OPEN TECH THOUGHTS

AV DATASETS & MULTIGPU TRAINING

Dr. Adolf Hohl

4.7.2018

NVIDIA GPU TECHNOLOGY CONFERENCE

OCTOBER 9 - 11, 2018 … in MUNICH!!!

NVIDIA's GPU Technology Conference (GTC) Europe is part of the largest global series of events focused on artificial intelligence and its applications across many important fields.

Join us in Munich and discover the latest breakthroughs in autonomous vehicles, high performance computing, smart cities, healthcare, big data, virtual reality and more.

https://www.nvidia.com/en-eu/gtc/

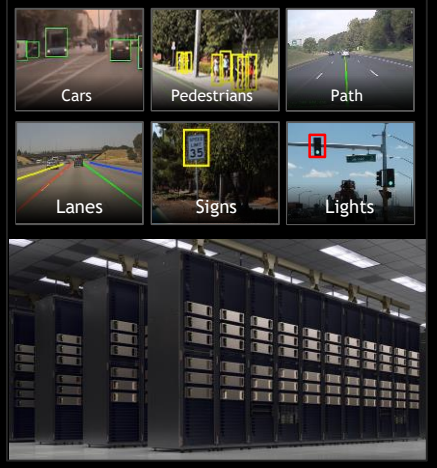
Enjoy Early Bird Pricing until 10th July and use following code for 25% discount:

NVAHOHL

END-TO-END SYSTEM FOR AV

COLLECT DATA

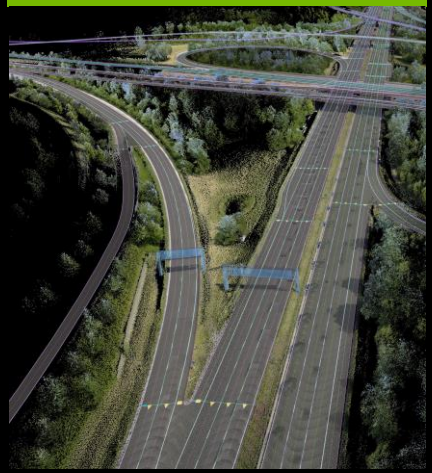
TRAIN MODELS



SIMULATE

RE-SIMULATE

MAPPING



Sources: NVIDIA, RAND Corporation

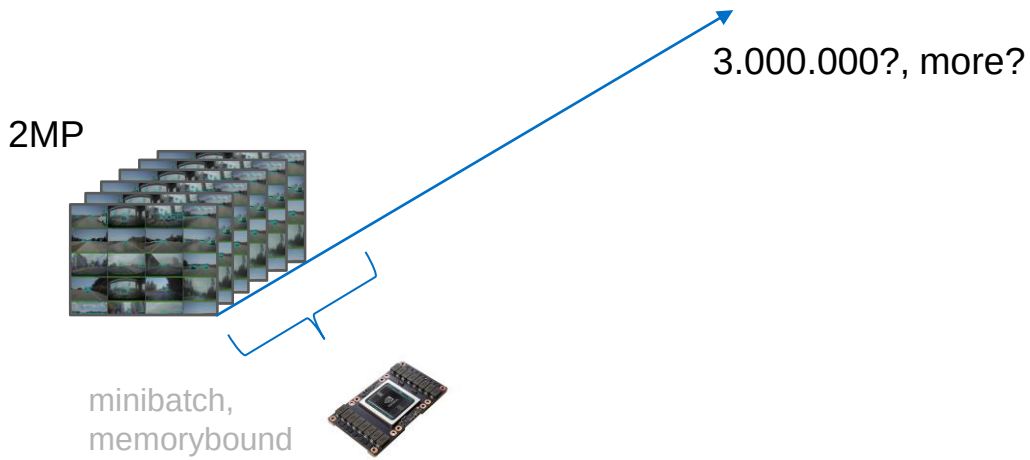
NVIDIA

RISING DATASETS AND DEMANDS

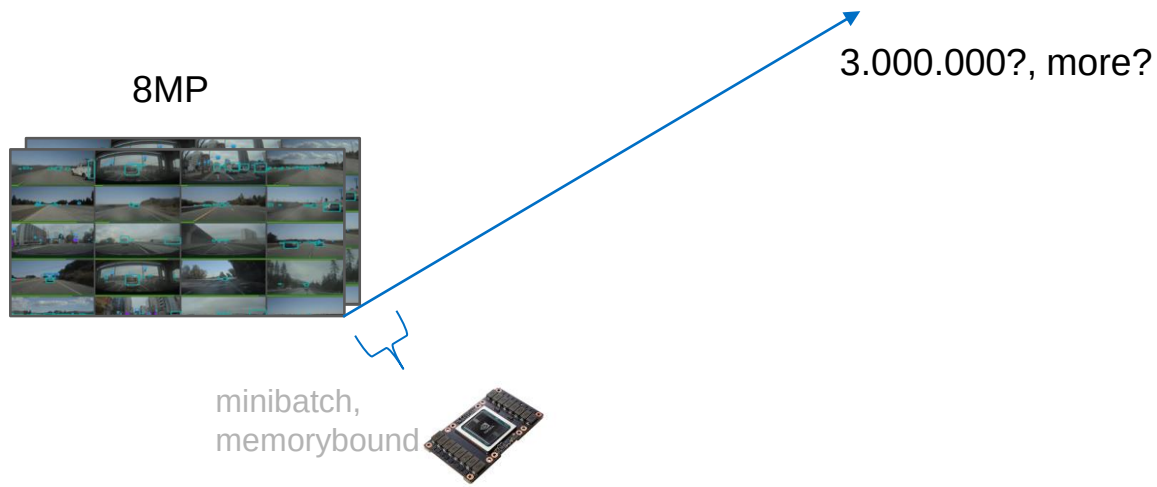
Estimate (moving):

