

Bringing an AI Ecosystem to the Domain Expert and Enterprise AI Developer

Fred Reiss

Vijay Bommireddipalli

IBM Center for Open-Source Data & AI Technologies
<http://codait.org>



IBM's history of strong AI leadership

1968, 2001: A Space Odyssey

- IBM was a technical advisor
- HAL is “the latest in machine intelligence”



1997: Deep Blue

- Deep Blue became the first machine to beat a world chess champion in tournament play



2011: Jeopardy!

- Watson beat two top Jeopardy! champions



2018: Open Tech, AI & emerging standards

- New IBM centers of gravity for AI
- OS projects increasing exponentially
- Emerging global standards in AI

Center for Open Source Data and AI Technologies

- CODAIT aims to make AI solutions dramatically easier to create, deploy, and manage in the enterprise
- Relaunch of the Spark Technology Center (STC) to reflect expanded mission

- CODAIT
- codait.org



- codait (French)
- = coder/coded
- <https://m.interglot.com/fr/en/codait>



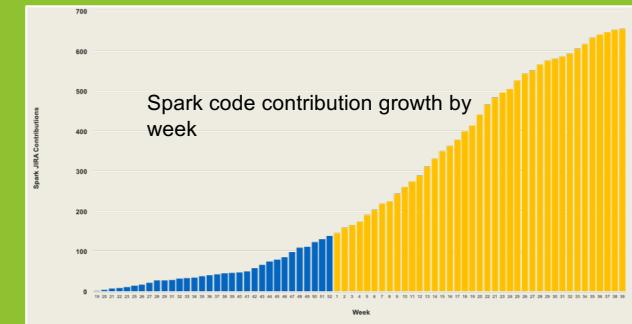
CODAIT by the numb3rs

- The team contributes to over 10 open source projects. These projects include - Spark, Tensorflow, Keras, SystemML, Arrow, Bahir, Toree, Livy, Zeppelin, R4ML, Stocator, Jupyter Enterprise Gateway
- 17 committers and many contributors in Apache projects- Spark, Arrow, systemML, Bahir, Toree, Livy
- Over 980 JIRAs and 50,000 lines of code committed to Apache Spark itself, and Over 65,000 LoC into SystemML
 - Established IBM as the number 1 contributor to Spark Machine Learning in Spark 2.0 release
- Over 25 product lines within IBM leveraging Apache Spark in some form or another. CODAIT engineers have interacted and interlocked with many of them.
- Speakers at over 100 conferences, MeetUps, un-conferences etc.

- CODAIT
- codait.org

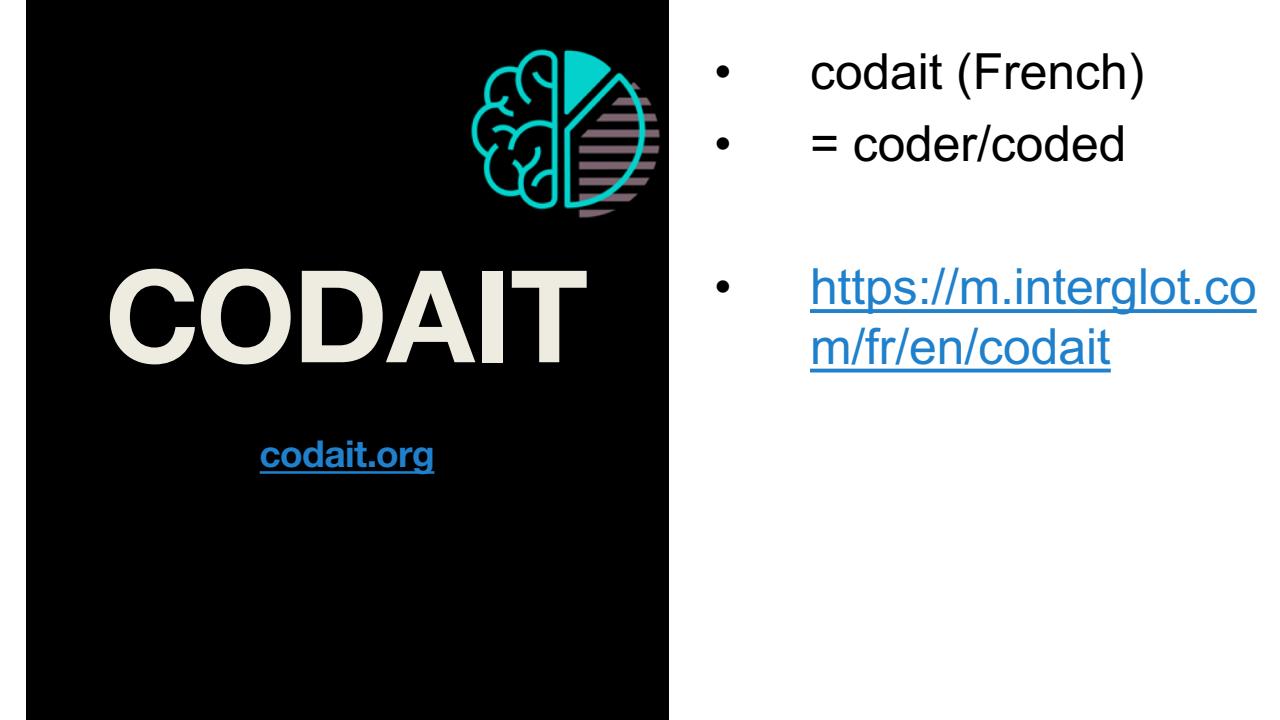


- codait (French)
- = coder/coded
- <https://m.interglot.com/fr/en/codait>

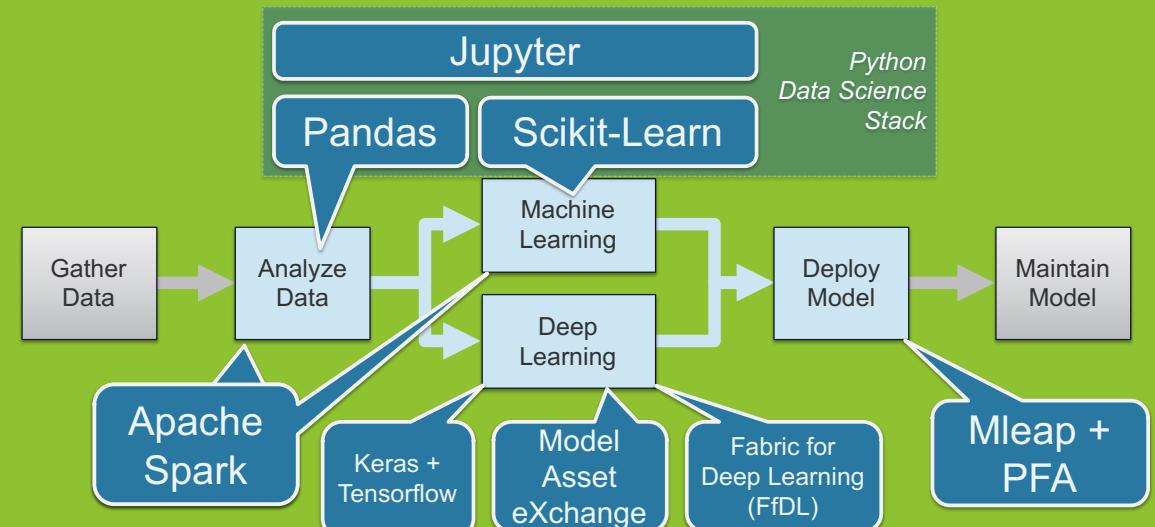


Center for Open Source Data and AI Technologies

- **Code** - Build and improve practical frameworks to enable more developers to realize immediate value (e.g. FfDL, Tensorflow Jupyter, Spark)
- **Content** – Showcase solutions to complex and real world AI problems
- **Community** – Bring developers and data scientists to engage with IBM (e.g. MAX)



Improving the Enterprise AI lifecycle in Open Source



Fabric for Deep Learning

<https://github.com/IBM/FfDL>

<https://www.youtube.com/watch?v=nQsYWmkfLP4>

- FfDL provides a scalable, resilient, and fault tolerant deep-learning framework
- Fabric for Deep Learning or FfDL (pronounced as ‘fiddle’) is an open source project which aims at making Deep Learning easily accessible to the people it matters the most i.e. Data Scientists, and AI developers.
- FfDL Provides a consistent way to deploy, train and visualize Deep Learning jobs across multiple frameworks like TensorFlow, Caffe, PyTorch, Keras etc.
- FfDL is being developed in close collaboration with IBM Research and IBM Watson. It forms the core of Watson’s Deep Learning service in open source.



