Microsoft Cognitive Toolkit (CNTK)

Philipp Kranen
Senior Software Engineer
Microsoft AI&Research

pkranen@microsoft.com

With many contributors:

A. Agarwal, E. Akchurin, C. Basoglu, B. Bozza, G. Chen, S. Cyphers, W. Darling, J. Droppo, A. Eversole, B. Guenter, M. Hillebrand, X.-D. Huang, Z. Huang, W. Richert, R. Hoens, V. Ivanov, A. Kamenev, N. Karampatziakis, P. Kranen, O. Kuchaiev, W. Manousek, C. Marschner, A. May, B. Mitra, O. Nano, G. Navarro, A. Orlov, P. Parthasarathi, B. Peng, M. Radmilac, A. Reznichenko, W. Richert, M. Seltzer, M. Slaney, A. Stolcke, T. Will, H. Wang, W. Xiong, K. Yao, D. Yu, Y. Zhang, G. Zweig, and many more

Special acknowledgment to Frank Seide, Dong Yu, Jasha Droppo, John Langford, Jacob Devlin, Sean McDirmid (Y-Combinator Research), Bruno Bozza

Microsoft

I. what
II. how
III. examples
IV. deep dive

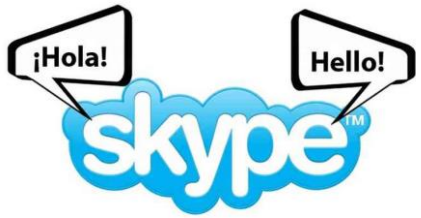
Microsoft Cognitive Toolkit, CNTK

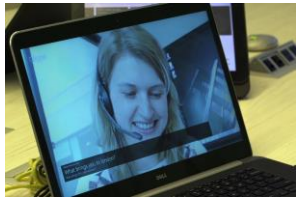
- CNTK is a library for deep neural networks
  - model definition
  - scalable training
  - efficient I/O
- Makes it easy to author, train, and use neural networks
  - think “what” not “how”
  - focus on composability
- Open source since 2015 https://github.com/Microsoft/CNTK

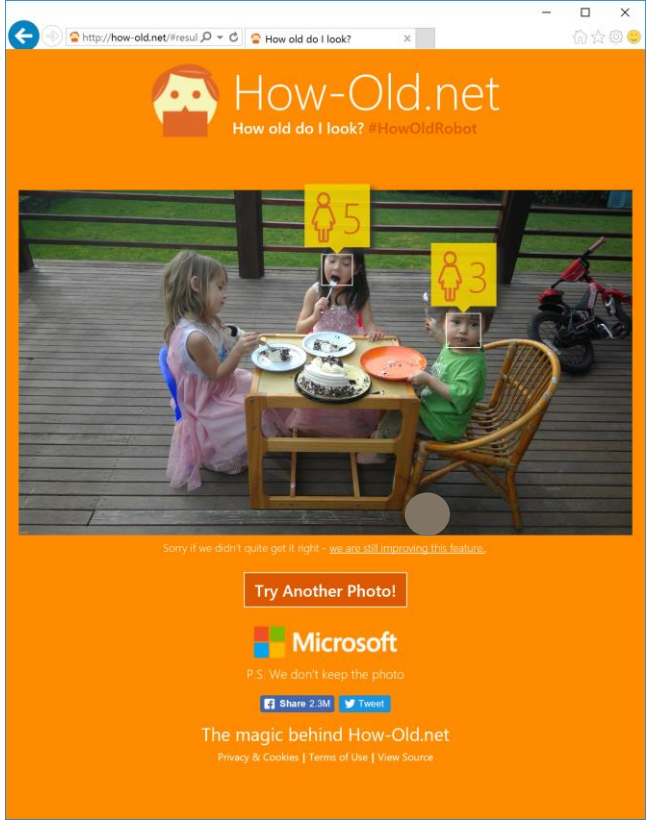
- CNTK 2.0 released June 1st, 2017
  - Core strengths: Performance and scalability
  - Major updates on API / flexibility (Python API, Layers Library, Keras etc.)

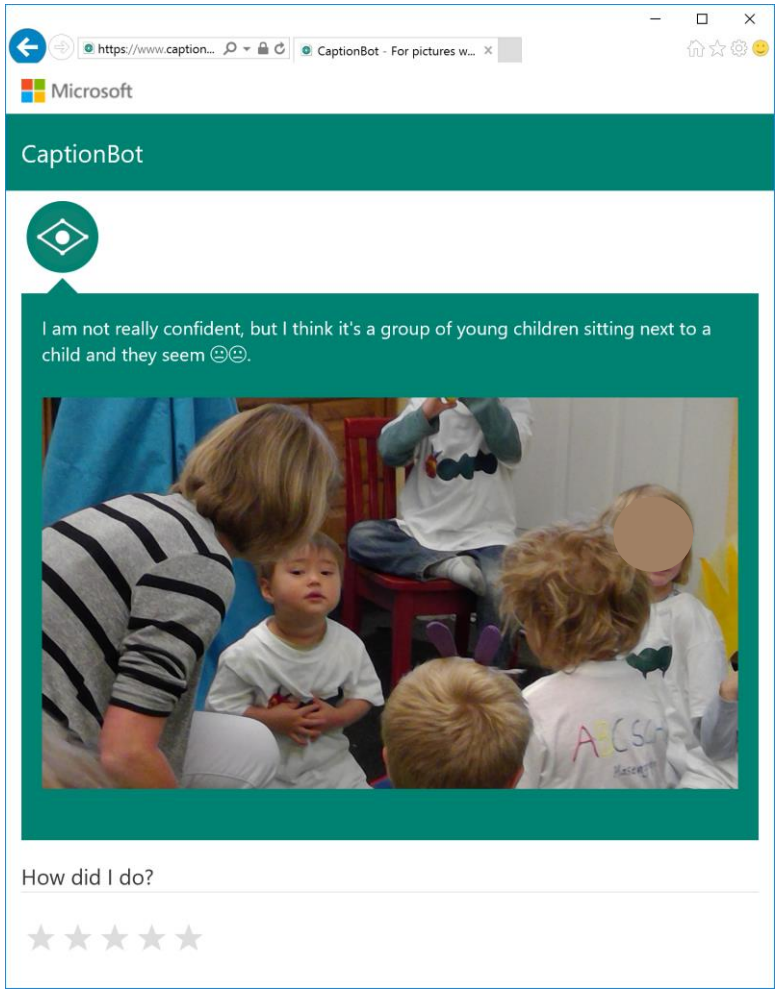
Deep Learning at Microsoft

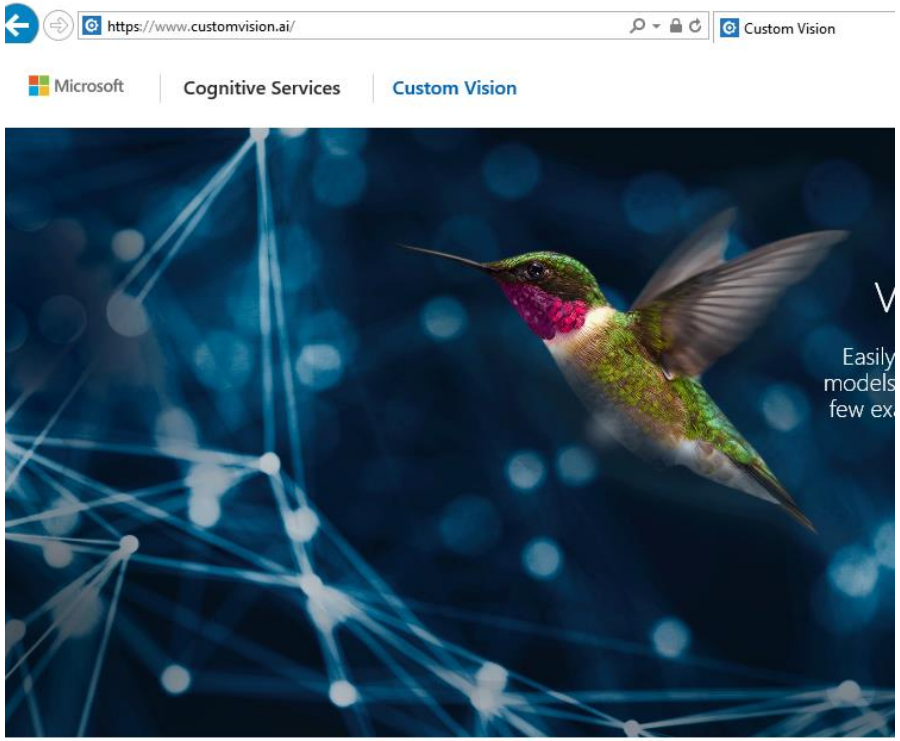
- Skype Translator
- Cortana
- Bing
- HoloLens
- Microsoft Cognitive Services
- Microsoft Research






How-Old.net
How old do I look? #HowOldRobot
5
3
Sorry if we didn't quite get it right - we are still improving this feature.
Try Another Photo!
Microsoft
P.S. We don't keep the photo
Share 2.3M
Tweet
The magic behind How-Old.net
Privacy & Cookies | Terms of Use | View Source


Microsoft
CaptionBot
I am not really confident, but I think it's a group of young children sitting next to a child and they seem 😐😐.
How did I do?


Microsoft
Cognitive Services
Custom Vision

Uber to require selfie security check from drivers

Using Microsoft Cognitive Services, Uber hopes to make riders feel safer by verifying the ID of drivers before rides are given.

By Jake Smith for iGeneration | September 23, 2016 -- 19:59 GMT (03:59 GMT+08:00) | Topic: Innovation

Uber announced on Friday a new security feature called Real-Time ID Check that will require drivers to periodically take a selfie before starting their driving shift.

The feature, which begins rolling out to US cities on Friday, uses Microsoft Cognitive Services to reduce fraud and give riders an extra sense of security.

Uber says Microsoft's feature instantly compares the selfie to the one corresponding with the account on file. If the two

SHARING ECONOMY

RELATED STORIES

Innovation

Victoria partners with Bosch for self-

Microsoft Customer Support Agent

TWCN Tech News

Home | TWC | Microsoft | Windows | Office | IE | Phone | Security | Social Media

September 6, 2016

Microsoft and Liebherr together to make Refrigerators smart

Smart refrigerators, Cortana, Microsoft and Liebherr

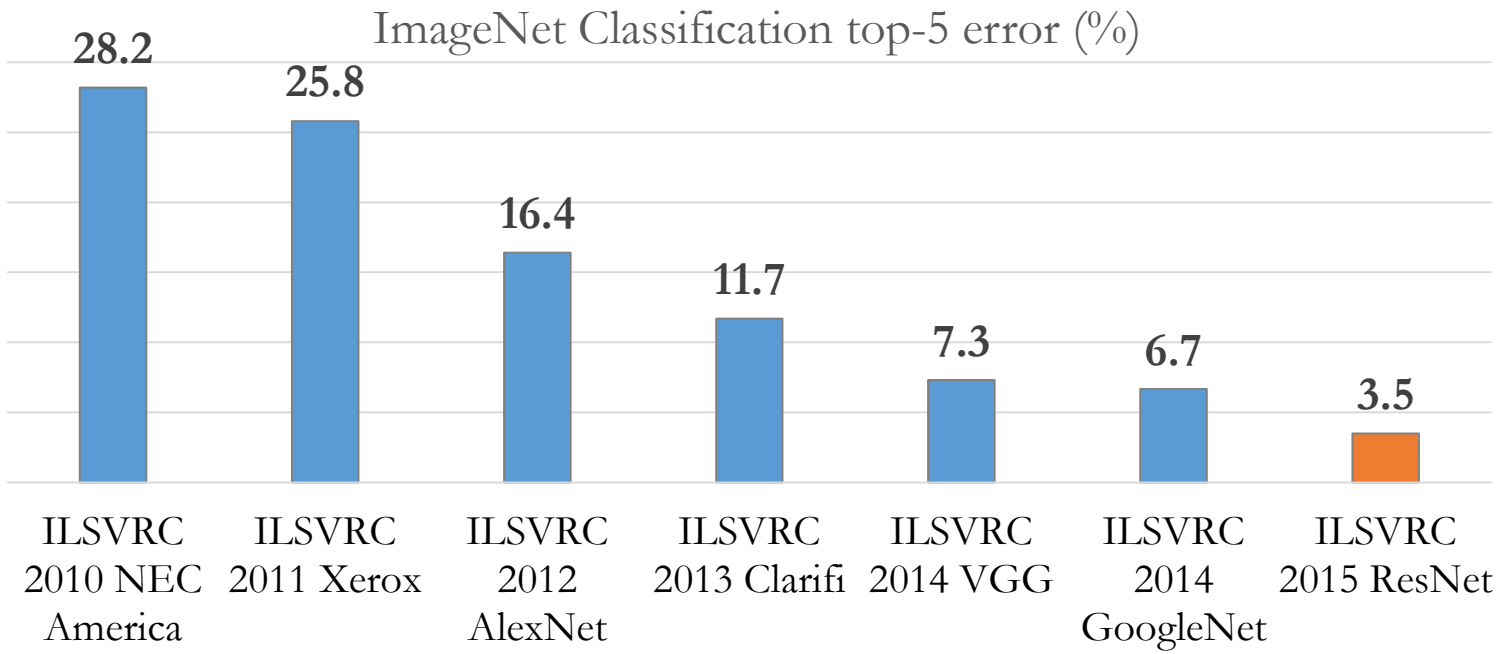
When this joint venture of Microsoft and Liebherr will come into reality, it will be the next level of machine learning. SmartDeviceBox is nothing a communication module which fits into Liebherr refrigerators and freezers, connecting them to the internet. The modular units can be integrated and upgraded at any time in existing SmartDevice-ready appliances to create value and comfort for customers through new digital features and solutions.

