Practical Exercises: Ficha

Iuliia Alekseenko, ajulyav@gmail.com

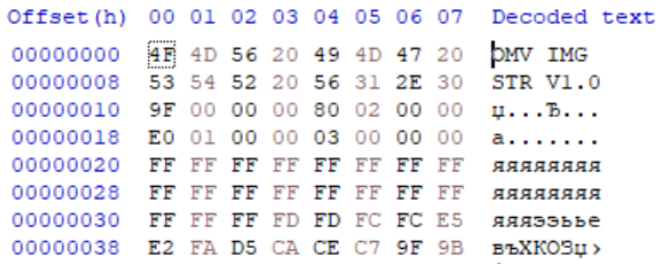


Fig. 1: The file representation

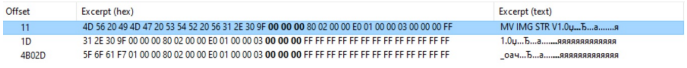


Fig. 2: Expected headers of the provided file

I. Image Parsing

In the course of the first part devoted to writing a program to read the raw images, there are several points that require more attention:

- a file can contain one or more files;
- information about the file header is unknown, for example, whether it is present or the file is a flat binary file (there is no header). It is also known that some formats of binary file have multiple headers, each denoting the start of a new block of data, for instance, animated GIFs, which are animations consisting of a series of images played one after another, have multiple headers. As the file may have more than one image, it is possible to assume that there are several headers in this case.

Thus, one potential solution is to quickly analyze the file in the binary file editor (Figure 1 and Figure 2), the file size (614 448 bytes) as well as the known information, for instance, RAW 8bit Bayer format and *sensor.VGA* which is 640x480px.

When taking into account the open source official information and provided details, there are four possible arrangements/patterns of how red, green and blue pixels can be placed on the camera sensor (Figure 3). And there is no universal standard for mapping colors (R, G, or B) to a pixel position.

If the resulting image still shows something like a Bayer pattern, it is very likely that the selected Bayer/Mosaic Pattern

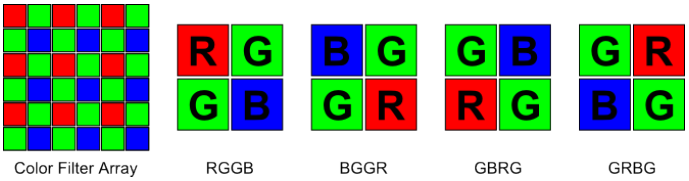
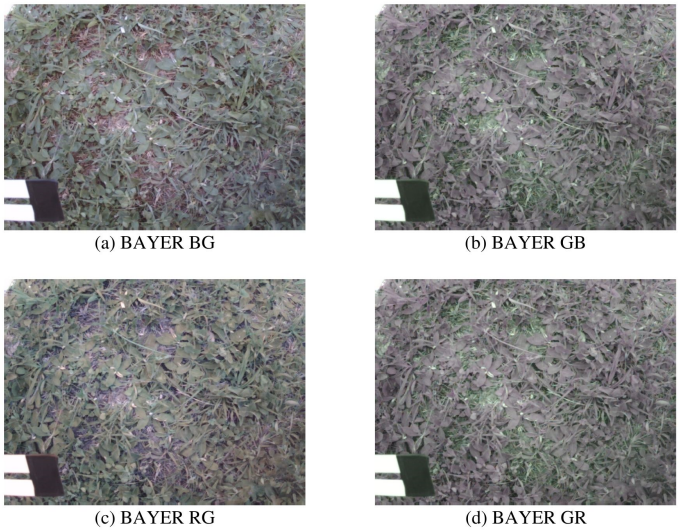


Fig. 3: Possible pixel arrangements



(a) BAYER BG (b) BAYER GB

(c) BAYER RG (d) BAYER GR

Fig. 4: Different pixel arrangement results

does not match the actual CFA pattern of the image.

To obtain a full-color image, various demosaicing algorithms can be used to interpolate a set of complete red, green, and blue values for each pixel. These algorithms make use of the surrounding pixels of the corresponding colors to estimate the values for a particular pixel.

Multiple interpolation methods exist for this task and the interpolation step has a significant effect on the output quality. For example, some of the common methods were analyzed for their advantages and disadvantages:

- Superpixel - this method is very fast. It has virtually no artifacts and is very well suited for demosaicing oversampled images.
- Bilinear - this method is very fast, but it tends to smooth sharp edges and generates color artifacts around edges. In general, other methods usually produce better results.
- VNG (Variable Number of Gradients) - this method preserves edges much better than bilinear interpolation. It also produces much less color artifacts and less chrominance noise compared to the earliest methods.

First, several pixel arrangements were tested in order to find the optimum one. When comparing the final results, extra knowledge about the lightning conditions (evening or day) would simplify the process as both BayerBG and BayerRG look quite promising. In the end, BayerBG was chosen, followed by a simple technique of color correction. Figure 4 represents the comparison.