Fabric for Deep Learning (FfDL)

Deep Learning Training, Monitoring and Management



Kubernetes – GPU/CPU/NFS Support

Cloud Hardware (GPUs and CPUs) SSD Backed NFS Volumes

6

Jupyter Enterprise Gateway



- A lightweight, multi-tenant, scalable and secure gateway that enables Jupyter Notebooks to share resources across an Apache Spark or Kubernetes cluster for Enterprise/Cloud use cases

data science security notebook
Spark kernel scalable multi-user available enterprise
Jupyter gateway remote cluster

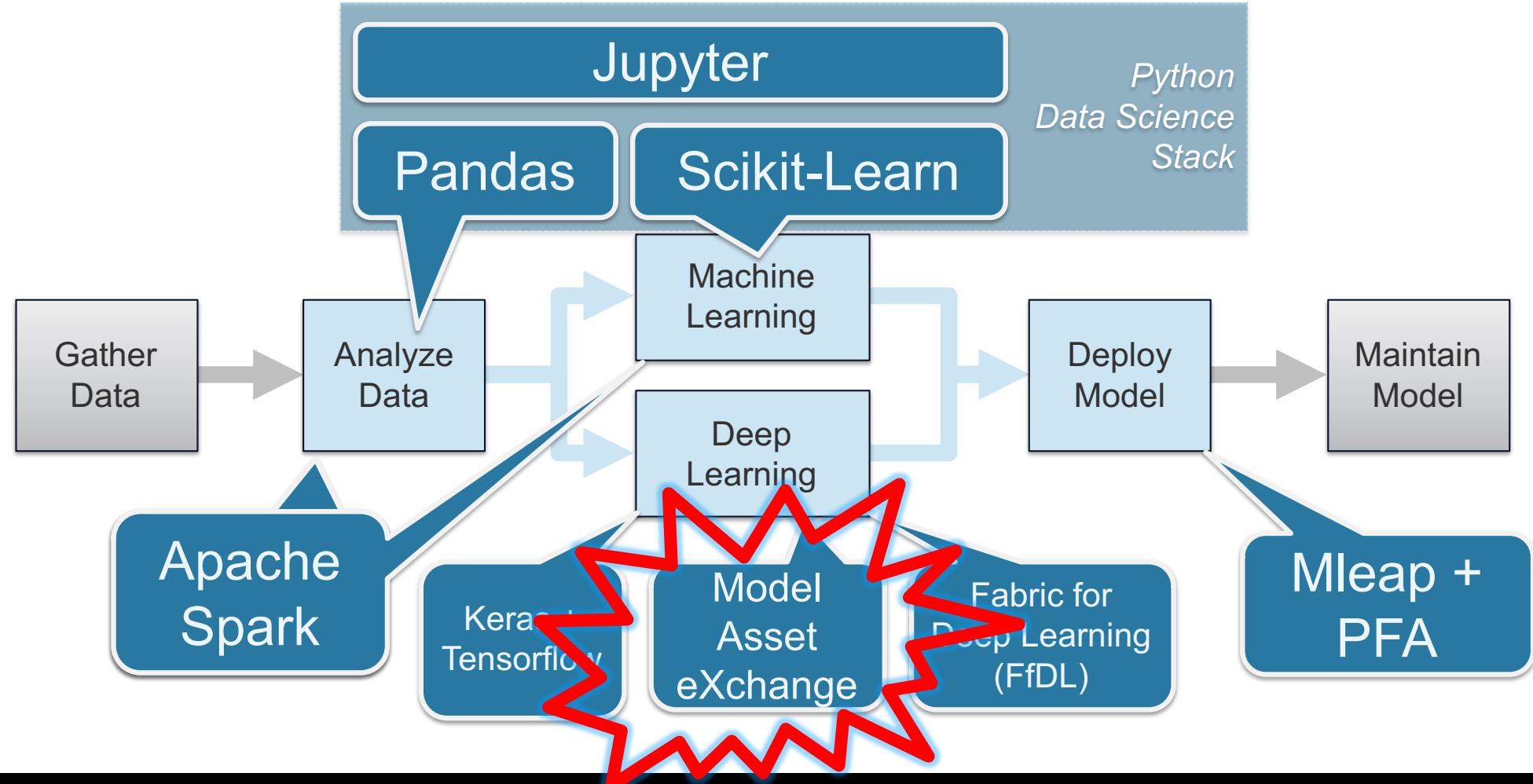


- Jupyter Enterprise Gateway at IBM Code
 - <https://developer.ibm.com/code/openprojects/jupyter-enterprise-gateway/>
-  Jupyter Enterprise Gateway source code at GitHub
 - https://github.com/jupyter-incubator/enterprise_gateway
- Jupyter Enterprise Gateway Documentation
 - <http://jupyter-enterprise-gateway.readthedocs.io/en/latest/>

Road Map

- Background: Deep Learning Models
- The IBM Code Model Asset Exchange
- Demo
- What's Next

CODAIT: Enabling End-to-End AI in the Enterprise





Making AI as Ubiquitous as the Telephone

This talk is about enabling **domain experts** to use **deep learning** in the enterprise.

Q: What is deep learning?

A: Machine learning using **deep neural networks**.

Q: What is a deep neural network?

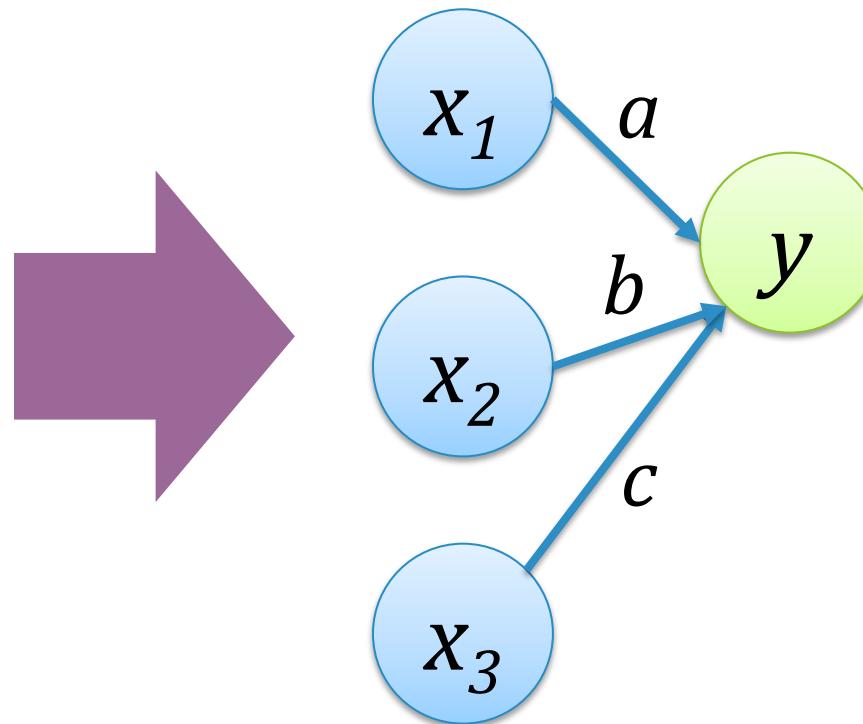
A: A neural network with **multiple hidden layers**.

Q: What is a **neural network**?

What is a neural network?

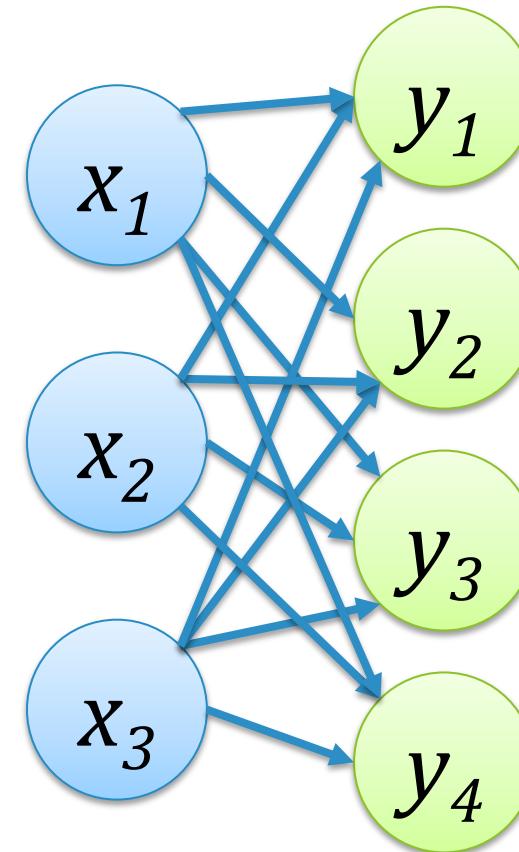
$$y = ax_1 + bx_2 + cx_3$$

Linear regression

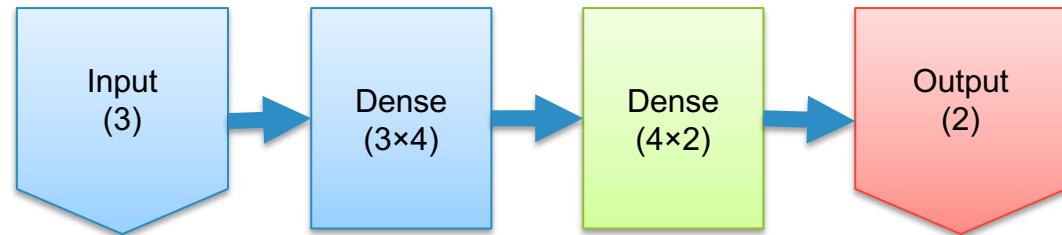


What is a neural network?

