Computer Vision
Improving Image Segmentation with Saliency Detection and GrabCut

Abstract:
The objective of this project is to create a computer vision pipeline that can automatically eliminate backgrounds from images thus extracting the salient foreground of the image.

We have tried implementing the aforementioned pipeline by dividing the pipeline into two main components :

1) Image Saliency Detection
2) GrabCut

The idea is to make segmentation more accurate by first finding important parts of the image using patterns and colors, and then refining the segmentation with GrabCut. This approach helps separate objects from the background more accurately, especially in complicated images.

Problem Statement:
We aim to develop a computer vision pipeline that is able to carry out effective foreground and background segmentation by first detecting the salient objects present in an image, create a rough mask of the object followed by a bounding box around the object and then finally extract the foreground by removing the non salient background of the image.

In this project we aim to explore the effectiveness of combining the two approaches: Saliency Detection and GrabCut together. Specifically, we have tried to explore the pipeline's effectiveness in isolating foreground objects from cluttered backgrounds and enhancing the visibility of salient regions within images.

Possible Use Cases:
This has wide ranging applications such as allowing users to replace the background portion of an image with transparent backgrounds or customized backdrops, providing unbiased datasets for ML/DL models etc. The process of eliminating backgrounds from photos has a wide range of uses, including producing high-quality portraits, capturing product images for online commerce, and improving the visual attractiveness of social media content.

Implementation:

Saliency Detection:

The saliency detection component involves several key steps:

1. **Segmentation:** The image is segmented into multiple regions using the SLIC (Simple Linear Iterative Clustering) algorithm, which partitions the image based on similarities in color and texture. This allows us to identify regions of the image that are similar to each other and the regions that are different to each other.

2. **Saliency Calculation:** In this step we aim to broadly identify the salient region of the image.

   Two types of saliency maps are computed:
   - **Pattern Saliency**
   - **Color Saliency**

   Pattern saliency highlights regions with distinct patterns or textures, while color saliency emphasizes regions with unique color distributions from the rest of the image.

3. **Combining Saliency Maps:** The individual pattern and color saliency maps are combined to produce a unified saliency map, which highlights the most visually significant (salient) regions in the image.

GrabCut:

In our pipeline, we have tried implementing the Grabcut algorithm using graph cuts. GrabCut is an iterative algorithm used for refining image segmentation. It begins by initializing Gaussian Mixture Models (GMMs) for both foreground and background based on the provided saliency masks. These masks serve as initializations for the segmentation process.

Next, GrabCut constructs a graph representation of the image, where pixels are represented as vertices and edges encode pixel affinities based on color similarity and spatial proximity. This graph serves as the basis for defining an energy function that GrabCut aims to minimize during segmentation.

The algorithm then iteratively optimizes segmentation by updating the foreground and background labels while minimizing the energy function. This optimization involves computing the probabilities of each pixel belonging to the foreground or background based on the GMMs and updating the segmentation accordingly.

Pipeline, Results and Discussions

Input Image

Image Saliency

The saliency generation process consists of two main parts, pattern distinctness and color distinctness. These are explained as follows :

