Automate Machine Learning with ProActive using Open Source technologies

Summary

Video demonstration is available on this link

ProActive
Machine Learning Studio

[— —Overview— —]

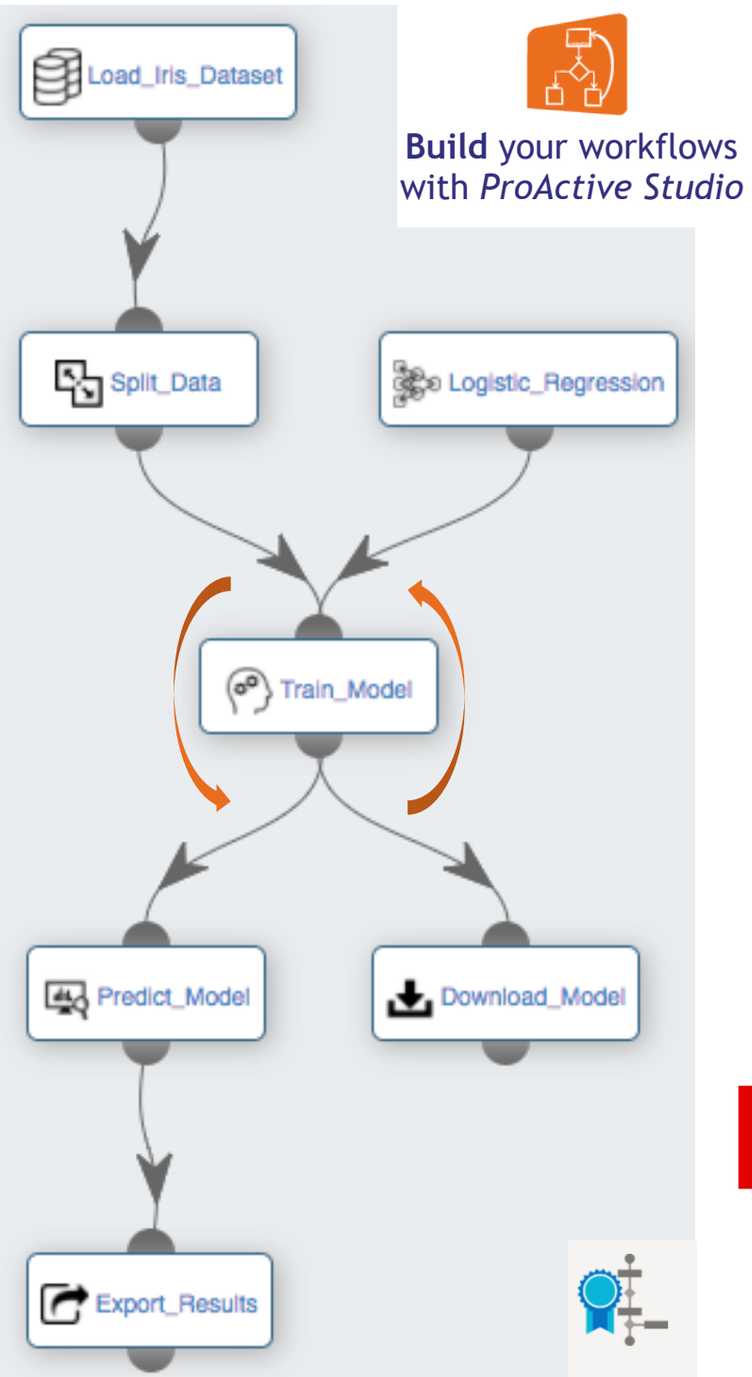
ActiveEon
SCALE BEYOND LIMITS

Introduction

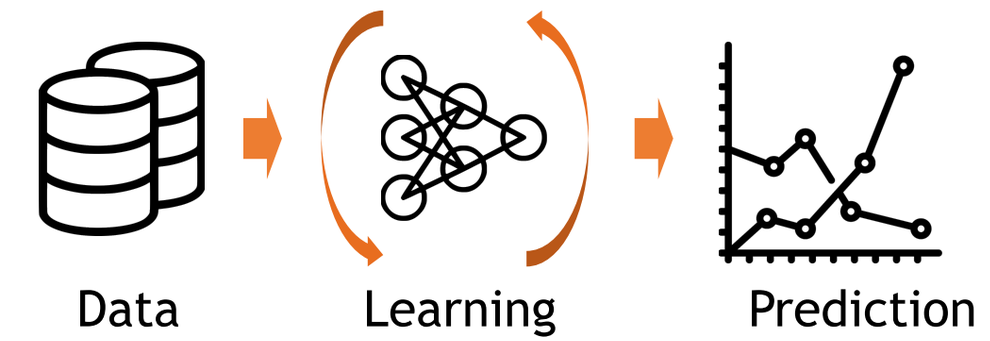
ProActive Machine Learning Open Studio is an interactive visual interface that allows to easily build, deploy, and test predictive on your data. It provides different tasks, for instance, import data, export data, save model, load a trained model, train prediction, and evaluate a model, etc. These tasks enable the user to develop powerful machine learning solutions, such as: fraud detection, text analysis, online offer recommendations, prediction of equipment failures, facial expression analysis, etc. Proactive Machine Learning Studio is open source and allows easy task parallelization, running them on resources matching constraints (Multi-CPU, GPU, data locality, library).

ProActive Machine Learning Studio

ACTIVEEON
SCALE BEYOND LIMITS



Build your workflows with *ProActive Studio*



python™

K + TensorFlow

scikit learn

Visdom

OpenCV

PYTORCH

docker

- Automate with **intelligence**.
- Compose **smart** workflows by visual drag-and-drop.
- Enable data scientists to graphically **build** and **deploy** large-scale experiments.
- Enabled real-time **data visualisation**.
- Use your favorite **machine learning** and **deep learning libraries**.
- Scaled-up under **local network resources** as well in the **cloud**.

ProActive
Machine Learning Studio

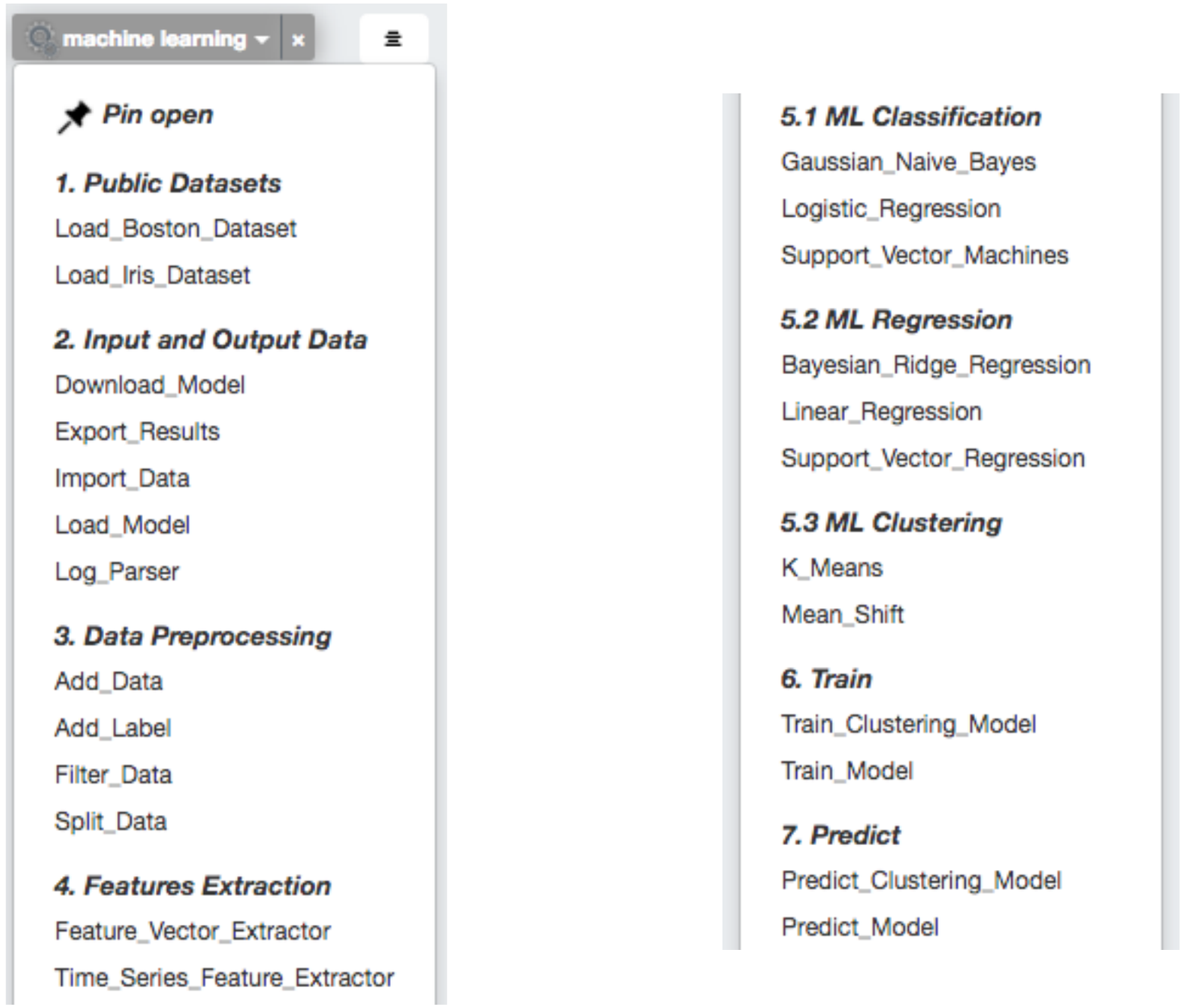
[— —Technical Report— —]

Summary

Summary

ProActive ML Open Studio Interface

ACTIVEEON
SCALE BEYOND LIMITS



Public Datasets

1.Task Name: Load_Boston_Dataset

2. Task Overview:
Load and return the Boston House-Prices dataset.

Features	Targets	Dimensionality	Samples total
Real, positive	Real 5. -50	13	506

3. How to Use Load_Boston_Dataset

- The Boston House-Prices is a dataset regression, you can only use it with a regression algorithm, such as Linear Regression and Support Vector Regression.

- After this task, you can use the **Split_Data** task to divide the dataset into training and testing sets.

More information about this dataset can be found **here**.

Public Datasets

ACTIVEEON
SCALE BEYOND LIMITS

1.Task Name: Load_Iris_Dataset

2. Task Overview:
Load and return the iris dataset.

Features	Classes	Dimensionality	Samples per Class	Samples total
Real, positive	3	4	50	150

3. How to Use Load_Iris_Dataset

- The Iris is a dataset for classification, you can only use it with a classification algorithm, such as Support Vector Machines and Logistic Regression.
- After this task, you can use the **Split_Data** task to divide the dataset into training and testing sets.

More information about this dataset can be found **here**.

Input and Output Data

1.Task Name: Dowload_Model

2. Task Overview:
Download a trained model on your computer device.

3. How to Use Dowload_Model

- The **Download_Model** task should be used after the **Train_Model** or **Train_Clustering_Model** tasks.

Input and Output Data

1.Task Name: Export_Results

2. Task Overview:

Export the results of the predictions generated by a classification, clustering or regression algorithm.

3. Task Variables

OUTPUT_FILE: string HTML or CSV

Converts the prediction results to HTML or CSV file.

4. How to Use Export_Results

- The **Export_Results** task should be used after **Predict_Model** or **Predict_Clustering_Model** tasks.

Input and Output Data

1.Task Name: Import_Data

2. Task Overview:
Load data from external sources.

3. Task Variables

FILE_URL: string
Enter you URL of the CSV file.
FILE_DELIMITER: str
Delimiter to use.
IS_LABELED_DATA: boolean
If your data have label data.

Input and Output Data

Remark: Your CSV file should be in a table format. See the example below.

	preg	plas	pres	skin	test	mass	pedi	age	class
0	1	85	66	29	0	26.6	351	31	0
1	8	183	64	0	0	23.3	672	32	1
2	1	89	66	23	94	28.1	167	21	0
3	0	137	40	35	168	43.1	2288	33	1
4	5	116	74	0	0	25.6	201	30	0
5	3	78	50	32	88	31.0	248	26	1
6	10	115	0	0	0	35.3	134	29	0
7	2	197	70	45	543	30.5	158	53	1
8	8	125	96	0	0	0.0	232	54	1
9	4	110	92	0	0	37.6	191	30	0
10	10	168	74	0	0	38.0	537	34	1
11	10	139	80	0	0	27.1	1441	57	0
12	1	189	60	23	846	30.1	398	59	1
13	5	166	72	19	175	25.8	587	51	1
14	7	100	0	0	0	30.0	484	32	1
15	0	118	84	47	230	45.8	551	31	1
16	7	107	74	0	0	29.6	254	31	1

Rows should be the samples

Columns should be the features

Input and Output Data

1.Task Name: Load_Model

2. Task Overview:
Load a trained model, and use it to make predictions for new coming data.

3. Task Variables

MODEL_URL: string, default: https://s3.eu-west-2.amazonaws.com/activeeon-public/models/pima-indians-diabetes.model
Type the URL to load your trained model.

4. How to Use Load_Model

- The **Load_Model** task should be used before **Predict_Model** or **Predict_Clustering_Model** to make predictions.

Input and Output Data

1.Task Name: Log_Parser

2. Task Overview:
Convert an **unstructured** raw log file into a structured one by matching a group of event patterns.

3. Task Variables:

LOG_FILE: string
refers to the URL of the raw log file that you need to parse.
PATTERNS_FILE: string
refers to the URL of the CSV file that contains the different REGEX expressions of each possible pattern and their corresponding variables.
The csv file must contain three columns:

A. **id_pattern**: integer
   specifies the identifier of each pattern
B. **Pattern**: Regex expression
   Defines the regex expression of a specific pattern
C. **Variables**: string
   refers to the name of each variable included in the pattern.
   N.B: Use the symbol '*' for variables that you need to neglect. (e.g. in the example below the 5th variable is neglected)
   N.B: All variables specified in each Regex expressions have to be mentioned in the column « Variables » in the right order.