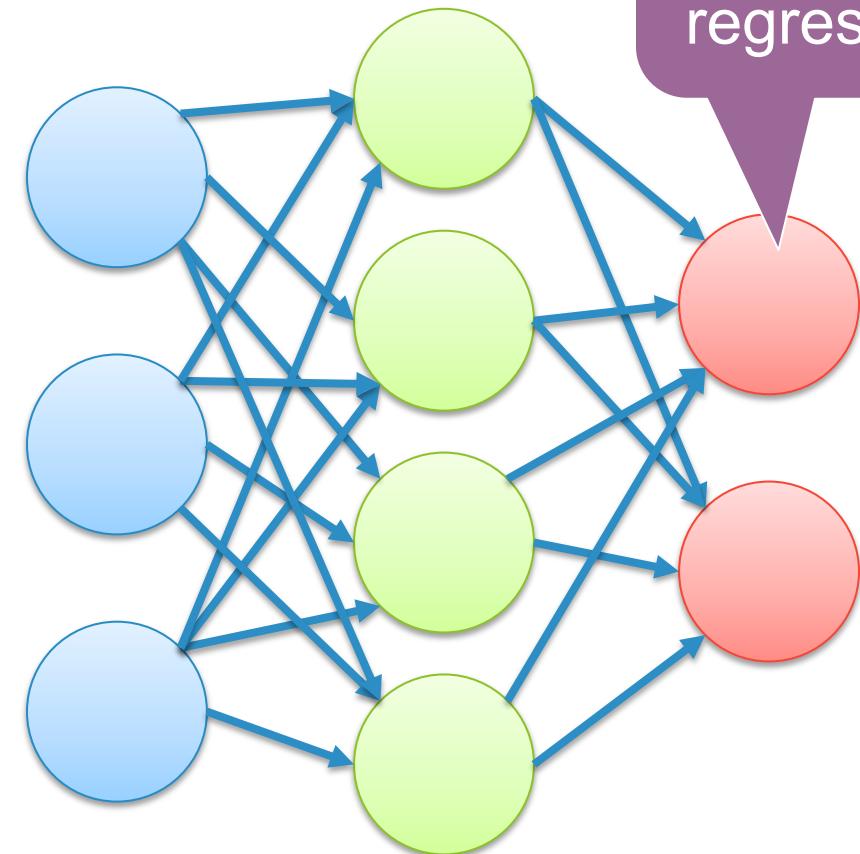
Multiple linear regressions at the same time



What is a neural network?



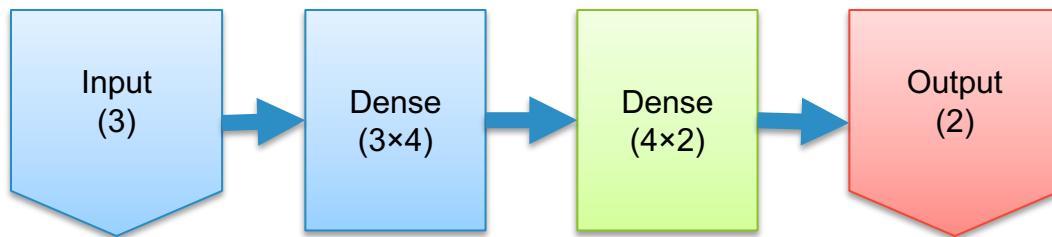
Same network in a more compact notation



Multilayer Perceptron
Neural Network

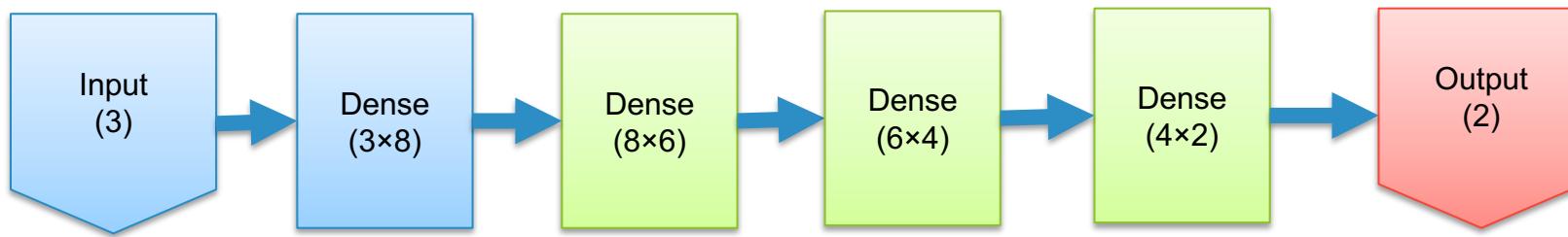
Q: What is a **deep neural network?**

A: A neural network with **multiple hidden layers.**



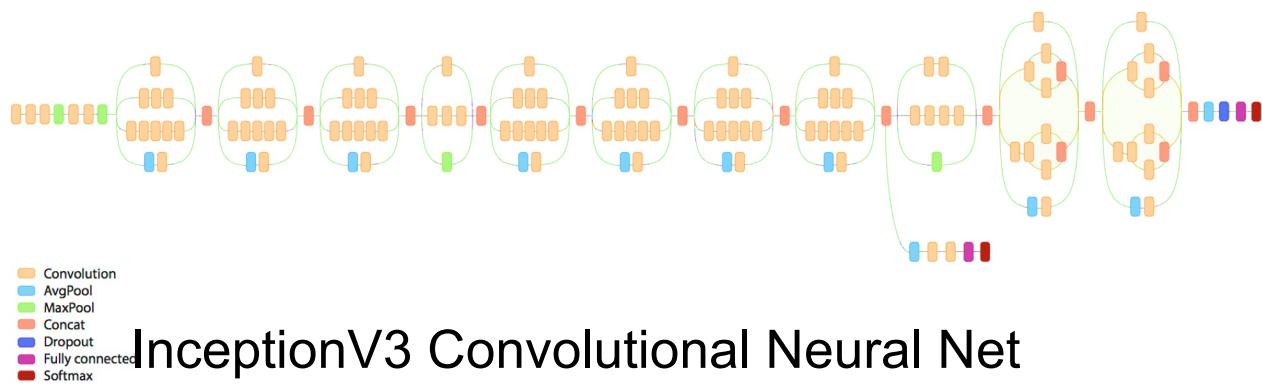
Q: What is a **deep neural network?**

A: A neural network with **multiple hidden layers.**



Q: What is deep learning?

A: Machine learning using deep neural networks.



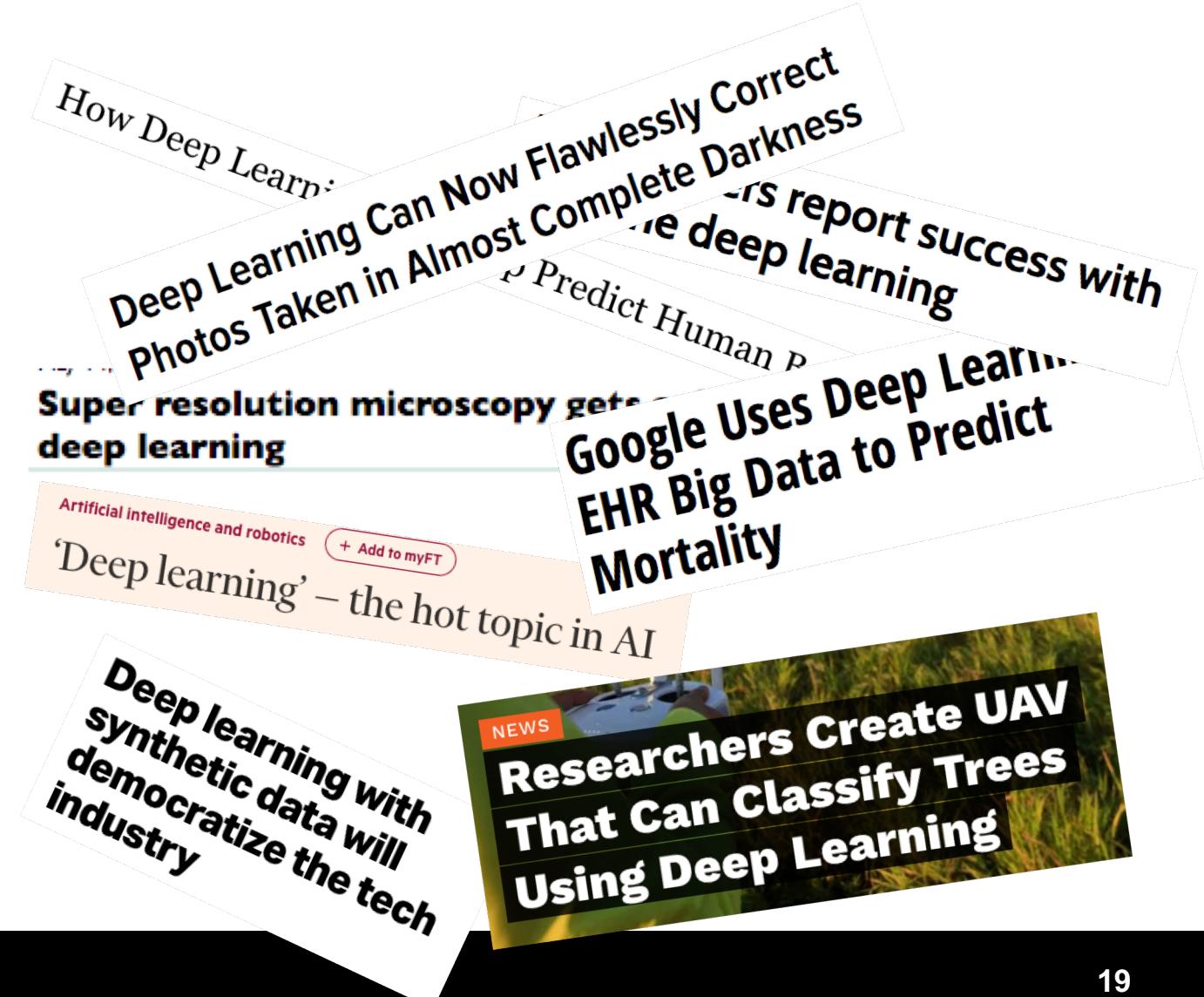
InceptionV3 Convolutional Neural Net
(A “medium-sized” deep learning model)

