Clever-Lab AI Framework, an extremely flexible AI framework for autonomous observation on long-running laboratory experiments

Hsieh, Kai-Chun

Munich University of Applied Science

Abstract

Experiments with visual observation in scientific laboratories can take long time before recognizing effects. As salaries of laboratory staff have significant effect on cost calculation (up to $80/h) and waiting of the unknown result from combination of several chemical, physical or biological components would be boring and wasting time – which means, meanwhile, wasting of pretty much money.

Under this circumstance, Machine Learning and Deep Learning may help.

However, among so many online and offline machine learning platforms, we found several problems: some of them having unclear pricing, some of them having bad performance, and some of them are hard to use. Especially, for data protection considerations, prohibit many companies the use of cloud-based services, although they provide often better user-experiences.

Considering AI and Machine Learning are too hard for scientific laboratorians to comprehensive, we would like therefore to build a machine learning framework, and the goal of which is to provide an API-lize platform, that is easy to use, open and comfortable to be extended, and convenient for users by saving their precious time.

Contents

Preliminary Knowledges

Machine Learning

Machine learning is a paradigm shift of computer programming. Traditionally, developers write codes manually, programs accept input data, process data then give the corresponding results out. Machine learning turns this pattern over: it accepts input and output data simultaneously, outputs “computer codes”. Machine learning is a useful tool, it generates code to solve problems, code of which are probably hard, or impossible to be implemented manually, voice recognition, time series analysis, and face recognition, for instance. Goal of machine learning is to discover patterns of data from a specific source and be able to make prediction on results from new potential data. (David Spieler, 2019)

Generally, most machine learning models can be sorted in to these three categories:

1. Supervised Learning:
   Preprocessed data as vectors of features are labeled with “tags” (the expecting output data) as training-data. During training, the model will be adjusted in order to make its output data match to the tags that the input data was given.
   Tasks of a supervised learning machine learning model is to infer patterns of input data and to make predictions according to the patterns they found.

2. Unsupervised Learning:
   Unlike supervised learning, input data as training data of an unsupervised learning model are not labeled with any tag. Unsupervised learning models are practically widely applied to gathering uncategorized data into different groups (cluster analysis), finding unknown associations between datasets (association rule learning) …etc.

3. Reinforcement learning.

Models, in machine learning discipline, is defined as the representation of mathematical functions and data structures.

Typically, 3 different phases with different operations will be performed during training a machine learning model:

1. Pre-process:
   Pre-processing of the training data, including normalization, reshaping of data, quantizing and extracting different features from data.

2. Train:
   “Training” is the process, that parameters of a machine learning model being tuned with training data in order to make the model be able to solve user specified problems.
   Training data will be usually randomly splitted in to 2 parts - training part, which will be used to train a model in this stage, and validation part, which will be used in the next stage to validate the effect of the trained model.

3. Validate:
   Validate the model trained in stage 2 with validate-dataset in order to detect whether the model is “overfitted”.
   A model is “overfitted” when it can solve problems with training-dataset specially well, however, has bad performance against other datasets.

Machine Learning and Support Vector Machine

In this project, we apply linear support vector machine in the “Local Foam Elimination Detector” plugin for detecting completeness of chemistry experiments and with which reducing laboratory works’ workload further.

Linear support vector machine is a supervised machine learning algorithm separating data points (vectors) into different groups with a “maximum margin”, a linear function as border.

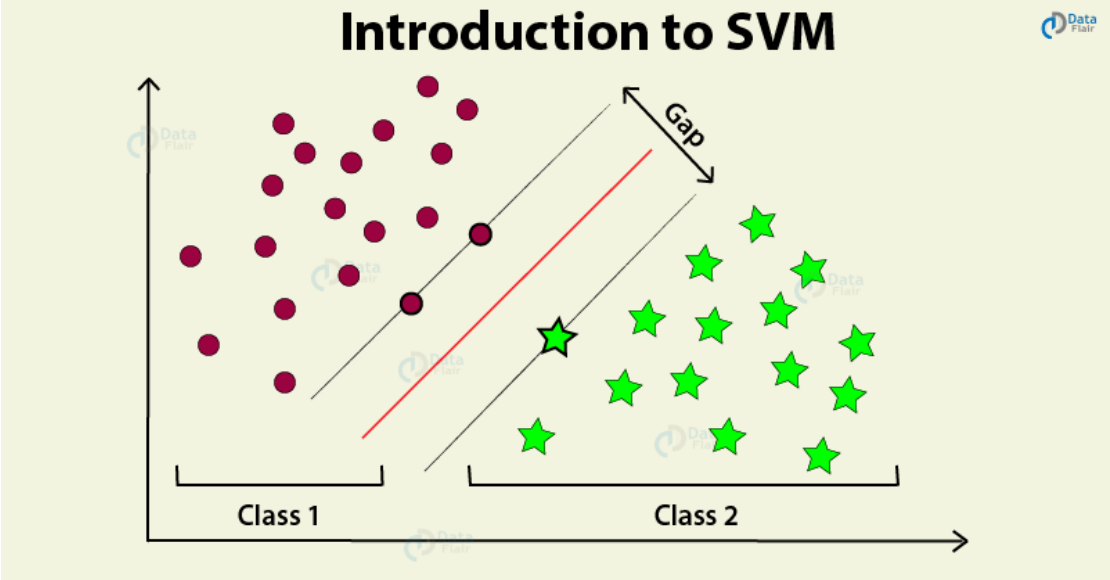


Figure 1 Illustration of an SVM classifying data into 2 groups. (DataFlair Team. Introduction to SVM. [Digital Image]. (2019). Introduction to Support Vector Machines: https://data-flair.training/blogs/svm-support-vector-machine-tutorial/

The “maximum margin” is the ultimate result that training of an SVM model is to find. Such margin can not only separate datasets properly but should also distance itself maximumly to support vectors.

Support vectors are the data point that are decisive on choosing margins. The green star and those 2 dark-red circles with black frames in figure 1 for instance.

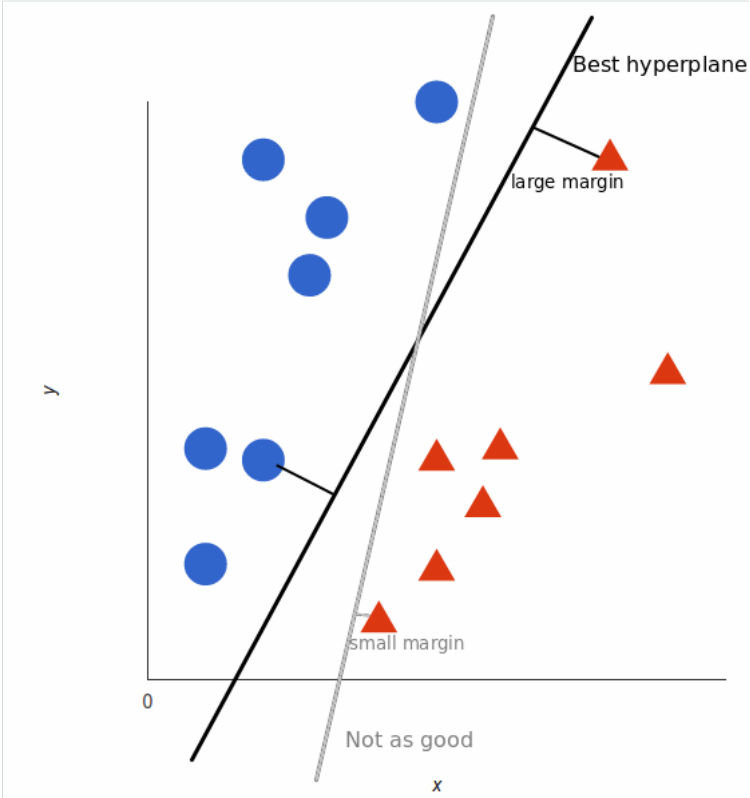


Figure 2 Example of choosing a better linear SVM margin. (Bruno Stecanella, 2007). Not all hyperplanes are created equal [Digital Image]. (2017). An Introduction to Support Vector Machines (SVM): https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

Clever-Lab AI Framework

To achieve our goal, we purpose a framework contains 3 different modules, 2 by the server side, and 1 by client side:

- Server side:
    1. Resource Management module
    2. Solution Management module
- Client side:
    1. Client SDK and Notification module

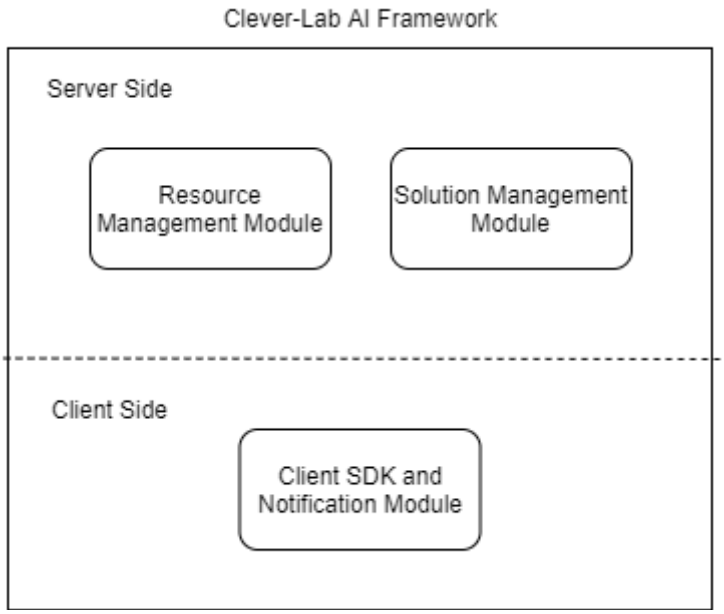


Figure 3 "Composition" of the Clever-Lab AI framework.

To maximize flexibility, compatibility and expandability of these modules, we designed the system to be distributed: modules are able to work independently to each other; each module is also extensible via extra “plugins”, which works also independently to each other. Moreover, to standardize the system and make developers have clue during implementing new plugins, we provided also interfaces with clear description for all 3 modules. With such interfaces, developers are also demanded to provide several essential information to help users to choose among numerous of plugins installed. For instance, effects of a plugin, price per operation by using the plugin…etc.

Practically, in our implementation, during server starting, modules by the server side (resource management and solution management module) scan under specified folders for plugins. Once a plugin is found, simple test for compatibility will be performed and an instance of the plugin will also be created and will make to be ready to be summoned.

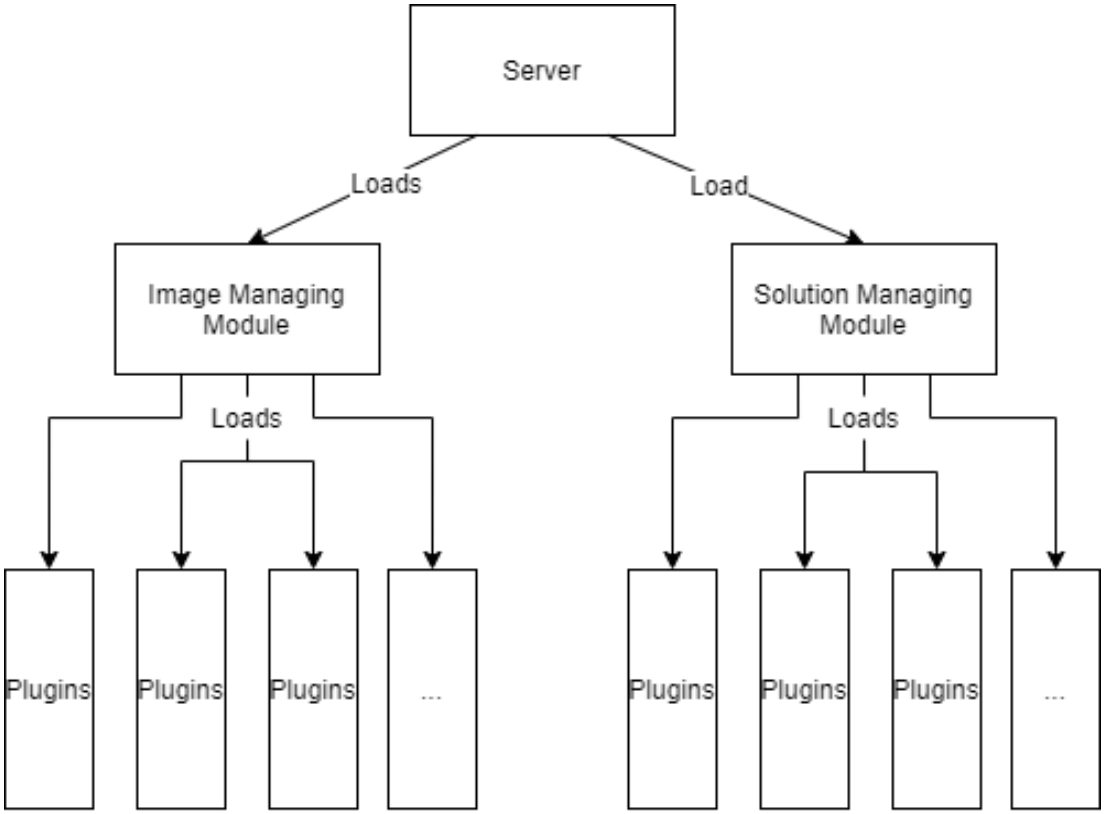


Figure 4 Loading-flow of components in a Clever-Lab AI server.

Whenever a user request is sent to server, modules in module-layer will handle it directly and pass the request into the correct plugin locating in the plugin-layer.

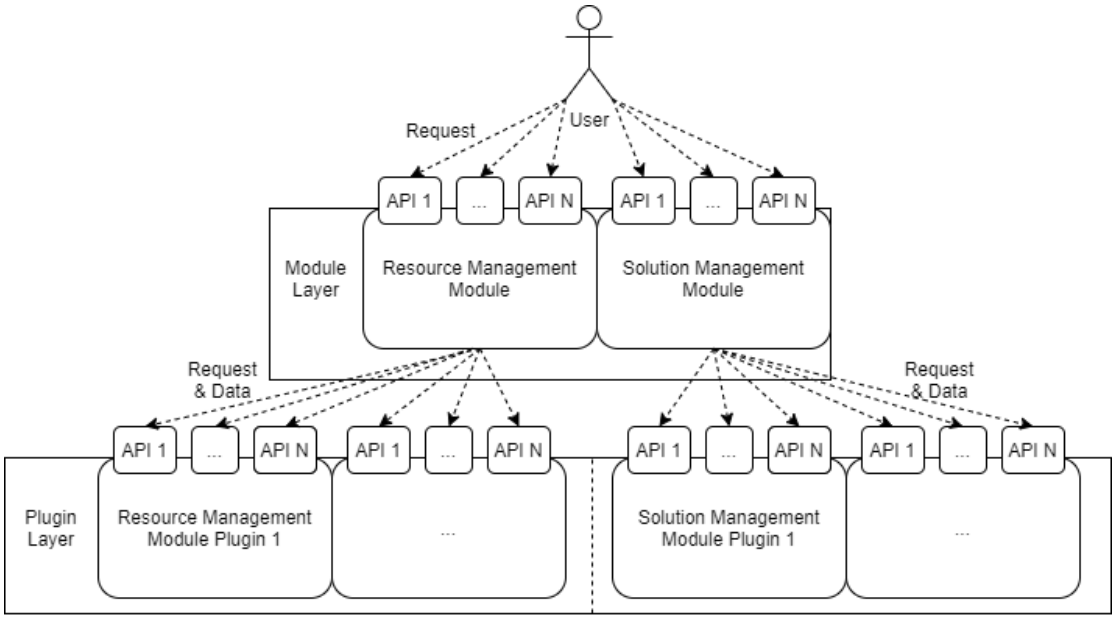


Figure 5 Information- and request-flow between different modules in a Clever-Lab AI server.

Moreover, for better compatibilities between different modules/plugins, we decide to make dataflows as simple as possible: data will be always passed with raw format or as a unique ID generated in the module-layer.

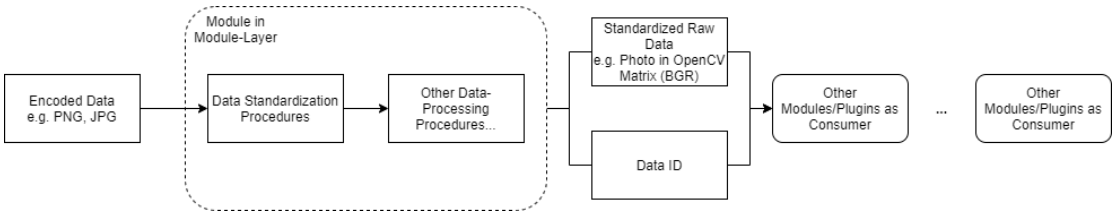


Figure 6 A figure suggesting how data transformed during different phases.

Therefore, in our design, whenever user requests to create/upload a resource on a Clever-Lab AI server, a plugin-ID shall be always given, with which the correct plugin instance can be found, and data can be consequently piped into the correct destination/processor. Metadata of the resource will then be recorded in a database and an unique resource-ID will be returned to user if the resource was successfully created.

Plugins are the real parts that taking effects in our framework. It contains idea, creativities of developers to process incoming data or to perform related actions according to requests: different plugin performs different procedure on data. With Clever-Lab AI framework, users are able to make their decision freely relaying on information from different plugins, that would always be provided by the system, to get different effect/approaching fulfilling their requirement.

For example, a “local_photo_loader” plugin under Resource Management module is able to save given images into the local filesystem of a Clever-Lab AI server. With proper implementations, the server will can save images also, for instance, in a NAS, and even Google Drive, Amazon AWS Cloud Storage...etc.

In following section, we will shortly introduce modules and their features that mentioned above:

Resource Management Module

This module manages the resources uploaded by users, it not only standardizes incoming data by transforming them into raw data types, such as images into OpenCV BGR (Blue, Green, Red) matrix, and passes them into the requested plugins for further processing, but also acts as a “proxy” bridging between users and plugins during clients’ query on resources or resources’ metadata.

Documentation of the public API and interface this module provides can be found in appendix 2.

Solution Management Module

This module manages solutions installed and proxying/processing user’s requests on creating, training and applying of machine learning models.

“Solution” represents, in this framework, the set of machine learning model type, hyperparameters of the machine learning model and data preprocessing algorithms it applies for normalizing, extracting features from input data.

“Model", is defined, in our framework, the storage of a trained model and its parameters, e.g., the slope and offset of the regression-function from a linear regression model.

Documentation of the public API and interface this module provides can be found in appendix 3.

Client SDK and Notificators

For furthermore user-friendly purpose, we implemented also a python library, which provides serialize functions applying Clever-Lab AI server APIs and is also able to visualize server responses automatically into several user specified forms, e.g., table-view and image-view.

Notificators are also a useful feature that we implemented in the client library. Notificators accept few parameters as trigger-condition and pull the trigger when the condition is fulfilled, e.g., target result with x% of confidence detected.

Following notificators are already available in our implementation:

1. Console Notificator:
   A notificator prints notifications directly to console.
2. MQTT Notificator:
   A Notificator publishes notifications via MQTT protocol[1].

Both notificators supports following triggering-modes:

1. By number of times a specified class detected.
2. By confidence a specified class detected.
3. Both 1 and 2
4. Either 1 or 2

Meanwhile, we provide also several useful, convenient, interesting utilities in the client library, such as /streamPredict, which takes photos periodically from camera connected to a Windows PC via DirectShow interface[2], uploads and make classification with taken images to a Clever-Lab AI server and then feeds classification results into a notificator instance provided by user. All automatically.

```
notificater = notification.MQTTNotification("broker.hivemq.com" ,
                                            1883, "[username]", "[password]",
                                            "KenMLClient", "ken/bachlorarbeit")
with notificater:
    model.streamPredict("../video_capturer/ffmpeg/bin/ffmpeg.exe", 60, 120,
                        "Lenovo EasyCamera", 1, notificater
                        ,messageType=notification.MESSAGE_TYPE_TEXT, classDetectedThreshold=2)
```

```
DEBUG Connected to MQTT-Broker
DEBUG A Photo was succesfully taken at 2020-03-02 09:22:45.452054
DEBUG A Photo was succesfully taken at 2020-03-02 09:23:46.892803
>>>>>>>>>>>>>>>>>>> Text Message at 2020-03-02 09:23:46.934797 : Target class '1'(100.0%) detected!
```

Example 1 Stream-Predict and notifications.

With such combination of function and notificators, one of our goals is achieved, an autonomous experiment-completion detector: Users

[1] MQTT is a reliable, lite weight messaging protocol using publish-subscribe pattern published by IBM in year 1999.
[2] DirectShow is a media framework and API published by Microsoft, based on Microsoft Windows Component Object Model (COM).

provide prepared notificators and models, they can then spare their hands keep on other important works, and, most importantly, they will always be notified when experiments are detected to be done.

Documentation and usages of this client module can be found in appendix 3.

```
import utils, time
import resource_managing, solution_managing

rm = resource_managing.ResourceManager("127.0.0.1", 5000)
sm = solution_managing.SolutionManager("127.0.0.1", 5000)

dataset0 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                              utils.getFileList("../Dataset1/foam", extension = "bmp")).uploaded
dataset1 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                              utils.getFileList("../Dataset1/nofoam", extension = "bmp")).uploaded

model = sm.createNewModel("edu.hm.hsieh_localfoameliminationdetector_1.0", "localModeltesting",
                          "its a test model")
model.feed_train_data(dataset0, 0)
model.feed_train_data(dataset1, 1)
model.train()
(159/0/172)
(160/0/172)
(161/0/172)
(162/0/172)
(163/0/172)
(164/0/172)
(165/0/172)
(166/0/172)
(167/0/172)
(168/0/172)
(169/0/172)
(170/0/172)
(171/0/172)
(172/0/172)
Caching image done: 2.49277400970459
Training model with 172 photos...
Training done saving model...
Model saved.
Done! Model successfully trained and saved.
```

Example 2 Train a model using client SDK library.

```
uploadPredict = pm.uploadPhotos("edu.hm.hsieh_mylocalphotoloader_1.0",
                                utils.getPhotoList("../Dataset1/nofoam"))
datasetPredict = uploadPredict.uploaded
model.predictWithPhotoList(datasetPredict,
                           view=model_managing.PREDICTION_VIEW_TABLE)
```

	id	class	score
0	416	0	100.0%
1	417	1	100.0%
2	418	1	100.0%
3	419	1	100.0%
4	420	1	100.0%
5	421	1	100.0%
6	422	1	100.0%
7	423	1	100.0%
8	424	1	100.0%
9	425	1	100.0%
10	426	1	100.0%
11	427	1	100.0%
12	428	1	100.0%
13	429	1	100.0%
14	430	1	100.0%
15	431	1	100.0%
16	432	1	100.0%
17	433	1	100.0%
18	434	1	100.0%
19	435	1	100.0%

Example 3 Make prediction with client SDK library.

