IBM

Visual Recognition
Analyst Tool

Version 2.0

User's Guide

August 2, 2018

Contents

Chapter 1

Installation

In this chapter we will describe installation methods of *IBM VR Analyst Tool*
on major operation systems.

1.1 Mac OS

1) Install Python 3 on your machine.

2) Check if Python 3 was successfully installed by typing **python** *or*
   python3 command in Terminal. It should run Python 3 environment
   in the Terminal window. Type *exit()* to close Python 3 environment.

3) Download the *IBM VR Analyst Tool* from GitHub url `https://github.`
   `ibm.com/OSzekely/ibm-vr-analyst-tool` via GIT program or down-
   load it with a browser as a *.zip* file. If you download a *.zip* file, please
   extract it.

   Place the folder with *IBM VR Analyst Tool* somewhere in the file sys-
   tem (set installation directory). Be aware, that the path to *IBM VR
   Analyst Tool* should not change in the future. If you change the loca-
   tion of *IBM VR Analyst Tool* , you need to perform step 5 again.

   Now, change the name of *IBM VR Analyst Tool* root folder as you
   like.

4) Open Terminal and type **cd /Users/userName/**.
   Type **sudo nano .bash_profile** and go to the end of file. Put there
   commands from *mac_os* file located in *computer_var_setting* subfolder.

You have to modify variable $ROOT \_ FOLDER$ and set a valid path to *IBM VR Analyst Tool* derived in step 4. Pay attention, path must not contain a backslash in the end of the path.

To apply changes you have to log out and sign in into system.

5) Install pip tool for Python 3 instance, if it is not already installed.

6) Install OpenCV3 for python. You can build it from source files or install it through Brew (guide https://www.codingforentrepreneurs.com/blog/install-opencv-3-for-python-on-mac/)

7) Install additional Python 3 packages with pip tool (in Terminal type ***pip install packageName***):

   – simplejson

   – tqdm

   – watson-developer-cloud

   – numpy

   – pillow

   – matplotlib

1.2   Linux

1) Install Python 3.

2) Check if Python 3 was successfully installed by typing ***python*** *or* ***python3*** command in Terminal. It should run Python 3 environment in the Terminal window. Type *exit()* to close Python 3 environment.

3) Download the *IBM VR Analyst Tool* from GitHub url `https://github.ibm.com/OSzekely/ibm-vr-analyst-tool` via GIT program or download it with a browser as a *.zip* file. If you download a *.zip* file, please extract it.

   Place the folder with *IBM VR Analyst Tool* somewhere in the file system (set installation directory). Be aware, that the path to *IBM VR Analyst Tool* should not change in the future. If you change the location of *IBM VR Analyst Tool* , you need to perform step 5 again.

Now, change the name of *IBM VR Analyst Tool* root folder as you like.

4) Open Terminal and type **cd /home/userName/**.
   Type **sudo nano .bashrc** and go to the end of file. Put there commands from *mac_os* file located in *computer_var_setting* subfolder.

   You have to modify variable *ROOT_FOLDER* and set a valid path to *IBM VR Analyst Tool* derived in step 4. Pay attention, path must not contain a backslash in the end of the path.

   To apply changes you have to log out and sign in into system.

5) Install pip tool for Python 3 instance, if it is not already installed.

6) Install OpenCV3 for python. You can build it from source files or install it through Brew (guide https://www.codingforentrepreneurs.com/blog/install-opencv-3-for-python-on-mac/)

7) Install additional Python 3 packages with pip tool (in Terminal type **pip install packageName**):

   - simplejson
   - tqdm
   - watson-developer-cloud
   - numpy
   - pillow
   - matplotlib

1.3   Windows

1) Install Python 3.

2) Check if Python 3 was successfully installed by typing **python** *or* **python3** command in Terminal. It should run Python 3 environment in the Terminal window. Type *exit()* to close Python 3 environment.

3) Download the *IBM VR Analyst Tool* from GitHub url `https://github.com/OndrejSzekely/vr_analyst_tool` via GIT program or download it with a browser as a *.zip* file. If you download a *.zip* file, please extract it.

Place the folder with *IBM VR Analyst Tool* somewhere in the file system (set installation directory). Be aware, that the path to *IBM VR Analyst Tool* should not change in the future. If you change the location of *IBM VR Analyst Tool* , you need to perform step 5 again.

Now, change the name of *IBM VR Analyst Tool* root folder as you like.

4) Right click on *This PC → Advanced system settings → Advanced → Environment Variables...*

Under *User variables for UserName* tab click on *New*, if PYTHONPATH variable does not exist, then type PYTHONPATH in *Variable name* entry in a new tab. Then click on *Browse Directory...* and browse for the root folder of *IBM VR Analyst Tool* . Submit all choices by *OK* buttons.

If PYTHONPATH is already defined in users variables, choose PYTHONPATH and click on *Edit...* Click in *Variable value* entry to the end, to append a new path. If there is not a semicolon in the end of path, place it. now click on *Browse Directory...* and browse for the root folder of *IBM VR Analyst Tool* . Submit all choices by *OK* buttons.

5) Install pip tool for Python 3 instance, if it is not already installed.

6) Install OpenCV3 for python. You can build it from source files or install it through Brew (guide https://www.codingforentrepreneurs.com/blog/install-opencv-3-for-python-on-mac/)

7) Install additional Python 3 packages with pip tool (in Terminal type ***pip install packageName***):

   – simplejson
   – tqdm
   – watson-developer-cloud
   – numpy
   – pillow
   – matplotlib

Chapter 2

Update to a new version

Download a new version of *IBM VR Analyst Tool* from `https://github.com/OndrejSzekely/vr_analyst_tool`. Copy all content from downloaded *IBM VR Analyst Tool* root folder and rewrite the old ones in installation directory of *IBM VR Analyst Tool* .

Chapter 3

JSON config and modules overview

3.1 Integration tests

There are 22 tests which are performed in series. To run integration test please fill API key info in `unit_and_integration_test/credentials.json` and then run `unit_and_integration_test/run_tests.sh`. Whole testing should take circa an 1.5 hours.

3.2 Run Scripts

Run scripts are the key files which are used by user to perform an interaction with the tool. They are located in **run_scripts** folder. Each run script file name is constructed in this way (except the **run_analyst_tool_setting.py**):

- **run_** prefix

- **IMAGE** or **CLASSIFIER**. **IMAGE** symbolizes operations related to image processing. **CLASSIFIER** represents operations related to classifiers of VR service.

- script's own name

Here we can see list of all run scripts:

- **run_ANALYST_TOOL_SETTING.py**: Needs to be run once after the VR Analyst Tool is installed. Sets used API version and API key.

- **run_IMAGE_download_images.py**: Downloads images from given file with URL links.

- **run_IMAGE_check_directory.py**: Checks images properties in given folder if they are suitable for VR service.

- **run_IMAGE_process_images.py**: Performs image processing described in given configuration file.

- **run_IMAGE_generate_negative_samples.py**: Generated negative images for a tiled classifier.

- **run_CLASSIFIER_classifiers_list.py**: Lists all trained custom classifiers in VR service.

- **run_CLASSIFIER_delete_classifier.py**: Deletes trained custom classifier from VR service.

- **run_CLASSIFIER_sets_creation.py**: Creates training and testing dataset for custom classifier.

- **run_CLASSIFIER_train_classifier.py**: Trains custom classifier.

- **run_CLASSIFIER_classify_images.py**: Classify images with a particular custom classifier.

- **run_CLASSIFIER_classify_cropped_images.py**: Classify only an particular area in the image.

  It can be used for following use cases:

  - **Inspection of particular areas of the product on production line**

Figure 3.1: Inspection of car engine on particular areas and checking if components align well

- **run_CLASSIFIER_measure_performance.py**: Measures performance of trained custom classifier on testing dataset.

- **run_CLASSIFIER_sort_images_based_on_classification.py**: Splits classified images into class folders based on classification results.

- **run_CLASSIFIER_update_classifier.py**: Updates already trained classifier.

- **run_CLASSIFIER_classify_tiled_images.py**: Classify tiled images.

  It can be used for following use cases:

  – **Detection of some visual aspect in high resolution images**

Figure 3.2: Detection of corrosion on power pole cables

Figure 3.3: Detection of broken warranty seals in car engines

- **run_CLASSIFIER_recursive_classifier.py**: Classify images with a stack of classifiers.

  It can be used for following use cases:

  - **Dynamical detection of visual aspect of various sizes + less scoring needed**
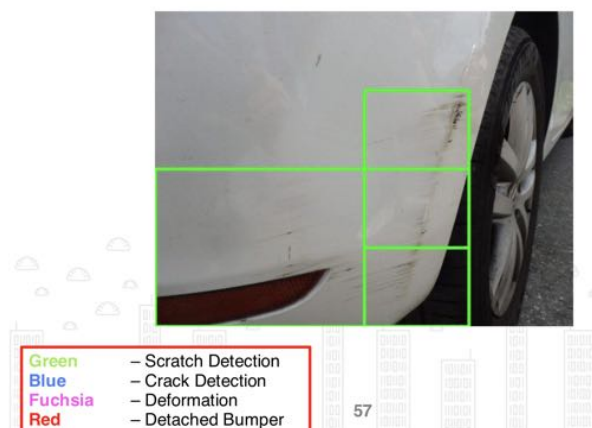

Figure 3.4: Detection of various defect types on car bumpers

Each run script should be run from the root VR Analyst Tool folder. So, each run script should be run in following manner:

`./run_scripts/run_script_name.py`

In addition you need to set a conf_file argument (if a run script demands it) representing path to conf file with json configuration of executed run script. In total, you should run run script in this way (if con file is required):

`python ./run_scripts/run_script_name.py --conf_file "/path/to/conf/file.json"`

3.2.1 JSON configs

You can find JSON configs templates in `config_templates`.

- **run_ANALYST_TOOL_SETTING.py**
  It has two mandatory keys:

  - `api_version`: Actual API version is `2018-03-19`. Value type is string.
  - `api_key`: VR service API key. Value type is string.

- **run_IMAGE_download_images.py**:
  It has four mandatory keys:

  - `output_folder`: Output folder where download the images. Value type is string
  - `url_file_path`: Path to text files storing urls of iamges. Each line in the file should represent one link. Value type is string.
  - `new_start`: When you start images downloading, it saves the actual progress. When you terminate the execution, you can continue downloading from last downloaded image. To continue downloading set value to 1. To start downloading from the beginning set value to 0.
  - `image_prefix` By default images are labeled by index, with prefix you can set a common prefix to all images. It is string, if you do not want to set a prefix, set empty string.

