EAN-Classifier Development

2020-09-25

Exports: Export Summary Sheet

EULA: End User License Agreement

Web: sightlineapplications.com

Sales: sales@sightlineapplications.com

Support: support@sightlineapplications.com

Phone: +1 (541) 716-5137

CAUTION: Alerts to a potential hazard that may result in personal injury, or an unsafe practice that causes damage to the equipment if not avoided

IMPORTANT: Identifies crucial information that is important to setup and configuration procedures.

Used to emphasize points or reminds the user of something. Supplementary information that aids in the use or understanding of the equipment or subject that is not critical to system use.

1 Overview

This document describes the process of training a classifier using Caffe deep learning framework software and a set of images to produce a classifier contained within a binary file. Setup time normally takes a few hours but is dependent on the size of training sets, number of iterations, size of intervals, and the number of layers used during the training process.

ⓘ **IMPORTANT**: Read this entire document before following the steps required to build and train a classifier to become familiar with the process.

1.1 Additional Support Documentation

Additional Engineering Application Notes (EANs) can be found on the Documentation page of the SightLine Applications website.

The Panel Plus User Guide provides a complete overview of settings and dialog windows located in the Help menu of the Panel Plus application.

The Interface Command and Control (IDD) describes the native communications protocol used by the SightLine Applications product line. The IDD is also available as a PDF download on the Documentation page under Software Support Documentation

1.2 SightLine Software Requirements

Building a custom classifier requires a 4000-OEM, 3000-OEM, or SLA-Library running version 3.2.xx software or later. Caffe software is based on working in a development environment using a PC running Windows 10.

1.3 Application Bit Requirements

The functions described in this EAN require Application Bits (app bits) purchased from SightLine. App bits are enabled with a license file provided by SightLine at initial unit purchase or during a license upgrade process. License files use a hardware ID that is applicable to a specific hardware serial number. For questions and upgrade support contact Sales.

Table 1: Application Bits Requirement Table

Function	Initial Software Release	Required Application Bit(s) v7 License
Classifier	3.2.x	(TBD)

2 Sample Application Installation

Download the *ARM Processor Code Examples* package from the example code page on the SightLine website. Launch the installer and follow the installation prompts. After running the installer navigate to *C:\SightLine Applications\SLA-Examples-ARM-<<version number>>\Classifier* and *extract caffe.zip* and *TrainingImages32.zip*. Extract the files to default locations as shown below.

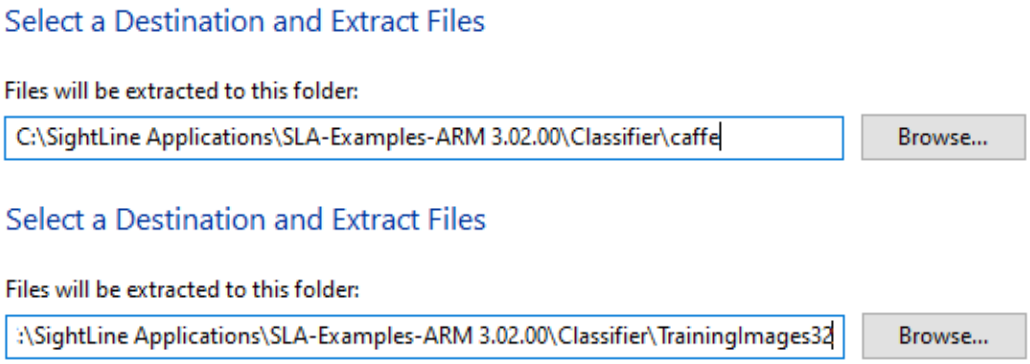


Figure 1: Extract Example Training Images and Caffe Tools

The files in TrainingImages64.zip are only needed when training a network with a different input image size. See Appendix B.

3 Third Party Software

3.1 Python

1. Download Python 2.7 from the Anaconda website.

Python Version 2.7 is required for this training example. Do not use any other versions of Python software.

2. Install the software to *C:\ProgramData\Anaconda2*.

IMPORTANT: During the installation *All Users* defaults to the destination folder of *C:\ProgramData\Anaconda2*. The remainder of this document assumes that the Python software is installed to this location.

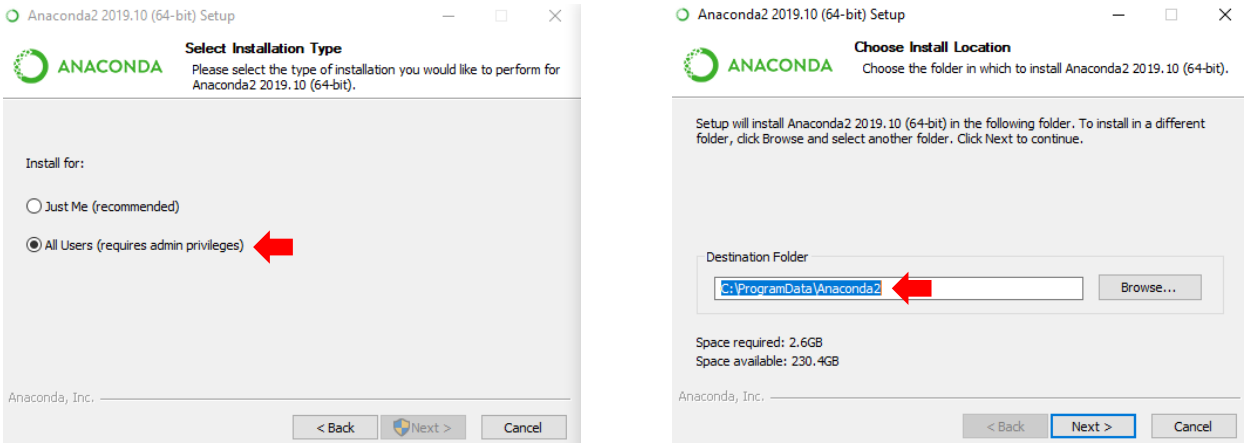


Figure 2: Python Installation Default Location

3. Locate the Windows Environment Variables on the PC. In the *System Variables* section add the following two *Path* directories:
   - *C:\ProgramData\Anaconda2*
   - *C:\ProgramData\Anaconda2\Library\bin*

ⓘ **IMPORTANT:** Make sure these are added to *System Variables* and not *User Variables*.

