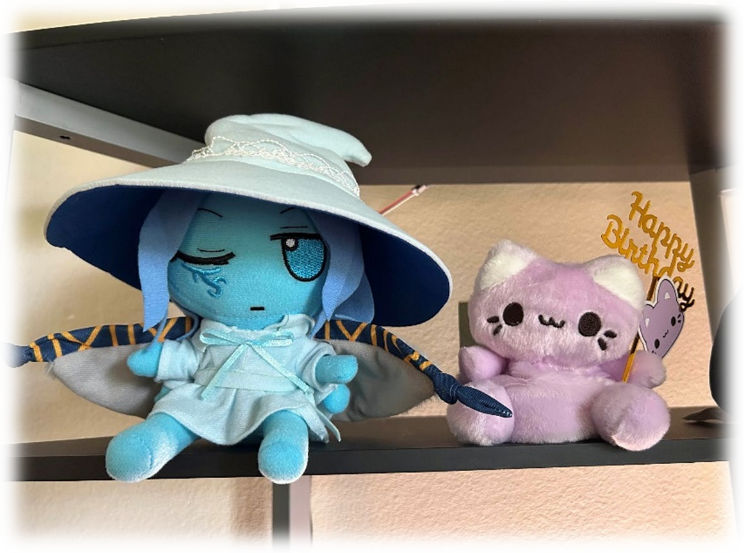
Part 1: Using the Basic Classifier and Implementing Grad-CAM

- Choose an image you'd like to classify. This can be a personal photo or one from a free image repository (e.g., Unsplash).



- If you do not have previous experience with Python, comment on whether the AI's explanations make sense to you. If you do have Python experience, comment on whether the AI's explanation agrees with your interpretation.

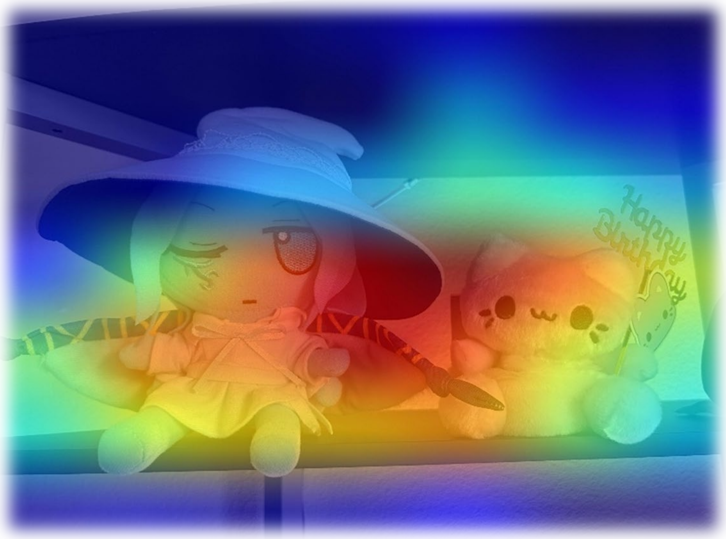
I don't have python experience; however, I can recognize that the dependencies were imported and configured. Then have the image be processed in the way that it can be processed, similarly to the way it would for training (normalization) and decoding, as well as importing a model with existing data-sets and pre-trained weights. My AI explains that the image gets resized to fit the requirements of the models and gets converted into a NumPy array (height x width x channels). Then the output of the model prints the top 3 labels with a confidence score for each. In my experience I assume the model converts to vector values and compares those values to their existing data-sets and weights, then the top closest matches get printed.

These were the **top-3** prediction results of **the default setup and image**:

1: tiger_cat (0.35), 2: tabby (0.34), 3: Egyptian_cat (0.07)

- Record the **top-3** predictions and their confidence scores. **(With my image)**
  1: teddy (0.73)
  2: bonnet (0.12)
  3: seat_belt (0.03)

- Run the updated classifier with Grad-CAM on your image.

