```cpp
#include <iostream>

#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>

using namespace cv;

//int* from original assignment replaced with pointer to the kernel instead, because that makes more sense
void convolve5 (Mat* inputImg, Mat* outImg, int (*kernel5)[5][5]) {
  //I assume the mat is CV_8UC1 since I want to process BGR channels individually
  Size s = inputImg -> size(); // get size of image
  const uint8_t kernel_size = 5; // todo: replace with sizeof
  uint8_t out[s.height][s.width]; // create output array
  int x ,y, h, w, i, j, sum; // declare variables

  std::cout << "Input type was : " << inputImg->type() << std::endl; //debug

  for (h=0; h<s.height; h++) { // for each row
    for(w=0; w<s.width; w++){ // for each column
      sum = 0; // reset sum
      for(i=0 ;i<kernel_size; i++ ){ // for each kernel row
        for(j=0; j<kernel_size; j++){ // for each kernel column
          y=h-i+1; x=w-j+1; // calculate the position of the pixel in the image
          // if the pixel is outside the image, set it to the border
          if(y<0) y=0;
          if(y>s.height-2) y=s.height-2;
          if(x<0) x=0;
          if(x>s.width-2) x=s.width-2;
          sum += ((*kernel5)[i][j]) * inputImg->at<uint8_t>(y,x); // add the product of the kernel and the pixel to the sum
        }
      }
      sum /= kernel_size*kernel_size; //divide the result of the pixel by 5^2
      if(sum<0) sum=0; // if the result is negative, set it to 0
      if(sum>255) sum=255; // if the result is greater than 255, set it to 255
      std::memcpy(outImg->data, out, s.height*s.width*sizeof(uint8_t)); // copy the result to the output array
      out[h][w]=(uint8_t)sum; // set the result to the output array
    }
  }
};


int main() {
  // Read the image (in BGR)
    Mat img = imread("ford_gt_final2.png", IMREAD_COLOR);
    if(img.empty())
    {
        std::cout << "Could not read the image: " << std::endl;
        std::cout << "Could not read the image: " << std::endl;
        return 1;
    }
    Size imgsize = img.size();

    // Split the image into 3 new images for blue, green and red.
    std::cout << "Splitting channels: " << std::endl;
  Mat bands[3];
  split(img, bands);

  //define our 5x5 kernel
  int kernel[5][5] = {{1,1,1,1,1},
                      {1,1,1,1,1},
                      {1,1,1,1,1},
                      {1,1,1,1,1},
                        {1,1,1,1,1}};

    /* example kernel with bottom sobel
  int kernel[5][5] = {{-1,-1,-1,-1,-1},
                      {-1,-1,-2,-1,-1},
                      {0,0,0,0,0},
                      {1,1,2,1,1},
                        {1,1,1,1,1}};
    */
    /* example kernel with identity
    int kernel[5][5] = {{0,0,0,0,0},
                      {0,0,0,0,0},
                      {0,0,1,0,0},
                      {0,0,0,0,0}
                        {0,0,0,0,0}};
    */

  Mat bandsConvoluted[3];
  bandsConvoluted[0] = Mat(imgsize.height, imgsize.width, CV_8U);
  bandsConvoluted[1] = Mat(imgsize.height, imgsize.width, CV_8U);
  bandsConvoluted[2] = Mat(imgsize.height, imgsize.width, CV_8U);

  convolve5(&bands[0],&bandsConvoluted[0],&kernel);
  convolve5(&bands[1],&bandsConvoluted[1],&kernel);
  convolve5(&bands[2],&bandsConvoluted[2],&kernel);

  Mat merged;
  std::vector<Mat> channels =
{bandsConvoluted[0],bandsConvoluted[1],bandsConvoluted[2]};
  merge(channels, merged);

    // Display the image until q is pressed
    std::cout << "Displaying result: " << std::endl;
    imshow("Display window", bands[0]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bands[1]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bands[2]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bandsConvoluted[0]);
    waitKey(0); // Wait for a keystroke in the window
```

```cpp
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bandsConvoluted[1]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", bandsConvoluted[2]);
    waitKey(0); // Wait for a keystroke in the window
    imshow("Display window", merged);
    waitKey(0); // Wait for a keystroke in the window
    return 0;
}

```