Also, three ideas of demosaicing were tested: the simplest one - Nearest Neighbour, Bilinear interpolation and VNG (Variable Number of Gradients) as shown in Figure 5. In case

(a) Nearest Neighbor (b) Bilinear Interpolation (c) VNG

Fig. 5: Demosaicing methods

of Nearest Neighbour, rather large blocks of the image can be seen, which is predictable behavior. Bilinear interpolation and VNG (Variable Number of Gradients) were simply implemented with the OpenCv as it is an optimized and easy-to-go solution. While Bilinear interpolation provides smoother results as it is always expected, the Variable Number of Gradients method preserves edges better.

As improvements, other demosaicing algorithms can be proposed, for example, Aliasing Minimization and Zipper Elimination (AMaZE) or Adaptive Homogeneity-Directed (AHD). Comparing and finding the optimal method can reduce the number of possible artifacts.

A. How to run

To run the *image_parser* script, place the script in the folder with the *raw_images.bin* file. The final files will be written to a sub-folder called *RgbOut* (the folder is created automatically if the user has not created it).

The python script runs as standard. In the console, go to the required folder and run the line: **python image_parser.py**

II. Data Generation

In this part, only a few short comments will be given, as well as ideas for improvement.

- Since the task was set to generate from 0 to 4 objects in the background, then in the case of generating 0 objects (no object in the background), the annotation is still written to the JSON file with the bbox equals to (0, 0, 0, 0). Although visually examining the annotations of the original COCO dataset, no such instances were found. So, the other potential solution may be needed.
- Currently, annotations contain only ID and bbox according to the task, however, as part of the improvement, a complete annotation file can be generated.
- Currently, the code only supports transforming a foreground object in the form of its rotation from 0 to 359 degrees. However, other options for transformations can also be considered, for example, scale.
- Perhaps the best cropping of an object from its background can be implemented.

Bboxes have been checked for their correct data format, as shown in the Figure 6.

A. How to run

To run the *coco_generator* script, place the script in the folder with *background.png* and *object.png*. The final files will

(a) 3 Foreground Objects (b) 2 Foreground Objects

Fig. 6: Bboxes around foreground objects

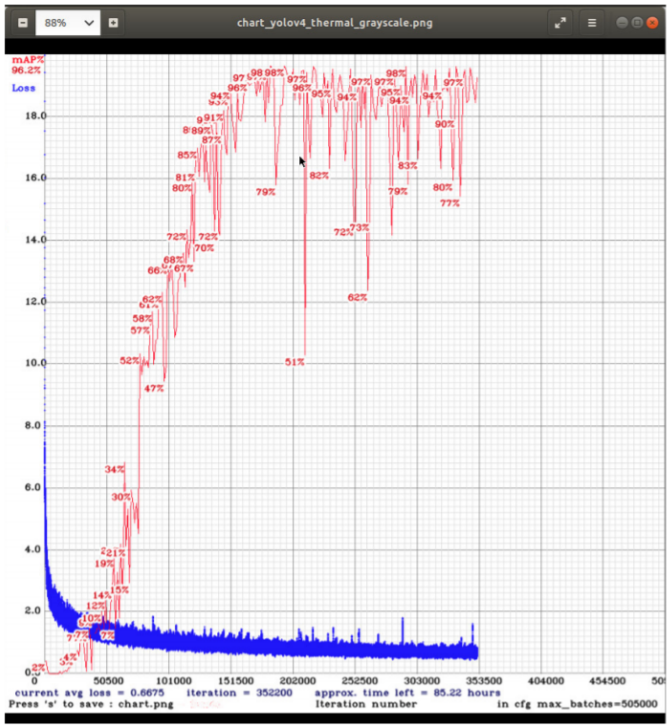


Fig. 7: Yolo v4 training

be written to a sub-folder called *GeneratedData* (the folder is created automatically if the user has not created it).

The python script runs as standard. In the console, go to the required folder and run the line: **python coco_generator.py**

III. Machine Learning

The plot shown in Figure 7 is given to analyze the behavior of the Yolo v4 model on the test data, in particular, some kind of decrease in its performance.

Without additional data about the training samples and training procedure, a number of hypotheses can be made.

It is important to notice a large number of drops on the map chart, it usually means that the mean precision is lower for certain mini-batch as compared to other mini-batches. Perhaps this is due to the fact that the training data is not balanced, i.e. some class dominates in the train dataset. When the drop occurs, it is possible to assume that the mini-batch contains

the alternative class, thus, the network does not train enough on it.

From this it can follow that during testing, that non-dominant dataset now makes up the largest part, at the same time, the model performance might be significantly lower.

Ideally, for each object to detect - there must be at least one similar object in the training dataset with about the same: shape, side of object, relative size, angle of rotation, tilt, illumination. Thus, it is important to build a training dataset with objects at different: scales, rotations, lighting, from different sides, on different backgrounds.

Additionally, the training dataset should have as many images of negative samples as there are images with objects.