FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Software Tools and Frameworks

Alexander Schindler

Information & Software Engineering
Institute of Information Systems Engineering
Faculty of Informatics
TU Wien
schindler@ifs.tuwien.ac.at

FFG
Forschung wirkt.

Agenda

- Overview
- Computation Graphs
- Deep Learning Frameworks
- Task specific Frameworks
- Hardware

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Overview

FFG
Forschung wirkt.

Overview

- How to choose the right Deep Learning framework?
- Depends on the task to solve
  - Highly experimental?
  - Only Experimenting?
  - Putting into production?
  - Large-scale deployment?
- Depends on the complexity of the models
  - Fine-tune pre-trained models?
  - Use adapt well-known architectures?
  - Small straigt forward models?
  - Complex architectures, custom loss functions or metrics, multi-task learning, etc.?

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Computation Graphs

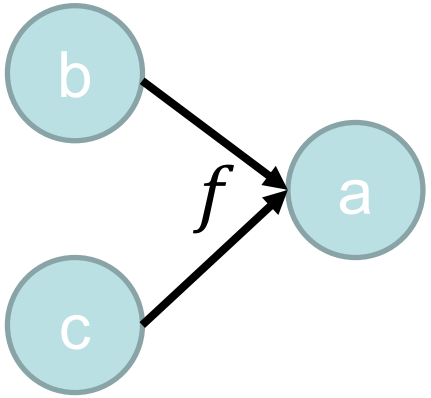
Static vs. Dynamic Graph Definitions

FFG
Forschung wirkt.

Computation Graphs

- Computational Graph
  - A “language” describing a function
- Model as Directed Acyclic Graph (DAG)
- Static Graph Definition
- Dynamic Graph Definition

$$a = f(b, c)$$



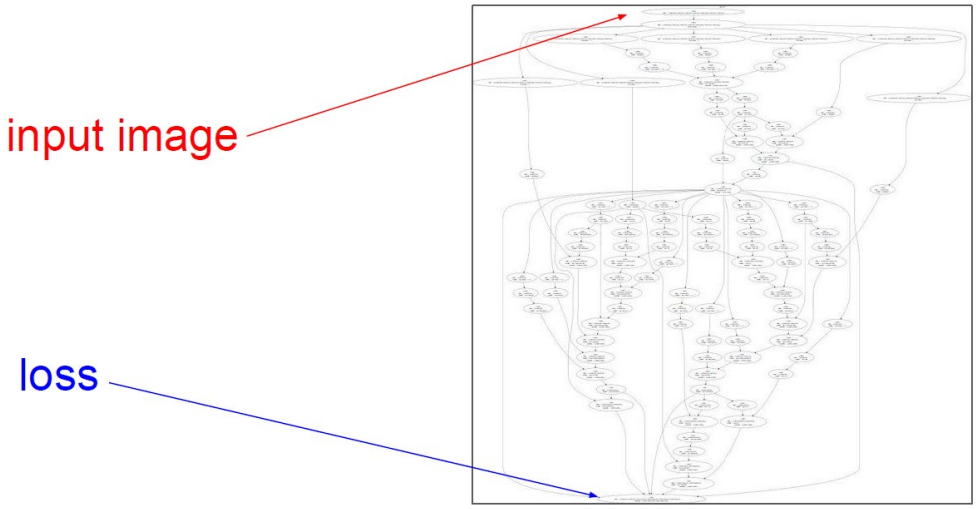


Figure reproduced with permission from a Twitter post by Andrej Karpathy.

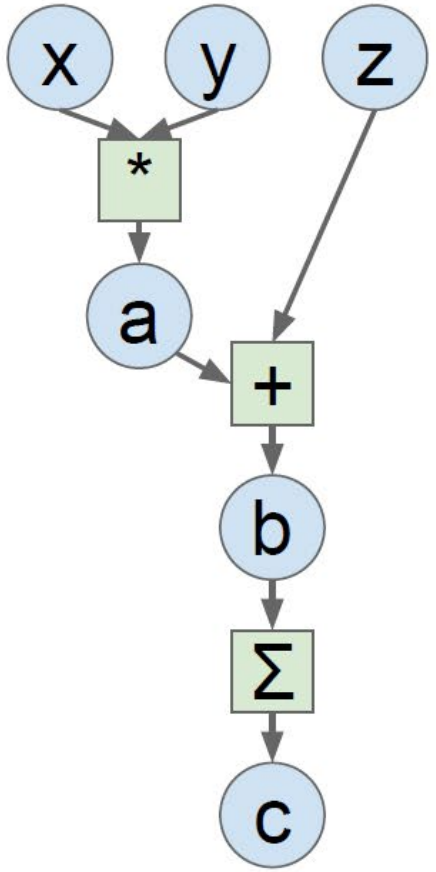
Main Responsibility of Deep Learning Frameworks

1. Efficiently build big computational graphs
2. Efficiently compute gradients in computational graphs
3. Run it all efficiently on GPUs

Computational Graphs

Numpy

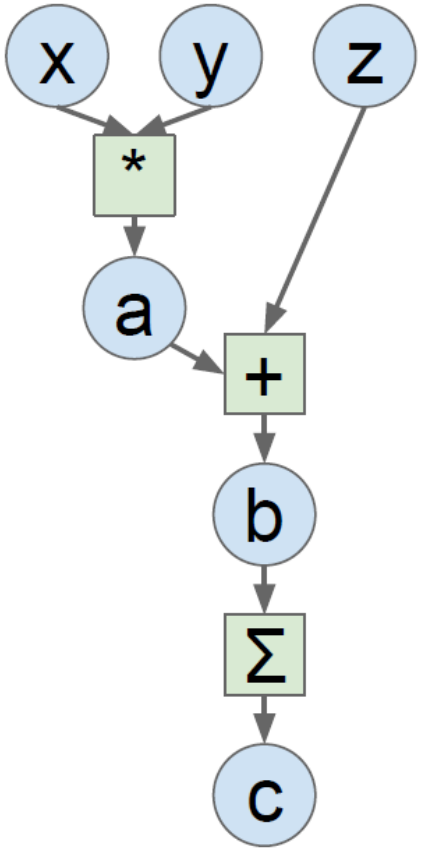
```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



TensorFlow

```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Computational Graphs

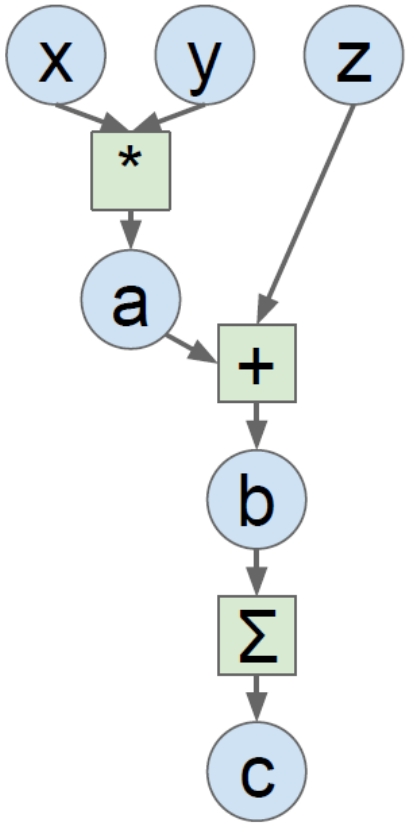


Create forward computational graph

TensorFlow

```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Computational Graphs

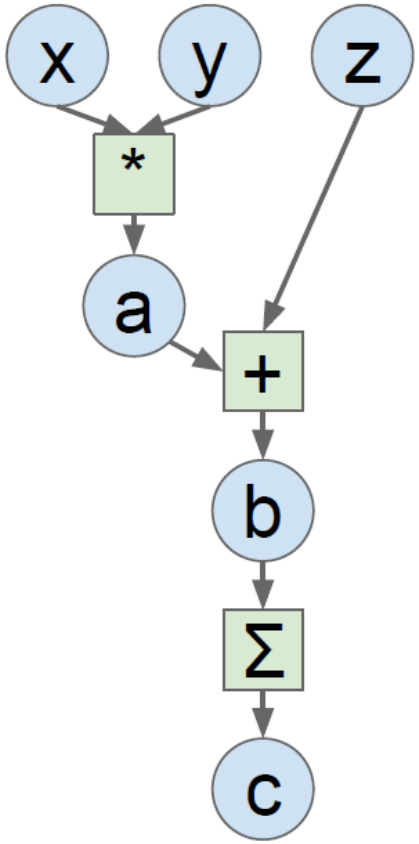


Ask TensorFlow to compute gradients

TensorFlow