Image Source:

https://github.com/tensorflow/models/blob/master/research/inception/g3doc/inception_v3_architecture.png

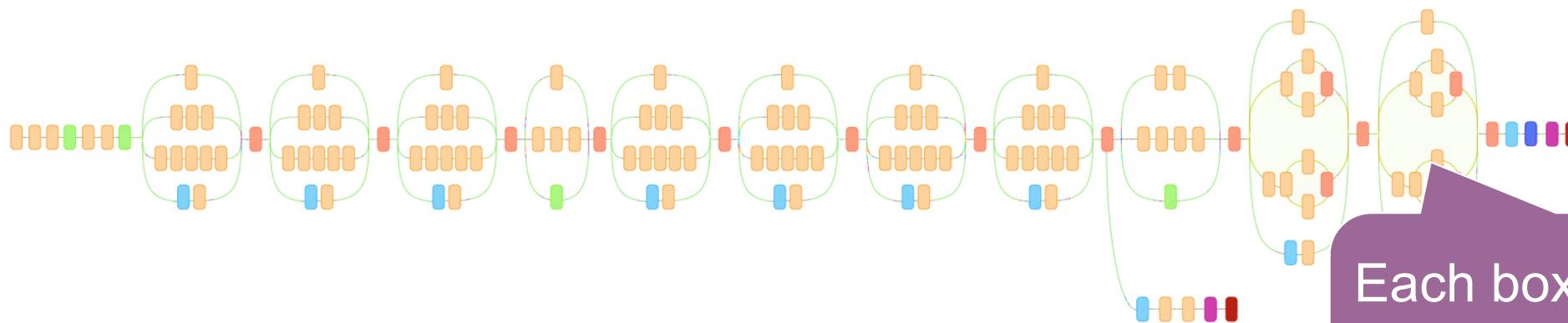
Characteristics of Deep Learning (1)

- State-of-the-Art prediction quality in many domains
 - Image classification
 - Machine translation
 - Facial recognition
 - Time series prediction
 - Many more



Characteristics of Deep Learning (2)

- Large, complex models
 - Model size generally determined by “how big a model can you fit on your device?”



Legend:
Orange square: Convolution
Blue square: AvgPool
Green square: MaxPool
Red square: Concat
Purple square: Dropout
Pink square: Fully connected
Dark red square: Softmax

InceptionV3 Convolutional Neural Net
(A “medium-sized” deep learning model)

Each box ≈ between
32 and 768 linear
regression models

Characteristics of Deep Learning (3)

Poorly understood today
...even by experts

- Why do the models converge?
- Why do the models converge with low loss?
- Why do the models generalize?

While these results are encouraging, questions remain on the exact dynamics at play.

Shake-Shake regularization

Xavier Gastaldi
xgastaldi.mba2011@london.edu

Abstract

The method introduced in this paper aims at helping deep learning practitioners faced with an overfit problem. The idea is to replace, in a multi-branch network, the standard summation of parallel branches with a stochastic affine combination. Applied to 3-branch residual networks, shake-shake regularization improves on the best single shot published results on CIFAR-10 and CIFAR-100 by reaching test errors of 2.86% and 15.85%. Experiments on architectures without skip connections or Batch Normalization show encouraging results and open the door to a large set of applications. Code is available at <https://github.com/xgastaldi/shake-shake>.

6 Conclusion

A series of experiments seem to indicate an ability to combat overfit by decorrelating the branches of multi-branch networks. This method leads to state of the art results on CIFAR datasets and could

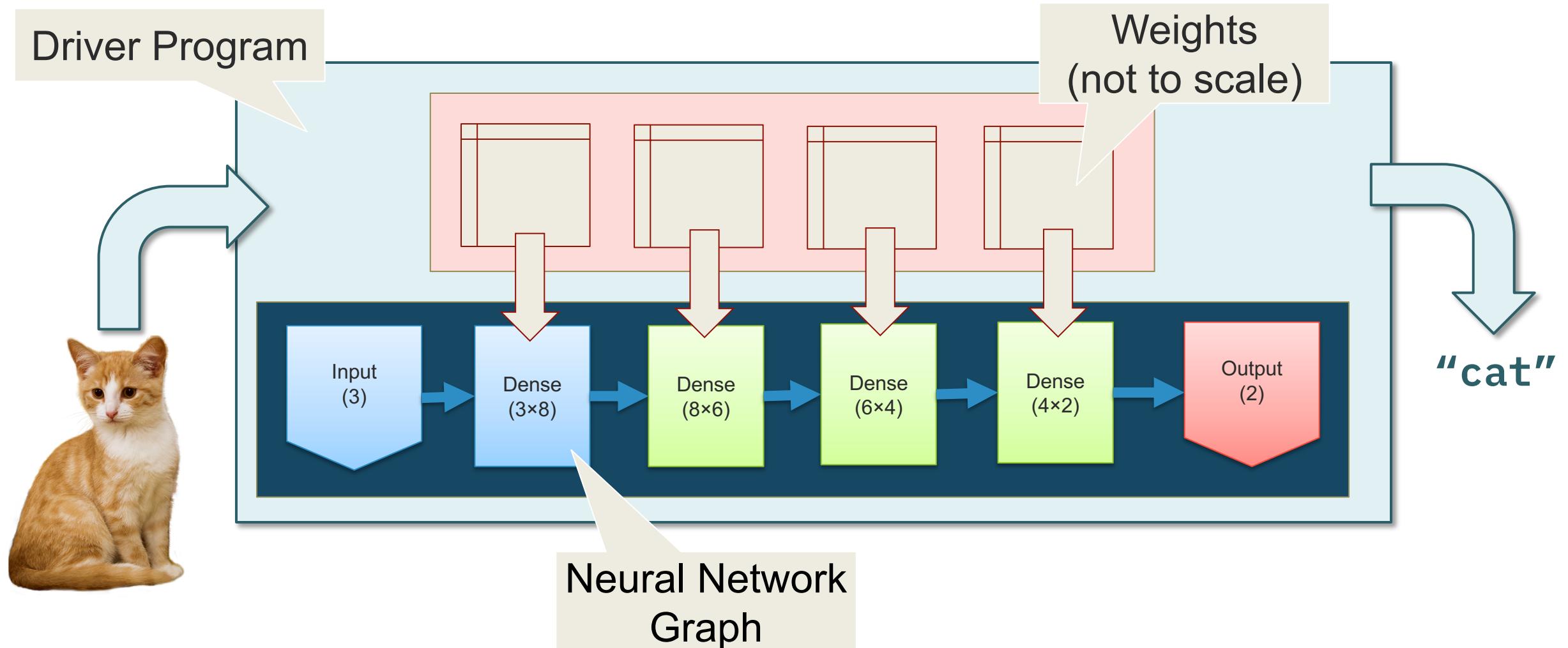
Focus of this Talk

Incorporating **well-understood** deep learning models into enterprise applications.



Sounds easy!

