

Rise of GPU Computing

PowerAI, *World's Fastest AI Platform*



Justin McCoy

justin.mccoy@us.ibm.com
@mccoyjus

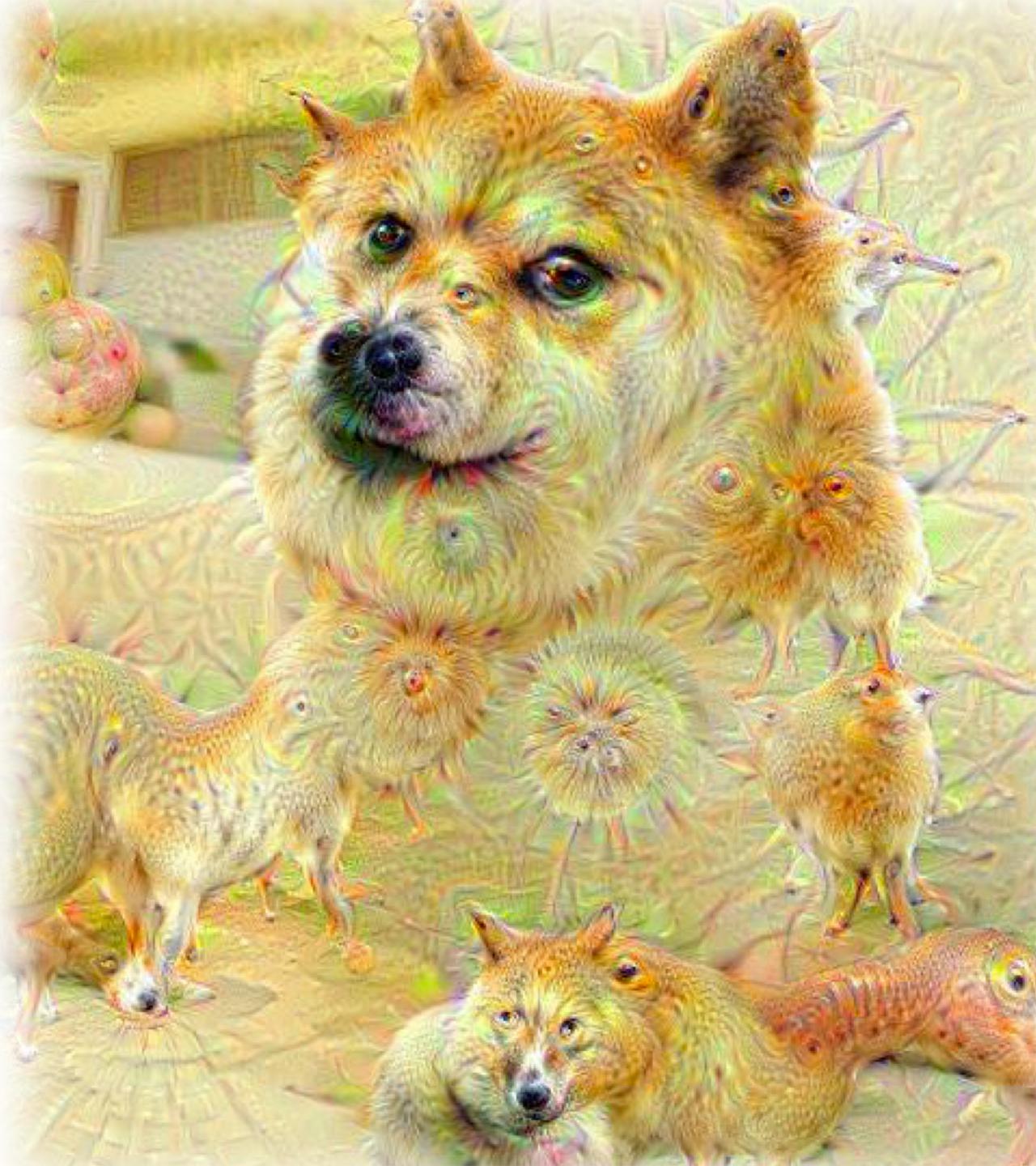
IBM

CODE

Our Journey Today

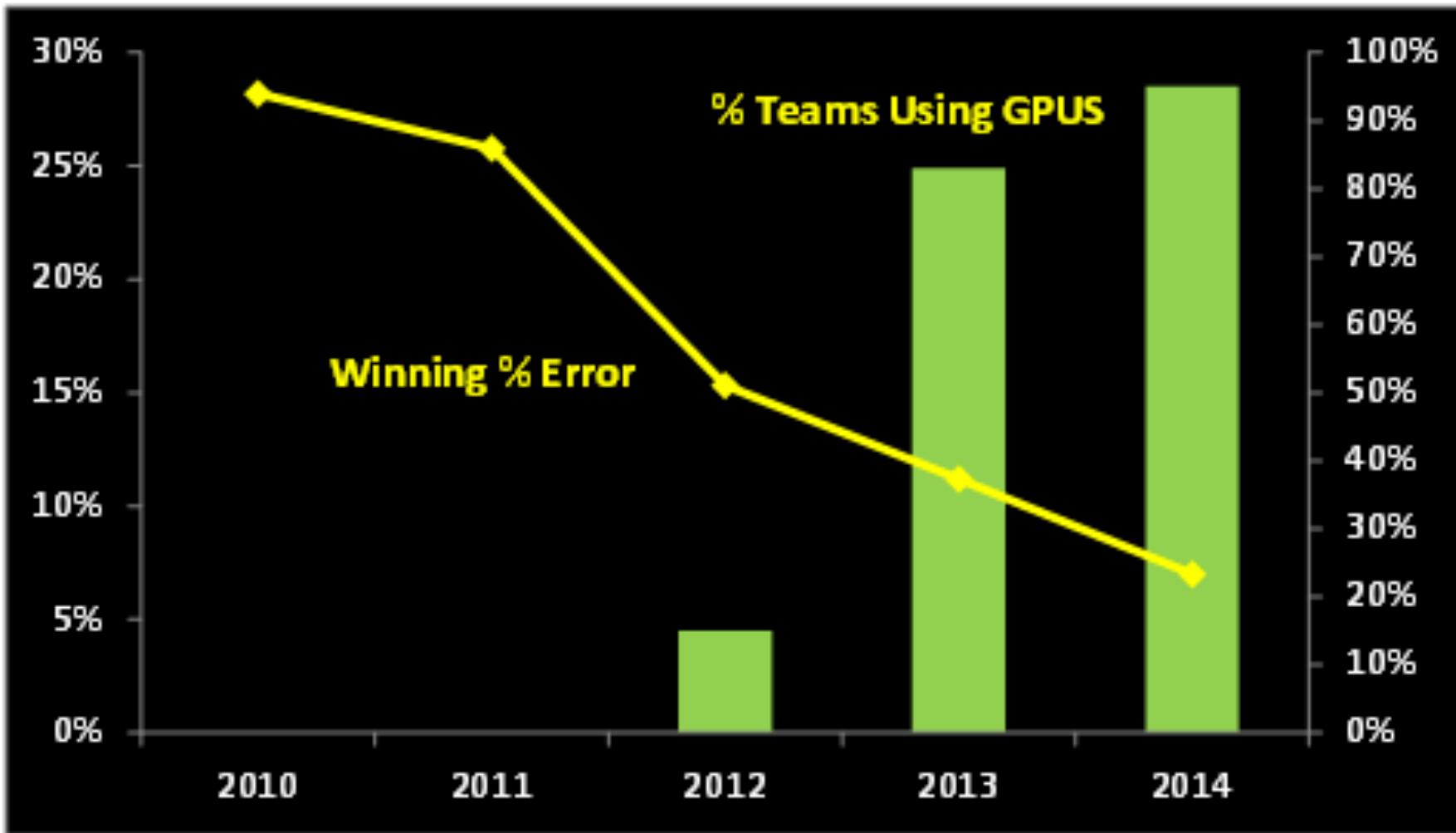
- Rise of GPU assisted Neural Networks
- PowerAI Platform
- Get started with PowerAI Platform