Speech Research: Microsoft's historic speech breakthrough

- CNTK was initiated in 2012 by Microsoft Speech Researchers
- In 2016 Microsoft's research system for conversational speech recognition reached 5.9% word-error rate
- enabled by CNTK's multi-server scalability

[W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, G. Zweig: "Achieving Human Parity in Conversational Speech Recognition," https://arxiv.org/abs/1610.05256]

Vision Research: Microsoft 2015 ResNet



Microsoft had all **5 entries** being the 1-st places in 2015: ImageNet classification, ImageNet localization, ImageNet detection, COCO detection, and COCO segmentation

Microsoft Cognitive Toolkit (CNTK)

Microsoft's open-source deep-learning toolkit

- https://github.com/Microsoft/CNTK
- Created by Microsoft Speech researchers (Dong Yu et al.) in 2012, "Computational Network Toolkit"
- Open sourced 2015, on GitHub since Jan 2016 under MIT license
- Python support since Oct 2016, rebranded as "Cognitive Toolkit"
- External contributions e.g. from MIT, Stanford and NVidia

Over 80% Microsoft internal DL workload runs CNTK

1st-class on Linux and Windows, docker support

Python, C++, C#, Java, Keras (beta)

Internal == External

why

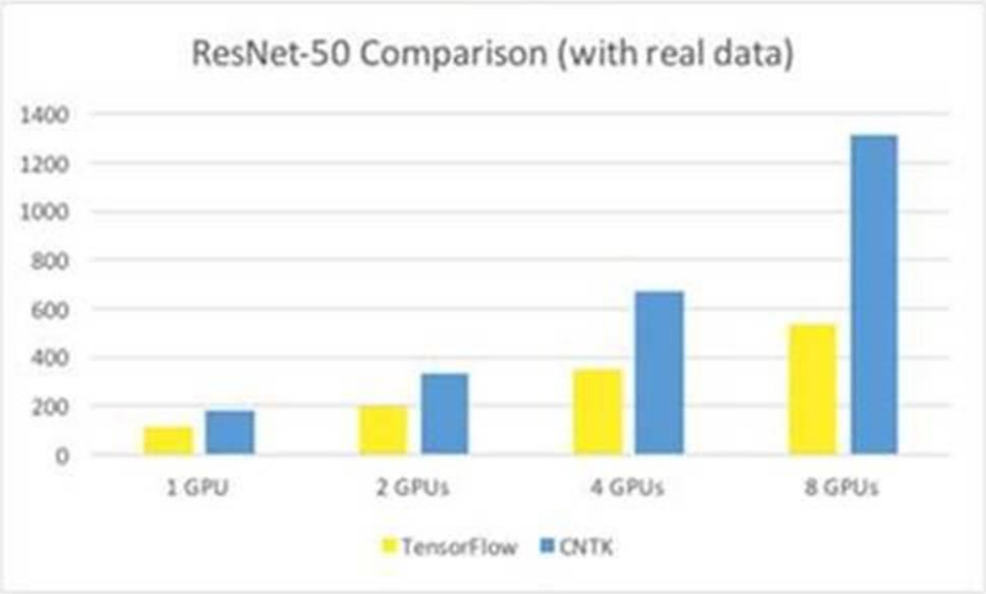
CNTK Speed

http://dlbench.comp.hkbu.edu.hk/
Benchmarking by HKBU, Version 8
Single Tesla K80 GPU, CUDA: 8.0 CUDNN: v5.1

Caffe: 1.0rc5(39f28e4)
CNTK: 2.0 Beta10(1ae666d)
MXNet: 0.93(32dc3a2)
TensorFlow: 1.0(4ac9c09)
Torch: 7(748f5e3)

	Caffe	CNTK	MxNet	TensorFlow	Torch
FCN5 (1024)	55.329ms	**51.038ms**	60.448ms	62.044ms	52.154ms
AlexNet (256)	36.815ms	**27.215ms**	28.994ms	103.960ms	37.462ms
ResNet (32)	143.987ms	**81.470ms**	84.545ms	181.404ms	90.935ms
LSTM (256) (v7 benchmark)	-	**43.581ms** (44.917ms)	288.142ms (284.898ms)	- (223.547ms)	1130.606ms (906.958ms)

CNTK Scalability



ResNet50 on DGX-1, tested by Nvidia, Dec. 2016

Image: Cray

Microsoft, Cray claim deep learning breakthrough on supercomputers

Steve Ranger
ZDNet

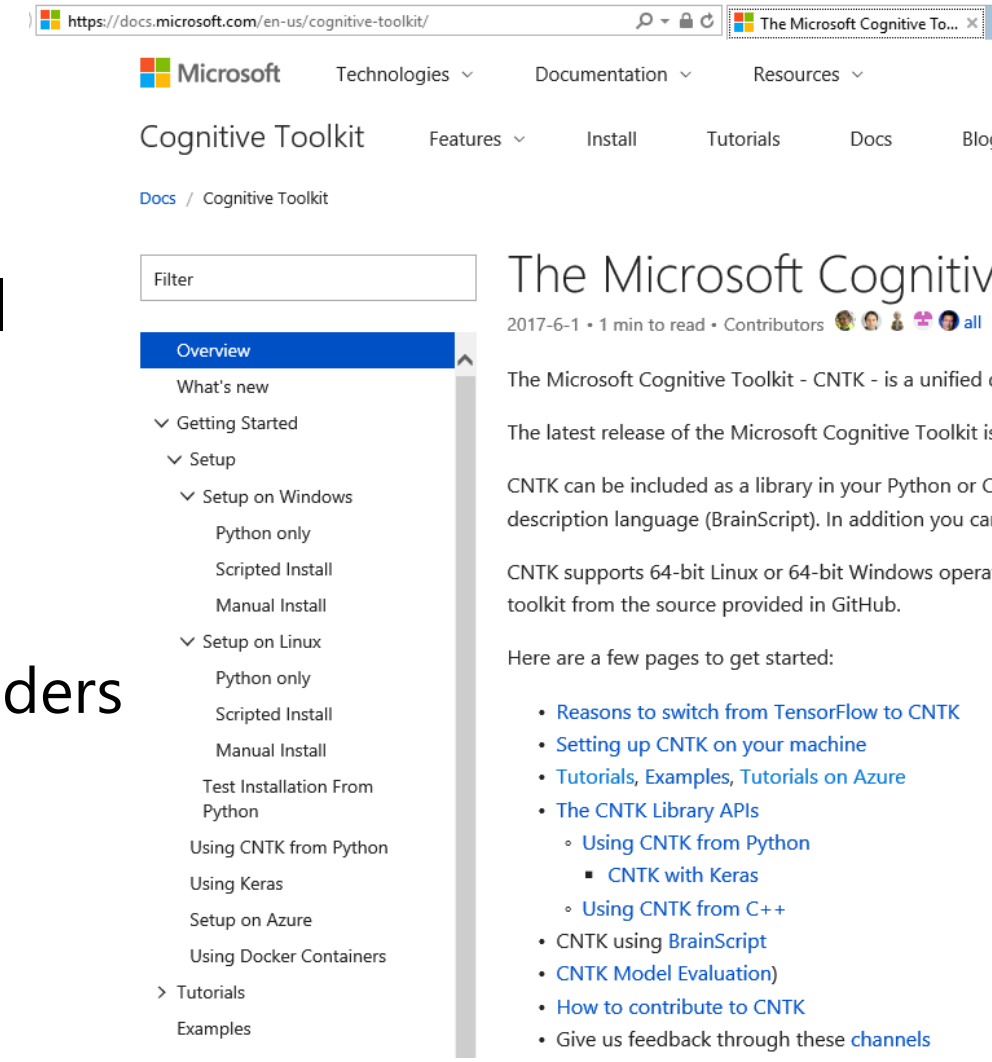
A team of researchers from Microsoft, Cray, and the Swiss National Supercomputing Centre (CSCS) have been working on a project to speed up the use of deep learning algorithms on supercomputers.

The team have scaled the Microsoft Cognitive Toolkit -- an open-source suite that trains deep learning algorithms -- to more than 1,000 Nvidia Tesla P100 GPU accelerators on the Swiss centre's Cray XC50 supercomputer, which is nicknamed Piz Daint.

CNTK Other Advantages

- Low level C++ and high level Python API
- Extensibility
  - User functions and learners in pure Python
- Readers
  - Distributed, highly efficient built-in data readers
- C#/.NET/Java inference support
- Internal == External

https://docs.microsoft.com/en-us/cognitive-toolkit/



I. what
II. how
III. examples
IV. deep dive

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

example: 2-hidden layer feed-forward NN

$$h_1 = \sigma(W_1 x + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$P = \text{softmax}(W_{\text{out}} h_2 + b_{\text{out}})$$

with input $x \in \mathrm{R}^M$

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

example: 2-hidden layer feed-forward NN

$$h_1 = \sigma(W_1 x + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$P = \text{softmax}(W_{\text{out}} h_2 + b_{\text{out}})$$

with input $x \in \mathrm{R}^M$ and one-hot label $y \in \mathrm{R}^J$
and cross-entropy training criterion

$$ce = y^{\mathrm{T}} \log P$$
$$\sum_{\text{corpus}} ce = \max$$

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

example: 2-hidden layer feed-forward NN

$$h_1 = \sigma(W_1 x + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$P = \text{softmax}(W_{\text{out}} h_2 + b_{\text{out}})$$

```
h1 = Sigmoid (W1   * x  + b1)
h2 = Sigmoid (W2   * h1 + b2)
P  = Softmax (Wout * h2 + bout)
```

with input $x \in \mathrm{R}^M$ and one-hot label $y \in \mathrm{R}^J$
and cross-entropy training criterion

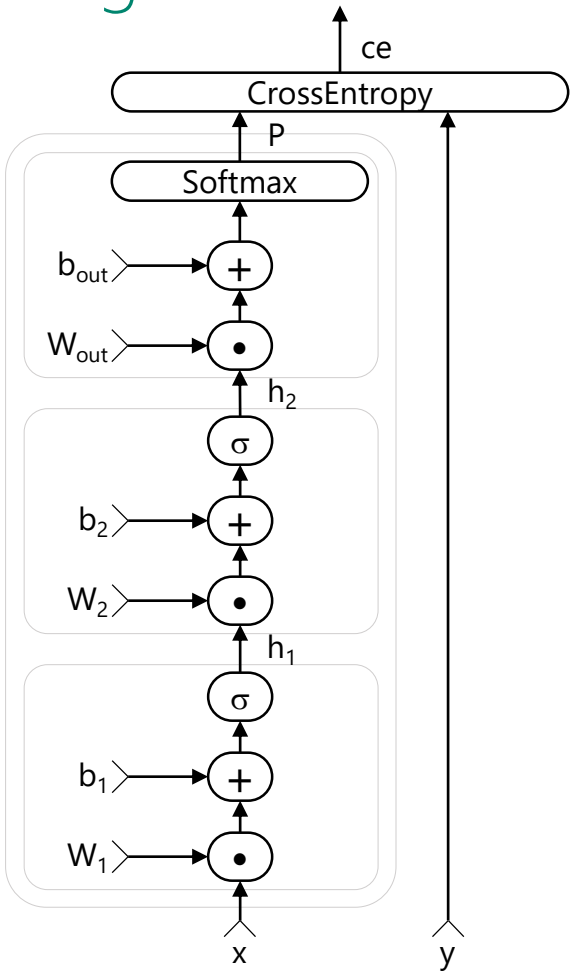
$$ce = y^{\mathrm{T}} \log P$$
$$\sum_{\text{corpus}} ce = \max$$

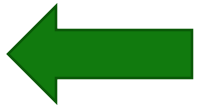
```
ce = CrossEntropy (y, P)
```

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."

```
h1 = Sigmoid (W1   * x  + b1)
h2 = Sigmoid (W2   * h1 + b2)
P  = Softmax (Wout * h2 + bout)
ce = CrossEntropy (y, P)
```

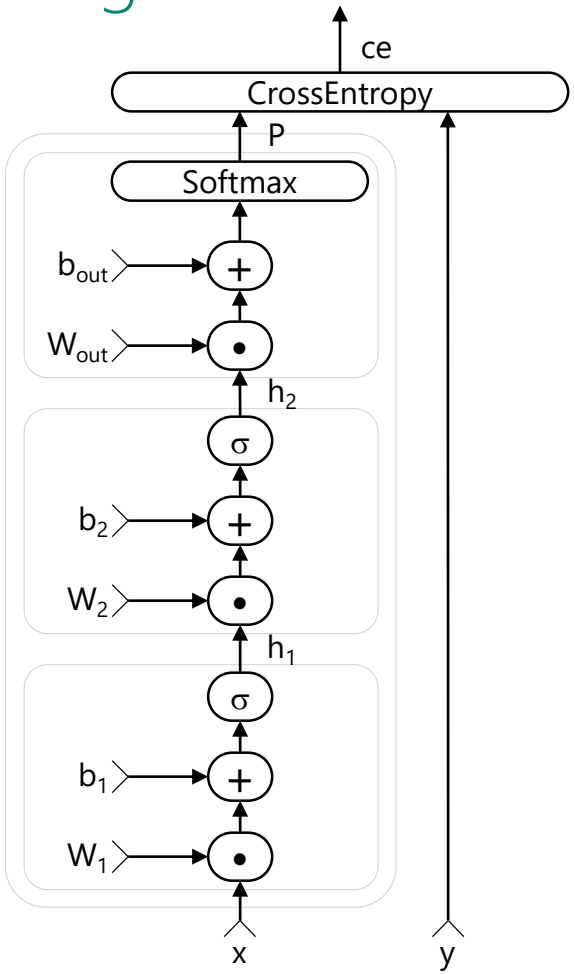
“CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications.”





```
h1 = Sigmoid (W1   * x  + b1)
h2 = Sigmoid (W2   * h1 + b2)
P  = Softmax (Wout * h2 + bout)
ce = CrossEntropy (y, P)
```

"CNTK expresses (nearly) arbitrary neural networks by composing simple building blocks into complex computational networks, supporting relevant network types and applications."