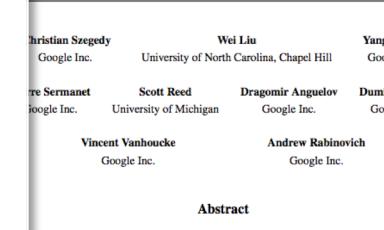
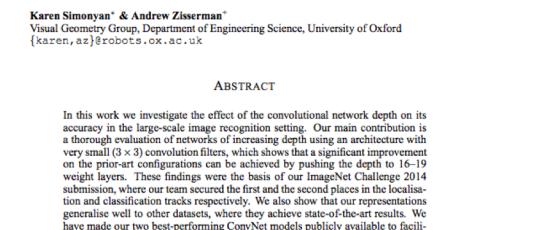
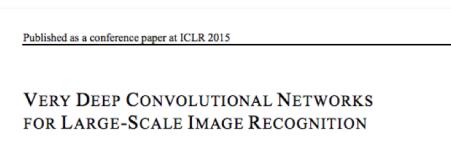
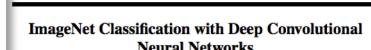
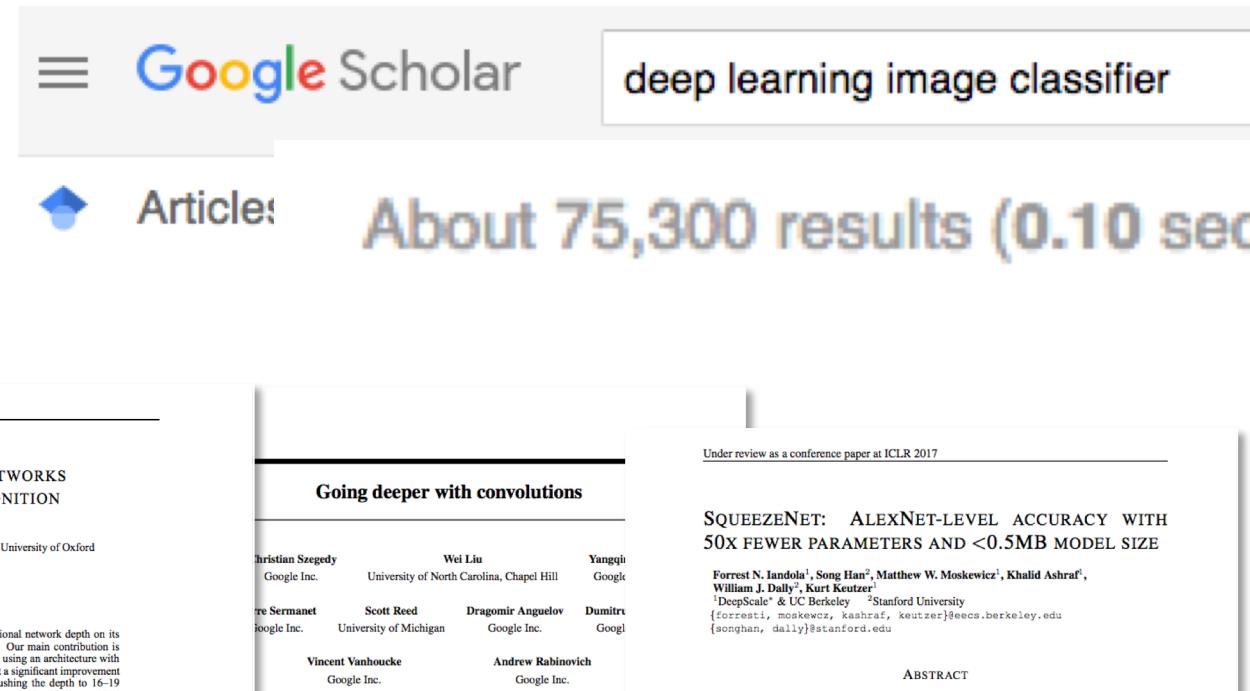
The Parts of a Deep Learning Model



Example: Get an Image Classifier

Step 1: Find a suitable neural network graph.

- Need to read some papers



SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

Forrest N. Iandola¹, Song Han², Matthew W. Moskewicz¹, Khalid Ashraf¹, William J. Dally³, Kurt Keutzer⁴
¹DeepScale² & UC Berkeley ³Stanford University
⁴{forrestn, moskewcz, kashraf, keutzer}@eecs.berkeley.edu
{songhan, dally}@stanford.edu

ABSTRACT

Recent research on deep convolutional neural networks (CNNs) has focused primarily on improving accuracy. For a given accuracy level, it is typically possible to identify multiple CNN architectures that achieve that accuracy level. With equivalent performance, smaller CNN architectures offer at least three advantages: (1) Smaller CNNs require less communication across servers during distributed training. (2) Smaller CNNs require less bandwidth to export a new model from the cloud to an autonomous car. (3) Smaller CNNs are more feasible to deploy on FPGAs and other hardware with limited memory. To provide all of these advantages, we propose a small CNN architecture called SqueezeNet. SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. Additionally, with model compression techniques, we are able to compress SqueezeNet to less than 0.5MB (510x smaller than AlexNet). The SqueezeNet architecture is available for download here: <https://github.com/DeepScale/SqueezeNet>

1 INTRODUCTION AND MOTIVATION

Most of the recent research on deep convolutional neural networks (CNNs) has focused on increasing accuracy on computer vision datasets. For a given accuracy level, there typically exist multiple CNN architectures that achieve that accuracy level. Given equivalent accuracy, a CNN architecture with fewer parameters has several advantages:

- **More efficient distributed training.** Communication among servers is the limiting factor to the scalability of distributed CNN training. For distributed data-parallel training, communication overhead is directly proportional to the number of parameters in the model (Janisch et al., 2016). In short, smaller models train faster due to requiring less communication.
- **Less overhead when exporting new models to clients.** For autonomous driving, companies such as Tesla periodically copy new models from their servers to drivers' cars. This practice can be refined by only updating the necessary components. Requirements for that the safety of Tesla's Autopilot semi-autonomous driving functionality has incrementally improved with respect over the six months (Consumer Reports, 2016). However, over-th