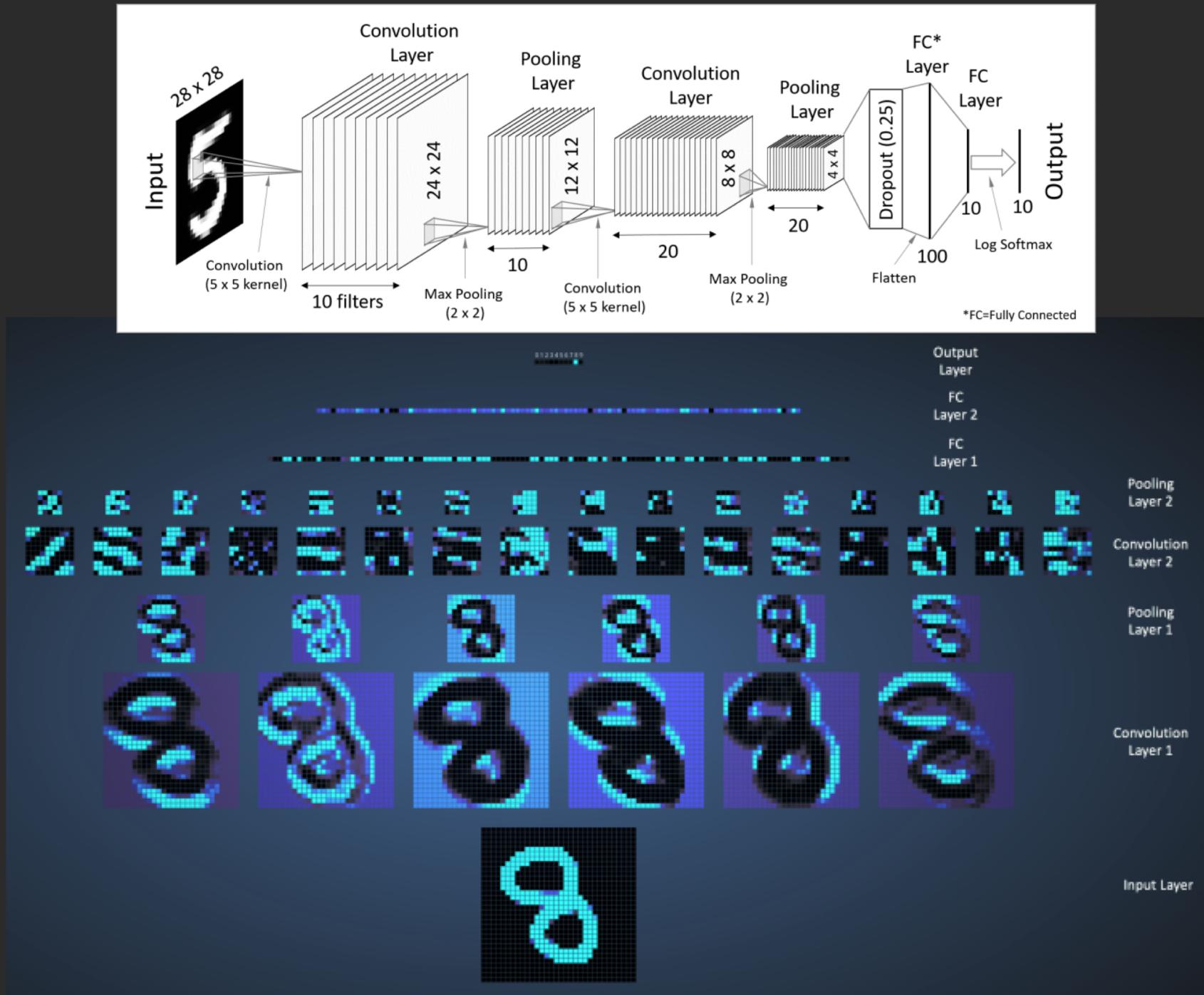
Let's Build Something: Hands on Demo



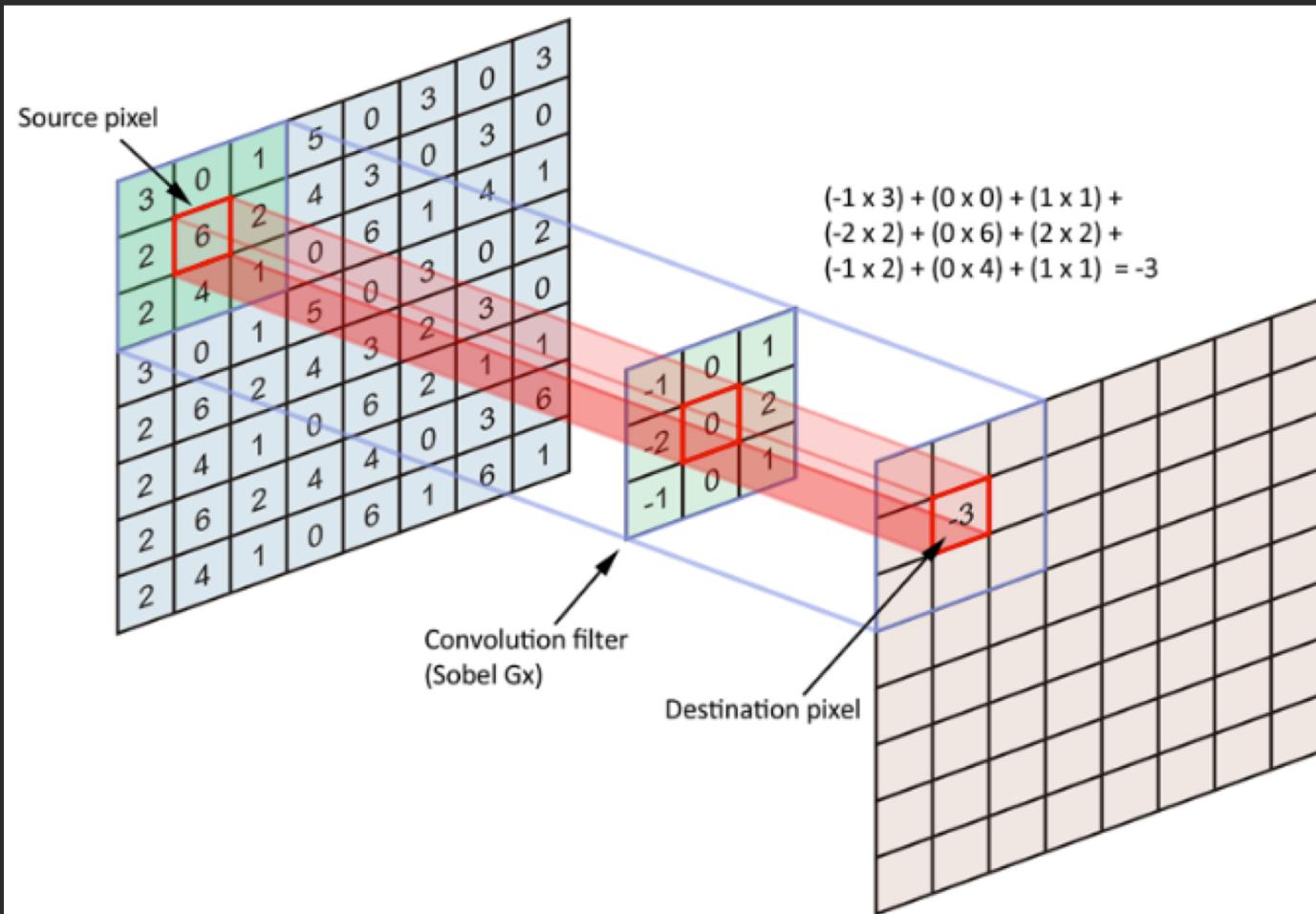
A subset of Machine Learning,
Deep Learning is the field of
building models to classify and
make predictions by
representation of the data in
successive layers with
increasing meaningful
representation.

