IBM Visual Insights

User Guide

Version 1 Release 1

IBM Visual Insights

User Guide

Version 1 Release 1

Note

Before using this information and the product it supports, read the information in “Notices” on page 25.

First edition (September 2017)

Contents

Chapter 1. Product overview

IBM® Visual Insights is a manufacturing quality monitoring and alerting solution that can take in images of in-process and finished products and assemblies, and classify them into defect categories.

Roles

To understand Visual Insights, it is helpful to understand how the different roles interact with the product.

Table 1. Visual Insights roles

Role	Description
Model Manager	Manages defect types and models, uploads image sets for specific defect types, creates model training requests for the Data Scientist, and distributes executable models to edges.
Data Scientist	Trains models based on image sets and defect types created by the Model Manager. The training of models is performed outside Visual Insights. It is recommended that the Data Scientist use the NVIDIA Deep Learning GPU Training System (DIGITS) to train models.
Inspector	Verifies the inspection results that are produced by the product, changes defect types if necessary, marks unknown defect types and passes them to the Inspector Supervisor for further evaluation.
Inspector Supervisor	Double-checks the Inspector's inspection results. Reviews and classifies unknown defect types. Reviews the KPI dashboard, which includes total defects, defect rate, overkill and escaping defect rates.

How is data backed up and restored?

IBM Open Platform redundancy is used to protect customer data in a big data environment. In addition, Tivoli® Storage Manager is used to back up data in the production environment, which includes Linux files. Linux files include customer uploaded files and middleware/application configuration/log files.

The following table shows the backup schedule for various aspects of the solution.

Table 2. Backup schedule for solution data

Data	Backup type	Frequency	Time (Central Time)	Retention period
Files	Full	Bi-weekly	00:00 - 03:00 on the 1st and 16th day of each month	5 weeks

Table 2. Backup schedule for solution data (continued)

Data	Backup type	Frequency	Time (Central Time)	Retention period
Files	Incremental	Twice daily	00:00 - 03:00 and 12:00 - 15:00	14 days

During the backup window, the solution is accessible. However, performance may be impacted.

In the event of a system failure that causes data damage or loss, IBM will help to restore the data to the recovery time points, according to its backup policy.

What's new in this release

The following new features are available in IBM Visual Insights.

New Features and Enhancements

- You can now use image groups to represent the same type of images by using one or more compressed image files.
- Added support for multiple model versions that share the same image groups, but use different image files to train the model.
- The model retrain process was added. You can automatically or manually retrain a new model version by using different image files.
- Added support for model validation. The validation process calculates and shows a model accuracy report based on validation image files.
- You can now use defect boxes and defect types on an image to mark the defect location for the inspector.
- You can now show multiple defect locations on one image. You can now add, adjust, and delete defect boxes on an image.
- Updated the KPI dashboard to include defect per unit and defect rate for the inspector supervisor.
- Added support for the object detection model to detect multiple defects in one image with the CNN model.

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products.

For information about the commitment that IBM has to accessibility, see the IBM Accessibility Center (www.ibm.com/able).

HTML documentation has accessibility features. PDF documents are supplemental and, as such, include no added accessibility features.

Chapter 2. Creating edge systems

Visual Insights consists of the center application and the edges. Edges are Linux systems that are used to perform runtime defect detection.

Edge systems use the Caffe deep-learning framework. Caffe is a dedicated artificial neural network (ANN) training environment. Deep learning requires significant processing resources. Deep learning can be performed efficiently by using a graphics processing unit (GPU). Although most deep learning frameworks also support CPU processing, GPU processing provides reasonable performance for production environments.

Edge system requirements

Before creating an edge system, ensure that your system meets the requirements.

- Ubuntu 16.04
- 4-core processor
- 64GB memory
- 2TB hard disk drive
- One or more NVIDIA GPU cards

Installing NVIDIA GPU packages

To enable GPU processing, you must install the required NVIDIA GPU packages.

Procedure

1. Download and install the drivers for your NVIDIA GPU. The NVIDIA driver list for Ubuntu is available at the following link: Binary Driver How to - Nvidia.
2. Download the NVIDIA CUDA 8.0 toolkit from the following link: CUDA 8.0 downloads On the download website, choose **Linux x86_64 Ubuntu 16.04 deb (network)** as the target platform. A `cuda-repo-ubuntu1604_8.0.61-1_amd64.deb` installation file is downloaded.
3. Install the CUDA file on the target server using the following commands:
   ```
   sudo dpkg -i cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
   sudo apt-get update
   sudo apt-get install cuda
   ```
4. Download the NVIDIA CUDA Deep Neural Network library CUDNN-v5.1 from the following link: NVIDIA cuDNN. You must register before downloading.
5. Unpack the downloaded file `cudnn-8.0-linux-x64-v5.1.tgz` to the cuda installation directory using the following command:
   ```
   sudo tar -xvf cudnn-8.0-linux-x64-v5.1.tgz -C /usr/local
   ```
6. Set the environment variable using the following command:
   ```
   export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
   ```
   Also add this command to the `~/.bashrc` script.
7. Install the NVIDIA NCCL package using the following commands:
   ```
   git clone https://github.com/NVIDIA/nccl.git
   cd nccl
   sudo make install -j4
   ```

Installing Caffe

Before using an edge, you must install the Caffe deep-learning framework and related packages. Caffe is used for model training and defect classification.