4. Open the Anaconda Prompt as administrator and run:

```
conda install protobuf
```

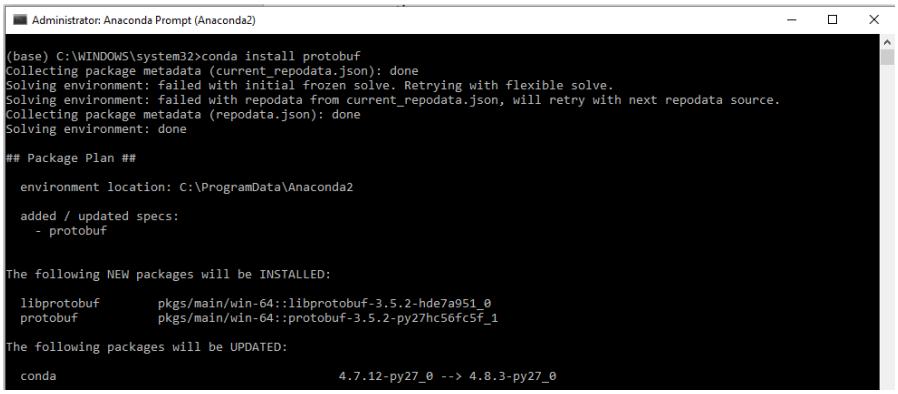


Figure 3: Install Protobuf

3.2 Caffe

All the Caffe tools to create the classifier can be found in *C:\SightLine Applications\SLA-Examples-ARM-<<version number>>\Classifier\caffe*.

No additional steps are necessary to build and install Caffe. For additional information see the Caffe website and the Caffe tutorial.

4 Custom Classifier Development

The following sections describe how to build a custom classifier based on sample data in the *C:\SightLine Applications\SLA-Examples-ARM-<<version number>>\Classifier\TrainingImages32* directory.

This process assumes that the default Caffe installation provided in the installer is being used and has been built with CPU support only. For details on training with GPU support see Appendix A.

Once the custom classifier is complete use a test pattern on the hardware to test it. The diagram in Figure 4 outlines the steps in the sections that follow.

If you would like to run a sample classifier that is pre-trained skip to the Testing the Classifier section and use the sample_classification_params.cls file located in C:\SightLine Applications\SLA-Examples-ARM-<<version number>>\Classifier\.