Pattern Distinctness

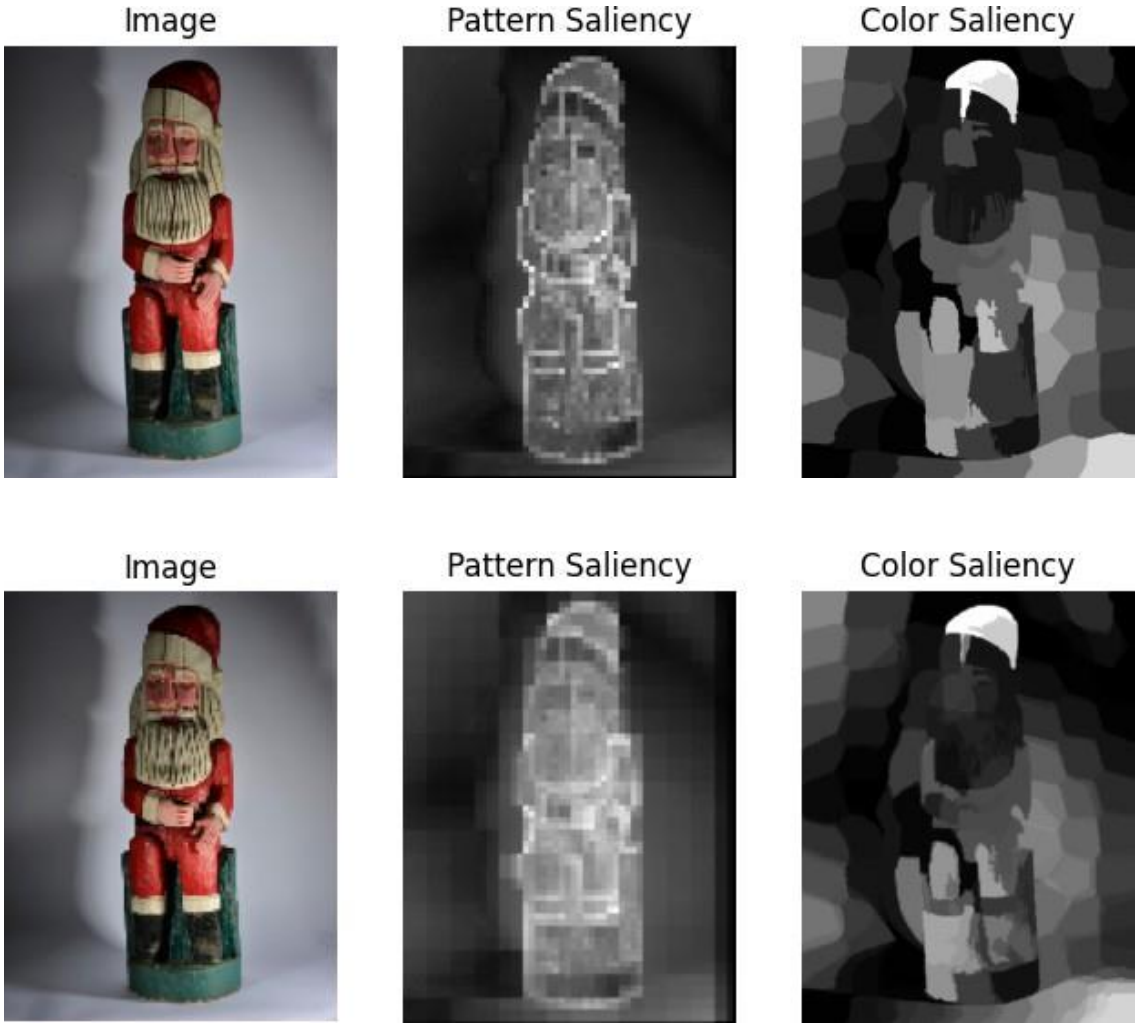
It is usually observed that the salient object in an image has a unique pattern. In the pattern saliency step we try to figure out the patches in the image that have a distinct pattern. To do this we first segment the image into multiple segments using the SLIC algorithm. This divided the image into regions on the basis of both color and region similarity. After this step we identify the patches which are having the highest variance (in RGB channels). These high variance patches are indicative of the salient region of the image but still may contain areas that are nowhere close to the actual salient region. To then better localize the actual salient patches we find the L1 Norm of each patch from the average patch in PCA coordinates. PCA is calculated using the high variance patches that were calculated. The patches that have a smaller L1 norm are considered to be similar to the average patch and the patches having a higher L1 norm are considered to be distinct from the average patch thus counted to be more salient than the rest of the image.

Color Distinctness

Along with pattern distinctness it is also usually observed that salient parts of the image also have a distinct color distribution as compared to the rest of the image. To do this we use the segments as generated using the SLIC algorithm in pattern distinctness. We first calculate the average color of each SLIC region thus giving us the color of each SLIC segment, then we calculate the overall average color of the image by averaging out the colors of the SLIC segments. FInally, we measure the euclidean distance of each SLIC segment's color from the overall average color of the image and normalize it to the maximum distance. Regions having a lower distance are considered to be non-distinct regions whereas regions having a higher distance are considered as distinct from the rest of the image.

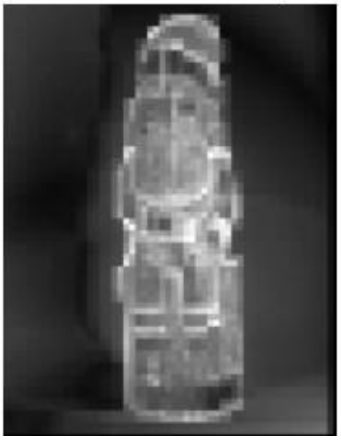
Finally the pattern saliency maps and the color saliency maps are combined using simple multiplication to get the final saliency map of the image.