Procedure

1. Install the packages that are required for Caffe by using the following commands:
```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install -y build-essential cmake git pkg-config
sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev
libhdf5-serial-dev protobuf-compiler
sudo apt-get install -y libatlas-base-dev libjasper-dev
sudo apt-get install -y --no-install-recommends libboost-all-dev
sudo apt-get install -y libgflags-dev libgoogle-glog-dev liblmdb-dev
sudo apt-get install -y python-pip
sudo apt-get install -y python-dev
sudo apt-get install -y python-numpy python-scipy
sudo apt-get install -y libopencv-dev
```
2. Download the Caffe source code by using the following command:
```
wget https://github.com/BVLC/caffe/archive/rc5.zip
```
3. Unpack the package and enter the package directory by using the following commands:
```
unzip rc5.zip
cd ./caffe-rc5
```
4. Make a copy of the make configuration file by using the following command:
```
cp Makefile.config.example Makefile.config
```
5. Add the following variables in the `Makefile.config` file:
```
USE_CUDNN := 1
CUDA_DIR := /usr/local/cuda-8.0
PYTHON_INCLUDE := /usr/include/python2.7 \
/usr/lib/python2.7/dist-packages/numpy/core/include
PYTHON_LIB := /usr/lib/x86_64-linux-gnu
WITH_PYTHON_LAYER := 1
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include
/usr/include/hdf5/serial
LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib
/usr/lib/x86_64-linux-gnu /usr/lib/x86_64-linux-gnu/hdf5/serial
```
6. In the `caffe-rc5` directory, run the following command:
```
find . -type f -exec sed -i -e 's^"hdf5.h"^"hdf5/serial/hdf5.h"^g' -e
's^"hdf5_hl.h"^"hdf5/serial/hdf5_hl.h"^g' '{}' \;
```
7. Run the following commands:
```
cd /usr/lib/x86_64-linux-gnu
sudo ln -s libhdf5_serial.so.10.1.0 libhdf5.so
sudo ln -s libhdf5_serial_hl.so.10.0.2 libhdf5_hl.so
```
8. Install the required Python packages in the `caffe-rc5/python` directory by using the following commands:
```
cd {caffe-installation-path}/caffe-rc5/python
for req in $(cat requirements.txt); do sudo -H pip install $req
--upgrade; done
```
where `{caffe-installation-path}` is the Caffe deployment path.

9. Open the makefile in the {caffe-installation-path} directory and change the parameter NVCCFLAGS to the following setting:
   ```
   NVCCFLAGS += -D_FORCE_INLINES -ccbin=$(CXX) -Xcompiler -fPIC
   $(COMMON_FLAGS)
   ```
10. In the main Caffe directory caffe-rc5, begin the Caffe build and installation by using the following commands:
    ```
    make all
    make test
    make runtest
    make pycaffe
    make distribute
    ```
11. Add the following line to the ~/.bashrc script:
    ```
    export PYTHONPATH="/usr/lib/python2.7:{caffe-installation-path}/caffe-rc5/python:$PYTHONPATH"
    ```
    where {caffe-installation-path} is the Caffe deployment path.

Troubleshooting the Caffe installation

If an error message displays in the log when you begin the Caffe build and installation, you can take steps to try to resolve the problem.

Symptoms

When you began the Caffe build and installation, the following message displays:

```
1. In file included from ./include/caffe/util/device_alternate.hpp:40:0,
2.                  from ./include/caffe/common.hpp:19,
3.                  from src/caffe/common.cpp:7:
4. ./include/caffe/util/cudnn.hpp: In function 'void caffe::cudnn::createPoolingDesc(cudnnPoolingStruct**,
 caffe::PoolingParameter_PoolMethod, cudnnPoolingMode_t*, int, int, int, int, int, int)':
5. ./include/caffe/util/cudnn.hpp:127:41: error: too few arguments to function 'cudnnStatus_t
 cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t, cudnnPoolingMode_t, cudnnNanPropagation_t, int,
 int, int, int, int, int)'
6.          pad_h, pad_w, stride_h, stride_w));
7.                                            ^
8. ./include/caffe/util/cudnn.hpp:15:28: note: in definition of macro 'CUDNN_CHECK'
9.      cudnnStatus_t status = condition; \
10.                            ^
11. In file included from ./include/caffe/util/cudnn.hpp:5:0,
12.                  from ./include/caffe/util/device_alternate.hpp:40,
13.                  from ./include/caffe/common.hpp:19,
14.                  from src/caffe/common.cpp:7:
15. /usr/local/cuda-7.5//include/cudnn.h:803:27: note: declared here
16.  cudnnStatus_t CUDNNWINAPI cudnnSetPooling2dDescriptor(
17.                            ^
18. make: *** [.build_release/src/caffe/common.o] Error 1
19.
```

