Xilinx AlexNet Test Drive Tutorial

Xilinx AlexNet Test Drive Tutorial

Table of Contents

Overview

The Xilinx AlexNet Test Drive is an early access look at the performance of image classification through an AlexNet model running on the Xilinx KCU1500 board, featuring the Kintex UltraScale™ KU115 FPGA. This demo will allow users to get an early look at Xilinx's machine learning software stack and offers a few customizations to the AlexNet demo.

The demo accelerates classification of images, taken from ImageNet, through an AlexNet neural network model. It's implemented through the open source framework, Caffe, and accelerated with the Xilinx Deep Neural Network (xfDNN) library to be fully optimized, delivering the highest compute efficiency for 8 bit inference.

Please note, as this is an early access demo, not all features are enabled and some flows may change when Xilinx publicly releases additional machine learning software later this year.

Getting Started

You can experience 8-bit precision using Xilinx technology. **If you don't have one or both of these accounts, click the links below to register. (please note: approval may take up to 48 hours)**

Step 1: These are the profiles you need to get started:

✓ **Nimbix Cloud Account**

✓ **Xilinx on Nimbix Cloud Account**

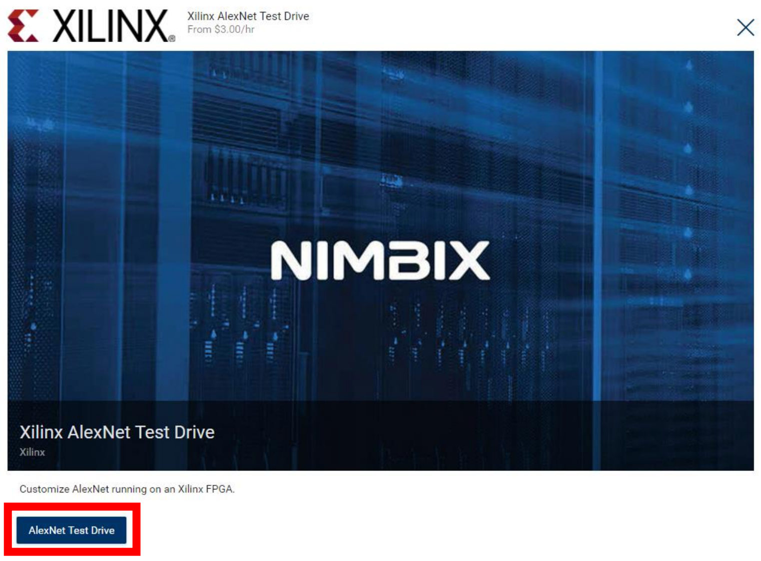
Please be sure to write **"AlexNet Test Drive"** in the comments/questions section, for approval.

For more information on how to use Nimbix, read the Xilinx Nimbix Getting Started Guide.

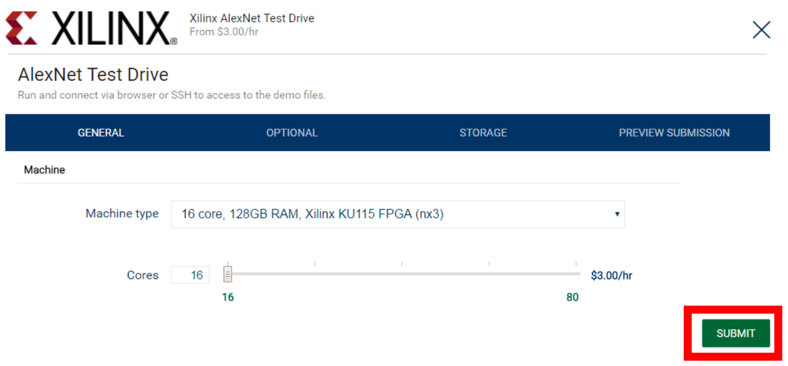
To learn how to transfer your data from and to Nimbix here is a tutorial.

Step 2: Launch the Xilinx AlexNet Test Drive
https://mc.jarvice.com/?page=compute&name=Xilinx%20AlexNet%20Test%20Drive

Click on "AlexNet Test Drive".


XILINX
Xilinx AlexNet Test Drive
From $3.00/hr
NIMBIX
Xilinx AlexNet Test Drive
Xilinx
Customize AlexNet running on an Xilinx FPGA.
AlexNet Test Drive

This will take you to the options screen. You don't need to make any changes, so click "submit".
Note: Standard Nimbix rates apply.


Xilinx AlexNet Test Drive
From $3.00/hr

AlexNet Test Drive

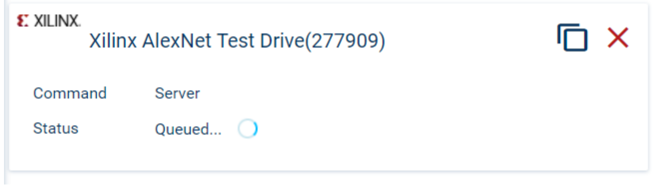
Run and connect via browser or SSH to access to the demo files.

Machine

Machine type: 16 core, 128GB RAM, Xilinx KU115 FPGA (nx3)

Cores: 16 — $3.00/hr

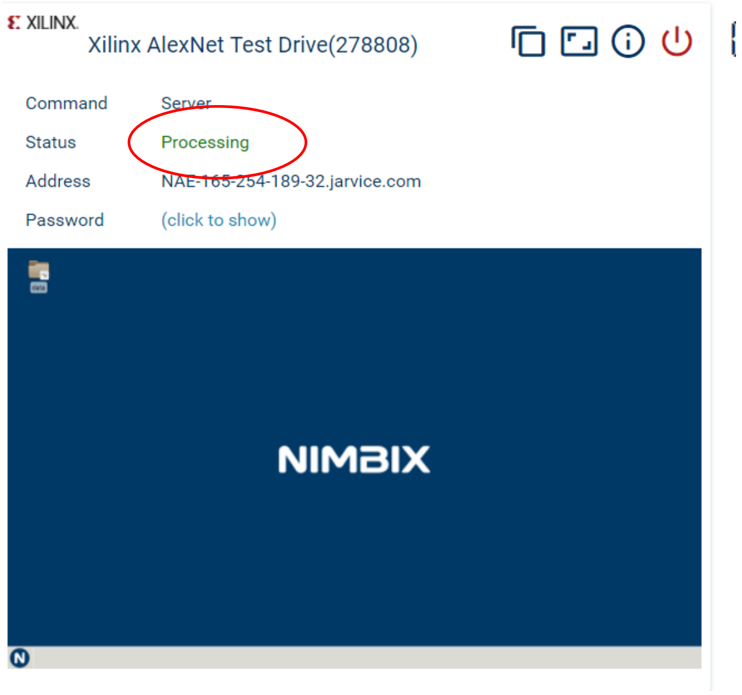
This will start the building of your session. It will take a couple of minutes, and will look like this:


Xilinx AlexNet Test Drive(277909)

Command: Server

Status: Queued...

Once the session is ready, the status will change to “Processing” and you can click the Nimbix desktop to connect via browser, or use the password to connect via ssh.


Xilinx AlexNet Test Drive(278808)