```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Computational Graphs

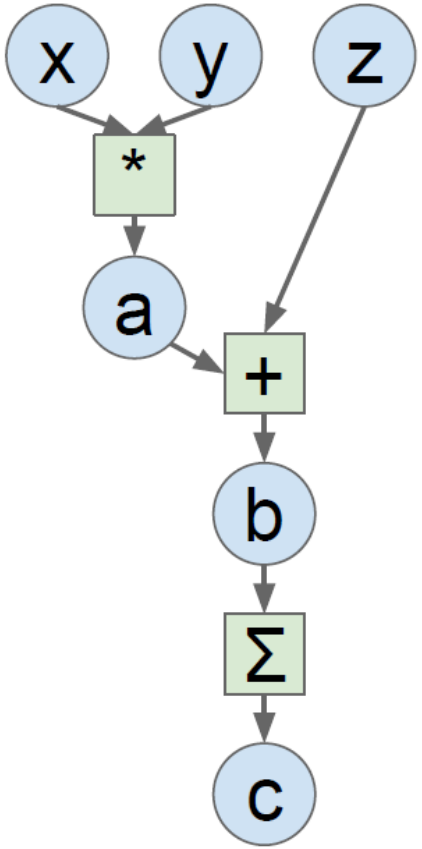


Tell TensorFlow to run on **GPU**

TensorFlow

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3000, 4000

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

Computational Graphs

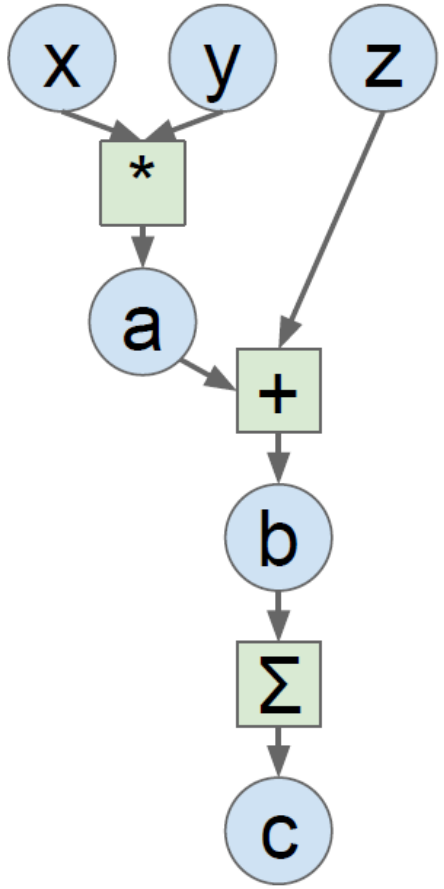


Define **Variables** to start building a computational graph

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D),
             requires_grad=True)
y = Variable(torch.randn(N, D),
             requires_grad=True)
z = Variable(torch.randn(N, D),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Computational Graphs

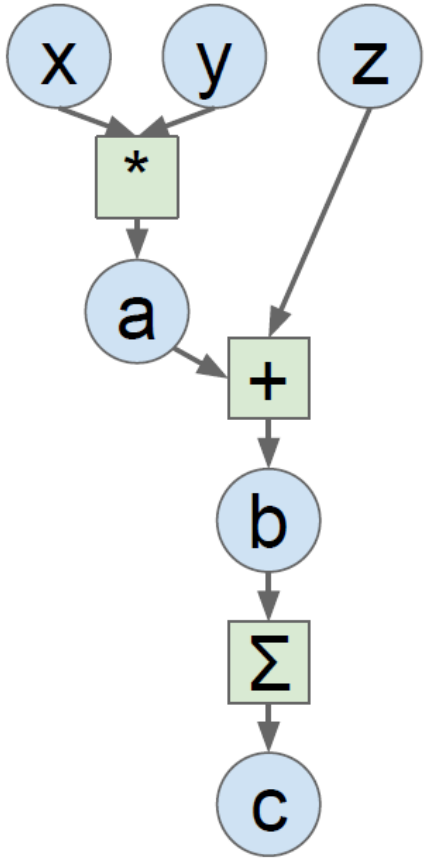


Forward pass looks just like numpy

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D),
             requires_grad=True)
y = Variable(torch.randn(N, D),
             requires_grad=True)
z = Variable(torch.randn(N, D),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Computational Graphs

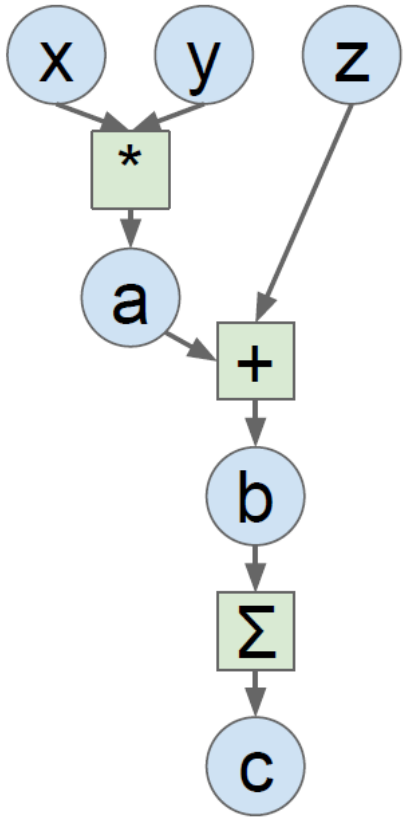


Calling c.backward() computes all gradients

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D),
             requires_grad=True)
y = Variable(torch.randn(N, D),
             requires_grad=True)
z = Variable(torch.randn(N, D),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Computational Graphs



Run on GPU by casting to .cuda()