- nodes: functions (primitives)
  - can be composed into reusable composites
- edges: values
- automatic differentiation
  - $\partial \mathcal{F} / \partial \text{in} = \partial \mathcal{F} / \partial \text{out} \cdot \partial \text{out} / \partial \text{in}$
- deferred computation → execution engine
- editable, clonable

LEGO-like composability allows CNTK to support wide range of networks & applications

Layers lib: DNN building blocks

- layers/layers.py:
  - `Dense(), Embedding()`
  - `Convolution(), Convolution1D(), Convolution2D(), Convolution3D(), Deconvolution()`
  - `MaxPooling(), AveragePooling(), GlobalMaxPooling(), GlobalAveragePooling(), MaxUnpooling()`
  - `BatchNormalization(), LayerNormalization()`
  - `Dropout(), Activation()`
  - `Label()`
- layers/blocks.py:
  - `LSTM(), GRU(), RNNUnit()`
  - `Stabilizer(), identity`
  - `ForwardDeclaration(), Tensor[], SparseTensor[], Sequence[], SequenceOver[]`
- layers/higher_order_layers.py:
  - `Sequential(), For(), operator >>, (function tuples)`
  - `ResNetBlock(), SequentialClique()`
- layers/sequence.py:
  - `Delay(), PastValueWindow()`
  - `Recurrence(), RecurrenceFrom(), Fold(), UnfoldFrom()`
- models/models.py:
  - `AttentionModel()`

Python API example

```
with default_options(activation=cntk.ops.relu, pad=False):
    conv1 = Convolution2D((5,5), 32, pad=True)(scaled_input)
    pool1 = MaxPooling((3,3), (2,2))(conv1)
    conv2 = Convolution2D((3,3), 48)(pool1)
    pool2 = MaxPooling((3,3), (2,2))(conv2)
    conv3 = Convolution2D((3,3), 64)(pool2)
    f4    = Dense(96)(conv3)
    drop4 = Dropout(0.5)(f4)
    z     = Dense(num_output_classes, activation=None)(drop4)

ce = cntk.losses.cross_entropy_with_softmax(z, label_var)
pe = cntk.metrics.classification_error(z, label_var)
```

extending CNTK in pure Python

New operator

```
from cntk.ops.functions import UserFunction

class MySigmoid(UserFunction):
    def __init__(self, arg, name='MySigmoid'):
        super(MySigmoid, self).__init__([arg], name=name)

    def forward(self, argument, device=None, outputs_to_retain=None):
        sigmoid_x = 1 / (1 + np.exp(-argument))
        return sigmoid_x, sigmoid_x

    def backward(self, state, root_gradients):
        sigmoid_x = state
        return root_gradients * sigmoid_x * (1 - sigmoid_x)

    def infer_outputs(self):
        return [output_variable(self.inputs[0].shape, self.inputs[0].dtype,
            self.inputs[0].dynamic_axes)]
```

New learner

```
from cntk.learner import UserLearner

class MySgd(UserLearner):

    def __init__(self, parameters, lr_schedule):
        super(MySgd, self).__init__(parameters, lr_schedule)

    def update(self, gradient_values, training_sample_count, sweep_end):
        eta = self.learning_rate() / training_sample_count
        for p, g in gradient_values.items():
            new_p = p - eta * C.constant(g)
            p.set_value(new_p.eval(as_numpy=False).data())
        return True
```

Keras example

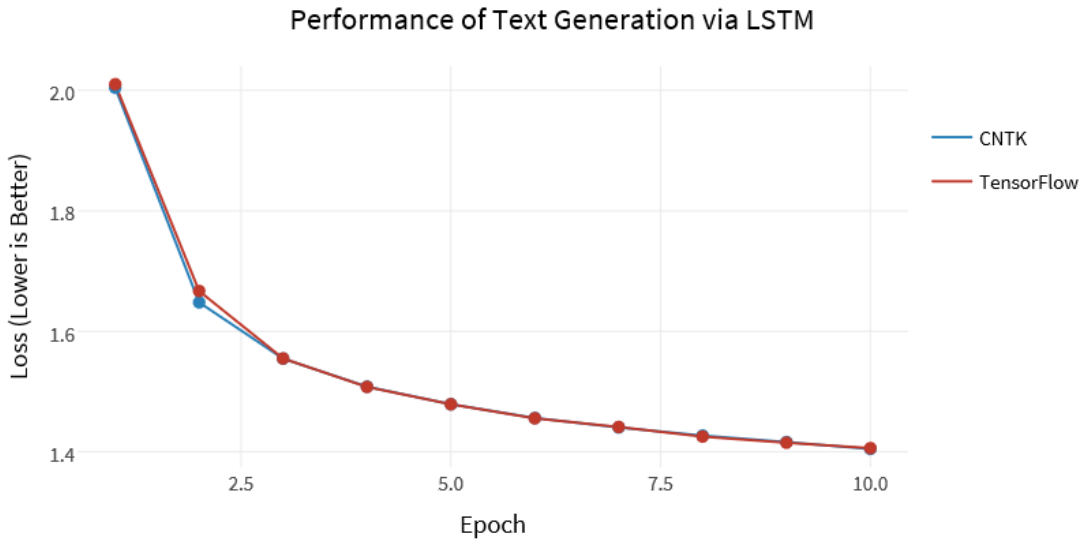
```
51  model = Sequential()
52  model.add(Conv2D(32, kernel_size=(3, 3),
53                   activation='relu',
54                   input_shape=input_shape))
55  model.add(Conv2D(64, (3, 3), activation='relu'))
56  model.add(MaxPooling2D(pool_size=(2, 2)))
57  model.add(Dropout(0.25))
58  model.add(Flatten())
59  model.add(Dense(128, activation='relu'))
60  model.add(Dropout(0.5))
61  model.add(Dense(num_classes, activation='softmax'))
62
63  model.compile(loss=keras.losses.categorical_crossentropy,
64                optimizer=keras.optimizers.Adadelta(),
65                metrics=['accuracy'])
```

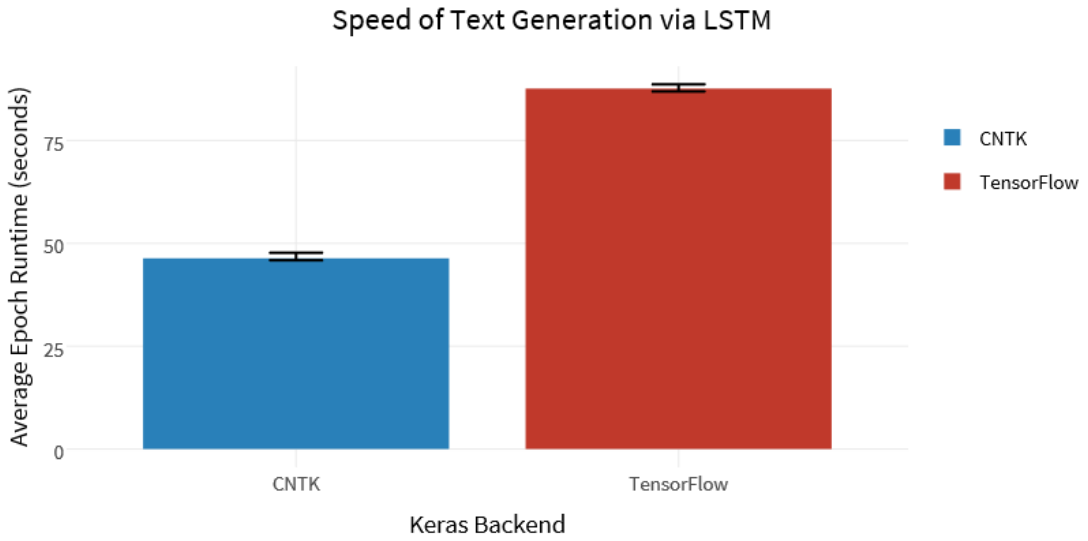
Keras benchmark

url:
http://minimaxir.com/2017/06/keras-cntk/

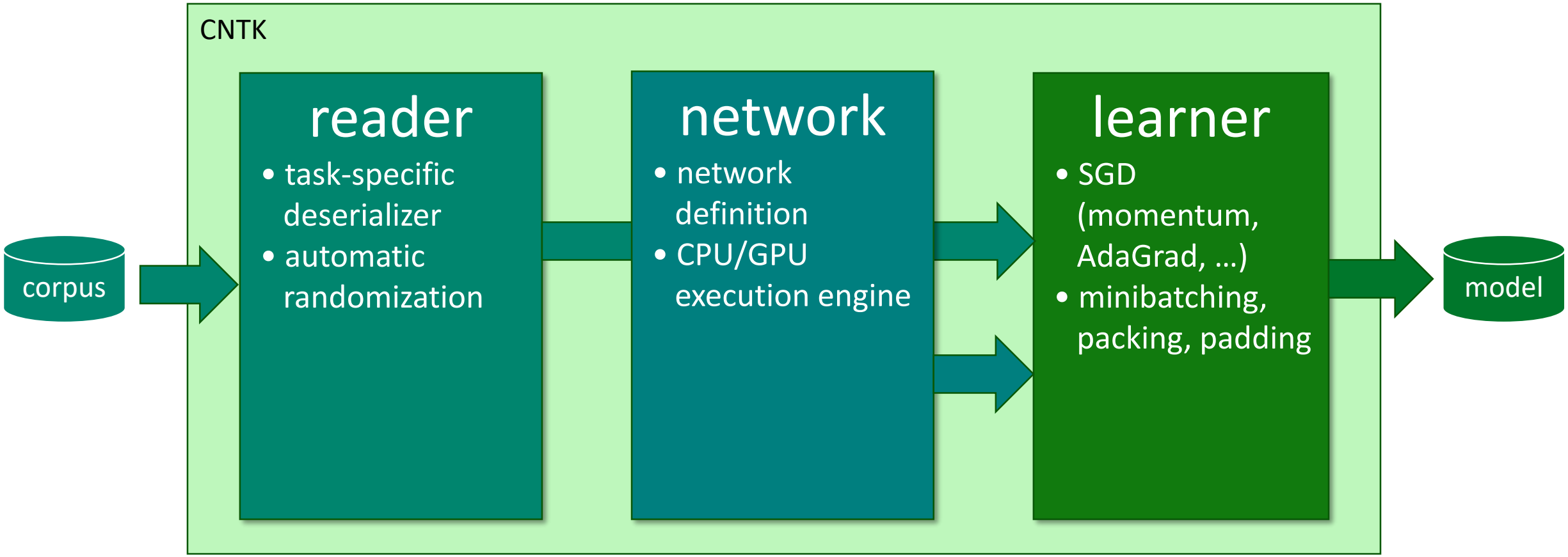
git repo:
https://github.com/minimaxir/keras-cntk-benchmark/

note:
CNTK Keras support is still beta and does not include performance optimizations such as 1-bit SGD or block momentum





CNTK architecture



I. what
II. how
III. examples
IV. deep dive

Examples

- Image classification recap
  - Terminology and basics
  - CNTK examples: VGG, ResNet
- Object recognition and segmentation
  - Object recognition algorithms
  - CNTK example: Fast R-CNN
  - Semantic segmentation and others

Computer vision tasks

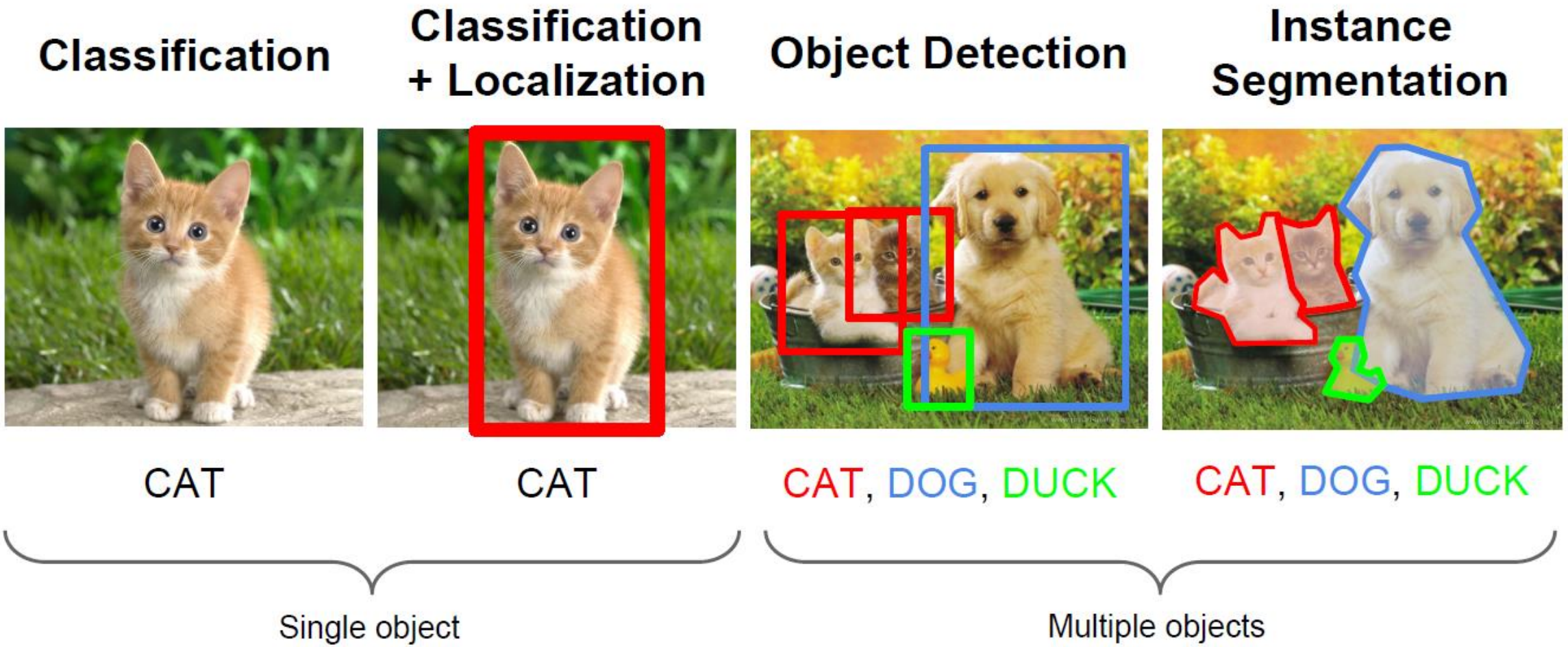
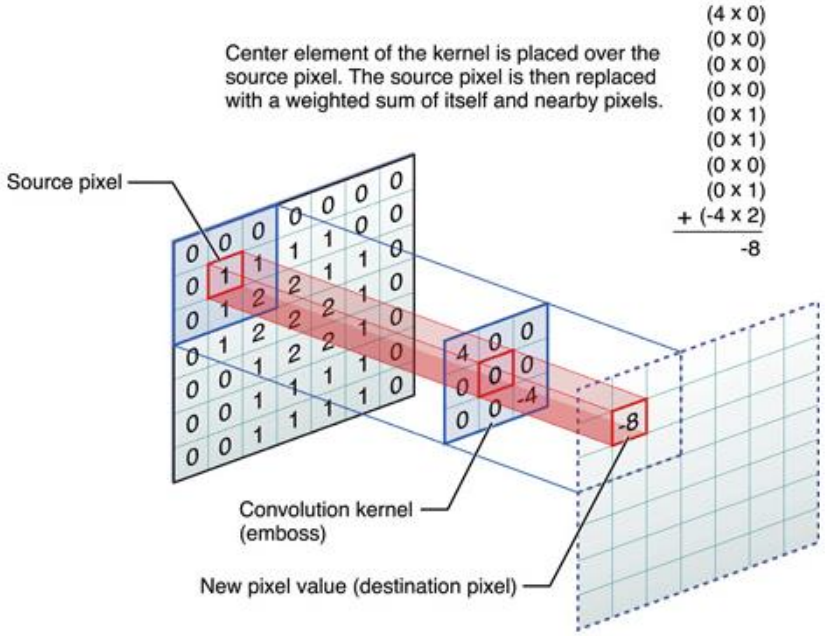


