



HEAT Documentation

Release 2.0

Nicolae Suditu

September 17, 2012

CONTENTS

1	COPYING	1
2	README	3
2.1	Description	3
2.2	Documentation	3
2.3	Reference papers	3
2.4	Demo web-server	4
2.5	Contact info	4
3	REQUIREMENTS	5
3.1	Required packages	5
3.2	Optional packages	5
4	INSTALLATION	7
4.1	Set up the Django project	7
4.2	Set up the MySQL database	7
5	GETTING STARTED	9
5.1	Run the Django web-server	9
5.2	Run the automatic application	9
5.3	Generate the Sphinx documentation	9
5.4	Access the MySQL database	10
6	LINUX CONFIGURATION	11
6.1	Set up the Apache server	11
7	DIRECTORY STRUCTURE	13
8	IMPLEMENTATION	15
8.1	settings	15
8.2	dbutils	16
8.3	apps	17
8.4	autoapp	20
8.5	libs	20
	Python Module Index	23

COPYING

HEAT is a content-based image retrieval application.

Copyright (c) 2012 Idiap Research Institute <<http://www.idiap.ch/>>

Written by Nicolae Suditu (Nicolae.Suditu@idiap.ch)

HEAT is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License version 3 (AGPLv3) as published by the Free Software Foundation.

HEAT is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License version 3 (AGPLv3) along with HEAT. If not, see <<http://www.gnu.org/licenses/>>.

README

HEAT is a content-based image retrieval web-application.

2.1 Description

HEAT is an image retrieval web-application that is intended for large unstructured collections of images without semantic annotations. The system implements a novel searching paradigm that does not require any explicit query, and it relies solely on an iterative relevance feedback mechanism. At each iteration, the system displays a small set of images and the user chooses the image that best matches what she is looking for. The system updates an internal state and displays a new set of images accordingly. After a few iterations, the sets of displayed images are gradually concentrated on images that satisfy the user. In principle, the system works on any multimedia collection for which one can provide vector-like indexing features, as for example bags-of-words based on the Scale Invariant Feature Transform (SIFT).

The web-application has an infrastructure for conducting user-based evaluations, which was actually used by the experiments published in the conference papers mentioned below. There is support for inter-changing image collections and different versions of the iterative relevance feedback mechanism. Moreover, there is infrastructure for handling user accounts, managing test configurations, recording searching sessions and plotting various statistics.

Besides the web-application, there is a test-platform for running searching sessions automatically. This test-platform implements a virtual user that interacts with the web-application in exactly the same way a human user does. In essence, the test-platform is a bridge between the web-application and the virtual user that knows to look at the currently displayed images and to simulate the relevance feedback events that are normally given by human users. This platform is very useful for abstract analyzes and code verification.

2.2 Documentation

The project documentation in PDF format is provided along with the archive:

`./docs/heat.pdf`

2.3 Reference papers

[1] N. Suditu and F. Fleuret, “HEAT: Iterative Relevance Feedback with One Million Images”, in Proceedings of the IEEE International Conference on Computer Vision (ICCV), November 2011.

[2] N. Suditu and F. Fleuret, “Iterative Relevance Feedback with Adaptive Exploration/Exploitation Trade-off”, in Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), October 2012.

2.4 Demo web-server

A demo web-server is available on-line at [<http://imr.idiap.ch/>](http://imr.idiap.ch/).

2.5 Contact info

Nicolae Suditu (Nicolae.Suditu@idiap.ch)

François Fleuret (Francois.Fleuret@idiap.ch)

Idiap Research Institute [<http://www.idiap.ch/>](http://www.idiap.ch/)

REQUIREMENTS

3.1 Required packages

- MySQL 1.5
- **Python 2.6 and the following packages**
 - matplotlib
 - numpy
 - scipy
- Django 1.2
- Cython 0.14

3.2 Optional packages

- Sphinx documentation generator
- **Packages for debugging**
 - Django-logging
 - Django-extensions
 - Pyximport
 - Pygments
 - Werkzeug
 - pep8
 - pylint
- **Add-ons for Mozilla Firefox**
 - Firebug
 - Dust-Me Selectors
 - CSS Refresh
 - Selenium

INSTALLATION

HEAT is a Django-based project, but basically it does not use much of the Django platform besides the view/template infrastructure. The project does not need a syncdb operation for creating the application model tables in MySQL, but only for creating the Django-related tables.