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

TensorFlow

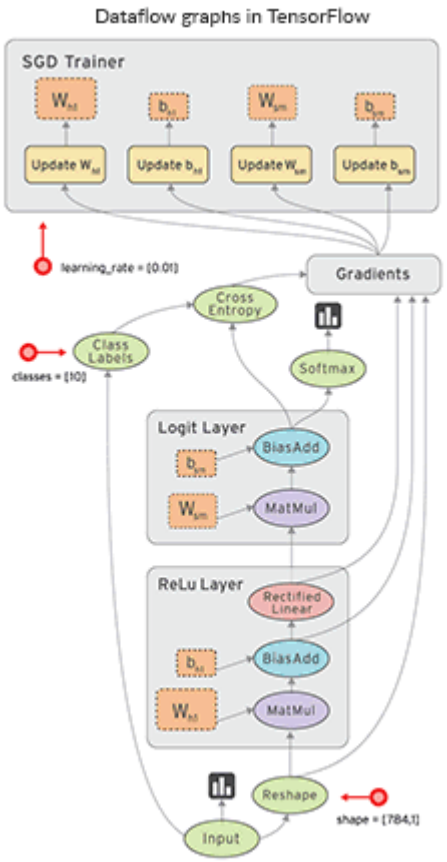
```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

Static Graph Definition

- Define graph statically before running the model
- All communication with outer world handled via
  - *Sessions*
  - *Placeholders*
    - Will be substituted at runtime



https://www.kdnuggets.com/2017/11/choosing-open-source-machine-learning-library.html

Dynamic Graph Definition

- Define computation graph at runtime
- Facilitates the usage of standard Python debuggers (pdb, ipdb, PyCharm, etc.)
- More imperative and dynamic
- No session interface or placeholders
- Define, change nodes on the go
- Integrates better into Python

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Deep Learning Frameworks

FFG
Forschung wirkt.

Deep Learning Frameworks

Top libraries by Github forks	
#1: 9004	tensorflow/tensorflow
#2: 6069	BVLC/caffe
#3: 1670	fchollet/keras
#4: 1367	Theano/Theano
#5: 1287	dmlc/mxnet
#6: 1252	torch/torch7
#7: 1076	Microsoft/CNTK
#8: 972	deeplearning4j/deeplearning4j
#9: 916	karpathy/convnetjs
#10: 585	Lasagne/Lasagne
#11: 376	NervanaSystems/neon
#12: 330	NVIDIA/DIGITS
#13: 312	tensorflow/skflow
#14: 281	pfnet/chainer
#15: 235	mila-udem/blocks
#16: 166	autumnai/leaf
#17: 148	tflearn/tflearn
#18: 99	IDSIA/brainstorm

Top libraries by Github contributors	
#1: 227	Theano/Theano
#2: 222	tensorflow/tensorflow
#3: 193	BVLC/caffe
#4: 174	fchollet/keras
#5: 129	dmlc/mxnet
#6: 96	torch/torch7
#7: 72	deeplearning4j/deeplearning4j
#8: 64	Microsoft/CNTK
#9: 54	pfnet/chainer
#10: 46	Lasagne/Lasagne
#11: 45	mila-udem/blocks
#12: 38	NervanaSystems/neon
#13: 24	NVIDIA/DIGITS
#14: 23	tensorflow/skflow
#15: 14	karpathy/convnetjs
#16: 14	autumnai/leaf
#17: 13	IDSIA/brainstorm
#18: 11	tflearn/tflearn

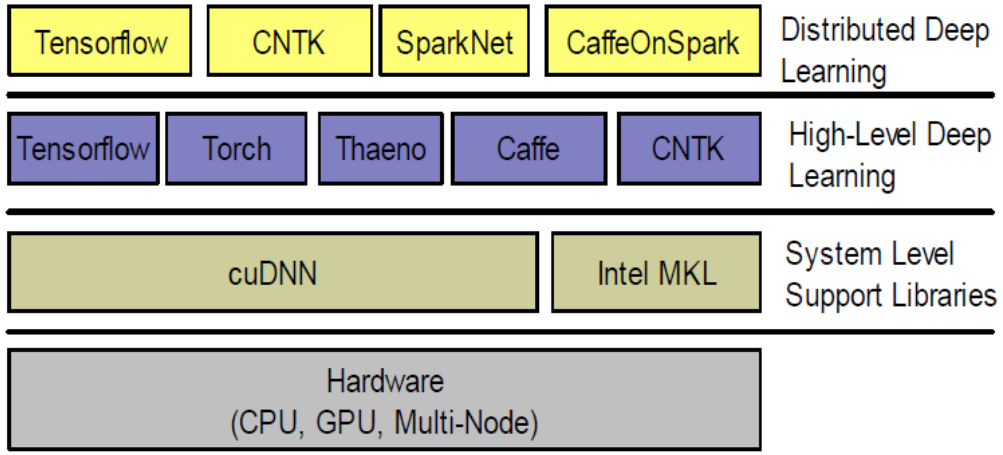
Top libraries by Github issues opened	
#1: 2587	BVLC/caffe
#2: 1946	fchollet/keras
#3: 1556	tensorflow/tensorflow
#4: 1537	Theano/Theano
#5: 1212	dmlc/mxnet
#6: 823	deeplearning4j/deeplearning4j
