XAI - Classification and Object Detection

(Explainable AI/ Interpretable AI)

Ankur Bhatia
Email: bhatia.ankur24@gmail.com
Website: https://ankurbhatia24.github.io/

Suppose that all the training images of *the bird* class contain a tree with leaves. How do we know whether CNN is using bird-related pixels, as opposed to some other features such as the tree or leaves in the image?

Explainability/Interpretability in DNN is essential because if we need to deploy the models in real-life scenarios like self-driving cars or in medical diagnosis, we can't just rely on a black box to make predictions for us. We need to be sure that the model will work accurately in possible conditions.

Some methods that are generally used for reasoning the classification label (model understanding) are:

1. CAM: Class Activation Maps - for identifying discriminative regions used by a particular class. CAM works for modified image classification CNN's (not containing any fully-connected layers). CVPR 2016 - Zhou et al. "Learning Deep Features for Discriminative Localization". The activation maps are generated using the global average pooling layers in the CNN. Here is the read: http://cnnlocalization.csail.mit.edu/
2. Grad-CAM (Gradient - weighted Class Activation Mapping) is a generalization of CAM for any CNN based architecture. ICCV 2017 - Selvaraju et al. "Visual Explanations from Deep Networks via Gradient base Localization".
3. Saliency maps is a technique that is used to see which part of the input image contributes to the final predictions (for classification: final class labels, for object detection: final bounding box prediction). This technique is based on computing the gradients of the class score wrt to the input image. This tells us how output category value changes with respect to a small change in input image pixels. All the positive values in the gradients tell us that a small change to that pixel will increase the output value. Hence, visualizing these gradients, which are the same shape as the image should provide some intuition of attention.

   As simple as this: $\frac{\partial output}{\partial input}$

Technique 1:
This work has implemented Visualizations for Grad-Cam and Saliency maps. In case of Grad-CAM, it can identify regions that CNN looks at for a particular class prediction. I have gone through this tutorial for understanding Grad CAM: https://medium.com/@mohamedchetoui/grad-cam-gradient-weighted-class-activation-mapping-ffd72742243a. In my code I have not implemented it by writing the whole functions as in

the tutorial (like calculating the gradients and backpropagating it to the input), rather I have used Keras-Vis library which has a good implementation of this.

1. Say for a class c, we compute the gradient of the last output layer of the network y^c with respect to the feature maps A of the last convolution layer $\frac{\delta y^c}{\delta A^K_{ij}}$.
2. These gradients flowing back are global-averaged-pooled to obtain weights w^C_K.

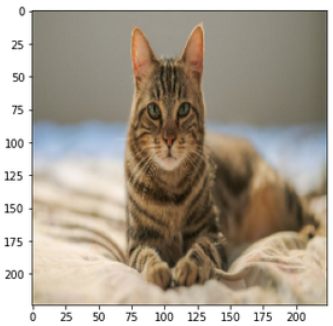
   $w^c_K = \frac{1}{Z}\sum_i \sum_j \frac{\delta y^c}{\delta A^K_{ij}}$, where w^C_K captures the importance of the feature map K for a target class c.
3. The Grad-CAM heat-map is a weighted combination of the feature maps with a ReLU (we want the +ve pixels, that are +vely correlated for the class prediction C).

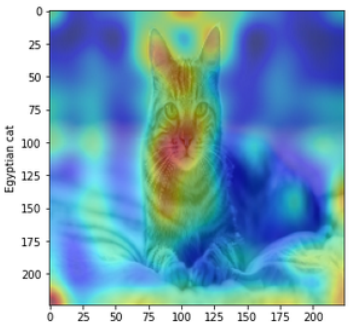
   $$L^c_{Grad-CAM} = ReLU(\sum_K w^c_K A^K)$$
4. **Disadvantage**: Can't properly localize an object, Can't localize multiple objects. Also since we are performing a global average pool for the weights w^C_K, it means that all the gradient feature maps are given equal weightage.
   Grad-CAM++ is a leveraged version of this. (Anirban Sarkar, Aditya Chattopadhyay, Prantik Howlader, Vineeth Balasubramanian - IIT Hyderabad)

