Image denoising using weak texture patches and genetic algorithms

Contents

Abstract

Image denoising using weak texture patches and genetic algorithms

Sam Drugan

Image denoising is an important process in the fields of computer vision and digital cameras. In this project we explore how applying denoising filters, such as Chambolle's total vision denoising algorithm and Gaussian blur, to the weak and rich textured area of an image can improve denoising. We use a genetic algorithm to select the filter and parameters that are used for each patch. Using the gradient of each patch in the image and its statistics, we generate the weak texture patches. This gives us a mask of the weak textured areas which we can use to extract the specific pixels needed using matrix multiplication. We can then combine the weak and rich texture patches, after applying the filter, to give us the complete denoised image. The evaluation methods we use are the root mean squared error, peak signal-to-noise ratio, image quality index, and structural similarity index. We tested the model on an image that had a layer of additive white Gaussian noise added. Early results show an improvement in each the previous scores when a different filter is applied to each type of patch over using a single filter on the whole image based on the estimated noise level. This could prove effective when trying to select an optimal filter to use or when trying to optimize the denoising as far as possible.

Declaration

Acknowledgements

List of Figures

Chapter 1

Introduction

1.1 Motivation

1.1.1 Image Noise

Noise is present in an image when there is a random variance in its pixel values. Image denoising is an important step in any image processing application. Fields such as computer vision rely on having high-quality images in order to work as efficiently as possible. Rafati, Mehravar, et al.[1] have shown how sensitive such systems can be to even low levels of noise. Digital cameras are also required to output images that closely represent the real world. It is therefore essential that as much of the process of this is as efficient as possible. The presence of noise in an image can cause problems for both perceived quality and effectiveness of algorithms using the image.

1.1.2 Evolutionary Algorithm

One problem with using denoising filters is the fact that input parameters are required to be selected manually. For example, a Gaussian filter requires an estimated noise level and Chambolles total variation filter requires a weight parameter[2]. Since there are many of these filters, it can take a long time to find the optimal one to use. The idea is to use an evolutionary algorithm to automatically select the filter and parameter(s).

1.1.3 Weak Texture Patches

Another problem faced in denoising images is the presence of naturally noisy areas. One example of this is a beach where the sand is naturally noisy but the sky is smooth. Many images with these areas lose much of their detail when a single filter is applied over the image. Dealing with rich/weak textured areas separately and then combining the results of the two could help address this issue.

1.2 Goals

The main goal of this project is to explore the effectiveness of automatically selecting a denoising filter and parameter. A secondary aim is to see if applying the filter separately to the weak and rich texture patches will have an effect on the outcome. The root mean squared error(RMSE)[3], peak signal to noise ratio(PSNR)[3], image quality index(IQI)[4] and structural similarity index metric(SSIM)[5] are used to evaluate the selected filters. The student aims to achieve better metrics then applying a single filter over the entire image. As a side goal, the student aims to learn more in the fields of image processing, linear algebra, and evolutionary algorithms.

1.3 Project Overview

Initially, the project takes in an image and applies a layer of additive white Gaussian noise to it. It then generates a mask covering the noisy images weak texture patches. The evolutionary algorithm creates a population of individuals which represent the filters to apply to the weak/rich texture patches. From here, the selected filter for each texture type is applied over a copy of the whole image and it uses the mask to take/remove the weak texture patch. The remaining patches in each denoised image are combined to give us the complete denoised image. It then measures the effectiveness of the filter by calculating one of the above metrics, RMSE for example. After enough generations of individuals are tested, it then tests the individual on a different image with a similar level of estimated noise.

1.4 Achievements

Chapter 2

Analysis

2.1 Image Processing

2.1.1 Images in Computers

There are many ways to represent images in memory. The standard way to represent natural pictures is as a raster image. A raster image is a rectangular matrix that can vary in depth. Each position in the matrix represents that locations colour/greyscale value. In this project, images are stored in a matrix with a depth of 3 with each channel representing red, blue, and green respectively. Each position in the matrix is called a pixel. This is an additive colour model as equal amounts of each will give us white.

