Rise of GPU Computing

PowerAI, *World's Fastest AI Platform*

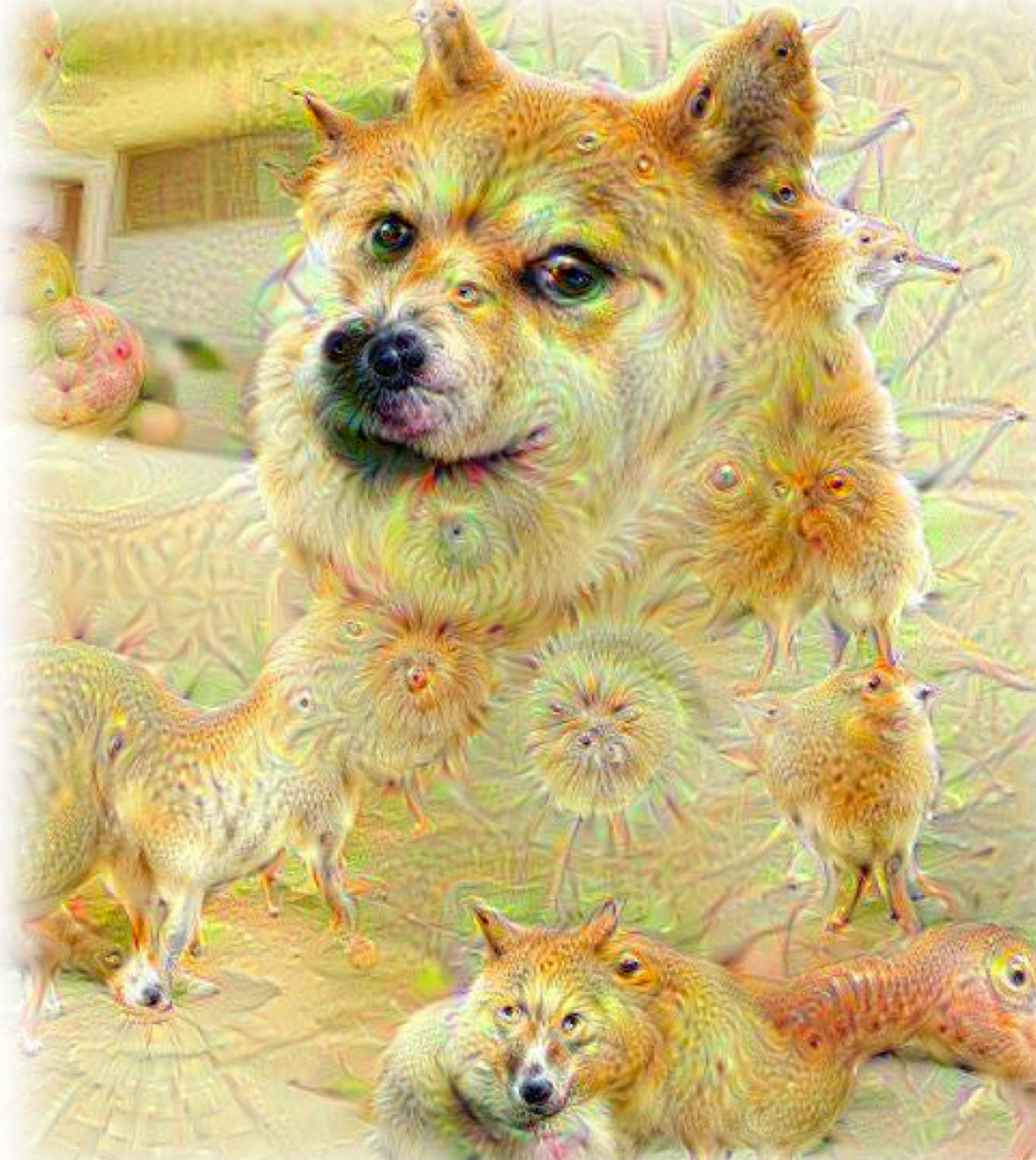
Justin McCoy
justin.mccoy@us.ibm.com
@mccoyjus