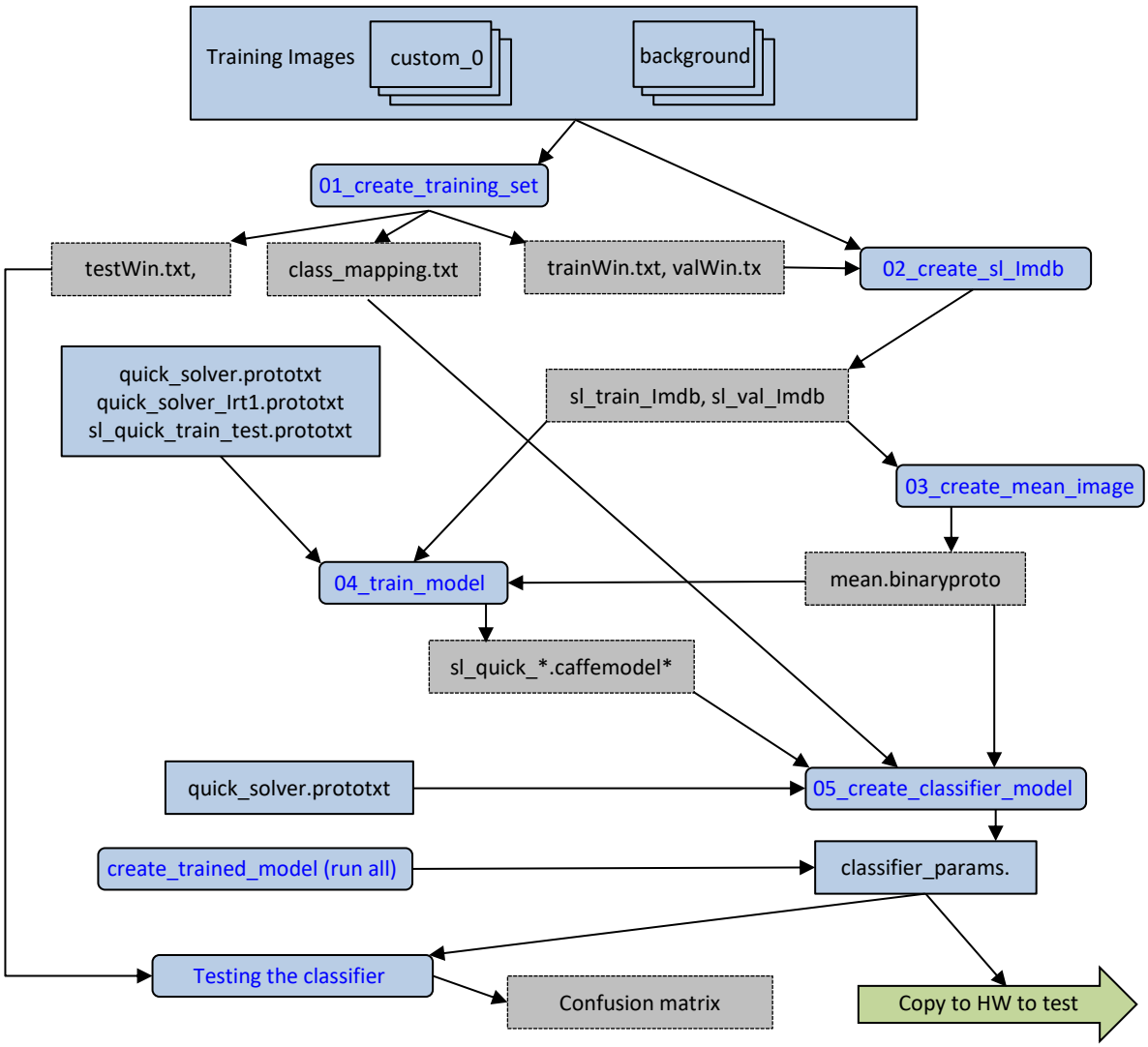


Figure 4: Training Steps

4.1 Generate Training Images

The first step in building a classifier is to generate a large set of training images. In this example, the example dataset contains the following two classes of data: *background* and *custom_0*. In this example *custom_0* represents images of a synthetic car.

If you are using the example dataset skip to the Create Training Set section.

To start training the data:

1. Create a set of *.png* greyscale images. See Appendix B for additional details on training different size patches.
2. Create folders based on the supported classifier types, e.g., *background, airplane, boat*.
3. Organize and place the images in their respective folders.

Additional changes to the create_training_set.py script may be needed depending on the naming scheme for class types, the number classes, and additional factors.

It is helpful to have a class that includes negative examples. For this example, negative examples are any images that are not a synthetic car and are represented in the background directory.

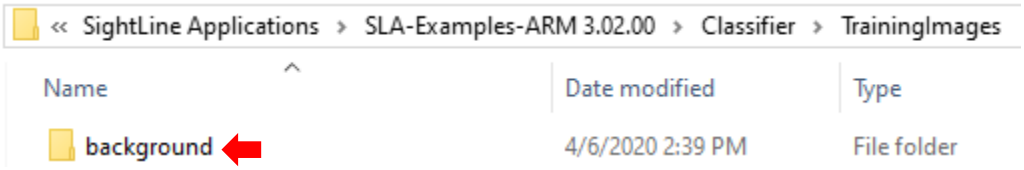


Figure 5: Example Training Images Directory Structure

4.2 Create Training Set

The *create_training_set.py* file is a python script used to create the training, validation, and testing data set from the training images described in the Generate Training Images section. It is also used to create the class mapping file as described in Create Custom Class Mapping section.

The output of this script is a series of files that contains a list of filenames. The first 80% of the data is put in the *trainWin.txt* file. The next 15% is put in the *valWin.txt* file. The final 5% of data is put in the *testWin.txt* file.

```
preDefinedTypes = [
    ("background",      0, "Bkgd"),
    ("rotaryWing",      1, "Drone Rotary"),
    ("fixedWing",       2, "Drone FW"),
    ("vehicles",        3, "Vehicle"),
    ("people",          4, "Person"),
    ("boats",           5, "Boat"),
    ("aircrafts",       6, "Aircraft"),
    ("custom_0",      100, "TestCar"),
    ("custom_1",      101, "Custom-002d"),
    ("custom_2",      102, "Custom-003d"),
```

Figure 6: Modify Class Names - PiP Labels

During the training process only the *trainWin.txt* and *valWin.txt* files are used. The *testWin.txt* file can be used for testing the classifier.

The folder names in the preDefinedTypes array *must exactly match the folder names of your training data.*

Even though the example in Figure 6 only shows three custom types, up to 100 custom types (number 100-199) can be added.

4.2.1 Modifying Default Definitions

The training process assumes that the directories provided are being used with the example and that the output will be placed in the *.output* directory. To configure these settings, edit the definitions near the bottom of the *created_trained_model.py* script.

Additional information on when to edit definitions can be found in the applicable sections of this EAN.

```
# Global Definitions
# - Edit these to match with your environment.
# ------------------------------------------------------------------------------
CAFFE_ROOT = R"./caffe"                          # Assuming /caffe directory exists.
CAFFE_GPU = R""  # R"./Release_CUDA_10_2"       # If you have NVidia CUDA, set CAFFE_GPU to GPU enabled Caffe.exe directory.


# Default definitions:
outputPath = R".output"                          # output directory.
imageDirPath = R"TrainingImages32"               # training image location.
imageSize = "32"                                 # height and width of the images.
snapshot = "sl_quick_iter_8000.solverstate"      # sl_quick_iter_4000.solverstate to shorten training time.
modelH5 = "sl_quick_iter_11000.caffemodel.h5"    # sl_quick_iter_7000.caffemodel.h5 to shorten training time.
```

Figure 7: Global Definitions

4.2.2 Create Custom Class Mapping

When the *create_training_set.py* script is running it is also responsible for generating a *class_mapping.txt* file. This file is used to describe the mapping of the classifier output type to the final SightLine reported types.