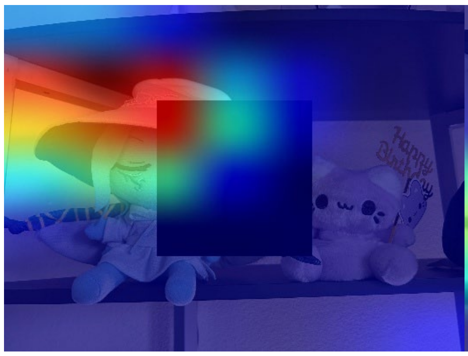


- Identify which parts of the image the classifier focuses on most heavily.

I was pleasantly surprised to see the top prediction as “teddy” even though the correct terminology would be plushy, but technically still correct! The heat map shows that the attention of the model was focused on the middle in between the two plushies, probably because of all the different fabric textures and colors found in that portion of the image, each color represents the model’s decision making, showing colors based on the weights of each feature map channel.

Part 2: Experimenting with Image Occlusion

Black Box of Pixels



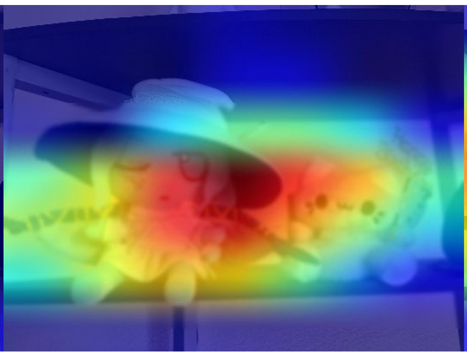
Top-3 Predictions:

1: mortarboard (0.26)

2: teddy (0.17)

3: carton (0.13)

Blur



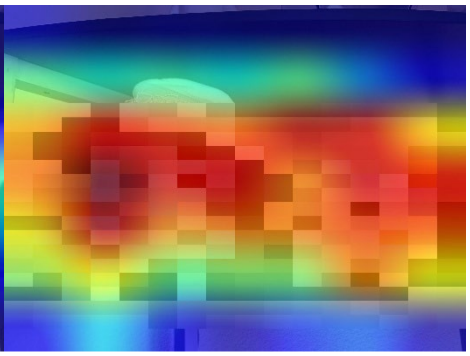
Top-3 Predictions:

1: teddy (0.14)

2: pinwheel (0.11)

3: carton (0.09)

Pixelate



Top-3 Predictions:

1: container_ship (0.07)

2: crossword_puzzle (0.03)

3: washbasin (0.02)

- Compare the classifier's performance on the original image vs. the occluded versions.

The classifier struggled very hard with the occluded versions, so much so that in the pixelated one the correct prediction isn't even listed, you can see that the focus areas on the pixelated ones are very large, almost as if it wasn't able to find something recognizable, the confidence scores are particularly low for all 3 of the predictions in that case.

The Blurred image has a lower confidence score than the non-occluded first iteration with the correct one only scoring 14%. However, the heatmap is surprisingly similar and the top prediction is accurate.

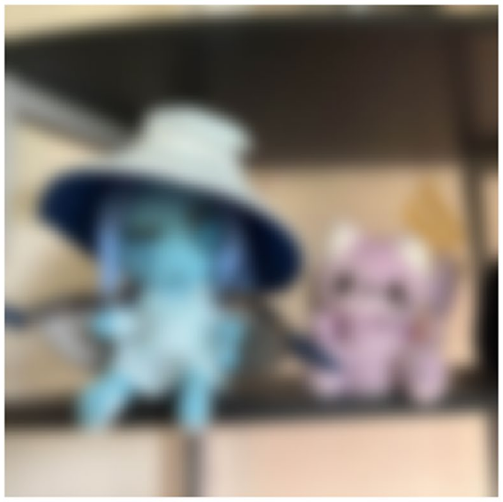
Lastly the image occluded with a black square show that the attention of the classifier focused on the top left of the image, the correct prediction, score of 17%, is in second place, and it's worth noting that the confidence scores are higher than the other occluded classified images. In the un-occluded classified image, the confidence scores for the top prediction is 73%, which is the highest out of all results, as expected.