IBM
CODE

Our Journey Today

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

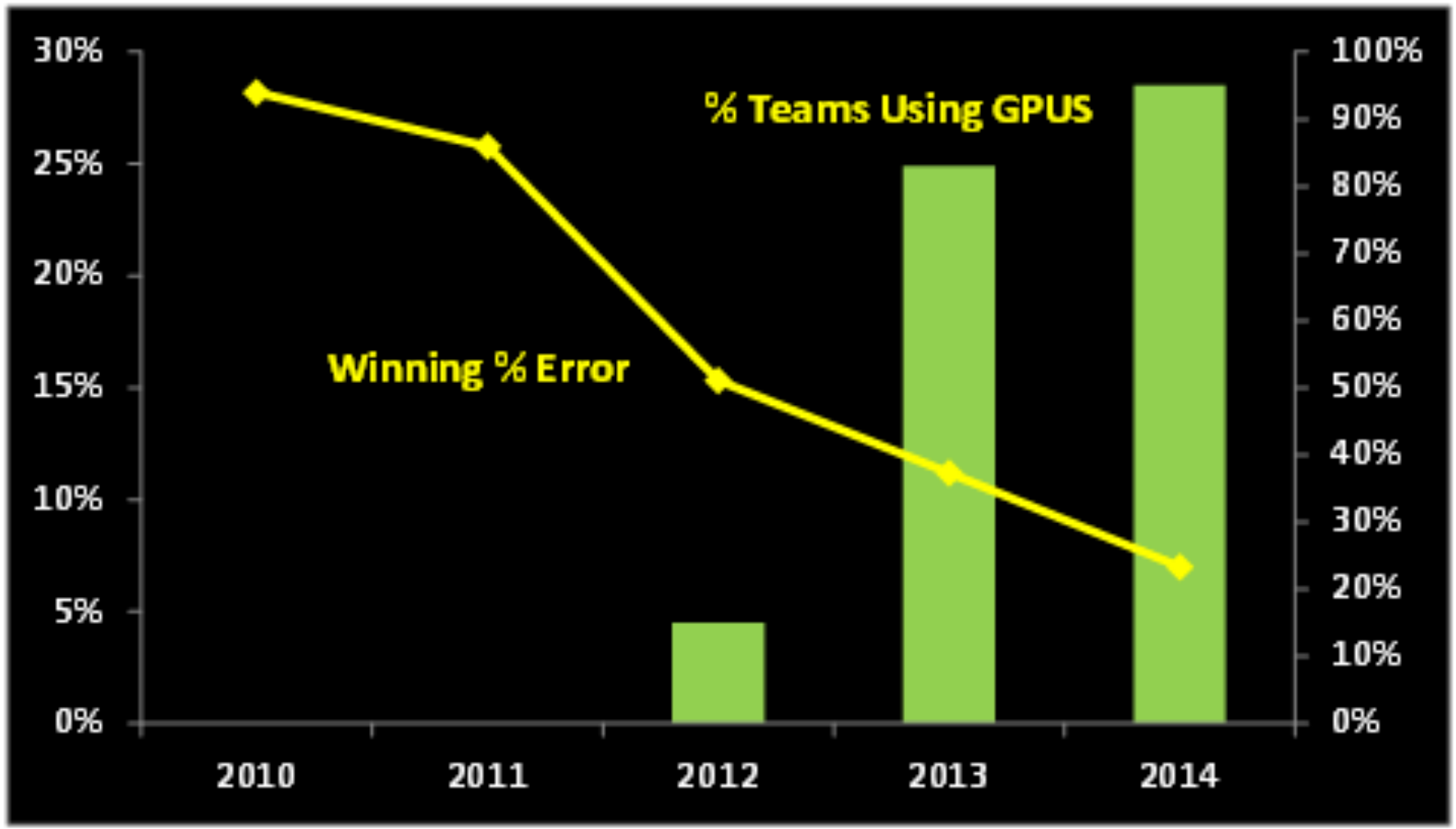
Let's Build Something: Hands on Demo



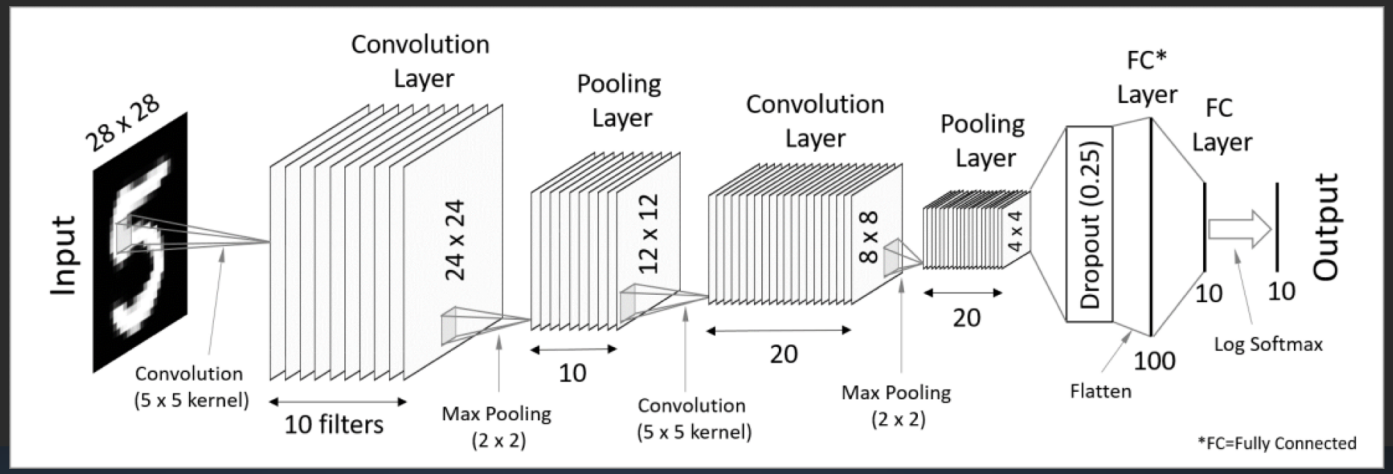
IBM CODE

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

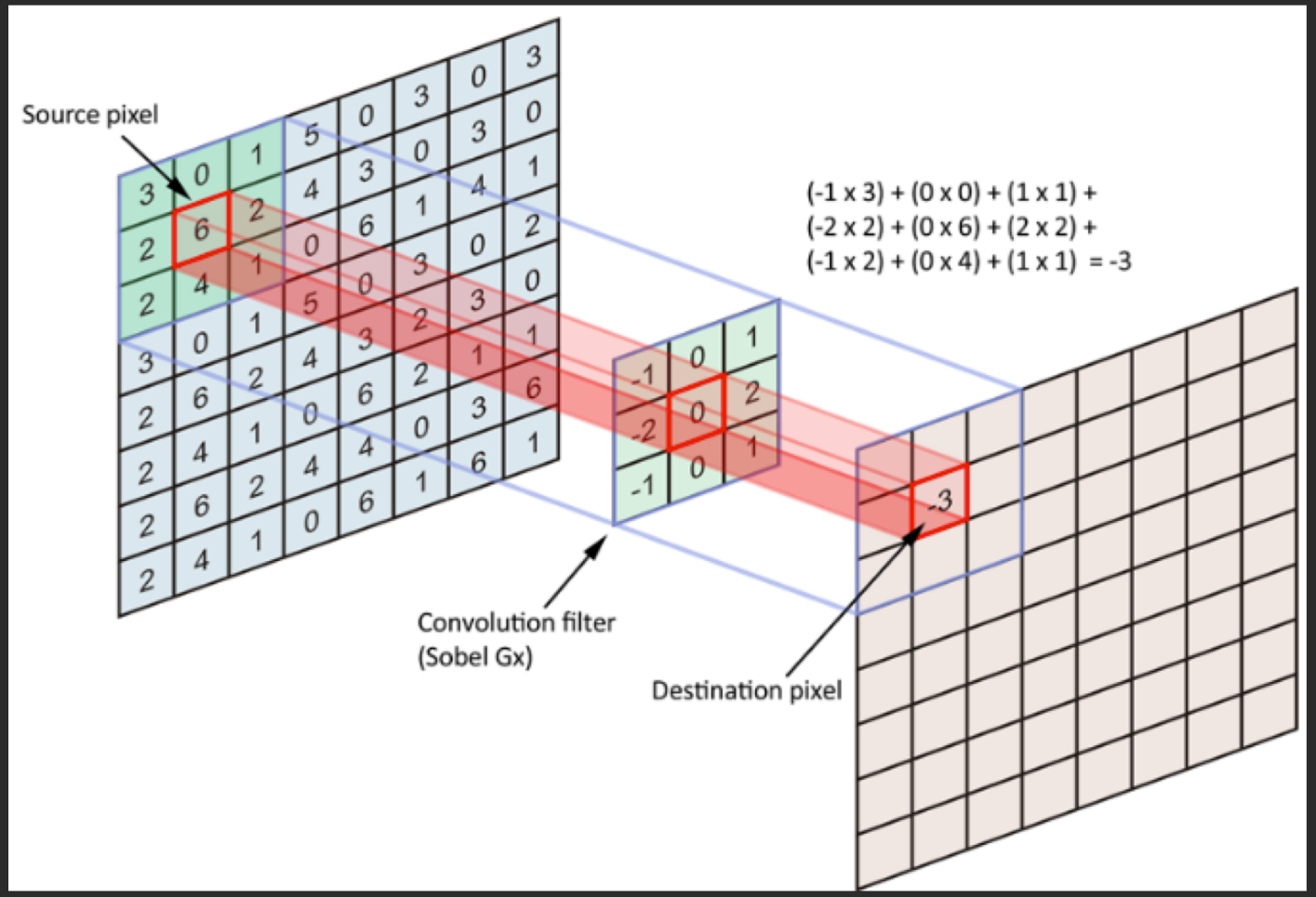


IBM CODE

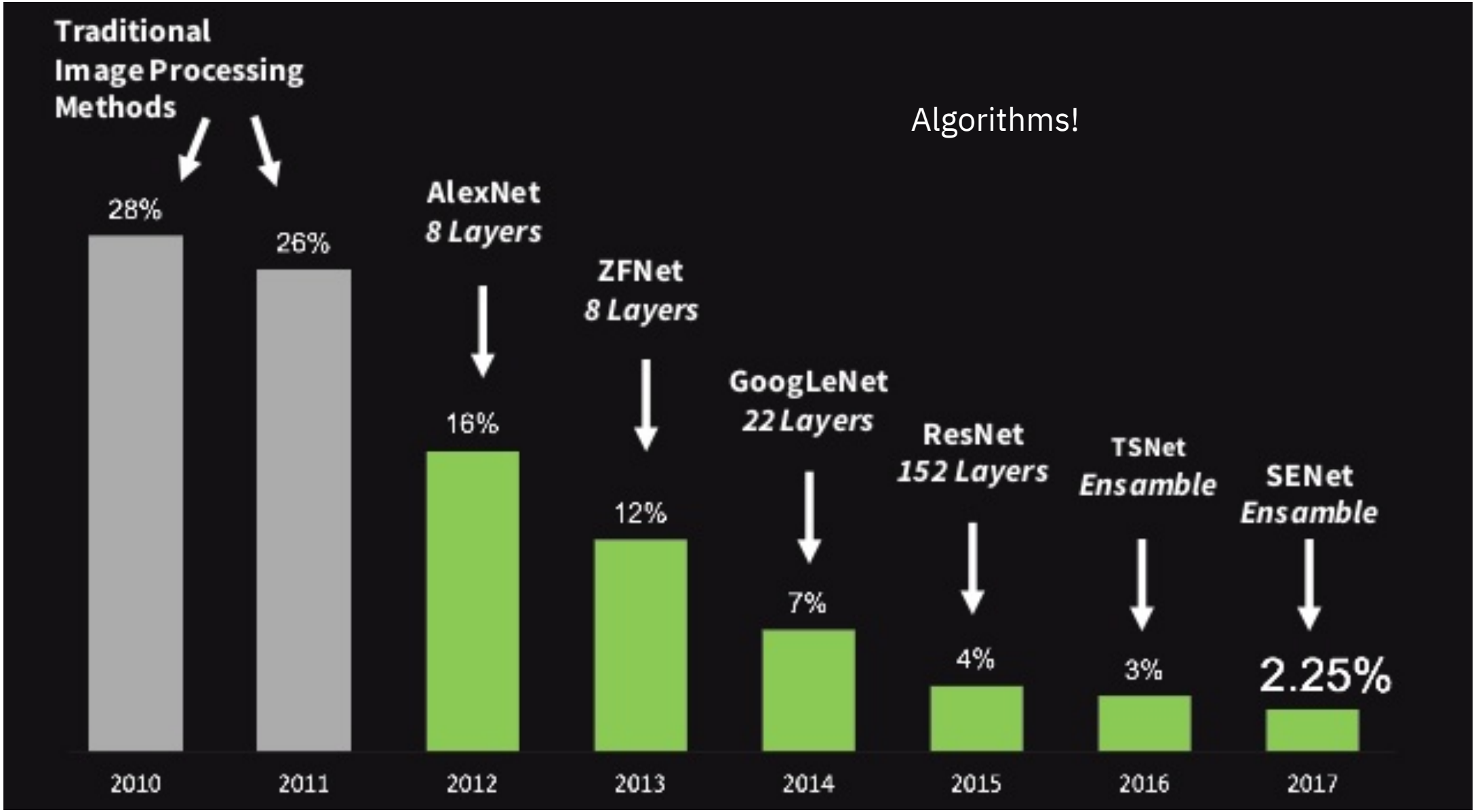

Input
28 x 28
Convolution
(5 x 5 kernel)
Convolution
Layer
24 x 24
10 filters
Max Pooling
(2 x 2)
Pooling
Layer
12 x 12
10
Convolution
(5 x 5 kernel)
Convolution
Layer
8 x 8
20
Max Pooling
(2 x 2)
Pooling
Layer
4 x 4
20
Dropout (0.25)
FC*
Layer
Flatten
100
FC
Layer
10
Log Softmax
10
Output
*FC=Fully Connected


Output
Layer
FC
Layer 2
FC
Layer 1
Pooling
Layer 2
Convolution
Layer 2
Pooling
Layer 1
Convolution
Layer 1
Input Layer