Input and Output Data

STRUCTURED_LOG: string
indicate the extension of the file where you will save the resulted structured logs.

Example of a PATTERN_File

id_pattern	Pattern	Variables
1	([0-9]+) ([0-9]+) ([0-9]+) ([A-Z]+)(.*) Adding an already existing block (.*)	date, time, pid, status, *, id_block
2	([0-9]+) ([0-9]+) ([0-9]+) ([A-Z]+)(.*) Verification succeeded for (.*)	date, time, pid, status, *, id_block

Regex Expressions to match

Variables that need to be extracted

N.B: The order of variables need to be respected :
([0-9]+) refers to date
([0-9]+) refers to time
([0-9]+) refers to pid
([A-Z]+) refers to status
(.*) refers to * (i.e. * means ignore this variable)
(.*) refers to id_block

Data Preprocessing

1.Task Name: Add_Data

2. Task Overview:
Concatenate the new added data to the original input data.

3. Task Variables:

FILE_URL: string
refers to the URL of the data that you need to add.
FILE_DELIMITER: string
Delimiter to use.
IS_LABELED_DATA: boolean
TRUE if the data is labeled.
IGNORE_INDEX: boolean
False if you need to ignore the index of the input data.

More details about this task can be found **here**.

Data Preprocessing

1.Task Name: Add_Label

2. Task Overview:
Add a new label column to the original input data.

3. Task Variables

FILE_URL: string
refers to the URL of the file containing the labels that you need to concatenate.
FILE_DELIMITER: string
Delimiters to use.

More details about this task can be found **here**.

Data Preprocessing

1.Task Name: Filter_Data

2. Task Overview:
Query the columns of your data with a boolean expression.

3. Task Variables:

Query: string
The query string to evaluate.
FILTERED_FILE_OUPUT: string
Refers to the extension of the file where the resulted filtered data will be saved.
Options [csv,html]

More details about this task can be found **here**.

Data Preprocessing

1.Task Name: Split_Data

2. Task Overview:
Separate data into train and test subsets.

3. Task Variables:

TRAIN_SIZE: float, default: 0.7
This parameter must be float within the range (0, 100), not including the values 0 and 100.

4. How to Use Split_Data

- The input data file must contains at least two rows.
- You should use the **Split_Data** task after the **Import_Data**, **Load_Boston_Dataset** and **Load_Iris_Dataset** tasks.

More information about this task can be found here.

Feature Extraction

1.Task Name: Feature_Vector_Extractor

2. Task Overview:
Encode **structured** data into numerical feature vectors whereby machine learning models can be applied.

3. Task Variables

SESSION_COLUMN: string
Refers to the ID of the entity that you need to represent.
FILE_OUT_FEATURES: string
Refers to the file name where the resulted features will be saved.
PATTERN_COLUMN: string
Refers to the index of column containing the log patterns.[specific to features extraction from logs]
PATTERNS_COUNT_FEATURES: boolean
Indicates if count features related to log patterns need to be extracted or not.
STATE_VARIABLES: string
Indicates the different variables that need to be considered to extract features according to their content.
COUNT_VARIABLES: string
Refers to the different variables that need to be considered to count their distinct content.
STATE_COUNT_FEATURES_VARIABLES: boolean
Indicates if state and count features need to be extracted or not.

Feature Extraction

1.Task Name: Time_Series_Feature_Extractor

2. Task Overview:
Extracts features by considering the **temporal** distribution of data.

3. Task Variables:

SESSION_COLUMN: string
Refers to the column name of the entity that you need to represent.
SLIDING_STEP: integer(in Minutes)
Specifies the number of minutes to jump while navigating from a window to another one.
WINDOW_SIZE: integer (in Minutes)
Indicates the window size in Minutes.
START_DATE_TIME: date
Indicates the date starting from it the time series vectors representing each entity [Session] will be extracted.
PATTERN_COLUMN: string
Refers to the column name containing the log patterns.[specific to features extraction from logs]
PATTERNS_COUNT_FEATURES: boolean
True if you want to count the number of detected patterns per entity [Session].
STATE_VARIABLES: string
Indicates the name of columns to use as features by counting the number of occurrence of their distinct content.

Feature Extraction

COUNT_VARIABLES: string
Specifies the name of the columns that you need to add as features by counting their distincts content per entity [Session].
STATE_COUNT_FEATURES_VARIABLES: boolean
True if you need to extract state and count features that you already specified in **COUNT_VARIABLES** and **STATE_VARIABLES**.

ML Classification

1.Task Name: Gaussian_Naive_Bayes

2. Task Overview:
Naive Bayes classifier is a family of simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

3. Task Variables:

PRIORS: array-like, shape (n_classes)
Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

4. How to Use Gaussian_Naive_Bayes

- The **Gaussian_Naive_Bayes** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found **here**.

ML Classification

1.Task Name: Logistic_Regression

2. Task Overview:
Logistic Regression is a regression model where the Dependent Variable (DV) is categorical.

3. Task Variables:

PENALTY: string, default: ‘l2’
Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only l2 penalties.
SOLVER: ‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’, default: ‘liblinear’
Algorithm to use in the optimization problem.
MAX_ITERATIONS: int, default: 100
Useful only for the newton-cg, sag and lbfgs solvers. Maximum number of iterations taken for the solvers to converge.
N_JOBS: int, default: 1
Number of CPU cores used. If given a value of -1, all cores are used.

3. How to Use Logistic_Regression

- The **Logistic_Regression** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found **here**.

ML Classification

1.Task Name: Support_Vector_Machines

2. Task Overview:
Support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification.

3. Task Variables:

C: float, optional (default=1.0)
Penalty parameter C of the error term.
KERNEL: string, optional (default='rbf')
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.

3. How to Use Support_Vector_Machines:

- The **Support_Vector_Machines** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found **here**.

ML Regression

1.Task Name: Bayesian_Ridge_Regression

2. Task Overview:
Bayesian linear regression is an approach to linear regression in which the statistical analysis is undertaken within the context of Bayesian inference.

3. Task Variables:

N_ITERATIONS: int, optional (default=300)
Penalty parameter C of the error term.
ALPHA_1: float, default 1.e-6
Hyper-parameter : shape parameter for the Gamma distribution prior over the alpha parameter.
ALPHA_2: float, default 1.e-6
Hyper-parameter : inverse scale parameter (rate parameter) for the Gamma distribution prior over the alpha parameter.
LAMBDA_1: float, default 1.e-6
Hyper-parameter : shape parameter for the Gamma distribution prior over the lambda parameter.
LAMBDA_2: float, default 1.e-6
Hyper-parameter : inverse scale parameter (rate parameter) for the Gamma distribution prior over the lambda parameter.

4. How to Use Bayesian_Ridge_Regression

- The **Bayesian_Ridge_Regression** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found **here**.

ML Regression

1.Task Name: Linear_Regression

2. Task Overview:
Linear regression is a linear approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X.

3. Task Variables:

N_JOBS: int, default:1
Number of jobs to run in parallel.

4. How to Use Linear_Regression

- The **Linear_Regression** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found **here**.

ML Regression

1.Task Name: Support_Vector_Regression

2. Task Overview:
Support vector regression are supervised learning models with associated learning algorithms that analyze data used for regression.

3. Task Variables:

C: float, optional (default=1.0)
Penalty parameter C of the error term.
KERNEL: string, optional (default='rbf')
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.
EPSILON: float, optional (default=0.1)
It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value.

4. How to Use Support_Vector_Regression

- The **Support_Vector_Regression** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found **here**.

ML Clustering

1.Task Name: K_Means

2. Task Overview:

K-means clustering aims to partition n observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

3. Task Variables:

N_CLUSTERS: int, optional, default: 8
The number of clusters to form as well as the number of centroids to generate.
MAX_ITERATIONS: int, default: 300
Maximum number of iterations of the k-means algorithm for a single run.
N_JOBS: int, default:1
The number of jobs to use for the computation. This works by computing each of the n_init runs in parallel. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all

4. How to Use K_Means

- The **K_Means** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found here.

ML Clustering

1.Task Name: Mean_Shift

2. Task Overview:
Mean shift is a non-parametric feature-space analysis technique for locating the maxima of a density function.

3. Task Variables:

CLUSTER_ALL: boolean, default True
The number of clusters to form as well as the number of centroids to generate.
N_JOBS: int, default:1
If true, then all points are clustered, even those orphans that are not within any kernel. Orphans are assigned to the nearest kernel. If false, then orphans are given cluster label -1.

4. How to Use Mean_Shift

- The **Mean_Shift** task should be connected with **Train_Model** / **Train_Clustering_Model** or **Predict_Model** / **Predict_Clustering_Model**.

More information about this task can be found **here**.

Train

1.Task Name: Train_Clustering_Model

2. Task Overview:
Train a clustering model.

3. How to Use Train_Clustering_Model

- The **Train_Clustering_Model** task should be used after a clustering algorithm such as **Bayesian_Ridge_Regression**, **Linear_Regression** or **Support_Vector_Regression**.

More information about this task can be found **here**.

Train

1.Task Name: Train_Model

2. Task Overview:
Train a model using a classification or regression algorithm..

3. How to Use Train_Clustering_Model

- The **Train_Model** task should be used after a classification or regression algorithm, such as **Support_Vector_Machines**, **Gaussian_Naive_Bayes** and **Linear_Regression**.

More information about this task can be found **here**.

Predict

1.Task Name: Predict_Clustering_Model

2. Task Overview:
Generate predictions using a trained model.

3. How to Use Train_Clustering_Model

- The **Predict_Clustering_Model** should be used after the **Train_Clustering_Model** task.

More information about this task can be found **here**.

Predict

1.Task Name: Predict_Model

2. Task Overview:
Generate predictions using a trained model.

3. How to Use Train_Clustering_Model

- The **Predict Model** should be used after the **Train Model** task.

More information about this task can be found **here**.

How to build a machine learning workflow using MLOS tasks

Log Analysis Use Case

Build your own Log Analysis Workflow

ACTIVEEON
SCALE BEYOND LIMITS

1. Create a new Workflow

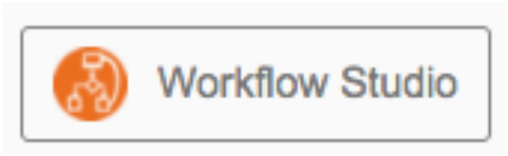
2. Add Machine Learning Bucket as Extra Catalog Menu

3. Compose your own Log Analysis Workflow using ML bucket

4. Execute your Log Analysis Workflow

5. Preview and Download the Output Results

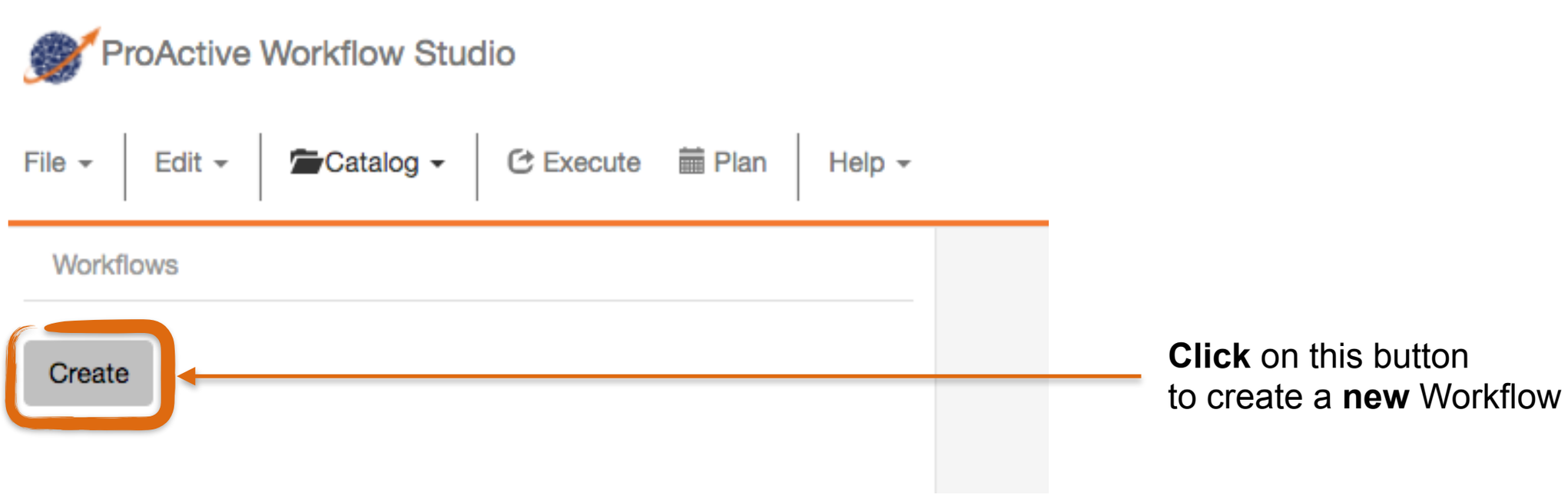
Build your own Log Analysis Workflow



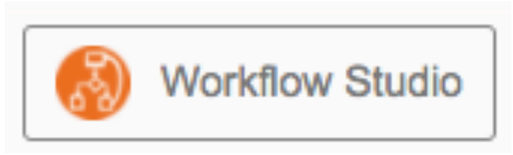
1. Create a new Workflow

Follow these steps to create the workflow:

- Open the ProActive Studio and Login using the user and password you received by e-mail when you first signed up.
- Create a new Workflow



