Accelerate ML inference on Raspberry Pi with PyArmNN

Non-Confidential

Copyright © 2020 Arm Limited (or its affiliates).
All rights reserved.

Issue 1.0
102107_0000_01

Accelerate ML inference on Raspberry Pi with PyArmNN

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	12th October 2020	Non-confidential	First release

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at **http://www.arm.com/company/policies/trademarks**.

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

www.arm.com

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Contents

1 Overview **5**

2 Before you begin **6**

3 What are Arm NN and PyArmNN? **9**

4 Run ML Inference with PyArmNN **10**

4.1 The example script 10

4.2 Running the example script 12

4.3 Explaining the example script 12

5 Performance comparison: PyArmNN and TensorFlow Lite **15**

6 Related information **18**

7 Next steps **19**

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1 Overview

This guide shows you how to train a neural network that can recognize fire in images. The ability to recognize fire means that the neural network can make fire-detection systems more reliable and cost-effective. This guide shows you how to use the Python Application Programming Interfaces (APIs) for the Arm NN inference engine to build a sample application that classifies images as fire or non-fire.

This guide uses a Raspberry Pi 3 or 4 device. The Raspberry Pi device is powered by an Arm CPU with Neon architecture. Neon is an optimization architecture extension for Arm processors. Neon is designed for:

- Faster video processing
- Image processing
- Speech recognition
- Machine Learning

Neon provides Single Instruction Multiple Data (SIMD) instructions, where multiple processing elements in the pipeline perform operations on multiple data points simultaneously. Arm NN provides the APIs to harness the power of the Neon backend.

At the end of this guide, you will be able to:

- Run a Python script that predicts whether a supplied image contains fire or not
- Explain what the differences are between Arm NN and PyArmNN

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2 Before you begin

To work through this guide, you need the following:

- A Raspberry Pi 3 or 4 device. This guide was developed using a Raspberry Pi 4 with Raspbian 10 OS.
- On your Raspberry Pi:
  - You must check out and build Arm NN version 20.05 or newer for your Raspberry Pi.
  - You must install the PyArmNN package.

    The Read Me, contains useful information. You must have SWIG installed.

    Note: For complete and up-to-date installation information, always refer to the previous README links. However, for your convenience, at the end of this section we provide a list of the commands we used to install Arm NN and PyArmNN.
- On your computer:
  - You must have `fire_detection.tflite`, generated from this guide and converted to a TensorFlow Lite model.
  - To convert your file to TensorFlow Lite model, use the following code:

```
tflite_convert \
--output_file=/tmp/fire_detection.tflite  \
--saved_model_dir=/tmp/keras-fire-detection/output/fire_detection.model
```

Note: You must convert your file to a tflite model on a machine running Windows or Linux.

Note: This Git hub repository has already converted the TensorFlow Lite model.

[NOTE FOR BUILD: PUT THE REST OF THIS SECTION AS A SINGLE CODE BLOCK INSIDE A COLLAPSIBLE ACCORDIAN, AND COLLAPSE IT BY DEFAULT]

The following code builds Arm NN and installs PyArmNN:

```
# Increase virtual memory swapfile allocation
sudo vi /etc/dphys-swapfile
# Find the following line:
#    CONF_SWAPSIZE=100
# Change this line to:
#    CONF_SWAPSIZE=1024
sudo /etc/init.d/dphys-swapfile stop
sudo /etc/init.d/dphys-swapfile start


# Install SCONS and CMAKE
sudo apt-get update
sudo apt-get install scons
sudo apt-get install cmake

mkdir armnn-tflite && cd armnn-tflite
export BASEDIR=`pwd`
git clone https://github.com/Arm-software/ComputeLibrary.git
git clone https://github.com/Arm-software/armnn
wget https://dl.bintray.com/boostorg/release/1.64.0/source/boost_1_64_0.tar.bz2
```