Image network components

- Convolution (filter)
- ReLu activation
- Pooling (max, avg)
- Batch normalization



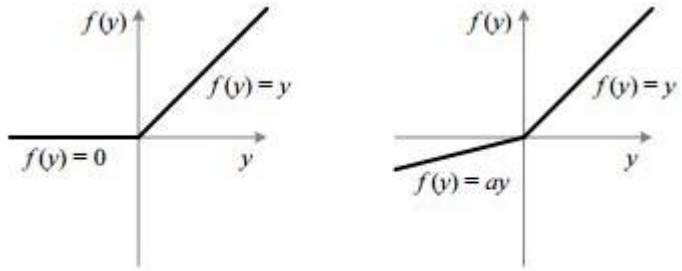
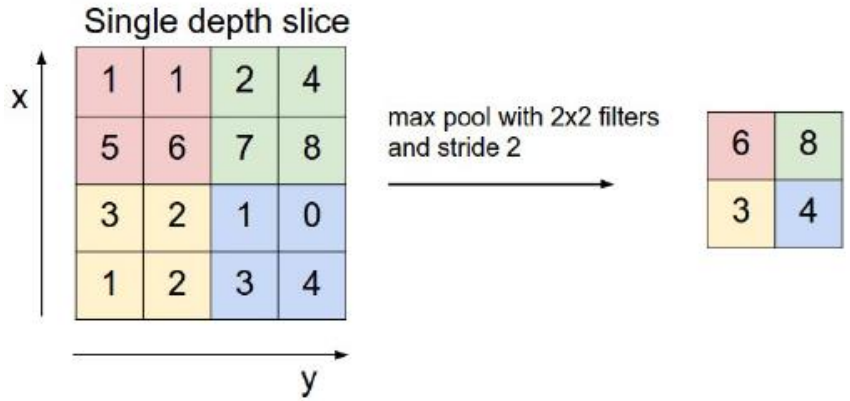
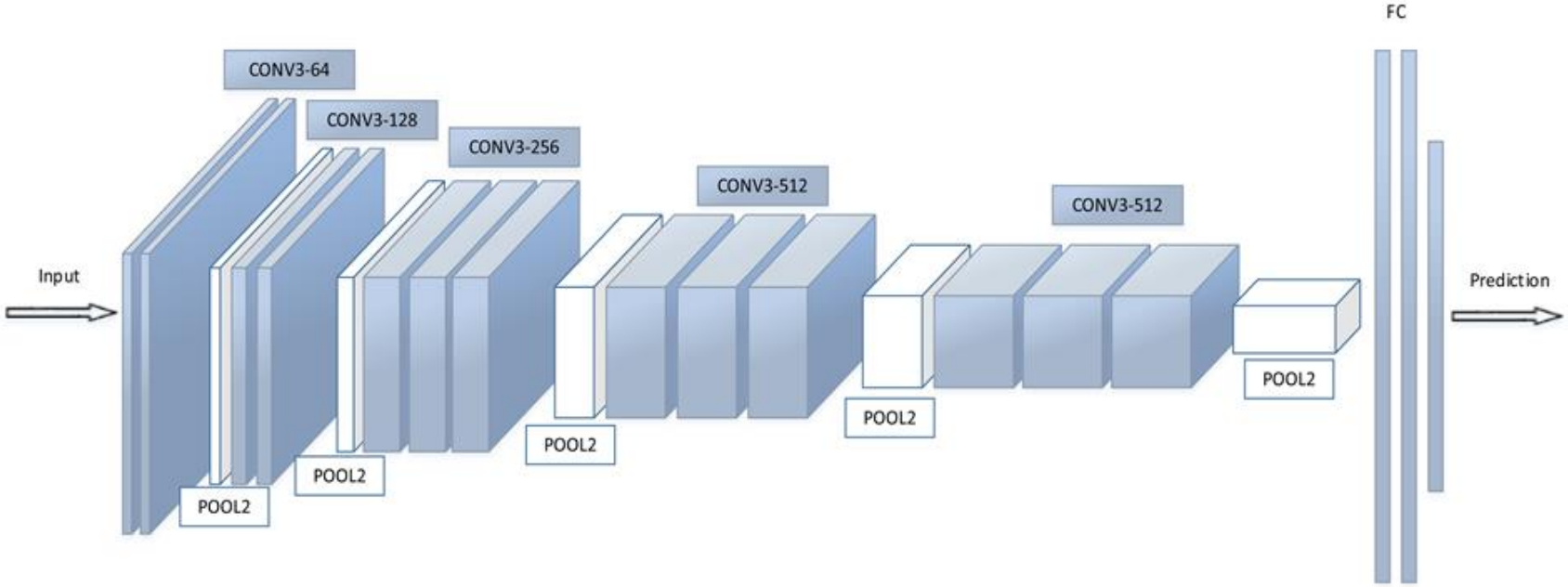


Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.



Example network: VGG



VGG Layer	Input Height x Width
Conv3-64	224 x 224
Conv3-64	224 x 224
maxpool	224 x 224
Conv3-128	112 x 112
Conv3-128	112 x 112
maxpool	112 x 112
Conv3-256	56 x 56
Conv3-256	56 x 56
Conv3-256	56 x 56
maxpool	56 x 56
Conv3-512	28 x 28
Conv3-512	28 x 28
Conv3-512	28 x 28
Max-pool	28 x 28
Conv3-512	14 x 14
Conv3-512	14 x 14
Conv3-512	14 x 14
maxpool	14 x 14
Fully connected 4096	7 x 7
Fully connected 4096	
Fully connected 1000	
Softmax	

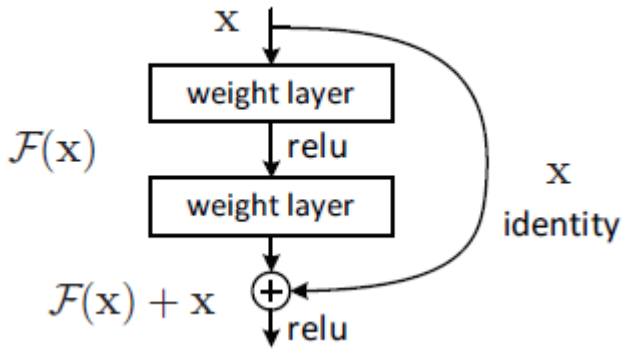
Python API example: VGG16 – model

```
with default_options(activation=None, pad=True, bias=True):
    z = Sequential([
        For(range(2), lambda i: [
            Convolution2D((3,3), 64, name='conv1_{}'.format(i)),
            Activation(activation=relu, name='relu1_{}'.format(i)),
        ]),
        MaxPooling((2,2), (2,2), name='pool1'),

        ...

        Dense(4096, name='fc6'),
        Activation(activation=relu, name='relu6'),
        Dropout(0.5, name='drop6'),
        Dense(4096, name='fc7'),
        Activation(activation=relu, name='relu7'),
        Dropout(0.5, name='drop7'),
        Dense(num_classes, name='fc8')
        ])(input)
```

VGG Layer	Input Height x Width
Conv3-64	224 x 224
Conv3-64	224 x 224
maxpool	224 x 224
Conv3-128	112 x 112
Conv3-128	112 x 112
maxpool	112 x 112
Conv3-256	56 x 56
Conv3-256	56 x 56
Conv3-256	56 x 56
maxpool	56 x 56
Conv3-512	28 x 28
Conv3-512	28 x 28
Conv3-512	28 x 28
Max-pool	28 x 28
Conv3-512	14 x 14
Conv3-512	14 x 14
Conv3-512	14 x 14
maxpool	14 x 14
Fully connected 4096	7 x 7
Fully connected 4096	
Fully connected 1000	
Softmax	

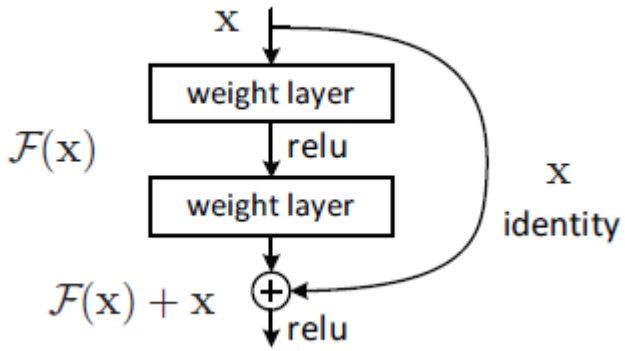
CNTK example: ResNet – component macros

```
def conv_bn(input, filter_size, num_filters, strides=(1,1), init=he_normal()):
    c = Convolution(filter_size, num_filters, activation=None, init=init, pad=True, strides=strides)(input)
    r = BatchNormalization(map_rank=1, normalization_time_constant=4096, use_cntk_engine=False)(c)
    return r
```



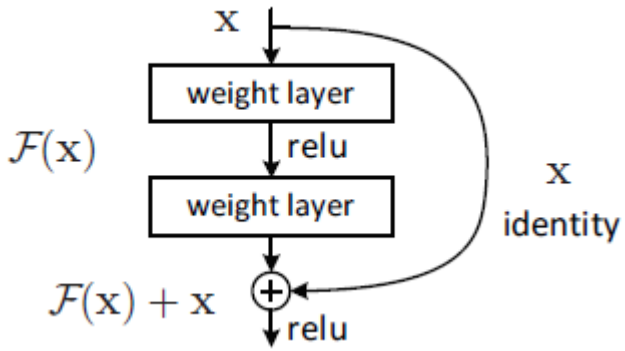
CNTK example: ResNet – component macros

```
def conv_bn(input, filter_size, num_filters, strides=(1,1), init=he_normal()):
    c = Convolution(filter_size, num_filters, activation=None, init=init, pad=True, strides=strides)(input)
    r = BatchNormalization(map_rank=1, normalization_time_constant=4096, use_cntk_engine=False)(c)
    return r

def conv_bn_relu(input, filter_size, num_filters, strides=(1,1), init=he_normal()):
    r = conv_bn(input, filter_size, num_filters, strides, init)
    return relu(r)
```



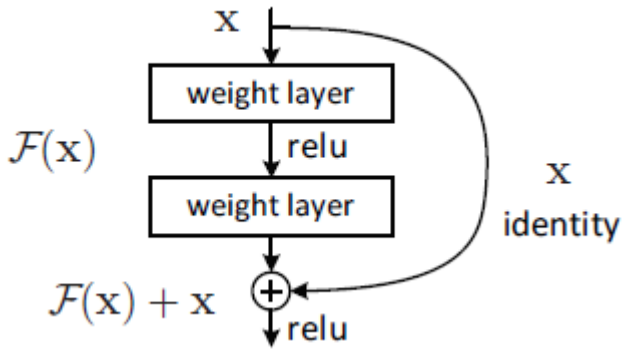
CNTK example: ResNet – component macros

```
def conv_bn(input, filter_size, num_filters, strides=(1,1), init=he_normal()):
    c = Convolution(filter_size, num_filters, activation=None, init=init, pad=True, strides=strides)(input)
    r = BatchNormalization(map_rank=1, normalization_time_constant=4096, use_cntk_engine=False)(c)
    return r

def conv_bn_relu(input, filter_size, num_filters, strides=(1,1), init=he_normal()):
    r = conv_bn(input, filter_size, num_filters, strides, init)
    return relu(r)

def resnet_basic(input, num_filters):
    c1 = conv_bn_relu(input, (3,3), num_filters)
    c2 = conv_bn(c1, (3,3), num_filters)
    p  = c2 + input
    return relu(p)
```



