Writing Neural Network Code: Introduction to TensorFlow, Hands-On

Class 5: TensorFlow Hands-On Part 3: Teaching and Testing and Conclusion

May 15, 2020

Charles J. Lord, PE
President, Consultant, Trainer
Blue Ridge Advanced Design and Automation

Presented by:

This Week's Agenda

5/11 A Brief History of Artificial Neural Networks

5/12 Neural Network Simulation and Programming

5/13 TensorFlow Hands-On Part 1: Hello World!

5/14 TensorFlow Hands-On Part 2: Defining and Building Your Network

5/15 TensorFlow Hands-On Part 3: Teaching and Testing and Conclusion

This Week's Agenda

5/11 A Brief History of Artificial Neural Networks

5/12 Neural Network Simulation and Programming

5/13 TensorFlow Hands-On Part 1: Hello World!

5/14 TensorFlow Hands-On Part 2: Defining and Building Your Network

5/15 TensorFlow Hands-On Part 3: Teaching and Testing and Conclusion

We're Almost There!

- Yesterday, we defined our neural network as a 784|128|10 model with an added dropout 'layer' while training.
- We need to actually build (compile) this network so that we can feed the parameters, run the training data through, and check for the differences
- For this last step we use an *optimizer*

Keras Optimizers

https://keras.io/api/optimizers/

- SGD – gradient descent
- RMSprop – Moving average RMS
- ADAM – Adaptive Descent Method
- Adamax – ADAM on steroids
- ADAM with Nesterov momentum
- Adagrad – parameters that get changed more change less
- Adadelta – adaptive learning rate per dimension – more robust version of adagrad
- Ftrl – algorithm that came from ad click-through optimization

Question 1 – What does RMS stand for?

Manual Optimization

```
# Instantiate an optimizer.
optimizer = tf.keras.optimizers.Adam()

# Iterate over the batches of a dataset.
for x, y in dataset:
    # Open a GradientTape.
    with tf.GradientTape() as tape:
        # Forward pass.
        logits = model(x)
        # Loss value for this batch.
        loss_value = loss_fn(y, logits)

    # Get gradients of loss wrt the weights.
    gradients = tape.gradient(loss_value, model.trainable_weights)

    # Update the weights of the model.
    optimizer.apply_gradients(zip(gradients, model.trainable_weights))
```

Keras to the Rescue

- We don’t have to be concerned with the internals when we use the high-level controls built into TensorFlow 2 (unless we want or need the extra control)
- When we compile a network, we define the optimizer we want to implement and use the model.fit method to go and do the actual training.
- But first, we need to determine the loss model:

class BinaryCrossentropy: Computes the cross-entropy loss between true labels and predicted labels.

class CategoricalCrossentropy: Computes the crossentropy loss between the labels and predictions.

class CategoricalHinge: Computes the categorical hinge loss between y_true and y_pred.

class CosineSimilarity: Computes the cosine similarity between y_true and y_pred.

class Hinge: Computes the hinge loss between y_true and y_pred.

class Huber: Computes the Huber loss between y_true and y_pred.

class KLDivergence: Computes Kullback-Leibler divergence loss between y_true and y_pred.

class LogCosh: Computes the logarithm of the hyperbolic cosine of the prediction error.

class Loss: Loss base class.

class MeanAbsoluteError: Computes the mean of absolute difference between labels and predictions.

class MeanAbsolutePercentageError: Computes the mean absolute percentage error between y_true and y_pred.

class MeanSquaredError: Computes the mean of squares of errors between labels and predictions.

class MeanSquaredLogarithmicError: Computes the mean squared logarithmic error between y_true and y_pred.

class Poisson: Computes the Poisson loss between y_true and y_pred.

class Reduction: Types of loss reduction.

class SparseCategoricalCrossentropy: Computes the crossentropy loss between the labels and predictions.

class SquaredHinge: Computes the squared hinge loss between y_true and y_pred.

tf.keras.losses.xx

Our Code from Yesterday

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
predictions = model(x_train[:1]).numpy()
print(predictions)
print(tf.nn.softmax(predictions).numpy())
```

Our New Code

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

loss_fn(y_train[:1], predictions).numpy()

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test,  y_test, verbose=2)
```

MNIST01 – MNIST01.py

File Edit View Navigate Code Refactor Run Tools VCS Window Help

MNIST01 〉 MNIST01.py

MNIST01.py scratch.py mnist02.py

```python
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])

predictions = model(x_train[:1]).numpy()
print(predictions)

print(tf.nn.softmax(predictions).numpy())

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

loss_fn(y_train[:1], predictions).numpy()

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test,  y_test, verbose=2)
```

Run: MNIST01

```
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0729 - accuracy: 0.9771
313/313 - 0s - loss: 0.0795 - accuracy: 0.9756

Process finished with exit code 0
```

6: TODO 4: Run Terminal Python Console Event Log

22:42 CRLF UTF-8 4 spaces Python 3.8 (venv)