IMAGENET

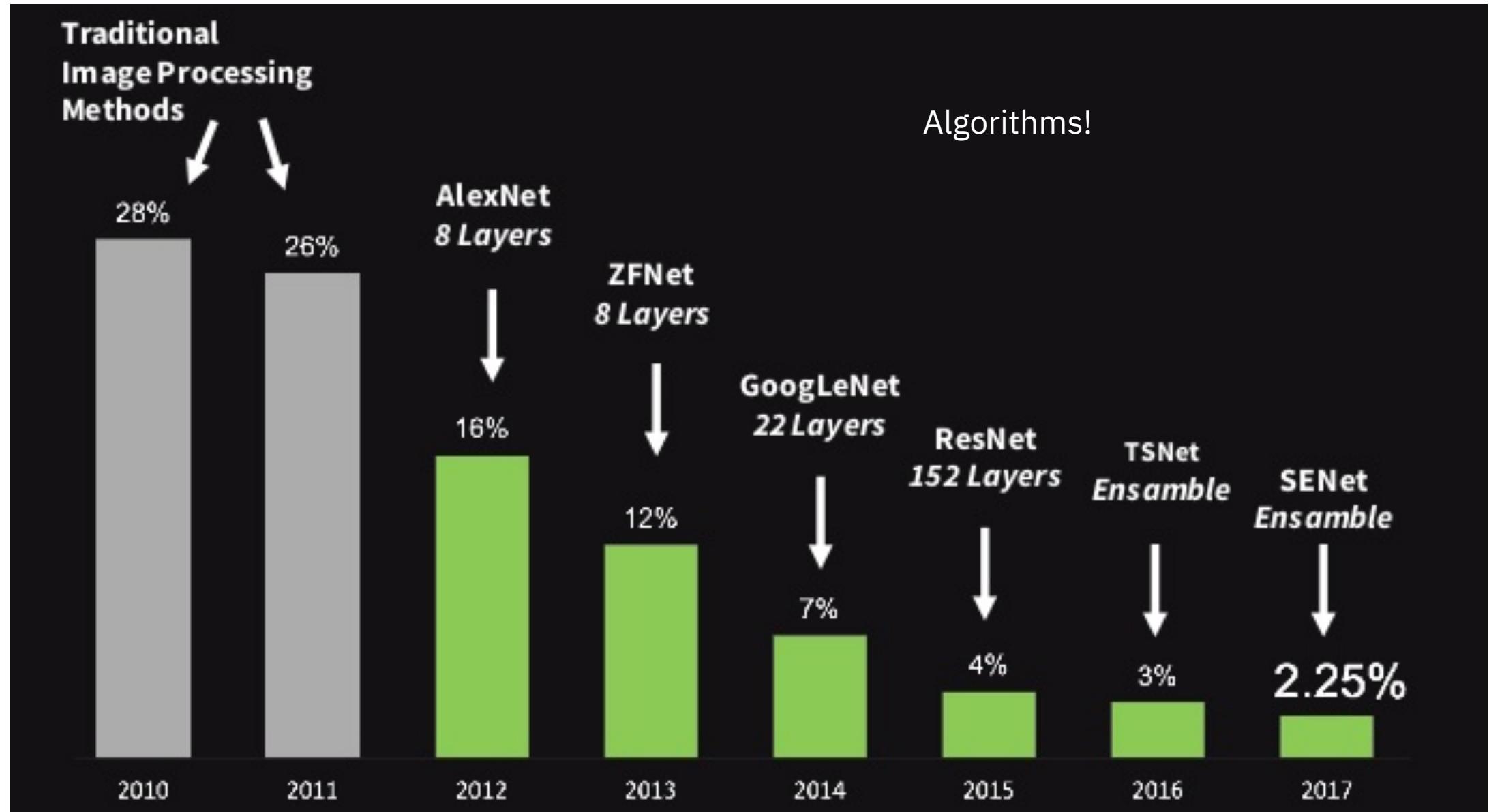




GPU Efficiency - Designed for Matrix Multiply

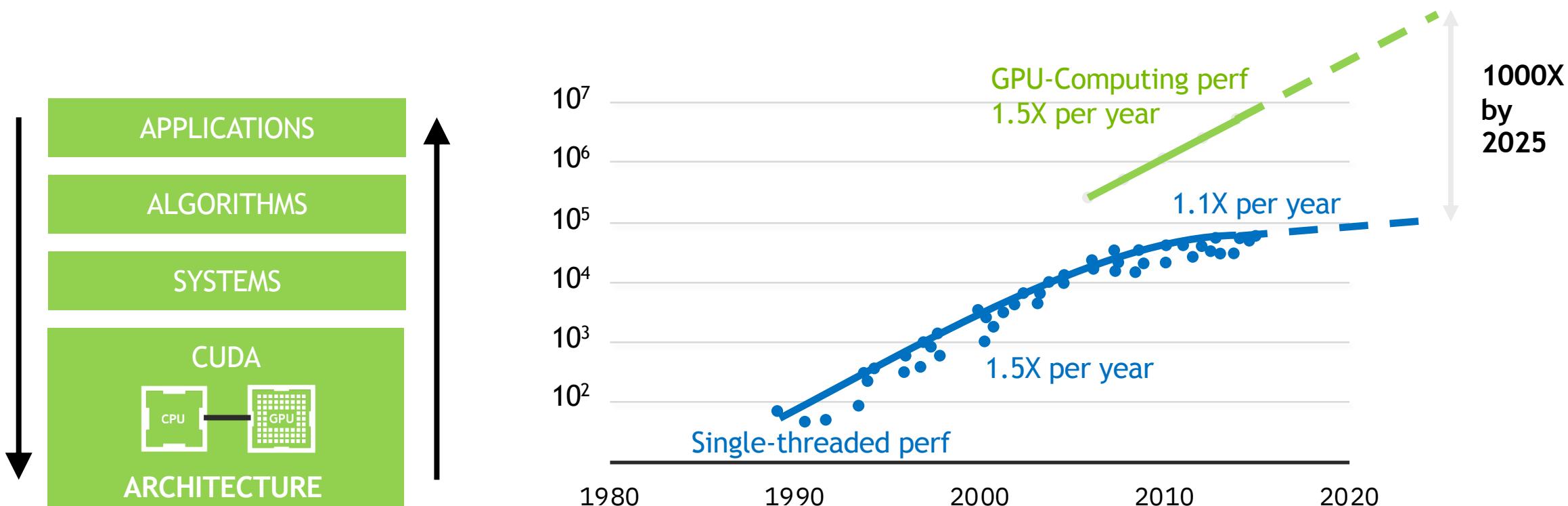


IMAGENET Top 5 Error Rate



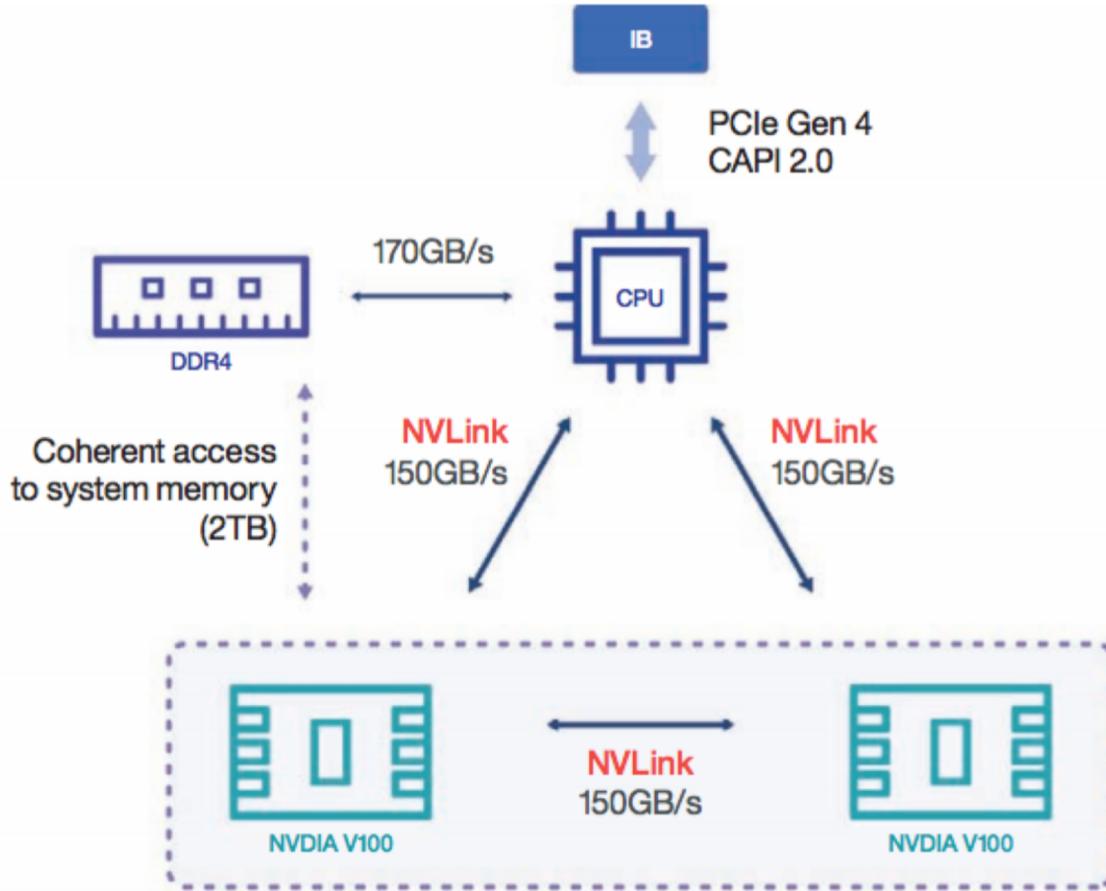


RISE of GPU computing



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.

Hardware Matters



where is Deep
Learning Being used
today?