Case: Foam decrease experiment

After the server of the Clever-Lab AI framework was built, we started to design a solution plugin containing a model to solve the rest of problems: a plugin being able to classify, whether a chemistry reaction is completed or not. After discussions, we decided to use linear Support Vector Machine as the classifier and started our experiments firstly with espressos.

We picked fresh espressos since it easy to access, and critically, since we found lots of similarities between chemical experiments and the process of espresso foam's elimination:

1. Many chemical reactions changes colors. Just like espresso's foam, it was brighter and more colorful when it was just made - part with foams are partially white, partially light brown and partially darker brown. After standing the coffee still for couples

of minutes, the liquid becomes darker and the color becomes more uniform due to foam's elimination.

2. Pictures of many mixture of chemical components will become more and more orderly as time passes.
   For instance, when salt firstly added into water, the water became chaotic: crystals sediment on the bottom of the glass making the water "angular". However, after few minutes of waiting, salts dissolve itself in the water completely, and the picture of the water become again still and transparent, as if nothing had happened. Just like espressos: white foams and small bubbles covered firstly on top of them. After the foam eliminated, the hue of the espresso will uniformly become dark.

With these observations we decide to extract image features with following algorithms:

1. To detect complexity of colors containing in pictures from an undergoing experiment, we referred algorithms purposed in paper *Algorithm and implementation of image classification based on SVM* (Zhang, Zhao & Li., 2007):
   a. Grayscale-Difference-Degree of an image is defined with:

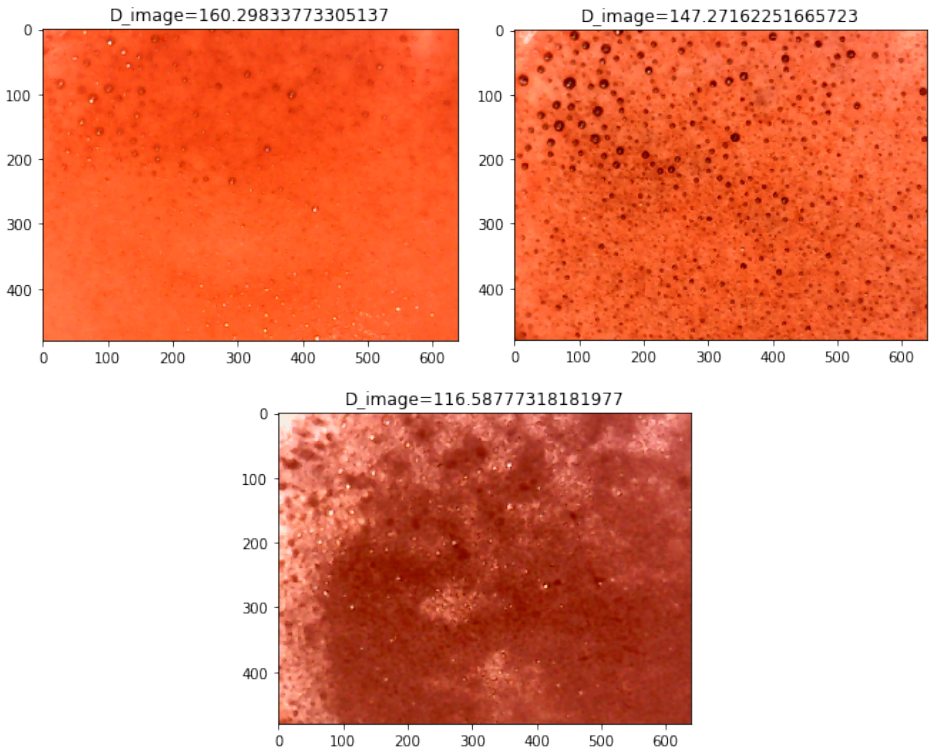
      $D_{image} = \frac{\sum_{i=1}^{N} d_i}{N}$, $N :=$ Number of pixels in the image.

   b. In which d_i is the Color-Difference-Degree from a pixel:

      $$d_i = \sqrt{\left(R_i - C_{average}\right)^2 + \left(G_i - C_{average}\right)^2 + \left(B_i - C_{average}\right)^2}$$

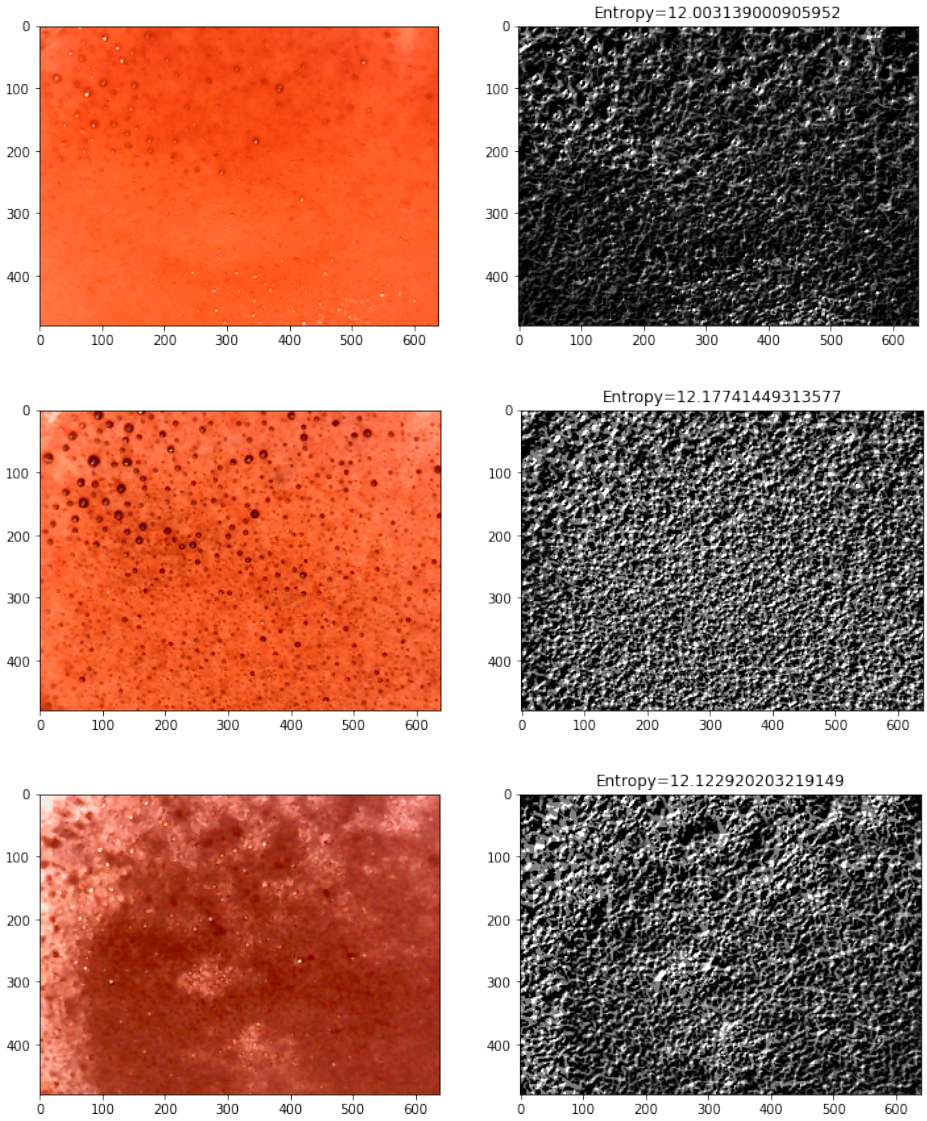
   c. $C_{average}$ is the average of grayscales of every color-channel from an image, which can be noted with:

      $C_{average} = \frac{\sum_{p=1}^{P}\sum_{c=1}^{C} c}{p \times c}, C =$ Number of color-channels, $P =$ Number of pixels from an image.



Graph 1 Espresso at different time and it's D_{image} values.

2. To detect the "order-ness-degree" (E_{Image}) of image from chemical components, we decided to extract our second feature with following algorithms:
   a. First step: Convert images into gray-scale images,
   b. Second step: Blur images with a 3x3 Gaussian filter.
   c. Third step: Apply Sobel Operation along both x and y axis in order to extract the "edges" from the image.
   d. Fourth step: Calculate the Shannon Entropy of the image from step 3 as a feature, which indicates the complexity of the edges of images.



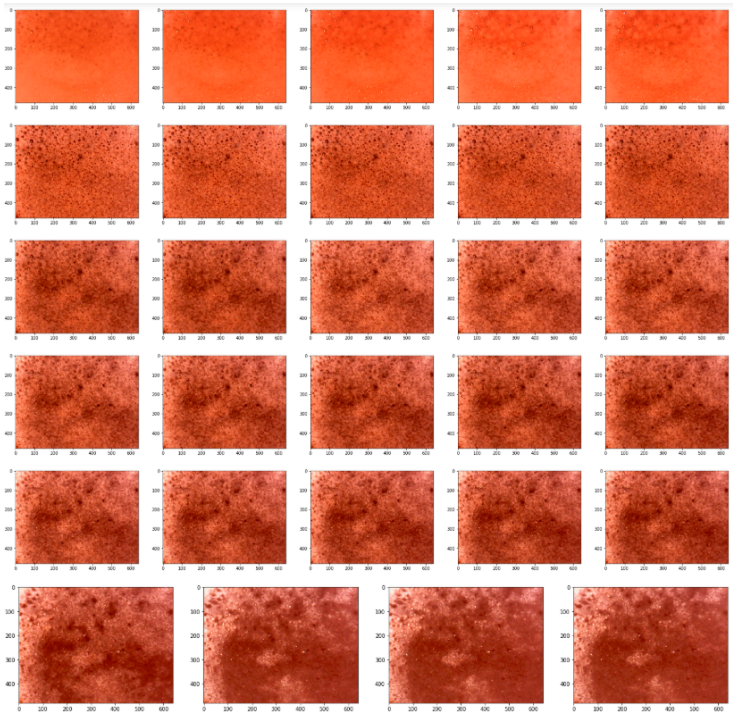
Graph 2 Espresso at different time and it's Sobel operation results and corresponded order-ness-degrees (Entropy, E_{Image}).

3. Normalize and map the features into (0, 1) for more stable results by using a MinMaxScaler.
4. Train a “Linear Support Vector Machine” with the normalized features.

To make sure our model is correctly train and is able to solve problems correctly, we

0. take serious of photos from a freshly made espresso regularly using a python script as the dataset,
1. take 29 images randomly from an original dataset as training-dataset, the others as validation-dataset,

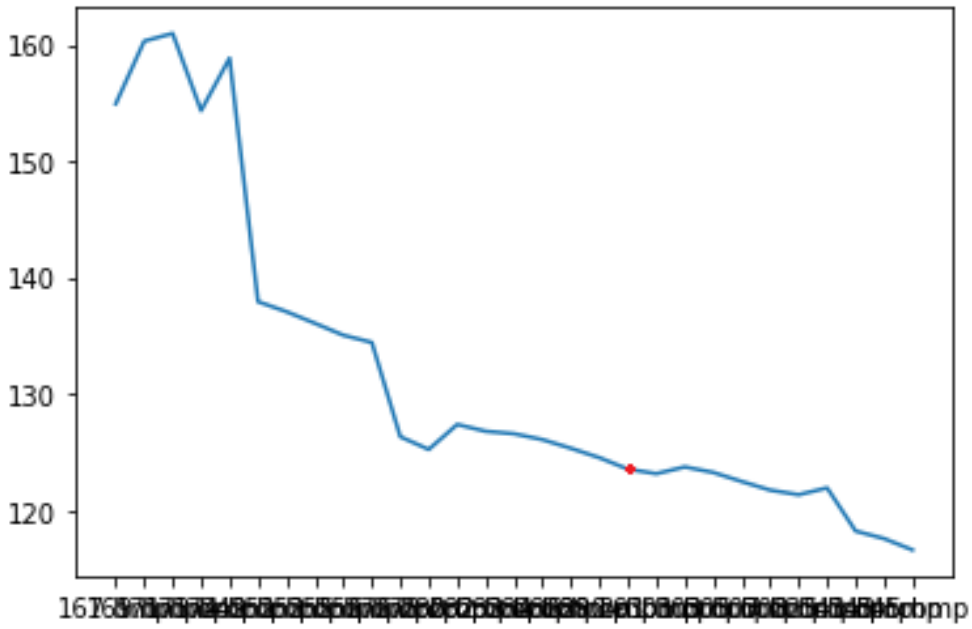
2. train a model with the training set, and finally
3. verify the trained model with the validation-dataset.



Graph 3 The training dataset.

Model Verifications:

Firstly, with apply algorithm 1 on all training data. And sort result by timeline ascendingly:

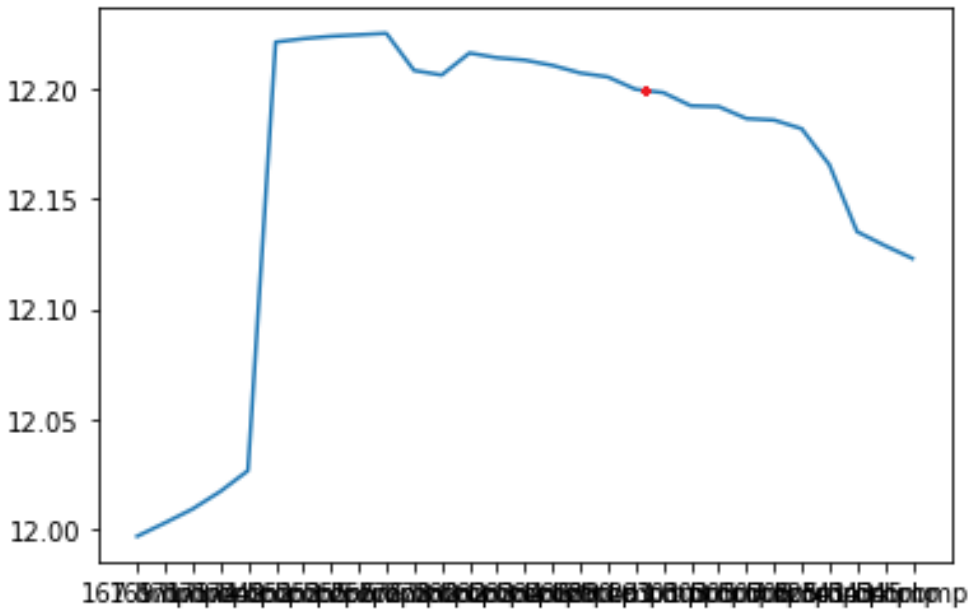


Graph 4 Images and their D_{image} (Color-Difference-Degree). X-axis are images names; Y-axis are the D_{images}. Data from the red dot on are "no-foam" photos.

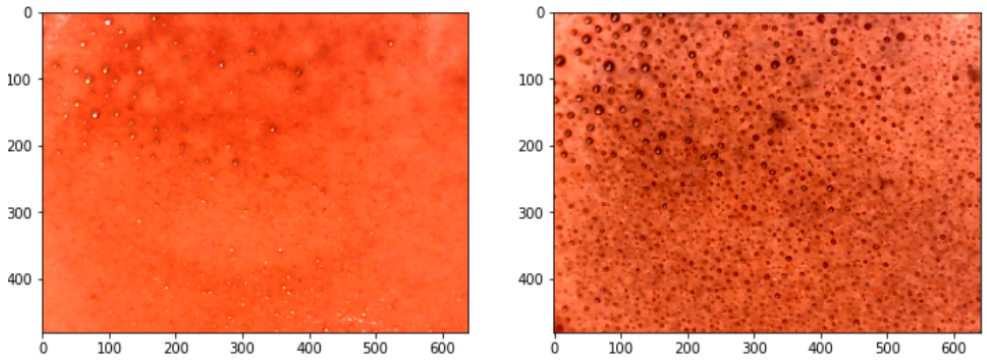
As we expected, the D_{images} dropped constantly as the color became more and more uniformly dark.

To the next, we applied algorithm 2 (chaotic-degree detection), and we found immediately that it is generating results exactly what we are expecting for: The E_{images} rises up at start, due to the bursting of bubbles making the picture of the espresso "more shuttered" and descends gradually down as time go by, since the elimination of foams.

Plus, the algorithm reacts also exactly to the dramatic change of espresso images from the 5th to the 6th photo, corresponding to the training dataset, between which, foam eliminated rapidly:

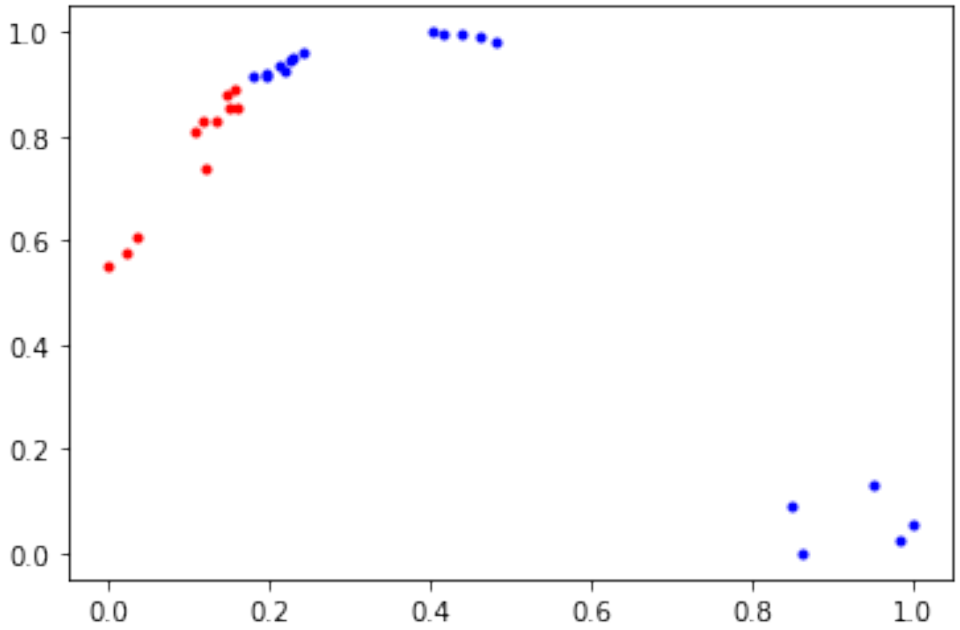


Graph 5 Images and their Entropies. X-axis are images names; Y-axis is the Entropies. Data from the red dot on are "no-foam" photos.

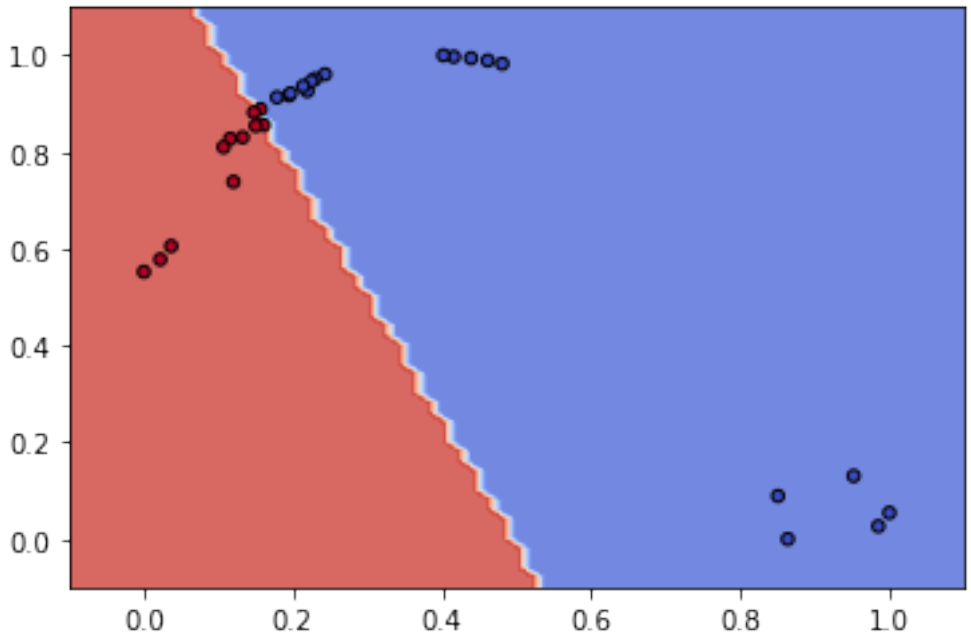


Graph 6 Two neighbor image datasets making the "peak" suggested in graph5.

Then, we normalize the extracted features and trained a linear SVM:

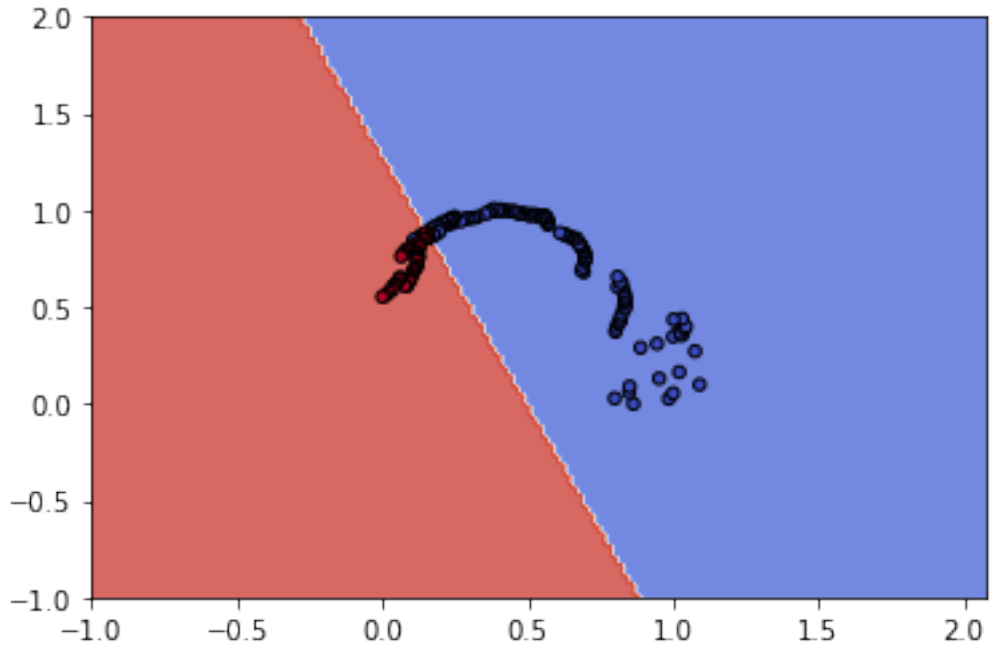


Graph 7 Normalized features of training-dataset. Blue dots are images with foam and red dots are “no-foam” images.