Resolving the problem

To fix the error, refer to the following steps:

1. In the /include/caffe/util/cudnn.hpp directory, replace the cudnn.hpp file with the newest cudnn.hpp file that is in the Caffe repository on GitHub.
2. In the /src/caffe/layers folder, replace all of the cudnn files that are in the /src/caffe/layers folder with the newest cudnn files that are in the Caffe repository on GitHub.

Installing Open CV

Before using an edge, you must install the Open Source Computer Vision (OpenCV) library.

Procedure

1. Get the OpenCV source code from Github:
   ```
   wget https://github.com/opencv/opencv/archive/3.2.0.zip
   ```
2. Unpack the downloaded package and change to the package directory:
   ```
   unzip 3.2.0.zip
   cd opencv-3.2.0
   ```
3. Create a building subdirectory and change to the directory:
   ```
   mkdir build
   cd build
   ```
4. Prepare and generate the building configuration:
   ```
   cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D
   WITH_TBB=ON -D WITH_V4L=ON ..
   ```
5. Compile and build the package:
   ```
   make –j $(nproc)
   ```
6. Install the package:
   ```
   sudo make install
   ```
7. Register the libraries and modules to the system:
   ```
   sudo /bin/bash -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/
   opencv.conf'
   sudo ldconfig
   ```
8. If required, uninstall the old opencv version to avoid version collision:
   ```
   sudo apt-get autoremove libopencv-dev
   ```

Installing object detection libraries

You install an object detection library so that you can run the object detection model on an edge.

About this task

IBM Visual Insights supports the following object detection libraries: YOLO (you only look once), Faster R-CNN, and SSD (Single Shot MultiBox Detector).

Procedure

1. Install the related Python packages by using the following commands:
   ```
   sudo apt-get install python-numpy
   sudo apt-get install python-scipy
   pip install cython
   pip install eaydict
   pip install shutil
   pip install cPickle
   pip install uuid
   pip install multiprocessing
   pip install xml
   ```
2. Install one of the following libraries:

Library	Installation Instructions
YOLO version 2 library	1. Run the following command to get the YOLO source code: **git clone --recursive http://github.com/pjreddie/darknet.git** 2. Edit the Makefile file by enabling the **GPU** and **CUDNN** parameters, and select the correct **GPU ARCH** parameter according to your machine configuration. 3. Run the following command to compile YOLO: **make**
Faster-RCNN Python library	1. Run the following command to get the Faster R-CNN source code: **git clone --recursive http://github.com/rbgirshick/py-faster-rcnn.git** 2. In the `pv-faster-rcnn` directory, in the lib folder, run the following command to Compile Cython: **make** 3. Compile Caffe and pycaffe inside the `py-faster-rcnn` folder. You must compile Caffe by using the Python layer.
SSD library	1. Run the following command to get the SSD source code: **git clone --recursive https://github.com/weiliu89/caffe.git** 2. Edit the Makefile file and change the **CUDA_ARCH**, **BLAS**, and **PYTHON_INCLUDE** parameters according to your machine configuration. 3. Compile the code by using following command: **make -j8** 4. Compile the Python layer by using the following command: **make py** 5. Compile the test by using the following command: **make test -j8**

3. Add the following environment variables at the user level: *YOLO_HOME*, *FRCNN_HOME*, and *SSD_HOME*. The following text is an example of adding environment variables: `YOLO_HOME=/home/user/darknet/, FRCNN_HOME=/home/user/py-faster-rcnn/`.
4. Optional: Manually deploy the object detection library.
   a. Unpackage the object detection package.
   b. Add the `yolo/detectorobj.c` file to the `darknet/examples` folder.
   c. Edit the `darknet/Makefile` file and indicate that `EXECOBJA=detectorobj.o`. The following code is an example of the code in the `Makefile` file:
   ```
   EXECOBJA=detectorobj.o captcha.o lsd.o super.o voxel.o art.o tag.o
   cifar.o go.o rnn.o rnn_vid.o compare.o segmenter.o regressor.o
   classifier.o coco.o dice.o yolo.o detector.o writing.o nightmare.o
   swag.o darknet.o
   ```
   d. In the `Makefile` file, add `$(EXECOBJ)` for the `$(SLIB)` and `$(ALIB)` objects. The following code is an example of the code in the `Makefile` file:
   ```
   $(ALIB): $(EXECOBJ) $(OBJS)
     $(AR) $(ARFLAGS) $@ $^
   $(SLIB): $(EXECOBJ) $(OBJS)
     $(CC) $(CFLAGS) -shared $^ -o $@ $(LDFLAGS)
   ```
   e. Run the following command: **make**
   f. Copy the `iotmyolo.py` file and add it to `YOLO_HOME` directory.

