**Name (netid):** Johnathon Small (jssmall3)
**CS 445 - Project 1: Hybrid Images**

Complete the claimed points and sections below.

| **Total Points Claimed** | **[130] / 130** |
|---|---|

1. Hybrid image main result
    a. Main result and description [45] / 45
    b. FFT images of main result [15] / 15
2. Hybrid images: two additional results [10] / 10
3. Image enhancement tasks (3rd is B&W)
    a. Contrast enhancement [10] / 10
    b. Color enhancement [10] / 10
    c. Color shift [10] / 10
4. Quality of results / report [10] / 10
5. Color Hybrid Image w/ explanation (B&W) [5] / 5
6. Gaussian / Laplacian Pyramids (B&W) [15] / 15

## 1. Hybrid image main result

I constructed a hybrid image from two pet photos provided by my brother (figure 1a). They are Tex the cat (image 1) and Malachai the dog (image 2). Illustrations of the Fourier transform of each image are provided (figure 1b).

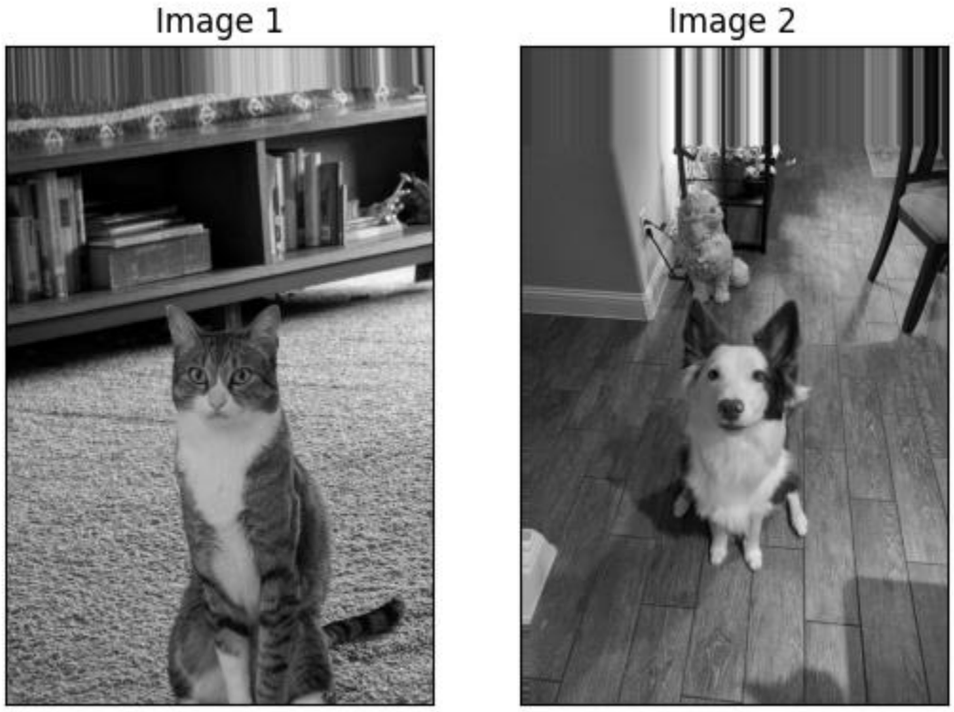



*Figure 1a: Tex & Malachai, unfiltered*

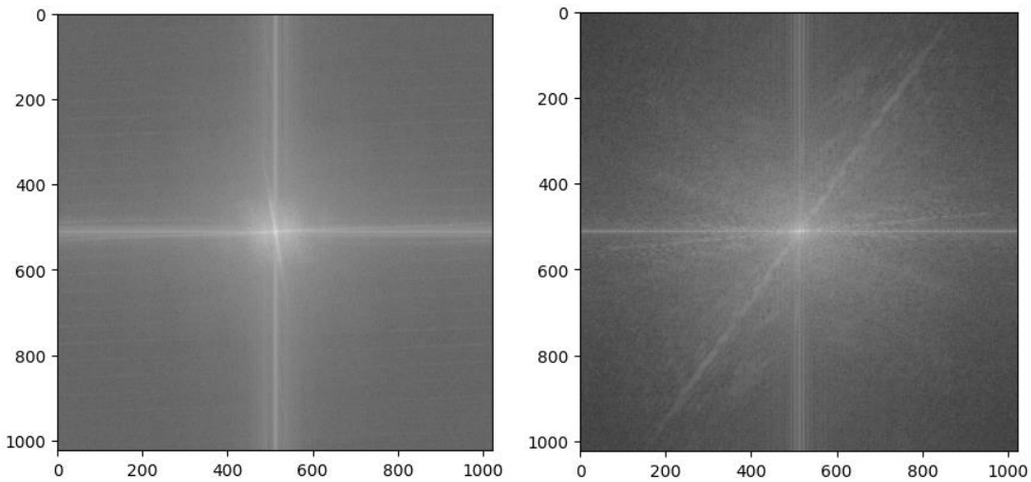



*Figure 1b: FFT of Tex & Malachai, respectively*

A low-pass filter was applied to the image of Tex to isolate the low frequency signals within the image (figure 2). The low pass filter was constructed from a Gaussian kernel of size KxK and a sigma value of 20, where K = 6 * sigma + 1. This K value (filter size) was chosen to minimize clipping of gaussian at its edges via the provided heuristic: kernel half-width should be greater than or equal to three times sigma. This ensures the edge values are close to zero.

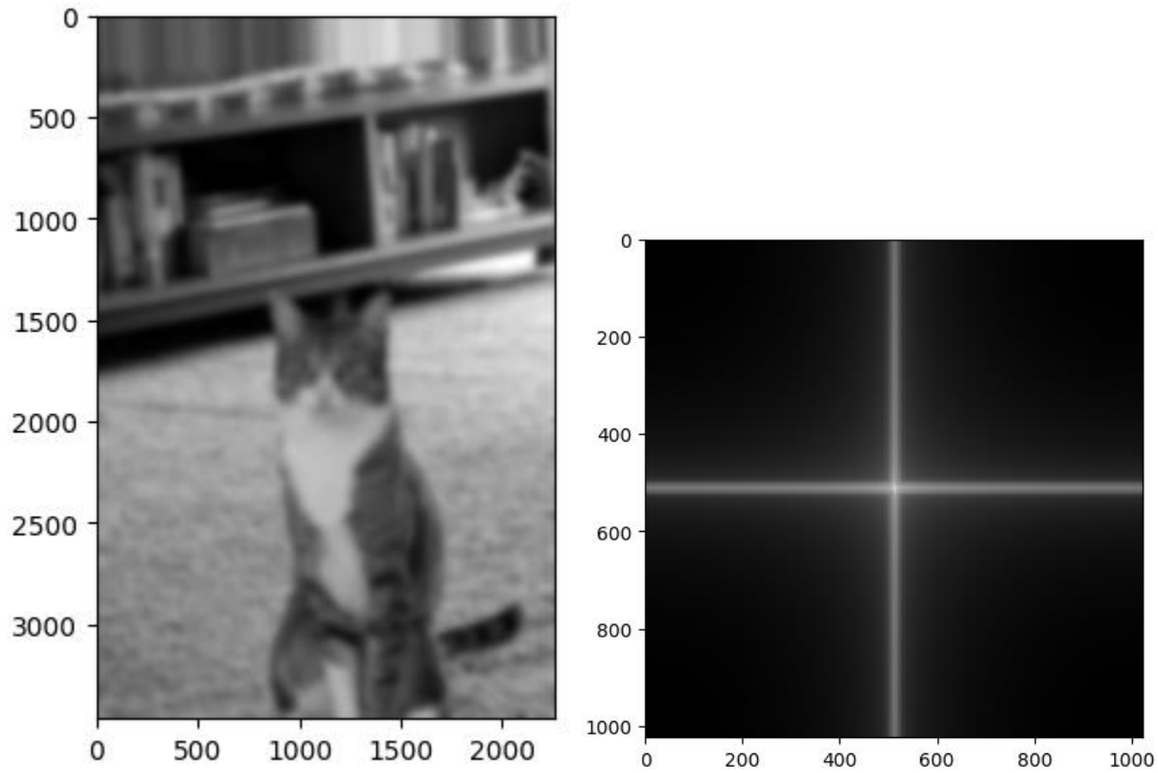



*Figure 2: left: Tex with a Gaussian blur (sigma=20), right: FFT of the blurred image*