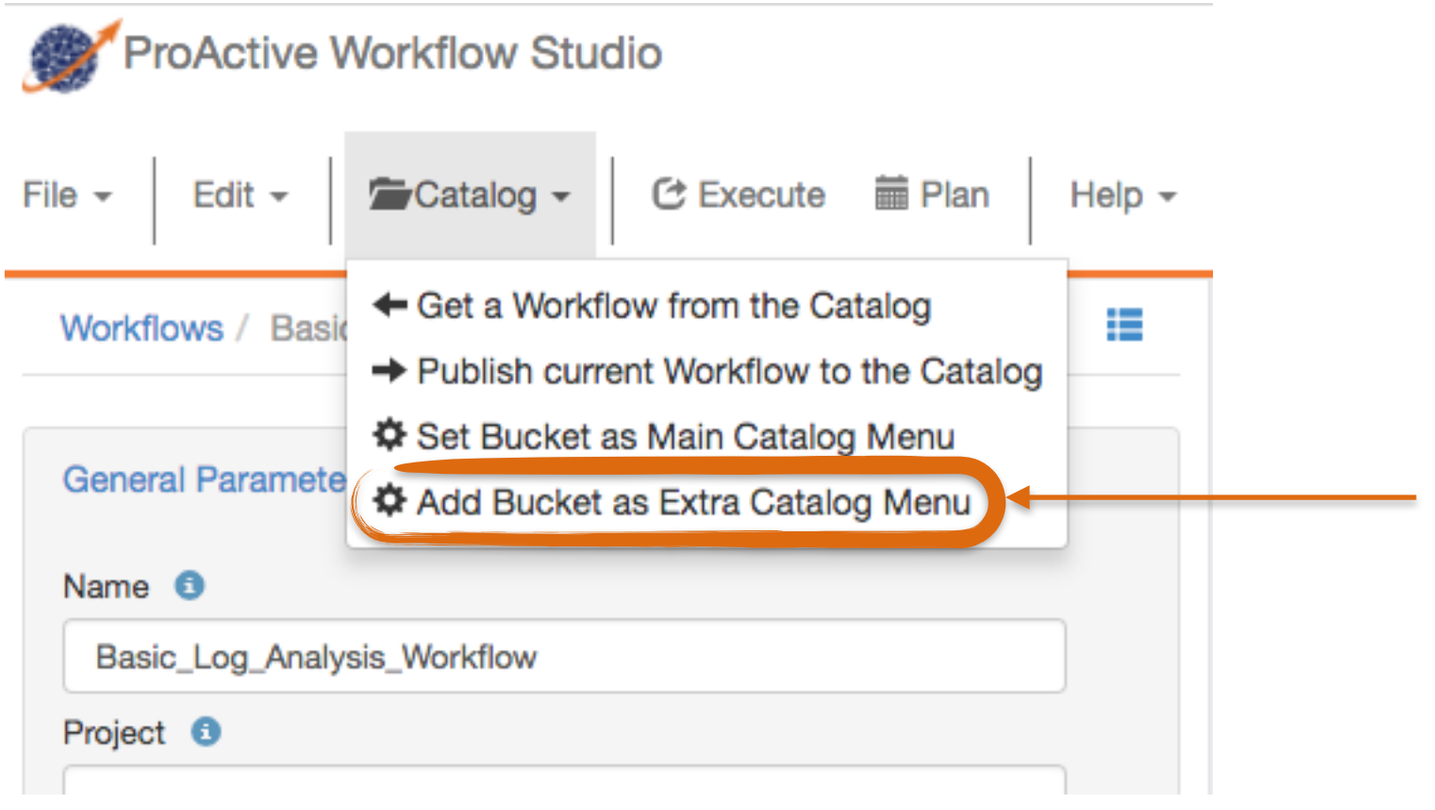
- Fill in the name of the workflow in the left panel, call it Basic_Log_Analysis_Workflow



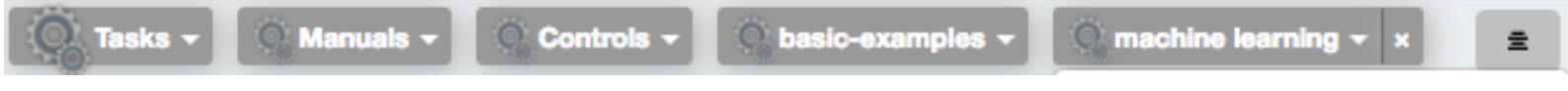
2. Add Machine Learning Bucket as Extra Catalog Menu

Follow these steps to add the machine learning bucket as extra menu:

- Click on Catalog>Add Bucket as Extra Catalog Menu

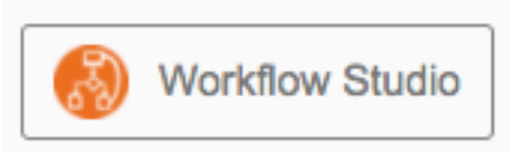


- Click on the machine-learning bucket then on the Select button.
  Now the machine-learning bucket should appear as follows:



Build your own Log Analysis Workflow

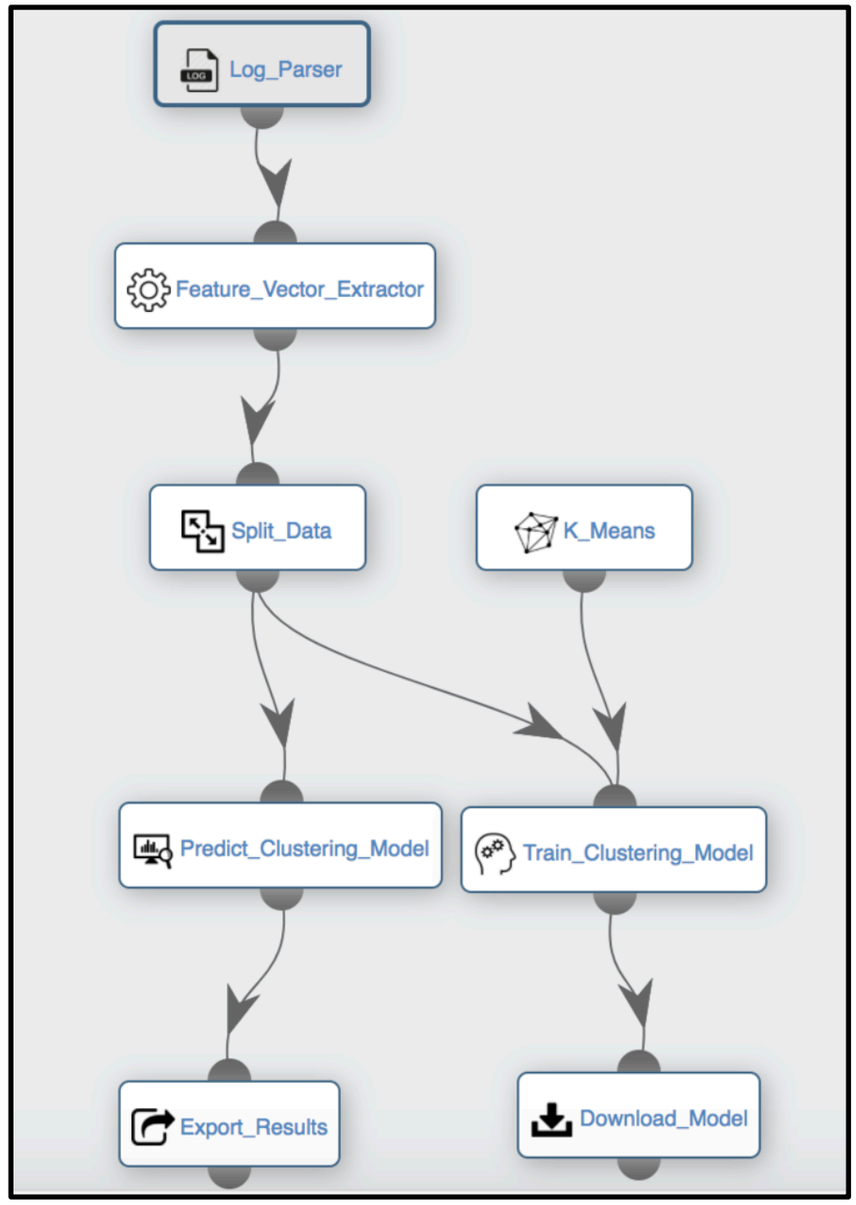
ACTIVEEON
SCALE BEYOND LIMITS



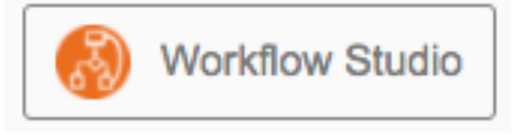
3. Compose your own Log Analysis Workflow using ML bucket

Follow these steps to compose your Log Analysis Workflow:

- **Open** the machine learning bucket
- **Drag and drop** the following tasks:
  Log_Parser
  Feature_Vector_Extractor
  Split_Data
  K_means
  Train_Clustering_Model
  Download_Model
  Predict_Clustering_Model
  Export_Results
- **Link** the different tasks in order to have a workflow similar to the workflow displayed in this image.
- [optional] **Click on** the tasks that you want to personalize by:
  modifying its parameters on the General Parameters Section
  modifying its code on the Task Implementation Section



Build your own Log Analysis Workflow



4. Execute your Log Analysis Workflow

Once your workflow is ready and you are logged in, click on Execute to submit your workflow as a job to the Scheduler.

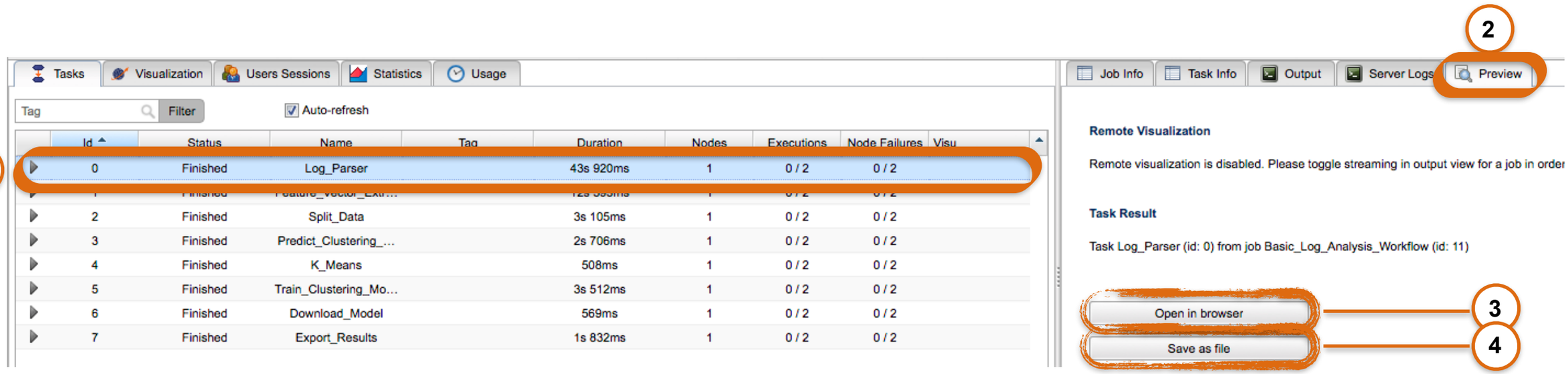
Build your own Log Analysis Workflow

ACTIVEEON
SCALE BEYOND LIMITS

Scheduling & Orchestration

5. Preview and Download the Output Results

Once your workflow is submitted, go to the **scheduler portal** and explore the results of each task.



1. Click on the task that you want to preview or download its results. For example, click on the Log Parser Task
2. Click on Preview
3. Click on Open in browser to visualize the results in your browser
4. Click on save as file to download the results in your laptop

Example of Workflows using customized machine learning tasks

Introduction

Machine Learning and Computer Vision are two core branches of computer science that allow us to create powerful AI applications. The ProActive Studio provides a fast, easy and practice way to execute different workflows using machine learning and computer vision algorithms. We present useful computer vision applications using deep learning based algorithms by convolutional neural networks for image recognition, object detection, anomaly detection, and image segmentation. In addition, two workflows to deploy and delete machine learning node source are shown. The ProActive Studio uses open source libraries, such as TensorFlow, Keras, Caffe, OpenCV, PyTorch and scikit-learn as a backend for AI workflows.

How does it work

1. Executing the functional Workflows

Image Recognition

TensorFlow_Demo_Prediction.xml returns the flower specie of a given input image.

TensorFlow_Demo_Parallel_Prediction.xml predicts three different images of flower species in parallel.

TensorFlow_Demo_Image_Classification.xml classifies a input image using deep ConvNets (specifically, VGG16)* pre-trained on the ImageNet dataset.

TensorFlow_Demo_Training.xml trains a deep ConvNets (specifically, Inception)* to recognize flower species.

Object Detection

YOLO_Demo_Object_Detection.xml detects real-world objects in a image with the YOLO* library using a pre-trained model.

Anomaly Detection

Anomaly_Detection_Demo.xml checks if exists an anomaly in a certain scene using the YOLO library. However, we show a scene where only people are allowed on a pedestrian street. Anything detected other than people is considered as anomaly.

Image Segmentation

PyTorch_Demo_Obj_Seg_Prediction.xml returns the horse segments of a given input image.

* https://www.tensorflow.org/

* http://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

How does it work

2. Training Workflows

TensorFlow_Demo_Training.xml trains a deep ConvNets (specifically, Inception)* to recognize flower species.

PyTorch_Demo_Obj_Seg_Training.xml trains an image segmentation algorithm using PyTorch* to segment horses. It segments horses in the input image using a deep neural network (DNN).

3. Node Source Deployment (ProActive, Infra, Admin aspects)

Node Source Deploy/Delete

MachineLearning_NodeSource_Deploy.xml deploys a machine learning node source.