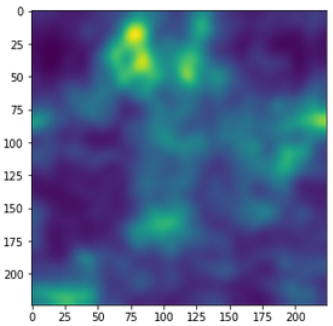
VISUALIZATIONS:

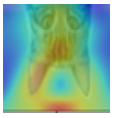
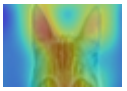
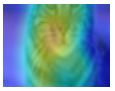
1. Visualizations have been done in order to see what features the model is using to predict the particular class.
   The 1st is the original image, 2nd is the Grad-CAM and 3rd is the Saliency Map.
2. Various augmentation on the input image has been performed to see the localization capability of the algorithm and the exact features learned.
3. Pre-trained VGG19 is used for the class prediction with imagenet weights.
4. Keras Visualization Toolkit: https://raghakot.github.io/keras-vis/ is used for generating the Grad-CAM's and Saliency maps for different classes. (By Kotikalapudi, Raghavendra and contributors)

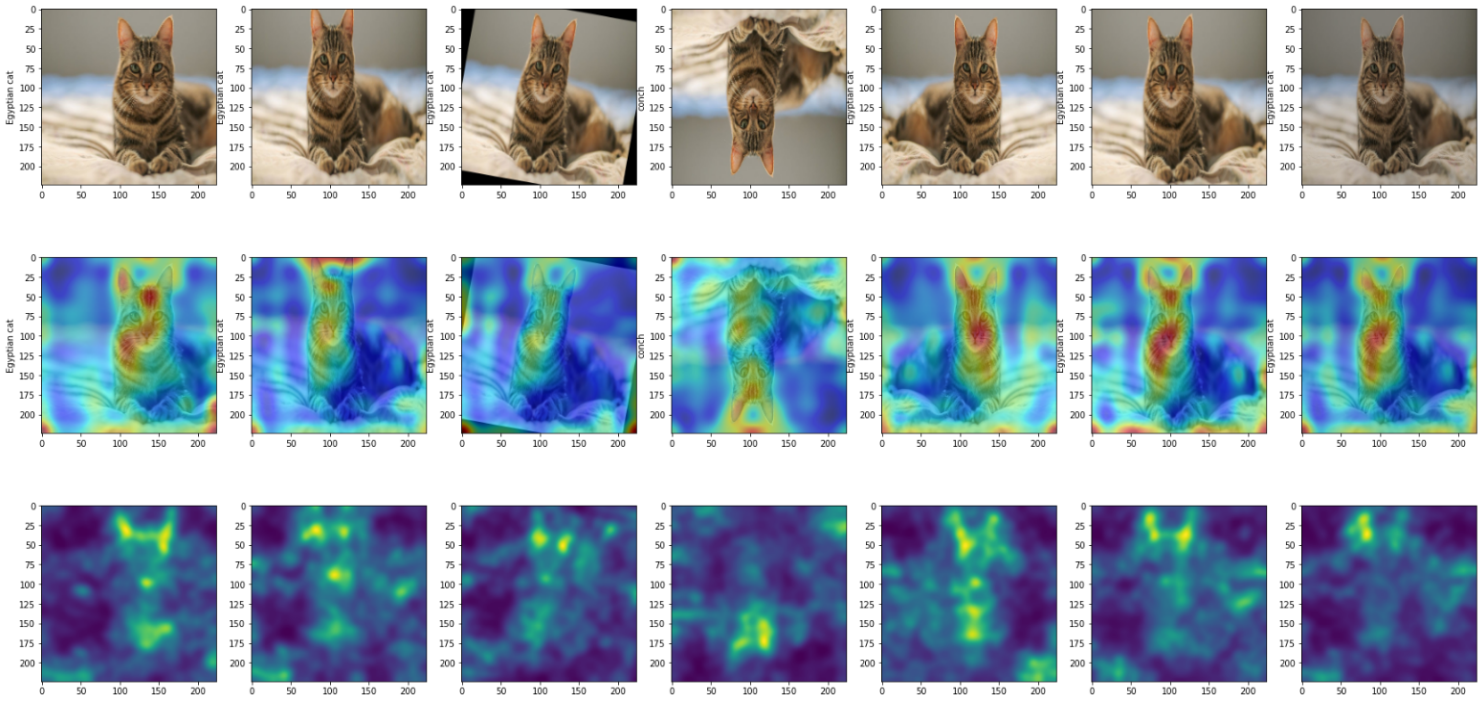
1. From the first example, we can see that the main contributing features cats ears and the middle of its face (with the mustache).