2.1.2 Image Noise/Denoising

Since storage space limits the ability to store large images due to the increasing space required by higher resolutions, a number of steps are taken to compress the data. One step is to apply colour quantisation to the image to reduce the range of values being stored. Quantisation is the process of estimating a range of values into a discrete value. This can reduce the amount of data stored but retain the same visual quality. A problem with this is it adds errors in the values to the image known as noise. Depending on the level of quantisation, the noise can be more or less noticeable.

The main goal the student will work towards is to visually reduce the effects of noise automatically. Currently, a person looking to denoise an image manually will need to select the filter and parameters themselves. This can be a cumbersome process if they don't understand what is required. Situations like this lead to a lot of trial and error which can be time-consuming. The student hopes to achieve scores in various quality measures that are better than applying simple filters.

This project will focus on additive white Gaussian noise. The main source of this type of noise is during the aquisition stage of the image due to faults in the sensor e.g. the sensors temperature is too high. A standard model for this type of noise is $X = Y + N$ where X is the noisy image, Y is the pure image and N is the layer of additive white gaussian noise. There a various methods already available to reduce this type of noise. Listed here are a few filter types:

2.1.2.1 Median Filter

Assigns the median value to the pixel of it and it's neighbours. It requires no input parameters to work. [4]

2.1.2.2 Gaussian Filter

A gaussian filter blurs an image causing a reduction in noise and detail. This is achieved by convolving the image using a gaussian function. The standard deviation of the noise is required.

2.1.2.3 Chambolle's Total Variation Filter

Attempts to reduce the total variance in the image based on a given weight parameter.[2] A higher weight reduces noise further but also reduces the level of detail.

2.1.2.4 Weiner Filter

Estimates the desired target image by applying a linear time-invarient filter to the signal. Similar to the Gaussian filter, it requires a noise level estimation.[4]

2.1.3 Weak Texture Patches

A weakly textured patch in an image is found where a cluster of pixels contains similar values to each other. Examples of this in natural images would be a wall or a clear sky. These patches are useful in noise estimation as it's easy to detect a disturbance. The main issue with this is the difficulty of detecting weak texture patches in noisy images as the noise variance causes pixel values to vary more.

Liu, Xinhao, et al.[6] propose a method to estimate noise levels of additive white Gaussian noise by analysing weak texture patches in an image. They also show a method for generating a mask of the weak texture patches in an image. The method they propose analyses statistics from the gradient covariance matrix of each patch. The process looks for what is expected in a weak texture patch after a layer of noise has affected the image. It then estimates a threshold such that a patch is weakly textured if the maximum eigenvalue of it's gradient covariance matrix falls below the threshold. This gives a matrix, with the same shape as the image, where there is a one at each position if that pixel is part of a weakly textured patch. The rest of the matrix contains zeros indicating the pixels that are part of richly textured patches.

2.2 Evolutionary Algorithms

Evolutionary algorithms(EA) attempt to mimic the process of natural evolution. Sistinct components work together to emulate this process in some manner. At the most basic level, an EA will maintain a population of individuals, evaluate the effectiveness of each individual using a fitness function, and create the population of the next generation. It repeats this step for a number of generations until an optimal solution is found. After the final generation, the EA will return the overall best-performing individual as a solution. This solution is optimized to the given fitness function. It is ideal when finding solutions to problems without making assumptions about the optimal result. In particular, it is useful in this project as it allows us to search for an optimal pair of denoising filters to use on each of the texture patch types. It also allows for easy and rapid expansion of the filters/parameter combinations being used. Below is a more detailed summary of the elements in an EA:

2.2.1 Individual & Population

An individual in an EA is usually represented as a bit string. The position of a sequence of bits in the string is used to represent what action to take depending on the situation. A population is the set of individuals used for any given generation.

2.2.2 Evaluation & Fitness

The evaluation step takes each individual in the population and returns a fitness value. This step will vary the most between different uses. In the case of this project, it will measure the effectiveness of the denoising filter.