Graph 8 Prediction result with training data of the trained model. Red dots in red area are images marked without foam and blue dots with foam. The white line in-between is the decision-boundary of the SVM model.

Last but not the least, we make prediction with the trained SVM on original dataset as validation:

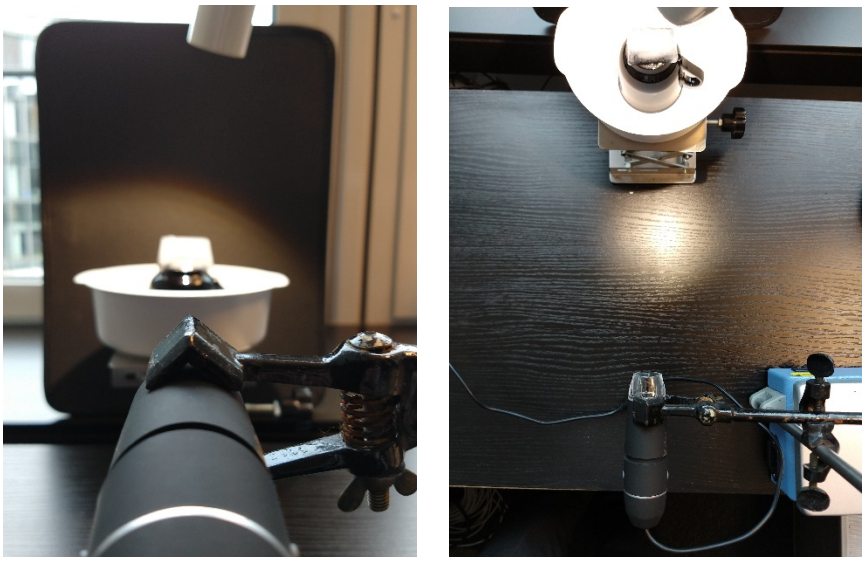


Graph 9 Validation result of the whole dataset. Red dots = “no-foam” images; blue dots = photos with foam.

With accuracy of the model on validation dataset up to 97.1% ($\frac{167}{172} \sim 97.1\%$), we have therefor sufficient reasons believe that the model was correctly built and trained.

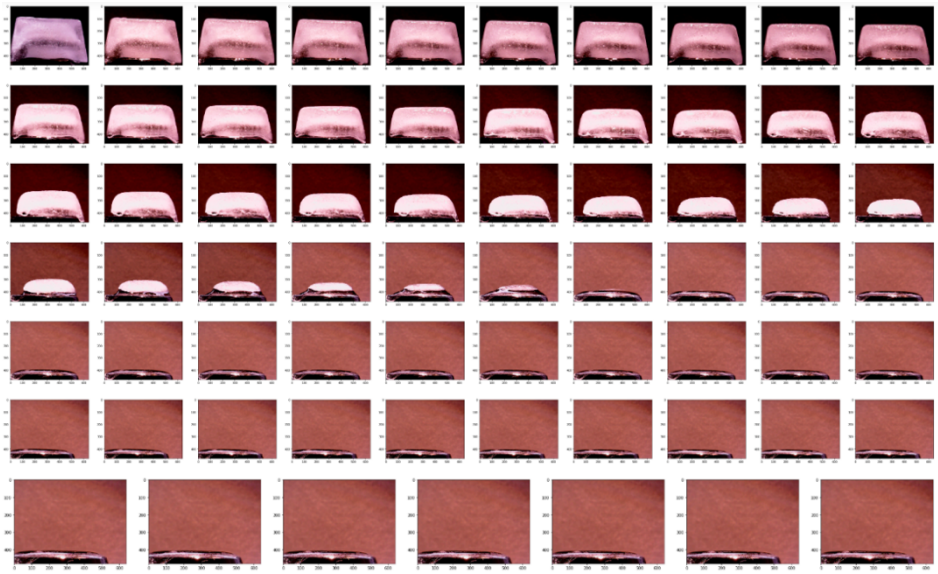
Case: ICE Cube experiment

To test the model further, we built up subsequently another experiment. This time with ice cubes.

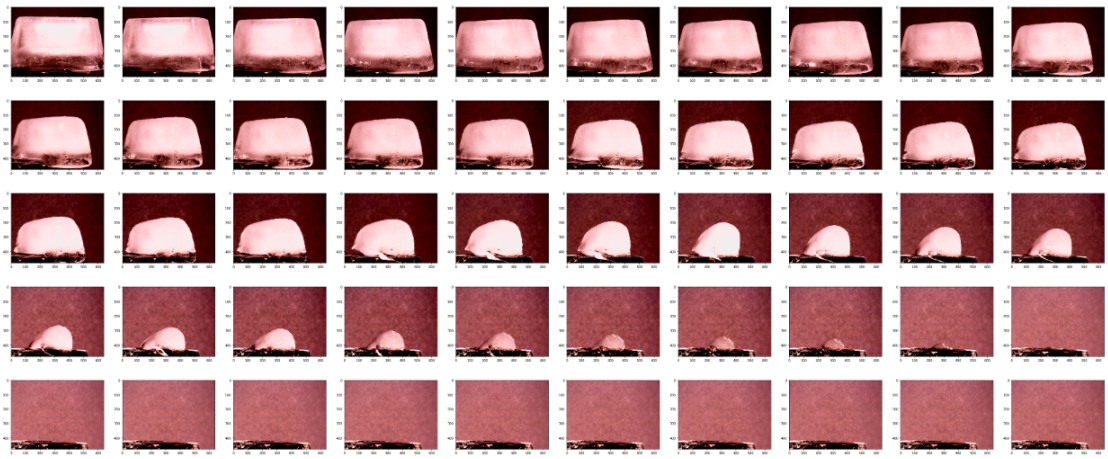


Graph 10 Setup for ice cube experiment.

The camera was setup to take photos every 2 minute and produced datasets as followings:

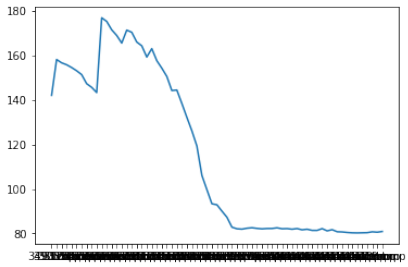
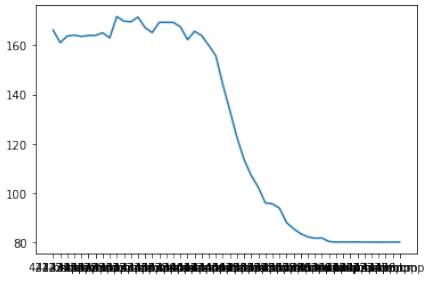
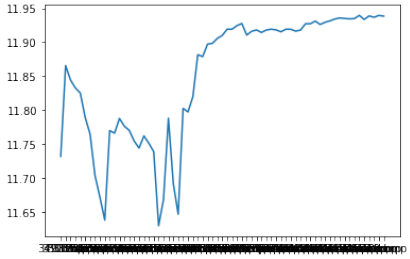
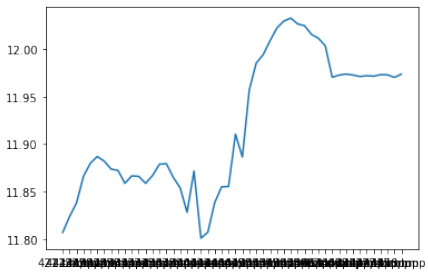
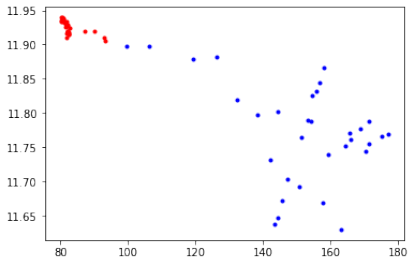
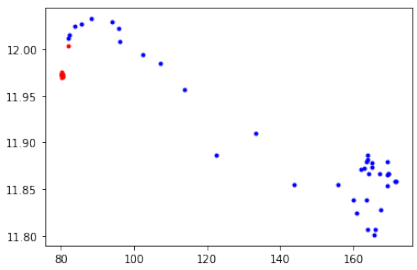
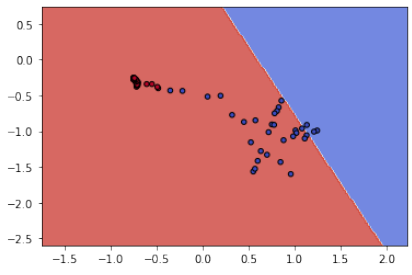
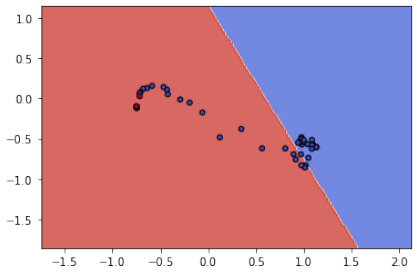


Graph 11 Ice cube dataset1.



Graph 12 Ice cube dataset2.

Results are listed in the following table:

D_{image}		
E_{Image}		
Features		
Predict		

Accuracy	$\frac{3+35}{32+35} \approx 56.7\%$	$\frac{18+13}{38+13} \approx 60.8\%$

Table 1 Make predictions on ice cube datasets with SVM model, MinMaxScale normalizer trained with espresso foams.

As result, the accuracy dropped dramatically, since the model and the normalizer were trained for espresso foam specifically. However, after training new model with ice images, accuracy on ice datasets raised back to normal level as expected:

Accuracy of prediction with SVM model trained with ice cube dataset**1** on ice cube dataset**2**: $\frac{27+13}{38+13} = 78.4\%$.

Accuracy of prediction with SVM model trained with ice cube dataset**2** on ice cube dataset**1**: $\frac{30+35}{32+35} = 97.0\%$;

Summary of Clever-Lab AI System

The raw format data-stream do make great effect on compatibilities; however, as side effect, in some cases, e.g. with the “Azure Custom Vision” plugin, performance lost maybe enormous (≈ 20% overhead).

Despite of the overheads by conversion in our implementation, this design still brings advantages.

For example, with such design, image resources can be fully reused: practically, it is often that a data scientist would like to compare effectiveness of different machine learning models with same batch of data. Under this circumstance, our design is comparing with other enterprise solutions much faster and brings more comfortableness.

To determine the performance improvements with our design under such scenario, we carried out another experiment: we measured the time that different phases of different services needed from very beginning till all data settled down and the model been ready to be trained with 100 uncompressed PNG (88 megabytes in total) from a client equips VDSL 100Mbps/40Mbps, i5-6500 CPU, 16GB DDR4 RAM, Windows 10 1909.

3 trails are performed in total and averages of time consumed are listed in the following table:

Service	Description	Score
IBM Cloud Visual Recognition	Create a custom image classification model	Create[3]: ≈ 25 sec Upload: 44.14 sec Loading[4]: 129.99 sec
Microsoft Azure Custom Vision	Create a custom image classification model	Create[5]: 1.22 sec Process[6]: 11.85 sec Upload[7]: 44.16 sec
Clever-Lab AI Framework	Create a model using solution-plugin running on a server locally.	Upload[8]: 7.75 sec Create: 1.13 sec Loading[9]: 0.95 sec

Table 2 performance comparison between different platforms.

Next, we would like to prove, that our "reusable resource" mechanism does improve efficiency in retraining the second or furthermore models with same data:

$$E = (1 - \frac{t_i}{t_o}) \times 100\%$$

In which, t_o is the time that the system needs for every retraining without applying the "reusable resource" mechanism, and t_i is the time that the system needs after applying the mechanism. Since we are not able to change the image uploading mechanism by IBM and Microsoft services, we assume easily that the $t_i = t_o - t_{upload}$, in which the t_{upload} represents to the time that uploading images from a user's computer to a server in the cloud needs.

---

[3] No SDK provided. Created via web browser, and not taking account since excessive overhead.
[4] Time that IBM Cloud Visual Recognition needed to load uploaded image from IBM Cloud Object Storage into the model.
[5] Via client python SDK.
[6] Overheads from Clever-Lab AI framework including saving, reading images from local filesystem using plugin "local_photo_loader" ≈ 1 x filesystem read and write operations + 2 x image-decoding and image-encoding (per image).
[7] Uploading decoded images from Clever-Lab AI server to Azure (including overheads of encoding decoded images back into PNG).
[8] Using plugin "local_photo_loader".
[9] Downloading image using plugin "local_photo_loader".

To prove our assumption of $t_i = t_o - t_{upload}$ is reasonable, we designed a python script using our own Cleve-Lab AI systems:

We firstly create a brand-new model with a dataset containing 126 bmp photos with foam and 46 bmp photos without foam with following script:

```
import utils, time
import resource_managing, solution_managing

rm = resource_managing.ResourceManager("127.0.0.1", 5000)
sm = solution_managing.SolutionManager("127.0.0.1", 5000)

start = time.time()
dataset0 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                              utils.getFileList("../Dataset1/foam", extension = "bmp")).uploaded
dataset1 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                              utils.getFileList("../Dataset1/nofoam", extension = "bmp")).uploaded
endUploading = time.time()

model = sm.createNewModel("edu.hm.hsieh_localfoameliminationdetector_1.0", "localModelmeasuring",
                          "its a test model")
model.feed_train_data(dataset0, 0)
model.feed_train_data(dataset1, 1)
model.train()
end = time.time()
```

Example 4 Python script with uploading resources (code 1).

Then, we reuse our uploaded image resources to create a new model with following script:

```
start = time.time()
model = sm.createNewModel("edu.hm.hsieh_localfoameliminationdetector_1.0",
                          "localModelmeasuringreusing", "its a test model")
model.feed_train_data(dataset0, 0)
model.feed_train_data(dataset1, 1)
model.train()
end = time.time()
```

Example 5 Python script with resource reusing (code 2).

We performed 3 trails and the average of results is listed in the following table:

	Time in total	Time of uploading
Code 1	17.38 sec	9.60 sec
Code 2	7.60 sec	0 sec

Table 3 Benchmarking result of Code 1 and Code 2.

According to results listed in table 2, we can confirm, our assumption on t_i is pretty correct, with error only up to 2.4%

($\frac{t_i - t_{real}}{t_{real}} \times 100\%$, in which $t_i = t_{total,code1} - t_{uploading,code1} = 17.38 - 9.6 = 7.78$ and $t_{real} = t_{total,code2} = 7.6$).

In the end, we could finally exanimate the performance improvement, the Es, of those 3 services we mentioned above.

Resultantly, we found, the “reusable resources” designed in the Clever-Lab AI framework do and could bring great performance improvements. It brings up to 25.3% ($1 - \frac{129.99}{129.99+44.14} \approx 25.3\%$) improvement to IBM Cloud Visual Recognition system; 97.3% ($1 - \frac{1.22}{1.22+44.16} \approx 97.3\%$) to Microsoft Azure Custom Vision service; also 78.8% ($1 - \frac{1.13+0.95}{7.75+1.13+0.95} \approx 78.8\%$) to our own framework.

Flexibility with the “plugin” design brings also another advantage: data security. Enterprise and laboratories users take privacy and security of data always in the first place, therefor, they often tend to use their own storages and custom protocols.

Our design is in this case useful, since the opensource property the framework owns, enterprises are always allowed to implement their own plugins as they desire.

Similarly, due to the opensource property that the python language brings, enterprise client can always inspect, adjust and even implement their security levels whenever they want to.

User-Friendliness is also a key feature that we care a lot during designing Clever-Lab AI framework.

During designing the client SDK, we emphasized on the ease-of-use of our libraries. For instance, training a new model needs with our SDK as result only 10 lines or less; training of new models with resource-reusing needs in the case even only 4 lines of python code.

```
import utils, time
import resource_managing, solution_managing

rm = resource_managing.ResourceManager("127.0.0.1", 5000)
sm = solution_managing.SolutionManager("127.0.0.1", 5000)

dataset0 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                              utils.getFileList("../Dataset1/foam", extension = "bmp")).uploaded
dataset1 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                              utils.getFileList("../Dataset1/nofoam", extension = "bmp")).uploaded

model = sm.createNewModel("edu.hm.hsieh_localfoameliminationdetector_1.0", "localModeltesting",
                          "its a test model")
model.feed_train_data(dataset0, 0)
model.feed_train_data(dataset1, 1)
model.train()
(159/0/172)
(160/0/172)
(161/0/172)
(162/0/172)
(163/0/172)
(164/0/172)
(165/0/172)
(166/0/172)
(167/0/172)
(168/0/172)
(169/0/172)
(170/0/172)
(171/0/172)
(172/0/172)
Caching image done: 2.49277400970459
Training model with 172 photos...
Training done saving model...
Model saved.
Done! Model successfully trained and saved.
```

Example 6 Code of training new models using client SDK of Clever-Lab AI framework.

```
model = sm.createNewModel("edu.hm.hsieh_localfoameliminationdetector_1.0",
                          "localModelmeasuringtestresusing", "its a test model")
model.feed_train_data(dataset0, 0)
model.feed_train_data(dataset1, 1)
model.train()
(159/0/172)
(160/0/172)
(161/0/172)
(162/0/172)
(163/0/172)
(164/0/172)
(165/0/172)
(166/0/172)
(167/0/172)
(168/0/172)
(169/0/172)
(170/0/172)
(171/0/172)
(172/0/172)
Caching image done: 2.4451379776000977
Training model with 172 photos...
Training done saving model...
Model saved.
Done! Model successfully trained and saved.
```

Example 7 Code of re-training new models using client SDK of Clever-Lab AI framework.

Moreover, implementing of data visualization tools also improves the “ease-of-use” of the client-side library further: many of our client SDK functions provide at least one data-visualization method.

```
rm.getPluginList(table_view = True)
```

	id	name	manufacturer	version	description	price description
0	edu.hm.hsieh_mylocalphotoloader_1.0	mylocalphotoloader	edu.hm.hsieh	1.0	This is a basic photo loader. This loader comp...	1 database operation per photo read/write. Pri...

Example 8 Visualization of resource management plugin list installed on server with table-view.

```
sm.getPluginList(table_view = True)
```

	id	name	manufacturer	version	description	price description
0	edu.hm.hsieh_azureimageclassifier_1.0	azureimageclassifier	edu.hm.hsieh	1.0	This model uses Microsoft Azure Custom Vision ...	Pricing see: https://azure.microsoft.com/en-us...
1	edu.hm.hsieh_localfoameliminationdetector_1.0	localfoameliminationdetector	edu.hm.hsieh	1.0	This is model extracts different features from...	Resource this model needs is only CPU resouces...

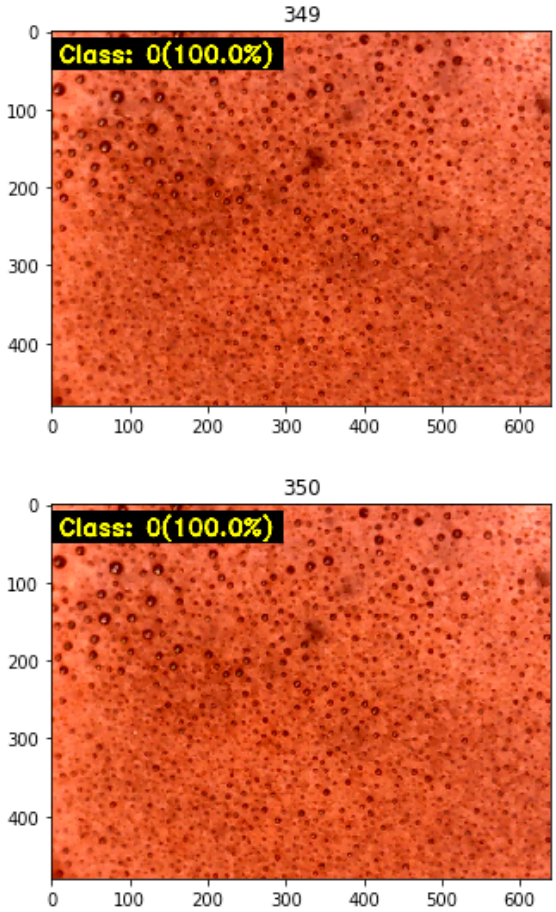
Example 9 Visualization of solution management plugin list installed on server with table-view.

```
sm.getModelList(table_view = True)
```

	id	nickname	created at	created by	plugin ID	state	description
0	5	azureTest6	Sun, 05 Apr 2020 15:19:10 GMT	webuser0	edu.hm.hsieh_azureimageclassifier_1.0	STATE_MODEL_CREATED	its a test model
1	4	localModelTest5	Sun, 05 Apr 2020 15:04:50 GMT	webuser0	edu.hm.hsieh_localfoameliminationdetector_1.0	STATE_MODEL_USABLE	its a test model
2	3	localModelTest4	Sun, 05 Apr 2020 15:04:22 GMT	webuser0	edu.hm.hsieh_localfoameliminationdetector_1.0	STATE_MODEL_CREATED	its a test model
3	2	localModelTest3	Sun, 05 Apr 2020 14:59:29 GMT	webuser0	edu.hm.hsieh_localfoameliminationdetector_1.0	STATE_MODEL_USABLE	its a test model
4	1	localModelTest2	Sun, 05 Apr 2020 14:59:07 GMT	webuser0	edu.hm.hsieh_localfoameliminationdetector_1.0	STATE_TRAINING	its a test model
5	0	localModelTest	Sun, 05 Apr 2020 14:57:48 GMT	webuser0	edu.hm.hsieh_localfoameliminationdetector_1.0	STATE_TRAINING	its a test model

Example 10 Visualization of models created on server and its metadata with table-view.

```
uploadPredict = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                                   utils.getFileList("../Dataset1/foam"))
datasetPredict = uploadPredict.uploaded
model.predictWithResourceList(datasetPredict,
                              view=solution_managing.PREDICTION_VIEW_PHOTO)
```



Example 11 Visualization of classification results with image-view.

Example plugin for Clever-Lab AI System

To show how users implement a plugin for a Clever-Lab AI system, we implemented a machine learning plugin using Microsoft's Azure Custom Vision.