Configuring the image server

You configure the image server on an edge machine so that the machine can store images that are captured by an industrial camera. The image server is monitored by the edge controller. When a new image is added, it is scored and the inspection result is sent to the center application so that an inspector can evaluate the image and the inspection results.

Before you begin

Add the following line to your export file:
`/imageserver "IP address"(rw,sync,no_root_squash,no_all_squash)`. For example, you would add the following line to the file:
`/imageserver 10.173.0.0/29 (rw,sync,no_root_squash,no_all_squash)`.

Procedure

1. Install and start the nfs service by using the following command:
   ```
   #>apt-get install nfs-kernel-server
   ```
2. Export the image server folder with the following commands:
   ```
   #>mkdir /imageserver
   #vi /etc/exports
   #>service nfs-server restart
   ```

Configuring the model store

You must configure the model store on the edge machine. The model store is a repository for executable models that are distributed from the center application.

Before you begin

Add the following line to your export file:
`/modelstore "IP address"(rw,sync,no_root_squash,no_all_squash)`. For example, add the following line to the file:
`/modelstore 10.173.0.0/29 (rw,sync,no_root_squash,no_all_squash)`.

Procedure

1. Export the model store folder by using the following commands:
   ```
   #>mkdir /modelstore
   #vi /etc/exports
   #>service nfs-server restart
   ```
2. Run the following commands:
   ```
   sudo pip install flask
   sudo pip install gevent
   sudo pip install requests
   sudo pip install pyinotify
   sudo pip install opencv-python
   sudo pip install lmdb
   sudo apt-get install dos2unix
   ```

Registering the edge to the center application

After you configure the edge machine, you must register it in the center application. You can create an edge or edit an existing one. Edges are used to run a scoring model.

Procedure

1. In the Model Manager, select **My Data** > **Edges**.
2. Select **Create new edge** and input the edge name.
3. Input the IP address, the SSH user name, and the password of the edge machine. The IP address must be accessible to the center application. The SSH user name and password are used to log in the edge machine to deploy the edge controller. The SSH user must be a root user or have SUDO privileges.
4. In the Physical Hierarchy box, add the cell hierarchy to the profiles. The hierarchy means that the edge covers those cells, and the images that are captured on those cells are sent to the edge for scoring. You can select one of the following combinations: plant, plant and line, or plant, line, and cell. If you specify plant, all of the cells that are under the plant are covered. If you specify plant and line, all of the cells that are under the plant and line are covered. If one profile is not enough, you can add more profiles.
5. Select **Ok** to create the edge. If the IP address, SSH user name, or password is not correct and an error message shows, make sure that the values are correct and that the edge machine can be connected.
6. Select **Create**. The edge controller and score engine deploy to the edge machine, and the edge is added into the registered list.
7. Perform an SSH login to the edge system.
8. Run the following command:
   `ps aux | grep python`
   The result should include the following Python process for the edge controller:
   `python controler.py`.
9. If you do not find the Python process on the edge machine, there is an issue with starting the edge controller. Check the log files *deployment_folder*`/vi_edge-bin_vi/vi_edge/nohup.out`. Try to start edge controller by running the following commands:
   `cd` *deployment_folder*`/vi_edge-bin_vi/vi_edge/`
   `nohup python controler.py &`
10. Run the following command again to look for Python processes for the score engine:
    `ps aux | grep python`
    The result should include the following Python processes:
    `python`
    deployment_folder`/vi_edge-bin_vi/vi_score_engine_restful/model`
    `/run.py 2001 0`
    `python`
    deployment_folder`/vi_edge-bin_vi/vi_score_engine_restful/model`
    `/run.py 2002 0`
    `python`
    deployment_folder`/vi_edge-bin_vi/vi_score_engine_restful/route`
    `/run.py 5005 2001 2002`.
11. If you do not find these Python processes on the edge machine, there is an issue with starting the score engine. Check the log files *deployment_folder*`/vi_edge-bin_vi/vi_score_engine_restful/route/log.txt` and *deployment_folder*`/vi_edge-bin_vi/vi_score_engine_restful/model/log.txt`. Try to start the score engine by running the following command:
    deployment_folder`/vi_edge-bin_vi/vi_score_engine_restful/startEngine.sh`
12. Run the following command again to look for Python processes for the score engine for Faster-RCNN and SSD:
    `ps aux | grep python`