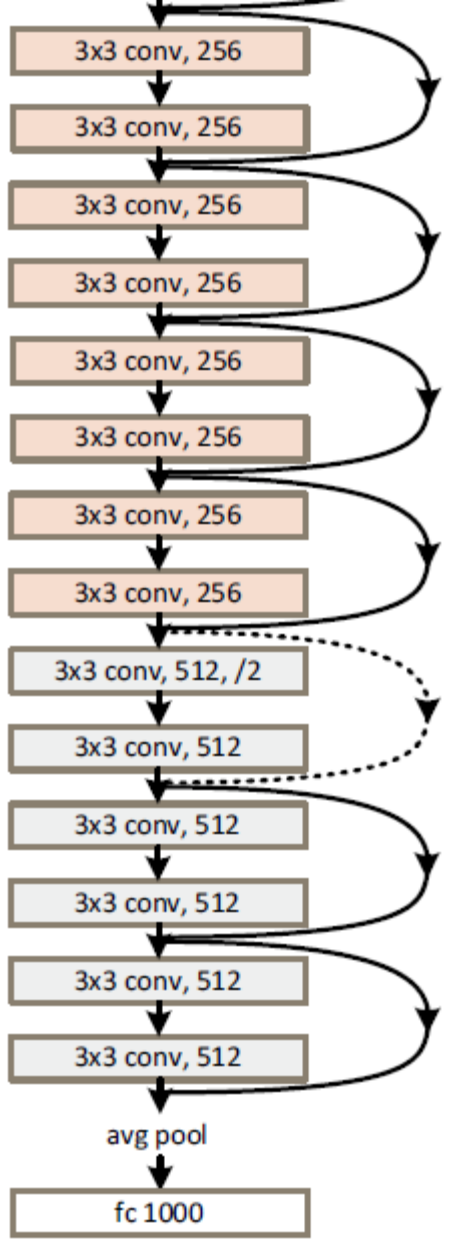
CNTK example: ResNet – component macros

```
def conv_bn(input, filter_size, num_filters, strides=(1,1), init=he_normal()):
    c = Convolution(filter_size, num_filters, activation=None, init=init, pad=True, strides=strides)(input)
    r = BatchNormalization(map_rank=1, normalization_time_constant=4096, use_cntk_engine=False)(c)
    return r

def conv_bn_relu(input, filter_size, num_filters, strides=(1,1), init=he_normal()):
    r = conv_bn(input, filter_size, num_filters, strides, init)
    return relu(r)

def resnet_basic(input, num_filters):
    c1 = conv_bn_relu(input, (3,3), num_filters)
    c2 = conv_bn(c1, (3,3), num_filters)
    p  = c2 + input
    return relu(p)

def resnet_basic_stack(input, num_stack_layers, num_filters):
    l = input
    for _ in range(num_stack_layers):
        l = resnet_basic(l, num_filters)
    return l
```

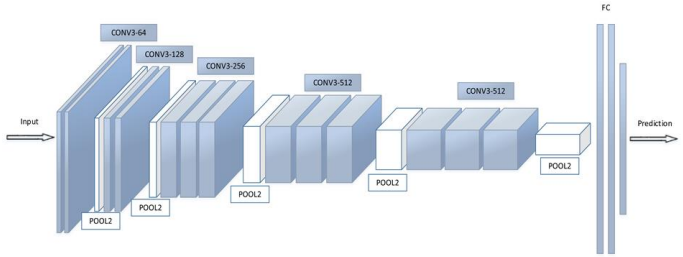


CNTK example: ResNet – model

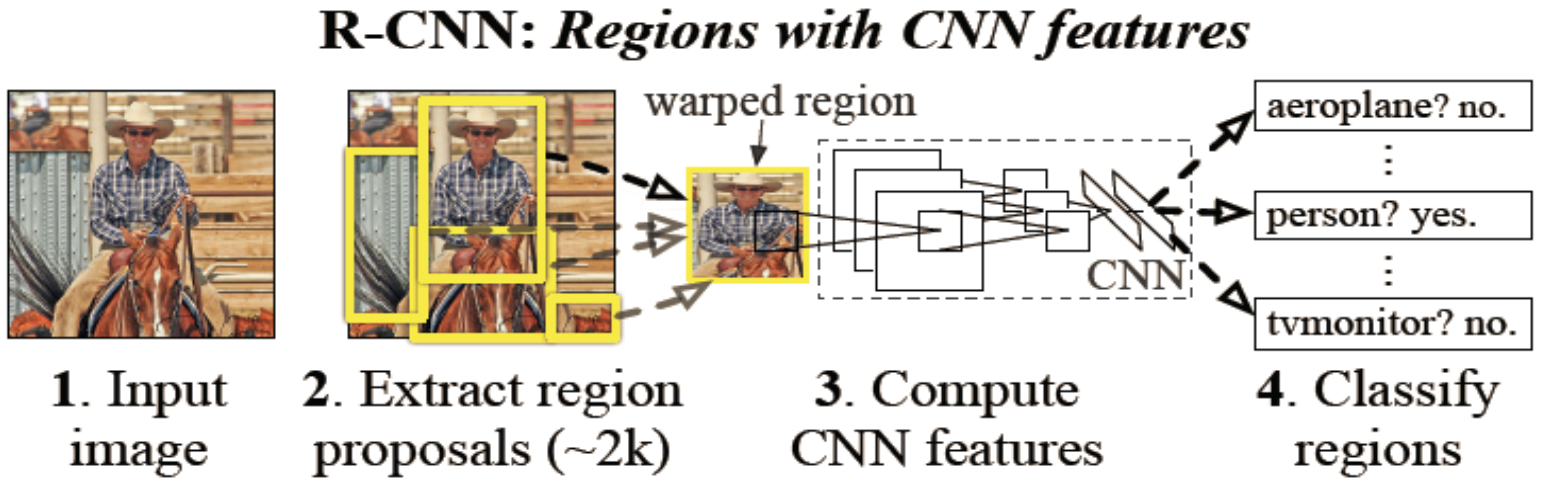
```
def create_cifar10_model(input, num_stack_layers, num_classes):
    c_map = [16, 32, 64]

    conv = conv_bn_relu(input, (3,3), c_map[0])
    r1 = resnet_basic_stack(conv, num_stack_layers, c_map[0])

    r2_1 = resnet_basic_inc(r1, c_map[1])
    r2_2 = resnet_basic_stack(r2_1, num_stack_layers-1, c_map[1])

    r3_1 = resnet_basic_inc(r2_2, c_map[2])
    r3_2 = resnet_basic_stack(r3_1, num_stack_layers-1, c_map[2])

    # Global average pooling and output
    pool = AveragePooling(filter_shape=(8,8))(r3_2)
    z = Dense(num_classes)(pool)
    return z
```



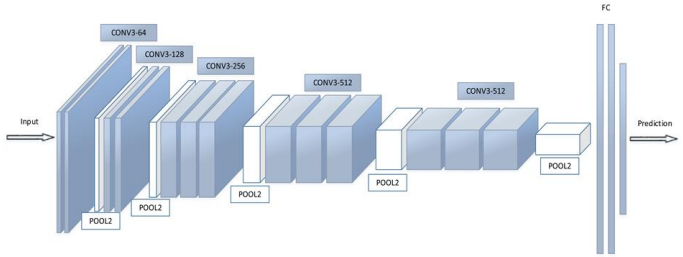
Object detection: R-CNN



- Image classification using transfer learning: use features from pretrained CNN (last-but-one layer) and train SVM/NN classifier
- R-CNN: similar approach with features computed from warped regions



Object detection: Fast R-CNN



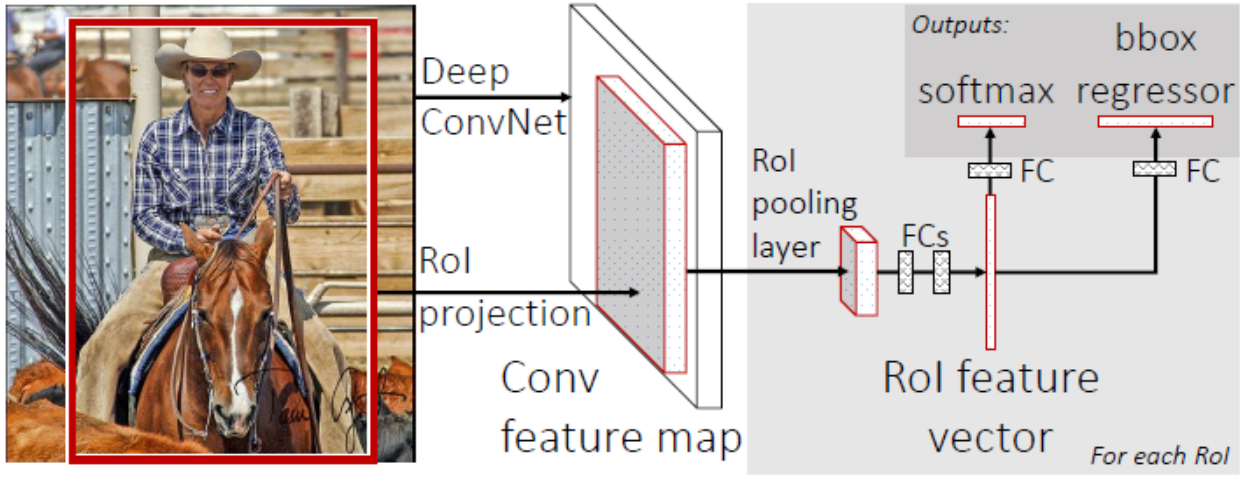


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

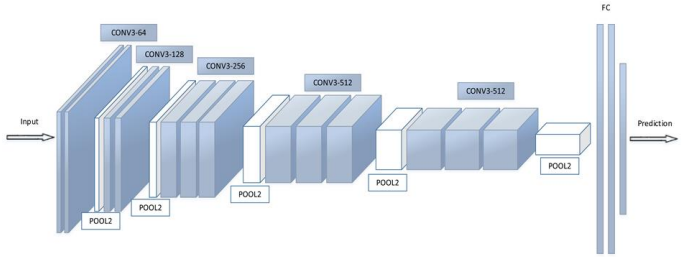
Advantages over R-CNN

- Compute conv layers only once
- Train the whole pipeline e2e at once

CNTK requirements

- RoI pooling node
- Axis aware error and CE
- Composite reader

CNTK example: Fast R-CNN




```
# Defines the Fast R-CNN network model for detecting objects in images
def frcn_predictor(features, rois, n_classes, model_path):
    # Load the pretrained classification net and find nodes
    loaded_model = load_model(model_path)

    # Clone the conv layers and the fully connected layers of the network
    conv_layers = base_model.clone([feature_node_name], [last_conv_node_name],
                                   clone_method=CloneMethod.freeze)
    fc_layers = base_model.clone([pool_node_name], [last_hidden_node_name],
                                 clone_method=CloneMethod.clone)

    # Create the Fast R-CNN model
    feat_norm = features - Constant(114)
    conv_out  = conv_layers(feat_norm)
    roi_out   = roipooling(conv_out, rois, (roi_dim, roi_dim))
    fc_out    = fc_layers(roi_out)

    # Add a dense layer for classification --- z = Dense(n_classes, activation=None)(fc_out)
    W = parameter(shape=(4096, n_classes))
    b = parameter(shape=n_classes, init=0)
    z = times(fc_out, W) + b
    return z
```

CNTK example: Fast R-CNN reader

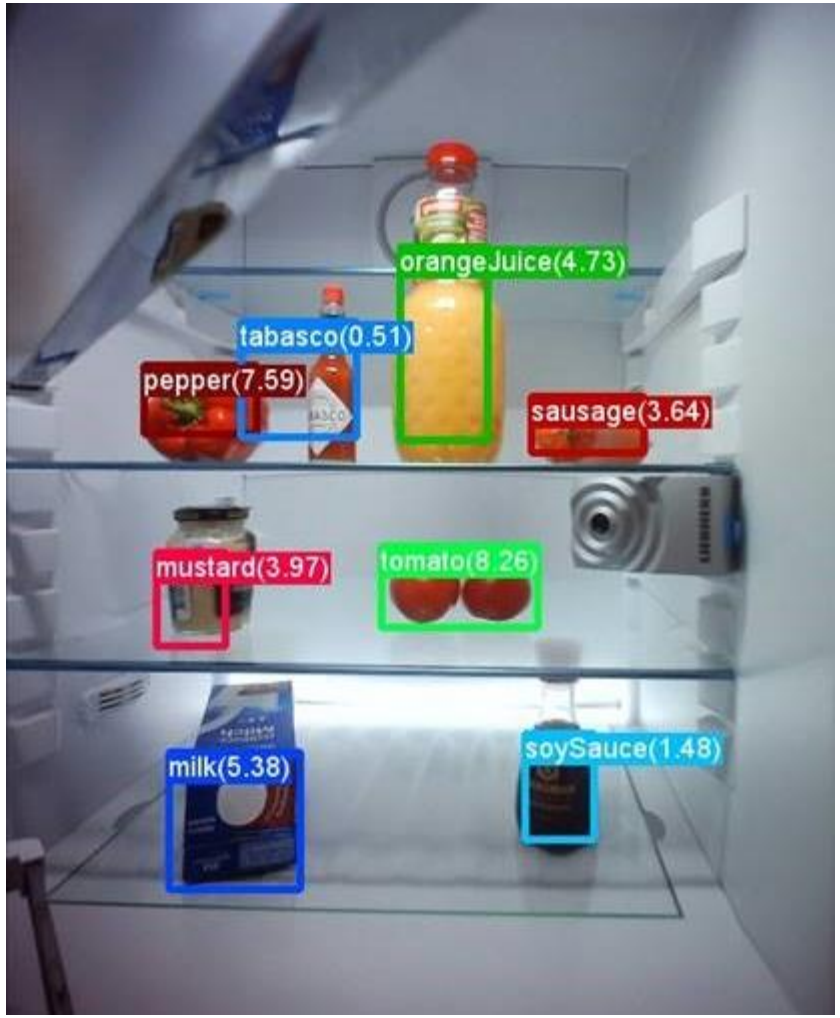
```
# read images
transforms = [scale(width=img_width, height=img_height, channels=img_channels,
                    scale_mode="pad", pad_value=114, interpolations='linear')]

image_source = ImageDeserializer(map_file, StreamDefs(
                  features = StreamDef(field='image', transforms=transforms)))

# read rois and labels
roi_source = CTFDeserializer(roi_file, StreamDefs(
                  rois = StreamDef(field=roi_stream_name, shape=rois_dim, is_sparse=False)))
label_source = CTFDeserializer(label_file, StreamDefs(
                  roiLabels = StreamDef(field=label_stream_name, shape=label_dim, is_sparse=False)))

# define a composite reader
return MinibatchSource([image_source, roi_source, label_source], randomize=data_set == "train")
```