Azure Custom Vision

Azure Custom Vision is the visual recognition service produced by Microsoft. In this project, we implemented a plugin for Clever-Lab AI

System that applies Azure Custom Vision APIs. With such plugin, users are able to train/predict/publish their model with Azure Custom Vision service within couples lines of code.

```
import utils
import resource_managing
import solution_managing
import notification

rm = resource_managing.ResourceManager("127.0.0.1", 5000)
sm = solution_managing.SolutionManager("127.0.0.1", 5000)

uploadResult0 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                                   utils.getFileList("../Dataset1_simple/foam"))
uploadResult1 = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                                   utils.getFileList("../Dataset1_simple/nofoam"))
dataset0 = uploadResult0.uploaded
dataset1 = uploadResult1.uploaded

model = sm.createNewModel("edu.hm.hsieh_azureimageclassifier_1.0", "azureModel1",
                          "its a test model")

model.feed_train_data(dataset0, 0)
model.feed_train_data(dataset1, 1)

model.train()
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Completed
Training done.
Done! Model successfully trained and saved.
```

Example 12 Train a model with Azure Custom Vision plugin.

```
uploadPredict = rm.uploadResources("edu.hm.hsieh_mylocalphotoloader_1.0",
                                   utils.getFileList("../Dataset1_simple/foam"))
datasetPredict = uploadPredict.uploaded
model.predictWithResourceList(datasetPredict,
                              view=solution_managing.PREDICTION_VIEW_TABLE)
```

	id	class	score
0	660	0	99.99%
1	661	1	68.29%
2	662	0	98.42%
3	663	0	90.22%
4	664	0	99.99%
5	665	0	97.12%
6	666	0	99.99%
7	667	0	76.27%
8	668	0	98.51%
9	669	0	99.99%
10	670	0	99.97%
11	671	0	99.92%
12	672	0	99.98%
13	673	0	99.07%
14	674	0	99.4%
15	675	0	99.65%
16	676	1	55.02%
17	677	0	99.99%

Example 13 Making prediction with a model trained with Azure Custom Vision plugin.

Comparison of Clever-Lab AI Platform with others

ML services

Ludwig

Ludwig is an open source machine learning toolbox made by Uber. It is built on top of TensorFlow (a framework, library, toolkit from Google, widely used in machine learning programming), and its goal is to provide a framework, with which building a machine learning model needs no more code-programming. Practically, with Ludwig, it is possible for user to train, predict and even

visualize different types of graphs, e.g. model performance per iteration, with only a single command line. (Getting Started, n.d.)

Advantages of Ludwig:

1. Trainings and predictions are made to be as easy as possible. Ideally, only one command-line per operation is needed.
2. Many built-in data preprocessing methods, machine learning models available.

Disadvantages of Ludwig:

1. Ludwig users must record their input data themselves into a CSV (Comma-Separated Value) file, which takes sometimes long and complicate without proper tools.
2. Product faces to data-scientists – users must equip with at least basic knowledge of machine learning, in order to design a new model suitable for them or to configure settings of a model correctly.

Comparison:

Our server API accepts JSON map as training-data, additionally, we provide also related functions in our client library, with which the JSON map would be generated and submitted to Clever-Lab AI servers automatically with only couples line of Python code.

Moreover, users of Clever-Lab AI system require much lower machine learning knowledge level: all those difficult proper nouns are packed into “plugins” in our system, and professional developers of plugins are also requested to provide human-readable/comprehensible descriptions to the models.

Kubeflow

Kubeflow is a machine learning tool introduced by Google aiming to make machine learning scalable based on demands, deploying of them to be easy, repeatable and transferrable by using Kubernetes. (An introduction to Kubeflow, n.d.)

Kubeflow is however a pretty different idea comparing with the Clever-Lab AI Framework. Clever-Lab AI framework focuses on resource management system with transparent privacy options, model management system being able to help users to choose between different models with ease and to get the one solving their problems the best. And Kubeflow is just a tool aiming to helping developer to deploy their product into a productive environment, and to extend their productive environment against growing demands.

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. (What is Kubernetes?, 2020)

MLFlow

MLFlow, represented by Databricks, a team founded by creators of Apache Spark team, believe that algorithm, parameters of machine learning experiments were hard to track, their results are hard to be reproduced, deploying of models and porting models to a different environment is suffering with so many opensource machine learning tools that we could find on the market. Thus, Databricks developed an open sources machine learning framework “MLFlow” to try to solve these problems.
(Matei Zaharia, June 2018)

MLFlow is a successful product, it does help data scientists to track their data experiments, such as libraries a model uses, different parameters of a model and it provides also charts, with which data scientists are able to determine quality of their data models with only a slight of glance. Additionally, it helps also computer engineers to deploy the models onto different local/cloud platforms easily, and to transfer between different platforms without worries of making it broken.
Generally speaking, MLFlow is still too difficult for general users without or with only few machines learning knowledges,

comparing to Clever-Lab AI framework. Additionally, MLFlow provides no resource managements.

Visual Recognition Services

IBM Cloud Visual Recognition

IBM Cloud Visual Recognition is an AI product published by IBM using its cloud computing platform. IBM Cloud Visual Recognition provides a pretty, user-friendly UI, with which, users are able to create a custom visual model “drag and drop”-ly without considering machine learning algorithms and writing codes.

Advantages:

1. User friendly GUI.
2. No preliminary machine learning knowledge needed. All operations can be done with clicks, drag-and-drops.
3. Auto model deployments.

Disadvantages:

1. Slow. The webpage UI takes usually 5-10 seconds to redirect itself to the next page to finish an operation. Queueing time of model training is also pretty long (at least 2-3 hours).
2. Expensive, according to prices list published by Mars 30, 2020, each training on new models costs 50 US dollars.
3. Only one default model is available. Users are not able to customize their model trainings.
4. Transferring of a trained model is not possible. Transferring a trained model is not available even between IBM Cloud accounts, needless to say, neither backing up or transferring to other platforms.

Comparison:

As one of the earliest enterprise solutions in visual recognition area with such beautiful GUI, IBM Cloud Visual Recognition was not always a perfect product. Accuracy of custom models and transparency on pricing was always their Achilles ankle. But, indeed, IBM has been making progress to make their product

better and better, price more and more favorable during these years of efforts.

As one of products that enlighten this project, we dedicated to make these parts better than IBM Cloud Visual Recognition:

1. Freedom: freedom on location, type, protocol of storage of data; freedom on choosing from different machine learning models, freedom on applying customized machine learning models.
2. Transparency: especially on pricing. Developers should reveal their plugins with descriptions of their efforts, pricing, and potential pricings in our framework.
3. Open: World in machine learning changes every day. We made our framework be opened for developers to build themselves their own customize solutions, with which their problems can be immediately and more properly solved.

Azure Custom Vision

Azure Custom Vision is online visual recognition platform represented by Microsoft. It provides beautiful GUI for user to build their models with “drag and drop”, possibility of deploying a trained model with only few clicks on its GUI, and SDKs in several popular program languages, such as, Python, Java, GO, Node.JS ...etc. The most astoundingly is, that the trained models of Azure Custom Vision can also be exported and run locally, even on mobile phones.

Advantages:

1. Much faster, comparing to IBM Cloud Visual Recognition.
2. Flexible. Models can be converted and exported, for instance, into TensorFlow models.
3. Many choices. Several models are available for user to choose to train their model, which are differently optimized to solve problems in specified areas.
4. Convenient. GUI, server-less deployment...

Disadvantages:

1. Locations of storage of uploaded photo are not clarified.
2. Models are not customizable.

Conclusion

In this project, we successfully built a framework, which, on users' side, benefits themselves with machine learning technologies and still remains, at the same time, user-friendliness; on developers side, has great potential to be easily extended; on data-scientists side, enhances user experiences with "resource-reusing" technology improving performance on training of different models using already uploaded data and, synchronously, owns client libraries simple and flexible enough to maximize user-friendliness; on enterprise side, ensures itself meeting enterprise's standard in data-protection.

Despite the trade-off between compatibility, convenience and flexibilities with little performance under few circumstances, the Clever-Lab AI do make big progress on enhancing user-experiences from different area, and thus, we believe that our framework and implementation can beat existing enterprises solutions currently on the market.

References

David Spieler. (2019). Maschinelles Lernen 01: Grundlagen [Maschine Learning 01: Fundamentales]. [PDF slides]. Munich University of applied Science Moodle: https://moodle.hm.edu/

DataFlair Team. Introduction to SVM. [Digital Image]. (2019). Introduction to Support Vector Machines. https://data-flair.training/blogs/svm-support-vector-machine-tutorial/

Bruno Stecanella. Not all hyperplanes are created equal [Digital Image]. (2017). An Introduction to Support Vector Machines (SVM):

https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

Zhang Shu-ya, Zhao Yi-ming & Li Jun-li. (2007). Algorithm and implementation of image classification based on SVM. *Computer Engineering and Applications*, 2007,43(25) p. 40-42

Getting Started. (n.d.). *Ludwig Uber*. Retrieved Mars 30, 2020 from https://uber.github.io/ludwig/getting_started/

An introduction to Kubeflow. (n.d.). *Kubeflow*. Retrieved Mars 30, 2020 from https://www.kubeflow.org/docs/about/kubeflow/

What is Kubernetes? (2020). *Cloud Native Computing Foundation*. Retrieved April 4, 2020 from https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Matei Zaharia. (June 5, 2018). Introducing MLflow: An Open Source Machine Learning Platform. https://databricks.com/blog/2018/06/05/introducing-mlflow-an-open-source-machine-learning-platform.html

Acknowledgement

I would like to express my special thanks to my supervisor Prof. Dr. Mr. Lars Wischof for providing his invaluable information, experiences and suggestions throughout this project.

Also, I would like to thank Dr. Mr. Thorsten Gresling for the chance letting me participate in this project, and also, especially, his creativities enlightening the project and his helping on experiments and related chemical knowledges.

Appendix 1 API Documents - Resource Management Module

Resource Management Module

Followings are the API that the Resource Management Module provides.

GET/resource_pluigins

Get the list of resource management plugins installed on server.

Parameters

No parameters

Responses

Code	Description

200	Successful operation.
	
	application/json
	
	• **Example Value**
	
	[
	{
	"id": 0,
	"manufacturer": "string",
	"author": "string",
	"name": "string",
	"version": "string",
	"description": "string",
	"price_description": "string"
	}
	]

GET/upload

A simple webpage to debug this API.

Parameters

No parameters

Responses

Code	Description

200	A simple webpage to debug this API.
	
	text/html
	
	• **Example Value**
	
	`<!doctype html>`
	`<title>Upload new File</title>`
	`<h1>Upload new File</h1>`
	`<form action="" method=post enctype=multipart/form-data>`
	`<input type=text name=plugin_name />`
	`<input type=file name="file[]" multiple="multiple"/>`
	`<input type=submit value=Upload />`
	`</form>`

POST /upload

Upload a file and save as a resource on the server. Supports BMP, JPG and PNG currently.

Parameters

No parameters

Request body

multipart/form-data

Information of data being fed to the server.

- **Example Value**

```
{
  "plugin_name": "string",
  "file[]": [
    "string"
```

```
  ]
}
```

Responses

Code	Description

200	Uploading completed.
	
	application/json
	
	• **Example Value**
	
	{
	  "OK": [
	  dummy.png
	  ],
	  "FAILED": [
	  broken.jpg
	  ],
	  "NOT-ALLOWED": [
	  malicious.exe
	  ]
	}

404	Plugin not found. text/plain • **Example Value** No plugin with given plugin_name was found.

GET /get_resource_metadata/{resourceID}

Get metadata of a resource from server with ID.

Parameters

resourceID *

`integer`

(path)

The ID of the resource of which metadata user wish to retrieve with.

Responses

Code	Description
200	Successful operation. application/json • **Example Value** `{ "ID": 0, "CREATED-AT": "2020-04-05T08:34:10.087Z", "CREATED-BY": "string", "PLUGIN-ID": "edu.hm.hsieh_mylocalphotoloader_1.0", "MIME": "image/png", "EXTRA-INFO": "string" }`
404	Resource with given ID not found. text/plain • **Example Value** `Not found.`
410	Resource/Plugin missing. text/plain • **Example Value** `Plugin handling this resource has been removed.`

Code	Description

```
Resource can no longer be found.
```

500

Unknown server error.

text/plain

- **Example Value**

```
Unknown error.
```

GET /get_resource_list

Get metadata of all resources.

Parameters

No parameters

Responses

Code	Description

200

Successful operation.

application/json

- **Example Value**

```
[
  {
    "ID": 0,
    "CREATED-AT": "2020-04-05T08:34:10.089Z",
    "CREATED-BY": "string",
    "PLUGIN-ID": "edu.hm.hsieh_mylocalphotoloader_1.0",
    "MIME": "image/png",
    "EXTRA-INFO": "string"
```

Code	Description

```
  }
]
```

GET /get_resource/{resourceID}

Get encoded resource from server with ID. Currently support image/*.

Parameters

resourceID * `integer` ***(path)***	ID of the resource user retrieving.	

Responses

Code	Description
200	Successful operation. */* • **Schema** `string($binary)`
404	Resource with given ID not found. text/plain • **Example Value** `Not found.`
410	Resource/Plugin missing. text/plain

Code	Description

- **Example Value**

```
Plugin handling this resource has been removed.
Resource can no longer be found.
```

500

Server error.

text/plain

- **Example Value**

```
Unknown error.
Unknown format error.
```

Schemas

Plugin {

```
id                integer($int64)
manufacturer      string
author            string
name              string
version           string
description       string
price_description string
```

}

Interface

class **ResourceLoader**(abc.ABC)

The interface for a resource management plugin.

Methods defined here:

getResource(self, database, id)

Retrieve a resource.

Args:
database (obj): The database instance spawned by server.
id (int): ID of the resource.

Returns:
object: An OpenCV matrix containing the raw resource.

getResourceExtraInfo(self, database, id)

Retrieve extra info of a resource.

Args:
database (obj): The database instance spawned by server.
id (int): ID of the resource.

Returns:
object: An opencv matrix containing the raw resource, "None" if failed

putResource(self, database, id, datastream, mime)

Decide and save a resource.

Args:
database (obj): The database instance spawned by server.
id (int): Global resource generated by module layer.
datastream (obj): InputStream of a flask's FileStorage instance.
mime (str): The MIME-type of the resource.

Returns:
bool: True if ok, False if failed.

Static methods defined here:

getAuthor()

Get the author's name.

Returns:

str: Author's name.

getDescription()

Get the description of this plugin.

!!!

This is a description for helping average users to choose plugins with.

Make sure to make the content as easy and clear as possible.

!!!

Returns:

str: Description of this plugin.

getManufacturer()

Get the manufacturer's name.

Returns:

str: Manufacturer's name.

getName()

Get the name of this plugin.

!!! Should be identical for every plugin under same manufacturer !!!

Returns:

str: Name of this plugin.

getPriceDescription()

Get the price description of this plugin.

!!!

This is a description for helping average users to choose plugins with.

Make sure to make the content as easy and clear as possible.

!!!

Returns:

str: Price description of this plugin.

getVersion()

Get the version of this plugin.

Returns:

str: Version of this plugin.

Appendix 2 API Documents - Solution Management Module

Solution Management Module

Followings are the API that the Solution Management Module provides.

Solution

GET /solution_plugins

Get the list of solutions installed on server.

Parameters

No parameters

Responses

Code	Description

200	Successful operation.
	
	application/json
	
	• **Example Value**
	
	[
	{
	"id": 0,
	"manufacturer": "string",
	"author": "string",
	"name": "string",
	"version": "string",
	"description": "string",
	"price_description": "string"
	}
	]

Model

POST /create_model

Create a brand-new machine learning model with specified solution.

Parameters

solutionID *

`string`

(query)

The solution that the model creating with.

Request body

application/json

Basic details of the new model.

- **Example Value**

```
{
  "nickname": "string",
  "description": "string"
}
```

Responses

Code	Description
200	ID of the new model. text/plain • **Example Value** `7`
400	Bad request. text/plain

- **Example Value**

```
Solution ID was not specified.
Nickname is already used, use another.
```

404

Such solution not found.

text/plain

- **Example Value**

```
Solution with specified ID is not found.
```

410

Plugin in charge failed to create the new model.

text/plain

- **Example Value**

```
Failed to create a new model.
```

GET /models

Get the list of models' metadata created on server.

Parameters

No parameters

Responses

Code	Description
200	successful operation application/json

Code **Description**

- **Example Value**


```
[
  {
    "id": 0,
    "nickname": "string",
    "created_at": "2020-04-05T02:50:44.113Z",
    "create_by": "string",
    "plugin_id": "string",
    "state": "STATE_MODEL_CREATED",
    "description": "string"
  }
]
```


POST /feed_train_data

Feed training data to a model.

Parameters

modelID *

`string`

(query)

ID or nickname of the model to feed data toward.

Request body

application/json

Training data. Dictionary of resourceID:tag of the resource.

- **Example Value**


```
{}
```


Responses

Code	Description
200	OK. text/plain • **Example Value** `OK`
400	Bad request. text/plain • **Example Value** `Bad training data.`
404	Solution not found. text/plain • **Example Value** `No model with given modelID was found.`
410	Plugin in charge failed to feed training data to the model. text/plain • **Example Value** `model can not be found from plugin database`

GET /train_model

Train the specified model.

Parameters

modelID *

`string`

(query)

ID or nickname of the model to feed data toward.

Responses

Code	Description

200	Training process started successfully. Server will keep sending training logs instantaneously in the response body.
	
	text/plain
	
	• **Example Value**
	
	`OK`
	
	Headers:
	
	Name **Description**
	
	`X-Content-Type-Options` `Always be set with nosniff for supporting of showing responses on browsers.`

400	Bad request.
	
	text/plain
	
	• **Example Value**
	
	`Bad training parameter format.`
	`Bad training data format.`
	`Model with state [STATE_TRAINING|STATE_MODEL_USABLE] is no longer allowed to give training data.`

Code	Description
404	Such solution not found. text/plain • **Example Value** `No model with given modelID was found.`

POST /train_model

Train the specified model.

modelID *

`string`

(query)

ID or nickname of the model to feed data toward.

Request body

application/json

Custom parameters for the training.

- **Example Value**

```
{}
```

Responses

Code Description

200

Training process started successfully. Server will keep sending training logs instantaneously in the response body.

text/plain

- **Example Value**

```
OK
```

Headers:

Name	Description	Type
X-Content-Type-Options	Always be set with nosniff for supporting of showing streaming responses on browsers.	string *Example: nosniff*

400

Bad request.

text/plain

- **Example Value**

```
Bad training parameter format.
Can't train model with state:
[STATE_MODEL_CREATED|STATE_TRAINING|STATE_MODEL_USABLE].
```

404

Model not found.

text/plain

- **Example Value**

```
No model with given modelID was found.
```

GET /predict

A simple webpage to debug this API.

Parameters

No parameters

Responses

Code	Description
200	A simple webpage to debug this API. text/html • **Example Value**

```
<!doctype html>
  <title>Upload new File to predit</title>
  <h1>Upload new File</h1>
  <form action="" method=post enctype=multipart/form-data>
     <input type=text name="modelName"/>
     <input type=file name="file[]" multiple="multiple"/>
     <input type=submit value=Upload />
</form>
```

POST /predict

Upload a resource and make prediction with a trained model. Supports BMP, JPG and PNG currently.

Parameters

No parameters

Request body

multipart/form-data

Information of data being feeded to be prediction.

- **Example Value**


```
{
  "modelID": "string",
  "file[]": [
    "string"
  ]
}
```


Responses

Code	Description
200	Successfully triggered prediction oepration. application/json • **Example Value**


```
{
  "ISOK": true,
  "RESULT": {
    "test.png": {
      "CLASS": "1",
      "SCORE": 0.95
    }
  }
}
{
  "ISOK": false
}
```


400	Bad request. text/plain • **Example Value**

Code Description

```
Can't predict with a untrained model.
```

404

Model not found.

text/plain

- **Example Value**

```
No model with given modelID was found.
```

POST /predict_w_list

Upload a list of resource ID and make prediction with resources listed.

Parameters

modelID * `string` ***(query)***	ID or nickname of the model to feed data toward.

Code Description

200

Successfully triggered prediction operation.

application/json

- **Example Value**


```
{
  "ISOK": true,
  "RESULT": {
    "175": {
      "CLASS": "1",
      "SCORE": 0.95
    }
  }
}
{
  "ISOK": false
}
```


400

Bad request.

text/plain

- **Example Value**

```
Can't predict with a untrained model.
```

404

Model not found.

text/plain

- **Example Value**

```
No model with given modelID was found.
```

Request body

application/json

Information of data being fed to be prediction.

- **Example Value**

```
[
  175
]
```

Responses

Schemas

Plugin {

id	integer($int64)
manufacturer	string
author	string
name	string
version	string
description	string
price_description	string

}

Model {

id	integer($int64)
nickname	string
created_at	string($date-time)
create_by	string
plugin_id	string
	The solution ID that the model created with.

```
    state                   stringEnum:
                            Array [ 4 ]


    description             string


}
```

Interface

class **SolutionManager**(abc.ABC)

The interface for a solution management plugin.

Methods defined here:

createModel(self, database, id)

Create a brand-new model.

Args:

database (obj): The database instance spawned by server.

id (int): Global modelID generated by module layer.

Returns:

bool: True if model created, resources for new model reserved successfully.

feedTrainData(self, database, id, trainData)

Feed training data to the model.

Args:

database (obj): The database instance spawned by server.

id (int): ID of the model to feed data with.

trainData(dict<int, string>):

New training data, dict of resourceID:class-tag of the resource.

Returns:

bool: True if training data fed successfully.

str: Error message if operation failed.

predictWithData(self, database, modelID, resourceNames, rawDatas, onFinished)

Make prediction with given list of rawDatas.

Args:

database (obj): The database instance spawned by server.

modelID (int): The model's ID to predict with.

resourceNames (List<str>): List of filename of uploaded resources.

rawDatas (List<int>): List of uploaded resources.

onFinished (func(obj)):

A callback function called exactly once whenever prediction is completed.

Type of prediction result: dict<str>obj

Format of prediction result:

ISOK (bool): Is prediction performed successfully.

RESULT (dict<int>str):

The dict of the prediction result. Form of resourceID:class-tag of the resource.

Returns:

No Return.

predictWithID(self, database, modelID, inputDataIDs, onFinished)

Make prediction with given list of resourceIDs.

Args:

database (obj): The database instance spawned by server.

modelID (int): The model's ID to predict with.

inputDataIDs (List<int>): List of resourceIDs.

onFinished (func(obj)): A callback function called exactly once whenever prediction is completed.

Type of prediction result: dict<str,obj>

Format of prediction result:

ISOK (bool): Is prediction performed successfully.

RESULT (dict<int,str>):

The dict of the prediction result. Form of resourceID:class-tag of the resource.

Returns:

No Return.

trainModel(self, database, modelID, parameters, onMessage, onFinished)

Start to train the model.

Args:
database (obj): The database instance spawned by server.
id (int): ID of the model to train.
parameters(dict<str,obj>): Parameters provided from user to costomize training.
onMessage (func(str)): A callback function for exporting training logging to user.
onFinished (func(bool)): A callback function called exactly once whenever training is co
mpleted. True: if all ok; False: if failed.

Returns:
No Return.

Static methods defined here:

getAuthor()

Get the author's name.

Returns:
str: Author's name.

getDescription()

Get the description of this plugin.
!!!
This is a description for helping average users to choose plugins with.
Make sure to make the content as easy and clear as possible.
!!!