The project is self-contained and it is meant to be installed in its own directory, and it is not deployed as an egg or plug-in application. This is mainly because it needs quite some manual set up and configuration.

4.1 Set up the Django project

1. Make sure the packages mentioned in the REQUIREMENTS are installed.
2. Configure the settings files. Set up the MySQL database access information and all the directories needed by the application. There is detailed info in the files themselves:

```
./heat/settings/common.py
./heat/settings/debug.py
./heat/settings/development.py
./heat/settings/production.py
./heat/settings/staging.py
./heat/settings/maintenance.py
```

3. Open a terminal and go to the project root directory:

```
cd project_root (the directory of this file)
```

4. Compile the Cython optimized code:

```
python setup.py build_ext --inplace
```

5. Initialize the MySQL tables needed by the Django project. There is detailed info at the Django's website <<https://www.djangoproject.com/>>:

```
python manage.py syncdb
```

4.2 Set up the MySQL database

There are provided scripts to set up the system for a synthetic image collection that is useful for illustrating the system behavior. There is detailed info in the file itself:

```
./heat/dbutils/imagesynthetic.py
```

There are provided scripts to set up the system for the ImageNet collection that is freely available at <<http://www.image-net.org/>>. The scripts get as input the files provided by ImageNet. There is detailed info in the files themselves:


```
./heat/dbutils/imagenet_1M.py  
./heat/dbutils/imagenet_33K.py  
./heat/dbutils/imagenet_60K.py
```

The settings file used by these scripts is:

```
./heat/settings/maintenance.py
```

Following these examples, one can set up the system for its own image collections. The settings files should be updated accordingly.

GETTING STARTED

5.1 Run the Django web-server

The Django command to start the lightweight development Web server on the local machine is:

```
./heat/manage.py runserver 8000 --settings=heat.settings.production
```

For developers, it is useful to install the optional package Django-extensions, and to start the Django Web server in the debug mode:

```
./heat/manage.py runserver_plus 8000 --settings=heat.settings.development
```

Then, the Django Web server can be accessed in one's preferred browser at the address:

```
http://127.0.0.1:8000/
```

5.2 Run the automatic application

There is provided a script to run the system programatically, in which the search sessions are driven by an automatic user:

```
./heat/autoapp/workflow.py
```

The settings file used by the automatic application is:

```
./heat/settings/staging.py
```

This file should be updated according to the test purpose. As a main setting, there are the collections to be tested together with the tasks and the configurations to be shuffled in the test. There is detailed info in the file itself.

5.3 Generate the Sphinx documentation

The project documentation in PDF format is provided along with the archive:

```
./docs/heat.pdf
```

The Sphinx documentation source files are provided as well:

```
./extra/sphinx
```

The project documentation in HTML format can be created by the following command:

```
sphinx-build -E -b dirhtml ./extra/sphinx ./docs/html
```

The project documentation in LATEX format can be created by the following command:


```
sphinx-build -E -b latex ./extra/sphinx ./docs/latex
```

5.4 Access the MySQL database

Usual commands in the terminal:

```
mysql --user=user --password=password database
```

```
SHOW TABLES;
```

```
DESCRIBE table;
```

```
SHOW INDEX FROM table;
```

```
SELECT * FROM table WHERE id=1;
```

```
SELECT COUNT(*) FROM table WHERE column LIKE 'xxx/%';
```

```
CREATE TABLE table2 LIKE table;
```

```
INSERT INTO table2 SELECT * FROM table WHERE id=1;
```

```
TRUNCATE TABLE table;
```

```
DROP TABLE table;
```

```
ALTER TABLE table RENAME TO table2;
```

```
DELETE FROM table WHERE id > 100;
```

Backup all the tables that have their names starting with “ImageNet_1M”:

```
mysqldump --user=user --password=password --host=mysqlserver database  
$(mysql --user=user --password=password --host=mysqlserver database  
-Bse "show tables like 'ImageNet_1M%'" ) > ./ImageNet_1M_bkp.sql
```