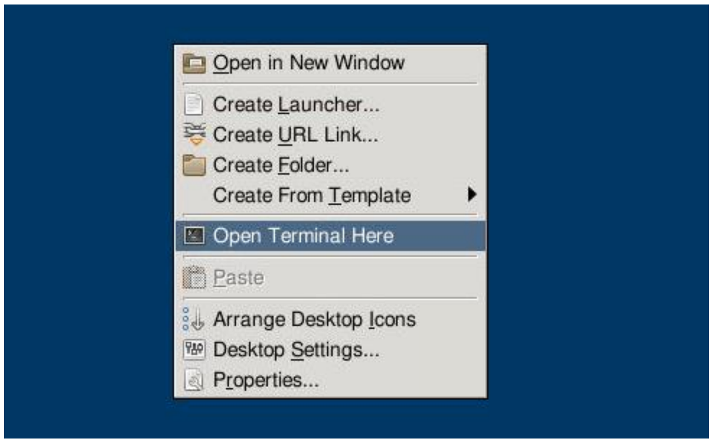
Command: Server

Status: Processing

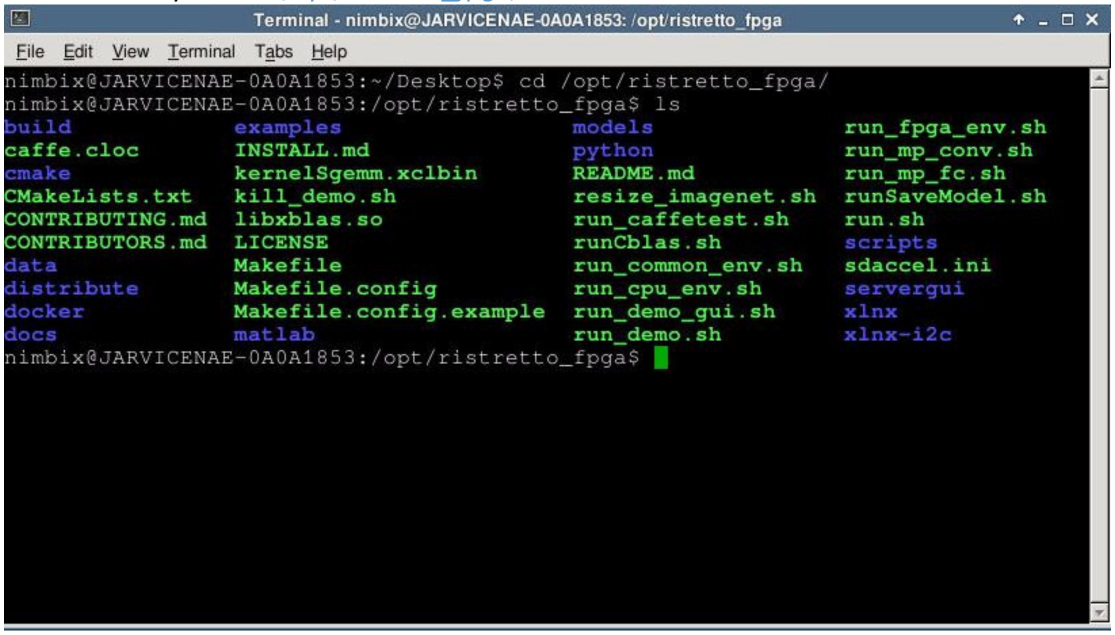
Address: NAE-165-254-189-32.jarvice.com

Password: (click to show)

Once you are in the session, right click on the desktop, and click “Open Terminal Here”

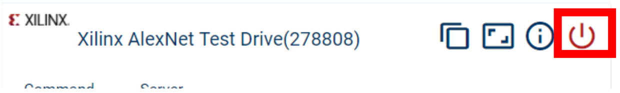


Navigate to the run directory with *cd /opt/ristretto_fpga/.*

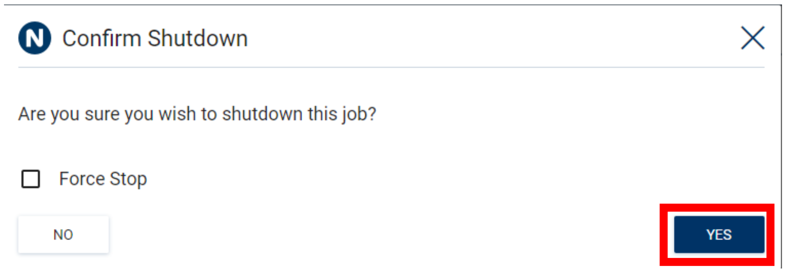

Terminal - nimbix@JARVICENAE-0A0A1853: /opt/ristretto_fpga
File Edit View Terminal Tabs Help

```
nimbix@JARVICENAE-0A0A1853:~/Desktop$ cd /opt/ristretto_fpga/
nimbix@JARVICENAE-0A0A1853:/opt/ristretto_fpga$ ls
build            examples                  models                run_fpga_env.sh
caffe.cloc       INSTALL.md                python                run_mp_conv.sh
cmake            kernelSgemm.xclbin        README.md             run_mp_fc.sh
CMakeLists.txt   kill_demo.sh              resize_imagenet.sh    runSaveModel.sh
CONTRIBUTING.md  libxblas.so               run_caffetest.sh      run.sh
CONTRIBUTORS.md  LICENSE                   runCblas.sh           scripts
data             Makefile                  run_common_env.sh     sdaccel.ini
distribute       Makefile.config           run_cpu_env.sh        servergui
docker           Makefile.config.example   run_demo_gui.sh       xlnx
docs             matlab                    run_demo.sh           xlnx-i2c
nimbix@JARVICENAE-0A0A1853:/opt/ristretto_fpga$
```

To end the session click on the red power icon



And then click yes.

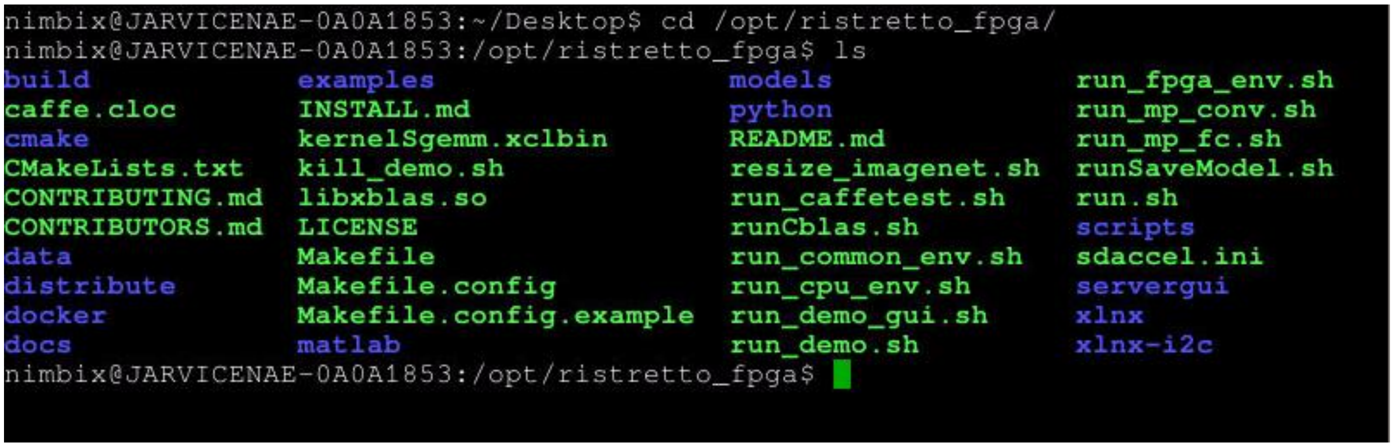


Step 3: Visit Xilinx Developer Forums (Optional)

If you have questions or comments about demo, please visit our Developer Forums:
https://forums.xilinx.com/t5/SDAccel/bd-p/SDx

File Structure

Most demo and runtime source is located in */ristrettto_fpga/.* This directory contains the run scripts and SDx runtime files.

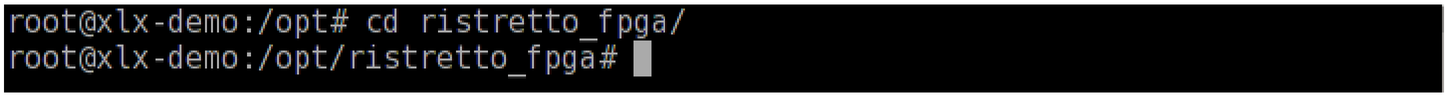

```
nimbix@JARVICENAE-0A0A1853:~/Desktop$ cd /opt/ristretto_fpga/
nimbix@JARVICENAE-0A0A1853:/opt/ristretto_fpga$ ls
build              examples                 models                run_fpga_env.sh
caffe.cloc         INSTALL.md               python                run_mp_conv.sh
cmake              kernelSgemm.xclbin       README.md             run_mp_fc.sh
CMakeLists.txt     kill_demo.sh             resize_imagenet.sh    runSaveModel.sh
CONTRIBUTING.md    libxblas.so              run_caffetest.sh      run.sh
CONTRIBUTORS.md    LICENSE                  runCblas.sh           scripts
data               Makefile                 run_common_env.sh     sdaccel.ini
distribute         Makefile.config          run_cpu_env.sh        servergui
docker             Makefile.config.example  run_demo_gui.sh       xlnx
docs               matlab                   run_demo.sh           xlnx-i2c
nimbix@JARVICENAE-0A0A1853:/opt/ristretto_fpga$
```

1. **Top Directory /ristretto_fpga/**
    1. All script files
        1. run_demo.sh –PoC performance demo
        2. run_caffetest.sh – main caffe script for running AlexNet
        3. run.sh – run script for running custom images
2. **/models/**
    1. Includes all models/prototext files included with Caffe
    2. /ristretto_fpga/models/bvlc_reference_caffenet – these are the models that will work for AlexNet
3. **/examples/images**
    1. Custom images directory
4. **../imagenet_val**
    1. Default directory with 50,000 images from ImageNet.