- **run_IMAGE_check_directory.py**:
  It has one mandatory key:

12

– `path` Path to analyzed directory. Value type is string.

- **run_IMAGE_process_images.py**:
  It has two mandatory keys:

  – `input_folder` Path to images folder which will be processed. Be careful, data processing overwrites original data, so for keeping original data, perform a backup. Value type is string.

  – `processing_tasks` Represents an array of processing tasks which should be performed on data in series. **Because some operations are not compatible, we recommend to apply only one image processing task in one run. To apply more image processing tasks, define multiple JSON configs and run them one by one.** Value type is an array of JSON objects.

  Now, we list JSON objects which could be put as a processing task:

  * **Standardize file names**
    The JSON object should have key `standardize_file_names` and the value is an empty JSON object.

    It renames all images with indices.

  * **Format Conversion**
    The JSON object should have key `convert_format` and the value is a JSON object with following key:
    · `format` String representing target format, values could be: `.jpg` or `.png`

  * **Resize to VR NN input resolution**
    This script resizes/squeezes images into resolution coming to the service - 224×224. The JSON object should have key `downsampleImageToNetworkSize` and the value is an empty JSON object.

  * **Augmentation by translation**
    It shifts image content randomly in x a y coordinates. The JSON object should have key `augmentation_translation` and the value is a JSON object with following keys:
    · `max_translation` Maximal pixels shift (+/-) in both axis from which the shift is generated. The value is an integer.
    · `min_translation` Minimal pixels shift (+/-) in both axis from which the shift is generated. The value is an integer.

- · `num_of_images` Number of images to generate. The value is integer.
- · `cropp_center` Image content shift causes that missing image areas are filled with black pixels. This is unwanted, so to avoid it we can crop the central part of the images. This parameter set the width and height of central part which is cropped. The value of the parameter is integer type. If you set 0 value, then no cropping is performed. Cropping is performed if parameter is bigger than zero.

* **Augmentation by rotation**
  It rotates image content randomly in x a y coordinates. The JSON object should have key `augmentation_rotation` and the value is a JSON object with following keys:
  - · `angle_step` It set rotation step. Value type of parameter is an integer and should be grated than 0.
  - · `samples_num` The value is an integer equal greater than zero. If the value is zero, then the augmentation is performed regarding the `angle_step` in regular constant step until 360 degrees rotation is performed. If the value is greater than zero, then `angle_step` is omited and it is generated `samples_num` new images with random angle rotation (positive or negative).
  - · `min_angle` It set minimal allowed angle which is randomly generated. It is valid when `angle_step` is grater than 0. The value is positive integer.
  - · `cropp_center` Image content rotation causes that missing image areas are filled with black pixels. This is unwanted, so to avoid it we can crop the central part of the images. This parameter set the width and height of central part which is cropped. The value of the parameter is integer type. If you set 0 value, then no cropping is performed. Cropping is performed if parameter is bigger than zero.

* **Images downsizing**
  It downsizes images to particular max size of max resolution. The JSON object should have key `downsample` and the value is a JSON object **which can have at least one of following keys:**
  - · `max_resolution` It set maximal allowed image resolution. The value of the key is a JSON with following keys:

· `width` Maximal allowed width of images. The value is a positive integer.
· `height` Maximal allowed height of images. The value is a positive integer.

item `max_size` It set maximal allowed image size in MB. The value of the key is a positive float.

– `Image Cropping` Image cropping crops areas from images on fixed position (It assumes that all images have same resolution). You can perform multiple crops at once, even crops can be overlaid. Each crop is defined by its name. This preprocessing is used for classification of image crops. Otherwise it has no sense. The JSON object should have key `cropping` and the value of the key is a JSON with following key:

  ∗ `crops` The value of this key is an array of JSON objects. Each JSON object in the array have to have following keys:
    · `position` The value is an array of four integers. It defines: x coordination. y coordination, width of crop, height of crop.
    · `name` It defined name of the crop. Value is an string.

– `Image Tiling` It defines tiling over the images. Image tiling is used when you want to use tiling classification of images. The JSON object have have key `tiling` and the value of the key is a JSON with following keys:

  ∗ `reference_image_tile` This key defined reference tiling mask which traverses over the image. The value is a JSON with two keys:
    · `width` Width of tiling mask. Value is positive integer.
    · `height` Height of tiling mask. Value is positive integer.
  ∗ `tile_variability` It defines ± percentual range derived from `reference_image_tile` in which the tiling mask will be sized. In other words it defines several tiling masks which will range from `reference_image_tile` $\times (1 - $ `tile_variability`$)$ to `reference_image_tile` $\times (1 + $ `tile_variability`$)$ with constant step. The parameter value is float. **Minimal value should be 0.5**
  ∗ `tile_translations` Tile translation define half mask shift for particular axis. By default no shift is done - tiling begins

15

in left upper corner. You can add shift in x axis, y axis, and in both axis. Shift means that tiling origin ill be shifted by half mask axis size. The value of the key is an array which could be empty or can have following values:

  · `x`

  · `y`

  · `xy`

- **run_IMAGE_generate_negative_samples.py**:
  It has three mandatory keys:

  - `input_folder` Input folder which images will be used for generation of negative class tiles. The value is a string.

  - `output_folder` Output folder where tiles will be placed. All tiles will be named by its index (order).

  - `tiling_conf` It represents configuration how tiles will be generated. The value of the key is a json object which has same structure/keys/values as `tiling` in image processing run script.

- **run_CLASSIFIER_classifiers_list.py**:
  It has no config file.

- **run_CLASSIFIER_delete_classifier.py**:
  It has two keys, but only one can be presented in the JSON:

  - `classifier_id` ID of the classifier wished to delete.

  - `classifier_name` Name of the classifier wished to delete. If there more classifiers of the same name, the first classifier which API retrieves is deleted.

- **run_CLASSIFIER_sets_creation.py**:
  It has two mandatory keys:

  - `output_path` Output path where sets will be created. This method does not overwrite original data. The value is string.

  - `classes` The value is an array of JSONs. Each JSON in array corresponds to one class. Number of array elements depends on if you plan to train binary of multiclass classifier. If you have

binary classifier, then the array has only one element. If you have multiclass classifier then the array has at least two elements. each array element (JSON) has follwoing keys:

* `percentages` Percentages is an array of three float elements. The sum of the array has to be 1.0. the first element represents percentage of data used for training dataset, the second one for testing dataset and the third one for validation dataset. Percentage of validation dataset could be 0, other ones has to be greater than zero.

* `path` It represents path to images folder which will be used for the classifier data split.

– `negative_class` This key is defined only when negative class is planned to use. It is a JSON object with following keys:

* `percentages` Percentages is an array of three float elements. The sum of the array has to be 1.0. the first element represents percentage of data used for training dataset, the second one for testing dataset and the third one for validation dataset. Percentage of validation dataset could be 0, other ones has to be greater than zero.

* `path` It represents path to images folder which will be used as negative class data split.

- **run_CLASSIFIER_train_classifier.py**:
  JSON object has following keys:

  – `classes` It is an array of JSON object. One element of the array represents one class in the classifier. Each class JSON has to have following keys:

  * `path` Path to the training data folder. If you used run script for sets creation, you should specify path to `Train` folder.

  * `class_name` Name of new class.

  – `negative_class` If you want to use negative class you have to specify this key, otherwise not. The value is a JSON object with following key:

  * `path` Path to negative class training data (`Train` folder in created negative set).

– `classifier_name` The value is a string representing a new classifier name.

- **run_CLASSIFIER_classify_images.py**:
  It has following keys:

  – `input_folder` Input images folder which will be scored. The value is string with images folder path.

  – `output_folder` Output path where scoring data will be stored. The value is string.

  – Now there has to be a classifier info which we want to use for scoring. It has to be `classifier_id` or `classifier_name`, not both. The value is string.

  – `file_prefix` It defines a prefix of generated csv file. The value is a string. If you left an empty string, the prefix of the generated csv file will be date and time of running the script. Otherwise it will be the prefix which you define.

- **run_CLASSIFIER_classify_cropped_images.py**:
  It has three keys:

  – `input_folder` Path to folder with cropped images (you need to use cropping preprocessing on these images).

  – `output_folder` Path to folder where output images will be generated. It generates original images and covers the particular areas with color if the classifier sees the visual aspect.

  – `layers` The value is an array which elements are JSON. Each element represents one classifier which will applied on particular crops, so you can have several classifiers applied on one image and can inspect various/same crops. Keys are:

    * It has to have `classifier_id` or `classifier_name` key. Not both. **Classifier has to be a binary one**.

    * `crops` It is an array of crop names. Crop name is an identifier of which crop should be used. It has to match to any crop name used in image crop preprocessing.

    * `threshold` Threshold defines minimal score that crops have to achieve to say that visual aspect is presented.

* `color` The value is an array of strings representing colors. The length of the array has to be same as number of crops. Values can be: `red`, `green`, `blue`, `fuchsia`.

- **run_CLASSIFIER_measure_performance.py**:
  It has to have following keys:

  - `output_folder` Output folder where statistics will be generated.

  - It has to have `classifier_id` or `classifier_name` key. Not both.

  - `classes` It is an array of JSON objects. Each element represents one class defined in the measured classifier. It has to have following keys:

    * `path` Path to class images which will be used for performance measurement. If you use sets creation run script, then the path should target `Test` set or validation `Validation`.
    * `class_name` Name of the class.

  - `negative_class` This key is optional. The value is a JSON with one key:

    * `path` Path to negative class images which will be used for performance measurement. If you use sets creation run script, then the path should target `Test` set or validation `Validation`.

- **run_CLASSIFIER_sort_images_based_on_classification.py**:
  It has following keys:

  - `scored_images_csv` It defines path to the generated scoring csv file from classify images run script.

  - `output_folder` Folder path where results will be placed.

  - `images_folder` Folder path where are the scored images from csv file.

  - `threshold` The value is a float from 0 to 1 which represent the threshold regarding negative class.

- **run_CLASSIFIER_update_classifier.py**:
  It has following keys:

– It has to have `classifier_id` or `classifier_name` key. Not both.