#7: 484	mila-udem/blocks
#8: 419	Microsoft/CNTK
#9: 412	pfnet/chainer
#10: 393	NVIDIA/DIGITS
#11: 357	Lasagne/Lasagne
#12: 304	torch/torch7
#13: 195	NervanaSystems/neon
#14: 105	tensorflow/skflow
#15: 81	IDSIA/brainstorm
#16: 56	tflearn/tflearn
#17: 43	autumnai/leaf
#18: 41	karpathy/convnetjs

Aggregate popularity (30•contrib + 10•issues + 5•forks)•10e-3	
#1: 67.24	tensorflow/tensorflow
#2: 62.01	BVLC/caffe
#3: 33.03	fchollet/keras
#4: 29.02	Theano/Theano
#5: 22.43	dmlc/mxnet
#6: 15.25	deeplearning4j/deeplearning4j
#7: 12.18	torch/torch7
#8: 11.49	Microsoft/CNTK
#9: 7.88	Lasagne/Lasagne
#10: 7.37	mila-udem/blocks
#11: 7.15	pfnet/chainer
#12: 6.30	NVIDIA/DIGITS
#13: 5.41	karpathy/convnetjs
#14: 4.97	NervanaSystems/neon
#15: 3.30	tensorflow/skflow
#16: 1.70	IDSIA/brainstorm
#17: 1.68	autumnai/leaf
#18: 1.63	tflearn/tflearn

François Chollet, https://twitter.com/fchollet/status/765212287531495424

Deep Learning Frameworks

- Deep Learning Frameworks are High-Level libraries
  - On top of a Deep Learning Stack
    - CUDA
    - CuDNN



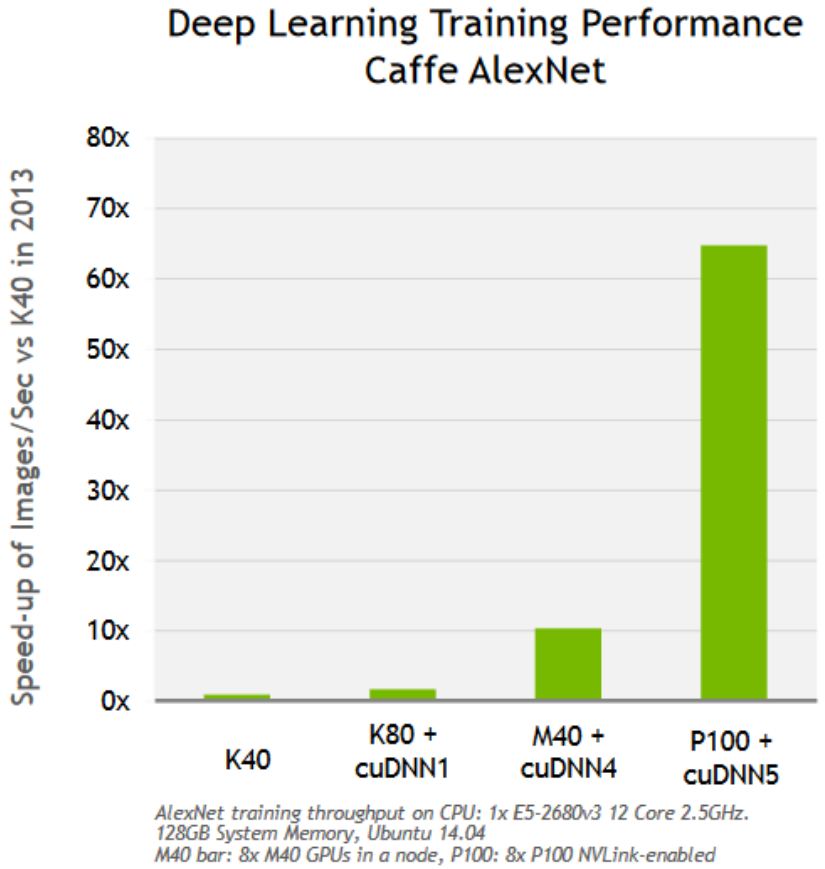
Pandey, H. M., & Windridge, D. (2019). A comprehensive classification of deep learning libraries. In *Third International Congress on Information and Communication Technology* (pp. 427-435). Springer, Singapore.

Compute Unified Device Architecture (CUDA)

- Parallel computing plattform by NVIDIA
- makes using a GPU for general purpose computing simple
- Runs only on NVIDIA hardware
- Responsible for
  - Memory management
  - Downloading and readbacks to and from the GPU
- Includes many accelerated libraries
  - Cublas, Cufft, Curand
- Other standards
  - OpenCL
    - Runs everywhere, incl. NVIDIA hardware
    - Not as fast as CUDA

CuDNN

- cuDNN is a library of primitives for deep learning
- High performance building blocks for deep learning frameworks
- Drop-in acceleration for widely used deep learning frameworks such as Caffe, CNTK, Tensorflow, Theano, Torch and others
