}$, $k \subset \{1, 2, 3, .., 128\}$). Thus, for unit $k$, the result of performing global average pooling, $F_k$ is $\sum_{x,y} f_k(x,y)$. For the class dog, the input to the dense layer, is $\sum_k w_k F_k$ where $w_k$ is the weight corresponding to class dog for unit $k$. Essentially, $w_k$ indicates the importance of the output of average pooling. In this work, we need to compute the sum of multiplication.

$$w_1 * f_1 + w_2 * f_2 + \ldots\ldots w_{128} * f_{128}$$

Where $w_i$ is a scalar and the $f_i$ is a matrix of size $128\text{x}128$.

## 3.4 Implementation

The training as the following steps:

1. Load the training data and randomly split the data into two parts, training and validation.
2. Define the network structure and training and validation parameters.
3. Define the loss function, accuracy.
4. Train the CNNs, logging the training/validation loss and accuracy every epoch.
5. Save the trained model.

6. Plot the logged loss and accuracy.

In the training phase of the CNNs where we estimate the CNNs performance, we use Adaptive Moment Estimation which computes adaptive learning rates for each parameter to reduce training losses. Adaptive Moment Estimation empirically works well in practice and compares favorably to other adaptive learning-method algorithms. We build the CNNs using Keras as provided part of this submission. We need to use the trained model to generate CAM when inputting a customer's image.

The CAMs phase can be split into the following steps:

1. Input and resize a new dog/cat image
2. Convert the shape of the image to (1, 128, 128, 3)
3. Extract the output of last convolutional layer as all activation maps from the trained model.
4. Redefine the new model to get CAMs and Pass the preprocessed image data through the new model
5. Get interested CAM via all activation maps and prediction
6. Plot the image of the original image and class activation map.

In the CAMs period, all code in the *get_model.py* file is finishing the work of the average pooling layer which outputs the spatial average of the feature map of each unit at the preceding layer. We redefined the output of the trained model and extract the output of the last convolutional layer ($12 \text{ x } 12 \text{ x } \mathbf{128}$). The vector of the prediction and the filtered images will output when inputting the preprocessed image data into the new model. Upsample to each filtered image to the original image size ($128 \text{ x } 128 \text{ x } \mathbf{128}$, the bold number represent the depth of one image) and reshape each unit to $16348 \text{ x } \mathbf{128}$ to multiplying with corresponding weights, ($128*128$ is equal to $16348$).

In our work, we have a set of $class\ activation\ map$ for each class of encoded output. There are *class activation maps* for cat and dog classes.

$$class\ activation\ map\ =\ w_0 \cdot f_0\ +\ w_1 \cdot f_0\ +\ .....\ +\ w_{128} \cdot f_{128}$$

## 3.5 Result

The output from the redefined model is a linear combination of all class activation maps. We will have a vector of class activation map and the vector of prediction after loading the testing image to the new model. We obtain the interested class activation map via the index of the maximum value in the predicted vector and get the class activation map of testing images during the CAM period.

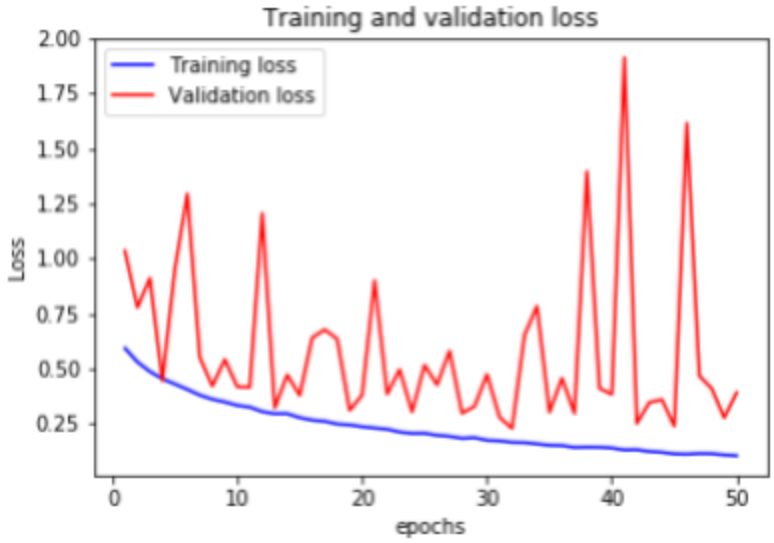



Figure 9 illustrates binary cross entropy losses of the predicted category and true label of the Dogs vs Cats dataset over 50 epoch (each epoch iterate 200 steps). As the number of iterations increases, the overall loss value decreases.