Running the Demo

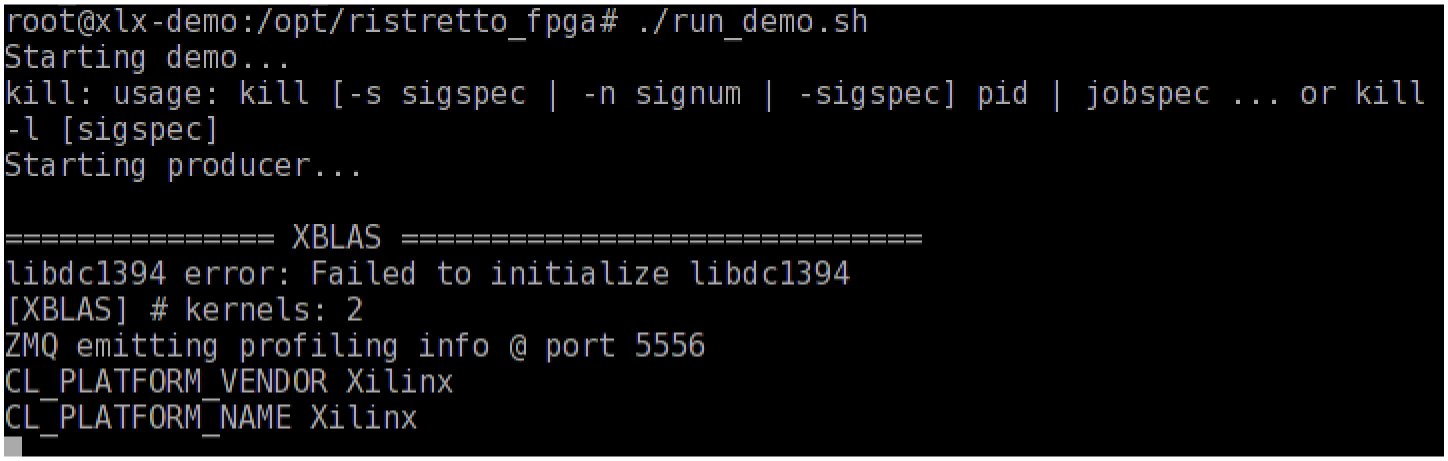
Launching Demo

1. Navigate to the */ristretto_fpga/* directory with *cd /ristretto_fpga*



```
root@xlx-demo:/opt# cd ristretto_fpga/
root@xlx-demo:/opt/ristretto_fpga# 
```

2. Execute *./run_demo.sh*

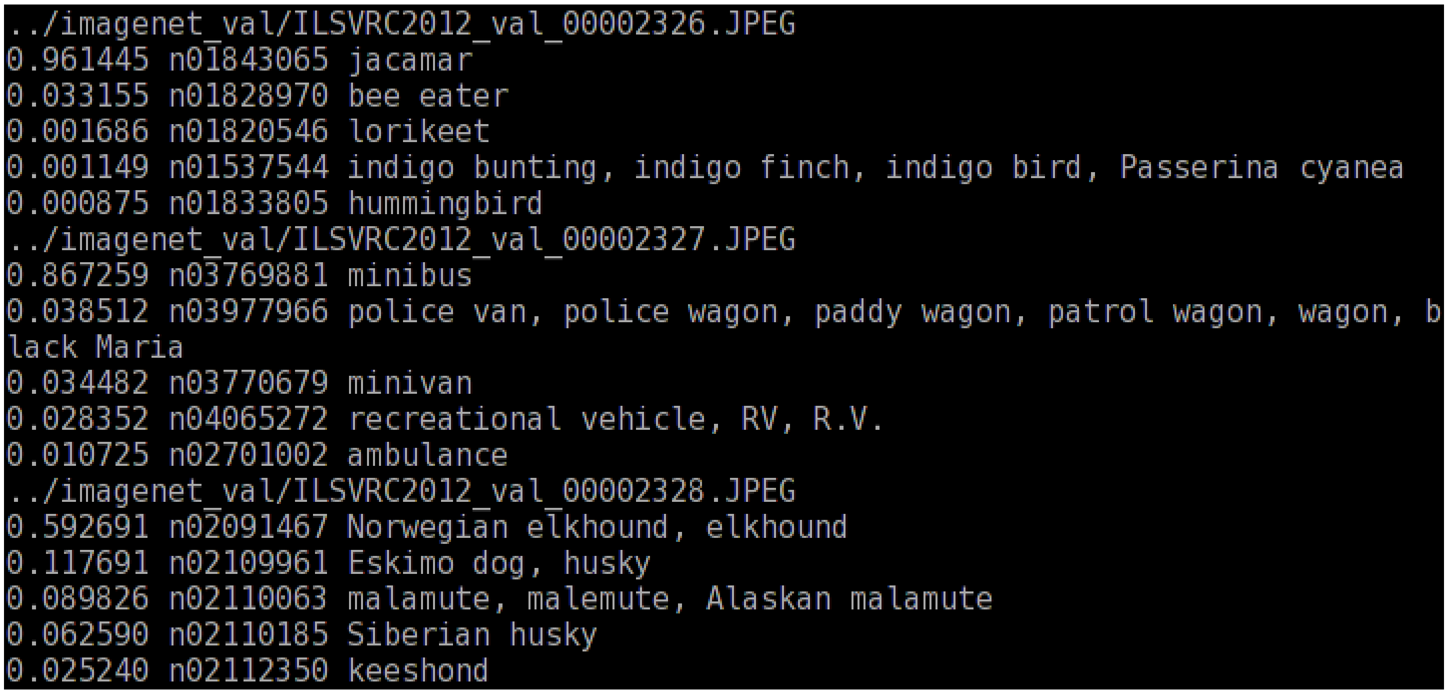


```
root@xlx-demo:/opt/ristretto_fpga# ./run_demo.sh
Starting demo...
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill
-l [sigspec]
Starting producer...

=============== XBLAS ===============================
libdc1394 error: Failed to initialize libdc1394
[XBLAS] # kernels: 2
ZMQ emitting profiling info @ port 5556
CL_PLATFORM_VENDOR Xilinx
CL_PLATFORM_NAME Xilinx
```

It will a take a few minutes to run through the runtime checks and verify that all of the dependencies are available. This console will display the running network and the images being classified as it runs through the images.