GPU Efficiency - Designed for Matrix Multiply



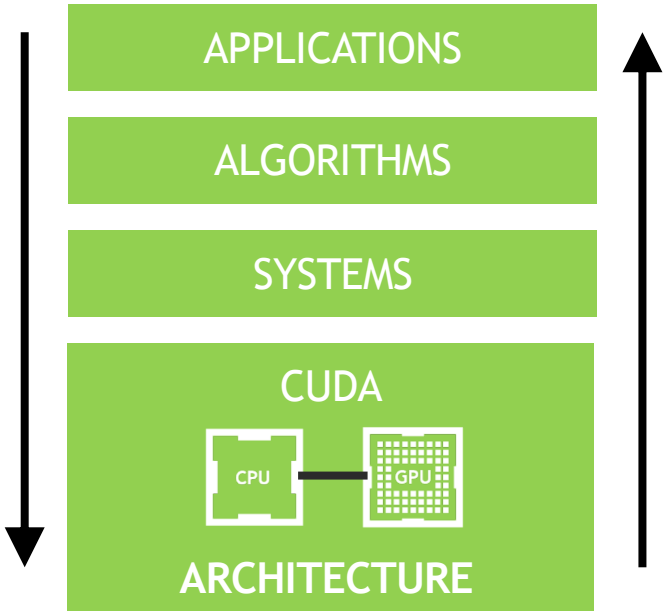
IMAGENET Top 5 Error Rate

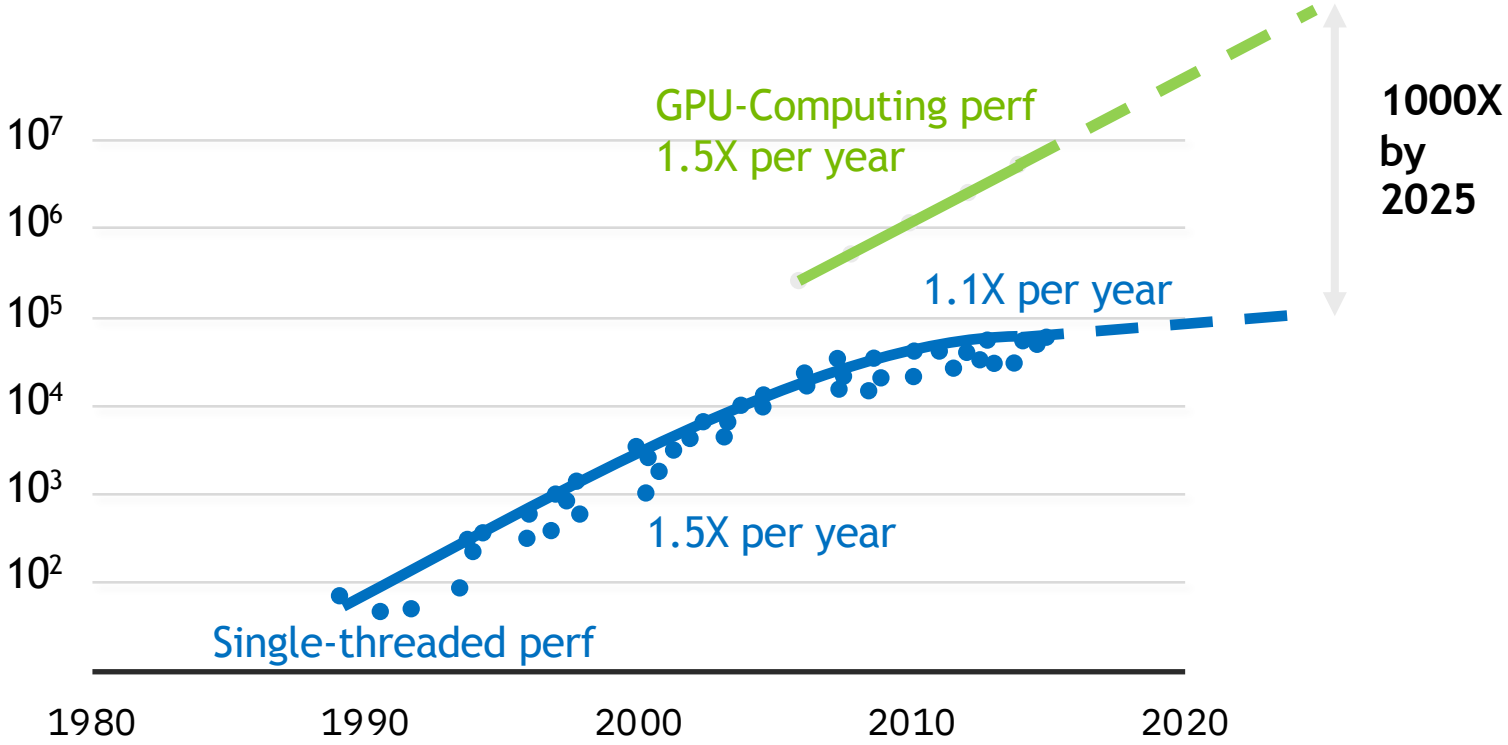


Algorithms!

IBM
CODE

RISE of GPU computing

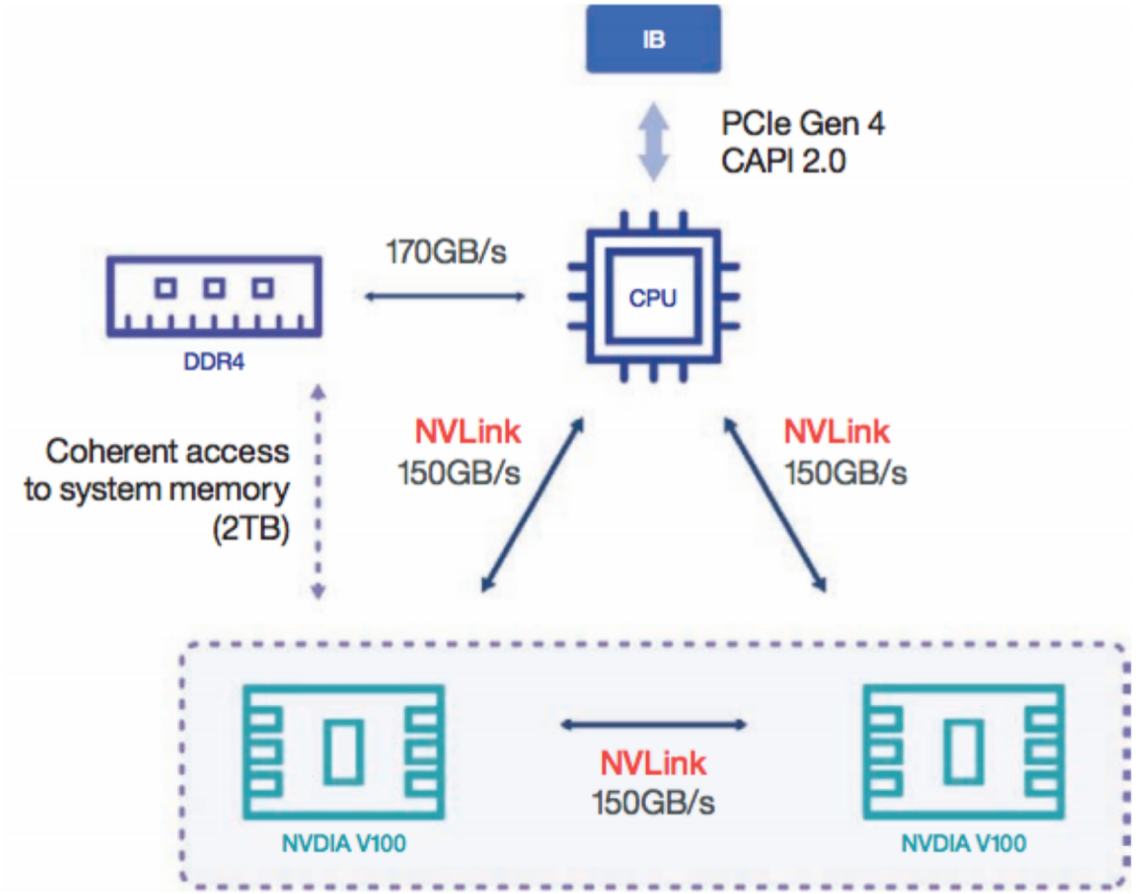




Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

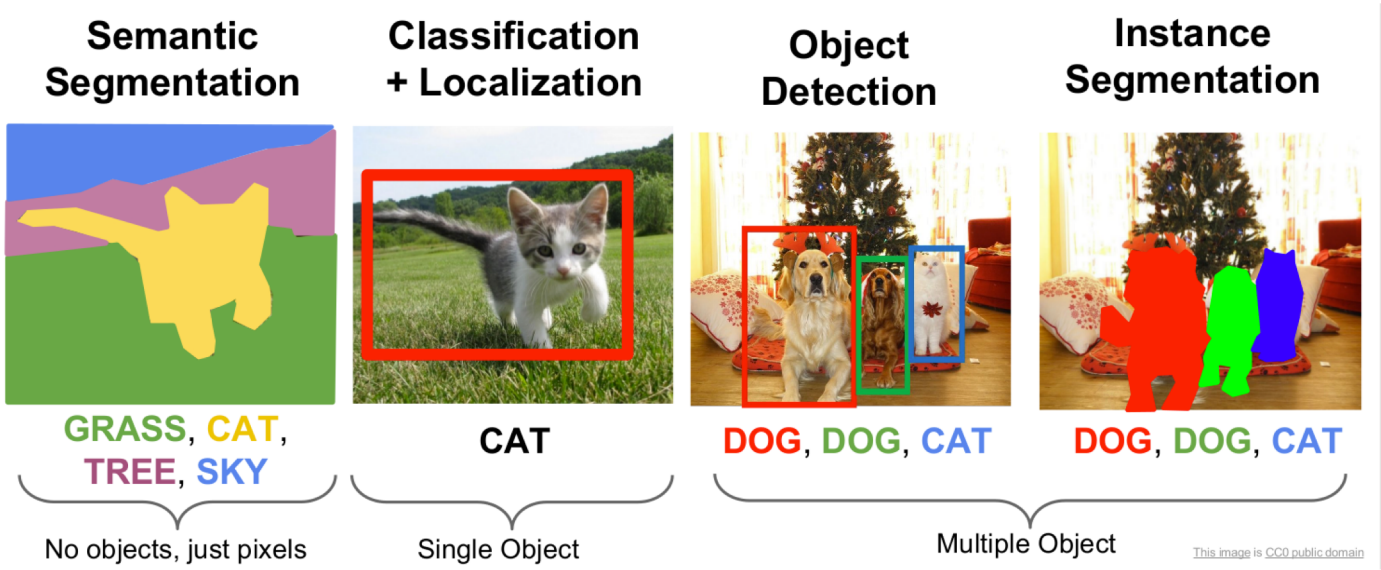
IBM CODE

Hardware Matters



IBM
CODE

Where is Deep Learning Being used today?