– `negative_class` This key is **optional**. If you do not want to update negative class, avoid it. The value is a JSON with following key:

  * `path` Path to folder with images which will be used to update negative class.

– `classes` It is an array of JSON objects. Each element represents one class which you want to update with following keys:

  * `path` Path to images which will be used to update the class.
  * `class_name` Name of the class which want to update.

- **run_CLASSIFIER_classify_tiled_images.py**:
  The JSON has three keys:

  – `input_folder` Path to folder with **tiled** images generated with preprocessing.

  – `output_folder` Output folder where results will be stored.

  – `layers` This define layers which will be executed in series. Which means, that you can stack multiple classifiers in chain. The value is a JSON with following keys:

    * It has to have `classifier_id` or `classifier_name` key. Not both. It defines which classifier will be used in particular layer. **It has to be a binary classifier**. `filter` This is is **optional**. if you do not set it, then all tiles with score above threshold are processed and put into next layer. If you set the key, there is only one value which can be used now - `maximum`. `maximum` means that only the tile with the highest score will flow into next layer (one for each tiling mask shift), all other tiles will be thrown away. It can be useful, when you know that the object has to be presented somewhere and only once, thus you want extract only the tile with highest score. `threshold` Minimal allowed score threshold to say that the object is there. `color` This key is **optional**. If you do not want to visualize the layer (mostly we visualize the last layer only), you can omit the key. The value is an array with one element representing color name. It can have following values: `red`, `green`, `blue`, `fuchsia`

- **run_CLASSIFIER_recursive_classifier.py**:
  It has following keys:

  - **input_folder** Path to folder with images to be analyzed. No preprocessing is needed like for tiling.

  - **output_folder** Output folder where results will be stored.

  - **layers** Layers represents a oriented graph with conditions how to move between them and actions which are performed if conditions pass or fail.

    * **classifier_ids** or **classifier_names** has to be specified. Not both. The value is an array of classifier names or ids which will be used in particualar layer.
    * **results_processing** This key defines config how results will be processed in the layer. The value is JSON with following keys:
      · **pass** It defines what will be done if condition pass for particular subimage. The value is JSON with following keys:
      · **handle_tiles** The value is string with one option now - **all**. It means that all subimages which pass the criterion in particular layer will be processed.
      · **handle_classifiers** The value is a string with two options - **all** or **max**. **all** means that all classifiers will be handled (if they pass the condition) on particular subimage. **max** means that the classifier with max score for particular subimage will be handled (if it pass the condition) on particular subimage.
      · **do_layer** defines which layer will be performed as a next if the condition pass for a particular classifier for particular subimage. Layers indices begin with 0. You can set -1 to omit any other layer execution if the condition passes. If you set the index of actual layer, then you can create a recursive loop. The number of elements has to math number of used classifier in particular layer. Each classifier can jump to different layer.
      · **criterion** Defines the criterion/condition which defines which branch will be executed for particualr subimage

- `pass` or `fail`. The value is JSON **can have** following keys:
- · `minimal_thresholds` This sets the score threshold which is needed to pass by the classifier to say that the criterion is passed. The value is an array with float thresholds. The length has to math the classifiers number in the layer. **Can not be set with with** `keep_growing` **key**
- · `keep_growing` This sets that the criterion is fullfiled for particular classifier if the score for particular subimage is bigger than the score for subimage parrent. It is used in quarters division. **Can not be set with with** `minimal_thresholds` **key**
- · `do_layer_on_chain_break` It means to which layer has to be jumped if we have series of successfully passed criterions (series parents subimages) and suddenly the chain is broken. Please do not confuse it with `fail` execution, it handles something different. The value is an array of layers indices. The array length has to match number of used classifiers in the layer. You can set -1 to perform no jump for particular classifier.
- · `do_output` It defines what will be done, if the particular classifier pass a criterion on particular subimage. The value is an array of JSON objects which can be empty, if you dont want to do an output or have some of following keys:
- · `text` Which defines a prefix for results logging for particular classifier.
- · `color` Which defines a color markup in the image. Values can be: `red`, `green`, `fuchsia`, `blue`, `orange`
- · `pass` It defines what will be done if condition fails for particular subimage. The value is JSON with following keys:
- · `handle_tiles` The value is string with one option now - `all`. It means that all subimages which fail the criterion in particular layer will be processed.
- · `handle_classifiers` The value is a string with two options - `all` or `min`. `all` means that all classifiers will be handled (if they fail the condition) on particular subimage. `min` means that the classifier with lowest score for

22

particular subimage will be handled (if it fail the condition) on particular subimage.

- · `do_layer` defines which layer will be performed as a next if the condition fail for a particular classifier for particular subimage. Layers indices begin with 0. You can set -1 to omit any other layer execution if the condition passes.

- · `do_output` It defines what will be done, if the particular classifier fail a criterion on particular subimage. The value is an array of JSON objects which can be empty, if you dont want to do an output or have some of following keys:

- · `text` Which defines a prefix for results logging for particular classifier.

- · `color` Which defines a color markup in the image. Values can be: `red`, `green`, `fuchsia`, `blue`, `orange`

- · `image_division` This key defines how we want to analyze the image. The value is JSON with following keys:

- · `type` It defines the type of image handling there are two string values which could be set.`none` to keep the original image in each layer execution or `halves` which divides the image into quarters in each layer execution. So, you start with the original image and divide it recursiverly into smaller pieces.

- · `stop_criterion` You can **optionaly** set stop criterion which is valid for `halves` and stops image division if quarters resolution would be smaller that the set one. If stop criterion is executed, then the `do_layer_on_chain_break` execution is called. The value is an array of two integers with width and height of the minimal subimage size.

3.2.2 Code Organization

Now lets see how run scripts are code organized.

Each run script is consisted of two code segments:

- Setting cmd arguments parser (this routine should be copied and pasted in new custom run scripts if conf file is required)

- Construction of Analyst Tool object and calling particular "run script" method and passing the JSON conf file object into (if run script demands it).

3.3   Other module

- `aux_functions.py` file which contains functions which are auxiliary for the tool and does not make sense to create a separate objects or files for them. Here you can add more aux functions.

3.4   Output Writer module

- `output_writer.py` This file defines `OutputWriter` object which is used to perform writing output into a target and define interface for output writing. However what is printed and the way how does it prints depends on the writer type.

  - `printOutputSegmentDelimiter` Prints new lines and ascii graphics representing new logical code execution segment.

  - `outputWarningDelimiter` Prints new lines and ascii graphics representing warning message.

  - `outputErrorDelimiter` Prints new lines and ascii graphics representing error message.

  - `printErrorText` Prints error text message, which is method input parameter.

  - `printWarning` Prints warning text message, which is method input parameter.

  - `writeTextOutput` Prints text message, which is method input parameter.

  - `setNewRecord` Writes new lines to divide logical text outputs.

  - `writeFormatedTextOutput` Writes formatted text with specific width. Its input parameters are printed text and alignment width.

- `writers.py` This file defines writers which are actually performing writing defined by interface in `OutputWriter`. There are two writers defined - console and text writer. You can create more writers types on your own. Just needs to have same methods definition as `FileWriter` and `ConsoleWriter`.

3.5   Argument Parser module

This module analyze and define arguments for the Analyst Tool. No modification is needed to do in this module

3.6   Analyst Tool Objects module

This is the key module containing all important object for run of the VR Analyst Tool. If you want to create a new run script, it should call a new methods of `AnalystTool object`. You can find at the end of class commented part representing a sample code how new method should be implemented.

All managers (object instances handling particular domain) are initialized at the constructor of `AnalystTool` object, thus if you need to define a new manager, it should be initialized in the similar way as already existing.

3.6.1   Visual Recognition object

Object `VisualRecognitionManager` defines an interface how programmer can work with the VR Service. You can check the interface in file `visual_recognition_manager.py`. This manager leverages class `VisualRecognition` located in `visual_recognition.py` which implements wrapper around the object `VrPythonApi` located in `vr_python.py` which calls WDC VR API SDK primitives.

3.6.2   UrlChecker object

This objects is responsible for images downloading from URLs. In other words it is used by run script `run_IMAGE_download|_images.py`.

3.6.3   TileInferenceManager object

This object is located in `tiling_inference_manager.py` and it is responsible for handling tiling method. It encapsulates tiling inferencing and how process

and merge tiles results. It leverages `TileOperator` located in `tile_operator.py` which is responsible for metadata stored in tiled images.

3.6.4   StatisticsManager object

This object is located in `statistics_manager.py` and it is responsible for all metrics done during performance measurement. If you need add new performance metrics, focus on this module.

3.6.5   ImagesDownloader object

It is located in `images_downloader.py` and it is responsible for process of downloading images from URL

3.6.6   UrlChecker object

This objects is responsible for URLs checking. It is used by run script `run_IMAGE_download_images.py`.

3.6.7   RecursiveClassifier object

It is located in `recursive_classifier.py` and it is responsible for the process of recursive classification.

3.6.8   ImageProcessor object

It is located in `image_processor.py` and it is responsible for performing image processing tasks. If you want to implement a new image processing task, it should be here.

3.6.9   FilesMaanager object

It is located in `files_manager.py` and it is responsible for filesystem and data manipulation.

3.6.10   CSVManager object

It is located in `csv_manager.py` is responsible for obtaining particular csv operators located in `csv_operators`.

3.6.11 CropInferenceManager object

This object is located in `crop_inference_manager.py` and it is responsible for handling cropping method. It encapsulates crops inferencing and how process and merge crops results. It leverages `CropOperator` located in `crop_operator.py` which is responsible for metadata stored in cropped images.

Chapter 4

Labs

In this lab we will go thru all important features of VR Analyst Tool and demonstrate it on one use case. For this lab we chose a biological cell image. Lets see the image:
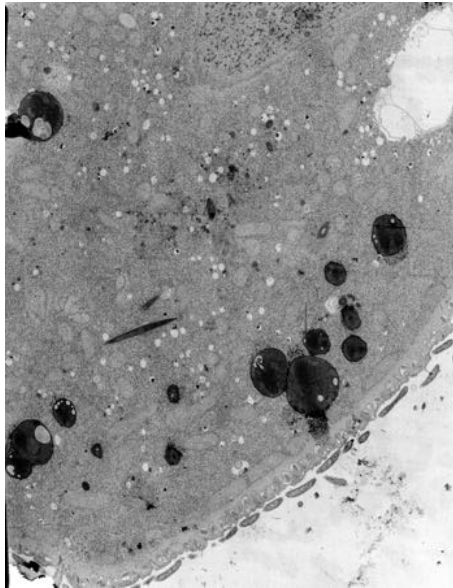
We can see that there are lot of details. The original image size is 3101×4000 and its size is 37 MB.