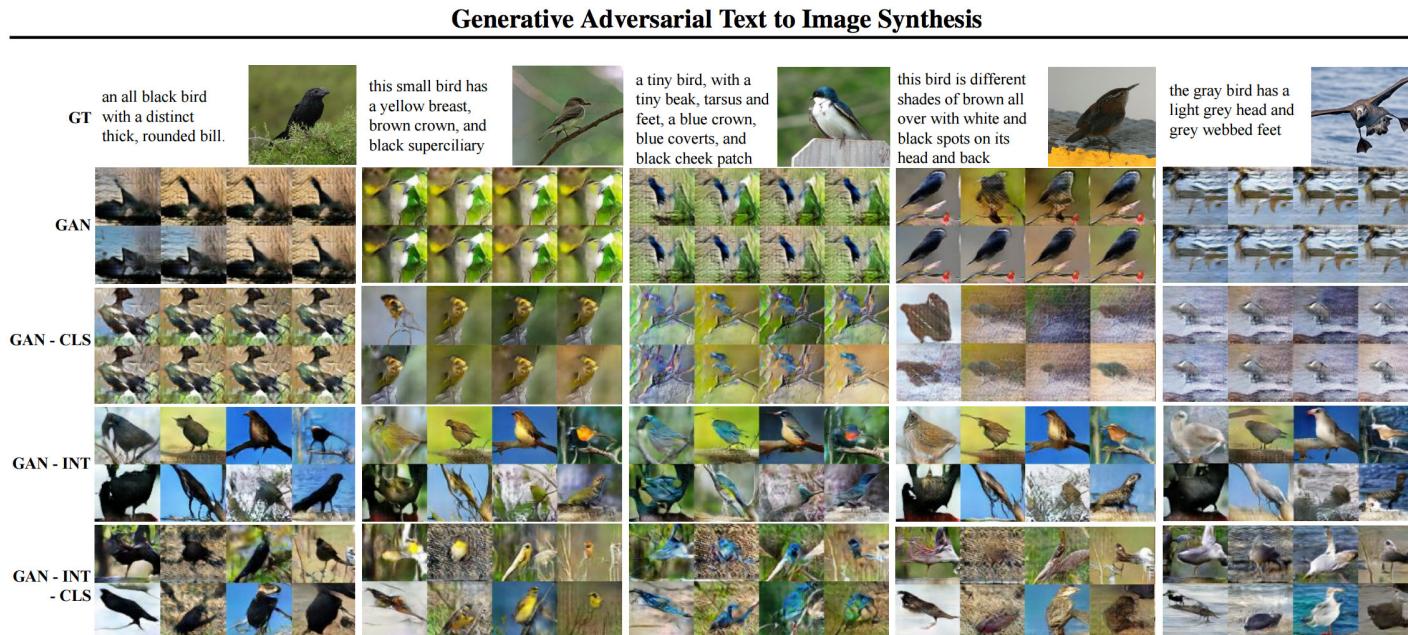
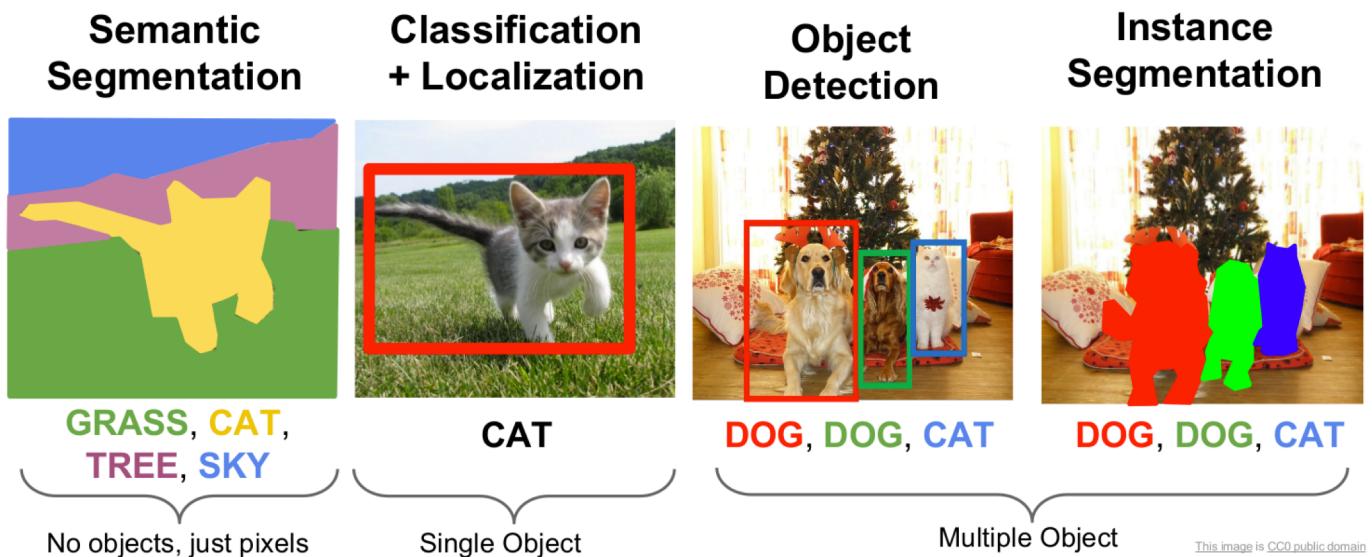


Figure 3. Zero-shot (i.e. conditioned on text from unseen test set categories) generated bird images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. We found that interpolation regularizer was needed to reliably achieve visually-plausible results.

Tools of the trade



IBM PowerAI Enterprise Platform

Enterprise-Grade AI Software, Optimized for Power

Deep Learning Frameworks & Enhancements



TensorFlow
Distributed Deep Learning

Caffe

Power Systems Large Model Support

IBM Caffe

AI Vision Tools



Watson APIs

Supporting Capabilities And Libraries

Distributed Frameworks



IBM Research AI Vision Runtime

OpenBLAS

IBM Spectrum Conductor



IBM Services And Support

IBM Entire Stack Support

IBM Research Pioneering AI Research

Cognitive Class Education & Certification



Optimization and testing

IBM Power Accelerated Servers: Ideal for Enterprise AI Workloads

IBM Power AC922

Acceleration Superhighway

Designed for The AI era

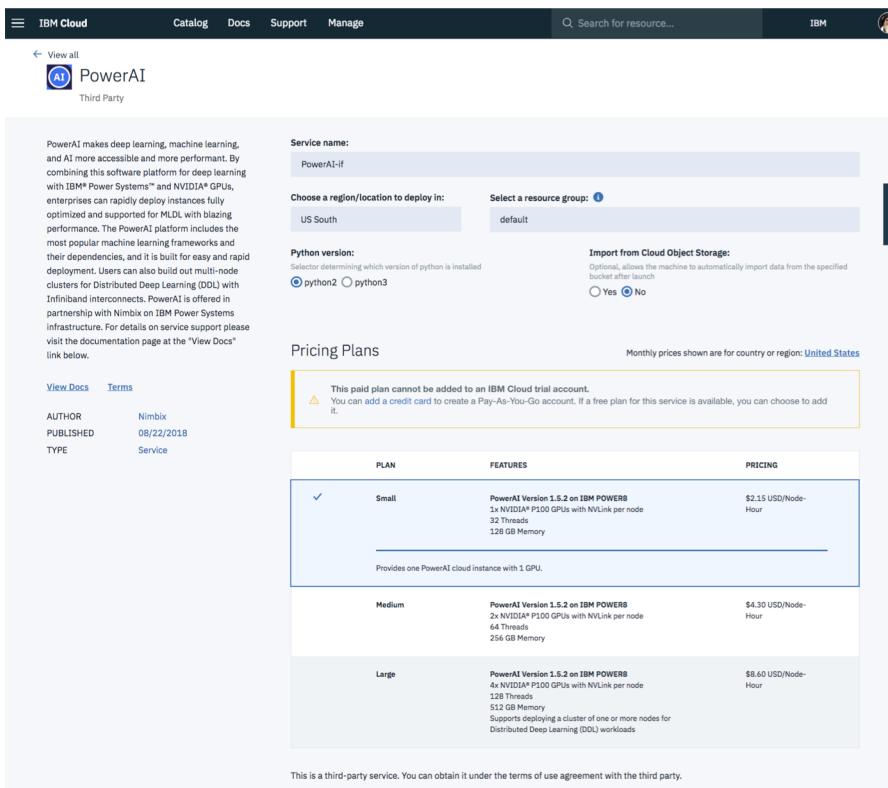
Enterprise Grade

POWER9 Performance

PowerAI

Access to the PowerAI platform is available in the Cloud with a pay-for-use model, or as a free download to install locally on your Power8/9 system with GPUs