The whole above process is done for 1 image. We can repeat the process on different scales of the image and then merge the output to get a better overall saliency map of the image.

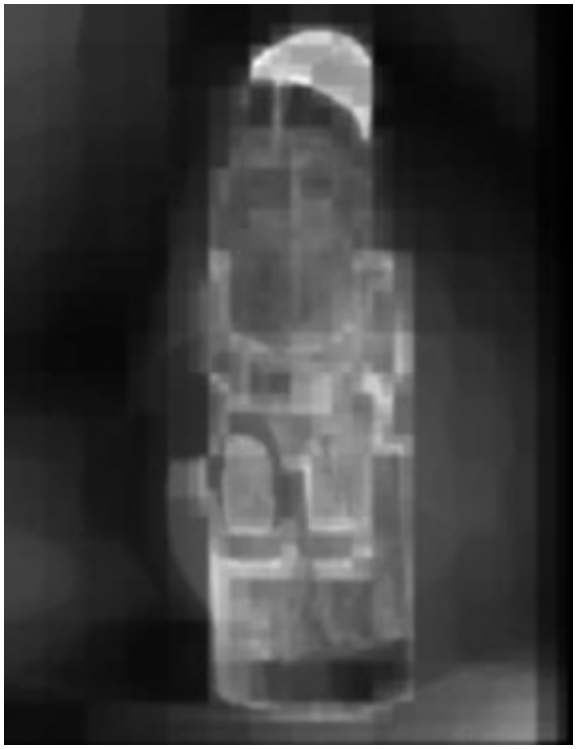


Image

Pattern Saliency



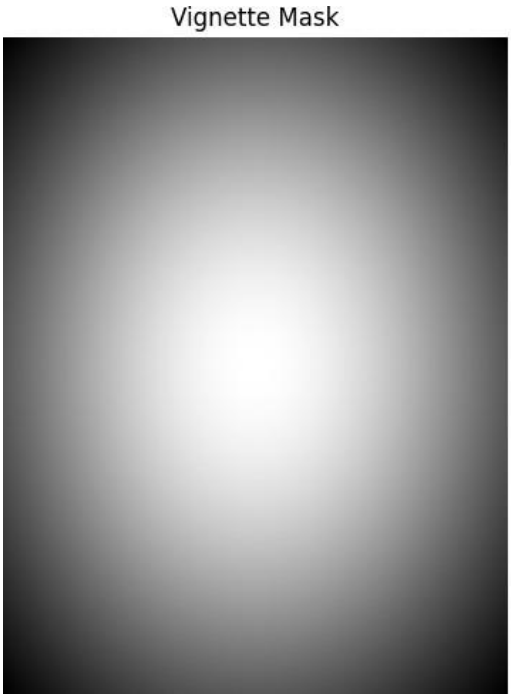
Color Saliency

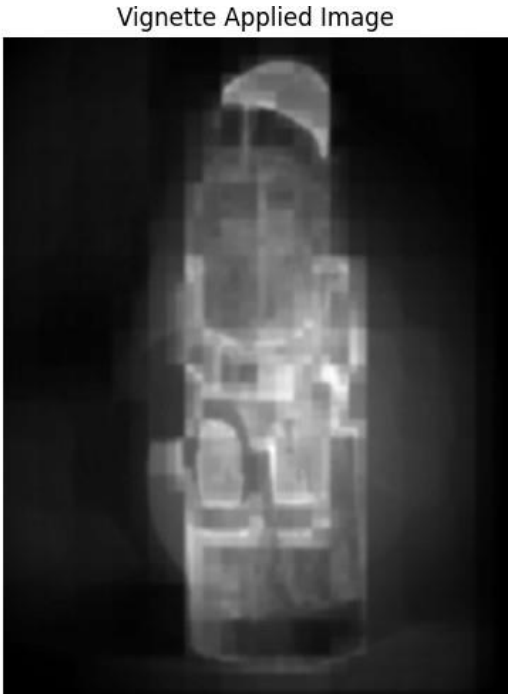


Final Saliency Map

Vignette Mask

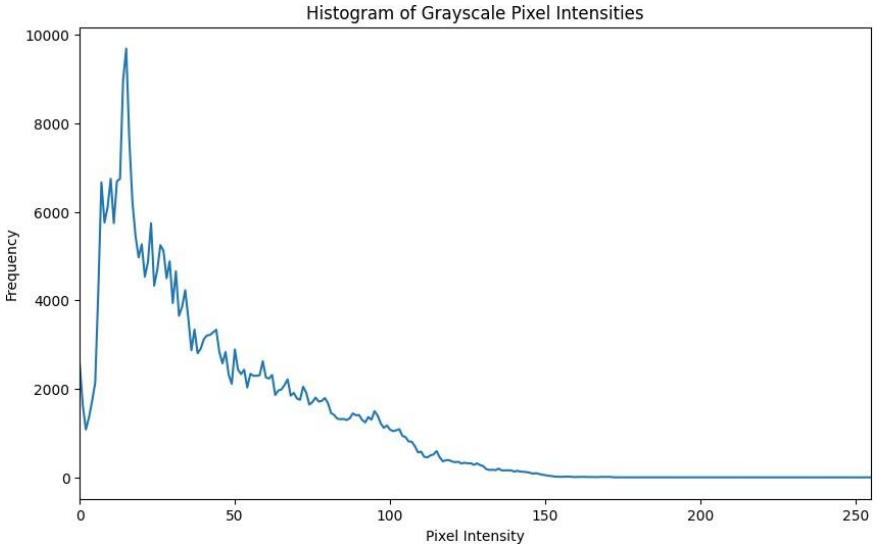
A vignette mask is applied on the saliency map of the image to highlight the salient regions of the image and suppress the non-salient regions of the image. We have currently assumed that the subject of the image