Upload the backup data into the database:

WARNING: It overwrites the existing tables with the same names.

```
mysql --user=user --password=password --host=mysqlserver database < ./ImageNet_1M_bkp.sql
```


LINUX CONFIGURATION

6.1 Set up the Apache server

Edit the configuration file “/etc/apache2/sites-available/default”:

```
<VirtualHost *:80>
```

```
    ServerAdmin EMAIL_ADDRESS
```

```
    ServerRoot PROJECT_ROOT
```

```
    DocumentRoot PROJECT_ROOT
```

```
<Directory PROJECT_ROOT/heat/>
```

```
    Options FollowSymLinks MultiViews
```

```
    AllowOverride None
```

```
    Order deny,allow
```

```
    Deny from all
```

```
    Allow from all
```

```
    AuthName "Protected Area"
```

```
    AuthType Basic
```

```
    AuthGroupFile /dev/null
```

```
    AuthUserFile PROJECT_ROOT/extra/config-apache/htpasswd
```

```
    Require user LIST_OF_USERS
```

```
    Satisfy any
```

```
</Directory>
```

```
Alias /files/ PROJECT_ROOT/media/
```

```
<Directory PROJECT_ROOT/media/>
```

```
    Options FollowSymLinks MultiViews
```

```
    AllowOverride None
```

```
    Order deny,allow
```

```
    Deny from all
```

```
    Allow from all
```

```
</Directory>
```

```
Alias /docs/ PROJECT_ROOT/docs/html/
```

```
<Directory PROJECT_ROOT/docs/html/>
```

```
    Options FollowSymLinks MultiViews
```

```
    AllowOverride None
```

```
    Options None
```



```
        Order allow,deny
        Allow from all
</Directory>

ErrorLog /var/log/apache2/error.log

# Possible values include: debug, info, notice, warn, error, crit, alert, emerg.
LogLevel warn

CustomLog /var/log/apache2/access.log combined

<Location />
    # Settings for Django
    SetHandler python-program
    PythonHandler django.core.handlers.modpython
    SetEnv DJANGO_SETTINGS_MODULE heat.settings.production

    #PythonOption django.root /
    PythonPath "['PATH_TO_YOUR_OWN_PYTHON_PACKAGES','PROJECT_ROOT',] + sys.path"
    PythonDebug On
    # enable expirations
    #<IfModule mod_expires.c>
    #     ExpiresActive On
    #     ExpiresDefault A1
    #</IfModule>
</Location>

<Location /files>
    SetHandler None
</Location>

<Location /docs>
    SetHandler None
</Location>

</VirtualHost>
```


DIRECTORY STRUCTURE

The project directory structure is as follows:

```
|project_root/
|  |-- COPYING
|  |-- README
|  |-- INSTALL
|  |-- REQUIREMENTS
|  |-- setup.py
|  |-- docs/
|  |   |-- heat.pdf
|  |-- extra/
|  |   |-- selenium/
|  |   |-- sphinx/
|  |-- media/
|  |   |-- css/
|  |   |-- js/
|  |   |-- ImageNet/
|  |   |-- ImageSynthetic/
|  |   |-- figures/
|  |-- tmp/
|  |   |-- django-cache/
|  |-- heat/
|  |   |-- manage.py
|  |   |-- settings/
|  |   |   |-- common.py
|  |   |   |-- debug.py
|  |   |   |-- production.py
|  |   |   |-- development.py
|  |   |   |-- staging.py
|  |   |   |-- maintenance.py
|  |   |-- dbutils/
|  |   |   |-- imagesynthetic.py
|  |   |   |-- imagenet_1M.py
|  |   |   |-- imagenet_33K.py
|  |   |   |-- imagenet_60K.py
|  |   |-- apps/
|  |   |   |-- registration/
|  |   |   |-- retrieval/
|  |   |-- autoapp\
|  |   |   |-- workflow.py
|  |   |-- templates/
|  |   |   |-- registration/
|  |   |   |-- retrieval/
|  |   |-- libs/
|  |   |   |-- retrieval_libs/
```


IMPLEMENTATION

8.1 settings

Provides the settings for running the Django application.

8.1.1 `settings.common`

Common settings.