1. Amont of training data
2. Training data size

Minibatching - memorybound



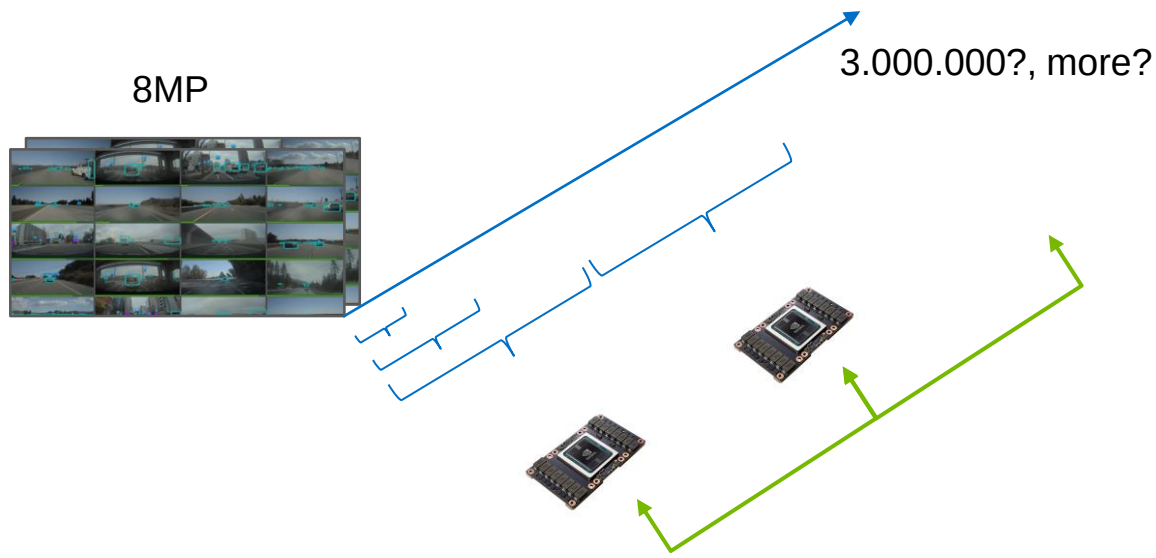
RISING DATASETS AND DEMANDS



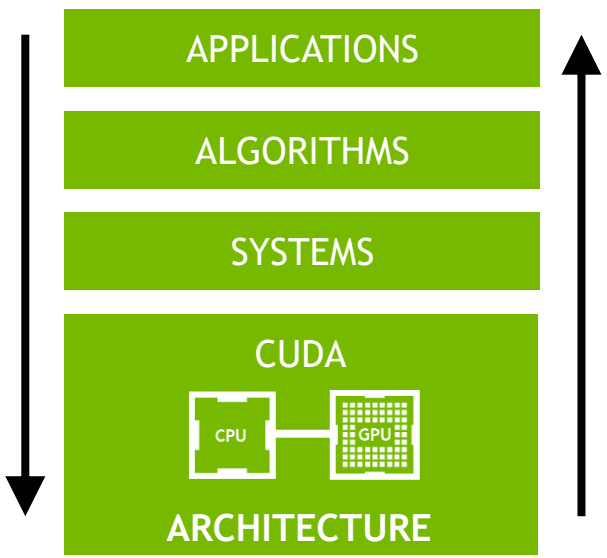
RISING DATASETS AND DEMANDS

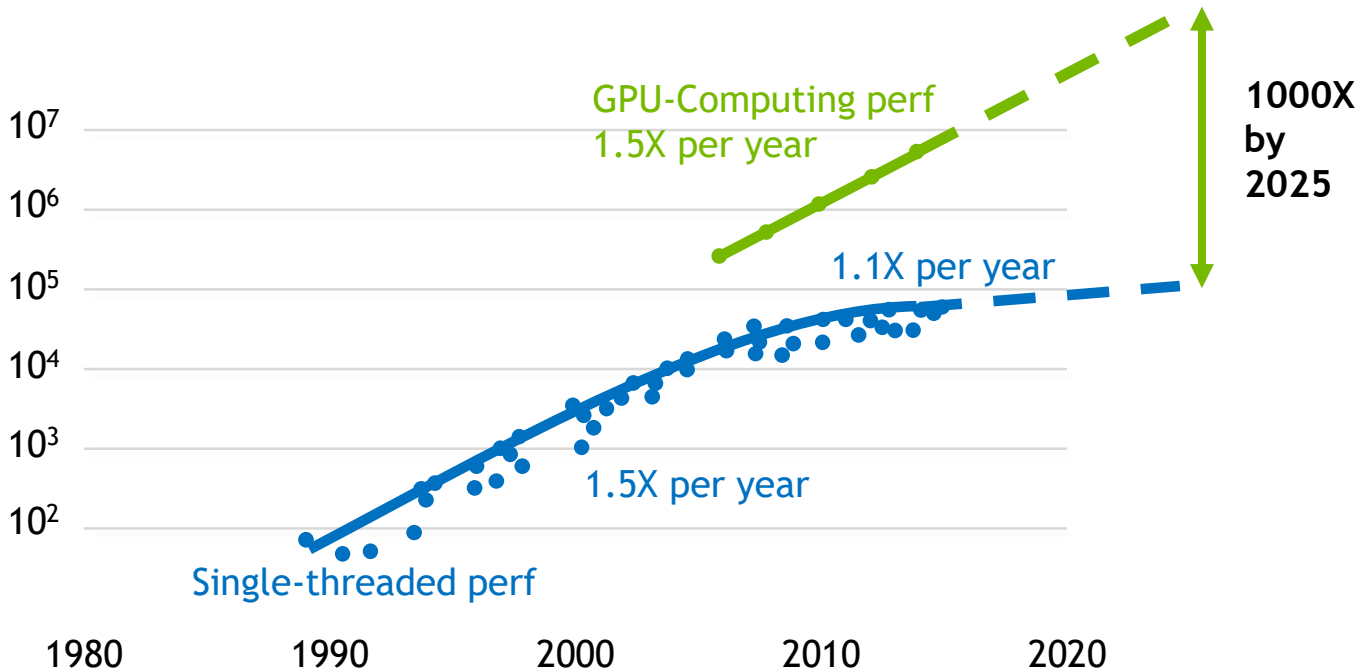
NVIDIA Innovations

1. 32 GB Memory
2. TensorCores & Mixed Precision
3. MultiGPU
4. NVLINK/NVSWITCH



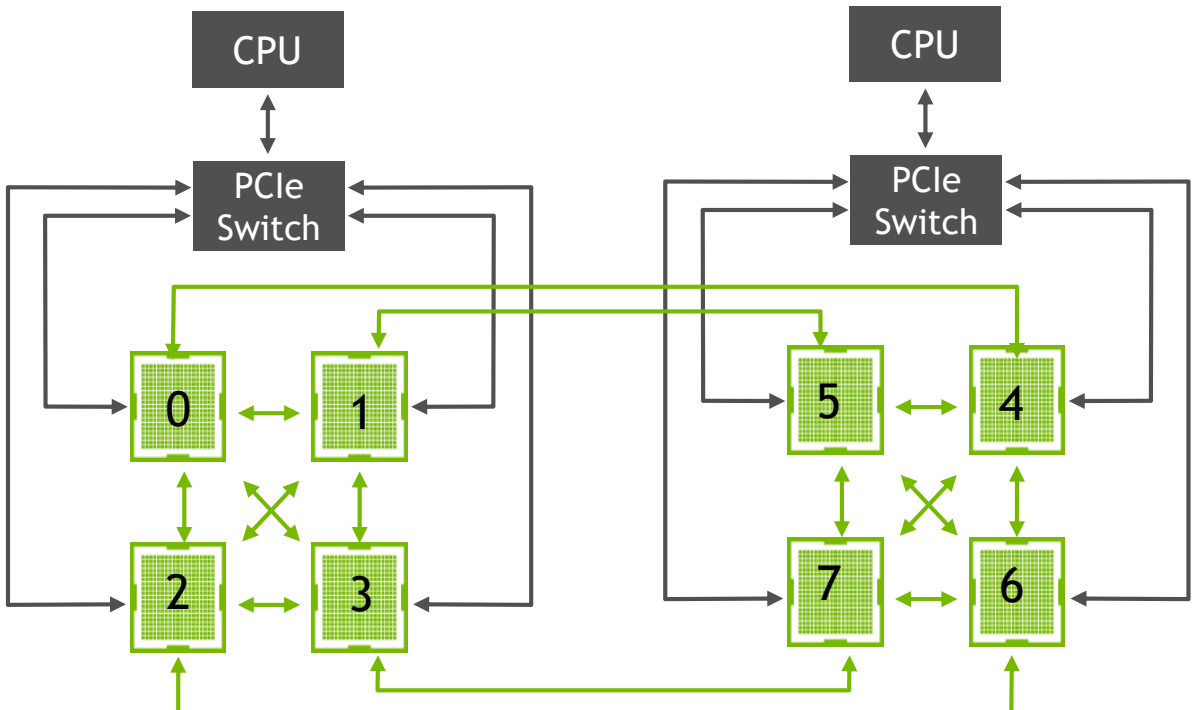
GPU COMPUTING TO THE RESCUE



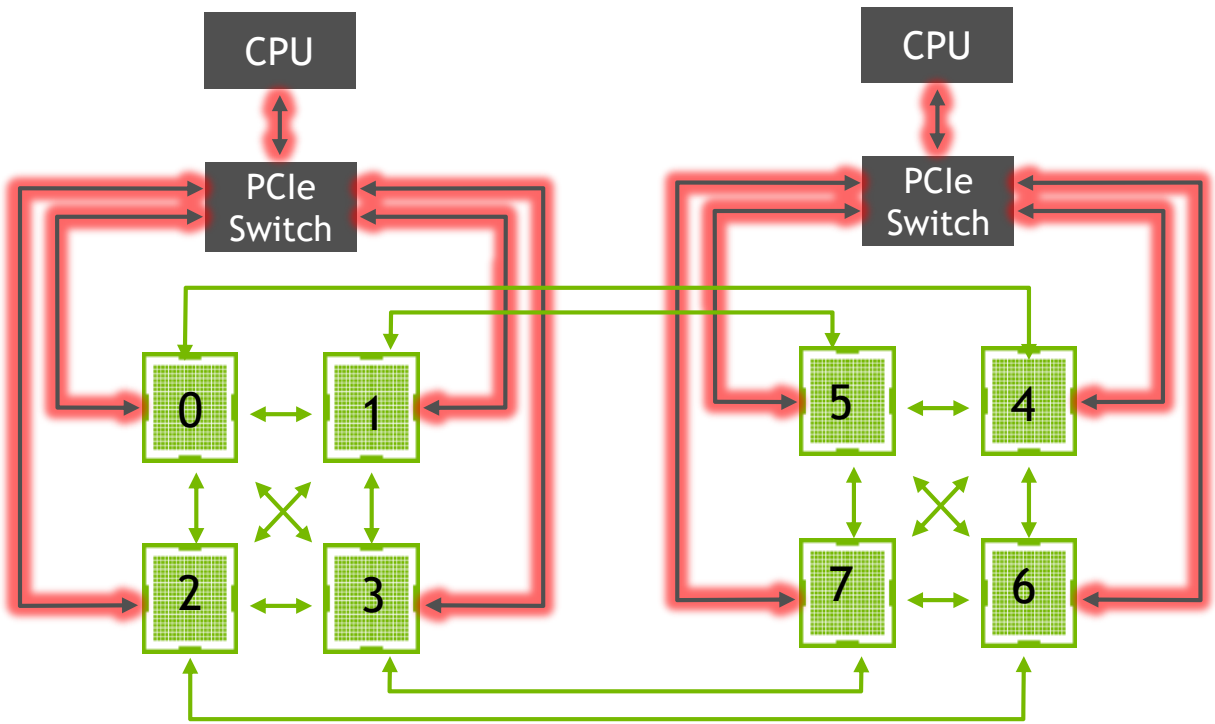


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

DATA PARALLELISM - NVLINK A KEY TO SCALE

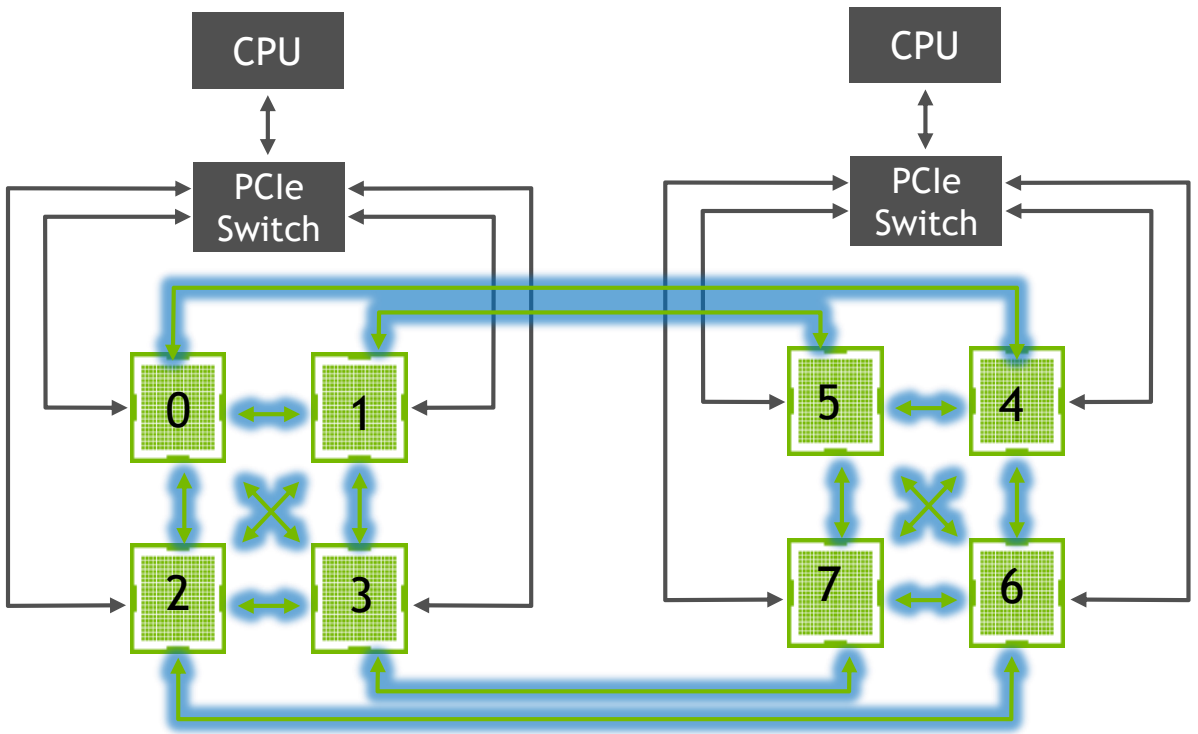


DATA PARALLELISM - NVLINK



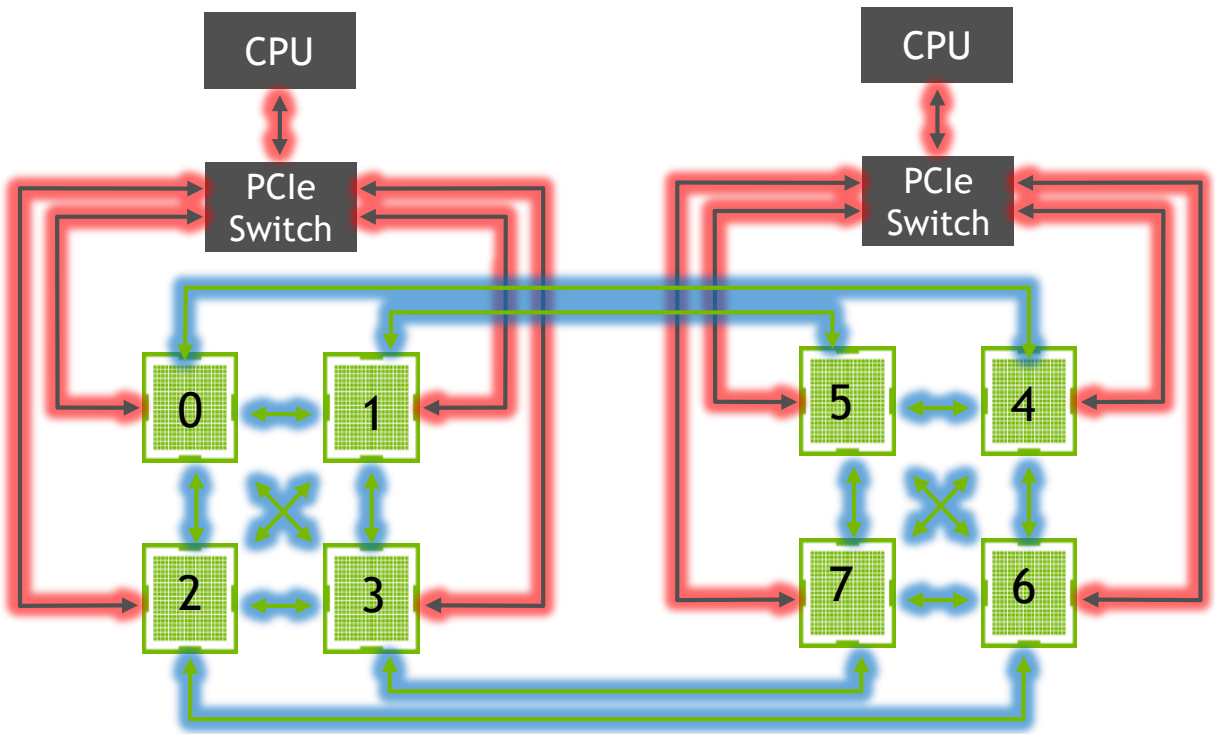
Data loading over PCIe

DATA PARALLELISM - NVLINK



Gradient averaging over NVLink

SPEEDUP TX TO NCCL AND NVLINK



Effective use of communication resources: No congestion

MULTIGPU TRAINING /W TF/KERAS

Recipe 1 – „Press the Sport Button“

1. Use the NVIDIA Container (Horovod, NCCL and MPI is installed)
2. Keep your model
3. Adapt the training procedure to make use of Horovod
    1. import horovod.tensorflow as hvd