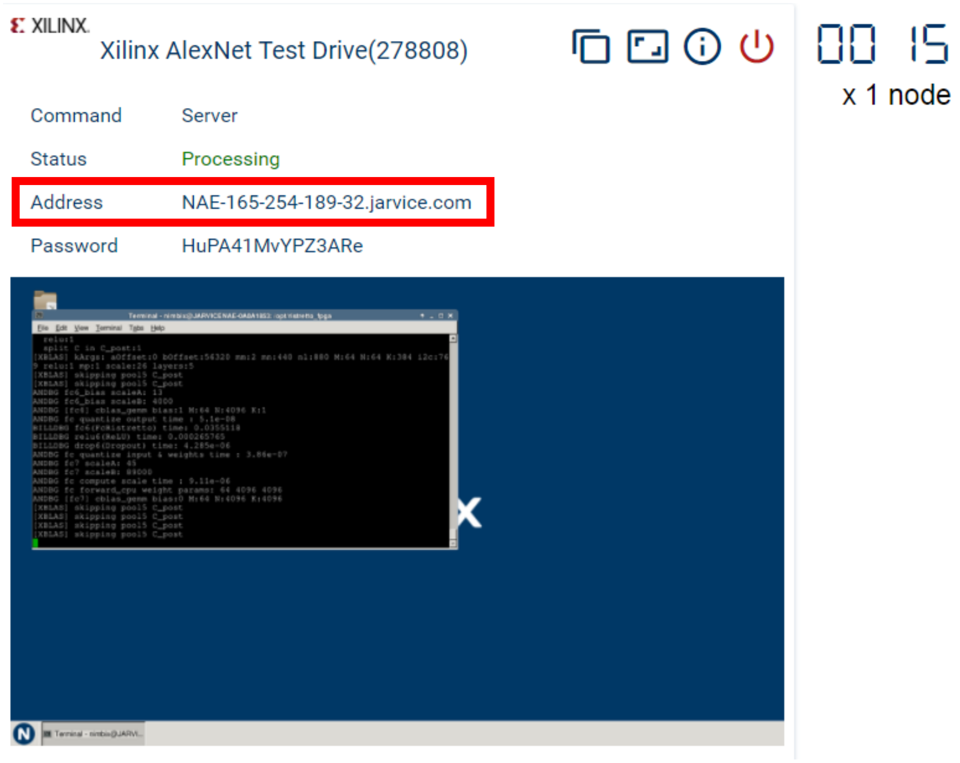
Once the design is fully running, the terminal will start showing the classifications scrolling by, as shown below.



```
../imagenet_val/ILSVRC2012_val_00002326.JPEG
0.961445 n01843065 jacamar
0.033155 n01828970 bee eater
0.001686 n01820546 lorikeet
0.001149 n01537544 indigo bunting, indigo finch, indigo bird, Passerina cyanea
0.000875 n01833805 hummingbird
../imagenet_val/ILSVRC2012_val_00002327.JPEG
0.867259 n03769881 minibus
0.038512 n03977966 police van, police wagon, paddy wagon, patrol wagon, wagon, b
lack Maria
0.034482 n03770679 minivan
0.028352 n04065272 recreational vehicle, RV, R.V.
0.010725 n02701002 ambulance
../imagenet_val/ILSVRC2012_val_00002328.JPEG
0.592691 n02091467 Norwegian elkhound, elkhound
0.117691 n02109961 Eskimo dog, husky
0.089826 n02110063 malamute, malemute, Alaskan malamute
0.062590 n02110185 Siberian husky
0.025240 n02112350 keeshond
```

This will also start a web server to send the output of the demo that can be accessed via any machine over the internet.

4. Open a Browser and navigate to: http://<sessionaddress>:8998/static/www/index.html. Replace the <sessionaddress> with the unique address of your nimbix session. To get this address, go back to your Nimbix dash board and copy the address.



Xilinx AlexNet Test Drive(278808)

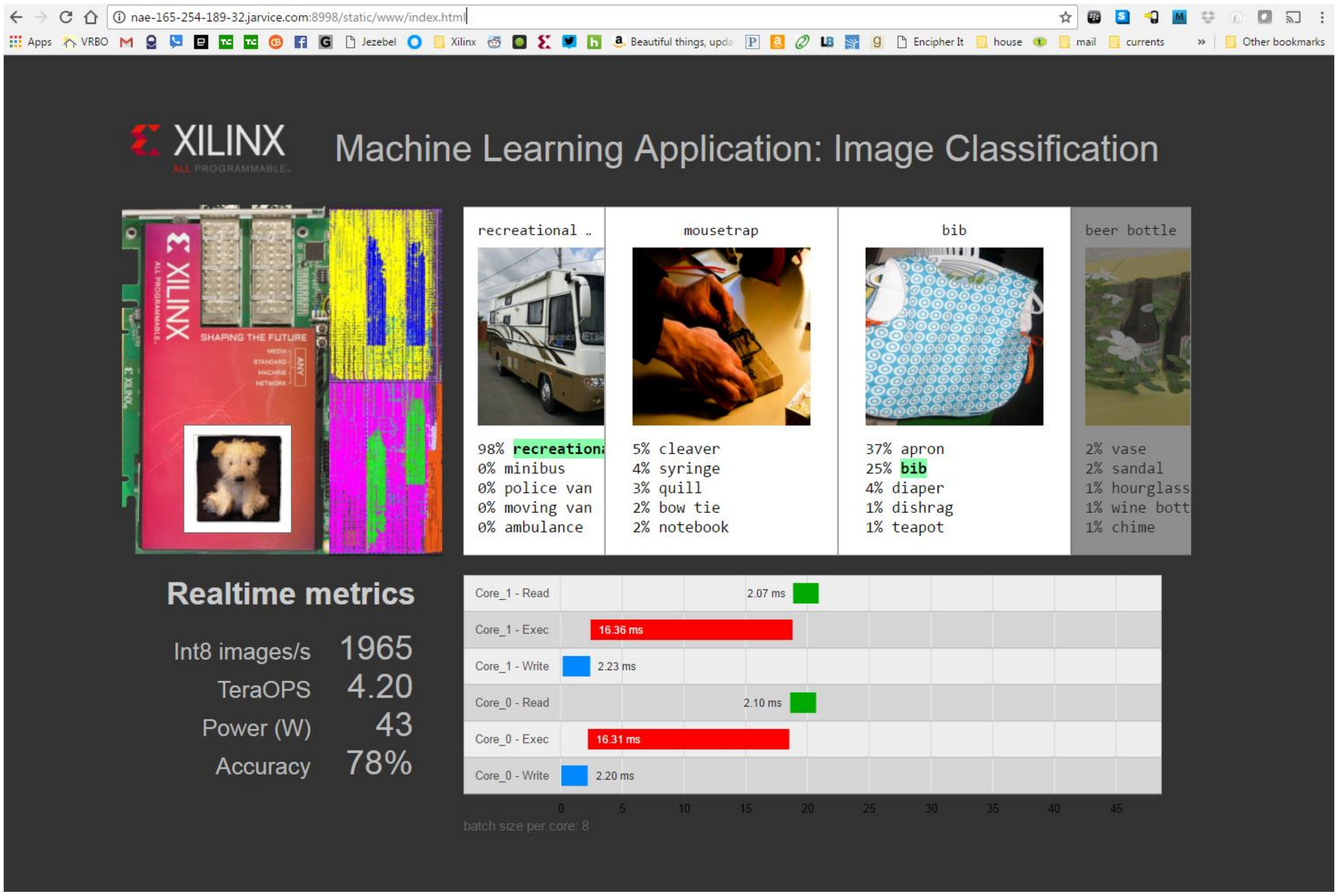
00 15
x 1 node

Command	Server
Status	Processing
Address	NAE-165-254-189-32.jarvice.com
Password	HuPA41MvYPZ3ARe

```
  relu:1
  split C in C_post:1
[XBLAS] kArgs: aOffset:0 bOffset:56320 mm:2 mn:440 nl:880 M:64 N:64 K:384 i2c:76
9 relu:1 mp:1 scale:26 layers:5
[XBLAS] skipping pool5 C_post
[XBLAS] skipping pool5 C_post
ANDBG fc6_bias scaleA: 13
ANDBG fc6_bias scaleB: 4000
ANDBG [fc6] cblas_gemm bias:1 M:64 N:4096 K:1
ANDBG fc quantize output time : 5.1e-08
BILLDBG fc6(FcRistretto) time: 0.0355118
BILLDBG relu6(ReLU) time: 0.000265765
BILLDBG drop6(Dropout) time: 4.285e-06
ANDBG fc quantize input & weights time : 3.86e-07
ANDBG fc7 scaleA: 45
ANDBG fc7 scaleB: 89000
ANDBG fc compute scale time : 9.11e-06
ANDBG fc forward_cpu weight params: 64 4096 4096
ANDBG [fc7] cblas_gemm bias:0 M:64 N:4096 K:4096
[XBLAS] skipping pool5 C_post
[XBLAS] skipping pool5 C_post
[XBLAS] skipping pool5 C_post
[XBLAS] skipping pool5 C_post
```

XILINX
Terminal - nimbix@JARVICENAE-0A8A1853: /opt/ristretto_fpga
File Edit View Terminal Tabs Help
Terminal - nimbix@JARVI...