CNTK Fast R-CNN in action

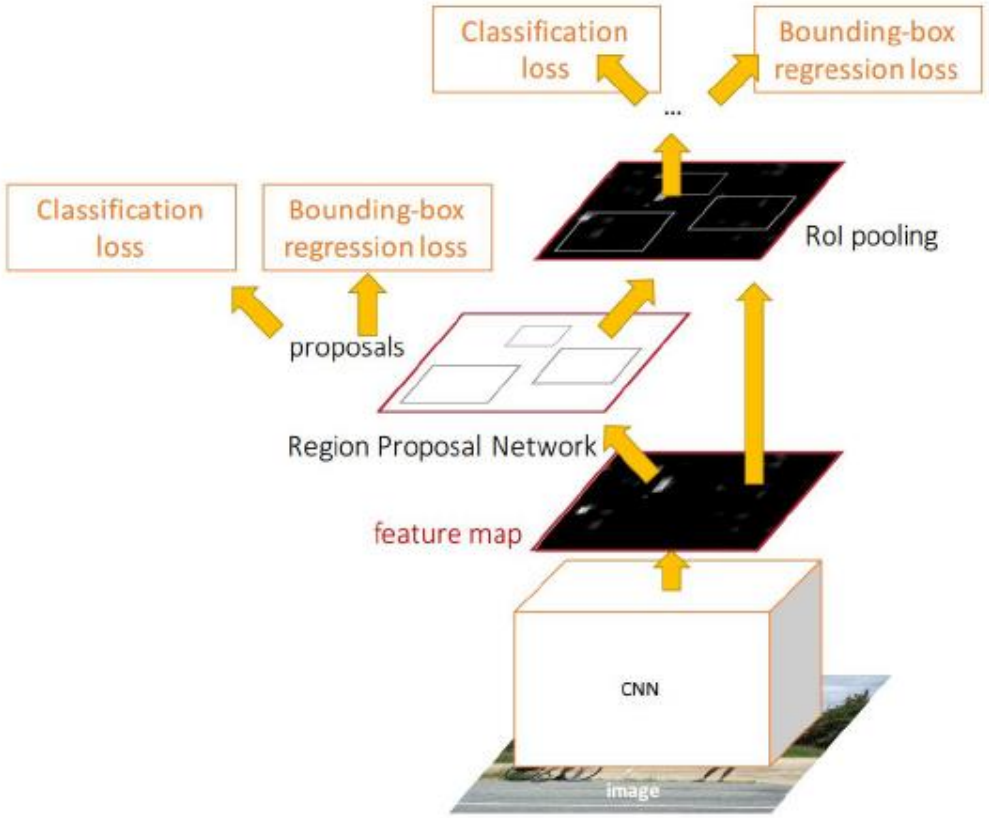
Liebherr und Microsoft, IFA 2016

Im Rahmen der IFA 2016 hat der Hausgerätehersteller Liebherr heute intelligente Kühlgeräte vorgestellt, die Kunden zukünftig auf Basis von Microsoft Technologien bei der Lagerung und Beschaffung von Lebensmitteln unterstützen. [...] Die Auswertung der Kamerabilder erfolgt über Deep-Learning-Technologien wie die Computer Vision API der Microsoft Cognitive Services und das Open-Source Computational Network Toolkit (CNTK).

[https://news.microsoft.com/de-de/liebherr-praesentiert-intelligente-kuehlgeraete-auf-basis-von-microsoft-azure-und-windows-10-iot/#sm.000012opr8wjiefadvdjggz0536ko]



Object detection: Faster R-CNN, Yolo



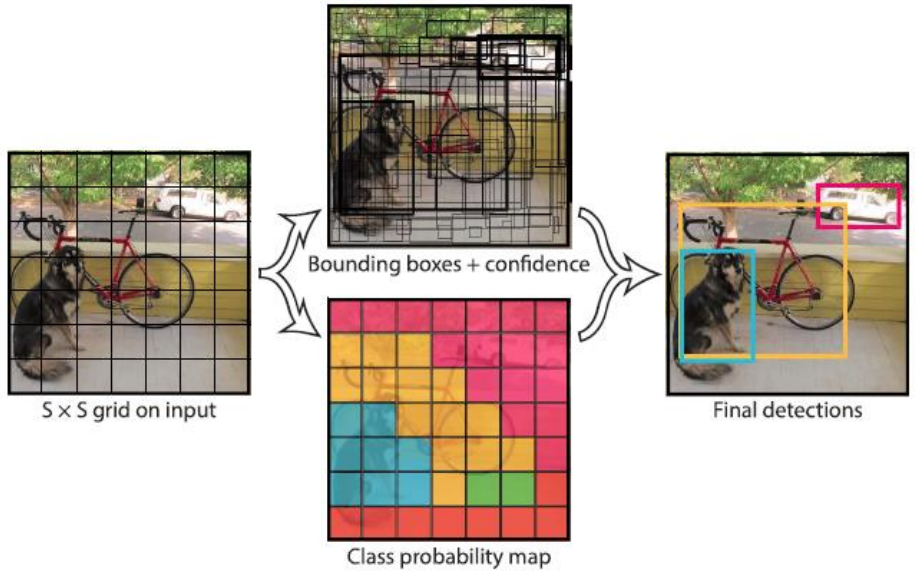


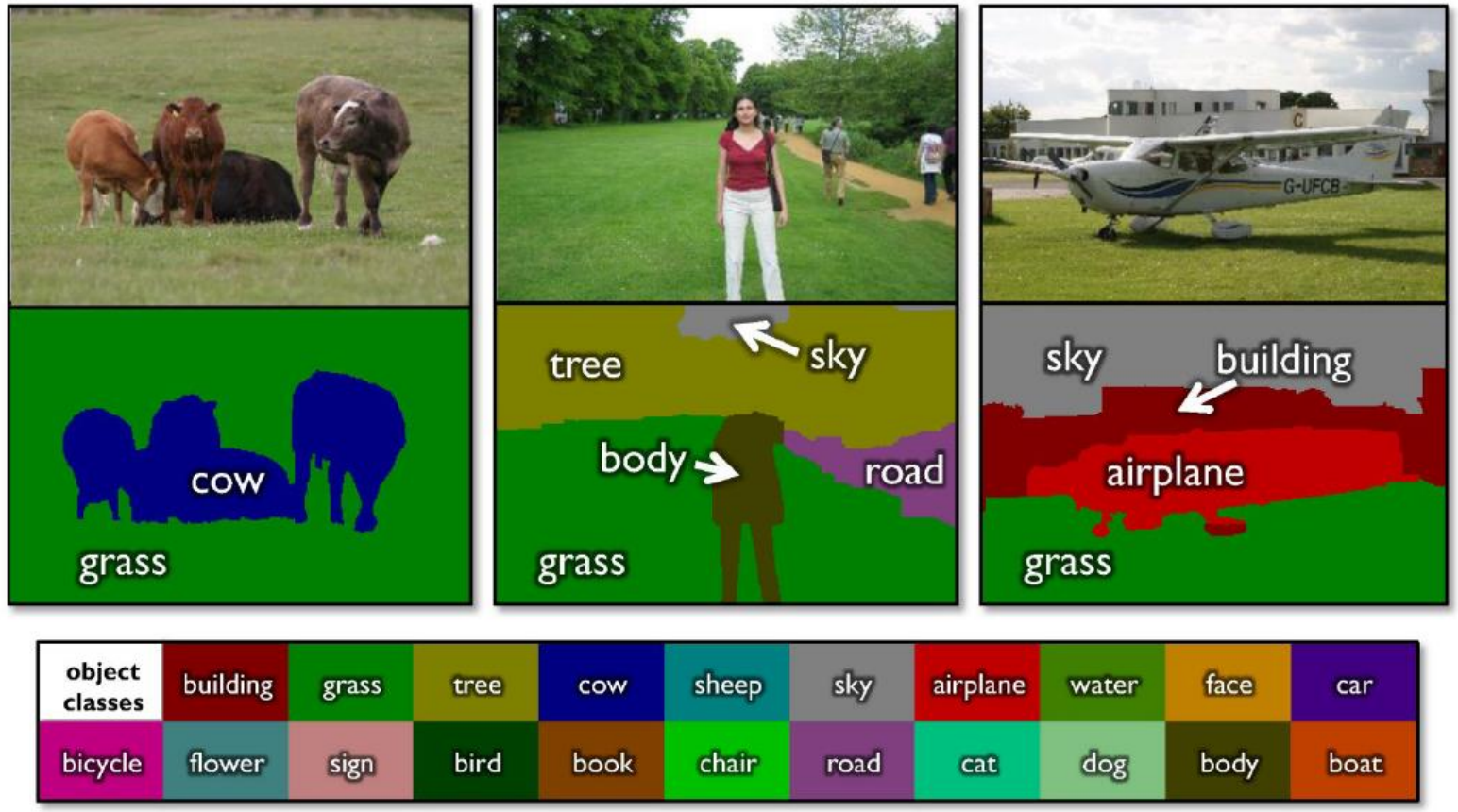
Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Semantic segmentation

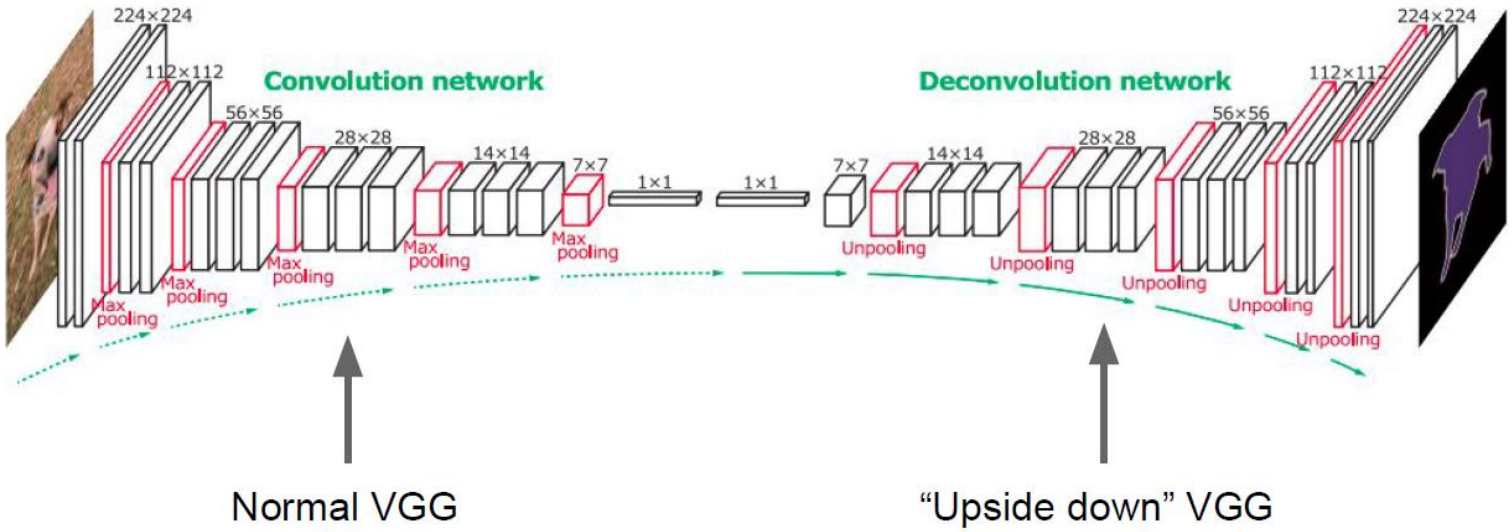
Label every pixel!

Don’t differentiate instances (cows)

Classic computer vision problem

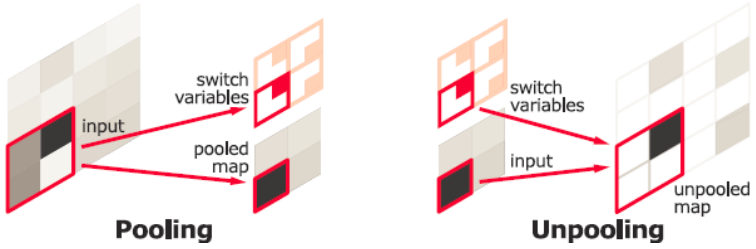


Semantic segmentation

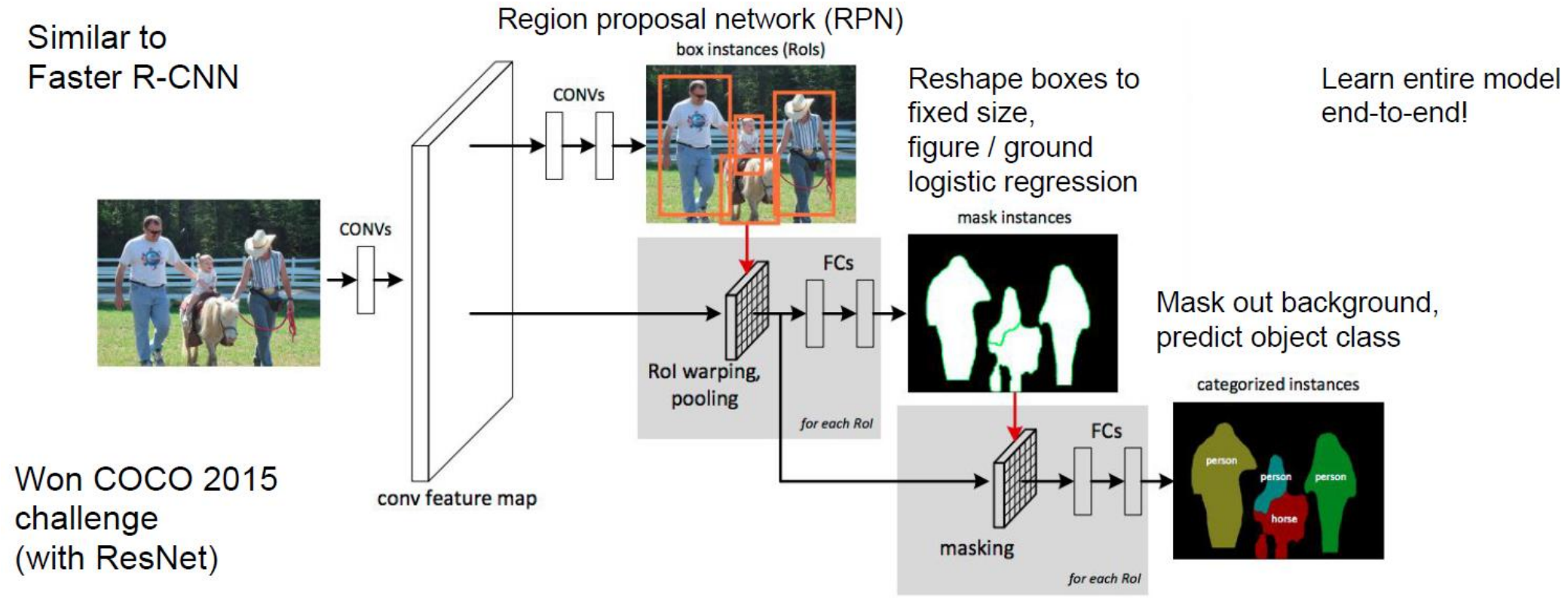


Requires

- Unpooling and Deconvolution nodes
- Crop node for deconvolution
- Multi-dimensional softmax (label per pixel) and loss
- Labels per pixel (reader)