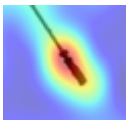
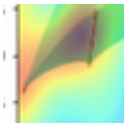
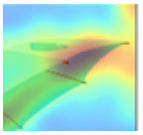


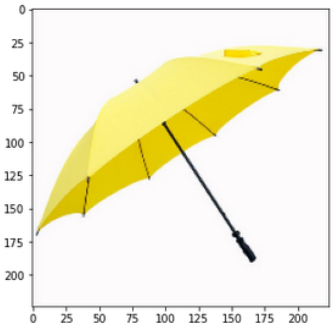


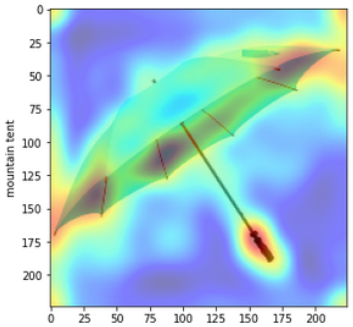


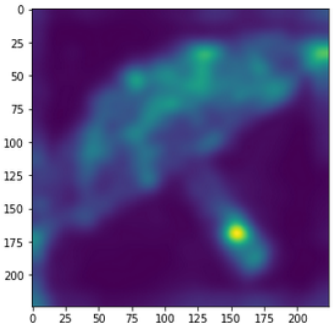
We can see in the 4th image (vertical flip), the prediction of the model is Conch instead of true label Egyptian Cat. On comparing the different visualizations we can say that  (something like a triangular feature) instead of  +  has caused the change of labels.



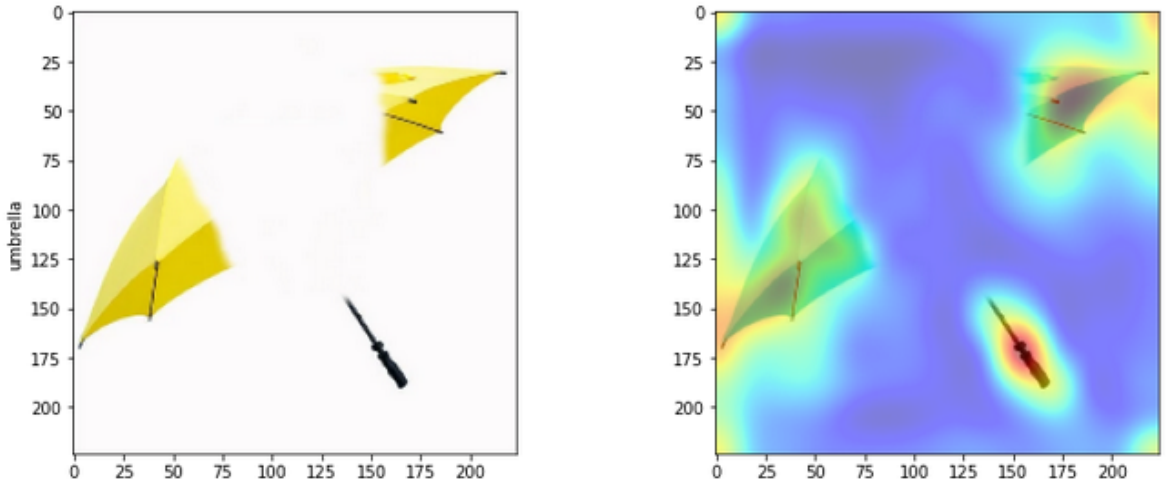
2. From the second example, we can see that the major contributing features are -  ,  and  (end of the stick, corners of the umbrella)