MachineLearning_NodeSource_Delete.xml deletes a machine learning node source.Object Detection

* https://pjreddie.com/darknet/yolo/

* http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

Contents

Image Recognition

TensorFlow_Demo_Prediction.xml

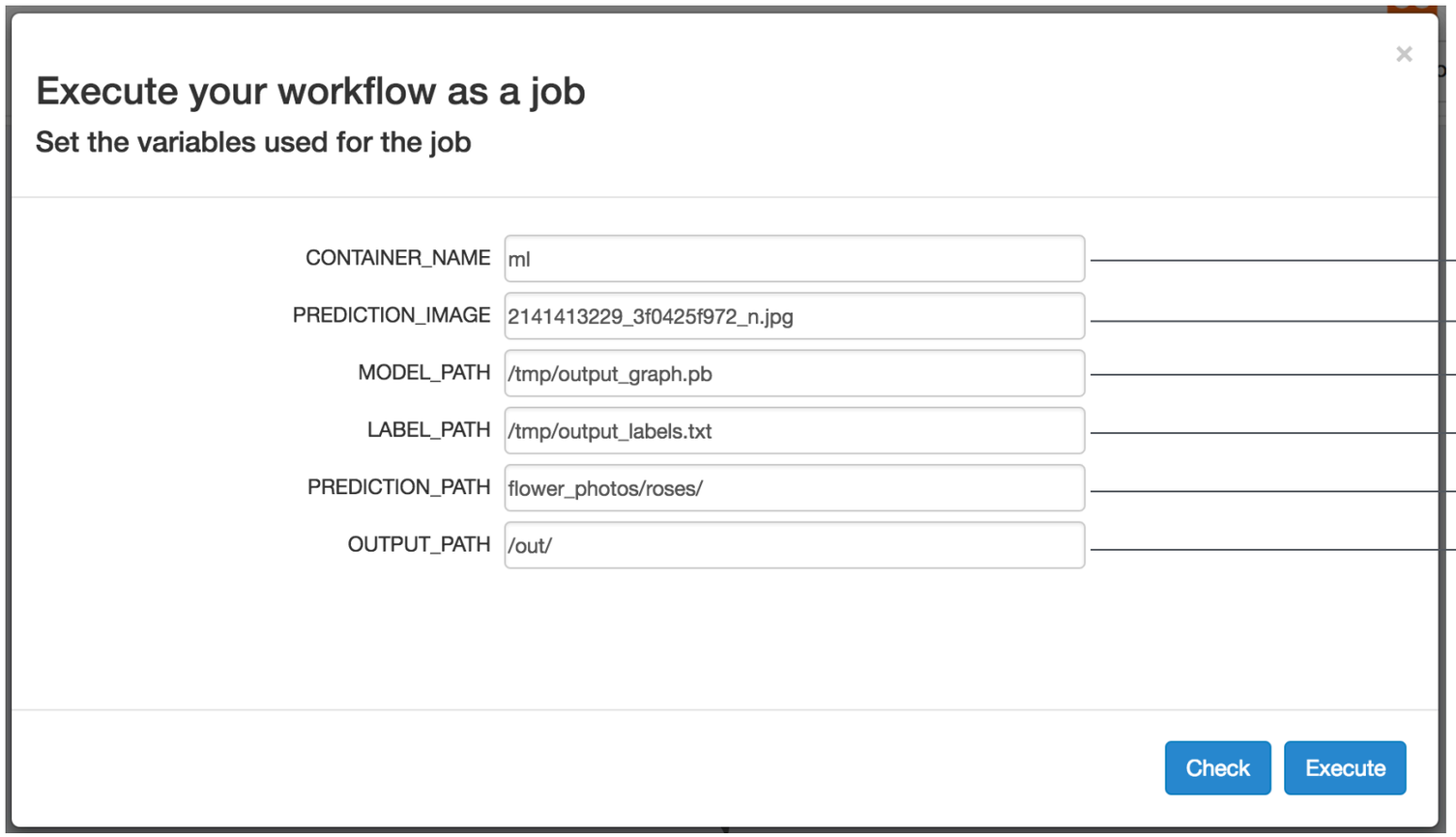


Image Recognition

TensorFlow_Demo_Prediction.xml

Load a dataset image

Show input image

Predict flower species using a previous trained model

Show the result of the classification (image label and its classification probability)

Image Recognition

TensorFlow_Demo_Prediction.xml

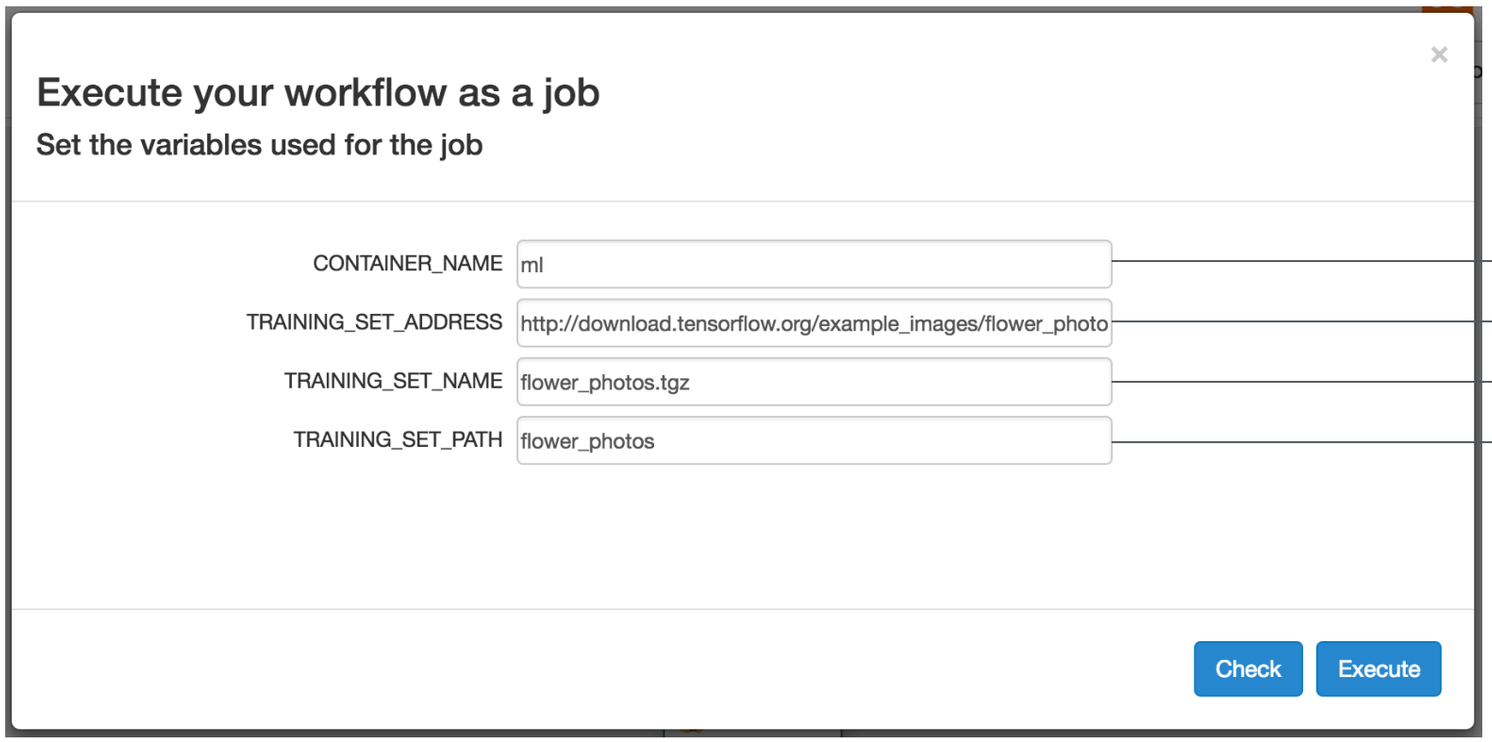


Image Recognition

ACTIVEEON
SCALE BEYOND LIMITS

TensorFlow_Demo_Prediction.xml

Input image

2141413229_3f0425f972_n.jpg

Prediction result

2141413229_3f0425f972_n.jpg

Contents

Image Recognition

TensorFlow_Demo_Parallel_Prediction.xml

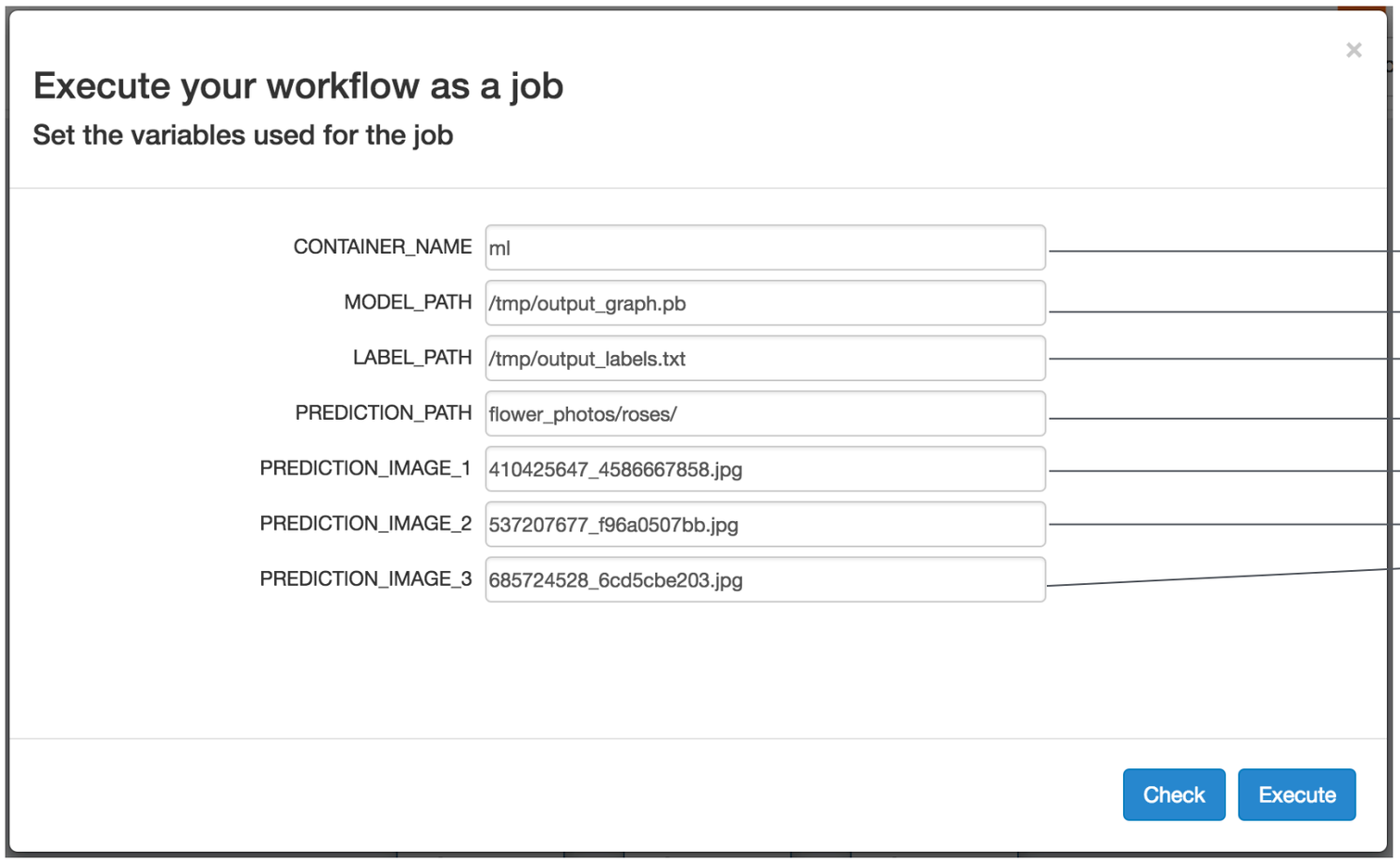


Image Recognition

TensorFlow_Demo_Parallel_Prediction.xml

Load a dataset image

Show the three input images

Predict three different images of flower species

Show the result of the predictions for the three flower species

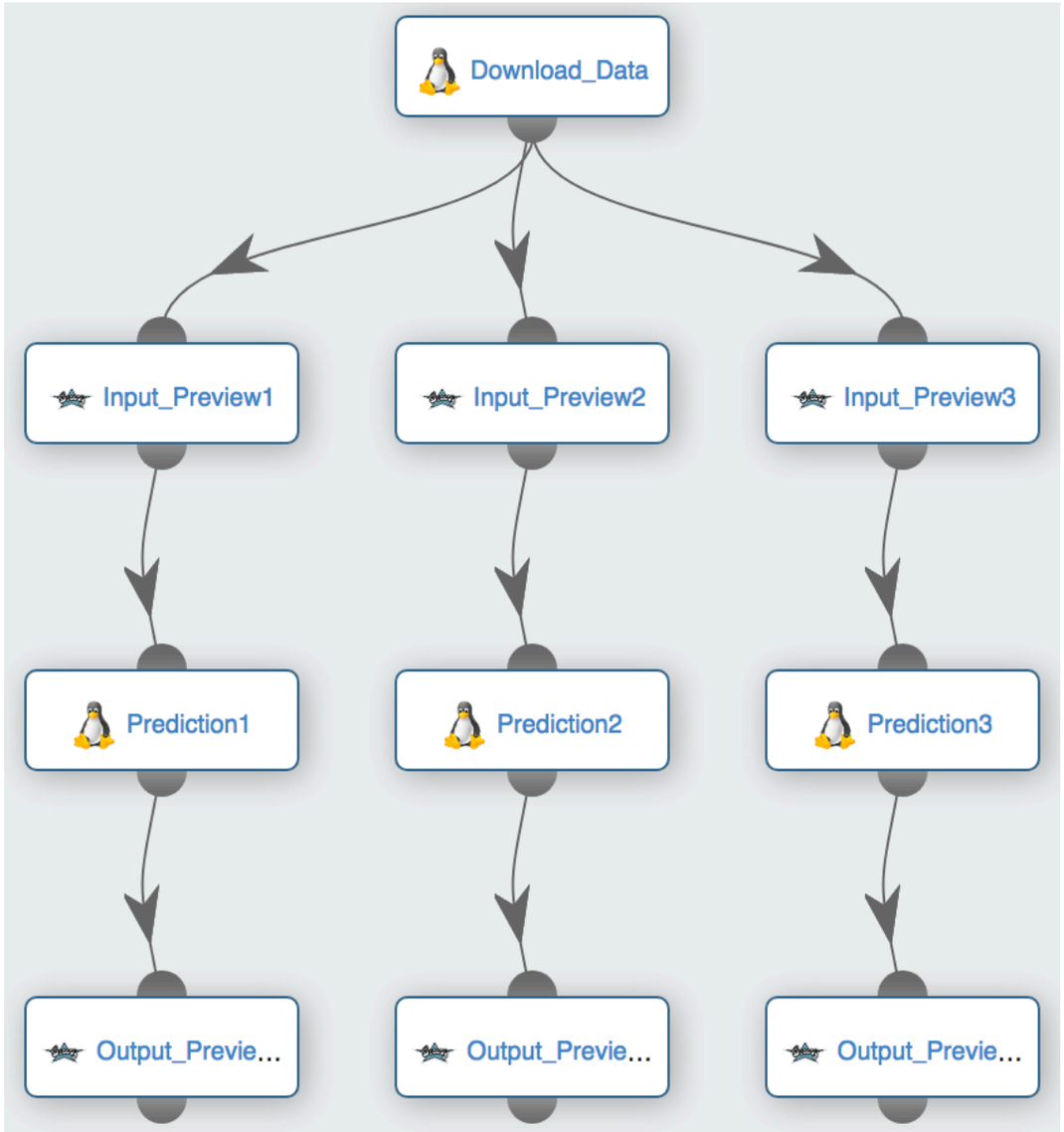


Image Recognition

ACTIVEEON
SCALE BEYOND LIMITS

TensorFlow_Demo_Parallel_Prediction.xml

410425647_4586667858.jpg

537207677_f96a0507bb.jpg

685724528_6cd5cbe203.jpg

Contents

ACTIVEEON
SCALE BEYOND LIMITS

1. **Executing the functional Workflows**

Image Recognition

* TensorFlow_Demo_Prediction.xml
* TensorFlow_Demo_Parallel_Prediction.xml
* **TensorFlow_Demo_Image_Classification.xml**