    2. hvd.init()
    3. Per tower/global optimizer
    4. get_or_create_global_step()
    5. global_optimizer.minimize

```
import DenseNet as dn
slim = tf.contrib.slim

tf.logging.set_verbosity(tf.logging.INFO)

growth_k = 12
nb_block = 2 # how many (dense block + Transition Layer) ?
class_num = 2

IMAGE_HEIGHT, IMAGE_WIDTH = 223,223
class_directories= ["mouse-click-events/pos", "mouse-click-events/neg"]

def grab_dir(path):

def Xy_shuffle_loader(pos_dir, neg_dir):

def main(_):

    hvd.init()

    X_train, y_train, X_validation, y_validation = Xy_shuffle_loader(class_directories[0], class_directories[1])

    graph = tf.Graph()

    with graph.as_default():

        tf.train.create_global_step()

        images = tf.placeholder(tf.float32, shape=[None, IMAGE_HEIGHT, IMAGE_WIDTH, 3], name='input_image')
        labels = tf.placeholder(tf.float32, shape=[None, class_num], name='training_labels')
        training_flag = tf.placeholder(tf.bool, name='training_flag')

        logits = dn.DenseNet(x=images, nb_blocks=nb_block, filters=growth_k, training=training_flag).model
        logits = tf.identity(logits, name='output_logits')

        loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))

        tower_optimizer = tf.train.RMSPropOptimizer(0.001 * hvd.size())
        global_optimizer = hvd.DistributedOptimizer(tower_optimizer)

        global_step = tf.train.get_or_create_global_step()

        train_op = global_optimizer.minimize(loss=loss, global_step=global_step)
```

MULTIGPU TRAINING /W TF/KERAS

Recipe 1 – „Press the Sport Button“

1. sync_hook to create stopping criteria and some logging
2. Some config (tf.ConfigProto())
3. Training session
    1. Use the Horovod tailored session
    2. Check if there is still data in the pipeline
4. Train

→ Your new bottleneck is likely the data pipeline now!

```
        train_op = global_optimizer.minimize(loss=loss, global_step=global_step)
        train_prediction = tf.nn.softmax(logits=logits)

        sync_hook = [
            hvd.BroadcastGlobalVariablesHook(0),
            tf.train.StopAtStepHook(last_step=20000 // hvd.size()),
            tf.train.LoggingTensorHook(tensors={'step': global_step, 'loss': loss}, every_n_iter=10),
            ]

        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        config.gpu_options.visible_device_list = str(hvd.local_rank())

        checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None

        batch_size = 128
        total_batch = int(len(X_train) / batch_size)
        step = 0

        # do the following on the same graph as on the one above
        with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir, hooks=sync_hook, config=config) as mtsess:

            while not mtsess.should_stop():
                if step > total_batch - 1:
                    step = 0

                batch_x = X_train[step * batch_size: (step+1) * batch_size]
                batch_y = np.eye(class_num)[y_train[step * batch_size: (step+1) * batch_size]]
                train_feed_dict = {images: batch_x, labels: batch_y, training_flag : True}

                _, l, t_predict = mtsess.run([train_op, loss, train_prediction], feed_dict=train_feed_dict)
                #print("loss:", l)

            mtsess.close()

if __name__ == "__main__":
    tf.app.run()
```