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
tar xf boost_1_64_0.tar.bz2
git clone -b v3.5.0 https://github.com/google/protobuf.git
git clone https://github.com/tensorflow/tensorflow.git
cd tensorflow/
git checkout 590d6eef7e91a6a7392c8ffffb7b58f2e0c8bc6b
git clone https://github.com/google/flatbuffers.git
cd $BASEDIR/ComputeLibrary
scons extra_cxx_flags="-fPIC" benchmark_tests=0 validation_tests=0 neon=1
cd $BASEDIR/boost_1_64_0
./bootstrap.sh
./b2 --build-dir=$BASEDIR/boost_1_64_0/build toolset=gcc link=static cxxflags=-fPIC --
with-filesystem --with-test --with-log --with-program_options install --
prefix=$BASEDIR/boost
cd $BASEDIR/protobuf
git submodule update --init --recursive
sudo apt-get install autoconf
sudo apt-get install libtool
./autogen.sh
./configure --prefix=$BASEDIR/protobuf-host
make
make install
cd $BASEDIR/tensorflow
../armnn/scripts/generate_tensorflow_protobuf.sh ../tensorflow-protobuf ../protobuf-
host
cd $BASEDIR
git clone https://github.com/google/flatbuffers.git
cd $BASEDIR/flatbuffers
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release
make

#Install SWIG
sudo apt-get install libpcre3 libpcre3-dev
cd $BASEDIR
mkdir swig
cd swig
wget http://prdownloads.sourceforge.net/swig/swig-4.0.2.tar.gz
chmod 777 swig-4.0.2.tar.gz
tar -xzvf swig-4.0.2.tar.gz
cd swig-4.0.2/
./configure --prefix=/home/pi/armnn-tflite/swigtool/
sudo make
sudo make install
sudo vi /etc/profile

# Add the following lines to /etc/profile
#   export SWIG_PATH=/home/pi/armnn-tflite/swigtool/bin
#   export PATH=$SWIG_PATH:$PATH

source /etc/profile
# Build Arm NN

cd $BASEDIR/armnn
mkdir build
cd build
cmake .. -DARMCOMPUTE_ROOT=$BASEDIR/ComputeLibrary -
DARMCOMPUTE_BUILD_DIR=$BASEDIR/ComputeLibrary/build -DBOOST_ROOT=$BASEDIR/boost -
DTF_GENERATED_SOURCES=$BASEDIR/tensorflow-protobuf -DPROTOBUF_ROOT=$BASEDIR/protobuf-
host -DBUILD_TF_LITE_PARSER=1 -
DTF_LITE_GENERATED_PATH=$BASEDIR/tensorflow/tensorflow/lite/schema -
DFLATBUFFERS_ROOT=$BASEDIR/flatbuffers -
DFLATBUFFERS_LIBRARY=$BASEDIR/flatbuffers/libflatbuffers.a -DSAMPLE_DYNAMIC_BACKEND=1 -
DDYNAMIC_BACKEND_PATHS=$BASEDIR/armnn/src/dynamic/sample -DARMCOMPUTENEON=1 -
DBUILD_TF_PARSER=1
make
cp $BASEDIR/armnn/build/*.so $BASEDIR/armnn/
cd /home/pi/armnn-tflite/armnn/src/dynamic/sample
```

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
mkdir build
cd build
cmake -DBOOST_ROOT=$BASEDIR/boost  -
DBoost_SYSTEM_LIBRARY=$BASEDIR/boost/lib/libboost_system.a -
DBoost_FILESYSTEM_LIBRARY=$BASEDIR/boost/lib/libboost_filesystem.a -
DARMNN_PATH=$BASEDIR/armnn/libarmnn.so ..
make


# Install PYARMNN
# Following instructions for "Standalone build" from:
# https://git.mlplatform.org/ml/armnn.git/tree/python/pyarmnn/README.md

export SWIG_EXECUTABLE=$BASEDIR/swigtool/bin/swig
export ARMNN_INCLUDE=$BASEDIR/armnn/include/
export ARMNN_LIB=$BASEDIR/armnn/build/
cd $BASEDIR/armnn/python/pyarmnn
sudo apt-get install python3.6-dev build-essential checkinstall libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev

python3 setup.py clean --all
python3 swig_generate.py -v
python3 setup.py build_ext --inplace
python3 setup.py sdist
python3 setup.py bdist_wheel
pip3 install dist/pyarmnn-21.0.0-cp37-cp37m-linux_armv7l.whl
sudo pip3 install opencv-python==3.4.6.27
sudo apt-get install libcblas-dev
sudo apt-get install libhdf5-dev
sudo apt-get install libhdf5-serial-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install libjasper-dev
sudo apt-get install libqtgui4
sudo apt-get install libqt4-test
```

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3 What are Arm NN and PyArmNN?

This section of the guide explains Arm NN and PyArmNN.