Lets define our goal for the lab! We would like to detect a small air/fluid capsule inside a black subcellular components (4.1).
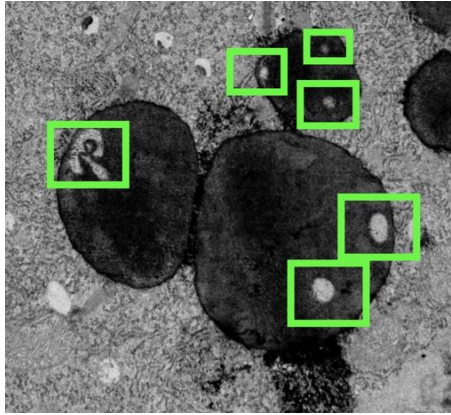
Figure 4.1: Highlighted air/fluid capsules inside a black subcellular components.

Now there is a question how to do that. We can not directly push image into VR Service, because it is designed for image recognition. In addition we need to keep in our mind a fact that each image coming into the service in downscaled into 224×224 pixels. Thus lot of details of high resolution images is lost.

There has been predefined

You can leverage for the lab a folder hierarchy which was set in VR Tool directory. It is called `lab`. You can find here predefined folders which should help you during the exercise, but you can definitely create your own hierarchy and some different location of filesystem. Lets take a look on a `lab` folder:
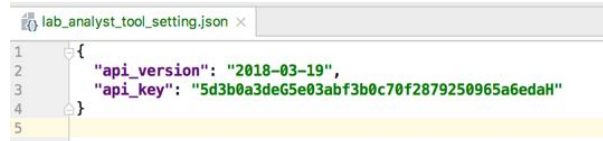
- `json_configs` Here should be put all config files for run scripts

- `data/source_image` Here you can find source image which we are working on

- `data/downloaded_source_image` Here you should download a source image

4.1 Setting VR Analyst Tool

As a first thing we need to setup VR Analyst Tool which means that we need to set API key and API version.
Firstly, you need to create a config JSON file. You can write a new one from

scratch or reuse JSON template file (`template_json_analyst_tool_setting.json`) located in folder `config_templates`.No matter which option you choose, as a result you should obtain JSON file with this structure:

Now, we we are ready to run the particular script which set the tool with this command:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_ANALYST_TOOL_SETTING.py
--conf_file "./lab/json_configs/lab_analyst_tool_setting.json"
```

where `--conf_file` should be the path to the created conf JSON file. **Note: You should run all scripts with Python 3 interpreter.** After the command is executed, you should see following output:

```
##########################################
Setting VR Analyst Tool ...
VR Analyst Tool was successfully set.
##########################################
```

Now you have successfully set VR Analyst Tool.

4.2   Downloading the cell image

Now we need to download the cell image. We do not need to do that manualy, VR Analyst Tool has capability to download series of images from URL links. You just need to put links into text file. For this task we prepared a text file with cell source image URL - it is located in `/source_image_url.txt`. As a backup you can find the image in `data/source_image`. To download the image we need to create an another JSON config, so lets do that. You can reuse template `template_json_download_images.json` from `config_templates`. In the end you should have a JSON with following structure:

```json
{
    "output_folder": "./lab/data/downloaded_source_image",
    "url_file_path": "./lab/data/source_image_url.txt",
    "new_start": 1,
    "image_prefix": ""
}
```

For better insight regarding meaning of key words, take a look into API doc part. Now, we we are ready to run the particular script which set the tool with this command:

```
[Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_IMAGE_download_images.py
    --conf_file "./lab/json_configs/lab_download_source_image.json"
```

where `--conf_file` should be the path to the created conf JSON file. After the command is executed, you should see following output:

```
#########################################
Setting VR Analyst Tool ...
VR Analyst Tool was successfully set.
#########################################
#########################################


Downloading images from URLs ...


Already processed 0.0000 % URLs.
- Processing URL index:  0
- Processing URL:        https://cildata.crbs.ucsd.edu/media/images/38862/38862.tif


ALL FILES HAS BEEN DOWNLOADED!
Downloaded 1(100.00 %) valid URLs.


#########################################
```

Now we have downloaded the source image. We can proceed to processing it.

4.3   Image conversion

We can see that the image is in `.tif` format which is not supported by the VR Service (`.jpeg`,`.png`). For this reason, we perform a preprocessing - conversion into `.png`. To process the image we need to create an another JSON config, so lets do that. You can reuse template `template_json_process_images.json` from `config_templates`. In the end you should have a JSON with following structure:

```
{
  "input_folder": "./lab/data/downloaded_source_image",
  "processing_tasks": [{"convert_format": {"format": ".png"}}]
}
```

For better insight regarding meaning of key words, take a look into API doc part. Now, we we are ready to run the particular script which set the tool with this command:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_IMAGE_process_images.py
--conf_file "./lab/json_configs/lab_normalize_image.json"
```

where `--conf_file` should be the path to the created conf JSON file. After the command is executed, you should see following output:

```
###########################################
Setting VR Analyst Tool ...
VR Analyst Tool was successfully set.
###########################################
###########################################


Performing images processing ...


Already processed 0.0000 % images.
Processing image: 0.tif ...
- Converting image ...


Processing of images is completed!
###########################################
```

Check the directory, image should be converted from `.tiff` to `.png`.

4.4   Classifier for fluid/gas capsule

Now, we move forward and try to train a classifier for recognition of fluid/gas capsule. The question how should we do that. As a first task we need to

create crops of the capsules from the image. To avoid that task we offer capsules crops in folder `lab/data/fluid_gas_capsules`. Check them now.

We can see that there is limited amount of images. Now, there is a question how to augment the data and create a negative dataset.

4.4.1 Data augmentation

VR Analyst tool has several options how to augment the data. We will use augmentation by rotation and translation. **We recommend to copy the source directory with capsules crops and apply the augmentation on the copied folder, because it overwrites original data, so we have a backup if anything goes wrong.**

We perform the augmentation in two phases. In the first phase we augment the data by translation. To do that, you need to define an another JSON config file. You can reuse template `template_json_process_images.json` from `config_templates`. In the end you should have a JSON file with following structure:

```
{
  "input_folder": "./lab/data/fluid_gas_capsules_copy",
  "processing_tasks": [
    {"augmentation_translation": {"max_translation": 40, "min_translation": 10, "num_of_images": 4, "cropp_center": 0}}
    ]
}
```

For better insight regarding meaning of key words, take a look into API doc part. Now, we we are ready to run the particular script which set the tool with this command:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_IMAGE_process_images.py --
conf_file "./lab/json_configs/lab_fluid_gas_capsules_augmentation_translation.json"
```

You should see execution of script and it should end with following:

33

```
Already processed 46.1538 % images.
Processing image: capsule_6.png ...
- Augmenting images with translation ...

Already processed 53.8462 % images.
Processing image: capsule_7.png ...
- Augmenting images with translation ...

Already processed 61.5385 % images.
Processing image: capsule_5.png ...
- Augmenting images with translation ...

Already processed 69.2308 % images.
Processing image: capsule_4.png ...
- Augmenting images with translation ...

Already processed 76.9231 % images.
Processing image: capsule_1.png ...
- Augmenting images with translation ...

Already processed 84.6154 % images.
Processing image: capsule_3.png ...
- Augmenting images with translation ...

Already processed 92.3077 % images.
Processing image: capsule_2.png ...
- Augmenting images with translation ...


Processing of images is completed!
#########################################
```
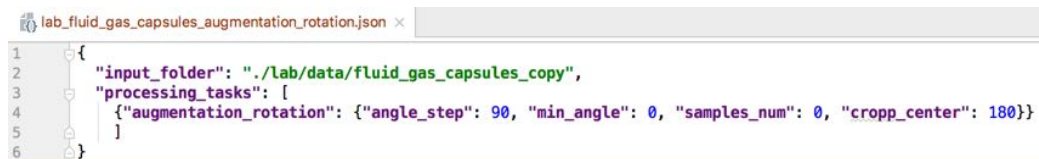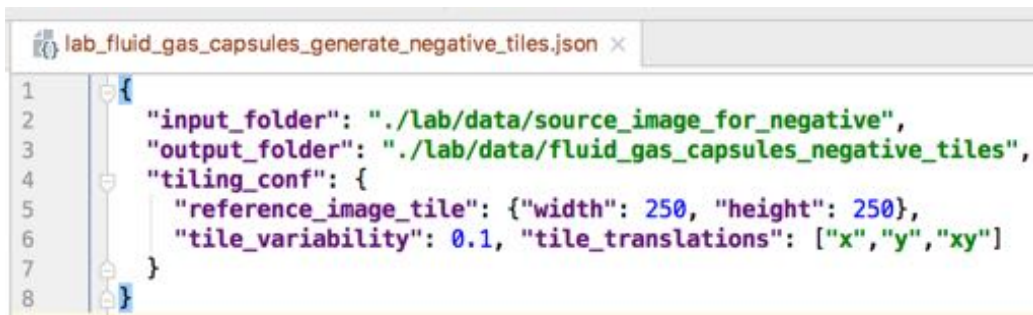
You can check the folder and see that data was augmented by random translation. Now we perform an augmentation by rotation. You need to create another JSON config or reuse the template and fill it with following:

```
lab_fluid_gas_capsules_augmentation_rotation.json ×
1  {
2    "input_folder": "./lab/data/fluid_gas_capsules_copy",
3    "processing_tasks": [
4      {"augmentation_rotation": {"angle_step": 90, "min_angle": 0, "samples_num": 0, "cropp_center": 180}}
5    ]
6  }
```

For better insight into JSON config check the API doc, but you can see that rotate images with 90 degree angle - we choose this angle because the capsules are near to each other in original image and we did not can crop a bigger tile with enough free space for different angle rotation (pixels out of

34

the image are filled with black color). To avoid out of image pixels caused by translation (with 90 degree rotation no black pixels are added) we crop the center of image which is set for our example to 180 pixels. Now we execute the command:

```
[Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_IMAGE_process_images.py  --]
 conf_file "./lab/json_configs/lab_fluid_gas_capsules_augmentation_rotation.json"▌
```

The script execution should end with following:

```
Already processed 93.8462 % images.
Processing image: capsule_11_translation__3.png ...
- Augmenting images with rotation ...