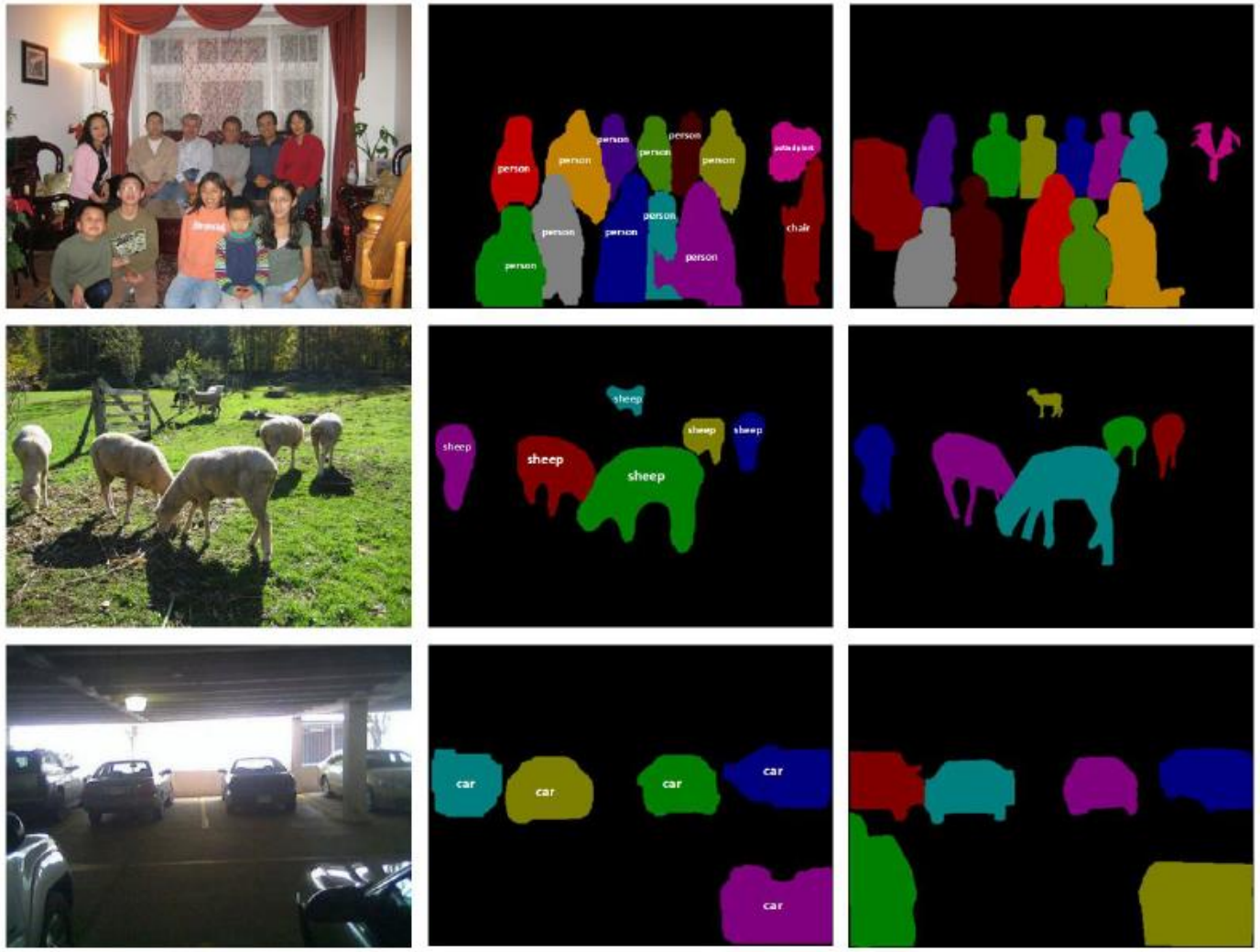


Instance segmentation (MSR 2015)



Dai et al, "Instance-aware Semantic Segmentation via Multi-task Network Cascades", arXiv 2015

Instance segmentation (MSR 2015) – results



I. what
II. how to
III. examples
IV. deep dive

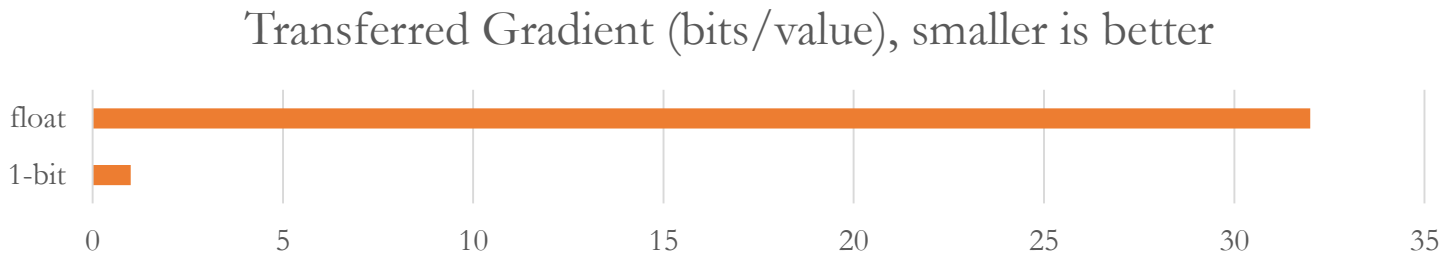
deep dive: data-parallel training

- data-parallelism: distribute each minibatch over workers, then aggregate
- challenge: communication cost
  - optimal iff
    compute and communication time per minibatch is equal (assuming overlapped processing)
- example: DNN, MB size 1024, 160M model parameters
  - compute per MB: 1/7 second
  - communication per MB: 1/9 second (640M over 6 GB/s)
  - can't even parallelize to 2 GPUs: communication cost already dominates!
- approach:
  - **communicate less** → 1-bit SGD
  - **communicate less often** → automatic MB sizing; Block Momentum

deep dive: 1-bit SGD

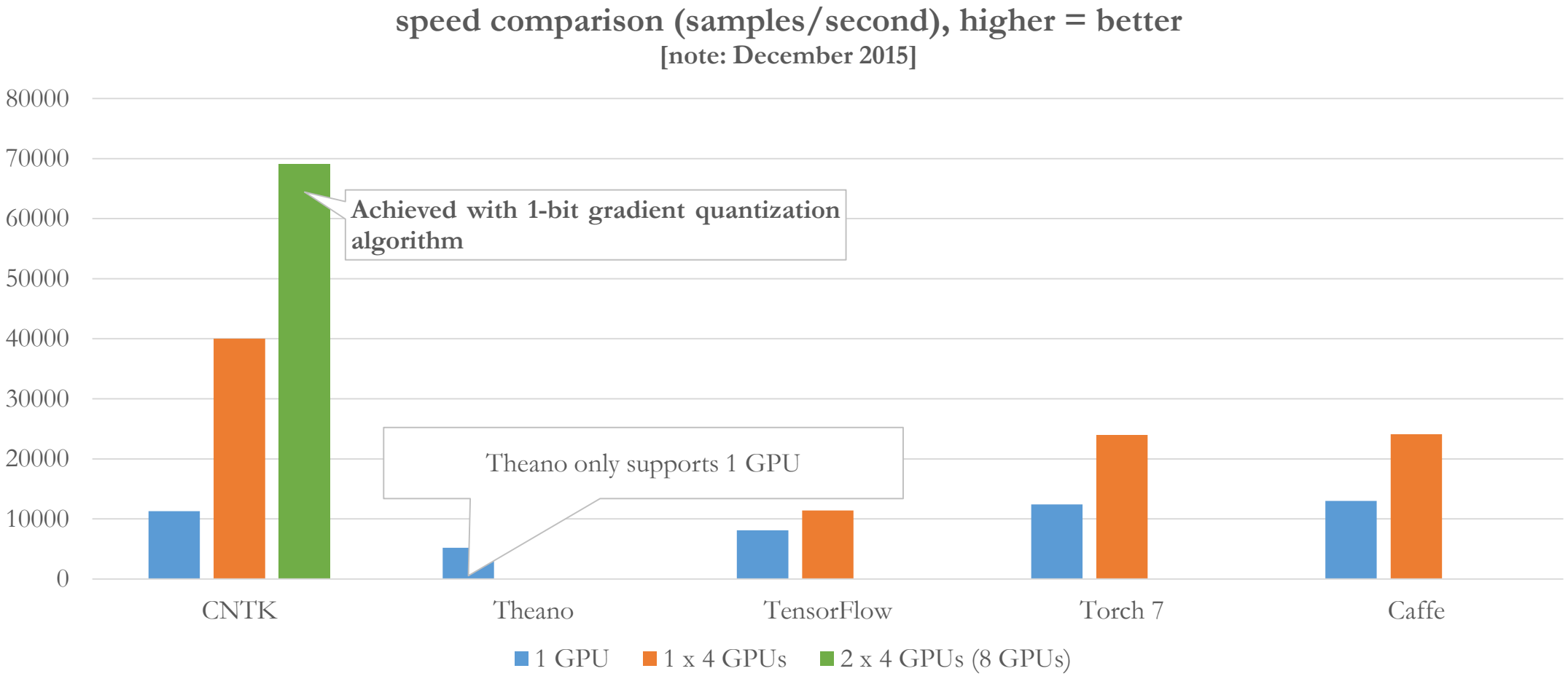
- quantize **gradients** to but **1 bit per value** with **error feedback**
  - carries over quantization error to next minibatch

$$G_{ij\ell}^{\text{quant}}(t) = \mathcal{Q}(G_{ij\ell}(t) + \Delta_{ij\ell}(t-N))$$
$$\Delta_{ij\ell}(t) = G_{ij\ell}(t) - \mathcal{Q}^{-1}(G_{ij\ell}^{\text{quant}}(t))$$



1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs, InterSpeech 2014, F. Seide, H. Fu, J. Droppo, G. Li, D. Yu

“CNTK is production-ready: State-of-the-art accuracy, efficient, and scales to multi-GPU/multi-server.”

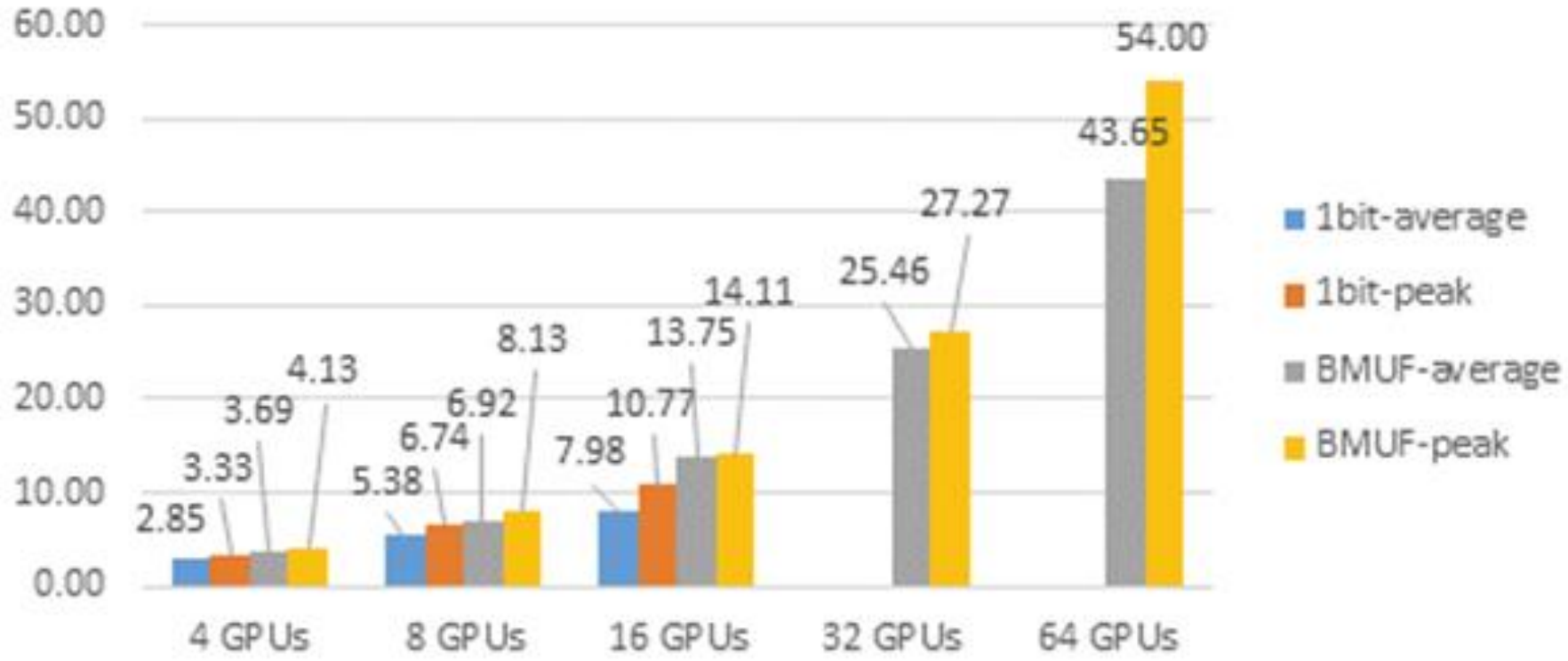


deep dive: Block Momentum

- very recent, very effective parallelization method
- goal: avoid to communicate after every minibatch
  - run a block of many minibatches without synchronization
  - then exchange and update with “block gradient”
- problem: taking such a large step causes divergence
- approach:
  - only add 1/K-th of the block gradient (K=#workers)
  - and carry over the missing (1-1/K) *to the next block update* (error residual like 1-bit SGD)
  - same as the common momentum formula