2.2.3 Selection, Crossover & Mutating

There are a number of different techniques for the selection of individuals that are used to create a population for the next generation. One example of a selection function that this project uses is tournament selection. Tournament selection randomly takes a small group of individuals and returns the best performing amongst them. The crossover step pairs up the selected individuals and creates two new individuals. It creates the new individuals by applying a point crossover on the bits strings and returning the two possible results. Finally, there is a chance, decided by the user, for an individual in the next population to be mutated in some way. One common way to do this is to flip one bit in the bit string. The EA does this to prevent every individual from becoming the exact same and preventing further experimentation.

2.3 Evaluation Metrics

As shown above, we have four methods of evaluating the effectiveness of a denoising filter. The main reason for this is that the EA optimizes the result based on the fitness function. One of the above metrics will be used as a the fitness function and we use the others to analyse it's effectiveness in other areas.

2.3.1 Root Mean Squared Error (RMSE)

The mean squared error(MSE) gives the average difference between pixel values in the original image and the denoised image. The RMSE is the square root of this.

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x(i,j) - y(i,j))^2$$

$$RMSE = \sqrt{MSE}$$

where x and y are the original image and the denoised image and M and N are the dimensions of the images.

2.3.2 Peak Signal To Noise Ratio (PSNR)

The PSNR is used as a quality measure when evaluating images. It represents the ratio between the average diference in pixel values and the maximum possible signal. It's particularly useful due to its low complexity which suits it's use as a fitness function.

$$PSNR = 10\log_{10}\frac{(2^n - 1)^2}{\sqrt{MSE}}$$

where n is the number of bits to represent each pixel. For greyscale images this value is 8 while for full colour 24 bits are required.

2.3.3 Image Quality Index (IQI)

IQI is another metric for image quality based on the degree of disturbance. It does this by analysing the loss of correlation, luminance distortion, and contrast distortion.

$$IQI = \frac{4 * \sigma_{xy} * \bar{x} * \bar{y}}{(\sigma_x^2 + \sigma_y^2) * ((\bar{x})^2 + (\bar{y})^2)}$$

where

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

$$\sigma_y^2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})^2$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

2.3.4 Structural Similarity Index Metric SSIM

SSIM is another metric that returns a result based on the luminance, contrast, and structural difference between the two images. While other metrics such as RMSE are purely a mathematical measure, SSIM gives us an idea of the percieved quality of the resulting image. A measure like this is useful in fields such as digital cameras where mathematical accuracy does not matter to the end user.[5]

$$SSIM = \frac{(2 * \bar{x} * \bar{y} + C_1)(2 * \sigma_{xy} + C_2)}{(\sigma_x^2 + \sigma_y^2 + C_2) * ((\bar{x})^2 + (\bar{y})^2 + C_1)}$$

where

$$C_1 = (k_1 L)^2$$
$$C_2 = (k_2 L)^2$$
$$L = (2^n - 1)$$

$k_1 = 0.01$ and $k_2 = 0.03$ by default.

Chapter 3

Design

3.1 Design Overview

As shown in the previous section, the main design for this project is split across the various aspects required to build an EA. The bulk of the work will be in implementing the evaluate function. The student will use python and a number of supporting libraries in order to achieve this. Prior to the EA running, the image dataset is generated.

3.2 Image Dataset

The image dataset is held in a custom object. It will maintain a copy of the original image, the noisy image and the weak texture patch mask. Since the generation of the weak texture patch also estimates the noise level it stores this as well to use when evaluating the result. It also provides functionality to reload the dataset and set the level of noise that's applied.

3.3 The Evaluate Function

This step in the EA comprises of 3 basic steps. Firstly, it takes the individual bit string and decodes it, mapping it to the actions required to be taken out. Next, the denoising filters are applied to copies of the noisy image. The texture patches are extracted using the mask that was previously generated. The extracted denoised images are then recombined to give the full image. Finally, the third step is to return a value as the fitness of the individual. This function will be one of the four evaluation functions outlined previously. Doing this allows for experimentation with optimizing the result to different evaluation metrics.