MULTIGPU TRAINING /W TF/KERAS

Recipe 2 – Optimize fuel flow

1. Avoid using `sess.run([ops], feed_dict={...})`
2. Load data using tfrecords
    1. Skeleton for efficient multithreaded input pipline
    2. Move data augmentation in here
3. Train

→ GPU utilization should rise even more. Check if there is still a data pipeline problem

A TFRECORD READER

Recipe 2 – Optimize fuel flow

TFRecord is a binary format

1. Protocol Buffers
2. Deserialization
3. Augmentation
4. Returns graph nodes

```python
def read_and_decode(filename_queue):

    reader = tf.TFRecordReader()

    _, serialized_example = reader.read(filename_queue)

    features = tf.parse_single_example(serialized_example, features={
        'height': tf.FixedLenFeature([], tf.int64),
        'width' : tf.FixedLenFeature([], tf.int64),
        'img_raw': tf.FixedLenFeature([], tf.string),
        'label': tf.FixedLenFeature([], tf.float32)
    })

    image = tf.decode_raw(features['img_raw'], tf.uint8)
    image = tf.cast(image, tf.float32)
    image = image/256.

    label = tf.cast(features['label'], tf.int32)
    label = tf.one_hot(label, depth=2)

    height = tf.cast(features['height'], tf.int32)
    width = tf.cast(features['width'], tf.int32)

    width, height = IMAGE_WIDTH, IMAGE_HEIGHT

    image_shape = tf.stack([height, width, tf.constant(3)], axis = 0)
    image = tf.reshape(image, [height, width, 3])
    image = tf.image.random_flip_left_right(image, seed=42)
    image = tf.image.random_flip_up_down(image, seed=42)

    vrot = 20
    degrees = np.random.random() * vrot  - vrot/2

    image = tf.contrib.image.rotate(image, degrees * np.pi / 180., interpolation='BILINEAR')
    #image_size_const = tf.constant((IMAGE_HEIGHT, IMAGE_WIDTH, 3), dtype=tf.int32)
    #transform_image = tf.image.resize_image_with_crop_or_pad(image=image, target_height=IMAGE_HEIGHT, target_width=IMAGE

    images, labels = tf.train.shuffle_batch([image, label], batch_size=20, capacity=30, num_threads=16, min_after_dequeue

    return images, labels
```

MULTIGPU TRAINING /W TF/KERAS

Recipe 2 – Optimize fuel flow

1. We use the same model as before
2. Replace images and labels placeholders
3. TF Record nodes appear directly on the graph

```python
import numpy as np
import skimage.io as io
import tensorflow as tf
import horovod.tensorflow as hvd

import DenseNet as dn
slim = tf.contrib.slim

tf.logging.set_verbosity(tf.logging.INFO)

IMAGE_HEIGHT, IMAGE_WIDTH = 223,223
tfrec_fname = "risse.tfrecords"

def read_and_decode(filename_queue):

growth_k = 12
nb_block = 2 # how many (dense block + Transition Layer) ?
class_num = 2

def main(_):

    hvd.init()

    graph = tf.Graph()

    with graph.as_default():

        filename_queue = tf.train.string_input_producer(([tfrec_fname]), num_epochs=10)
        images, labels = read_and_decode(filename_queue)
        training_flag = tf.placeholder(tf.bool, name="training_flag")
        tf.train.create_global_step()

        images = tf.identity(images, name='input_image')

        logits = dn.DenseNet(x=images, nb_blocks=nb_block, filters=growth_k, training=training_flag).model

        assert(logits.shape == labels.shape)

        loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels))

        tower_optimizer = tf.train.RMSPropOptimizer(0.001 * hvd.size())
        global_optimizer = hvd.DistributedOptimizer(tower_optimizer)

        global_step = tf.train.get_or_create_global_step()

        train_op = global_optimizer.minimize(loss=loss, global_step=global_step)
        train_prediction = tf.nn.softmax(logits=logits)

```

MULTIGPU TRAINING /W TF/KERAS

Recipe 2 – Optimize fuel flow

1. Skipped manual data loading
2. Start reader threads to put data into our input nodes
    1. tf.train.Coordinator()
    2. tf.train.start_queue_runners()
    3. Reference input nodes as ops
3. Graceful shutdown
4. Train