Returns:
str: Description of this plugin.

getManufacturer()

Get the manufacturer's name.

Returns:
str: Manufacturer's name.

getName()

Get the name of this plugin.
!!! Should be identical for every plugin under same manufacturer !!!

Returns:

str: Name of this plugin.

getPriceDescription()

Get the price description of this plugin.

!!!

This is a description for helping average users to choose plugins with.

Make sure to make the content as easy and clear as possible.

!!!

Returns:

str: Price description of this plugin.

getVersion()

Get the version of this plugin.

Returns:

str: Version of this plugin.

Appendix 3 Client SDK and Notification Module

Solution Manager

class **SolutionManager**()

SolutionManager(address, port)

Client library for managing solutions and models.

Methods defined here:

__init__(self, address, port)

Constructor.

Args:

address (str): IP or URL of the server.

port (str): Port of the server.

Returns:

No Returns.

createNewModel(self, solutionID, modelNickName, modelDescription)

Create a new model with specified solution.

Args:

solutionID (str): ID of the solution plugin.

modelNickName (str):

Nickname of the new model. Must be server-

wide unique, otherwise, 400 Bad Request will be produced.

modelDescirption (str): Random description for the new model.

Returns:

newModel (Model): An model object as handle.

Raises:

RuntimeError: If failed to fetch valid response from server.

getModelList(self, table_view=False)

Get the list of models created on the server.

Args:

table_view (bool): Whether render response with table-view.

Returns:

response (obj):

Json by default; pandas Dataframe by table-

view mode. Contains colums: ID, CREATED-BY, CREATED-AT, PLUGIN-

ID, MIME, EXTRA_INFO.

Raises:

RuntimeError: If failed to fetch valid response from server.

getPluginList(self, table_view=False)

Get the list of solution managment plugins installed on the server.

Args:

table_view (bool): Whether render response with table-view.

Returns:

response (obj): Json by default; pandas Dataframe by table-

view mode. Contains columns: id, name, manufacturer, version, description and price des

cription.

Raises:

RuntimeError: If failed to fetch valid response from server.

modelFactory(self, modelID)

A factory method to create an instance model.

Args:

modelID (int): ID of the model.

Returns:

response (obj): Json by default; pandas Dataframe by table-view mode.

Raises:

RuntimeError: If failed to fetch valid response from server.

class **Model**()

Model(modelID, serverAddres, serverPort)

Model instance and its APIs.

Methods defined here:

__init__(self, modelID, serverAddres, serverPort)

Constructor

Args:

modelID (int): ID of the model.

serverAddress (str): IP or URL of the server.

serverPort (str): Port of the server.

feed_train_data(self, resourceList, dataClass)

Feed training data into model.

Args:

resourceList (obj): A resource_managing.ResourceList resource list.

dataClass (str): Class-tag that resources in the resourceList will be labeled with.

Returns:

result (str): "OK" if ok.

Raises:

RuntimeError: If failed to fetch valid response from server.

getID(self)

Get the id of this model.

Returns:

modelID (int): ID of this model.

predict(self, resourceFileName, view=None)

Upload and make prediction on a file.

Args:

resourceFileName (str): Filename of the resource to be predicted.

view (int):

Type of renderer appling on prediction-

responses, supporting PREDICTION_VIEW_TABLE and

REDICTION_VIEW_PHOTO.

Returns:

result (obj):

Parsed Json object if no view-type is set. pandas.DataFrame if table-view set. Nothing returned by photo-view but will displays rendered photos automatically.

Raises:

RuntimeError: If failed to fetch valid response from server.

predictWithResourceList(self, resourceList, view=None)

Make predictions on the resources listed.

Args:

resourceList (obj): A resource_managing.ResourceList resource list.

view (int):

Type of renderer appling on prediction-responses, supporting PREDICTION_VIEW_TABLE and REDICTION_VIEW_PHOTO.

Returns:

result (obj): Parsed Json object if no view-type is set. pandas.DataFrame if table-view set. Nothing returned by photo-view but will displays rendered photos automatically.

Raises:

RuntimeError: If failed to fetch valid response from server.

predictWithResources(self, resourceFilenameList, view=None)

Upload files and make prediction on them.

Args:

resourceFilenameList (List<str>): List of filenames that will be uploaded.

view (int):

Type of renderer appling on prediction-responses, supporting PREDICTION_VIEW_TABLE and REDICTION_VIEW_PHOTO.

Returns:

result (obj):

Parsed Json object if no view-type is set. pandas.DataFrame if table-
view set. Nothing returned by photo-
view but will displays rendered photos automatically.

Raises:

RuntimeError: If failed to fetch valid response from server.

streamPredict(self, ffmpegPath, captureInterval, experimentLength, cameraName, targetPredictionClass, notificator, messageType=0, triggerMode=1, classDetectedThreshold=1, scoreThreshold=1.0)

Make autonomous real-time classification with cameras.

Args:

ffmpegPath (str): Path of the ffmpeg execuable.

captureInterval (int): Interval of taking a photo.

experimentLength (int): Interval of the whole experiment.

cameraName (int): Name of the camera connecting to.

targetPredictionClass (str): The target class this script to be aware of.

notificator (Notifcation): The notificator to send message with.

messageType (int): Type of the notificator, e.g. notification.MESSAGE_TYPE_TEXT.

triggerMode (int): Mode of trigger producing notifications.

classDetectedThreshold (int):

Threshold of trigger on number of detections against target-class.

classDetectedThreshold (float): Threshold of trigger on confidience of target-class.

train(self, parameter=None)

Train this model with dataset fed earlier.

Args:

parameter (obj): A dict containing parameters for customizing training.

Returns:

training_logs (str): Live logging of the training process.

Raises:

RuntimeError: If failed to fetch valid response from server.

Solution Managing - Enums

Data

NUMBER_DETECTION = 1
SCORE = 2
NUMBER_DETECTION_AND_SCORE = 3
NUMBER_DETECTION_OR_SCORE = 4

PREDICTION_VIEW_TABLE = 1
PREDICTION_VIEW_PHOTO = 2

Resource Manager

class **ResourceManager**()

ResourceManager(address, port)

A Clever-Lab AI server resource manager.

Methods defined here:

__init__(self, address, port)

Constructor.

Args:

address (str): IP or URL of the server.

port (str): Port of the server.

getPluginList(self, table_view=False)

Get the list of resource management plugins installed on the server.

Args:

table_view (bool): Whether render response with table-view.

Returns:

response (obj):

Json by default; pandas Dataframe by table-

view mdoe. Contains columns: id, name, manufacturer, version, description and pric

e description.

Raises:

RuntimeError: If failed to fetch valid response from server.

getResourceList(self, table_view=False)

Get the list of resources created on the server.

Args:

table_view (bool): Whether render response with table-view.

Returns:

response (obj):

Json by default; pandas Dataframe by table-

view mdoe. Contains columns: ID, CREATED-BY, CREATED-AT, PLUGIN-

ID, MIME, EXTRA_INFO.

Raises:

RuntimeError: If failed to fetch valid response from server.

uploadResources(self, pluginID, resourceFileList)

Upload resources with a list of files.

Args:

pluginID (str): ID of the plugin handles this uploading.

resourceFileList (List<str>): List of filenames of uploading resources.

Returns:

response (obj):

A _UploadResourceResult object, containing fields uploaded, rejected, failed.

Raises:

RuntimeError: If failed to fetch valid response from server.

class **ResourceList**()

ResourceList(resourceIDMapList)

A class maintaining list of resources.

Methods defined here:

__init__(self, resourceIDMapList)

Constructor.

Args:

resourceIDMapList (obj):

Can be either List<int>, list of resourceIDs. Or Dict<int, str>, Dict of resourceID to its original filenames.

getIDs(self)

Get ID of the resources tracking by this instance.

Returns:

idList (List<int>): List of the resourceIDs.

getResourceIDFilenameMap(self)

Get the map of resourceIDs to their original filename.

Returns:

resourceIDMapList (obj): Either List<int> or Dict<int, str>

merge(self, another)

Merge this instance with another.

Args:

another (ResourceList): another instance.

Notification Module

Interface

class **Notification**(abc.ABC)

Interface of a notification class.

Methods defined here:

__enter__(self)

__exit__(self, exeType, exeValue, exeTraceback)

sendMessage(self, messageType, payload)

```
        Send a message.

        Args:
            messageType (int): type of the message in the payload.
            payload (obj): payload.
```

Notificators

class **ConsoleNotification**(Notification)

A notificator print notifications directly on a console.

Methods defined here:

__enter__(self)

__exit__(self, exeType, exeValue, exeTraceback)

sendMessage(self, messageType, payload)

Print notification directly on the console if payload is a text message, otherwise, save in coming message under CWD and print a text message on the console.

Args:

messageType (int):

type of the payload. Either MESSAGE_TYPE_TEXT or MESSAGE_TYPE_IMAGE currently

---

class **MQTTNotification**(Notification)

MQTTNotification(brokerAddress, brokerPort, username, password, clientID, topicID)

A notificator sending notifications via MQTT.

Methods defined here:

__enter__(self)

__exit__(self, exeType, exeValue, exeTraceback)

__init__(self, brokerAddress, brokerPort, username, password, clientID, topicID)

Initialize self. See help(type(self)) for accurate signature.

sendMessage(self, messageType, payload)

Send notification directly via MQTT protocol if payload is a text message, otherwise save i ncoming message under CWD and send a text message via MQTT.

Args:

messageType (int):

Type of the payload. Supports MESSAGE_TYPE_TEXT and MESSAGE_TYPE _IMAGE currently.

Notificators Enums

Data

MESSAGE_TYPE_IMAGE= 1

MESSAGE_TYPE_TEXT= 0

Appendix 4 Source Codes

Server

Project tree:

```
Server
├── main.py
├── requirement.txt
├── resource_loader.py
├── server_api.py
├── solution_manager.py
├── utils.py
├── resource_loader_plugins
│   ├── __init__.py
│   ├── interface.py
│   └── local_photo_loader.py
└── solution_plugins
    ├── __init__.py
    ├── interface.py
    ├── azure_image_classifier.py
    ├── interface.py
    └── local_foam_elimination_detector.py
    └── utils.py
```

```
##################################
    main.py
##################################
    import server_api
    import solution_manager
    import resource_loader
    from flask import Flask
    import psycopg2

    database = psycopg2.connect(
        database="[DATABASE]",="postgres",
        user="[USERNAME]",="postgres",
        password="[PASSWORD]",="postgres",
```

```
    host="[HOST]",="127.0.0.1",
    port="5432")

# initialize global setting datatable
session = database.cursor()
session.execute("CREATE TABLE IF NOT EXISTS resource_indexes\
            (\
                module_id text NOT NULL,\
                next_index integer NOT NULL,\
                PRIMARY KEY (module_id)\
            )\
            WITH (\
                OIDS = FALSE\
            );")
database.commit()

app = Flask(__name__)

# initialize different modules
serverAPI = server_api.ServerAPI()

resource_loader.initialize(database, app, serverAPI)

solution_manager.initialize(database, app, serverAPI)


if __name__ == "__main__":
    app.run()
```