3.3.1 Individual Decoding

In this case, the first half of the bit string will map to the denoising filter and parameters to use on the weak texture patches while the rest represents the filter to use on the rich texture patches.

3.3.2 Denoising the image

This step in the evaluation function is the most important as the effectiveness of the denoising filter is what gets measured in the final step. The filters selected by the individual are applied to separate copies of the image. The weak/rich texture patches are extracted using the previously generated mask. This can be achieved by the following: $DW = DI_1 * W$ where W is the weak texture mask, DI_1 is the denoised image using the filter selected for the weak textures, and DW is the denoised weak textures. Then, the denoised rich texture, DR, is obtained from the following: $DR = DI_2 - (DI_2 * W)$ where DI_2 is the denoised image using the filter selected for the rich textures. Finally, the full denoised image is reformed as $DI = DW + DR$ where DI is the full denoised image.

3.3.3 Fitness

The image dataset generated above retains the original, untouched, image. The denoised image and the original image are put into one of the fitness functions mentioned above. This is set by the user but uses RMSE by default. The EA will optimize for the lowest possible RMSE score but optimise for the highest score if PSNR, IQI, or SSIM are used.

3.4 Final EA components (Selection, Crossover & Mutation)

This project uses a simple tournament selection as a means to pick the individuals used in the crossover. Since individuals comprise of two decisions, a single point crossover is used. For each pair of individuals, a new pair is created such that the first individual is made up from the first half of one parent, and the second half of the other. The second individual is then created using the remaining parts of the parents. As for the mutation step, each individual will have a 20% chance to have one of their bits flipped. ____

Chapter 4

Implementation

4.1 Technologies Used

The student chose to implement the bulk of his project in Python3. Pyhton3 has a lot of supporting libraries to aid in complex maths, image processing and evolutionary algorithms. It's ease of use also allowed for quick prototyping and debugging. This project also uses a virtual environment to manage the various libraries and maintain a standard development environment across machines. Below is a list of the supporting libraries and a brief description of each one:

4.1.1 Python Libraries

- Numpy: A large library that provides access to N-dimensional arrays with many, fast, supporting functions. THe images used in this project will be stored in numpy arrays. Numpy also contains many useful linear algebra functions which are used to extract the differant texture patches.
- Scikit-Image: This library implements functions to achieve many common image processing tasks. The main ones this project will use are the denoising filters that were previously mentioned. Functions for getting statistics/analysis from images such as some of the previously mentioned evaluation function.
- MatPlotLib: Due to the nature of an image processing project, there is a lot of visual data at each stage. MatPlotLib allows for easy visualisation of data such as the images before and after, and the weak texture mask.
- DEAP[7]: An distributed evolutionary algorithm library that implements a lot of the basic functionality of an EA. It runs the EA in a parallel manner allowing for quicker runtime. This allows the student to experiment and find the most effective method of returning a result.

4.1.2 Jupyter Notebook

The student uses a jupyter notebook to implement and test the workflow of the denoising proccess. A jupyter notebook is a self-hosted web app that allows for live-code, visualisations and text fields. It allows quick and easy tweaking of any specific values and quickly seeing the resulting effect. Since it allows for text fields, sections of code are easily documented and the workflow can be easily followed.

4.2 Images

Images from the Berkely segmentation dataset[8] will be used for testing and evaluation purposes. This allows us to ensure our weak texture mask lines up with the human segmented images as the border between rich and weak textures is often a harsh change. The images available in this dataset are also at a reasonable resolution. This prevents the EA run time from becoming too excessive as applying many denoising filters becomes more cumbersome on the CPU as resolution increases.

The implementation of the dataset object is trivial as it requires a single load images function that accepts the number of images to load and the level of Gaussian noise to add. When images are loaded in, the colour channels are initially in the order of blue, green, red (BGR) while the standard used in display functions is red, green, blue (RGB). Changing the order in this case only requires a reversing of the order of the channels. Python3's list indexing simplifies this as shown below:

```
new_image = new_image[:,:,::-1]
```

4.3 Weak Texture Patches

4.4 Evaluation Function

The evaluation function requires two parameters to be given, the individual, and the image dataset to be used. There are also three optional parameters that can be set. One of these is a curried performance function that takes two images and returns a number. This allows us to define a generic evaluate function that can be changed depending on which metric it is optimizing towards. There is also the option to display the image at each stage of the denoising (original image, noisy image and denoised image). This is false by default due to the frequency that this function gets called during the running of the EA.