Example: Get an Image Classifier

Step 2: Find code to generate the neural network graph

```

100 self.bottleneck = bottleneck
101 # bottleneck
102 if self._version == 1:
103     self.block_fn = _bottleneck_block_v1
104     strides = []
105     self.block_fn = _bottleneck_block_v2
106 else:
107     self._version == 2
108     self.block_fn = _building_block_v2
109     strides = [1, 2, 2]
110
111 def _block_fn(self, inputs, filters, training,
112             projection_shortcut, kernel_size3, stride0,
113             kernel_size2, kernel_size1, strides,
114             data_format):
115
116     """Applies standard residual block with 3 convolution layers.
117
118     This function is a custom
119     same signature as tf.layers.conv2d()
120     but has extra short cuts and
121     variable_scope. Then
122     returns the output of the
123     block.
124
125     The variable_scope
126     then
127     gets passed to
128     the types of variables
129     get
130     passed to
131     the
132     'beta'
133     parameter
134     beta
135     is
136     variable with
137     'ptrtrn'
138
139     This custom get will
140     (e.g. float32) variable
141
142     Creates variables in
143     the
144     scope.
145
146     def _building_block_v2(inputs, filters, training, projection_shortcut, strides,
147                           data_format):
148
149     """Single block for ReLU v2, without a bottleneck.
150
151     Batch normalization then ReLu then convolution as described by:
152
153     Identity Mapping in Deep Residual Networks
154     https://arxiv.org/pdf/1603.05027.pdf
155     by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Jul 2016.
156
157     Args:
158         inputs: A tensor of size [batch, channels, height_in, width_in] or
159                 [batch, height_in, width_in, channels] depending on data_format.
160         filters: The number of filters for the convolution.
161         training: A Boolean for whether the model is in training or inference
162                 mode. Needed for batch normalization.
163         projection_shortcut: The function to use for projection shortcuts
164                 (typically a 1x1 convolution when reducing channels)
165         kernel_size3: Stride of the first layer. If greater than 1, this block will ultimately
166                     downsample the input.
167         kernel_size2: The kernel for "channels_last" or "channels_first".
168         kernel_size1: The kernel for "channels_last" or "channels_first".
169
170     Returns:
171         The output tensor of the block, shape should match inputs.
172
173     ---
```

```

    ---Creates one layer blocks for the model.
    312
    313    Arg:
    314        inputs: A tensor or list [batch, channels, height_in, width_in] or
    315        [batch, height_in, width_in, channel] depending on data_format.
    316        filters: The number of filters for each output channel of the layer.
    317        bottleneck_fn: The block to use instead of the default "building_block"
    318        block_fn: The block to use within the model, either "building_block" or
    319        "bottleneck".
    320        blocks: The number of blocks contained in the layer.
    321        The layer will be trained to completion if the value of the layer.
    322        greater than zero. If set to None, the layer will always return training.
    323        training: Either True or False, whether we are currently training the
    324        layer.
    325        name: A string name for the tensor output of the black layer.
    326        data_format: The input format ("channels_last" or "channels_first").
    327
    328    Returns:
    329        --- The output tensor of the block layer.
    330
    331    # Bottlenecks blocks end with the number of filters as they start with
    332    # filters_out = filters + (filters_out - filters) * (blocks - 1)
    333
    334    def projection_shortcut(inputs):
    335        return conv2d_fixed_padding(
    336            inputs=inputs, filters=filter_size_out, kernel_size=1, strides=strides,
    337            data_format=data_format)
    338
    339    # Only the first block per block_layer uses projection_shortcut and strides
    340    # are set to 1, all other blocks have stride = 1.
    341    # Inputs = [batch, height, width, channel], training, projection_shortcut, strides,
    342    # data_format
    343
    344    for _ in range(blocks):
    345        inputs = block_fn(inputs, filters, training, None, 1, data_format)
    346
    347    return tf.identity(inputs, name)
    348
    349
    350    shortcut = inputs
    351
    352    # If projection_shortcut is not None
    353    # shortcut = projection_shortcut(inputs)
    354    shortcut = conv2d_fixed_padding(
    355        inputs=inputs, filters=filter_size_out, kernel_size=1, strides=1,
    356        data_format=data_format)
    357
    358    inputs = conv2d_fixed_padding(
    359        inputs=inputs, filters=filter_size_out, kernel_size=1, strides=1,
    360        data_format=data_format)
    361
    362    inputs = tf.nn.relu(shortcut)
    363    inputs += shortcut
    364
    365    inputs = conv2d_fixed_padding(
    366        inputs=inputs, filters=filter_size_out, kernel_size=1, strides=1,
    367        data_format=data_format)
    368
    369    inputs = tf.nn.relu(inputs)
    370    inputs += shortcut
    371
    372    inputs = tf.nn.relu(inputs)
    373
    374
    375    return inputs
    376
    377
    378    def _bottleneck_block(inputs, filters, training, projection_shortcut,
    379        filter_size_out, kernel_size, strides, data_format, name):
    380        """A single block for ResNet v2, without a bottleneck.
    381
    382        Similar to _building_block_v1, except using the "bottleneck" blocks
    383        described in [1].
    384        [1] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition.
    385        https://arxiv.org/pdf/1512.03385.pdf
    386        by Kaiming He, Xiangyu Zhang, Shengxiong Ren, and Jian Sun, Dec 2015.
    387
    388        Adapted to the ordering conventions of
    389        https://arxiv.org/pdf/1701.04862.pdf
    390        Identity Mappings in Deep Residual Networks
    391
    392
```

```

    Returns:
        A logits Tensor with shape [batch_size, self.num_classes].
    
```

```

    with self._model_variable_scopes():
        # Set data_format = "channels_first".
        # This provides a large performance boost on GPU. See
        # https://www.tensorflow.org/performance/performance_guide#data_formats
        inputs = tf.convert_to_tensor(inputs, dtype=dtype, name="x")
        strides = [1, 1, 1, 1]
        kernel_size = [1, 1, 1, 1]
        padding = "VALID"
        dilation_rate = [1, 1, 1, 1]
        data_format = "NHWC"
        kernel_size = [1, 1, 1, 1]
        strides = [1, 1, 1, 1]
        padding = "VALID"
        dilation_rate = [1, 1, 1, 1]
        data_format = "NHWC"
        inputs = tf.cast(inputs, tf.float32)
        inputs = tf.reshape(inputs, [-1, 1, 1, 1])
        inputs = tf.nn.conv2d(
            inputs=inputs,
            filters=conv_filters,
            kernel_size=kernel_size,
            strides=strides,
            data_format=data_format,
            padding=padding,
            dilations=dilation_rate,
            name="conv1")
        inputs = tf.nn.relu(inputs, name="relu1")

        if self._use_max_pooling:
            inputs = tf.nn.max_pool(
                inputs=inputs,
                pool_size=[1, 1, 1, 1],
                stride=[1, 1, 1, 1],
                padding="VALID",
                data_format="NHWC",
                name="max_pool1")
            inputs = tf.nn.conv2d(
                inputs=inputs,
                filters=maxpool_filters,
                kernel_size=kernel_size,
                strides=strides,
                data_format=data_format,
                padding=padding,
                dilations=dilation_rate,
                name="conv2")
            inputs = tf.nn.relu(inputs, name="relu2")

        for i, num_blocks in enumerate(unrolled["block_sizes"]):
            num_filters = unrolled["num_filters"][(2*i)+1]
            inputs = tf.layers.conv2d(
                inputs=inputs,
                filters=num_filters,
                kernel_size=3,
                strides=1,
                padding="same",
                data_format="NHWC",
                activation=tf.nn.relu,
                name="conv%d" % (2*i+1))
            inputs = tf.layers.max_pooling2d(
                inputs=inputs,
                pool_size=2,
                strides=2,
                data_format="NHWC",
                name="max_pool%d" % (2*i+1))

        # Only apply the BN and Relu for model that does pre_activation in each
        # building blocks, block or resnet V2.
        if not self._pre_activation:
            inputs = tf.layers.batch_normalization(
                inputs=inputs,
                training=self.data_format == "NHWC",
                name="bn1")
            inputs = tf.nn.relu(inputs, name="relu3")

        # Args:
        #   inputs: A tensor of size [batch, channels, height, width] as
        #   defined by the data_format argument on data_format.
        #   kernel_size: The size of the convolutional kernel.
        #   strides: The stride of the convolution.
        #   padding: A Boolean for whether the model is in training or inference
        #   mode. Needed for batch normalization.
        #   project_shortcut: Whether to project the input before projection shortcuts
        #   (typically a 1x1 convolution when downscaling the input).
        #   shortcut: The residual block. If greater than 1, this block will utilize
        #   downsample the input.
        #   data_format: The input format ("channels_last" or "channels_first").
        #   name: The name of the layer.
        # Returns:
        #   The output tensor of the block; shape should match inputs.
        #   -->
        shortcut = inputs
        inputs = tf.layers.conv2d(
            inputs=inputs,
            filters=unrolled["num_filters"], training, data_format,
            strides=1, padding="same", data_format="NHWC",
            name="conv%d" % (2*i+2))
        inputs = tf.layers.batch_normalization(
            inputs=inputs,
            training=training,
            name="bn%d" % (2*i+2))
        inputs = tf.nn.relu(inputs, name="relu4")

        if project_shortcut:
            if project_shortcut > 1:
                shortcut = project_shortcut(shortcut)
            else:
                shortcut = tf.layers.conv2d(
                    inputs=inputs,
                    filters=filter_size,
                    kernel_size=1,
                    strides=1,
                    data_format="NHWC",
                    name="conv%d" % (2*i+3))
                shortcut = tf.layers.batch_normalization(
                    inputs=shortcut,
                    training=training,
                    name="bn%d" % (2*i+3))
                shortcut = tf.nn.relu(shortcut, name="relu5")

        inputs = tf.layers.conv2d(
            inputs=inputs,
            filters=filter_size,
            kernel_size=1,
            strides=1,
            data_format="NHWC",
            name="conv%d" % (2*i+4))
        inputs = tf.layers.batch_normalization(
            inputs=inputs,
            training=training,
            name="bn%d" % (2*i+4))
        inputs = tf.nn.relu(inputs, name="relu6")

        if project_shortcut:
            inputs = inputs + shortcut
        else:
            inputs = inputs + shortcut * 0.33333333333333337

        return inputs + shortcut

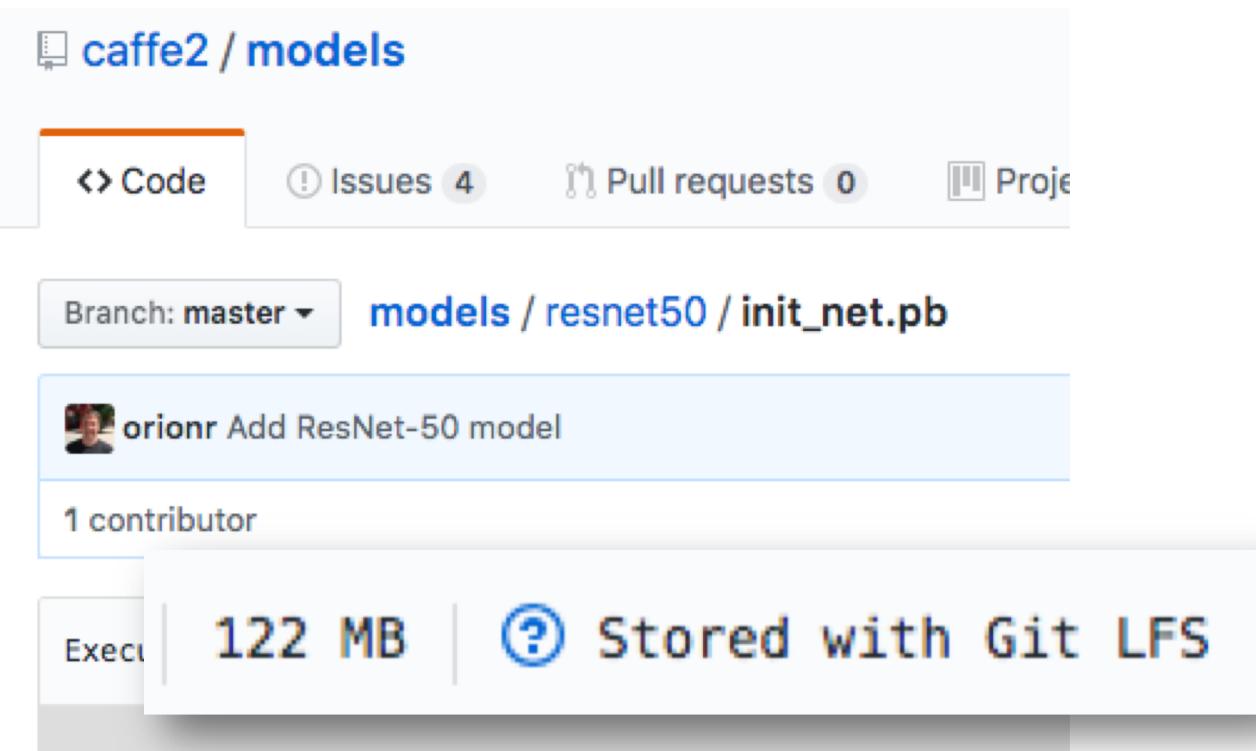
    def block_layer(inputs, filters, bottleneck, block_fn, blocks,