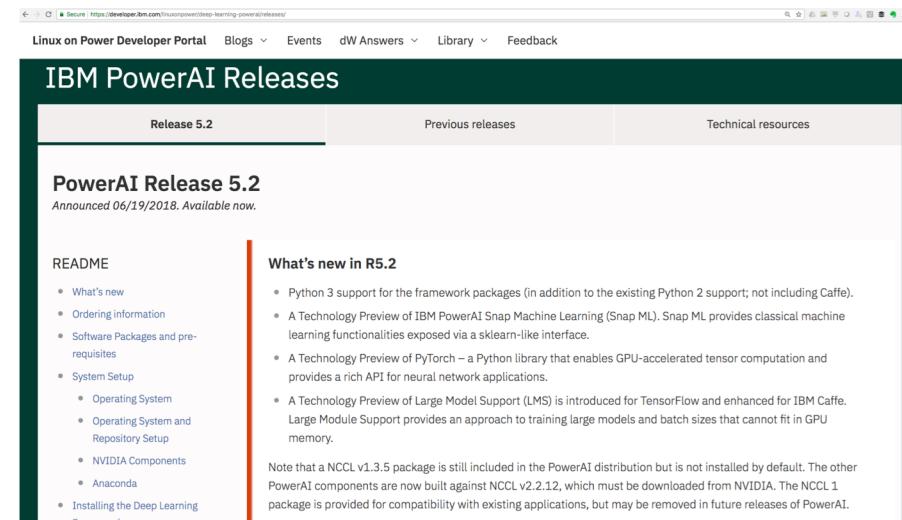
IBM Cloud Service <https://bluemix.net>



The screenshot shows the IBM Cloud Service interface for deploying a PowerAI service. It includes fields for Service name (PowerAI-if), Region (US South), Resource group (default), Python version (python3 selected), and Import from Cloud Object Storage (No selected). Below this, there's a Pricing Plans section with three options: Small, Medium, and Large. The Small plan is selected, showing details like PowerAI Version 1.5.2 on IBM POWER8, 1x NVIDIA P100 GPU with NVLink per node, 32 threads, and 128 GB Memory. The Medium plan shows 2x NVIDIA P100 GPUs with NVLink per node, 64 threads, and 256 GB Memory. The Large plan shows 4x NVIDIA P100 GPUs with NVLink per node, 128 threads, and 512 GB Memory. A note at the bottom states: "This is a third-party service. You can obtain it under the terms of use agreement with the third party."

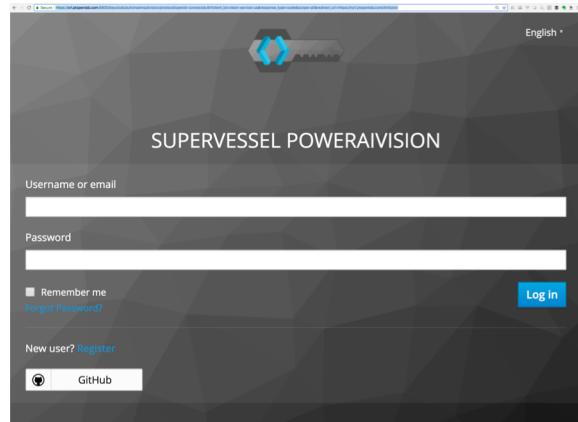


Download for PowerAI HW <https://ibm.biz/powerai>



The screenshot shows the IBM PowerAI Releases page for Release 5.2. It features a navigation bar with tabs for Release 5.2, Previous releases, and Technical resources. The main content area is titled "PowerAI Release 5.2" (Announced 06/19/2018. Available now.) and includes sections for "README" and "What's new in R5.2". The "README" section lists items such as What's new, Ordering information, Software Packages and prerequisites, System Setup, and Installing the Deep Learning Containerization. The "What's new in R5.2" section lists several new features and improvements, including Python 3 support, Snap ML, PyTorch, and Large Model Support (LMS).

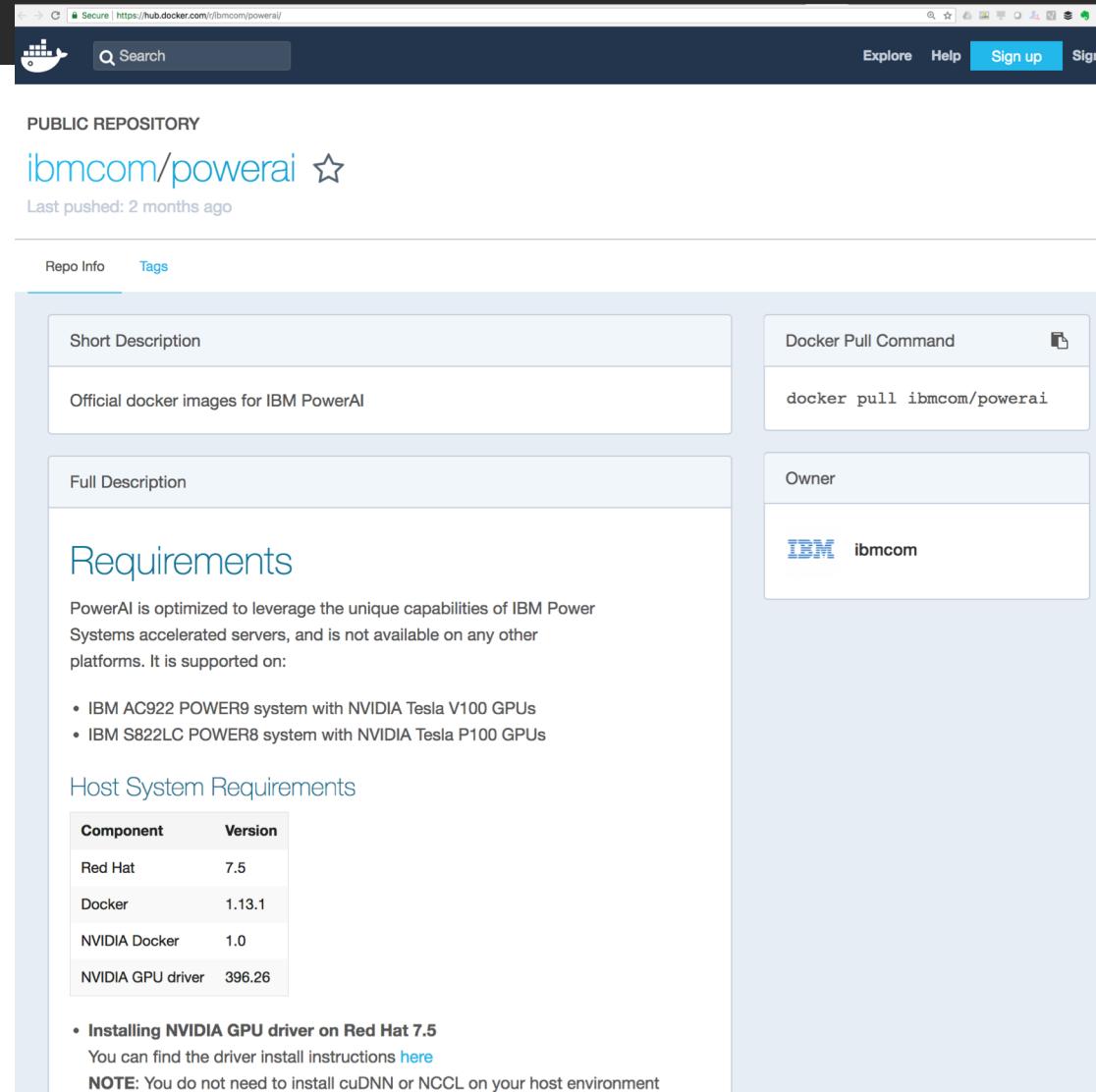
PowerAI Vision Preview <https://ibm.biz/powerai>



The screenshot shows the PowerAI Vision Preview login page. It has fields for Username or email and Password, a Remember me checkbox, a Forget Password link, and a GitHub integration button. The background features a dark, geometric pattern with the text "SUPERVESSEL POWERAIVISION" and the IBM logo.