Output

```
MNIST01
C:\Users\Charles\venv\Scripts\python.exe C:/Users/Charles/PycharmProjects/MNIST01/MNIST01.py
2020-05-13 17:31:28.201987: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0xe7d52c0 initialized fo
2020-05-13 17:31:28.202987: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, D
[[ 0.19646935  0.3767267   0.12971145 -0.33407652 -0.2645662  -0.20949584
   0.44157678 -0.83076787  0.33970034 -0.13830277]]
[[0.11732346 0.14049783 0.1097469  0.06901949 0.07398772 0.07817654
  0.14991106 0.04200118 0.13539085 0.08394507]]
Epoch 1/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2921 - accuracy: 0.9161
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1440 - accuracy: 0.9571
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1073 - accuracy: 0.9671
Epoch 4/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0874 - accuracy: 0.9730
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0759 - accuracy: 0.9759
313/313 - 0s - loss: 0.0712 - accuracy: 0.9785

Process finished with exit code 0
```

97.85% Accurate

Question 2 – What was our approximate accuracy after one pass?

Hello World!

- We have our working network!
- But can we do better?
  - What if we have 200 hidden nodes?
  - What if we use linear instead of ReLu?
  - What if we optimize using SGD?
  - What if we use CategoricalCrossentropy for loss?
- This is where the fun starts!

Plugging these changes in…

- Original: 97.85%
- 200 hidden nodes: 98.11%
- linear activation function: 91.85%
- optimize using SGD? 92.42%
- CategoricalCrossentropy for loss: 97.74%

- The only change that made any sort of positive difference was almost doubling the hidden nodes – at twice the computational time!

CPU versus GPU

- A simple test can be done in Colab to compare typical performance between a CPU and GPU
- You can perform the same test if you have both processors in your development or target system – but you usually don't

  https://colab.research.google.com/notebooks/gpu.ipynb

```
Time (s) to convolve 32x7x7x3 filter over random 100x100x100x3 images (batch x height x width x channel). Sum of ten runs.
CPU (s):
3.41443132400002
GPU (s):
0.052692100000001574
GPU speedup over CPU: 64x
```

TPU

- Tensor Processing Unit
- ASIC developed by Google
- Optimized for TensorFlow
- Colab has a TPU available but no CPU vs GPU vs TPU benchmarks yet

https://colab.research.google.com/notebooks/tpu.ipynb

Google Coral TPU (tensor processing unit)

MediaTek 8167s SoC (Quad-core Arm Cortex-A35)

IMG PowerVR GE8300 (integrated in SoC)

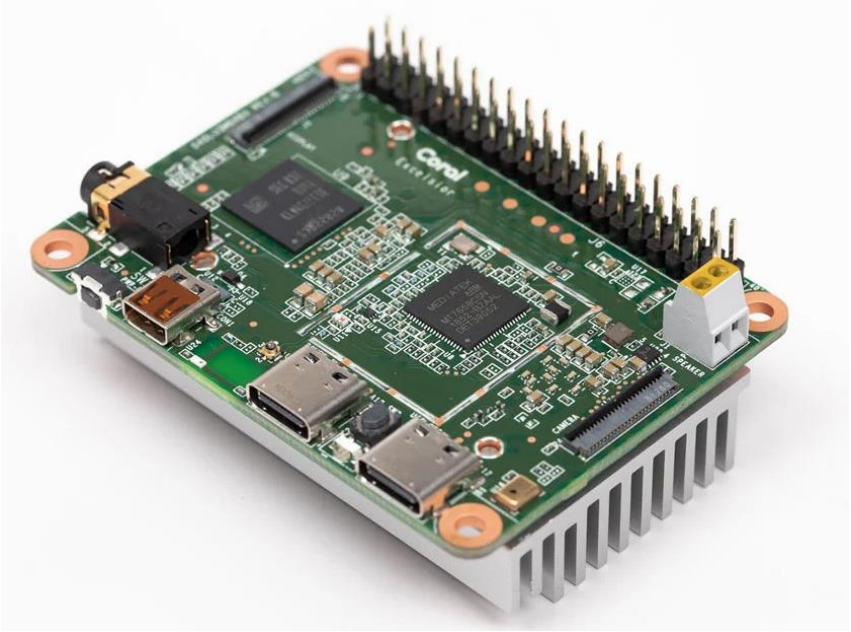
Google Edge TPU coprocessor

2 GB DDR3L

8 GB eMMC

Typical RPi-type I/O

Announced at CES2020, still “coming soon”