```

TensorFlow code to build ResNet50 neural network graph

Example: Get an Image Classifier

Step 3: Find some pre-trained weights for your graph



Caffe2 ResNet50 model weights*

Example: Get an Image Classifier

Step 4: Find example code that performs model inference

inference

```
1 # Copyright 2017 The TensorFlow Authors. All Rights Reserved.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 #
15 """Runs a ResNet model on the ImageNet dataset."""
16
17 from __future__ import absolute_import
18 from __future__ import division
19 from __future__ import print_function
20
21 import os
22
23 from absl import app as absl_app
24 from absl import flags
25 import tensorflow as tf # pylint: disable=g-bad-import-order
26
27 from official.utils.flags import core as flags_core
28 from official.resnet import imagenet_preprocessing
29 from official.resnet import resnet_model
30 from official.resnet import resnet_run_loop
31
32 _DEFAULT_IMAGE_SIZE = 224
33 _NUM_CHANNELS = 3
34 _NUM_CLASSES = 1000
35
36 _NUM_IMAGES = {
37     'train': 1281167,
38     'validation': 50000,
39 }
40
41 _NUM_TRAIN_FILES = 1024
42 _SHUFFLE_BUFFER = 1500
```

```
88     Returns:
89     image_buffer: Tensor tf.string containing the contents of a JPEG file.
90     label: Tensor tf.int32 containing the label.
91
92     ding boxes arranged [1, num_boxes, coords]
93     1) and the coordinates are arranged as
94
95     feature[1], dtype=tf.string,
96     default_value=''),
97     lenFeature[1], dtype=tf.int64,
98     default_value=0),
99     lenFeature[1], dtype=tf.string,
100     default_value=''),
101     default_value=''),
102
103     return [
104         os.path.join(data_dir, 'train-%05d-of-01824' % i)
105         for i in range(_NUM_TRAIN_FILES)]
106     else:
107         return [
108             os.path.join(data_dir, 'validation-%05d-of-00128' % i)
109             for i in range(128)]
110
111
112     def _parse_example_proto(example_serialized):
113         """Parses an Example proto containing a training example of an image.
114
115         The output of the build_imagenet_data.py image preprocessing script is a dataset
116         containing serialized Example protocol buffers. Each Example proto contains
117         the following fields (values are included as examples):
118
119             image/height: 462
120             image/width: 581
121             image/colorspace: 'RGB'
122             image/channels: 3
123             image/class/label: 615
124             image/class/synset: 'n03623198'
125             image/class/text: 'knee pad'
126             image/object/bbox/min_x: 0.1
127             image/object/bbox/max_x: 0.9
128             image/object/bbox/min_y: 0.2
129             image/object/bbox/max_y: 0.6
130             image/object/bbox/label: 615
131             image/format: 'JPEG'
132             image/filename: 'ILSVRC2012_val_00041207.JPG'
133             image/encoded: <JPEG encoded string>
134
135     Args:
136         example_serialized: scalar Tensor tf.string containing a serialized
137             Example protocol buffer.
```

```
133     The input record is parsed into a label and image, and the image is passed
134     through preprocessing steps (cropping, flipping, and so on).
135
136
137     record: scalar Tensor tf.string containing a serialized
138     Example protocol buffer.
139     training: A boolean denoting whether the input is for training.
140
141     ms:
142     file with processed image tensor and one-hot-encoded label tensor.
143
144     _buffer, label, bbox = _parse_example_proto(raw_record)
145
146     = imagenet_preprocessing.preprocess_image(
147         image_buffer=image_buffer,
148         bbox=bbox,
149         input_height=_DEFAULT_IMAGE_SIZE,
150         input_width=_DEFAULT_IMAGE_SIZE,
151         num_channels=_NUM_CHANNELS,
152         is_training=is_training)
153
154     = tf.one_hot(tf.reshape(label, shape=[]), _NUM_CLASSES)
155
156     image, label
157
158     ut_fn(is_training, data_dir, batch_size, num_epochs=1):
159         put function which provides batches for train or eval.
160
161
162     training: A boolean denoting whether the input is for training.
163     a_dir: The directory containing the input data.
164     ch_size: The number of samples per batch.
165     _epochs: The number of epochs to repeat the dataset.
166
167     ms:
168     dataset that can be used for iteration.
169
170     names = get_filenames(is_training, data_dir)
171     et = tf.data.Dataset.from_tensor_slices(filenames)
172
173     _training:
174     shuffle the input files
175     asset = dataset.shuffle(buffer_size=_NUM_TRAIN_FILES)
176
177
178     def run_imagenet(flags_obj):
179         """Run ResNet ImageNet training and eval loop.
180
181     Args:
182         flags_obj: An object containing parsed flag values.
183
184         """
185
186         input_function = [flags_obj.use_synthetic_data and get_synth_input_fn()
```

```
187
188         # Convert to individual records
189         dataset = dataset.flat_map(tf.data.TFRecordDataset)
190
191
192         return resnet_run_loop.process_dataset(
193             dataset, is_training, batch_size, _SHUFFLE_BUFFER, parse_record,
194             num_epochs
195         )
196
197         selected Resnet size.\n"
198         owed: {}.".format(
199             {}
200         )
201
202         def get_synth_input_fn():
203             return resnet_run_loop.get_synth_input_fn(
204                 _DEFAULT_IMAGE_SIZE, _DEFAULT_IMAGE_SIZE, _NUM_CHANNELS, _NUM_CLASSES)
205
206
207         modes, params:
208             with our Estimator.""
209                 # Running the model
210                 training_rate_with_decay(
211                     batch_denom=256,
212                     learning_rate=_LEARNING_RATE,
213                     boundary_epochs=[30, 60, 80, 90],
214                     scale_factor=4)
215
216
217         def __init__(self, resnet_size, data_format=None, num_classes=_NUM_CLASSES,
218             resnet_version=resnet_model.DEFAULT_VERSION,
219             dtype=resnet_model.DEFAULT_DTYPE):
220
221             """These are the parameters that work for Imagenet data.
222
223             Args:
224                 resnet_size: The number of convolutional layers needed in the model.
225                 data_format: Either 'channels_first' or 'channels_last', specifying which
226                     data format to use when setting up the model.
227                 num_classes: The number of output classes needed from the model. This
228                     enables users to extend the same model to their own datasets.
229                 resnet_version: Integer representing which version of the ResNet network
230                     to use. See README for details. Valid values: [1, 2].
231
232                 dtype: The TensorFlow dtype to use for calculations.
233
234             # For bigger models, we want to use "bottleneck" layers
235             if resnet_size < 50:
236                 bottleneck = False
237                 final_size = 512
238             else:
239                 bottleneck = True
240                 final_size = 2048
241
242             resnet_size
243
244
245             if __name__ == '__main__':
246                 tf.logging.set_verbosity(tf.logging.INFO)
247                 define_imagenet_flags()
248                 absl_app.run(main)
```

*TensorFlow code for **training** and **batch inference*** on ResNet50*

 SPARK+AI
SUMMIT 2018

Example: Get an Image Classifier

Step 5: Write your own code to perform model inference on one image at a time

Step 6: Package your inference code, graph creation code, and pre-trained weights together

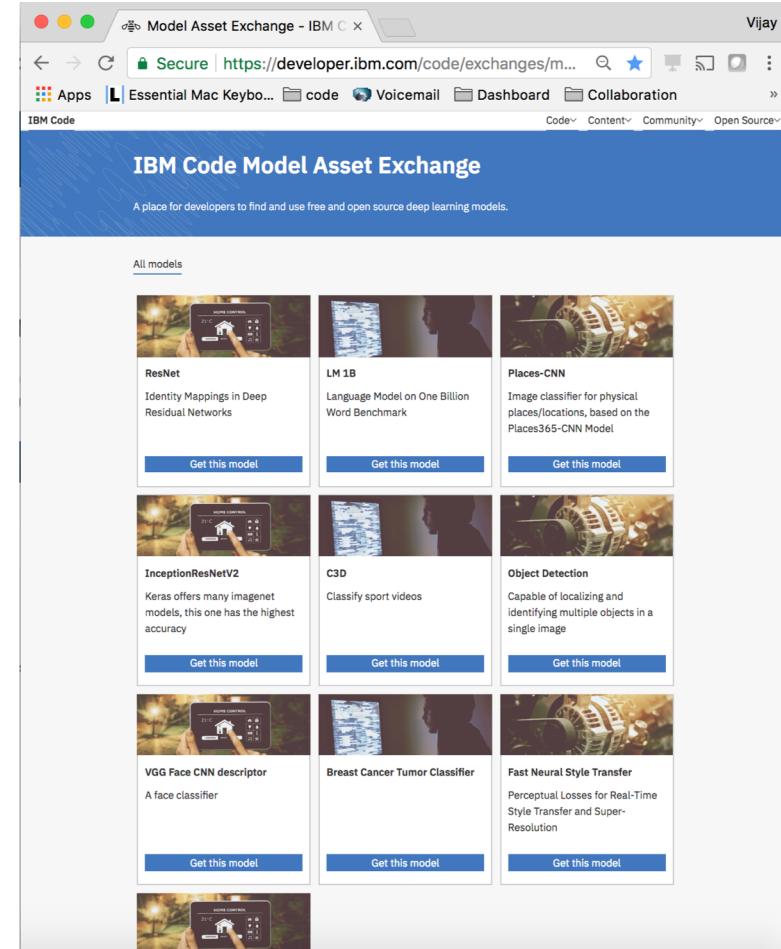
Step 7: Deploy your package

Model Marketplaces

- Collections of well-understood deep learning models
- Provide a central place to find known-good implementations of these models