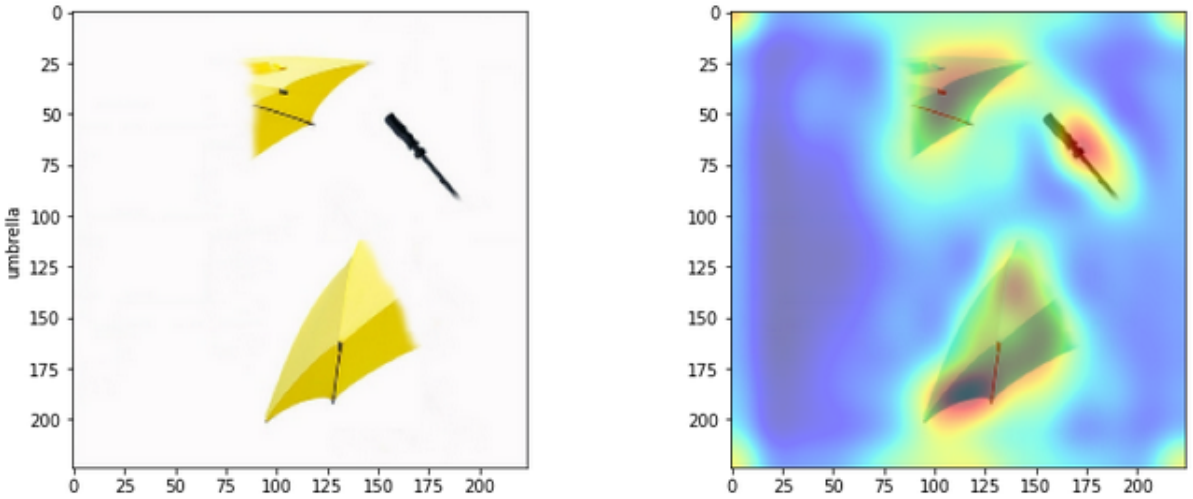


To check the accuracy of this, I just edited out the unimportant part of the image and tested the model for the leftover image.