The GUI should look like this, if everything has been launched correctly.


nae-165-254-189-32.jarvice.com:8998/static/www/index.html

Machine Learning Application: Image Classification

recreational ..	mousetrap	bib	beer bottle
98% recreationa	5% cleaver	37% apron	2% vase
0% minibus	4% syringe	25% bib	2% sandal
0% police van	3% quill	4% diaper	1% hourglass
0% moving van	2% bow tie	1% dishrag	1% wine bott
0% ambulance	2% notebook	1% teapot	1% chime

Realtime metrics

Int8 images/s	1965
TeraOPS	4.20
Power (W)	43
Accuracy	78%

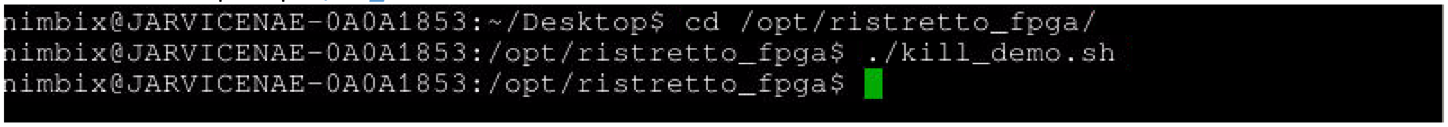
Core_1 - Read	2.07 ms
Core_1 - Exec	16.36 ms
Core_1 - Write	2.23 ms
Core_0 - Read	2.10 ms
Core_0 - Exec	16.31 ms
Core_0 - Write	2.20 ms

batch size per core: 8

Stopping the demo

1. In the terminal, navigate to the */ristretto_fpga/* directory with *cd /ristretto_fpga*

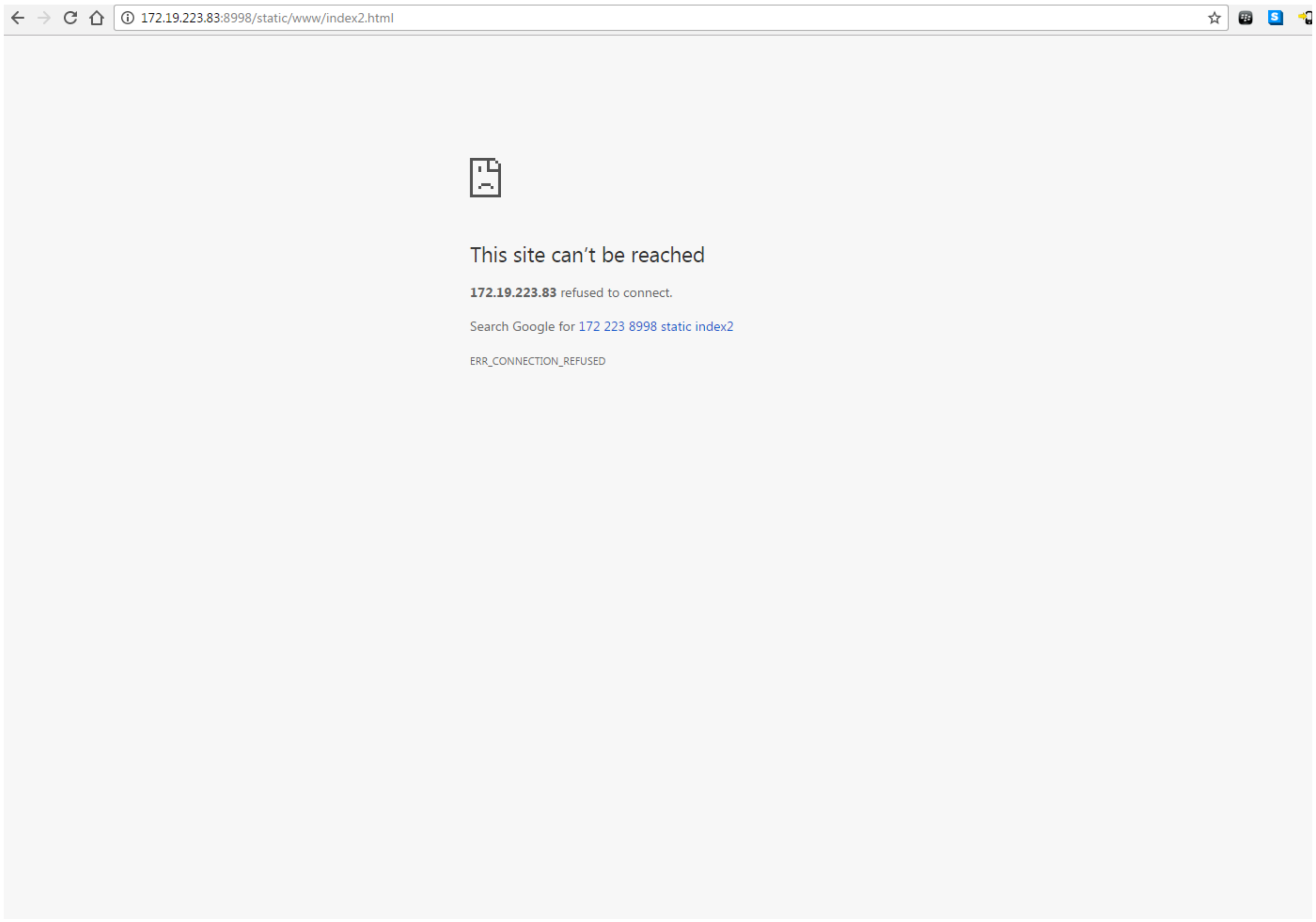
2. Execute the stop script: *./kill_demo.sh*


```
nimbix@JARVICENAE-0A0A1853:~/Desktop$ cd /opt/ristretto_fpga/
nimbix@JARVICENAE-0A0A1853:/opt/ristretto_fpga$ ./kill_demo.sh
nimbix@JARVICENAE-0A0A1853:/opt/ristretto_fpga$
```

3. Now in both of the open terminals you should see that the demo has stopped.

Troubleshooting

1. Cannot load GUI at the web address.



172.19.223.83:8998/static/www/index2.html

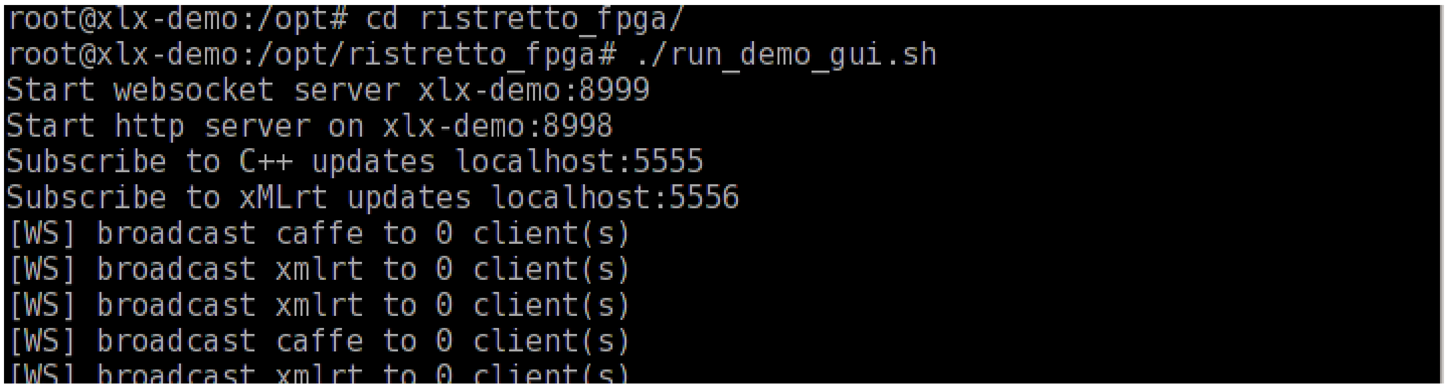
This site can't be reached

172.19.223.83 refused to connect.