K. Chen, Q. Huo: “Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering,” ICASSP 2016

data-parallel training



LSTM SGD baseline	11.08				
Parallel Algorithms	4-GPU	8-GPU	16-GPU	32-GPU	64-GPU
1bit	10.79	10.59	11.02		
BMUF	10.82	10.82	10.85	10.92	11.08

Table 2: WERs (%) of parallel training for LSTMs

[Yongqiang Wang, IPG; internal communication]

example: plugging in BlockMomentum

- configure reader, network, learner
- train & evaluate

```
python my_cntk_script.py
```

example: plugging in BlockMomentum

- configure reader, network, learner
- train & evaluate

```
mpiexec --np 16 --hosts server1,server2,server3,server4    \
python my_cntk_script.py

# change my_cntk_script.py to use BlockMomentum (one-liner)
learner = block_momentum_distributed_learner(learner, block_size=block_size)
```

recap

- CNTK can parallelize training across multiple GPUs and multiple servers
- extremely easy to use: mpiexec + parallelTrain option
- enables to handle production workloads
- two methods based on error-feedback: 1-bit, Block Momentum

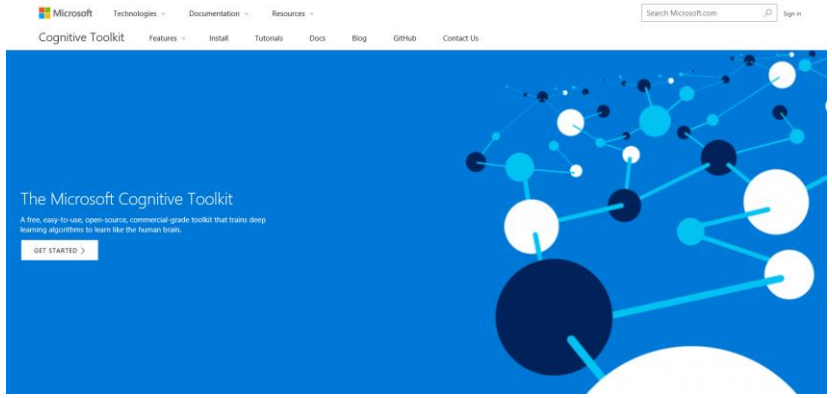
I. what
II. how
III. examples
IV. deep dive

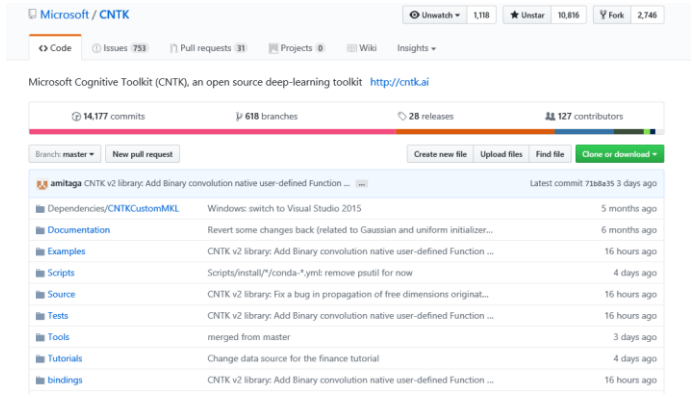
Conclusion

- ease of use
  - *what*, not *how* (sequence batching/unrolling, gradients, memory reuse, MPI)
  - Examples and tutorials: feed-forward DNN, recurrent, convolution, DSSM; speech, vision, text
- fast and scalable
  - best-in-class multi-GPU/multi-server algorithms (1-bit SGD, Block Momentum)
  - optimized for GPU and NVidia GPU libraries
- flexible
  - Python API, extensibility, support for C#, Java, Keras
- open-source, cross-platform and production-ready
  - Linux, Windows, docker, .Net, Azure, (ARM, FPGA)
  - growing use and contribution by various product teams
- internal=external version

Cognitive Toolkit resources

- Web site: `https://www.microsoft.com/en-us/cognitive-toolkit/`
- Github: `https://github.com/Microsoft/CNTK`
- Tutorials: `https://github.com/Microsoft/CNTK/tree/master/Tutorials`





The Microsoft Cognitive Toolkit

2017-6-1 • 1 min to read • Contributors all

The Microsoft Cognitive Toolkit - CNTK - is a unified deep-learning toolkit by Microsoft. This video provides a high-level overview of the toolkit.

The latest release of the Microsoft Cognitive Toolkit is 2.0.

CNTK can be included as a library in your Python or C++ programs, or used as a standalone machine learning tool through its own model description language (BrainScript). In addition you can use the CNTK model evaluation functionality from your C# or Java program.

CNTK supports 64-bit Linux or 64-bit Windows operating systems. To install you can either choose pre-compiled binary packages, or compile the toolkit from the source provided in GitHub.

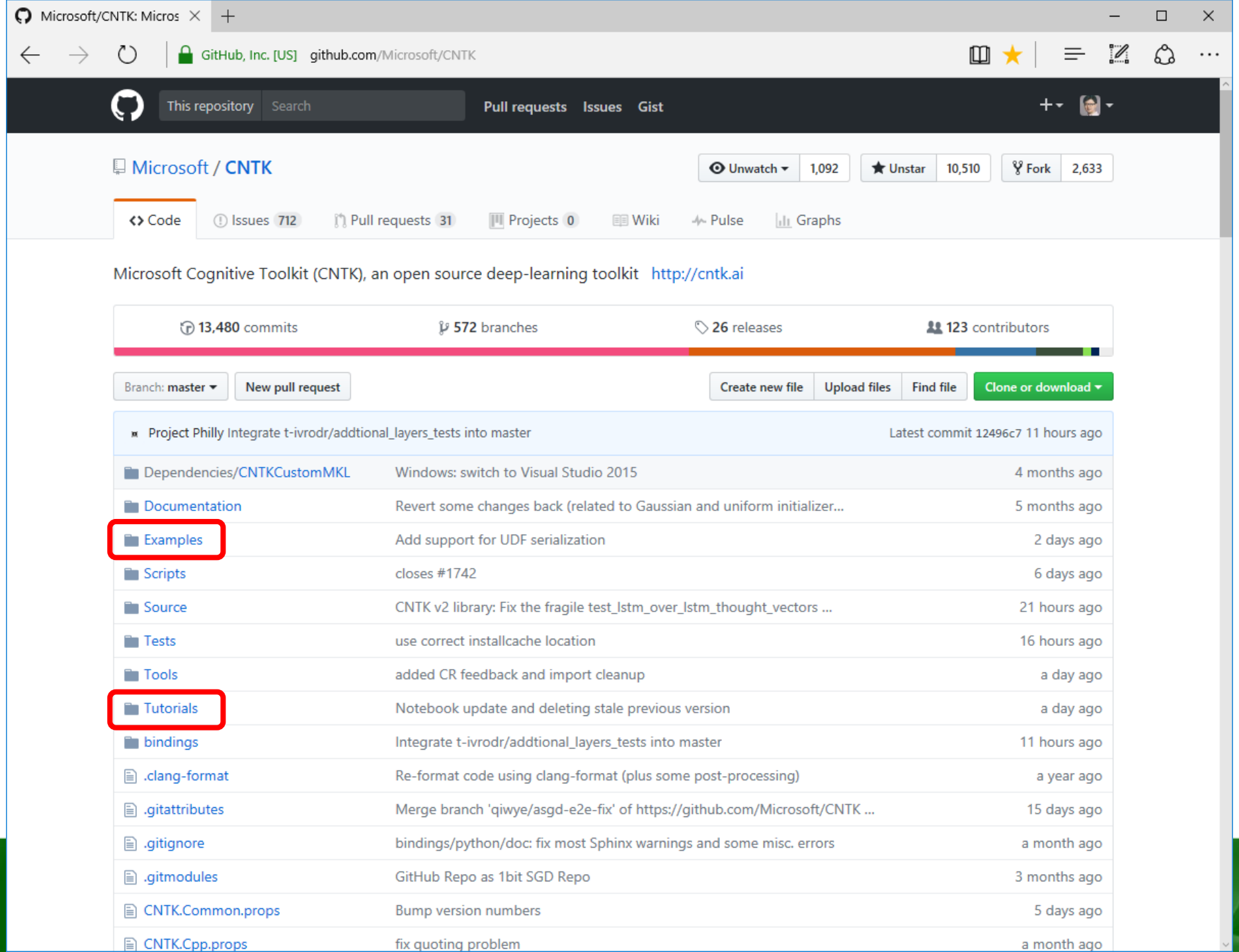
Here are a few pages to get started:

- Reasons to switch from TensorFlow to CNTK
- Setting up CNTK on your machine
- Tutorials, Examples, Tutorials on Azure
- The CNTK Library APIs
  - Using CNTK from Python
    - CNTK with Keras
  - Using CNTK from C++
- CNTK using BrainScript
- CNTK Model Evaluation)
- How to contribute to CNTK
- Give us feedback through these channels

Python API for CNTK (2.0)

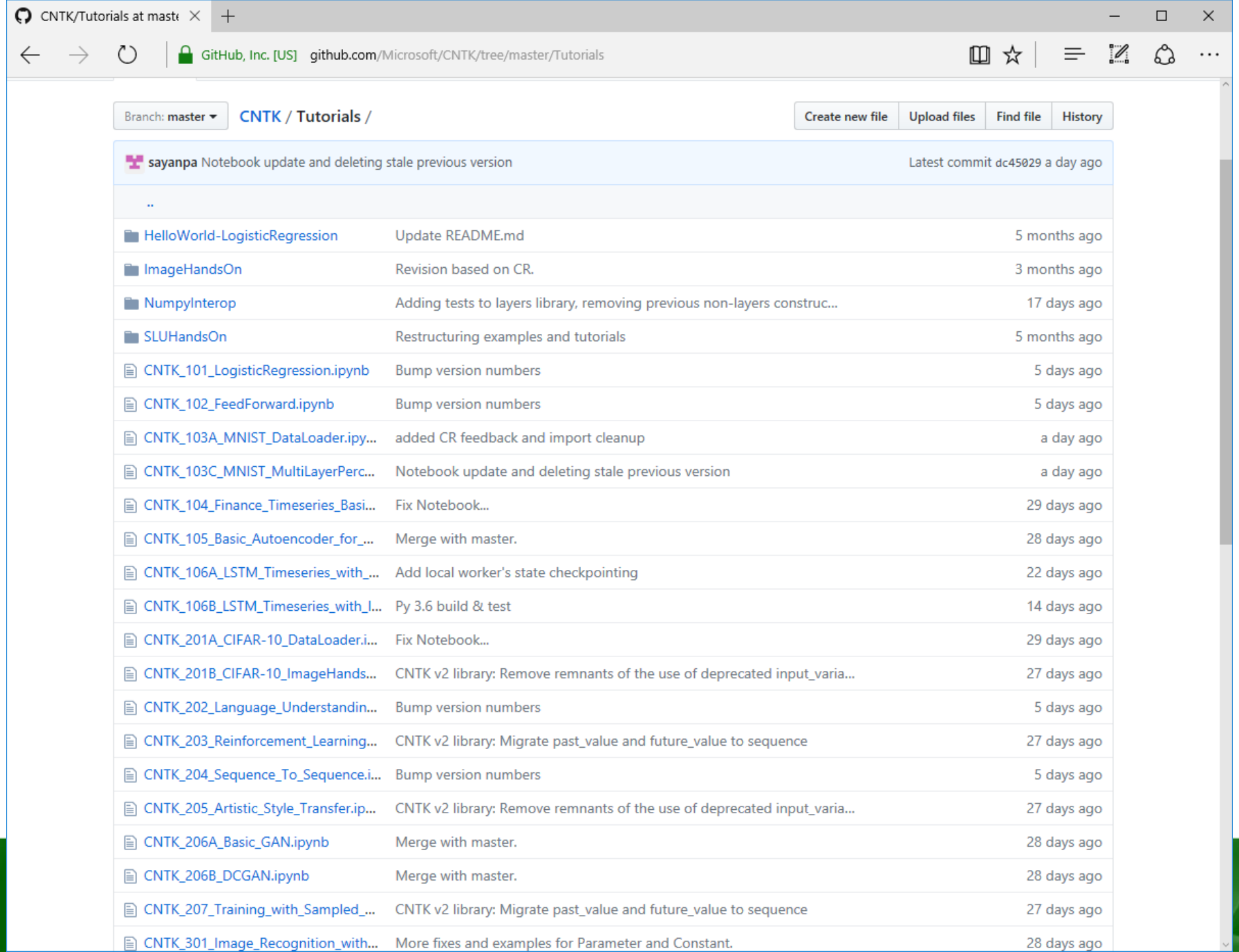
CNTK, the Microsoft Cognitive Toolkit, is a system for describing, training, and executing computational networks. It is also a framework for describing arbitrary learning machines such as deep neural networks (DNNs). CNTK is an implementation of computational networks that supports both CPU and GPU.

This page describes the Python API for CNTK version 2.0. This is an ongoing effort to expose such an API to the CNTK system, thus enabling the use of higher-level tools such as IDEs to facilitate the definition of computational networks, to execute them on sample data in real time. Please give feedback through these channels.



Microsoft Cognitive Toolkit

Microsoft



Microsoft Cognitive Toolkit

Microsoft

Thanks!

https://github.com/Microsoft/CNTK