The IBM Code Model Asset eXchange

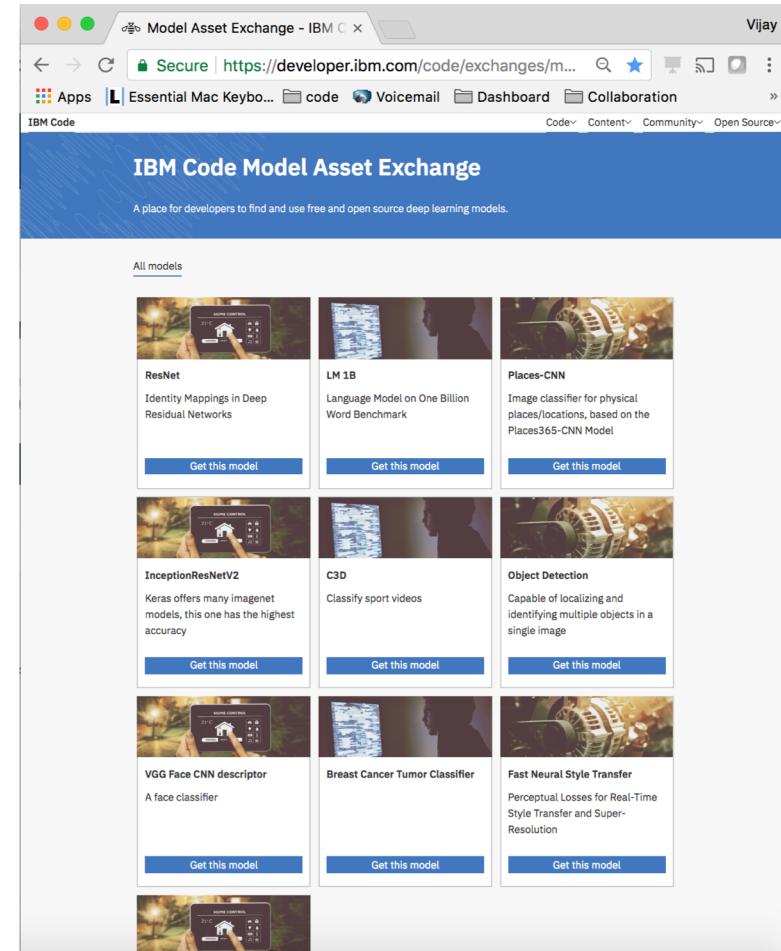
- Free, open-source models.
- Wide variety of domains.
- Multiple deep learning frameworks.
- Vetted and tested code and IP.
- Build and deploy a web service in 30 seconds.
- Start training on **Watson Studio** in minutes.



- Demo!

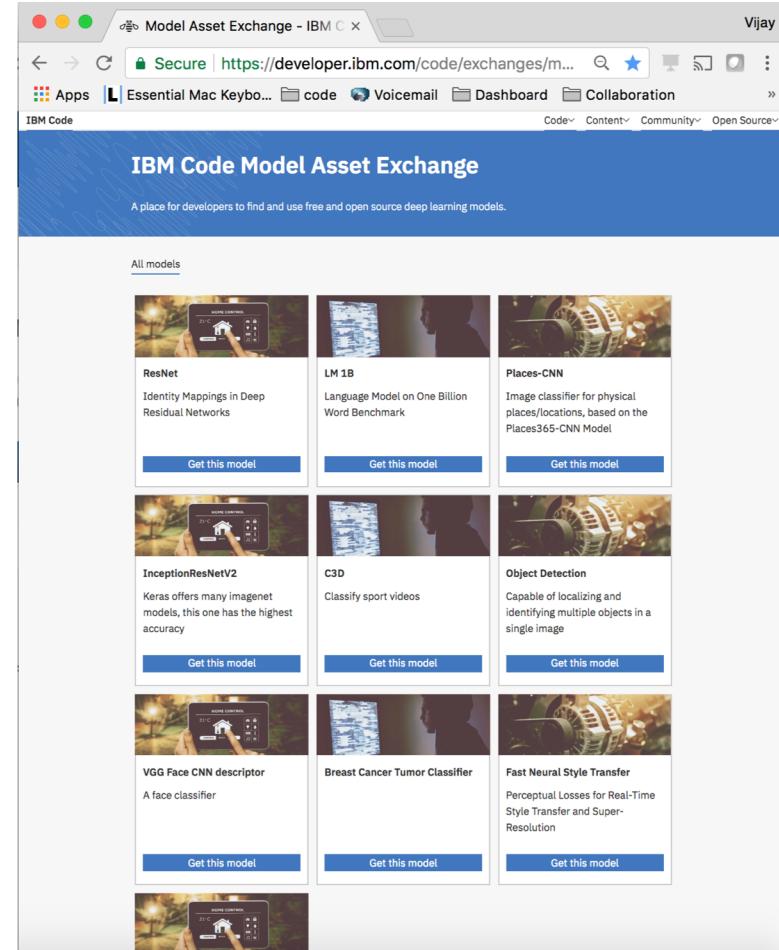
Model Asset eXchange: Summary

- Free, open-source models.
- Wide variety of domains.
- Multiple deep learning frameworks.
- Vetted and tested code and IP.
- Build and deploy a web service in 30 seconds.
- Start training on **Watson Studio** in minutes.



Model Asset eXchange: What's Next

- More models
- More deployment options
- Code Patterns showing how to use the models (including today's demo!)



Thank you!



- <http://codait.org>
- <https://developer.ibm.com/code/exchanges/models/>



- github.com/codait
- developer.ibm.com/code





CALL FOR CODE

GLOBAL INITIATIVE 2018

Commit to the cause. Push for change.

Call for Code inspires developers to solve **pressing global problems** with **sustainable software solutions**, delivering on their vast potential to do good.

Bringing together NGOs, academic institutions, enterprises, and startup developers to compete build effective **disaster mitigation solutions**, with a focus on health and well-being.

International Federation of Red Cross/Red Crescent, The American Red Cross, and the United Nations Office of Human Rights combine for the ***Call for Code Award*** to elevate the profile of developers.

Award winners will receive **long-term support** through **open source foundations**, **financial prizes**, the **opportunity to present their solution to leading VCs**, and will deploy their solution through **IBM's Corporate Service Corps**.

Developers will jump-start their project with dedicated **IBM Code Patterns**, combined with **optional enterprise technology** to build projects over the course of three months.

Judged by the world's most **renowned technologists**, the **grand prize** will be presented in **October** at an Award Event.

developer.ibm.com/callforcode

IBM Sessions at Spark+AI Summit 2018 (Tuesday, June 5)

Date, Time, Location & Duration	Session title and Speaker
Tue, June 5 11 AM 2010-2012, 30 mins	Productionizing Spark ML Pipelines with the Portable Format for Analytics Nick Pentreath (IBM)
Tue, June 5 2 PM 2018, 30 mins	Making PySpark Amazing—From Faster UDFs to Dependency Management and Graphing! Holden Karau (Google) Bryan Cutler (IBM)
Tue, June 5 2 PM Nook by 2001, 30 mins	Making Data and AI Accessible for All Armand Ruiz Gabernet (IBM)
Tue, June 5 2:40 PM 2002-2004, 30 mins	Cognitive Database: An Apache Spark-Based AI-Enabled Relational Database System Rajesh Bordawekar (IBM T.J. Watson Research Center)
Tue, June 5 3:20 PM 3016-3022, 30 mins	Dynamic Priorities for Apache Spark Application's Resource Allocations Michael Feiman (IBM Spectrum Computing) Shinnosuke Okada (IBM Canada Ltd.)
Tue, June 5 3:20 PM 2001-2005, 30 mins	Model Parallelism in Spark ML Cross-Validation Nick Pentreath (IBM) Bryan Cutler (IBM)
Tue, June 5 3:20 PM 2007, 30 mins	Serverless Machine Learning on Modern Hardware Using Apache Spark Patrick Stuedi (IBM)
Tue, June 5 5:40 PM 2002-2004, 30 mins	Create a Loyal Customer Base by Knowing Their Personality Using AI-Based Personality Recommendation Engine; Sourav Mazumder (IBM Analytics) Aradhna Tiwari (University of South Florida)
Tue, June 5 5:40 PM 2007, 30 mins	Transparent GPU Exploitation on Apache Spark Dr. Kazuaki Ishizaki (IBM) Madhusudanan Kandasamy (IBM)
Tue, June 5 5:40 PM 2009-2011, 30 mins	Apache Spark Based Hyper-Parameter Selection and Adaptive Model Tuning for Deep Neural Networks Yonggang Hu (IBM) Chao Xue (IBM)

IBM Sessions at Spark+AI Summit 2018 (Wednesday, June 6)

Date, Time, Location & Duration	Session title and Speaker
Wed, June 6 12:50 PM	Birds of a Feather: Apache Arrow in Spark and More Bryan Cutler (IBM) Li Jin (Two Sigma Investments, LP)
Wed, June 6 2 PM 2002-2004, 30 mins	Deep Learning for Recommender Systems Nick Pentreath (IBM))
Wed, June 6 3:20 PM 2018, 30 mins	Bringing an AI Ecosystem to the Domain Expert and Enterprise AI Developer Frederick Reiss (IBM) Vijay Bommireddipalli (IBM Center for Open-Source Data & AI Technologies)

Meet us at IBM booth in the Expo area.

Thank you!

- ∞ <http://codait.org>
- <https://developer.ibm.com/code/exchanges/models/>
- <https://www.linkedin.com/in/fred-reiss/>
- <https://www.linkedin.com/in/vijayrb/>



github.com/codait



developer.ibm.com/code