Object Detection

* YOLO_Demo_Object_Detection.xml

Anomaly Detection

* Anomaly_Detection_Demo.xml

Image Segmentation

* PyTorch_Demo_Obj_Seg_Prediction.xml

2. **Training Workflows**

* TensorFlow_Demo_Training.xml
* PyTorch_Demo_Obj_Seg_Training.xml

3. **Node Source Deployment (ProActive, Infra, Admin aspects)**

* MachineLearning_NodeSource_Deploy.xml
* MachineLearning_NodeSource_Delete.xml

Image Recognition

TensorFlow_Demo_Image_Classification.xml

Image Recognition

TensorFlow_Demo_Image_Classification.xml

Load the input image

Show the input image

Classify the input image using deep ConvNet

Show the result of the classification (image label and its classification probability)

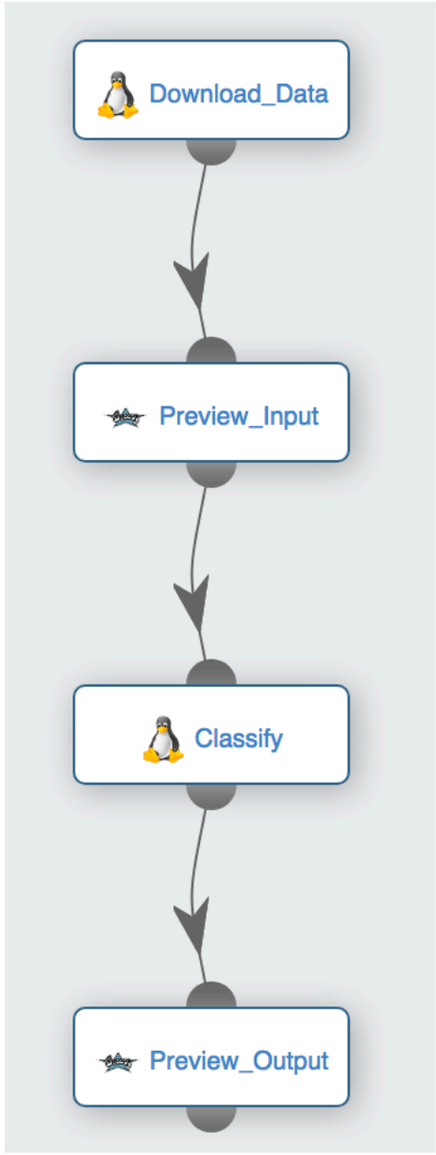


Image Recognition

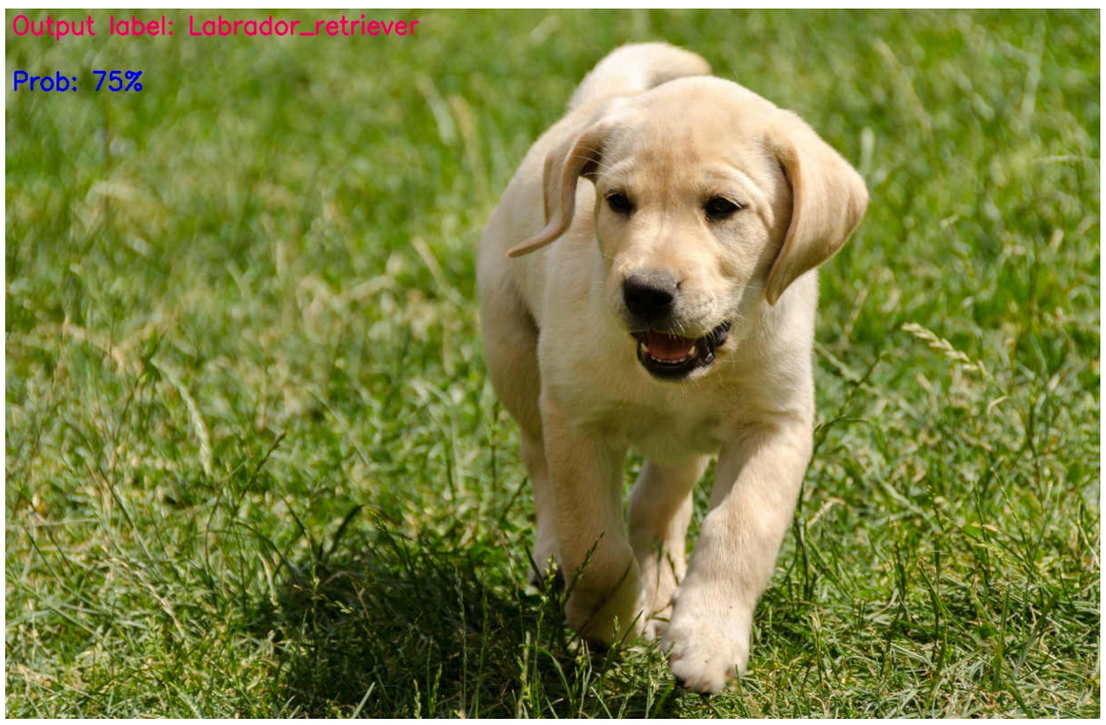
ACTIVEEON
SCALE BEYOND LIMITS

TensorFlow_Demo_Image_Classification.xml

Input image

dog.jpg

Classification result



dog.jpg

Contents

1. Executing the functional Workflows

Image Recognition

* TensorFlow_Demo_Prediction.xml
* TensorFlow_Demo_Parallel_Prediction.xml
* TensorFlow_Demo_Image_Classification.xml

Object Detection

*** YOLO_Demo_Object_Detection.xml**

Anomaly Detection

* Anomaly_Detection_Demo.xml

Image Segmentation

* PyTorch_Demo_Obj_Seg_Prediction.xml

2. Training Workflows

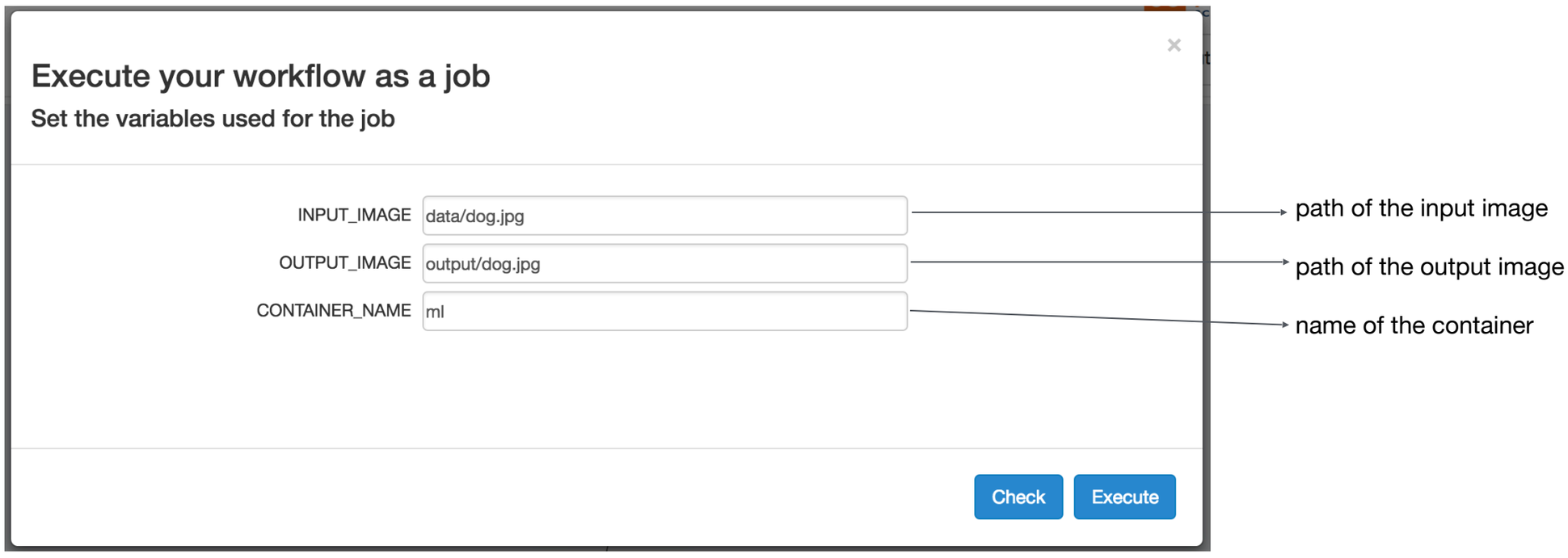
* TensorFlow_Demo_Training.xml
* PyTorch_Demo_Obj_Seg_Training.xml

3. Node Source Deployment (ProActive, Infra, Admin aspects)

* MachineLearning_NodeSource_Deploy.xml
* MachineLearning_NodeSource_Delete.xml

Object Detection

YOLO_Demo_Object_Detection.xml



Object Detection

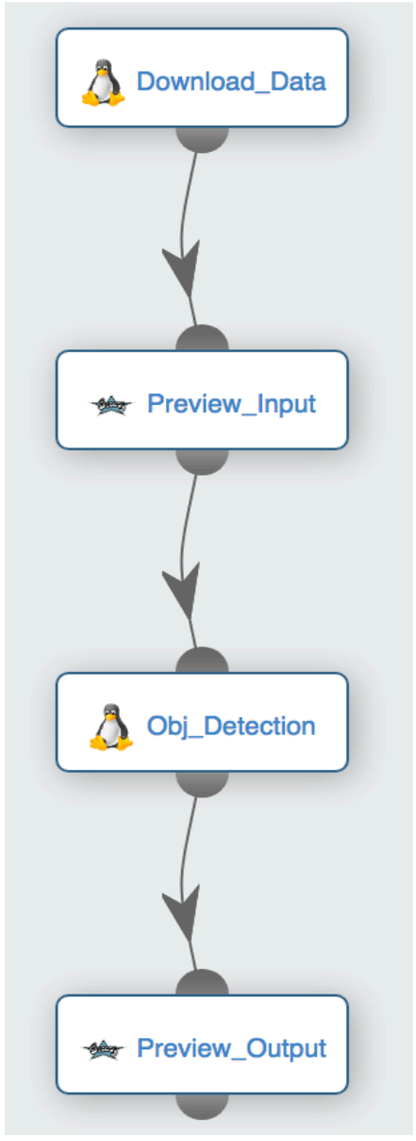
YOLO_Demo_Object_Detection.xml

Load the input image

Show the input image

Detect different objects in the input image using a deep ConvNet

Show the result of the detection (the objects found in the image)



Object Detection

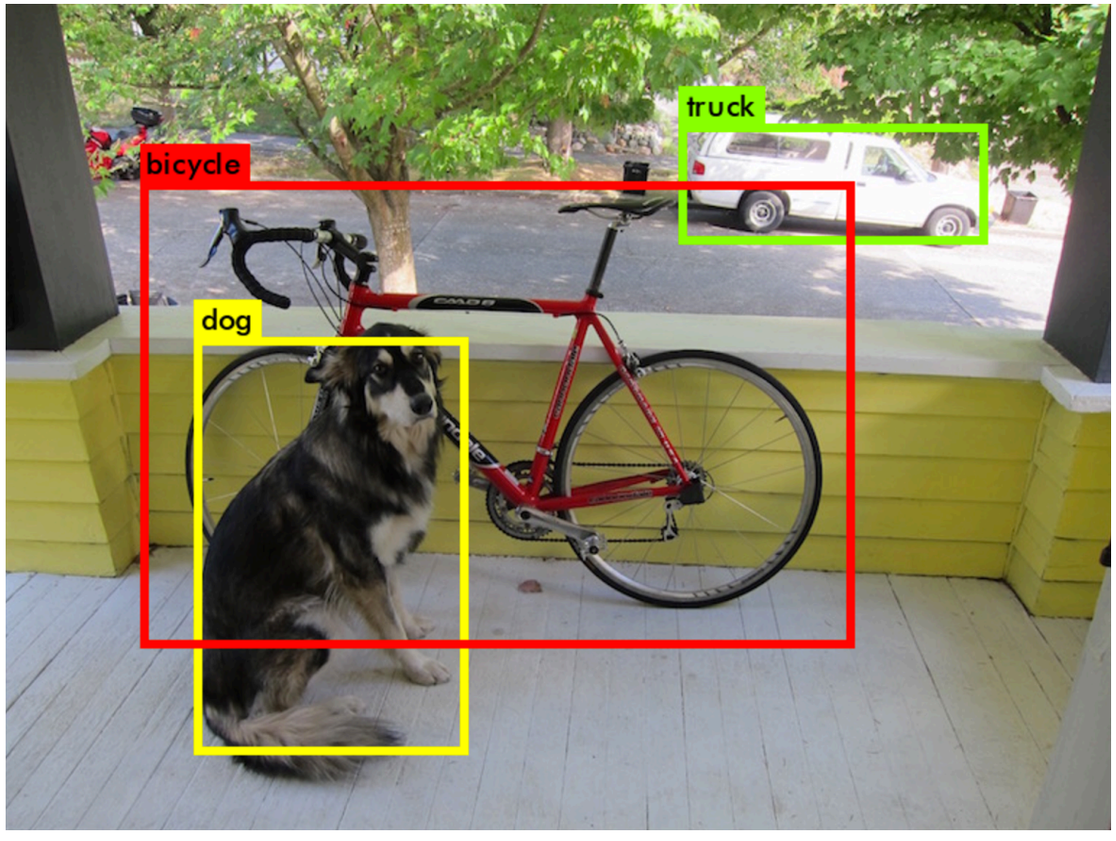
ACTIVEEON
SCALE BEYOND LIMITS

YOLO_Demo_Object_Detection.xml

Input image

dog.jpg

Detection results



dog.jpg

Contents

ACTIVEEON
SCALE BEYOND LIMITS

1. **Executing the functional Workflows**

Image Recognition

* TensorFlow_Demo_Prediction.xml
* TensorFlow_Demo_Parallel_Prediction.xml
* TensorFlow_Demo_Image_Classification.xml

