The Texture Tiling Model (TTM)

The Texture Tiling Model is an image processing-based computational model that predicts what information is preserved or lost in the human peripheral visual field. The current package generates visualizations of predictions from the Texture Tiling Model for a given image with a specified fixation. We call these visualizations "mongrels".

A detailed description of the Texture Tiling Model can be found in, for example, our review paper:

> Rosenholtz, R., Yu, D., & Keshvari, S. (2019). Challenges to pooling models of crowding: Implications for visual mechanisms. Journal of Vision, vol. 19.

A number of researchers contributed to the code, including: Ruth Rosenholtz, Alvin Raj, Krista Ehinger, Lavanya Sharan, Dian Yu, Shaiyan Keshvari, and Lex Fridman. Thanks also to Maverick Smith for extensive work testing the code on a large dataset. This release contains the code as of March 2020.

In addition to software developed in the Rosenholtz lab, this also includes code from other sources, as described below.

Copyright info

Contact: Ruth Rosenholtz, rruth@mit.edu, 32-D426, MIT, Cambridge, MA 02139

This copyright *does not apply* to the matlabPyrToolbox, nor to the original or modified versions of the texture analysis/synthesis toolbox (Eero Simoncelli, eero.simoncelli@nyu.edu), nor to the FastICA toolbox (Aapo Hyvärinen, aapo.hyvarinen@helsinki.fi), bundled alongside TTM.

Prerequisites

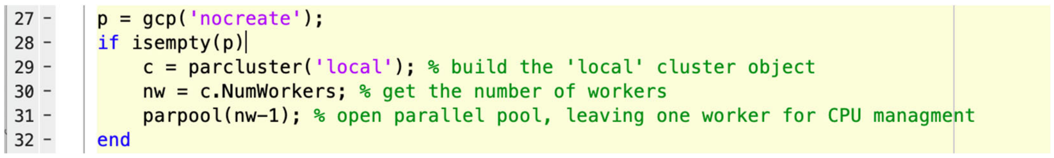
Our software uses the matlabPyrToolbox from Eero Simoncelli. This toolbox can also be downloaded from the dspace directory containing the TTM code. Within the matlabPyrToolbox folder, there is a folder called MEX with various binary executables, precompiled for various platforms. You may need to recompile them by running compilePyrTools.m.

TTM also extensively uses the texture analysis synthesis toolbox from Portilla & Simoncelli (2000) (http://www.cns.nyu.edu/~lcv/texture/). *Use the version provided alongside the TTM code.* If you have already installed the original toolbox, ensure that your path specifies the TTM version when running TTM. We have made small changes to some routines to make it more robust on a wider range of images.

The current program also uses the fastICA toolbox (https://research.ics.aalto.fi/ica/fastica/). Again this toolbox is included alongside the TTM code and you *do not need to* download it again.

The current package uses the MATLAB parallel toolbox to speed up generating multiple mongrels by using multiple processors. Please note that when running large jobs (many images at once), garbage collection does not occur for some of the mex functions; this can cause crashes as MATLAB runs out of memory. A temporary workaround is to use fewer workers or generate mongrels in batches. You can disable use of this toolbox by editing the routing generateMultipleMongrelsFromList.m as follows:

1. Comment out lines 52-57



```
p = gcp('nocreate');
if isempty(p)
    c = parcluster('local'); % build the 'local' cluster object
    nw = c.NumWorkers; % get the number of workers
    parpool(nw-1); % open parallel pool, leaving one worker for CPU managment
end
```

2. Change parfor in line 74 to for

Running the program

Creating mongrels for a given image, manual fixation selection

1. Run convertImageForMongrels. Example call:

```
convertImageForMongrels('demo/a1.png', 'num_mongrels', 10);
```

This function takes an image filename as input (if the image is not in the path, this includes its directory, as shown above), and a number of optional parameters. Shown above: the number of mongrels you wish to generate for this image (default = 1). This function allows you to select the fixation and fovea size by clicking on the image. It will generate the number of mongrels you specified given that fixation. For any given image and fixation point, the information encoded by TTM is consistent with a number of possible images. To get a sense of what information the model says is preserved and lost in peripheral vision, one typically wants to generate a number of mongrels for each (image, fixation) pair. Other options include 'fovea_radius', to specify the radius in pixels.