Search Google for 172 223 8998 static index2

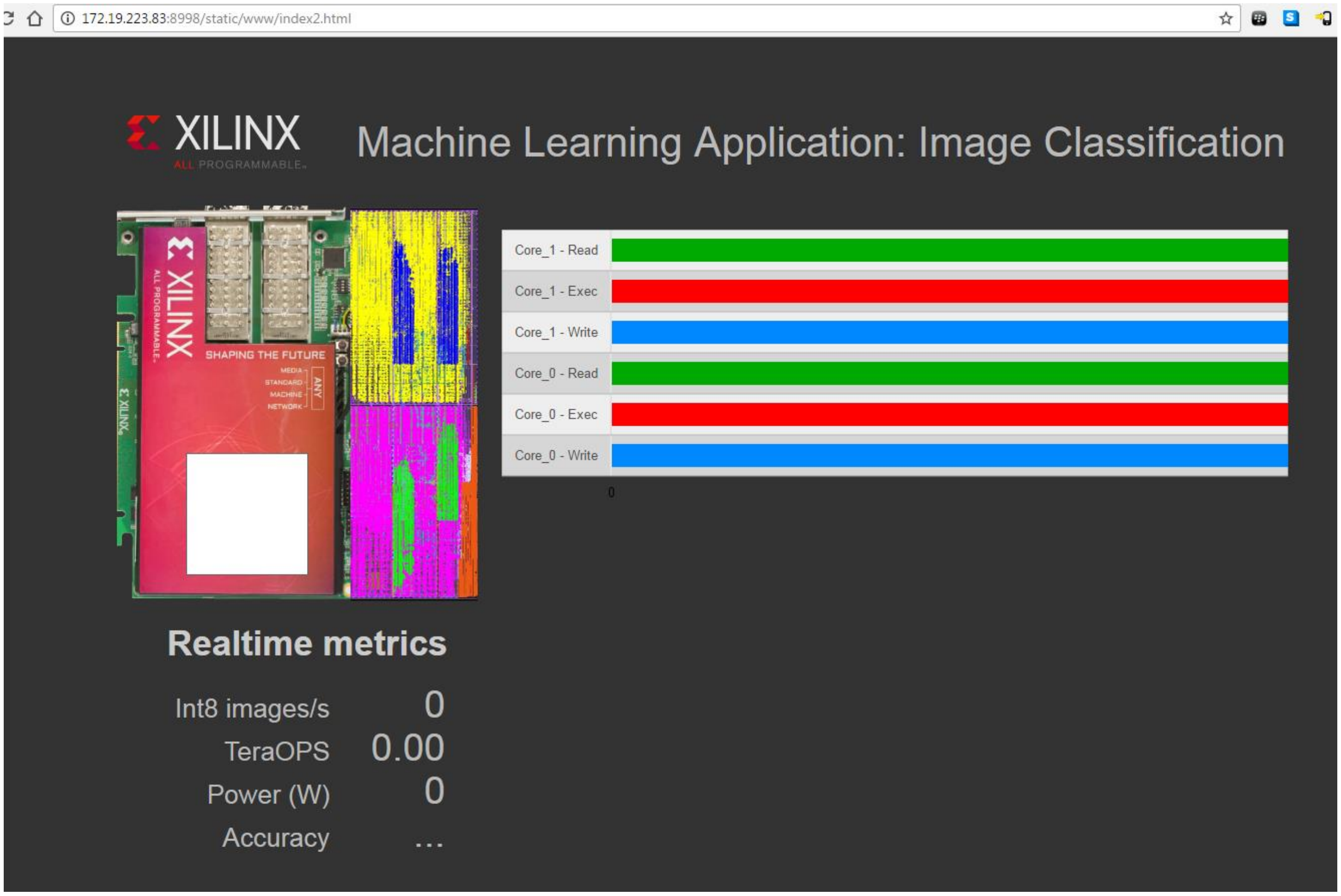
ERR_CONNECTION_REFUSED

This is caused by either the IP address of the host server being incorrect, or the *./run_demo_gui.sh* wasn't executed. Check that the IP address is correct and the terminal is running the GUI script. Below is an example of the running script.



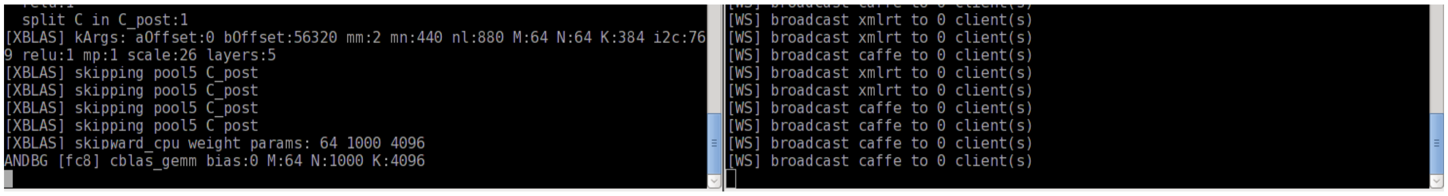
```
root@xlx-demo:/opt# cd ristretto_fpga/
root@xlx-demo:/opt/ristretto_fpga# ./run_demo_gui.sh
Start websocket server xlx-demo:8999
Start http server on xlx-demo:8998
Subscribe to C++ updates localhost:5555
Subscribe to xMLrt updates localhost:5556
[WS] broadcast caffe to 0 client(s)
[WS] broadcast xmlrt to 0 client(s)
[WS] broadcast xmlrt to 0 client(s)
[WS] broadcast caffe to 0 client(s)
[WS] broadcast xmlrt to 0 client(s)
```

2. The website loads, but there isn't any data.



If the demo looks like this, the webserver is running, but the demo kernel may not be running. Check the terminal where the *./run_demo.sh* script was executed and make sure it's still running.

Both of the open terminals should look like this:



```
  split C in C_post:1
[XBLAS] kArgs: aOffset:0 bOffset:56320 mm:2 mn:440 nl:880 M:64 N:64 K:384 i2c:76
9 relu:1 mp:1 scale:26 layers:5
[XBLAS] skipping pool5 C_post
[XBLAS] skipping pool5 C_post
[XBLAS] skipping pool5 C_post
[XBLAS] skipping pool5 C_post
[XBLAS] skipward_cpu weight params: 64 1000 4096
ANDBG [fc8] cblas_gemm bias:0 M:64 N:1000 K:4096
```

```
[WS] broadcast xmlrt to 0 client(s)
[WS] broadcast xmlrt to 0 client(s)
[WS] broadcast caffe to 0 client(s)
[WS] broadcast xmlrt to 0 client(s)
[WS] broadcast xmlrt to 0 client(s)
[WS] broadcast caffe to 0 client(s)
[WS] broadcast caffe to 0 client(s)
[WS] broadcast caffe to 0 client(s)
[WS] broadcast caffe to 0 client(s)
```

Defining Networks

Networks are defined by passing a prototxt file as a command line argument. An example of this is shown in the */ristretto/run_caffetest.sh* script:

```
./build/tools/caffe test --
model=models/bvlc_reference_caffenet/AlexOwt_iter_450000.prototxt.quantized --
weights=models/bvlc_reference_caffenet/AlexOwt_iter_450000.caffemodel --
iterations=100
```

./build/tools/caffe – C++ exe application to execute AlexNet through Caffe
test – Mode of application – Train, Test and Time
Leave in Test mode, but you can try in Time mode as well.

--model – network definition file. This test drive supports the following prototxt files:
AlexOwt_iter_450000.prototxt - floating point reference prototxt the demo is build on (not supported).
AlexOwt_iter_450000.prototxt.quantized – 8bit AlexNet Implementation. – Use for experimentation and customization.
AlexOwt_iter_450000_dummydata.prototxt.quantized – Optimized for to be used with the *run_dem.sh* script only.

--weights – Caffe model / pretrained weights. Must be trained on the same model as referenced above.

--iterations – how many iterations to run.

To change parameters or to load in alternate AlexNet Prototxt files edit the run_caffetest.sh and replace the model reference

- Original:

  -
    ```
    ./build/tools/caffe test --
    model=models/bvlc_reference_caffenet/AlexOwt_iter_450000.prototxt.quantized
    --weights=models/bvlc_reference_caffenet/AlexOwt_iter_450000.caffemodel --
    iterations=1000
    ```
- New Prototxt:

  -
    ```
    ./build/tools/caffe test --
    model=models/bvlc_reference_caffenet/AlexOwt_iter_450000_16.prototxt.quanti
    zed --weights=models/bvlc_reference_caffenet/AlexOwt_iter_450000.caffemodel
    --iterations=1000
    ```

For Iterations – adjust the argument:

-
  ```
  ./build/tools/caffe test --
  model=models/bvlc_reference_caffenet/AlexOwt_iter_450000.prototxt.quantized
  --weights=models/bvlc_reference_caffenet/AlexOwt_iter_450000.caffemodel --
  iterations=1000
  ```

Execute *run_caffetest.sh* to see results.

Defining Weights

Changing the weights are also done as a command line argument. In the *run_caffetest.sh* above, change the default location of the weights to the new weights you would like to use.

```
./build/tools/caffe test --
model=models/bvlc_reference_caffenet/AlexOwt_iter_450000.prototxt.quantized --
weights=models/bvlc_reference_caffenet/AlexOwt_iter_450000.caffemodel --
iterations=100
```

The Test Drive doesn't include any other weights you can try, but you may upload your own Caffe model and try it with this AlexNet test drive. Note, only AlexNet is supported in this test drive.