NVIDIA Jetson Nano

GPU 128-core Maxwell

CPU Quad-core ARM A57 @ 1.43 GHz

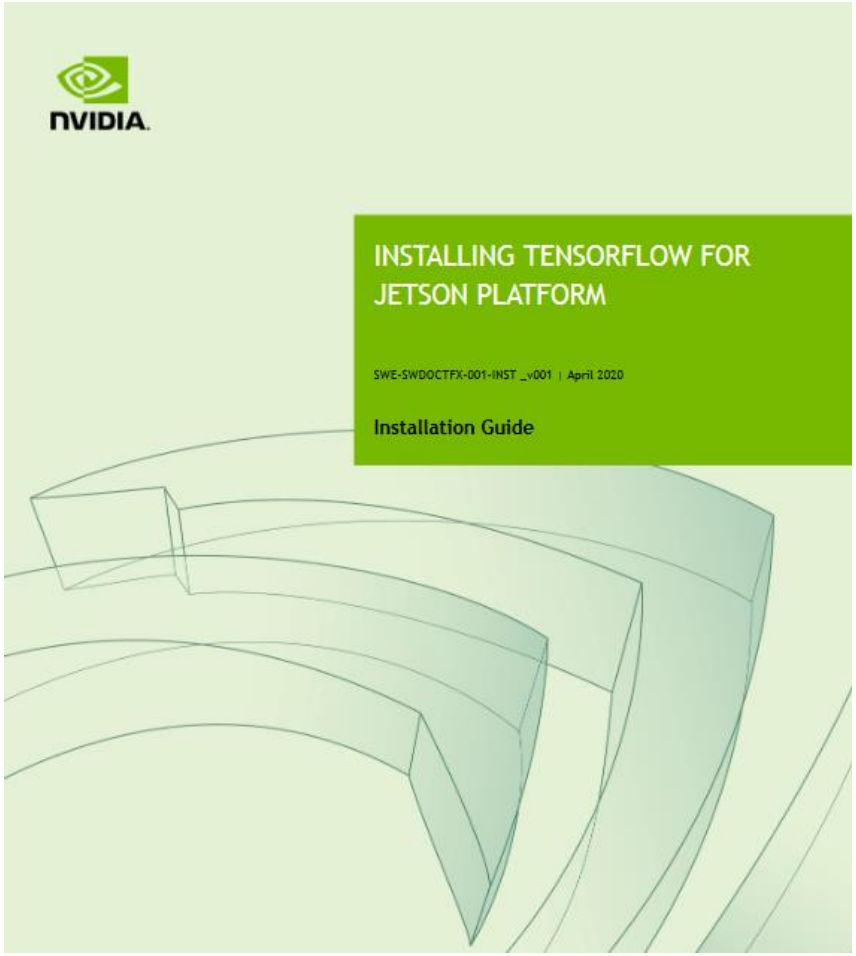
Memory 4 GB 64-bit LPDDR4 25.6 GB/s

Storage microSD (not included)

$120

https://www.digikey.com/product-detail/en/seeed-technology-co-ltd/102110417/1597-102110417-ND

https://docs.nvidia.com/deeplearning/frameworks/pdf/Install-TensorFlow-Jetson-Platform.pdf



- Latest Version
- April 2020
- Installs in the linux distribution for the Nano
- NVIDIA also has open source TensorRT that is optimized for the Jetson architecture