PowerAI Docker Container

Download the latest PowerAI distribution for free as a Docker container
<https://hub.docker.com/r/ibmcom/powerai/>



The screenshot shows the Docker Hub interface for the 'ibmcom/powerai' repository. At the top, there's a search bar and navigation links for 'Explore', 'Help', 'Sign up', and 'Sign In'. Below the header, the repository name 'ibmcom/powerai' is displayed with a star icon, and it's noted that it was 'Last pushed: 2 months ago'. There are two tabs: 'Repo Info' (which is active) and 'Tags'. The 'Repo Info' section contains fields for 'Short Description' (containing 'Official docker images for IBM PowerAI') and 'Docker Pull Command' (containing 'docker pull ibmcom/powerai'). It also shows the 'Owner' as 'IBM ibmcom'. The 'Full Description' section includes a 'Requirements' section detailing compatibility with specific IBM Power Systems and a list of supported GPUs. It also lists host system requirements, including Red Hat 7.5, Docker 1.13.1, NVIDIA Docker 1.0, and NVIDIA GPU driver 396.26. A note at the bottom states that cuDNN or NCCL are not required for the host environment.

PUBLIC REPOSITORY

ibmcom/powerai ☆

Last pushed: 2 months ago

Repo Info Tags

Short Description

Official docker images for IBM PowerAI

Docker Pull Command

docker pull ibmcom/powerai

Full Description

Requirements

PowerAI is optimized to leverage the unique capabilities of IBM Power Systems accelerated servers, and is not available on any other platforms. It is supported on:

- IBM AC922 POWER9 system with NVIDIA Tesla V100 GPUs
- IBM S822LC POWER8 system with NVIDIA Tesla P100 GPUs

Host System Requirements

Component	Version
Red Hat	7.5
Docker	1.13.1
NVIDIA Docker	1.0
NVIDIA GPU driver	396.26

• Installing NVIDIA GPU driver on Red Hat 7.5
You can find the driver install instructions [here](#)

NOTE: You do not need to install cuDNN or NCCL on your host environment

PowerAI Code

```
In [3]: # Create some wrappers for simplicity
def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)

def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1],
                          padding='SAME')

# Create model
def conv_net(x, weights, biases, dropout):
    # MNIST data input is a 1-D vector of 784 features (28*28 pixels)
    # Reshape to match picture format [Height x Width x Channel]
    # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
    x = tf.reshape(x, shape=[-1, 28, 28, 1])

    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)

    # Convolution Layer
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling)
    conv2 = maxpool2d(conv2, k=2)

    # Fully connected layer
    # Reshape conv2 output to fit fully connected layer input
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    # Apply Dropout
    fc1 = tf.nn.dropout(fc1, dropout)

    # Output, class prediction
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out
```

```
In [4]: # Store layers weight & bias
weights = {
    # 5x5 conv, 1 input, 32 outputs
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    # fully connected, 7*7*64 inputs, 1024 outputs
    'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])),
    # 1024 inputs, 10 outputs (class prediction)
    'out': tf.Variable(tf.random_normal([1024, num_classes]))
}
```

```
biases = {
    'bc1': tf.Variable(tf.random_normal([32])),
    'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([num_classes]))
}

# Construct model
logits = conv_net(x, weights, biases, keep_prob)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)

# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

```
In [5]: # Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    for step in range(1, num_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: dropout})
        if step % display_step == 0 or step == 1:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                                Y: batch_y,
                                                                keep_prob: 1.0})
            print("Step " + str(step) + ", Minibatch Loss= " + \
                  "{:.4f}".format(loss) + ", Training Accuracy= " + \
                  "{:.3f}".format(acc))

    print("Optimization Finished!")