The function generates a text file in the same directory as the image. Each row specifies the parameters for running TTM in the following tab-separated format:

```
directory/filename	fix_x	fix_y	radius-of-fovea	mongrel_index
```

where fix_x, fix_y, and radius-of-fovea are specified in pixels (the top-left corner of the image is pixel (1,1), and mongrel_index runs from from 1 to num_mongrels. *Note that the model may produce unreliable results if radius-of-fovea is less than 16 (pooling regions near the fovea will be too small). Typical use is 16-32 pixels.* For an example, see

```
demo/a1.png_mongrel_list_example.txt
```

2. Run generateMultipleMongrelsFromList.m with the previously generated txt file. Include directory if the textfile is not in the path. Example:

```
generateMultipleMongrelsFromList('demo/a1.png_mongrel_list.txt');
```

Final output will be saved in a folder under the current parent directory. The directory name will be of the following format, where the x- and y-coordinates specify the modeled fixation:

```
imageName_Xcoordinate_Ycoordinate_date_mongrelIndex
```

Create mongrels given an image and known fixation coordinates (i.e. no need to hand select fixation)

1. Create a txt file where each row follows the following tab-separated format:

```
directory/filename    fix_x    fix_y    radius-of-fovea  mongrel_index
```

where fix_x, fix_y, and radius-of-fovea are specified in pixels, and mongrel_index runs from 1 to num_mongrels. The text file 'demo/a1.png_mongrel_list_example.txt' shows an example for the case in which there is one input image, two fixations, and four mongrels per fixation. *Note that the model may produce unreliable results if radius-of-fovea is less than 16 (pooling regions near the fovea will be too small), and typical use is 16-32 pixels.*

2. run generateMultipleMongrelsFromList.m with the txt file you created. Example:

```
generateMultipleMongrelsFromList('demo/a1.png_mongrel_list_example.txt');
```

Final output will be saved in a folder under the current parent directory with a name in the following format:

```
imageName_Xcoordinate_Ycoordinate_date_mongrelIndex
```

Synthesis parameters

The current program runs with a set of parameters specified in default.job. The list of parameters are as follows, with notes relating to each:

Random number generation

```
random_mode 'shuffle'
random_type []
random_seed []
```

Generating a mongrel visualization is a stochastic process. It generates a different image depending on how one seeds MATLAB's random number generator. Usually one just wants to use an essentially "random" seed, e.g. generated based on the date and time. To do that, use the default parameters shown above. Sometimes one may instead want to use a previously used seed, for example to start two syntheses with different parameters but the same starting point of the synthesis, or when rerunning a previous synthesis. In this case, set random_mode to anything other than 'shuffle', and set random_type and random_seed to the previous type and seed. (Type `help rng` for options for random_type.)

Debugging

```
saveintermediate 1     % save intermediate results or no?
```

Pooling region parameters

```
poolingRate .5         % poolingRate * eccentricity specifies pooling region size
radialOverlap .25      % specifies the amount that neighboring pooling regions overlap radially
numAngular 36          % specifies how many pooling regions at each (quantized) eccentricity
```

Parameters governing the optimization process

```
scalesToRun 1:4
niters [10 10 10 10]
```

TTM does a sort of coarse-to-fine optimization to generate each mongrel. The parameters nIters specifies how many global optimization steps to perform (how many times the software iterates across the entire image), for each scale specified in scalesToRun. In default mode, the code generates the mongrel starting with the coarsest scale (1), and then progressively adds additional scales to generate finer detail. nIters must be a vector of the same length as scalesToRun. For example, scalesToRun = [1 2 3 4] and nIters = [10 20 5 10] means 10 iterations at scale 1; 20 at scales 1 and 2; 5 iterations with the coarsest 3 scales, and 10 iterations at the end with all scales.

```
prIters 3
```

prIters specifies, *within each pooling region*, how many iterations of optimization on a given round. In other words, this is the number of *local* iterations of a given pooling region, whereas nIters is the number of global iterations of optimization across the entire image.

Parameters inherited from Portilla and Simoncelli (2000)

```
Nor 4               % number of filter orientations
Na 7                % number of autocorrelation taps
```

Note that the number of scales parameter from Portilla and Simoncelli is determined by the maximum of scalesToRun.

Notes on using non-default parameters

One can change certain parameters in the job file and pass it to generateMultipleMongrelsFromList.m. Example:

```
generateMultipleMongrelsFromList('demo/a1.png_mongrel_list.txt','newJobFile.job')
```

One can play with some of the parameters specified in default.job without thinking too much about the model's architecture, including scalesToRun, Nor, Na, nIters, and prIters. Changing other parameters, including poolingRate, numAngular, and radialOverlap, which may be useful for changing the critical spacing predicted by the model, require deep understanding about what the algorithm does.