The text in the *preDefinedTypes* array can also be edited to add a better descriptive name for the classified object to the *class_mapping.txt* file. This name will be used when viewing the class label in the PiP window (see the Picture-in-Picture (PiP) Label section). In the example in Figure 6 *Custom-0001d* was changed to *TestCar*.

Names must be limited to 15 characters.

4.2.3 Running the create_training_set Script

Once modifications are complete run the script from a command prompt or power shell window.

`01_create_training_set.bat`

```
PS C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier> .\01_create_training_set.bat

C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier>python create_trained_model.py  create_training_set
Executing create_training_set...
SLA script run successfully.
```

Figure 8: Example - 01_create_training_set.bat

Verify that this step was successful by checking the trainWin.txt file as shown in Figure 9. This can be found in the .output directory.

```
background\img_tp13_4114_222_0.png 0
background\img_tp13_6329_319_0.png 0
custom_0\img_tp13_6699_335_0.png 1
background\img_tp13_7544_372_0.png 0
custom_0\img_tp13_8640_486_0.png 1
custom_0\img_tp13_5079_264_1.png 1
```

Figure 9: Example trainWin.txt Data

4.3 Generating LMDB Files

The next step in training is to create Lighting Mapped Database (LMDB) files. Use the script *02_create_sl_lmdb.bat* to generate the LMDB files. In this example no modifications are required to this script.

This script can be run directly as follows:

```
02_create_sl_lmdb.bat
```

The output of the script will describe what has been processed. An example of a portion of the output are shown in Figure 10.

```
PS C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier> .\02_create_sl_lmdb.bat

C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier>python create_trained_model.py   create_sl_lmdb
Executing create_sl_lmdb...
Creating train lmdb...
Directory doesn't exist: .output/sl_train_lmdb
.output/: set GLOG_logtostderr=1 & "C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier\caffe\convert_imageset" --resize_height=32
--resize_width=32 --gray=true "C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier\TrainingImages32/" trainWin.txt sl_train_lmdb
I0618 13:10:33.217770 15128 convert_imageset.cpp:89] A total of 800 images.
I0618 13:10:33.225740 15128 db_lmdb.cpp:40] Opened lmdb sl_train_lmdb
I0618 13:10:33.655113 15128 convert_imageset.cpp:153] Processed 800 files.
Creating val lmdb...
Directory doesn't exist: .output/sl_val_lmdb
.output/: set GLOG_logtostderr=1 & "C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier\caffe\convert_imageset" --resize_height=32
--resize_width=32 --gray=true "C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier\TrainingImages32/" valWin.txt sl_val_lmdb
I0618 13:10:33.746657 22824 convert_imageset.cpp:89] A total of 150 images.
I0618 13:10:33.749650 22824 db_lmdb.cpp:40] Opened lmdb sl_val_lmdb
I0618 13:10:34.064605 22824 convert_imageset.cpp:153] Processed 150 files.
```

Figure 10: Example Output of Create LMDB Process

Once the script is finished running, the folders shown in Figure 11 are generated. Each folder will contain the .mdb files that contain a key-value database of image buffers for later use.

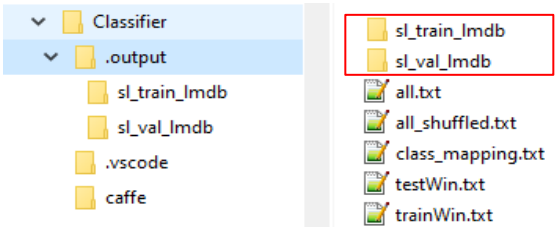


Figure 11: Train and Validation Database Folders

Additional information on the Caffe tool convert_imageset can be found by looking at the Caffe Source.

When running *convert_imageset* in the script the `-shuffle` parameter has been removed. Removing this parameter allows the script to run multiple times with the same result. You can add this parameter back in if you want to run this process over and over with multiple initial datasets and choose the best result.

Since everything is treated as grayscale the `--gray=true` parameter is used in this example.

```
print("Creating train lmdb...")
sl_util.delete_directory(outputPath + 'sl_train_lmdb')
cmd = 'set GLOG_logtostderr=1 & "' + CAFFE_ROOT + 'convert_imageset" --resize_height=' + str(imageSize) + " --resize_width=" + str(imageSize)
cmd += " --gray=true " + imageDirPath + " trainWin.txt sl_train_lmdb"
ret = sl_util.execute_dos_command(cmd, outputPath)
```

Figure 12: Example Creating LMDB Files From create_trained_model.py

4.4 Generating Mean Image Data

The next step in building the trained model is to generate a mean image data file. To create this data run the following file:

```
03_create_mean_image.bat
```

The output of this process is the mean.binaryproto file that contains all the mean image data. Customizing this file is not necessary.

4.5 Training the Classifier

This step can take several hours or several days if CPU is used depending on the complexity of the network, the number of images, and other training parameters. For this example, expect two to four hours if the number of training iterations are not reduced, or one to two hours if they are reduced.

Before running 04_train_model.bat it is important to make the required modifications.

4.5.1 Changing the Number of Outputs

Two files define the network structure used for both training and actual deployment of the network: *sl_quick.prototxt* and *sl_quick_train_test.prototxt*.