The result should include the following Python processes:

```
python run.py 5060 1 FRCNN
python run.py 5061 1 SSD
```

13. If you do not find these Python processes on the edge machine, there is an issue with starting the score engine. Check for the `frcnn_log.txt` and `ssd_log.txt` log files in *`deployment_folder`*`/vi_edge-bin_vi/vi_obj_detection/RESTAPI/model`. Try to start the score engine by running the following command:

```
cd deployment_folder/vi_edge-bin_vi/vi_obj_detection/RESTAPI/model
nohup python run.py 5060 1 FRCNN > frcnn_log.txt 2>&1 &
nohup python run.py 5061 1 SSD > ssd_log.txt 2>&1 &
```

Chapter 3. Creating and using models

You create models to collect historical images and defect information. The information is sent to the center application and used to train the model. After the model is trained, it is distributed to edges so that it can be inspected.The information is used to train the model. After the model is trained, it must be validated before it is deployed to an edge. Validating the model provides model accuracy information. There can be multiple versions of a model. Models can share defect information but have different image files from different product lines. You can retrain a model to attempt to get a higher model accuracy so that the model version can be replaced with a newer model version. There are two types of models implementations, the classification model and the object detection model. The object detection model does not support retrain.

Structure of compressed image files

Before you add historical images for image groups, you must have files that contain the image files that you need for either the classification model or the object detection model.

Classification model

Add the images into compressed files. One compressed file must contain all of the images that belong to the same image group. You must put all of the images in a flat structure with no subfolders in the compressed file.

Object detection model

The structure for the object detection model must contain two compressed folders in one compressed file. One folder must be named `JPEGImages` and the other must be named `Annotations`. In addition to the two folders, the compressed file must also contain a `labels.txt` file.

Add all of the image files to the `JPEGImages` folder. Add all of the annotations files to the `Annotations` folder. An annotation file must have the same file name as it's image file. The files must be in XML format. The following information is an example of an annotation file:

```
<annotation>
  <folder>JPEGImages</folder>
  <filename>000001.jpg</filename>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>864</width>
    <height>1296</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>defect1</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>474</xmin>
```

```
      <ymin>368</ymin>
      <xmax>540</xmax>
      <ymax>448</ymax>
    </bndbox>
  </object>
  <object>
    <name>defect2</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>303</xmin>
      <ymin>387</ymin>
      <xmax>369</xmax>
      <ymax>452</ymax>
    </bndbox>
  </object>
</annotation>
```

The `labels.txt` file contains the names of all of the defects types that are in the compressed image folder. Each defect must be on a separate line, as shown in the following example:

```
defect1
defect2
defect3
```

Adding historical images for image groups

The model manager can create image groups by using historical images to train the model.

Procedure

1. Select **Data** > **Image groups** > **New Image Group**.
2. Add a unique image group name and description, select the image group type, and select **Next**.

   For the image group type, single characteristics means that all of the images that are in the group have the same type of defect. For the single characteristics group type, each image in the group has one defect. Multiple characteristics means that all of the images that are in the group contain more than one defect. For the multiple characteristics group type, the defect types can be the same or different. For the image group type, not a defect means that the images in the group do not contain defects.

   Note: After the image set is uploaded, the image group type that you select cannot be changed.
3. In the Image sets pane, add images and select **Add Image Group**.
4. In the Image groups pane, open the defect that you added and select **Preview** to view the image files.

Results

In the Image groups pane, you can select an image group and select edit. You can add or delete the image sets, update the image group name or description, or change the image group type.

Note: If the image set is cited in a model instance, the image set cannot be deleted.

Creating model requests

After the defect types are added, the model manager creates a model request. The model request is submitted to send the model to a data scientist. The data scientist trains the model.

About this task

You view the details of a model on the **All Models** tab. The model details include the versions of the model. Different model versions are built by different image sets.

The model name and the product type are unique. You cannot use existing model names and product types when you create a new model. However, the same model name can exist in two separate lists to represent a different version of the same model. After the new model is distributed to the edges, the older version of the model is replaced.

Procedure

1. Select **Create New Model**.
2. On the **General** tab, update the information. The product type is used to map the model. The edge controller selects the model based on the product type information that is on the image.
3. On the **Global Policies** tab, set the retrain policy and the manual inspection policy settings. If you select auto retrain, the retraining is automatically triggered when the conditions are fulfilled. If you select manual retrain you can select the confidence level for the model manager to manually retrain.
4. On the **Add Defects** tab, select the image sets that you want to use to train the model and select **Done**. The model version status is set to Draft.
5. In the All Models pane, view the model. If the model version status is set to Draft, you can select an image group and edit.
6. Select **Request training**. The status of the model version changes from Draft to Submitted.