Object Detection

* YOLO_Demo_Object_Detection.xml

Anomaly Detection

* **Anomaly_Detection_Demo.xml**

Image Segmentation

* PyTorch_Demo_Obj_Seg_Prediction.xml

2. **Training Workflows**

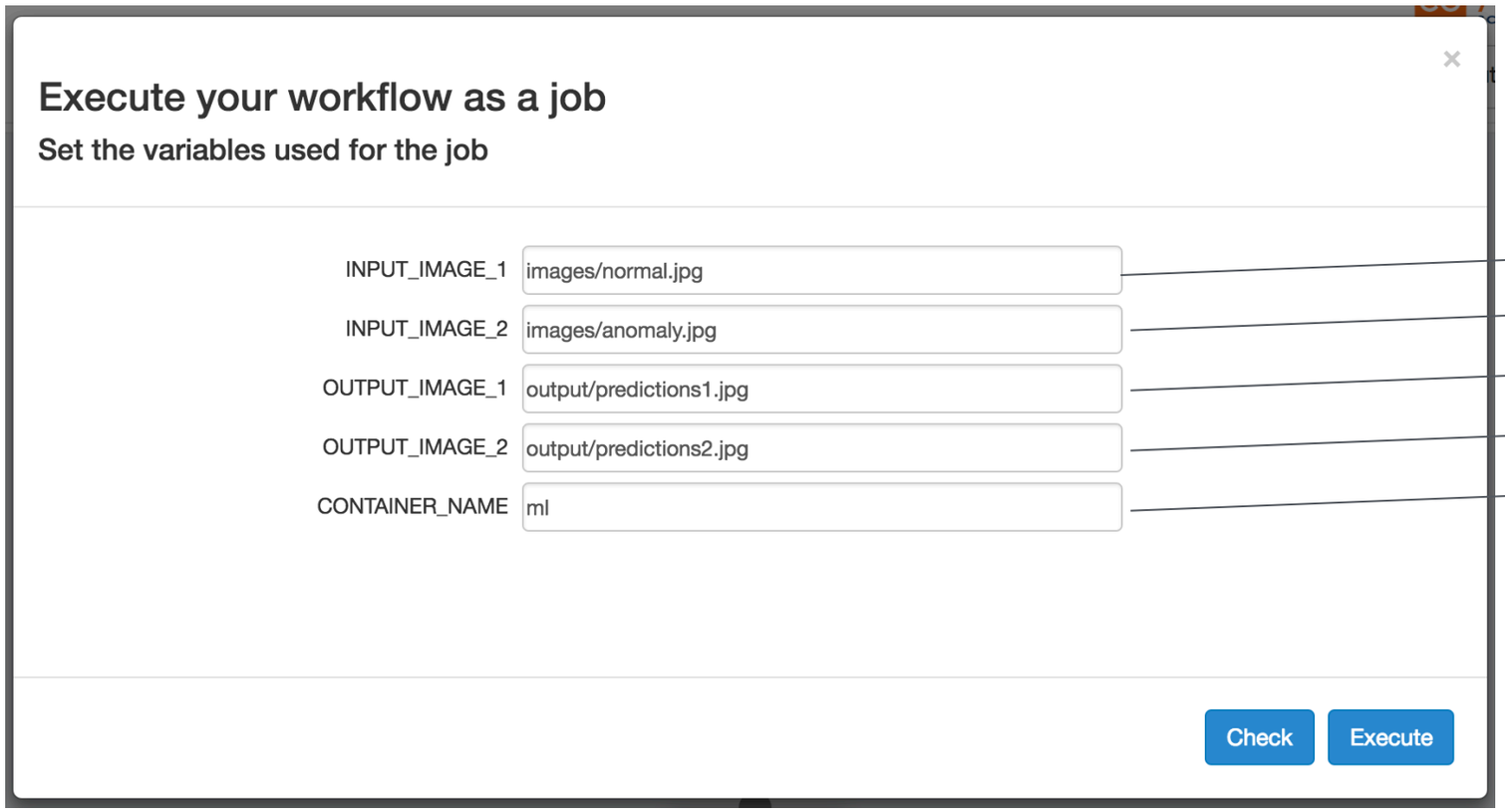
* TensorFlow_Demo_Training.xml
* PyTorch_Demo_Obj_Seg_Training.xml

3. **Node Source Deployment (ProActive, Infra, Admin aspects)**

* MachineLearning_NodeSource_Deploy.xml
* MachineLearning_NodeSource_Delete.xml

Anomaly Detection

Anomaly_Detection_Demo.xml



Anomaly Detection

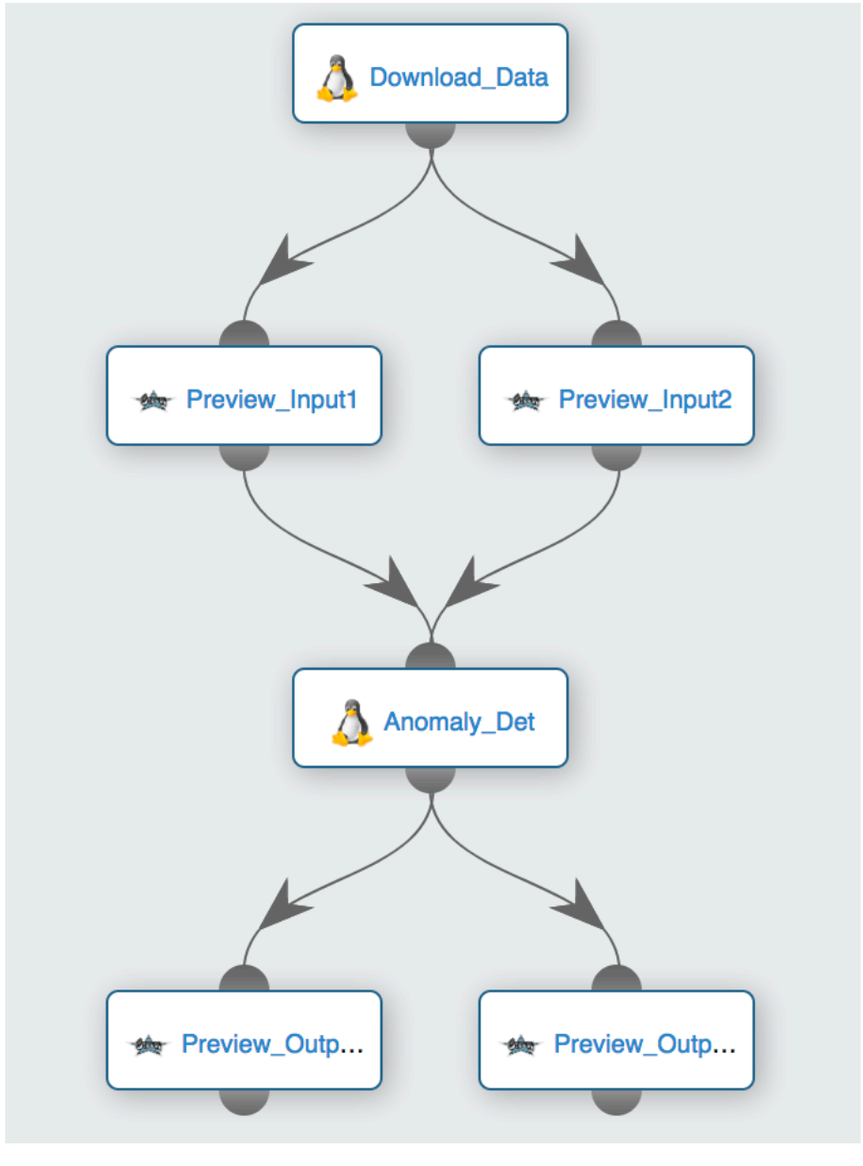
Anomaly_Detection_Demo.xml

Load the dataset

Show the input image1 and the input image 2

Detect different objects in the input image using a deep ConvNet

Show the result of the detection (image 1 and image 2)



Anomaly Detection

ACTIVEEON
SCALE BEYOND LIMITS

Anomaly_Detection_Demo.xml

Input image

normal.jpg

Output image

predictions.jpg

Anomaly Detection

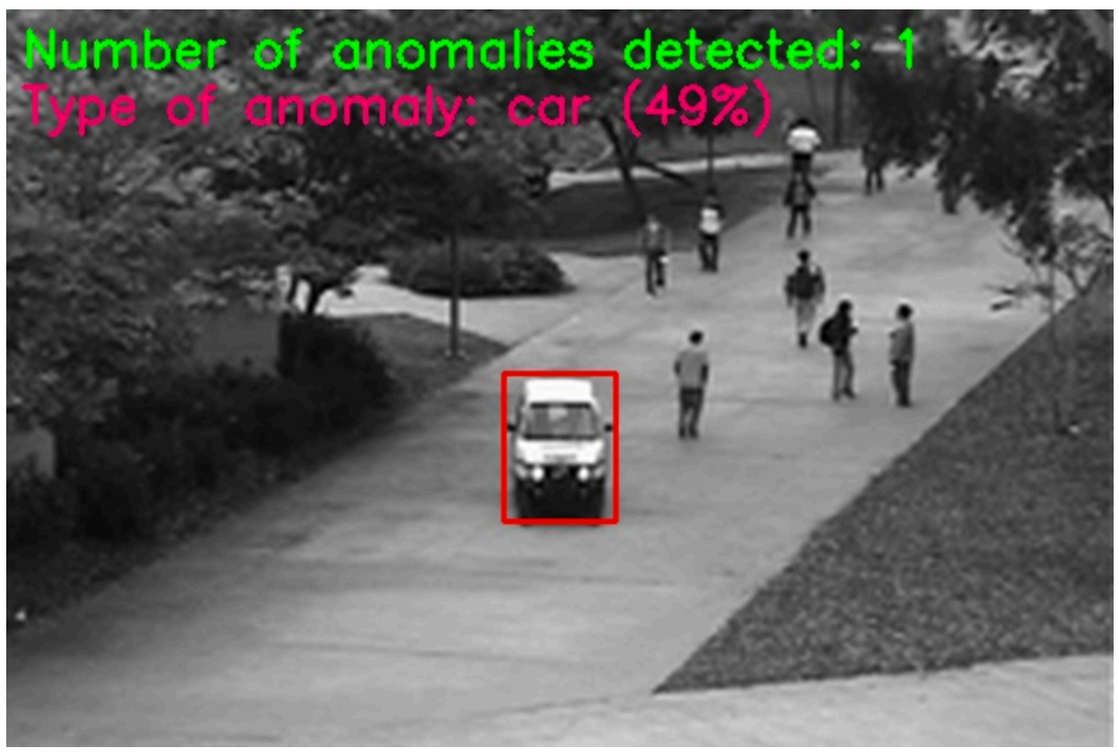
ACTIVEEON
SCALE BEYOND LIMITS

Anomaly_Detection_Demo.xml

Input image

anomaly.jpg

Output image



predictions.jpg

Contents

ACTIVEEON
SCALE BEYOND LIMITS

1. Executing the functional Workflows

Image Recognition

* TensorFlow_Demo_Prediction.xml
* TensorFlow_Demo_Parallel_Prediction.xml
* TensorFlow_Demo_Image_Classification.xml