##################################

resourece_loader.py

##################################

```
# imports
import os
import importlib
import utils
import mimetypes
```

```
import cv2
from flask import jsonify, make_response, request

# static parameters
MODULE_TAG_RESOURCE = "resource_manager"

ALLOWED_EXTENSIONS = ['png', 'jpg', 'jpeg', 'bmp']

# global runtime parameters
resourcePlugins = {}
mimeCache = {}


def initialize(database, app, serverAPI):
    # initialize resource database
    session = database.cursor()
    session.execute("CREATE TABLE IF NOT EXISTS resources\
                (\
                    id integer NOT NULL,\
                    created_at timestamp without time zone NOT NULL,\
                    create_by text NOT NULL,\
                    plugin_id text NOT NULL,\
                    mime     text NOT NULL,\
                    PRIMARY KEY (id)\
                )\
                WITH (\
                    OIDS = FALSE\
                );")
    database.commit()

    # initialize resource index
    session = database.cursor()
    session.execute("INSERT INTO resource_indexes (module_id,
next_index)\
                SELECT '" + MODULE_TAG_RESOURCE + "', 0\
                WHERE NOT EXISTS (SELECT 1 FROM resource_indexes WHERE
module_id = '" + MODULE_TAG_RESOURCE + "');")
    database.commit()
```

```
    # load resource loaders
    for plugin in os.listdir("./resource_loader_plugins"):
        pluginFilename = plugin

        if pluginFilename.endswith(".py") and not
pluginFilename.startswith(
                "interface") and not
pluginFilename.startswith("__init__"):
            pluginModule = importlib.import_module(
                "resource_loader_plugins." + pluginFilename[:-3])

            try:
                pluginClass = pluginModule.reflector()
            except BaseException:
                print("ERROR Resource plugin", pluginFilename,
                      "was refused to load: ERR_NO_REFLECTOR_METHOD")
                continue

            try:
                instanceID = utils.getPluginID(pluginClass)
            except BaseException:
                print("ERROR Resource plugin", pluginFilename,
                      "was refused to load: ERR_METADATA_INVALID")
                continue

            databaseID = utils.getPluginDataTableID(instanceID)

            try:
                # create a datatable for this plugin if it not exists
                session = database.cursor()
                session.execute("CREATE TABLE IF NOT EXISTS " + databaseID
+ "\
                                (\
                                    id integer NOT NULL,\
                                    remote_id text NOT NULL,\
                                    extra_info json,\
                                    PRIMARY KEY (id)\
```

```
                        )\
                        WITH (\
                            OIDS = FALSE\
                        );")
            database.commit()

        except BaseException:
            print("ERROR Resource plugin", pluginFilename,
                 "was refused to load: ERR_DATATABLE_CREATE_FAILED")
            continue

        resourcePlugins[instanceID] = {
            "instance": pluginClass(
                databaseID,
                serverAPI),
            "class": pluginClass}

print("INFO", len(resourcePlugins), "resource plugins loaded.")

@app.route('/resource_pluigins', methods=['GET'])
def resourceLoaderInfoProvider():
    loaders = []
    for loader in resourcePlugins:
        pluginClass = resourcePlugins[loader]["class"]

        loaders.append({
            "id": utils.getPluginID(pluginClass),
            "manufacturer": pluginClass.getManufacturer(),
            "author": pluginClass.getAuthor(),
            "name": pluginClass.getName(),
            "version": pluginClass.getVersion(),
            "description": pluginClass.getDescription(),
            "price_description": pluginClass.getPriceDescription(),
        })

    return jsonify(loaders)

@app.route('/upload', methods=['GET', 'POST'])
```

```
    def dataUploader():
        if request.method == 'POST':
            files = request.files.getlist("file[]")
            pluginName = request.form["plugin_name"]

            print("DEBUG", len(files), "files uploading.")

            uploadOK = []
            uploadFailed = []
            uploadNotallowed = []

            if pluginName and (pluginName in resourcePlugins):
                for file in files:

                    if file and utils.isFileAllowed(
                            file.filename, ALLOWED_EXTENSIONS):
                        mime = file.mimetype
                        if len(mime) < 3:  # mime type is invalid, guess it
                            mime = _getMimeFromExtension(file.filename)

                        newID = utils.reserveNewID(
                            database, MODULE_TAG_RESOURCE)
                        result =
resourcePlugins[pluginName]["instance"].putResource(
                            database, newID, file, mime)

                        if result:
                            print(
                                "DEBUG Plugin",
                                pluginName,
                                "saved a resource successfully: " +
                                file.filename)

                            # update resource record
                            session = database.cursor()
                            session.execute(
                                "INSERT INTO resources (id, created_at,
create_by, plugin_id, mime) VALUES (%s, NOW(), %s, %s, %s)",
```

```
                        (newID,
                         "webuser0",
                         pluginName,
                         mime))
                    database.commit()

                    resourceIDMap = {}
                    resourceIDMap[newID] = file.filename
                    uploadOK.append(resourceIDMap)

                else:
                    print(
                        "ERROR",
                        pluginName,
                        "failed to store a uploaded photo.")
                    uploadFailed.append(file.filename)
            else:
                print(
                    "ERROR file",
                    file.filename,
                    ", file type is not allowed.")
                uploadNotallowed.append(file.filename)

            return jsonify(
                {"OK": uploadOK, "FAILED": uploadFailed, "NOT-ALLOWED":
uploadNotallowed}), 200
        else:
            return "Plugin " + pluginName + " not found.", 404

    return '''
        <!doctype html>
        <title>Upload new File</title>
        <h1>Upload new File</h1>
        <form action="" method=post enctype=multipart/form-data>
            <input type=text name=plugin_name />
            <input type=file name="file[]" multiple="multiple"/>
            <input type=submit value=Upload />
        </form>
```

```
        '''

    @app.route('/get_resource_metadata/<resourceID>', methods=['GET'])
    def resourceMetadataGetter(resourceID):
        record, err = getResourceMetadata(database, resourceID)

        if err:
            if err == "err_not_found":
                return "Not found.", 404

            elif err == "err_plugin_removed":
                return "Plugin handling this resource has been removed.",
410

            elif err == "err_plugin_record_removed":
                return "Resource can no longer be found.", 410

            else:
                return "Unknown error.", 500

        return jsonify(record)

    @app.route('/get_resource_list', methods=['GET'])
    def resourceListGetter():
        return jsonify(getAllResourceMetadata(database))

    @app.route('/get_resource/<resourcID>', methods=['GET'])
    def resourceGetter(resourcID):

        resource, mime, err = getResource(database, resourcID)

        if err:
            if err == "err_not_found":
                return "Not found.", 404

            elif err == "err_plugin_removed":
                return "Plugin handling this resource has been removed.",
410
```

```
            elif err == "err_plugin_record_removed":
                return "Resource can no longer be found.", 410

            else:
                return "Unknown error.", 500

        if mime.startswith("image/"):
            _, buffer = cv2.imencode('.png', resource)
            response = make_response(buffer.tobytes())
            response.headers['Content-Type'] = 'image/png'
        else:
            return "Unknown format error.", 500

        return response

# APIs


def getResource(database, resourceID):
    # get the record
    session = database.cursor()
    session.execute(
        "SELECT created_at, create_by, mime, plugin_id FROM resources
WHERE id=%s",
        (resourceID,
         ))
    result = session.fetchone()
    session.close()

    if result:
        _, _, mime, pluginID = result
        if pluginID not in resourcePlugins:
            return None, None, "err_plugin_removed"

        plugin = resourcePlugins[pluginID]["instance"]
        resource = plugin.getResource(database, resourceID)
```

```
        if resource is None:
            return None, None, "err_plugin_record_removed"

        return resource, mime, None

    else:
        return None, None, "err_not_found"


def getResourceMetadata(database, resourceID):
    # get the record
    session = database.cursor()
    session.execute(
        "SELECT created_at, create_by, mime, plugin_id FROM resources
WHERE id=%s",
        (resourceID,
         ))
    result = session.fetchone()
    session.close()

    if result:
        createdAt, createBy, mime, pluginID = result

        if pluginID not in resourcePlugins:
            return None, "err_plugin_removed"

        plugin = resourcePlugins[pluginID]["instance"]
        extraInfo = plugin.getResourceExtraInfo(database, resourceID)

        if extraInfo:
            return {"CREATED-AT": createdAt, "CREATED-BY": createBy,
                    "PLUGIN-ID": pluginID, "EXTRA-INFO": extraInfo, "MIME":
mime}, None
        else:
            return None, "err_plugin_record_removed"

    else:
        return None, "err_not_found"
```

```
def getAllResourceMetadata(database):
    # get the record
    session = database.cursor()
    session.execute(
        "SELECT id, created_at, create_by, plugin_id, mime FROM resources
ORDER BY id DESC;")
    result = session.fetchall()
    session.close()

    resourceLists = []

    for resource in result:
        resourceID, createdAt, createBy, pluginID, mime = resource

        if pluginID in resourcePlugins:

            plugin = resourcePlugins[pluginID]["instance"]
            extraInfo = plugin.getResourceExtraInfo(database, resourceID)

            if extraInfo:
                resourceLists.append({"ID": resourceID,
                                      "CREATED-AT": createdAt,
                                      "CREATED-BY": createBy,
                                      "PLUGIN-ID": pluginID,
                                      "MIME": mime,
                                      "EXTRA-INFO": extraInfo})

    return resourceLists


def _getMimeFromExtension(filename):

    if filename in mimeCache:
        return mimeCache[filename]
    else:
        typeGuess, _ = mimetypes.guess_type(filename)
        if typeGuess:
```

```
            mimeCache[filename] = typeGuess
            return typeGuess
        else:
            return None


##################################
    server_api.py
##################################
    import resource_loader


    class ServerAPI():

        def __init__(self):
            self.getResource = resource_loader.getResource
            self.getResourceMetadata = resource_loader.getResourceMetadata


##################################
    solution_manager.py
##################################
    # imports
    import os
    import importlib
    import utils
    import queue
    import threading
    from flask import jsonify, request, Response
    from resource_loader import ALLOWED_EXTENSIONS, _getMimeFromExtension
    import json
    from cv2 import cv2 as cv
    import numpy as np

    # static parameters
    MODEL_TAG_MODEL = "model_manager"

    SOLUTION_PLUGINS_PATH = "solution_plugins"
```

```
STATE_MODEL_CREATED = 0
STATE_DATA_FEEDING = 10
STATE_TRAINING = 20
STATE_MODEL_USABLE = 30

stateID2State = {
    0: "STATE_MODEL_CREATED",
    10: "STATE_DATA_FEEDING",
    20: "STATE_TRAINING",
    30: "STATE_MODEL_USABLE",
}


solutionPlugins = {}


def initialize(database, app, serverAPI):
    # initialize model database
    session = database.cursor()
    session.execute("CREATE TABLE IF NOT EXISTS models\
                (\
                    id integer NOT NULL,\
                    nickname text,\
                    created_at timestamp without time zone NOT NULL,\
                    create_by text NOT NULL,\
                    plugin_id text NOT NULL,\
                    state integer NOT NULL,\
                    description text,\
                    PRIMARY KEY (id)\
                )\
                WITH (\
                    OIDS = FALSE\
                );")
    database.commit()

    # initialize model-resource index
    session = database.cursor()
```

```
    session.execute("INSERT INTO resource_indexes (module_id,
next_index)\
                    SELECT '" + MODEL_TAG_MODEL + "', 0\
                    WHERE NOT EXISTS (SELECT 1 FROM resource_indexes WHERE
module_id = '" + MODEL_TAG_MODEL + "');")
    database.commit()

    # load solution managers
    for plugin in os.listdir("./" + SOLUTION_PLUGINS_PATH):
        pluginFilename = plugin

        if pluginFilename.endswith(".py") and not
pluginFilename.startswith(
                "interface") and not
pluginFilename.startswith("__init__"):
            pluginModule = importlib.import_module(
                SOLUTION_PLUGINS_PATH + "." + pluginFilename[:-3])

            try:
                pluginClass = pluginModule.reflector()
            except BaseException:
                print("ERROR Solution plugin", pluginFilename,
                      "was refused to load: ERR_NO_REFLECTOR_METHOD")
                continue

            try:
                instanceID = utils.getPluginID(pluginClass)
            except BaseException:
                print("ERROR Solution plugin", pluginFilename,
                      "was refused to load: ERR_METADATA_INVALID")
                continue

            databaseID = utils.getPluginDataTableID(instanceID)

            try:
                # create a datatable for this plugin if it not exists
                session = database.cursor()
```

```
                session.execute("CREATE TABLE IF NOT EXISTS " + databaseID
+ "\
                                (\
                                    id integer NOT NULL,\
                                    remote_id text NOT NULL,\
                                    training_data json,\
                                    extra_info json,\
                                    PRIMARY KEY (id)\
                                )\
                                WITH (\
                                    OIDS = FALSE\
                                );")
                database.commit()

            except BaseException:
                print("ERROR Solution plugin", pluginFilename,
                      "was refused to load: ERR_DATATABLE_CREATE_FAILED")
                continue

            solutionPlugins[instanceID] = {
                "instance": pluginClass(
                    databaseID,
                    serverAPI),
                "class": pluginClass}

    print("INFO", len(solutionPlugins), "solution plugins loaded.")

    @app.route('/solution_plugins', methods=['GET'])
    def solutionPluginInfoProvider():
        loaders = []

        for loader in solutionPlugins:
            pluginClass = solutionPlugins[loader]["class"]

            loaders.append({
                "id": utils.getPluginID(pluginClass),
                "manufacturer": pluginClass.getManufacturer(),
                "author": pluginClass.getAuthor(),
```

```
                "name": pluginClass.getName(),
                "version": pluginClass.getVersion(),
                "description": pluginClass.getDescription(),
                "price_description": pluginClass.getPriceDescription(),
            })

        return jsonify(loaders)

    @app.route('/create_model', methods=['POST'])
    def modelCreater():
        solutionID = request.args.get('solutionID')

        if solutionID is None:
            return "Solution ID was not specified.", 400

        elif solutionID not in solutionPlugins:
            return "Solution with specified ID is not found.", 404
        else:
            modelInfo = request.get_json()
            if "nickname" in modelInfo:
                nickName = modelInfo['nickname']
                # check is nickname unique
                session = database.cursor()
                session.execute("SELECT 1 from models WHERE nickname=%s;",
(nickName,))
                numberOfNickname = session.rowcount
                session.close()

                if numberOfNickname > 0:
                    return "Nickname is already used, use another.", 400

            else:
                nickName = None

            if "description" in modelInfo:
                description = modelInfo['description']
            else:
                description = None
```

```
            newID = utils.reserveNewID(database, MODEL_TAG_MODEL)
            result =
solutionPlugins[solutionID]["instance"].createModel(database, newID)

            # write the record of the created model into database
            session = database.cursor()
            session.execute(
                "INSERT INTO models (id, nickname, created_at, create_by,
plugin_id, state, description) VALUES (%s, %s, NOW(), %s, %s, %s, %s);",
                (newID,
                 nickName,
                 "webuser0",
                 solutionID,
                 STATE_MODEL_CREATED,
                 description))

            database.commit()

            if result:
                return str(newID), 200
            else:
                return "Failed to create a new model.", 410

    @app.route('/models', methods=['GET'])
    def modelInfoProvider():
        session = database.cursor()
        session.execute(
            "SELECT id, nickname, created_at, create_by, plugin_id, state,
description FROM models ORDER BY id DESC;")
        results = session.fetchall()
        session.close()

        models = []

        for result in results:
            id, nickname, created_at, create_by, plugin_id, state,
description = result
```

```
            models.append({
                "id": id,
                "nickname": nickname,
                "created_at": created_at,
                "create_by": create_by,
                "plugin_id": plugin_id,
                "state": stateID2State[state],
                "description": description,
            })

        return jsonify(models)

    @app.route('/feed_train_data', methods=['POST'])
    def feedModelTrainData():
        modelName = request.args.get('modelID')  # can be number:
modelID, or model nickname

        # parse body: training data
        if not request.is_json:
            return "Bad training data.", 400

        trainingData = request.get_json()
        if trainingData is None:
            return "Bad training data format.", 400

        # get modelID from given id or nickname
        session = database.cursor()
        try:
            queryModelID = int(modelName)
            queryModelNickName = None
        except BaseException:
            queryModelID = -1
            queryModelNickName = modelName

        session.execute(
            "SELECT id, plugin_id, state FROM models WHERE id=%s OR
nickname=%s;",
```

```
            (queryModelID,
             queryModelNickName))
        result = session.fetchone()
        session.close()

        if result is None:
            return "No model with given modelID was found.", 404

        modelID, pluginID, modelState = result

        if modelState > STATE_DATA_FEEDING:
            return "Model with state " + \
                stateID2State[modelState] + " is no longer allowed to give
training data.", 400

        result, err =
solutionPlugins[pluginID]["instance"].feedTrainData(
            database, modelID, trainingData)
        if result:
            # update model state
            session = database.cursor()
            session.execute("UPDATE models SET state=%s WHERE id=%s;",
                            (STATE_DATA_FEEDING, modelID))
            database.commit()

            return "OK"

        else:
            return err, 410

    @app.route('/train_model', methods=['GET', 'POST'])
    def modelTrainer():
        modelName = request.args.get('modelID')  # can be number:
modelID, or model nickname

        trainingParameter = None

        if request.is_json:
```

```
            trainingParameter = request.get_json()
            if trainingParameter is None or not
isinstance(trainingParameter, dict):
                return "Bad training parameter format.", 400

        # get modelID from given id or nickname
        session = database.cursor()
        try:
            queryModelID = int(modelName)
            queryModelNickName = None
        except BaseException:
            queryModelID = -1
            queryModelNickName = modelName

        session.execute(
            "SELECT id, plugin_id, state FROM models WHERE id=%s OR
nickname=%s;",
            (queryModelID,
             queryModelNickName))
        result = session.fetchone()
        session.close()

        if result is None:
            return "No model with given modelID was found.", 404

        modelID, pluginID, modelState = result

        if modelState != STATE_DATA_FEEDING:
            return "Can't train model with state: " +
stateID2State[modelState], 400

        # update model state to training
        session = database.cursor()
        session.execute("UPDATE models SET state=%s WHERE id=%s;",
(STATE_TRAINING, modelID))
        database.commit()

        outputPipe = queue.Queue(8)
```

```
        def onMessage(message):
            outputPipe.put(message)

        def onFinished(isSucceed):
            if isSucceed:
                # update model state to training
                session = database.cursor()
                session.execute("UPDATE models SET state=%s WHERE id=%s;",
                            (STATE_MODEL_USABLE, modelID))
                database.commit()

                outputPipe.put("Done! Model successfully trained and
saved.")
            else:
                # training failed. revert to last step
                session = database.cursor()
                session.execute("UPDATE models SET state=%s WHERE id=%s;",
                            (STATE_DATA_FEEDING, modelID))
                database.commit()

                outputPipe.put("Failed!")

            outputPipe.put(None)  # terminate outputstream to client
            print("Model " + str(modelID) + " is trained...successfully?",
isSucceed)


threading.Thread(target=solutionPlugins[pluginID]["instance"].trainModel
,
                     args=(database, modelID, trainingParameter,
onMessage, onFinished)).start()

        def events():
            message = outputPipe.get()
            while message is not None:
                yield message + "\n"
                message = outputPipe.get()
```

```
        res = Response(events(), mimetype='text/plain')
        res.headers["X-Content-Type-Options"] = "nosniff"
        return res

    @app.route('/predict', methods=['GET', 'POST'])
    def uploadPhotoAndPredict():
        if request.method == 'POST':
            files = request.files.getlist("file[]")
            modelName = request.form["modelID"]

            # get modelID from given id or nickname
            session = database.cursor()
            try:
                queryModelID = int(modelName)
                queryModelNickName = None
            except BaseException:
                queryModelID = -1
                queryModelNickName = modelName

            session.execute(
                "SELECT id, plugin_id, state FROM models WHERE id=%s OR
nickname=%s;",
                (queryModelID,
                 queryModelNickName))
            result = session.fetchone()
            session.close()

            if result is None:
                return "No model with given modelID was found.", 404

            modelID, pluginID, modelState = result

            if modelState != STATE_MODEL_USABLE:
                return "Can't predict with a untrained model.", 400

            # deal with gives files
            print("DEBUG ", len(files), "files uploading.")
```

```
            resources = []
            resourceNames = []

            for file in files:
                if file and utils.isFileAllowed(file.filename,
ALLOWED_EXTENSIONS):
                    mimetype = file.mimetype
                    if len(mimetype) < 3:  # mime type is invalid, guess it
                        mimetype = _getMimeFromExtension(file.filename)

                    if mimetype.startswith("image/"):
                        # load image
                        opencvImage = cv.imdecode(np.fromstring(
                            file.read(), np.uint8), cv.IMREAD_COLOR)

                        if opencvImage is None:
                            print("DEBUG Image", file.filename, "decode
failed.")
                            continue

                        resources.append(opencvImage)
                        resourceNames.append(file.filename)

                        # print("DEBUG Image", image.filename,"upload OK.")
                    else:
                        print(
                            "DEBUG Resource",
                            file.filename,
                            "upload rejected. Not supported currently.")

                else:
                    print("DEBUG Resource", file.filename, "upload
rejected.")

            outputPipe = queue.Queue(1)

            def onFinished(resultList):
                outputPipe.put(resultList)
```

```
threading.Thread(target=solutionPlugins[pluginID]["instance"].predictWit
hData,
                         args=(database, modelID, resourceNames,
resources, onFinished)).start()

            def events():
                predictResult = outputPipe.get()
                yield json.dumps(predictResult)

            return Response(events(), mimetype='application/json')

        return '''
            <!doctype html>
            <title>Upload new File to predit</title>
            <h1>Upload new File</h1>
            <form action="" method=post enctype=multipart/form-data>
                <input type=text name="modelName"/>
                <input type=file name="file[]" multiple="multiple"/>
                <input type=submit value=Upload />
            </form>
            '''

    @app.route('/predict_w_list', methods=['POST'])
    def feedJSONResouceIDListAndPredict():
        modelName = request.args.get('modelID')  # can be number:
modelID, or model nickname

        resourceIDList = None

        if request.is_json:
            resourceIDList = request.get_json()
            if resourceIDList is None or not isinstance(resourceIDList,
list):
                return "Bad resource list format.", 400

        # get modelID from given id or nickname
```

```
        session = database.cursor()
        try:
            queryModelID = int(modelName)
            queryModelNickName = None
        except BaseException:
            queryModelID = -1
            queryModelNickName = modelName

        session.execute(
            "SELECT id, plugin_id, state FROM models WHERE id=%s OR
nickname=%s;",
            (queryModelID,
             queryModelNickName))
        result = session.fetchone()
        session.close()

        if result is None:
            return "No model with given modelID was found.", 404

        modelID, pluginID, modelState = result

        if modelState != STATE_MODEL_USABLE:
            return "Can't predict with a untrained model.", 400

        outputPipe = queue.Queue(1)

        def onFinished(resultList):
            outputPipe.put(resultList)


threading.Thread(target=solutionPlugins[pluginID]["instance"].predictWit
hID,
                         args=(database, modelID, resourceIDList,
onFinished)).start()

        def events():
            predictResult = outputPipe.get()
            yield json.dumps(predictResult)
```

```
            return Response(events(), mimetype='application/json')


##################################
    utils.py
##################################
    def getPluginName(plugin):
        return plugin.getManufacturer() + "_" + plugin.getName()


    def getPluginID(plugin):
        return plugin.getManufacturer() + "_" + plugin.getName() + "_" +
    plugin.getVersion()


    def getPluginDataTableID(pluginID):
        return pluginID.replace(".", "_")


    def isFileAllowed(filename, allowedExtension):
        filename = filename.lower()
        return '.' in filename and filename.rsplit('.', 1)[1] in
    allowedExtension


    def reserveNewID(database, moduleTag):
        session = database.cursor()
        session.execute("UPDATE resource_indexes\
                        SET next_index = next_index + 1\
                    WHERE module_id = '" + moduleTag + "' RETURNING
    next_index;")

        database.commit()

        return session.fetchone()[0] - 1
```

```
################################
    resource_loader_plugins/interface.py
################################
    import abc


    class ResourceLoader(abc.ABC):
        """
        The interface for a resource management plugin.
        """

        # getters of basic metadata
        @staticmethod
        @abc.abstractmethod
        def getManufacturer():
            """
            Get the manufacturer's name.

            Returns:
                str: Manufacturer's name.
            """

            raise NotImplementedError("Not implemented function")

        @staticmethod
        @abc.abstractmethod
        def getAuthor():
            """
            Get the author's name.

            Returns:
                str: Author's name.
            """

            raise NotImplementedError("Not implemented function")

        @staticmethod
        @abc.abstractmethod
```

```
    def getName():
        """
        Get the name of this plugin.
        !!! Should be identical for every plugin under same
manufacturer !!!

        Returns:
            str: Name of this plugin.
        """

        raise NotImplementedError("Not implemented function")

    @staticmethod
    @abc.abstractmethod
    def getVersion():
        """
        Get the version of this plugin.

        Returns:
            str: Version of this plugin.
        """

        raise NotImplementedError("Not implemented function")

    @staticmethod
    @abc.abstractmethod
    def getDescription():
        """
        Get the description of this plugin.
        !!!
            This is a description for helping average users to choose
plugins with.
            Make sure to make the content as easy and clear as possible.
        !!!

        Returns:
            str: Description of this plugin.
        """
```

```
        raise NotImplementedError("Not implemented function")

    @staticmethod
    @abc.abstractmethod
    def getPriceDescription():
        """
        Get the price description of this plugin.
        !!!
            This is a description for helping average users to choose
plugins with.
            Make sure to make the content as easy and clear as possible.
        !!!

        Returns:
            str: Price description of this plugin.
        """
        raise NotImplementedError("Not implemented function")

    # service handlers
    # return opencv2 image
    @abc.abstractmethod
    def getResource(self, database, id):
        """
        Retrieve a resource.

        Args:
            database (obj): The database instance spawned by server.
            id      (int): ID of the resource.

        Returns:
            object: An OpenCV matrix containing the raw resource.
        """

        raise NotImplementedError("Not implemented function")

    # return printable
    @abc.abstractmethod
```

```
    def getResourceExtraInfo(self, database, id):
        """
        Retrieve extra info of a resource.

        Args:
            database (obj): The database instance spawned by server.
            id      (int): ID of the resource.

        Returns:
            object: An OpenCV matrix containing the raw resource, "None"
if failed
        """

        raise NotImplementedError("Not implemented function")

    @abc.abstractmethod
    def putResource(self, database, id, datastream, mime):
        """
        Decide and save a resource.

        Args:
            database (obj): The database instance spawned by server.
            id      (int): Global resource generated by module layer.
            datastream (obj): InputStream of a flask's FileStorage
instance.
            mime      (str): The MIME-type of the resource.

        Returns:
            bool: True if ok, False if failed.
        """

        raise NotImplementedError("Not implemented function")
```

```
########################################
    resource_loader_plugins/local_photo_loader.py
########################################
    from . import interface
    import cv2
    import numpy as np
    import time
    import os

    PHOTO_DIRECTORY = "data/mylocalphotoloader"


    def reflector():
        return LocalPhotoLoader


    class LocalPhotoLoader(interface.ResourceLoader):

        def __init__(self, datatableName, serverAPI):
            self._datatableName = datatableName

            import os
            if not os.path.exists(PHOTO_DIRECTORY):
                os.makedirs(PHOTO_DIRECTORY)

        # getters of basic metadata
        @staticmethod
        def getManufacturer():
            return "edu.hm.hsieh"

        @staticmethod
        def getAuthor():
            return "hsieh"

        @staticmethod
        def getName():
            return "mylocalphotoloader"

        @staticmethod
```

```
    def getVersion():
        return "1.0"

    @staticmethod
    def getDescription():
        return "This is a basic photo loader. This loader compress all
photo recieving with PNG formate and saves and loads photos directly
from local file-system."

    @staticmethod
    def getPriceDescription():
        return "1 database operation per photo read/write. Price of
storage depends on the size of the photo."

    # service handlers
    # return opencv image
    def getResource(self, database, id):
        session = database.cursor()
        session.execute("SELECT remote_id, extra_info FROM " +
                    self._datatableName + " WHERE id = %s", (id,))
        imageInfo = session.fetchone()
        session.close()

        if imageInfo is None:
            print("INFO (", self._datatableName, ") Provided id",
                  id, "is not found in plugin database.")
            return None

        filename, _ = imageInfo
        imagePath = os.path.join("./" + PHOTO_DIRECTORY, filename)

        if not os.path.isfile(imagePath):
            print("INFO (", self._datatableName, ") Photo", filename,
                  "can no longer be found from the filesystem.")
            return None

        frame = cv2.imread(imagePath)
        return frame
```

```
    def getResourceExtraInfo(self, database, id):
        session = database.cursor()
        session.execute("SELECT extra_info FROM " + self._datatableName +
" WHERE id = %s", (id,))
        imageInfo = session.fetchone()
        session.close()

        if not imageInfo:
            return None

        return "No Extra Infomation Available."

    def putResource(self, database, id, photoStream, mime):
        if not mime.startswith("image/"):
            print("DEBUG This resource-loader doesn't support ", mime)
            return False

        filename = LocalPhotoLoader.getName() + "-" +
str(int(time.time_ns())) + ".png"

        savePhotoResult = photoWriter(os.path.join(PHOTO_DIRECTORY,
filename), photoStream)
        if savePhotoResult:
            # update database record
            session = database.cursor()
            session.execute(
                "INSERT INTO " +
                self._datatableName +
                " (id, remote_id) VALUES (" +
                str(id) +
                ", '" +
                filename +
                "');")
            database.commit()

            return True
```

```
            return False

    # return true if ok, false if failed


    def photoWriter(filename, photoStream):
        frame = cv2.imdecode(np.fromstring(photoStream.read(), np.uint8),
    cv2.IMREAD_COLOR)
        return cv2.imwrite(filename, frame)
```