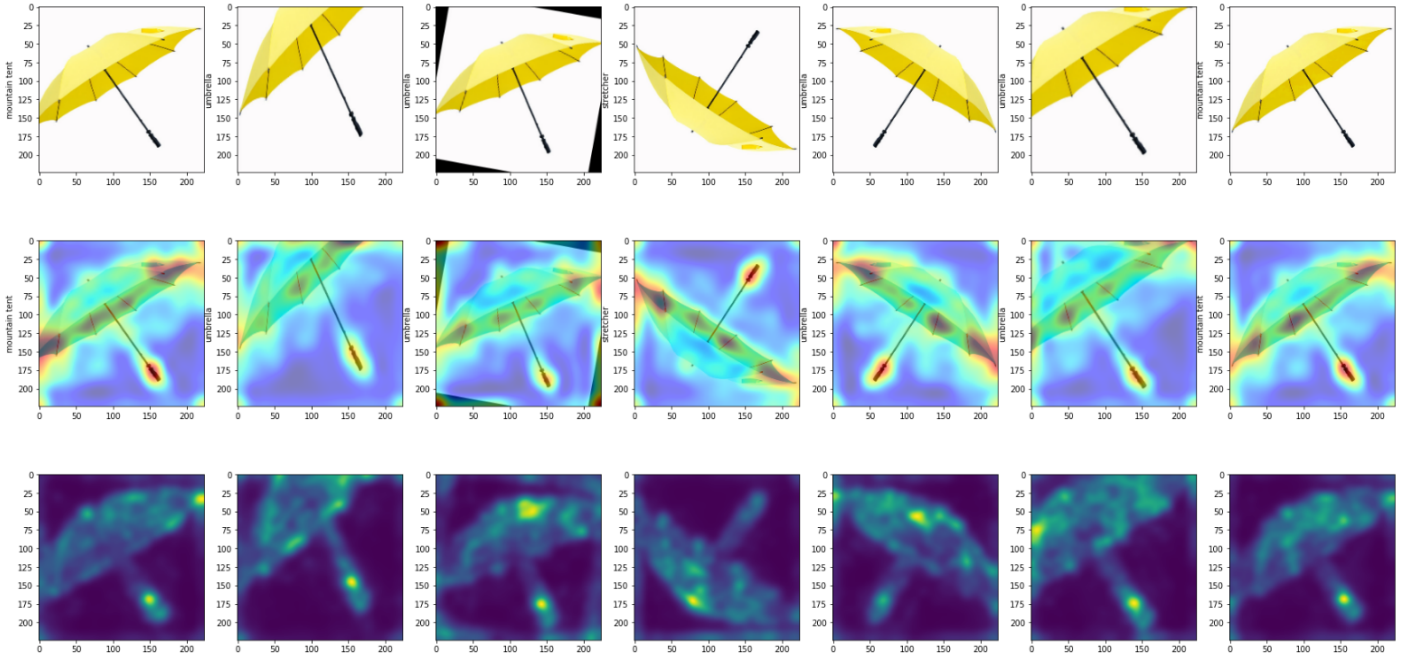


Still, the model predicted an umbrella - which shows that the model is predicting only from the selected important regions.

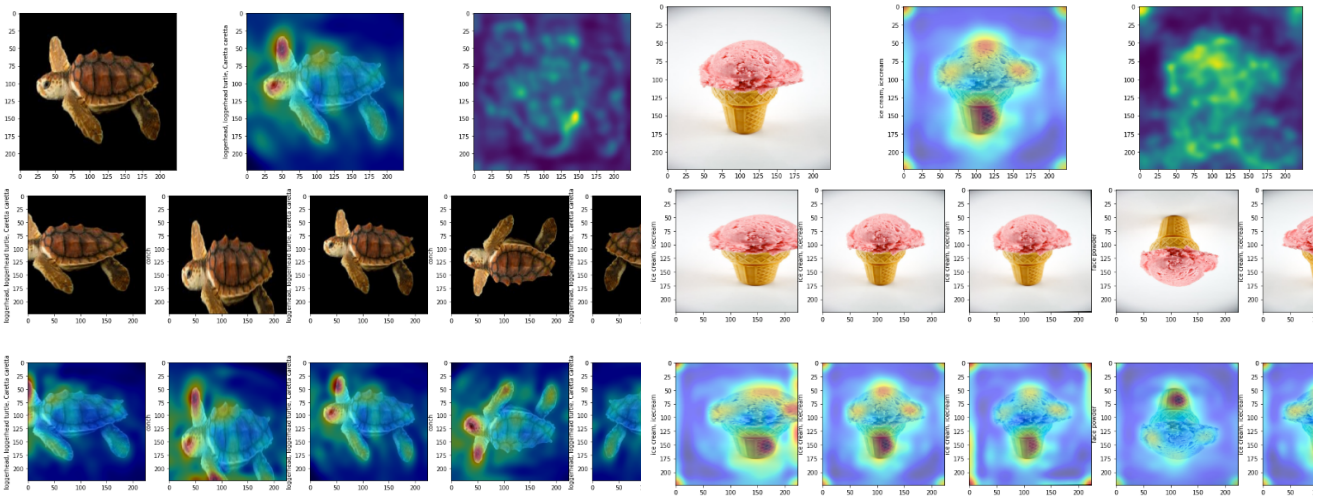
Also, as we know that the convolutions are spatially invariant - so changing the localization of the umbrella (containing the same parts as above) also produces the correct label. It recognizes the umbrella from the features it requires.