- It configures the Django project and its applications
- It configures the loggers and sets them at the INFO level

`settings.common.DATABASES`
connection settings for the MySQL database

`settings.common.ALL_DB`
list of system setups containing the specific settings for each image collection

`settings.common.DEFAULT_DB`
the default system setup

`settings.common.FLAG_TRACEGFX = {'TRIX_2D': True, 'PROB_2D': False}`
enable/disable the trace representations for display

`settings.common.ITER_MAX = 20`
the maximum number of iterations for the sessions during testing

8.1.2 `settings.debug`

Debug settings.

- It sets the loggers at the DEBUG level
- It activates the django-logging middleware

8.1.3 `settings.production`

Settings for running the application in production mode.

8.1.4 `settings.development`

Settings for running the application in development mode.

8.1.5 `settings.staging`

Settings for running the application programmatically/automatically.

`settings.staging.FLAG_TRACEGFX = False`
enable/disable the trace representation for display

`settings.staging.PROFILE = False`
for profiling/timing the program execution

`settings.staging.REINIT_LOOP = 1`
the period of the re-initialization of the automatic tasks

If it is set to `TESTS_MAX`, the tasks will be fixed once for all the tests. This is useful for getting the statistical performance of the automatic user for one specific task.

If it is set to 1, the tasks will be randomized after each test, so after each complete set of sessions. This is useful for getting the statistical performance of the automatic user for a class of tasks.

`settings.staging.TESTS_MAX = 1000`
the number of tests to be performed by the automatic users

A `**test**` is a set of sessions. A `**session**` refers to one search/retrieval process. Each session is performed on one image database and it is configured with one task-config peer.

The number of sessions in one `*test*` is equal to the sum of all possible task-config peers for all image databases.

8.1.6 `settings.maintenance`

Settings for uploading the image collections in the database.

8.2 `dbutils`

Provides functionality for uploading the image collections in the database.

8.2.1 `dbutils.imagesynthetic`

Uploads the ImageSynthetic collection, where each image contains a 2D point and the image features are the 2D coordinates of that point.

The hierarchical organization of the collection is as follows. All the points are within the square (0.0, 0.0, 1.0, 1.0), and this is the root square. For any node, its children are obtained by dividing its square into 4 squares, and its representative image is the center of its square.

For example, the root node represents the full square (0.0, 0.0, 1.0, 1.0), and its representative image is the central point (0.5, 0.5). Then, its 4 children nodes would represent the squares:

- (0.0, 0.0, 0.5, 0.5) - lower left
- (0.0, 0.5, 0.5, 1.0) - upper left
- (0.5, 0.5, 1.0, 1.0) - upper right
- (0.5, 0.0, 1.0, 0.5) - lower right

8.2.2 `dbutils.imagenet_1M`

Uploads the ImageNet_1M collection.

- ImageNet collection is provided freely at <http://www.image-net.org/>
- It is assumed that the ImageNet files have been downloaded

8.2.3 `dbutils.imagenet_33K`

Uploads the ImageNet_33K collection.

- ImageNet_33K is generated by sub-sampling ImageNet_1M
- It is assumed that ImageNet_1M was already set up

8.2.4 `dbutils.imagenet_60K`

Uploads the ImageNet_60K collection.

- ImageNet_60K is generated by sub-sampling ImageNet_1M
- It is assumed that ImageNet_1M was already set up

8.3 apps

Repository for the Django applications in the project.

8.3.1 registration

The registration application provides functionality to differentiate between anonymous, regular and staff users.

Anonymous users can access only minimal functionality:

- perform search sessions

Regular users can access additional functionality:

- see the training examples
- perform evaluation tests

Staff users can access additional functionality:

- manage (create/delete/modify) the User, Task, Config models
- view the statistics of the system performance
- watch the test search sessions saved in the database

There is some functionality for managing user groups, e.g. expert users, but there is not much use of it for now.

`registration.models.User`

```
class registration.models.User(*args, **kwargs)
    User model is a sub-class of the DjangoUser model.
```


`registration.models.Group`

`class registration.models.Group(*args, **kwargs)`
Group model is a sub-class of the DjangoGroup model.

8.3.2 retrieval

The content-based image retrieval application.

`retrieval.middleware.DbSetup`

`class retrieval.middleware.DbSetup`
Set up the system for the current database.

`retrieval.models.Config`

Provides the system configuration.

`retrieval.models.Config.get_config_model(active_db)`
Creates the corresponding Config model.

`class retrieval.models.Config._Config(*args, **kwargs)`
Abstract base-class for the Config models.

Variables

- **name** – the label and the display properties
- **TRACE** – the trace size
- **SYS_THARGS** – the image similarity thresholds
- **SYS_WEIGHTS** – the image feature’s weights
- **SYS_CONSISTENCY** – the consistency type
- **SYS_VERSION** – the algorithm version
- **datetime** (<http://docs.python.org/library/datetime.html#datetime>) – the time of the last saving in the database

`retrieval.models.Task`

Provides the target that is given as reference to the users during the testing and training.

`retrieval.models.Task.get_task_model(active_db)`
Provides access to the corresponding Task model.

`class retrieval.models.Task._Task(*args, **kwargs)`
Abstract base-class for the Task models.

Variables

- **query** – the text query
- **target** – the target image set
- **USR_THARGS** – the image similarity thresholds
- **USR_WEIGHTS** – the image feature’s weights
- **USR_VERSION** – the algorithm version
- **datetime** (<http://docs.python.org/library/datetime.html#datetime>) – the time of the last saving in the database

retrieval.models.Session

Manages the searching session data.

The session data is accumulated iteration by iteration. In between the iterations, the session data is stored in the Django-cache-per-user. At the end of the searching session, the session data is usually deleted, but in case of testing it stored in the database.

`retrieval.models.Session.get_session_model(active_db)`

Provides access to the corresponding Session model.

class `retrieval.models.Session._Session(*args, **kwargs)`

Abstract base-class for the Session models.

Variables

- **user** (<http://docs.python.org/library/user.html#user>) – the user that performed the session
- **task** – the task given to the user
- **config** – the system configuration
- **iterations** – the number of iterations (redundant)
- **evaluation** – the evaluation given by the user
- **actionH** – the actions performed on the input data
- **thargsH** – the history of the image similarity thresholds
- **weightsH** – the history of the image feature's weights
- **consistencyH** – the history of the consistency scores
- **dispfdbkH** – the displayed image sets and the corresponding relevance feedback that was given
- **traceH** – the evolution of the size of the trace
- **timingH** – the timing of system response and user feedback
- **datetime** (<http://docs.python.org/library/datetime.html#datetime>) – the time of the saving in the database

retrieval.models.Statistics

Provides the statistics of the system performance.

The model retrieves the session data from the database and generates various plots. There is user interface to filter the data and to select the statistics of interest.

`retrieval.models.Statistics.get_statistics_model(active_db)`

Provides access to the corresponding Statistics model.

class `retrieval.models.Statistics._Statistics(*args, **kwargs)`

Abstract base-class for the Statistics models.

Variables

- **users** – the list of users
- **tasks** – the list of tasks
- **configs** – the list of system configurations

8.4 autoapp

Performs automatic evaluation tests.

8.4.1 autoapp.workflow

Manages the workflow of the automatic user actions and the system responses during the automatic evaluation tests. The main user actions that are involved are:

- start a new searching session
- provide the relevance feedback

8.4.2 autoapp.browser

Encapsulates the functionality for the Django-view and the Html-page that corresponds to the user intelligence on how to use the application:

- what are the actions required in a certain state
- what are the buttons that trigger the actions

8.5 libs

Repository for the Django application libraries.

8.5.1 dblayer

Provides functionality for interacting with the database.

This functionality is somehow similar with the Django models, but it is handy for the MySQL tables that need only low level access.

`retrieval_libs.dblayer.DbAccess`

Provides the DbTable instances.

class `retrieval_libs.dblayer.DbAccess.DbAccess`
Provides the DbTable instances.

Note: Implemented as a very simple singleton.

`retrieval_libs.dblayer.DbCnx`

Manages the database connection and interaction.

`retrieval_libs.dblayer.DbTable`

Manages the generic interaction with the tables.

retrieval_libs.dblayer.DbTableImg

Manages the table of the image data, e.g. filenames, features and distances.