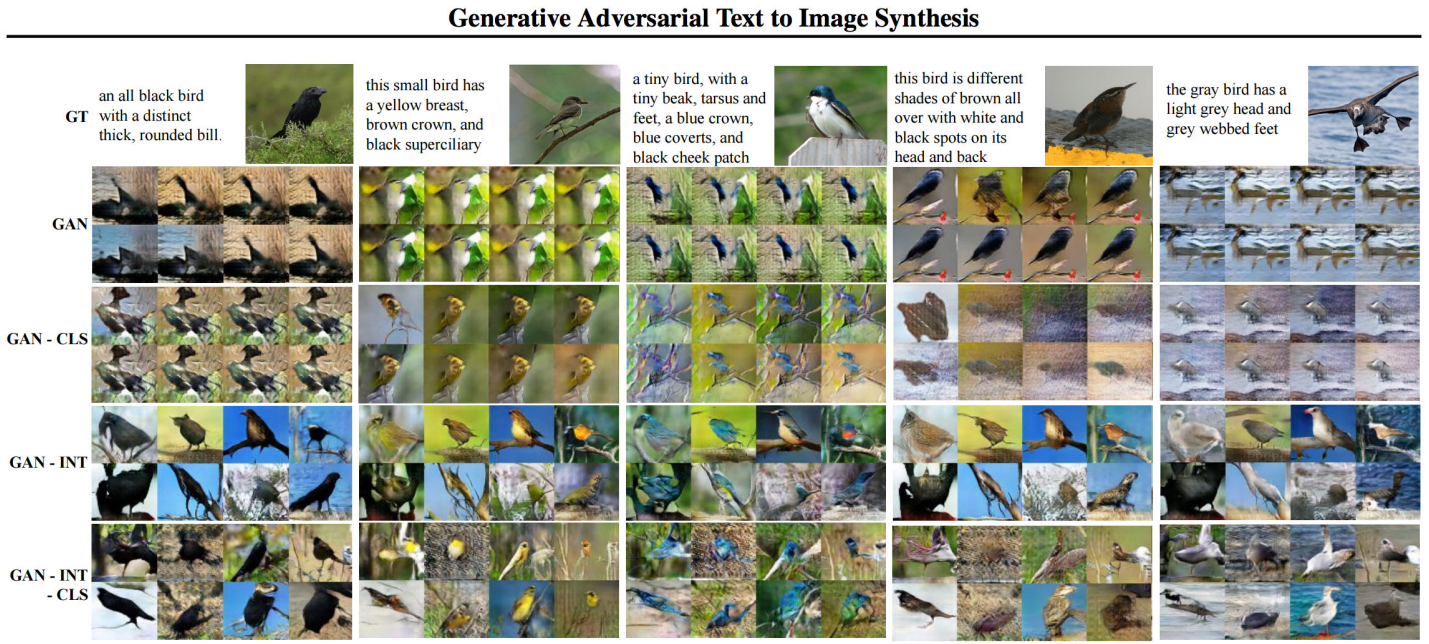
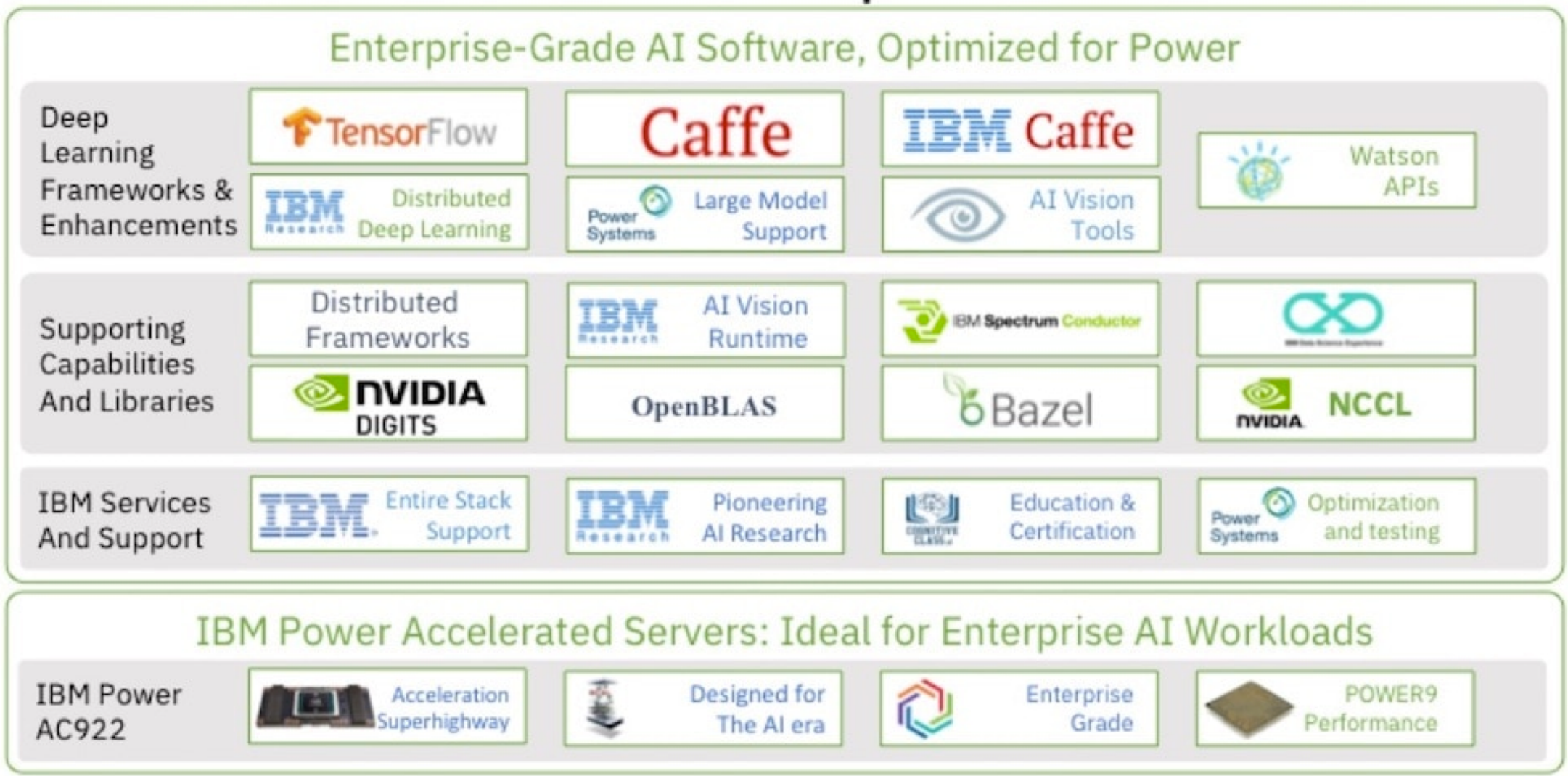


Figure 3. Zero-shot (i.e. conditioned on text from unseen test set categories) generated bird images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. We found that interpolation regularizer was needed to reliably achieve visually-plausible results.