4.4.1 Individual Decoding and Action Mapping

Individuals are passed into the evaluation function as a Python list of ones and zeros. A list comprehension is used to converts each half into an int which maps to the filter to be used.

""“python3 def bits_to_int(bit_list): return sum([(x*(2**i)) for i, x in enumerate(bit_list[::-1])])”""

Each filter with predetermined paramters is placed in a list of lamda functions that take in a single argument and return a denoised image. This allows the denoising process to call a position in the list as a function with one argument. Functionality like this is particularily helpful as it lets a user expand or reduce the number/types of filters used easily without worrying about breaking functionality. Below is a short example of this with two entries:

""“python3 denoising_filters = [lambda x : (filters.gaussian(x ,sigma=1)*255).astype(numpy.uint8), lambda x : (restoration.denoise_tv_chambolle(x, weight=0.01)*255).astype(numpy.uint8),]”""

4.4.2 Applying Denoising Filters

The filter used on each type of texture is applied over a copy of the full image. Initially in the project, the textures were extracted from the image and had the relevant filter applied. This caused neighbouring pixels not in the same texture group to be modified due to the nature of how filters worked. After applying the filters, the weak texture mask is used to extract the weak and rich texured areas. As mentioned earlier, Numpy provides functionality for N-dimensional array manipulation. This allows us to extract the weak texture areas by element-wise multiplication of the weak texture mask and the denoised image. This works as the weak texture mask stores a one in each pixels colour channel if that pixels colour channel is in a weak texture patch. Element-wise subtraction is then used to retrieve the rich textures. Finally, element-wise addition of the two texture types gives us back a full image. Bellow is the code required t implement this:

```
denoised_example_image_weak = filters.gaussian(noisy_image, sigma=2)
denoised_example_image_rich = restoration.denoise_tv_chambolle(noisy_image, wei

weak_texture = (denoised_example_image_weak * noisy_image_mask)
strong_texture = (denoised_example_image_rich - (denoised_example_image_rich * 

denoised_image = weak_texture + strong_texture
```

4.5 Evolutionary Algorithm

References

[1] Hossein Hosseini, Baicen Xiao, and Radha Poovendran. Google's cloud vision api is not robust to noise. In Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on, pages 101–105. IEEE, 2017

[2] Duran, Joan & Coll, Bartomeu & Sbert, Catalina. (2013). Chambolle's Projection Algorithm for Total Variation Denoising. Image Processing On Line. 3. 301-321. 10.5201/ipol.2013.61.

[3] Rafati, Mehravar, et al. "Fuzzy Genetic-Based Noise Removal Filter for Digital Panoramic X-Ray Images." Biocybernetics and Biomedical Engineering, vol. 38, no. 4, 2018, pp. 941–965., doi:10.1016/j.bbe.2018.08.005.

[4] Zhou Wang and A. C. Bovik, "A universal image quality index," in IEEE Signal Processing Letters, vol. 9, no. 3, pp. 81-84, March 2002. doi: 10.1109/97.995823

[5] Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in

IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004.

[6] Liu, Xinhao, et al. “Single-Image Noise Level Estimation for Blind Denoising.” IEEE Transactions on Image Processing, vol. 22, no. 12, 2013, pp. 5226–5237., doi:10.1109/tip.2013.2283400.

[7] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau and Christian Gagné, “DEAP: Evolutionary Algorithms Made Easy”, Journal of Machine Learning Research, pp. 2171-2175, no 13, jul 2012.

[8] D. Martin and C. Fowlkes and D. Tal and J. Malik. “A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics.” {Proc. 8th Int'l Conf. Computer Vision , July 2001, vol. 2, pp. 416-423.