Trained models

The data scientist trains the model by using the model description and the image sets. The model manager validates the model and either accepts or rejects the trained model. The model manager validates the model by using validation image sets.

It is recommended that the data scientist use the NVIDIA development tool to train the model. The data scientist packages the output of the Caffe model and the parameter files and sends the compressed file to the model manager. The model manager selects the model and attaches the compressed file. After the compressed file is uploaded, the model status is changed to Trained.

Structure of model files

Visual Insights supports the convolutional neural network (CNN) classification and object detection model types.

CNN classification models

The CNN classification model must be a single compressed file and contain the correct directory structure and files.

The compressed model file

The compressed model file must contain the following directories and files.

- `model.config` (file, required)
- `sink.config` (file, required)
- `parameter.config` (file, optional)
- `cnet` (directory). The `cnet` directory must contain the following files:
  - `labels.txt` (file, required)
  - `deploy.prototxt` (file, required)
  - `mean.binaryproto` (file, required)
  - `info.json` (file, required)
  - `snapshot.caffemodel` (file, required)
  - `solver.prototxt` (file, required)
  - `train_val.prototxt` (file, required)

Each file must have the correct structure and keywords. The files are described in the following sections.

model.config

The following text is an example of the contents of the `model.config` file. Keywords are shown in bold text.

```
submodel{
  module {
       type:"ChipROIExtractor"
       ref_file:"parameter.config"
  }
  module {
       type:"ClassificationNet"
       net_name:"cnet"
  }
}
sink{
    type:"SinkFilter"
    ref_file:"sink.config"
}
```

This file must have at least one module that has the **`ClassificationNet`** type in the submodel. The ref_file keyword points to the other configuration files within the CNN classification model compressed file. The net_name keyword refers to the folder name that contains the CNN model. You do not need to change the sink information unless you have a different name for the `sink.config` file.

sink.config

The contents of the `sink.config` file are as follows. You do not need to edit the contents.

```
keyword:"position"
keyword:"probableTypes"
```

labels.txt

The `labels.txt` file contains all the class names with which this classification model is classified. Each class name must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

info.json

The `info.json` file holds all of the metadata information of the CNN model. All of the file names must match the file names that are included in this model. Keywords in the following example file are shown in bold text.

```
{
    "caffe flavor": "BVLC",
    "caffe version": "1.0.0-rc5",
    "creation time": "2017-05-02 16:26:18.957631",
    "dataset_id": "20170502-162536-2027",
    "deploy file": "deploy.prototxt",
    "digits version": "4.1-dev",
    "framework": "caffe",
    "id": "20170502-162618-b701",
    "image dimensions": [
        227,
        227,
        3
    ],
    "image resize mode": "squash",
    "job id": "20170502-162618-b701",
    "labels file": "labels.txt",
    "mean file": "mean.binaryproto",
    "model file": "original.prototxt",
    "name": "C_3_227_227",
    "snapshot file": "snapshot_iter_138.caffemodel",
    "solver file": "solver.prototxt",
    "status": "Aborted",
    "train_val file": "train_val.prototxt",
    "username": "coco"
}
```

Other files

All *filename*`.prototxt` files are model definition files that are required when you train a CNN model.

The `snapshot.caffemodel` and `mean.binaryproto` files are output files that are created after the model training is complete.

Object detection model

The object detection model must be a single compressed file and contain the correct directory structure and files. The following object detection algorithms are supported: You Only Look Once (YOLO) V2, faster region-based convolutional neural network (Faster R-CNN), and single shot multibox detector (SSD).

YOLO V2:

The YOLO V2 object detection model must be a single compressed file and contain the correct directory structure and files.

Compressed model file

The compressed model file must contain the following files:

- `labels.txt`
- `model.config`

- yolo_final.weights
- Yolo.cfg

Each file must have the correct structure and keywords. The files are described in the following sections.

labels.txt

The labels.txt file contains the names of all of the objects that this object detection model detects. Each object must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

model.config

Keywords in the following example are shown in bold text:

```
{
    "modelType": "YOLO",
    "modelCfg": "Yolo.cfg",
    "model": " yolo_final.weights",
    "parameters": {
        "iteration": "40000",
        "batchSize": 16,
        "learningRate": 0.001,
        "subBatchSize": 2,
        "steps": "100, 15000, 25000,35000",
        "scales": "-1,10, 0.1, 0.1"
    }
}
```

The value of modelType is always YOLO. The value of modelCfg is the name of the deep learning network definition file. The value of model is the name of the actual model file. The values in parameters are hyper-parameters of the YOLO V2 model.

yolo_final.weights

This file contains the output after the YOLO model is trained. The file name must match the definition of **model** in model.config.

Yolo.cfg

This model definition file includes network definitions, hyper-parameters, and anchor settings. A template of this file can be found in the darknet/cfg/ YOLO installation directory. This file must match the weights file.

Faster R-CNN:

The Faster R-CNN object detection model must be a single compressed file and contain the correct directory structure and files.

Compressed model file

The compressed model file must contain all of the following files:

- labels.txt

- faster_rcnn_final.caffemodel
- model.config
- stage1_fast_rcnn_solver30k40k.pt
- stage1_fast_rcnn_train.pt
- stage1_rpn_solver60k80k.pt
- stage1_rpn_train.pt
- stage2_fast_rcnn_solver30k40k.pt
- stage2_fast_rcnn_train.pt
- stage2_rpn_solver60k80k.pt
- stage2_rpn_train.pt
- faster_rcnn_test.pt

Each file must have the correct structure and keywords. The files are described in the following sections.

labels.txt

The `labels.txt` file contains the names of all of the objects that this object detection model detects. Each object must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

faster_rcnn_final.caffemodel

This file contains the output after the Faster R-CNN model is trained. The file name must match the definition of **`model`** in `model.config`.

model.config

Keywords in the following example are shown in bold text:

```
{
    "modelType": "FRCNN",
    "model": " faster_rcnn_final.caffemodel",
    "solvers": "stage1_rpn_solver60k80k.pt,stage1_fast_rcnn_solver30k40k.pt,
    stage2_rpn_solver60k80k.pt,stage2_fast_rcnn_solver30k40k.pt",
    "net_file": "stage1_rpn_train.pt,stage1_fast_rcnn_train.pt,
    stage2_rpn_train.pt,stage2_fast_rcnn_train.pt",
    "deploy_net": "faster_rcnn_test.pt",
    "parameters": {
        "iteration": "40000,80000,40000,80000",
        "learningRate": 0.001,
        "stepsize": "10000",
        "gamma": "0.1"
    }
}
```

The value of modelType is always FRCNN. The value of model is the name of the model file. The value of solvers is the list of solver files that are used during model training. The value of net_file is the list of network definition files. The value of deploy_net is the name of the scoring network definition. The values in parameters are all hyper-parameters of the Faster R-CNN model.

***.pt files**

The files with the .pt extension are model definition files. The pascol_voc model that is provided by Faster R-CNN is supported. The template file can be found in the `models/pascal_voc/`*`netname`*`/faster_rcnn_alt_opt/` Faster R-CNN installation directory, where *`netname`* is `ZF` or `VGG16`.

SSD:

The SSD (Single Shot MultiBox Detector) object detection model must be a single compressed file and contain the correct directory structure and files.

Compressed model file

The compressed model file must contain all of the following files:

- `labels.txt`
- `solver.prototxt`
- `deploy.prototxt`
- `model.config`
- `SSD.caffemodel`

Each file must have the correct structure and keywords. The files are described in the following sections.

labels.txt

The `labels.txt` file contains the names of all of the objects that this object detection model detects. Each object must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

solver.prototxt

This file contains all hyper-parameters for the SSD model.

deploy.prototxt

This file contains the network definition of the trained model.

model.config

Keywords in the following example are shown in bold text:

```
{
    "modelType": "SSD",
    "seldef_parameters": {
        "TestRatio": 0.3,
        "Train_batch_size": 8,
        "Validation_batch_size": 8
    }
}
```

The value of modelType is always SSD. The values of seldef_parameters are parameters that are used for training and validation.

SSD.caffemodel

This file contains the output after the SSD model is trained.

Validated models

The model manager validates the model version and either accepts or rejects the trained model. The model manager validates the model by using validation image sets. After a model is validated, it can be used and deployed.

You can validate a model version that has a status of Trained. A trained or retrained model version can trigger the validation process. When you validate a report, you must manage the image sets. Every image group must have at least one validation image set to validate the model. You must use different image sets and training image sets to validate the model version. To begin the validation process, select **Validate**.

After the validation process is finished, a report is generated that shows the model accuracy. You can create the following types of reports:

Classification model report
In the classification model report, one image has one defect at most. The confusion matrix is used to generate the report where each column represents one real image group type in the validation data sets. Each row represents the predicted image group type. The last row in the chart represents the aggregate results.

Object detection model report
In the object detection model report, one image has multiple defects. In this report, the mean average precision and the recall are calculated to indicate the model accuracy.

To view the report, select **Show Report**. From the Validation Report window, you can revalidate, reject, or accept and deploy the report.

If the first model version is rejected, you can attach a new model file. If a model version that is not the first version is rejected, the model manager can retrain a new model version.

Distributing trained models to edges

After the model manager accepts the trained model version, the model is distributed to edges so that it can be inspected.

Procedure

1. Select a model version with a status of Accepted. Select **Deploy**.
2. Select the plant, line, or cell filter to search for edges that the model will deploy on.
3. Select the edges from the Candidate List and move them to the Selected List.
4. Select **Deploy**.

Retraining models

When you retrain a model, a new model version is created. When you create a model request, you define the retrain policy. The retrain policy is the condition that triggers auto-retrain. If there is concern about the current model accuracy, the model manager can manually select the image files to trigger the retrain process.

Procedure

1. Select a model version that has a status of Deployed and select **Retrain**.
2. Manage the image sets. Every image group must have at least one retrain image set to retrain the model.

   Note: You must use different image sets to retrain the model version.
3. Select **Retrain** to begin the retrain process.

What to do next

The model manager validates if the retrained model is accepted. The model manager can deploy the new version, and the old model version is no longer deployed.

Chapter 4. Checking inspection results

After the inspection results are sent to the center application, the inspector and the inspector supervisor can go to the Defect Check tab to view and filter the inspection results and make any necessary changes.

Images

The inspector and the inspector supervisor can view images to see if they are classified as existing defects or not, and to find out if someone else checked the images. Viewing the images determines what the inspector or supervisor must do when they check the defects.

The inspector views unchecked and checked images. Unchecked images mean that the image was only scored by the model and was not checked by an inspector. Checked images mean that the image was scored by the model and was already checked by an inspector.

The inspector views unconfirmed and confirmed images. Unconfirmed images mean that the image was only scored by the model and was not confirmed by an inspector. Confirmed images mean that the image was scored by the model and was confirmed by an inspector.

The inspector supervisor can view objects and unknown objects. Unknown objects mean that the image was marked by an inspector as an unknown defect because the inspector did not classify the image as an existing defect. These images are highlighted on the inspector supervisor's list.

Filtering defects

The inspector and the inspector supervisor can apply filters to the cell overview and the defect list.

Procedure

1. On the All Workstations window, select a workstation to view the list of unconfirmed, confirmed, and unknown objects.
2. Select the filter icon.
3. Input a value for a condition to set the filter and then select the add icon. The filter applies to the list immediately.

Checking defects

The inspector and inspector supervisor review the inspection results and make any necessary changes.

About this task

When you select an image, the defect candidate and corresponding confidence display. The first defect is selected by default. The inspector can select unknown for a defect type if the defect type is unknown.

Procedure

1. Select an image to view the image details and inspection results.
2. Select **Edit Zoom** to zoom in and out of the image or drag the image to locate a position.
3. Select **Set Zoom** to change back to edit mode. You can add, resize, move, and view the details of a defect box.
4. Select the defect box that you want to confirm and view the details of the defect type and confidence level. You can change the defect type or delete the position.
5. Select **Confirm**.
6. If the image does not belong to any existing defects, the inspector supervisor can create a new image group. The new defect is added into the candidate list for images that are under the same model.

Chapter 5. KPI dashboard

An inspector supervisor uses the KPI dashboard to help manage all of the inspectors. The inspector supervisor also uses the KPI dashboard to check the image level defect rate and the location level defect rate. These metrics can help provide information so that you can ask the IT team to retrain the model or adjust the manufacturing procedure.

The KPI dashboard is on the KPI tab. You can select all workstations or a specific workstation. This selection impacts the scope that you are working on. You can also switch between real-time and historical views. In the real-time view, KPI data is refreshed every 5 seconds. The KPI values include defect per unit and defect rate. The defect per unit is calculated as a specific defect number divided by the total image number. The defect per unit value represents the occurrence rate of a defect type. The defect rate is calculated as the number of images with one or more defects divided by the total image number. The defect rate represents the product defect rate. Each line in the chart represents the KPI value in the current 5-second interval. The historical view shows historical KPI data. You can edit the start date and end date to determine the time range. KPIs in the historical view include defect per unit and defect rate.

The historical view has three granularities: hourly, daily, and monthly. In the hourly chart, each point represents 1 hour. For example, a KPI value that is 24 points represents 24 hours. In the daily chart, each point represents one day. For example, a KPI value that is 30 points represents 30 days. In the monthly chart, each point represents one month. For example, a KPI value that is 12 points represents 12 months. There are relationships between the selected time range and granularity. If there are too many points in a granularity for a selected time range, then that granularity is disabled until you shorten the time range. If you want to refresh the chart, you can change time range or click **Refresh**. The historical KPI data is calculated periodically by the server, so there is some time delay based on the configuration. The default period to calculate historical KPI is 1 hour.

By default, only the top five defects types are displayed in the Defect Per Unit chart. If you want to check other defect types, you can select one or more defect types under the chart and then click **Show trend**. A new KPI chart displays the selected defect types. If the KPI value of a defect type is 0, it cannot be selected to show a trend.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample

programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user’s name, user name, password, or other personally identifiable information for purposes of session management, authentication, single sign-on configuration or other usage tracking or functional purposes. These cookies can be disabled, but disabling them will also likely eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies” and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Printed in USA