Here also some augmentations - vertical flip (has predicted **stretcher**) and brightness (has caused to predict **mountain tent**)
These examples like brightness etc. can be treated as an adversarial example.

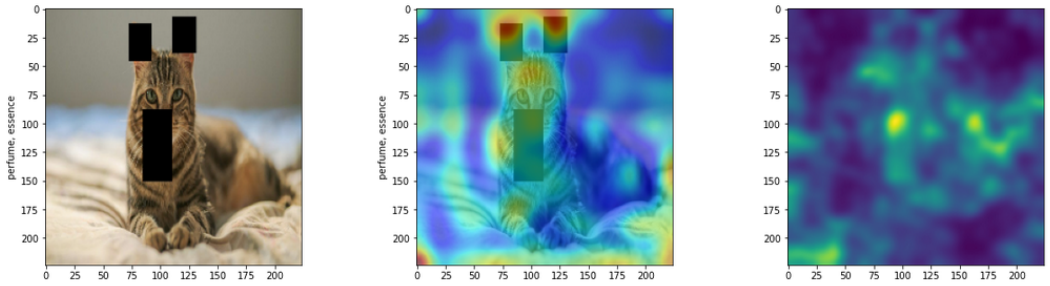


Similarly, I have tested this above on 7 other classes:
Tractor, Ice-cream, Mushroom, turtle, dogs and cats, Pineapple, Potato (spaghetti squash)

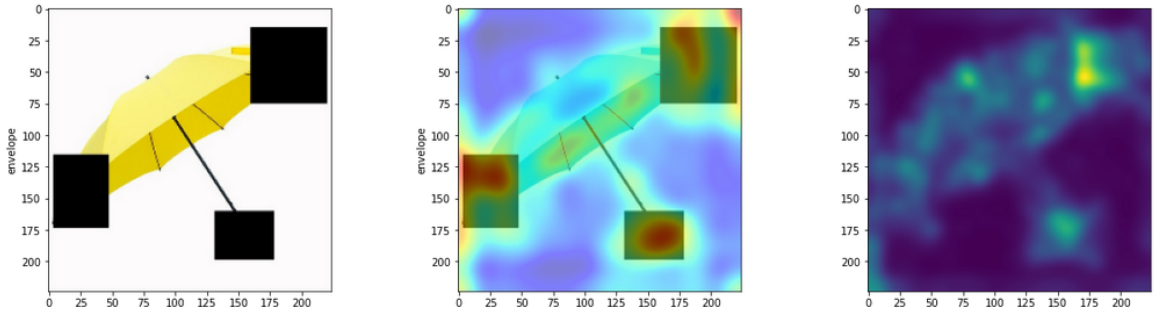


Further, I have performed a few experiments to see if masking the areas that are essential (as by saliency maps and Grad-cam), will there be a change in the classification label.
I have targeted the brightest areas from the salience maps and masked those areas with black rectangles.

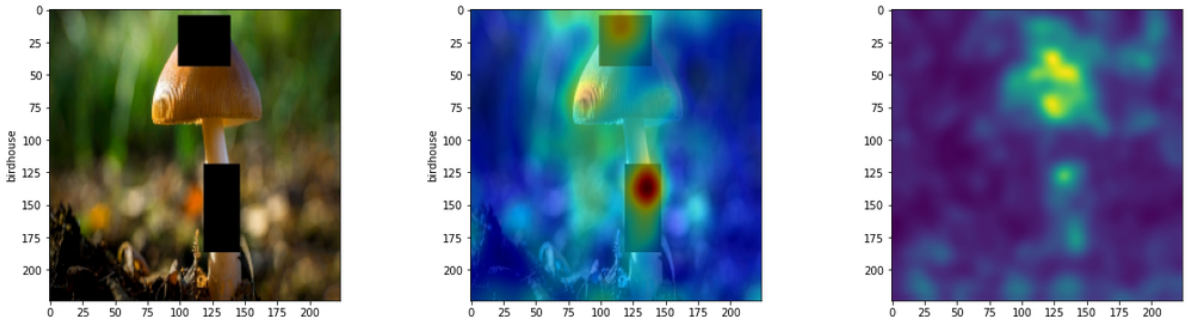
1. We can see that in the **Egyptian cat** example the classification is changed to **perfume, essence** label.