In this example both files must be updated to set the number of outputs to *2* because there are only two class types (background and custom_0). Modifications are made to the *ip2* layer on both files as shown in Figure 13 and Figure 14.

```
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 2
```

Figure 13: File Update - sl_quick.prototxt

```
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "drop2"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }

  inner_product_param {
    num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.1
    }
    bias_filler {
      type: "constant"
```

Figure 14: File Update - sl_quick_train_test.prototxt

4.5.2 Reducing Training Time (optional)

Skip this section if GPU is being used for training.

The default training time can be long. To help reduce training time, modify the following three files that define the parameters used in the training process:

- *quick_solver.prototxt* - Change `max_iter` from `8000` to `4000` (Figure 15).
- *quick_solver_lr1.txt* - Change `max_iter` from `11000` to `7000` (Figure 16).
- *create_trained_model.py* - Change default definitions for snapshot and modelH5 (Figure 17).

```
# The learning rate policy
lr_policy: "fixed"
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 4000
# snapshot intermediate results
snapshot: 500
snapshot_prefix: "sl_quick"
```

Figure 15: File Modification - quick_solver.prototxt

```
# The learning rate policy
lr_policy: "fixed"
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 7000
# snapshot intermediate results
snapshot: 500
snapshot_format: HDF5
snapshot_prefix: "sl_quick"
```

Figure 16: File Modification - quick_solver_lr1.prototxt

```
# Default definitions:
outputPath = R".output"
imageDirPath = R"TrainingImages32"
imageSize = "32"
snapshot = "sl_quick_iter_4000.solverstate"
modelH5 = "sl_quick_iter_7000.caffemodel.h5"
```

Change 8000 to 4000
Change 11000 to 7000

Figure 17: File Modification - create_trained_model.py

4.5.3 Run the Training

After completing the changes to the network and training files, run the training by executing the *04_train_model.bat* file from either a command prompt or power shell window.

```
04_train_model.bat
```

4.6 Generating the Binary File

Once the training is complete convert the data into a binary classifier file that the SightLine system can process.

Read this entire section before starting this process to ensure the necessary changes are made to the commands based on any modifications to the training process in the previous sections.

To convert the data run the following script:

```
05_create_classifier_model.bat
```

If the classifier was modified for shorter training as outlined in the Reducing Training Time section make sure *modelH5* in *create_trained_model.py* is set to sl_quick_iter_7000.caffemodel.h5 instead of _11000 (Figure 17).

When the script is complete it will generate a *classifier_params.cls* file. Since this filename is the same as the default classifier filename it is important to rename the file to avoid confusion. For this example, the file has been renamed to *custom.cls*.

4.7 Testing the Classifier with Caffe

After training is complete use the *sl_test_model.py* script to test the model on a series of images or on individual images as shown in Figure 18.

This testing uses Caffe software. SightLine hardware uses an optimized version of Caffe, so the results are not guaranteed to be the same.

```
python sl_test_model.py .output/testWin.txt --image_root=TrainingImages32
```

The output of this step is a confusion matrix and an accuracy score. The confusion matrix shows the predicted classes compared to the actual classes. The example in Figure 18 shows one instance where the predicted class was *Bkgd* but the actual class is *Custom_001d*.

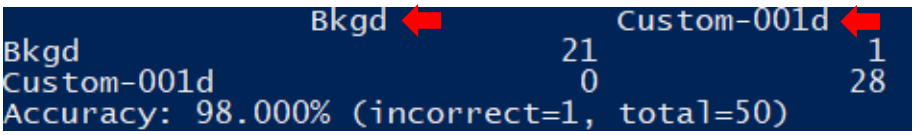


Figure 18: Caffe Classifier Test Example

This process can also be used to test individual images:

```
python sl_test_model.py TrainingImages32/custom_0/img_tp13_5604_288_1.png
```

This example will show the predicted output class of the image.

4.8 One Step Training

SightLine has created a *create_trained_model.py* script that runs all the steps in the Custom Classifier Development section. Before using this script, it is important to become familiar with all the steps in this section to make sure any additional changes that need to be made can be completed at this time.

The following is an example of using this script to complete the entire process:

```
python create_trained_model.py
```

5 Testing the Classifier

The following sections describe the steps required to test the custom classifier created in Custom Classifier Development section based on the SightLine example training images. These same steps will be useful in understanding how to upload and test any custom classifier.

This section assumes connection to the 4000-OEM with Panel Plus running 3.2.xx software. The same steps are valid for the 3000-OEM.

5.1 Copy the Binary File to the 4000-OEM

To copy the file, use the *SLA-4000 Upgrade Utility*. See the EAN-Firmware Upgrade Utility for more information on using this application.

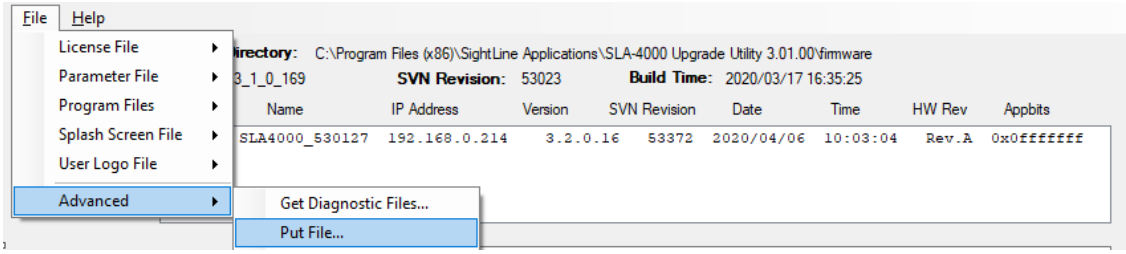


Figure 19: SLA-4000 Upgrade Utility

5.2 Testing - Multi Car Test Pattern

To use the *Multi Car* test pattern in Panel Plus to test the classifier:

1. From the main menu in Panel Plus go to *Configure » Acquisition Settings*.
2. Set up one of the cameras to be a multi car test pattern.
3. Click *Apply*.
4. Main menu » *Parameters* » *Save to Board*.
5. Main menu » *Reset* » *Board*.
6. Wait for the system to boot, and then reconnect to the board.