IBM CODE

Tools of the trade

IBM PowerAI Enterprise Platform

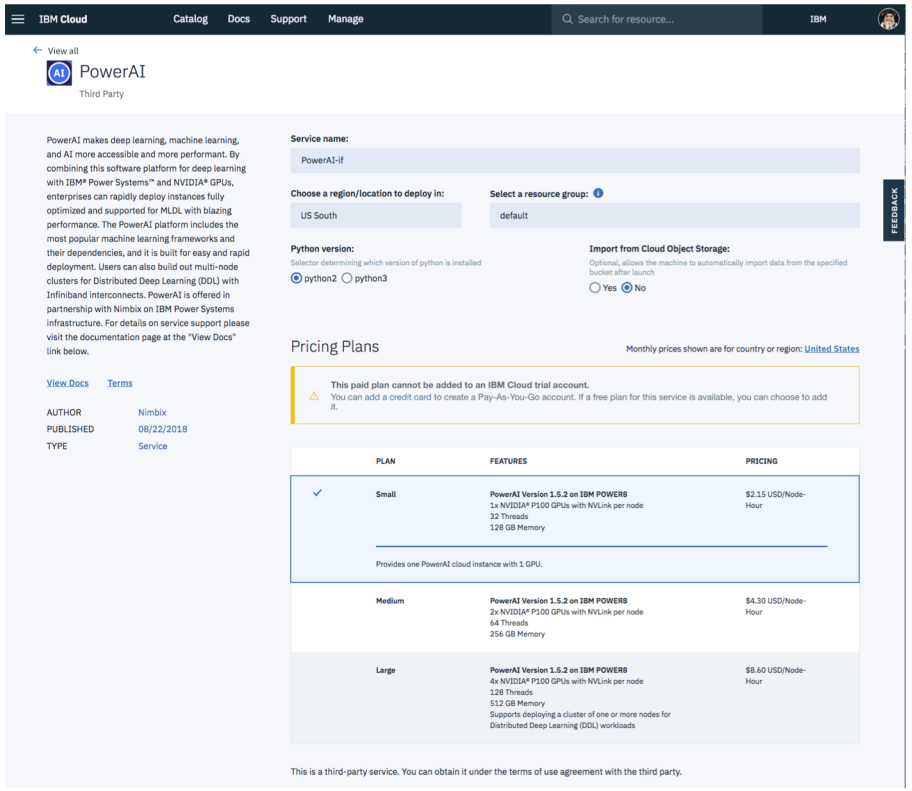


IBM CODE

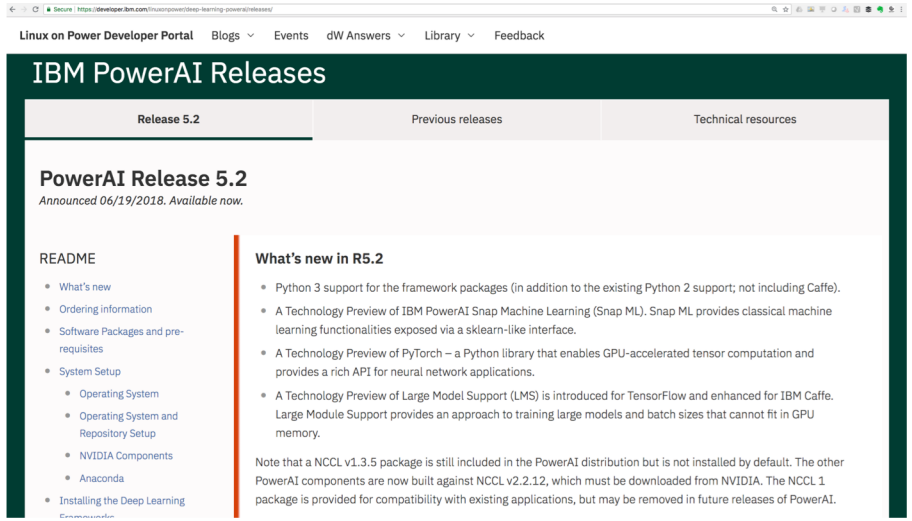
PowerAI

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

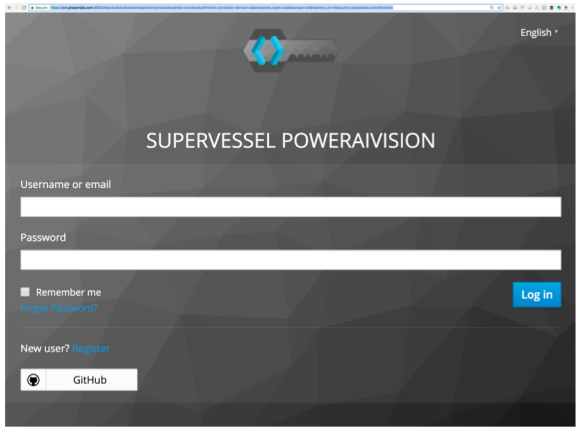
IBM Cloud Service
https://bluemix.net



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



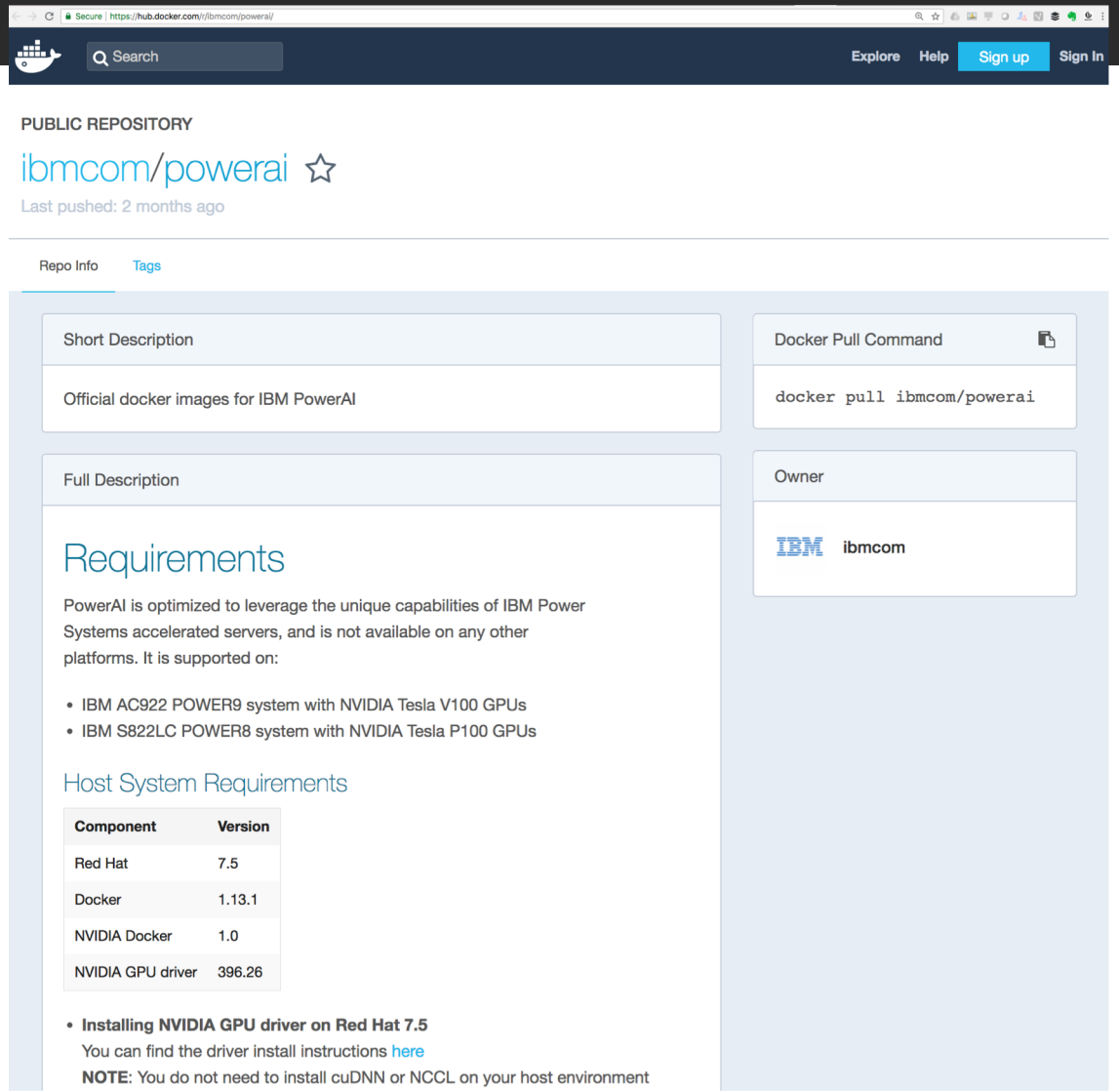
PowerAI Vision Preview
https://ibm.biz/powerai