- Accelerates industry vetted deep learning algorithms, such as convolutions, LSTM, fully connected, and pooling layers
- Fast deep learning training performance tuned for NVIDIA GPUs



DEEP LEARNING FRAMEWORKS

Tensorflow

Developers	Google Brain
Initial release	Nov. 2015
Stable Release	r.1.13 / 2.0-alpha0
Programming Languages	Python, C++, Javascript, Go, Java, Swift
Platform	Linux, macOS, Windows, Android, Raspberry Pi 0-3
License	Apache 2.0 open-source license
Repository	https://github.com/tensorflow/tensorflow
Website	https://www.tensorflow.org/

Image source: wikipedia

Tensorflow

Pros

- Actively used at Google for Research and Production
- Visualization Tool *Tensorboard*
- Easy production-ready model sharing
- Multiple language support
- Good Documentation
- Great Community
  - Almost every problem already discussed on Stackoverflow
- Tensorflow Lite enables on-device inference with low latency for mobile devices
- High-Level API

Cons

- Cluttered code
  - Inconsistent naming conventions (tf.nn.conv2d, tf.layers.conv2d, tf.layers.Conv2d, …)
- Tensorflow equivavlents for basic code such as for-loops
- Significant learning curve
- Frequent releases

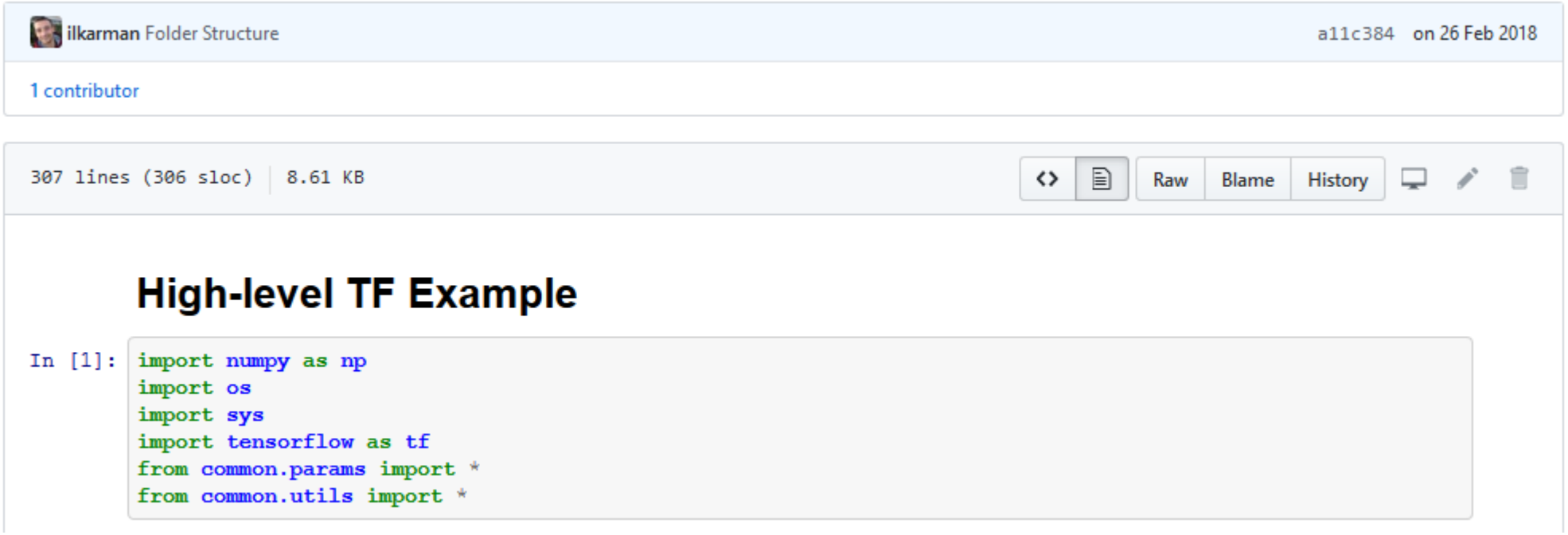
Tensorflow - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks



Tensorflow - Tensorboard

Pros

- Actively used at Google for Research and Production
- Great Visualization Tool Tensorboard

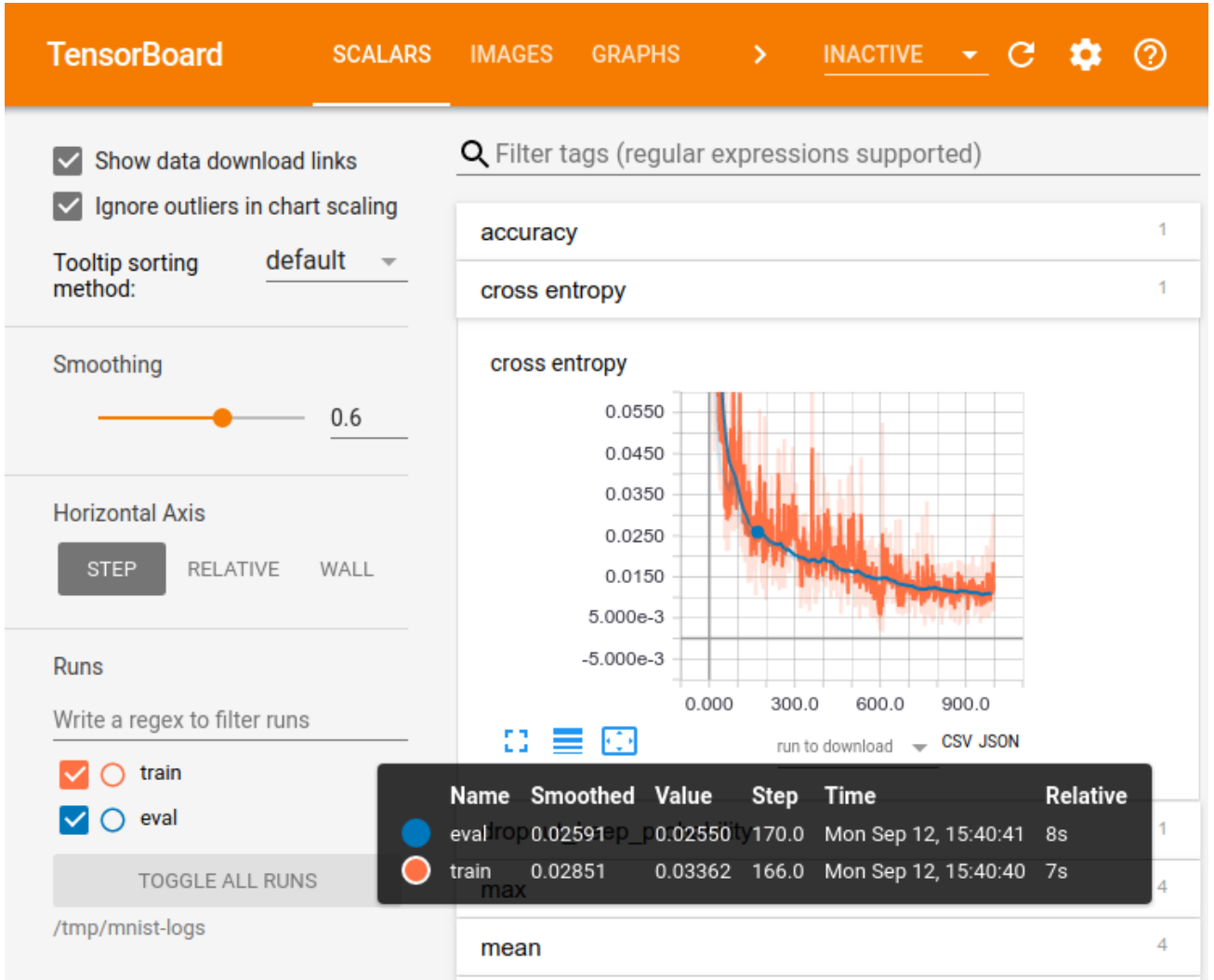
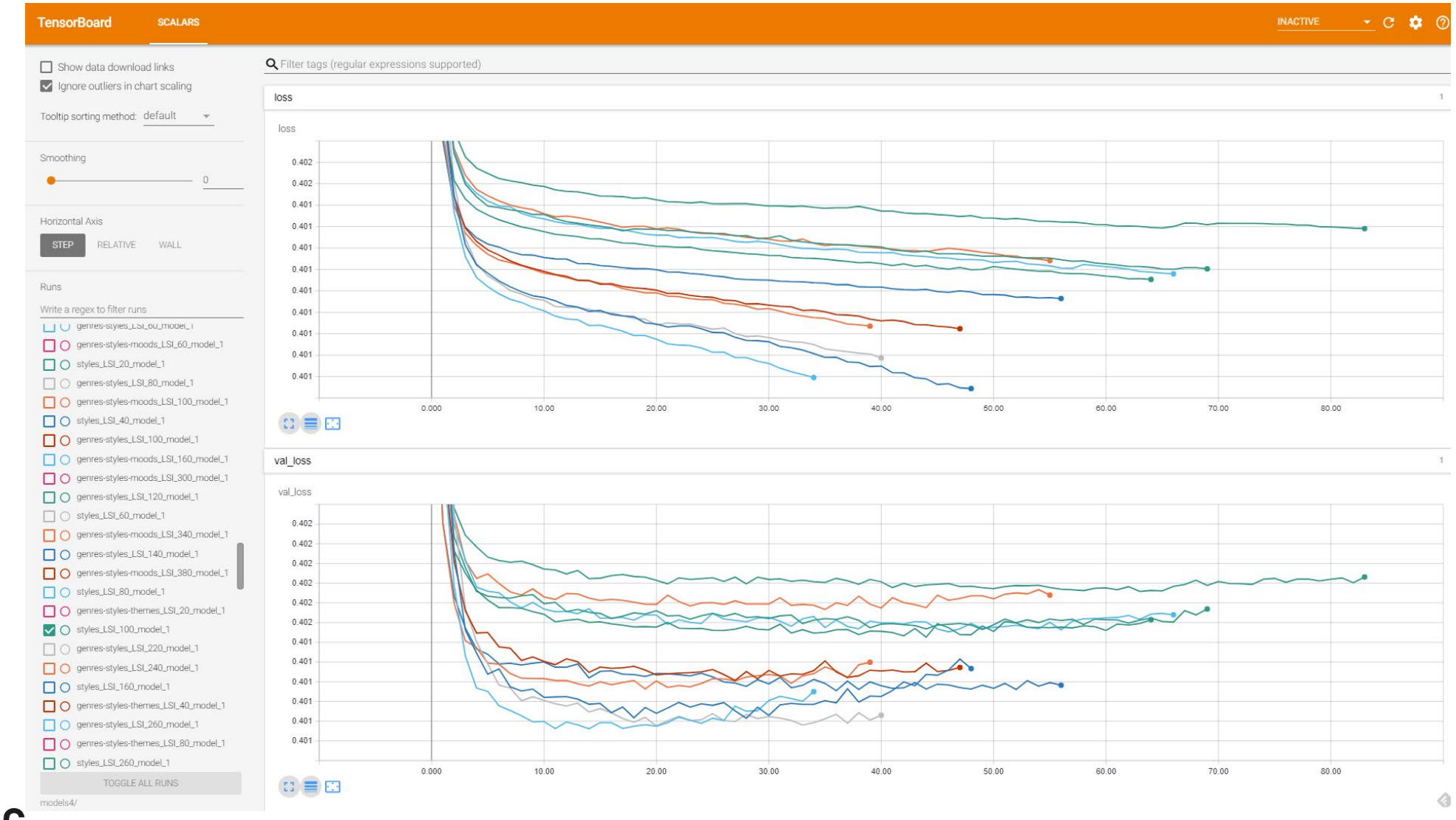


Image source: https://www.tensorflow.org/guide/summaries_and_tensorboard

Tensorflow - Tensorboard

- Compare different
  - Models
  - Parametrization
  - Learning rates
  - Etc.



Tensorflow - Tensorboard

- Visualize and inspect Network Graph

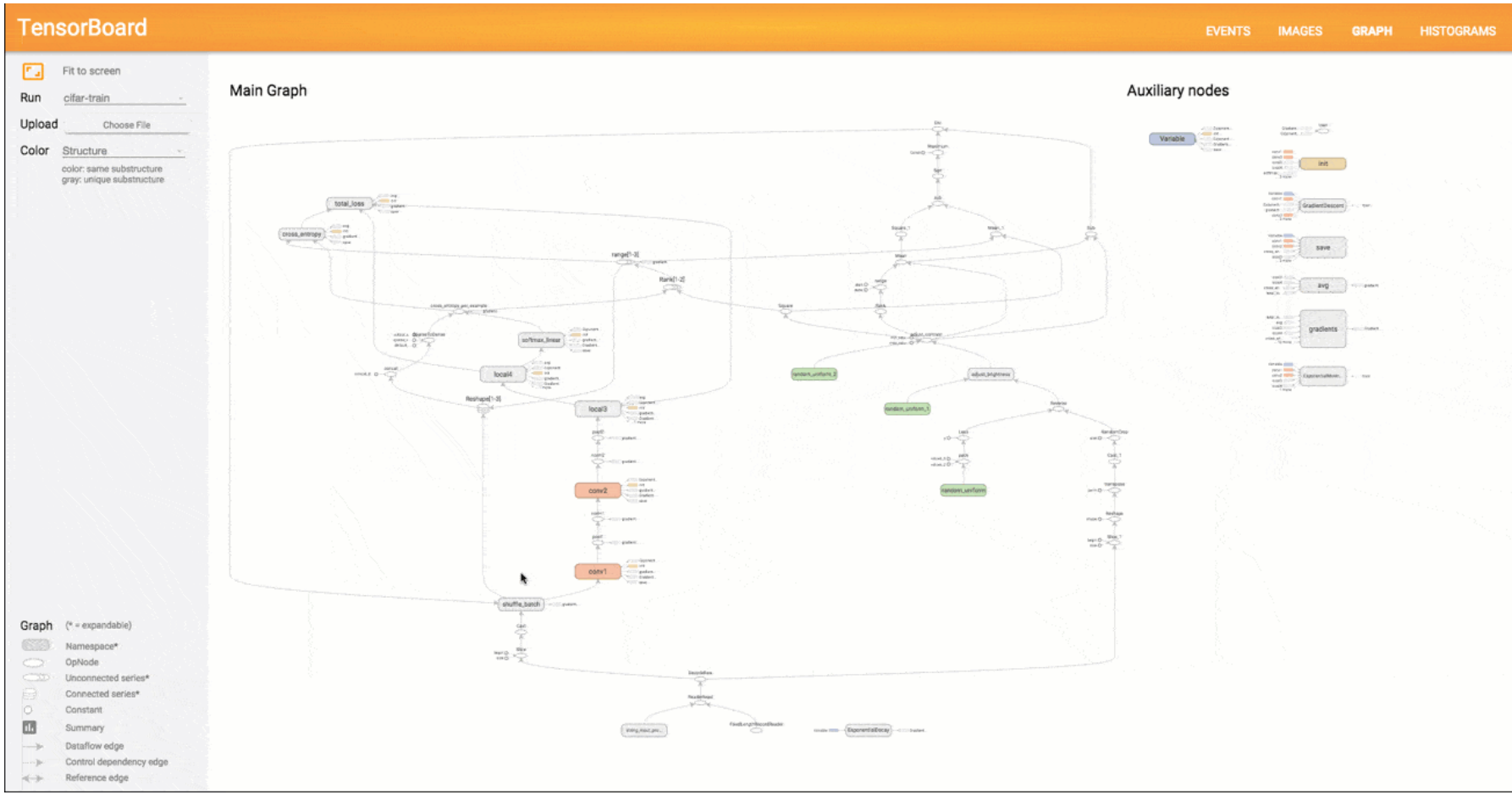


Image source: https://www.tensorflow.org/guide/summaries_and_tensorboard

Tensorflow - Tensorboard

- Create Custom Visualizations
  - Featuremaps
  - Kernels

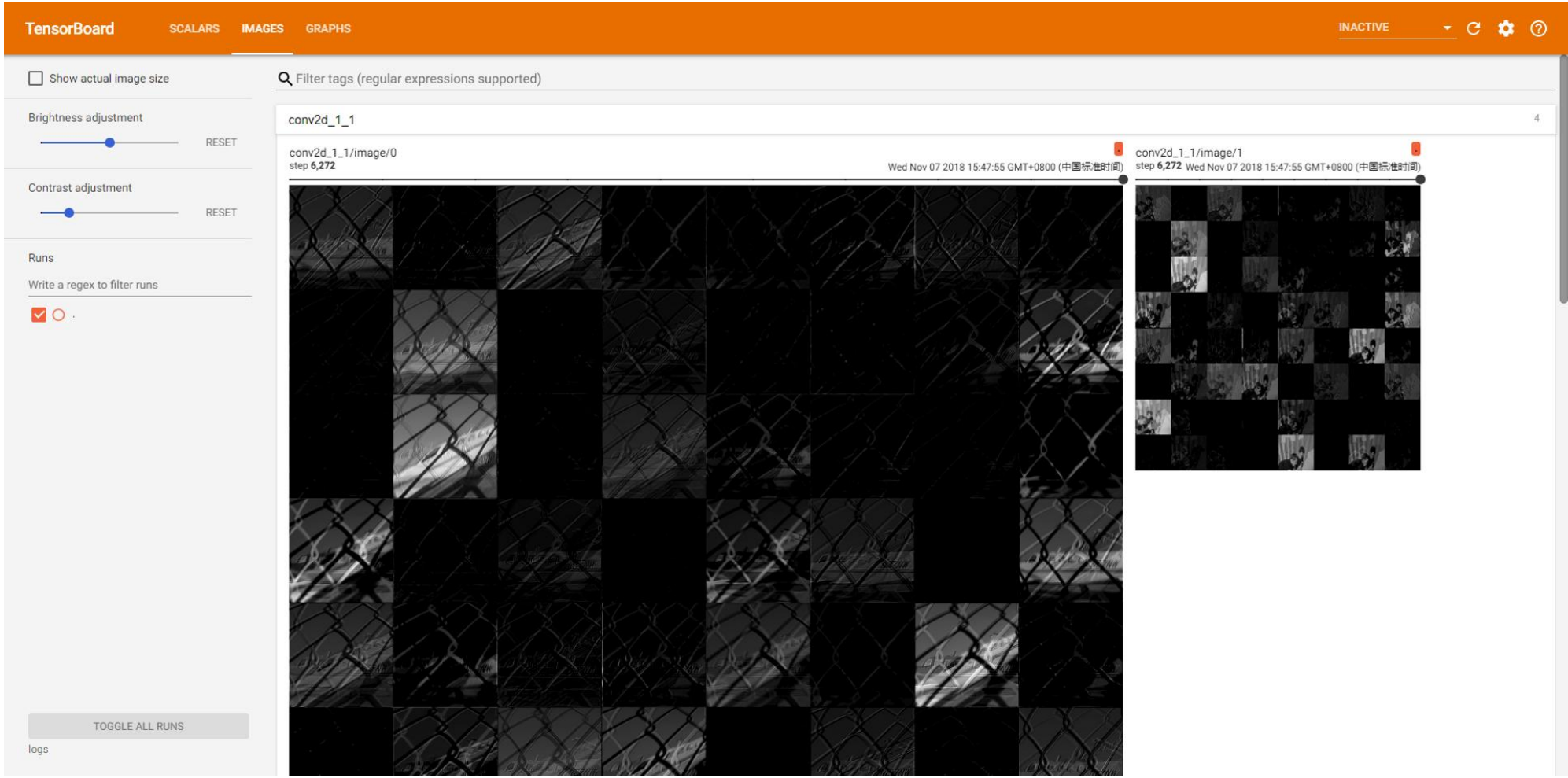


Image source: http://www.programmersought.com/article/5738128912/

Tensorflow - Tensorboard

In Tensorflow