A high-pass filter was applied to the image of Malachai to isolate the high-frequency signals from the image (figure 3). The high-pass filter was constructed from a Gaussian kernel of size KxK with a sigma value of 25 subtracted from a unit impulse filter of size KxK, where K was determined in the manner described above.

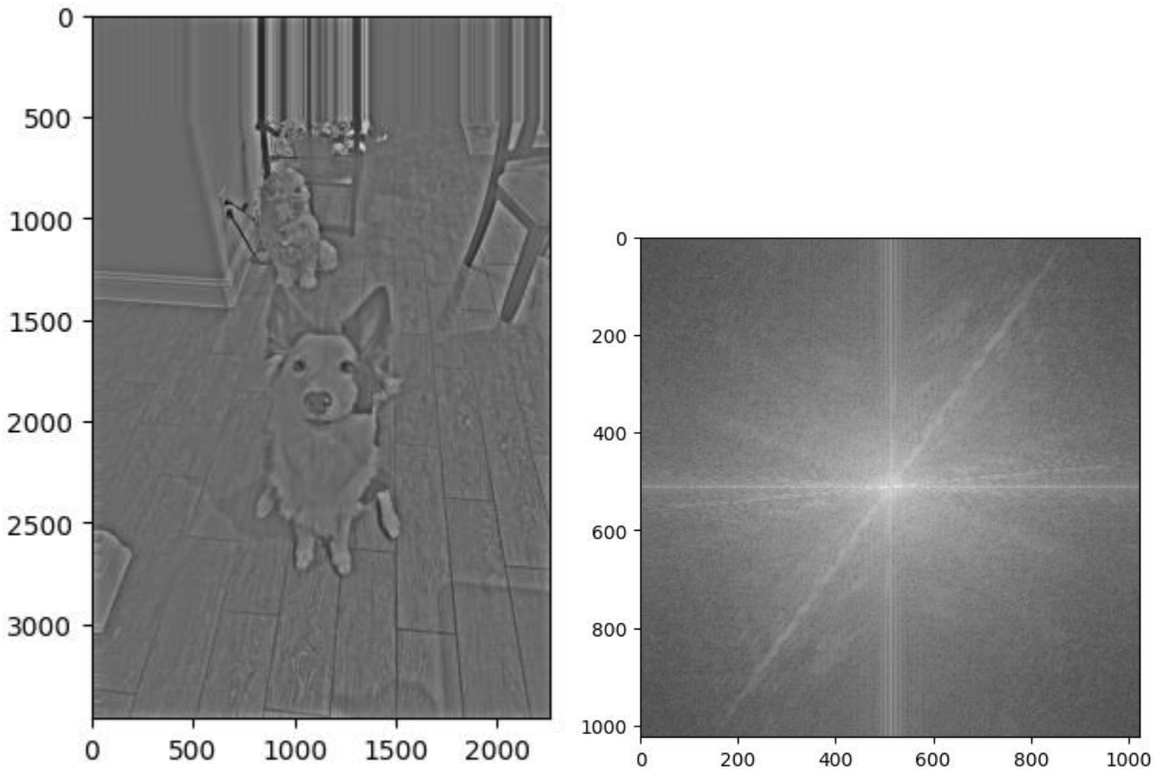



*Figure 3: left: Malachai with a Gaussian subtracted from the image (sigma=25); right: FFT of the image*

The final hybrid image (Figure 4) was constructed simply by adding the low & high frequency images together. From afar, we receive limited light information from the screen, so the low frequency information dominates and we are only able to discern Tex the cat. As we get closer, our eyes can perceive more information and we pick up on the finer, high frequency detail of the image, which causes Malachai the dog to be more visible.