Object Detection

* YOLO_Demo_Object_Detection.xml

Anomaly Detection

* Anomaly_Detection_Demo.xml

Image Segmentation

* **PyTorch_Demo_Obj_Seg_Prediction.xml**

2. Training Workflows

* TensorFlow_Demo_Training.xml
* PyTorch_Demo_Obj_Seg_Training.xml

3. Node Source Deployment (ProActive, Infra, Admin aspects)

* MachineLearning_NodeSource_Deploy.xml
* MachineLearning_NodeSource_Delete.xml

Image Segmentation

PyTorch_Demo_Obj_Seg_Prediction.xml

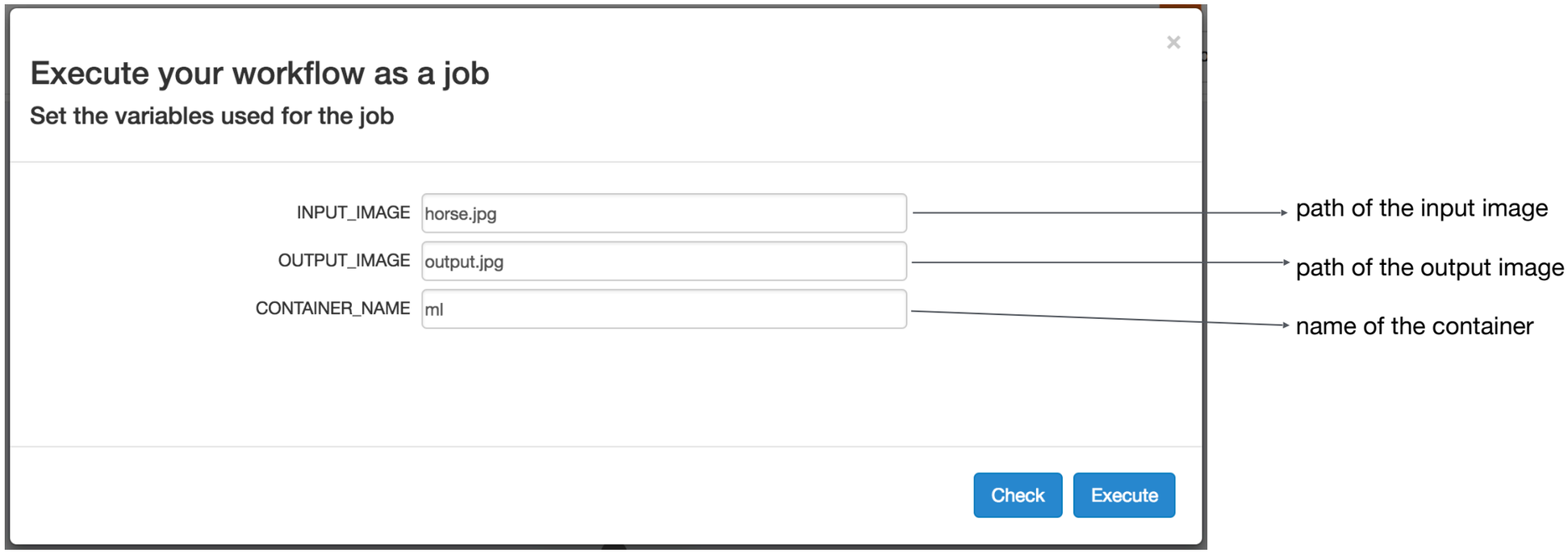


Image Segmentation

PyTorch_Demo_Obj_Seg_Prediction.xml

Load the input image

Show the input image

Segment the input image using a deep ConvNet

Show the result of the segmentation

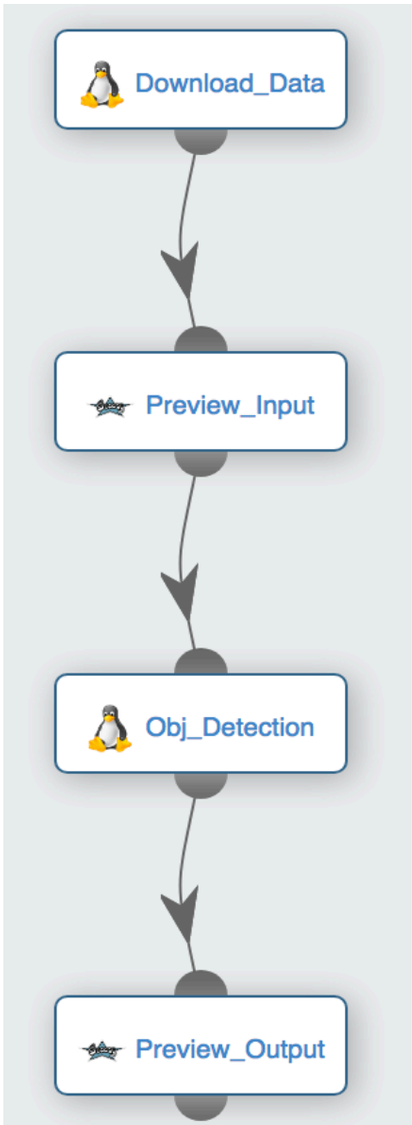


Image Segmentation

ACTIVEEON
SCALE BEYOND LIMITS

PyTorch_Demo_Obj_Seg_Prediction.xml

Input image

horse.jpg

Image segmentation results

horse.jpg

Contents

1. Executing the functional Workflows

Image Recognition

* TensorFlow_Demo_Prediction.xml
* TensorFlow_Demo_Parallel_Prediction.xml
* TensorFlow_Demo_Image_Classification.xml

Object Detection

* YOLO_Demo_Object_Detection.xml

Anomaly Detection

* Anomaly_Detection_Demo.xml

Image Segmentation

* PyTorch_Demo_Obj_Seg_Prediction.xml

2. Training Workflows

* **TensorFlow_Demo_Training.xml**
* PyTorch_Demo_Obj_Seg_Training.xml

3. Node Source Deployment (ProActive, Infra, Admin aspects)

* MachineLearning_NodeSource_Deploy.xml
* MachineLearning_NodeSource_Delete.xml

Image Recognition

TensorFlow_Demo_Training.xml

Load a dataset image

Train a deep ConvNet to recognize flower species

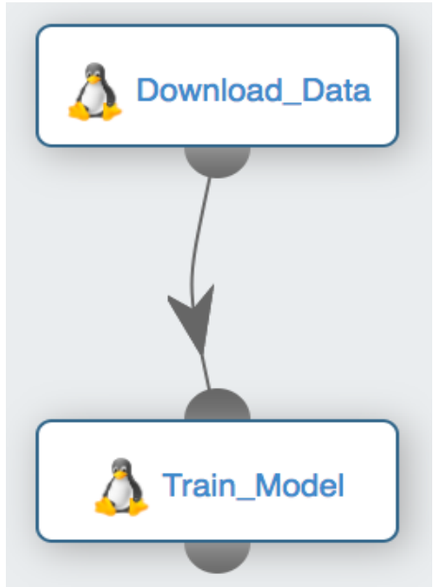
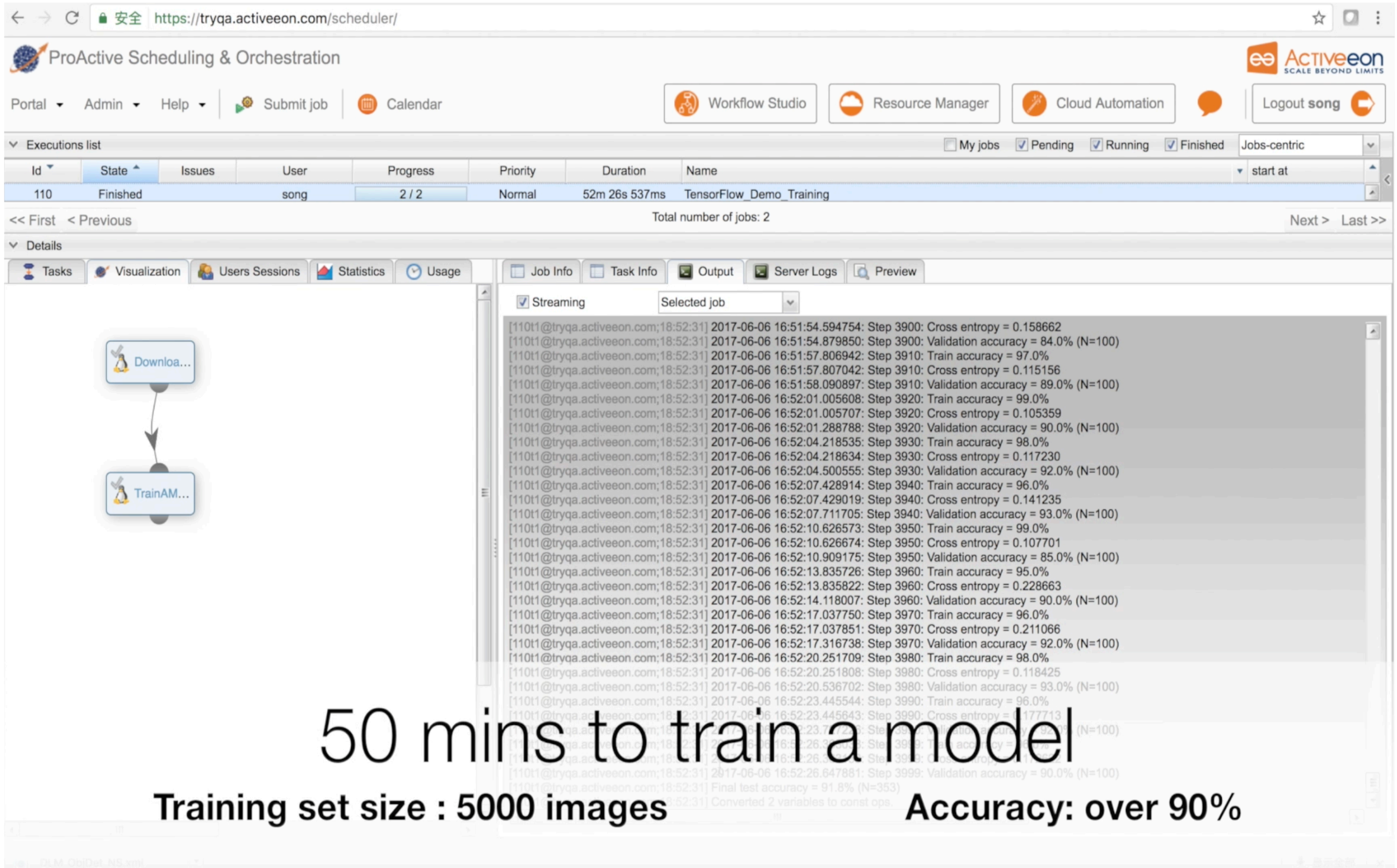


Image Recognition

TensorFlow_Demo_Training.xml

Image Recognition

TensorFlow_Demo_Training.xml



50 mins to train a model

Training set size : 5000 images **Accuracy: over 90%**

Contents

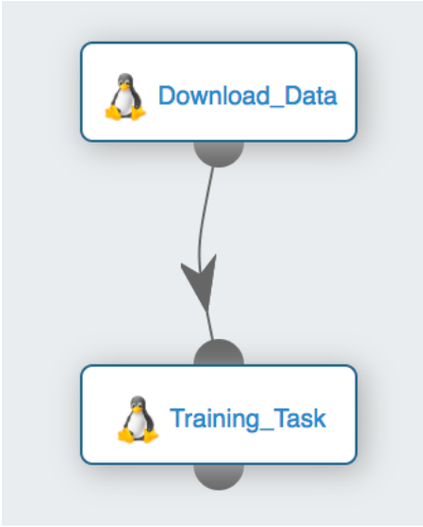
Image Segmentation

PyTorch_Demo_Obj_Seg_Training.xml

Image Segmentation

PyTorch_Demo_Obj_Seg_Training.xml

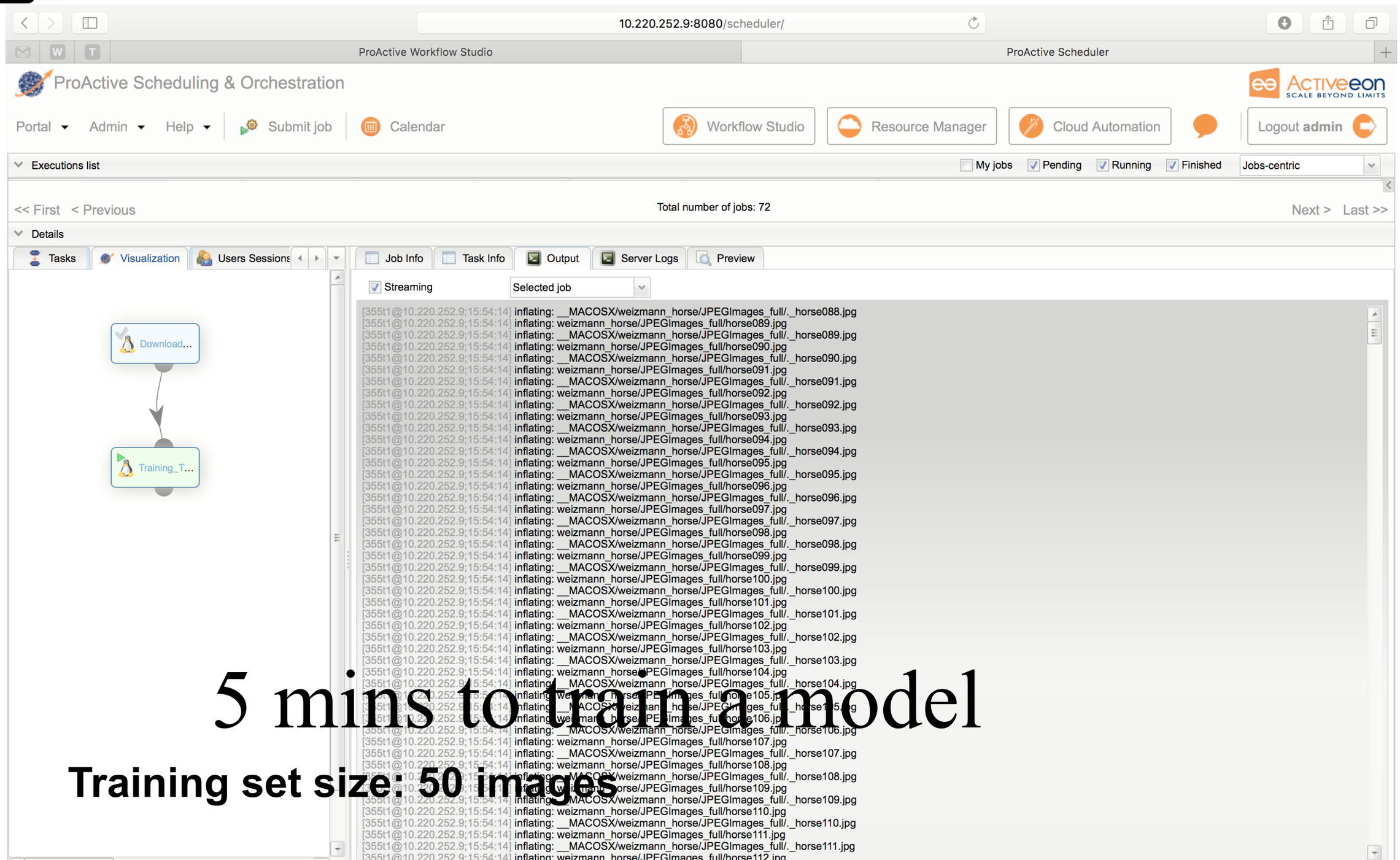
Load a dataset image



Train an image segmentation algorithm with deep neural network to recognise horses.