```
# define from which variables generate summary from
loss_summary  = tf.summary.scalar('loss_1', loss1)
image_summary = tf.summary.image('generated_image', result)

# merge whole summary into one instruction
summary = tf.summary.merge([loss_summary, image_summary])

# define summary writer
summary_writer = tf.summary.FileWriter('path/to/summary/`,
                                       graph=tf.Session().graph)

# run summary along computation
result, summary_values = tf.Session().run([network_output, summary],
                                          feed_dict={: input_data})

#write summary to disk and view it in TensorBoard
summary_writer.add_summary(summary_values)
```

In Keras

```
# define model
model = Sequential()
model.add(Dense(10, input_dim=784))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')


# define Tensorboard Callback
tb_callback = keras.callbacks.TensorBoard(
                    log_dir                = './logs`,
                    histogram_freq         = 0,
                    batch_size             = 32,
                    write_graph            = True,
                    write_grads            = False,
                    write_images           = False,
                    embeddings_freq        = 0,
                    embeddings_layer_names = None,
                    embeddings_metadata    = None,
                    embeddings_data        = None,
                    update_freq            = 'epoch`)

# train model with automatic Tensorboard logging
model.fit(x_train, y_train, batch_size=128,
          epochs=20, verbose=0,
          callbacks=[tb_callback])
```

Image source: https://www.tensorflow.org/guide/summaries_and_tensorboard

Tensorboard

- tensorboardX
  - PyTorch Interface to Tensorboard
  - https://github.com/lanpa/tensorboardX

Microsoft Cognitive Toolkit (CNTK)

Developers	Microsoft Research
Initial release	Jan 2016
Stable Release	2.7
Programming Languages	C++, C#/.NET, Python, Java
Platform	Windows, Linux
License	MIT
Repository	https://github.com/Microsoft/CNTK
Website	https://docs.microsoft.com/en-us/cognitive-toolkit/

Microsoft Cognitive Toolkit (CNTK)

Pros

- Fast!
  - a lot of highly optimized components
  - Especially fast RNN implementations
    - tf.contrib.cudnn_rnn cought up
- Passing sequences that vary in length
  - Would require padding, masking, etc. in Tensorflow, also partially in Pytorch
- good scalability
  - very efficient in terms of resource usage
- Static Graphs, but better communication with Numpy
- supports simple integration with Azure Cloud

Cons

- Cumbersome to install
  - Especially on Linux Systems
- Limited community support

Microsoft Cognitive Toolkit (CNTK)

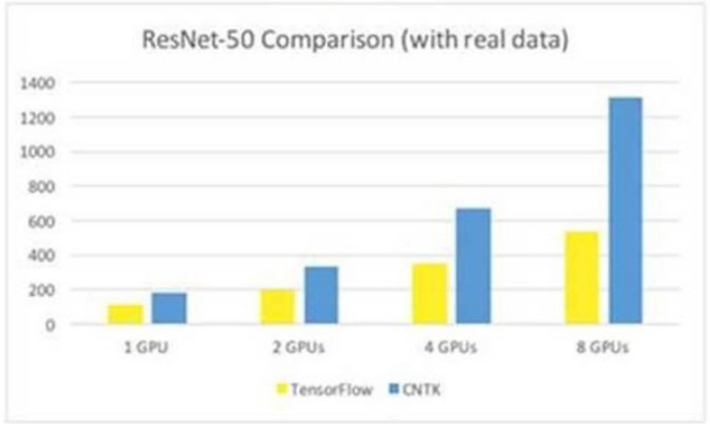
CNTK Speed

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

Philipp Kranen, Microsoft Research. Microsoft Cognitive Toolkit (CNTK). 12th Vienna Deep Learning Meetup. 20.06.2017. https://github.com/vdlm/meetups/tree/master/Meetups/Meetup_12

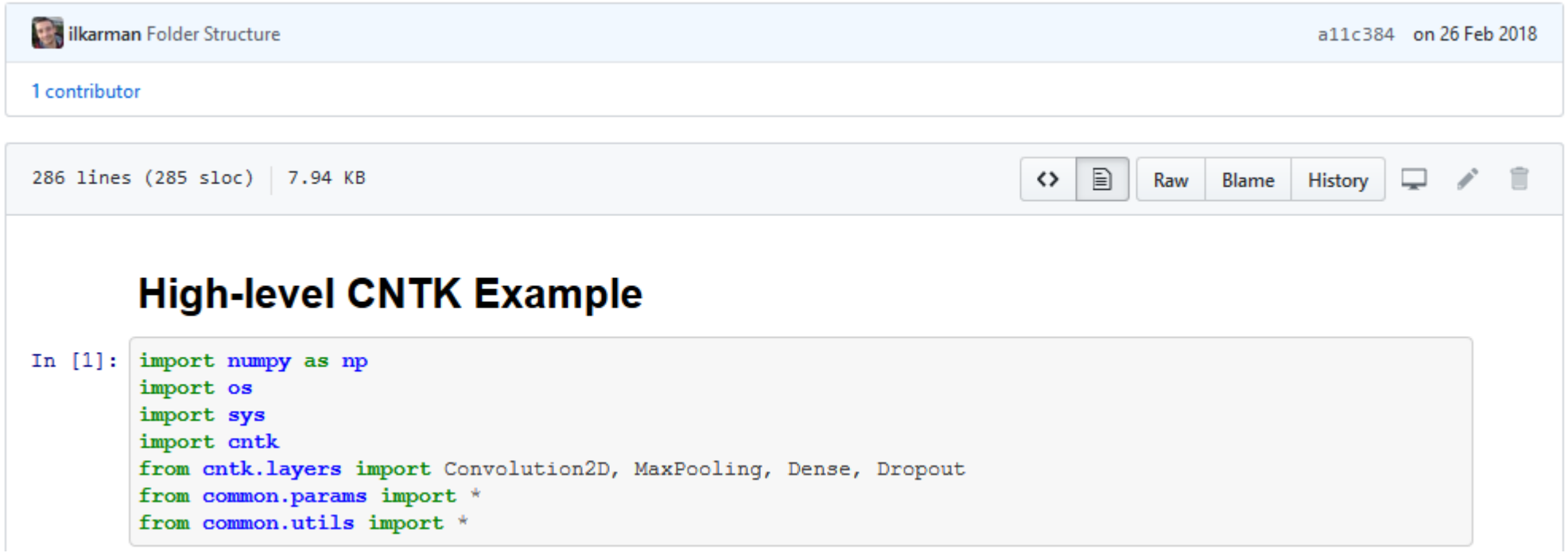
CNTK - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks



Keras

Developers	Francois Chollet
Initial release	Mar 2015
Stable Release	2.2.4
Programming Languages	Python
Platform	Cross-Plattform
License	MIT
Repository	https://github.com/keras-team/keras
Website	https://keras.io/

Keras

Pros

- High-level Deep Learning API
- Flat learning curve
  - simplistic and intuitive interface – fantastic for newbies
  - API aligned to scikit-learn (model.fit, .predict, .evaluate)
  - Comprehensible code
- Facilitates rapid experimentation
  - lightweight in terms of building DL models with a lot of layers
- Can use different backends (Tensorflow, CNTK, Theano)
- built-in support for training on multiple GPUs
- can be turned into Tensorflow estimators and trained on clusters of GPUs on Google Cloud
- Huge Community

Cons

- Slightly slower than vanilla backend modelling
- Price of abstraction
  - Less flexible and less extensible
- Not possible to pass additional data to metric or loss function
  - Requires to „hack“ the API (e.g. passing additional inputs through to the loss function)

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Keras – pretrained models

Classify ImageNet classes with ResNet50

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
# Predicted: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265', u'tusker', 0.1122357,
```