Arm NN is an inference engine for CPUs, GPUs, and NPUs. Arm NN executes a Machine Learning (ML) model on-device to make predictions based on input data. Arm NN enables efficient translation of existing neural network frameworks, such as TensorFlow Lite, TensorFlow, ONNX, and Caffe, allowing them to run efficiently and without modification across Arm Cortex-A CPUs, Arm Mali GPUs, and Arm Ethos NPUs.

PyArmNN is a Python extension for Arm NN SDK. In this guide, we use PyArmNN APIs to run the fire detection image classification model `fire_detection.tflite` described in Fire and smoke detection with Keras and Deep Learning. In this guide we compare the inference performance with TensorFlow Lite on Raspberry Pi.

Arm NN provides a TFLite parser: `armnnTfLiteParser`, which is a library for loading neural networks defined by TensorFlow Lite FlatBuffers files into the Arm NN runtime. We are going to use the TFLite parser to parse our fire detection model for fire or non-fire image classification.

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4 Run ML Inference with PyArmNN

This section of the guide shows you how to set up a Machine Learning (ML) model on your Raspberry Pi. You will perform the following steps:

- Import the PyArmNN module
- Load an input image
- Create a parser and load the network.
- Choose backends, create the runtime, and optimize the model
- Perform inference
- Interpret and report the output

The example code that performs these steps is `predict_pyarmnn.py`. To add `predict_PyArmNN.py` to your Raspberry Pi, you can clone it from the GitHub repository, or you can copy the following code into a python file on your Raspberry Pi.

The example code is examined in Explaining the example script.

4.1 The example script

The complete `predict_pyarmnn.py` code is as follows:

```
import pyarmnn as ann
import numpy as np
import cv2
import argparse

print('Working with Arm NN version ' + ann.ARMNN_VERSION)

#Load an image
parser = argparse.ArgumentParser(
      formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument(
      '--image', help='File path of image file', required=True)
args = parser.parse_args()

image = cv2.imread(args.image)
image = cv2.resize(image, (128, 128))
image = np.array(image, dtype=np.float32) / 255.0
print(image.shape)
```

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
# ONNX, Caffe and TF parsers also exist.
parser = ann.ITfLiteParser()
network = parser.CreateNetworkFromBinaryFile('./fire_detection.tflite')

graph_id = 0
input_names = parser.GetSubgraphInputTensorNames(graph_id)
input_binding_info = parser.GetNetworkInputBindingInfo(graph_id, input_names[0])
input_tensor_id = input_binding_info[0]
input_tensor_info = input_binding_info[1]
print('tensor id: ' + str(input_tensor_id))
print('tensor info: ' + str(input_tensor_info))
# Create a runtime object that will perform inference.
options = ann.CreationOptions()
runtime = ann.IRuntime(options)

# Backend choices earlier in the list have higher preference.
preferredBackends = [ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]
opt_network, messages = ann.Optimize(network, preferredBackends,
runtime.GetDeviceSpec(), ann.OptimizerOptions())

# Load the optimized network into the runtime.
net_id, _ = runtime.LoadNetwork(opt_network)
print("Loaded network, id={net_id}")
# Create an inputTensor for inference.
input_tensors = ann.make_input_tensors([input_binding_info], [image])

# Get output binding information for an output layer by using the layer name.
output_names = parser.GetSubgraphOutputTensorNames(graph_id)
output_binding_info = parser.GetNetworkOutputBindingInfo(0, output_names[0])
output_tensors = ann.make_output_tensors([output_binding_info])

runtime.EnqueueWorkload(0, input_tensors, output_tensors)
results = ann.workload_tensors_to_ndarray(output_tensors)
print(results[0])
j = np.argmax(results[0])
if j == 0:
    print("Non-Fire")
else:
    print("Fire")
```

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.2 Running the example script

Run the Python script from the command line, as shown in the following code:

```
$ python3 predict_pyarmnn.py --image ./images/opencountry_land663.jpg
```

In the preceding code, the following picture was used:

You should get the following output:

```
Working with Arm NN version 21.0.0
(128, 128, 3)


tensor id: 15616,
tensor info: TensorInfo{DataType: 1, IsQuantized: 0, QuantizationScale: 0.000000,
QuantizationOffset: 0, NumDimensions: 4, NumElements: 49152}
[0.9967675, 0.00323252]
Non-Fire
```

In our example, the image has a 0.9967675 probability of being class 0, and a 0.00323252 probability of being class 1. Class 0 means fire, and class 1 means non-fire. The example did not detect fire in the image.

4.3 Explaining the example script

The following steps break down what is happening and why in `predict_pyarmnn.py`:

1. Import the PyArmNN module, to define the location of our model, image, and label file. This is shown in the following code:

   ```
   import pyarmnn as ann
   import numpy as np
   import cv2


   print('Working with Arm NN version ' + ann.ARMNN_VERSION)
   print(ann.ARMNN_VERSION)
   ```

2. Load and pre-process an input image. To do this, load the image specified in the following code example, then resize it to the model input dimension. In `predict_pyarmnn.py`, the model

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

accepts 128x128 input images. The input image is wrapped in a const tensor and bound to the input tensor. This is shown in the following code:

```
parser = argparse.ArgumentParser(
      formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument(
      '--image', help='File path of image file', required=True)
args = parser.parse_args()

# Load an image.
image = cv2.imread(args.image)
image = cv2.resize(image, (128, 128))
image = np.array(image, dtype=np.float32) / 255.0
print(image.size)
```

The model is a floating-point model. Therefore, we must scale the input image values to a range of -1 to 1.

3. Create a parser object that will be used to load the network file. Arm NN has parsers for various model file types, including TFLite, ONNX, and Caffe. Parsers create the underlying Arm NN graph, so that you do not need to construct your model graph by hand.

   The following code creates a TfLite parser to load our TensorFlow Lite model from the specified path:

```
parser = ann.ITfLiteParser()
network = parser.CreateNetworkFromBinaryFile('./fire_detection.tflite')
```

4. Use the parser to extract the input information for the network.

   You can extract all the input names by calling `GetSubgraphInputTensorNames()` and use those input names to get the input binding information. For this example, the model only has one input layer. This means that you use `input_names[0]` to obtain the input tensor, then use this string to retrieve the input binding info.

   The input binding information is a tuple consisting of integer identifiers for the following:

   - Bindable layers, for example, inputs, and outputs
   - The tensor information, for example, data type, quantization information, number of dimensions, and total number of elements.

   The following code extracts the input binding information:

```
graph_id = 0
input_names = parser.GetSubgraphInputTensorNames(graph_id)
input_binding_info = parser.GetNetworkInputBindingInfo(graph_id, input_names[0])
input_tensor_id = input_binding_info[0]
input_tensor_info = input_binding_info[1]
print('tensor id: ' + str(input_tensor_id))
print('tensor info: ' + str(input_tensor_info))
```

5. Specify the backend list so that you can optimize the network. This is shown in the following code:

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
options = ann.CreationOptions()
runtime = ann.IRuntime(options)


# Backend choices earlier in the list have higher preference.
preferredBackends = [ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]
opt_network, messages = ann.Optimize(network, preferredBackends,
runtime.GetDeviceSpec(), ann.OptimizerOptions())
```

If your device has an Arm CPU and a Mali GPU, you could define the backend list as follows:

```
preferredBackends = [ann.BackendId('CpuAcc'), ann.BackendId('GpuAcc'),
ann.BackendId('CpuRef')]
```

6. Load the optimized network in the run-time context. `LoadNetwork()` creates the backend-specific workloads for the layers. This is shown in the following example:

```
# Load the optimized network into the runtime.
net_id, _ = runtime.LoadNetwork(opt_network)
print("Loaded network, id={net_id}")
input_tensors = ann.make_input_tensors([input_binding_info], [image])
```

7. Get the output binding information and make the output tensor. This is done in a similar way to the input binding information. We can use the parser to retrieve the output tensor names and get the binding information.

   The following code assumes that an image classification model has only one output. Therefore, the code only uses the first name from the list returned. You could easily extend this code to process multiple outputs by iterating over the `output_names array`:

```
# Get output binding information for an output layer by using the layer name.
output_names = parser.GetSubgraphOutputTensorNames(graph_id)
output_binding_info = parser.GetNetworkOutputBindingInfo(0, output_names[0])
output_tensors = ann.make_output_tensors([output_binding_info])
```

8. Perform inference. The `EnqueueWorkload()` function of the run-time context executes the inference for the network loaded. This is shown in the following example:

```
runtime.EnqueueWorkload(0, input_tensors, output_tensors)
results = ann.workload_tensors_to_ndarray(output_tensors)
```

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5 Performance comparison: PyArmNN and TensorFlow Lite

In this section of the guide, we compare the performance of PyArmNN and the TensorFlow Lite Python API on a Raspberry Pi.

TensorFlow Lite uses an interpreter to perform inference. The interpreter uses static graph ordering and a custom (less-dynamic) memory allocator. For more information on how to load and run a model with Python API, see the TensorFlow Lite documentation.