is usually present at the center of the image. This step is necessary to properly apply thresholding to the image and then generate the mask.

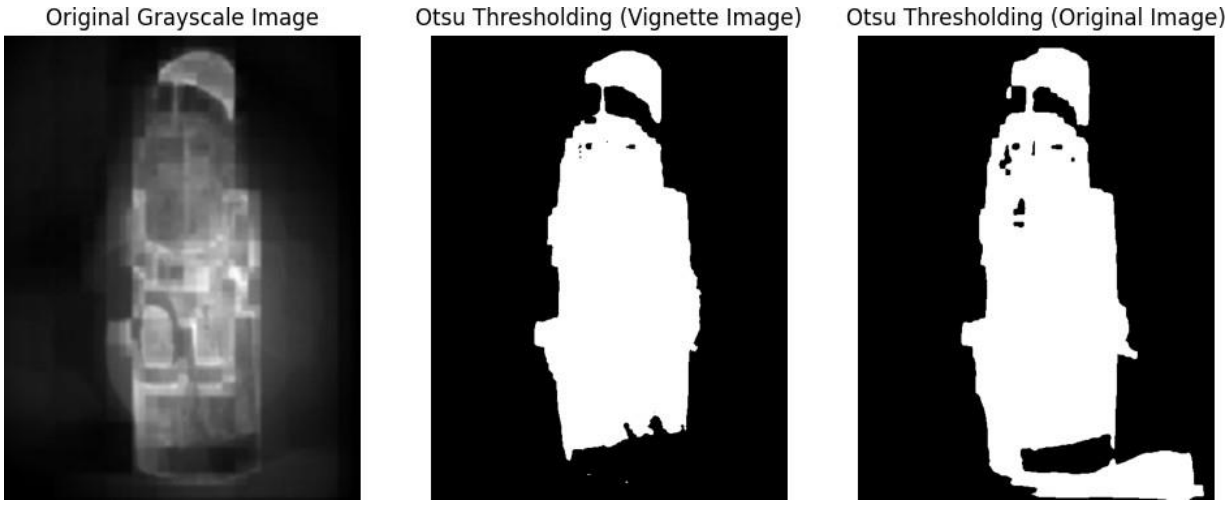




Otsu Thresholding

After applying the vignette mask, we have now gotten an image where the salient part of the image is mostly highlighted in whitish pixels and the non salient part of the image in blackish pixels. To generate the rough mask of the salient region, we can use Otsu thresholding. In Otsu thresholding we threshold at the value which maximizes the between class variance thus giving us the best possible thresholding of the image.