Already processed 95.3846 % images.
Processing image: capsule_1_translation__4.png ...
- Augmenting images with rotation ...


Already processed 96.9231 % images.
Processing image: capsule_11_translation__2.png ...
- Augmenting images with rotation ...


Already processed 98.4615 % images.
Processing image: capsule_2.png ...
- Augmenting images with rotation ...


Processing of images is completed!
#########################################
```

4.4.2   Generation of negative data set

Now we need to generate a negative dataset. Because we want to use tiling, we need to crop the source image into smaller tiles, similar to capsules tiles. We can do that manually or automatically with our tool. You just need to cover areas representing class data with some color. Check the modified source image in `data/source_image_for_negative`. You can see that we overlaid capsules with grey color.

Now we specify JSON file for this task
"(you can reuse template `template_json_generate_negative_dataset`). In the folder hierarchy we created a folder where you can target the output of the script. In the end the JSON files should look like: `data/fluid_gas_capsules_negative_tiles`:

35

Now we execute the script:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_IMAGE_generate_negative_samples_for_tiling.py
 --conf_file "./lab/json_configs/lab_fluid_gas_capsules_generate_negative_tiles.json"
```

and the script execution should end with following:

```
########################################
Setting VR Analyst Tool ...
VR Analyst Tool was successfully set.
########################################
########################################
Processing generating negative image samples ...


Performing images processing ...


Already processed 0.0000 % images.
Processing image: negative.png ...
- Cropping images into tiles ...


Processing of images is completed!


Moving images tiles into output folder:
########################################
```

Lets check the generated dataset. You have to create another JSON config or reuse template `template_json_check_directiory`. You should have following JSON file:

36

Run it with this command:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_IMAGE_check_directory.py
--conf_file "./lab/json_configs/lab_fluid_gas_capsules_negative_tiles_info.json"
```

and you should see following output:

```
FOLDER STATISTICS:
 - Number of files:                      2499

 - Number of recognized images:          2498

 - Number of recognized VR images:       2498

 - Number of recognized videos:          0

 - Number of folders with VR images:     0

 - Number of image tiles:                0

 - Number of image crops:                0

########################################
```

You can see that there were generated 2499 tiles, which is pretty high number.

4.4.3 Sets creation

Now we need to create training and testing dataset (we avoid validation dataset). To do that we need to define a JSON file (you can reuse template `template_json_sets_creation.json`). You can produce output in the prepared folder `fluid_gas_capsules_sets`. JSON file should look like:

```
lab_fluid_gas_capsules_sets_creation.json ✕
1   {
2     "classes": [
3       {
4         "percentages": [0.8, 0.2, 0.0],
5         "path": "./lab/data/fluid_gas_capsules_copy"
6       }
7     ],
8     "output_path": "./lab/data/fluid_gas_capsules_sets",
9     "negative_class": {
10      "percentages": [0.9, 0.1, 0.0],
11      "path": "./lab/data/fluid_gas_capsules_negative_tiles"
12    }
13  }
```

Now we execute the command:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_CLASSIFIER_sets_creation.py
 --conf_file "./lab/json_configs/lab_fluid_gas_capsules_sets_creation.json" ▊
```

and see following output:

```
FOLDER STATISTICS:
 - Number of files:                          2499

 - Number of recognized images:             2498

 - Number of recognized VR images:          2498

 - Number of recognized videos:                0

 - Number of folders with VR images:           0

 - Number of image tiles:                       0

 - Number of image crops:                       0


All sets were created succesfully ...
############################################
```

You can check created dataset structure now.

4.4.4   Classifier training

Now we train the classifier for gas/fluid capsules. To do that we need to specify a JSON file (you can reuse `template_json_train_classifier.json`) and define as following.**Pay attention, sou need to specify path to** `Train` **dataset**:
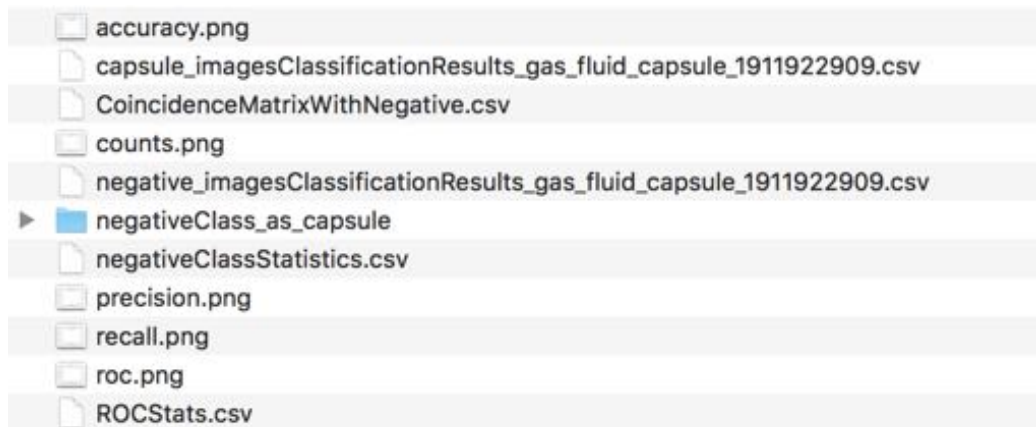
```json
lab_fluid_gas_capsules_train_classifier.json ×
1  {
2      "classes": [
3          {
4              "path": "./lab/data/fluid_gas_capsules_sets/fluid_gas_capsules_copy/Train",
5              "class_name": "capsule"
6          }
7      ],
8      "classifier_name": "gas_fluid_capsule",
9      "negative_class": {"path": "./lab/data/fluid_gas_capsules_sets/fluid_gas_capsules_negative_tiles/Train"}
10 }
```

Now we run the script:

```
[Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_CLASSIFIER_train_classifier.
 py  --conf_file "./lab/json_configs/lab_fluid_gas_capsules_train_classifier.json"  ▊
```

Now a classifiers starts to train. It can take lot of time (hour) and the computer should not fall into sleep during the process.

After a training is complete, we check whether a classifier is correctly trained. We can do that with follwing command:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_CLASSIFIER_classifiers_list.py▊
```

You should see classifiers listing and try to find the trained classifier. Find the trained classifier and you should see following:

```
gas_fluid_capsule
- classifier ID:        gas_fluid_capsule_1911922909
- status:               ready
- date created:         2018-07-27T21:07:37.798Z
- classes:

                        capsule
```

We can see that the classifier was correctly trained.

4.4.5   Performance measurement

Now we need to measure the performance of the classifier on a Testing dataset. You need to create another JSON file or reuse template (`template_json_measure_performance.json`). You need to specify id of the trained classifier and target folder (there is prepared folder `fluid_gas_capsules_performance`). You need to specify path to `Test` dataset. The JSON file should look like this:

```
    lab_fluid_gas_capsules_performance.json  ×
1    {
2       "classes": [
3         {
4           "path": "./lab/data/fluid_gas_capsules_sets/fluid_gas_capsules_copy/Test",
5           "class_name": "capsule"
6         }
7       ],
8       "negative_class": {"path": "./lab/data/fluid_gas_capsules_sets/fluid_gas_capsules_negative_tiles/Test"},
9       "classifier_id": "gas_fluid_capsule_1911922909",
10      "output_folder": "./lab/data/fluid_gas_capsules_performance"
11    }
```

Then run it with:

```
Ondrejs-MBP:AnalystTool ondra$ python3 ./run_scripts/run_CLASSIFIER_measure_performance.py
conf_file "./lab/json_configs/lab_fluid_gas_capsules_performance.json"
```

It will generate statistics inside target folder. Now, check the statistics. It should generate following files:

```
     accuracy.png
     capsule_imagesClassificationResults_gas_fluid_capsule_1911922909.csv
     CoincidenceMatrixWithNegative.csv
     counts.png
     negative_imagesClassificationResults_gas_fluid_capsule_1911922909.csv
 ▶   negativeClass_as_capsule
     negativeClassStatistics.csv
     precision.png
     recall.png
     roc.png
     ROCStats.csv
```

Check the results, meaning of these statistics could be found on the internet. We tested the performance on testing data, now lets see how the classifier will work on tileld image.

4.4.6   Testing the classifier

There is a folder `/data/testing_images` which contains two images - one used for classifier training and the other unseen image. Lets apply the classifier on the images. To do that we need to perform tiling over those images.

You need to specify a JSON file (you can reuse `template_json_process_images.json`) with the following content. To spare a time, we perform only shift in both x and y axis and drop variance in tile window. Be careful, tiling image processing will overwrite original data, to avoid that copy images to prepared folder (`data/fluid_gas_capsules_images_tiling`) or somewhere else.

41

```
lab_fluid_gas_capsules_tiling.json  ×

{
    "input_folder": "./lab/data/fluid_gas_capsules_images_tiling",
    "processing_tasks": [
        {"tiling": {"reference_image_tile": {"width": 250, "height": 250}, "tile_translations": ["xy"]}}
    ]
}
```

Now run the script:

```
Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_IMAGE_process_images.py
--conf_file "./lab/json_configs/lab_fluid_gas_capsules_tiling.json"
```

After the processing is complete, check the folder, instead of images there should be folders with image tiles.

Now we need to apply the classifier on the tiled images. To do that, we need to define a JSON file (you can reuse template `template_json_classify_tiled_image.json`). There is prepared folder regarding the classification outputs - `/data/fluid_gas_capsules_images_tiling_output`. In the end the JSON should look like as following:


```
lab_fluid_gas_capsules_tiling_classification.json  ×

{
    "input_folder": "./lab/data/fluid_gas_capsules_images_tiling",
    "layers": [{"classifier_name": "gas_fluid_capsule", "threshold": 0.7, "colors": ["green"]}],
    "output_folder": "./lab/data/fluid_gas_capsules_images_tiling_output"
}
```

Now execute the command:

```
Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_CLASSIFIER_classify_tiled_images.py
[--conf_file "./lab/json_configs/lab_fluid_gas_capsules_tiling_classification.json"
```

Now, you should see images scoring results. After the script is completed you should see in output folder two images covered with tiles (tile represents classification match). It should look like following:
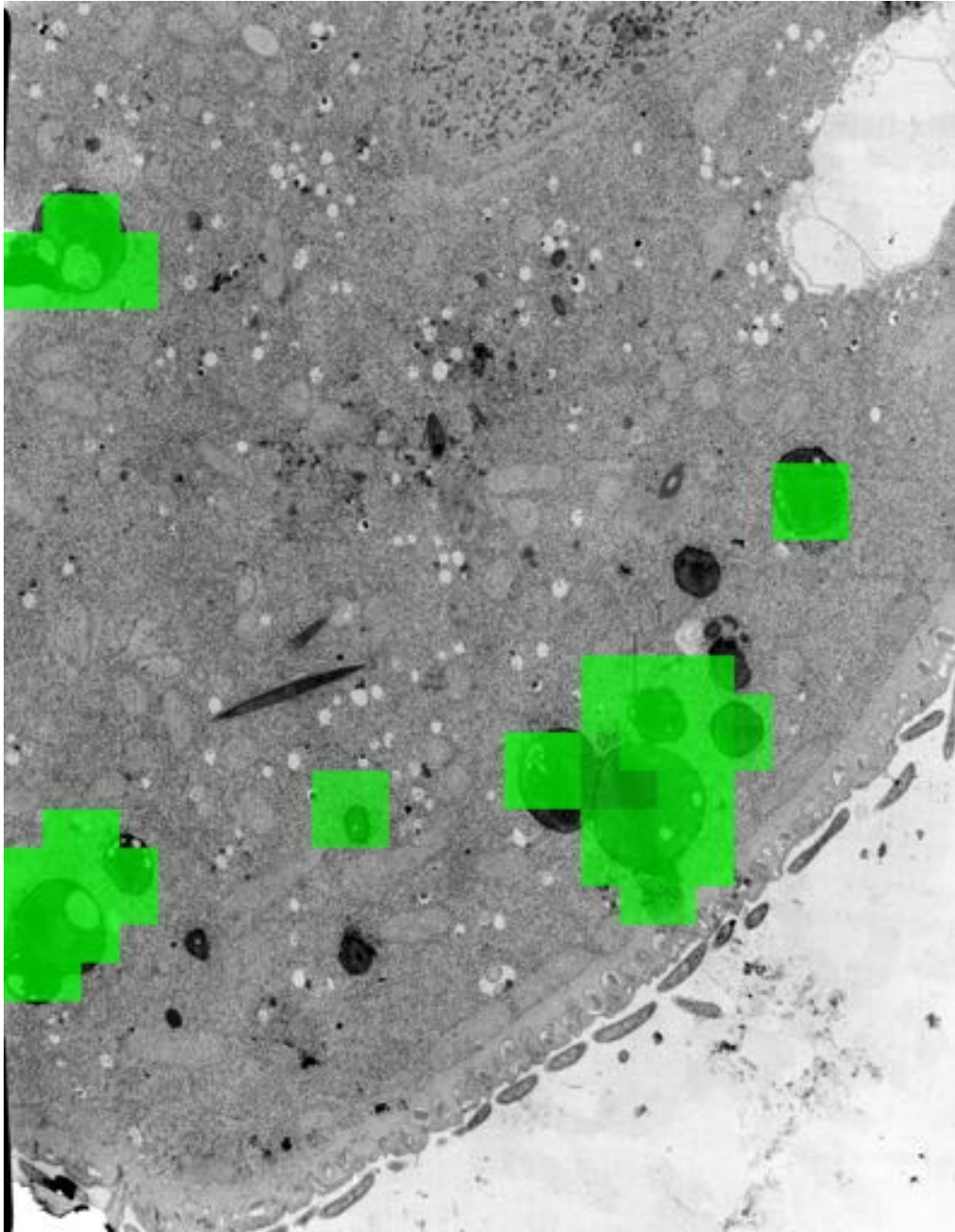
Figure 4.2: Training image
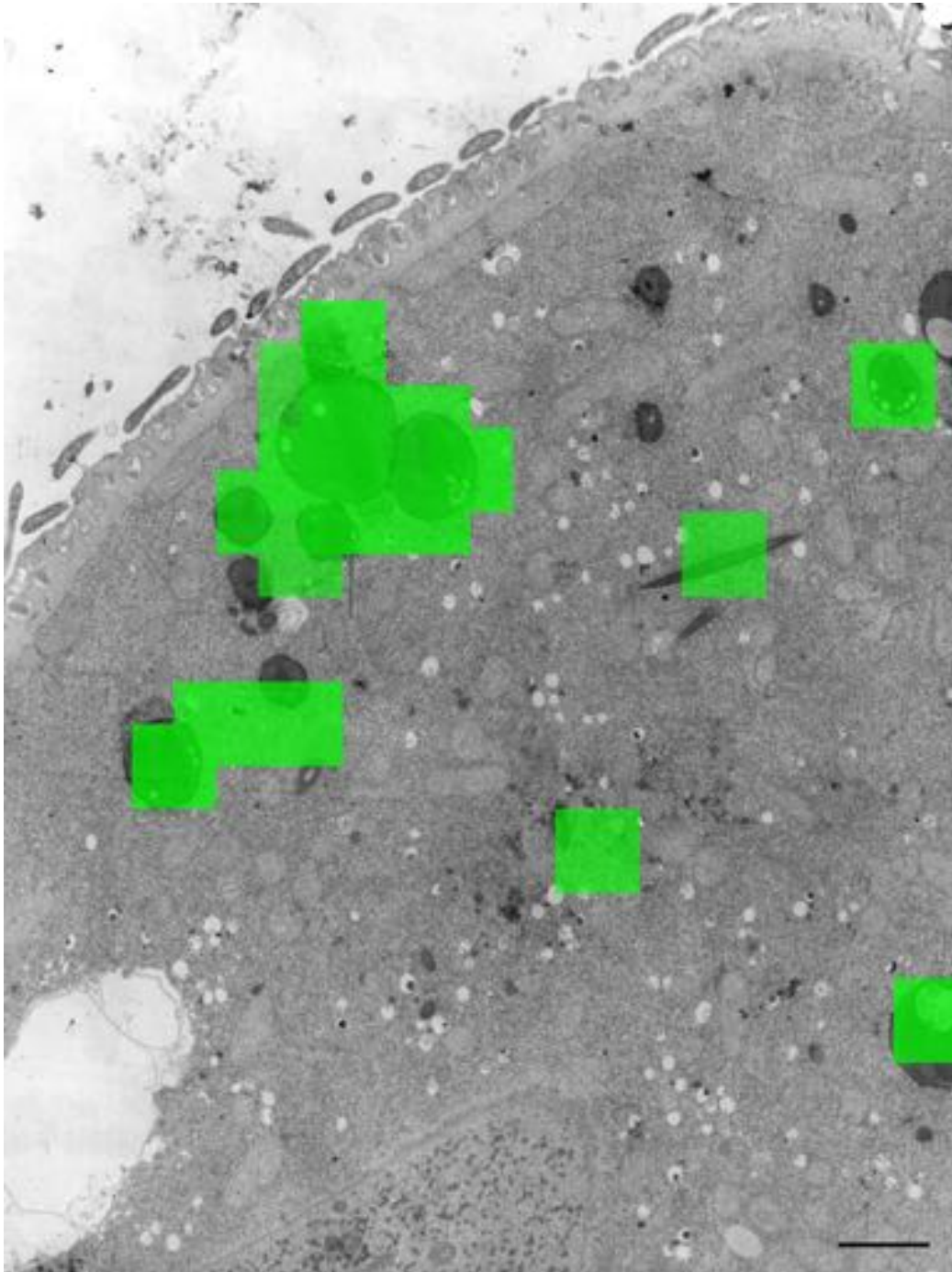
Figure 4.3: Testing image

Regarding the results we can see that the classifiers are working well even on unseen image, even though that the histogram is not equalized.

4.5 Recursive Classifier

On the other hand we see that to process one image we need in total circa 200 images scoring + another 200 scoring for tile window shift in x and y axis. This might inefficient. To avoid this there is a way called recursive classification.

The basic idea is recursive division of input image. We divide an input image into quarters, classify each quarter and based on the score + demanded margin we go deeper and divide the quarter into quarters. This is repeated while the resolution is bigger than some minimal res + score is increasing in each repetition. This process dynamically select areas which are suspicious from recognized object presence + reduce number or acted scoring + is able to dynamically recognize object in various sized without fixed tiling. When the chain rule is broken - we zoomed to the object enough, we can apply another classifiers.

In this lab we use the method for recognition of black subcellular components. To do that we need a training dataset containing crops of black components with various fields of view. Take a look on prepared crops in `data/black_component_crops`.

4.5.1 Data augmentation

Now we need to augment the data. We will use only rotation by 90 degrees. Please copy the images to preserve the original ones, you can use predefined folder (`data/black_component_crops_copy`). The configured JSON should look like:

```
black_components_augmentation_rotation.json ×
{
  "input_folder": "./lab/data/black_component_crops_copy",
  "processing_tasks": [
    {"augmentation_rotation": {"angle_step": 90, "min_angle": 0, "samples_num": 0, "cropp_center": 0}}
  ]
}
```

and run it with:

```
Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_IMAGE_process_images.py
--conf_file "./lab/json_configs/black_components_augmentation_rotation.json"
```

You should see processing images. After the script is finished lets proceed to the generation of negative tiles.

4.5.2   Negative tiles generation

To do that, you need to specify JSON file or reuse the template, like for the previous classifier. You need to set a target folder, you can use folder data/black_component_crops_negative_tiles.

and run

```
Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_IMAGE_generate_negative_samples_for_tiling.py
--conf_file "./lab/json_configs/black_components_generate_negative_tiles.json"
```

4.5.3   Sets creation

Now we need to create datasets split. Lets do it in same way as for last classifier. We can use predefined folder data/black_component_sets.

and run it

```
Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_CLASSIFIER_sets_creation.py
--conf_file "./lab/json_configs/black_components_sets_creation.json"
```

You should see in the target folder created datasets.

4.5.4   Classifier Training

Now we should train the classifier for black components. Specify the JSON file like this. Be careful, you need to specify path to **Train** dataset:

and run it with following command. The training will take about one hour or more.