For performance comparison, inference was carried out with our fire detection model. In our example, we only run inference once. We can also run the model multiple times and take the average inferencing time.

The `predict_tflite.py` script compares performance. To add `predict_tflite.py` to your Raspberry Pi you can clone it from the GitHub repository, or you can copy the following code into a Python file on your Raspberry Pi.

The following code shows `predict_tflite.py`:

```
import tensorflow as tf
import argparse
import io
import time
import cv2
import numpy as np
from timeit import default_timer as timer

# Load TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path="./fire_detection.tflite")
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
_,height,width,_ = input_details[0]['shape']
floating_model = False
if input_details[0]['dtype'] == np.float32:
    floating_model = True

parser = argparse.ArgumentParser(
      formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument(
```

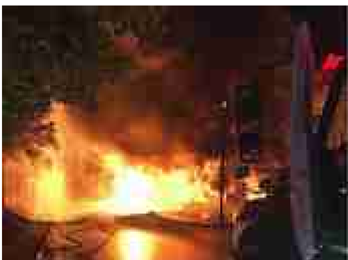
Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
      '--image', help='File path of image file', required=True)
args = parser.parse_args()

image = cv2.imread(args.image)
image = cv2.resize(image, (width, height))
image = np.expand_dims(image, axis=0)
if floating_model:
image = np.array(image, dtype=np.float32) / 255.0

# Test model on image.
interpreter.set_tensor(input_details[0]['index'], image)
start = timer()
interpreter.invoke()
end = timer()
print('Elapsed time is ', (end-start)*1000, 'ms')

# The function `get_tensor()` returns a copy of the tensor data.
# Use `tensor()` in order to get a pointer to the tensor.
output_data = interpreter.get_tensor(output_details[0]['index'])
print(output_data)
j = np.argmax(output_data)
if j == 0:
    print("Non-Fire")
else:
print("Fire")
```

You can run `predict_tflite.py` by using the following command:

```
$ python3 predict_tflite.py --image ./images/746.jpg
```

In the preceding command, the following picture was used:



When you have run `predict_tflite.py` you will get the following output:

```
2020-01-01 11:32:33.609188: E
Elapsed time is  38.02500700112432 ms
[[9.9076563e-04 9.9900925e-01]]
Fire
```

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

We can extend `predict_pyarmmnn.py` with the same code for inference benchmarking. To extend `predict_pyarmmnn.py` you must add the following code before the line `runtime.EnqueueWorkload(0, input_tensors, output_tensors):`

```
start = timer()
```

After the line `runtime.EnqueueWorkload(0, input_tensors, output_tensors)` you must add the following code:

```
end = timer()

print('Elapsed time is ', (end - start) * 1000, 'ms')
```

Therefore, you will have the following code in `predict_tflite.py`:

```
start = timer()

runtime.EnqueueWorkload(0, input_tensors, output_tensors)

end = timer()

print('Elapsed time is ', (end - start) * 1000, 'ms')
```

Run the predict_pyarmnn.py script again:

```
$ python3 predict_pyarmnn.py --image ./images/746.jpg
```

In the preceding code, the following picture was used:

You will get the following output:

```
Working with Arm NN version 21.0.0

(128, 128, 3)


tensor id: 15616,

tensor info: TensorInfo{DataType: 1, IsQuantized: 0, QuantizationScale: 0.000000,
QuantizationOffset: 0, NumDimensions: 4, NumElements: 49152}


Elapsed time is  21.224445023108274 ms

[0.0009907632, 0.99900925]

Fire
```

From the preceding code, you can observe an inference performance enhancement by using the Arm NN Neon optimized computational backend.

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6 Related information

Here are some resources related to the material in this guide:

- Arm architecture and reference manuals
- Arm Community - Ask development questions, and find articles and blogs on specific topics from Arm experts.
- Arm Machine Learning - Intro to Arm NN
- Arm NN SDK
- Arm Software Developer Kit (SDK)
- Configure the Arm NN SDK build environment for Caffe
- Deploying a Caffe MNIST model using the Arm NN SDK
- Neon
- Project Trillium Framework

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7 Next steps

This guide shows you how to train a neural network that can recognize fire in images using PyArmNN.

You also can use this guide as a starting point to handle other types of neural networks with PyArmNN.

Arm Developer provides How-to guides to help you get started with Machine Learning on Arm-based devices.

Copyright © 2020 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential