

# SIGNALPOP

## AI DESIGNER

GETTING STARTED

## CONTENTS

Getting Started .....	1
Overview.....	7
Product Minimum Requirements .....	7
Datasets.....	8
Creating Datasets .....	8
Creating the MNIST Dataset .....	9
Creating the MNIST 'Source' and 'Target' Datasets.....	10
Creating the CIFAR-10 Dataset.....	11
Creating an Image Dataset.....	12
Viewing Datasets .....	14
Analyzing Datasets .....	15
Iterative PCA Analysis .....	16
t-SNE Analysis.....	18
Gym Datasets .....	20
Creating a Gym Datasets.....	21
Projects.....	22
Creating New Projects .....	22
Project Editing .....	23
Visual Configuration Dialogs .....	24
Text vs Graphical Editing.....	25
Model Toolbox .....	26
Layer Updates .....	27
Half Sized Memory.....	28
Freezing Learning.....	29
Opening Projects .....	30
Project Settings.....	31
Image Loading – LOAD_FROM_SERVICE Configuration.....	33
Training and Testing Projects.....	34
Scheduling Projects.....	37
Adding a Project to the Schedule .....	39

Setting up Secure Database Access.....	40
Setting the Scheduling Database .....	44
Scheduling States.....	45
Importing and Exporting .....	47
Importing a Project .....	47
Importing Pre-Trained Models .....	48
Importing Weights .....	49
Importing Weights for Transfer Learning .....	50
Exporting Projects.....	51
Exporting to Docker .....	52
Custom Trainers.....	56
Custom Trainer Settings - Trainers.....	57
Custom Trainer Settings - Properties .....	57
Debugging .....	61
Real-Time .....	61
Histogram Blob Visualization .....	62
Image Blob Visualization .....	63
Model Debugging.....	64
Visualizations .....	66
Weight Visualization .....	66
Network Visualization .....	67
Label Impact Visualization .....	68
Evaluators .....	69
Image Evaluation .....	69
Dream Evaluation.....	70
Neural Style Transfer Evaluation .....	71
Hardware .....	75
Processor Resource Window .....	75
Project Throughput .....	76
Real-Time Debug Timing .....	76
Memory Tester.....	77

Example Models.....	78
Domain-Adversarial Neural Networks (DANN) .....	78
Datasets.....	78
DANN Model.....	79
Validating the DANN Model.....	81
Deep Auto-Encoder Networks .....	83
Datasets.....	83
Auto-Encoder Model.....	83
Full Model .....	84
Model Analysis .....	85
Using Pre-Trained Auto-Encoder .....	89
Training Comparison.....	92
Siamese Network with Contrastive Loss for One-Shot Learning.....	94
Datasets.....	94
Siamese Net Model.....	94
Full Model .....	95
Training and Testing.....	99
Policy Gradient Reinforcement Learning .....	100
Datasets.....	100
Policy Gradient Model.....	101
Training.....	103
Deep Q-Learning (DQN) .....	105
NOISYNET Model .....	106
Training.....	108
LSTM Recurrent Learning .....	110
Shakespeare Output .....	110
LSTM Layer.....	112
LSTM Layer Internals .....	114
Training LSTM.....	116
LSTM_SIMPLE Layer.....	118
Training LSTM_SIMPLE .....	120



Encoder-Decoder Transformers (ChatGPT like) .....	122
Create the Transformer Model .....	123
Temporal Fusion Transformer Model for Time Series Prediction.....	126
Data Preparation.....	130
Data Preprocessing .....	130
TFT for Electricity Use Prediction .....	133
TFT for Traffic Flow Prediction .....	139
Summary .....	145
Appendix A – SignalPop Universal Miner .....	146
Hardware Monitoring.....	146
Appendix B – Dataset Creator Interface .....	148
IXDatasetCreator Interface .....	148
IXDatasetCreator::Name.....	148
IXDatasetCreator::QueryConfiguration.....	148
IXDatasetCreator::Create .....	148
DatasetConfiguration Object .....	149
DatasetConfiguration::IsReadOnly.....	149
DatasetConfiguration::ID .....	149
DatasetConfiguration::Name .....	149
DatasetConfiguration::SelectedGroup .....	150
DatasetConfiguration::Settings.....	150
DatasetConfiguration::Sort.....	150
DatasetConfiguration::Clone.....	150
DatasetConfiguration::SaveToFile .....	150
DatasetConfiguration::LoadFromFile.....	150
DataConfigSettingCollection Object.....	151
DataConfigSetting object .....	151
DataConfigSetting::VerifyInterface.....	151
DataConfigSetting::Name.....	151
DataConfigSetting::Extra .....	152
DataConfigSetting::Value .....	152

DataConfigSetting::Type .....	152
DataConfigSetting::Clone .....	152
DataConfigSetting::ToSaveString.....	152
DataConfigSetting::Parse .....	153
IXDatasetCreatorSettings Interface .....	153
IXDatasetCreatorSettings::VerifyConfiguration .....	153
IXDatasetCreatorSettings::GetCustomSetting.....	153
IXDatasetCreatorProgress Interface .....	154
IXDatasetCreatorProgress::OnProgress.....	154
IXDatasetCreatorProgress::OnCompleted .....	154
CreateProgressArgs Object.....	154
CreateProgressArgs::Aborted .....	155
CreateProgressArgs::PercentComplete.....	155
CreateProgressArgs::PercentCompleteAsText .....	155
CreateProgressArgs::Message .....	155
CreateProgressArgs::Error .....	155
CreateProgressArgs::Abort .....	155
Example Source Code .....	156
References .....	157

## OVERVIEW

The SignalPop® AI Designer is an application designed to help you develop, train, test and debug your deep learning models that use the MyCaffe™ AI Platform [1]. The MyCaffe AI Platform was inspired by the original open-source C++ Caffe project created by [2] while at Berkeley. For more information on MyCaffe, see us on **GitHub** at <http://github.com/mycaffe> or **NuGet** at <https://www.nuget.org/packages?q=MyCaffe>.

When discussing the designer, we will focus on several areas that are oriented around the common tasks that take place when building, training and debugging a deep learning model. The main areas of focus are:

- **Datasets;** these are the fundamental collections of data that models are trained on. Each dataset contains a set of images that are separated into a training and testing set. This section describes how to create and analyze datasets visually via the Iterative PCE and t-SNE algorithms.
- **Projects;** each project contains a model description, a reference to the dataset for which the model is run, and the model weights making up the trained model. In this section we discuss building and editing models with the visual editor and then discuss how to train and test your model.
- **Debugging;** debugging is an essential step in model development. Models can blow-up (drive to infinity or NaN) during training and when this happens it is nice to have a set of tools that help diagnose why. This section gives tips on how to better understand your models through the debugging tools offered by the SignalPop AI Designer. We also discuss some of the tools offered by the designer that help you diagnose your GPU hardware and describe how it is used.

So, let's get started, but before we do, please make sure that you are running on a system that meets the minimum requirements listed below.

## PRODUCT MINIMUM REQUIREMENTS

The SignalPop AI Designer was built for Windows developers running on either Windows 7 or Windows 10 PC's that have at least one NVIDIA CUDA based GPU installed. With that in mind the minimum requirements for the designer are as follows:

- Operating System: **64-bit Windows 7 or Windows 10**
- System Memory (PC side): **8 GB or more**
- Hard Disk Space: **10 GB free disk space**
- GPU Model: **NVIDIA 750Ti or above** (must have CUDA support and the latest NVIDIA driver<sup>1</sup>)
- GPU Memory: **2 GB** (4 GB or more is recommended)
- Multi-GPU Training: requires at least two headless GPUs that meet the GPU requirements above.
- Database: **Microsoft SQL or Microsoft SQL Express 2008 R2** or above (2016 on Windows 10 recommended) running in Windows Authentication Mode.

---

<sup>1</sup> To get the latest NVIDIA driver, please see <https://www.geforce.com/drivers>.

## DATASETS

Datasets are the fundamental store of data used to train and test each deep learning model. A list of images organized into a test set and training set make up each dataset. Image data is an ideal medium for training deep learning systems for they are visual and therefore are easy for a human to quickly understand - which can be very helpful when diagnosing their validity.

## CREATING DATASETS

The first step when using the SignalPop AI Designer is to create the datasets that you wish to train against. Dataset 'Creators' are plug-ins used to create each dataset. See the 'Dataset Creators' pane to view each creator already installed with the product.

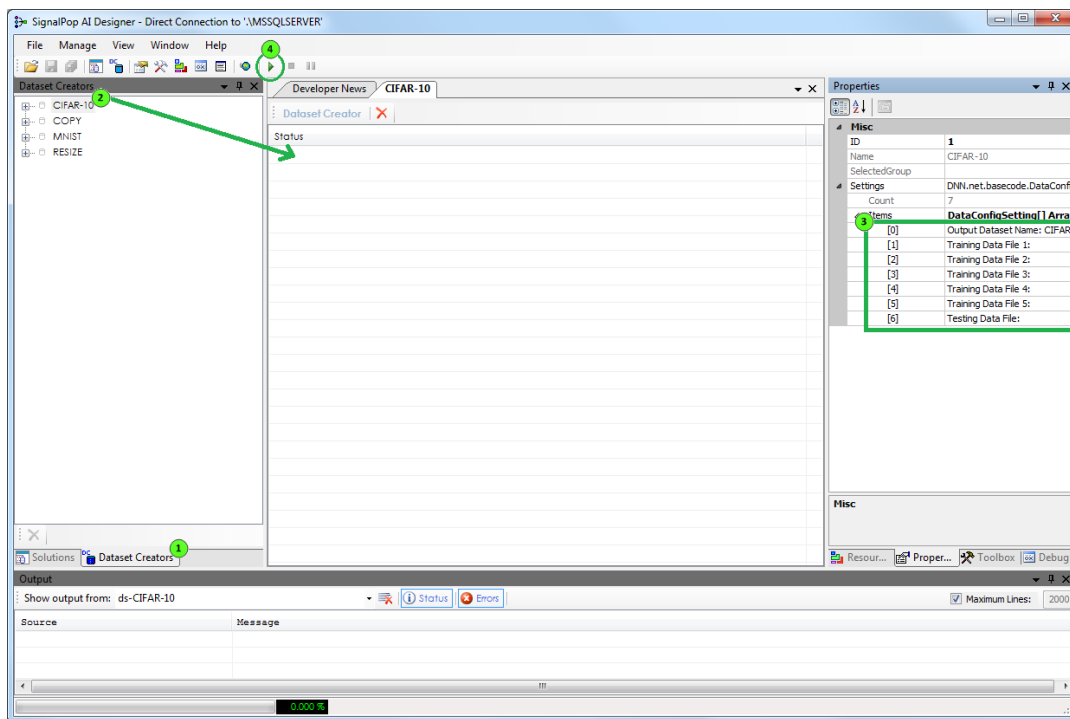


Figure 1 Creating Datasets

Take the following steps to create a new Dataset:

- 1.) Select the 'Dataset Creators' tab.
- 2.) Double-click the Dataset Creator (i.e., 'CIFAR-10') that you want to use, which will open the Dataset Creator window for that dataset creator.
- 3.) Fill out the Dataset Creator properties.
- 4.) Run the Dataset Creator by pressing the 'Run' (▶) button.

When the Dataset Creator completes running, the new dataset will appear under the Dataset Creator name in the tree-view.

## CREATING THE MNIST DATASET

The MNIST dataset is a dataset of 70,000 28x28 images of handwritten characters from the number zero (0) through the number nine (9). 60,000 of the characters are used for training and 10,000 are used for testing. This section describes how to create the MNIST dataset in the SignalPop AI Designer.

Before starting, first download the four GZ data files below from <http://yann.lecun.com/exdb/mnist/>.

<http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz>  
<http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz>  
<http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz>  
<http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz>

Once downloaded the MNIST Dataset Creator will use these files to create the MNIST Dataset.

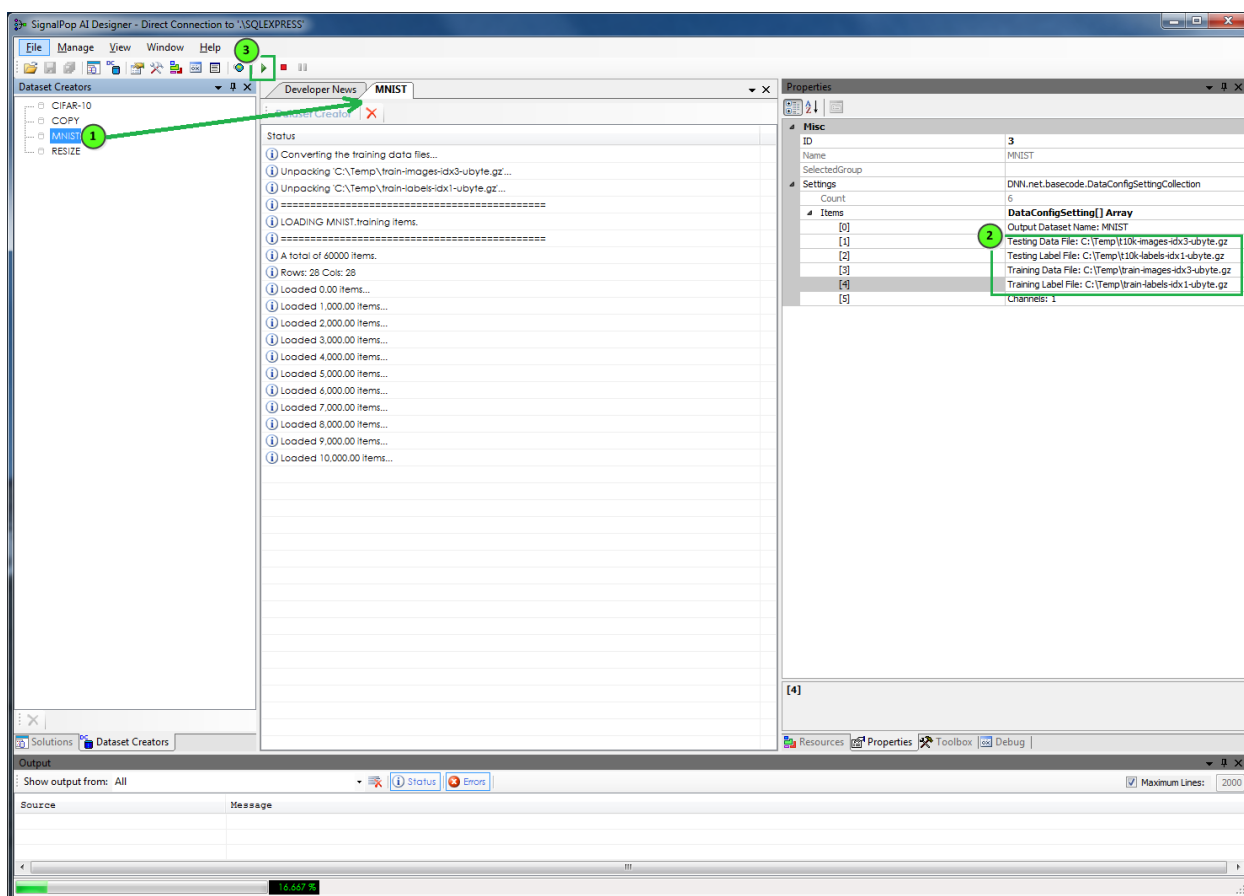


Figure 2 Creating the MNIST Dataset

To create the MNIST Dataset, do the following:

- 1.) Double-click the MNIST Dataset Creator to open the MNIST Creator window.
- 2.) Next, add the MNIST \*.gz files as shown above.
- 3.) Press the 'Run' (▶) button to start creating the dataset.

## CREATING THE MNIST 'SOURCE' AND 'TARGET' DATASETS

In addition to creating the standard MNIST dataset as described in the previous section, the MNIST Dataset Creator also allows you to create the MNIST dataset that is overlaid on top of an image of your choice to simulate environmental noise within the dataset images. This new dataset can be used as a 'target' dataset such as is used with Domain-Adversarial Training [3].

To create a 'source' MNIST dataset, follow the steps previously discussed to create the MNIST dataset, but with the 'Channel' set to 3. Both the source and target datasets must have the same shape.

To create the 'target' MNIST dataset, do the same by setting the 'Channel' to 3, but also set the 'Target Overlay File' to an image such as the standard 512x512 sized 'LENA.JPG' testing image found at <https://en.wikipedia.org/wiki/Lenna>. Setting a target overlay file causes the MNIST dataset creator to create the MNIST hand-written characters superimposed on top of the target overlay image.

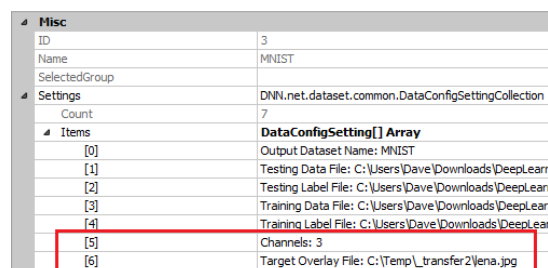


Figure 3 Settings for Target MNIST Dataset

Once created your target dataset will be named 'MNIST\_Target.3\_ch' and look as follows:

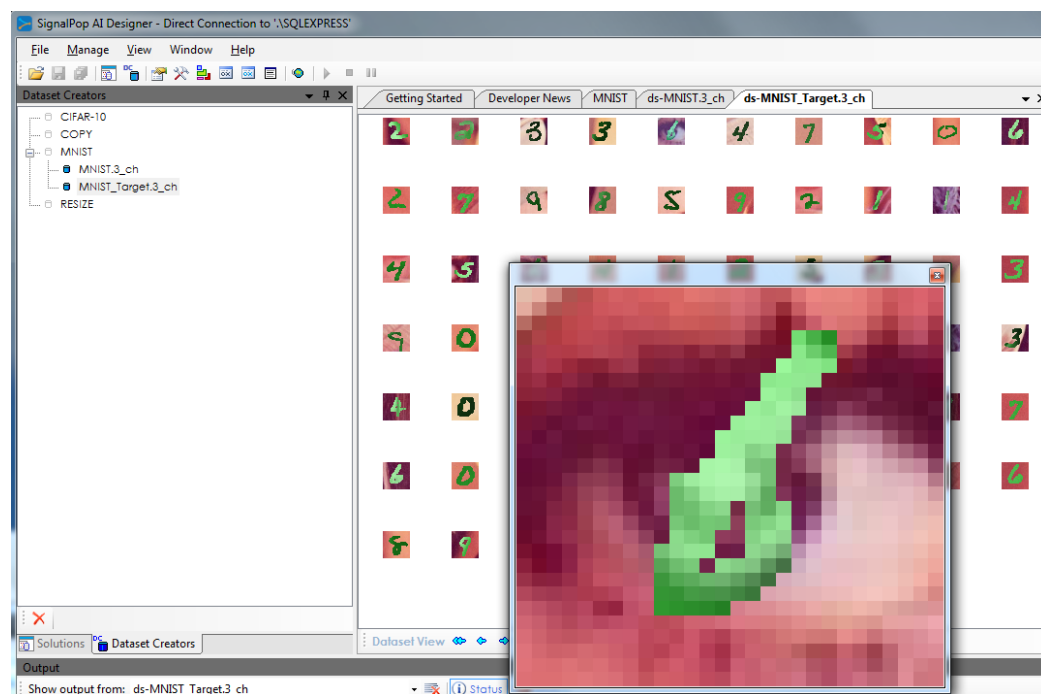


Figure 4 Target MNIST Dataset

## CREATING THE CIFAR-10 DATASET

The CIFAR-10 dataset is a dataset of 50,000 32x32 images of 10 classes including cats, dogs, boats, cars, etc. This section describes how to create the CIFAR-10 Dataset in the SignalPop AI Designer.

Before starting you will need to download the data file <http://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz> from Alex Krizhevsky's website. Using a tool like **7-zip** (<http://www.7-zip.org/>), this file expands into a \*.tar file which **7-zip** expands into 6 \*.bin files:

```
data_batch_1.bin
data_batch_2.bin
data_batch_3.bin
data_batch_4.bin
data_batch_5.bin
test_batch.bin
```

Once downloaded and expanded, the CIFAR-10 Dataset Creator will use these files to create the MNIST Dataset.

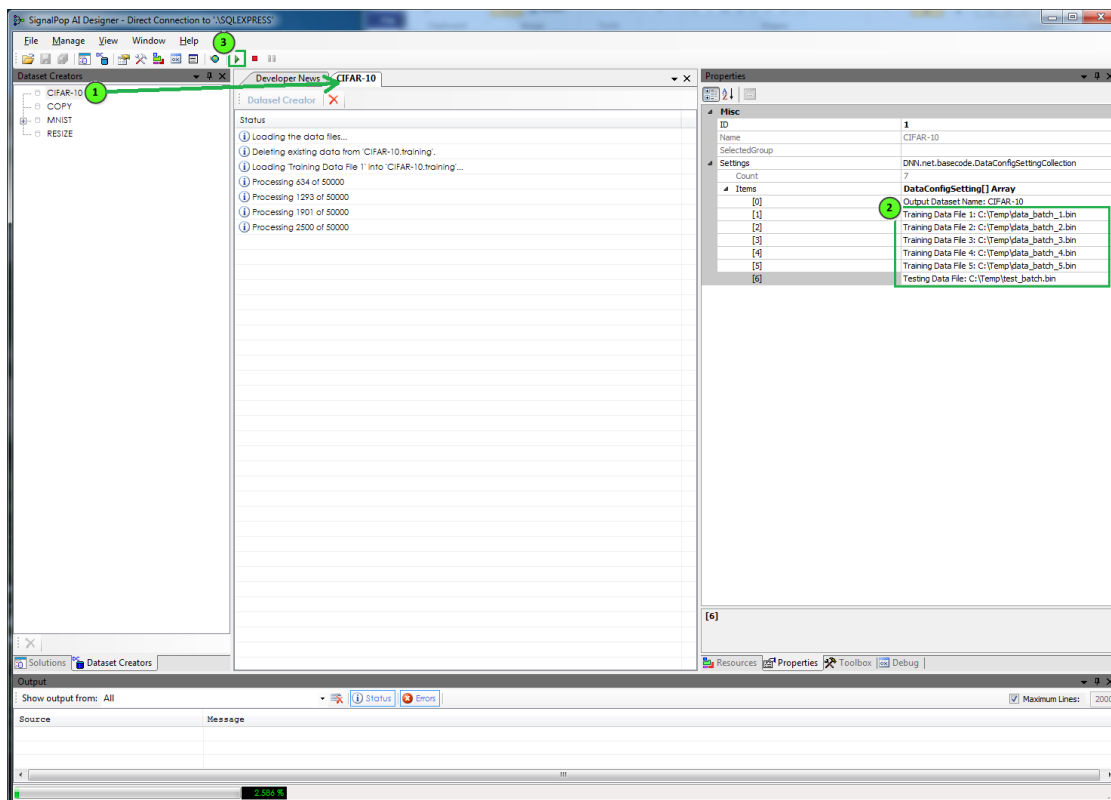


Figure 5 Creating the CIFAR-10 Dataset

To create the CIFAR-10 Dataset, do the following:

- 1.) Double-click the CIFAR-10 Dataset Creator to open the CIFAR-10 Creator window.
- 2.) Next, add the CIFAR-10 \*.bin files as shown above.
- 3.) Press the 'Run' (▶) button to start creating the dataset.

## CREATING AN IMAGE DATASET

The IMPORT.IMG dataset creator creates a dataset from a set of images located in a folder on your computer. In the example below, we have imported a subset of the MNIST images, previously created by exporting a portion of the MNIST dataset.

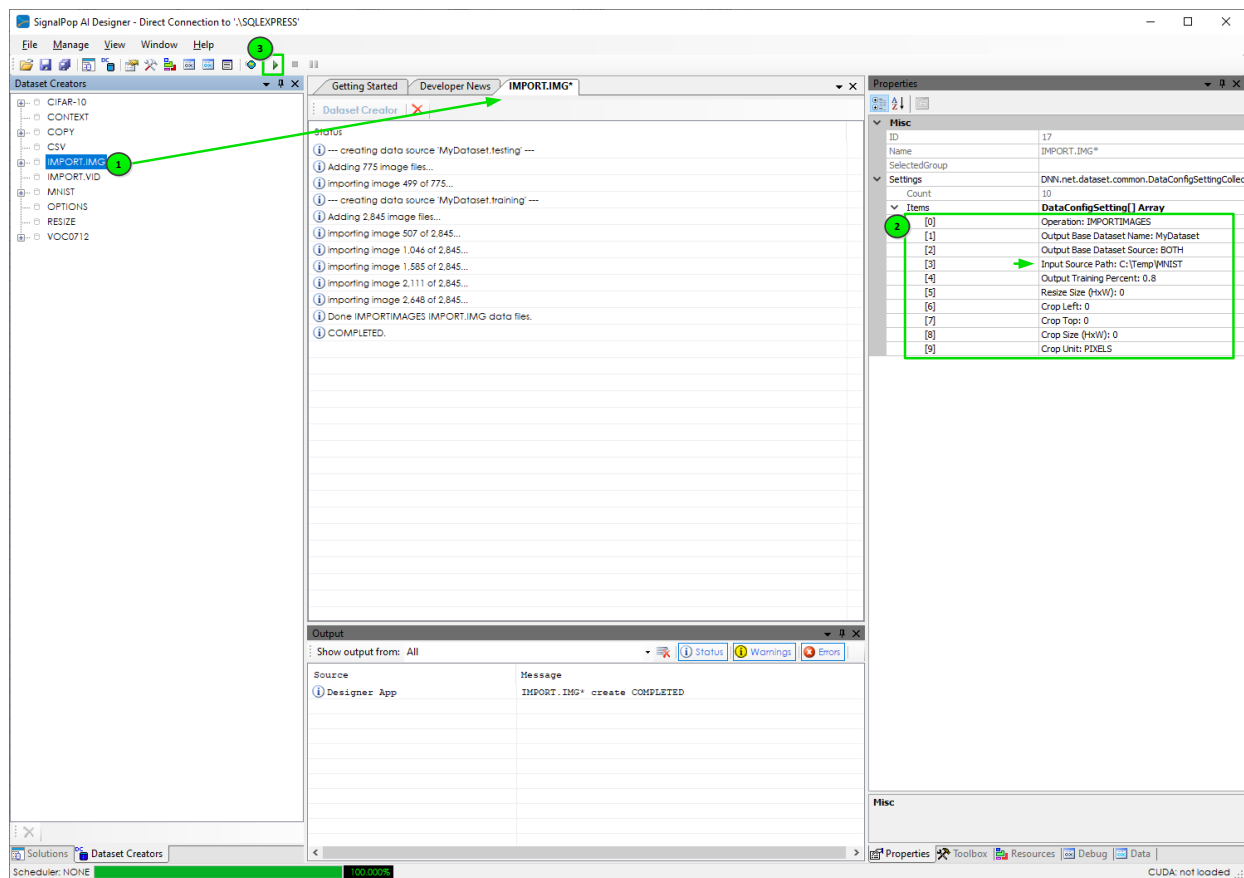


Figure 6 IMPORT.IMG Dataset Creator

To create a new dataset with this dataset creator, follow the steps below.

- 1.) Double click on the 'IMPORT.IMG' dataset creator name to open the 'IMPORT.IMG' dataset status window.
- 2.) Set the 'IMPORT.IMG' properties to include the 'Input Source Path' which should be the directory on your computer containing the images to import.
- 3.) Select the 'Run' button to start the import.

The properties of this dataset creator are as follows:

<b>Operation</b>	IMPORTIMAGES – specifies to import the images.
<b>Output Base Dataset Name</b>	Specifies the name of the new dataset to be created.
<b>Output Base Dataset Source</b>	BOTH, TRAINING, TESTING – specifies which dataset source to fill during the import (default = BOTH).



<b>Input Source Path</b>	Specifies the directory containing the images to import.
<b>Output Training Percent</b>	Specifies a value between [0.0, 1.0] that determines the percentage of images allotted to the training set (e.g., a value of 0.8 = 80%).
<b>Resize Size (HxW)</b>	When > 0, specifies the new size H x W of the images output into the dataset. Default = 0, which ignores resizing.
<b>Crop Left</b>	When > 0, specifies the left starting point of the crop in either pixels or percentage of the original image width (before resizing). Default = 0, which ignores this setting.
<b>Crop Top</b>	When > 0, specifies the top starting point of the crop in either pixels or percentage of the original image height (before resizing). Default = 0, which ignores this setting.
<b>Crop Size (HxW)</b>	When > 0, specifies the size of the crop region in either pixels or percentage of the original image width and height (before resizing). Default = 0, which ignores cropping.
<b>Crop Unit</b>	PIXELS = crop left, top and sizing values specify pixel values.  PERCENT = crop left, top and sizing values specify percentage values.

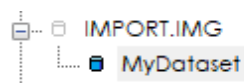
By default, all images imported are given a label of -1. However, if you would like to import labeled images, you can create a file named '\_filelist.txt' that contains the label information for each image. The format of the '\_filelist.txt' file is as follows:

```
<image file path> <label>
```

The following is an example section of the '\_filelist.txt' file used to import the subset of MNIST images.

```
c:\temp\MNIST\img_000000.png 5
c:\temp\MNIST\img_000001.png 0
c:\temp\MNIST\img_000002.png 4
c:\temp\MNIST\img_000003.png 1
c:\temp\MNIST\img_000004.png 9
c:\temp\MNIST\img_000005.png 2
c:\temp\MNIST\img_000006.png 1
c:\temp\MNIST\img_000007.png 3
c:\temp\MNIST\img_000008.png 1
```

Once created, you will see the name of your new dataset residing under the IMPORT.IMG dataset creator.



**Figure 7 New 'MyDataset' dataset**

## VIEWING DATASETS

Once created, viewing datasets helps you understand what types of data you will be creating a model for. For example, picture data can appear very different from heat-map data – both are images, but you may want to use different models for these two different types of data. The CIFAR-10 dataset is shown below.

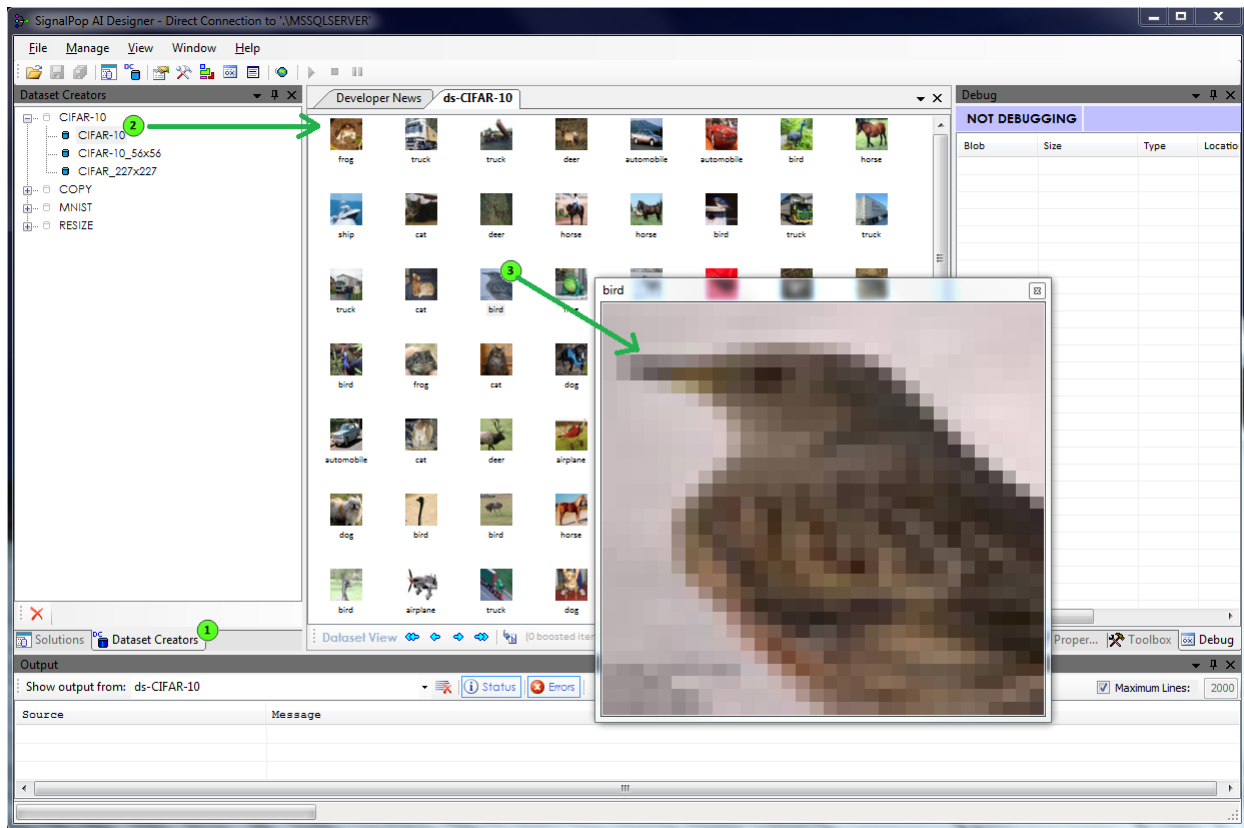


Figure 8 Viewing Datasets

To view datasets installed within the SignalPop AI Designer, take the following steps:

- 1.) Select the 'Datasets Creators' tab.
- 2.) Expand a Dataset Creator within the 'Dataset Creators' pane and double-click the dataset that you want to view, and it will appear in a new window.
- 3.) Double-click any image to enlarge.
- 4.) Training a project on a given dataset will create the image mean for the dataset. To view the image mean, right click within the white space of the dataset view window and select the 'Show Image Mean' menu item.

## ANALYZING DATASETS

Once a dataset is created, analyzing it visually can provide helpful insights on how 'learnable' the dataset actually is. The SignalPop AI Designer offers two types of visual dataset analysis: Iterative PCA and t-SNE.

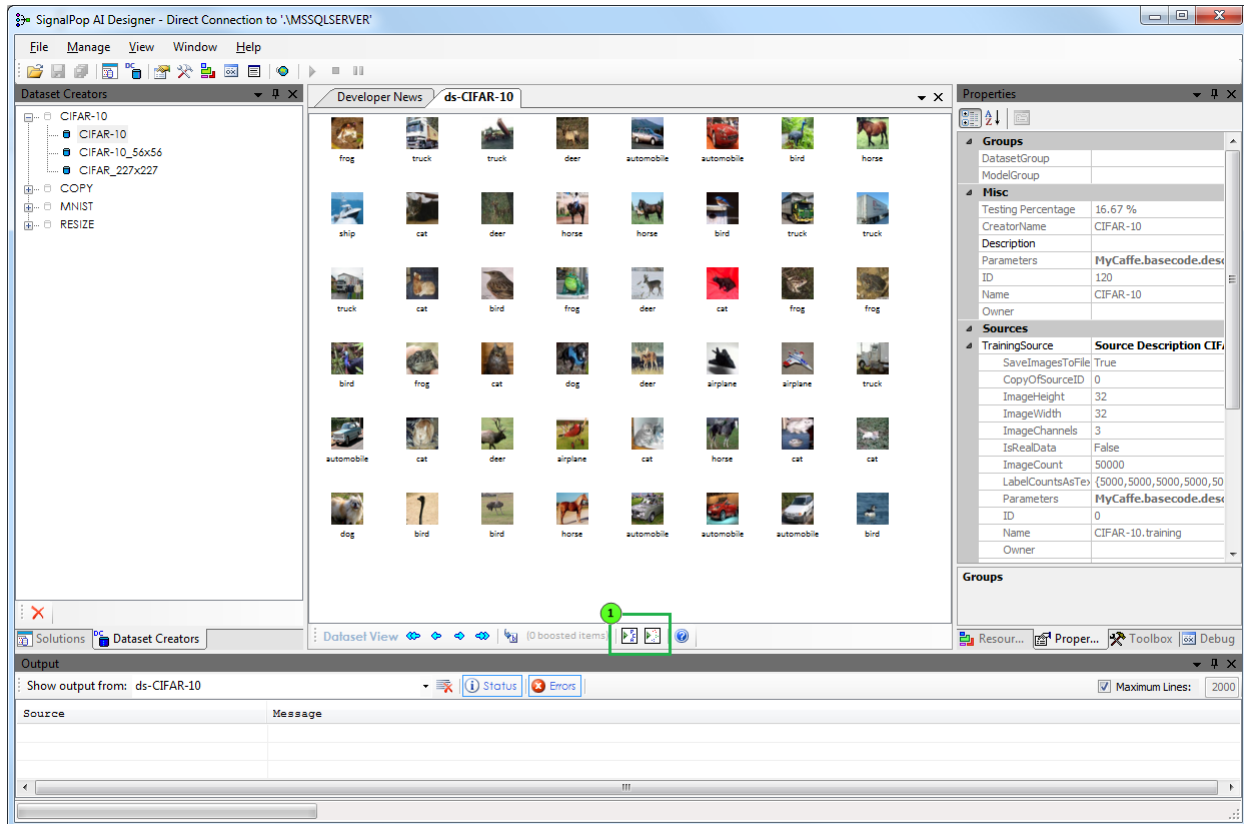


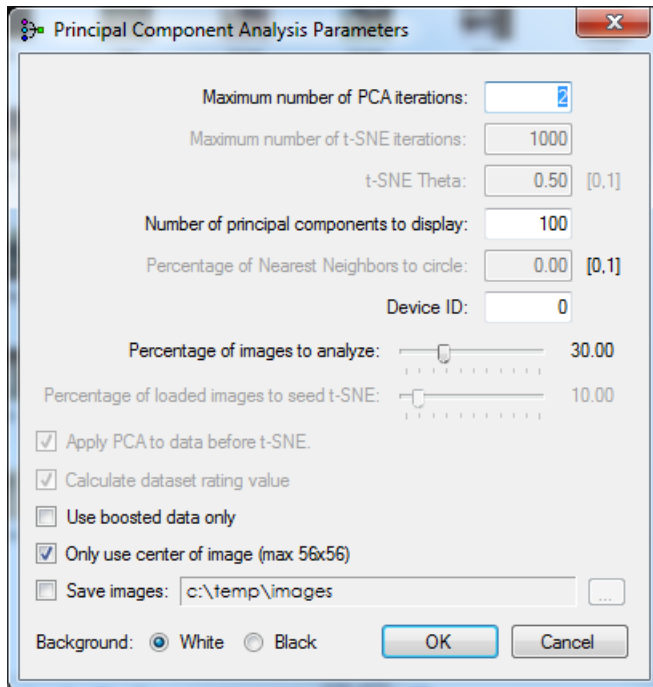
Figure 9 Dataset Visual Analysis

Select either the 'Run PCA Analysis' or 'Run T-SNE Analysis' at the bottom of the dataset window to start either analysis.

---

## ITERATIVE PCA ANALYSIS

The Iterative PCA Analysis uses an algorithm from [4] which leverages the speed of the underlying GPU to calculate the PCA values in refining iterations.



**Figure 10 Iterative PCA Parameters**

The following parameters are available when calculating the Iterative PCA of the dataset.

**Maximum number of PCA iterations:** Specifies the number of iterations to run the iterative PCA algorithm.

**Number of principal components to display:** Specifies the number of principal components that we want to calculate.

**Device ID:** specifies the ID of the GPU to use.

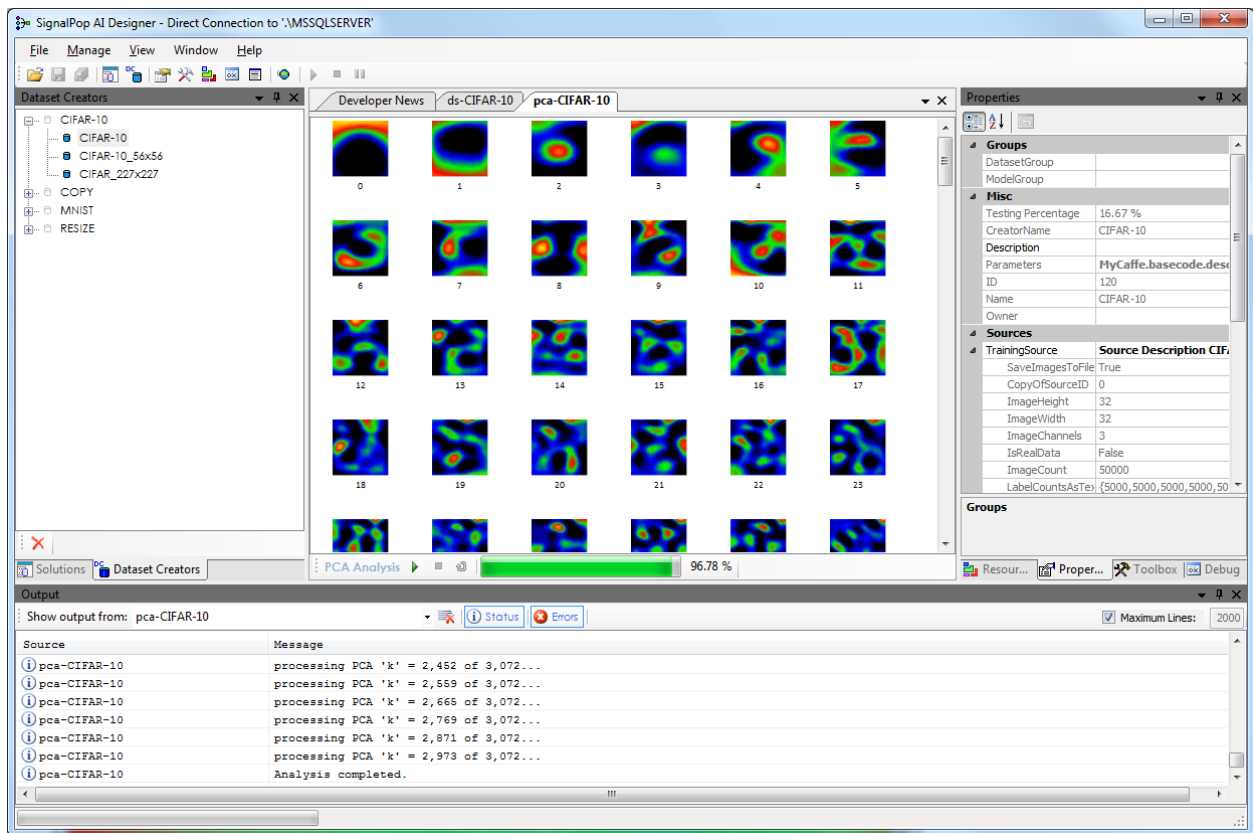
**Percentage of images to analyze:** Specifies the percentage of images to analyze with the algorithm. If you run out of memory, reduce this setting.

**Use boosted data only:** When set, only images that have a boost value are used.

**Only use center of image (max 56x56):** Specifies to only use up to 56x56 of the center of the image when calculating the PCA.

**Save images:** Specifies to save the resulting images to file in the directory specified.

**Background:** Specifies the color of the background to use.



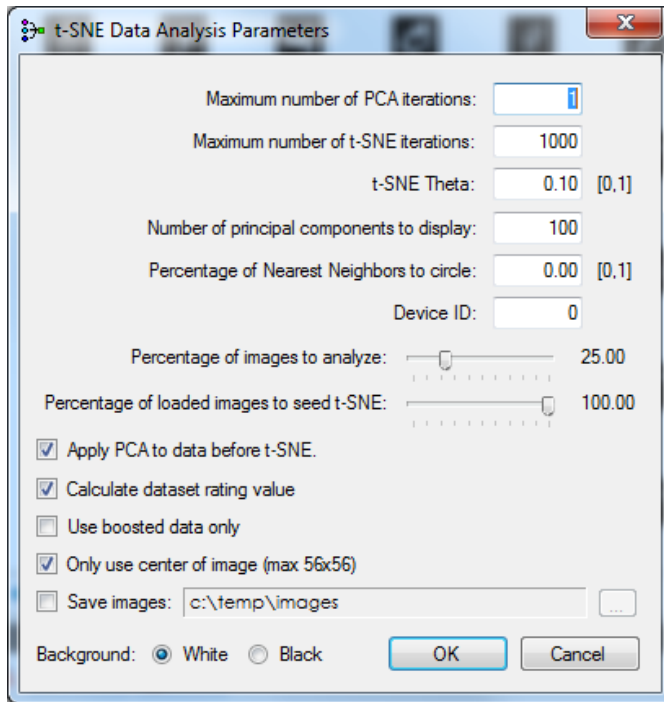
**Figure 11 Iterative PCA Results**

Upon completion, the images created for each PCA are shown in the 'pca-<dataset>' window.

---

## T-SNE ANALYSIS

The t-SNE Analysis uses an algorithm from [5] that leverages the speed of the underlying GPU to calculate the t-SNE animation.



**Figure 12 t-SNE Parameters**

The following parameters are available when calculating the t-SNE of the dataset.

**Maximum number of PCA iterations:** Specifies the number of iterations to run the iterative PCA algorithm when 'Apply PCA to data before t-SNE' is checked for dimension reduction.

**Maximum number of t-SNE iterations:** Specifies the number of t-SNE iterations to run.

**t-SNE theta:** Specifies the theta value in the range [0, 1]. According to [6], the theta value "specifies how coarse the Barnes-Hut approximation is".

**Number of principal components to display:** Specifies the number of principal components that we want to calculate when 'Apply PCA to data before t-SNE' is checked for dimension reduction.

**Percentage of Nearest Neighbors to circle:** When greater than zero, this specifies the percentage of neighbors to consider when drawing the centroid circles around each classification.

**Device ID:** Specifies the ID of the GPU to use.

**Percentage of images to analyze:** Specifies the percentage of images to analyze with the algorithm. *If you run out of memory, reduce this setting.*

**Percentage of loaded images to seed t-SNE:** Specifies the number of loaded images to use when seeding the t-SNE algorithm.

**Apply PCA to data before t-SNE:** When selected, the PCA algorithm is run first and then the t-SNE is run on the PCA results.

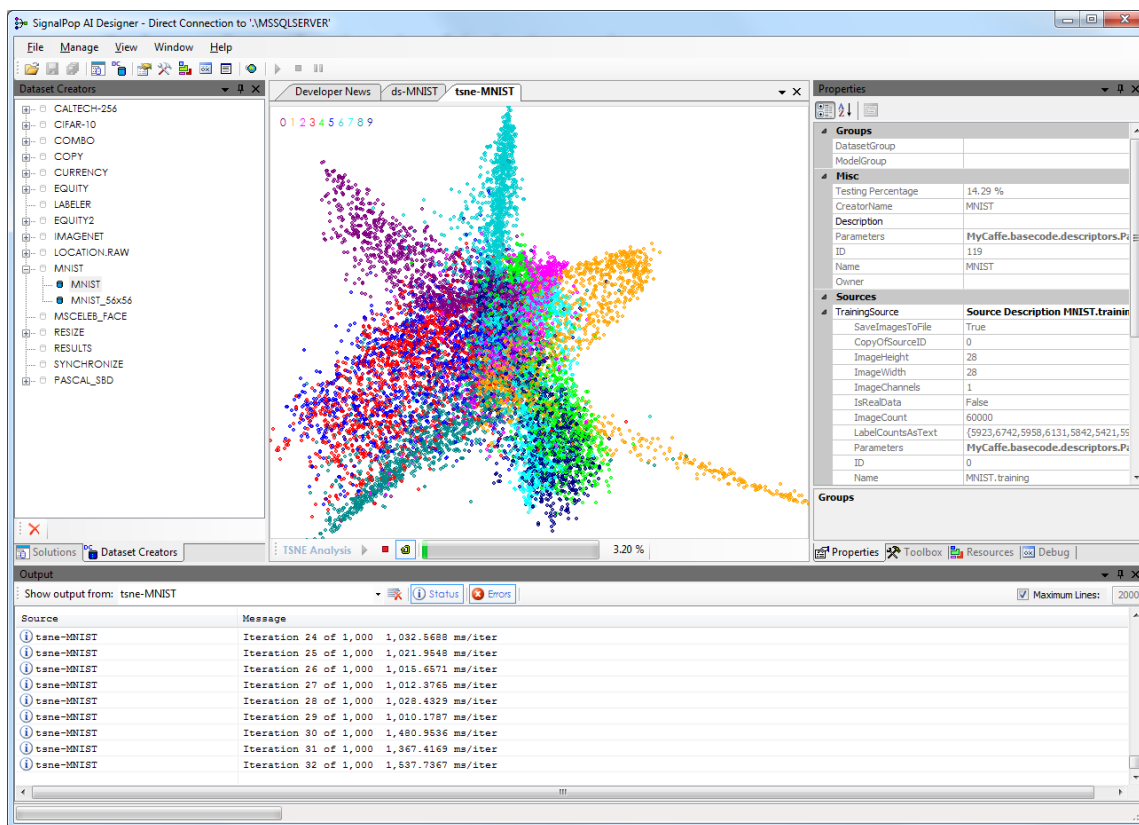
**Calculate dataset rating value:** When selected, the dataset 'learnability' rating value is calculated.

**Use boosted data only:** When set, only images that have a boost value are used.

**Only use center of image (max 56x56):** Specifies to only use up to 56x56 of the center of the image when calculating the PCA and t-SNE.

**Save images:** Specifies to save the resulting images to file in the directory specified.

**Background:** Specifies the color of the background to use.



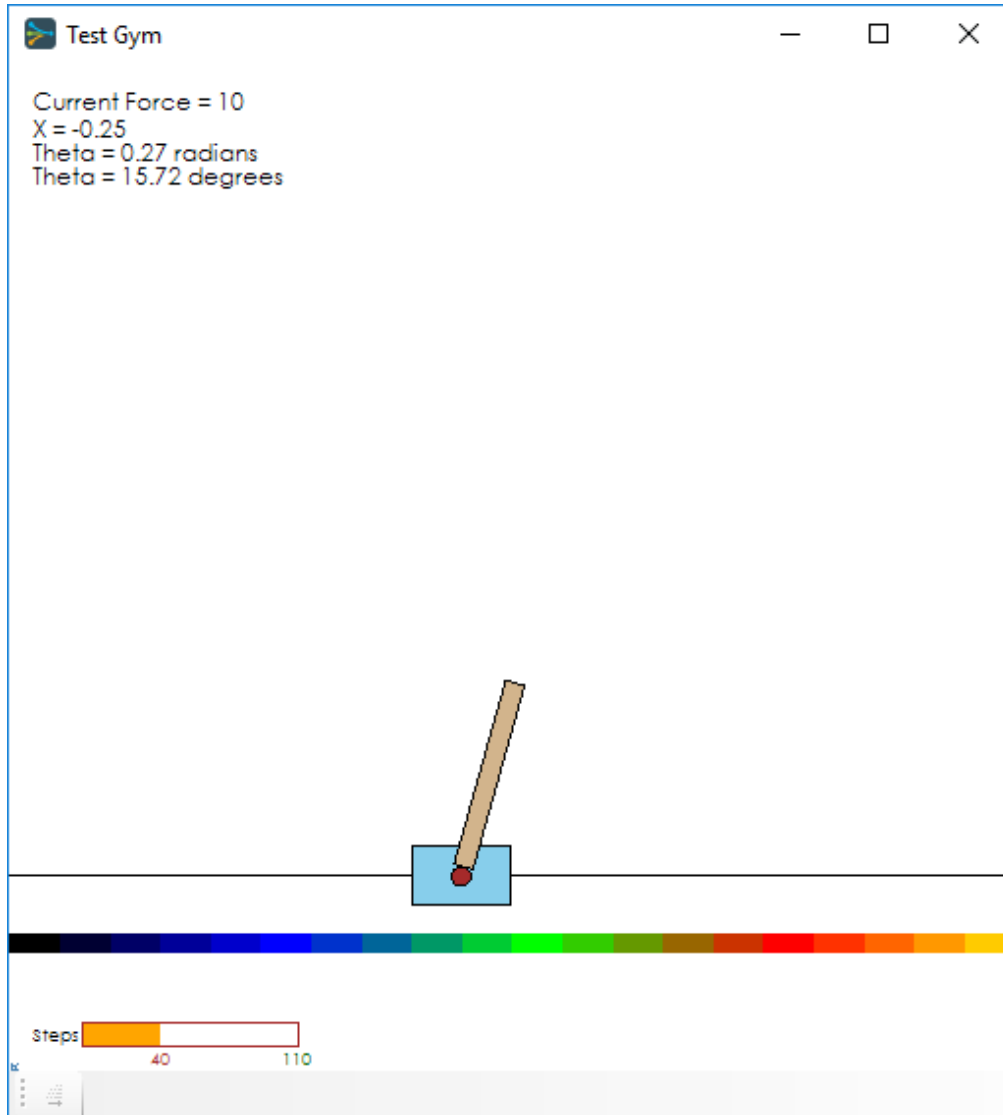
**Figure 13 t-SNE Animation**

While the t-SNE calculates, an animation of the results are shown in the 't-sne-<dataset>' window. The final image shows the full data separation.

To see the specific images making up the animation, left click in the image and drag to select the points to view and the images associated with each point will then be displayed in a new window along with the % of images falling into each class within the region.

## GYM DATASETS

Gym datasets are dynamic datasets that create new data on each iteration. For example, the Cart-Pole [7] gym available in Python from OpenAI on GitHub [8] (originally copied from Rich Sutton et al. [9] [10]), provides a simulation of a cart balancing a pole. Our C# version of the cart-pole simulation was inspired by the OpenAI Python version.



**Figure 14** Cart-Pole Gym

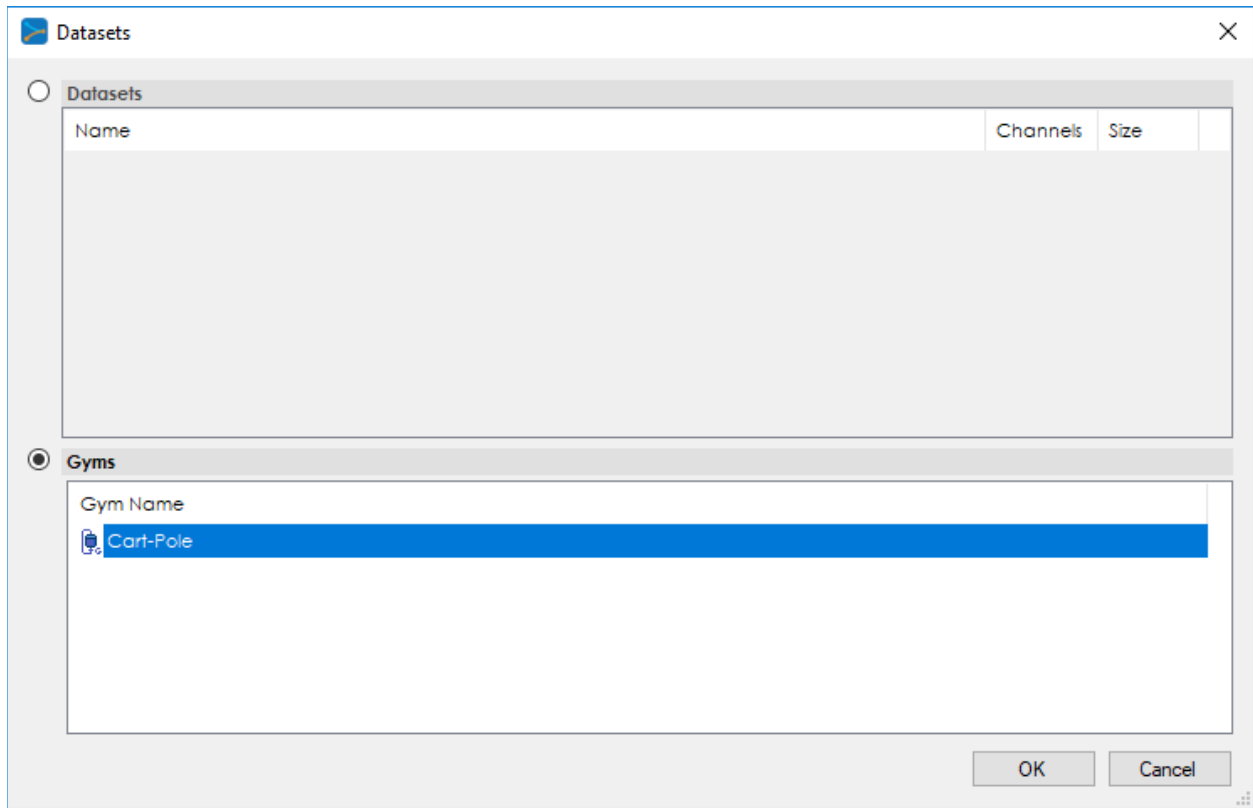
In our simulation, the cart (in light blue) moves left and right with the tan pole balanced on top of the cart. Users of this simulation apply a force to the left and right of the cart to balance the pole.

Gyms are treated like any other data set with one main exception – gym datasets have no labels for they dynamically create their data as the gym runs.



## CREATING A GYM DATASETS

The process of creating a gym dataset is no different from any other dataset. When creating a project, selecting the dataset will show the *Datasets Dialog*.



**Figure 15 Datasets Dialog**

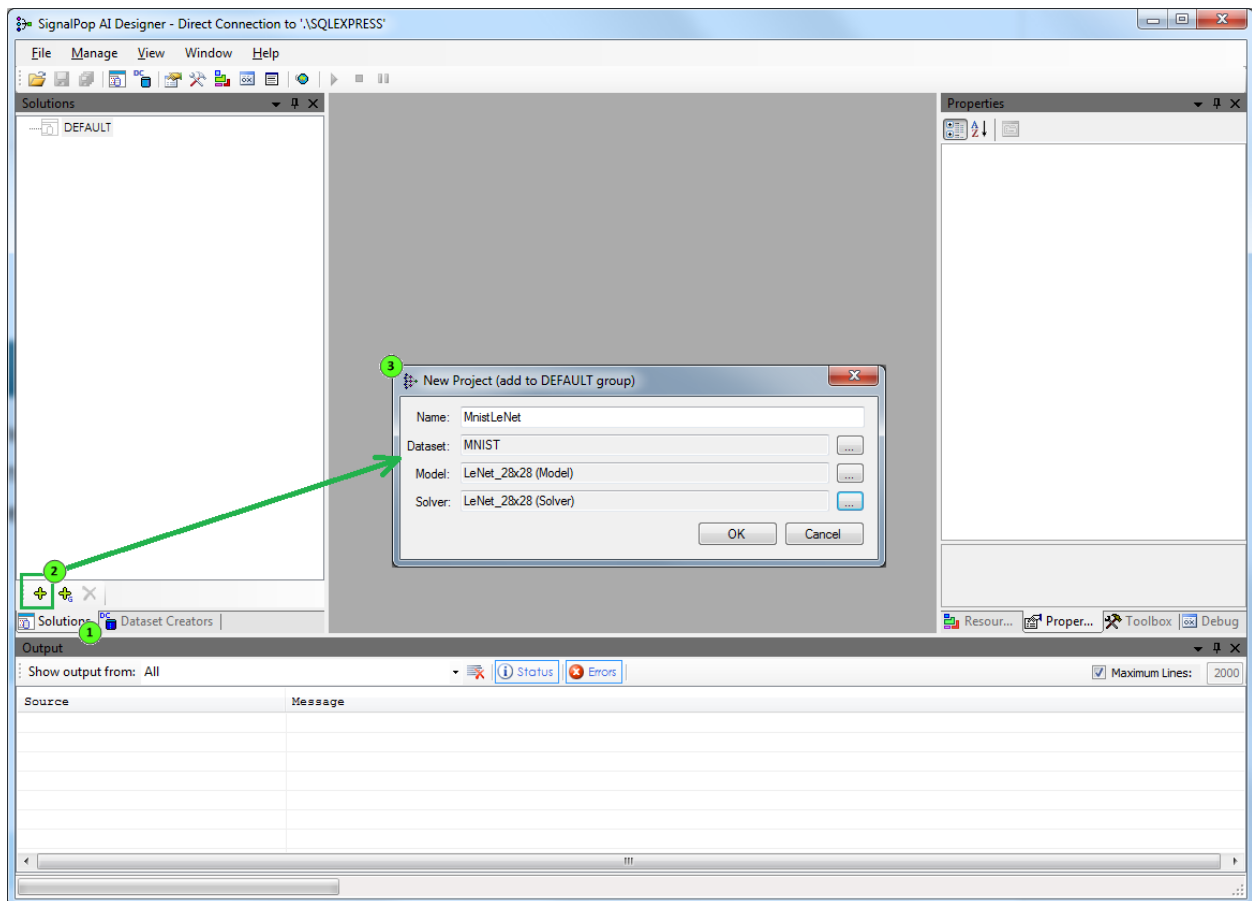
From this dialog you will see both datasets and gyms, where the gyms produce a dynamic dataset that (other than its missing labels) look just like any other dataset.