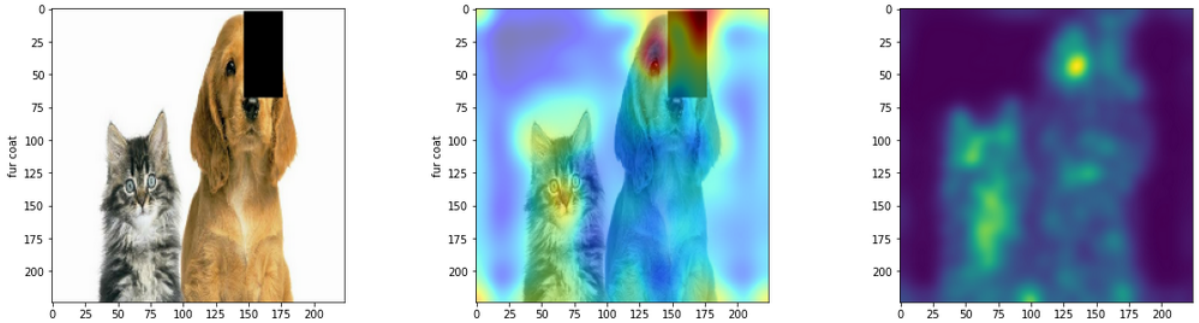
2. For the **umbrella**, we can see that the label is changed to an **envelope.**



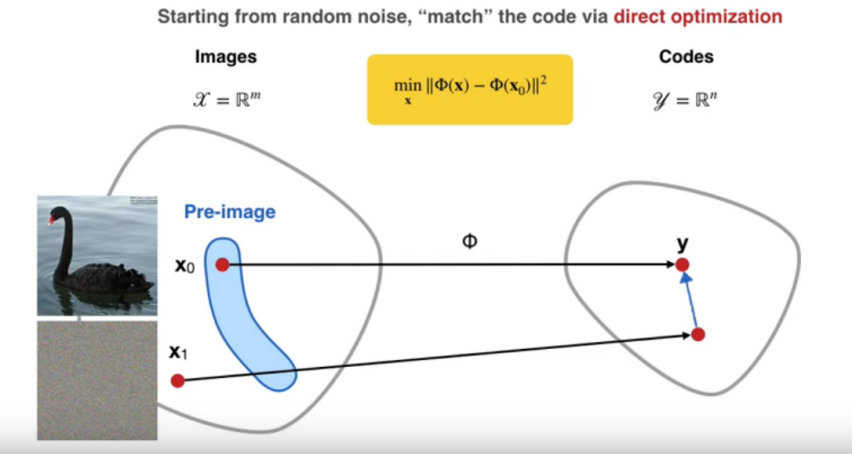
3. For the **mushroom**, we can see that the label is changed to **a birdhouse.**