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet valida
dataset.

https://keras.io/applications/

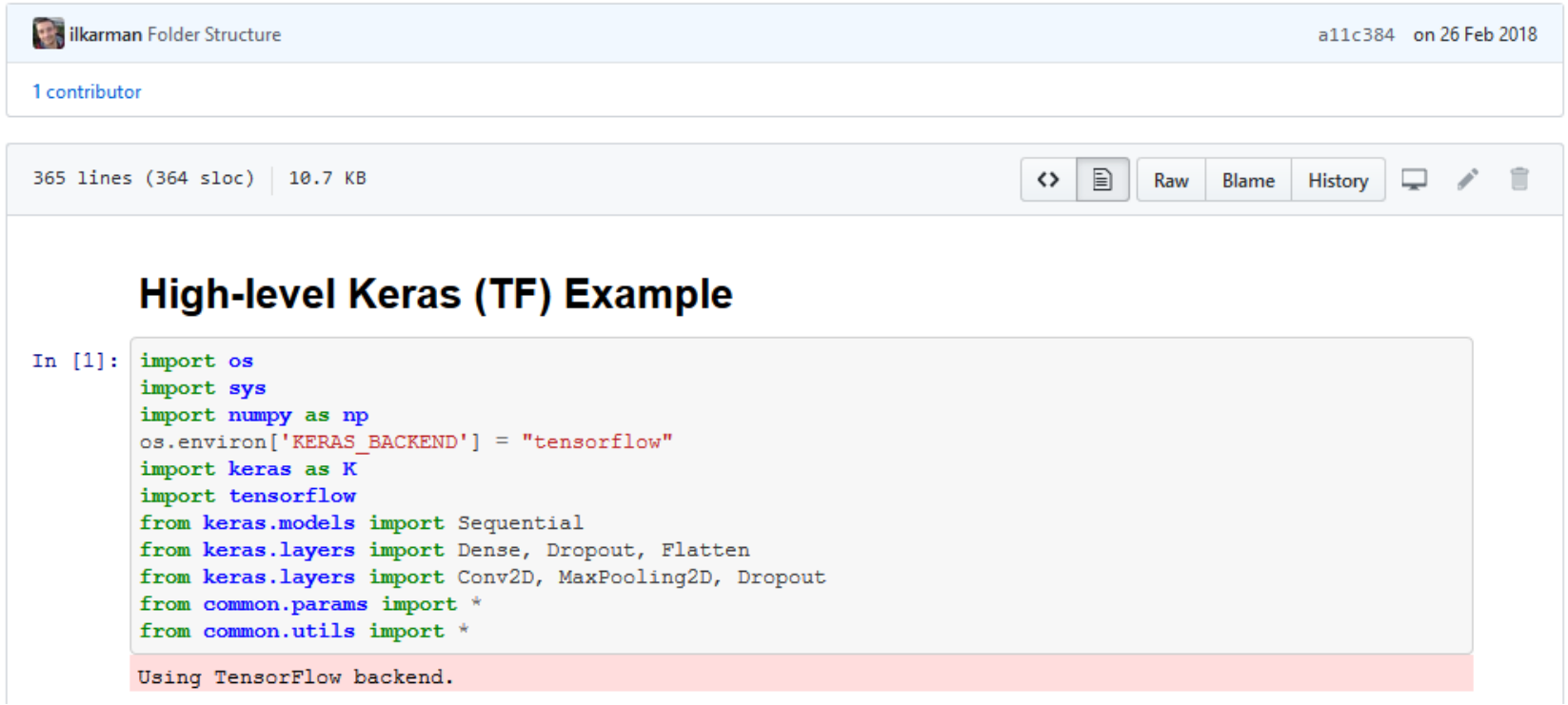
Keras - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks



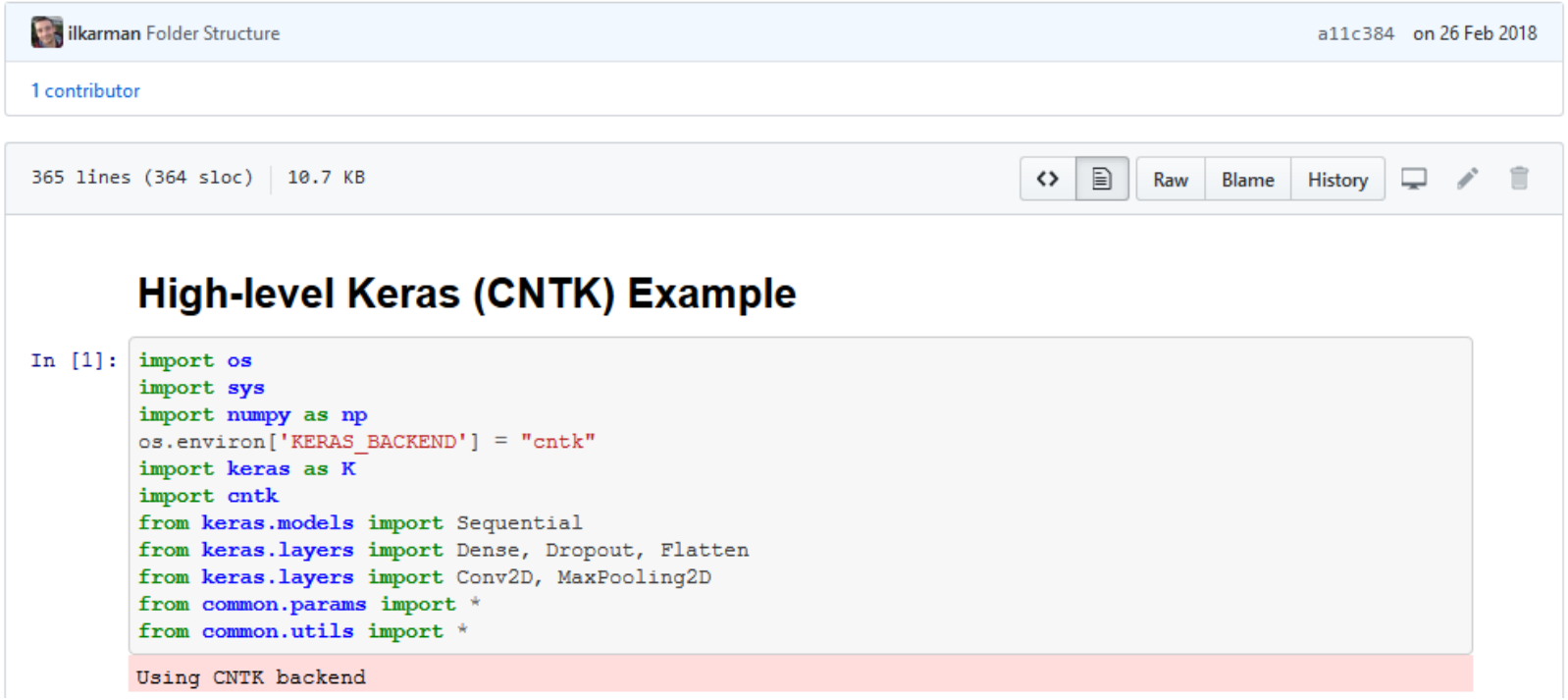
Keras - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks



ilkarman Folder Structure a11c384 on 26 Feb 2018

1 contributor

365 lines (364 sloc) | 10.7 KB

Raw Blame History

High-level Keras (CNTK) Example

In [1]:
```
import os
import sys
import numpy as np
os.environ['KERAS_BACKEND'] = "cntk"
import keras as K
import cntk
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from common.params import *
from common.utils import *
```

```
Using CNTK backend
```