We have also used normalization of the convolutional layer weights to get rid of the change of the parameters from the previous layer and speed up convergence. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. We use the Batch Normalization function provided by Keras which normalize the activation of the previous layer at each training mini-batch.

Use the simple CNN architecture described in Section 3.3, we achieve 80% accuracy in Dogs vs Cats dataset after 1 hours of training with GPU on the Floydhub cloud. Next, we apply this model to discriminative localization.

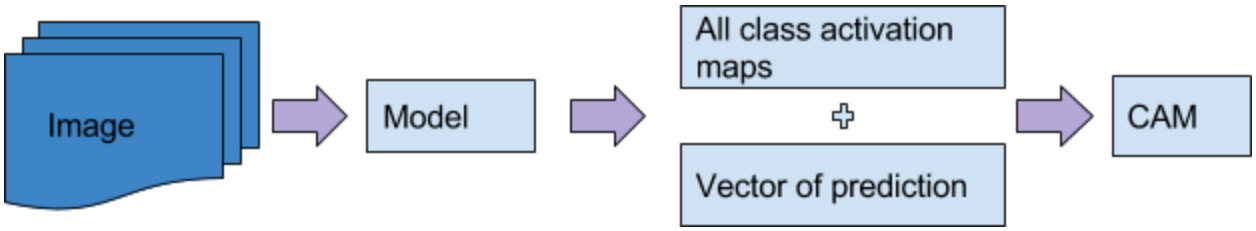
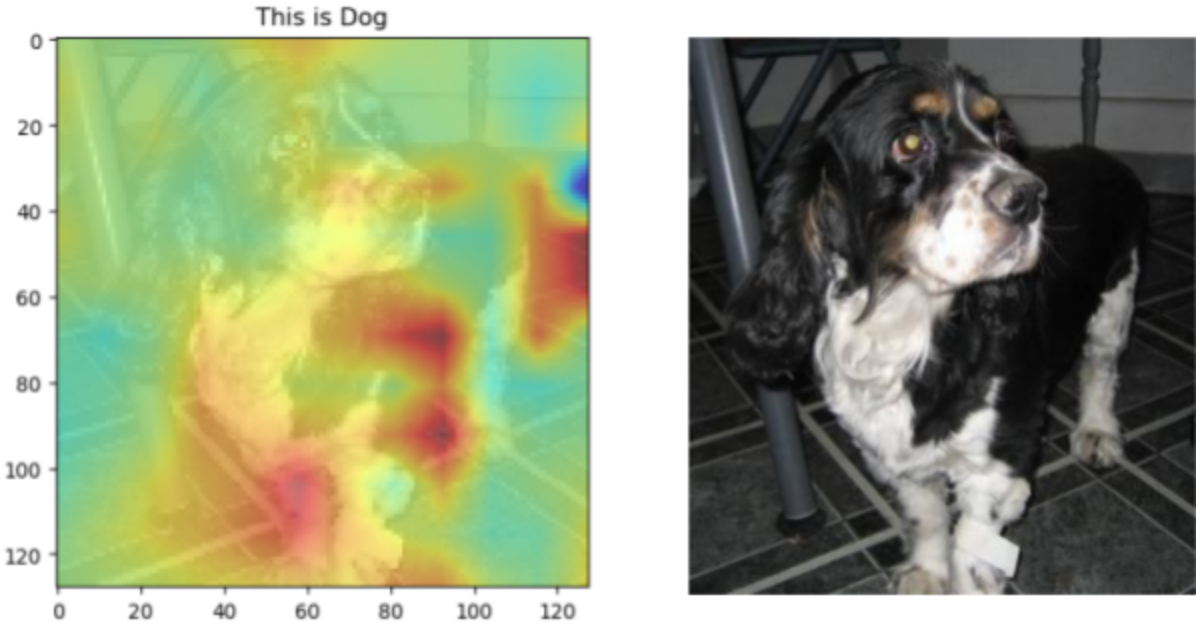



Figure 10. Shows the flow of the discriminative localization

We randomly choose some images from the test file (the images is unlabeled in this file) and create them as testing images for discriminative localization as demonstrated in Figure 11. Then we load these images to the *get_cam.py* file and highlight the discriminative areas where the model classify that this is a dog. The class activation map are plotted to observe the

discriminative localization ability of our simple model. The output in a class activation map with size 128x128 compared with the original image.