4. For the **`bloodhound`**, we can see that the label is changed to a fur **coat.**

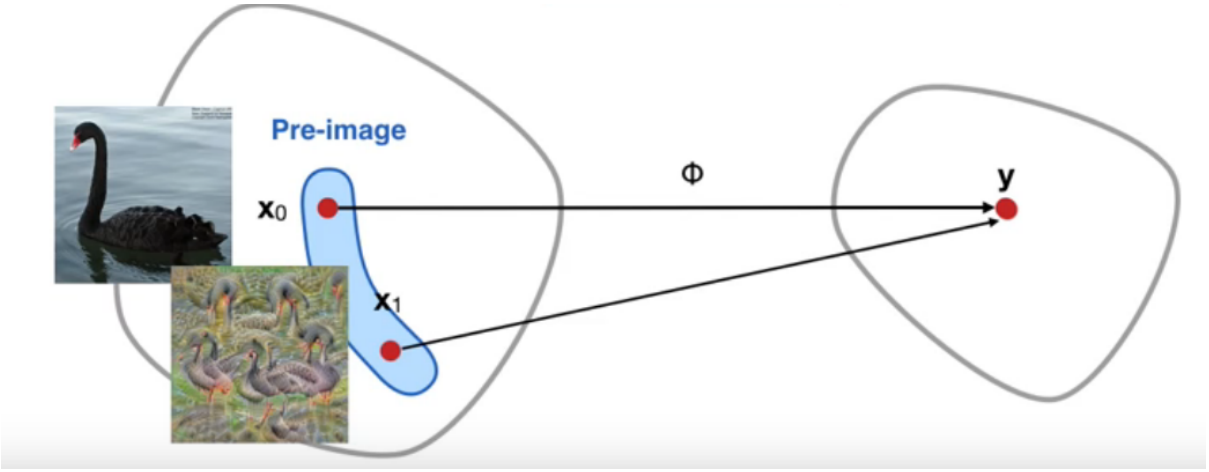


Technique 2 (not in the code - can be implemented):
I have thought of implementing this (below) type of architecture for understanding the model more. This technique has been explained in the CVPR 2018 Tutorial on Interpretable Machine Learning for Computer Vision (here).



In this method, for understanding what the model learns or is the model learning the right features, what we can do is -

1. Take a pretrained network (eg vgg19) on certain class labels (eg imagenet).
2. Then remove the classification (last layer) layer of the network (or remove the layers until which layer you want to figure out that whether this layer has learned something related to the class or not?).
3. Say we have removed the last layer from the vgg19, so we have got a fully connected dense layer of 4096 units.
4. Now take a random noisy image (x1 as shown) and x0 (the image from the class C for which you want to observe the learning).
5. Now for every example of class C, feed it through the network to get the 4096 sized vectors, do the same with noise and use L2 distance for calculating the losses and backpropagate the losses to update the input noisy image (Minimize using SGD)
6. In this process, the model will pull (sample) the random noise image on the pre-images of the function.



I didn't get much time for its deep understanding and implementation.

WEBSITES I REFERRED TO:

1. CVPR Tutorial on Interpretable Machine Learning for Computer Vision: https://www.youtube.com/watch?v=1aSS5GEH58U
2. CVPR 2020 - https://interpretablevision.github.io/
3. MIT CAM - http://cnnlocalization.csail.mit.edu/
4. Blog: https://www.machinecurve.com/index.php/2019/11/18/visualizing-keras-model-inputs-with-activation-maximization/
5. Blog: https://www.machinecurve.com/index.php/2019/11/25/visualizing-keras-cnn-attention-saliency-maps/#using-saliency-maps-to-visualize-attention-at-mnist-inputs
6. Github: https://github.com/raghakot/keras-vis/tree/master/vis (https://raghakot.github.io/keras-vis/)
7. Blog: https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-gradcam-554a85dd4e48
8. Blog: https://medium.com/@mohamedchetoui/grad-cam-gradient-weighted-class-activation-mapping-ffd72742243a
9. Siraj Raval - XAI - https://www.youtube.com/watch?v=Y8mSngdQb9Q
10. Blog: https://medium.com/swlh/deep-learning-and-medical-imaging-interpret-what-the-model-sees-3a1a35b2a323

Readings and Other Resources:

1. https://github.com/topics/explainable-ai
2. https://gist.github.com/sooheang/2b1eb9b6f3a52b4ca9c6d62ebad3dce4
3. https://arxiv.org/abs/1312.6034
4. http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf
5. https://arxiv.org/abs/1610.02391
6. https://github.com/pbiecek/xai_resources