Image Segmentation

PyTorch_Demo_Obj_Seg_Training.xml



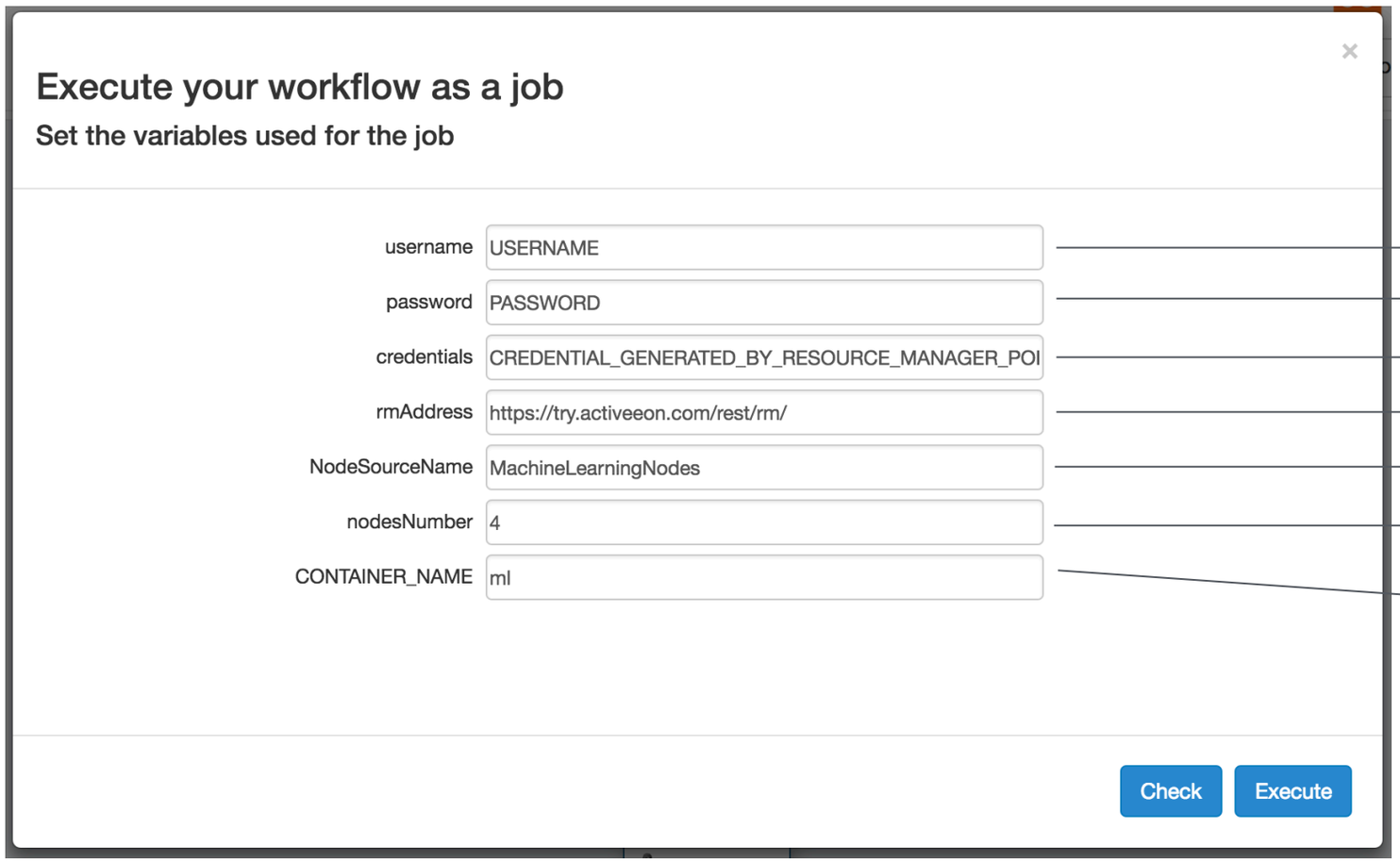
5 mins to train a model

Training set size: 50 images

Contents

Node Source

MachineLearning_NodeSource_Deploy.xml

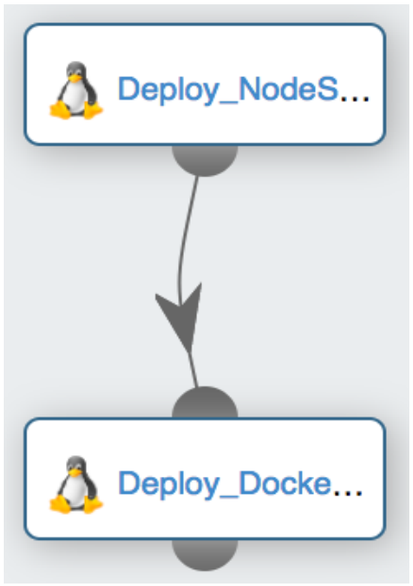


Node Source

MachineLearning_NodeSource_Deploy.xml

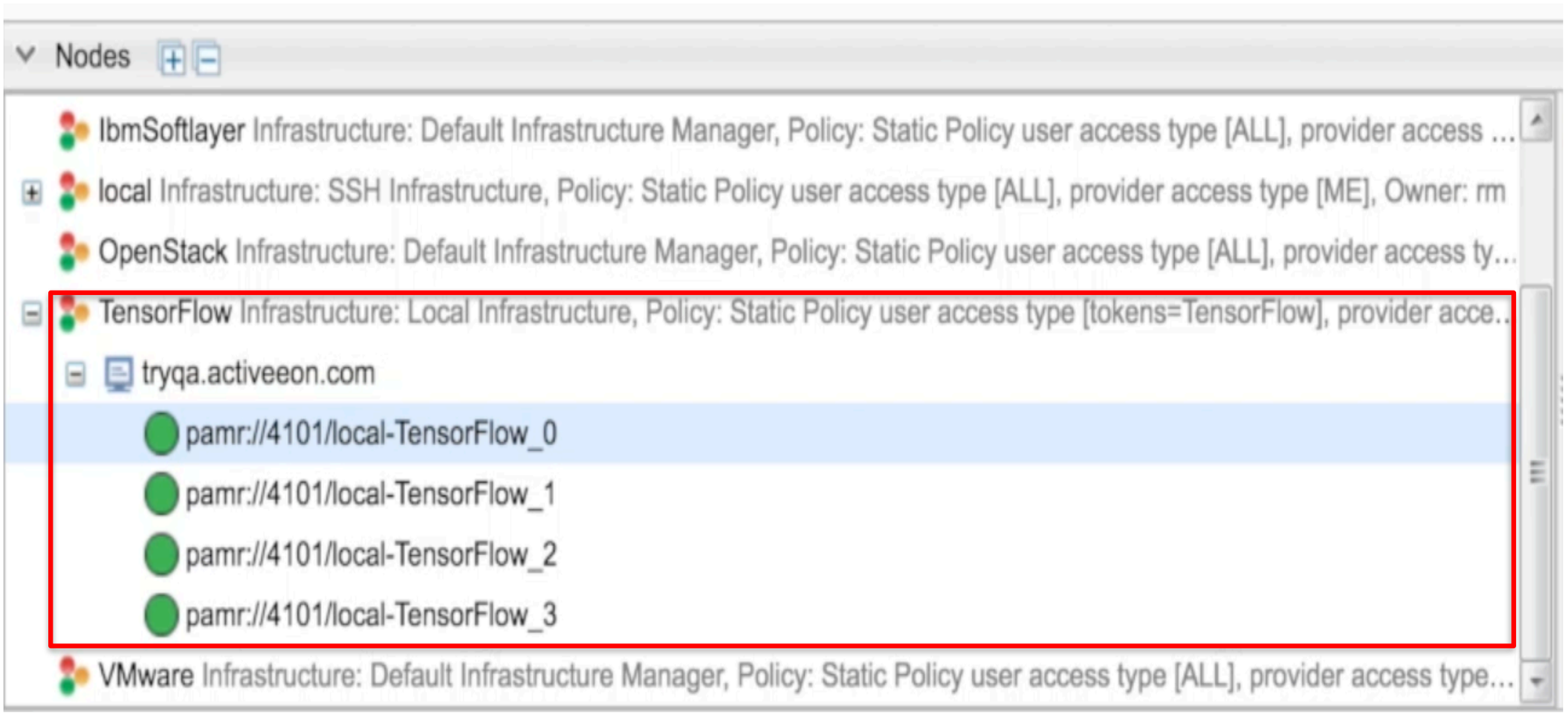
Deploy node source

Deploy docker container



Node Source

MachineLearning_NodeSource_Deploy.xml



Contents

1. Executing the functional Workflows

Image Recognition

* TensorFlow_Demo_Prediction.xml
* TensorFlow_Demo_Parallel_Prediction.xml
* TensorFlow_Demo_Image_Classification.xml

Object Detection

* YOLO_Demo_Object_Detection.xml

Anomaly Detection

* Anomaly_Detection_Demo.xml

Image Segmentation

* PyTorch_Demo_Obj_Seg_Prediction.xml

2. Training Workflows

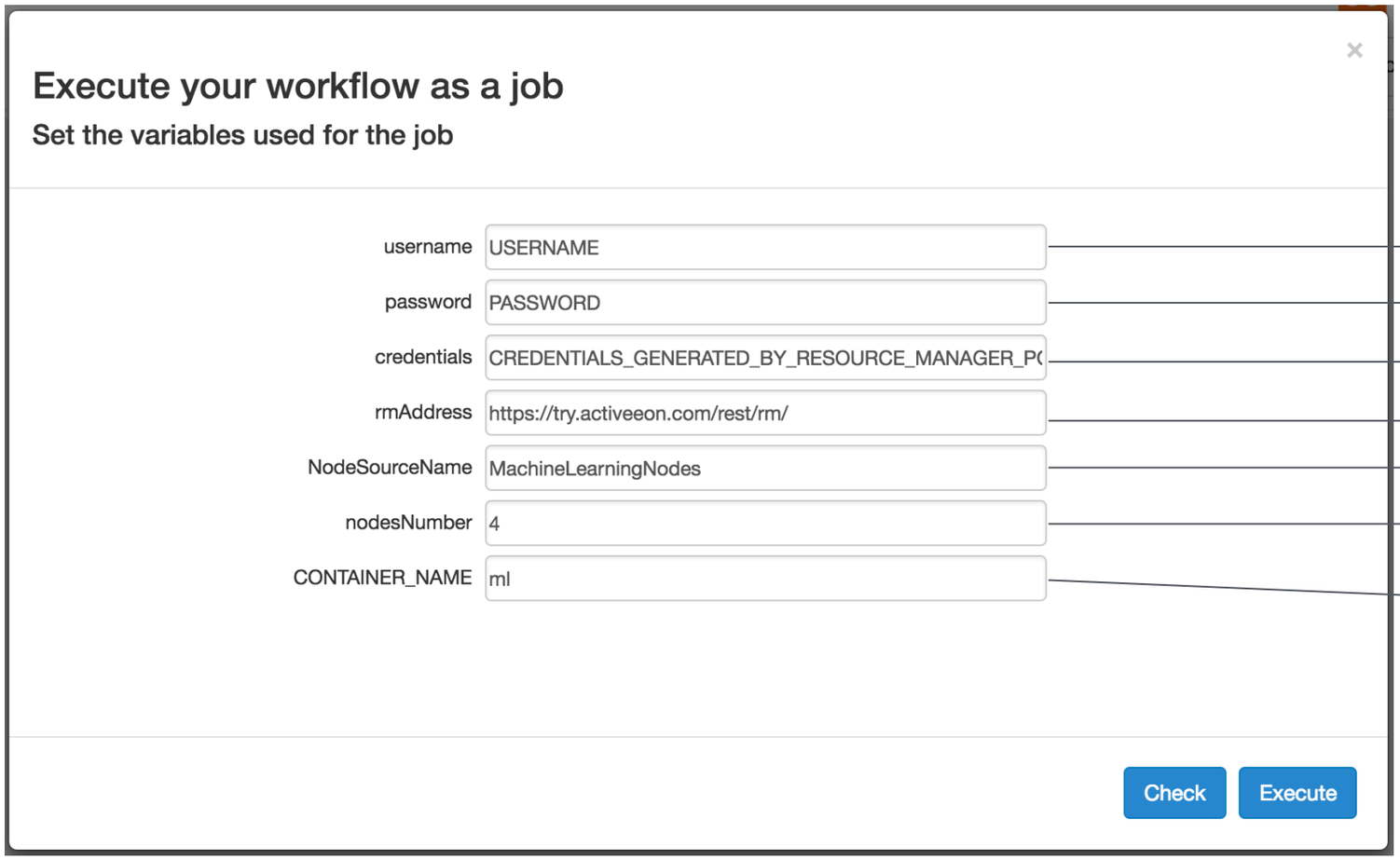
* TensorFlow_Demo_Training.xml
* PyTorch_Demo_Obj_Seg_Training.xml

3. Node Source Deployment (ProActive, Infra, Admin aspects)

* MachineLearning_NodeSource_Deploy.xml
* **MachineLearning_NodeSource_Delete.xml**

Node Source

MachineLearning_NodeSource_Delete.xml

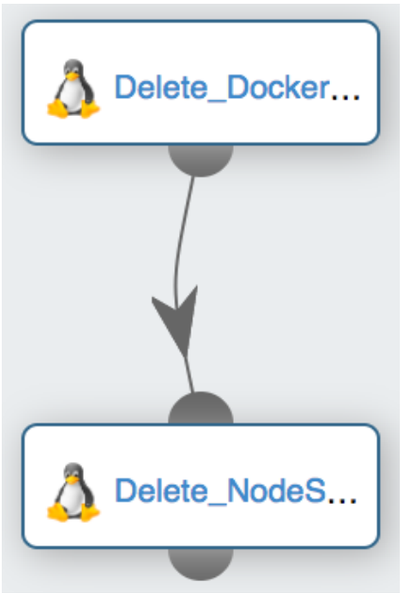


Node Source

MachineLearning_NodeSource_Delete.xml

Delete docker container

Delete node source



Node Source

MachineLearning_NodeSource_Delete.xml