Sigma values were chosen (low-pass=20, high-pass=25) to optimize for the distinction between images at different viewing angles. As Oliva et al. describe in their paper, the cut-off frequencies should be separate by some margin to reduce filter overlap, which creates ambiguity in the image. The high-frequency sigma value is held higher than the low-frequency sigma value for this reason (distinction between frequency bands).

Higher sigma values for the low-frequency image cause greater blurring and a greater loss of information. If this value is too high (>80 for these images), Tex the cat becomes difficult to discern at any distance as there is not enough signal remaining for the brain to construct a cat.

Conversely, higher sigma values for the high-frequency image reduce how much of the low-frequency components are removed from the image. If this value is too high compared to the low-pass filter value, the images compete during low-frequency viewing while the high-frequency image dominates during nearby viewing.

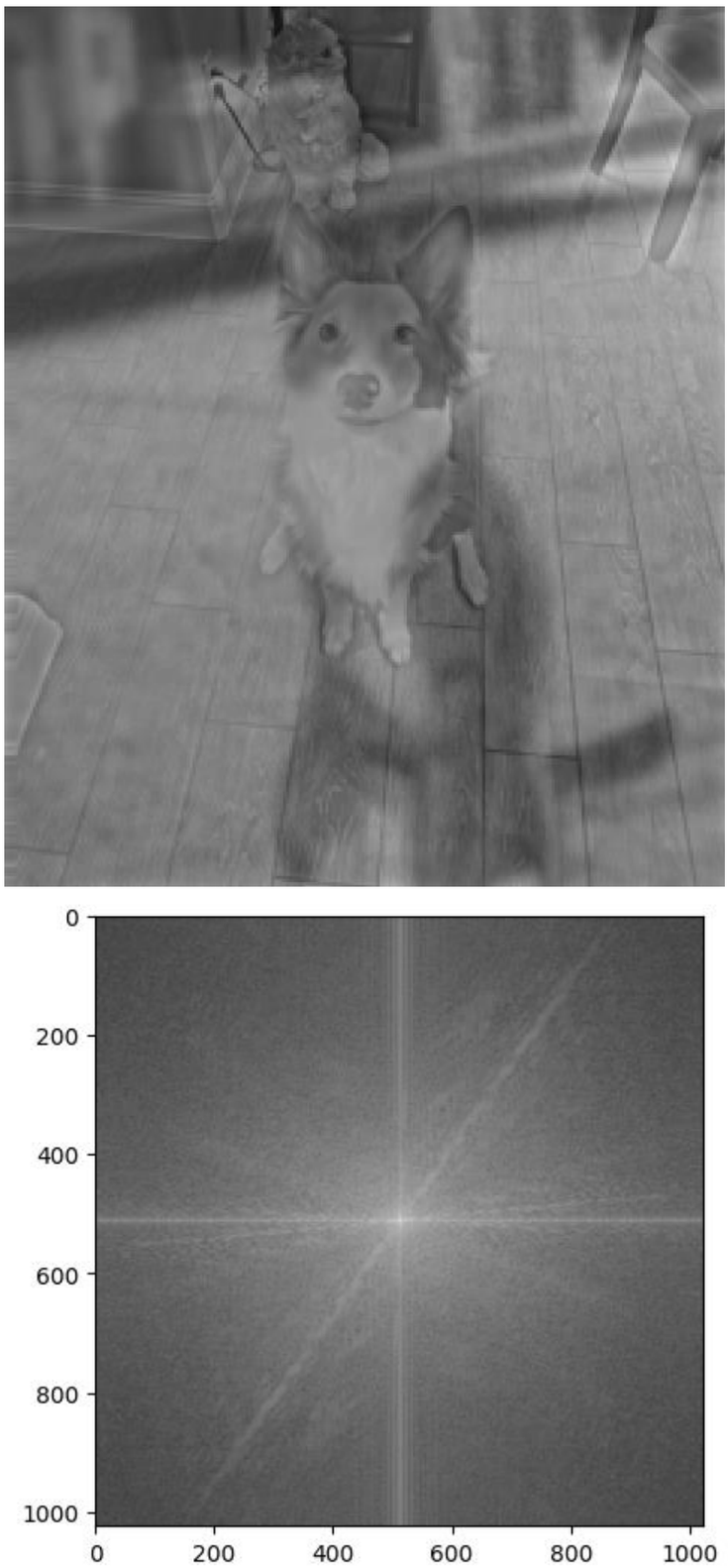


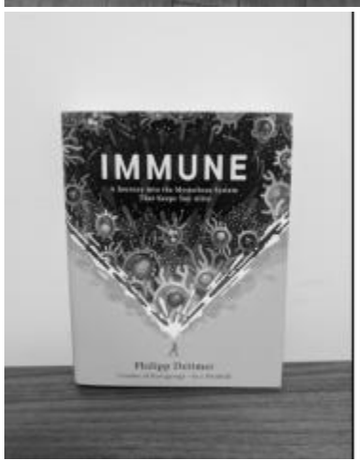

*Figure 4: Above: The hybrid image, cropped; below: FFT of the hybrid image*

These images were constructed with a modified version of the provided `align_images` utility function. The original implementation translated the images and padded the empty space with `np.zeros` arrays, which render as black. The resulting hybrid images tended to have clear edge lines as the translation from the lightly-colored image to the black fill area caused a strong high-frequency response. To address this, I experimented with a few strategies