## PROJECTS

Projects are the basic unit used by the SignalPop AI Designer to manage each model and its associated data. Each project contains a reference to the dataset used, the model description, the solver description, and the trained model weights.

## CREATING NEW PROJECTS

Creating a project associates the dataset used to the model and solver that will be trained on it.



**Figure 16 Creating a Project**

To create a project, do the following:

- 1.) Select the *Solutions* pane.
- 2.) Select the *Add Project* (+) button at the bottom of the pane.
- 3.) Fill out the 'New Project' dialog with the project name, and the model and solver to use.

Once added, you will see the new project appear in the *Solutions* pane.

## PROJECT EDITING

Double-clicking the Model (📄) name within a project displays the Model Editor window where you can view the model visually, or in its text-script form.

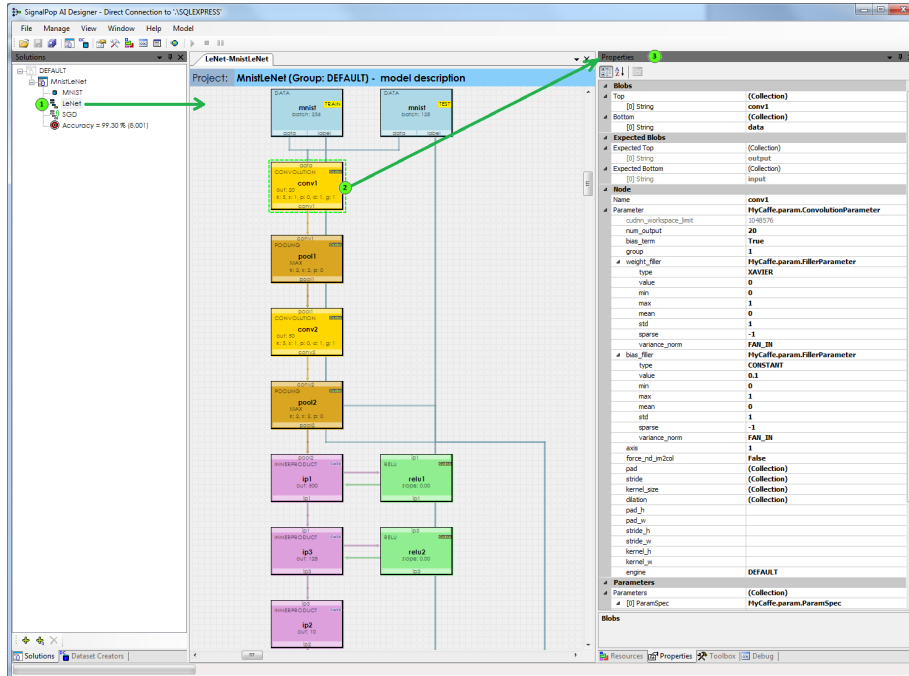


Figure 17 Editing Projects

From the Model View Editor, you can easily edit the properties of each layer, add, and remove layers and even visually configure layers.

To edit a model, do the following:

- 1.) Select the 'Solutions' tab and double click on the Model (📄) name to open the Model Editor window. NOTE: The model editor remains in a read-only mode when the project is open. To edit, close the project first.
- 2.) From the Model Editor window, select a layer to see and edit its properties...
- 3.) ...in the Properties window.

To better help you analyze your model, hovering the mouse over a selected layer displays a tooltip with information describing the layer.

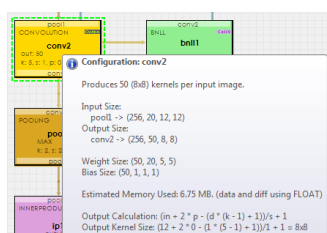
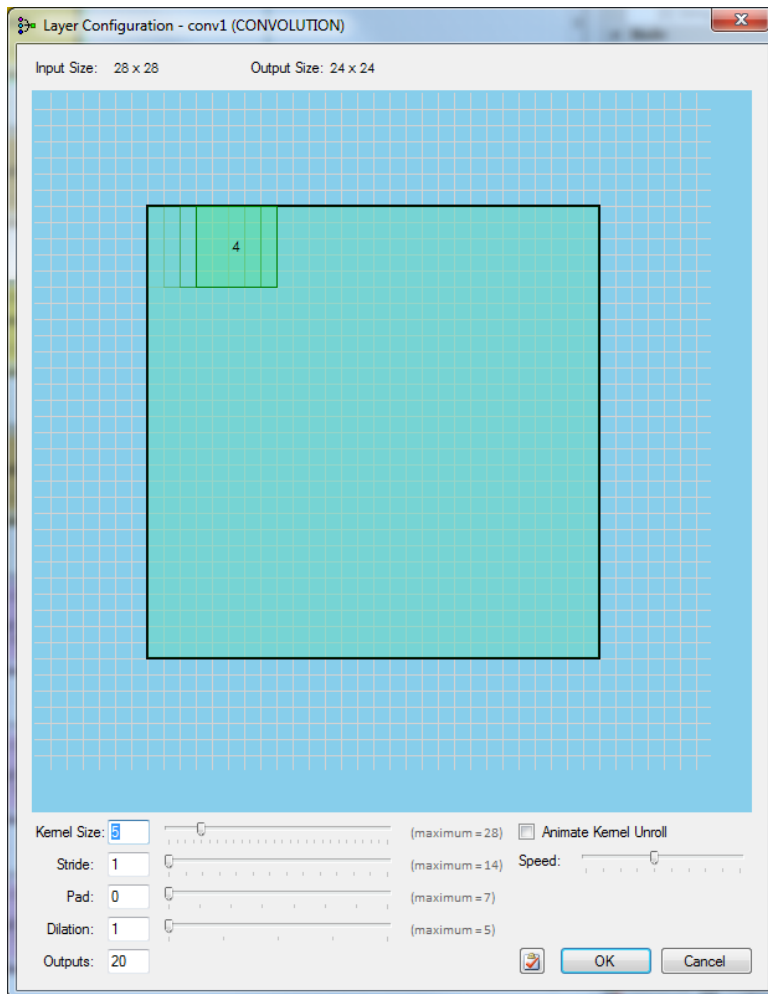


Figure 18 Layer Tooltip

## VISUAL CONFIGURATION DIALOGS

Several layers, such as the Convolution and Pooling layers, support visual configuration dialogs.



**Figure 19 Visual Configuration Dialog**

Visual configuration dialogs allow you to visually set the kernel, stride and pad settings and show visually how the settings will impact the way the layer uses its input Blobs.

## TEXT VS GRAPHICAL EDITING

The Model Editor window supports both graphical and text editing.

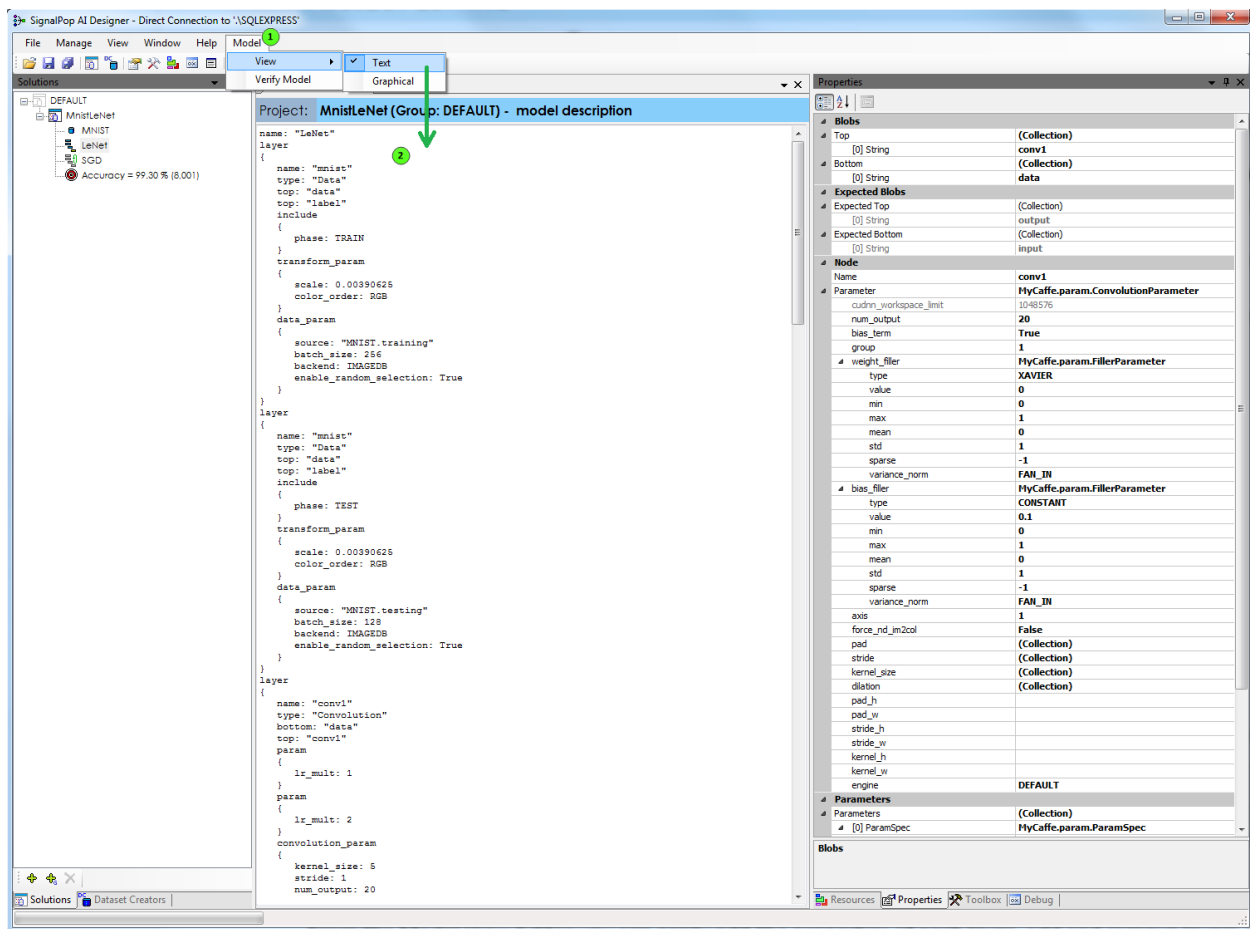


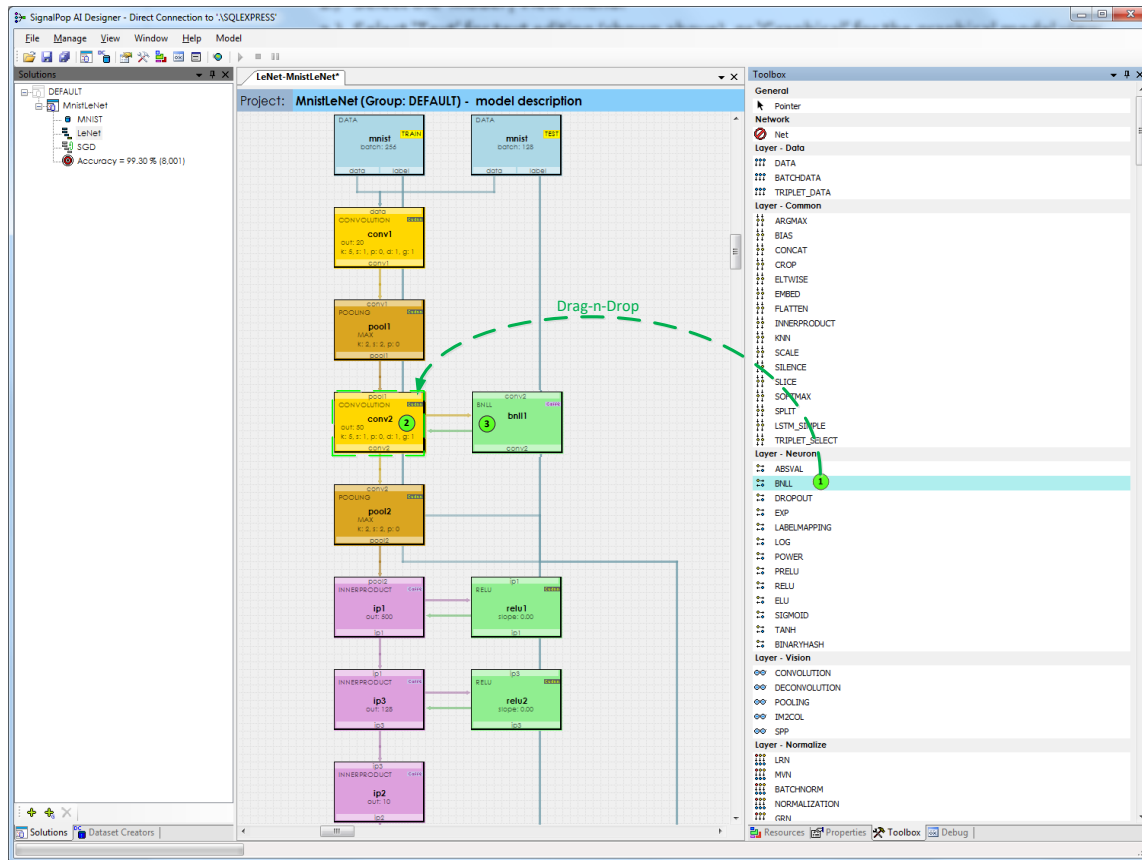
Figure 20 Text and Graphical Editing

To switch between text and graphical editing, do the following:

- 1.) Select the 'Model | View' menu.
- 2.) Select 'Text' for text editing (shown above), or 'Graphical' for the graphical model view.

## MODEL TOOLBOX

To build your model further, you can use the Model Toolbar window.



**Figure 21 Model Toolbox Window**

To add layers to your model, just drag and drop them from the toolbox window onto other existing layers using the following steps:

- 1.) Select the new layer to add from the 'Toolbox' window.
- 2.) Drag the item over to and drop it onto an existing layer.
- 3.) Upon drop, the new layer will appear attached to the existing layer for which it was dropped.

After editing your model, pressing the Save (💾) button then saves the updated model to your project.

## LAYER UPDATES

Right clicking in the model edit window allows you to change settings that apply to all layers in the model. When right clicking on a selected layer, these settings only apply to the selected layer.

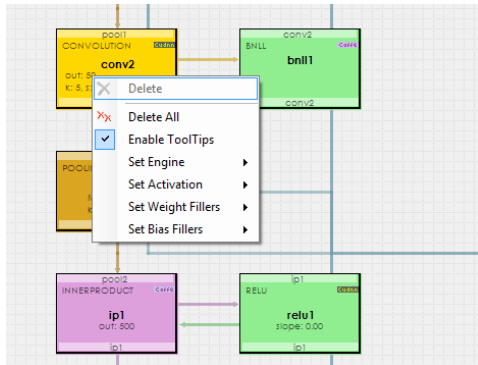


Figure 22 Changing Model Settings

Right clicking on the model (when no layers are selected) allows you to change the following items for each layer for which the settings apply:

- **Set Engine;** sets the engine used by each layer to DEFAULT, CUDNN or CAFFE.
- **Set Activation;** sets all activation layers to RELU, TANH, SIGMOID, PRELU or ELU.
- **Set Weight Fillers;** sets all weights fillers to UNIFORM, GAUSSIAN, XAVIER, MSRA, POSITIVE UNIT BALL, or CONSTANT.
- **Set Bias Fillers;** sets all bias fillers to UNIFORM, GAUSSIAN, XAVIER, MSRA, POSITIVE UNIT BALL, or CONSTANT.

Right clicking on a selected node allows you to quickly make these settings and immediately go to the programming help for the node – just select the 'Help' menu item to go there.

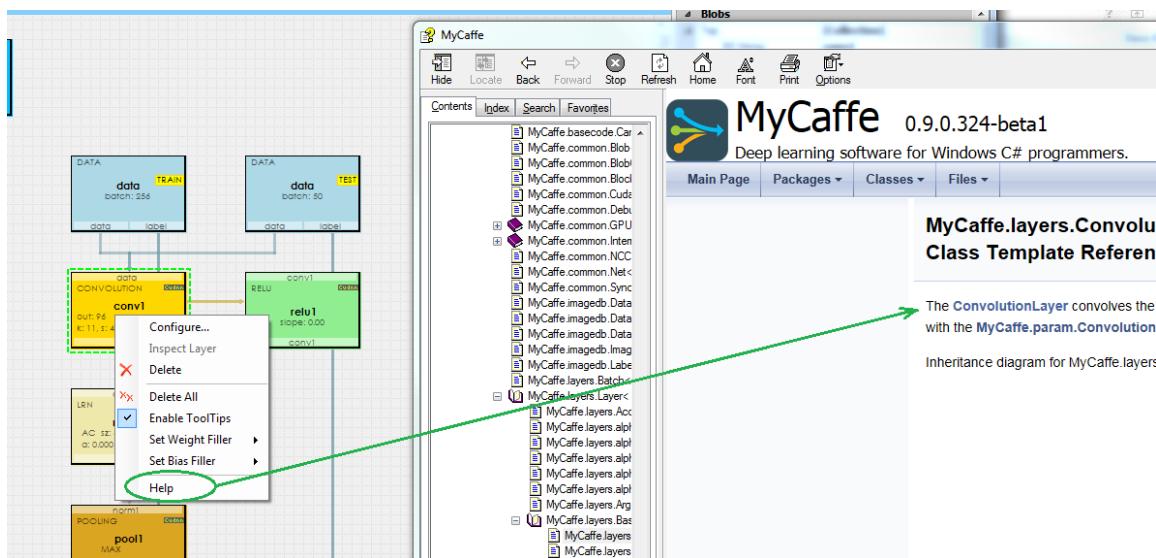


Figure 23 Extensive Layer Help

## HALF SIZED MEMORY

By default, the SignalPop AI Designer uses the *float* base type, which also allows you to further optimize select layers to use half-sized memory. When using half-sized memory, the weights, top and bottom blobs of the layer are configured to use the half-size type which is  $\frac{1}{2}$  the size of a floating-point number. This does impact the precision of the layer but also reduces its memory footprint and can improve training times.

To enable half-sized memory, simply right click on a layer and select the *Enable Half Size* menu option. Once enabled, a small 'half' will appear on the layer graphic.

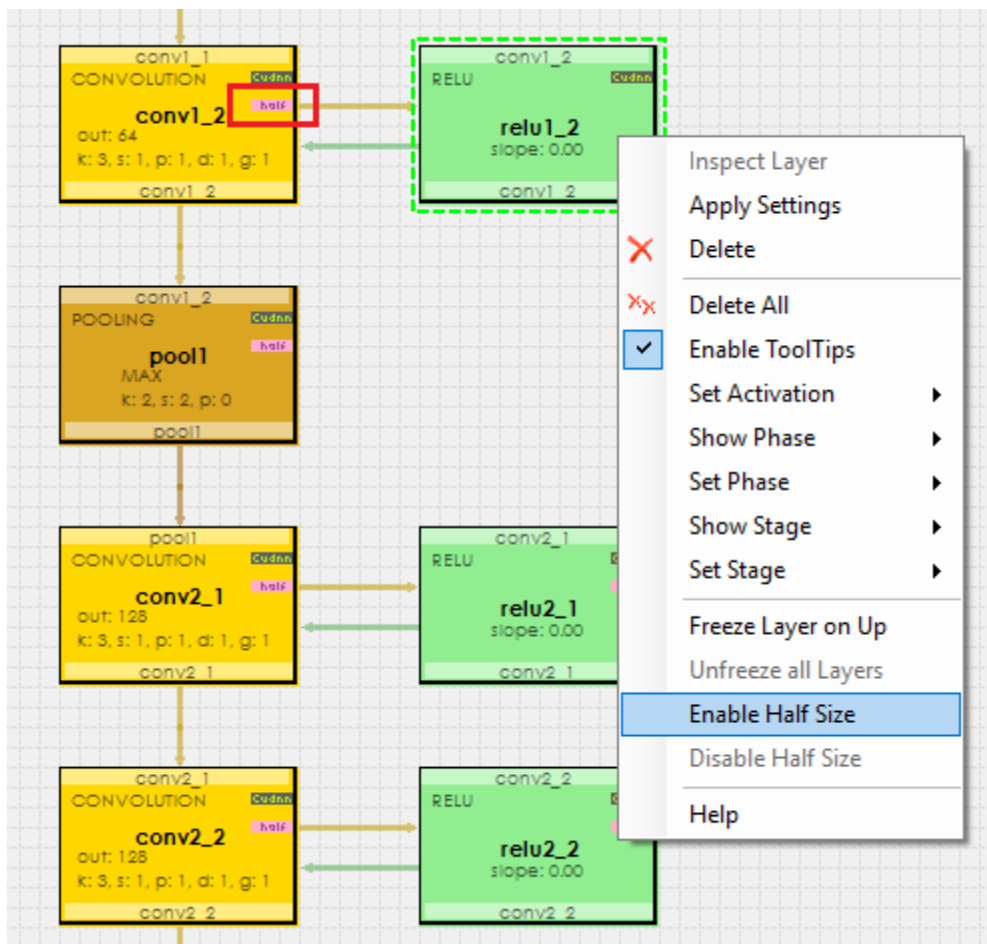


Figure 24 Half-Sized Memory

NOTE: when using half-sized memory, the precision of that layer will be reduced to a 2-byte floating point number which can cause the model to blow up with an infinity or NaN loss. When this occurs, try reducing the scaling of the input data.

Currently half-sized memory is available on the CONVOLUTION, POOLING, RELU, TANH, and SIGMOID layers when run with the CUDNN engine and the INPUT layer.



## FREEZING LEARNING

During the learning process, you may want to freeze the learning in each layer. When a layer is frozen, training proceeds in the normal manner yet the weight updates are not applied to any frozen layer. To freeze a layer either edit its properties or select the *Freeze Layer on Up* option to freeze the layer and all its predecessors.

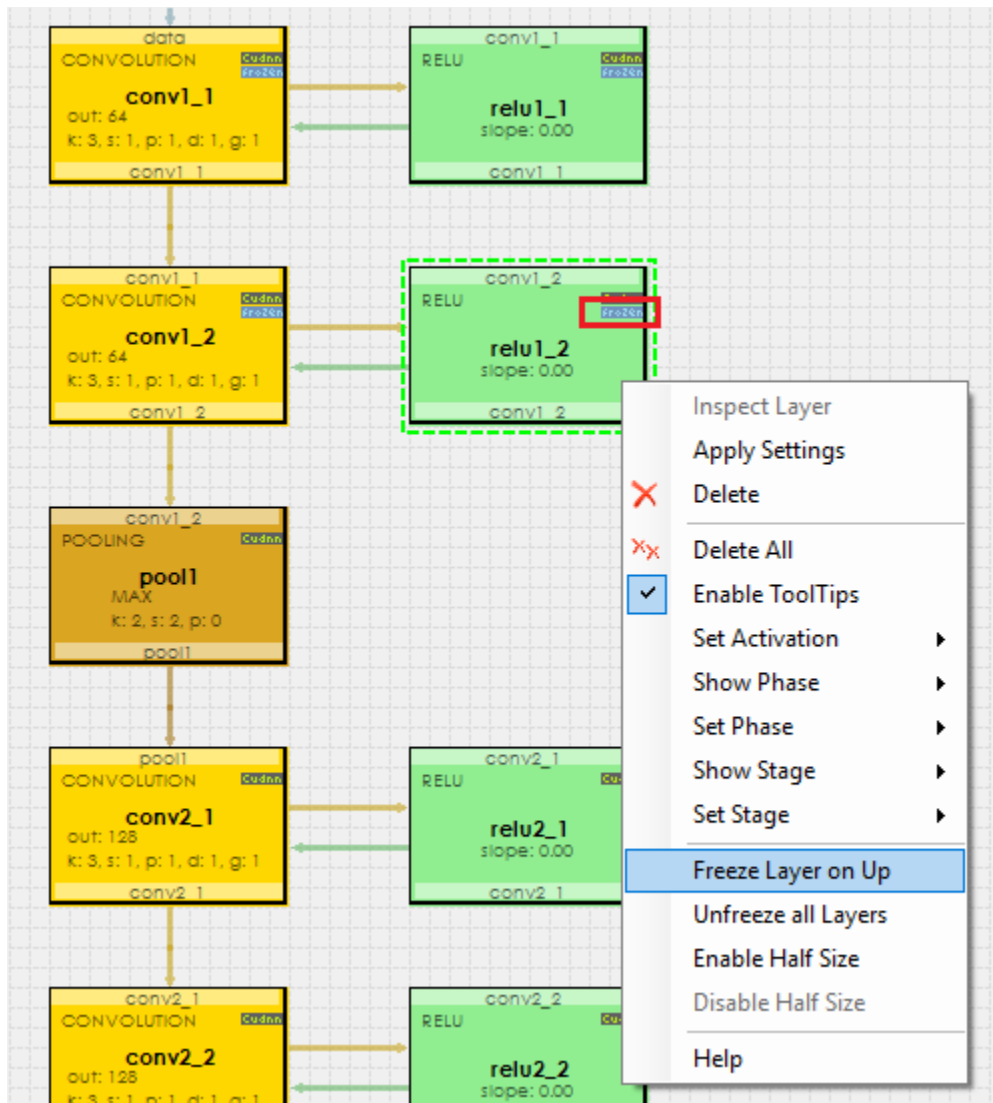


Figure 25 Freezing Learning

When frozen, a small 'frozen' is displayed on the layer graphic.

Layer freezing can be very helpful when performing transfer learning (e.g., importing a pre-trained model and fine tuning the end layers to the problem at hand).

## OPENING PROJECTS

When working with a project, you must first 'open' the project which loads the dataset into memory, creates the model scaffolding and then loads any trained weights into the model used.

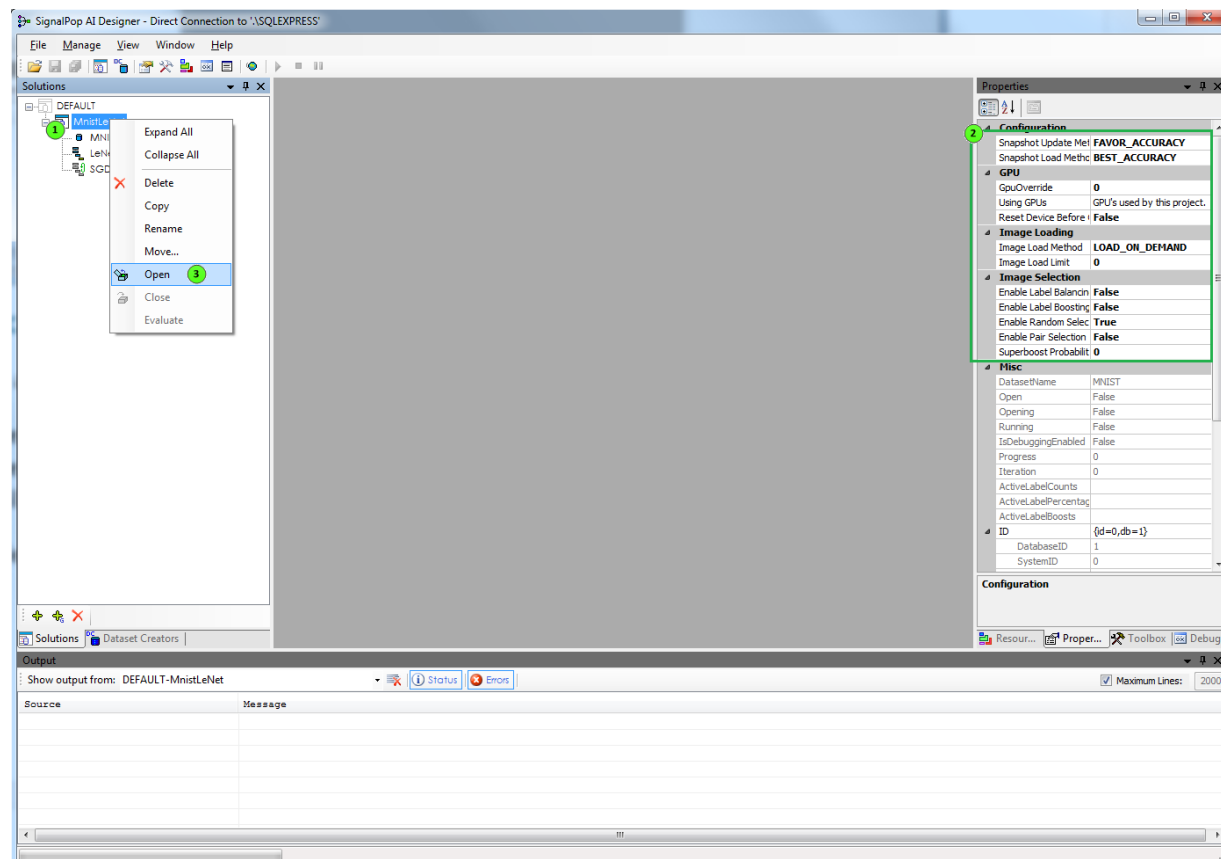



Figure 26 Opening a Project

To open a project, do the following:

- 1.) First select the project so that its properties show in the 'Properties' pane.
- 2.) Set the properties to use for the project, including the 'GPU Override' which specifies the GPU on which to open the project. In addition, make sure to select the 'Image Load Method'. When opening a project you can choose to 'LOAD\_ON\_DEMAND' which loads the images into memory as needed (and caches them), 'LOAD\_ALL' which loads all images into memory before using them (which dramatically improves training speeds), 'LOAD\_FROM\_SERVICE' which provides the best of both options, where by the images are loaded and retained in memory by the SignalPop In-Memory Database Service, or 'LOAD\_ON\_DEMAND\_BACKGROUND' which does not wait for all images to load, but starts loading them into memory in the background while returning the image requested<sup>2</sup>.

<sup>2</sup> The LOAD\_ON\_DEMAND\_BACKGROUND option is only available in the MyCaffe Image Database version 2.

3.) Next, right click on the project and select the 'Open' menu item to open the project.

Once open, the project name will appear in bold  **MnistLeNet** with a green light attached.

**NOTE:** The database icon in the lower right-hand side of the Project Window, displays the % for which the in-memory database is loaded. Optimal speeds are reached only after the database is fully loaded.

---

## PROJECT SETTINGS

Before opening a project, you will want to set the configuration under which the project is to be opened. This section describes these settings and how they impact the project opened.

Configuration	
Snapshot Update Method	FAVOR_ACCURACY
Snapshot Load Method	BEST_ACCURACY
GPU	
GpuOverride	0
Using GPUs	GPU's used by this project.
Reset Device Before Opening Project	False
Image Loading	
Image Load Method	LOAD_ON_DEMAND
Image Selection	
Enable Label Balancing	False
Enable Label Boosting	False
Enable Random Selection	True
Enable Pair Selection	False
Superboost Probability	0

**Figure 27 Project Settings**

The following project settings impact how the project behaves when it is opened.

**Snapshot Update Method;** there are three types of snapshot update methods: FAVOR\_ACCURACY which favors triggering snapshots when the accuracy improves (increases), FAVOR\_ERROR which favors triggering snapshots when the error improves (decreases), and FAVOR\_BOTH which favors triggering a snapshot when either improve.

**Snapshot Load Method;** the snapshot load method is used when loading the weights into the project as it is opened. There are three snapshot load methods: BEST\_ACCURACY which loads the last snapshot with the best accuracy, BEST\_ERROR which loads the last snapshot with the best error, and LAST\_STATE which loads the last snapshot taken.

**GpuOverride;** the GPU override specifies the GPU ID(s) on which to train, test and run the project. GPU IDs are zero bases and increase up to one minus the number of GPUs that you have installed. Multi-GPU training is supported on headless GPU's (the GPU's that do not have a monitor plugged into them). Both TCC and WDM mode multi-GPU training are supported, by you will want to make sure that all GPU's used in the multi-GPU training are set to the same mode (i.e., either all set to TCC, or all set to WDM). See the [nvidia-smi documentation](#) for more information on setting the GPU mode.

**Reset Device Before Opening Project;** enabling this setting resets the GPU used before opening the project. **IMPORTANT:** This setting is **ONLY** recommended when testing as it will wipe the GPU and can disrupt other software using the GPU.


**Image Load Method;** as described previously, there are three image loads methods:

- 1.) *LOAD\_ON\_DEMAND*; Loads the images as they are used (which is slower),
- 2.) *LOAD\_ALL*; Loads all images into memory before using (faster training, slower load),
- 3.) *LOAD\_FROM\_SERVICE*; Uses the SignalPop In-Memory Database Service to load the images (faster training and faster load once loaded).
- 4.) *LOAD\_ON\_DEMAND\_BACKGROUND*; Loads the images as they are used while simultaneously starts loading all the images in the background<sup>3</sup>.
- 5.) *LOAD\_ON\_DEMAND\_NOCACHE*; Loads the images as they are used and does not cache the images which uses less memory but is slower<sup>4</sup>.

**Enable Label Balancing;** when enabled, the images are selected first by randomly selecting the label and then selecting the image from the label selected. Using this setting with datasets that have uneven data across classes can help balance the classes presented to the network and thus make training more accurate. When datasets have balanced data across classes (i.e., CIFAR-10 or MNIST), this setting is not needed.

**Enable Random Selection;** when enabled, the images are randomly selected from the dataset, or from the label set selected if label balancing is enabled. When disabled, images are selected sequentially which is not recommended for training.

**Enable Pair Selection;** when enabled, the first image is selected using the 'Enable Random Selection' setting and the second image query selects the image immediately following (in sequence) the first image selected.

**Superboost Probability;** this is a value within the range [0, 1] that specifies the probability that an image is selected from the 'boosted' set of images if any exist where a value of 1 = 100%. Boosted images are images marked with a boost value set via the Dataset Viewer window. To increase the boost value of an image, left click on the image while holding down the ALT key. To reset all boost values, select the Reset All Boosts () button.

---

<sup>3</sup> The *LOAD\_ON\_DEMAND\_BACKGROUND* option is only available in the MyCaffe Image Database version 2.

<sup>4</sup> The *LOAD\_ON\_DEMAND\_NOCACHE* option is only available in the MyCaffe Image Database version 2.

---

## IMAGE LOADING – LOAD\_FROM\_SERVICE CONFIGURATION

When using the LOAD\_FROM\_SERVICE image loading, the SignalPop AI Designer uses the SignalPop In-Memory Database Windows Service. To configure the service, you must make sure that the 'LogOn As' account used has access to your database – otherwise the service will fault when attempting to load images from the database.

By default, the 'LogOn As' account was set when you installed the SignalPop AI Designer. However, if you need to change this account, simply do the following:

- 1.) Open the 'Services' Configuration Window by selecting the 'Administrative Tools' from the Control Panel, and then select the 'Services' administrative tool.
- 2.) From within the 'Services' Configuration Window, scroll down to the 'SignalPop In-Memory Database Service' and double click it.
- 3.) Select the 'Log On' tab and configure the account that you would like the Service to use.
- 4.) Completely stop the service and then start to make sure that any connections to the previously run service are killed. A service restart does not always do this.

Next, make sure that your DNN database gives this account access to both 'Connect' and 'Select' for the SignalPop In-Memory Database only reads from the database. To do this, simply do the following:

- 1.) Open the 'SQL Server Management Studio'.
- 2.) Open the main, root-level 'Security | Logins' tab and make sure that your account has log-in access.
- 3.) Next, right click the 'DNN' database and select 'Properties' and then click the 'Permissions' page from the left panel.
- 4.) On the 'Permissions' add your account and grant 'Connect' and 'Select' access to your account.

---

## SQL DATABASE ACCOUNT

When accessing a SQL database from a Service, the account used must be granted special access on the SQL database.

The account you use must be granted the following generate 'Securables' found by right clicking on the username within the 'Security|Logins' tree and selecting the *Properties* menu item. The following securables must be granted for the machine on which the user is associated:

*Connect SQL*

*View any definition*

Next, from the 'Databases|DNN' database tree, select the *Properties* menu and then select the *Permissions* page. Select the user account and grant the following permissions:

*Connect*

## TRAINING AND TESTING PROJECTS

After opening a project, you are now ready to train and test it. Double clicking on the open project name opens the *Project* window.

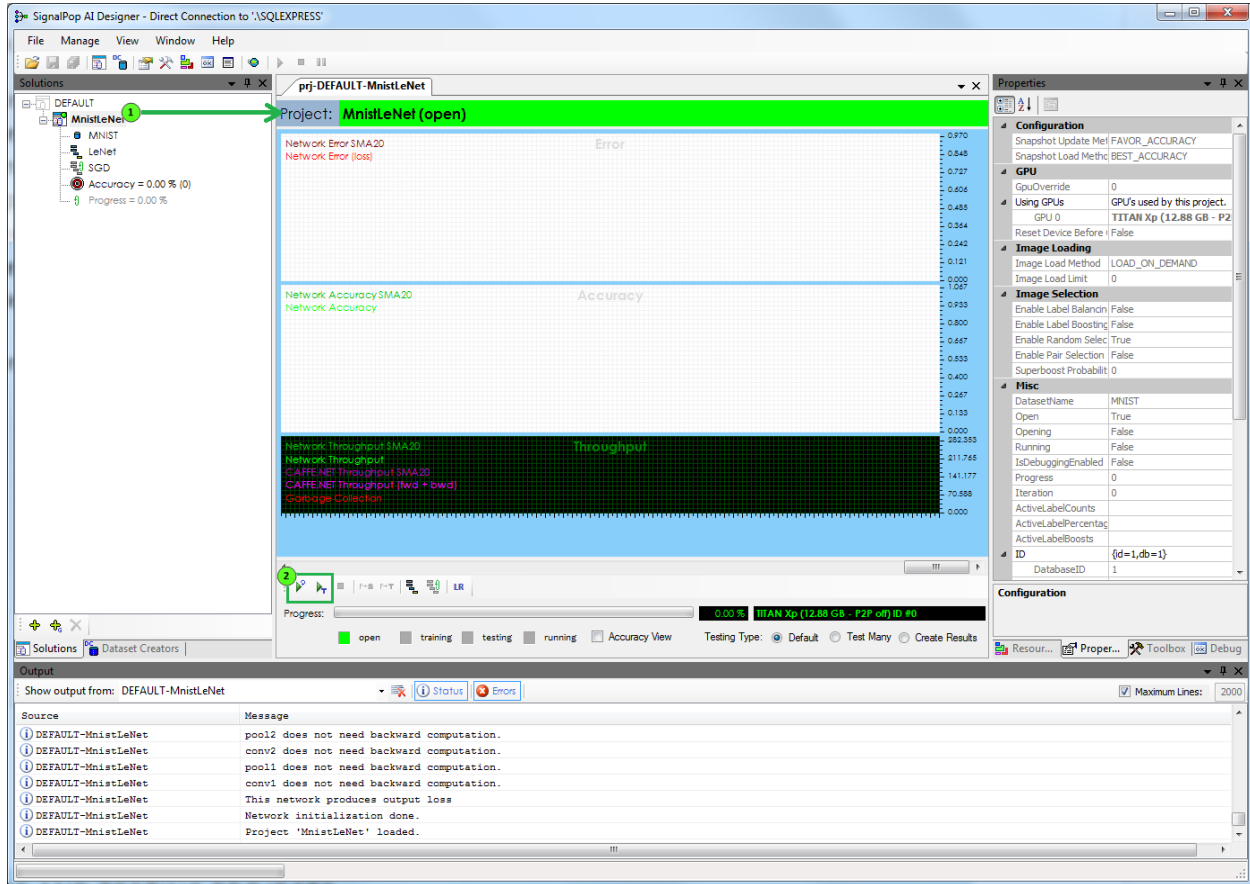


Figure 28 Training and Testing Projects

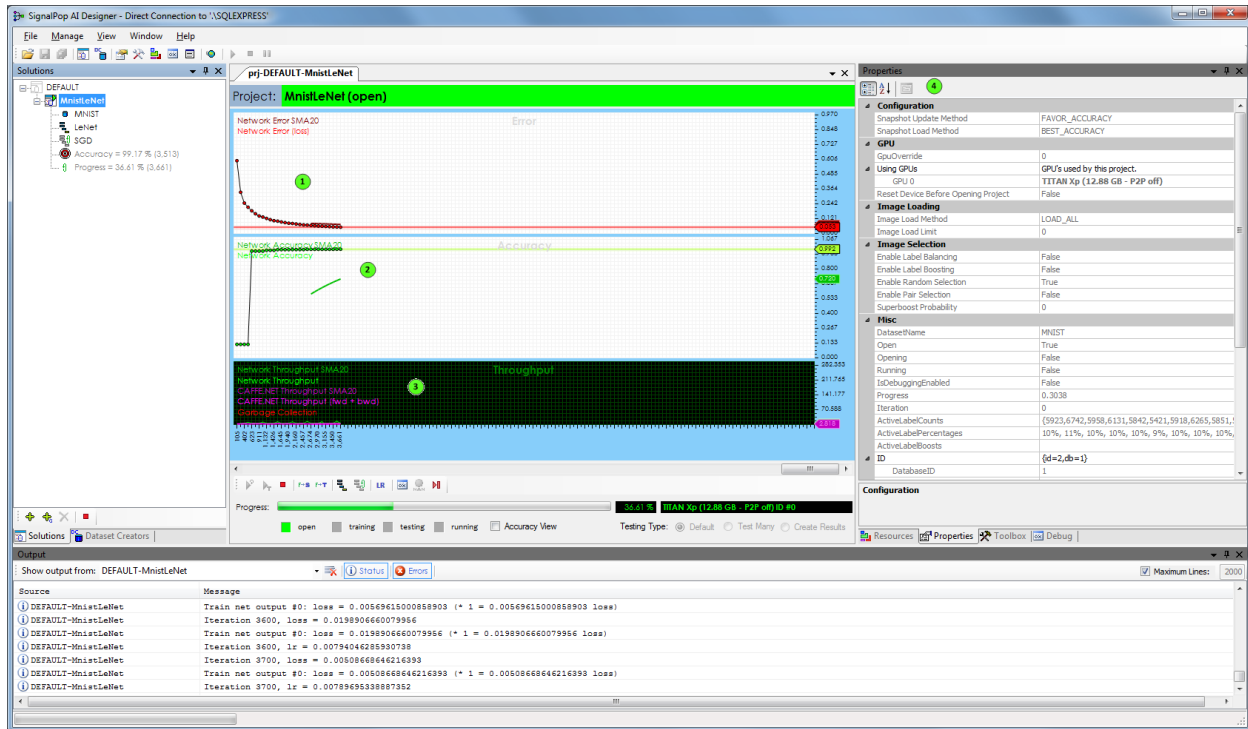
To train or test a project, do the following:

- 1.) Double-click the project name (of an open project) to open the *Project* window.
- 2.) At the bottom of the *Project* window select the *Run Training* (🟢) or *Run Testing* (🔵) buttons to start training or testing.

When testing, there are three types of testing that are available:

- **Default**; runs the default MyCaffe testing pass which also runs during training to calculate the accuracy of the model.
- **Test Many**; selects images at random from the testing set and runs them through the model in the same way any other outside image is run through the model and tallies up the results.
- **Create Results**; runs each of the *training* images through the model and saves the results in the database – this last type of training can be used to build higher level *bagged* results sets that combine the results of several models into higher level models.


During training, the results of the training process are shown in the graphical 'Project' window.





**Figure 29 Training a Project**


While running, the project window visually shows the progress of the training via the graph of the (1) error, (2) accuracy, calculated on each test pass, and (3) the network and MyCaffe throughput shown in milliseconds. In addition, the (4) 'Properties' pane shows the properties under which the training is taking place.


Several buttons at the bottom of the 'Project' window allow you to further control the training process:


 **Stop**; the stop button halts the training (or testing) process.


 **Force Snapshot**; the force snapshot button forces a snapshot at the current position of training. Normally, snapshots automatically occur based on the settings within the solver description, and when the accuracy increases.


 **Force Testing**; the force testing button forces a testing cycle to take place. Normally, the testing cycle frequency is determined by the settings within the solver description.


 **View Model**; the view model button opens the model editor for visual viewing of the model. Note, when a project is open, selecting the 'Inspect Layer' from several of the layers displays a visualization of the layer's internals (weights, etc.). See Model Editing for more information on the model editor.

 **View Solver**; the view solver button opens the solver editor for visual viewing of the solver.

 **Show Learning Rate**; the show learning rate button toggles between the 'Learning Rate' graph and 'Throughput' graph.

 **Blob Debugging**; the debugging button, when pressed, enables debugging by sending debug information to the 'Debug' windowpane. See the section on Debugging for more information.

 **Break on first NaN**; enabling the NaN breakpoint causes the model to immediately stop training upon detecting the first NaN or infinity value (e.g., model blow-up). This setting is only available when the 'Blob Debugging' is enabled. See the section on Debugging for more information.

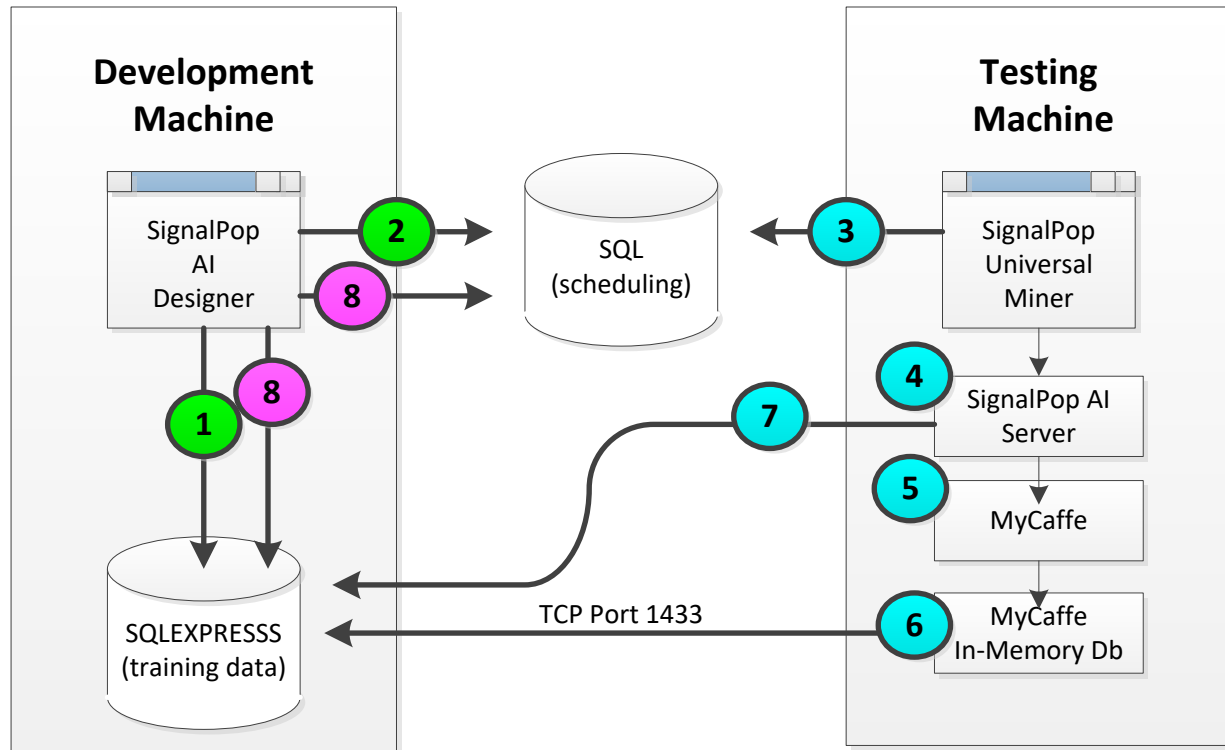
 **Enable Single Step**; enabling the single-step feature causes the model to run a single iteration and stop which can be helpful when debugging.

**Accuracy View**; checking the 'Accuracy View' toggles the scaling of the error and accuracy graphs between the error scale and accuracy scale as the accuracy values are only calculated on each test cycle.



## SCHEDULING PROJECTS

In some situations, the designer may want to schedule a project so that is available for training on a different machine. For example, say you have two development machines; one that you actively develop on and another that you use for testing – scheduling a project makes the project available for training by the SignalPop Universal Miner™ that runs on the test machine thus freeing up the GPU on your development machine for other tasks.



**Figure 30 Scheduling Projects**

When scheduling a project, the following steps take place.

- 1.) First the designer uses the SignalPop AI Designer™ on the Development Machine to create the dataset and the work-package data (model and solver descriptors) which are stored on the local copy of Microsoft SQLEXPRESS, running on the same machine as the SignalPop AI Designer application. To allow instances of MyCaffe running on the remote Test Machine to access the database, the designer must open access to port 1433 and create a database user account that has read-only access rights to the database on the local machine. Update access rights are only made available on the work package results table.
- 2.) Next, the designer schedules a project by adding a new work package to the work package scheduling database. The work package contains encrypted data describing the location of the dataset and work package data used by the remote Testing Machine to build the model and access the dataset during training.

- 3.) Next, when the SignalPop Universal Miner™ running on the remote Testing Machine notices the newly scheduled project, the scheduled project is assigned to the SignalPop Universal Miner software on the Testing Machine.
- 4.) Upon being assigned to the project, the SignalPop Universal Miner on the remote Testing Machine uses the SignalPop AI Server™ to load the work package data and use it to open and train the project.
- 5.) The SignalPop AI Server on the remote Testing Machine creates an instance of MyCaffe and loads the project into it.
- 6.) In addition, the SignalPop AI Server on the Testing Machine creates an instance of the MyCaffe In-Memory Database and sets the connection credentials to those specified in the scheduled work package thus allowing the MyCaffe In-Memory Database to access the dataset residing on the designer's Development Machine (or in some other location specified by the designer). This communication takes place over SQL port 1433.
- 7.) After the training of the model completes, the SignalPop Universal Miner™ software on the Testing Machine saves the weights and state back to the developer's Development Machine and then marks the project as completed in the scheduling database.
- 8.) Back on the designer's Development Machine, when the SignalPop AI Designer™ detects that the project is done, the project is displayed as completed with results. At this point the designer may copy the scheduled results from their local work packages results table into the projects local results residing on the local SQLEXPRESS instance used by the SignalPop AI Designer.

The example above shows how the designer can off-load the training to the testing machine thus freeing the designer to work with their local resources on other tasks.

## ADDING A PROJECT TO THE SCHEDULE

To schedule a project, right click on the project name within the *Solutions Window* and select the 'Schedule | Add' menu item, which will display the *Schedule Project* dialog.

**Schedule Project**

Project: **AlexNet**

**Task Type**

☒ Training ☐ Testing

**Target Resource**

☒ Any GPU ☐ 6 GB GPU ☐ 8 GB GPU ☐ 12 GB GPU ☐ 16 GB GPU ☐ 24 GB GPU ☐ 48 GB GPU

**Dataset Location**

☐ Local Machine ☒ Remote Machine

Server: MYMACHINE (Must have port 1433 accessible)

Database: DNN

Username: joe

Password: password

☐ Azure (public share)

**Work Package Data Location**

☐ Local Machine ☒ Remote Machine **same as dataset**

Server: MYMACHINE (Must have port 1433 accessible)

Database: DNN

Username: joe

Password: password

OK Cancel

**Figure 31 Scheduling a Project**

From this dialog you can select the type of task to run, the target resource to use and the dataset location information.

**Task Type**; currently only the *Training* task is supported which when run trains the scheduled project.

**Target Resource**; the target resource defines the type of GPU for which you would like to run the schedule project. Different GPU types can vary in price depending on the resources they have.

**Dataset Location;** the dataset location specifies all information needed by the remote SignalPop Universal Miner™ to connect to the dataset for which the model is to be trained. Note, all dataset location information is encrypted before sending or storing in the scheduling database.

**Server;** specifies the machine name or URL of the server where the database resides.

**Database;** specifies the name of the database, typically 'DNN'.

**Username;** specifies the SQL username that has read-only access to the database.

**Password;** specifies the SQL password for the SQL user.

**Work Package Data Location;** the work package data location specifies all information needed by the remote SignalPop Universal Miner™ to connect to the database containing the work package data (e.g., model descriptor and solver descriptor) and work package results. Note, all dataset location information is encrypted before sending or storing in the scheduling database. When loading a project, any existing weights and state within the work package results are loaded into the project. And once training is completed, or stopped, the current weights and state are saved back into the work package results table.

**Server;** specifies the machine name or URL of the server where the database resides.

**Database;** specifies the name of the database, typically 'DNN'.

**Username;** specifies the SQL username that has read-only access to the database.

**Password;** specifies the SQL password for the SQL user.

Select the '*same as dataset*' button to use the same connection settings for both the dataset and work package data.

---

## SETTING UP SECURE DATABASE ACCESS

Before the SignalPop Universal Miner™ can train a model from a different machine using the data residing in your local SQL (or SQL Express) database, you must give the SignalPop Universal Miner secure access to the database. This section describes how to do just that.

Setting up secure access involves two main steps: 1.) Setting up SQL and 2.) Setting up a SQL user that has read-only access rights to your training database.

---

## SETTING UP SQL

To setup SQL for remote access, you will need to take the following steps.

- 1.) First run the *Sql Server Configuration Manager* located at  
`c:\WINDOWS\SysWOW64\SQLServerManager14.msc`
- 2.) From within the SQL Server Configuration Manager, select and expand the '*SQL Server Network Configuration*' item, and then select the '*Protocols for MSSQLSERVER*' or '*Protocols for SQLEXPRESS*' depending on the type of instance you are using.
- 3.) Make sure the '*TCP/IP*' protocol is **enabled**.
- 4.) Next, double click on the '*TCP/IP*' protocol and select the '*IP Addresses*' tab. Make sure that the **TCP Port** setting is set to 1433 for each network for which you intend to access the database and make sure that the last **IPAll** setting has its **TCP Port** set to 1433.
- 5.) As a final step, re-start the SQL Server (or SQL Express) instance that you are using and **make sure that the SQL Server Browser is running and set to Automatic startup**. You may have to perform this latter step from the '*Services*' window that starts by entering 'Services' in the Windows Startup Search window.

Next, you will need to configure the Windows Firewall to open port 1433. The following steps show how to make this configuration.

- 1.) From the start search window (lower left side of the Windows-10 screen) enter 'Windows Firewall' and run 'Windows Defender Firewall'.
- 2.) From the Windows Defender Firewall, select the '*Advanced settings*'.
- 3.) Right click the '*Inbound Rules*' and select '*New Rule*'.
- 4.) For the *rule type* select '*Port*' and select the '*Next*' button.
- 5.) Select '*TCP*' as the port type and enter the '*Specific local ports:*' as **1433** and select the '*Next*' button.
- 6.) Select '*Allow the connection*' and select the '*Next*' button.
- 7.) Select how the rule is to apply (e.g., '*Private*' only for local use) and select the '*Next*' button.
- 8.) Give the rule a name such as 'SQL Server' and select the '*Finish*' button to complete adding the new rule.

You can now access SQL on port 1433 over the network for which you allowed the rule to apply (e.g., '*Private*' for your internal network).

Next, you will need to create a SQL user and give them read-only access to your database.

---

## SETTING UP SQL USER

To set up a SQL user, open the Microsoft SQL Server Management Studio and follow the steps below.

- 1.) To create a new user, select the 'Security | Logins' tab in the 'Object Explorer'.
- 2.) Right-click on the 'Logins' item and select 'New Login'.
- 3.) From the 'Login-New' dialog, enter the new 'Login name', select the 'SQL Server authentication' option and enter in the password for the new user.
- 4.) Press 'OK' to add the new user.
- 5.) Double click on the new user now listed under the 'Logins' item.
- 6.) From the 'Login Properties' dialog, select the 'Server Roles' item and grant access to both the 'public' and 'bulkadmin' Server roles.
- 7.) From the same dialog, select the 'User Mapping' item and check the 'DNN' database, and check the database role membership 'public' and 'db\_datareader'.
- 8.) Press 'OK' to accept the changes.
- 9.) Next, right click on the 'WorkPackageData' table and select the 'Permissions' tab. Grant INSERT, DELETE, and SELECT on this table for your new SQL user account.
- 10.) Next, right click on the 'WorkPackageResults' table and select the 'Permissions' tab. Grant INSERT, DELETE, and UPDATE on this table for your new SQL user account.
- 11.) Next, expand the 'Databases | DNN | Programmability | Stored Procedures' item and right click on the 'dbo.GetRawData' stored procedure. **If this item does not appear, you must install, run, and connect to the database with the SignalPop AI Designer.**
- 12.) From the 'Stored Procedure Properties' dialog select the 'Permissions' item and search for and select the new user that you previously added, and grant 'Execute' permission.

**IMPORTANT:** For remote users to access your database securely, you must enable both SQL Server and Windows Authentication mode so that they can access the DNN database. Using the Microsoft SQL Server Management Studio, right click on your database server and select 'Properties'. To enable this, from the 'Server Properties' dialog, select the 'Security' tab and make sure to check the 'SQL Server and Windows Authentication mode' radio button. To test your new user's access rights, connect to your database server using the SQL Server Management Studio and your new user's name and password.

Your database and machine are now ready to allow the SignalPop Universal Miner to train your local projects on a remote machine.

For a simple example, we will use the new username 'joe' with the password 'new\*test99' who was added to your local machine named 'TEST'. To schedule a project in a way that allows a remote Signal Universal Miner to train the project, you will want to use the following scheduling settings.

The 'Schedule Project' dialog box is shown with the following settings:

- Project:** AlexNet
- Task Type:** Training (selected), Testing
- Target Resource:** Any GPU (selected), 6 GB GPU, 8 GB GPU, 12 GB GPU, 16 GB GPU, 24 GB GPU, 48 GB GPU
- Dataset Location:** Remote Machine (selected)
  - Server: TEST (Must have port 1433 accessible)
  - Database: DNN
  - Username: joe
  - Password: new\*test99
- Work Package Data Location:** Remote Machine (selected)
  - Server: TEST (Must have port 1433 accessible)
  - Database: DNN
  - Username: joe
  - Password: new\*test99
  - A button labeled 'same as dataset' is visible next to the Remote Machine selection.

Buttons: OK, Cancel

**Figure 32 Sample Configuration Settings**

After pressing OK, this information is encrypted and stored along with the project model descriptor, solver descriptor and trained weights (if any exists) in the scheduling database. When an external instance of the SignalPop Universal Miner is assigned the project, it decrypts the dataset location information and starts training the project with MyCaffe and the MyCaffe In-Memory database. During training, the MyCaffe In-Memory database loads the dataset information into memory from the location pointed to by the dataset location information that you configured when scheduling the project.

To learn more about the SignalPop Universal Miner™, see the [SignalPop Universal Miner Getting Started Guide](#).

---

## SETTING THE SCHEDULING DATABASE

The scheduling database is initially set to 'NONE' which disables the work package scheduling. To change this setting, select the 'File | Settings' menu from the SignalPop AI Designer and change the 'SchedulingDatabaseServer' setting under 'Scheduling'. To enable work package scheduling, enter the specific scheduling database server information as follows: 'server;database;username;password'. Before saving this information, it is encrypted.

For example, to connect to the server 'MYMACHINE' with database 'DNN' and database user name 'joe' and password 'password', you would enter 'MYMACHINE;DNN;joe;password'.

**NOTE:** The SQL user used to access the scheduling database, must be given 'Execute' privileges on both the 'QueryAndAssignWorkPackages' and 'DeleteWorkPackages' stored procedures.

In addition, the SQL user must be granted the following access on the tables below:

Table	Access Rights Needed
WorkPackages	INSERT, SELECT, UPDATE
WorkPackageParameters	INSERT, SELECT, UPDATE
WorkPackageCommands	INSERT, SELECT, UPDATE, DELETE

**IMPORTANT:** When setting a machine up with the Scheduling Database, your machine must be visible to other machines. For example, to make the machine visible on your private network, you will need to set your 'Network Profile' to 'Private'. To do this, set the connection properties of your internet connection by selecting the 'Windows Settings | Network & Internet' settings. Next, select the 'Change connection properties' link and change your 'Network profile' to 'Private'.

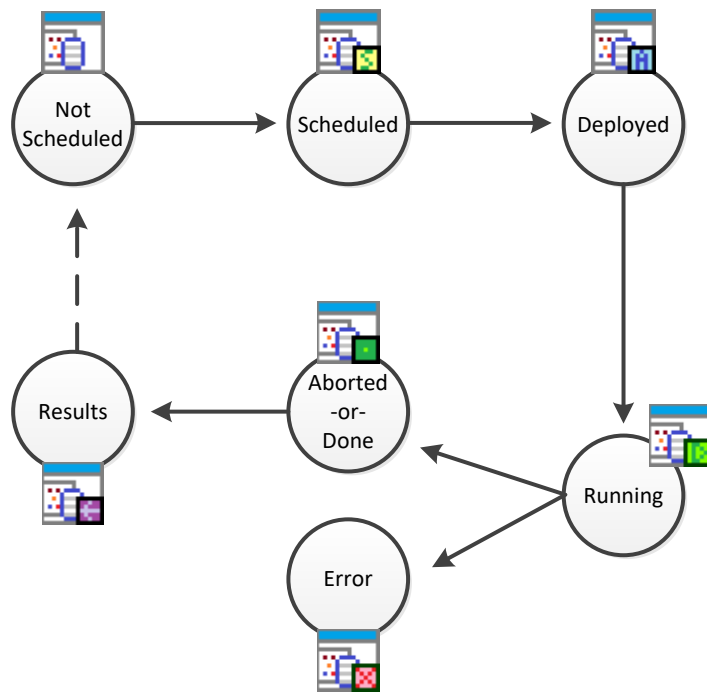
See the previous section on setting up secure database access for more details.



---




## SCHEDULING STATES





When using scheduled project, the state of the scheduled project is reflected in the project icon and follow the state transitions as shown below.



**Figure 33 Scheduling State Transitions**

Once scheduled, a project displays one of several special icons as described below.

Icon	Meaning
	Unscheduled project
	Scheduled project – the project is in the database ready to be assigned.
	Scheduled and assigned project – the project has been assigned to a miner.

	Scheduled and assigned running project – the project has been assigned to a miner and the miner is running the project.
	Scheduled and assigned project in an error state – the project has been assigned to a miner; the miner attempted to run the project, but an error occurred.
	Scheduled project is done (or aborted) and has no results pending.
	Scheduled project is done and has results pending.

After a scheduled project is completed successfully, the results are stored in the scheduling database where the designer can opt to copy the results over to their local database. The following project icon indicates that schedules results are ready to copy.



To copy the results over to the local database (replacing the existing local results), select the '*Schedule | Copy to Local*' menu item. After the results are copied, the scheduled project is deleted, and the project icon changes back to the normal project icon.



## IMPORTING AND EXPORTING

When working with AI models you may want to share your models with others and/or use models trained by others. Importing and exporting helps you do just that. There are several ways to import and export models and model descriptions that are discussed in the following sections.

### IMPORTING A PROJECT

Importing a project involves creating a new project from an existing Solver Descriptor, Model Descriptor and Model weight file. To import a new project, select the 'File | Import' menu item which displays the 'Import Project' dialog show below.

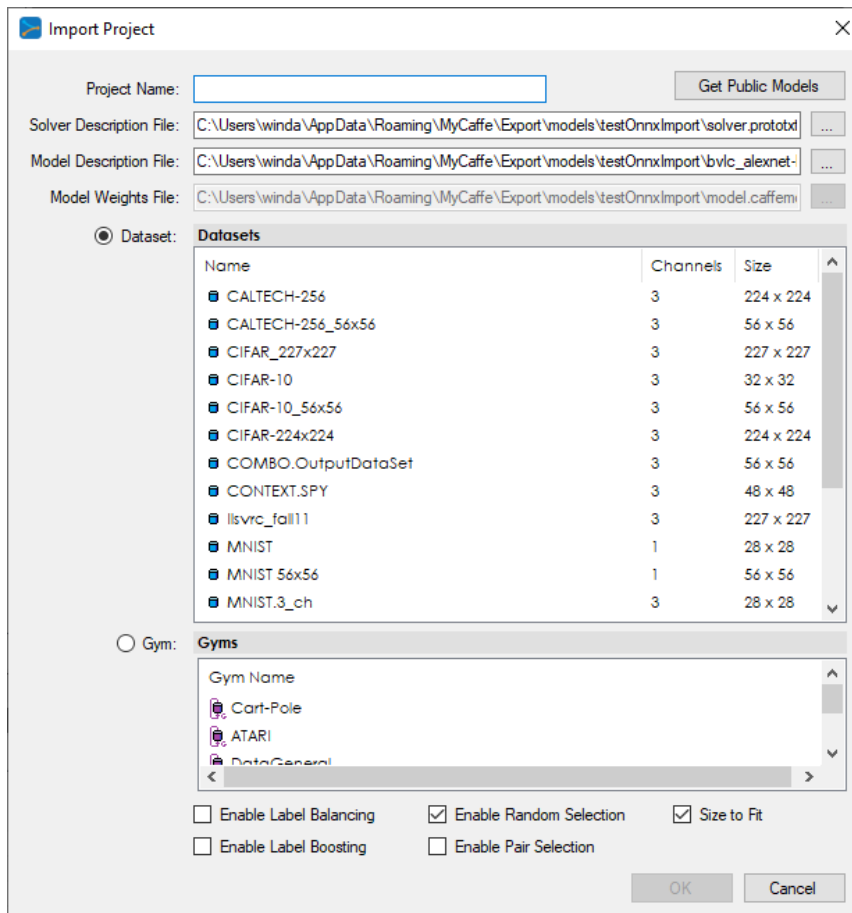


Figure 34 Import Project Dialog

From this dialog, enter the new project name and select the prototxt files for both the Solver and Model where the prototxt files use the native Caffe format. In addition, you can select a weight file that is either a 'mycaffemodel' or 'caffemodel' file format. And finally, select the Dataset to use with your project and add the new project.

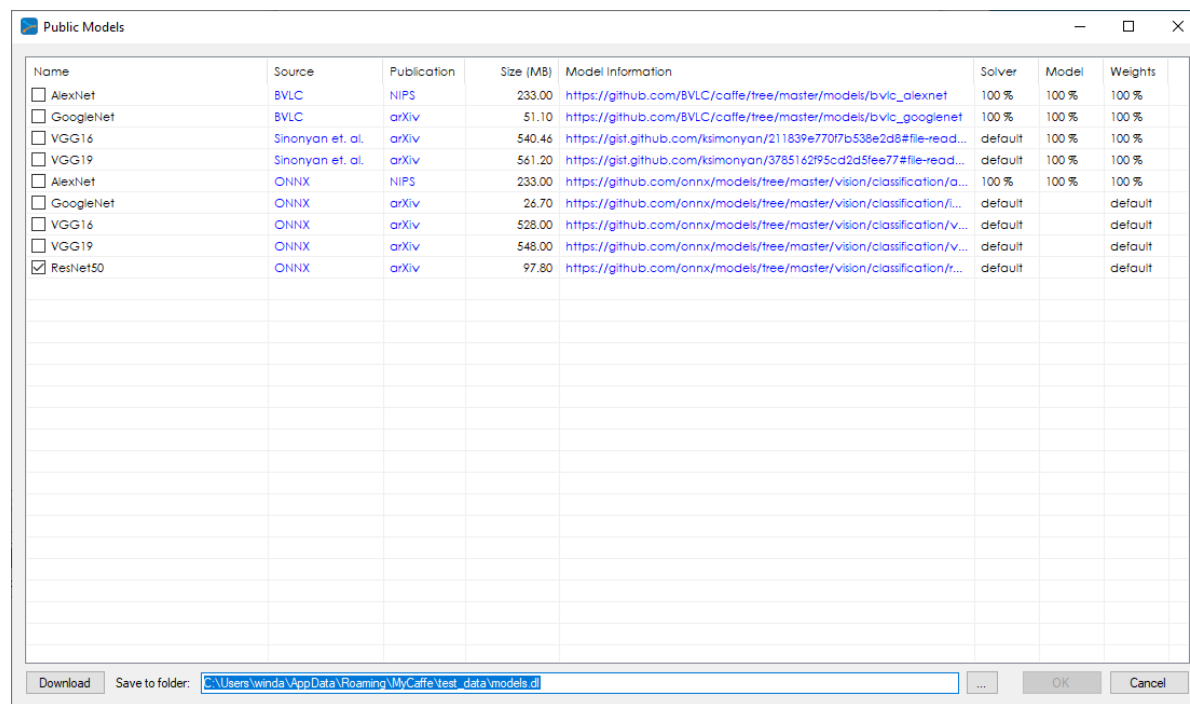
**Note:** To import an ONNX (\*.onnx) model file, just enter the \*.onnx file name in the 'Model Description Field' and it will be automatically converted into a model description and weights file that are then imported into your project.

**Note:** When importing a project, the following model fields are changed on the imported model:

```
transform_param
{
    crop_size: if greater than image size, changed to image size, otherwise unchanged.
    mean_file: if exists, removed but use_imagedb_mean added and set to True.
    use_imagedb_mean: True (if mean_file exists with a value).
    color_order: BGR (if a caffemodel is imported), RGB otherwise.
}
data_param
{
    source: set to dataset source name for TRAIN or TEST phase.
    backend: changed to IMAGEDB.
}
```

## IMPORTING PRE-TRAINED MODELS

To import already pre-trained models, select the 'Get Public Models' button in the upper right corner of the *Import Project* dialog.



**Figure 35** Public Models Dialog

From this dialog, select the public model that you would like to import, download it by selecting the 'Download' button and then select OK.

**Note:** When importing ONNX models, the downloaded \*.onnx file is automatically converted into a model description file and weight file that are then imported into your project.

## IMPORTING WEIGHTS

After opening a project, you can import the weights (or a portion of the weights) of an already trained model that matches (or closely) matches the project.

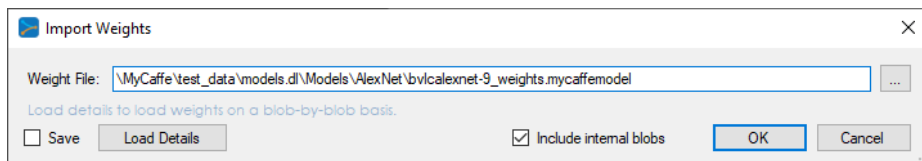


Figure 36 Import Weights Dialog

To import new weights into a project, do the following:

- 1.) Open the project.
- 2.) Right click on the 'Accuracy' (🎯) sub-tree item and select the 'Import' menu item which displays the 'Import Weights' dialog shown above.
- 3.) If you want to save the imported weights (like a snapshot), select the 'Save' check box. Otherwise, when unselected, the weights are just loaded into memory.
- 4.) Also, if you want to import on a blob-by-blob basis, select the 'Load Details' button which expands the dialog and allows you to select each blob to import. The blobs imported must match the blob size for which they are imported. By default, all matching blobs are imported.
- 5.) After selecting OK, the weights are imported into the open project.

**IMPORTANT:** Native Caffe uses a 'B'lue-'G'reen-'R'ed (BGR) color ordering whereas most images use the 'R'ed-'G'reen-'B'lue (RGB) ordering. When importing native Caffe models, you will want to make sure that each *DataLayer* within your model is set to use the BGR color ordering as shown below.

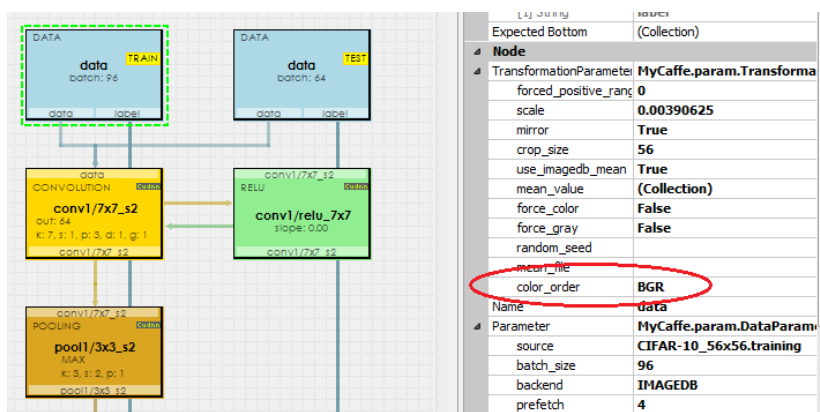


Figure 37 Setting BGR Color Ordering

To set the BGR ordering, do the following:

- 1.) Close your project and open the model in the model editor.
- 2.) Select each DATA layer as shown above.
- 3.) Change the 'color\_order' property to BGR for all DATA layers and save the project.

## IMPORTING WEIGHTS FOR TRANSFER LEARNING

In some cases, you may only want to import only a portion of a trained model to accommodate for different sizing or to only import initial layers.

To import only a portion of a trained model (e.g., for Transfer Learning), select the 'Show Details' button and the weights for the file to be imported and for the target model are displayed side-by-side.

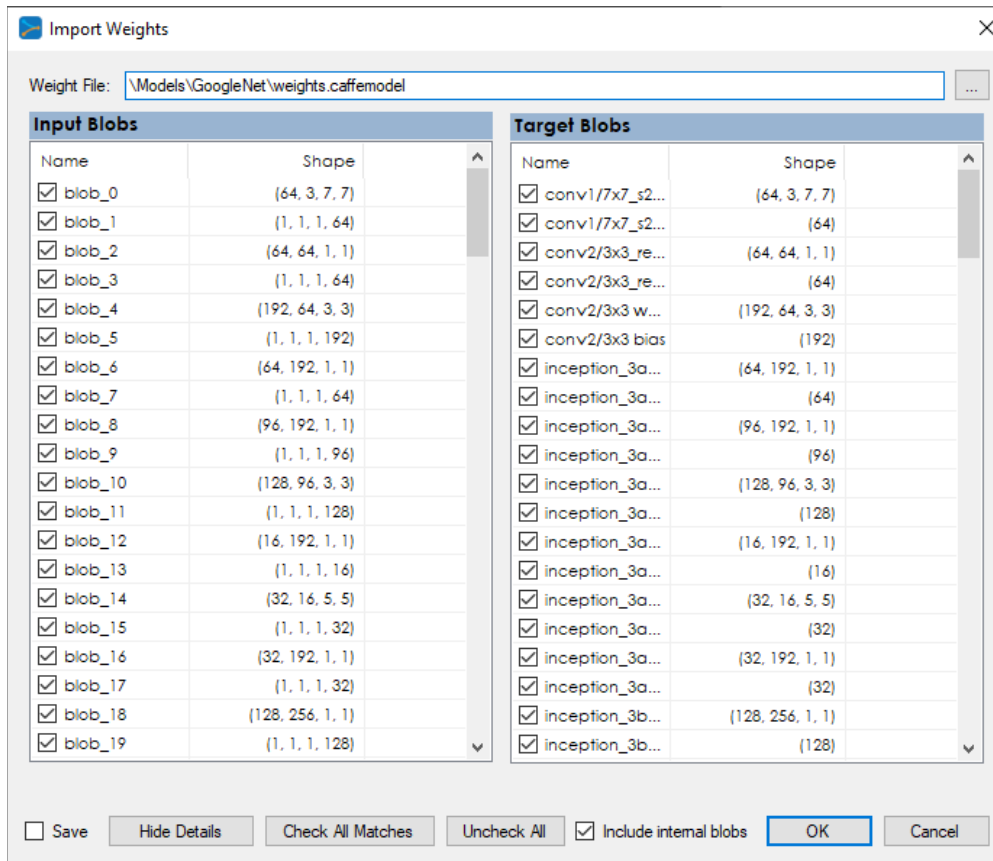


Figure 38 Weight Details Dialog

The blob sizes are automatically selected when the sizes match. Check the blobs that you want to import and select OK to add them to your model.

## EXPORTING PROJECTS

You can also export project descriptors and weights. Weights are exported in the 'mycaffemodel' file format which uses the same Google ProtoBuf binary file format and organization as that of native Caffe. The main difference between the 'mycaffemodel' file format and that of the 'caffemodel' format is that MyCaffe adds extra MyCaffe specific data *after* all the native Caffe data.

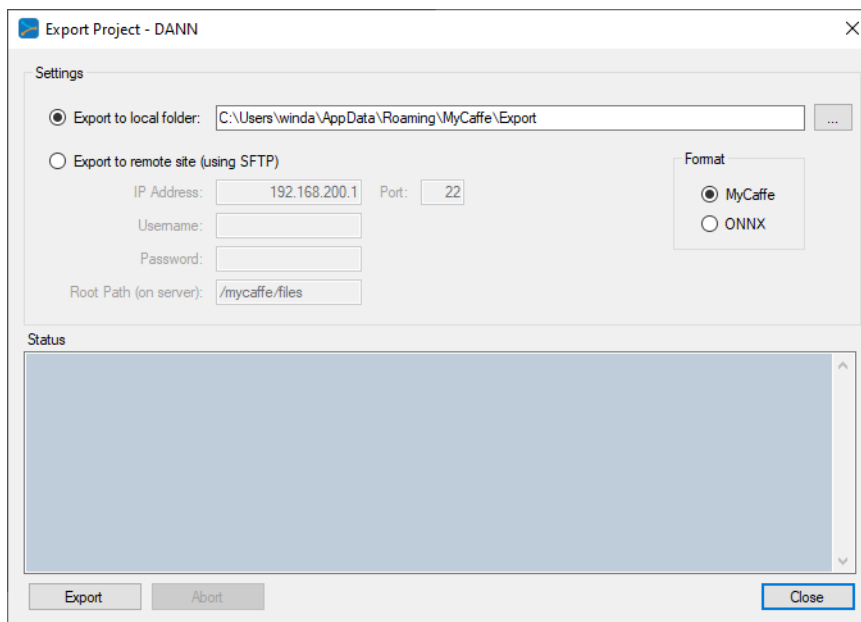
To export descriptors for the solver or model, do the following:

- 1.) From the 'Solutions' pane, right click the Model (📁) sub-tree item or the Solver (🔧) sub-tree item and select the 'Export' menu item.
- 2.) When exporting a Solver, select the 'Export | Solver' menu item.
- 3.) When exporting a Model, you can export either the train/test model by selecting the 'Export | Train/Test Model' menu item, or a deploy model by selecting the 'Export | Deploy Model' menu item.

To export models, do the following:

- 1.) Right click the 'Accuracy' (🎯) sub-tree item and select the 'Export | Weights' menu item.

To export the entire project (e.g., model description, solver description and weights), right click on the project name itself and select the 'Export' menu item.



**Figure 39** Export Project Dialog

The 'Export Project' dialog allows you to export to either a folder or over SFTP to a remote target. In addition, projects can be exported in the MyCaffe or ONNX formats.

**Note**, when exporting in the ONNX format, only the model description and weights are exported into the resulting \*.onnx file.

---

## EXPORTING TO DOCKER

We now support exporting both projects and datasets to your Docker containers! This section walks you through the steps necessary to do the export.

---

### DOCKER SETUP

Before exporting to your Docker container, you will want to perform a few initial setup steps to configure a Docker volume that is shared between your native Caffe Docker Container and an SFTP server such as the one made available by [atmoz/sftp](#) (available on Docker Hub).

Once you have your SFTP server pulled and ready to go, you will want to run the following Docker commands to configure it.

First, we need to create a Docker volume that will be shared between the SFTP server and the native Caffe Docker container. The following command will create a new volume named 'mycaffe-vol'.

```
$ docker volume create mycaffe-vol
```

Next, we need to create the SFTP Docker container – the following commands will start it up.

```
$ Docker container run -v mycaffe-vol:/home/signalpop/mycaffe -p  
2222:22 -d atmoz/sftp signalpop:password:1001::mycaffe/files
```

What this command does is start the SFTP server running with the 'mycaffe-vol' volume mapped to the '/home/signalpop/mycaffe' directory within the SFTP server. In addition, the port 2222 is mapped to the server's port 22 (sftp port) and the user 'signalpop' is logged in with the password 'password'. Files are then uploaded to the '/home/signalpop/mycaffe/files' directory within the SFTP server.

You will want to change the password for security reasons.

**IMPORTANT:** Make sure to use the same username and password set on the SFTP server here when you export SignalPop projects and datasets to your Docker container. We will get to that later, but for now, remember this username, the password and the Port used.

Next, we need to start the native Caffe Docker container. The following command will start the native Caffe Docker container made available by [nvidia/caffe](#) (available on Docker Hub).



```
$ docker container run -it -v mycaffe-vol:/workspace/mycaffe  
nvidia/caffe
```

What this command does is start the '*nvidia/caffe*' image in interactive mode (*-it* option) with the volume '*mycaffe-vol*' mapped to the '*/workspace/mycaffe*' directory within the Caffe container.

When you enter the Caffe container and run the '*ls*' command from the '*/workspace*' directory, you will see a new directory called '*mycaffe*'. This is where all exported files will be placed as discussed below.

All datasets exported from the SignalPop AI Designer are placed into the directory '*/workspace/mycaffe/files/data*' where a subdirectory exists for each dataset exported. Each dataset has two more subdirectories, one for all training images and another for all test images. Currently, datasets are exported as PNG images and are exported along with a *\_filelist.txt* that contains a listing of pairings where each line in the file lists the file path followed by the label of the file. In addition, a *\_info.txt* file is exported that describes the image sizes (h x w x c) and the number of images in each set. Use the '*caffe-master/tools/convert\_imageset.cpp*' tool to convert the images into the LMDB database used by the native Caffe in your Caffe container.

All projects exported from the SignalPop AI Designer are placed into the directory '*/workspace/mycaffe/files/models*' where a subdirectory exists for each project exported. For each project exported, the trained weights are placed in the *\*.caffemodel* file, and the model and solver description files are placed in a '*train\_test.prototxt*' and '*solver.prototxt*' files respectively.

The file structure used by the SignalPop AI Designer on the native Caffe container is as follows:

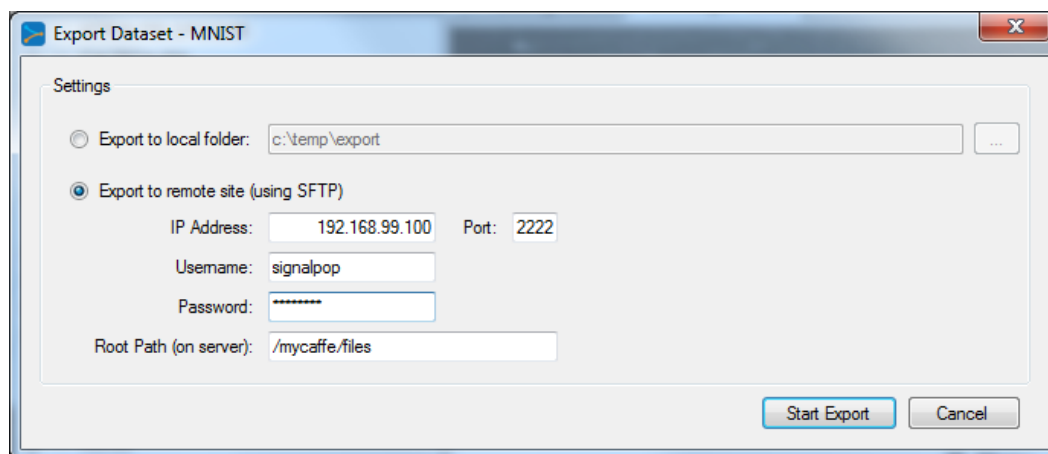
```
/workspace  
  /mycaffe  
    /files  
      /data  
        <your exported dataset name>  
          /train  
          /test  
      /models  
        <your exported project name>
```

Native Caffe model descriptors use the '*/workspace*' directory as the root for the sources referenced for training and testing. Each project exported by the SignalPop AI Designer, changes the training and testing sources to match this structure but also adds '*/lmdb*' to the end of each source within the model description, with the anticipation that the dataset will be converted into an LMDB database within this directory.

---

## EXPORTING DATASETS TO DOCKER

To export a dataset to your Docker container (setup as described above), first right-click on the dataset within the Dataset Creators window to display the 'Export Dataset' dialog.



**Figure 40** Export Dataset Dialog

Next, select the 'Export to remote site (using SFTP)' radio button and fill out the fields as follows:

**IP Address:** set this to the IP address where your SFTP Docker container is running.

**Port:** set the Port to the port that you mapped to port 22 above with the `-P` option when starting up the SFTP server.

**Username:** enter the username specified when starting the SFTP server.

**Password:** enter the password specified when starting the SFTP server.

**Root Path (on server):** for now, just leave this set to `/mycaffe/files` which is also set when starting up the SFTP server.

Next with your Docker SFTP server running, press the **Start Export** button and a new 'Dataset Creator' window will appear and show the status of the export. Once completed, your files will be on the native Caffe Docker container ready to use!

In the example above where we exported the MNIST dataset, you can find these files in the following directories on your native Caffe Docker container:

`/workspace/mycaffe/files/data/MNIST/train` – training mnist data files.

`/workspace/mycaffe/files/data/MNIST/test` – testing mnist data files.

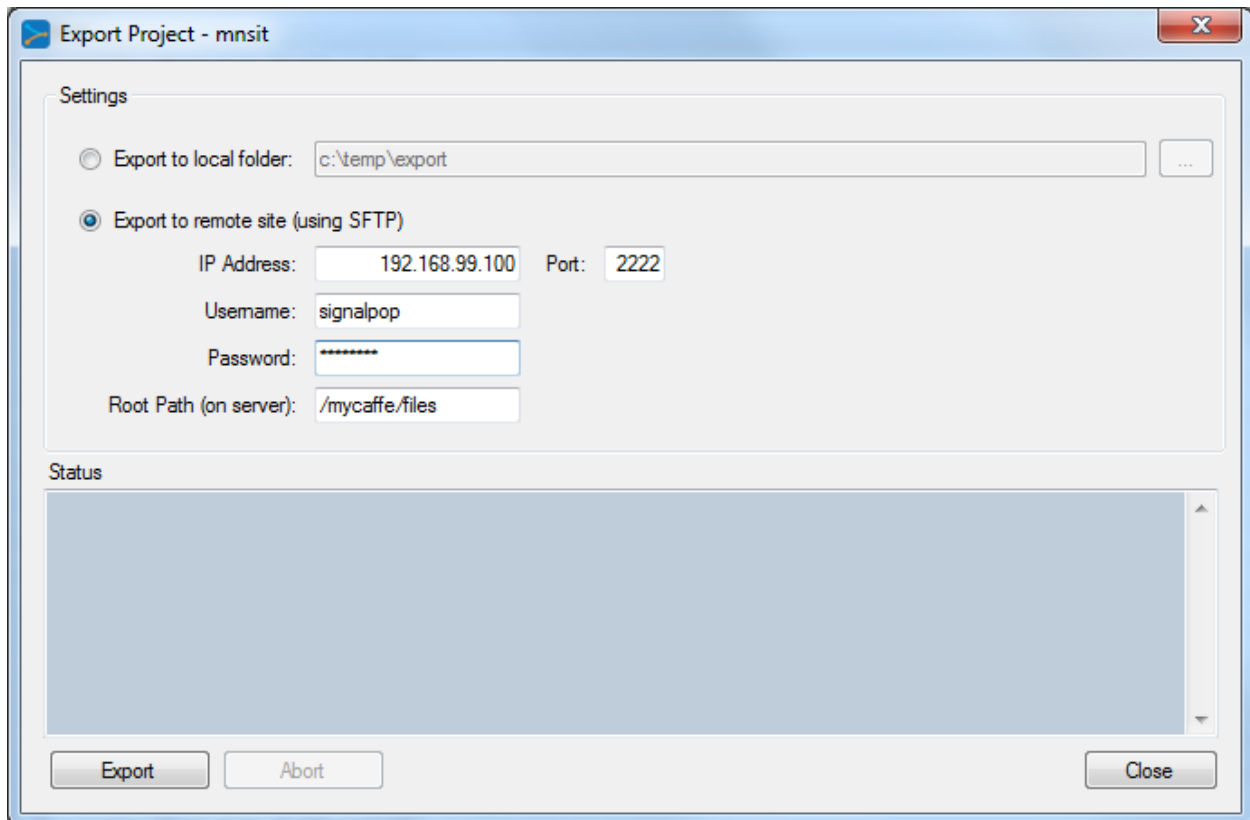
As mentioned above, to complete the process, run the `'caffe-master/tools/convert_imageset.cpp'` tool to convert each directory into an LMDB.

As a final note, when uploading a dataset, an image mean file named `'mean.png'` is also uploaded to the training directory.

---

## EXPORTING PROJECTS TO DOCKER

To export a project to your Docker container (setup as described above), go to the 'Solutions' window and right click on the **unopened** project that you want to export. Right click on the project to display the 'Export Project' dialog.



**Figure 41** Export Project Dialog

The steps are very similar to exporting a dataset. Again, you will select the radio button to 'Export to remote site (using SFTP)' and fill out the IP Address, Port, Username and Password. Next with your Docker SFTP server running, press the 'Export' button and the project is exported to your Docker container.

In the example above, the export places the files in the following directory on the Docker container:

*/workspace/mycaffe/files/models/MNIST*

Within this directory you will see the following three files:

*train\_test.prototxt* – this is your model descriptor file.

*solver.prototxt* – this is your solver descriptor file.

*model.caffemodel* – these are your trained weights, trained in MyCaffe, yet native Caffe compatible.

The '*train\_test.prototxt*' file has both train and test sources pointing to the '*mycaffe/files/data/MNIST/train/lmdb*' and '*mycaffe/files/data/MNIST/train/lmdb*' directories.

## CUSTOM TRAINERS

Custom trainers provide a way to train existing MyCaffe models in a recurrent and/or reinforcement manner. The MyCaffe Dual trainer offers recurrent and reinforcement training. When using a custom trainer, the training, testing, and running functions are performed by the custom trainer which then takes care of feeding data into the model based on the training method used. Typically, a Gym is used as the data source with a custom trainer.

Each custom trainer is set up through the properties of the solver.

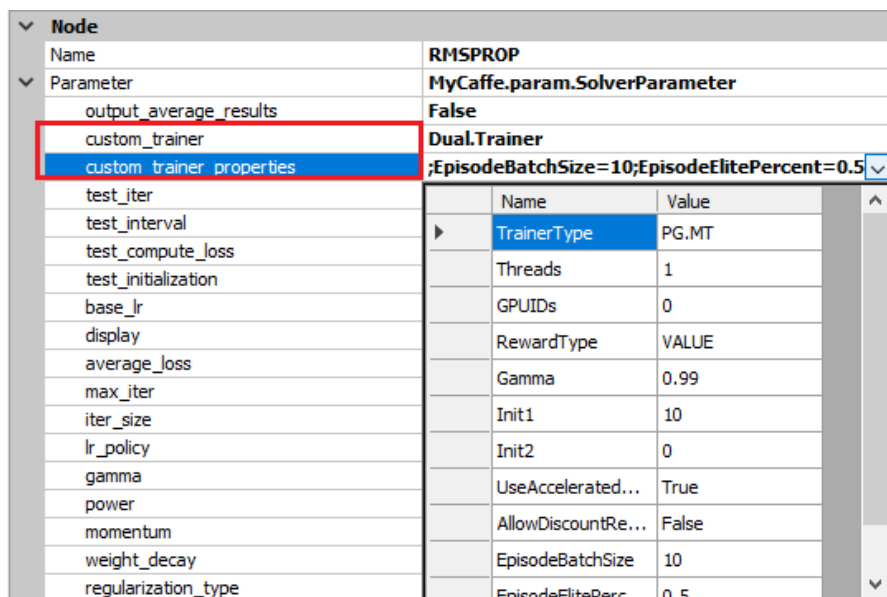


Figure 42 Custom Trainer Settings

The **Dual.Trainer** is a MyCaffe trainer that supports both recurrent and reinforcement learning. Depending on the trainer type used, the Dual.Trainer sets the MyCaffe model stage to either 'RNN' for recurrent learning or 'RL' for reinforcement learning. With these stages, each model can then activate each layer used during each stage.

---

## CUSTOM TRAINER SETTINGS - TRAINERS

Currently the following trainer types are supported:

<b>RNN.SIMPLE</b>	This trainer is used for recurrent training and when run sets the MyCaffe stage to 'RNN'.
<b>PG.MT</b>	This trainer provides a multi-threaded policy gradient reinforcement training and when run sets the MyCaffe stage to 'RL'.
<b>PG.ST</b>	This trainer provides a single threaded policy gradient reinforcement training and when run sets the MyCaffe stage to 'RL'.
<b>PG.SIMPLE</b>	This trainer provides a very simple policy gradient reinforcement training and when run test the MyCaffe stage to 'RL'.
<b>DQN.SIMPLE</b>	This trainer provides a simple DQN implementation derived from the Google Dopamine DQNAgent <sup>5</sup> licensed under the Apache 2.0 License.
<b>DQN.ST</b>	This trainer provides a single threaded DQN implementation derived from the Google Dopamine DQNAgent <sup>6</sup> licensed under the Apache 2.0 License.
<b>C51.ST</b>	This trainer provides a single threaded C51 (Rainbow) implementation derived from the Google Dopamine RainbowAgent <sup>7</sup> licensed under the Apache 2.0 License.

---

## CUSTOM TRAINER SETTINGS - PROPERTIES

Depending on the trainer used, there are various additional custom trainer settings that apply.

Property	Description
<b>TrainerType</b>	<i>RNN.SIMPLE</i> – use the RNN trainer with the RNN stage.  <i>PG.MT</i> – use the multi-threaded policy gradient reinforcement learning trainer with the RL stage.  <i>PG.ST</i> – use the single-threaded policy gradient reinforcement learning trainer with the RL stage.  <i>DQN.SIMPLE</i> – use the simple DQN reinforcement learning trainer with the RL stage.

---

<sup>5</sup> See [https://github.com/google/dopamine/blob/master/dopamine/agents/dqn/dqn\\_agent.py](https://github.com/google/dopamine/blob/master/dopamine/agents/dqn/dqn_agent.py)

<sup>6</sup> See [https://github.com/google/dopamine/blob/master/dopamine/agents/dqn/dqn\\_agent.py](https://github.com/google/dopamine/blob/master/dopamine/agents/dqn/dqn_agent.py)

<sup>7</sup> See [https://github.com/google/dopamine/blob/master/dopamine/agents/rainbow/rainbow\\_agent.py](https://github.com/google/dopamine/blob/master/dopamine/agents/rainbow/rainbow_agent.py)

	DQN.ST – use the single threaded DQN reinforcement learning trainer with the RL stage.
<b>Threads</b>	[PG.MT trainer only] Specifies the number of threads to run for multiple training sessions.
<b>GPUIDs</b>	[PG.MT trainer only] Specifies which GPU's to use per thread.
<b>RewardType</b>	<i>VAL (or VALUE)</i> – display the actual reward value as the reward.  <i>MAX</i> – display the maximum reward received.
<b>Gamma</b>	[PG.MT, PG.ST, DQN.SIMPLE, DQN.ST trainers only] Default Value = 0.99 Specifies the discount rate used for past rewards.
<b>MiniBatch</b>	[PG.MT, DNQ.SIMPLE, and DNQ.ST trainers only] Default Value = batch setting from project or 10 Specifies the override (if any) for the mini-batch which defines how often the accumulated gradients are applied. This setting is useful with Recurrent models that use a batch size of 1 for the <i>MiniBatchOverride</i> can then define how often the gradients are applied thus operating as though batching is in use.
<b>UseAcceleratedTraining</b>	[PG.MT, PG.ST, DQN.SIMPLE, DQN.ST trainers only] Default Value = True Specifies whether to use accelerated training. When enabled, gradient changes are applied twice to accelerate the areas where the gradient is changing. Accelerated training only applies when <i>MiniBatch</i> > 1
<b>AllowDiscountReset</b>	[PG.MT, PG.ST trainers only] Default Value = False Specifies whether to recent the discount values are reset during accumulation or accumulated.
<b>Preprocess</b>	Default Value = True Specifies whether to preprocess the data into 1 and 0 values.
<b>ActionForceGray</b>	Default Value = False Specifies whether to force the action values into a single channel of data.
<b>UseRawInput</b>	Default Value = True

	Specifies whether to use the input values directly (true) or to use the difference between the current value and the previous value (false).
<b>EpsStart</b>	[PG.MT trainer only] Specifies the starting exploration rate used to randomly select actions. For example, a value of 0.1 directs the trainer to randomly select the action 10% of the time.
<b>EpsEnd</b>	[PG.MT trainer only] Specifies the ending exploration rate.
<b>EpsSteps</b>	[PG.MT trainer only] Specifies the number of steps to use the exploration rate which starts at the EpsStart rate and decreased over the steps to the EpsEnd rate.
<b>SequenceLength</b>	[RNN.SIMPLE trainer only] Specifies the length of the data sequence fed into the recurrent model.
<b>Lookahead</b>	[RNN.SIMPLE trainer only] Specifies the length of sequence to look into the future. The full sequence length = SequenceLength + Lookahead.
<b>VocabularySize</b>	[RNN.SIMPLE trainer only] Specifies the number of vocabulary buckets to use. Data values are fit into a set of buckets that span a given data range. Character values are each fit into their own bucket whereas numeric values are fit into the bucket that has a matching value range. For example, when using numeric values, a value of 0.12 is placed in the bucket with data value range [0.0, 0.5].
<b>VocabularyMin</b>	[RNN.SIMPLE trainer only] Specifies the minimum value of the data range spanning all buckets. Each bucket fills the range between the VocabularyMin and VocabularyMax. For example, if the VocabularyMin = -1.0 and the VocabularyMax = 1.0 a VocabularySize = 4 will produce four buckets with the following data ranges: [-1.0, -0.5], [-0.5,0.0],[0.0,0.5],[0.5,1.0] Note, when data values exceed the end bucket ranges, the values are placed within those buckets. With the example above a value of -2 is placed in the first bucket whereas a value of 3 is placed in the last bucket.
<b>VocabularyMax</b>	[RNN.SIMPLE trainer only]

	Specifies the maximum value of the data range spanning all buckets.
<b>LastLabelOnly</b>	[RNN.SIMPLE trainer only] Specifies that only the last label is used in the sequence, instead of the entire past and future sequence.
<b>FrameSkip</b>	[ATARI Gym] Specifies the frame skip to apply, where 1 = no frame skip.
<b>AllowNegativeRewards</b>	[ATARI Gym] When true, negative rewards are returned after our player misses the ball.
<b>TerminateOnRallyEnd</b>	[ATARI Gym] Specifies to give a terminate state on the end of a rally as opposed to the end of a game.



## DEBUGGING

When developing models, often it is important to understand what is occurring within the model. This is especially true when your model blows up and you are trying to figure out why. The debugging tools of the SignalPop AI Designer are provided to help you better understand what your model is doing.

There are four types of debugging tools provided by the SignalPop AI Designer: Real-time, Visualizations, Evaluators and Hardware diagnostic tools. The following sections discuss each of these in more detail.

## REAL-TIME

Selecting the 'Blob Debugging' (🔍) button from a 'Project' window that is training a model, causes the data flowing through the model to also display in the 'Debug' window.

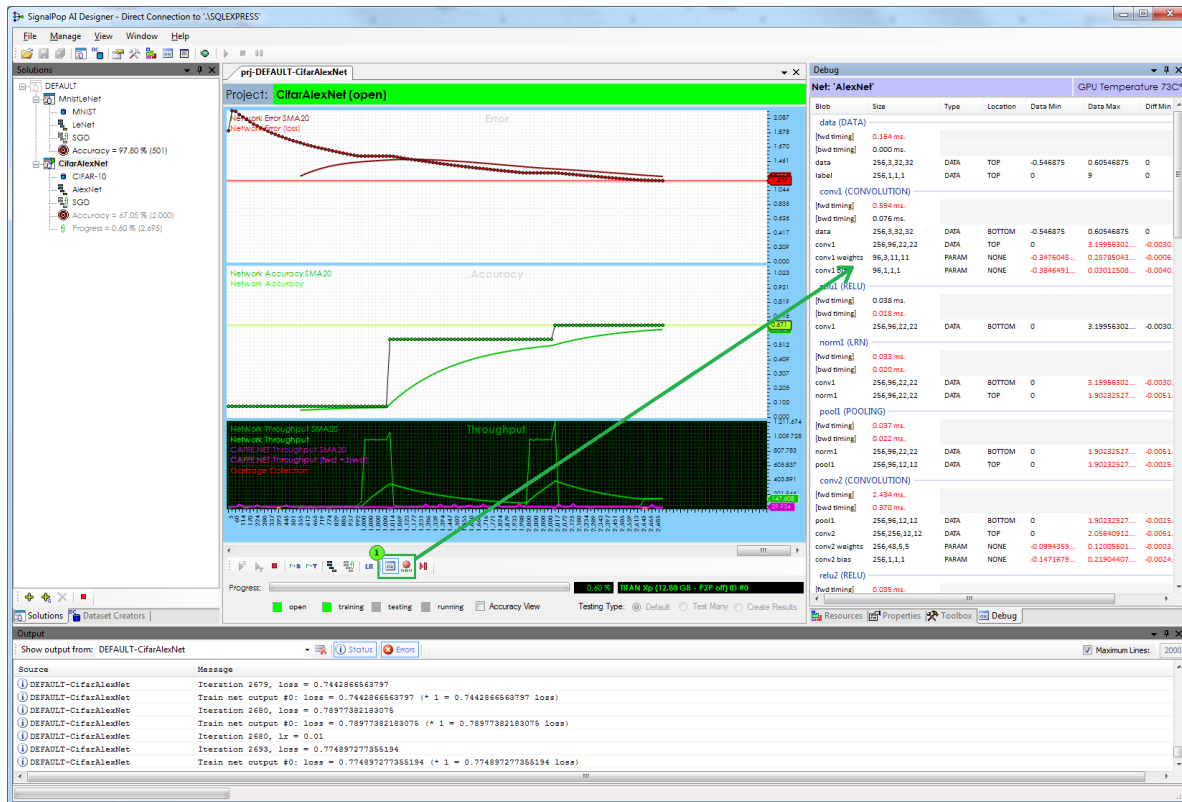


Figure 43 Real-time Debugging

To enable the real-time debugging to do the following:

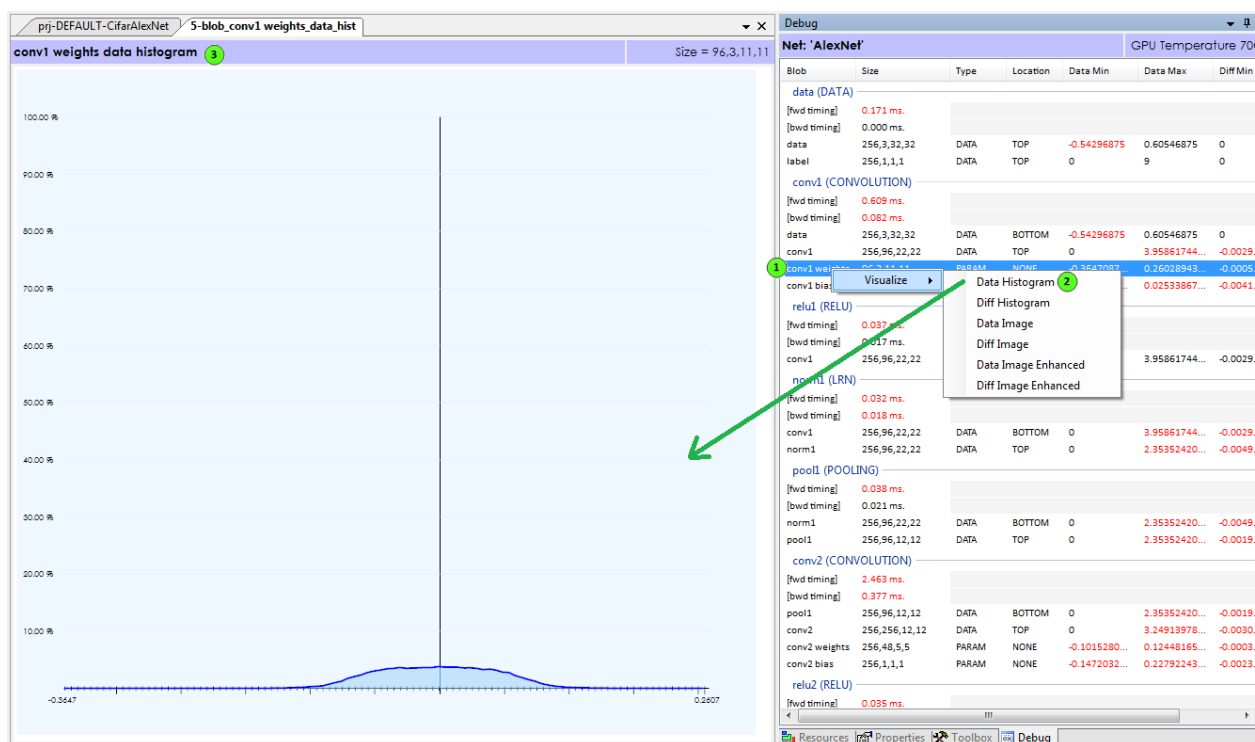
- 1.) Select the 'Blob Debugging' (🔍) button while training and the data values flowing through the network will also be displayed in the 'Debug' window as they flow through the network.

Note, selecting the 'Break on first NaN' (NaN) button causes the network to immediately stop training upon detecting the first NaN or Infinity value which means the model most likely blew up.

After stopping the training session, with the 'Debug' window still open, you can further analyze the Blob values of each layer by right clicking on each within the 'Debug' window. The debug window allows you to view each Blob's contents as a Histogram or as an Image for both the Data and Diff portions of the Blob.

## HISTOGRAM BLOB VISUALIZATION

The histogram Blob visualization shows the histogram created from the values within the Blob thus giving you a better idea of where the data values reside in the overall Min/Max data range of the Blob.



**Figure 44 Blob Histogram Visualization**

To view the Blob Histogram Visualization, do the following:

- 1.) Stop the training session with the 'Blob Debugging' enabled, and right click on the Blob that you want to visualize.
- 2.) Select the 'Data Histogram' visualization to view the Data portion of the Blob (selecting the Diff Histogram visualizes the Diff portion of the Blob).
- 3.) View the histogram of the Data portion of the Blob in the histogram window.

## IMAGE BLOB VISUALIZATION

The image Blob visualization shows the image created by converting each value within the blob to a coloring that makes up the Blob image. This type of information can tell you where data values drop out when driven to zero, etc.

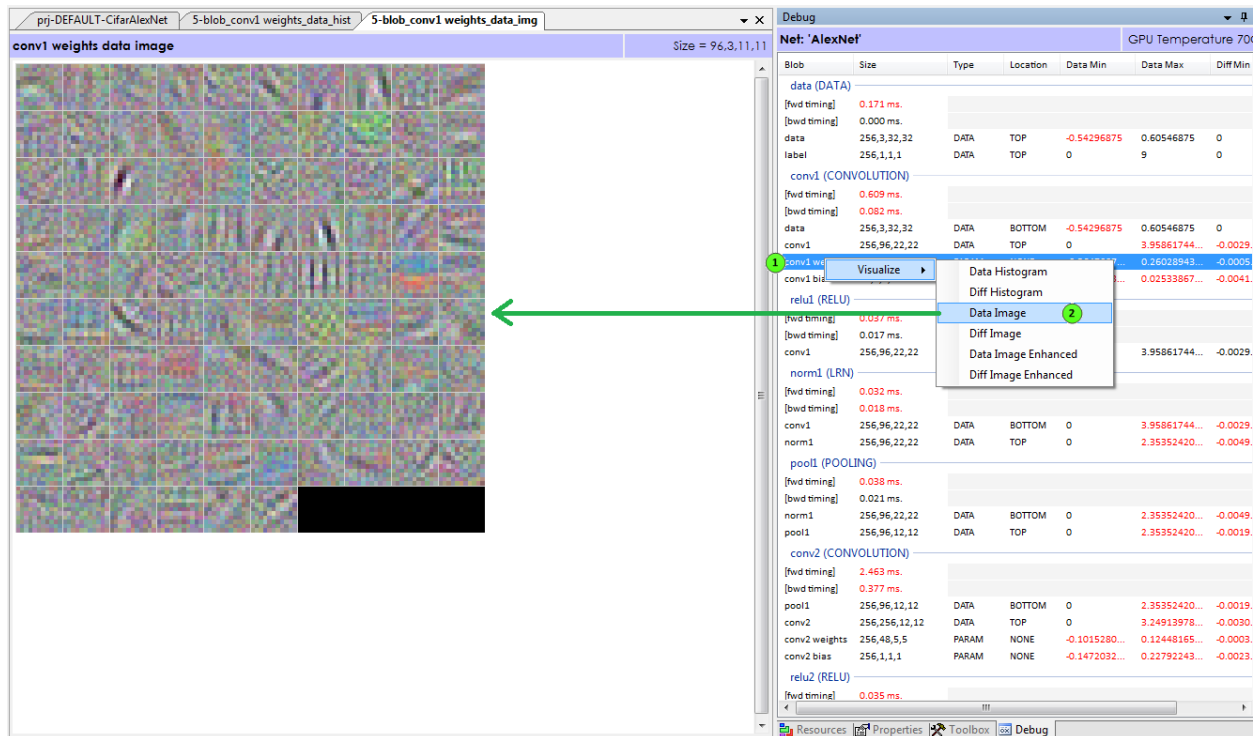


Figure 45 Blob Image Visualization

To view the Blob Image Visualization, do the following:

- 1.) Stop the training session with the 'Blob Debugging' enabled, and right click on the Blob that you want to visualize.
- 2.) Select the 'Data Image' visualization to view the Data portion of the Blob (selecting the Diff Image visualizes the Diff portion of the Blob).
- 3.) View the image of the Data portion of the Blob in the image window.

The enhanced image Blob visualization is like the image Blob visualization, but each color is enhanced to hopefully bring out more information that helps better understand the network internals.

## MODEL DEBUGGING

When developing a model, it is often helpful to visualize the actual data that flows from one layer to another. The Blob Data Debugging window allows you to do just that. To use the Blob Data debugging feature, first open your project as described in [Opening Projects](#). Next, open the model editor by double clicking on the model's name as described in the [Project Editing](#) section, above. At this point the project is read-only, but layers and links are now live and can be inspected.

## INSPECTING LINKS

To view the blob data that travels between layers, first select the link that you want to inspect by clicking on it. Selected links are highlighted in light green. Next, left mouse click and select the 'Inspect Link Data' to view the blob data, or select the 'Inspect Link Diff' to view the blob diff.

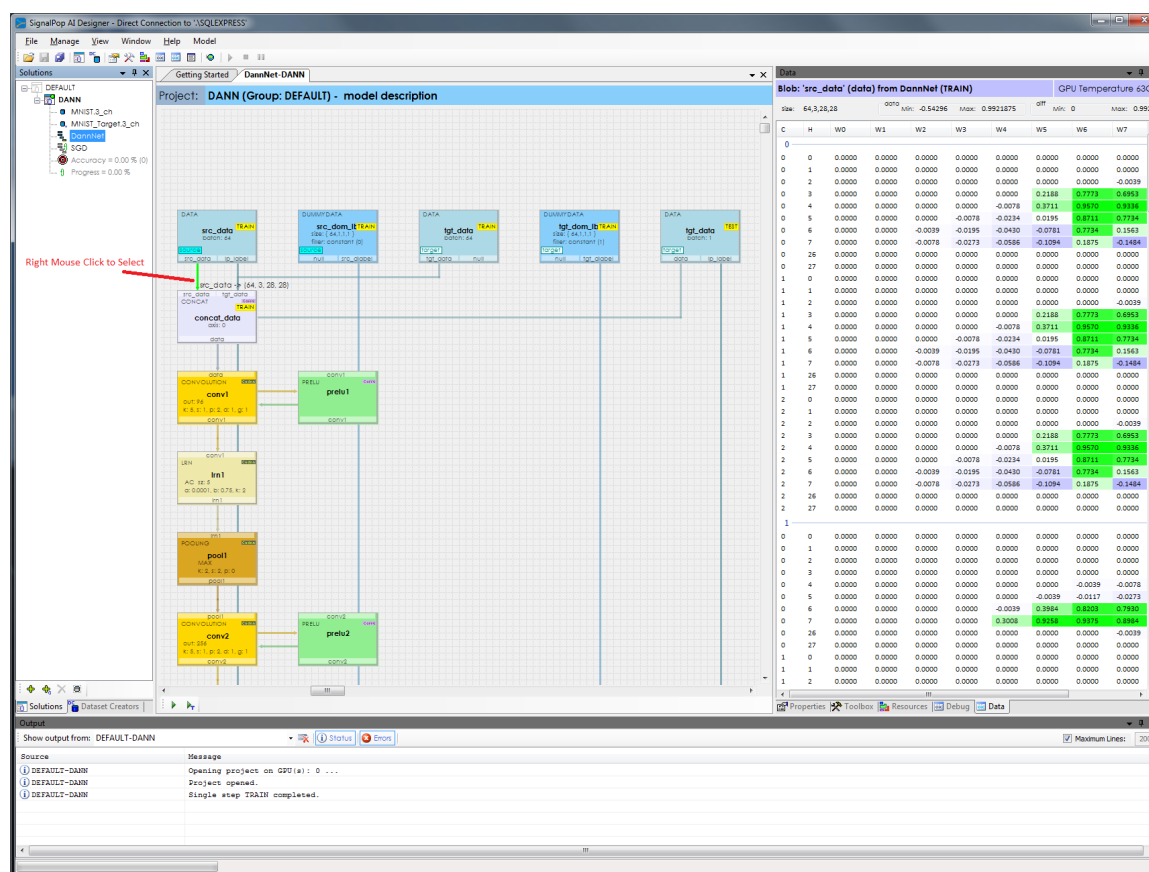


Figure 46 Blob Data Debugging

The actual blob data (or diff) contents are then displayed in the Data debugging window to the right of the editor.

When viewing the model of an open project, the 'Step Training' (▶) and 'Step Testing' (▶) buttons appear in the lower right section of the model editor – these buttons allow you to single step through the training or testing of the model so that you can then easily inspect a link or layer again to see the new impact of the step made.

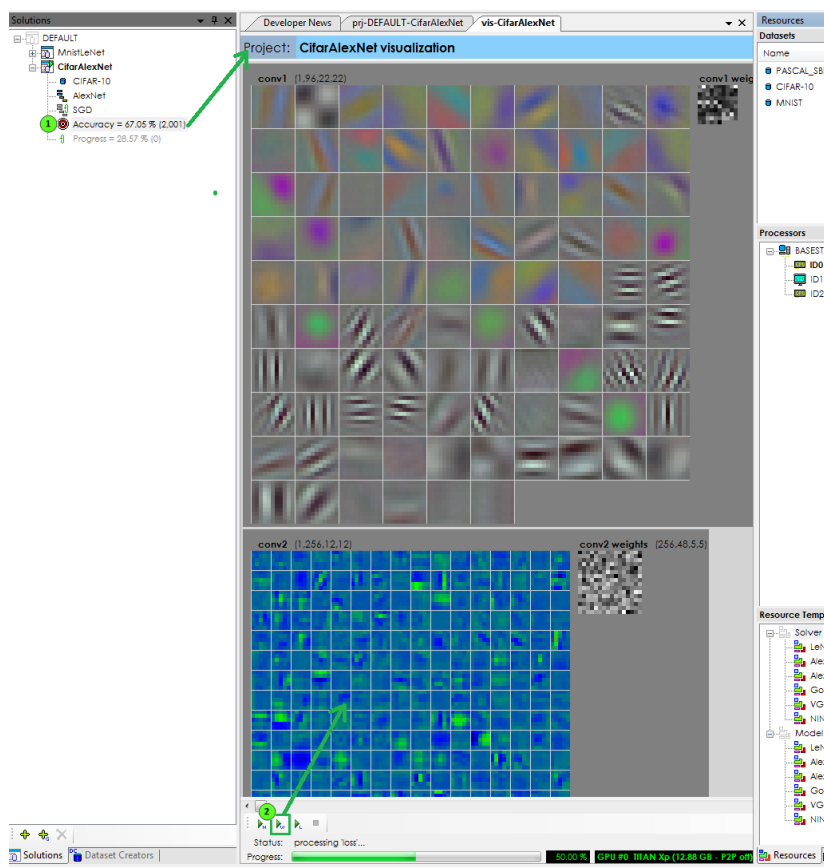


## VISUALIZATIONS

Visualizations allow you to see the insides of the network by viewing its data and weights and viewing the impact of the network on a given input to see which portions of the input have the highest impact on the detected label.

### WEIGHT VISUALIZATION

The weight visualization draws the weights (of each layer that has learnable parameters) as an image allowing you to visually see whether the network is learning features or not. For example, the 'conv1' layer of the ConvAlexNet model shows visually that it has learned patterns of edges and other simple features<sup>8</sup>.




**Figure 4.8 Visualizing the Network Weights**

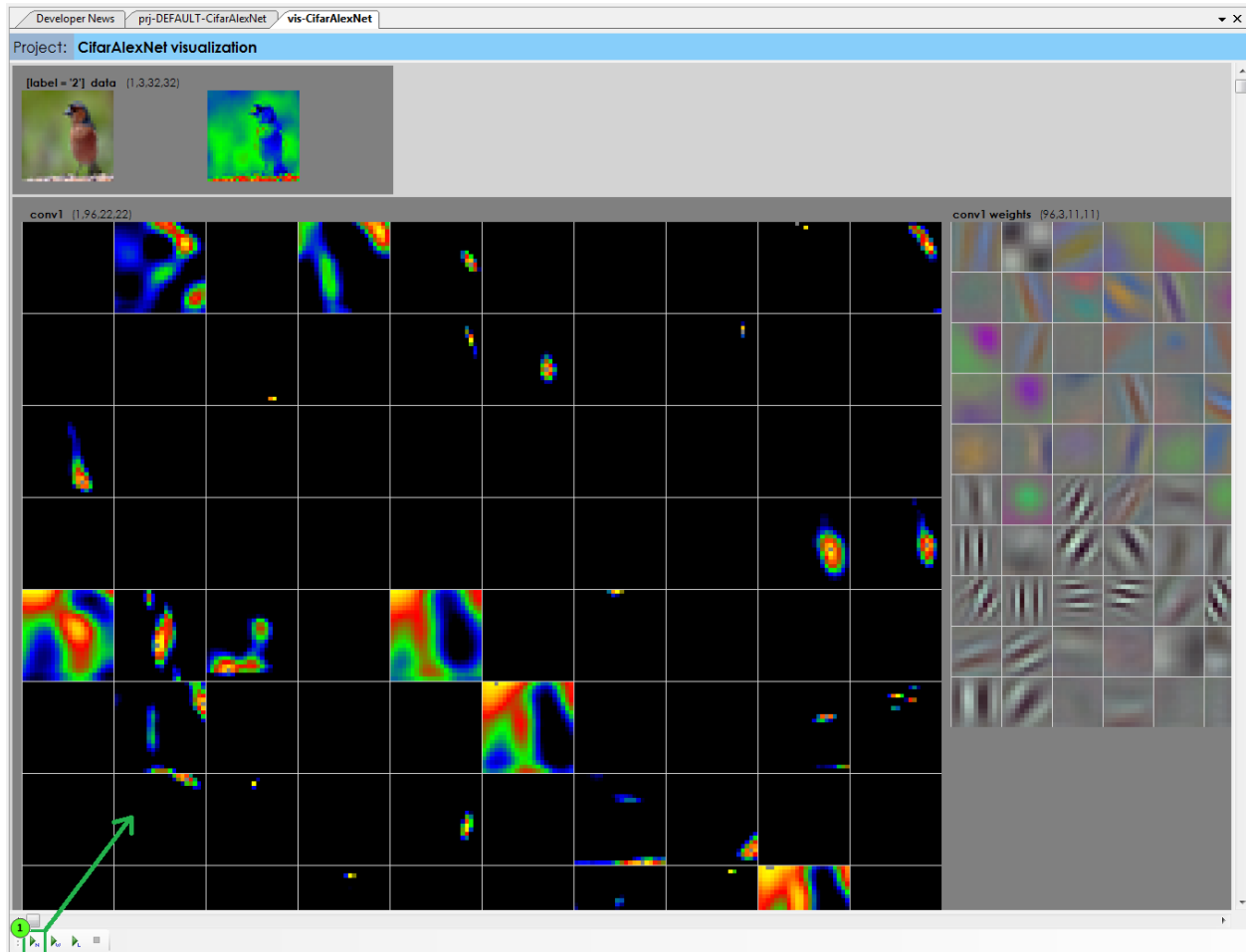
To view the Weight Visualization, do the following:

- 1.) Double click the 'Accuracy' project item from the 'Solutions' pane.
- 2.) Select the 'Run weights visualization' (🎮) button.

<sup>8</sup> The model shown was imported from the ImageNet AlexNet model trained by Berkeley and found on GitHub at [https://github.com/BVLC/caffe/tree/master/models/bvlc\\_alexnet](https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet).

## NETWORK VISUALIZATION

In addition to just visualizing the network weights, selecting the 'Run network visualization' (  ) button visualizes the entire network when running a single image.



**Figure 49 Visualizing the Network**

Visualizing the network shows the results of a given image when moved through the network along with the weights (as shown in the previous section).

With this view you can see which neurons fire given the various weight features discovered.

## LABEL IMPACT VISUALIZATION

The label impact visualizations take a different approach to visualization in that they attempt to show which areas of a given image trigger the detected label. When creating the label impact, we use an algorithm inspired by [11] where the network is run repeatedly on the same image where on each run a small window blocks out a portion of the image to see which portions of the image have a higher impact on the network end result.

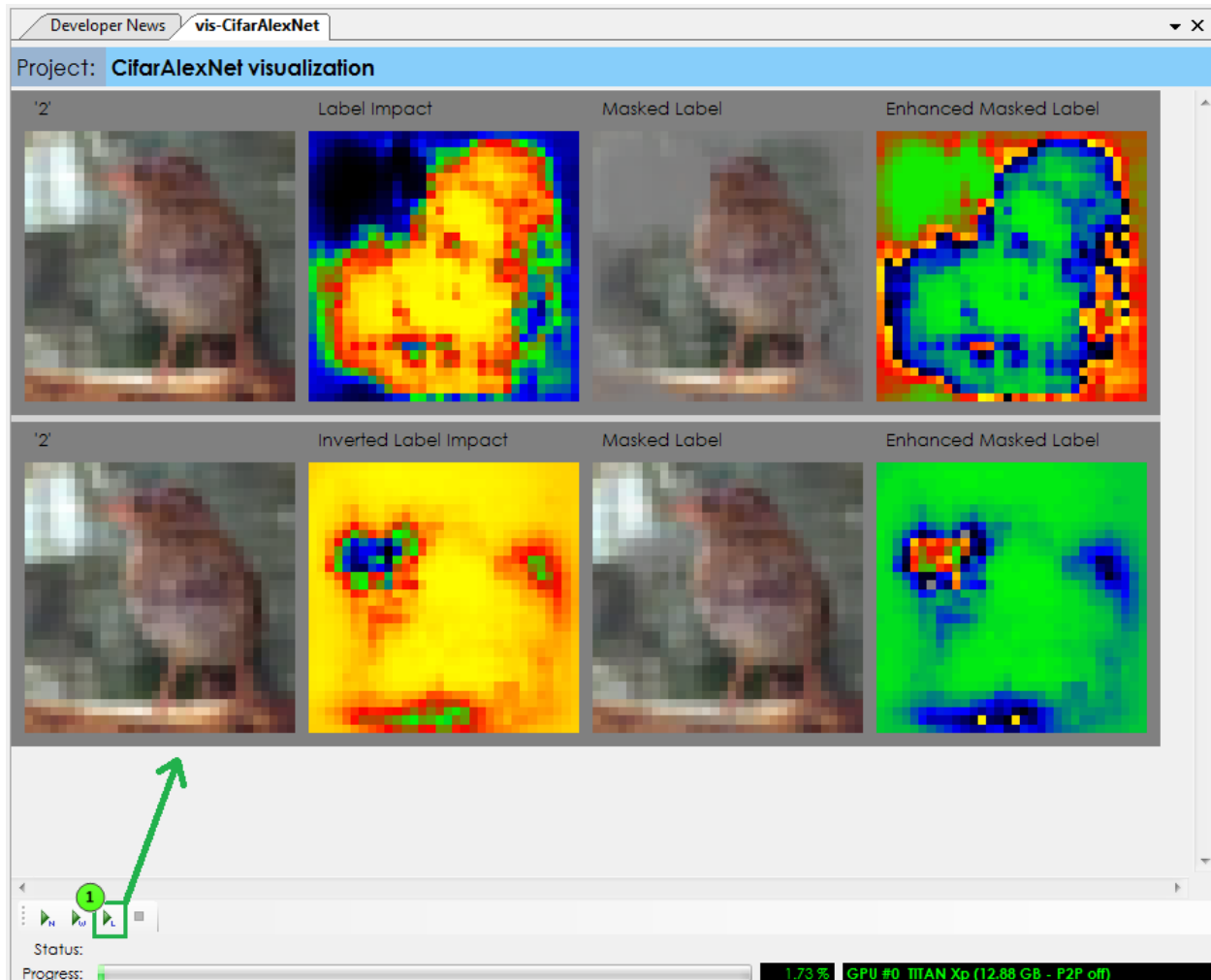


Figure 50 Labe Impact Visualization

In the first row of the label impact visualization, we block out a portion of the image with a small black window.

In the second row we take the opposite approach and run only a small portion of the image through the network with the remaining portion of the image set to back.

To run the label impact visualization, select the 'Run label visualization' (▶) button.



## EVALUATORS

Evaluators are plug-ins that give you different views of each network. Some are for debugging and some are for artistic fun.

### IMAGE EVALUATION

Image evaluation runs the network in reverse and displays the results for each layer thus allowing you to see where the network has difficulty learning. The image evaluation uses the deconvolution and unpooling algorithm inspired by [11].

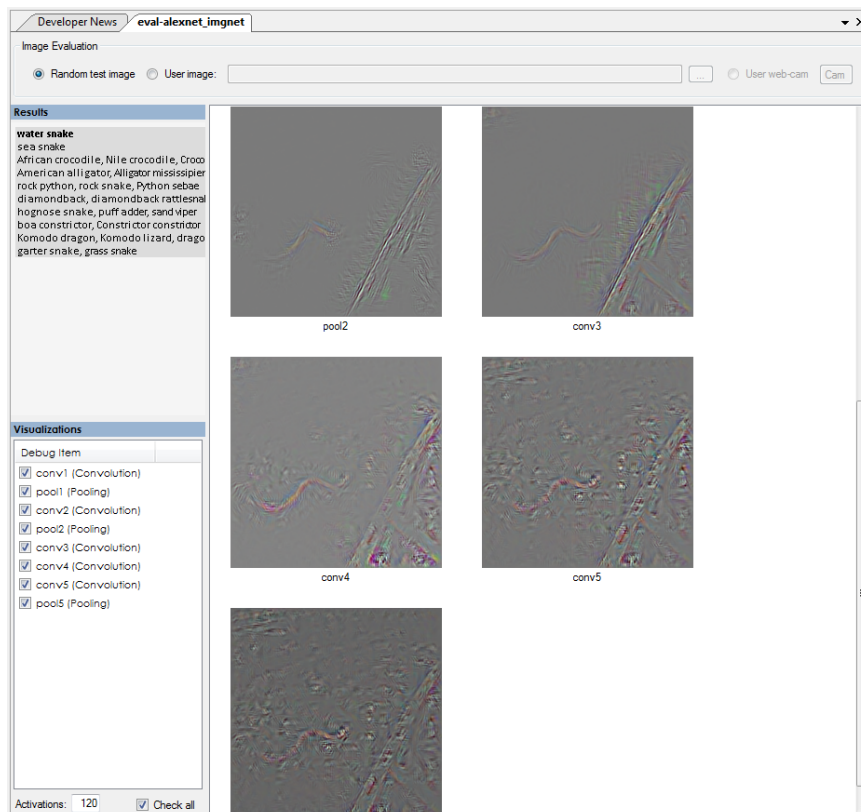


Figure 51 Image Evaluator

To run the image evaluator<sup>9</sup>, do the following:

- 1.) Right click an open project and select 'Evaluate'.
- 2.) Select the 'Image' Evaluator.
- 3.) Run the evaluator by pressing the 'Run' (▶) button.

**NOTE:** Image evaluation is not supported on projects that use a gym dataset.

<sup>9</sup> The model shown above was imported from the ImageNet AlexNet model trained by Berkeley and found on GitHub at [https://github.com/BVLC/caffe/tree/master/models/bvlc\\_alexnet](https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet)

## DREAM EVALUATION

The 'Dream' evaluator runs the Deep Dream [12] algorithm which allows you to see what the network sees when detecting one label or another.

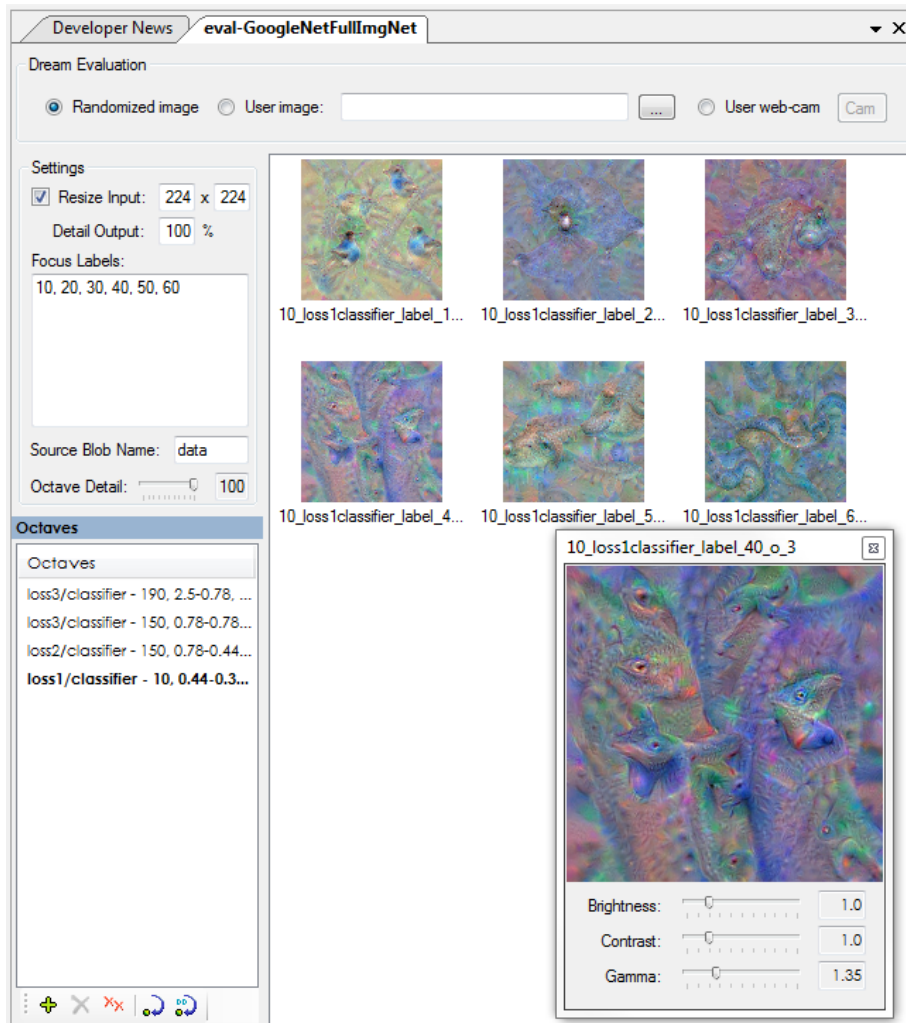


Figure 52 Deep Dream Evaluator

To run the deep dream evaluator<sup>10</sup>, do the following:

- 1.) Right click an open project and select 'Evaluate'.
- 2.) Select the 'Dream' Evaluator.
- 3.) Add several (or all) of the labels that the network detects into the 'Focus Labels' field.
- 4.) Run the evaluator by pressing the 'Run' (▶) button.

**NOTE:** Dream evaluation is not supported on projects that use a gym dataset.

<sup>10</sup> The model shown above was imported from the ImageNet AlexNet model trained by Berkeley and found on GitHub at [https://github.com/BVLC/caffe/tree/master/models/bvlc\\_alexnet](https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet)

## NEURAL STYLE TRANSFER EVALUATION

The 'Neural Style Image' evaluator runs the 'Neural Style Transfer' [13] algorithm to learn the style provided by one image and applies it to another 'content' image.

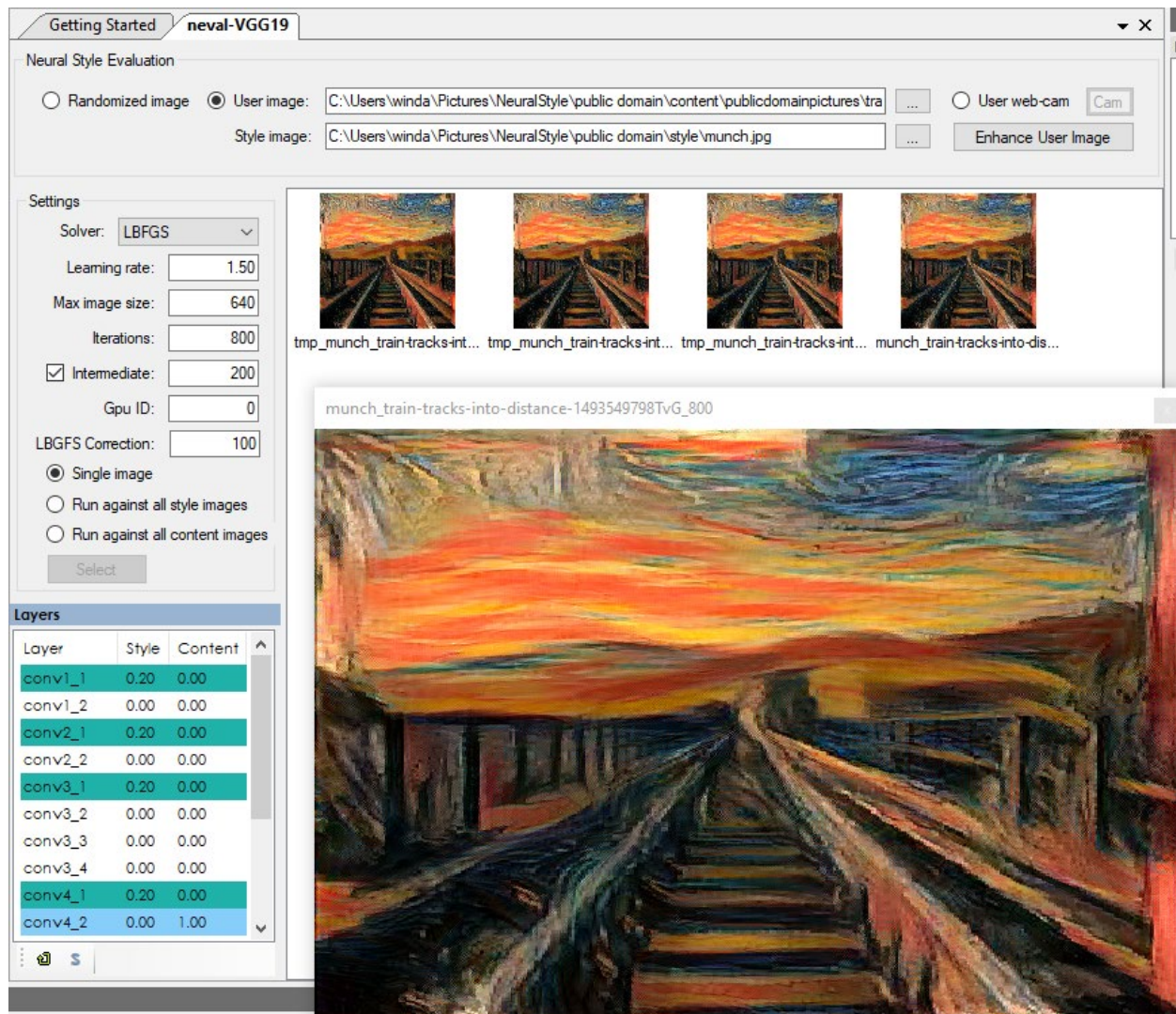


Figure 53 Neural Style Transfer

To run the Neural Style Transfer evaluator, just follow the steps below.

- 1.) Create a project using the VGG19 model as described by [14] using an arbitrary dataset.
- 2.) Open the project with the 'LOAD\_ON-DEMAND' image load method.
- 3.) Once open, right click on the project and select the *Evaluate* menu item.
- 4.) From the *Project Evaluators* dialog, select the 'Neural Style Image Evaluator'.
- 5.) Select the *User Image* and *Style Image* and press the 'Run' (▶) button to start the style transfer.
- 6.) Images will appear in the main window on each *Intermediate* iteration, or upon completion.
- 7.) Double-click any image to enlarge to its original size.
- 8.) Right clicking on the image allows you to save it.

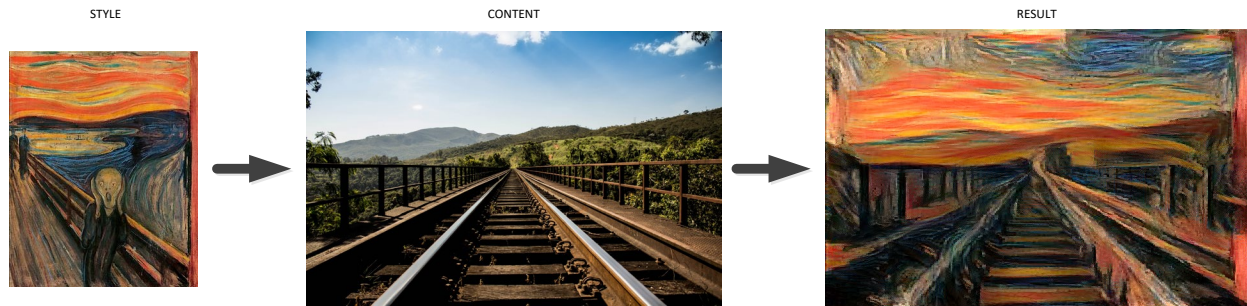


---

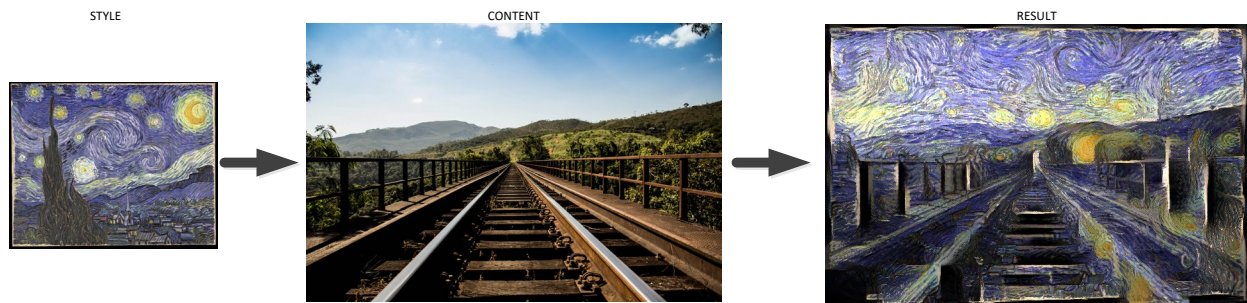
## WHAT NEURAL STYLE TRANSFER DOES

In the example above, we learn the style of Edvard Munch's *The Scream*<sup>11</sup> and applied that style to a public domain image of a train bridge<sup>12</sup> provided by [www.pexels.com](http://www.pexels.com) under the CCo license.

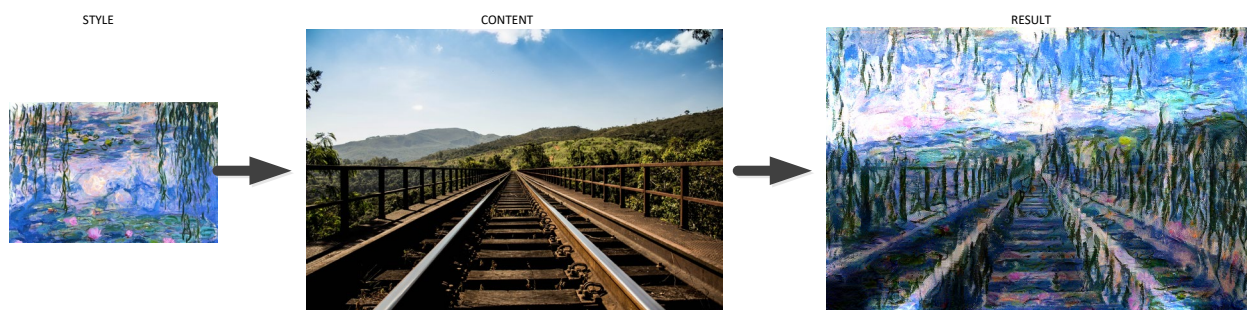
Selecting Edvard Munch's, *The Scream* as the style image tells the application to learn the style of his painting and then apply that style to the content features of the train bridge picture.



The same process works for any style and content image combination. To demonstrate this, we use the same content image as above, but this time apply Van Gogh's *Starry Night*<sup>13</sup> style to it.



As a final example, we apply Claude Monet's *Water Lilies* 1916<sup>14</sup> to the same content image.



---

<sup>11</sup> *The Scream* by Edvard Munch is available to the public domain at [https://commons.wikimedia.org/wiki/File:The\\_Scream.jpg](https://commons.wikimedia.org/wiki/File:The_Scream.jpg).

<sup>12</sup> The train bridge photo is provided by [www.pexels.com](http://www.pexels.com) at <https://www.pexels.com/photo/bridge-clouds-forest-guidance-461772/> under the CCo Licenses with the original source of Pixabay.

<sup>13</sup> *Starry Night* by Vincent van Gogh is available to the public domain at [https://commons.wikimedia.org/wiki/File:VanGogh-starry\\_night.jpg](https://commons.wikimedia.org/wiki/File:VanGogh-starry_night.jpg).

<sup>14</sup> *Water Lilies 1916* by Claude Monet is available to the public domain at [https://commons.wikimedia.org/wiki/File:Monet\\_Water\\_Lilies\\_1916.jpg](https://commons.wikimedia.org/wiki/File:Monet_Water_Lilies_1916.jpg).

---

## NEURAL STYLE SETTINGS

The following settings allow you to alter how the neural style transfer progresses.

**Solver;** several solvers are available, however we have found the LBFGS to work the best. Selecting each solver automatically sets the other settings to the best defaults for that solver.

**Learning Rate;** specifies the learning rate to use with the solver. If your learning rate is too low or high your model may blow up with NaN or infinity.

**Max image size;** specifies the maximum image size used for input and output. Images input to the system are scaled and resized using this setting if they exceed the maximum size specified.

**Iterations;** specifies the number of iterations to run when learning the style of the style image.

**Immediate;** specifies whether to output immediate images and at what iteration interval. For example if your iterations are set at 1000 and the immediate value is set at 100, the style transfer will run for 1000 iterations and output an immediate image at iterations 100, 200, 300, ..., and 1000.

**Gpu ID;** specifies the GPU on which to run the style transfer. Typically, this is the GPU for which the project was opened. However, if you run out of memory when using the main GPU and are running a multi-GPU system, specifying a secondary GPU ID causes the neural style transfer to run on that secondary GPU and not on the one for which the project was opened.

**LBFGS Correction;** specifies the number of LBFGS corrections to use. A smaller number uses less memory but also slows down the style transfer process. If you run out of memory when performing a style transfer, first try reducing this setting. This setting only applies when using the LBFGS solver.

**Single image;** specifies to transfer the style from one style image to one content image.

**Run against all style images;** specifies to transfer the style from all images in the directory for which the style image resides against a single content image.

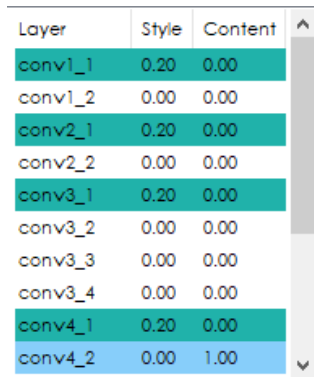
**Run against all content images;** specifies to transfer the style of one style image against all content images in the directory for which the user image resides.

**NOTE:** *Some of the settings may have limited functionality in the general product release and require purchasing the Extended Neural Style Transfer feature to unlock the full power of the neural style transfer.*

---

## LAYER SETTINGS

Changing the layer settings allows you to alter how the style features (learned from the style image) and how the content features (learned from the content image) are applied to the resulting image.



Layer	Style	Content
conv1_1	0.20	0.00
conv1_2	0.00	0.00
conv2_1	0.20	0.00
conv2_2	0.00	0.00
conv3_1	0.20	0.00
conv3_2	0.00	0.00
conv3_3	0.00	0.00
conv3_4	0.00	0.00
conv4_1	0.20	0.00
conv4_2	0.00	1.00

**Figure 54 Layer Settings**

When enabled, double clicking on each of the layers allows you to alter the weights for both style and/or content for that layer.

In addition, pressing the *Set Default* (🔧) button resets the weights back to their defaults and pressing the *Use style only* (S) button sets the defaults to only use style weights (e.g., no content weights).

**NOTE:** Some of the settings may have limited functionality in the general product release and require purchasing the *Extended Neural Style Transfer* feature to unlock the full power of the neural style transfer.

---

## NEURAL STYLE LIMITATIONS

The following neural style limitations apply when running the SignalPop AI Designer in different modes.

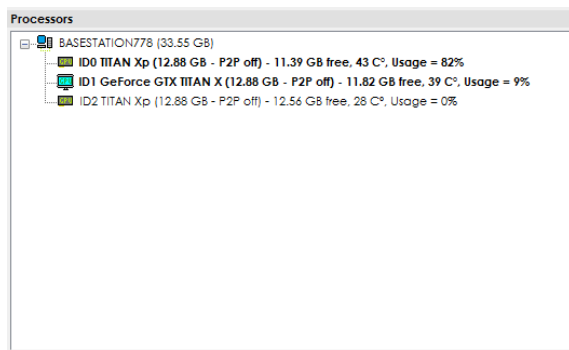
	DEMO	General	Full Neural Style
Max Image Size	220 pixels	440 pixels	Unlimited
Multi-GPU	NO	YES	YES
Multi-Style Images	NO	NO	YES
Multi-Content Images	NO	NO	YES
Layer Weight Editing	NO	YES	YES
Solver Editing	NO	YES	YES
Learning Rate Editing	NO	YES	YES
Intermediate Output	NO	YES	YES
Image Saving	NO	YES	YES
Maximum Iterations	200	Unlimited	Unlimited

## HARDWARE

When using the SignalPop AI Designer it is important to see how your hardware is performing and what portions of the hardware are in use. Four views give better insights on how the hardware GPUs are being used: The 'Processors' resource window, the 'Project' window throughput graph, the Real-time debug timing, and the 'Memory Test' window.

### PROCESSOR RESOURCE WINDOW

The 'Processor' window shows which GPUs are installed within your computer, and the current settings and usage of each GPU.



**Figure 55 Processors Window**

This window lets you know the following information about each GPU in your computer:

- The number of GPU's installed and the model's name of each.
- The amount of total memory of each GPU.
- The current mode that the GPU is running in (TCC - 'P2P on' or WDM - 'P2P off')
- The amount of free GPU memory.
- The current usage as a % of total GPU processor usage.
- The GPU(s) with Monitors attached.

## PROJECT THROUGHPUT

The throughput graph in the 'Project' window shows the timing on each cycle of the model as it is being trained. Both the MyCaffe low-level throughput for each forward + backward pass and the overall network throughput at the application level are plotted in milliseconds.

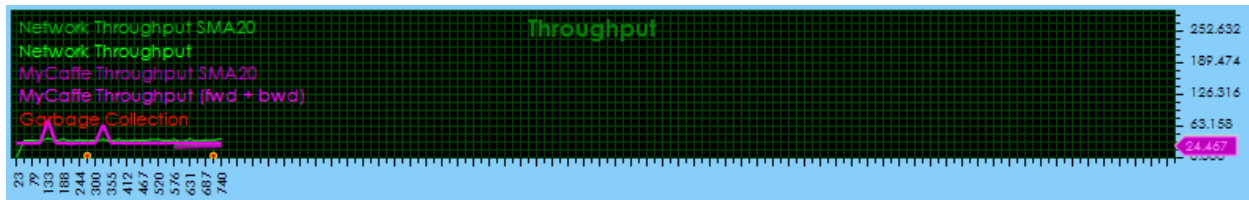


Figure 56 Throughput Graph

Note, at the bottom of the throughput graph are plotted small red/yellow points where a system garbage collection was detected.

## REAL-TIME DEBUG TIMING

To help further diagnose bottlenecks in the system, the real-time 'Debug' window shows the timing of the forward and backward pass for each layer thus helping pinpoint where a network slows down during training.

The figure shows a 'Debug' window for a network named 'AlexNet'. The GPU temperature is 80C. The table below lists the layers and their forward and backward timing in milliseconds. Excessive times are highlighted in yellow.

Blob	Size	Type	Location	Data Min	Data Max	Diff Min	Diff
<b>data (DATA)</b>							
[fwd timing]	0.167 ms.						
[bwd timing]	0.000 ms.						
data	256,3,32,32	DATA	TOP	-0.546875	0.60546875	0	0
label	256,1,1,1	DATA	TOP	0	9	0	0
<b>conv1 (CONVOLUTION)</b>							
[fwd timing]	0.571 ms.						
[bwd timing]	0.070 ms.						
data	256,3,32,32	DATA	BOTTOM	-0.546875	0.60546875	0	0
conv1	256,96,22,22	DATA	TOP	0	5.46550178...	-0.0043...	0.0
conv1 weights	96,3,11,11	PARAM	NONE	-0.5407485...	0.49263706...	-0.0006...	0.0
conv1 bias	96,1,1,1	PARAM	NONE	-0.8040485...	0.10103730...	-0.0020...	0.0
<b>relu1 (RELU)</b>							
[fwd timing]	0.046 ms.						
[bwd timing]	0.016 ms.						
conv1	256,96,22,22	DATA	BOTTOM	0	5.46550178...	-0.0043...	0.0
<b>norm1 (LRN)</b>							
[fwd timing]	0.033 ms.						
[bwd timing]	0.016 ms.						
conv1	256,96,22,22	DATA	BOTTOM	0	5.46550178...	-0.0043...	0.0
norm1	256,96,22,22	DATA	TOP	0	3.24905967...	-0.0073...	0.0
<b>pool1 (POOLING)</b>							
[fwd timing]	0.035 ms.						
[bwd timing]	0.024 ms.						
norm1	256,96,22,22	DATA	BOTTOM	0	3.24905967...	-0.0073...	0.0
pool1	256,96,12,12	DATA	TOP	0	3.24905967...	-0.0054...	0.0

Figure 57 Layer Timings in the Debug Window

Note, excessive times are highlighted in yellow.



## MEMORY TESTER

The memory tester runs a memory test across all (or a portion of) the memory in each GPU. This test can be helpful in determining whether your hardware is functioning as expected.

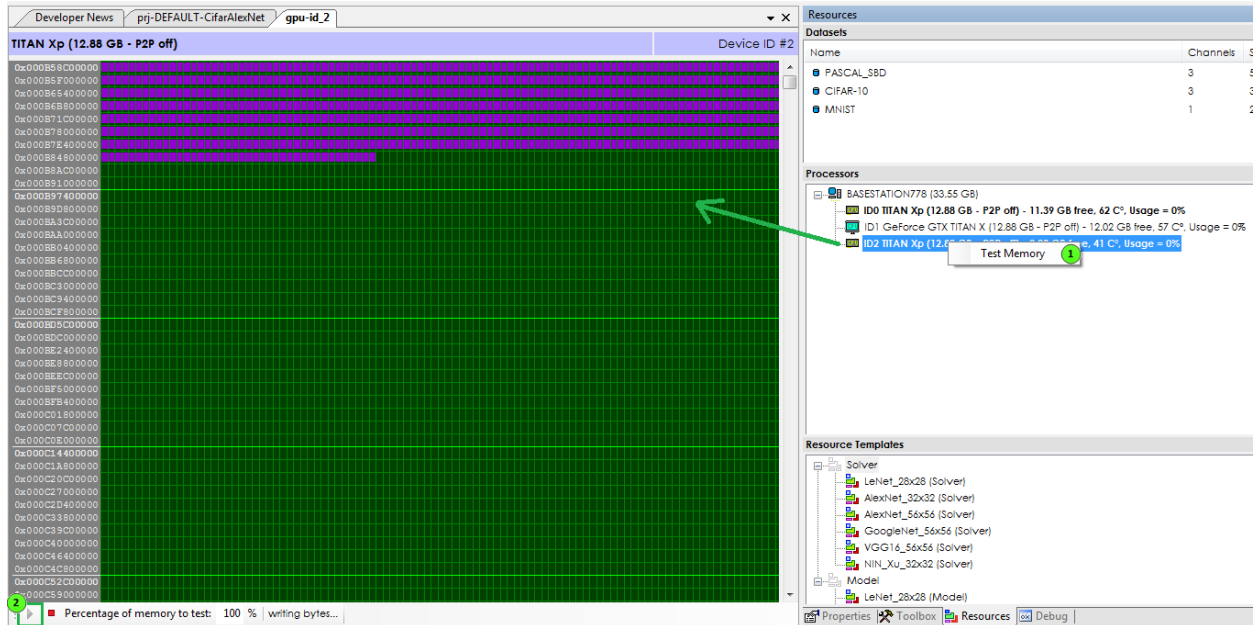


Figure 58 Memory Testing

To run the memory test, do the following:

- 1.) First, right click on the GPU within the 'Processors' window that you want to test, which will display the 'Memory Test' window for that GPU.
- 2.) Next, select the 'Start' (▶) button, in the Memory Test window, to start the memory test.

## EXAMPLE MODELS

This section provides the step-by-step process of setting specific more complicated models using the SignalPop AI Designer.

### DOMAIN-ADVERSARIAL NEURAL NETWORKS (DANN)

The Domain-Adversarial Neural Network (DANN) is an adversarial network inspired by [3] with the goal of learning domain adaptation where the source and target domains differ.

#### DATASETS

We can simulate an environment using the MNIST dataset, where a 3-channel version of the standard MNIST dataset is used as the 'source' dataset.

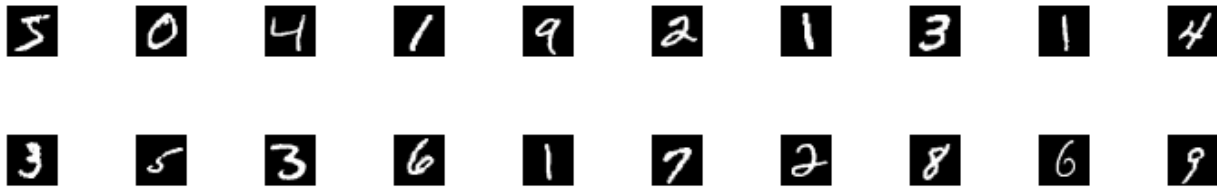


Figure 59 MNIST 'source' Dataset

The 3-channel 'source' MNIST dataset has two data sources: *MNIST.training.3\_ch* and *MNIST.testing.3\_ch*.

And a version of the MNIST dataset that is superimposed on another image (e.g., simulating an environment) is used as the 'target' dataset.



Figure 60 MNIST 'target' Dataset

Both datasets are easily created using the SignalPop AI Designer's MNIST Dataset Creator. See section [Creating the MNIST 'source' and 'target' Datasets](#) above for details on creating these datasets.

The 3-channel 'target' MNIST dataset has two data sources: *MNIST\_Target.training.3\_ch* and *MNIST\_Target.testing.3\_ch*.

Once the datasets are created, we are ready to create the model.

## DANN MODEL

To create the DANN model, we must create the model and add the new 'target' MNIST dataset to it. The following steps will guide you through creating this model.

- 1.) First, select the 'Solutions' tab in the SignalPop AI Designer.
- 2.) Select the Add Project (+) button at the bottom of the pane.
- 3.) Fill out the 'New Project' dialog with the project name, the MNIST dataset, and the DANN model and solver templates as shown below.

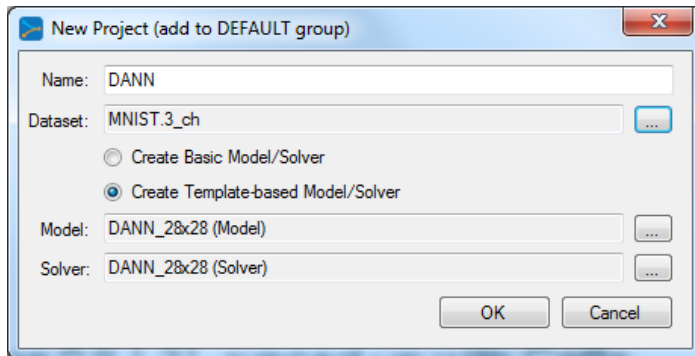


Figure 61 New Project Dialog

- 4.) After pressing OK, you will see the new DANN project added to the Solutions window.
- 5.) To add the 'target' dataset, select the 'MNIST\_Target.3\_ch' dataset from the Datasets pane within the Resources window and drag-n-drop it onto the DANN project.

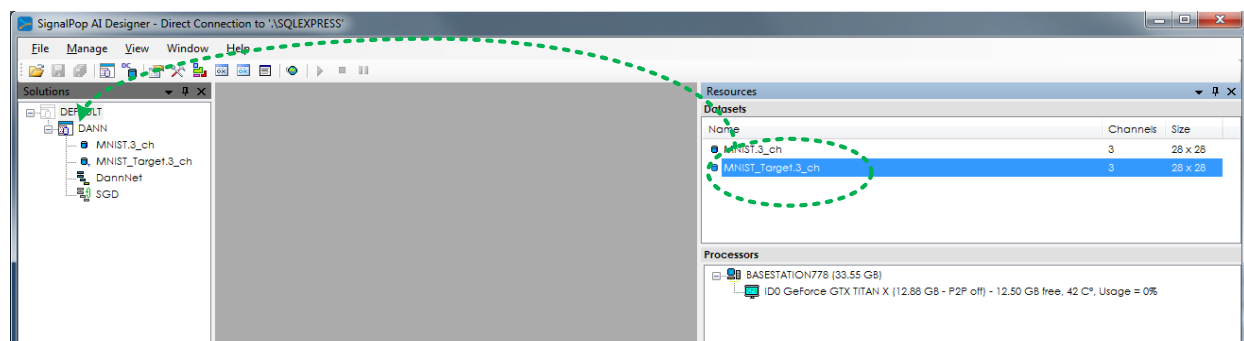


Figure 62 Drag-n-drop the Target Dataset

- 6.) Upon dropping the dataset, you will be presented with the 'Drop Dataset Action' dialog. From this dialog, select the 'Add this dataset to the 'target' dataset' radio button.
- 7.) After pressing OK, the DANN project updates with the new dataset listing both the source MNIST.3\_ch and target MNIST\_Target.3\_ch datasets.

At this point, you are ready to open and train your model just like any other model. See the section [Opening Projects](#) above to open your new project, and see section [Training and Testing Projects](#) above to start testing your model.

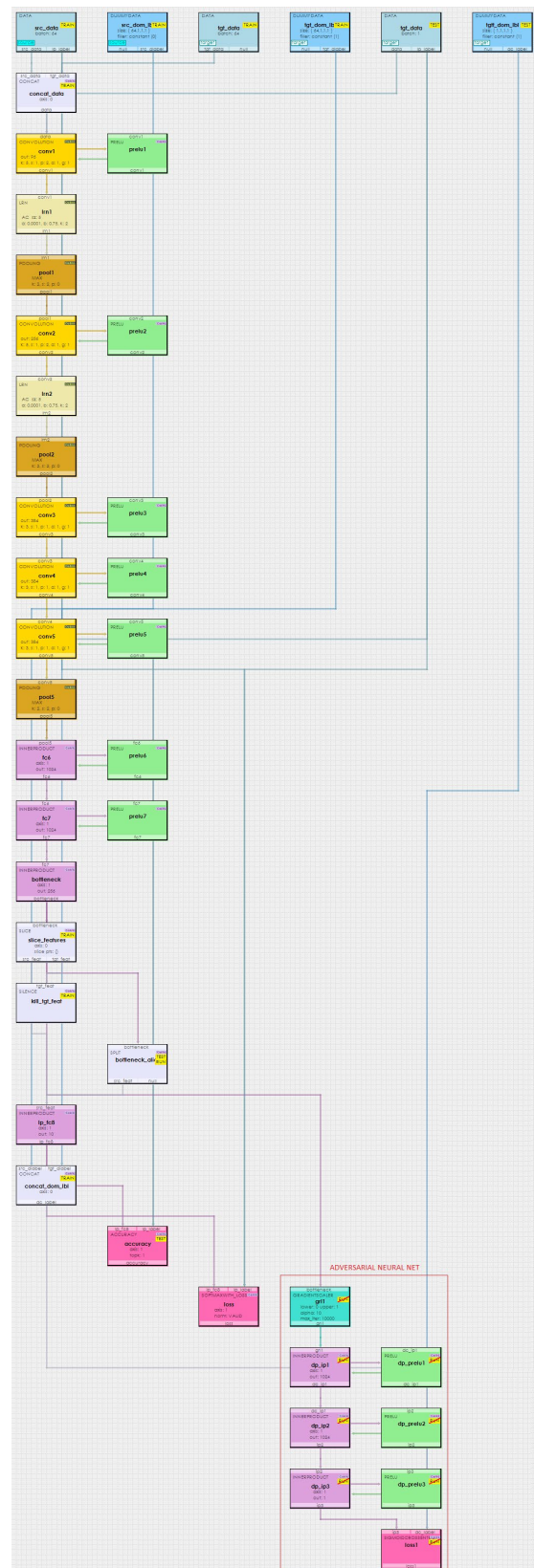
## FULL MODEL

The DANN network is made up of two neural networks that are averse to one another. The first, main network is a regular convolutional network that learns the pattern of the source domain.

Bottleneck layer that links both networks together →

And the second network learns the target domain and operates adversely to the first network via the GRADIENTSCALE 'grl1' layer shown to the right →

The gradient scaler layer is a gradient reversing layer introduced by [3]. The reversed gradients are then added back into the bottleneck layer shown above.



## VALIDATING THE DANN MODEL

Once you have trained your model for around 10,000 iterations you should see an accuracy of around 80% or greater. The example below was run on an NVIDIA TITAN X (Maxwell). The next step is to run the model on a validation set of data.

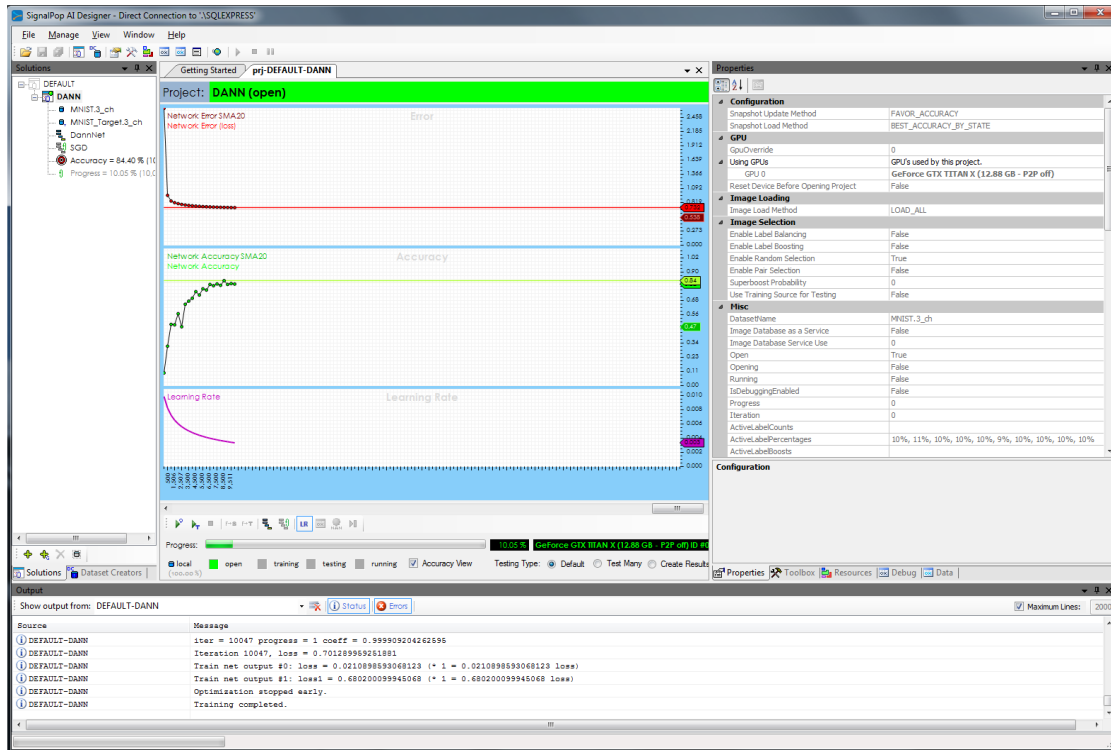


Figure 63 Model Training

The next step is to validate your model by running it against the training data source of the MNIST\_Target.3\_ch data source.

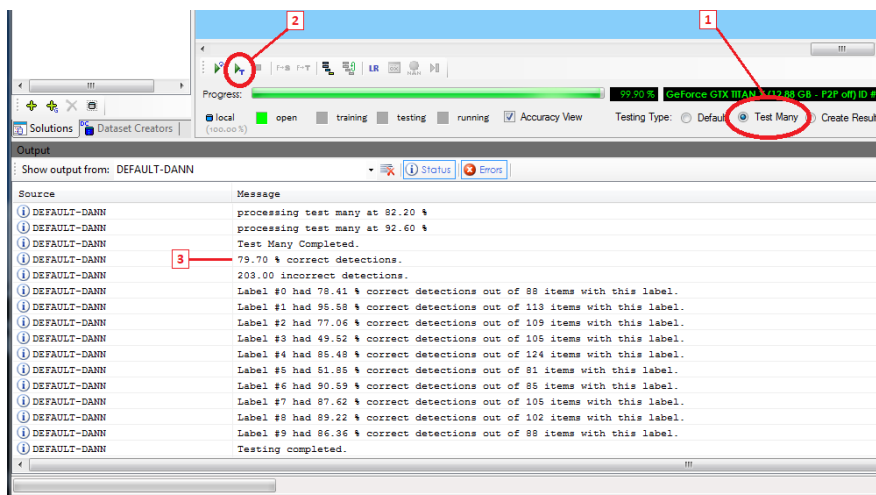



Figure 64 Validating Your Model

To validate your model, follow these steps:

- 1.) First select the 'Test Many' radio button at the bottom of the Project window.
- 2.) Next, select the 'Run Testing' button  to start the test run. After starting the test run, the 'Select Dataset for Testing' dialog will appear. From this dialog, select the 'Target Dataset' to run the model on images randomly selected from the MNIST\_Target.3\_ch dataset.
- 3.) Upon completing the test run, which runs 1000 images through the model, the statistics for the run appear in the output window and show the accuracy of the model on the 'target' data set.

## DEEP AUTO-ENCODER NETWORKS

In addition to data compression and dimension reduction tasks, auto-encoders provide an unsupervised learning method often used to pre-train deep neural networks [15]. The auto-encoder uses an encoding-decoding method that learns the input features by learning to re-create the input data. Our auto-encoder models were inspired by [16] where we have modified the original model by adding Batch Normalization layers to help stabilize the data flowing through the network and better reproduce the input data set. In this section, we will walk through the steps necessary to re-create both the deep convolution auto encoder and the deep convolution with pooling auto encoder models.

### DATASETS

Both models discussed in this section use the 1-channel MNIST dataset shown below.



Figure 65 MNIST 'source' Dataset

To create the MNIST dataset see the section [Creating the MNIST Dataset](#) above.

### AUTO-ENCODER MODEL

To create the Auto-Encoder model, we must add a new project to the Solutions window. The following steps will guide you through creating this model.

- 1.) First, select the 'Solutions' tab in the SignalPop AI Designer.
- 2.) Select the Add Project (+) button at the bottom of the pane.
- 3.) Fill out the 'New Project' dialog with the project name, the MNIST dataset, and either the Auto Encoder, or Auto Encoder Pool model and solver templates as shown below.

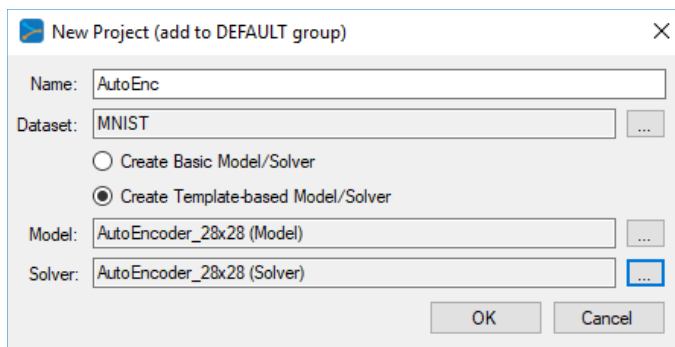


Figure 66 New Project Dialog

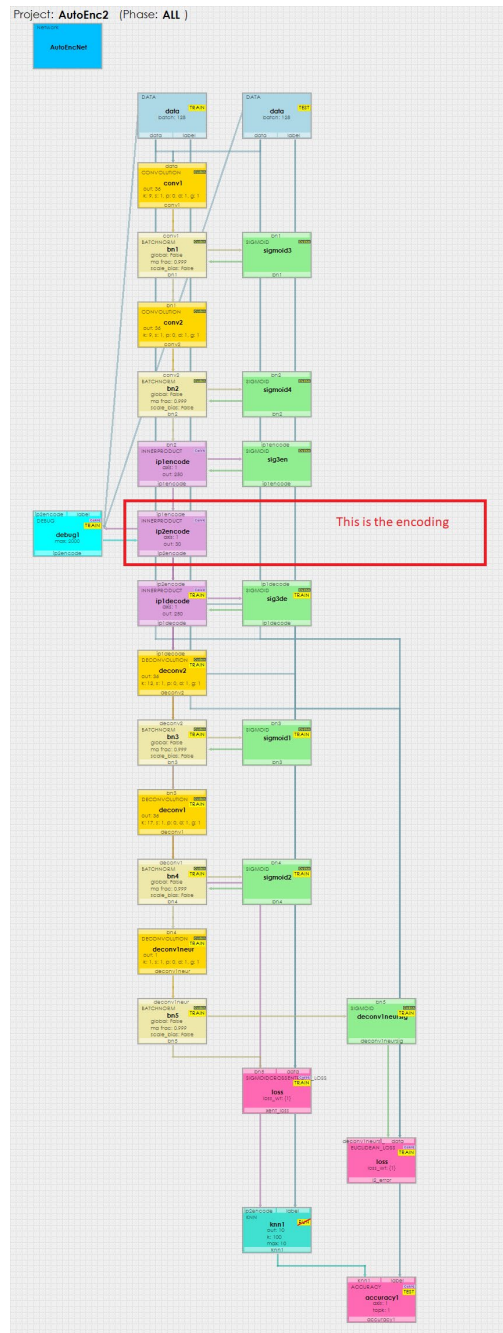
- 4.) After pressing OK, you will see the new AutoEnc project added to the Solutions window.



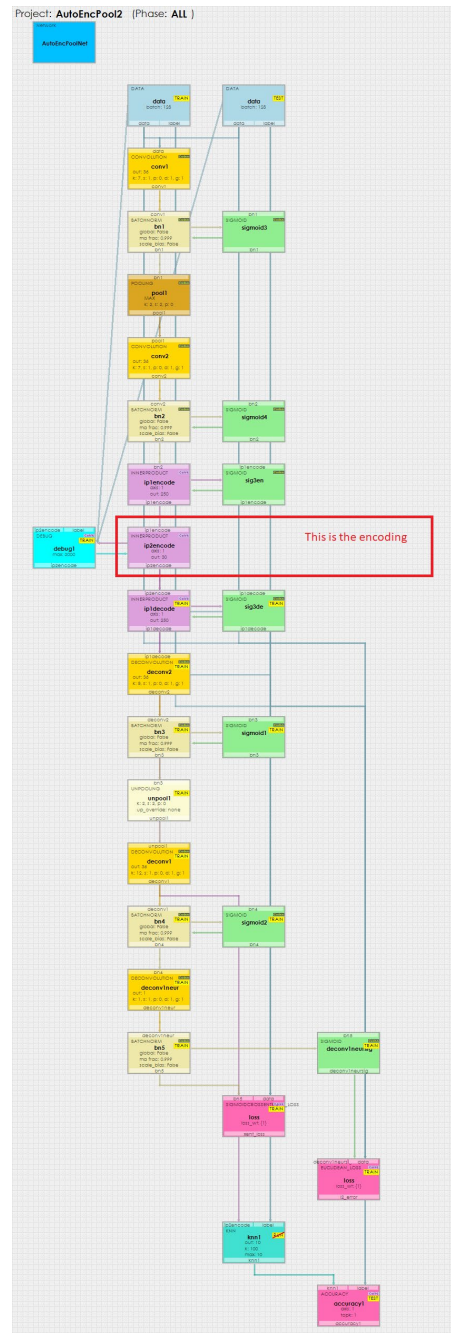
## FULL MODEL

To view your new model, expand the new 'AutoEnc' project and double click on its 'AutoEncNet' sub-tree item. When opened in the model editor, the auto-encoders will look as follows:

Deep Convolution Auto-Encoder



Deep Convolution Auto-Encoder with Pooling



For the full details on viewing your new model see the section [Opening Projects](#) above to open your new project, and see section [Training and Testing Projects](#) above to start testing your model.



## MODEL ANALYSIS

When training, visualizing the model's internals can be helpful in determining whether you are on the right track. The SignalPop AI Designer has many helpful visualization tools for this purpose – we will use three of these tools here to show how well the model works.

### LAYER INSPECTION

Auto-encoders are trained by using the model to re-create the input. Using the layer inspection visualization on the last layer, we can see how well the model does this recreation.

To inspect the output layer data, follow the steps below.

- 1.) First train your model up through around 1000 iterations using the steps described in section [Training and Testing Projects](#).
- 2.) Next, open the model editor as described above, and make sure that the model itself is loaded and in the 'Open' state as described in section [Opening Projects](#).
- 3.) From within the model editor, scroll down to the bottom of the model and select the 'bn5' node. As shown below, you will see that this node produces a batch of 128, single channel 28x28 outputs – which are the images re-created by the model.

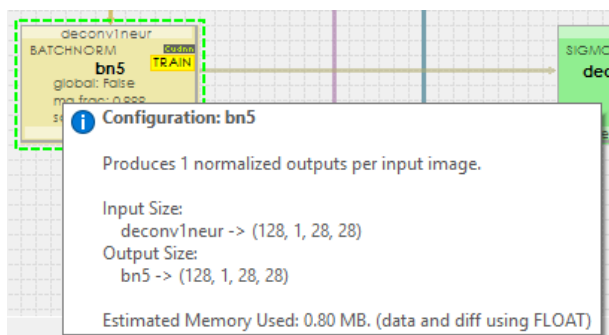


Figure 67 Node Selection

- 4.) Right click on the selected 'bn5' node and select the 'Inspect Layer' menu item, which opens a window containing a visualization of the data in that layer. As shown below, the 'bn5' layer clearly re-creates the MNIST data images.

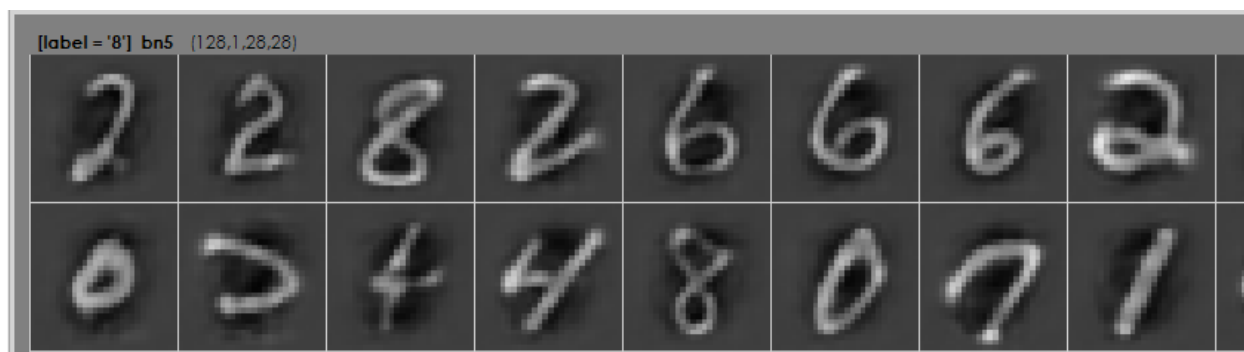


Figure 68 Visualizing the 'bn5' layer.

## DATA INSPECTION

Sometimes your training may not go as expected and never converge to a solution. In such cases visualizing the last layer may show all black images or all white, washed-out images. This occurs when your model 'blows-up' by either driving the gradients to very small or very large numbers.

Inspecting the data flowing between layers can be a helpful tool that helps you hunt down the source of the model blow-up.

To inspect a data link, follow the steps below.

- 1.) As discussed in the previous section, first train your model up through around 1000 iterations using the steps described in section [Training and Testing Projects](#).
- 2.) Next, open the model editor as described above, and make sure that the model itself is loaded and in the 'Open' state as described in section [Opening Projects](#).
- 3.) From within the model editor, scroll down to the bottom of the model and select the line connecting the 'bn5' node to the Sigmoid Cross Entropy 'loss' layer – when selected, the line will highlight bright green and display the size of the data flowing between the two layers which in this case will read 'bn5 -> (128, 1, 28, 28)'.

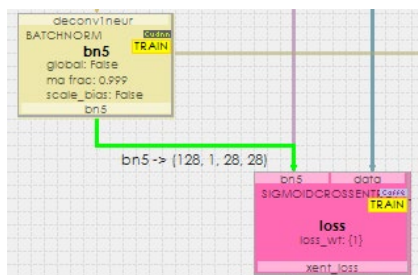




Figure 69 Selecting a Data Link

- 4.) Next, select the 'Inspect Single Items' () button at the bottom of the model editor – this button is only displayed when the project is in the 'Open' state.
- 5.) Now, right-click on the selected link and select the 'Inspect Data Link' menu item which will send the data flowing between the 'bn5' and 'loss' layer to the Data window ().

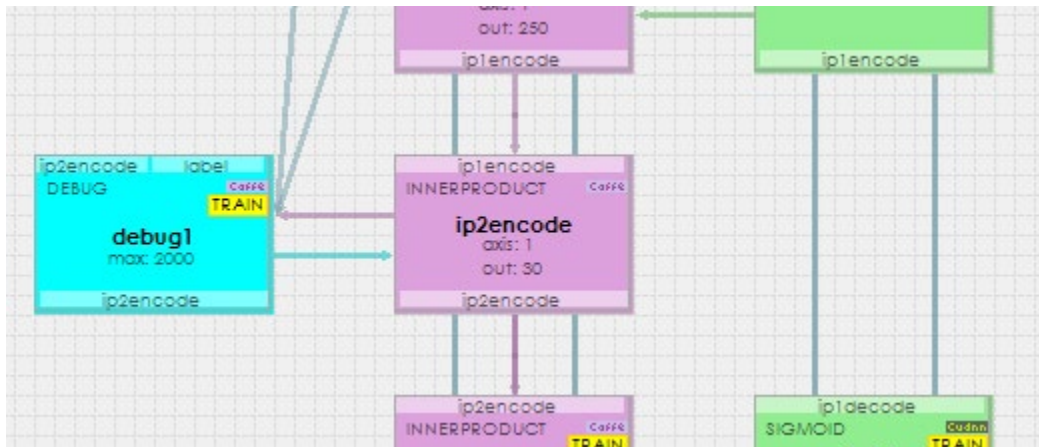
C	H	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26
0	0	-0.1243	-0.1282	-0.1191	-0.1147	-0.1136	-0.1122	-0.1145	-0.1284	-0.1298	-0.1203	-0.1178	-0.1289	-0.1313	-0.1284	-0.1286	-0.1288	-0.1443	-0.1568	-0.1565	-0.1609	-0.1580	-0.1379	-0.1242	-0.1348	-0.1441	-0.1441	-0.1385
0	1	-0.1237	-0.1220	-0.1074	-0.1065	-0.1043	-0.1103	-0.1174	-0.1074	-0.0949	-0.0748	-0.0684	-0.0923	-0.1236	-0.1139	-0.1371	-0.1452	-0.1707	-0.1785	-0.1782	-0.1746	-0.1518	-0.1392	-0.1354	-0.1571	-0.1619	-0.1484	-0.1329
0	2	-0.1220	-0.1059	-0.0947	-0.0927	-0.1146	-0.1332	-0.1115	-0.0616	-0.0458	-0.0406	-0.0665	-0.1145	-0.1438	-0.2492	-0.3533	-0.2768	-0.3538	-0.2372	-0.2340	-0.2170	-0.2188	-0.1958	-0.1773	-0.1767	-0.1614	-0.1307	-0.1296
0	3	-0.1228	-0.1087	-0.0782	-0.0776	-0.1057	-0.1160	-0.0883	-0.0494	-0.0927	-0.1228	-0.0893	-0.1118	-0.2442	-0.3815	-0.3287	-0.2573	-0.2655	-0.3348	-0.4279	-0.3731	-0.2707	-0.2387	-0.2242	-0.1893	-0.1632	-0.1420	-0.1392
0	4	-0.1195	-0.1043	-0.0838	-0.0852	-0.0901	-0.0935	-0.1007	-0.1437	-0.2006	-0.1876	-0.2178	-0.2586	0.0324	1.2657	2.8518	3.5831	3.3364	2.0794	0.3820	-0.3998	-0.4107	-0.3266	-0.3234	-0.2598	-0.1587	-0.1459	-0.1444
0	5	-0.1141	-0.1046	-0.0953	-0.0879	-0.0716	-0.0895	-0.1248	-0.2038	-0.2574	-0.2989	-0.2211	0.5103	2.1735	3.1380	2.8880	2.7799	3.0183	3.1723	2.3458	0.3412	-0.4302	-0.4129	-0.3921	-0.3041	-0.1659	-0.1561	-0.1619
0	6	-0.1109	-0.0968	-0.0793	-0.0795	-0.0601	-0.1041	-0.1447	-0.2421	-0.3123	-0.3724	0.1235	2.6948	2.8664	1.6641	-0.0774	-0.0800	0.0320	1.8539	2.6121	1.0961	-0.5095	-0.5982	-0.5039	-0.4439	-0.2543	-0.1920	-0.1808
0	7	-0.1058	-0.0837	-0.0676	-0.0608	-0.1053	-0.1342	-0.2566	-0.3951	-0.5407	-0.6024	-0.0941	0.8571	0.5721	0.3563	0.7529	0.5812	0.2383	0.3413	1.7260	-0.3654	-0.5842	-0.6983	-0.6045	-0.3350	-0.2118	-0.1870	-0.1870
0	8	-0.1029	-0.0717	-0.0457	-0.0552	-0.0872	-0.1678	-0.2929	-0.3800	-0.5562	-0.8215	-0.7651	-0.7033	-1.0264	-0.5332	-1.5287	-1.6381	-1.8051	0.0699	2.3802	1.4075	-1.1992	-0.9564	-0.7478	-0.6289	-0.3263	-0.1806	-0.1616
0	9	-0.0980	-0.0518	-0.0338	-0.0239	-0.0554	-0.1214	-0.2187	-0.3236	-0.6334	-0.9649	-1.1030	-1.3168	-0.9358	-1.4203	-1.3096	-1.4518	-1.5279	0.9646	2.5241	0.6417	-1.2204	-0.8771	-0.6275	-0.4743	-0.2346	-0.1522	-0.1570
0	10	-0.0883	-0.0480	-0.0256	0.0004	-0.0039	-0.0614	-0.1944	-0.3850	-0.7126	-0.9703	-1.1196	-1.2422	-1.1877	-1.0419	-0.9814	-1.2683	-0.9871	1.9450	2.5548	0.0236	-1.1056	-0.7204	-0.4560	-0.3686	-0.1961	-0.1517	-0.1717
0	11	-0.0967	-0.0552	-0.0229	0.0183	-0.0463	-0.0374	-0.2083	-0.3912	-0.6768	-0.9812	-1.0534	-1.0770	-0.9412	-0.8106	-0.9051	-1.0150	0.1705	2.4326	1.9783	-0.8878	-0.5882	-0.6337	-0.4641	-0.4012	-0.1629	-0.2068	-0.1990
0	12	-0.0889	-0.0618	-0.0430	0.0133	-0.0406	-0.0645	-0.2224	-0.4519	-0.7280	-1.0276	-1.1246	-1.0144	-0.9152	-0.9646	-1.2651	-1.2331	1.1277	2.0823	0.3589	-1.2743	-1.0010	-0.7214	-0.5717	-0.4273	-0.1508	-0.2197	-0.1904
0	13	-0.0909	-0.0701	-0.0503	-0.0501	-0.0516	-0.1757	-0.2439	-0.4671	-0.9026	-1.1970	-1.1972	-1.0543	-0.9884	-1.2561	-1.1811	-0.7567	1.4896	1.2594	0.9614	-1.1581	-0.8426	-0.6537	-0.6520	-0.4105	-0.2809	-0.1959	-0.1856
0	14	-0.0962	-0.0851	-0.0839	-0.0218	-0.0863	-0.2945	-0.3075	-0.5222	-1.0236	-1.2030	-1.1889	-1.1099	-1.1444	-1.4637	-1.3054	0.6011	1.3848	-0.2191	1.4286	-1.0549	-0.6044	-0.5580	-0.7304	-0.3691	-0.1929	-0.2023	-0.1679
0	15	-0.0871	-0.0899	-0.1013	-0.0371	-0.1400	-0.4082	-0.4315	-0.5175	-0.8375	-1.0120	-1.1126	-1.0474	-0.9090	-0.8408	-0.0217	1.2897	0.7079	2.0474	1.3168	-1.0436	-0.5946	-0.6170	-0.6790	-0.3649	-0.2083	-0.2073	-0.1729
0	16	-0.0853	-0.0776	-0.0859	-0.0316	-0.1371	-0.4053	-0.4946	-0.5627	-0.7407	-0.8510	-0.9552	-0.7038	-1.3074	-0.6186	1.6671	1.7777	-0.4968	1.4098	-1.1337	-0.9990	-0.6976	-0.6286	-0.5029	-0.3122	-0.2129	-0.1809	-0.1532
0	17	-0.0803	-0.0851	-0.0803	-0.0150	-0.0919	-0.2747	-0.4702	-0.6684	-0.8314	-0.9284	-0.8445	-0.7003	-1.6147	2.6492	1.9452	-1.1958	-1.3312	-1.1248	-0.9351	-0.7427	-0.5598	-0.3661	-0.2829	-0.2172	-0.1657	-0.1314	-0.1314
0	18	-0.1046	-0.0829	-0.0888	-0.0935	-0.0572	-0.2032	-0.4581	-0.6882	-0.7915	-0.4157	-0.0568	0.1705	1.0456	2.5183	2.7261	1.4097	1.2026	-1.3961	-1.1999	-1.0046	-0.7438	-0.4185	-0.2714	-0.3002	-0.2220	-0.1481	-0.1125
0	19	-0.1116	-0.1019	-0.1000	-0.1572	-0.1596	-0.4852	-0.6889	-0.4895	-1.1498	1.7719	1.8159	2.0889	2.3611	1.9621	0.0111	-0.4158	-0.1850	-0.5920	-0.8036	-0.4907	-0.2504	-0.2713	-0.2995	-0.2129	-0.1251	-0.1004	-0.1004
0	20	-0.1097	-0.1126	-0.1306	-0.1776	-0.2104	-0.4338	-0.6230	-0.7235	0.5414	1.5162	2.5884	2.5883	2.1295	1.6037	0.4884	-0.2553	0.1213	0.3396	-0.2603	-0.7011	-0.5912	-0.3514	-0.2587	-0.2703	-0.2077	-0.1271	-0.1024
0	21	-0.1123	-0.1157	-0.1453	-0.1812	-0.2136	-0.4061	-0.6430	-0.3383	2.3707	3.3113	2.6802	2.1622	1.5979	0.2276	1.1851	-1.0948	-0.0240	0.4026	-0.2649	-0.6705	-0.5859	-0.3662	-0.2587	-0.2138	-0.1736	-0.1289	-0.1070
0	22	-0.1129	-0.1120	-0.1512	-0.1700	-0.1692	-0.2671	-0.5085	0.3943	3.2662	3.6378	2.9778	1.9487	0.1534	1.2434	-1.6554	-1.2548	-0.4804	-0.2411	-0.5729	-0.6510	-0.4195	-0.2252	-0.1131	-0.1338	-0.1246	-0.1092	-0.1098
0	23	-0.1307	-0.1464	-0.1377	-0.1330	-0.1934	-0.2279	-0.3177	0.3102	2.4582	3.4338	2.3610	0.1120	-0.9432	-1.1548	-1.1052	-0.9756	-0.6090	-0.4285	-0.4655	-0.3799	-0.1917	-0.0870	-0.0513	-0.0779	-0.1136	-0.1026	-0.1026
0	24	-0.1209	-0.1497	-0.1116	-0.0728	-0.1855	-0.2470	-0.2170	-0.1790	0.0076	-0.1201	-0.5149	-0.8409	-0.7881	-0.7165	-0.7325	-0.6839	-0.6126	-0.2143	-0.2414	-0.1289	-0.1617	-0.0544	-0.0317	-0.0902	-0.1138	-0.1136	-0.1136
0	25	-0.1244	-0.1182	-0.1106	-0.0952	-0.1789	-0.2339	-0.2005	-0.1686	-0.2826	-0.4790	-0.5872	-0.4325	-0.2491	-0.3315	-0.3097	-0.3084	-0.1614	-0.0806	-0.1088	-0.1277	-0.0772	-0.0282	-0.0575	-0.1086	-0.1148	-0.1187	-0.1141
0	26	-0.1259	-0.1191	-0.1030	-0.0976	-0.1085	-0.1251	-0.1518	-0.2003	-0.2636	-0.3164	-0.2736	-0.2057	-0.1889	-0.1583	-0.1084	-0.0950	-0.0470	-0.0308	-0.0582	-0.0689	-0.0689	-0.1012	-0.1054	-0.1102	-0.1196	-0.1161	-0.1125
0	27	-0.1236	-0.1332	-0.1210	-0.1129	-0.1165	-0.1172	-0.1084	-0.1094	-0.1096	-0.1112	-0.0904	-0.0957	-0.1344	-0.1156	-0.1202	-0.1042	-0.0792	-0.0887	-0.0961	-0.1052	-0.1132	-0.1250	-0.1218	-0.1167	-0.1149	-0.1164	-0.1157

Figure 70 Data Window

## ENCODING SEPARATION

Once you can stabilize your training, you will also want to verify that your model learns an encoding that has good data separation.

To view the encoding data separation, we will use the Debug Layer, which is attached to the 'ipzencode' Inner Product Layer – our encoding layer.

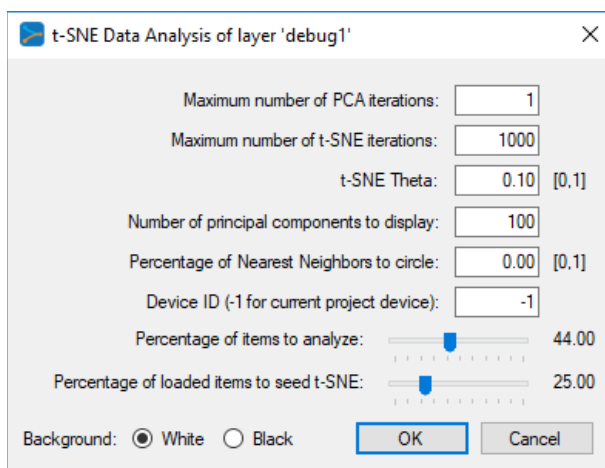


### Figure 71 Using the Debug Layer

During training, the Debug Layer merely stores the outputs of the attached layer (e.g., the 'ipzencode' layer in this case) in a revolving buffer.

When inspecting the Debug Layer, we run the T-SNE [5] algorithm to visualize the data separation of the items output by the layer being debugged. In this case, the T-SNE algorithm is run on our embedding which visually shows how well the encoding works.

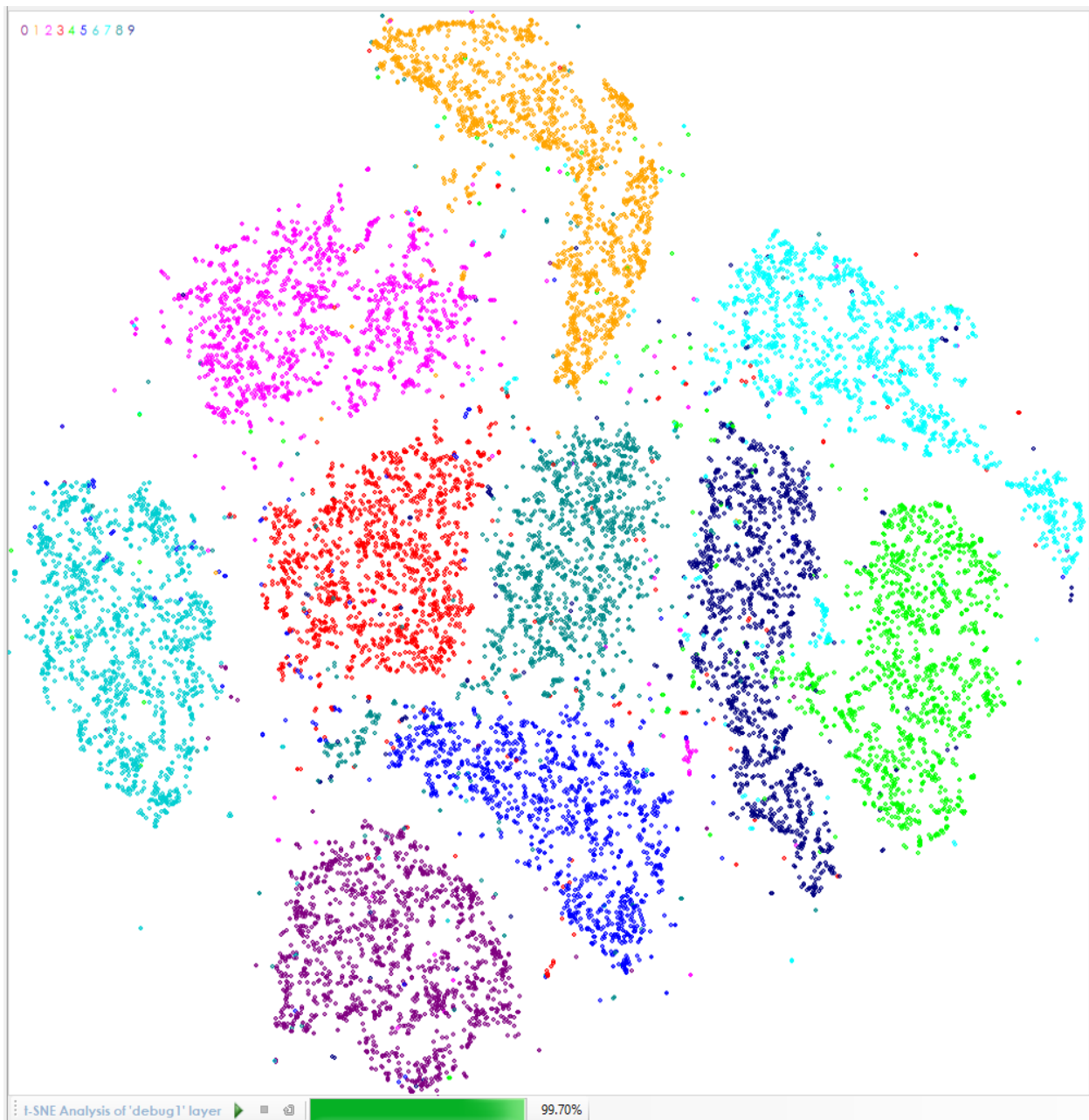
After first selecting 'Inspect Layer' you will be presented with the 't-SNE' settings dialog.



### Figure 72 t-SNE Settings Dialog

See the [T-SNE Settings](#) Section above for more information on this dialog.

The process will animate while the solution is found to the t-SNE data separation. Once complete, you can visually inspect the data separation of the embedding itself.



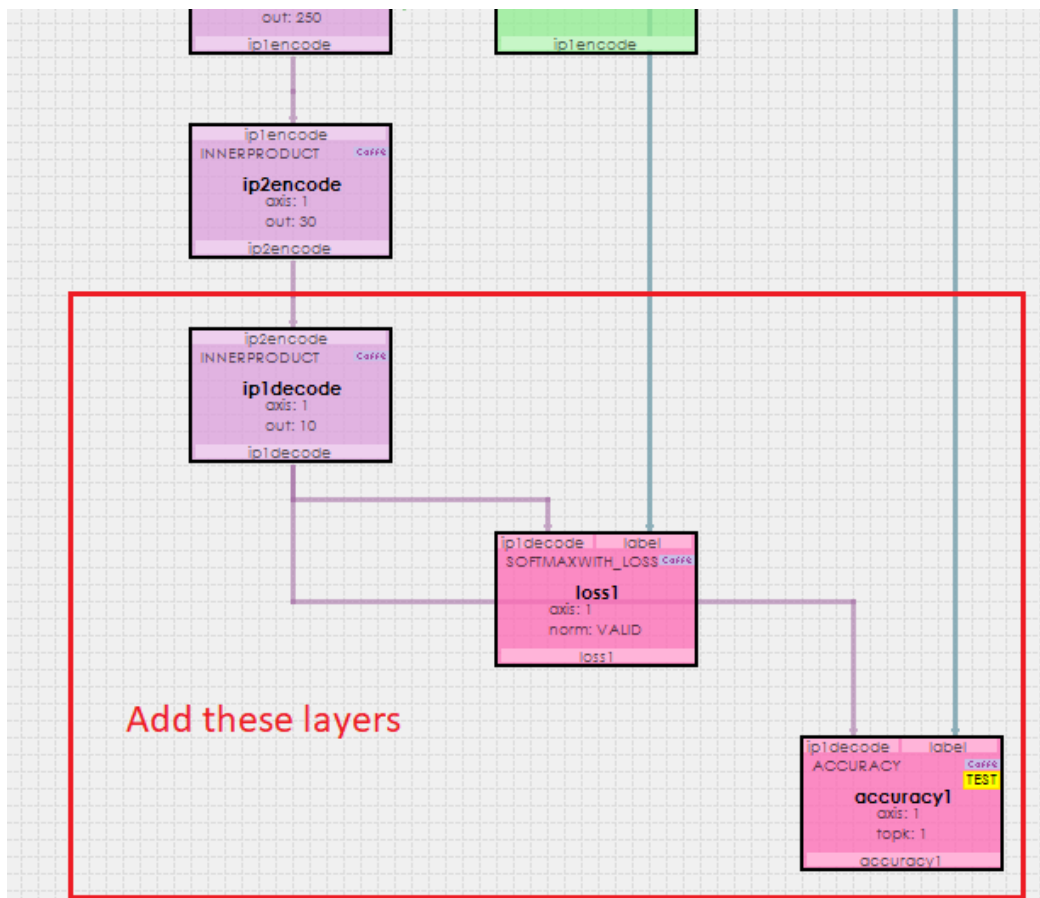
**Figure 73 T-SNE Data Separation**

At this point your model is ready to use as a pre-trained set of weights used to accelerate the training of a network that uses the nodes at least up through the 'ipzencode' layer.

## USING PRE-TRAINED AUTO-ENCODER

After training the 'AutoEncPool' project, you can use the trained weights to pre-train a classifier. To do this, first copy the 'AutoEncPool' project, by right clicking on the project in the 'Solutions' pane and selecting the 'Copy' menu item. When copying you do not need to copy the weights for, we will re-import them.

Load the newly copied project's model description into the model editor (e.g., double-click on the 'AutoEncPoolNet' sub-tree item of the newly copied project named 'AutoEncPool2'). Remove all layers below the 'ip2encode' layer and make the as shown below.



**Figure 74** Converting Auto-encoder Model.

After removing the layers below the 'ip2encode' layer, we will add back a 10 output INNER PRODUCT layer after the 'ip2encode' layer along with the standard SOFTMAXWITH\_LOSS and ACCURACY layers used for classification problems.

The new INNER PRODUCT layer can use the Xavier weight filler, and Constant (0.2) bias filler.

**NOTE:** If you do not want to fiddle around with copying and editing a project, you can merely just create a new project and use the 'AutoEncoderPoolRun\_28x28 (Model)' and associated solver.

The complete model should then look as follows.

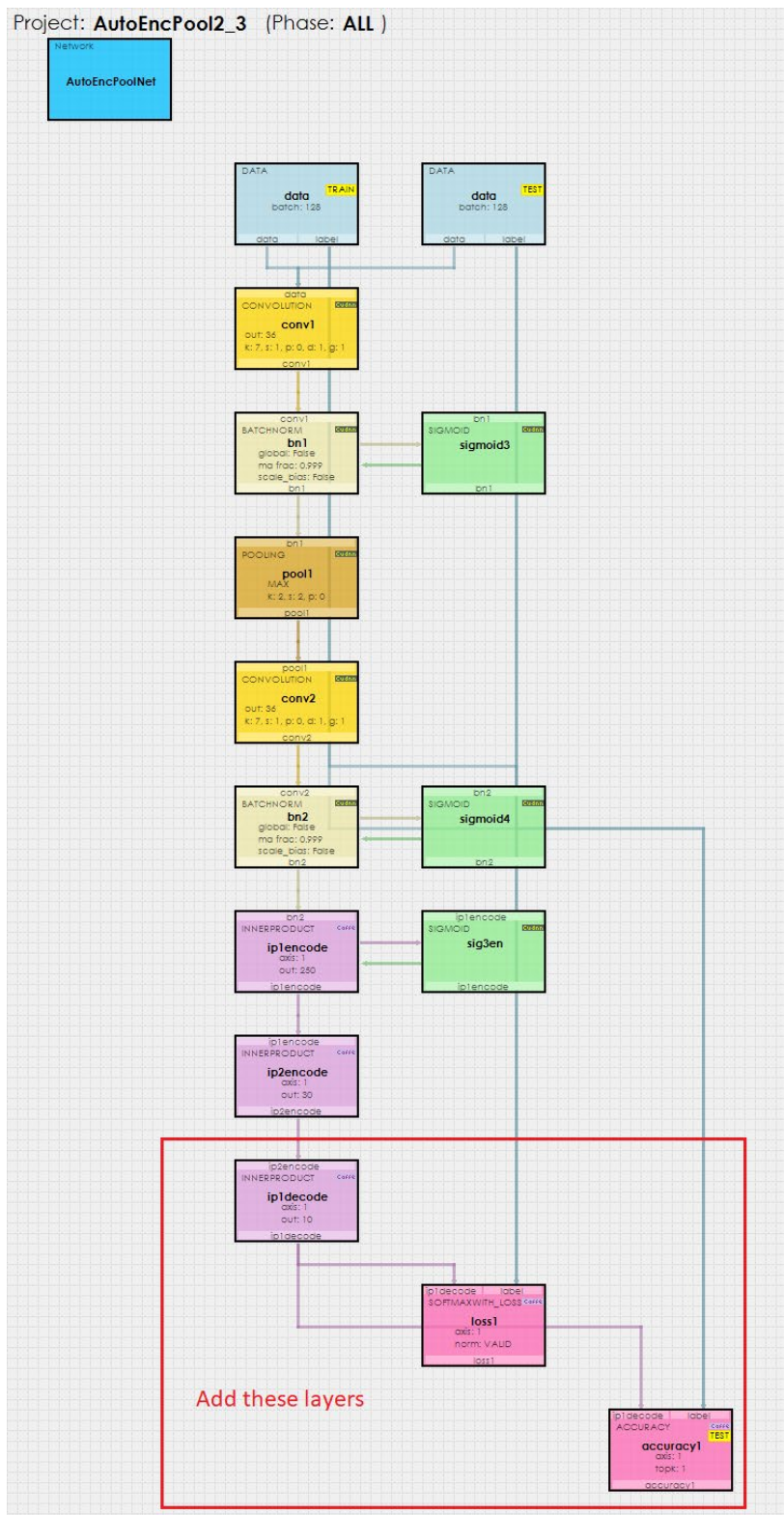


Figure 75 Complete Classifier Model



## IMPORTING AUTO ENCODER WEIGHTS

Next, you will want to import the weights up through the encoding layer into your new classifying model shown in the previous section.

First, we need to export the weights from your original 'AutoEncPool' model. To do this, follow the steps below.

- 1.) Expand the 'AutoEncPool' project in the 'Solutions' tree view.
- 2.) Right-click on the 'Accuracy' sub-tree item from the model and select the 'Export | Weights' menu item.
- 3.) Save the weights to a file of your choosing. In our example, we have saved the weights to the file 'C:\Users\win\Pictures\temp.mycaffemodel'.

Next, we need to import the weights into your newly copied model 'AutoEncPool2'. To do this, follow the steps below.

- 1.) First, open the newly copied 'AutoEncPool2' model by right clicking on it and selecting the 'Open' menu item.
- 2.) Next, right-click on the 'Accuracy' sub-tree item within the 'AutoEncPool2' model and select the 'Import' menu item which will display the dialog below.

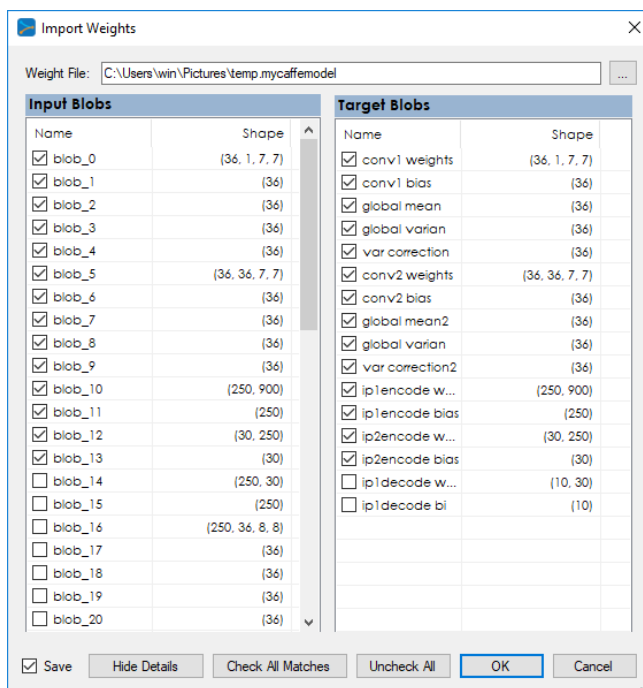
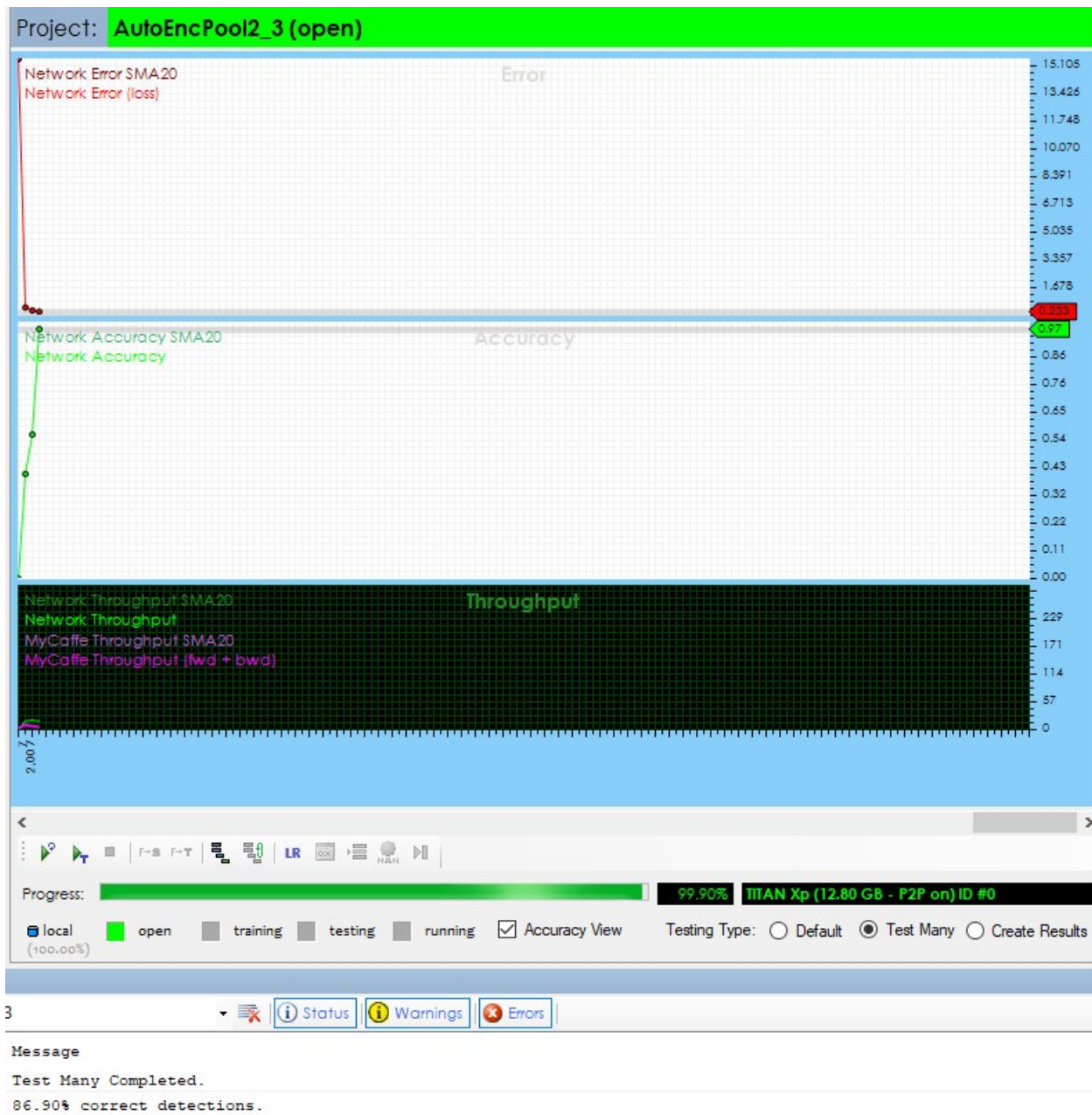


Figure 76 Weight Import Dialog

- 3.) All matching blobs will be checked when opening the dialog as only matching blob sizes can be imported. Make sure to check the 'Save' check box to save the weights into your new project and press 'OK' to import the weights.

## TRAINING COMPARISON

Now that you have the pre-trained weights imported, you are ready to train your classifier network. When training you should see the accuracy jump up to around 97% within 3000 iterations.

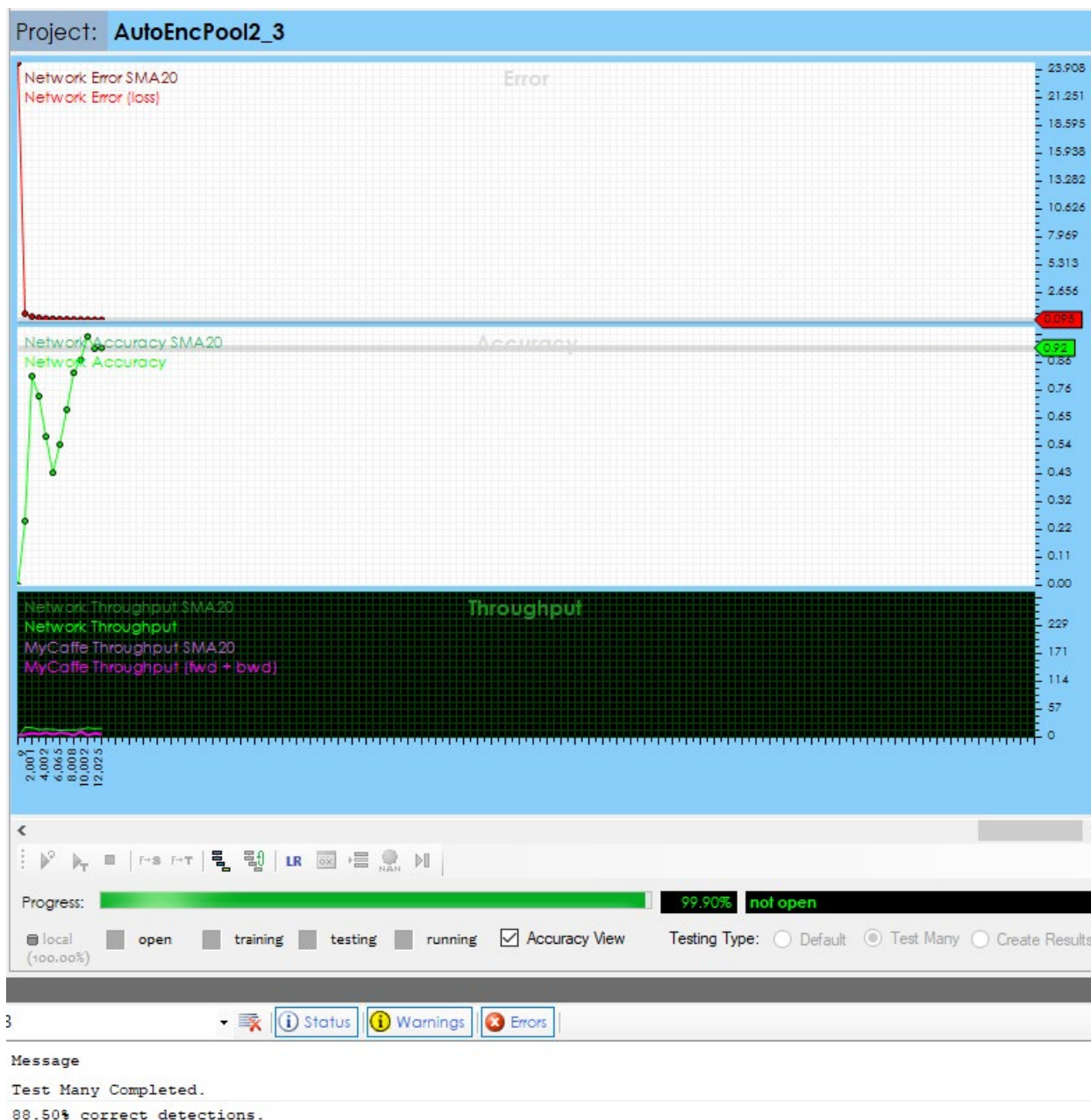


**Figure 77 Pre-trained Network Performance**

At this point you may want to stop training and run a simple test on a set of 1000 images selected randomly from within your test set. In our test, the 'Test Many' had an accuracy of 86.9%.



How does the pre-trained network compare to one that was not pre-trained? The following training was performed with the standard random weight initialization.



**Figure 78 Non-Pre-Trained Network Performance**

Like the pre-trained network, the non-pre-trained network reaches 90%+ accuracy but does so after 10,000 iterations as opposed to the 3,000 iterations of the pre-trained network. This clearly shows that pre-training can dramatically help reduce the training time.

## SIAMESE NETWORK WITH CONTRASTIVE LOSS FOR ONE-SHOT LEARNING

In this example, we show how to create, train and use a Siamese Net to learn hand written character classification using the MNIST dataset similar to [17] and [18]. The Siamese Net provides the ability to perform 'One-Shot' learning where an image that the network has never seen before, is quickly matched with already learned classifications if such a match exists. For several examples of one-shot learning, see [19] who use a Siamese Net for image retrieval, [20] who use a Siamese Net for content based retrieval, and [21] who use a Siamese Net to detect railway assets such as switches on the railway track.

### DATASETS

The Siamese Net discussed in this section uses the 1-channel MNIST dataset shown below.



Figure 79 MNIST 'source' Dataset

To create the MNIST dataset see the section [Creating the MNIST Dataset](#) above.

### SIAMESE NET MODEL

To create the Siamese model, we must add a new project to the Solutions window. The following steps will guide you through creating this model.

- 1.) First, select the 'Solutions' tab in the SignalPop AI Designer.
- 2.) Select the Add Project (+) button at the bottom of the pane.
- 3.) Fill out the 'New Project' dialog with the project name, the MNIST dataset and the Siamese\_28x28 model and solver templates as shown below.

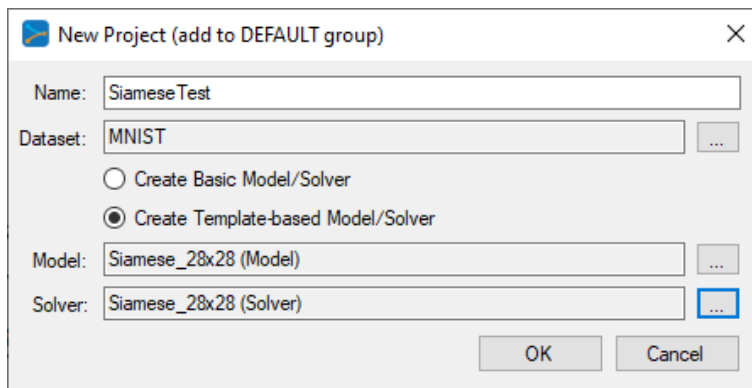
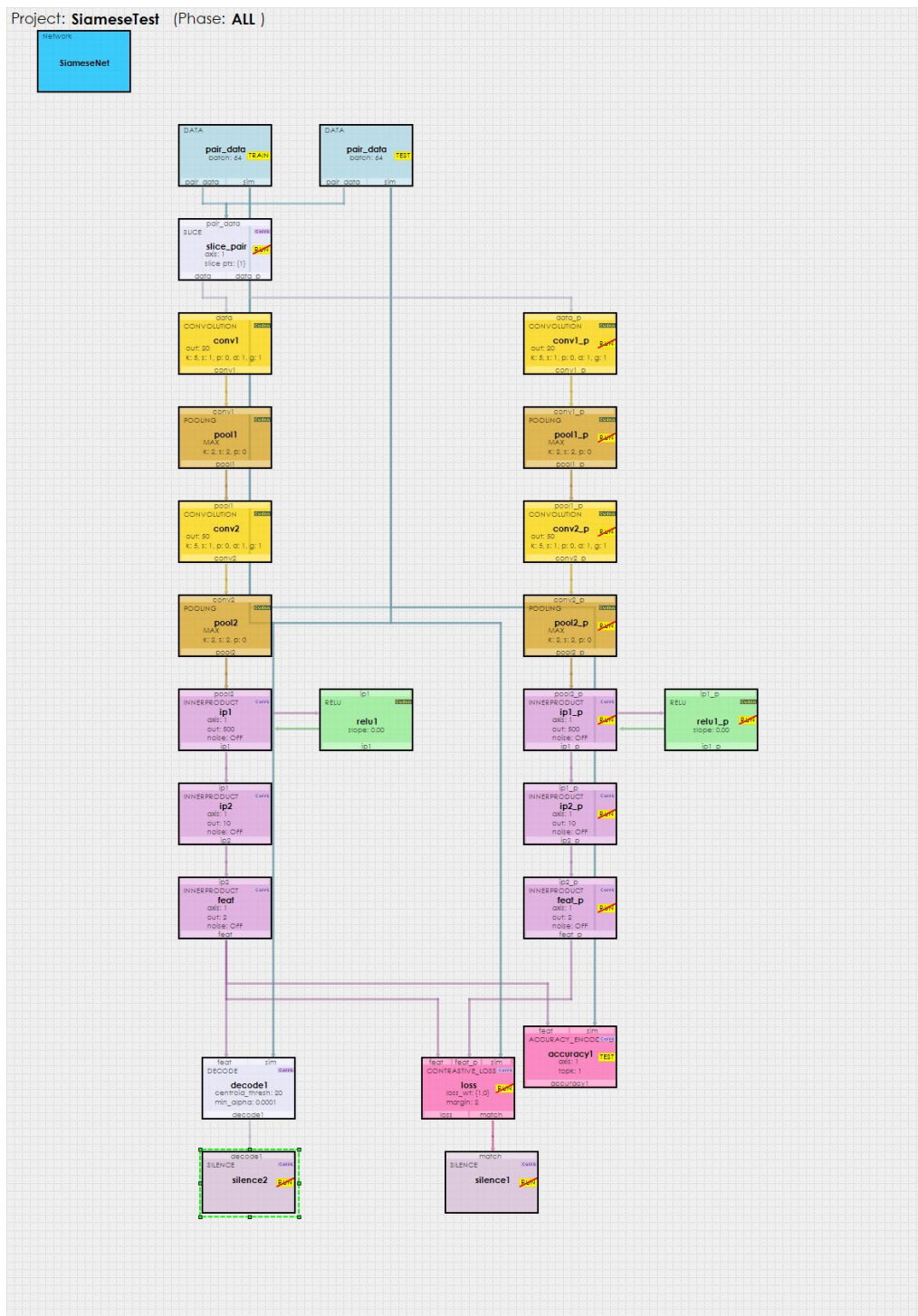


Figure 80 New Project Dialog

- 4.) After pressing OK, you will see the new SiameseTest project added to the Solutions window.

## FULL MODEL

To view your new model, expand the new 'SiameseTest' project and double click on its 'SiameseNet' sub-tree item. When opened in the model editor, the auto-encoders will look as follows:



For the full details on viewing your new model see the section [Opening Projects](#) above to open your new project, and see section [Training and Testing Projects](#) above to start testing your model.

As shown above, the Siamese Net comprises two separate, matching networks (on the left, and on the right) that both share the same weights. During training and testing, the DataLayer is configured to pack two images per channel which are then separated using the Slice layer into 'data' and 'data\_p'. The 'data' images are fed to the left network and the 'data\_p' are fed to the right.

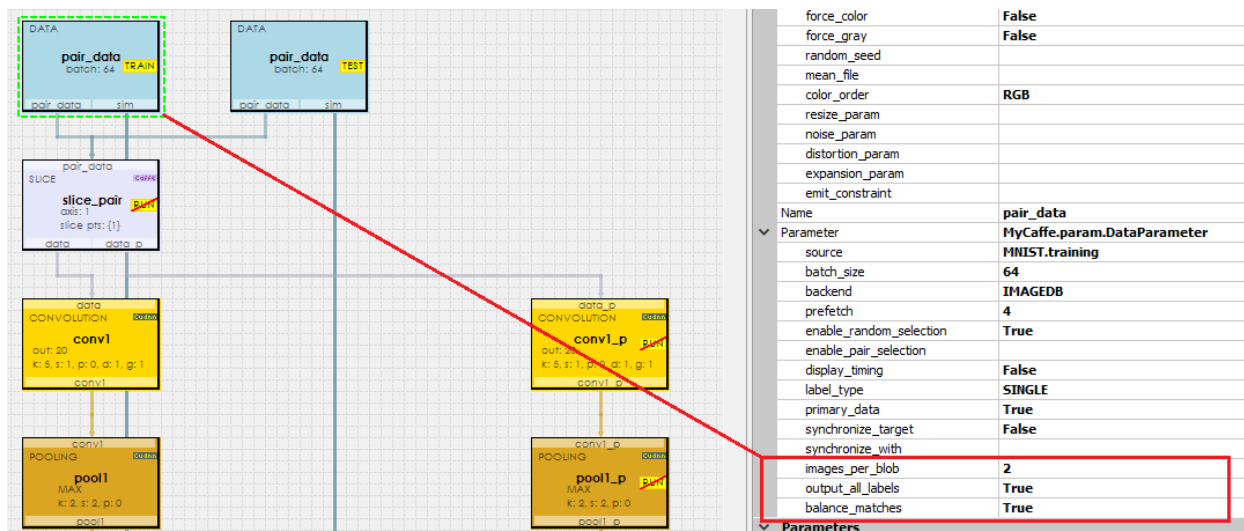


Figure 81 Siamese Data Layer

The 'images\_per\_blob' DataLayer setting directs the data layer to load two images per channel. With the 'balance\_matches' setting = True, images are loaded alternating between two images that are of the same class, and two images that are from two different classes. By balancing the matching and non-matching pairs, the network can learn the correct distance (small when images are the same, large when they are different) using the ContrastiveLoss layer. Also, note that each DataLayer is configured with 'output\_all\_labels' = True. The 'output\_all\_labels' directs the DataLayer to output each image's label in the same order that the images are packed into the channels. When false, the DataLayer just outputs the similarity of the two images where a value of '1' signifies that the images are of the same class and a value of '0' signifies that the images are of different classes.

Note, the model uses different phases where the two parallel networks are only both loaded during the TRAIN and TEST phases, whereas only a single network (the left network) is loaded during the RUN phase.

The following sections describe the difference between each phase.

## TRAINING AND TESTING PHASE

During the training phase, both the left and right networks output the learned encoding for each input image (originally split into the two images using the SliceLayer and fed into each network as described above). These two encodings are fed into the ContrastiveLoss layer along with the label of each Image (or the similarity value).

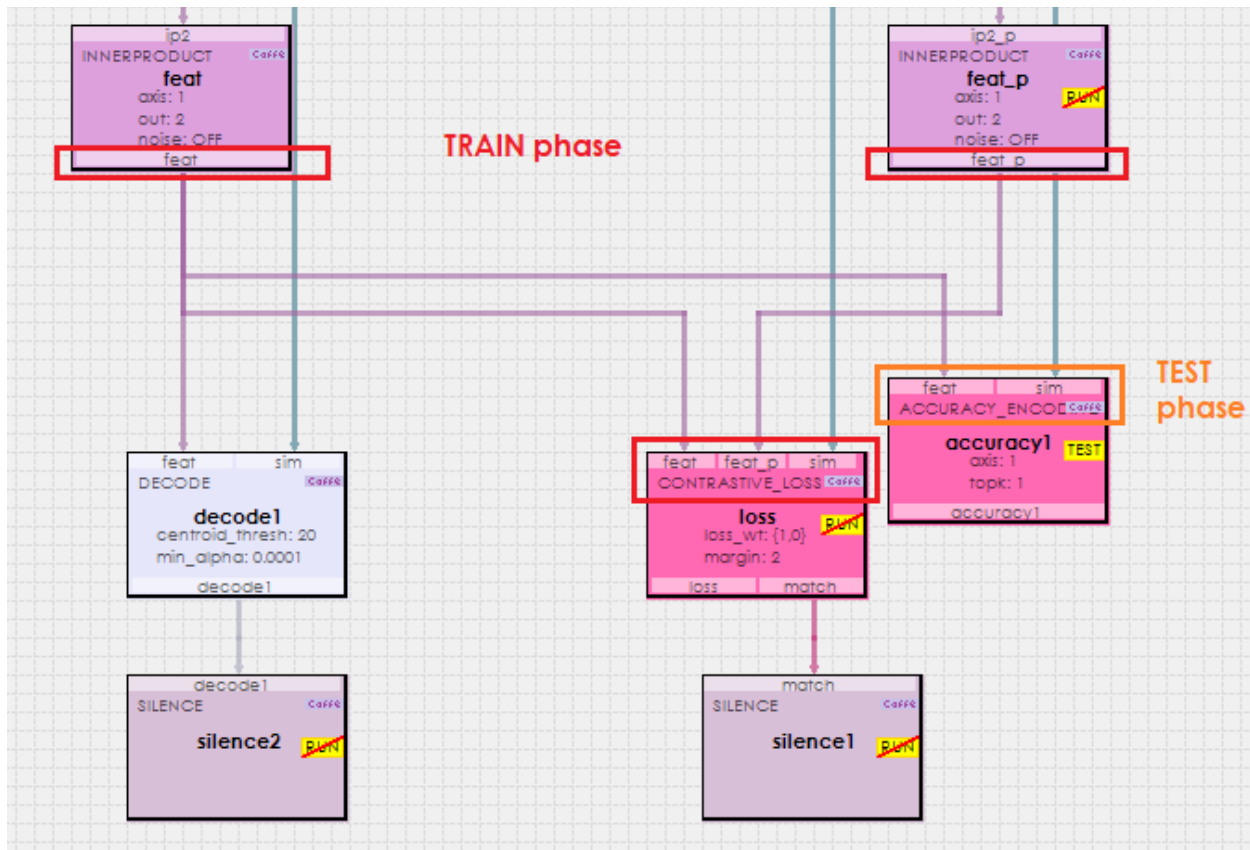


Figure 82 Siamese Net Training and Testing Phase

The ContrastiveLoss layer then calculates the distance between the two encodings and calculates the loss where the loss is set to the squared distance between the two images when they are from the same class, and the squared difference between the margin and the distance when they are from different classes, which moves the image encodings toward one another when they are from the same class and further apart when they are not.

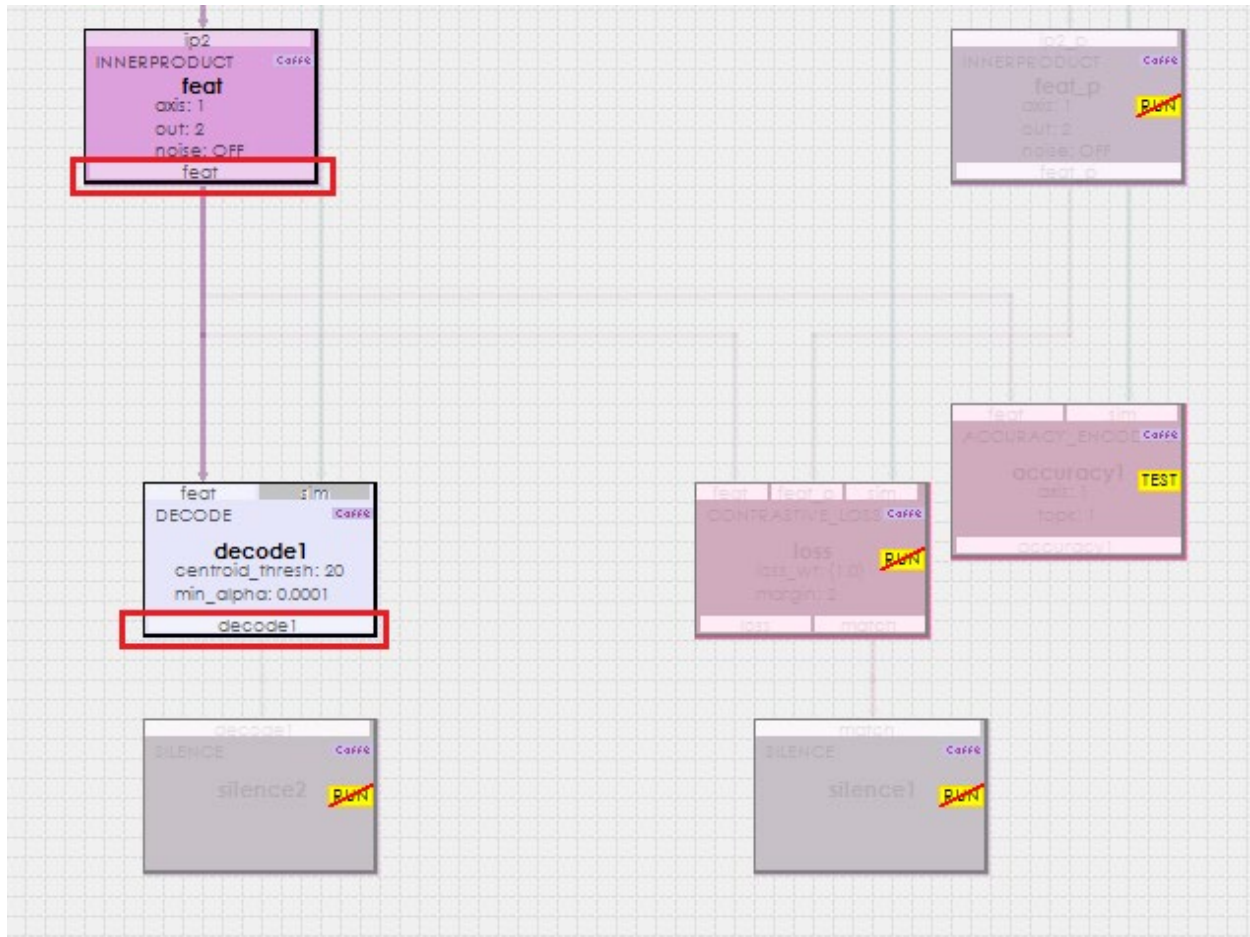
The DecodeLayer, later used when running the net in the RUN phase, must also run during the training phase, for while training, this layer calculates the encoding centroid of all classes. The encoding centroid is then stored in the DecodeLayer's learned parameters for later use during the RUN phase.

During the TEST phase, the encoding, and labels from the first net are fed into the AccuracyEncoding Layer which calculates the class based on the encoding centroid of each class.



## RUN PHASE

During the RUN phase, only one image is loaded into a single network (just the left side) which then produces the image's encoding. The image encoding is then fed into the DecodeLayer which calculates the distances of the encoding each class encoding centroid learned during the TRAIN phase.



### Figure 83 Siamese Net Run Phase

The DecodeLayer outputs the distances much in the same way that a SoftMax output probabilities, except in this case the smallest distance signifies the label whereas with a SoftMax the maximum probability signifies the label.

When using a one-shot learning application, each image is fed into the network which then produces a set of distances. If a very low distance is observed for a given class, then that class is determined to be the matching class. However, if no low distance is observed for any class, then the image is determined to NOT be in any of the classes.

## TRAINING AND TESTING

Training and testing are performed in the same manner used with other models as described in section [Training and Testing Projects](#) above.

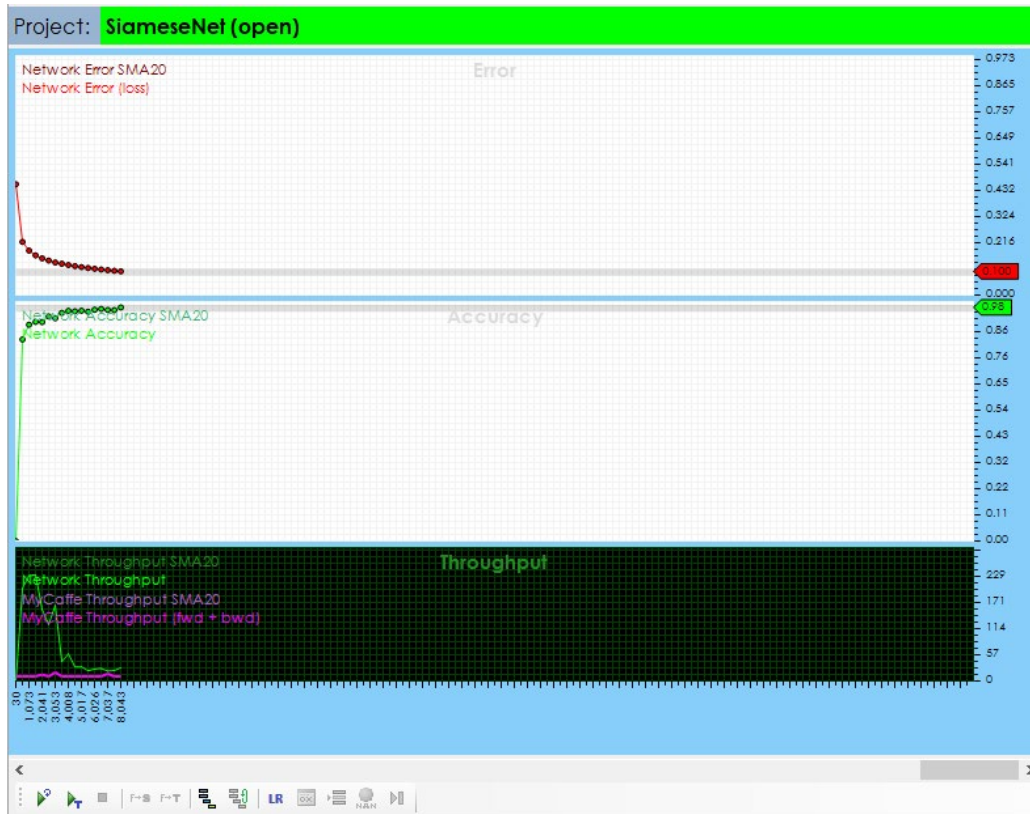


Figure 84 Training a Siamese net.

To test your model, select the 'Test Many' radio button and then press the 'Test' button. Once completed, the accuracy per label is output as follows.

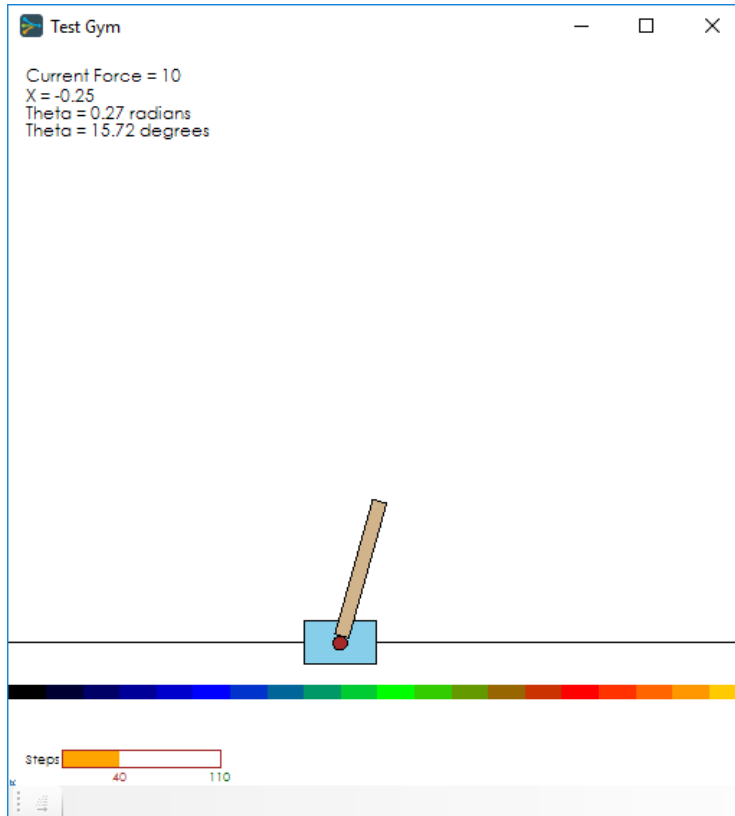
Output	
Show output from: DEFAULT-SiameseNet	
<div>Status Warnings Errors</div>	
Source	Message
DEFAULT-SiameseNet	Test Many Completed.
DEFAULT-SiameseNet	97.00% correct detections.
DEFAULT-SiameseNet	30.00 incorrect detections.
DEFAULT-SiameseNet	Label #0 had 98.11% correct detections out of 106 items with this label.
DEFAULT-SiameseNet	Label #1 had 100.00% correct detections out of 135 items with this label.
DEFAULT-SiameseNet	Label #2 had 94.31% correct detections out of 123 items with this label.
DEFAULT-SiameseNet	Label #3 had 96.84% correct detections out of 95 items with this label.
DEFAULT-SiameseNet	Label #4 had 93.90% correct detections out of 82 items with this label.
DEFAULT-SiameseNet	Label #5 had 97.47% correct detections out of 79 items with this label.
DEFAULT-SiameseNet	Label #6 had 100.00% correct detections out of 82 items with this label.
DEFAULT-SiameseNet	Label #7 had 94.50% correct detections out of 109 items with this label.
DEFAULT-SiameseNet	Label #8 had 96.81% correct detections out of 94 items with this label.
DEFAULT-SiameseNet	Label #9 had 97.89% correct detections out of 95 items with this label.
DEFAULT-SiameseNet	Testing completed.

Figure 85 Testing a Siamese Net

## POLICY GRADIENT REINFORCEMENT LEARNING

In this example, we will show how to use policy gradient-based reinforcement learning inspired by Andrej Karpathy [22] [23], to solve the Cart-Pole gym.

During training, each step of the Cart-Pole gym is run repeatedly until a failure occurs, which in Cart-Pole occurs if the angle of the pole exceeds  $\pm 20$  degrees, or the cart runs off the track to the right or left.



**Figure 86** Cart-Pole Gym

The following sections discuss how to create and train the project.

---

## DATASETS

The cart-pole datasets produce a dynamic set of data where on each step, four data items are produced:

- 1.) The position of the cart.
- 2.) The acceleration of the cart.
- 3.) The position of the pole (radians)
- 4.) And the acceleration of the pole.

This information is then used by the model along with the MyCaffe Reinforcement Learning Trainer to solve the cart-pole problem.



---

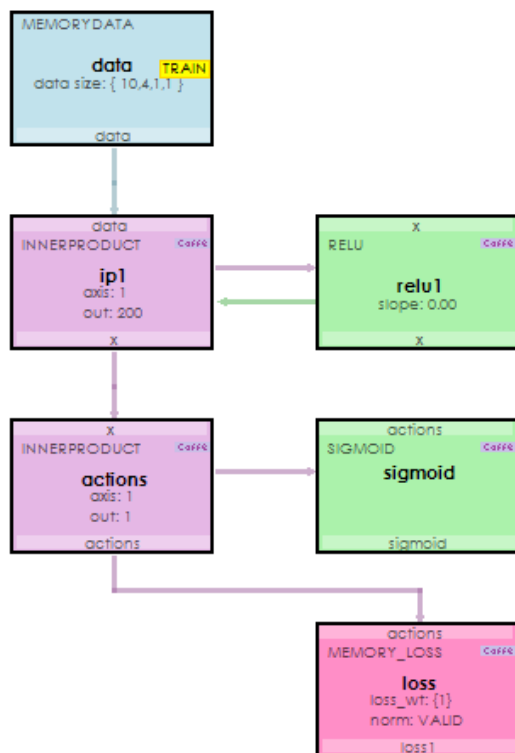
## POLICY GRADIENT MODEL

Two different models are used to solve the cart-pole problem, a Sigmoid based model and a Softmax based model.

---

### SIGMOID POLICY GRADIENT MODEL

The Sigmoid based policy gradient model is a very simple two-layer model consisting of two inner-product layers. Input data from the cart-pole gym is fed into the Memory Data layer and then the loss (and gradients) is calculated in the Memory Loss layer at the bottom of the model. The entire model is trained using the RMSProp solver along with the MyCaffe Reinforcement Learning Trainer which then trains the model loaded into the open MyCaffe project.



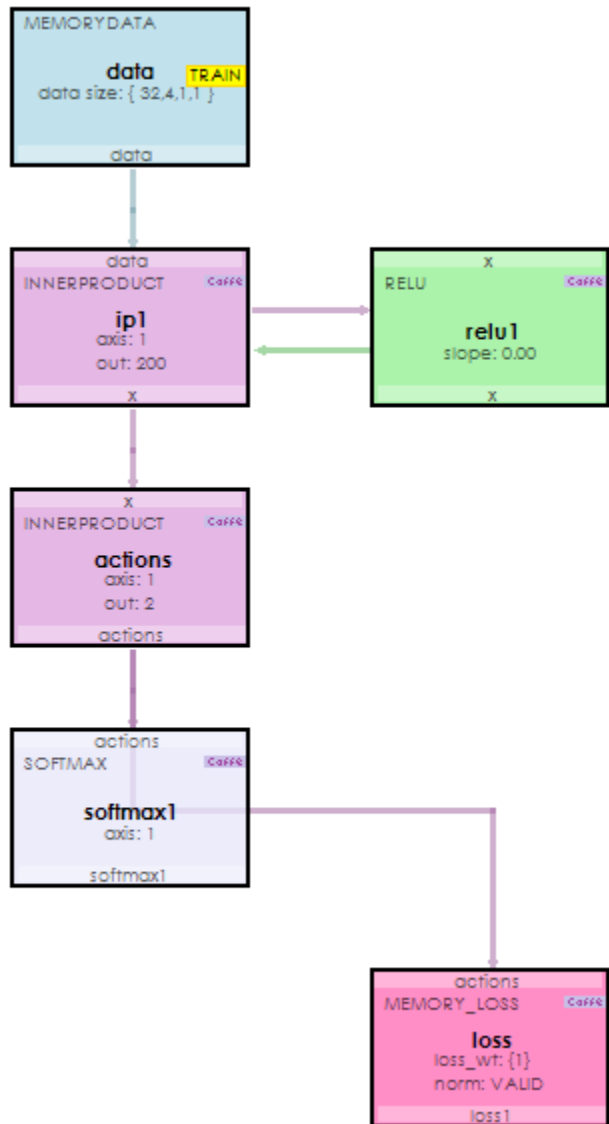
**Figure 87 Sigmoid Policy Gradient Model**

On each step, the sigmoid layer outputs a probability which is used to determine whether to move the cart to the left or right. This probability is also used to calculate the gradient that encourages to move toward the action that should have taken place [24].

---

## SOFTMAX POLICY GRADIENT MODEL

The Softmax based policy gradient model is also a very simple two-layer model consisting of two inner-product layers. Input data from the cart-pole gym is fed into the Memory Data layer and then the loss (and gradients) is calculated in the Memory Loss layer at the bottom of the model. However, the Softmax based model is not limited to two actions like the Sigmoid based model – instead the Softmax supports as many actions as you need. The same RMSProp solver is again used along with the MyCaffe Reinforcement Learning Trainer to train the model loaded into the open MyCaffe project.



**Figure 88 Softmax Policy Gradient Model**

On each step, the softmax layer outputs a probability which is used to determine whether to move the cart to the left or right. This probability is also used to calculate the gradient that encourages to move toward the action that should have taken place [24].

---

## SOLVER SETTINGS

The following solver settings are used to train the model.

**TrainerType=PG.MT**; this directs the trainer to use the policy gradient trainer.

**RewardType=VAL**; this directs the trainer to output the actual reward values. Alternatively, a setting of MAX returns the maximum reward.

**Gamma=0.99**; specifies the discount rate used on the rewards back into time.

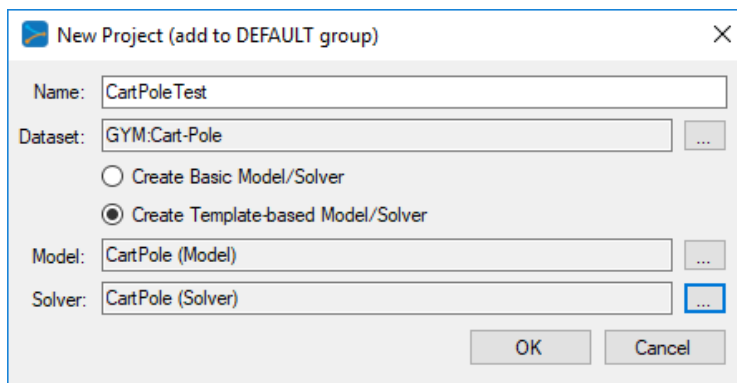
**Init1=10**; directs the trainer to apply +/- 10 for a force setting during training.

**Init2=0**; directs the trainer to use non-additive forces, when 1, each new force applied is added to the previous force used.

---

## TRAINING

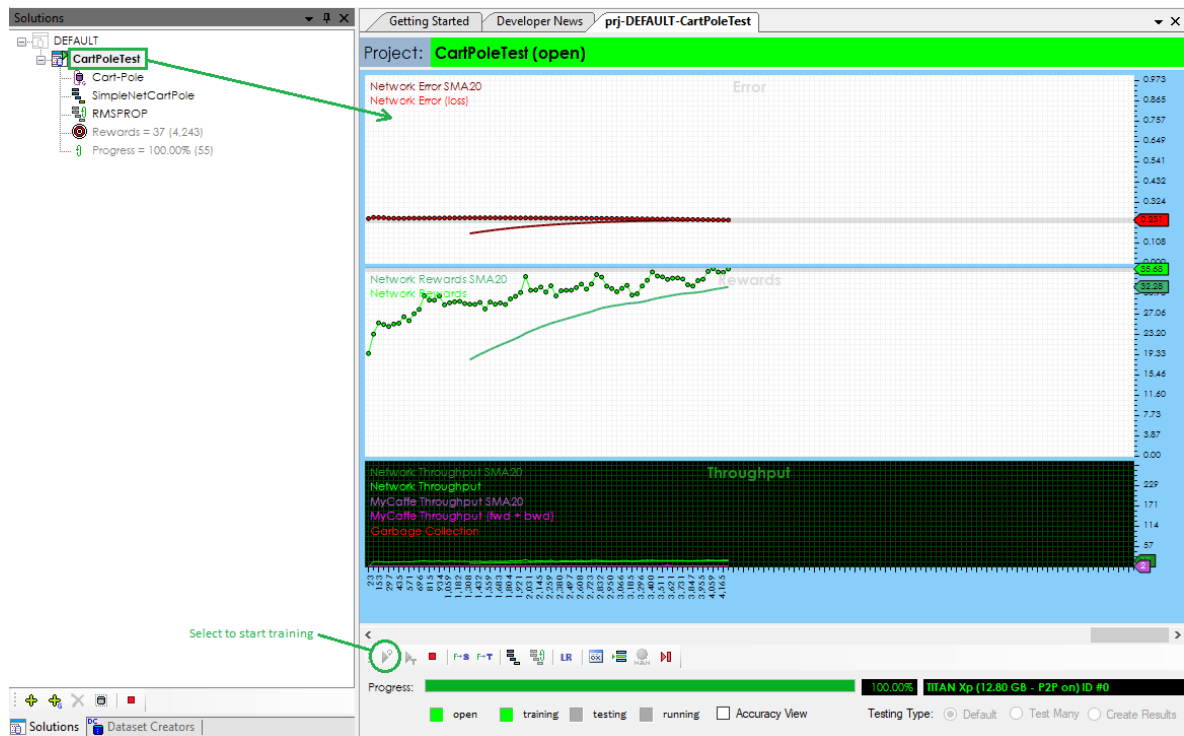
Before training, first create a new project with the Cart-Pole gym and model in it. To create a new project, select the *Add Project* (+) button at the bottom of the *Solutions* window.



**Figure 89 Creating the Cart-Pole Project**

From the *New Project* dialog, select the Cart-Pole gym and Cart-Pole model and solver and press *OK* to create the new project.

Open the project, by right clicking on the *CartPoleTest* project and selecting *Open* from the menu. Once, open, double click on the project name to open the *CartPoleTest* project and select the *Run Training* (🏃) button.



**Figure 90 Start Training Cart-Pole**

After a few minutes of training, you should see the rewards start to rise above 200 and continue to rise as you train. The higher the reward, the longer the algorithm was able to balance the pole. Run the training for an hour or so and you should see that the training is able to balance the pole for several thousand steps where each step takes place over about 20 milliseconds – so you should be able to balance the pole for a minute or more after training for a few hours.

To see the pole balancing in action, just double click on the *Cart-Pole* (🖱️) gym from within the CartPoleTest project – note you will need to do this while the project is training.

**NOTE:** When the gym window is open, the training steps slow down to 20 milliseconds per step to facilitate a 30 frames per second in the Gym window. To speed the training back up, just close the gym window.

## DEEP Q-LEARNING (DQN)

Dopamine is an open-source research project from Google that provides the Deep Q-Learning style of reinforcement learning via a DQN Agent [25]. This learning technique works well with Noisy-Nets by expanding the overall search scope by randomly altering the weights of the layer. In addition, a prioritized memory collection [26] can help focus the learning. MyCaffe offers DQN learning through the optional DQN trainer. Other alternative trainers include the PG (policy gradient) trainers such as the PG.MT.

Dopamine uses “the DQN architecture as a starting point” to add more complex DQN variants such as double DQN and the use of “Prioritized experience replay”. [25] DQN [27] [28] is a learning algorithm that uses an agent to train two networks: an ‘online’ network that is trained to learn the optimal action-value function (Q-values) produced by a second ‘target’ network.

The MyCaffe DQN and C51 trainers use modified versions of the Dopamine<sup>15</sup> DQNAgent and RainbowAgent both found on GitHub at:

DQNAgent:

[https://github.com/google/dopamine/blob/master/dopamine/agents/dqn/dqn\\_agent.py](https://github.com/google/dopamine/blob/master/dopamine/agents/dqn/dqn_agent.py)

RainbowAgent:

[https://github.com/google/dopamine/blob/master/dopamine/agents/rainbow/rainbow\\_agent.py](https://github.com/google/dopamine/blob/master/dopamine/agents/rainbow/rainbow_agent.py)

In addition, MyCaffe uses a Prioritized Replay Buffer [26] to optimize the sample selection of experiences used during training. The trainer is tuned further by using a model that employs two Noisy Dense layers which help optimize the solution search by randomly altering the weights of the layer.

The following sections discuss the NoisyNet model used to learn how to beat the ATARI game ‘breakout’ using the DQN trainer.

---

<sup>15</sup> The Dopamine open-source is licensed by Google under the Apache 2.0 License (<http://www.apache.org/licenses/LICENSE-2.0>) and made available on GitHub at <https://github.com/google/dopamine>.

## NOISYNET MODEL

The Noisy-Net [29] model uses one or more Inner Product layers that add noise to their weight and bias values. With this configuration, the solution space is searched via by randomly altering the learned parameters. Using the noisy-net supports numerous actions and is commonly used in conjunction with a Deep Q-Network Agent and prioritized memory collection.

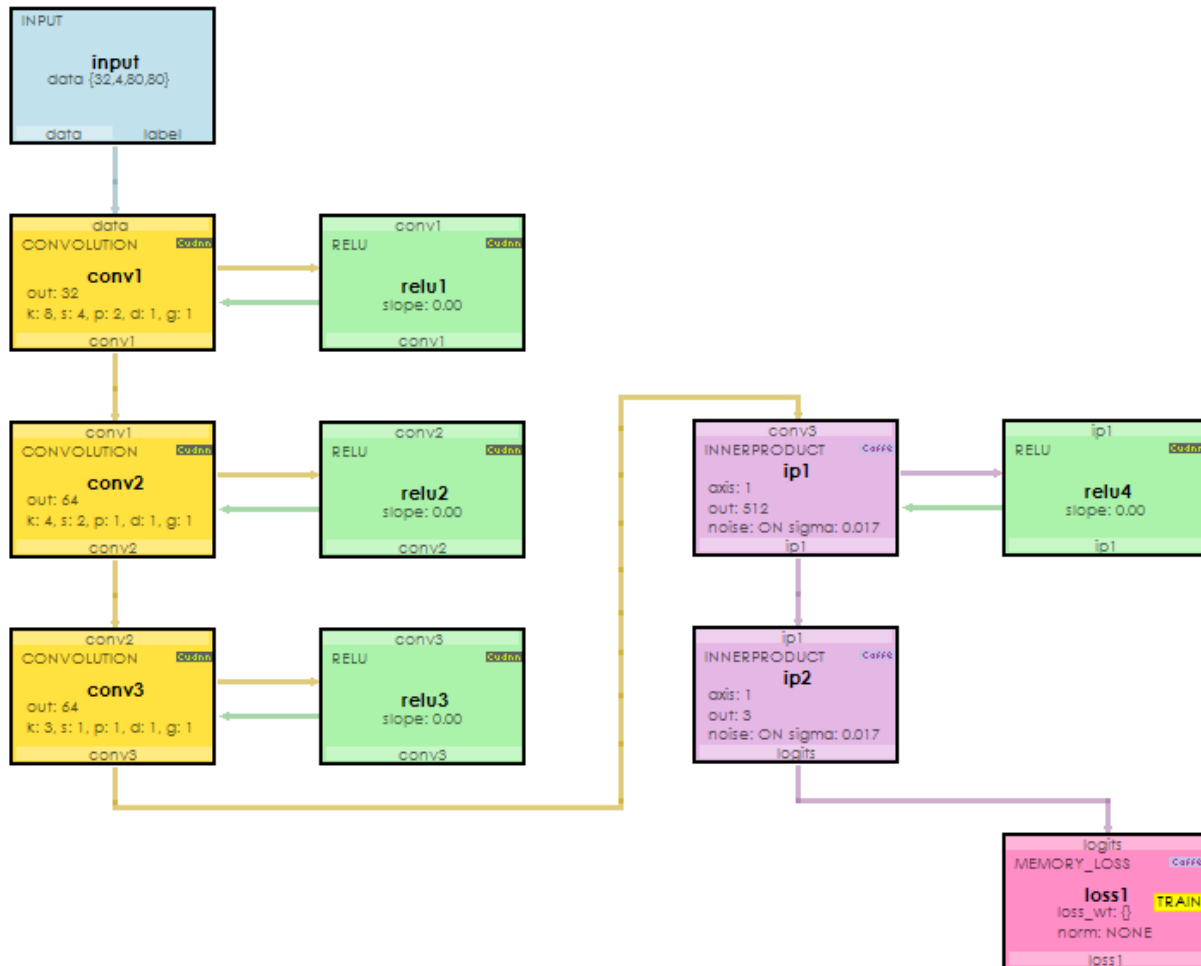


Figure g1 Noisy-Net Model

On each step the model outputs raw logits, however a softmax layer may also be used to convert the logits into probabilities. The memory loss layer calculates the gradients which are then back propagated through the model.

---

## SOLVER SETTINGS

The following solver settings are used to train the model.

**TrainerType=DQN.ST**; this directs the trainer to use the policy gradient trainer.

**RewardType=VAL**; this directs the trainer to output the actual reward values. Alternatively a setting of **MAX** returns the maximum reward.

**Gamma=0.99**; specifies the discount rate used on the rewards back into time.

**MiniBatch=1**; specifies the mini-batch over which gradients are accumulated.

**UseAcceleratedTraining=False**; specifies to both accumulate the gradients (when **MiniBatch>1**) and apply the gradients on each step thus doubling up on the gradient updates.

**AllowDiscountReset=False**; specifies to reset the discount on negative rewards.

**ValueType=BLOB**; specifies to use three dimensional data (h x w x c).

**InputSize=80**; specifies an input size (h & w) of 80.

**EpsStart=0.99**; specifies the starting percentage of randomly selected actions.

**EpsEnd=0.01**; specifies the ending percentage of randomly selected actions.

**EpsSteps=200000**; specifies number of steps to use randomly selected actions.

**FrameSkip=1**; specifies the number of frames to skip (1 = none) on each step.

**Preprocess=False**; specifies whether or not to preprocess values to 1 or 0.

**UseRawInput=True**; specifies to use the raw input as opposed to using a difference between the current and previous input.

**ActionForceGray=True**; specifies to force the data into a single channel.

**AllowNegativeRewards=True**; directs the Gym to allow negative rewards when a ball is missed.

**TerminateOnRally=True**; directs the Gym to force a termination state once a rally has completed as opposed to a game completing.

**GameROM=C:\Program~Files\SignalPop\AI~Designer\roms\breakout.bin**; specifies the ATARI game to play.

## TRAINING

Before training, first create a new project with the ATARI gym and model in it. To create a new project, select the *Add Project* (+) button at the bottom of the *Solutions* window.

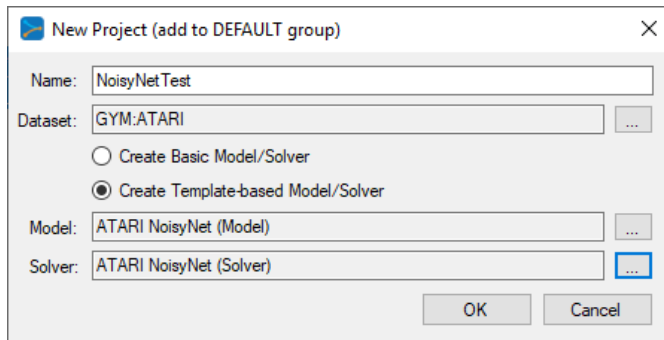


Figure 92 Creating the NoisyNet Project

From the *New Project* dialog, select the ATARI gym and ATARI NoisyNet model and solver and press *OK* to create the new project.

Open the project, by right clicking on the *NoisyNetTest* project and selecting *Open* from the menu. Once, open, double click on the project name to open the NoisyNetTest project and select the *Run Training* (🏃) button.

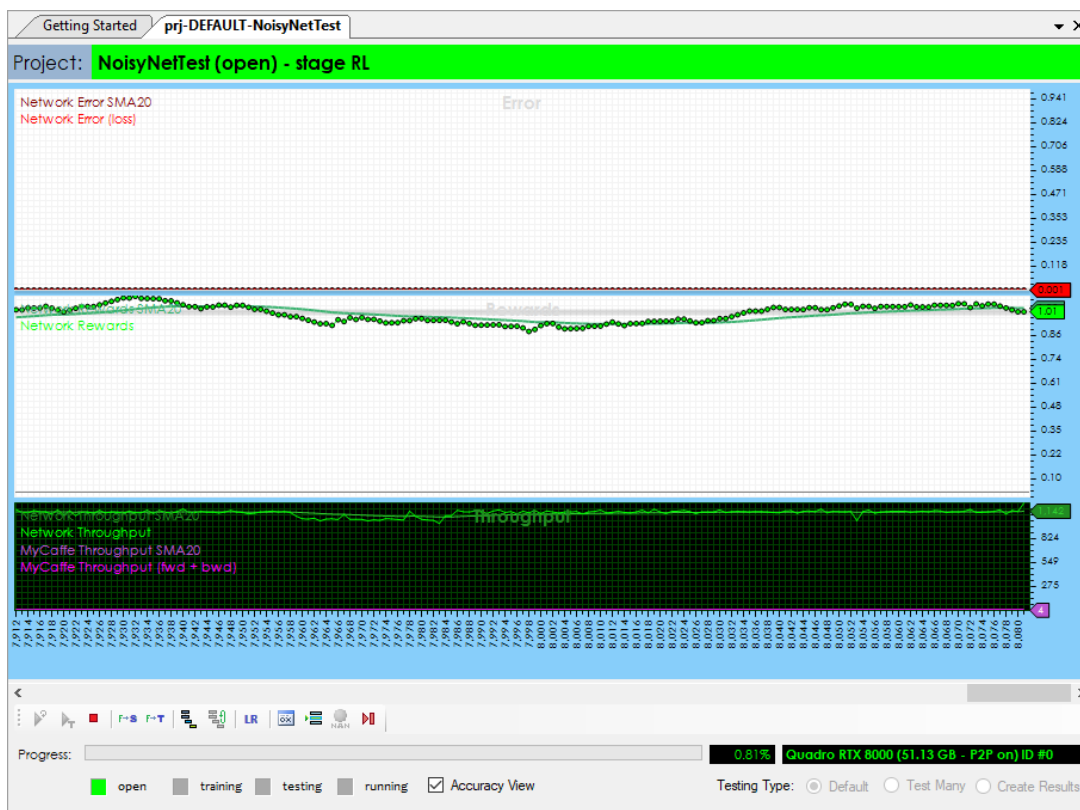


Figure 93 Training the NoisyNet



When training, the game progress is easily viewed by double clicking on the *ATARI* gym of the project which displays the Gym Window shown below.



**Figure 94 ATARI Gym - Breakout Game**

During training, an overlay is drawn on the game to show the actual action probabilities used during each step of the game.

## LSTM RECURRENT LEARNING

In this next model, we use LSTM based recurrent learning to learn Shakespeare as inspired by Andrej Karpathy [30] and adepierre [31]. With this example we demonstrate how to learn Shakespeare with both the LSTM layer [32] and the LSTM\_SIMPLE layer [33].

---

### SHAKESPEARE OUTPUT

Both the LSTM and LSTM\_SIMPLE layers learn and output Shakespeare like results, as shown below.

#### LSTM Output

```
The thought of his but is the queen of the wind:
Thou hast done that with a wife of bate are to the
earth, and straker'd them secured of my own to with the
more.

CORIOLANUS:
My lord, so think'st thou to a play be with thee,
And mine with him to me, which think it be the gives
That see the componted heart of the more times,
And they in the farswer with the season
That thou art that thou hast a man as belied.

PRONEES:
That what so like the heart to the adficer to
the father doth and some part of our house of the server.

DOMIONA:
What wishes my servant words, and so dose strack,
here hores ip a lord.

PARELLO:
And you are grace; and a singer of your life,
And his heart mistress fare the dear be readors
To the buse of the father him to the sone.

HOMITIUS ENOBARY:
And they are not a his wonders for thy greater;
But but a plotering pastice and every sirs.

PAPOLLES:
I will not as my lord, and the prince and house,
But that is scort to my wanter with her than.
```

Figure 95 LSTM output after 444k iterations.

## LSTM\_SIMPLE Output

Sir, he shall speak the duster his freender,  
And strengener the casse to this hearer,  
And see things and leaders of the stronge.

LAUCETHER:  
He do you the strange be the braces; thy some to the sense.

MAUNIA:  
And and wear the could to the strange of my more by for the meanting you this  
The walless before the some down to this presence to with the tonger;  
With the shall be not the which do come to be the good  
To the wearence of the earth, the consent me a stander  
That the beander with presently to me to the bearden,  
That with the weeples to and the priest of the  
biden hath the weeth and anster this bornes, my lord,  
The will strenge her and here the stare to that thy  
With of my faith, he thinks to make me like  
The foulder this nother his world be this nother.

LOCENTIO:  
What I do you to have a some of thee as the gentleman,  
And with you have the stelless and with thy groce  
Here the prominess fored for the fail fairer's live:  
The come some to the starest the things to some some  
True this

Figure g6 LSTM\_SIMPLE output after 600k iterations

As you can see, both models output Shakespeare, however, the LSTM model appears to learn some of the subtle nuances such as line spacing better than the LSTM\_SIMPLE model.

So how do each of these models work? The following sections go into more detail on the inner workings of the LSTM and LSTM\_SIMPLE layers.

When running

## LSTM LAYER

The LSTM layer was originally created by Donahue, et al. [32] and is now a part of the original C++ Caffe [2]. Two LSTM layers are used in the overall model shown below.

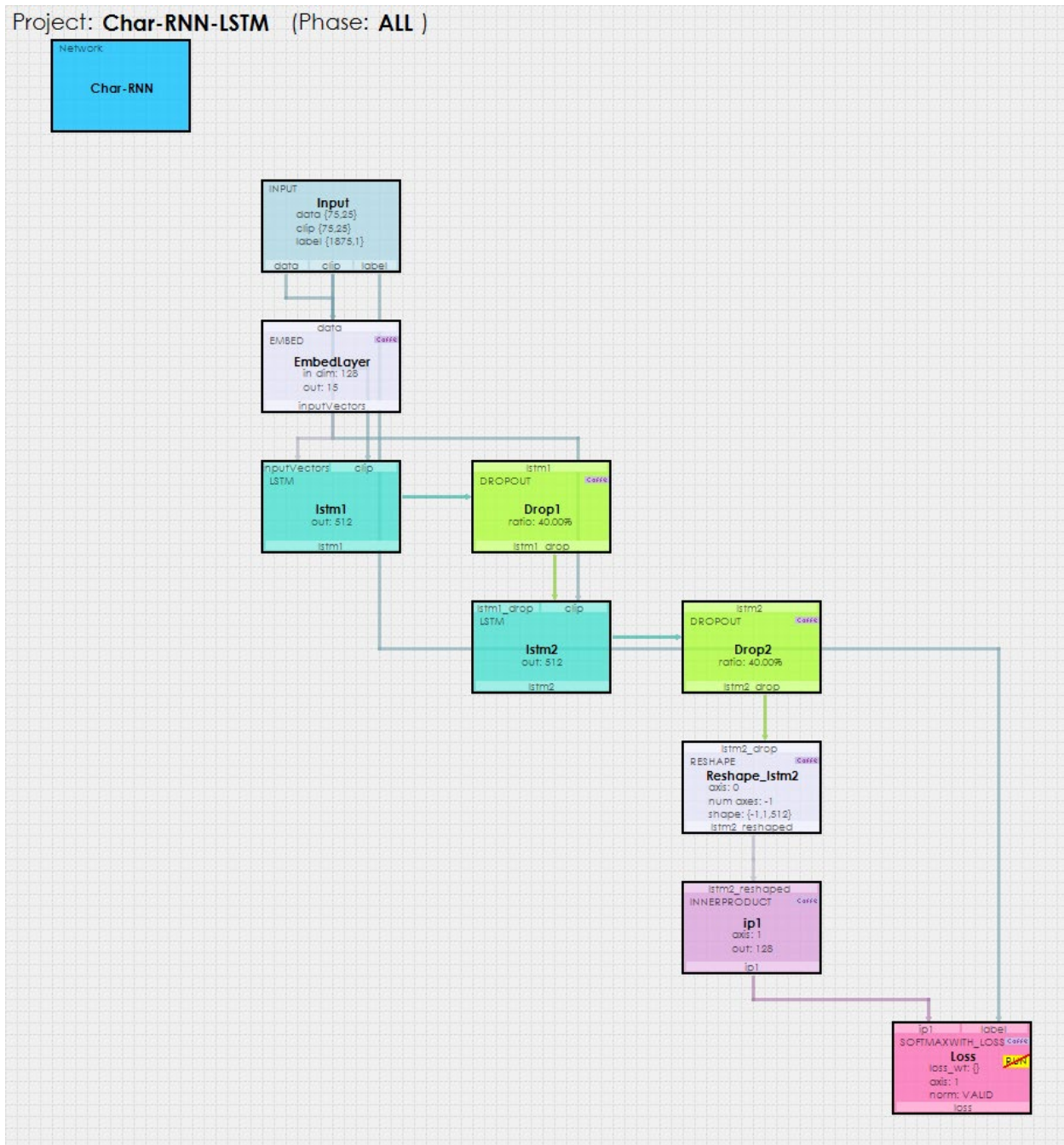


Figure 97 LSTM Model

The first LSTM layer is fed output from an EMBED layer used to transform each character within the range [1,128] into a 15-output embedding of the character. A set of 25 batches of 75 element sequences are embedded (using the EMBED) layer and then fed into the LSTM layer.

When feeding data into the LSTM layer, the following data ordering is used:

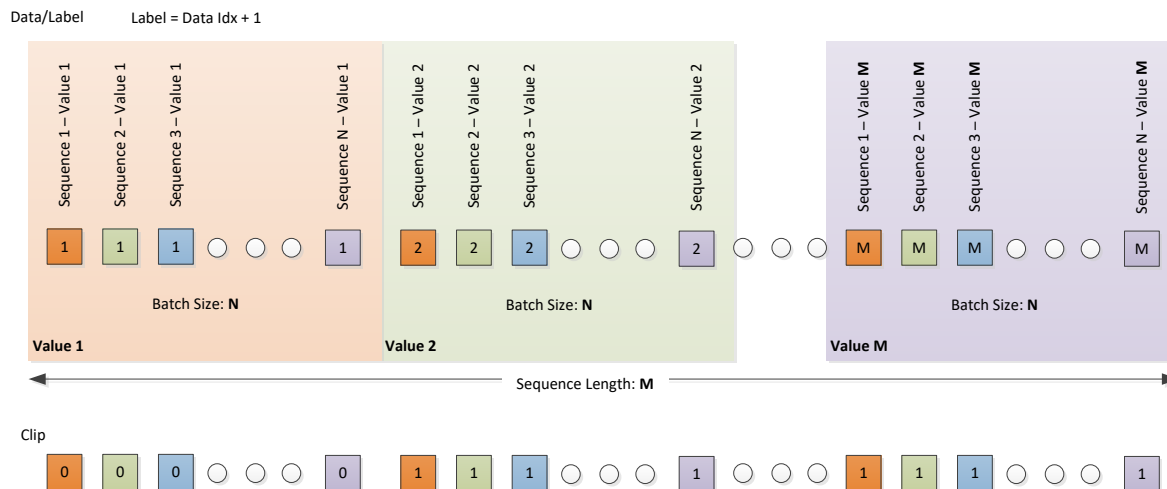


Figure g8 Data Input

As shown in the model above, the INPUT layer has three top values: data, clip, and label. The data and clip values eventually make their way to the LSTM layer (after embedding the data) and the label is used by the SOFTMAXWITH\_LOSS layer. When loading these values, each item in the label is set to the character 1 index past the character placed in the corresponding data item. The clip is set to zero (0) for the first item within a sequence and one (1) otherwise.

Note the EMBED input dim and INNER\_PRODUCT output dim is dynamically set to the actual vocabulary size, where the vocabulary size is determined by the number of unique characters found in the input data set.

Also note, that the actual character values are not used as input to the data and label blobs, but instead the index within the vocabulary of each character is used.

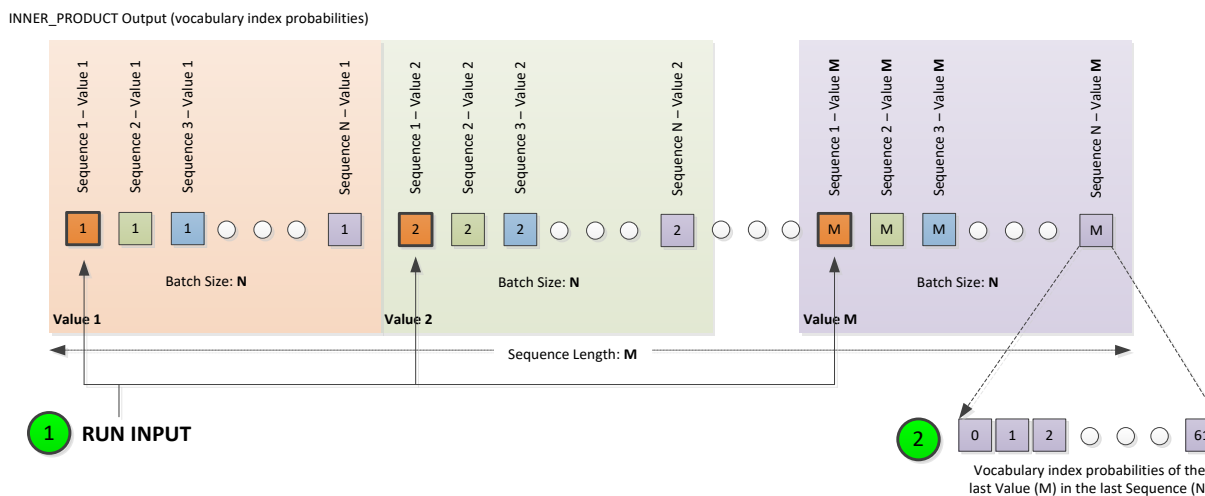


Figure g9 Run Input and Output

When running, a single sequence of characters is entered into the first sequence only with zero (o) in all other sequences. After running, the probabilities of the last value in the last sequence are used to determine the next character.

## LSTM LAYER INTERNALS

What occurs within each LSTM layer? When training or running, the LSTM layer constructs the unrolled network, which 'unrolls' the processing of each value within the sequence. Below, is the model of the unrolled network for a 75-item sequence – this type of unrolled network exists in each LSTM layer.

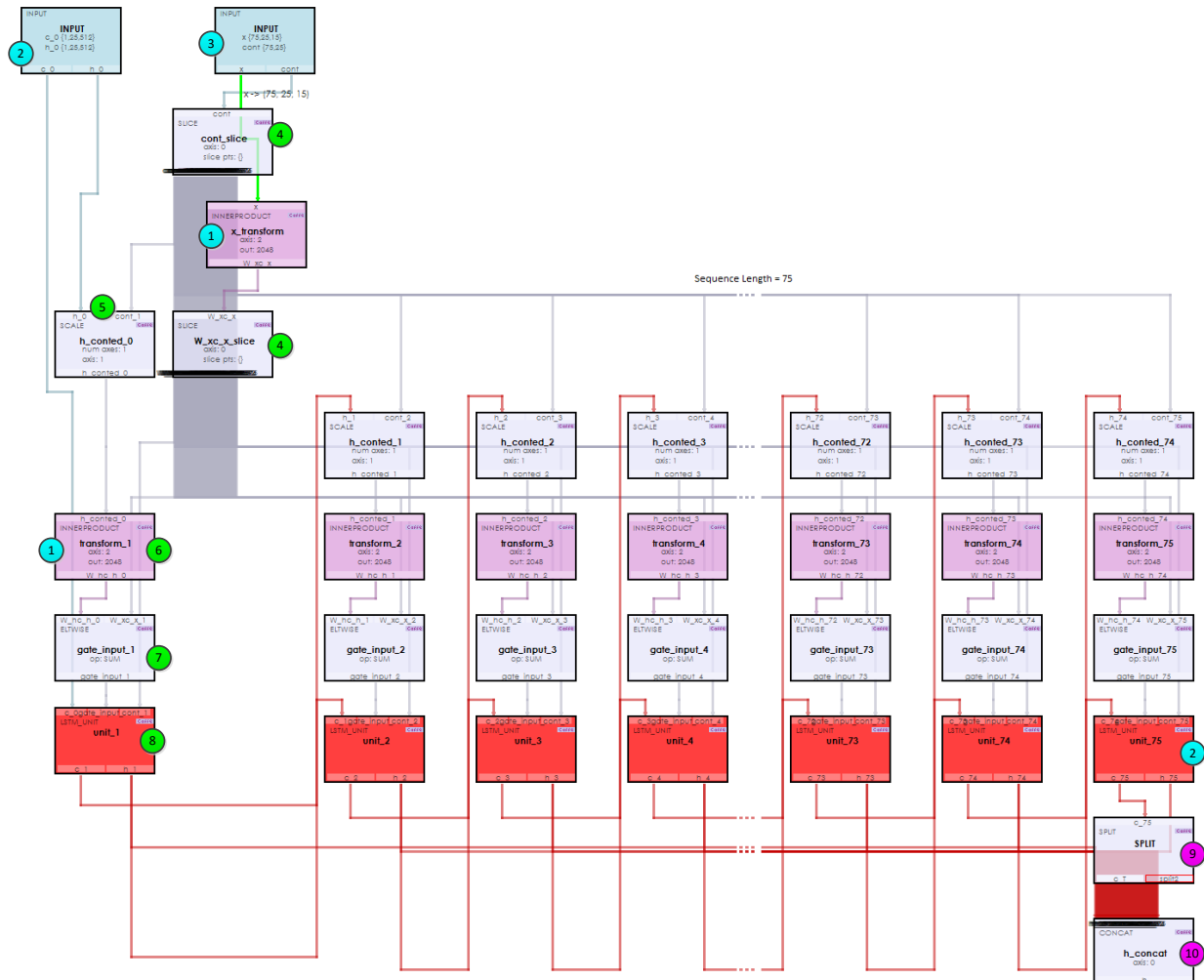


Figure 100 LSTM Internal Unrolled Network

When running through a forward pass on the LSTM layer, the following steps take place.

- 1.) During initialization, the weights are loaded into the **x\_transform** and **transform\_1** INNER\_PRODUCT layers.
- 2.) Next, the previous **c\_75** (split into **c\_T**) and **h\_75** values are copied into the start **c\_o** and **h\_o** INPUT layer values.

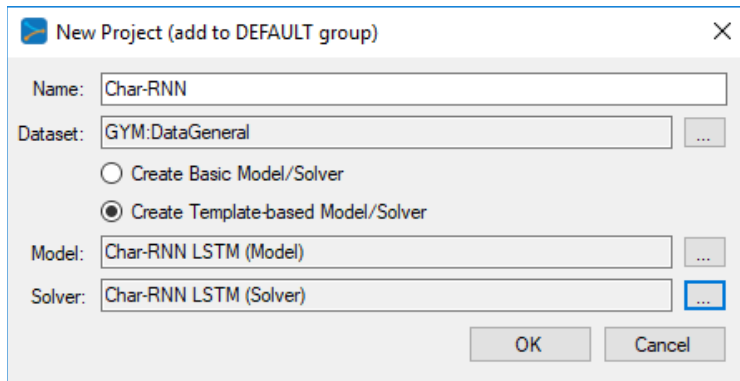
- 3.) The **data** inputs are then fed into the **x** values and the **clip** values are fed into the **cont** values within the second INPUT layer.
- 4.) During the forward pass, a SLICE layer splits each of the 75 sequence clip values into separate outputs named **cont\_1**, ... **cont\_75**. And the output of the **x** values fed through an initial INNER\_PRODUCT layer is also split into their individual sequence values and are named **W\_xc\_x\_1**, ... **W\_xc\_x\_75**.
- 5.) A SCALE layer multiplies (e.g., scales) the initial **h\_o** value by the first **cont\_1** value (which is the clip value for the first sequence value).
- 6.) An INNER\_PRODUCT layer then transforms the scaled result from 5 above to produce the **W\_hc\_h\_o** output.
- 7.) The result from 6 above is then summed with the slice of the **W\_xc\_x** output from 4 corresponding to the sequence element in play to produce the **gate\_input**.
- 8.) The **c** value, **gate\_input** value and **cont** value for this sequence element are next fed through the LSTM\_UNIT layer which applies the various gate transformations of Long-Short Term Memory. The process returns to step 5 and repeats until each of the sequence items have been processed (which in this case is 75 items).
- 9.) Upon completing processing, the 75-sequence item, the **c\_75** is split to produce the **c\_T** and **h\_75** values are then fed back into the INPUT layer described in step 2 above.
- 10.) The resulting **h\_o**, ... **h\_75** values are then concatenated together by a CONCAT layer to produce the output **h** value which is returned as **top[o]**. The **c\_T** value is returned as **top[1]**.

The LSTM model is a truly deep model. What appears to be a 9-layer model is a 617-layer model for each LSTM layer internally contains 305 layers within each of their unrolled networks.

---

## TRAINING LSTM

To set up this model and start training, first create the model. To create a new project, select the *Add Project* (+) button at the bottom of the *Solutions* window.

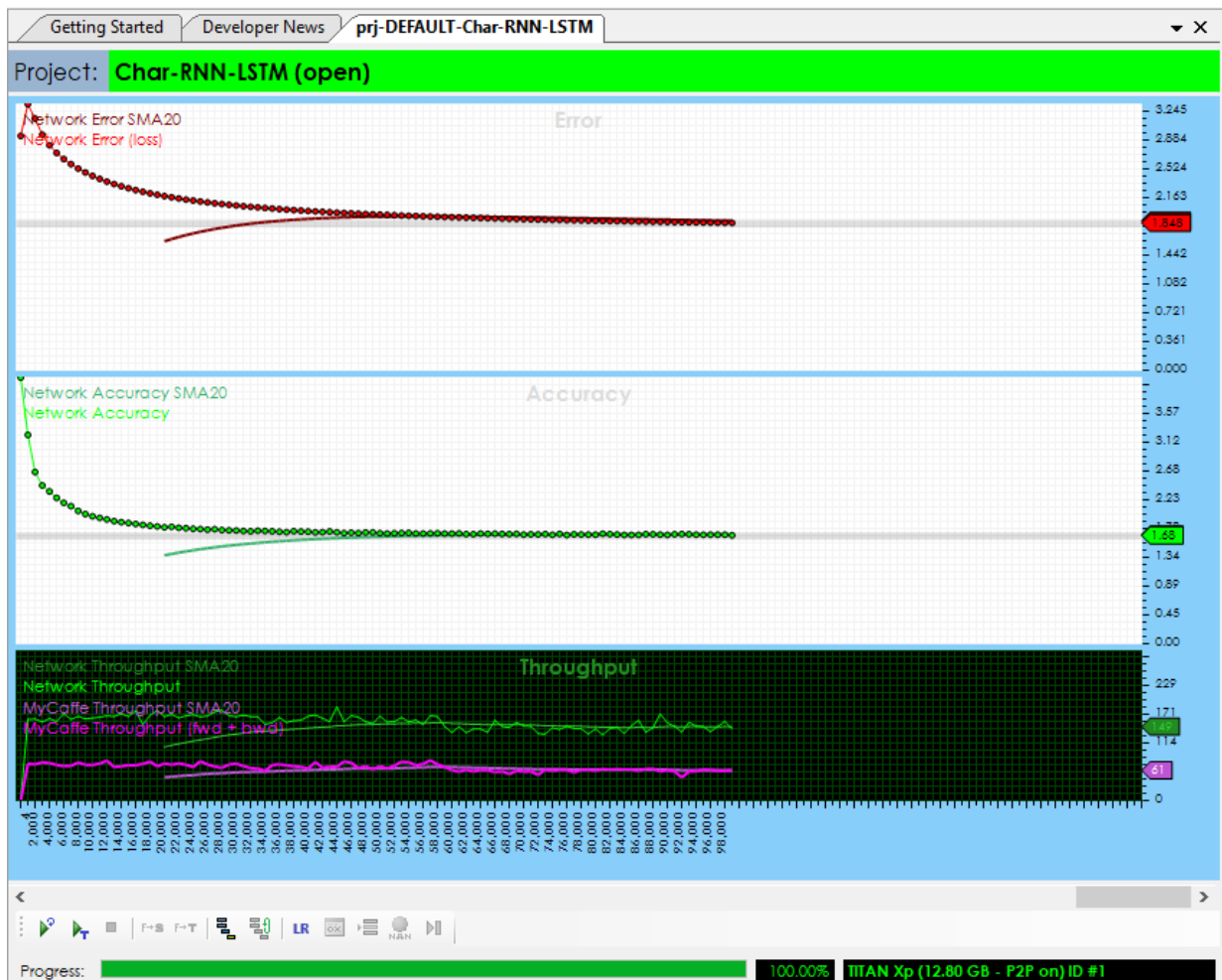


**Figure 101** Creating the Cart-Pole Project

From the *New Project* dialog, select the DataGeneral gym and Char-RNN LSTM model and solver and press *OK* to create the new project.

Open the project, by right clicking on the *Char-RNN* project and selecting *Open* from the menu. Once, open, double click on the project name to open the Char-RNN project and select the *Run Training* (🏃) button.





**Figure 102 Training LSTM**

Even with such a large network, the operations on a Titan Xp are very quick with each forward/backward pass completing in around 64 milliseconds or so – this really shows the power of GPU based deep learning!

## LSTM\_SIMPLE LAYER

The LSTM\_SIMPLE layer was originally created by *junhyukoh* [33] and takes a different approach to that of the LSTM layer in that the LSTM\_SIMPLE layer completes all operations without the use of a large unroll network. When using the LSTM\_SIMPLE layer, the model looks as follows.

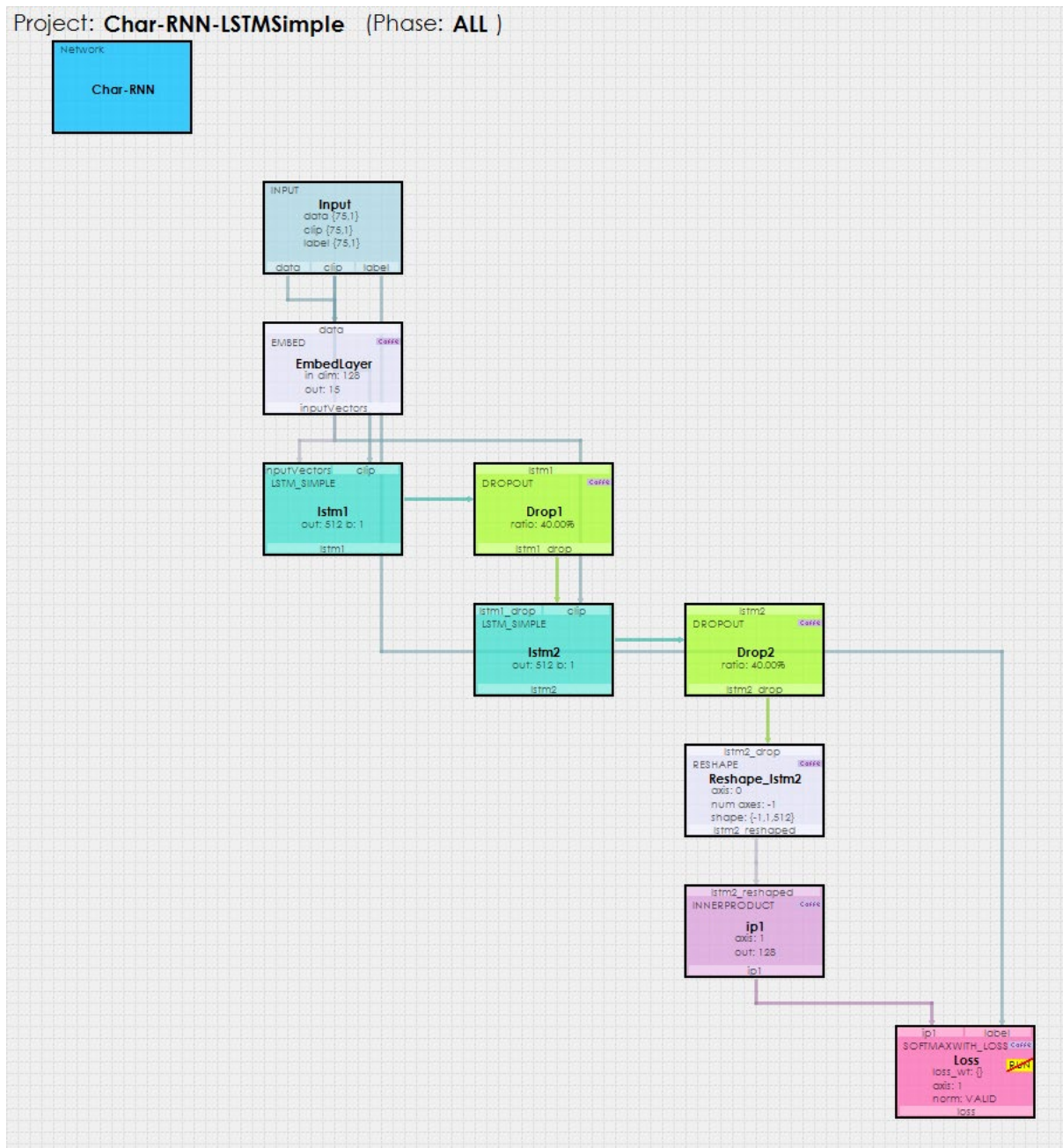


Figure 103 LSTM Model

As you can see the model is very similar to the model used by the LSTM layer with the main differences being in the data shapes input into the model and the use of the LSTM\_SIMPLE layer instead of the LSTM layer.

When feeding data into the LSTM\_SIMPLE layer, the following data ordering is used with batch size = 1.

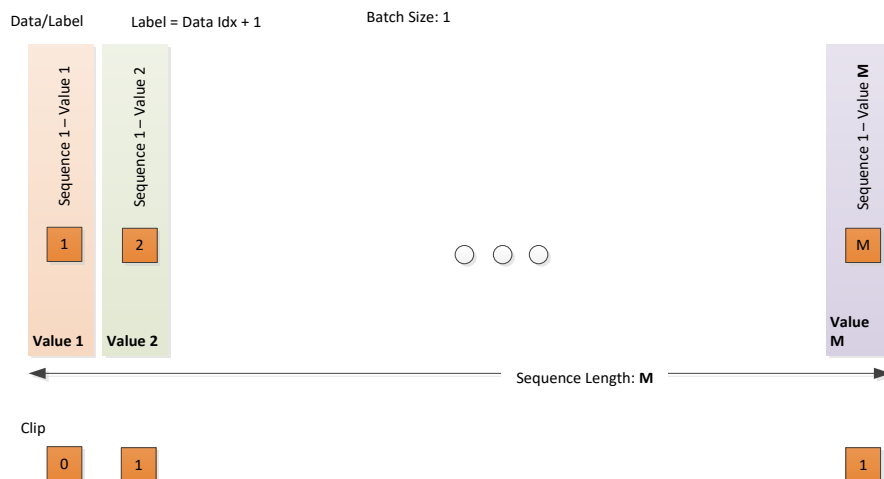


Figure 104 Data Input

As shown in the model above, the INPUT layer still has three top values: data, clip, and label. The data and clip values eventually make their way to the LSTM\_SIMPLE layer (after embedding the data) and the label is again used by the SOFTMAXWITH\_LOSS layer. When loading these values, each item in the label is set to the character 1 index past the character placed in the corresponding data item. The clip is set to zero (0) for the first item within a sequence and one (1) otherwise.

Note the EMBED input dim and INNER\_PRODUCT output dim is dynamically set to the actual vocabulary size, where the vocabulary size is determined by the number of unique characters found in the input data set.

Also note, that the actual character values are not used as input to the data and label blobs, but instead the index within the vocabulary of each character is used.

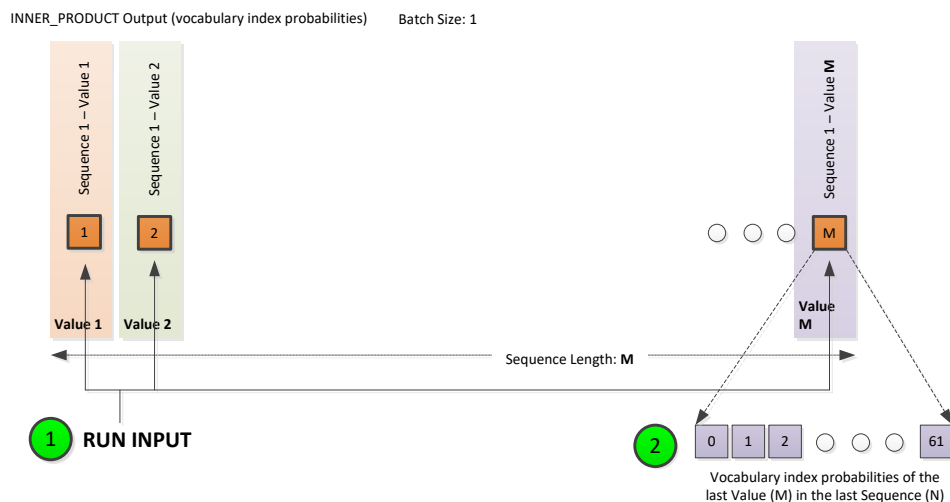
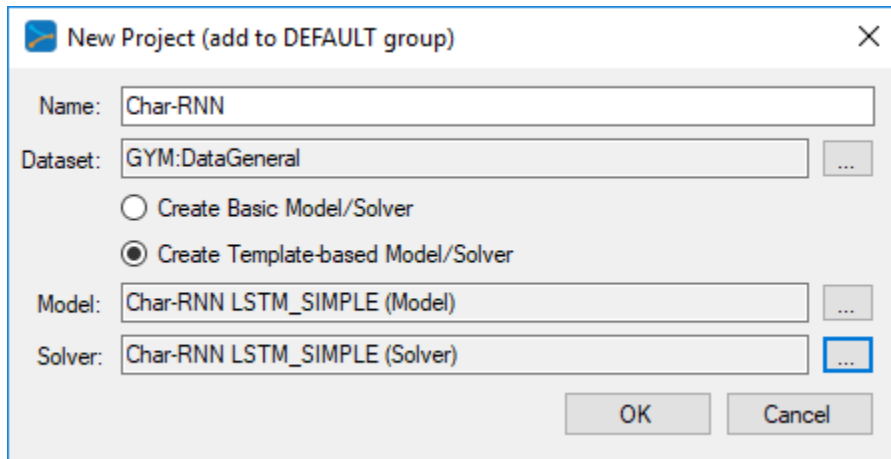


Figure 105 Run Input and Output

---

## TRAINING LSTM\_SIMPLE

To set up this model and start training, first create the model. To create a new project, select the *Add Project* (+) button at the bottom of the *Solutions* window.



**Figure 106** Creating the Cart-Pole Project

From the *New Project* dialog, select the DataGeneral gym and Char-RNN LSTM\_SIMPLE model and solver and press *OK* to create the new project.

Open the project, by right clicking on the *Char-RNN* project and selecting *Open* from the menu. Once, open, double click on the project name to open the Char-RNN project and select the *Run Training* (▶) button.

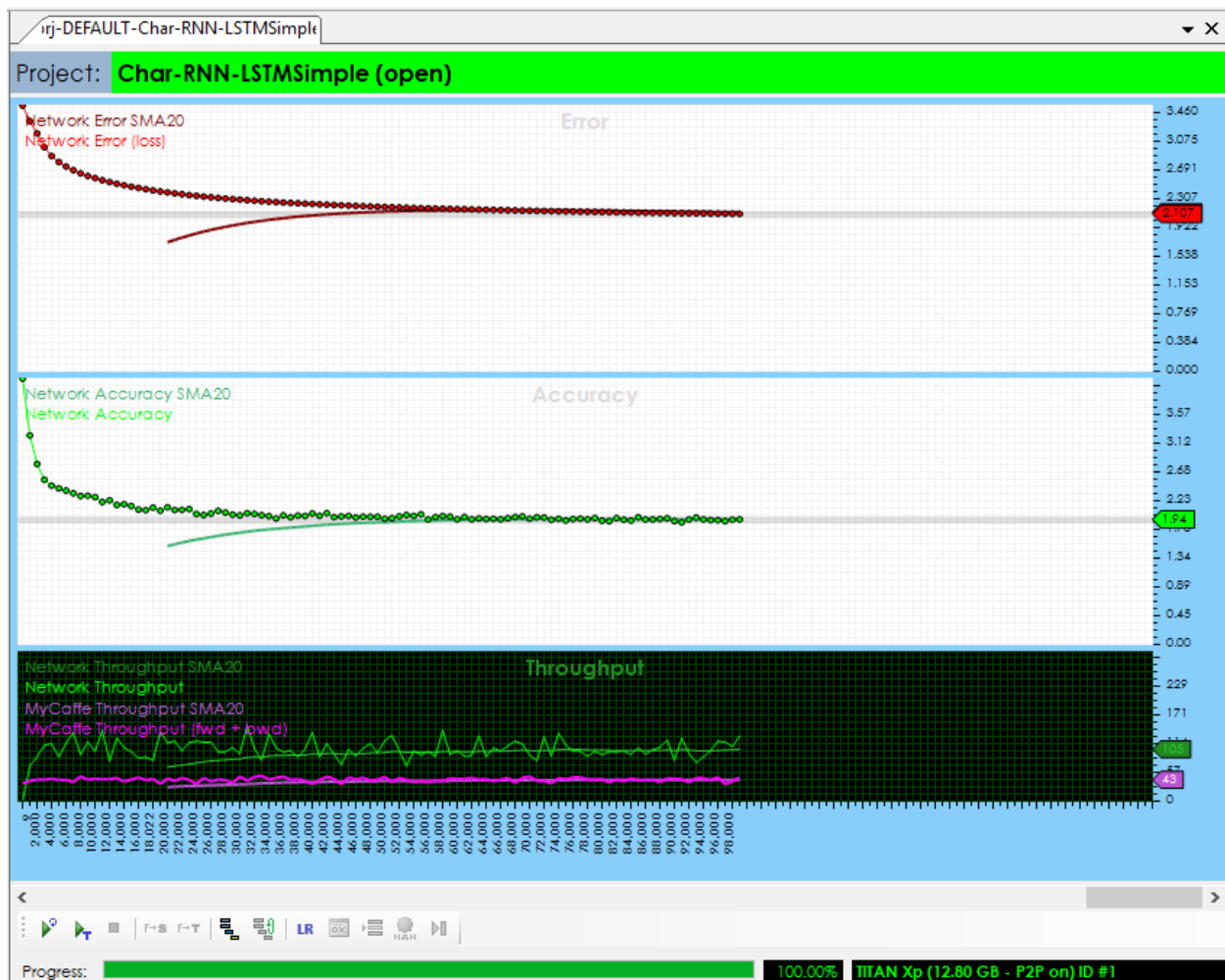


Figure 107 Training LSTM\_SIMPLE

As expected, the LSTM\_SIMPLE with a batch size = 1, runs about 30% faster than the LSTM, however, the learning is not as smooth and as noted above, the model may miss important details picked up by the LSTM layer.

## ENCODER-DECODER TRANSFORMERS (CHATGPT LIKE)

The encoder/decoder transformer models, initially introduced by Vaswani et al. [34], are also like the models used to power the popular ChatGPT service from OpenAI. These models comprise an encoder side that learns the probability distribution of the query or first language, and a decoder that first learns the probability distribution of the response or second language, then learns to map the two probability distributions via a multi-headed attention layer.

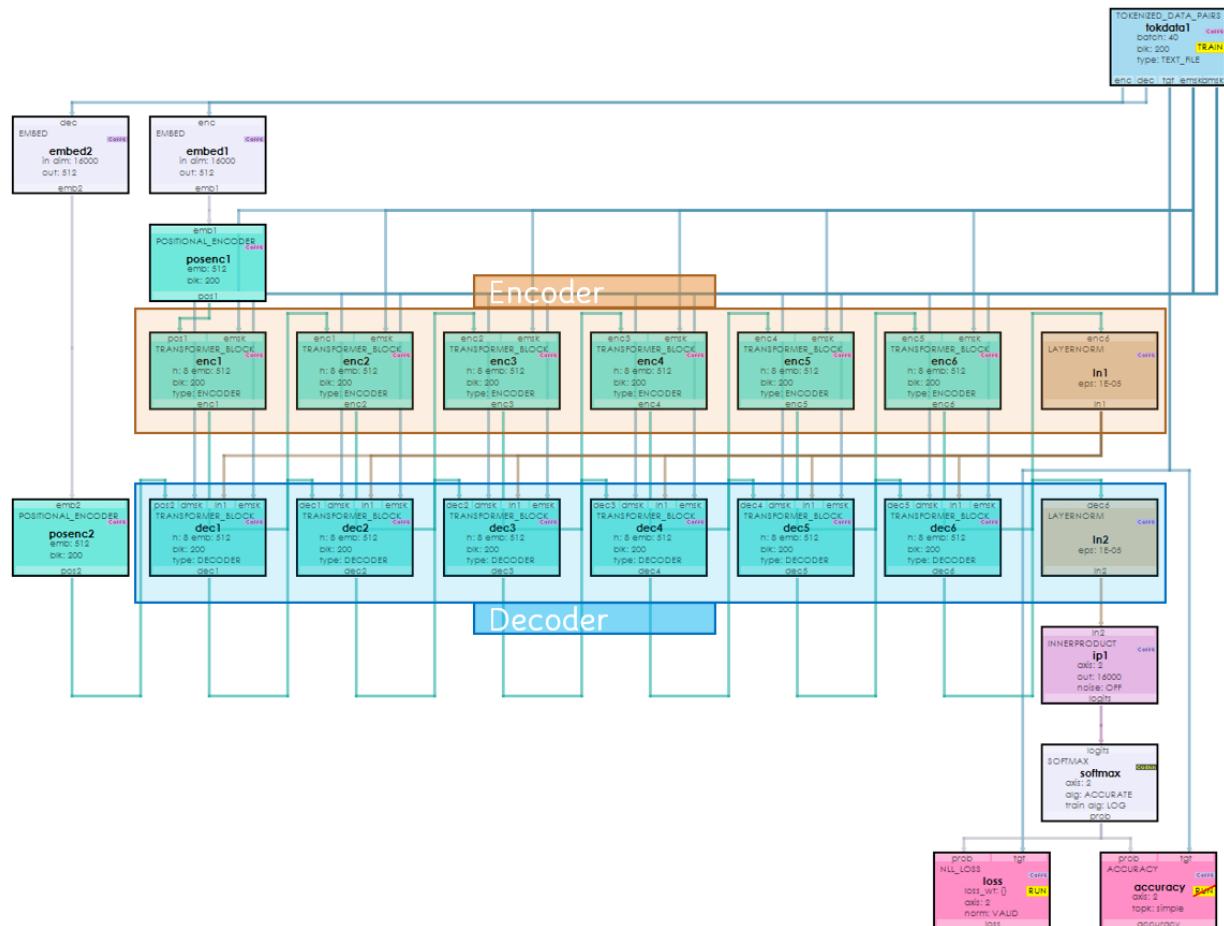


Figure 108 Encoder/Decoder Transformer Model

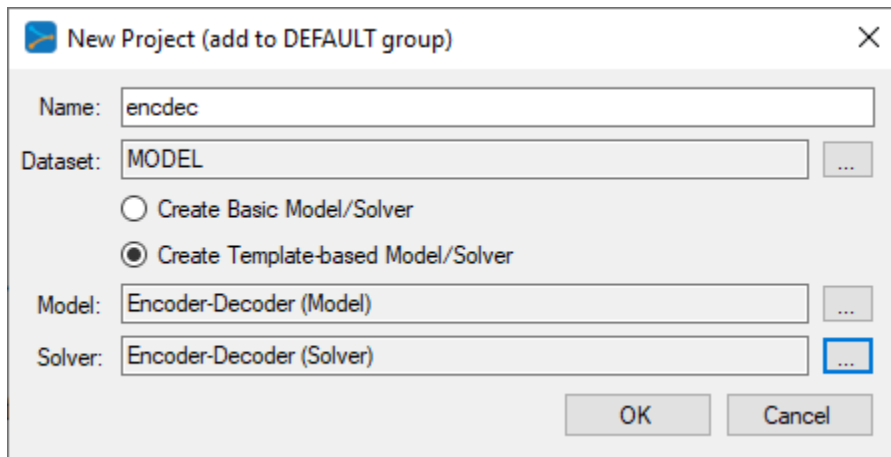
The Encoder/Decoder Transformer model is also useful for language translation, and the following steps describe how to create and train a model to translate from English to French.

---

## CREATE THE TRANSFORMER MODEL

Using the pre-created encoder/decoder transformer model template, we can easily create a new model for language translation.

To create a new project, select the *Add Project* (+) button at the bottom of the *Solutions* window.



**Figure 109** Creating the Cart-Pole Project

From the *New Project* dialog, select the MODEL dataset type and Encoder-Decoder model and solver and press *OK* to create the new project.

Open the project, by right clicking on the *encdec* project and selecting *Open* from the menu. Once, open, double click on the project name to open the *encdec* project and select the *Run Training* (▶) button.

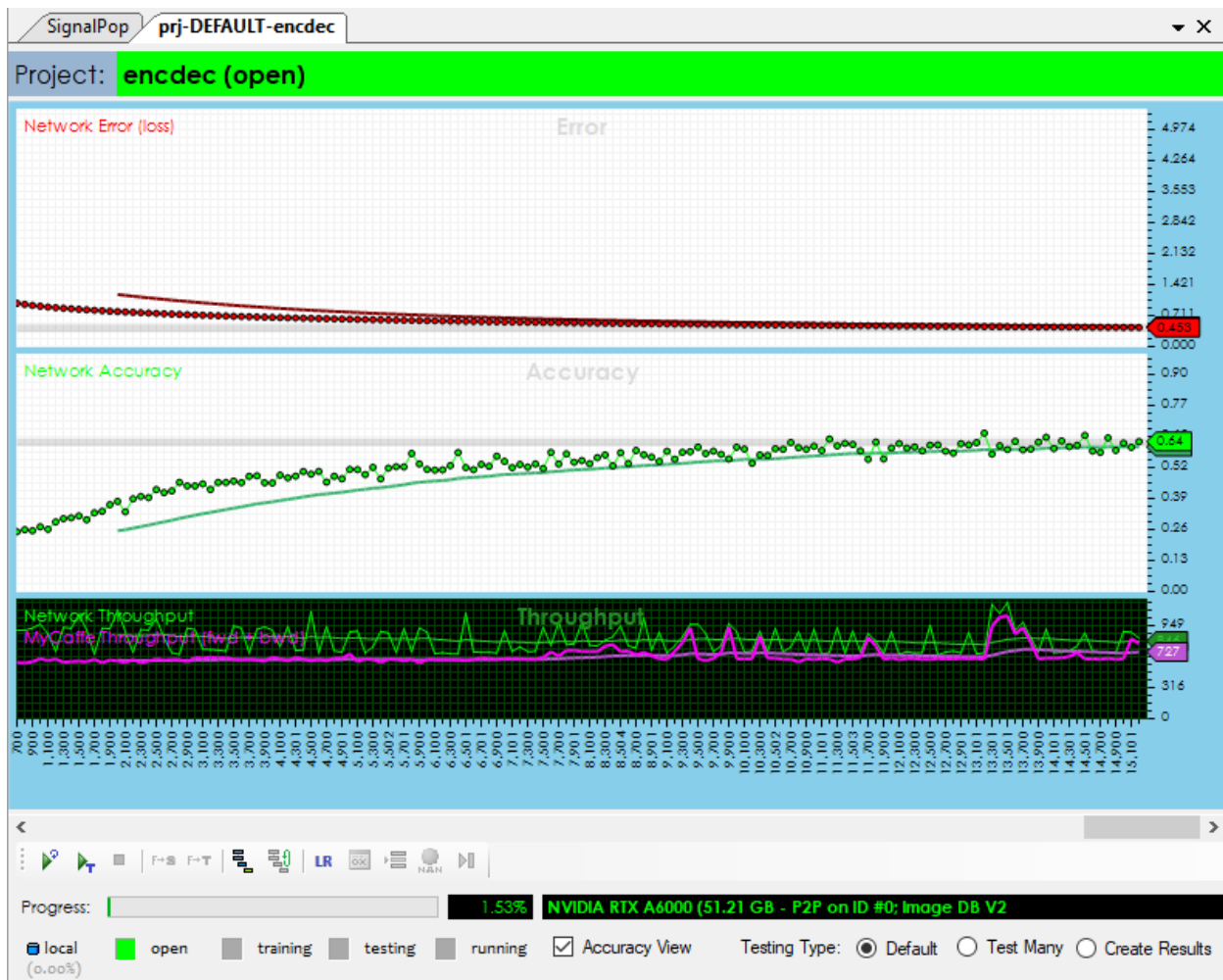



Figure 110 Training the Encoder/Decoder model for Language Translation

After training for around 15,000 iterations, you are ready to test the model. To test the model, select the *Test Many* radio button in the lower right corner of the project window. Next, select the *Test* (  ) button to start testing.

The Test Many dialog will display, allowing you to enter the English language you want translated.

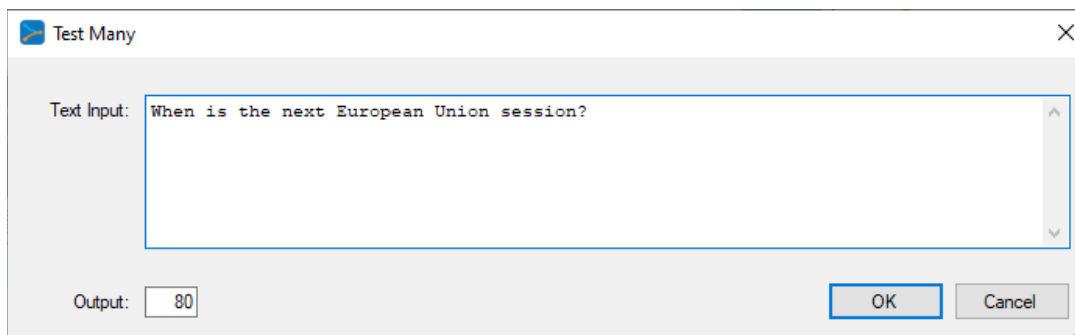


Figure 111 Test Many Input Dialog Box



After pressing, OK, the model is run in an inference mode over the inputs to produce the translated text which is output in the results window.

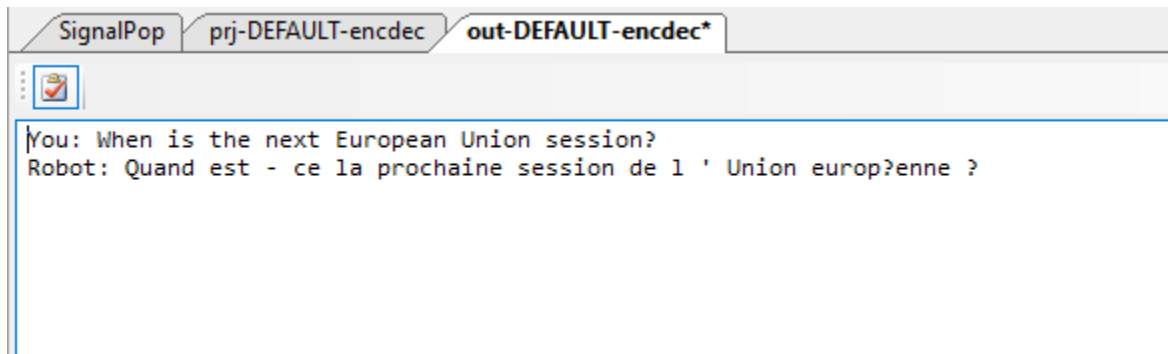


Figure 112 Test Many Results

To check your results, you can pass the same resulting output into Google Translate and verify the English translation with your inputs.

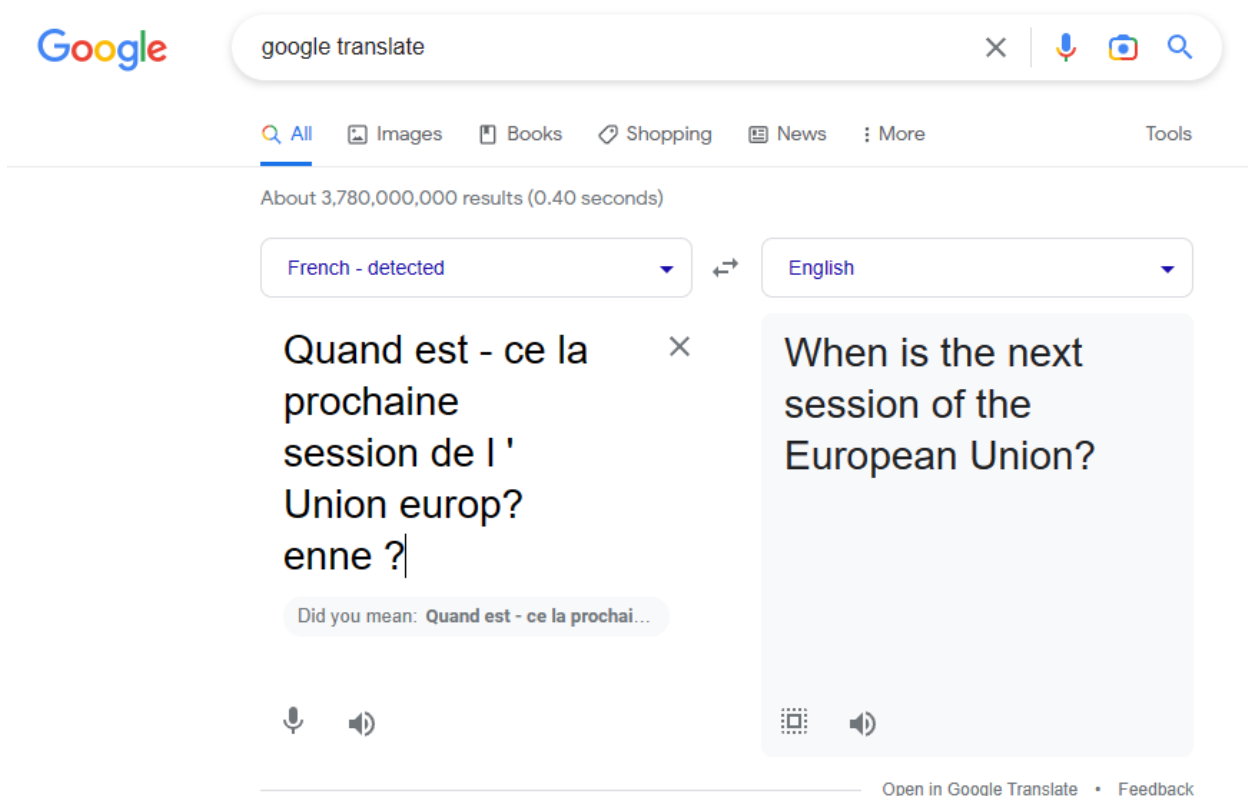
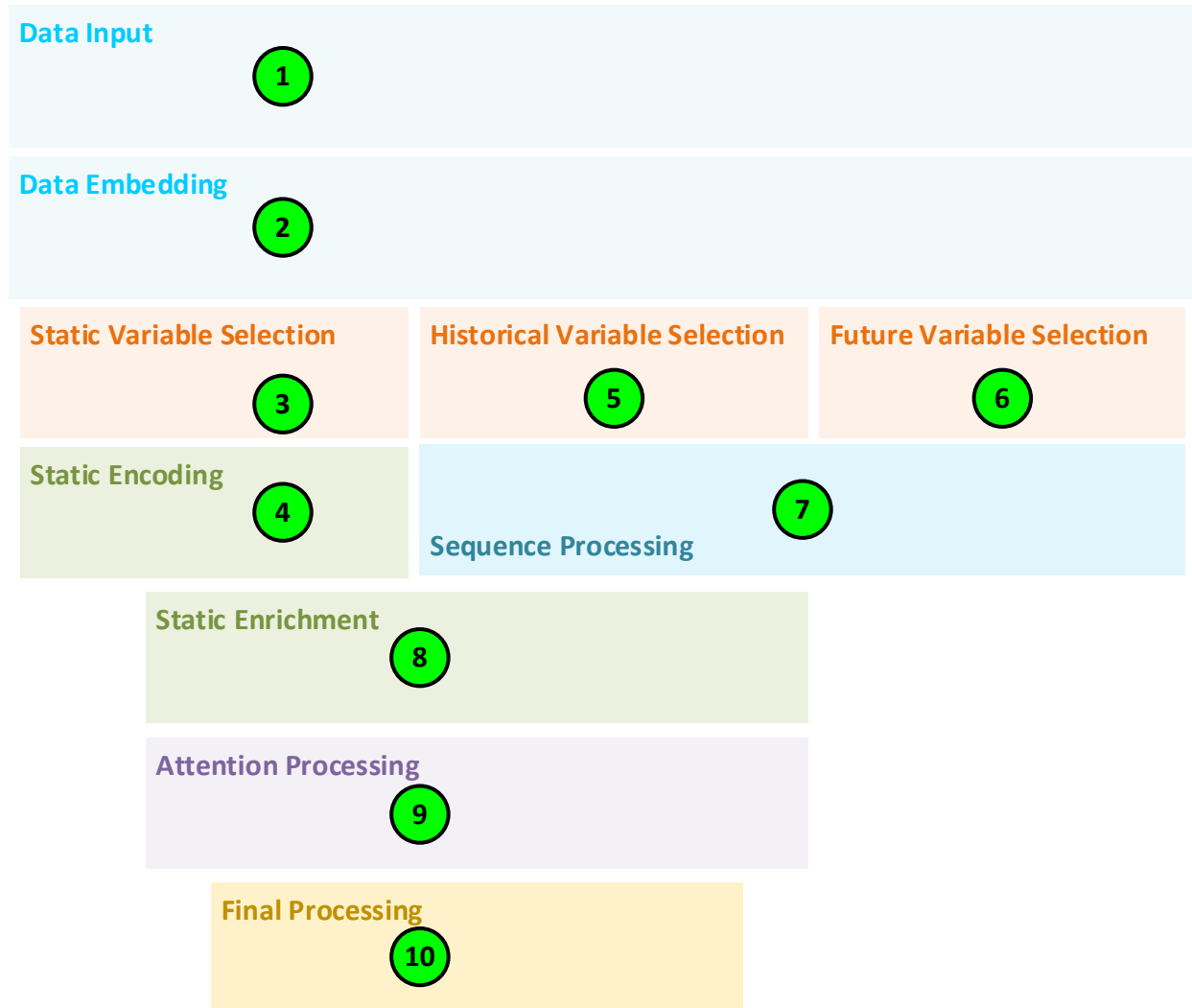


Figure 113 Using Google Translate to Verify Results

You have now created and trained your first Encoder/Decoder Transformer model for language translation!

## TEMPORAL FUSION TRANSFORMER MODEL FOR TIME SERIES PREDICTION

In this next example, we will use a Temporal Fusion Transformer (TFT) model, initially introduced by Lim, et. al. [35], for time series prediction. We will use this model to predict the use of electricity and in a separate sample to predict the flow of traffic.



**Figure 114** TFT Model Flow

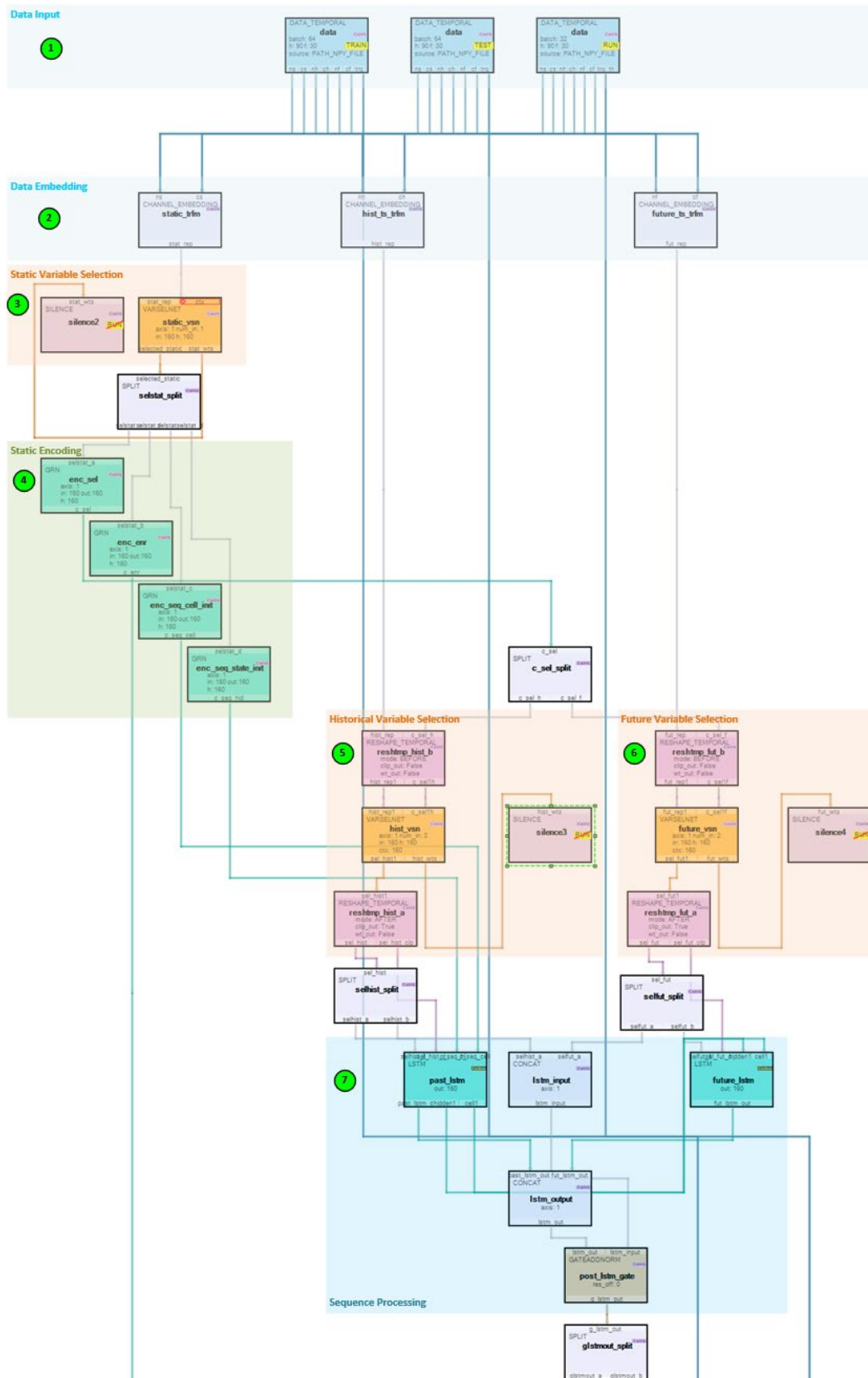
The TFT takes various forms of input data that are static, observed or known. Static data include items that are not tied to temporal measurements (e.g., not tied to time). Observed data include items that are observable only up to the present and are not known in the future. The price of a stock is an example of observed data. Known data include items that are known both in the past and future. The time of day is an example of a known data. Each data type (static, observed and known) may be numeric data or categorical data. Numeric data values are continuous values whereas categorical data items are discrete values that represent a category from a set of categories. For example, a stock price

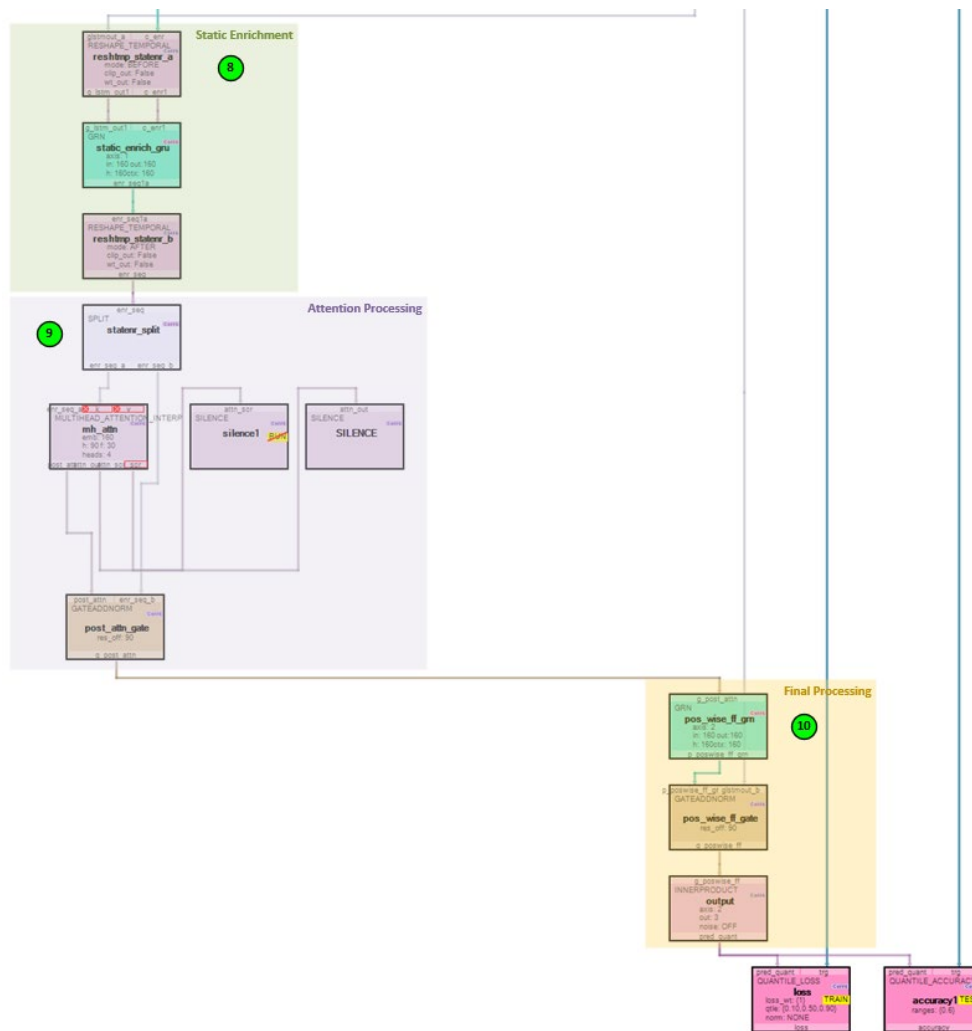
at a given point in time would be numeric data whereas the symbol from a set of symbols would be considered categorical data.

When running the TFT model, the following steps take place.

- 1.) First the data input is collected from a CSV or NPY file, or from a database. Typically, the input data is already pre-processed in that all numerical values are normalized by centering and dividing by the standard deviation. Data inputs associated with differing categorical inputs are normalized within each category. For example, data from one stock symbol is only normalized against data from the same stock symbol.
- 2.) An embedding is created for each data input where numeric data items are fed into linear projections and categorical data are fed into embedding layers.
- 3.) The static embeddings are fed into the static variable selection network used to weigh the static variables that contribute most to the prediction. A side output of the static VSN provides the weighting which allows for post model analysis of the variable contributions.
- 4.) Static encodings are created for the static VSN outputs which are used for static selection in the historical and future VSNs, static enrichment, and as inputs to the first sequence processing LSTM.
- 5.) The historical encodings are fed into the historical variable selection network used to weigh the historical variables that contribute most to the prediction. Like the static VSN, the historical VSN also outputs the variable weights allowing for post model analysis of the variable contributions to the model.
- 6.) The future encodings are fed into the future variable selection network used to weigh the future variables that contribute most to the prediction. These weights can also be analyzed during post model to show the variable contributions to the model.
- 7.) The historical and future VSN outputs are fed into the sequence processing which also have static encodings for the initial LSTM cell and hidden inputs. A two LSTM stack form an encoder/decoder that processes the sequential nature of the data.
- 8.) The sequential processing and static encoding are then fed into the static enrichment phase, which...
- 9.) ... passes down through the attention processing that uses a multi-headed attention layer designed for interpretability. The attention scores output is useful to see where most of the attention is placed within the time series data.
- 10.) The attention output is passed to the final processing that is either output as a set of quantile future predictions, fed into the loss layer during training, or the accuracy layer during testing.

As you can imagine, the final model is quite complex.





**Figure 115 Temporal Fusion Transformer Model**

The following sections describe the steps needed to get started training a TFT to predict electricity use and traffic flow.

---

## DATA PREPARATION

Before training the models, you will need to get and pre-process the data sets. To get the data, you can either download the datasets from the following links:

[https://archive.ics.uci.edu/ml/machine-learning-databases/00321/LD2011\\_2014.txt.zip](https://archive.ics.uci.edu/ml/machine-learning-databases/00321/LD2011_2014.txt.zip) (electricity)

<https://archive.ics.uci.edu/ml/machine-learning-databases/00204/PEMS-SF.zip> (traffic)

Alternatively, run the MyCaffe Test Application and select the '*Test | Download Test Data | TFT (Data Only)*' menu item which will place the model data in the following directories:

C:\ProgramData\MyCaffe\test\_data\tft\data\electricity

C:\ProgramData\MyCaffe\test\_data\tft\data\traffic

---

## DATA PREPROCESSING

Once you have the data downloaded, you can use the SignalPop AI Designer™ data set creators to load, preprocess and save the data as a set of NPY files that the **DataTemporalLayer** can load.

## ELECTRICITY DATA PREPROCESSING

From within the SignalPop AI Designer <sup>TM</sup>, select the *Dataset Creators* tab and double click on the *TFT.Electricity* dataset creator.

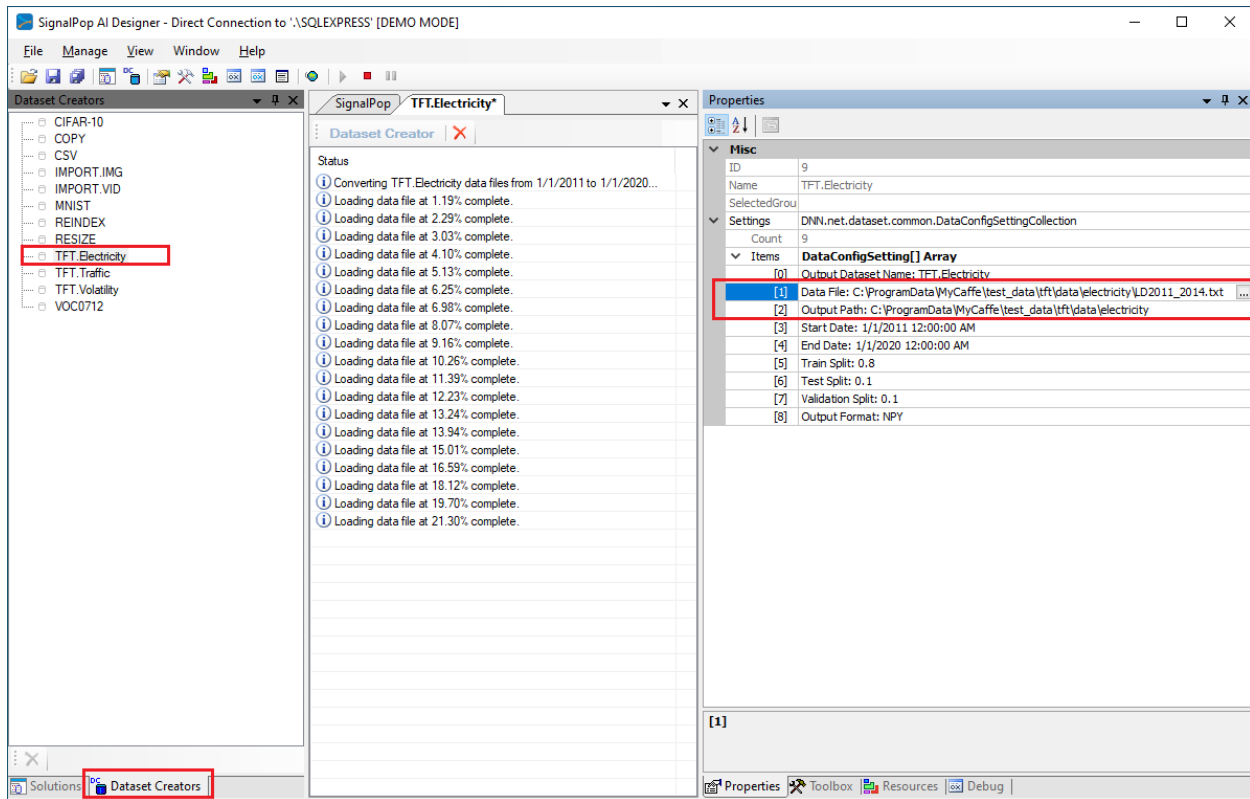


Figure 116 TFT Electricity Dataset Creator

Enter in the following settings in the property window and select the *Run* (▶) button to start creating the pre-processed dataset:

**Data File:** C:\ProgramData\MyCaffe\test\_data\tft\data\electricity\LD2011\_2014.txt

**Output Path:** C:\ProgramData\MyCaffe\test\_data\tft\data\electricity

Upon completion, the pre-processed files are placed in the following output directory.

C:\ProgramData\MyCaffe\test\_data\tft\data\electricity\preprocessed

In addition to normalizing the data, the datasets are split into train, test, and validation sets.

## TRAFFIC DATA PREPROCESSING

From within the SignalPop AI Designer <sup>TM</sup>, select the *Dataset Creators* tab and double click on the *TFT.Traffic* dataset creator.

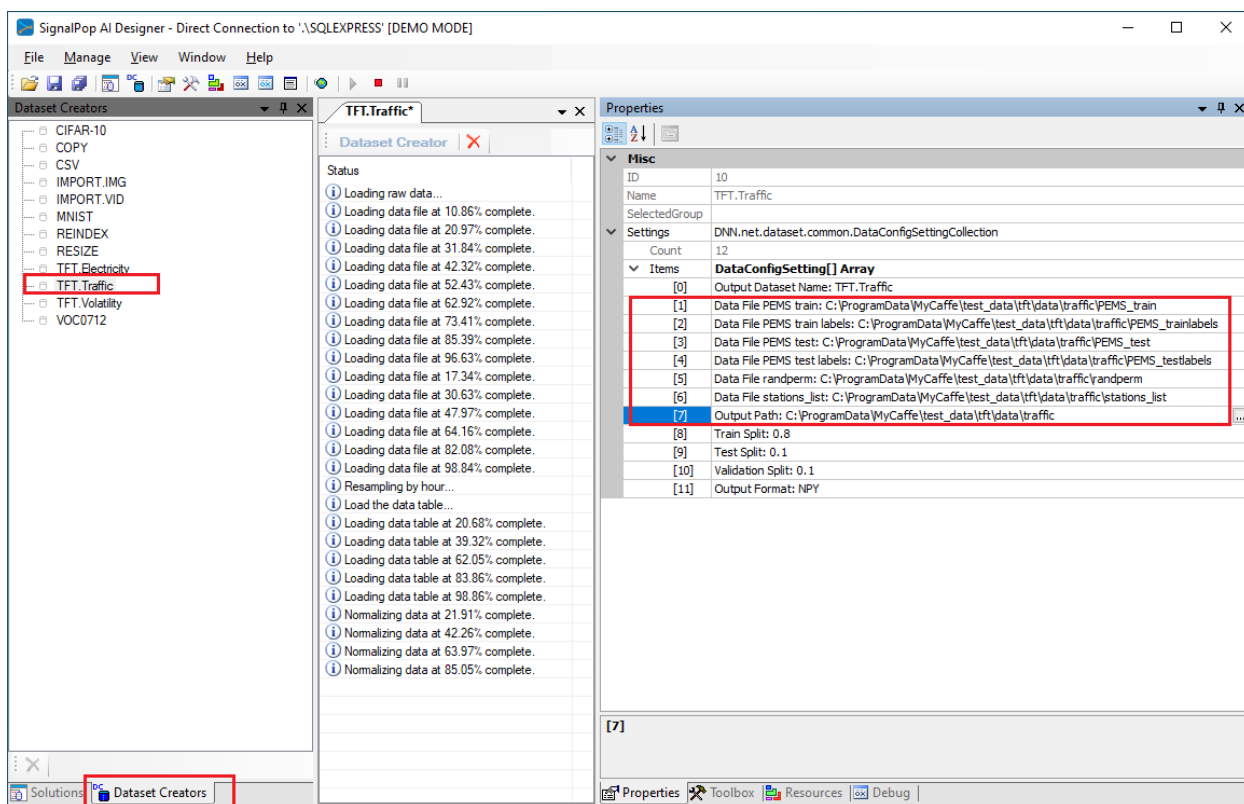


Figure 117 TFT Traffic Dataset Creator

Enter in the following settings in the property window and select the *Run* (▶) button to start creating the pre-processed dataset:

**Data File PEMS train:** C:\ProgramData\MyCaffe\test\_data\tft\data\traffic\PEMS\_train

**Data File PEMS train labels:** C:\ProgramData\MyCaffe\test\_data\tft\data\traffic\PEMS\_trainlabels

**Data File PEMS test:** C:\ProgramData\MyCaffe\test\_data\tft\data\traffic\PEMS\_test

**Data File PEMS test labels:** C:\ProgramData\MyCaffe\test\_data\tft\data\traffic\PEMS\_testlabels

**Data File randperm:** C:\ProgramData\MyCaffe\test\_data\tft\data\traffic\randperm

**Data File stations\_list:** C:\ProgramData\MyCaffe\test\_data\tft\data\traffic\stations\_list

**Output Path:** C:\ProgramData\MyCaffe\test\_data\tft\data\traffic

Upon completion, the pre-processed files are placed in the following output directory.

C:\ProgramData\MyCaffe\test\_data\tft\data\traffic\preprocessed

In addition to normalizing the data, the datasets are split into train, test, and validation sets.



## TFT FOR ELECTRICITY USE PREDICTION

To set up this model and start training, first create the model. To create a new project, select the *Add Project* (+) button at the bottom of the *Solutions* window.

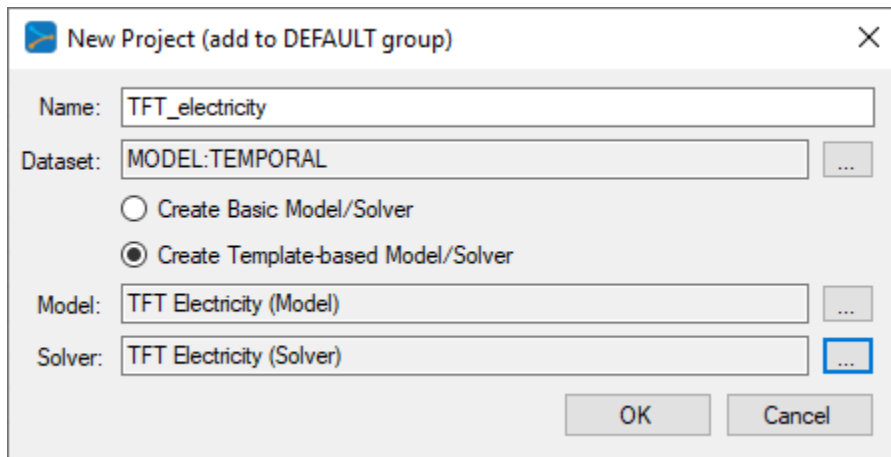
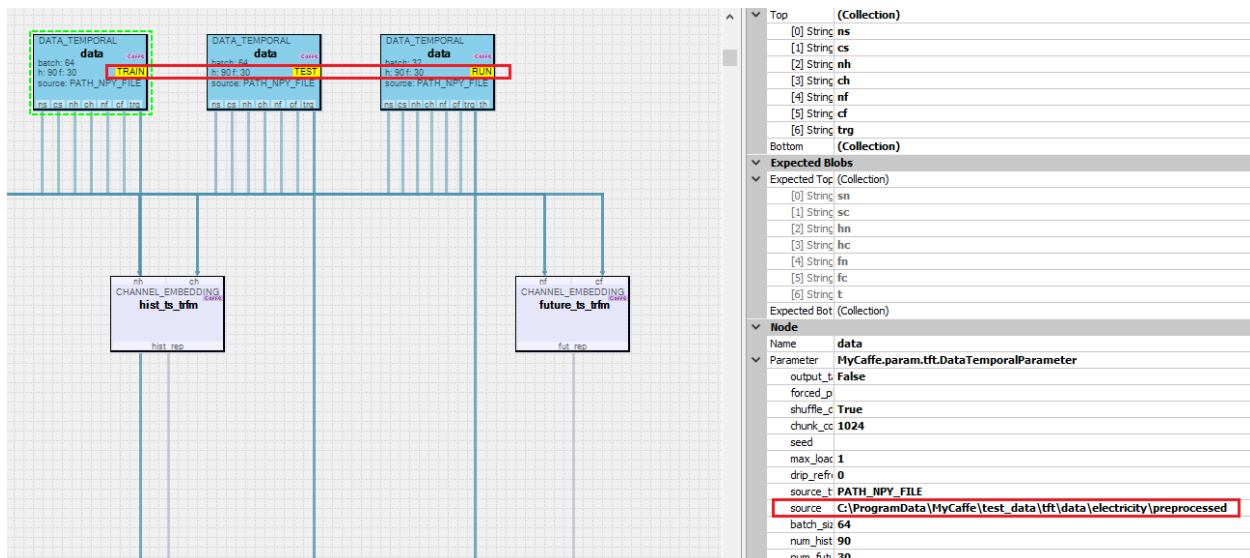


Figure 118 Creating the TFT Electricity Project

From the *New Project* dialog, select the *MODEL (temporal)* and TFT Electricity model and solver and press *OK* to create the new project.

Open the project, by right clicking on the *TFT\_electricity* project and selecting *Open* from the menu.



Double clicking on the *tft\_net* displays the model architecture. By selecting the *DataTemporal* layer you can then see its properties in the *Properties* window to the right. Note, each *DataTemporal* layer points to the same *source folder*. The data file with the matching phase is loaded from this directory.

Next, double click on the project name *TFT\_electricity* to open the *TFT\_electricity* project and select the *Run Training* (🚀) button.

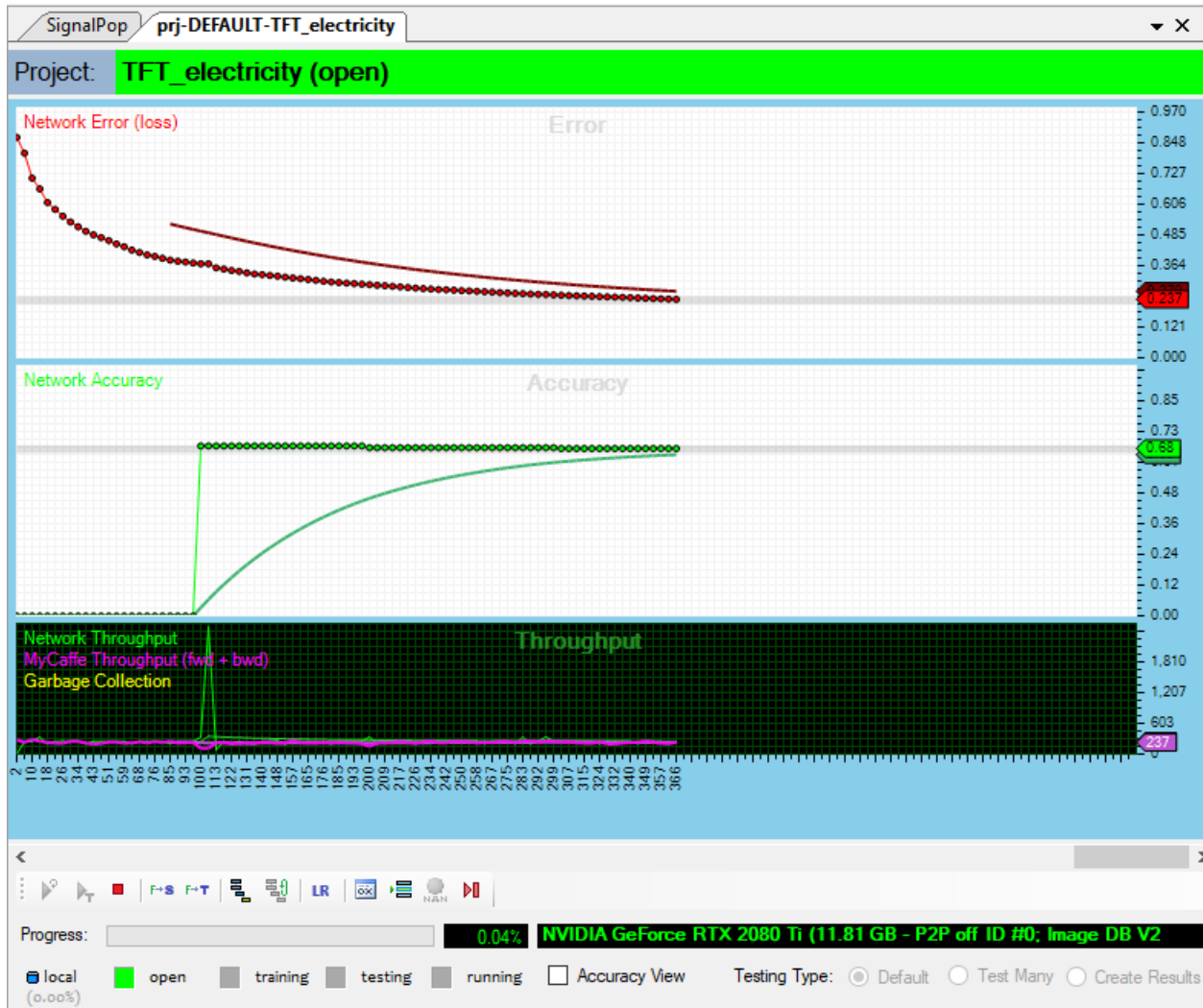


Figure 119 Training the *TFT\_electricity* Model.

The model uses the ADAMW solver with the following settings:

**Learning Rate:** 0.001


**Weight Decay:** 0.0

**Lr Policy:** SIGMOID

**AdamW Decay:** 0.0

**Momentum:** 0.9

**Momentum2:** 0.999

After training for around 1000 iterations, try running the *Test* (  ) with the *Test Many* radio button selected. This will first prompt you for the schema file which is in the preprocessed directory.

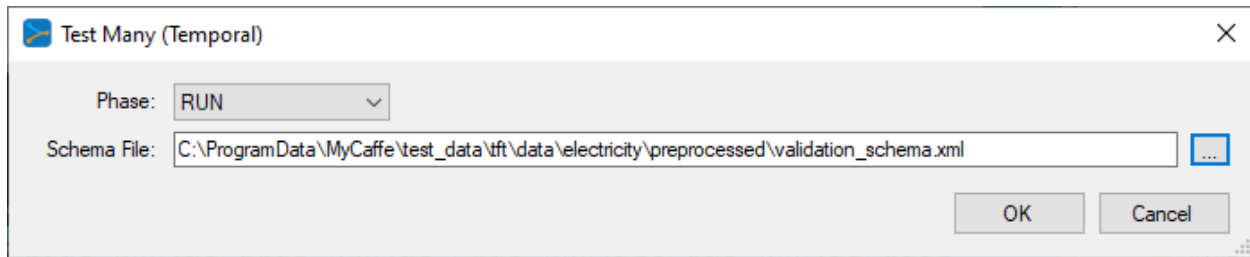


Figure 120 Run Test Many for Analysis

Running the *Test Many* runs a detailed analysis on the data that first shows several predictions against the actuals used during training.

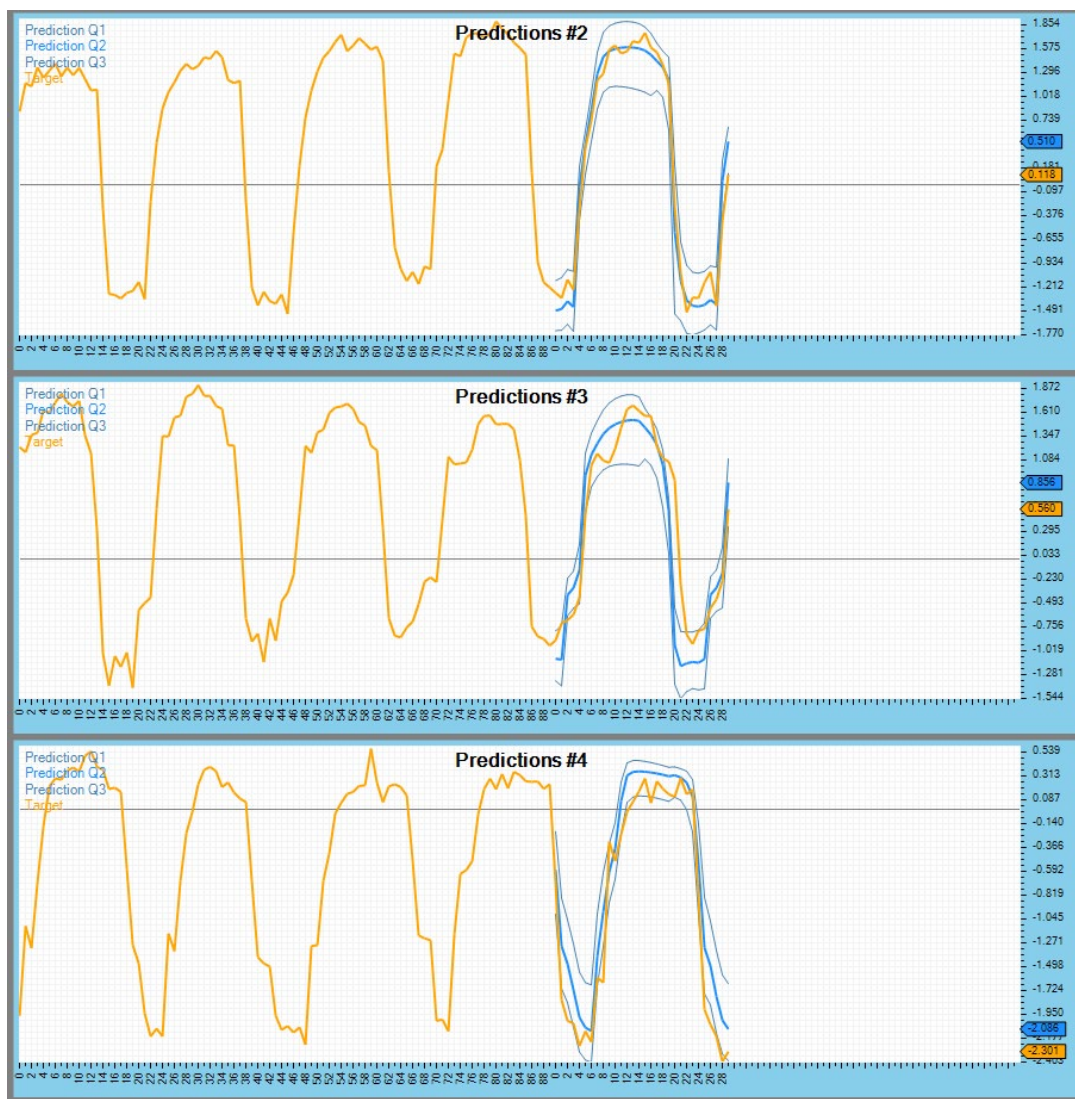
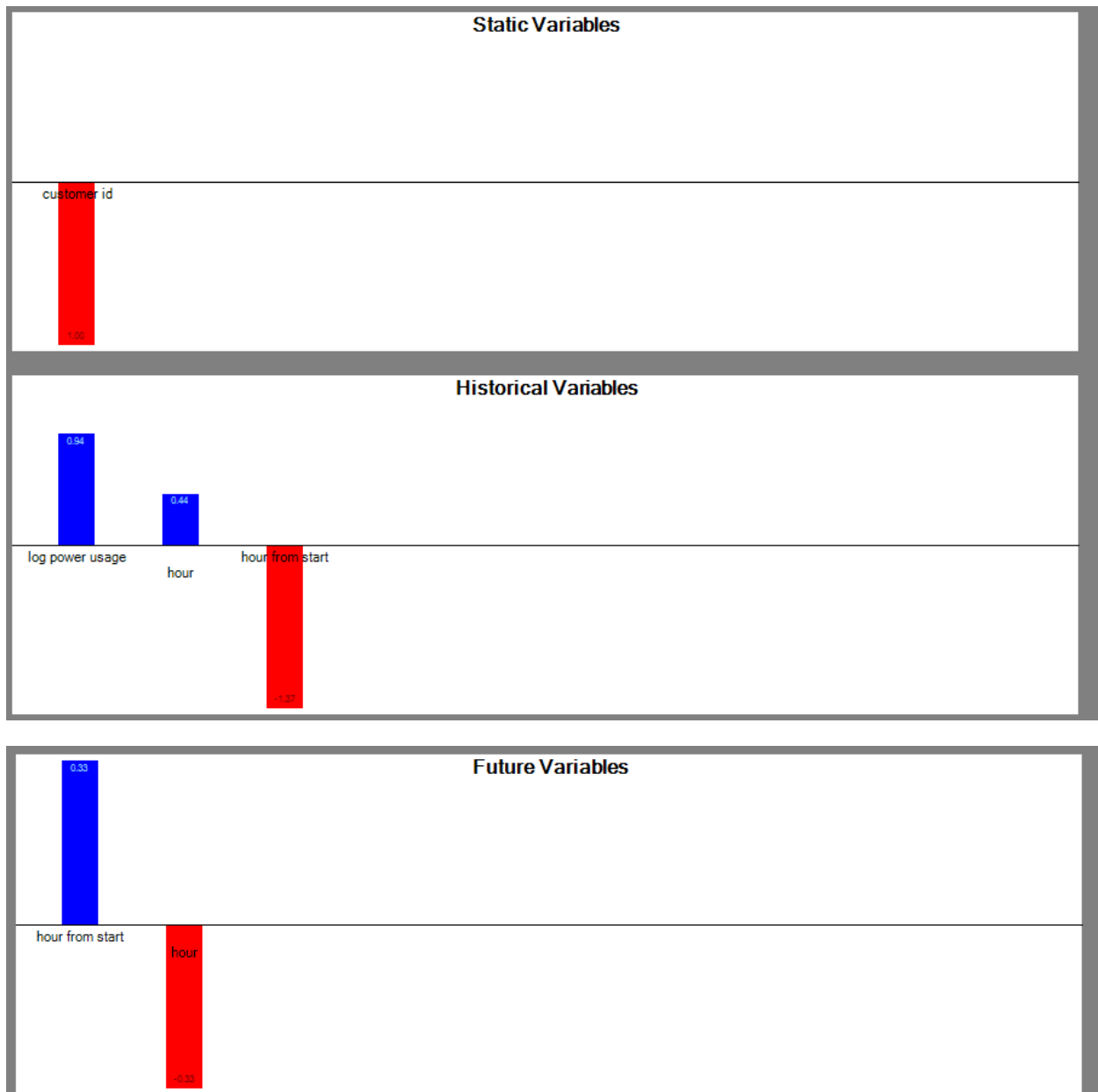


Figure 121 Predictions vs Actuals

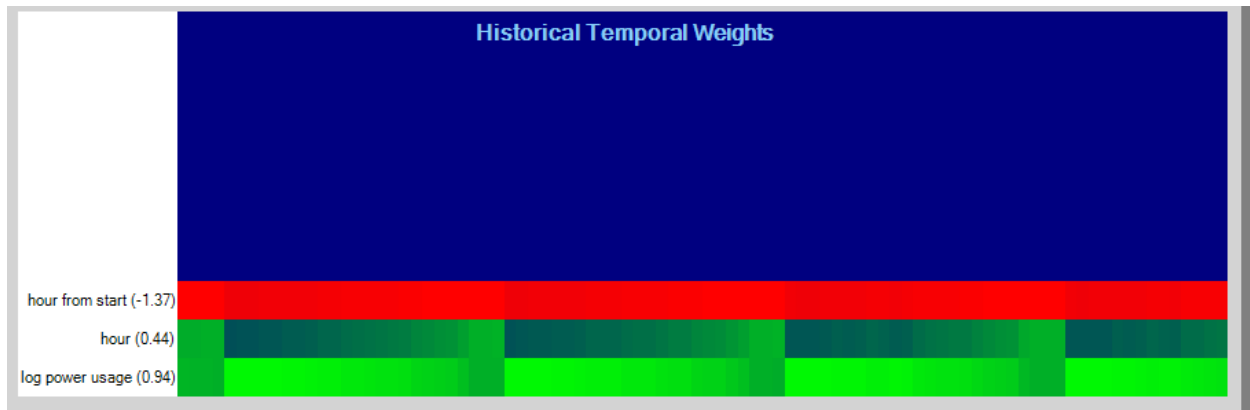
Next the static, historical and future variables are analyzed by showing their relative contributions to the model.



**Figure 122 Variable Contributions**

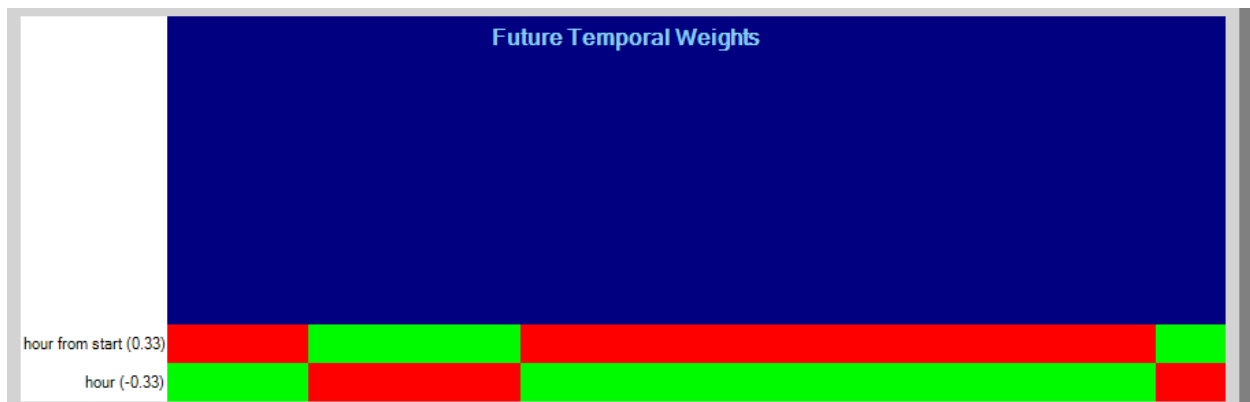
As you can see, *log-power usage* has the highest impact in the historical variables followed by the *hour*. And the *hour from start* has the highest impact in the future variables.

Next, the temporal weights for each variable are analyzed from a random sample to show where in time the variable contributions were made. The following shows the weights for the historical variables.



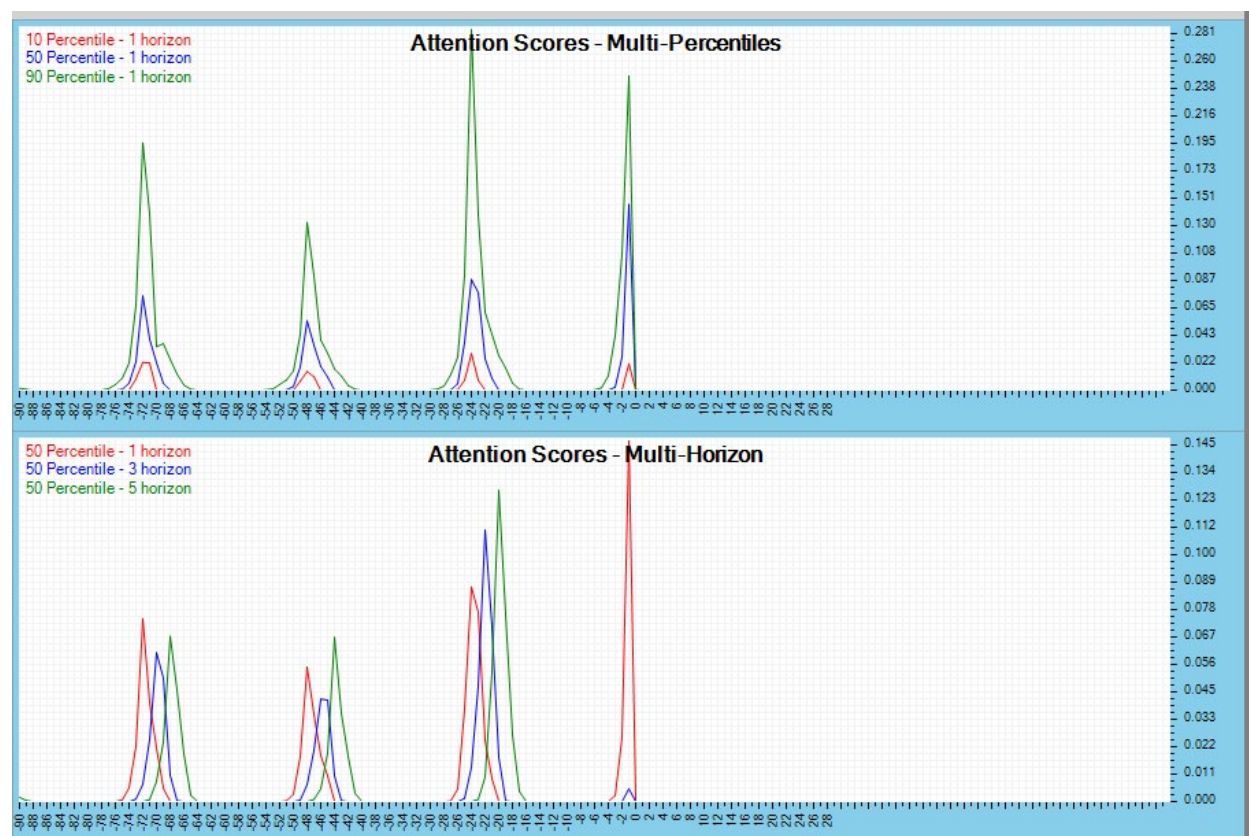
**Figure 123 Historical Temporal Weights**

And the following shows the weights for the future variables.



**Figure 124 Future Temporal Weights**

And finally, the attention scores are displayed to show where in the temporal data the focus was spent during the learning process.



**Figure 125 Temporal Attention Scores**

The temporal attention scores are shown in a multi-percentile view (10, 50 and 90 percentiles) followed by a multi-horizon view (1, 3 and 5 horizons at 50 percentile).

As shown above, a 24-hour cycle was observed by the attention scores.

## TFT FOR TRAFFIC FLOW PREDICTION

To set up this model and start training, first create the model. To create a new project, select the *Add Project* (+) button at the bottom of the *Solutions* window.

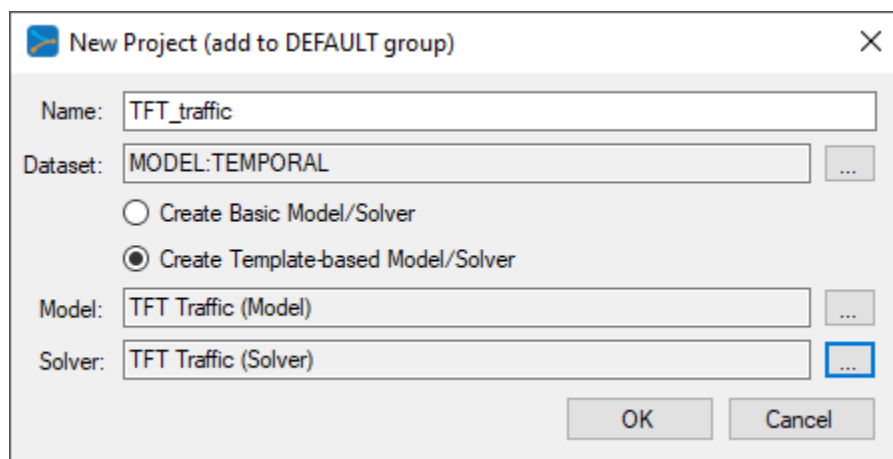
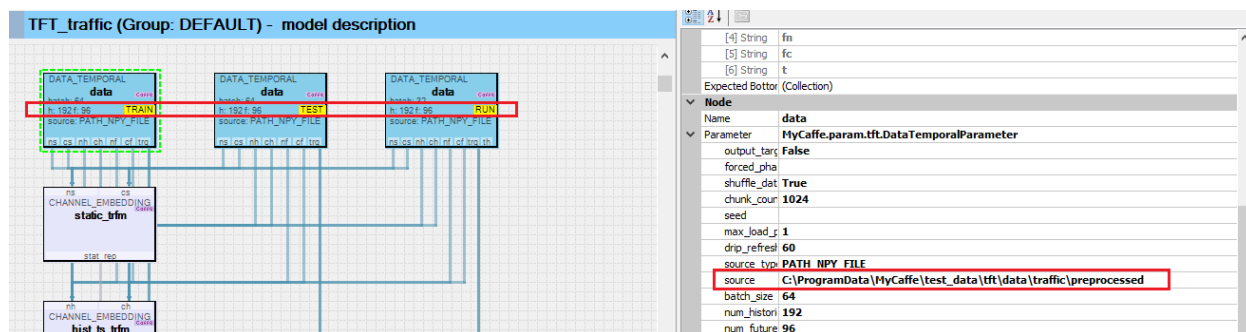


Figure 126 Creating the TFT Traffic Project

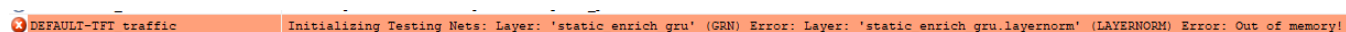
From the *New Project* dialog, select the *MODEL (temporal)* and TFT Traffic model and solver and press *OK* to create the new project.

Open the project, by right clicking on the *TFT\_traffic* project and selecting *Open* from the menu.



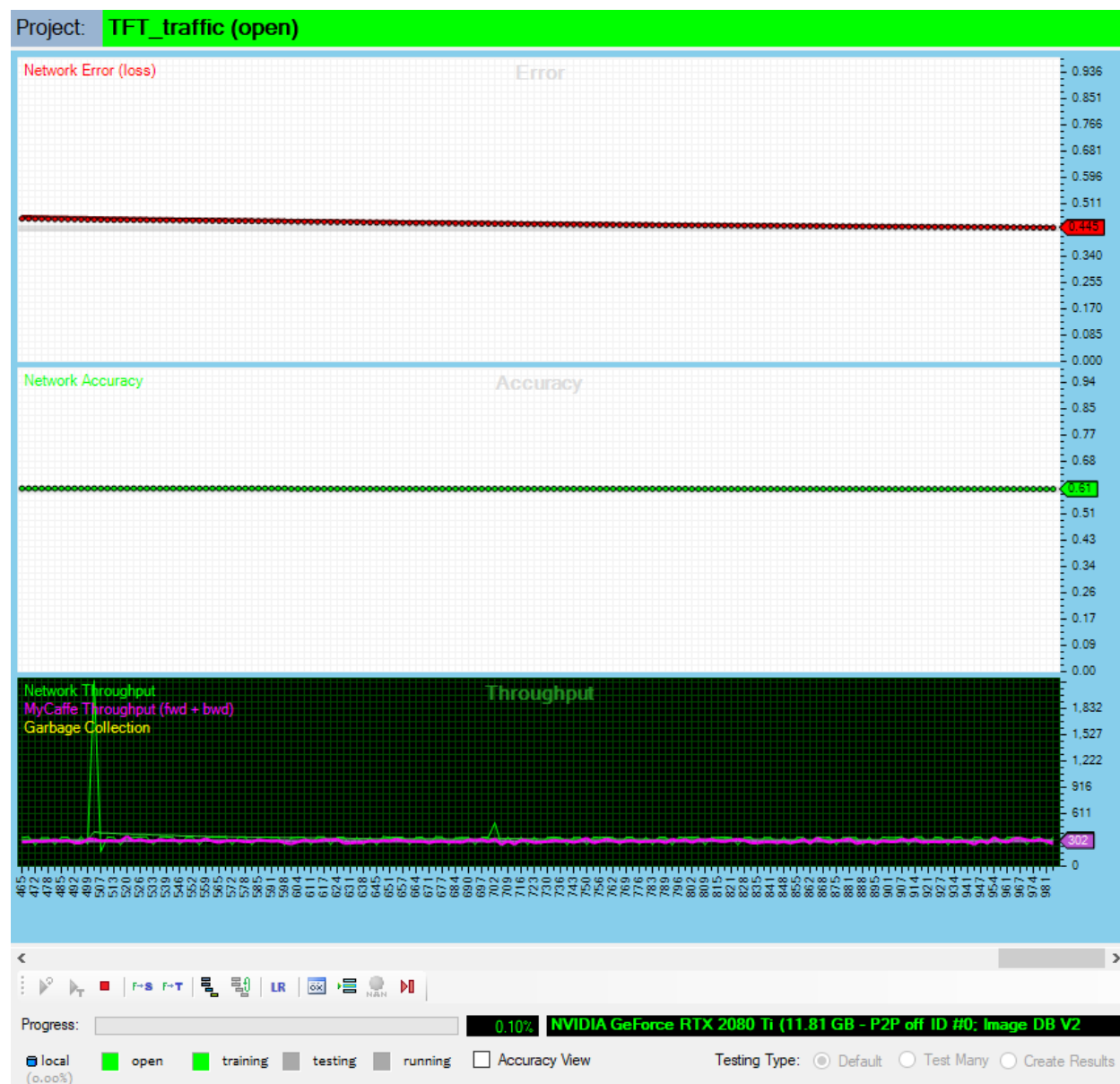
Double clicking on the *tft\_net* displays the model architecture. By selecting the **DataTemporal** layer you can then see its properties in the *Properties* window to the right. Note, each **DataTemporal** layer points to the same *source folder*. The data file with the matching phase is loaded from this directory.

Note, when training, if you do not have enough memory to load the model, you will receive an error.



When this occurs, select each *DataTemporal* layer and reduce the batch size from 64 to 32, or 16 depending on the amount of memory in your GPU.


Next, double click on the project name *TFT\_traffic* to open the *TFT\_traffic* project and select the *Run Training* (🚀) button.



**Figure 127 Training the TFT\_traffic Model.**

The model uses the same ADAMW solver and settings used by the TFT electricity model discussed above.



After training for around 1000 iterations, try running the *Test* (  ) with the *Test Many* radio button selected. This will first prompt you for the schema file which is in the preprocessed directory.

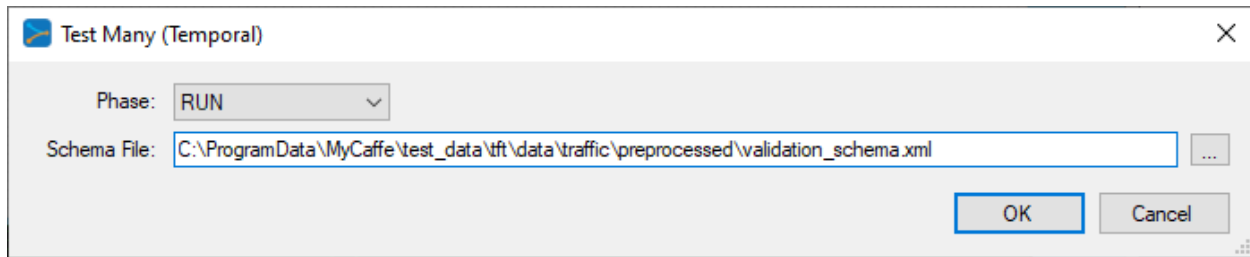


Figure 128 Run Test Many for Analysis

Running the *Test Many* runs a detailed analysis on the data that first shows several predictions against the actuals used during training.

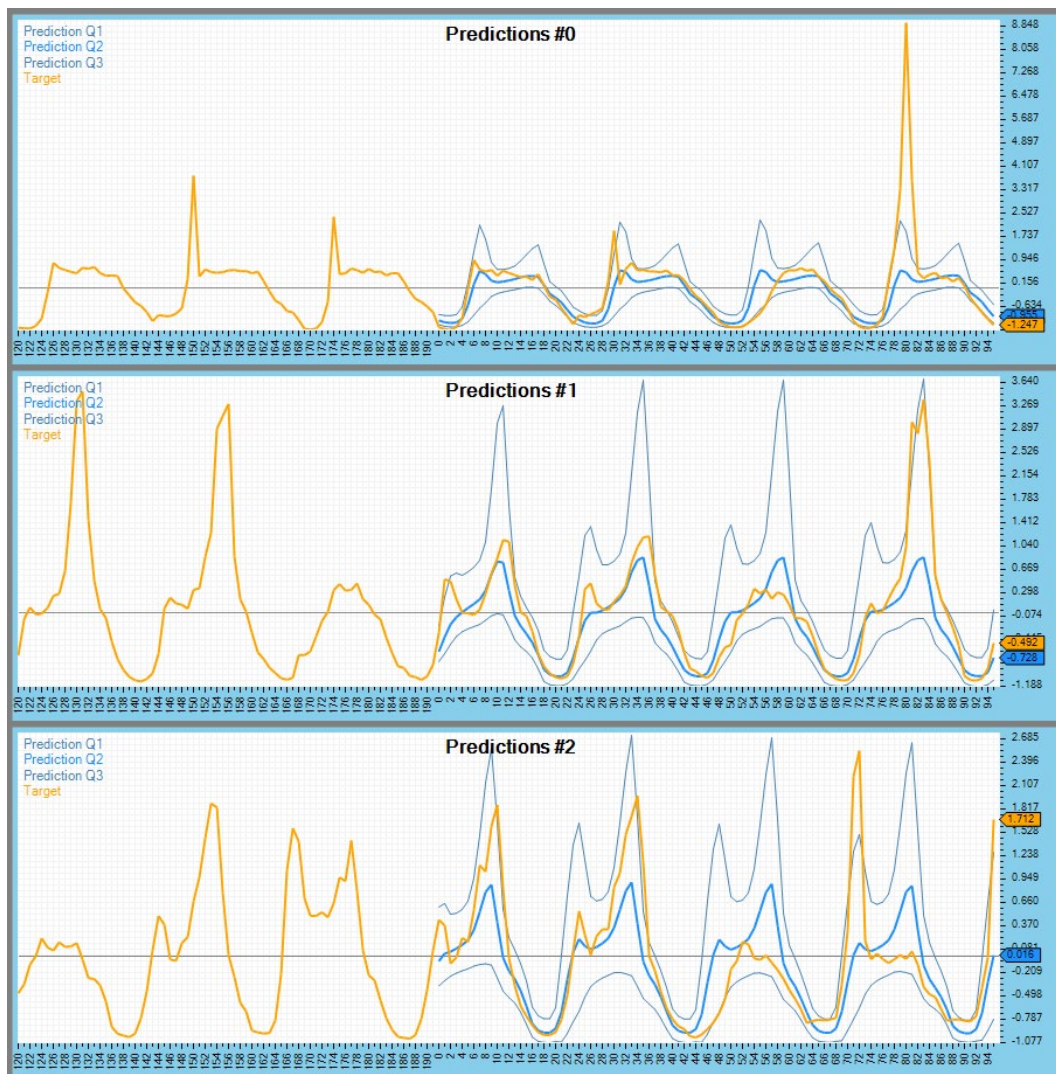
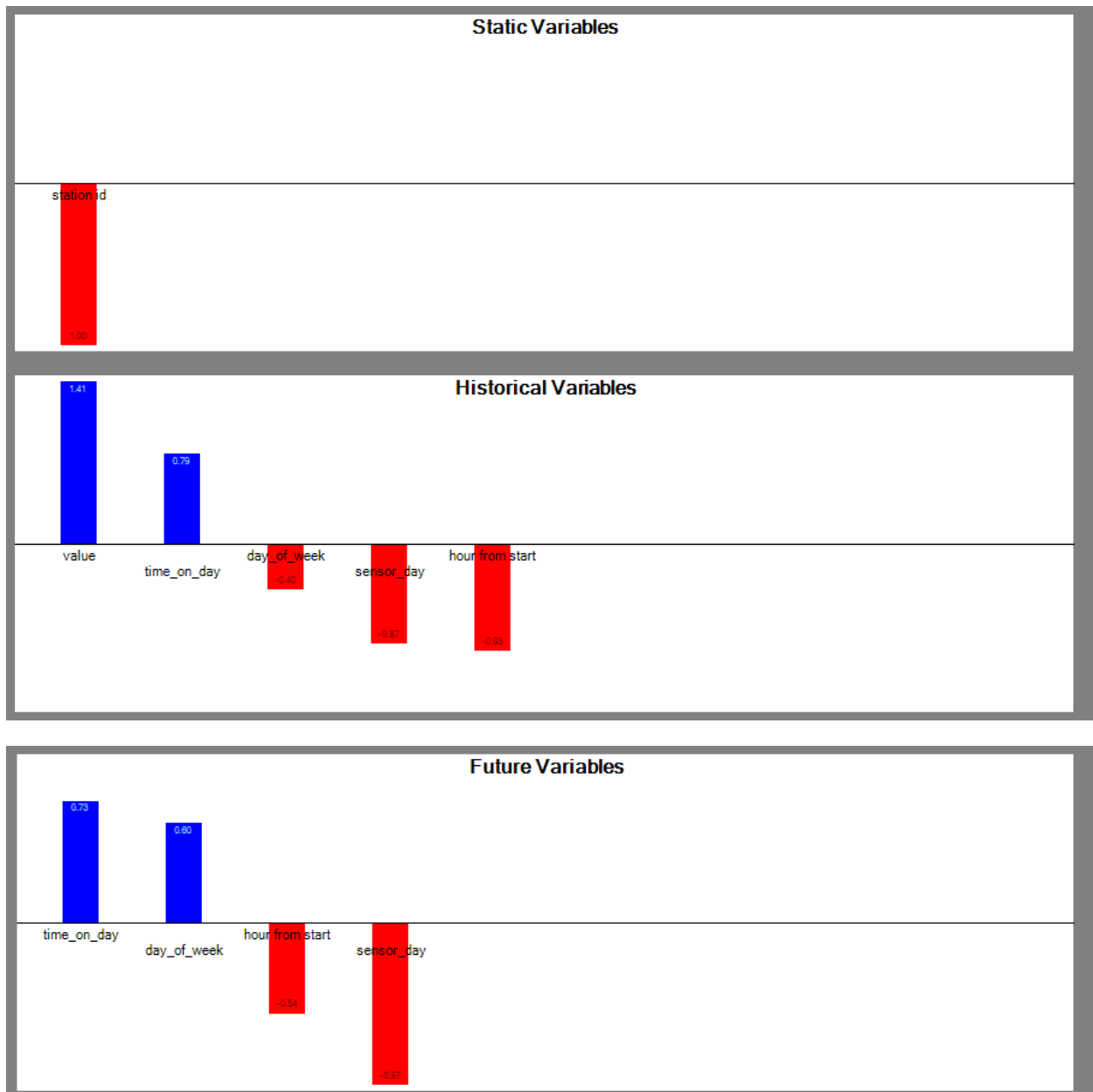


Figure 129 Predictions vs Actuals

Next the static, historical and future variables are analyzed by showing their relative contributions to the model.



**Figure 130 Variable Contributions**

As you can see, *value (traffic flow)* has the highest impact in the historical variables followed by the *time on day*. And the *time on day* has the highest impact in the future variables followed by the *day of week*.

Next, the temporal weights for each variable are analyzed from a random sample to show where in time the variable contributions were made. The following shows the weights for the historical variables.

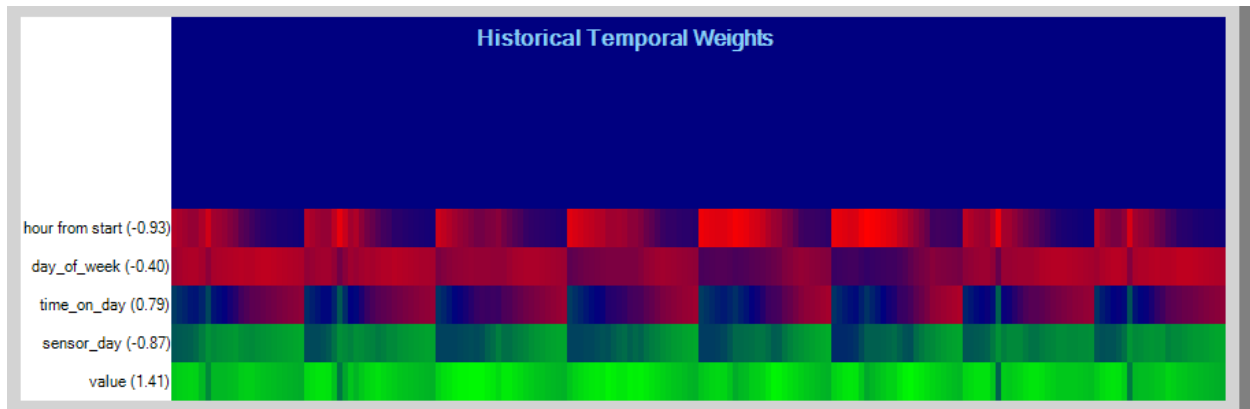


Figure 131 Historical Temporal Weights

And the following shows the weights for the future variables.

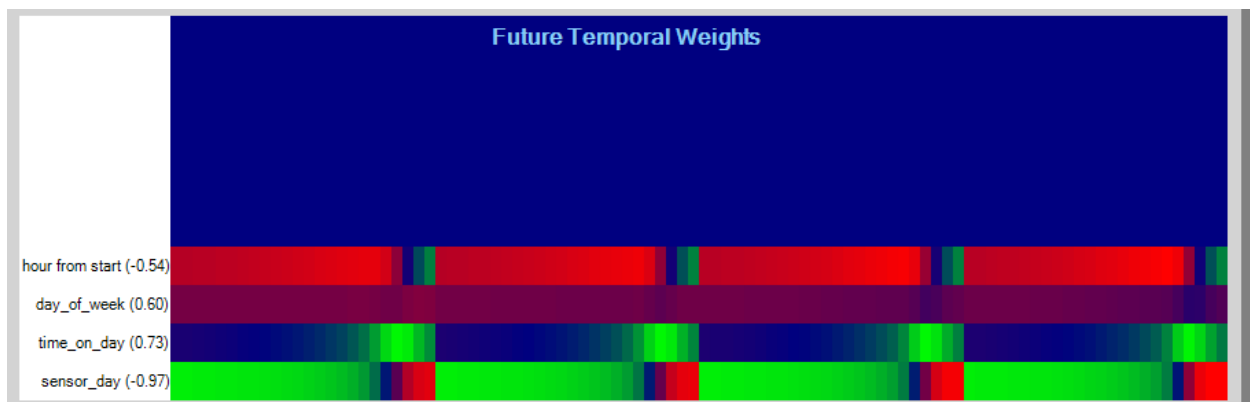
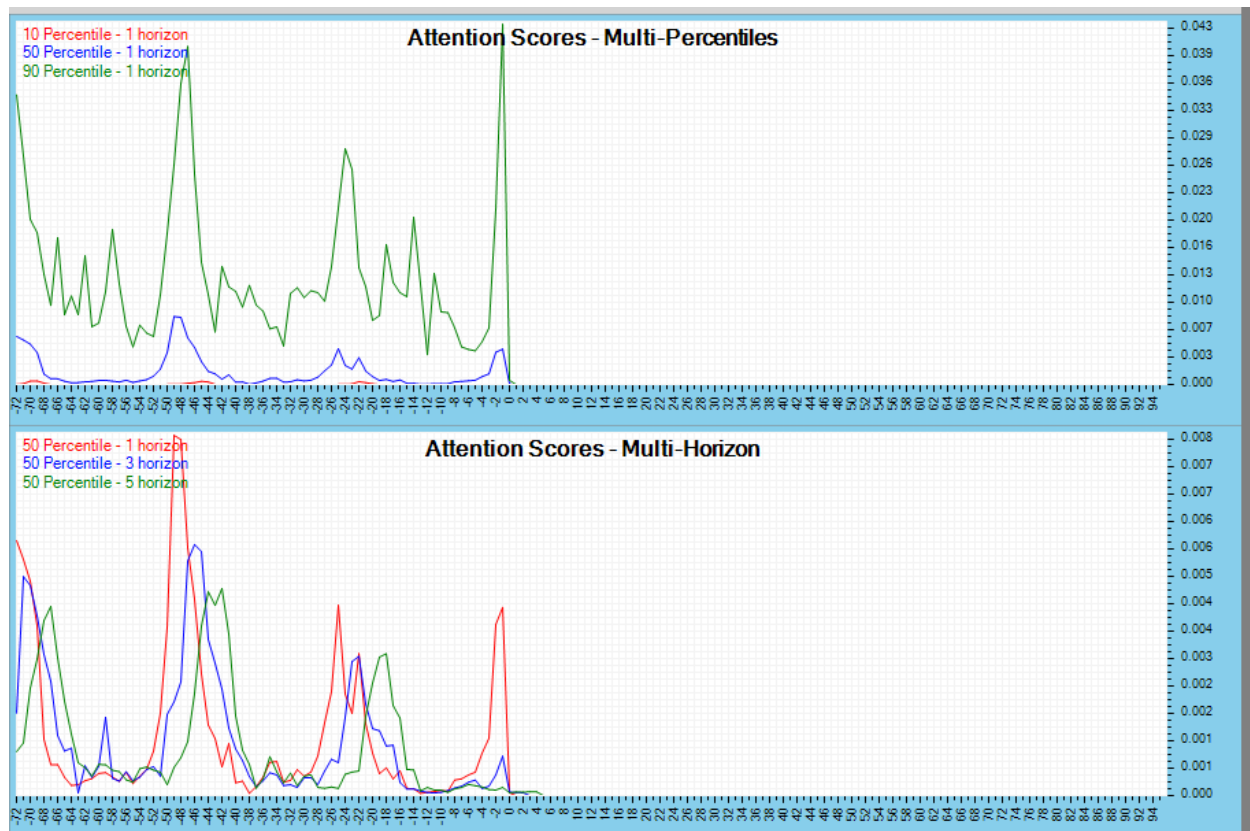


Figure 132 Future Temporal Weights

And finally, the attention scores are displayed to show where in the temporal data the focus was spent during the learning process.



**Figure 133 Temporal Attention Scores**

The temporal attention scores are shown in a multi-percentile view (10, 50 and 90 percentiles) followed by a multi-horizon view (1, 3 and 5 horizons at 50 percentile).

Although not as periodic as the electricity data, the traffic attention scores also show a 24-hour cycle which would account for rush-hour.

## SUMMARY

As you use the SignalPop AI Designer, make sure to check the following resources for helpful hints and updates related to the product.

**SignalPop Site** - see <http://www.signalpop.com> for new MyCaffe based products that leverage the models you build with the SignalPop AI Designer.

**SignalPop Blog** – see <https://www.signalpop.com/blog/> for updated news on the product, new models that we support and general product information.

**MyCaffe on GitHub** – see <http://github.com/mycaffe> for updates on the MyCaffe AI Platform including its full source code and automated tests. The SignalPop AI Designer is designed to work directly with the MyCaffe AI Platform.

**MyCaffe on NuGet** – see <http://www.nuget.org/packages?q=MyCaffe> for updates of the MyCaffe binary components that are easily integrated into your own C# applications for Windows.

**SignalPop AI Designer Main Help** – see <https://www.signalpop.com/help/> or select the 'Help | Main Help' menu in the SignalPop AI Designer for detailed help on the MyCaffe AI Platform. Within this help you can find descriptions of each Activation type, Filler type and much more.

## APPENDIX A – SIGNALPOP UNIVERSAL MINER

Originally created to mine Ethereum, the SignalPop Universal Miner™ provides an easy way to monitor your GPU temperatures and fan speeds.

**Free Download**<sup>16</sup> <https://signalpop.blob.core.windows.net/wpeupdate/wpe.net.app.setup.exe>

### HARDWARE MONITORING

Monitor current GPU temperatures, usage, fan speeds and clock settings with the *Hardware Window*.

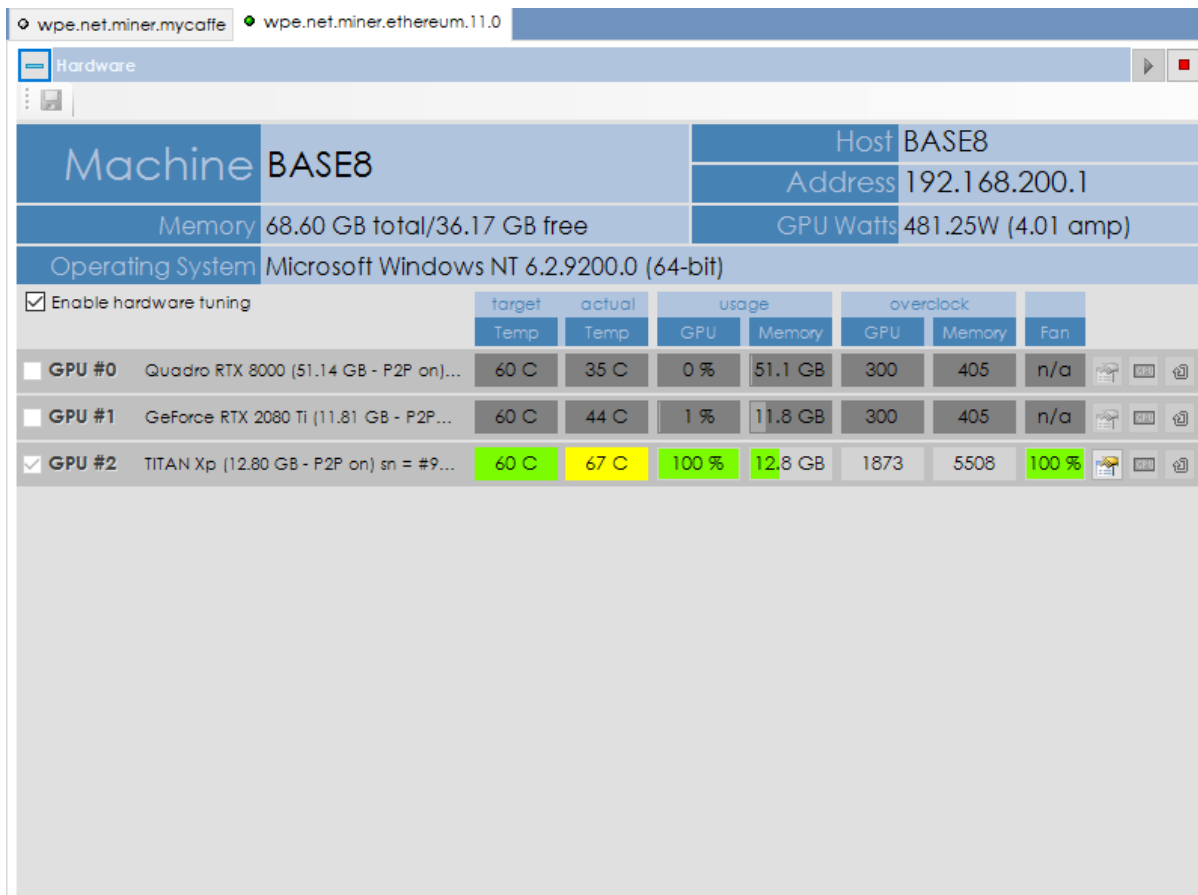


Figure 134 Hardware Window

<sup>16</sup> A small portion of daily mining time is run using our mining address to help pay for maintaining the SignalPop Universal Miner – this is otherwise known as the 'dev fee'.

Monitor hardware changes over time (such as fan speeds and temperatures) with the *History Window*.

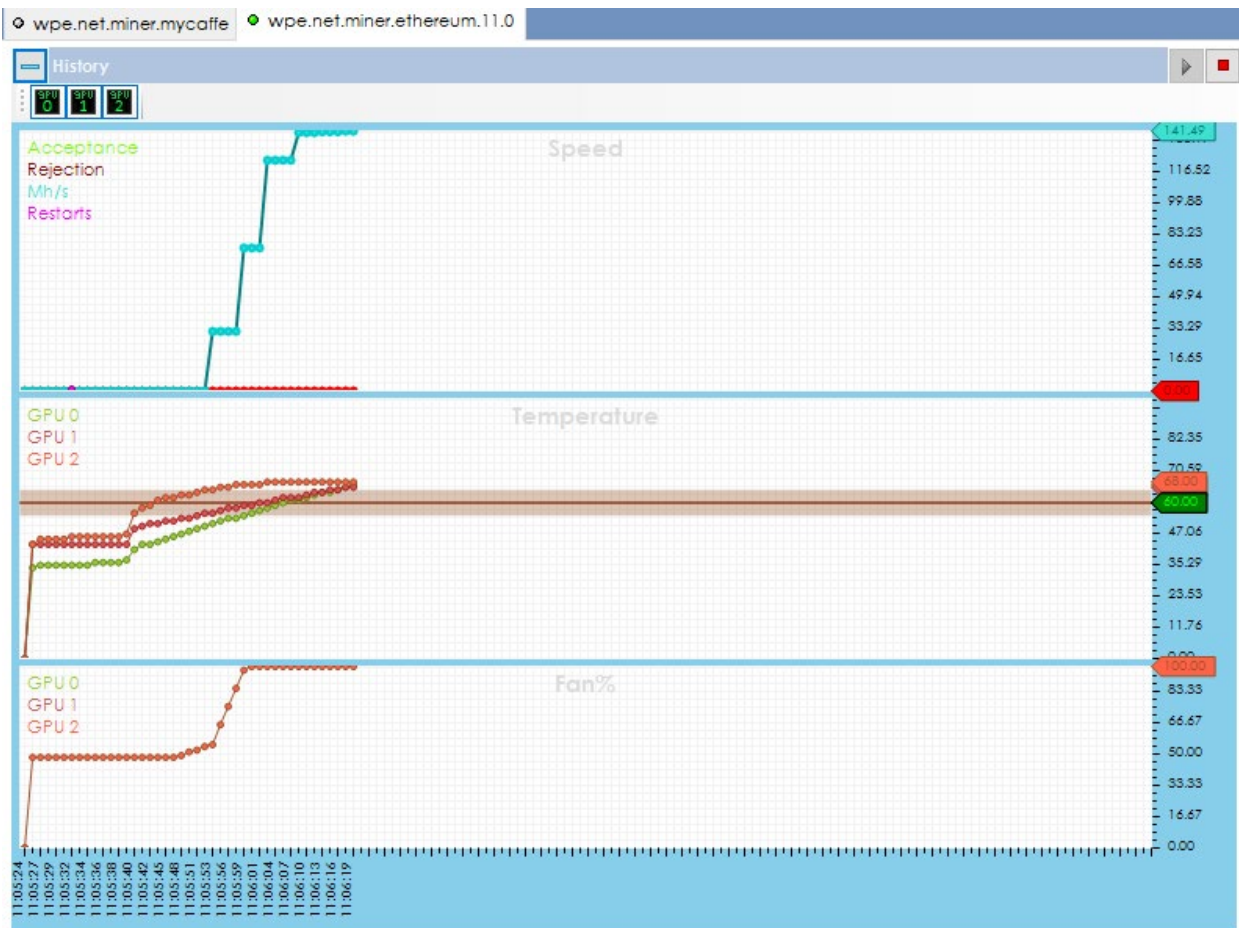


Figure 135 History Window

## APPENDIX B – DATASET CREATOR INTERFACE

Each dataset creator is a plug-in module to the SignalPop AI Designer. At some point you may want to create your own dataset creators. To create a dataset creator, you will need to create a C# component that implements the **IXDatasetCreator** interface as described below.

### IXDATASETCREATOR INTERFACE

The **IXDatasetCreator** interface is the main interface used by the SignalPop AI Designer to create datasets and has the following methods:

**Name**; returns the name of the creator.

**QueryConfiguration**; returns the configuration settings of the creator that show up on the property window.

**Create**; creates the dataset and gives feedback to the application via the **IXDatasetCreatorProgress** interface.

---

#### IXDATASETCREATOR::NAME

**Syntax**      `string Name { get; }`

**Description** Returns the name of the dataset creator.

---

#### IXDATASETCREATOR::QUERYCONFIGURATION

**Syntax**      `void QueryConfiguration(DatasetConfiguration config)`

**Description** Returns the configuration settings for the dataset creator in a DatasetConfiguration object.

---

#### IXDATASETCREATOR::CREATE

**Syntax**      `void Create(DatasetConfiguration config,  
                                                 IXDatasetCreatorProgress progress)`

**Description** Creates the dataset using the configuration settings in the *config* parameter all the while calling the *progress* object to display the progress updates in the SignalPop AI Designer.



## DATASETCONFIGURATION OBJECT

The **DatasetConfiguration** object contains the set of settings used when creating the dataset. This object has the following methods:

**DatasetConfiguration constructor**; used to create a new collection.

**IsReadOnly**; specifies whether the configuration is read-only or not.

**ID**; specifies the ID of the dataset creator.

**Name**; specifies the name of the dataset creator.

**SelectedGroup**; optionally, specifies the selected group of the dataset creator.

**Settings**; specifies the collection of settings used when creating the dataset.

**Sort**; sorts the settings within the Settings collection.

**Clone**; duplicates this configuration object and returns it as a new one.

**SaveToFile**; saves the configuration settings to a file.

**LoadFromFile**; loads a set of configurations from file.

---

### DATASETCONFIGURATION::ISREADONLY

**Syntax**      `bool IsReadOnly { get; }`

**Description** Returns whether the configuration is read-only or not.

---

### DATASETCONFIGURATION::ID

**Syntax**      `int ID { get; set; }`

**Description** Get/set the ID of the dataset creator.

---

### DATASETCONFIGURATION::NAME

**Syntax**      `string Name { get; }`

**Description** Returns the name of the dataset creator.

DATASETCONFIGURATION::SELECTEDGROUP	
<b>Syntax</b>	<code>string SelectedGroup { get; }</code>
<b>Description</b>	Returns the name of the dataset creator.

**Description** Returns the name of the dataset creator.

DATASETCONFIGURATION::SETTINGS	
<b>Syntax</b>	<code>DataConfigurationSettingCollection Settings { get; }</code>
<b>Description</b>	Returns the collection of settings used to create the dataset.

**Description** Returns the collection of settings used to create the dataset.

DATASETCONFIGURATION::SORT	
<b>Syntax</b>	<code>void Sort()</code>
<b>Description</b>	Sorts the Settings collection.

<b>Description</b>	Sorts the Settings collection.
--------------------	--------------------------------

DATASETCONFIGURATION::CLONE	
<b>Syntax</b>	<code>DatasetConfiguration Clone()</code>
<b>Description</b>	Returns a copy of the configuration settings.

**Description** Returns a copy of the configuration settings.

DATASETCONFIGURATION::SAVETOFILE	
<b>Syntax</b>	<code>void SaveToFile(DataConfiguration[] settings, string strFile)</code>
<b>Description</b>	Save the settings to the file specified.

<b>Description</b>	Save the settings to the file specified.
--------------------	------------------------------------------

DATASETCONFIGURATION::LOADFROMFILE	
<b>Syntax</b>	<code>DataConfiguration[] LoadFromFile(string strFile)</code>
<b>Description</b>	Load the settings from the file specified.

<b>Description</b>	Load the settings from the file specified.
--------------------	--------------------------------------------

## DATACONFIGSETTINGCOLLECTION OBJECT

The **DataConfigSettingCollection** provides a standard List interface to a collection of **DataConfigSetting** objects.

## DATACONFIGSETTING OBJECT

Each **DataConfigSetting** object contains the data for one setting of the settings used to create the dataset. The **DataConfigSetting** has the following methods.

**VerifyInterface**; interface used to verify the setting, when set.

**Name**; returns the name of the setting.

**Extra**; returns extra string data of the setting (sometimes this is used to specify a file extension).

**Value**; get/set the value of the setting.

**Type**; returns the type of the setting value.

**Clone**; returns a copy of the setting.

**ToString**; returns a string representation of the setting.

**ToString**; returns a string used for saving.

**Parse**; parses a saved string into a setting.

---

### DATACONFIGSETTING::VERIFYINTERFACE

**Syntax** `IXDatasetCreatorSettings VerifyInterface { get; }`

**Description** Returns the interface used to verify the setting (or `null` if not set).

---

### DATACONFIGSETTING::NAME

**Syntax** `string Name { get; }`

**Description** Returns the name of the setting.

---

#### DATACONFIGSETTING::EXTRA

**Syntax**      `string Extra { get; }`

**Description** Returns extra information about the setting such as an expected file extension.

---

#### DATACONFIGSETTING::VALUE

**Syntax**      `object Value { get; set; }`

**Description** Get/set the actual setting value.

---

#### DATACONFIGSETTING::TYPE

**Syntax**      `DataConfigSetting::TYPE Type { get; }`

**Description** Returns the type of the setting where TYPE is one of the following values:

TEXT – specifies the setting value is a string type.

FILENAME – specifies the setting value is a string type that represents a file name.

DIRECTORY – specifies the setting value is a string type that represents a directory.

LIST – specifies the setting value is a list of setting values.

DATETIME – specifies the setting value is a `DateTime` object.

INTEGER – specifies the setting value is an `int` value.

REAL – specifies the setting value is a `double` value.

CUSTOM – specifies the setting value is a custom value.

HELP – specifies the setting value is help information in string form.

---

#### DATACONFIGSETTING::CLONE

**Syntax**      `DataConfigSetting Clone()`

**Description** Returns a copy of the setting.

---

#### DATACONFIGSETTING::TOSAVESTRING

**Syntax**      `string ToSaveString()`

**Description** Returns a string representing the settings of the configuration used for saving.

## DATACONFIGSETTING::PARSE

[illegible]

**Description** Returns a string representing the settings of the configuration used for saving.

**Parameters** `string` `str`; specifies the string containing the settings to parse.

`IXDatasetCreatorSettings` `iVerify`; specifies optional the interface to use for setting verification, default = null.

## IXDATASETCREATORSETTINGS INTERFACE

Optionally, you may also want to implement the **IXDatasetCreatorSettings** interface which then gives your dataset creator the chance to validate settings before accepting them. The **IXDatasetCreatorSettings** interface has the following methods:

**VerifyConfiguration;** allows you to verify the settings before accepting them.

**GetCustomSetting;** allows you to query the user for a custom setting type.

## IXDATASETCREATORSETTINGS::VERIFYCONFIGURATION

**Syntax**     `void VerifyConfiguration(DataConfigSetting[] settings)`

**Description** Verify the *settings* and if invalid, throw an exception.

**Parameters** `DataConfigSetting[]` settings; specifies the settings to verify.

---

IXDATASETCREATORSETTINGS::GETCUSTOMSETTING

**Syntax**

```
void GetCustomSetting(string strName,  
                     string strCustomSettingType,  
                     DataConfigSetting[] settings)
```

**Description** Retrieve the custom setting with the name *strName* and place it within the *settings* array.

**Parameters**

- `string` strName; specifies the name of the setting to retrieve.
- `String` strCustomSettingType; optionally, specifies a custom setting type, or null when not used.
- `DataConfigSetting[]` settings; specifies the settings to search.

## IXDATASETCREATORPROGRESS INTERFACE

The **IXDatasetCreatorProgress** interface is implemented by the SignalPop AI Designer and passed as a parameter to the **IXDatasetCreator::Create** method thus allowing you to give feedback of the dataset creation process. The **IXDatasetCreatorProgress** interface has the following methods:

**OnProgress**; called to pass along general progress information including the percentage complete.

**OnCompleted**; called upon completing the dataset creation process.

---

### IXDATASETCREATORPROGRESS::ONPROGRESS

**Syntax**      `void OnProgress(CreateProgressArgs args)`

**Description**    Call this method (implemented by the SignalPop AI Designer) when you want to update the designer of the status and progress taking place when creating your dataset.

---

### IXDATASETCREATORPROGRESS::ONCOMPLETED

**Syntax**      `void OnCompleted(CreateProgressArgs args)`

**Description**    Call this method (implemented by the SignalPop AI Designer) when you are done creating your dataset.

## CREATEPROGRESSARGS OBJECT

The **CreateProgressArgs** object describes the progress status taking place when creating your dataset and has the following methods.

**Aborted**; specifies that the creation process was aborted.

**PercentComplete**; specifies the percentage of completion of the creation process.

**PercentCompleteAsText**; a text representation of **PercentComplete**;

**Message**; specifies a message describing the current status of the creation process.

**Error**; specifies an error that occurred during the creation process.

**Abort**; specifies whether to abort the creation process.

---

#### CREATEPROGRESSARGS::ABORTED

**Syntax** `bool Aborted{ get; }`

**Description** Returns whether the process was aborted.

---

#### CREATEPROGRESSARGS::PERCENTCOMPLETE

**Syntax** `double Aborted{ get; }`

**Description** Returns the percentage of completion as a number between 0.0 and 1.0.

---

#### CREATEPROGRESSARGS::PERCENTCOMPLETEASTEXT

**Syntax** `string PercentCompleteAsText{ get; }`

**Description** Returns the percentage of completion as string such as 20.0%.

---

#### CREATEPROGRESSARGS::MESSAGE

**Syntax** `string Message{ get; }`

**Description** Returns the status of the creation process.

---

#### CREATEPROGRESSARGS::ERROR

**Syntax** `Exception Error{ get; }`

**Description** Returns an error that occurred during the creation process.

---

#### CREATEPROGRESSARGS::ABORT

**Syntax** `bool Abort{ get; set; }`

**Description** Specifies whether the SignalPop AI Designer should abort the creation process.

## EXAMPLE SOURCE CODE

To interact with the database, you will want to use the **DatasetFactoryEx** object within the **DNN.net.data** namespace, which is derived from the **DatasetFactory** object within the **MyCaffe.imagedb** namespace.

These two objects allow you to easily create new Datasets, Data Sources, Labels and Raw Images.

For more information and example source code that shows how to create your own dataset creator, see us on GitHub at: <https://github.com/MyCaffe/AiDesigner>.



## REFERENCES

- [1] D. W. Brown, "MyCaffe: A Complete C# Re-Write of Caffe with Reinforcement Learning," Cornell University, 4 October 2018. [Online]. Available: <https://arxiv.org/abs/1810.02272>.
- [2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," 20 June 2014. [Online]. Available: <https://arxiv.org/abs/1408.5093>.
- [3] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand and V. Lempitsky, "Domain-Adversarial Training of Neural Networks," *Journal of Machine Learning Research* 17, pp. 1-35, 2016.
- [4] M. Andrecut, "Parallel GPU Implementation of Iterative PCA Algorithms," *arXiv*, vol. arXiv:0811.1081, p. 45, 7 November 2008.
- [5] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, no. 9, pp. 1-27, 2008.
- [6] L. van der Maaten and G. Hinton, "User's Guide for t-SNE Software," 2016. [Online]. Available: [https://lvdmaaten.github.io/tsne/User\\_guide.pdf](https://lvdmaaten.github.io/tsne/User_guide.pdf).
- [7] OpenAI, "CartPole-V0," OpenAI, [Online]. Available: <https://gym.openai.com/envs/CartPole-v0/>.
- [8] OpenAI, "cartpole.py on GitHub," GitHub, 27 April 2016. [Online]. Available: [https://github.com/openai/gym/blob/master/gym/envs/classic\\_control/cartpole.py](https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py).
- [9] R. e. a. Sutton, "http://incompleteideas.net/sutton/book/code/pole.c," 1983. [Online]. Available: <https://perma.cc/C9ZM-652R>.
- [10] A. G. Barto, R. S. Sutton and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE*, Vols. SMC-13, no. 5, pp. 834-846, September 1983.
- [11] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," *arXiv*, vol. arXiv:1311.2901, 28 November 2013.
- [12] Wikipedia, "DeepDream".

- [13] L. A. Gatys, A. S. Ecker and M. Bethge, "A Neural Algorithm of Artistic Style," 26 Aug 2015. [Online]. Available: <https://arxiv.org/abs/1508.06576>.
- [14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 4 Sep 2014. [Online]. Available: <https://arxiv.org/abs/1409.1556>.
- [15] Q. V. Le, "A Tutorial on Deep Learning Part 2: Autoencoders, Convolution Neural Networks and Recurrent Neural Networks," 20 October 2015. [Online]. Available: <http://robotics.stanford.edu/~quocle/tutorial2.pdf>.
- [16] V. Turchenko, E. Chalmers and A. Luczak, "A Deep Convolutional Auto-Encoder with Pooling - Unpooling Layers in Caffe," *arXiv*, p. 21 pages, Jan 18, 2018.
- [17] Berkeley Artificial Intelligence Research (BAIR), "Siamese Network Training with Caffe," Berkeley Artificial Intelligence Research, [Online]. Available: <https://caffe.berkeleyvision.org/gathered/examples/siamese.html>.
- [18] G. Koch, R. Zemel and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition," *ICML 2015 Deep Learning Workshop*, p. 8, 2015.
- [19] K. L. Wiggers, A. S. Britto, L. Heutte, A. L. Koerich and L. S. Oliveira, "Image Retrieval and Pattern Spotting using Siamese Neural Network," *arXiv*, vol. 1906.09513, p. 8, 22 June 2019.
- [20] Y.-A. Chung and W.-H. Weng, "Learning Deep Representations of Medical Images using Siamese CNNs with Application to Content-Based Image Retrieval," *arXiv*, vol. 1711.08490, p. 8, 22 November 2017.
- [21] D. J. Rao, S. Mittal and S. Ritika, "Siamese Neural Networks for One-shot detection of Railway Track Switches," *arXiv*, vol. 1712.08036, p. 6, 21 December 2017.
- [22] A. Karpathy, "Deep Reinforcement Learning: Pong from Pixels," Andrej Karpathy blog, 31 May 2016. [Online]. Available: <http://karpathy.github.io/2016/05/31/rl/>.
- [23] A. Karpathy, "karpathy/pg-pong.py," GitHub, 2016. [Online]. Available: <https://gist.github.com/karpathy/a4166c7fe253700972fcbc77e4ea32c5>.
- [24] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," Stanford University, [Online]. Available: <http://cs231n.github.io/neural-networks-2/#losses>.

- [25] P. S. Castro, S. Moitra, C. Gelada, S. Kumar and M. G. Bellemare, "Dopamine: A Research Framework for Deep Reinforcement Learning," *arXiv*, vol. arXiv:1812.06110v1, p. 11, 14 December 2018.
- [26] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized Experience Replay," *arXiv*, no. arXiv:1511.05952, p. 21, 18 November 2015.
- [27] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wiestra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, p. 13, 25 February 2015.
- [28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wiestra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv*, no. arXiv:1312.5602v1, p. 9, 19 December 2013.
- [29] M. Fortunato, M. A. Gheshlaghi, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell and S. Legg, "Noisy Networks for Exploration," *arXiv*, no. arxiv:1706.10295v2, p. 21, 30 June 2017.
- [30] A. Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks," Andrej Karpathy blog, 21 May 2015. [Online]. Available: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>.
- [31] adepierre, "adepierre/caffe-char-rnn Github," Github.com, 25 January 2017. [Online]. Available: <https://github.com/adepierre/caffe-char-rnn>.
- [32] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko and T. Darrell, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description," Cornell University Library, arXiv.org, 17 November 2014. [Online]. Available: <https://arxiv.org/abs/1411.4389>.
- [33] junhyukoh, "Junhyukoh/caffe-lstm Github," Github.com, 30 August 2016. [Online]. Available: <https://github.com/junhyukoh/caffe-lstm>.
- [34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," *arXiv*, 12 June 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762>.

- [35] B. Lim, S. O. Arik, N. Loeff and T. Pfister, "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting," *arXiv*, vol. 19, p. Dec, 2019.