Among other stuff, it encapsulates the functionality for providing the image similarity distances. It knows to choose between computing the distances on-the-fly or reading the files with the pre-computed distances.

class `retrieval_libs.dblayer.DbTableImg.DbTableImgSynthetic`
 DbTableImgSynthetic is a sub-class of DbTableImg.

Note: It generates the synthetic images on-the-fly.

retrieval_libs.dblayer.DbTableTree

Manages the table of the hierarchical tree-like organization of the images.

retrieval_libs.dblayer.FileAccess

Provides methods to access the data files for R/W/A operations.

retrieval_libs.dblayer.FileManager

Generates unique file names.

8.5.2 search**retrieval_libs.search.SearchMain**

Manages the workflow within one iteration of the retrieval process.

Based on the system configuration (see [retrieval.models.Config](#) (page 18)) that is currently active, the retrieval iteration employs one of the following algorithms:

retrieval_libs.search.im_orig

Performs the original retrieval algorithm.

class `retrieval_libs.search.im_orig.QueryImg.QueryImg`
 Manages the original retrieval algorithm.

class `retrieval_libs.search.im_orig.ProbabilityUpdater.ProbabilityUpdater`
 Updates the probabilities of image relevance.

class `retrieval_libs.search.im_orig.DisplaySetSelector.DisplaySetSelector`
 Selects the image set for display.

Note: The images are chosen by performing an heuristic algorithm.

class `retrieval_libs.search.im_orig.TraceUpdater.TraceUpdater`
 Manages the rankdata.

Note: Rankdata has the size of the collection at hand.

class `retrieval_libs.search.im_orig.ConsistencyUpdater.ConsistencyUpdater`
Estimates the consistency based on the relevance feedback history.

Note: The consistency scores are used to compute the zoom-in factor.

`retrieval_libs.search.im_heat`

Performs the HEAT retrieval algorithm.

class `retrieval_libs.search.im_heat.QueryImg.QueryImg`
Manages the HEAT retrieval algorithm.

class `retrieval_libs.search.im_heat.DisplaySetSelector.DisplaySetSelector`
Selects the image set for display.

Note: The images are chosen by performing an heuristic algorithm.

class `retrieval_libs.search.im_heat.TraceUpdater.TraceUpdater`
Manages the rankdata.

Note: Rankdata has the size of the trace within the hierarchical tree, and it is dynamically updated by collapse/expand operations.

`retrieval_libs.search.im_rnd`

Performs the random retrieval algorithm.

class `retrieval_libs.search.im_rnd.QueryImg.QueryImg`
Manages the random retrieval algorithm.

class `retrieval_libs.search.im_rnd.DisplaySetSelector.DisplaySetSelector`
Selects the image set for display.

Note: The images are chosen randomly.

class `retrieval_libs.search.im_rnd.TraceUpdater.TraceUpdater`
Manages the rankdata.

Note: Rankdata has the size of the collection at hand.

PYTHON MODULE INDEX

a

autoapp, 20
autoapp.browser, 20
autoapp.workflow, 20

d

dbutils, 16
dbutils.imagenet_1M, 17
dbutils.imagenet_33K, 17
dbutils.imagenet_60K, 17
dbutils.imagesynthetic, 16

r

registration.models, 17
retrieval.models, 18
retrieval.models.Config, 18
retrieval.models.Session, 19
retrieval.models.Statistics, 19
retrieval.models.Task, 18
retrieval_libs.dblayer, 20
retrieval_libs.dblayer.DbAccess, 20
retrieval_libs.dblayer.DbCnx, 20
retrieval_libs.dblayer.DbTable, 20
retrieval_libs.dblayer.DbTableImg, 21
retrieval_libs.dblayer.DbTableTree, 21
retrieval_libs.dblayer.FileAccess, 21
retrieval_libs.dblayer.FileManager, 21
retrieval_libs.search.im_heat, 22
retrieval_libs.search.im_orig, 21
retrieval_libs.search.im_rnd, 22
retrieval_libs.search.SearchMain, 21

s

settings, 15
settings.common, 15
settings.debug, 15
settings.development, 15
settings.maintenance, 16
settings.production, 15
settings.staging, 16