to fill the void space, such as reflected tiling (which had the downside of ghost images, like a copy of Nutmeg the cat peaking down from the ceiling). Ultimately, the following implementation, which simply repeats the edge, had the best results since it blends in with the original image.

Handling of the fill-space is of particular importance on the image contributing the high-frequency signals since edges cause a strong response to the high-pass filter, making them visible in the final image. An ideal implementation might apply a gaussian to the fill pixels such that they only contain low-frequency information, making them effectively invisible in the final image.

```
    # translate first so that center of ref points is center of image
    def translate(image, translation: tuple[int, int]):
        x, y = translation
        y_pad_term = (y, 0) if y > 0 else (0, -y)
        x_pad_term = (x, 0) if x > 0 else (0, -x)

        # Tuples are height (start, end); width (start, end); color (start, end)
        return np.pad(image, (y_pad_term, x_pad_term, (0, 0)), mode="edge")

    tx = np.around((w1 / 2 - center_im1[0]) * 2).astype(int)
    ty = np.round((h1 / 2 - center_im1[1]) * 2).astype(int)

    img1 = translate(img1, (tx, ty))

    tx = np.around((w2 / 2 - center_im2[0]) * 2).astype(int)
    ty = np.round((h2 / 2 - center_im2[1]) * 2).astype(int)
```

**2. Hybrid image additional results**




**3. Image enhancement tasks (2 required, 3 for B&W)**

Include

- For at least two out of three enhancement tasks (each is worth 10 points), display original image, modified image, and explanation of how the image was modified

# Color Shift

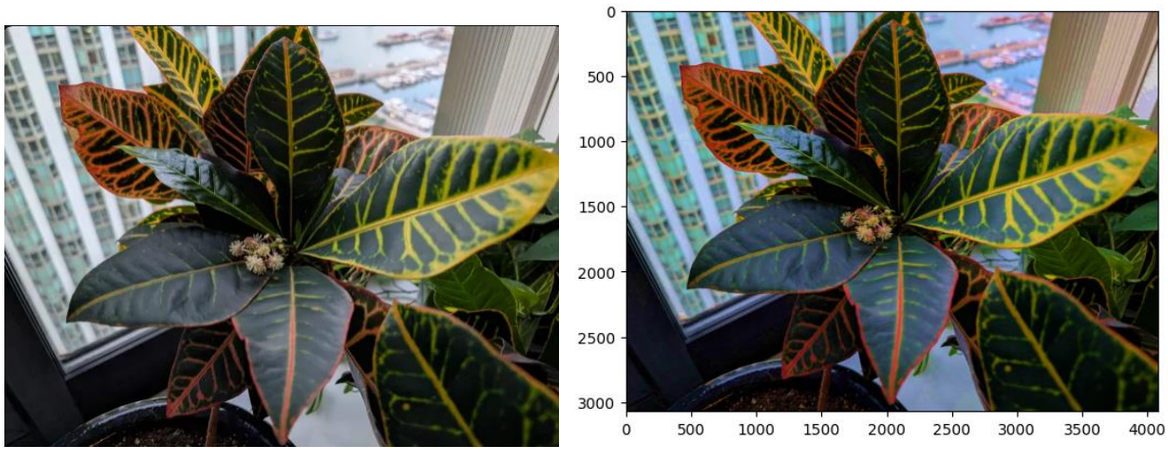



To perform the color shift, I first converted the image from the CIERGB to CIELAB. The use of a color space with chroma & luminance allowed me to modify the chroma channels without affecting luminance. Next, I wanted to boost or decrease the appropriate chroma channels without causing clipping or overflows. The gamma function used for scaling luminance is a perfect fit because it outputs numbers [0, 1] for all inputs [0, 1]. Opencv2's implementation of CIELAB stores each channel as a uint8 between 0 and 255, so I scaled the channels, applied the gamma function, then re-scaled and cast them back to uint8.

For the "more red" image, I scaled the channels with factors (1, .8, 1) to increase the strength of the red/green channel while leaving the other channels unchanged. For the "less yellow" image, I scaled the channels with factors (1, 1, 1.25) to decrease the strength of the yellow/blue channel while, again, leaving the others unchanged.

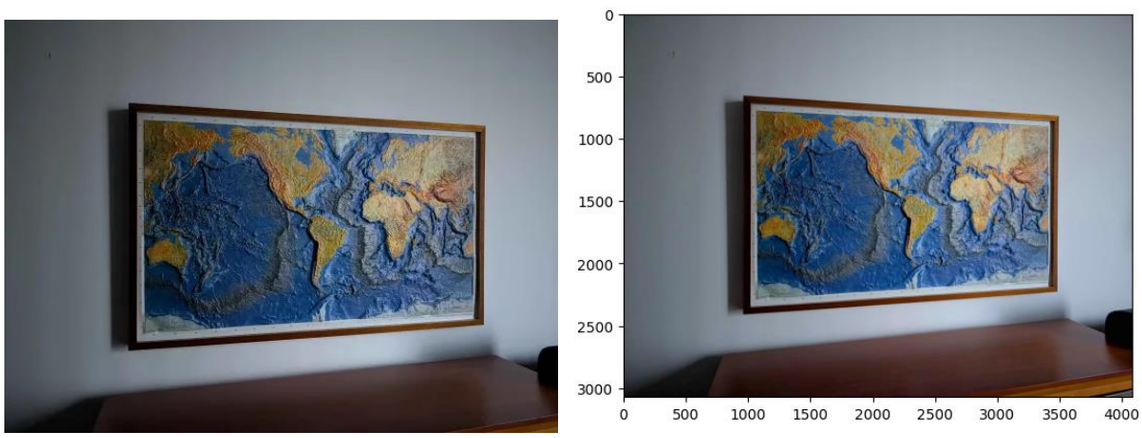
While the algorithm above was sufficient for the other enhancement tasks, I upgraded it to perform masking in order to reduce the effect on "complementary" colors. The user can instead pass an argument to indicate whether the gamma effect should be applied to the top or bottom half of the channel. In this way, we selectively affect only the chroma values desired.

## Color Enhancement




Color enhancement can be performed in the same way as color shifting as long as we pick the right color space. By instead moving to CIEHSV and applying the gamma shift (1, .5, 1) to boost the saturation channel, we can enhance the colors in the image, as shown above.

## Contrast Enhancement




Again, in the same way we can move an image to the CIEHSV color space and shift the luminance channel to enhance contrast. In this example, we apply the shift (1, 1, .75) to brighten the image and reveal more detail in the mountains under the Pacific. As an aside, it turns out that taking a photo with bad contrast was more difficult than expected due to the auto-correction and night-sight features of modern smartphone cameras.

**4. Quality of results and report**

Nothing extra to include.

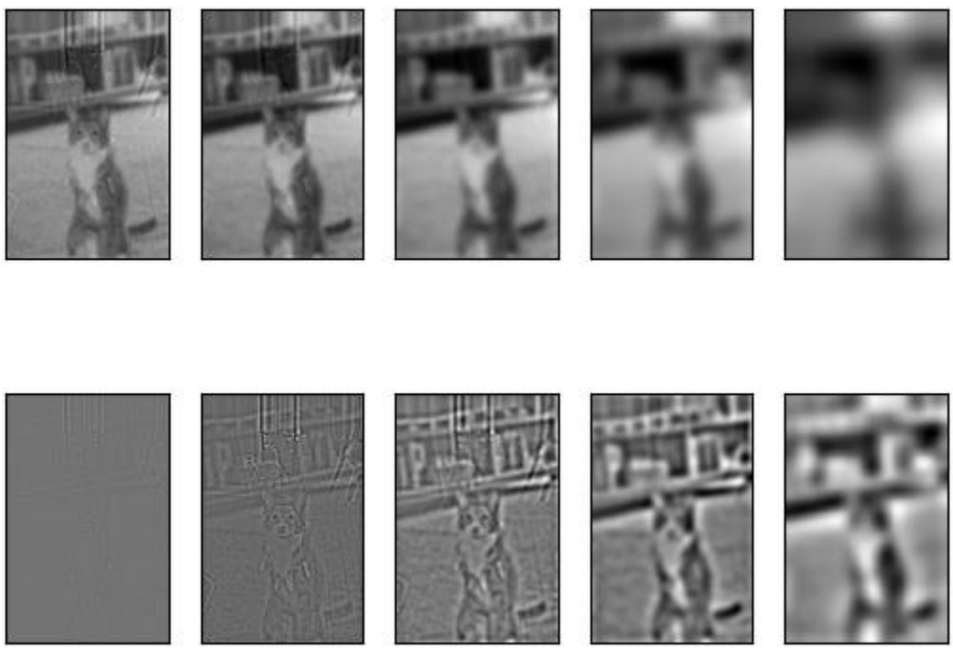
**5. Color hybrid result (B&W)**

*Left: colored low-frequency; middle: colored high-frequency; right: both colored*

In this experiment, I passed color channels for one or both of the component images prior to combination. To ensure images had the same amount of channels, I converted the grayscale images to gray, then back to RGB prior to running the hybridization algorithm.

Coloring in the high-frequency image (middle) is very muted. There is little high-frequency color data available in the source image, leading to an effectively grayscale result. Color is very visible in the low-frequency image, but care needs to be taken such that it does not overpower the high-frequency image. In this example, because of the poor alignment of the animals' bodies, the colored low-frequency image dominates perception. Color would best be applied when both layered images are very similar. The image with both components colored is not meaningfully different than the colored low-frequency image. It follows that it is best to use color in the low-frequency image only.

**6. Gaussian and Laplacian Pyramids (B&W)**



The top row shows the Gaussian pyramid using sigma=10. The bottom row is the Laplacian pyramid, which is simply the respective Gaussian image subtracted from the input image. In this way, any given column added together will yield the input image at that step. Each step from left to right is downsampled by a factor of 2 in each direction.

**Acknowledgments / Attribution**
https://docs.opencv.org/3.4/d4/d1f/tutorial_pyramids.html
https://docs.opencv.org/4.12.0/d4/d86/group__imgproc__filter.html
https://stackoverflow.com/questions/75334695/rotate-an-image-in-python-and-fill-the-cropped-area-with-image
https://docs.opencv.org/3.4/d4/d61/tutorial_warp_affine.html
https://stanford.edu/class/ee367/reading/OlivaTorralb_Hybrid_Siggraph06.pdf
Pet pictures courtesy of Matt Small
All other photos mine