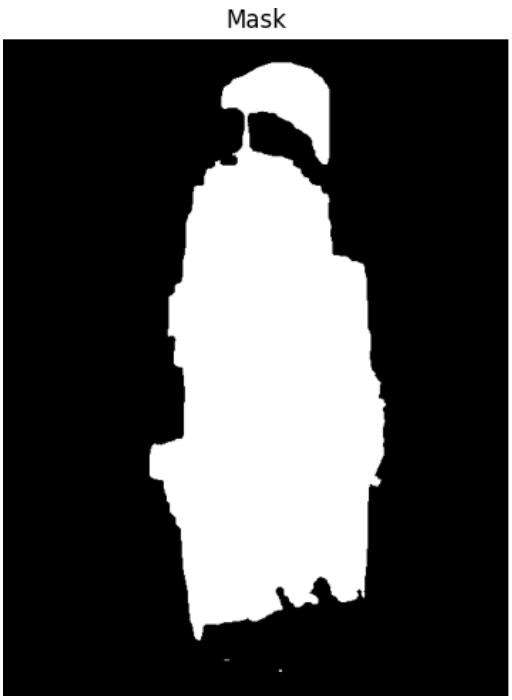
The reason why we have applied otsu thresholding onto the vignette applied saliency map is that if applied on the non-vignette applied vignette map, otsu thresholding will take into account various regions of the saliency map that are not even salient in the first place. Vignette helps suppress these erroneous regions of the saliency map.

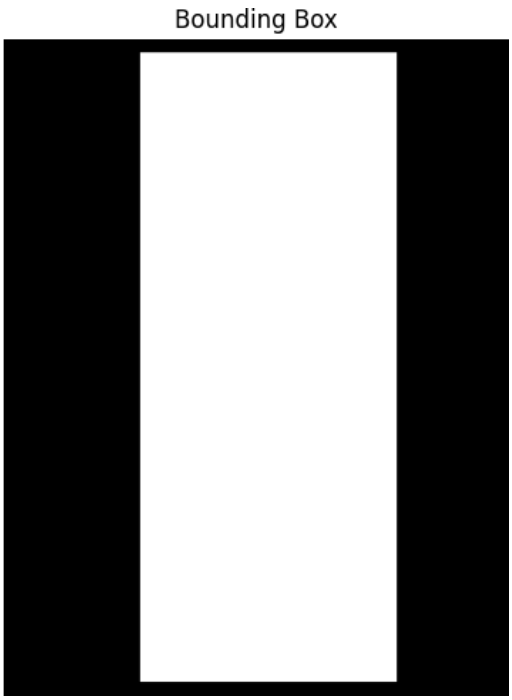
Hole Filling

In this step we try to fill the small holes within the mask that was generated in the previous step. Here, we have used openCV's flood fill algorithm to fill the holes.

Bounding Box Generation

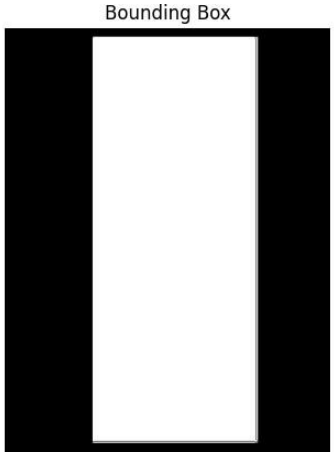
Next, we generate a bounding box over the mask. The main idea of using a bounding box instead of the mask generated in the further pipeline is because the mask generated in the previous step is not perfect and may have imperfections in certain places which may cause problems later on in the pipeline. However, what the mask is good at is to give us a very concrete idea of where the salient part of the image is located. Hence, creating a bounding box would in most cases encompass only the region containing the salient object, thus giving better results down the pipeline.