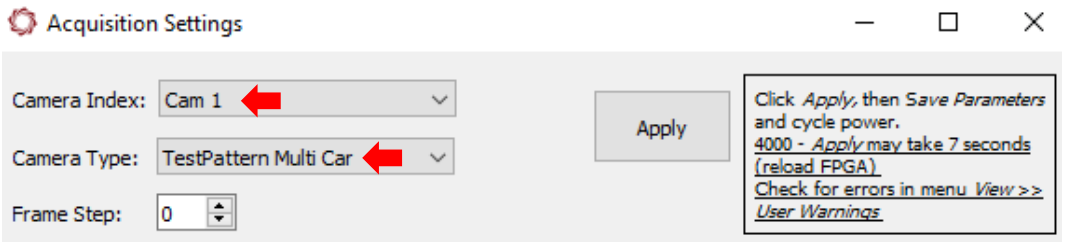


Figure 20: Multi Car Test Pattern

5.3 Verify Telemetry

The following sections discuss the different methods to look at classifier related telemetry. These commands are all based on the commanded camera.

Ensure the commanded camera in Panel Plus matches the camera setup as the Multi Car test pattern. In the example in Figure 20 it is Cam 1.

5.3.1 Enable Vehicle Detection

1. From the *Tracking* tab in Panel Plus click on the *Detect* tab.
2. From the *Mode* dropdown menu select *Vehicle* to enable the vehicle detection mode.

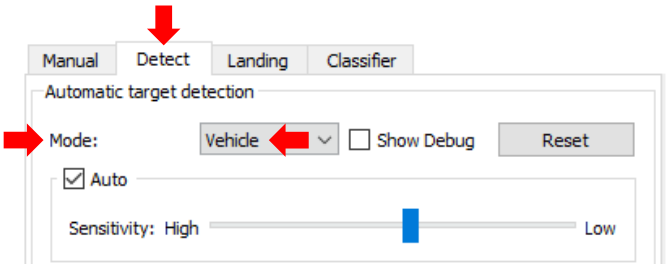


Figure 21: Enable Vehicle Detection

5.3.2 Enable Classification of Vehicle Detections

1. From the *Tracking* tab in Panel Plus click on the *Classifier* tab.
2. In the *Vehicle, Drone, Person* dropdown menu select *All Tracks - Including Vehicle/Drone Detect Tracks* to enable this classification.

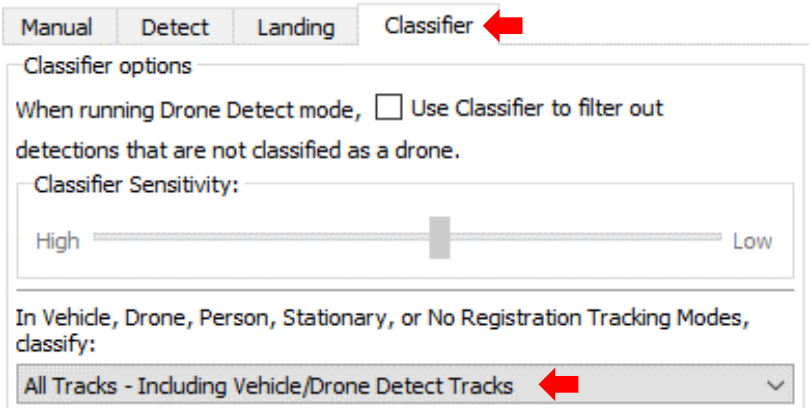


Figure 22: Enable Classifier

5.3.3 Update the Custom Classifier

To use a different classifier on the OEM, update the binary classifier file name.

1. From the *Tracking* tab in Panel Plus click on the *Classifier* tab.
2. Update the name in the *Custom Classifier File Name* field.
3. Click *Update File*. The new classifier parameters will update prior to the next call to the classifier.

When updating files look for any user warnings for potential issues.

Figure 23: Classifier Settings

5.3.4 Telemetry Dialog

1. Make sure that the classifier is enabled as shown in Figure 24.
2. To show the telemetry dialog, click the *Show Telemetry* button on the bottom of the *Tracking* tab or go to the Panel Plus main menu » *View* » *Telemetry*.
3. In the Telemetry dialog window select the *Send Telemetry 2Me* check box and the *Extended* check box.

When a track is created telemetry is shown in their respective sections. Object type (ClassID), and confidence in this type (ClassConf) are shown in the Extended section.