```
Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_CLASSIFIER_train_classifier.py
--conf_file "./lab/json_configs/black_components_train_classifier.json"
```

When the training is complete, we need to proceed to performance measurement.

4.5.5   Performance measurement

Now, we need to measure the performance of the trained classifier. To do that we need to specify JSON file. There is created folder for storing performance results - `black_component_performance`.

and run it

```
[Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_CLASSIFIER_measure_performance.py
[ --conf_file "./lab/json_configs/black_components_performance.json"
```

Check the results in the folder.

4.6   Recursive classification

Now we will try to recursively classify testing images with black component classifier. Define a JSON file as following or reuse a template (`template_json_recursive_classif`

```json
{
    "input_folder": "./lab/data/testing_images",
    "layers": [
        {
            "classifier_names": ["black_component_detection"],
            "results_processing": {
                "pass": {
                    "handle_tiles": "all",
                    "handle_classifiers": "all",
                    "do_layer": [0],
                    "criterion": {"keep_growing": ""},
                    "do_output": [
                        {"text": "Suspicious area: ", "color": "red"}
                    ]
                },
                "fail": {
                    "handle_tiles": "all",
                    "handle_classifiers": "all",
                    "do_layer": [-1],
                    "do_output": [-1]
                }
            },
            "image_division": {"type": "halves"},
            "stop_criterion": {"min_res": [100,100]}}
    ],
    "output_folder": "./lab/data/black_component_recursive"
}
```

and run it with

```
[Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_CLASSIFIER_recursive_classifier.py]
    --conf_file "./lab/json_configs/black_components_recursive_classifier.json"
```

It will start analysis of images. In console you can see actual square which is processing by its coordinates and layer which is executed on it.When script is done you should see following files in output directory.

training_image.png.txt
training_image.png
testing_image.png.txt
testing_image.png

In txt file you can find scores for each layer for particular square. More important are images. You should obtain similar images like these:
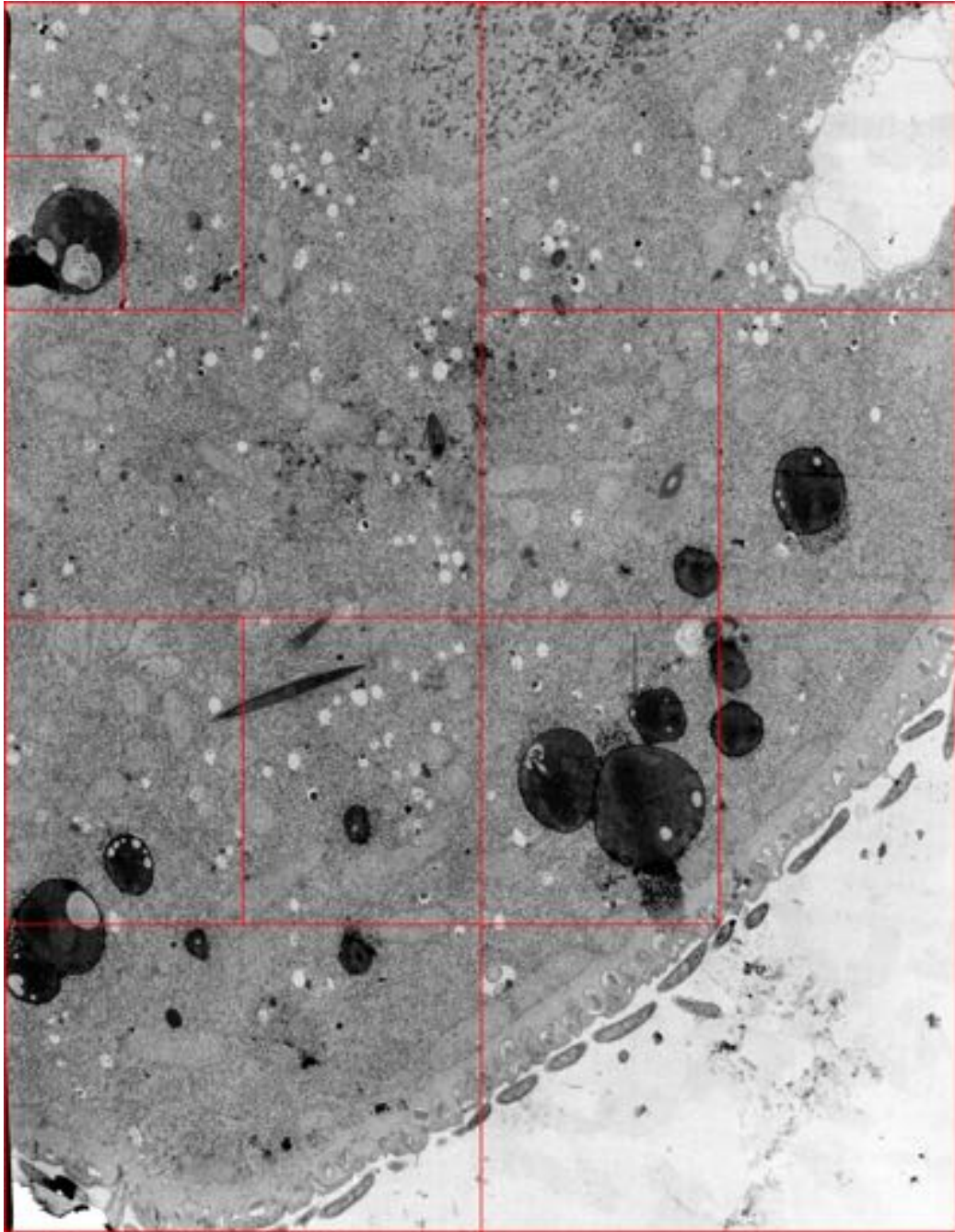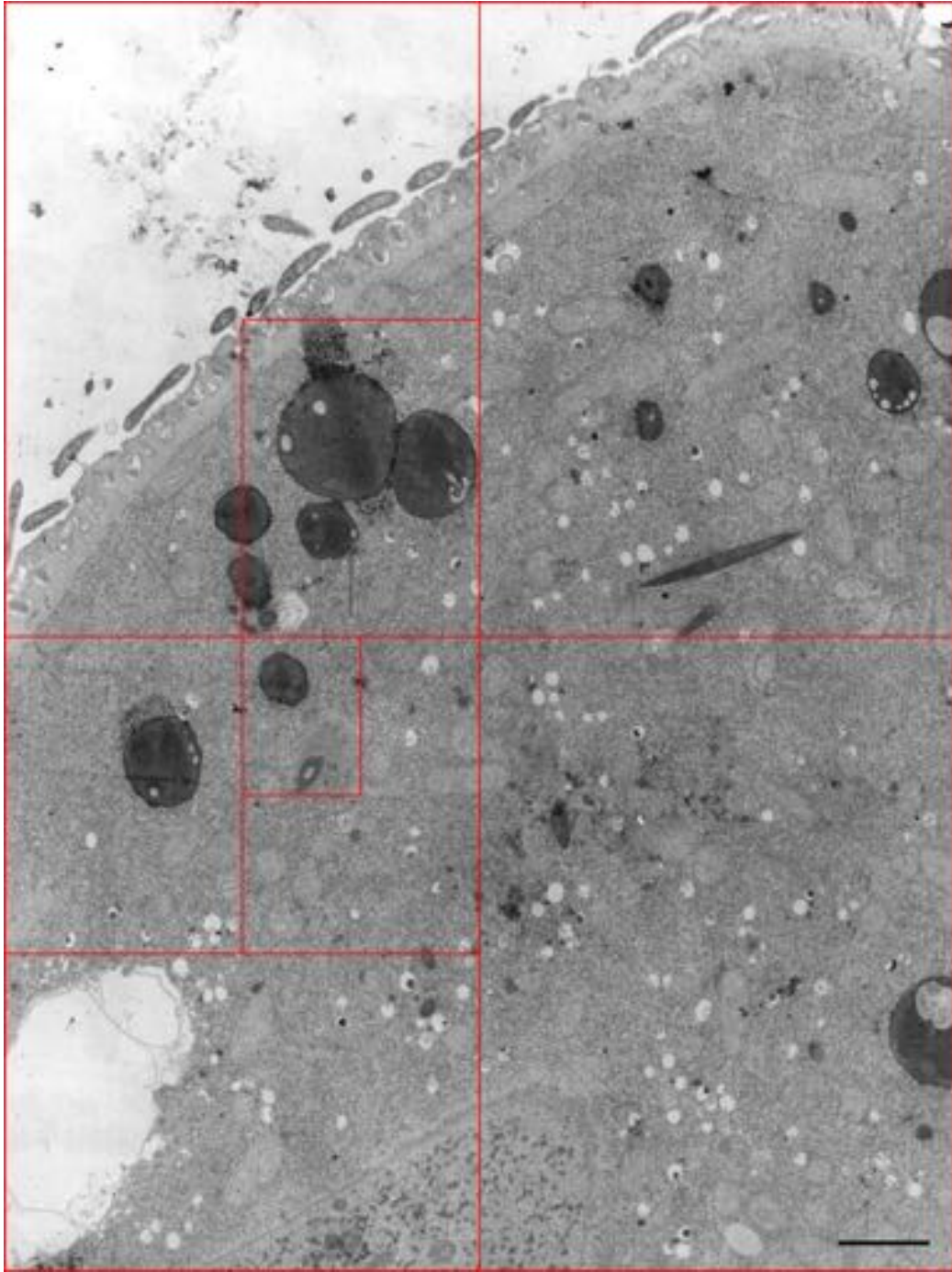
Figure 4.4: Training image

Figure 4.5: Testing image

52

We can see that we detected most of the black cell components with much smaller number of scoring. 44 for training image and 32 for testing image. It is 80 percent reduction. We did not captured all components, but this could be improved by more training data, more distances, more augmentation etc.

4.7 Overall recursive classification

In this section we put all classifiers together and apply recursive classifier for both, black components detection and gas/fluid capsules detection. Lets define a JSON file.