IBM CODE

PowerAI Docker Container

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



IBM CODE

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
```

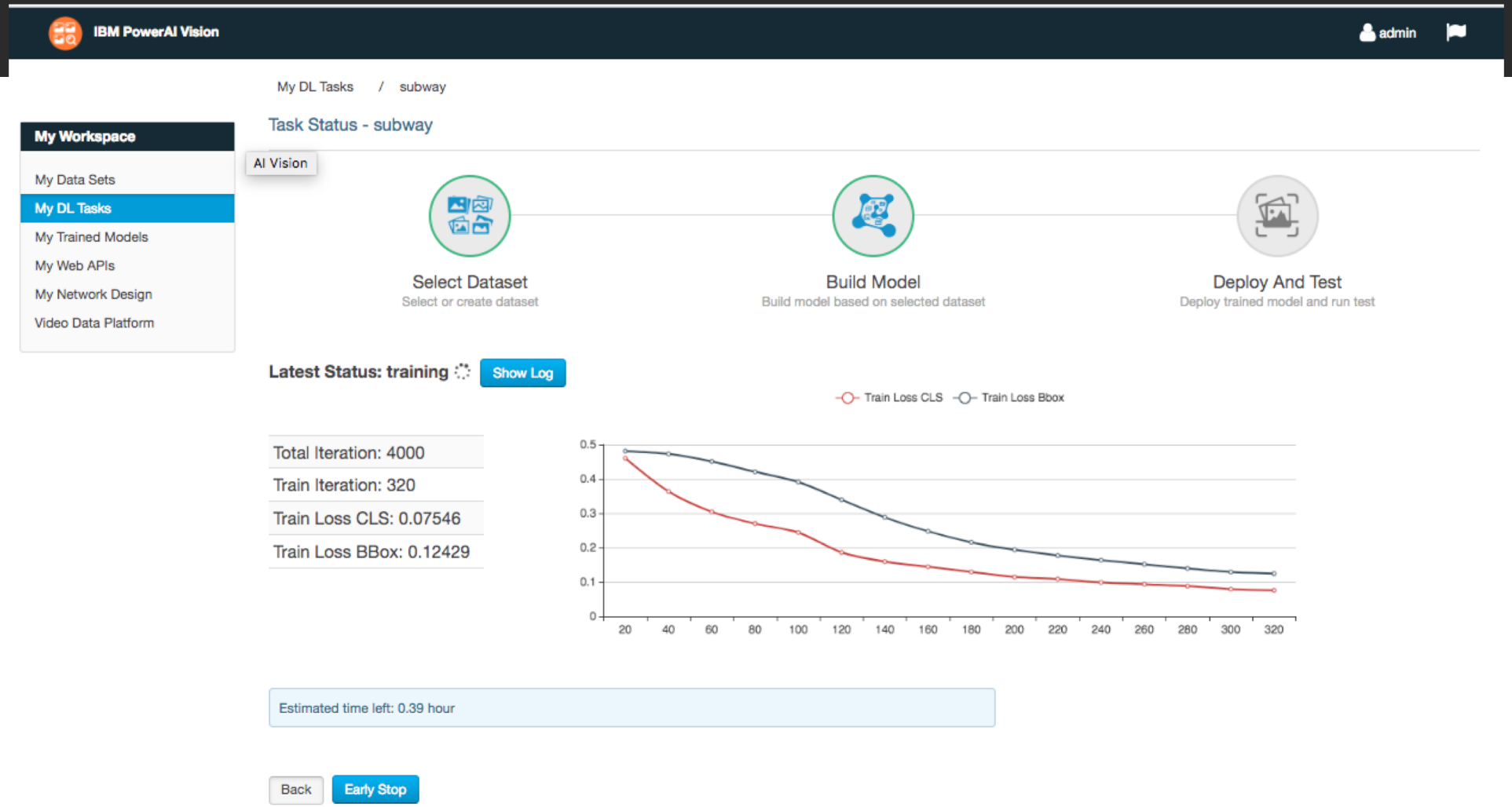
In [4]:
```
# Store layers weight & bias
weights = {
    # 5x5 conv, 1 input, 32 outputs
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    # fully connected, 7*7*64 inputs, 1024 outputs
    'wd1':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