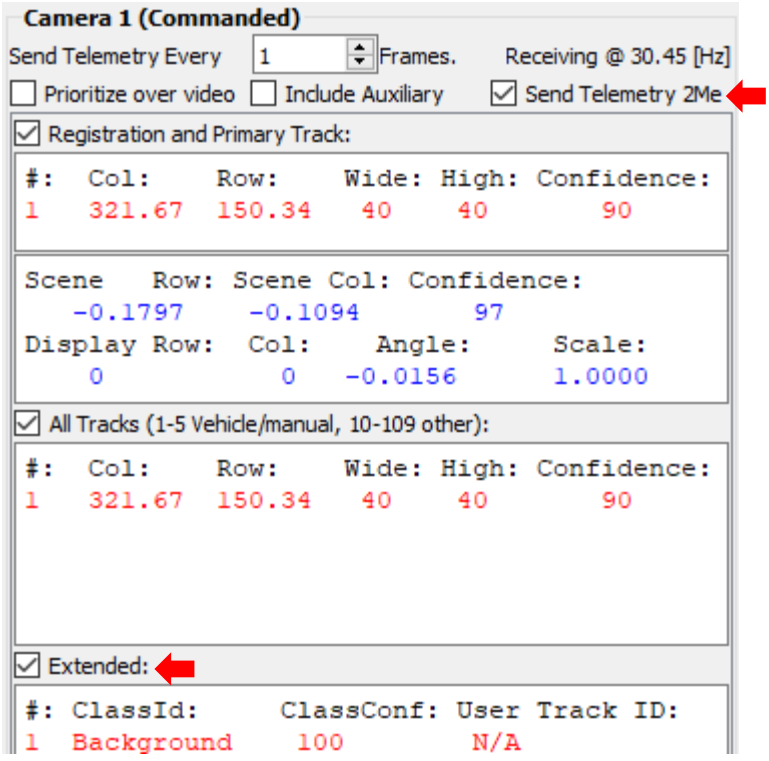


Figure 24: Telemetry Dialog Window

If the ClassID shows that it unclassified make sure that the classifier is enabled.

5.3.5 Picture-in-Picture (PiP) Label

To view the classification of the tracks in a detection mode go to the *Multi Camera* tab. Verify the following:

- ✓ The *Video Source* is set to *Multi*
- ✓ Select Zoom on All Tracks
- ✓ The *PiP Label* is set to *Classification*
- ✓ The *Main index* and *PiP index* are both set to the active camera.

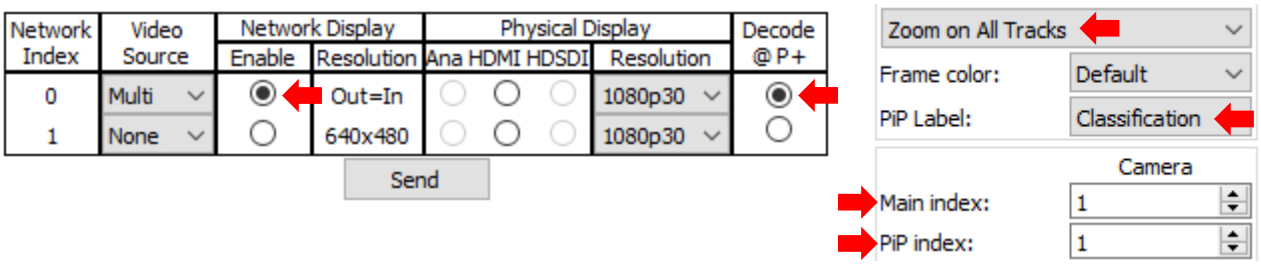


Figure 25: Multi Camera Setup

The class names labels created in Generating Training Image Lists will be used as the PiP labels shown in Figure 26.

Figure 26: PiPs Display - Custom Classifier Labels

6 Troubleshooting

Question	Answer
Can more images be added to the training set?	Adding more images for different classes can improve the performance of the classifier. It is important to consider what types of images are being used. A variety of images is necessary, otherwise overfitting can occur, and classifier performance will decline. It is also important to consider whether a balanced dataset is necessary. This requires a deeper understanding of the issues involved and cannot be answered generically for all use cases.
Can color be used in the training?	Color is not supported at this time but may be added in future updates.
Can a system with 1000 classes be trained?	SightLine supports up to 100 custom classes and 100 SLA predefined classes. For more information about specific use cases, contact Sales for recommendations. It is also important to understand that the SightLine network deployed for drone detection is streamlined to run at frame rate on an embedded device. A much larger network may be required for other problems and may not be able to run fast enough on SightLine hardware.
Is GPU training supported?	Yes. GPU training is supported for some NVIDIA GPUs. See Appendix A for more information. When training with a GPU that is not supported, Caffe can be built directly and used in place of the SightLine provided version.
In Drone mode, classification can be used to filter out false positives. Is this also possible when developing a custom classifier?	Currently SightLine software does not specifically show detections that meet a certain class. However, all the data exists for the customer to do this outside of the SightLine board. The detection and classification interface may be expanded in the future to support this capability.
Can a more complicated model structure be used on the 3000-OEM or 4000-OEM?	Yes, alternate model structures are supported, but there are limitations. See Appendix C for more details.
Does SightLine provide tools for extracting images for training?	Yes, using the Detection to Snapshot feature a large number of images can quickly be generated for training. See Appendix D for more details.

6.1 Questions and Additional Support

For questions and additional support, please contact Support. Additional support documentation and Engineering Application Notes (EANs) can be found on the Documentation page of the SightLine Applications website.

Appendix A - Caffe GPU Support

An NVIDIA GPU can be used to accelerate the training process. These instructions assume familiarity with the training process using the CPU version and a compatible NVIDIA GPU.

1. Determine the version of CUDA support:
   a. Open a command prompt and run `nvcc -V`.

If nvcc does not exist, the CUDA Tool Kit may need to be installed. See the NVIDIA Developer website to download the CUDA Toolkit.

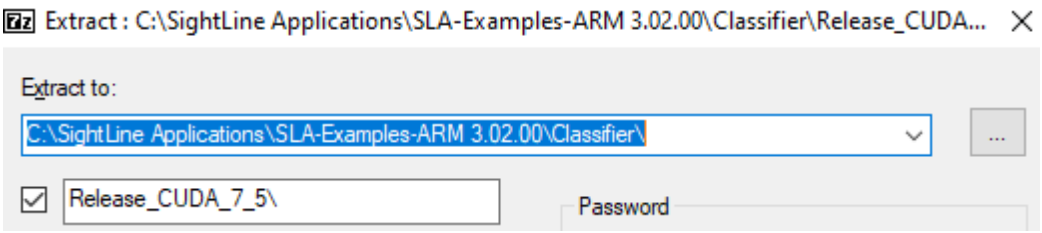
   b. Record the release version located after *Cuda compilation tools, release*.

```
PS C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier> nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2015 NVIDIA Corporation
Built on Tue_Aug_11_14:49:10_CDT_2015
Cuda compilation tools, release 7.5, V7.5.17
PS C:\SightLine Applications\SLA-Examples-ARM 3.02.00\Classifier>
```

2. Download the compatible Caffe build based on the version of CUDA identified in step 1 from the following list of links:

   CUDA 7.5 CUDA 8.0 CUDA 9.2 CUDA 10.2

3. Copy the file to the *C:\SightLine Applications\SLA-Examples-ARM-<<version number>>\Classifier*\.
4. Extract the file using 7zip.



5. Edit *create_trained_model.py* to set CAFFE_GPU to point to the new directory.

```
# ------------------------------------------
# Global Definitions
# - Edit these to match with your environment.
# ------------------------------------------
CAFFE_ROOT = R"./caffe"
CAFFE_GPU = R"./Release_CUDA_7_5"
```

Do not change CAFFE_ROOT
Change CAFFE_GPU to the new directory

6. Train the network by running:

   `04_train_model.bat`

When GPU is used for training, the created models are slightly different each time it is created. This is due to a non-deterministic nature of CUDA implementation.

Appendix B - Training Different Size Patches

32x32 training images are used in the training process described in Custom Classifier Development section. To train a network with a different size use the following modifications.

A set of 64x64 images included in the TrainingImages64.zip file can also be used for training.

1. In the *create_trained_model.py* file the directory name and image size needs to be changed as follows:

```
# Default definitions:
outputPath = R".output"
imageDirPath = R"TrainingImages64"
imageSize = "64"
snapshot = "sl_quick_iter_8000.solverstate"
modelH5 = "sl_quick_iter_11000.caffemodel.h5"
```

Change to `TrainingImage64`
Change `32` to `64`

2. In the *sl_quick.prototxt* file update the *dim* parameter near the top of the file to the correct size.

```
input_param { shape: { dim: 1 dim: 1 dim: 64 dim: 64 } }
```

Using a larger model will result in slower performance when running the classification on the hardware. Running a 32x32 classification on the 4000-OEM takes about 1.1 msec and running a 64x64 classification takes about 4.2 msec.

Appendix C - Using Alternate Model Structure

The model structure used in training is defined in *sl_quick_train_test.prototxt*. The model structure used for deployment (also called inference) is defined in *sl_quick.prototxt.* SightLine recommends starting with this model structure for training and deployment.

SightLine supports different model structures with some limitations. Contact us for more information.

The complexity of the model structure impacts runtime. Before investing a lot of time in training an overly complicated model SightLine recommends verifying that it will run on the 3000-OEM or 4000-OEM without impacting frame rates.

The Performance graphs in Panel Plus shows the time used by the processing steps. From the main menu go to View » Show Performance Graphs (see EAN-Performance-and-Latency).

C1 Supported Deployed Network Layers

There are a limited set of layers that are supported for the deployed network (*sl_quick.prototxt)*. If a binary file is loaded that includes layers that are not supported a user warning will be displayed. SightLine supports the following layers:

- Convolution
- Pooling
- Activation (ReLU, Sigmoid, TanH)
- Batch Normalization
- Inner Product
- Softmax

Contact Technical Support for any additional layers you may need that are not shown in this list.

During training, any layer that Caffe supports can be used, however it is important that the training network correctly trains for the final deployed network. For example, *Dropout* is a layer type used in *sl_quick_train_test.prototxt*. This is a normal training layer that is not used in the deployed network, which is acceptable. But, if a *threshold* layer is added to the training, that same layer may be needed in the deployed network, which is not a supported layer at this time.

Appendix D - Creating Training Images

Extracting images patches can be a very time-consuming part of the classifier training process. The snapshot detection feature described in the EAN-File-Recording can streamline this process. This feature can be used when streaming live video or on previously recorded video (see the EAN-Decoder).

Snapshot Detection setting notes:

- *Snapshot Mode: Detections*
- *Snapshot Size Type: Auto.* This will result in the actual detection being scaled to the setting in *Snapshot Size.*
- *Max Snapshots Per Frame: 1* to *3* depending on the storage media.
  - Use *1* for microSD cards.
  - Use up to *3* for an SSD USB drive.
- *Min Frames Between Snapshots*: *~30*

  For example, there are 30 detections in every frame. *Max Snapshots Per Frame is set to 1* and *Min Frames Between Snapshots* is set to 30. This means there will be one snapshot of each detection every second. After 10 minutes there will be over 10,000 snapshots.

Detection snapshots are not labeled and must be manually sorted prior to training.

If training a classifier for a specific detection algorithm, use that detection algorithm to gather snapshots so that the data you train on is a good representation of your real data.