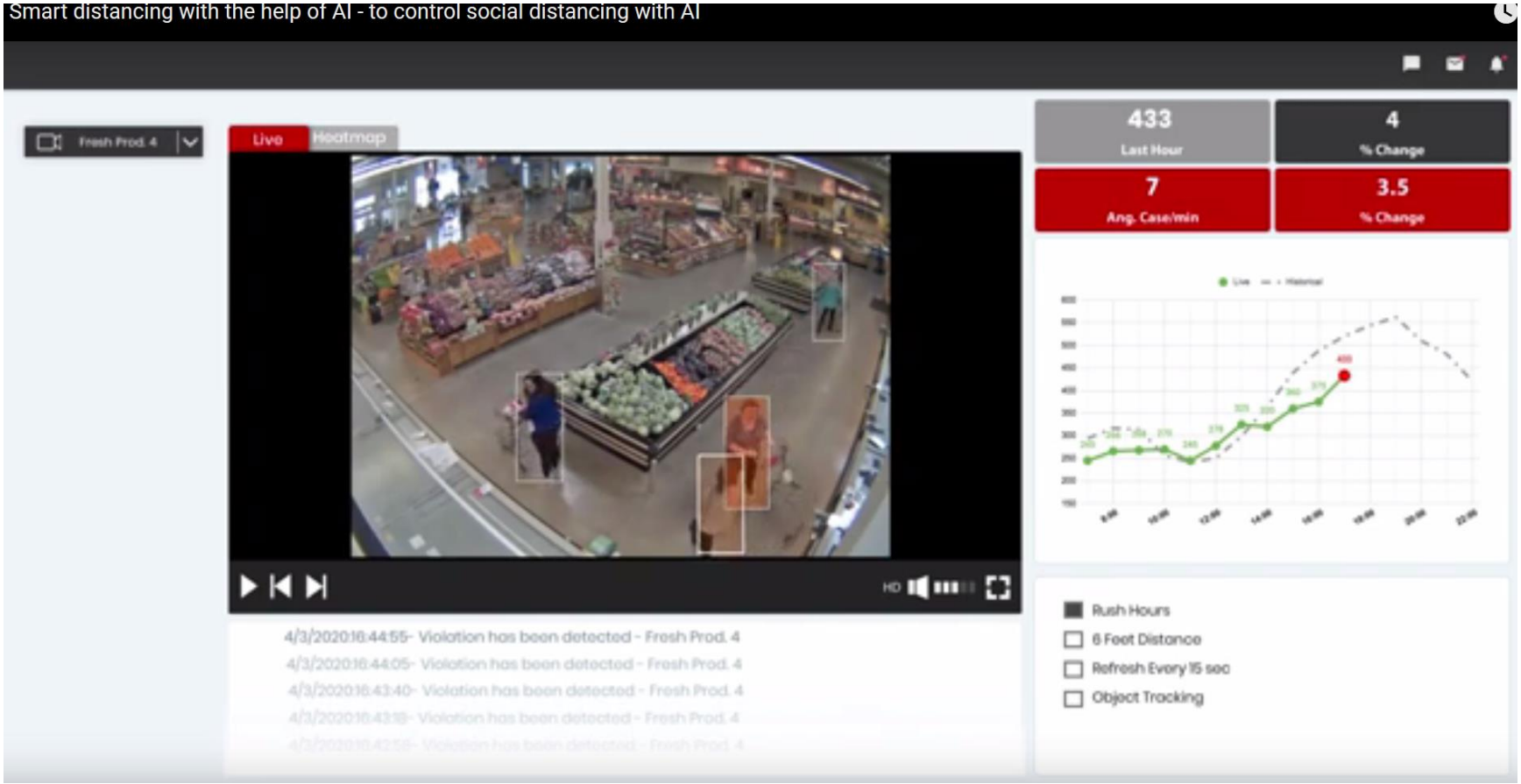
TensorFlow on the Nano

https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html

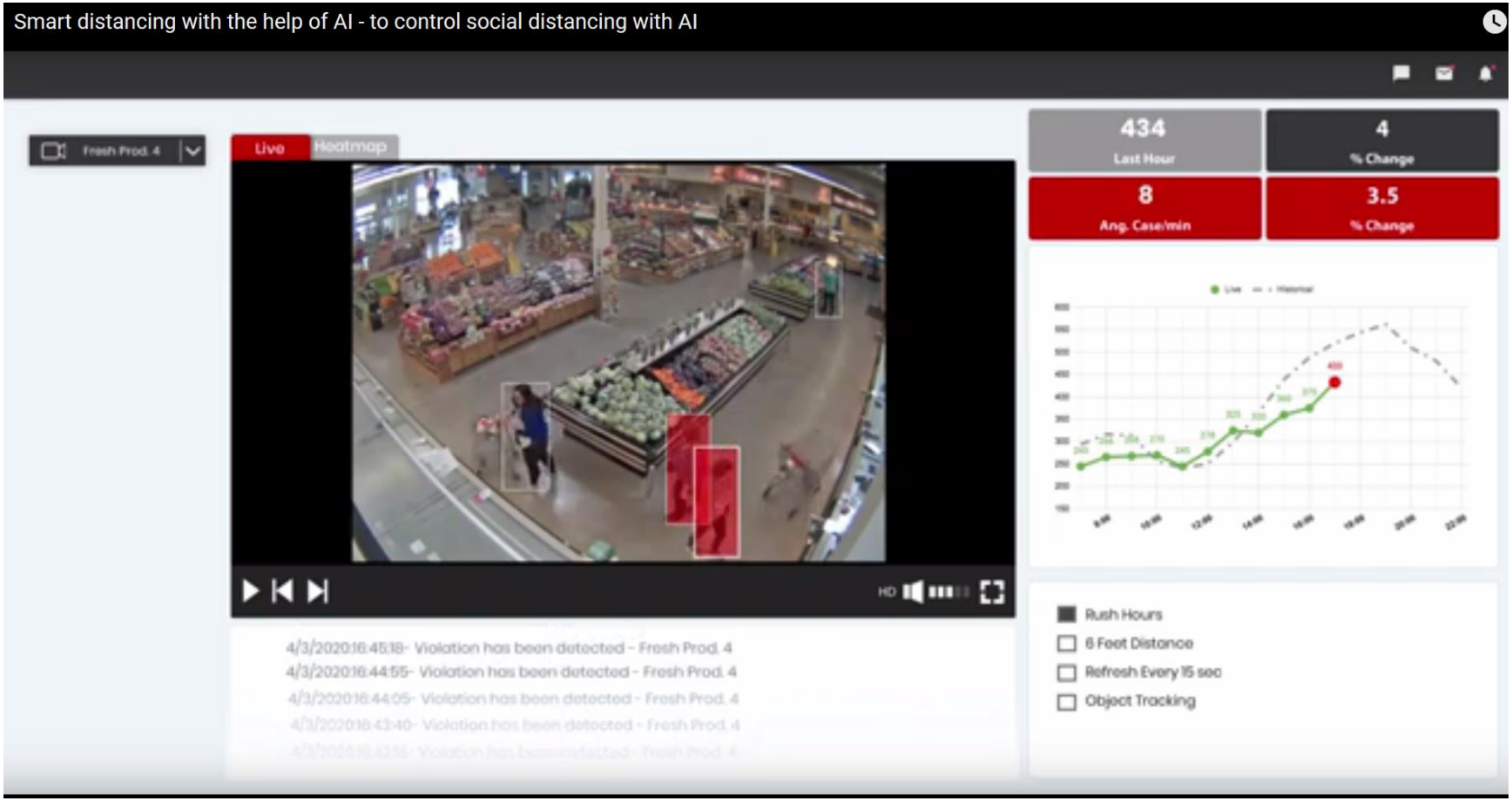
https://developer.nvidia.com/embedded/community/jetson-projects