Part 3: Creating and Experimenting with Image Filters

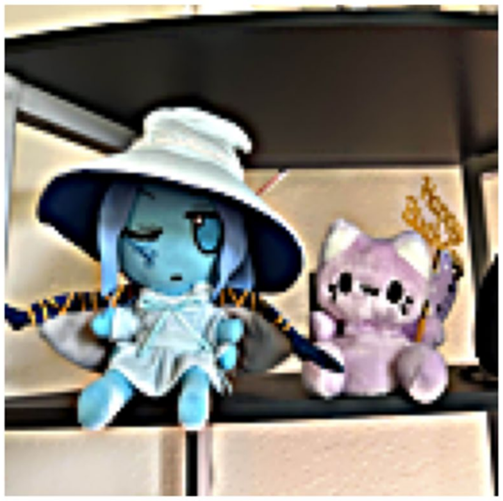
- If you do not have previous experience with Python, comment on whether the AI's explanations make sense to you. If you do have Python experience, comment on whether the AI's explanation agrees with your interpretation.

A class for manipulating images is used for this script and the image path is defined and resized, then the filter gets applied from an existing library of functions, theres some error handling involved as usual, then the output is an image file with the filter name in the same directory as the original image.

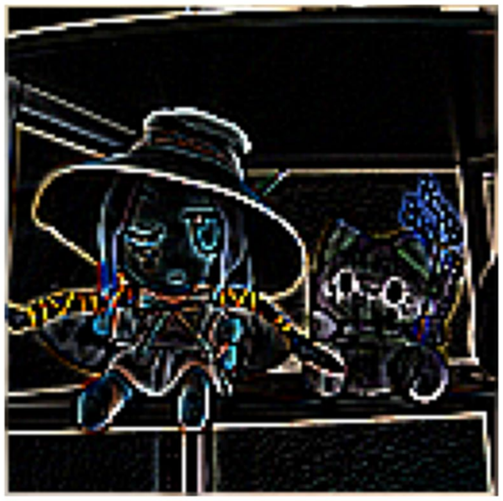
- Start with the provided basic_filter.py program, which applies a simple blur to an image.



- Modify the basic_filter.py program to implement the three new filters suggested by the AI.



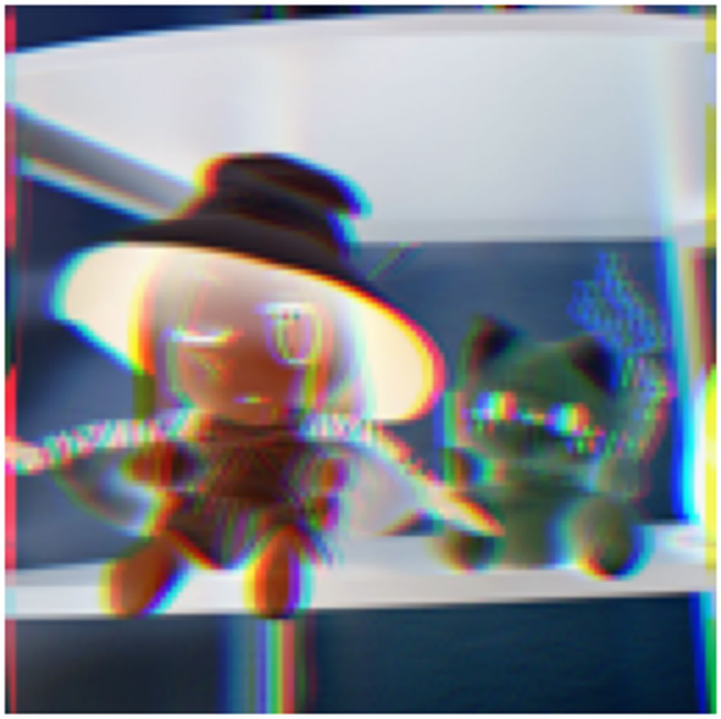
Sharpen Filter



Edge Detection Filter

Emboss Filter

- Use creative prompts to direct the AI to develop an artistic filter



Custom Chromatic Aberration + Negative Filter