If your model is not 8-bit, the next section is a tutorial for porting your higher precision model to lower precision.

Quantizing Weights to 8 bit

We are discussing two approaches to convert the double/ single precision Caffe weights into lower precision weights.

- Approach#1: Ristretto-Caffe (Fine-tuning)
- Approach#2: Quick Evaluation (Direct Float to Fixed Conversion)

Approach#1 (in the order of few hours) is more time-consuming when compared to approach#2 (in the order of few seconds). The table below compares the accuracies obtained with different approaches.

Table 1: Accuracy Comparison

Network	TestData Set	Default Caffe	Approach#1	Approach#2
GoogleNet	ImageNet Validation set(50k Images)	67.35	66.15	65.33
AlexNet	ImageNet Validation set(50k Images)	55.66	54.01	53.55

Ristretto-Caffe Approach (Fine-tuning):

Here, we use Ristretto-Caffe which is used to find the optimal bit widths based on the span of the range of values of weights & biases (layer-wise) and then perform fine-tuning of these parameters (using backpropagation) for a few iterations. This helps the weights to re-adjust themselves under the specified bit widths during the fine-tuning process and results in a new set of weights that incur lesser loss in accuracy when compared to direct float to fixed conversion (Approach#2).

The fine-tuning stage is compute intensive as it involves a backpropagation technique, but it results in slightly better accuracy. Details on how to perform fine-tuning can be found here.

Assumptions:

1. Caffe Ristretto is installed successfully(http://lepsucd.com/?page_id=621) & added to the system path variable($PATH)
2. 32-bit Caffe weights & their deploy.prototxt are available.

Steps:

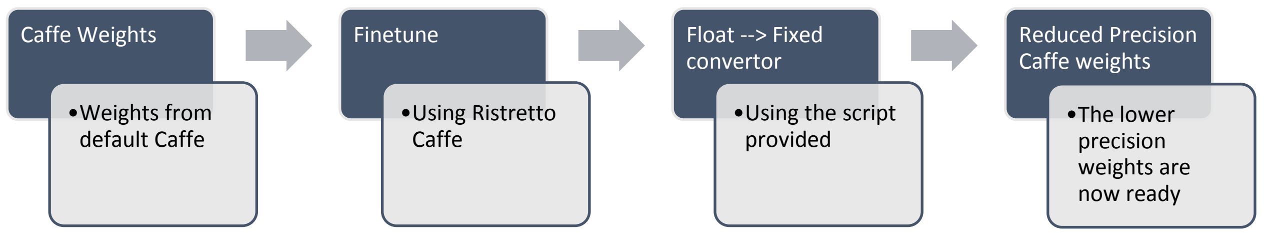

Caffe Weights
•Weights from default Caffe
Finetune
•Using Ristretto Caffe
Float --> Fixed convertor
•Using the script provided
Reduced Precision Caffe weights
•The lower precision weights are now ready

Figure 1: Ristretto-Caffe

Quick Evaluation Approach (Direct Float to Fixed Conversion):

In this approach, we perform the standard floating point to fixed point conversion. The bit widths for the integer as well as the fractional parts can be supplied as command line arguments to this script. By default, bit width for the parameters (params) is set to 8bits and the fractional part is set to 7bits. We use a round to the nearest even rounding scheme during the format conversion.

Assumptions:

1. Caffe installed successfully & added to the system path variable($PATH)
2. 32-bit Caffe weights & its deploy.prototxt available.

Steps:

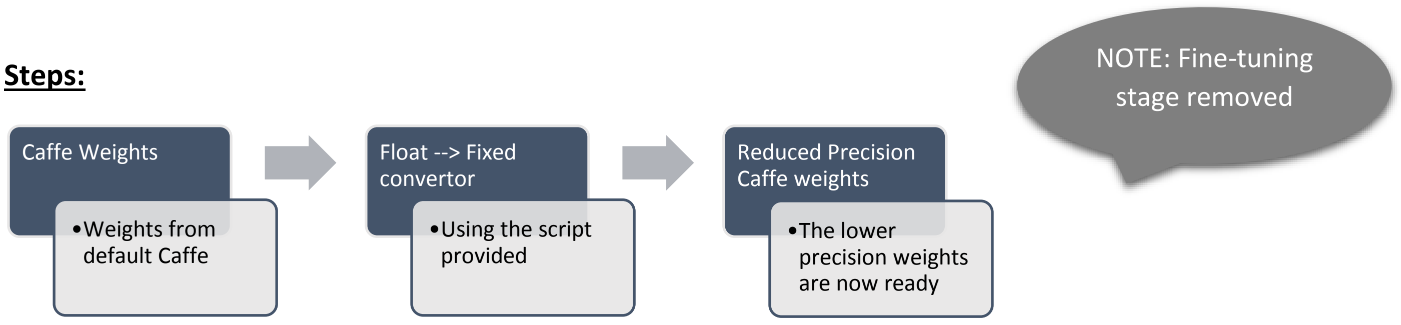

NOTE: Fine-tuning stage removed
Caffe Weights
•Weights from default Caffe
Float --> Fixed convertor
•Using the script provided
Reduced Precision Caffe weights
•The lower precision weights are now ready

Figure 2: Quick Evaluation Flow

The script takes the following arguments:

--deploy: Path for the Caffe deploy file.

--caffemodel: Path for the Caffe model file (default).

--outputmodel: Name for the output Caffe model whose values are in the lower precision range.

--bwparams: (Optional) Default value is set to 8 bits. Specifies the BitWidth for the parameters of the Caffe model.

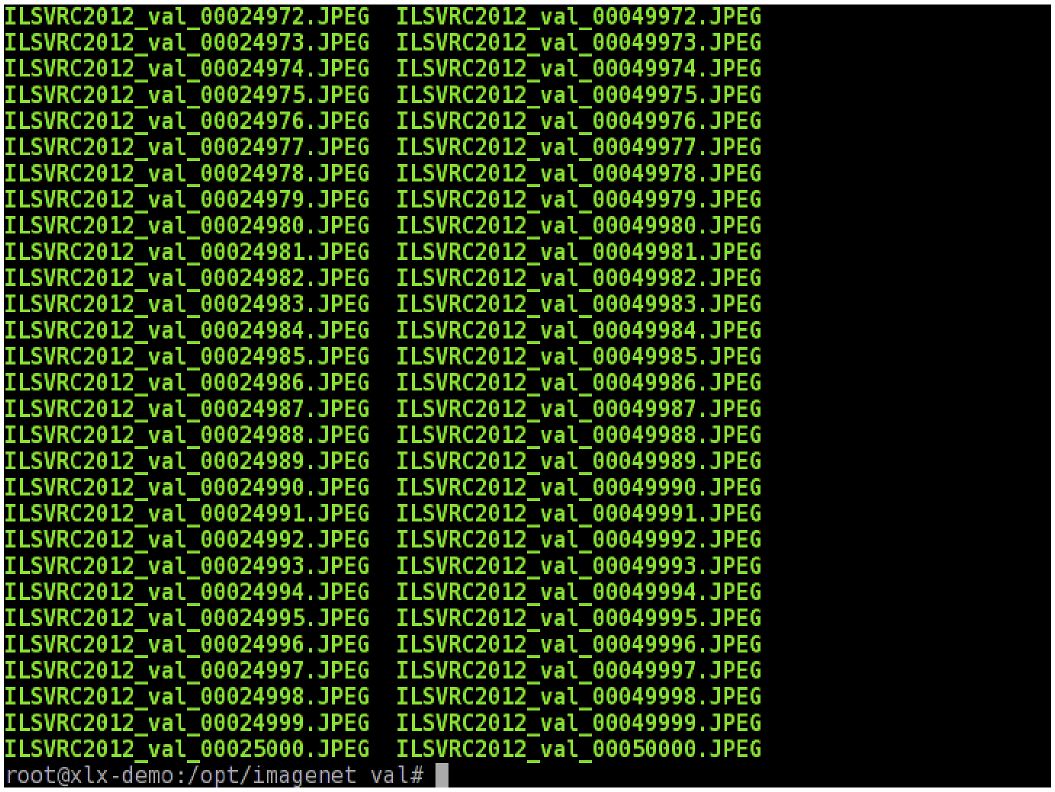
--fracparams: (Optional) Default value is set to 7 bits. Specifies the BitWidth for the fractional part.