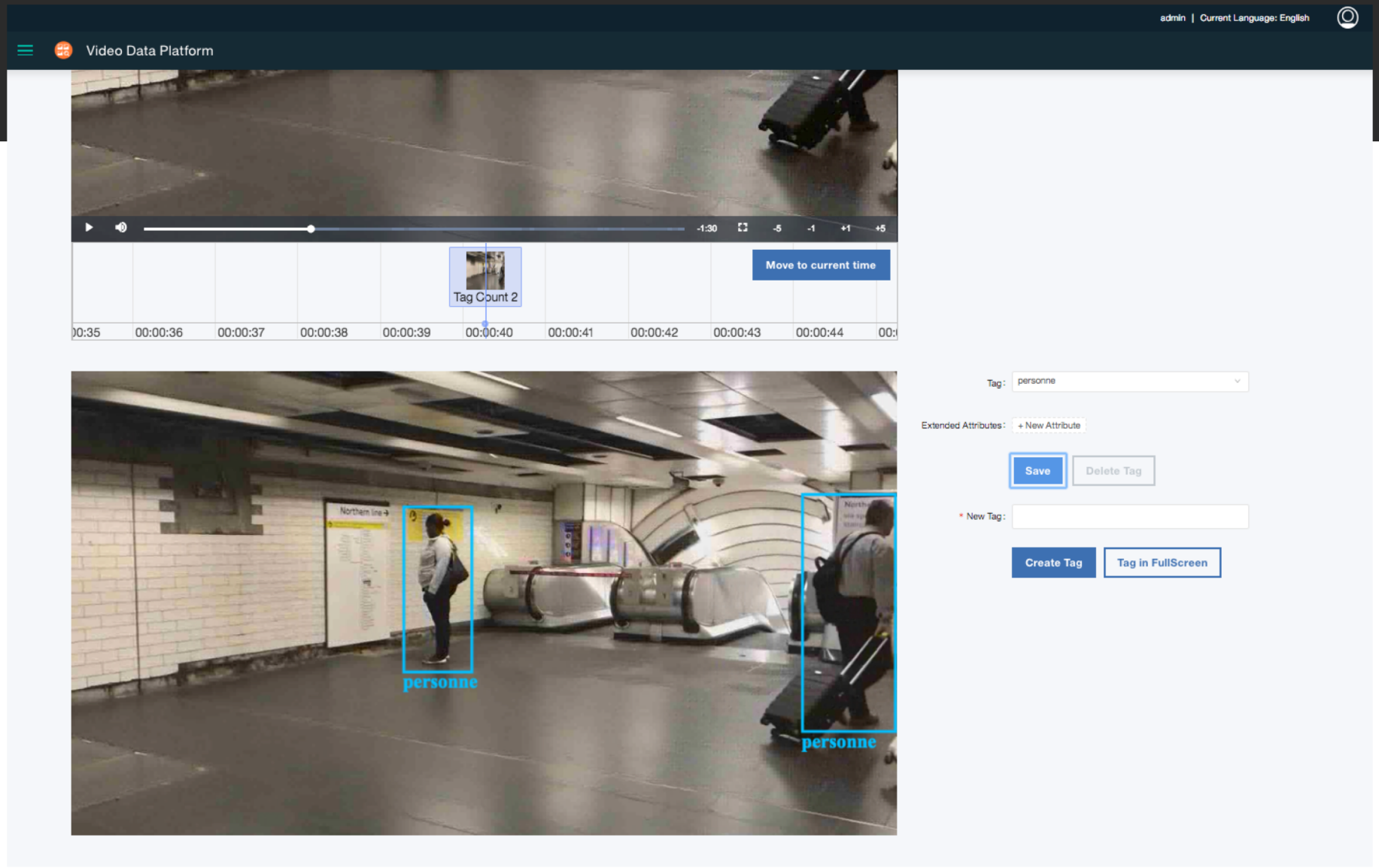
IBM CODE

PowerAI Vision Platform



IBM CODE

PowerAI Video Data Platform



IBM CODE

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

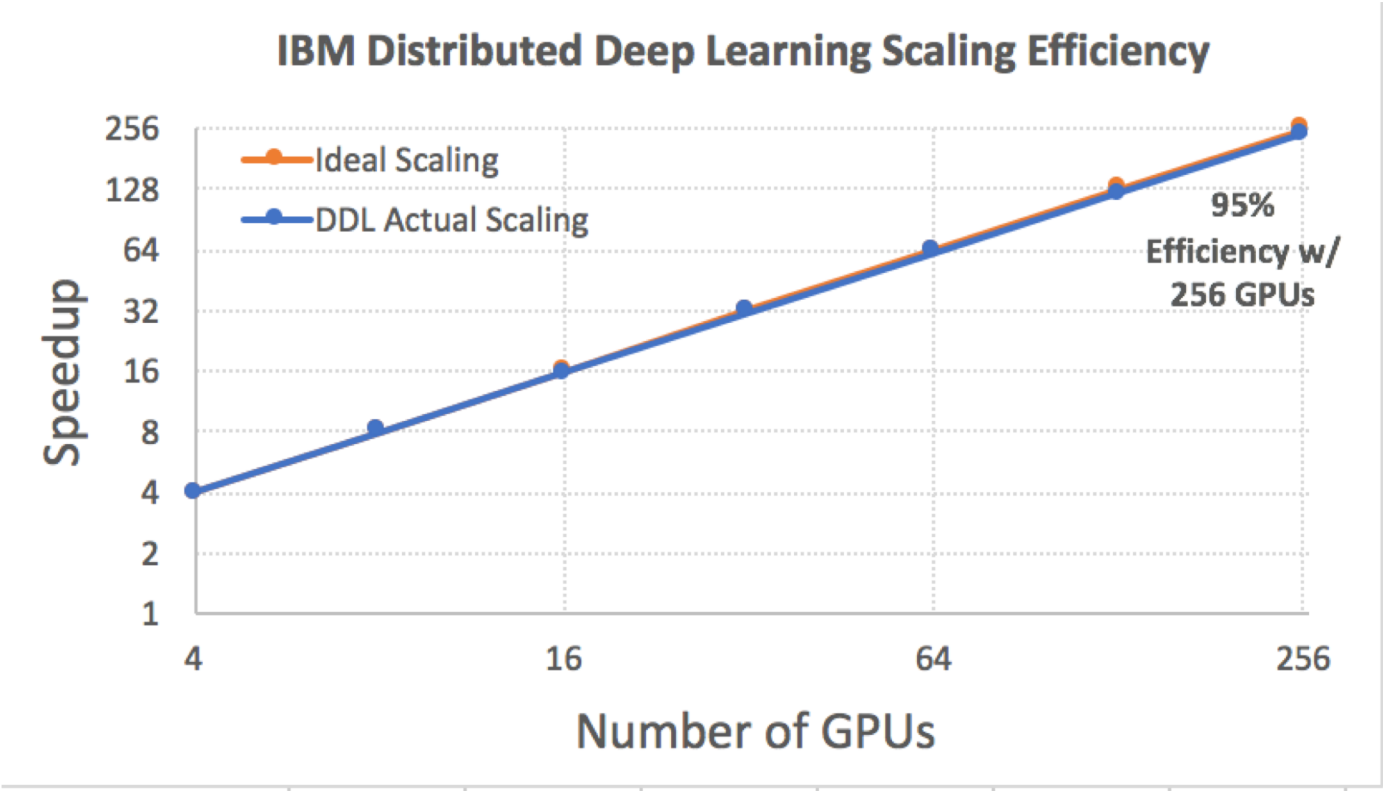
IBM CODE

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



IBM CODE