Question 3 – Any experience with the Jetson Nano (or other GPU board)?

Solution of the Month (timely!)



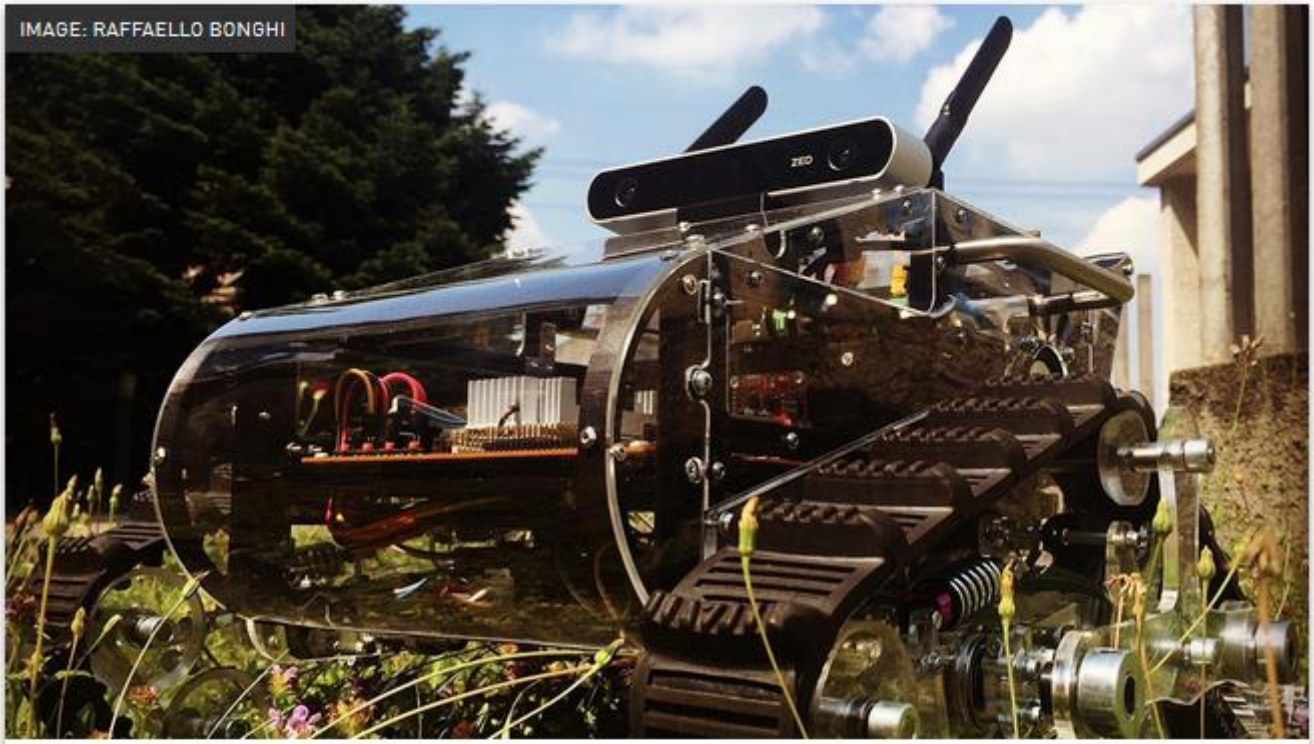
Don’t get so close to me….



P.A.N.T.H.E.R.: Powerful Autonomous eNTity High-End Robot

Share

By Raffaello Bonghi



Jetson TX2

"Using its two tracks, ZED stereo camera and the NVIDIA Jetson TX2, this robot explores the outdoors and interacts with its surroundings. Weighing 9kg (20lbs), with 7cm (2.7in) of ground clearance, and a track system composed of three different dampers to absorb vibrations when drifting on grass, P.A.N.T.H.E.R. can climb little rocks and bumps. P.A.N.T.H.E.R. is built with plexiglass, aluminium, plastic, and other materials, is integrated with ROS, and all code is available on GitHub."