From the examples shows in Figure 11, we can discover the discriminative areas in the image that the model classifies the object to dog or cat. The animal in the second image is a cat, though the model classifies it in the dog category. The activation in the second grouped images is much smaller than the others. Our model successfully gives an outline of the dog by activating the area concentrated on the tail, thigh, and head.



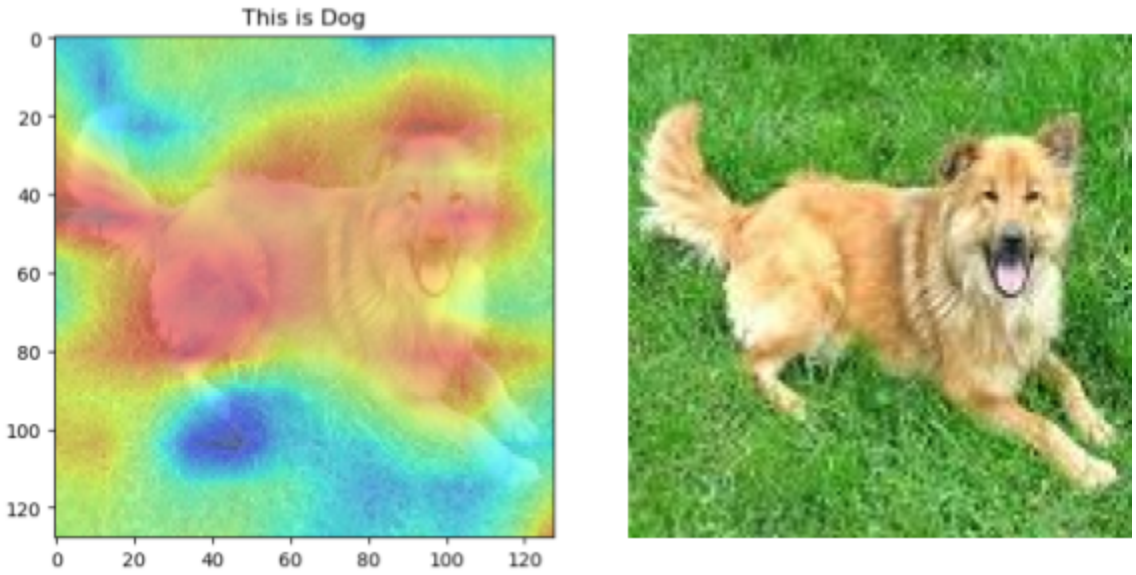



Figure 11 Examples of Discriminative Localization

# 4. Conclusion

In this work, we train a simple CNN model algorithm for size 128x128x3 on Dogs vs Cats dataset and utilize it to dog/cat images for discriminative localization. We show that the discriminative localization of the similar objects is possible according to the proposed model architecture. Our discriminative localization algorithm shows about 80% accuracy and around 0.5 binary cross entropy losses. The training parameters need to be adjusted to meet more stable losses and higher accuracy.

Despite the results are encouraging, from the loss and accuracy, the next step is reducing the degree of oscillation in the training period. In the training, selecting an appropriate *batch size* and the number of epochs is trouble since the inappropriate value making the loss fluctuate violently. For instance, a model with larger batch size couldn't work more stable if the number of epochs isn't fit. We can increase the number of convolutional layers or build parallel convolutional layers to extracting deeper features from the training images. This might help the model performance more robust. The discriminative localization can help us better understand how the convolutional neural network recognize the images and it also can be applied to pattern discovery.

# Reference:

[1]B. Zhou, A. Khosia, A. Lapedriza, A. Torralba. Object detectors emerge in deep scene cnns. International Conference on Learning Representations, 2015.
[2]https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/

[3]CheXNet: RAdiologist-Level Pneumonia Detection on Chest X-Ray with Deep Learning.
[4]Bolei Zhou, Aditya. Learning Deep Features for Discriminative Localization.
[5]https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
[6]https://www.tensorflow.org/tutorials/layers
[7]https://arxiv.org/pdf/1312.4400.pdf