In this lab, you will be implementing some simple image processing commands in CUDA. Begin by following the directions to set up CUDA on Google Colab.

`https://github.com/TeachingUndergradsCHC/modules/blob/master/Architecture/gpu_memory_hierarchy/colab.md`

As you enter each command, be sure to remove leading spaces.

While you're doing that or once it's done, you'll need to upload the given code. These these files from the repo and upload them to Colab (select the folder icon in the left margin and then the file with an upward arrow): `ppmFile.c`, `ppmFile.h`, `main.c`, and our sample image `640x426.ppm`. (You can also use another .ppm file.)

Once colab is set up and you've run the same "Hello World" program at the bottom of the setup page, paste in the contents of `noRed.cu` into another code window and run it. This file is the code for a kernel that will remove all the red from our sample image. In includes non-conventional includes of .c files near the top and at the bottom. These paste in the code for dealing with images and the program's main (which calls the kernel). We're #including these files to cut down on the amount of code that has to be pasted into the code box.

When you run the code, a new file `out.ppm` will be created. (You may have to refresh the file panel to the left by hitting the button with circular arrow to see it.) Download this file and observe that it looks basically the same as our original image, but the colors are slightly shifted. The image is stored as an array of RGB values, each color in a separate *channel* of the pixel. Look through the kernel to see how it works.

As a first task, let's change this kernel to convert an image into grayscale (black and white). To do this, each pixel should be set to the average of its input red, green, and blue values. In the kernel, this corresponds to adding up the values for each channel, dividing by three, and then setting each channel of the output image to the result. Do this and make sure it's working before proceeding.

Creating the blur effect is the final challenge. This is also an average, but a different kind. Each channel is blurred separately— each gets the average of that channel's value for nearby pixels. "Nearby" in this case means all pixels for which each coordinate is within a specified radius of the pixel being set (i.e. its the L_∞ distance $\leq$ the radius).