Code on GitHub | Demo Video | Link to Page

Autonomous Tank

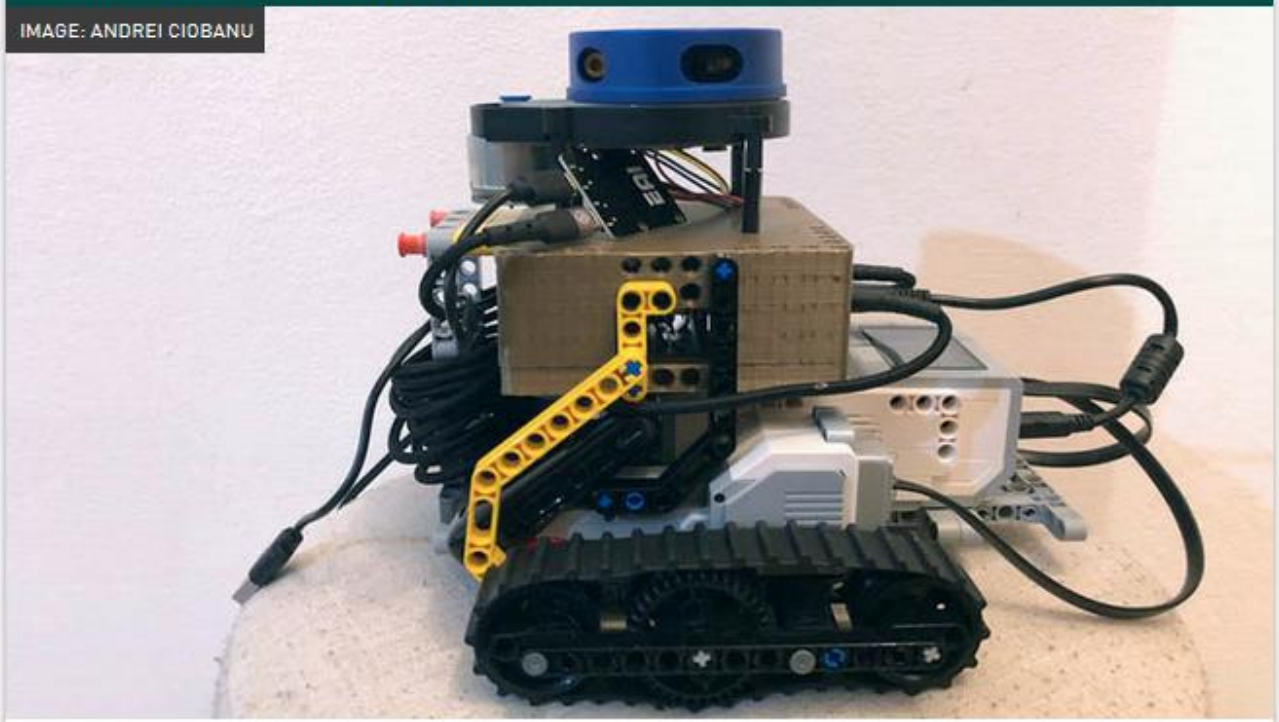
Share

By Andrei Ciobanu

NVIDIA / HACKSTER AI AT THE EDGE CHALLENGE 3RD PLACE

CATEGORY: AUTONOMOUS MACHINES & ROBOTICS

IMAGE: ANDREI CIOBANU



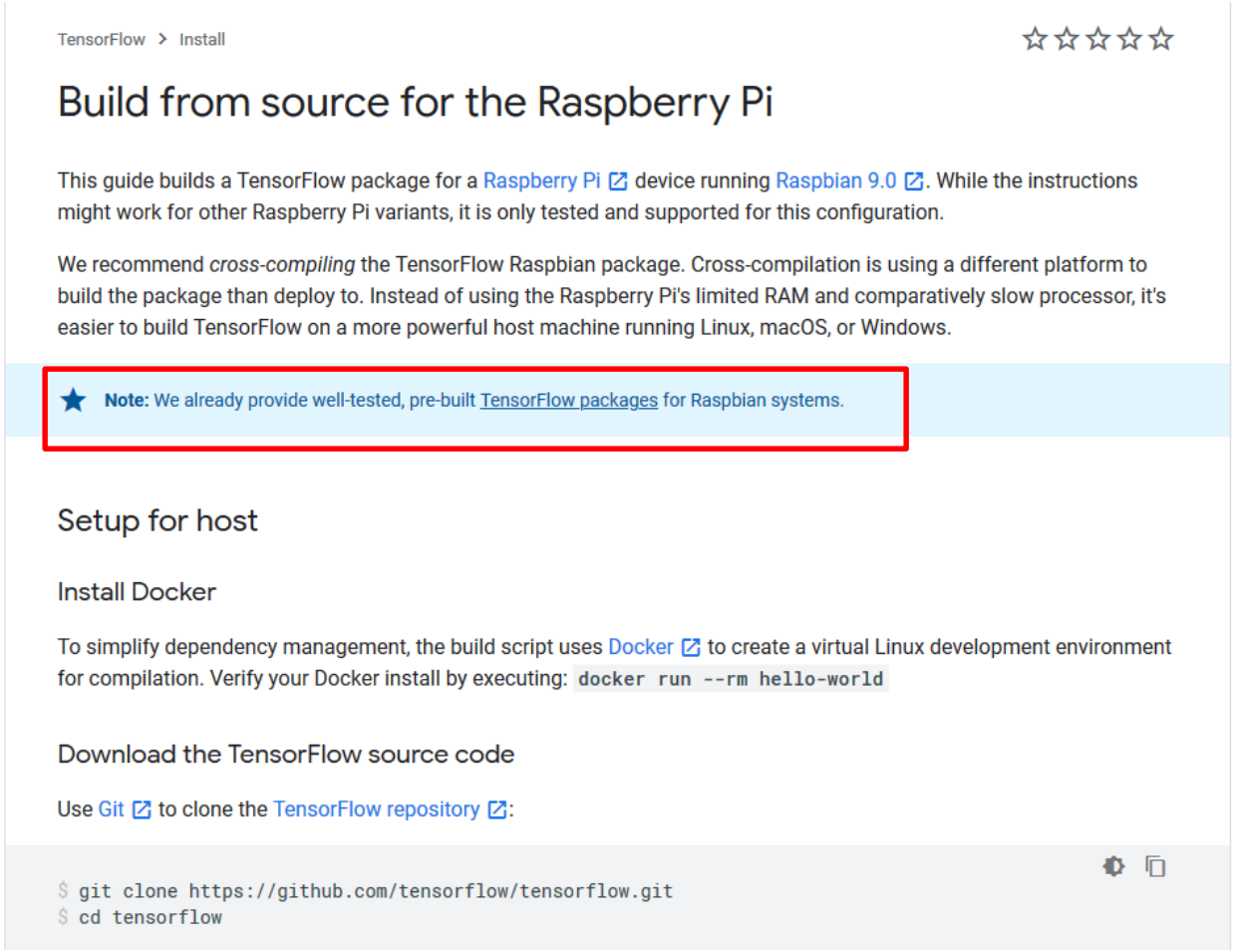
Jetson Nano

" Tracked vehicle made with Lego Technic parts and motors, enhanced with LiDAR and controlled by a Jetson Nano board running the latest Isaac SDK. Issue voice commands and get the robot to move autonomously. Create missions: navigate [and] set where the tank should go. If [the camera] detects the target object, it will get closer and shoot it with... the camera. It'll just take a picture, no real weapons :)"

Code on GitHub | Demo Video | Learn More

Tensorflow on RPi

https://www.tensorflow.org/install/source_rpi



TensorFlow > Install

Build from source for the Raspberry Pi

This guide builds a TensorFlow package for a Raspberry Pi device running Raspbian 9.0. While the instructions might work for other Raspberry Pi variants, it is only tested and supported for this configuration.

We recommend *cross-compiling* the TensorFlow Raspbian package. Cross-compilation is using a different platform to build the package than deploy to. Instead of using the Raspberry Pi's limited RAM and comparatively slow processor, it's easier to build TensorFlow on a more powerful host machine running Linux, macOS, or Windows.

Note: We already provide well-tested, pre-built TensorFlow packages for Raspbian systems.

Setup for host

Install Docker

To simplify dependency management, the build script uses Docker to create a virtual Linux development environment for compilation. Verify your Docker install by executing: `docker run --rm hello-world`

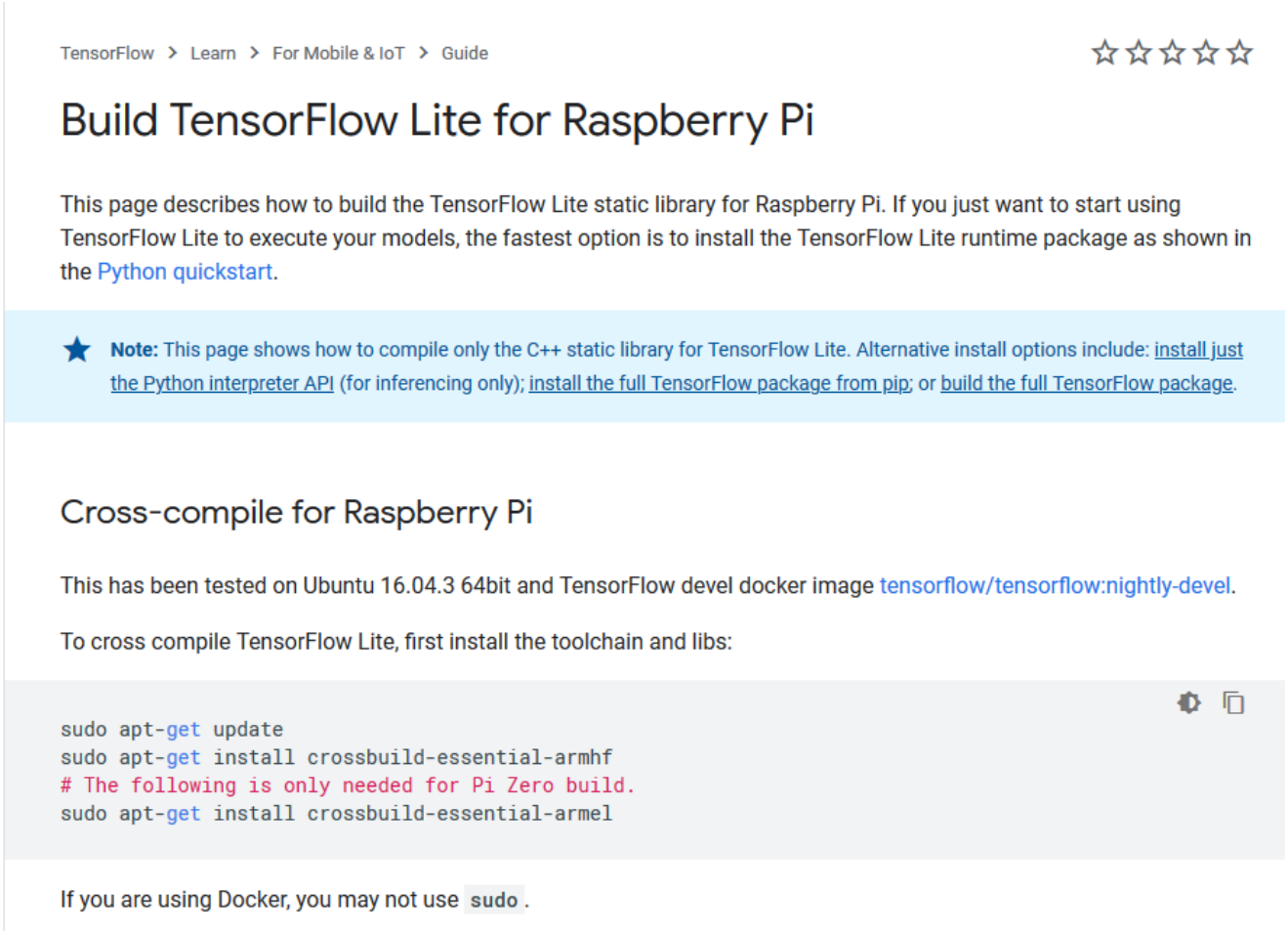
Download the TensorFlow source code

Use Git to clone the TensorFlow repository:

```
$ git clone https://github.com/tensorflow/tensorflow.git
$ cd tensorflow
```

TensorFlow Lite on RPi

https://www.tensorflow.org/lite/guide/build_rpi



TensorFlow > Learn > For Mobile & IoT > Guide

Build TensorFlow Lite for Raspberry Pi

This page describes how to build the TensorFlow Lite static library for Raspberry Pi. If you just want to start using TensorFlow Lite to execute your models, the fastest option is to install the TensorFlow Lite runtime package as shown in the Python quickstart.

Note: This page shows how to compile only the C++ static library for TensorFlow Lite. Alternative install options include: install just the Python interpreter API (for inferencing only); install the full TensorFlow package from pip; or build the full TensorFlow package.

Cross-compile for Raspberry Pi

This has been tested on Ubuntu 16.04.3 64bit and TensorFlow devel docker image tensorflow/tensorflow:nightly-devel.

To cross compile TensorFlow Lite, first install the toolchain and libs:

```
sudo apt-get update
sudo apt-get install crossbuild-essential-armhf
# The following is only needed for Pi Zero build.
sudo apt-get install crossbuild-essential-armel
```

If you are using Docker, you may not use `sudo`.

Conclusion

- We have just glanced at the tip of the iceberg
- This is not an exact science (actually it is but the math is beyond where we typically talk)
- Trial, error, experimentation is the way to become proficient – more than 'book learning'
- You now have the tools to go make trouble – go forth and convolute! (break some eggs!)

This Week's Agenda

5/11 A Brief History of Artificial Neural Networks

5/12 Neural Network Simulation and Programming

5/13 TensorFlow Hands-On Part 1: Hello World!

5/14 TensorFlow Hands-On Part 2: Defining and Building Your Network

5/15 TensorFlow Hands-On Part 3: Teaching and Testing and Conclusion

Coming Up Next!

- Our next course titled “Building Machine Vision Applications using OpenMV” will take place June 8-12 with Jacob Beningo.
- For more information and to sign up for the course, go to the curriculum calendar.

Question 4 – What future classes would you like to see?

Please stick around as I answer your questions!

- Please give me a moment to scroll back through the chat window to find your questions
- I will stay on chat as long as it takes to answer!
- I am available to answer simple questions or to consult (or offer in-house training for your company)
  c.j.lord@ieee.org
  http://www.blueridgetechnc.com
  http://www.linkedin.com/in/charleslord
  Twitter: @charleslord
  https://www.github.com/bradatraining