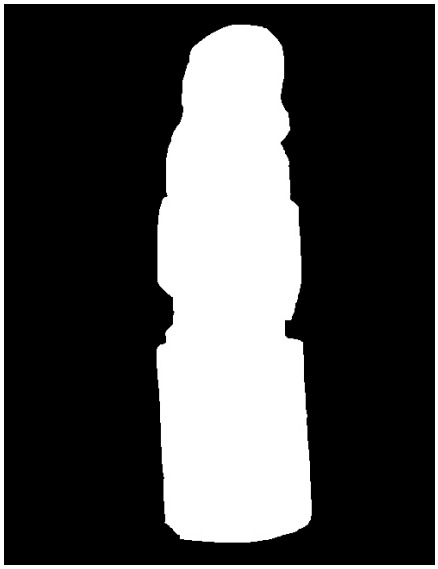


GrabCut

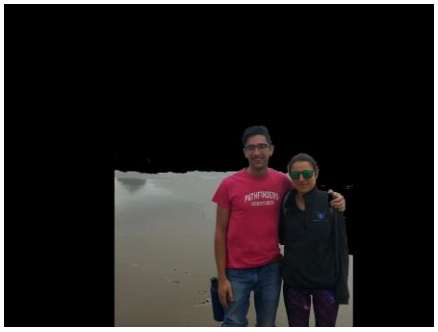
In GrabCut, we aim to separate the foreground and background from each other with the help of graph cuts. Due to the mask (bounding box) provided to the algorithm, we have a rough idea of the foreground and background pixels on which we fit GMMs. Then, in the graph cuts algorithm, we construct a graph where each pixel acts as a vertex of the graph and each vertex is connected to its immediate surrounding neighbor pixels (8-way connection) and each edge's weight is computed and assigned. Then using min-cuts we segment the graph into foreground and background portions and then assign the new alpha values, ie: refine the segmentation into accurate foreground(white) and background(black) portions. This process is continued until flow (min-cut value) converges or a specified maximum number of iterations is reached.



Results and Observations











There were many key observations during the process of making this pipeline:

The pipeline works best when the salient region of the image is very clearly distinct from the background of the image. If the salient region has similar patterns as compared to the rest of the image or if its color is similar to the other parts, the effectiveness of the pipeline starts decreasing but is countered to some extent using techniques such as applying the vignette mask and thresholding.

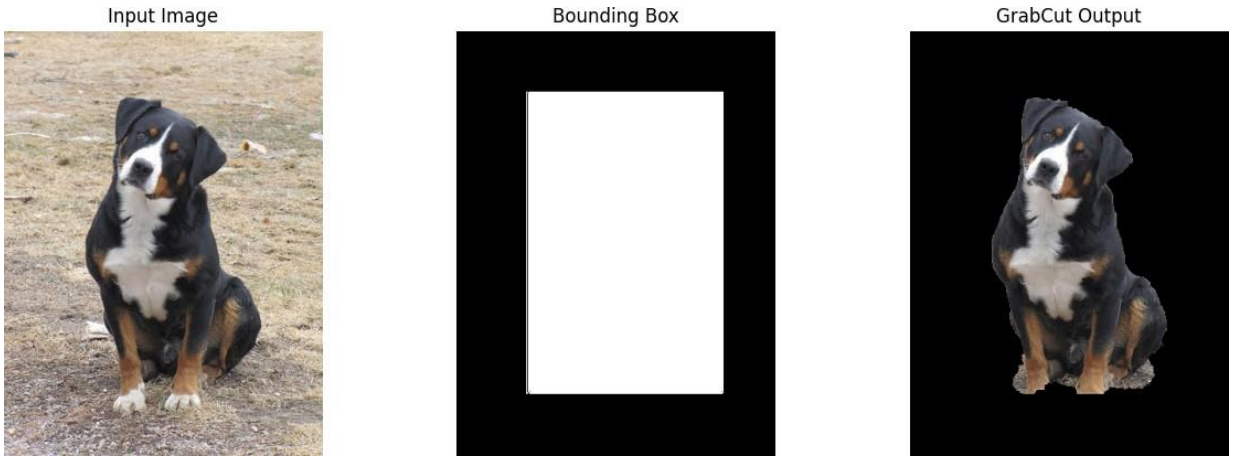
There may be certain regions in the salient part of the image itself that may not have such a distinct pattern or color. Due to this these particular regions within the salient regions end up getting classified as non salient thus affecting the mask generation later in the pipeline.

The pipeline may detect in certain scenarios more than 1 localized region as the salient region. Due to this at the end of the pipeline we may end up getting other not so salient parts of the image as well along with the main salient region.

Uses

As mentioned in the possible use cases, image biasness is a problem that is faced in training ML/DL models. It may be possible that for an image the model is learning not only the foreground of the image but is also being influenced by the background of the image as well in cases where the background of the image does not hold any importance. We applied this idea by running the VGG16 model on the following image

Input Image



Predicted class

Predicted class: **Appenzeller**
Probability: **0.6096334**

Predicted class: **EntleBucher**
Probability: **0.4337105**

As we see that the model has given different output classes for the dog present in the image even though it should have been invariant to the background of the image and should have given roughly the same result.

References

[What Makes a Patch Distinct?]
["GrabCut": interactive foreground extraction using iterated graph cuts]
[GitHub - JakeOliger/What-Makes-A-Patch-Distinct]
[GrabCut/README.md at master - MoetaYuko]