##################################
solution_manager/interface.py
##################################

```
    import abc


    class SolutionManager(abc.ABC):
        """
        The interface for a solution management plugin.
        """

        # getters of basic metadata
        @staticmethod
        @abc.abstractmethod
        def getManufacturer():
            """
            Get the manufacturer's name.

            Returns:
                str: Manufacturer's name.
            """
            raise NotImplementedError("Not implemented function")

        @staticmethod
        @abc.abstractmethod
        def getAuthor():
            """
            Get the author's name.
```

```
        Returns:
            str: Author's name.
        """

        raise NotImplementedError("Not implemented function")

    @staticmethod
    @abc.abstractmethod
    def getName():
        """
        Get the name of this plugin.
        !!! Should be identical for every plugin under same
manufacturer !!!

        Returns:
            str: Name of this plugin.
        """
        raise NotImplementedError("Not implemented function")

    @staticmethod
    @abc.abstractmethod
    def getVersion():
        """
        Get the version of this plugin.

        Returns:
            str: Version of this plugin.
        """
        raise NotImplementedError("Not implemented function")

    @staticmethod
    @abc.abstractmethod
    def getDescription():
        """
        Get the description of this plugin.
        !!!
```

```
            This is a description for helping average users to choose
plugins with.
            Make sure to make the content as easy and clear as possible.
        !!!

        Returns:
            str: Description of this plugin.
        """
        raise NotImplementedError("Not implemented function")

    @staticmethod
    @abc.abstractmethod
    def getPriceDescription():
        """
        Get the price description of this plugin.
        !!!
            This is a description for helping average users to choose
plugins with.
            Make sure to make the content as easy and clear as possible.
        !!!

        Returns:
            str: Price description of this plugin.
        """
        raise NotImplementedError("Not implemented function")

    # service handlers
    @abc.abstractmethod
    def createModel(self, database, id):
        """
        Create a brand-new model.

        Args:
            database (obj): The database instance spawned by server.
            id      (int): Global modelID generated by module layer.

        Returns:
```

```
            bool: True if model created, resources for new model reserved
successfully.
        """
        raise NotImplementedError("Not implemented function")

    @abc.abstractmethod
    def feedTrainData(self, database, id, trainData):
        """
        Feed training data to the model.

        Args:
            database (obj): The database instance spawned by server.
            id      (int): ID of the model to feed data with.
            trainData(dict<int,string>): New training data, dict of
resourceID:class-tag of the resource.

        Returns:
            bool: True if training data fed successfully.
            str: Error message if operation failed.
        """
        raise NotImplementedError("Not implemented function")

    @abc.abstractmethod
    def trainModel(self, database, modelID, parameters, onMessage,
onFinished):
        """
        Start to train the model.

        Args:
            database (obj): The database instance spawned by server.
            id      (int): ID of the model to train.
            parameters(dict<str,obj>): Parameters provided from user to
costomize training.
            onMessage (func(str)): A callback function for exporting
training logging to user.
            onFinished (func(bool)): A callback function called exactly
once whenever training is completed. True: if all ok; False: if failed.
```

```
        Returns:
            No Return.
        """
        raise NotImplementedError("Not implemented function")

    @abc.abstractmethod
    def predictWithID(self, database, modelID, inputDataIDs,
onFinished):
        """
        Make prediction with given list of resourceIDs.

        Args:
            database (obj): The database instance spawned by server.
            modelID  (int): The model's ID to predict with.
            inputDataIDs (List<int>): List of resourceIDs.
            onFinished (func(obj)): A callback function called exactly
once whenever prediction is completed.
                                    Type of prediction result: dict<str,obj>
                                    Format of prediction result:
                                        ISOK (bool): Is prediction performed
successfully.
                                        RESULT (dict<int,str>): The dict of
the prediction result. Form of resourceID:class-tag of the resource.

        Returns:
            No Return.
        """
        raise NotImplementedError("Not implemented function")

    @abc.abstractmethod
    def predictWithData(self, database, modelID, resourceNames,
rawDatas, onFinished):
        """
        Make prediction with given list of rawDatas.

        Args:
            database (obj): The database instance spawned by server.
            modelID  (int): The model's ID to predict with.
```

```
            resourceNames (List<str>): List of filename of uploaded
    resources.
            rawDatas (List<int>): List of uploaded resources.
            onFinished (func(obj)): A callback function called exactly
    once whenever prediction is completed.
                                Type of prediction result: dict<str>obj
                                Format of prediction result:
                                    ISOK (bool): Is prediction performed
    successfully.
                                    RESULT (dict<int>str): The dict of the
    prediction result. Form of resourceID:class-tag of the resource.

        Returns:
            No Return.
        """
        raise NotImplementedError("Not implemented function")


########################################
    solution_manager/azure_image_classifier.py
########################################
    from . import interface

    import cv2
    import numpy as np
    import time
    import os
    import json
    import pickle

    from azure.cognitiveservices.vision.customvision.training import
    CustomVisionTrainingClient
    from azure.cognitiveservices.vision.customvision.training.models import
    ImageFileCreateEntry
    from azure.cognitiveservices.vision.customvision.prediction import
    CustomVisionPredictionClient

    MODEL_ID_PREFIX = "azureimageclassifier-1.0"
```

```
def reflector():
    return AzureCustomVision


class AzureCustomVision(interface.SolutionManager):

    def __init__(self, datatableName, serverAPI):
        self._datatableName = datatableName
        self._serverAPI = serverAPI
        self._endPoint =
"https://westeurope.api.cognitive.microsoft.com/"
        self._trainingKey = "[TRAINING_KEY]"
        self._predictionKey = "[PREDICTION_KET]"
        self._resourceID = "[RESOURCE_ID]"

    # getters of basic metadata
    @staticmethod
    def getManufacturer():
        return "edu.hm.hsieh"

    @staticmethod
    def getAuthor():
        return "hsieh"

    @staticmethod
    def getName():
        return "azureimageclassifier"

    @staticmethod
    def getVersion():
        return "1.0"

    @staticmethod
    def getDescription():
        return "This model uses Microsoft Azure Custom Vision to provide
better image classification experience."
```

```
    @staticmethod
    def getPriceDescription():
        return "Pricing see: https://azure.microsoft.com/en-us/pricing/"

    # service handlers
    def createModel(self, database, id):

        trainer = CustomVisionTrainingClient(self._trainingKey,
endpoint=self._endPoint)
        project = trainer.create_project(MODEL_ID_PREFIX + "-" +
str(time.time_ns()))

        # reserve neccessory resources from the new model and fill the
record into database
        session = database.cursor()
        session.execute("INSERT INTO " + self._datatableName +
                        " (id, remote_id) VALUES (%s, %s)", (id,
project.id))
        database.commit()

        return True

    def feedTrainData(self, database, id, newTrainData):
        # get model record from database

        session = database.cursor()
        session.execute("SELECT remote_id, training_data, extra_info FROM
" +
                        self._datatableName + " WHERE id = %s", (id,))
        result = session.fetchone()
        session.close()

        if result is None:
            print("INFO (", self._datatableName, ") Model", id,
                  "can no longer be found from the plugin database.")
            return False, "model can not be found from plugin database"

        _, trainingData, _ = result
```

```
        if trainingData is None:
            trainingData = {}

        trainingData = {**trainingData, **newTrainData}

        # update record
        session = database.cursor()
        session.execute(
            "UPDATE " +
            self._datatableName +
            " SET training_data=%s WHERE id=%s;",
            (json.dumps(trainingData),
             id))
        database.commit()

        return True, None

    def trainModel(self, database, modelID, parameters, onMessage,
onFinished):
        onMessage("Trainer fetching model settings.")
        session = database.cursor()
        session.execute("SELECT remote_id, training_data, extra_info FROM
" +
                        self._datatableName + " WHERE id = %s", (modelID,))
        result = session.fetchone()
        session.close()

        if result:
            projectID, trainingData, _ = result

            onMessage("Training starting...")

            onMessage("Retrieving model...")
            trainer = CustomVisionTrainingClient(self._trainingKey,
endpoint=self._endPoint)
            project = trainer.get_project(projectID)
```

```
            onMessage("Downloading/Caching and Analyzing training
data...")

            imageList = []
            dataClassList = {}

            try:
                start = time.time()
                # retrieve information of created tags
                createdTags = trainer.get_tags(projectID)
                for tag in createdTags:
                    dataClassList[tag.name] = tag

                imageOK = 0
                imageFailed = 0
                imageTotal = len(trainingData)

                def visualizeImageDownload():
                    return "(" + str(imageOK) + "/" + str(imageFailed) +
"/" + str(imageTotal) + ")"

                for photoID in trainingData:
                    image, _, err = self._serverAPI.getResource(database,
photoID)

                    if err:
                        imageFailed += 1
                        onMessage(
                            "Failed to download image " +
                            str(photoID) +
                            ". Error: " +
                            err +
                            " " +
                            visualizeImageDownload())
                    else:
                        imageOK += 1

                        classOfData = str(trainingData[photoID])
```

```
                    # create tag if not exists
                    if classOfData not in dataClassList:
                        dataClassList[classOfData] =
trainer.create_tag(project.id, classOfData)

                    isOK, encodedImage = cv2.imencode('.png', image)
                    imageList.append(
                        ImageFileCreateEntry(
                            name=str(photoID) + ".png",
                            contents=encodedImage,
                            tag_ids=[
                                dataClassList[classOfData].id]))

                    onMessage(visualizeImageDownload())
            end = time.time()
            onMessage("Image caching done. Used: " + str(end - start))

            start = time.time()
            for i in range(0, len(imageList), 64):
                batch = imageList[i:i + 64]
                upload_result =
trainer.create_images_from_files(project.id, images=batch)

                if not upload_result.is_batch_successful:
                    onMessage("Image batch upload failed.")

                    for image in upload_result.images:
                        onMessage("Image status: ", image.status)

                    onFinished(False)
                    return
            end = time.time()
            onMessage("Image upload done. Used: " + str(end - start))

            onMessage("Training model with " + str(imageOK) + "
photos...")
```

```
            iteration = trainer.train_project(project.id)
            while (iteration.status != "Completed"):
                iteration = trainer.get_iteration(project.id,
iteration.id)
                onMessage("Training status: " + iteration.status)
                time.sleep(3)

            # The iteration is now trained. Publish it to the project
endpoint
            trainer.publish_iteration(project.id, iteration.id,
projectID, self._resourceID)
            onMessage("Training done.")
            onFinished(True)

        except Exception as err:
            onMessage("Failed to train.")
            onMessage("Error Message: " + str(err))
            onFinished(False)
    else:
        onMessage("The trainer can't recognize the given model any
more.")
        onFinished(False)

def predictWithID(self, database, modelID, inputDataIDs,
onFinished):

    # load model record
    session = database.cursor()
    session.execute(
        "SELECT remote_id FROM " +
        self._datatableName +
        " WHERE id = %s",
        (modelID,
         ))
    result = session.fetchone()
    session.close()

    if result:
```

```
            projectID = result[0]

            predictOK = True
            resultMap = {}

            if len(inputDataIDs) > 0:
                try:
                    # Now there is a trained endpoint that can be used to
make a prediction
                    trainer = CustomVisionTrainingClient(self._trainingKey,
endpoint=self._endPoint)
                    predictor = CustomVisionPredictionClient(
                        self._predictionKey, endpoint=self._endPoint)
                    project = trainer.get_project(projectID)

                    # load photos
                    for photoID in inputDataIDs:

                        image, _, err =
self._serverAPI.getResource(database, photoID)
                        if err is None:
                            isOK, encodedImage = cv2.imencode('.png', image)
                            predictResponse = predictor.classify_image(
                                project.id, projectID, encodedImage)
                            predictResult = predictResponse.predictions
                            if len(predictResult) > 0:
                                resultMap[photoID] = {
                                    "CLASS": predictResult[0].tag_name,
                                    "SCORE": predictResult[0].probability}

                except Exception as err:
                    predictOK = False
                    resultMap = {}
                    print("ERROR (", self._datatableName, ") Model",
                          modelID, "failed to predict: " + str(err))

                    #raise err
```

```
            onFinished({"ISOK": predictOK, "RESULT": resultMap})

        else:
            print("ERROR (", self._datatableName, ") Model", modelID,
                  "failed to predict, model record not found by plugin.")
            onFinished({"ISOK": False})

    def predictWithData(self, database, modelID, imageNames,
inputOpenCVImages, onFinished):
        # load model record
        session = database.cursor()
        session.execute(
            "SELECT remote_id FROM " +
            self._datatableName +
            " WHERE id = %s",
            (modelID,
             ))
        result = session.fetchone()
        session.close()

        if result:
            projectID = result[0]

            predictOK = True
            resultMap = {}

            if len(inputOpenCVImages) > 0:
                try:
                    # Now there is a trained endpoint that can be used to
make a prediction
                    trainer = CustomVisionTrainingClient(self._trainingKey,
endpoint=self._endPoint)
                    predictor = CustomVisionPredictionClient(
                        self._predictionKey, endpoint=self._endPoint)
                    project = trainer.get_project(projectID)

                    # load photos
                    for index in range(len(inputOpenCVImages)):
```

```
                        photoID = imageNames[index]
                        image = inputOpenCVImages[index]

                        isOK, encodedImage = cv2.imencode('.png', image)
                        predictResponse = predictor.classify_image(
                            project.id, projectID, encodedImage)
                        predictResult = predictResponse.predictions
                        if len(predictResult) > 0:
                            resultMap[photoID] = {
                                "CLASS": predictResult[0].tag_name,
                                "SCORE": predictResult[0].probability}

                    except Exception as err:
                        predictOK = False
                        resultMap = {}
                        print("ERROR (", self._datatableName, ") Model",
                              modelID, "failed to predict: " + str(err))

                        #raise err

                    onFinished({"ISOK": predictOK, "RESULT": resultMap})
                else:
                    print("ERROR (", self._datatableName, ") Model", modelID,
                          "failed to predict, model record not found by plugin.")
                    onFinished({"ISOK": False})
```

```
##############################################
    solution_manager/local_foam_elimination_detector.py
##############################################
    from sklearn import svm
    from sklearn import preprocessing
    from scipy.stats import entropy
    from . import interface

    import cv2
    import numpy as np
    import time
```

```
import os
import json
import pickle

MODEL_DIRECTORY = "data/localfoameliminationdetector"
MODEL_ID_PREFIX = "foameliminationdetector-1.0"


def reflector():
    return LocalFoamEliminationDetector


class LocalFoamEliminationDetector(interface.SolutionManager):

    def __init__(self, datatableName, serverAPI):
        self._datatableName = datatableName
        self._serverAPI = serverAPI

        import os
        if not os.path.exists(MODEL_DIRECTORY):
            os.makedirs(MODEL_DIRECTORY)

    # getters of basic metadata
    @staticmethod
    def getManufacturer():
        return "edu.hm.hsieh"

    @staticmethod
    def getAuthor():
        return "hsieh"

    @staticmethod
    def getName():
        return "localfoameliminationdetector"

    @staticmethod
    def getVersion():
        return "1.0"
```

```
    @staticmethod
    def getDescription():
        return "This is model extracts different features from input
photos and train/classify using a SVM."

    @staticmethod
    def getPriceDescription():
        return "Resource this model needs is only CPU resouces of the
local computer. However user should also be aware of costs of
downloading/caching input photos."

    # service handlers

    def createModel(self, database, id):
        # reserve neccessory resources from the new model and fill the
record into database

        session = database.cursor()
        session.execute("INSERT INTO " +
                        self._datatableName +
                        " (id, remote_id) VALUES (%s, %s)", (id,
MODEL_ID_PREFIX +
                                                             "_" +

str(int(time.time_ns())) +
                                                             ".model"))
        database.commit()

        return True

    def feedTrainData(self, database, id, newTrainData):
        # get model record from database

        session = database.cursor()
        session.execute("SELECT remote_id, training_data, extra_info FROM
" +
                        self._datatableName + " WHERE id = %s", (id,))
        result = session.fetchone()
```

```
        session.close()

        if result is None:
            print("INFO (", self._datatableName, ") Model", id,
                  "can no longer be found from the plugin database.")
            return False, "model can not be found from plugin database"

        _, trainingData, _ = result
        if trainingData is None:
            trainingData = {}

        trainingData = {**trainingData, **newTrainData}

        # update record
        session = database.cursor()
        session.execute(
            "UPDATE " +
            self._datatableName +
            " SET training_data=%s WHERE id=%s;",
            (json.dumps(trainingData),
             id))
        database.commit()

        return True, None

    def trainModel(self, database, modelID, parameters, onMessage,
onFinished):
        onMessage("Trainer fetching model settings.")
        session = database.cursor()
        session.execute("SELECT remote_id, training_data, extra_info FROM
" +
                        self._datatableName + " WHERE id = %s", (modelID,))
        result = session.fetchone()
        session.close()

        if result:
            onMessage("Training starting...")
            modelSaveName, trainingData, _ = result
```

```
            onMessage("Downloading/Caching photos...")

            start = time.time()

            images = []
            dataClasses = []
            imageOK = 0
            imageFailed = 0
            imageTotal = len(trainingData)

            def visualizeImageDownload():
                return "(" + str(imageOK) + "/" + str(imageFailed) + "/" +
str(imageTotal) + ")"

            for photoID in trainingData:
                image, _, err = self._serverAPI.getResource(database,
photoID)

                if err:
                    imageFailed += 1
                    onMessage(
                        "Failed to download image " +
                        str(photoID) +
                        ". Error: " +
                        err +
                        " " +
                        visualizeImageDownload())
                else:
                    imageOK += 1
                    images.append(image)
                    dataClasses.append(trainingData[photoID])

                    onMessage(visualizeImageDownload())

            end = time.time()
            onMessage("Caching image done: " + str(end - start))
```

```
            if imageOK < 2:
                onMessage(
                    "Model training aborted.  Number of successfully
downloaded training data is not enough.")
                onFinished(False)
                return

            onMessage("Training model with " + str(imageOK) + "
photos...")

            # get photo features
            features, scaler = self._getImageFeatures(images)

            newModel = svm.LinearSVC()

            try:
                train(newModel, features, dataClasses)

            except Exception as err:
                onMessage("Failed to train model, debug message: " +
str(err))
                onFinished(False)
                return

            onMessage("Training done saving model...")
            try:
                with open("./" + MODEL_DIRECTORY + "/" + modelSaveName,
'wb') as modelFile:
                    pickle.dump(newModel, modelFile,
protocol=pickle.HIGHEST_PROTOCOL)
                with open("./" + MODEL_DIRECTORY + "/" + "scaler-" +
modelSaveName, 'wb') as scalerFile:
                    pickle.dump(scaler, scalerFile,
protocol=pickle.HIGHEST_PROTOCOL)

            except Exception as err:
                onMessage("Failed to save the trained model, debug
message: " + str(err))
```

```
                onFinished(False)
                return

            print("Model saved as " + modelSaveName)

            onMessage("Model saved.")
            onFinished(True)
        else:
            onMessage("The trainer can't recognize the given model any
more.")
            onFinished(False)

    def predictWithID(self, database, modelID, inputDataIDs,
onFinished):

        # load model record
        session = database.cursor()
        session.execute(
            "SELECT remote_id FROM " +
            self._datatableName +
            " WHERE id = %s",
            (modelID,
             ))
        result = session.fetchone()
        session.close()

        if result:
            resultMap = {}

            photoIDs = []
            opencvImages = []

            if len(inputDataIDs) > 0:
                modelFileName = result[0]

                # load photos
                for photoID in inputDataIDs:
```

```
                image, _, err = self._serverAPI.getResource(database,
photoID)
                if err is None:
                    photoIDs.append(photoID)
                    opencvImages.append(image)

            predictResult = self._predict(modelFileName, opencvImages)

            if predictResult is None:
                onFinished({"ISOK": False})
                return

            for index in range(len(photoIDs)):
                resultMap[photoIDs[index]] = {"CLASS":
str(predictResult[index]), "SCORE": 1.0}

            onFinished({"ISOK": True, "RESULT": resultMap})
        else:
            print("ERROR (", self._datatableName, ") Model", modelID,
                  "failed to predict, model record not found by plugin.")
            onFinished({"ISOK": False})

    def predictWithData(self, database, modelID, imageNames,
inputOpenCVImages, onFinished):
        # load model record
        session = database.cursor()
        session.execute(
            "SELECT remote_id FROM " +
            self._datatableName +
            " WHERE id = %s",
            (modelID,
             ))
        result = session.fetchone()
        session.close()

        if result:
            modelFileName = result[0]
```

```
            resultMap = {}
            predictResult = []

            if len(inputOpenCVImages) > 0:
                predictResult = self._predict(modelFileName,
inputOpenCVImages)

                if predictResult is None:
                    onFinished({"ISOK": False})
                    return

                for index in range(len(predictResult)):
                    resultMap[imageNames[index]] = {
                        "CLASS": str(predictResult[index]), "SCORE": 1.0}

            onFinished({"ISOK": True, "RESULT": resultMap})
        else:
            print("ERROR (", self._datatableName, ") Model", modelID,
                  "failed to predict, model record not found by plugin.")
            onFinished({"ISOK": False})

    def _getImageFeatures(self, opencvPhotos, scaler=None):
        features = getFeatures(opencvPhotos)
        features, scaler = normalizeDataset(features, scaler)

        return features, scaler

    def _predict(self, modelFileName, opecvPhotos):

        predictResult = []

        assert (len(opecvPhotos) > 0)

        # load model and scalers
        try:
            with open("./" + MODEL_DIRECTORY + "/" + modelFileName, "rb")
as modelStream:
                model = pickle.load(modelStream)
```

```
            with open("./" + MODEL_DIRECTORY + "/" + "scaler-" +
modelFileName, "rb") as scalerStream:
                scaler = pickle.load(scalerStream)
        except Exception as err:
            print("ERROR (", self._datatableName, ") Model",
modelFileName,
                  "failed to be loaded, debug message: " + str(err))
            return None

        # get photo features
        features, _ = self._getImageFeatures(opecvPhotos, scaler)
        predictResult = model.predict(features)

        return predictResult


### The Solution this model using ###


def getOpenCVImageColorMSE(image):
    # calculate color difference factor D:
    # source: 基于SVM的图像分类算法与实现

    averageColor = np.average(image)
    averageFilter = np.full((len(image), len(image[0]), 3),
averageColor)

    return np.sqrt(((image - averageFilter) ** 2).sum(axis=2)).mean()


def getFeatures(images):
    dataset = []

    for image in images:
        features = []

        # calculate feature 1: MSE of colors of a image
        features.append(getOpenCVImageColorMSE(image))
```

```
        # calculate feature 2: entropy from edges of a image

        # image prepocessing
        grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(grayImage, (3, 3), 0)

        # graphic processing
        sobelX = cv2.Sobel(blurred, -1, 1, 0, ksize=5)
        sobelY = cv2.Sobel(blurred, -1, 0, 1, ksize=5)
        sobelXY = cv2.addWeighted(sobelX, 0.5, sobelY, 0.5, 0)
        #sobelYnp.arctan2(np.absolute(sobelX), np.absolute(sobelY))

        # compute entropy
        features.append(entropy(sobelXY.flatten()))

        dataset.append(features)

    return dataset


def normalizeDataset(dataset, scaler=None):

    if scaler is None:
        #scaler = preprocessing.RobustScaler()
        #scaler = preprocessing.StandardScaler()
        scaler = preprocessing.MinMaxScaler()
        scaler.fit(dataset)

    return scaler.transform(dataset), scaler


def train(model, dataset, dataTags):
    model.fit(dataset, dataTags)  # training the svc model
```

Client SDK

Project tree:

Client SDK
├── requirements.txt
├── resource_managing.py
├── solution_managing.py
├── notification.py
└── utils.py

```
################################
    resource_managing.py
################################
    import requests
    import os
    import pandas
    import numbers


    class ResourceList:
        """
        A class maintaining list of resources.
        """

        def __init__(self, resourceIDMapList):
            """
            Constructor.

            Args:
                resourceIDMapList (obj): Can be either List<int>, list of
    resourceIDs. Or Dict<int, str>, Dict of resourceID to its original
    filenames.
            """

            self._resourceIDMapList = resourceIDMapList
            self._IDs = []
```

```
        if len(resourceIDMapList) > 0 and
isinstance(resourceIDMapList[0], numbers.Number):
            self._IDs = resourceIDMapList

        else:
            for entry in resourceIDMapList:
                idEntrys = entry.keys()
                if len(idEntrys) != 1:
                    raise ValueError("Bad server response formate at: " +
str(entry))

                for idEntry in idEntrys:
                    self._IDs.append(idEntry)

    def getIDs(self):
        """
        Get ID of the resources tracking by this instance.

        Returns:
            idList (List<int>): List of the resourceIDs.
        """
        return self._IDs

    def getResourceIDFilenameMap(self):
        """
        Get the map of resourceIDs to their original filename.

        Returns:
            resourceIDMapList (obj): Either List<int> or Dict<int, str>
        """

        return self._resourceIDMapList

    def merge(self, another):
        """
        Merge this instance with another.

        Args:
```

```
            another (ResourceList): another instance.
        """

        listFromAnother = another.getIDs()
        self._resourceIDMapList = {**self._resourceIDMapList,
**self.getResourceIDFilenameMap()}
        self._IDs += listFromAnother

        return self


class ResourceManager:
    """
    A Clever-Lab AI server resource manager.
    """

    def __init__(self, address, port):
        """
        Constructor.

        Args:
            address (str): IP or URL of the server.
            port (str): Port of the server.
        """

        self._address = address
        self._port = port

    def getPluginList(self, table_view=False):
        """
        Get the list of resource management loader_plugins installed  on
the server.

        Args:
            table_view (bool): Whether render response with table-view.

        Returns:
```

```
            response (obj): Json by default; pandas Dataframe by table-
view mdoe. Contains columns: id, name, manufacturer, version,
description and price description.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        res = requests.get("http://" + self._address + ":" +
str(self._port) + "/resource_pluigins")
        if res.status_code == 200:

            if table_view:
                tableHeaders = [
                    "id",
                    "name",
                    "manufacturer",
                    "version",
                    "description",
                    "price description"]
                data = []

                for plugin in res.json():
                    data.append([plugin["id"],
                                 plugin["name"],
                                 plugin["manufacturer"],
                                 plugin["version"],
                                 plugin["description"],
                                 plugin["price_description"]])

                return pandas.DataFrame(data, columns=tableHeaders)

            else:
                return res.json()

        else:
            raise RuntimeError("Unknown error code: " +
str(res.status_code))
```

```
    def uploadResources(self, pluginID, resourceFileList):
        """
        Upload resources with a list of files.

        Args:
            pluginID (str): ID of the plugin handles this uploading.
            resourceFileList (List<str>): List of filenames of uploading
resources.

        Returns:
            response (obj): A _UploadResourceResult object, containing
fields uploaded, rejected, failed.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        files = []

        for resourceFile in resourceFileList:
            file = open(resourceFile, "rb")

            files.append(("file[]", (resourceFile, file)))

        res = requests.post("http://" + self._address + ":" +
str(self._port) +
                            "/upload", files=files, data={"plugin_name":
pluginID})

        if res.status_code == 200:
            content = res.json()
            return _UploadResourceResult(content["OK"], content["FAILED"],
content["NOT-ALLOWED"])

        elif res.status_code == 404:
            raise RuntimeError("Plugin not found: " + res.text)
```

```
        else:
            raise RuntimeError("Unknown error: " + res.text +
                               ", with status " + str(res.status_code) + "")

    def getResourceList(self, table_view=False):
        """
        Get the list of resources created on the server.

        Args:
            table_view (bool): Whether render response with table-view.

        Returns:
            response (obj): Json by default; pandas Dataframe by table-
view mdoe. Contains columns: ID, CREATED-BY, CREATED-AT, PLUGIN-ID,
MIME, EXTRA_INFO.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        res = requests.get("http://" + self._address + ":" +
str(self._port) + "/get_resource_list")
        if res.status_code == 200:

            if table_view:
                tableHeaders = [
                    "id",
                    "created by",
                    "created at",
                    "upload-plugin id",
                    "resource type",
                    "extra-info"]
                data = []

                for resource in res.json():
                    data.append([resource["ID"],
                                 resource["CREATED-BY"],
                                 resource["CREATED-AT"],
```

```
                                resource["PLUGIN-ID"],
                                resource["MIME"],
                                resource["EXTRA-INFO"]])

                return pandas.DataFrame(data, columns=tableHeaders)

            else:
                return res.json()

        else:
            raise RuntimeError("Unknown error code: " +
    str(res.status_code))

    # Privte classes


    class _UploadResourceResult:
        def __init__(self, filesOK, filesFailed, filesNotAllowed):
            self.uploaded = ResourceList(filesOK)
            self.failed = filesFailed
            self.rejected = filesNotAllowed
```

```
##################################
    solution_managing.py
##################################
    import requests
    import os
    import pandas
    from cv2 import cv2 as cv
    import numpy as np
    import time
    import datetime
    import subprocess
    import io
    import notification
    import utils
```

```
NUMBER_DETECTION = 1
SCORE = 2
NUMBER_DETECTION_AND_SCORE = 3
NUMBER_DETECTION_OR_SCORE = 4

PREDICTION_VIEW_TABLE = 1
PREDICTION_VIEW_PHOTO = 2


class SolutionManager:
    """
    Client library for managing solutions and models.
    """

    def __init__(self, address, port):
        """
        Constructor.

        Args:
            address (str): IP or URL of the server.
            port (str): Port of the server.

        Returns:
            No Returns.
        """

        self._address = address
        self._port = port

    def getPluginList(self, table_view=False):
        """
        Get the list of solution managment plugins installed on the
server.

        Args:
            table_view (bool): Whether render response with table-view.

        Returns:
```

```
            response (obj): Json by default; pandas Dataframe by table-
view mode. Contains columns: id, name, manufacturer, version,
description and price description.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        res = requests.get("http://" + self._address + ":" +
str(self._port) + "/solution_plugins")
        if res.status_code == 200:

            if table_view:
                tableHeaders = [
                    "id",
                    "name",
                    "manufacturer",
                    "version",
                    "description",
                    "price description"]
                data = []

                for plugin in res.json():
                    data.append([plugin["id"],
                                 plugin["name"],
                                 plugin["manufacturer"],
                                 plugin["version"],
                                 plugin["description"],
                                 plugin["price_description"]])

                return pandas.DataFrame(data, columns=tableHeaders)
            else:
                return res.json()

        else:
            raise RuntimeError("Unknown error code: " +
str(res.status_code))
```

```
    def createNewModel(self, solutionID, modelNickName,
modelDescription):
        """
        Create a new model with specified solution.

        Args:
            solutionID (str): ID of the solution plugin.
            modelNickName (str): Nickname of the new model. Must be
server-wide unique, otherwise, 400 Bad Request will be produced.
            modelDescirption (str): Random description for the new model.

        Returns:
            newModel (Model): An model object as handle.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        payload = {}
        if modelNickName:
            payload["nickname"] = modelNickName

        if modelDescription:
            payload["description"] = modelDescription

        res = requests.post("http://" + self._address + ":" +
str(self._port) + "/create_model",
                            params={"solutionID": solutionID},
                            json=payload)

        if res.status_code == 200:
            return Model(int(res.text), self._address, self._port)
        elif res.status_code == 410:
            raise RuntimeError("Failed to create a new model: " +
res.text)
        else:
            raise RuntimeError("Unknown server error: " +
str(res.status_code))
```

```
    def getModelList(self, table_view=False):
        """
        Get the list of models created on the server.

        Args:
            table_view (bool): Whether render response with table-view.

        Returns:
            response (obj): Json by default; pandas Dataframe by table-
view mode. Contains colums: ID, CREATED-BY, CREATED-AT, PLUGIN-ID, MIME,
EXTRA_INFO.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        res = requests.get("http://" + self._address + ":" +
str(self._port) + "/models")
        if res.status_code == 200:

            if table_view:
                tableHeaders = [
                    "id",
                    "nickname",
                    "created at",
                    "created by",
                    "plugin ID",
                    "state",
                    "description"]
                data = []

                for plugin in res.json():
                    data.append([plugin["id"],
                                 plugin["nickname"],
                                 plugin["created_at"],
                                 plugin["create_by"],
                                 plugin["plugin_id"],
```

```
                                plugin["state"],
                                plugin["description"]])

                return pandas.DataFrame(data, columns=tableHeaders)
            else:
                return res.json()

        else:
            raise RuntimeError("Unknown error code: " +
str(res.status_code))

    def modelFactory(self, modelID):
        """
        A factory method to create an instance model.

        Args:
            modelID (int): ID of the model.

        Returns:
            response (obj): Json by default; pandas Dataframe by table-
view mode.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        return Model(modelID, self._address, self._port)


class Model:
    """
    Model instance and its APIs.
    """

    def __init__(self, modelID, serverAddress, serverPort):
        """
        Constructor
```

```
        Args:
            modelID (int): ID of the model.
            serverAddress (str): IP or URL of the server.
            serverPort (str): Port of the server.
        """

        self._modelID = modelID
        self._serverAddress = serverAddress
        self._serverPort = serverPort

    def getID(self):
        """
        Get the id of this model.

        Returns:
            modelID (int): ID of this model.
        """

        return self._modelID

    def feed_train_data(self, resourceList, dataClass):
        """
        Feed training data into model.

        Args:
            resourceList (obj): A resource_managing.ResourceList resource
list.
            dataClass (str): Class-tag that resources in the resourceList
will be labeled with.

        Returns:
            result (str): "OK" if ok.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """
        resourceIDs = resourceList.getIDs()
```

```
        # tag data
        dataset = {}
        for resourceID in resourceIDs:
            dataset[resourceID] = dataClass

        res = requests.post("http://" +
                            self._serverAddress +
                            ":" +
                            str(self._serverPort) +
                            "/feed_train_data", params={"modelID":
self._modelID}, json=dataset)

        if res.status_code == 200 and res.text == "OK":
            return "OK"
        elif res.status_code == 400 or res.status_code == 404 or
res.status_code == 410:
            raise RuntimeError(res.text)
        else:
            raise RuntimeError("Unknown server error: " + res.text)

    def train(self, parameter=None):
        """
        Train this model with dataset fed earlier.

        Args:
            parameter (obj): A dict containing parameters for customizing
training.

        Returns:
            training_logs (str): Live logging of the training process.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        session = requests.Session()
```

```
        with session.post("http://" + self._serverAddress + ":" +
str(self._serverPort) + "/train_model", params={"modelID":
self._modelID}, json=parameter, stream=True) as res:
            print("INFO Server answered training model result=" +
str(res.status_code))
            if res.status_code == 200:
                print("INFO Training starting:")

                for line in res.iter_lines():
                    if line:
                        print(line.decode())
            else:
                raise RuntimeError("Server rejected the training request,
err: " + res.text)

    def predictWithResourceList(self, resourceList, view=None):
        """
        Make predictions on the resources listed.

        Args:
            resourceList (obj): A resource_managing.ResourceList resource
list.
            view (int): Type of renderer appling on prediction-responses,
supporting PREDICTION_VIEW_TABLE and REDICTION_VIEW_PHOTO.

        Returns:
            result (obj): Parsed Json object if no view-type is set.
pandas.DataFrame if table-view set. Nothing returned by /local_photo-
view but will displays rendered photos automatically._loader.py

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        res = requests.post("http://" + self._serverAddress + ":" +
str(self._serverPort) +
                            "/predict_w_list", params={"modelID":
self._modelID}, json=resourceList.getIDs())
```

```
        if res.status_code == 200:
            res = res.json()

            if res["ISOK"]:
                if view == PREDICTION_VIEW_TABLE:
                    return _toTableView(res["RESULT"])
                elif view == PREDICTION_VIEW_PHOTO:
                    return _toPhotoView(res["RESULT"], self._serverAddress,
self._serverPort)
                else:
                    return res["RESULT"]
            else:
                raise RuntimeError("Predict failed.")
        else:
            raise RuntimeError("ERROR Unknown server error with code: " +
str(res.status_code))

    def predict(self, resourceFileName, view=None):
        """
        Upload and make prediction on a file.

        Args:
            resourceFileName (str): Filename of the resource to be
predicted.
            view (int): Type of renderer appling on prediction-responses,
supporting PREDICTION_VIEW_TABLE and REDICTION_VIEW_PHOTO.

        Returns:
            result (obj): Parsed Json object if no view-type is set.
pandas.DataFrame if table-view set. Nothing returned by photo-view but
will displays rendered photos automatically.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        return self.predictWithResources([resourceFileName], view)
```

```
    def predictWithResources(self, resourceFilenameList, view=None):
        """
        Upload files and make prediction on them.

        Args:
            resourceFilenameList (List<str>): List of filenames that will
be uploaded.
            view (int): Type of renderer appling on prediction-responses,
supporting PREDICTION_VIEW_TABLE and REDICTION_VIEW_PHOTO.

        Returns:
            result (obj): Parsed Json object if no view-type is set.
pandas.DataFrame if table-view set. Nothing returned by photo-view but
will displays rendered photos automatically.

        Raises:
            RuntimeError: If failed to fetch valid response from server.
        """

        files = []

        modelName = self.getID()

        for resourceFile in resourceFilenameList:
            file = open(resourceFile, "rb")
            files.append(("file[]", (resourceFile, file)))

        res = requests.post("http://" +
                            self._serverAddress +
                            ":" +
                            str(self._serverPort) +
                            "/predict", files=files, data={"modelID":
modelName})

        if res.status_code == 200:
            res = res.json()
```

```
            if res["ISOK"]:
                if view == PREDICTION_VIEW_TABLE:
                    return _toTableView(res["RESULT"])
                elif view == PREDICTION_VIEW_PHOTO:
                    return _toPhotoView(res["RESULT"], self._serverAddress,
self._serverPort)
                else:
                    return res["RESULT"]
            else:
                raise RuntimeError("Predict failed.")
        else:
            raise RuntimeError("ERROR Unknown server error with code: " +
str(res.status_code))

    def streamPredict(
            self,
            ffmpegPath,
            captureInterval,
            experimentLength,
            cameraName,
            targetPredictionClass,
            notificator,
            messageType=notification.MESSAGE_TYPE_TEXT,
            triggerMode=NUMBER_DETECTION,
            classDetectedThreshold=1,
            scoreThreshold=1.0):
        """
        Make autonomous real-time classification with cameras.

        Args:
            ffmpegPath (str): Path of the ffmpeg execuable.
            captureInterval (int): Interval of taking a photo.
            experimentLength (int): Interval of the whole experiment.
            cameraName (int): Name of the camera connecting to.
            targetPredictionClass (str): The target class this script to
be aware of.
            notificator (Notifcation): The notificator to send message
with.
```

```
            messageType (int): Type of the notificator, e.g.
notification.MESSAGE_TYPE_TEXT.
            triggerMode (int): Mode of trigger producing notifications.
            classDetectedThreshold (int): Threshold of trigger on number
of detections against target-class.
            classDetectedThreshold (float): Threshold of trigger on
confidience of target-class.
        """

        cameraName = cameraName.strip()

        assert len(cameraName) > 0
        triggerCounter = 0

        for _ in range(experimentLength // captureInterval):
            tempFileName = "temp-" + str(time.time_ns()) + ".png"

            result, err = _ffmpegCaptureAFrame(ffmpegPath, cameraName,
tempFileName)

            if result:
                print("DEBUG A Photo was succesfully taken at",
datetime.datetime.now())
                predictResult = self.predictWithResources([tempFileName])
                if predictResult:
                    if len(predictResult.keys()) != 1:
                        print("ERROR Invalid predicted entry numbers:",
len(predictResult.keys()))
                    else:
                        sortedToClass =
predictResult[list(predictResult.keys())[0]]["CLASS"]
                        scoreRaw =
predictResult[list(predictResult.keys())[0]]["SCORE"]
                        tag, score = _toImageTag(sortedToClass, scoreRaw)

                        trigger = False

                        if triggerMode == SCORE:
```

```
                        if scoreRaw >= scoreThreshold:
                            trigger = True

                    elif triggerMode == NUMBER_DETECTION_AND_SCORE:
                        if scoreRaw >= scoreThreshold and sortedToClass
== str(
                                targetPredictionClass):
                            triggerCounter += 1
                        else:
                            triggerCounter = 0

                        if triggerCounter >= classDetectedThreshold:
                            trigger = True

                    elif triggerMode == NUMBER_DETECTION_OR_SCORE:

                        if sortedToClass == str(targetPredictionClass):
                            triggerCounter += 1
                        else:
                            triggerCounter == 0

                        if triggerCounter >= classDetectedThreshold or
scoreRaw >= scoreThreshold:
                            trigger = True

                    else:
                        # default: Number of detection mode
                        if sortedToClass == str(targetPredictionClass):
                            triggerCounter += 1
                        else:
                            triggerCounter == 0

                        if triggerCounter >= classDetectedThreshold:
                            trigger = True

                    if trigger:
                        if messageType ==
notification.MESSAGE_TYPE_IMAGE:
```

```
                            try:
                                with open(tempFileName, "rb") as
imageFile:
                                    # load image
                                    opencvImage =
cv.imdecode(np.fromstring(
                                        imageFile.read(), np.uint8),
cv.IMREAD_COLOR)
                                    _tagPhoto(opencvImage, "Class: " + tag
+ "(" + score + ")")

                                    _, buffer = cv.imencode(".png",
opencvImage)
                                    notificator.sendMessage(
                                        notification.MESSAGE_TYPE_IMAGE,
io.BytesIO(buffer))
                            except Exception as err:
                                raise RuntimeError(
                                    "Failed to read saved temp image file:
" + tempFileName + str(err))
                        else:
                            notificator.sendMessage(
                                notification.MESSAGE_TYPE_TEXT,
                                "Target class '" + tag + "'(" + score + ")
detected!")

                else:
                    print("ERROR Stream predict failed.")

                # all done, remove the temp image file
                os.remove(tempFileName)

            else:
                print("DEBUG Photo taken was failed. Debug messages:")

                output = err.readline()

                while len(output) > 0:
```

```
                trimedOutput = output.decode().strip()
                print(trimedOutput)

                output = err.readline()

            time.sleep(captureInterval)


# Private static functions

def _toImageTag(tag, score):
    return str(tag), str(int(score * 10000) / 100) + "%"


def _toTableView(predictionResult):
    tableHeaders = ["id", "class", "score"]
    data = []

    for imageID in predictionResult.keys():
        tag, score = _toImageTag(predictionResult[imageID]
                                 ["CLASS"],
predictionResult[imageID]["SCORE"])
        data.append([imageID, tag, score])

    return pandas.DataFrame(data, columns=tableHeaders)


def _tagPhoto(opencvImage, tag):
    font = cv.FONT_HERSHEY_DUPLEX

    textsize = cv.getTextSize(tag, font, 1, 2)[0]

    cv.rectangle(opencvImage, (10 - 10, 40 + 10),
                 (10 + textsize[0] + 10, 40 + 10 - textsize[1] - 20), (0,
0, 0), -1)
    cv.putText(opencvImage, tag, (10, 40), font, 1, (0, 255, 255), 2,
cv.LINE_AA)
```

```
def _toPhotoView(predictionResult, severAddress, serverPort):

    for photoID in predictionResult:
        isID = True

        try:
            int(photoID)
        except BaseException:
            isID = False

        image = None

        if isID:
            # download from server
            res = requests.get(
                "http://" +
                severAddress +
                ":" +
                str(serverPort) +
                "/get_resource/" +
                photoID,
                stream=True)
            if res.status_code == 200:
                image = np.asarray(bytearray(res.raw.read()),
dtype="uint8")
                image = cv.imdecode(image, cv.IMREAD_COLOR)

            else:
                raise RuntimeError("Cache image " + str(photoID) +
                                   " failed with code: " +
str(res.status_code))

        else:
            image = cv.imread(photoID)

        if image is not None:
            tag, score = _toImageTag(
```

```
                predictionResult[photoID]["CLASS"],
    predictionResult[photoID]["SCORE"])
                label = "Class: " + tag + "(" + score + ")"
                _tagPhoto(image, label)

                utils.showOpenCVImage(image, photoID)


    def _ffmpegCaptureAFrame(ffmpegPath, cameraName, outputFilename):
        captureFrame = subprocess.run(
            [ffmpegPath, "-n", "-hide_banner", "-f", "dshow", "-i",
             "video=" + cameraName, "-vframes", "1", outputFilename],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE)

        if captureFrame.returncode == 0:
            return True, None
        else:
            return False, io.BytesIO(captureFrame.stderr)
```

```
##################################
    notification.py
##################################
import paho.mqtt.client as mqttClient
import abc
import time
import datetime

MESSAGE_TYPE_TEXT = 0  # Enum. Set Message payload type to text-messages.
MESSAGE_TYPE_IMAGE = 1  # Enum. Set Message payload type to image-messages.


class Notification(abc.ABC):
    """
    Interface of a notification class.
    """

    @abc.abstractmethod
```

```
    def __enter__(self):
        pass

    @abc.abstractmethod
    def __exit__(self, exeType, exeValue, exeTraceback):
        pass

    @abc.abstractmethod
    def sendMessage(self, messageType, payload):
        """
        Send a message.

        Args:
            messageType (int): Type of the message in the payload.
            payload (obj): payload.
        """
        pass


class ConsoleNotification(Notification):
    """
    A notificator print notifications directly on a console.
    """

    def __enter__(self):
        pass

    def __exit__(self, exeType, exeValue, exeTraceback):
        pass

    def sendMessage(self, messageType, payload):
        """
        Print notification directly on the console if payload is a text
message, otherwise, save incoming message under CWD and print a text message
on the console.

        Args:
```

```
            messageType (int): type of the payload. Either MESSAGE_TYPE_TEXT
or MESSAGE_TYPE_IMAGE currently
        """

        if MESSAGE_TYPE_TEXT == messageType:
            print(">>>>>>>>>>>>>>>>>>> Text Message at",
datetime.datetime.now(), ":", payload)

        elif MESSAGE_TYPE_IMAGE == messageType:
            filename = "consolenotification-" + str(time.time_ns()) + ".png"
            print(
                ">>>>>>>>>>>>>>>>>>> Image Message at",
                datetime.datetime.now(),
                "Saved as",
                filename)

            with open(filename, "wb") as file:
                file.write(payload.read())


class MQTTNotification(Notification):
    """
    A notificator sending notifications via MQTT.
    """

    def __init__(self, brokerAddress, brokerPort, username, password,
clientID, topicID):
        self._brokerAddress = brokerAddress
        self._brokerPort = brokerPort
        self._username = username
        self._password = password
        self._clientID = clientID
        self._topicID = topicID
        self._connected = False

    def __enter__(self):
        self._client = mqttClient.Client(self._clientID)  # create new
instance
```

```
        self._client.username_pw_set(self._username,
                                     password=self._password)  # set username and
password

        def on_connect(client, userdata, flags, rc):
            if rc == 0:
                print("DEBUG Connected to MQTT-Broker")
                self._connected = True  # Signal connection
            else:
                print("Connection failed")
        self._client.on_connect = on_connect  # attach function to callback

        #self._client.on_message= on_message

        def on_log(client, userdata, level, buf):
            print("DEBUG MQTT LOG: ", buf)
        # self._client.on_log=on_log

        self._client.connect(self._brokerAddress, port=self._brokerPort)  #
connect to broker
        self._client.loop_start()  # start the loop

        while self._connected != True:  # Wait for connection
            time.sleep(0.1)

    def __exit__(self, exeType, exeValue, exeTraceback):
        self._client.disconnect()
        self._client.loop_stop()

    def sendMessage(self, messageType, payload):
        """
        Send notification directly via MQTT protocol if payload is a text
message, otherwise save incoming message under CWD and send a text message
via MQTT.

        Args:
            messageType (int): type of the payload. Supports MESSAGE_TYPE_TEXT
and MESSAGE_TYPE_IMAGE currently.
```

```
        """

        if MESSAGE_TYPE_TEXT == messageType:
            print(">>>>>>>>>>>>>>>>>>> Text Message at",
datetime.datetime.now(), ":", payload)

            self._client.publish(self._topicID, payload, qos=2)

        elif MESSAGE_TYPE_IMAGE == messageType:
            filename = "mqttnotification-" + str(time.time_ns()) + ".png"
            print(
                ">>>>>>>>>>>>>>>>>>> Image Message at",
                datetime.datetime.now(),
                "Saved as",
                filename)

            with open(filename, "wb") as file:
                file.write(payload.read())
            self._client.publish(self._topicID, "Image Message Saved as " +
filename, qos=2)


##################################
    utils.py
##################################
    import os
    import subprocess
    import io
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt


    def getFileList(directory, extension="bmp"):
        names = []

        for filename in os.listdir(directory):
            processedFilename = os.path.join(directory, filename.lower())
```

```
        if processedFilename.endswith(extension):
            names.append(processedFilename)

    return names


def getDirectShowDevices(ffmpegPath):
    cameraInfo = subprocess.run(
        [ffmpegPath, "-hide_banner", "-list_devices", "true", "-f",
"dshow", "-i", "dummyffmpeg"],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)

    # print(cameraInfo.returncode)

    outputStream = io.BytesIO(cameraInfo.stderr)
    output = outputStream.readline()

    while len(output) > 0:
        trimedOutput = output.decode().strip()
        if trimedOutput.startswith("[dshow @"):
            print(trimedOutput)

        output = outputStream.readline()


def showOpenCVImage(openCVImage, title=None, cols=1):

    fig = plt.figure()
    a = fig.add_subplot(cols, 1.0, 1)
    if openCVImage.ndim == 2:
        plt.gray()
    plt.imshow(cv2.cvtColor(openCVImage, cv2.COLOR_BGR2RGB))

    if title is not None:
        a.set_title(title)
    fig.set_size_inches(np.array(fig.get_size_inches()))
    plt.show()
```