```json
{
    "input_folder": "./lab/data/testing_images",
    "layers": [
      {"classifier_names": ["black_component_detection"],
        "results_processing": {
          "pass": {
            "handle_tiles": "all",
            "handle_classifiers": "all",
            "do_layer": [0],
            "criterion": {"keep_growing": "", "do_layer_on_chain_break": [1]},
            "do_output": [{"color": "red" , "text": "Detected black component"}]},
          "fail": {
            "handle_tiles": "all",
            "handle_classifiers": "all",
            "do_layer": [-1],
            "do_output": [-1]}},
        "image_division": {"type": "halves", "stop_criterion": {"min_res": [160,160]}}},
      {"classifier_names": ["gas_fluid_capsule"],
        "results_processing": {
          "pass": {
            "handle_tiles": "all",
            "handle_classifiers": "all",
            "do_layer": [1],
            "criterion": {"keep_growing": "", "do_layer_on_chain_break": [2]},
            "do_output": [{"color": "green" , "text": "Suspicious capsule"}]},
          "fail": {
            "handle_tiles": "all",
            "handle_classifiers": "all",
            "do_layer": [-1],
            "do_output": [-1]}},
        "image_division": {"type": "halves", "stop_criterion": {"min_res": [160,160]}}},
      {"classifier_names": ["gas_fluid_capsule"],
        "results_processing": {
          "pass": {
            "handle_tiles": "all",
            "handle_classifiers": "all",
            "do_layer": [-1],
            "criterion": {"minimal_thresholds": [0.7]},
            "do_output": [{"color": "orange" , "text": "Detected capsule"}]},
          "fail": {
            "handle_tiles": "all",
            "handle_classifiers": "all",
            "do_layer": [-1],
            "do_output": [-1]}},
        "image_division": {"type": "none"}}
    ],
    "output_folder": "./lab/data/overall_classification"
}
```

Lets run it with follwing command:

```
Ondrejs-MBP:AnalystTool ondrejszekely$ python3 ./run_scripts/run_CLASSIFIER_recursive_classifier.py
  --conf_file "./lab/json_configs/overall_classification.json"
```

Figure 4.6: Testing image

It will take some time, because we reduced number of scored tiles, but we increased number of API calls, which is time consuming. In addition there are many if conditions, etc. This tool is dev tool, not production tool which means, with this tool you should test your hypothesis which then should be implemented in more efficient way.

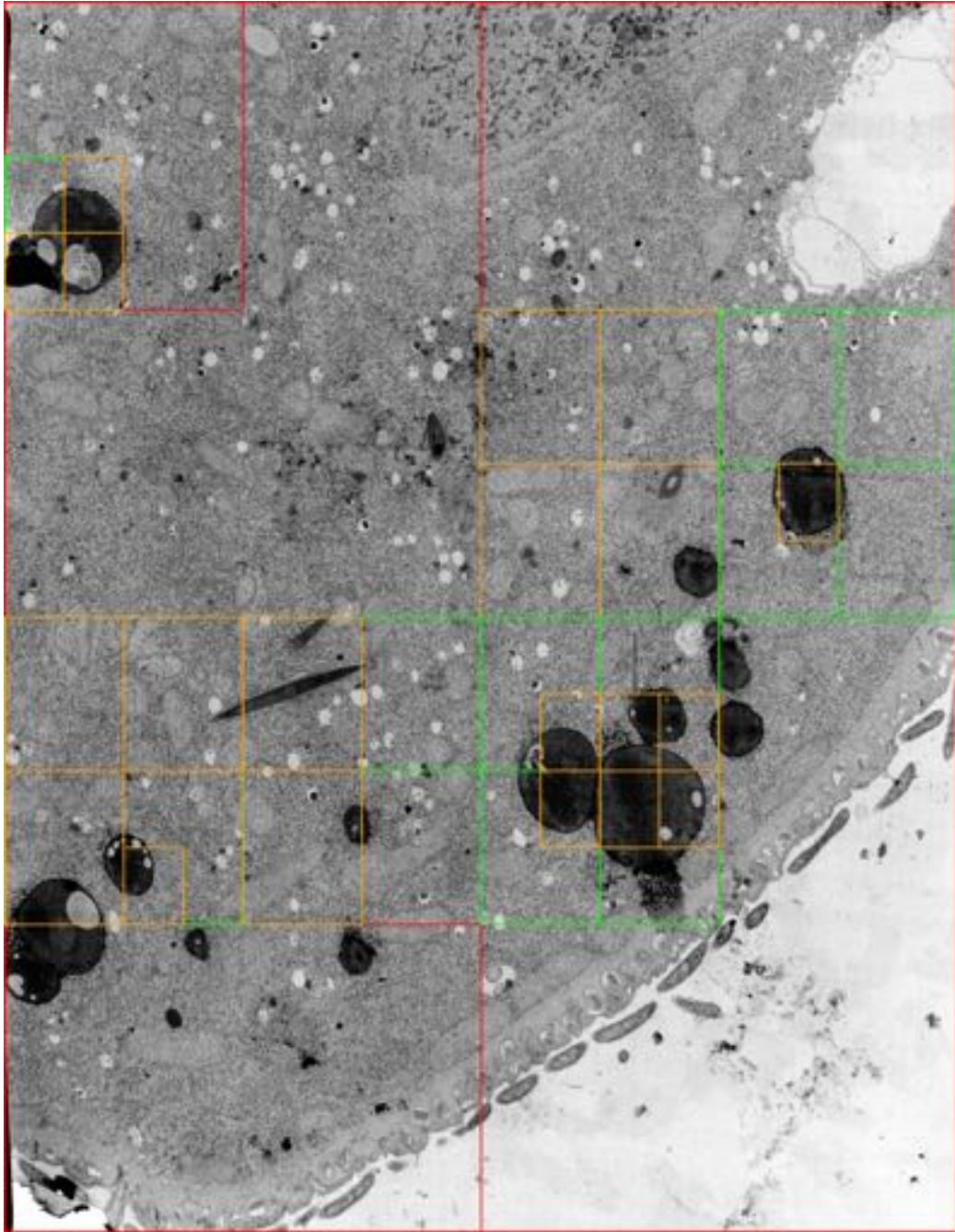When the training is completed you should obtain similar images:
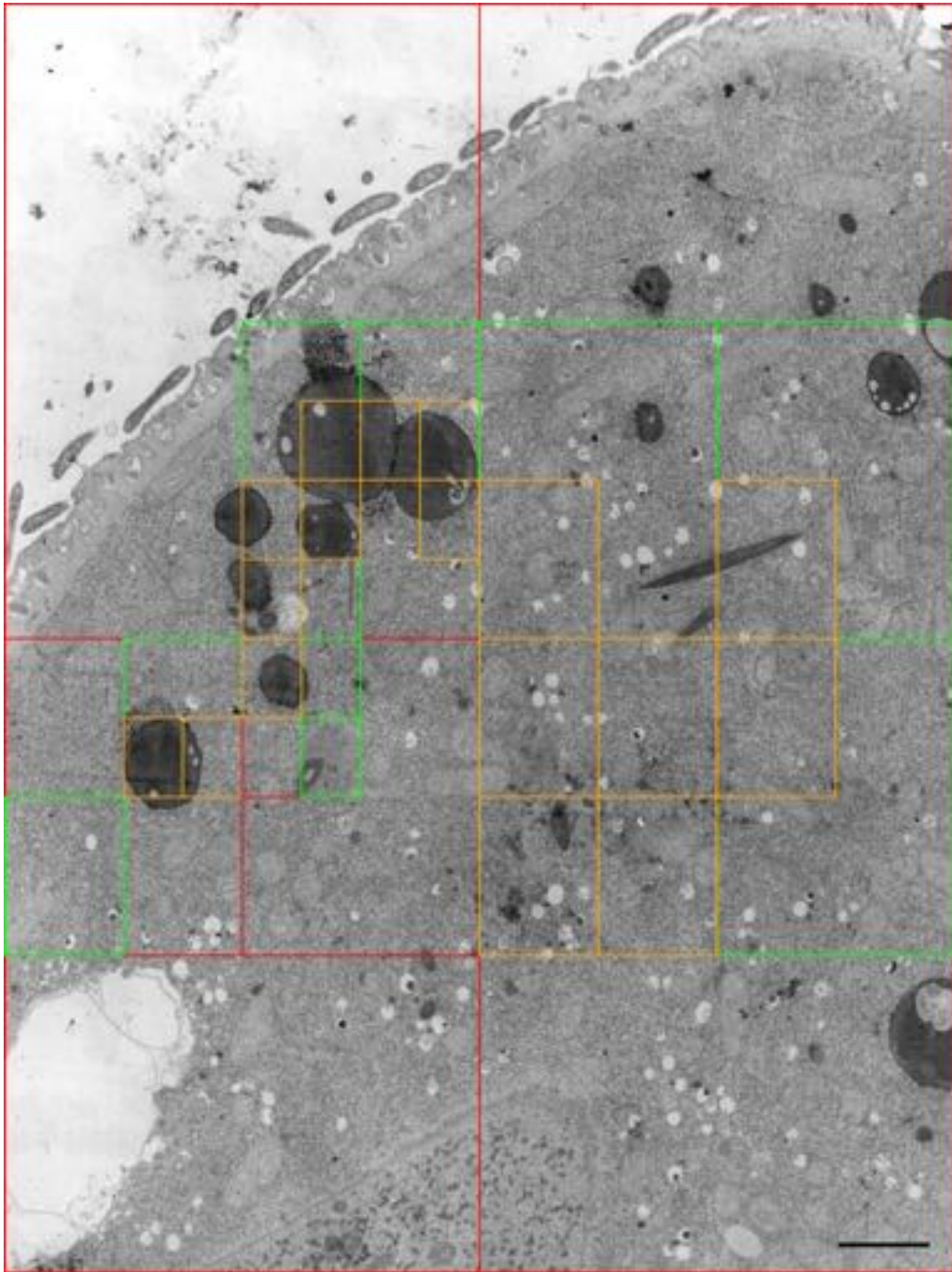
Figure 4.7: Training image

Figure 4.8: Testing image

Now, **red** color represents areas which are suspicious by presence of black component (mostly it is covered by top layers). **Green** represents areas which are suspicious of presence of areas gas/fluid capsule and the **orange** represents areas which are confident that there is a capsule with confidence higher than 0.7 and tile could not be further divided.

We can see that it works some way, but if we compare results of directly used gas/fluid capsule classifier with tiling and this one, we can see that the classifier has lower performance. **The reson is that during training of gas/fluid classifier we used square like tiles with one resolution for negative class. Thus you should be aware of that and if you want to use dynamical tiling you need as a first step crop source images into squares/add more additional data with rectangle like tiles + add more images from various distances as a training data.**