    # Calculate accuracy for 256 MNIST test images
    print("Testing Accuracy:"),
    sess.run(accuracy, feed_dict={X: mnist.test.images[:256],
                                 Y: mnist.test.labels[:256],
                                 keep_prob: 1.0})
```

PowerAI Vision Platform

IBM PowerAI Vision admin

My DL Tasks / subway

Task Status - subway

My Workspace

- My Data Sets
- My DL Tasks**
- My Trained Models
- My Web APIs
- My Network Design
- Video Data Platform

AI Vision

Select Dataset Build Model Deploy And Test

Latest Status: training Show Log

Total Iteration: 4000
Train Iteration: 320
Train Loss CLS: 0.07546
Train Loss BBox: 0.12429

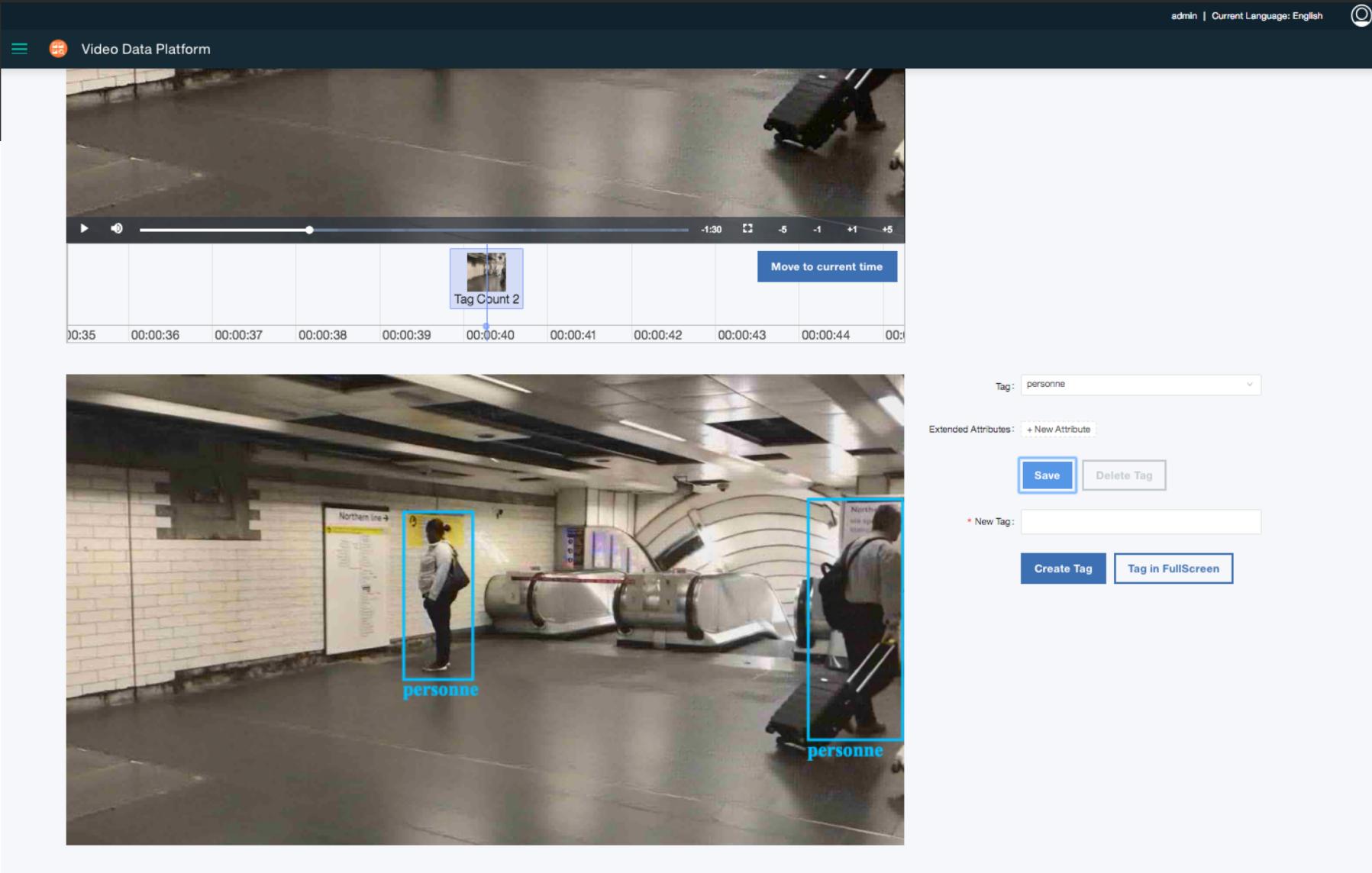
Estimated time left: 0.39 hour

Train Loss CLS Train Loss Bbox

Iteration	Train Loss CLS	Train Loss Bbox
20	0.45	0.48
40	0.38	0.45
60	0.32	0.42
80	0.28	0.38
100	0.25	0.35
120	0.20	0.30
140	0.18	0.28
160	0.16	0.25
180	0.14	0.23
200	0.12	0.20
220	0.11	0.18
240	0.10	0.17
260	0.09	0.16
280	0.08	0.15
300	0.08	0.14
320	0.08	0.13