→ Check GPU/CPU utilization. Further bottleneck analysis required?

```
        sync_hook = [
            hvd.BroadcastGlobalVariablesHook(0),
            tf.train.StopAtStepHook(last_step=20000 // hvd.size()),
            tf.train.LoggingTensorHook(tensors={'step': global_step, 'loss': loss}, every_n_iter=10),
            ]

        config = tf.ConfigProto()
        config.gpu_options.allow_growth = True
        config.gpu_options.visible_device_list = str(hvd.local_rank())

        checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None

        # do the following on the same graph as on the one above
        with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir, hooks=sync_hook, config=config) as mtsess:
            coord = tf.train.Coordinator()
            threads = tf.train.start_queue_runners(sess=mtsess, coord=coord)

            while not mtsess.should_stop():
                img, lbls, _, l, t_predict = mtsess.run([images, labels, train_op, loss, train_prediction], feed_dict={trainin
                #print("loss:", l)

            coord.request_stop()
            coord.join(threads)

            mtsess.close()

if __name__ == "__main__":
    tf.app.run()
```

```
root@8eb16c5b074d:/workspace/home/Risse# mpirun -np 8 --allow-run-as-root -H localhost:8 -bind-to none -map-by
slot -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH -mca pml ob1 -mca btl ^openib python trainontfrecord.py
u@  16.04 0:-* 1:-- 2:-                                    40d1h 5.34 80x2.1GHz 503.8G1% 2018-06-13 16:13:11
```

CALL TO ACTION

Roll up the sleeves!

NVIDIA GPU CLOUD: Register free @ NGC.NVIDIA.COM

NVIDIA Docker Runtime: github nvidia-docker

Turn your training code into MultiGPU capable training code!

Register for GTC – join more sessions on MultiGPU and Tensorcores!

https://www.nvidia.com/en-eu/gtc/

Enjoy Early Bird Pricing until 10th July and use following code for 25% discount:

NVAHOHL

NVIDIA DEEP LEARNING INSTITUTE

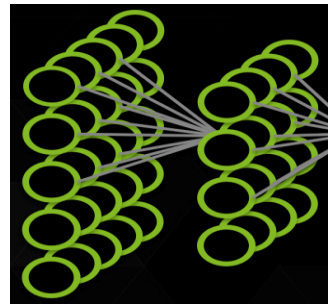
Hands-on, self-paced and instructor-led training in deep learning and accelerated computing for developers and data scientists

Request onsite instructor-led workshops at your organization: **www.nvidia.com/requestdli**

Take self-paced courses online: **www.nvidia.co.uk/dlilabs**

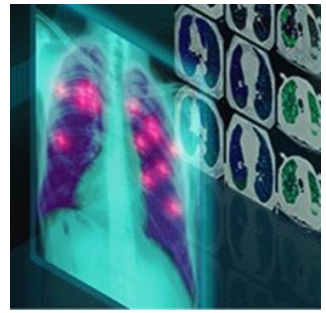
Download the course catalog and view upcoming public workshops: **www.nvidia.co.uk/dli**

Caffe2 · Microsoft Cognitive Toolkit · mxnet · TensorFlow · PYTORCH

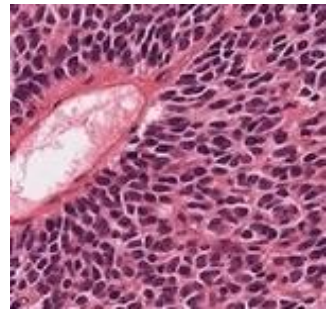


Deep Learning Fundamentals

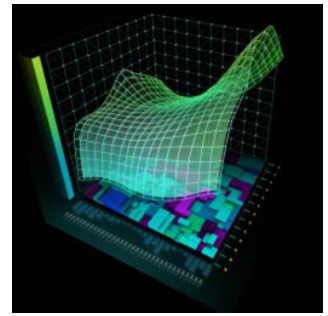
Autonomous Vehicles



Medical Image Analysis

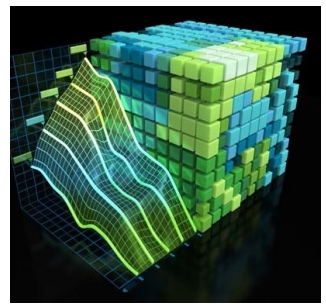


Genomics



Finance

Game Development & Digital Content



Accelerated Computing Fundamentals

More industry-specific training coming soon…


NVIDIA