Usage Example:

python convertmodel.py --deploy=deploy.prototxt --caffemodel=alexnet_train_iter_360000.caffemodel --outputmodel=new.caffemodel --bwparams=8 --fracparams=7

Custom Images

By default, the demo comes preloaded with 50,000 ImageNet images, all of which are 227x227 in size. These are located in the ..*/imagenet_val/* directory.



```
ILSVRC2012_val_00024972.JPEG  ILSVRC2012_val_00049972.JPEG
ILSVRC2012_val_00024973.JPEG  ILSVRC2012_val_00049973.JPEG
ILSVRC2012_val_00024974.JPEG  ILSVRC2012_val_00049974.JPEG
ILSVRC2012_val_00024975.JPEG  ILSVRC2012_val_00049975.JPEG
ILSVRC2012_val_00024976.JPEG  ILSVRC2012_val_00049976.JPEG
ILSVRC2012_val_00024977.JPEG  ILSVRC2012_val_00049977.JPEG
ILSVRC2012_val_00024978.JPEG  ILSVRC2012_val_00049978.JPEG
ILSVRC2012_val_00024979.JPEG  ILSVRC2012_val_00049979.JPEG
ILSVRC2012_val_00024980.JPEG  ILSVRC2012_val_00049980.JPEG
ILSVRC2012_val_00024981.JPEG  ILSVRC2012_val_00049981.JPEG
ILSVRC2012_val_00024982.JPEG  ILSVRC2012_val_00049982.JPEG
ILSVRC2012_val_00024983.JPEG  ILSVRC2012_val_00049983.JPEG
ILSVRC2012_val_00024984.JPEG  ILSVRC2012_val_00049984.JPEG
ILSVRC2012_val_00024985.JPEG  ILSVRC2012_val_00049985.JPEG
ILSVRC2012_val_00024986.JPEG  ILSVRC2012_val_00049986.JPEG
ILSVRC2012_val_00024987.JPEG  ILSVRC2012_val_00049987.JPEG
ILSVRC2012_val_00024988.JPEG  ILSVRC2012_val_00049988.JPEG
ILSVRC2012_val_00024989.JPEG  ILSVRC2012_val_00049989.JPEG
ILSVRC2012_val_00024990.JPEG  ILSVRC2012_val_00049990.JPEG
ILSVRC2012_val_00024991.JPEG  ILSVRC2012_val_00049991.JPEG
ILSVRC2012_val_00024992.JPEG  ILSVRC2012_val_00049992.JPEG
ILSVRC2012_val_00024993.JPEG  ILSVRC2012_val_00049993.JPEG
ILSVRC2012_val_00024994.JPEG  ILSVRC2012_val_00049994.JPEG
ILSVRC2012_val_00024995.JPEG  ILSVRC2012_val_00049995.JPEG
ILSVRC2012_val_00024996.JPEG  ILSVRC2012_val_00049996.JPEG
ILSVRC2012_val_00024997.JPEG  ILSVRC2012_val_00049997.JPEG
ILSVRC2012_val_00024998.JPEG  ILSVRC2012_val_00049998.JPEG
ILSVRC2012_val_00024999.JPEG  ILSVRC2012_val_00049999.JPEG
ILSVRC2012_val_00025000.JPEG  ILSVRC2012_val_00050000.JPEG
root@xlx-demo:/opt/imagenet_val#
```

To test other images through the model, use the *run.sh* script. This script exposes the data directories, and custom images can be added here.

1. Load the image to the *./examples/images* directory

2. Edit the *run.sh* script to include the new images

```
./build/examples/cpp_classification/classification.bin
models/bvlc_reference_caffenet/AlexOwt_iter_450000_dummydata.prototxt.quantized
\


models/bvlc_reference_caffenet/AlexOwt_iter_450000.caffemodel \


data/ilsvrc12/imagenet_mean.binaryproto data/ilsvrc12/synset_words.txt \

./examples/images/cat_gray.jpg  ./examples/images/cat_gray.jpg \
   ./examples/images/cat.jpg       ./examples/images/fish-bike.jpg \
```

Add the new images or directory of images to the bottom of the script:

```
   ./examples/images/cat_gray.jpg  ./examples/images/cat_gray.jpg \
   ./examples/images/cat.jpg       ./examples/images/fish-bike.jpg \
   ./examples/images/newimage1.jpg    ./examples/images/newimage2.jpg \
   ./examples/images/newimage3jpeg   ./examples/images/newimage4.jpg \
```

3. Execute *run.sh.* The output will be the prediction results for the new images.

```
---------- Prediction 6 for ./examples/images/coffee.jpeg -------
---
0.7616 - "n07920052 espresso"
0.0752 - "n03249569 drum, membranophone, tympan"
0.0328 - "n03250847 drumstick"
0.0148 - "n03297495 espresso maker"
0.0055 - "n02823750 beer glass"
```

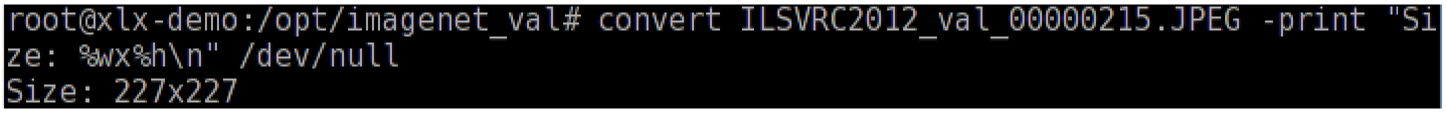
The images need to be compliant with ImageNet images:

- File Type: JPEG image data, JFIF standard 1.01
- Resolution/size: 227x227

If the images you want to use are not 227x227, they will need to be resized to the supported resolution.

To check the resolution of the image you want to use, type: *convert <file name>.JPEG -print "Size: %wx%h\n" /dev/null*

This will display the image size of the image:



```
root@xlx-demo:/opt/imagenet_val# convert ILSVRC2012_val_00000215.JPEG -print "Si
ze: %wx%h\n" /dev/null
Size: 227x227
```

To resize custom images to the supported resolution, use the convert command with –resize option: *convert -resize 227x227\! $name $name*

Specifications / Definitions:

Application: Image Classification

Image classification is the task of accurately identifying or classifying images based on a set of classifiers defined in the pre-trained model.

Framework: Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.

Link to Caffe: http://caffe.berkeleyvision.org/ , https://github.com/BVLC/caffe

This demo uses the Ristretto branch from Caffe, that adds support for fixed point networks. Ristretto is an automated CNN-approximation tool which condenses 32-bit floating point networks. Ristretto is an extension of Caffe and allows to test, train and fine-tune networks with limited numerical precision. Ristretto allows users to automatically quantize networks using different bit-widths, deploy Caffe network layers and train new models in lower precision.

Links to Ristretto: http://lepsucd.com/?page_id=621 , https://github.com/pmgysel/caffe

Network: AlexNet

This demo uses the default Alexnet network as described in the BVLC Reference CaffeNet in models/bvlc_reference_caffenet: AlexNet trained on ILSVRC 2012, with a minor variation from the version as described in ImageNet classification with deep convolutional neural networks by Krizhevsky et al. in NIPS 2012. (Trained by Jeff Donahue @jeffdonahue)

The trained model comes from the Caffe Model Zoo.

Dataset: ImageNet

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.

To use custom images they need to be compliant with ImageNet images (i.e. 227x227)

Custom images need to be in the */imagenet_val* directory.

Links to ImageNet: http://image-net.org/index

Precision: 8Bit

Tools: SDx 2016.4

Platform: XLX KCU115

The Kintex® UltraScale™ FPGA Acceleration Development Kit is an excellent starting point for cloud application developers. This kit is based on a production-ready PCI card accessible in the cloud with the frameworks, libraries, drivers and development tools to support easy application programming with OpenCL, C, C++ through SDAccel. Get started in the cloud, with this kit or both to move through development with Xilinx and go to production with one of Xilinx's ecosystem partners.

For more information visit: https://www.xilinx.com/products/boards-and-kits/xcku115-2flvb2104e.html

References

Comparative Study of Deep Learning Software Frameworks https://arxiv.org/pdf/1511.06435.pdf

XILINX
ALL PROGRAMMABLE™