Back Early Stop

PowerAI Video Data Platform



Demo and Code

Code Patterns

Locate and count items with object detection

Breast Cancer Classification with IBM PowerAI Vision

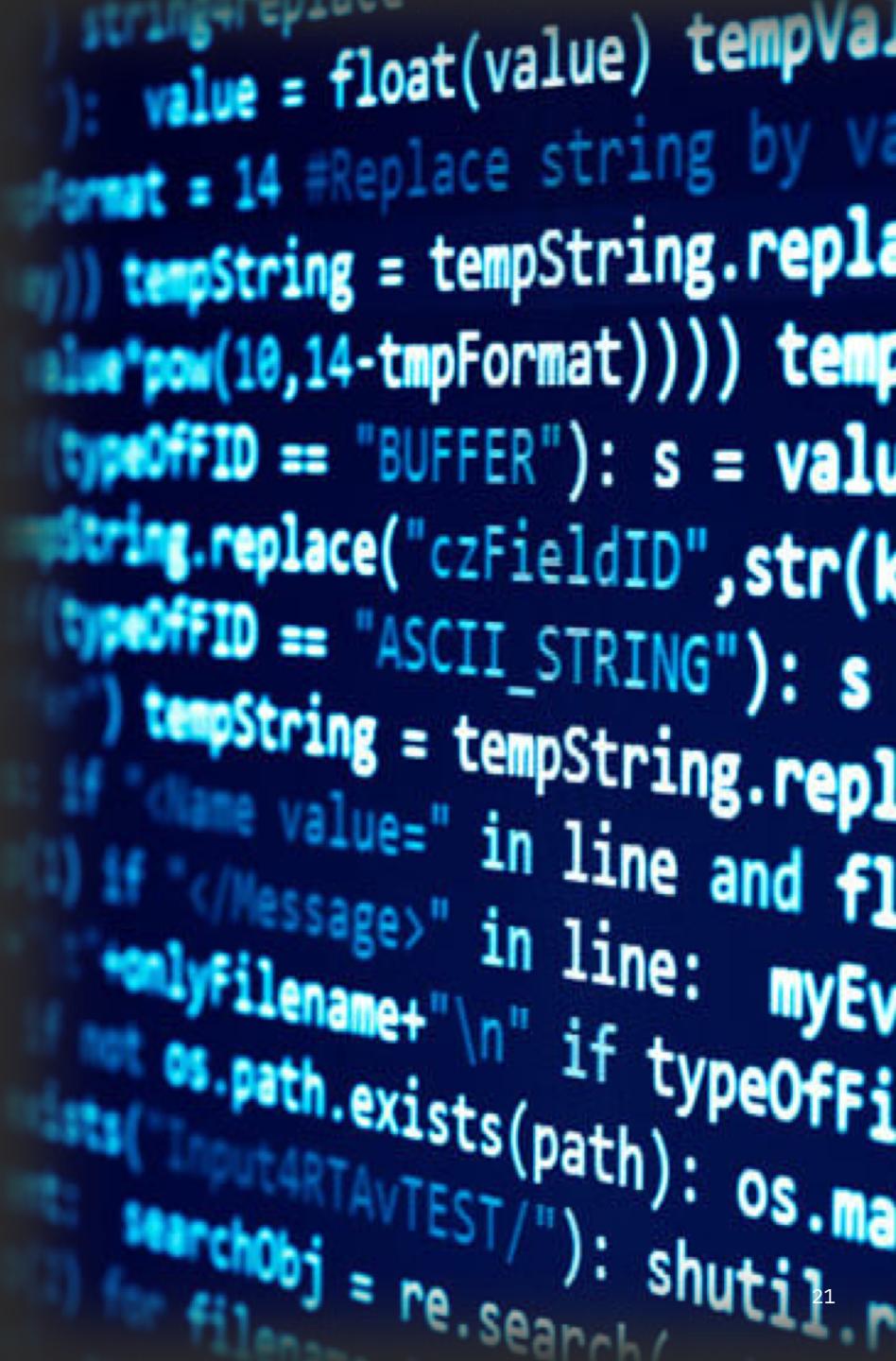
Detect, track, and count cars in a video

Accelerate training of Machine Learning Algorithms

Image recognition training with PowerAI Notebooks

SETI Signal Classification on PowerAI

Image classifier on iOS w/CoreML



Thank You

If you're interested in learning more about AI, Machine Learning, Deep Learning or anything IBM Cloud, reach out to Lennart or Justin via email or come talk to us.

Code and examples:

<https://developer.ibm.com>

<https://ibm.biz/powerai>

https://ibm.biz/powerai_counting_cars



Lennart Frantzell
alf@us.ibm.com



Justin McCoy
justin.mccoy@us.ibm.com

Distributed Deep Learning (DDL)

IBM Research solved the scale-out problem in deep learning!

Almost Linear Scale up to 100's of GPUs at 95%

Reduces training time from weeks to hours for complex deep neural networks