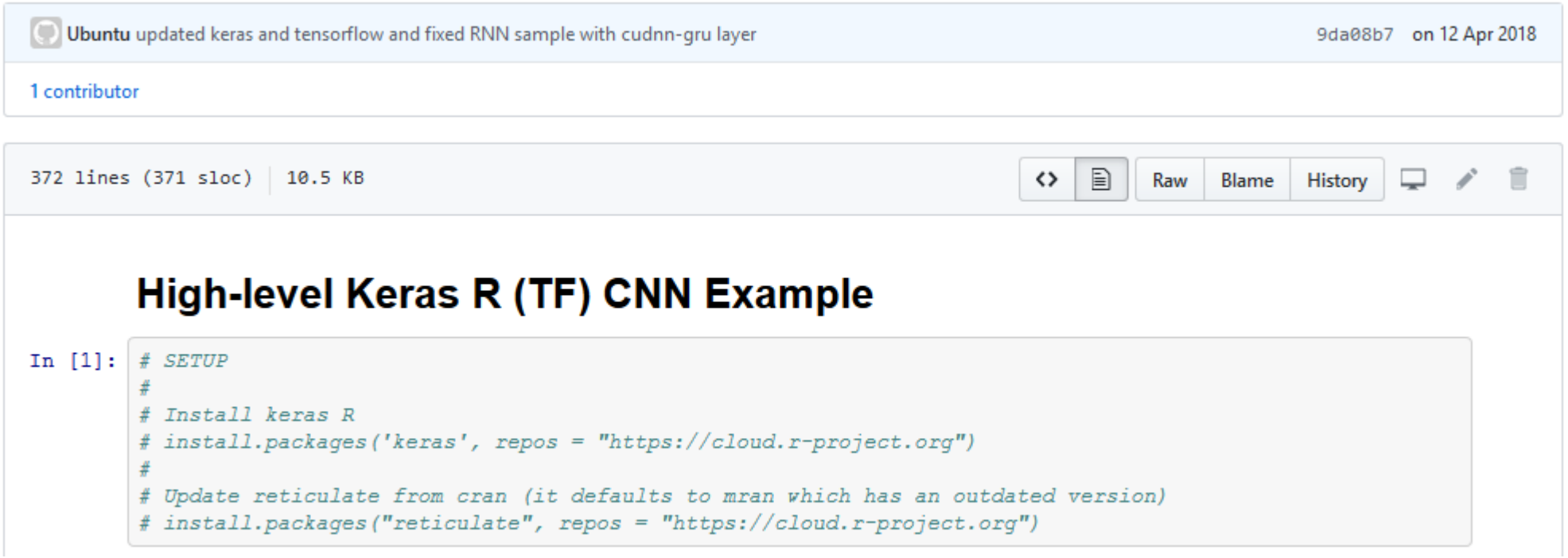
Keras - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks



PyTorch

Developers	Primarily Facebook
Initial release	Oct 2016
Stable Release	1.0.0
Programming Languages	Python
Platform	Linux, macOS,Windows
License	
Repository	https://github.com/pytorch/pytorch
Website	https://pytorch.org/

PyTorch

Pros

- Define computation graph at runtime
- Facilitates interactive debugging
  - And the usage of standard Python debuggers (pdb, ipdb, PyCharm, etc.)
- No session interface or placeholders
- Integrates better into Python and Numpy
- Less boilerplate code is needed to get a basic model running
- Elaborated data loading concept
- features a lot of pretrained models and modular parts that are ready and easy to combine

Cons

- Younger than other frameworks
  - Smaller community
  - Less documentation
- Drawbacks at large-scale deployment
- lacks interfaces for monitoring and visualization such as Tensorboard
  - Wrapper for Tensorboard available

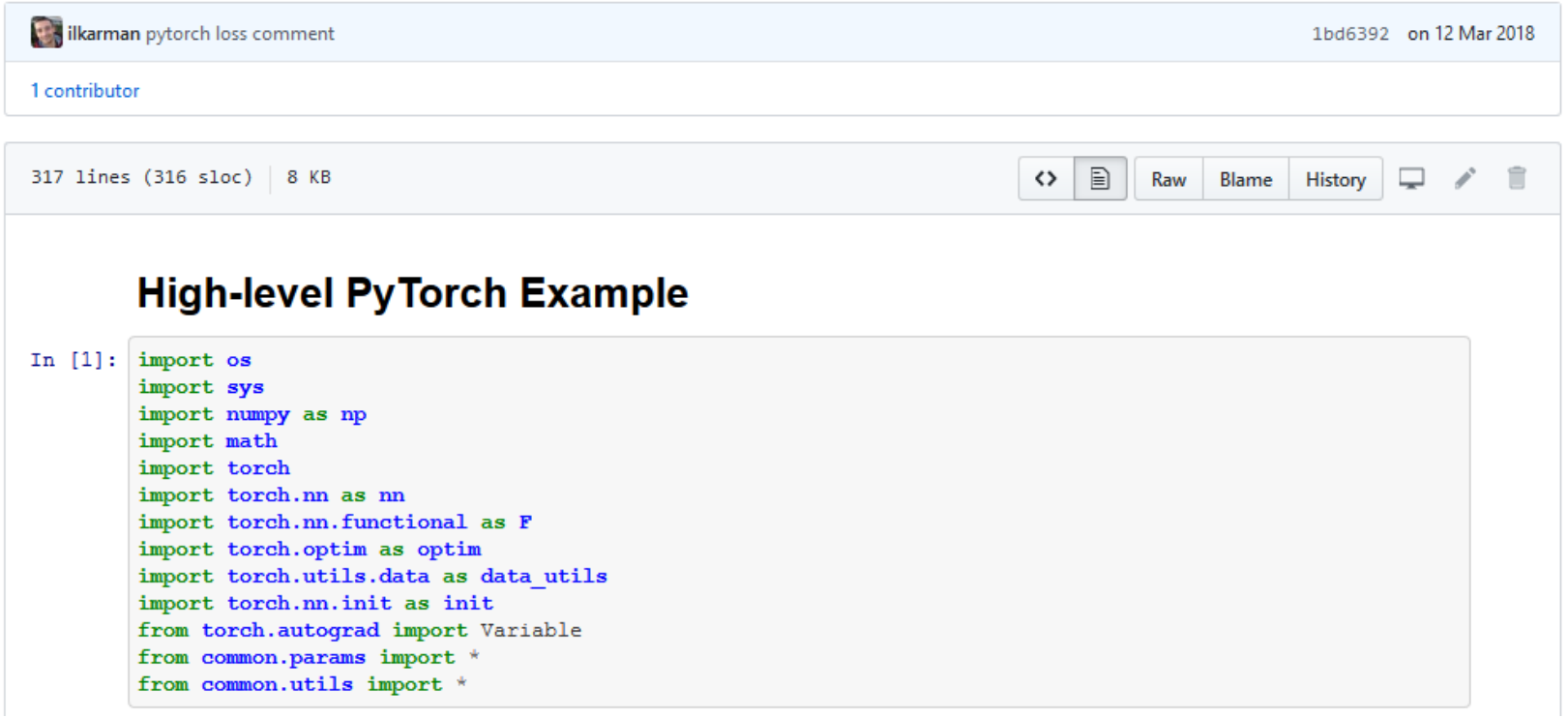
PyTorch - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks



Apache MXNet

Developers	Apache Software Foundation, Amazon
Initial release	Jan 2017
Stable Release	1.4.0
Programming Languages	C++, Python, Java, Scala, R, Julia, Perl, go, JNI(Android), Javascript
Platform	Windows, macOS, Linux
License	Apache 2.0
Repository	https://github.com/apache/incubator-mxnet
Website	https://mxnet.apache.org/

Apache MXNet

Pros

- Fast
- Efficient memory footprint
  - Caveat = Configuration
- Community Support
  - Conversation via Github issues. Direct answers by core developers
- Huge number of supported programming languages
- HybridBlocks
  - Support dynamic and static computational graphs
- can be run on any device
- highly scalable

Cons

- has a much smaller community behind it compared with Tensorflow
- not so popular among the research community

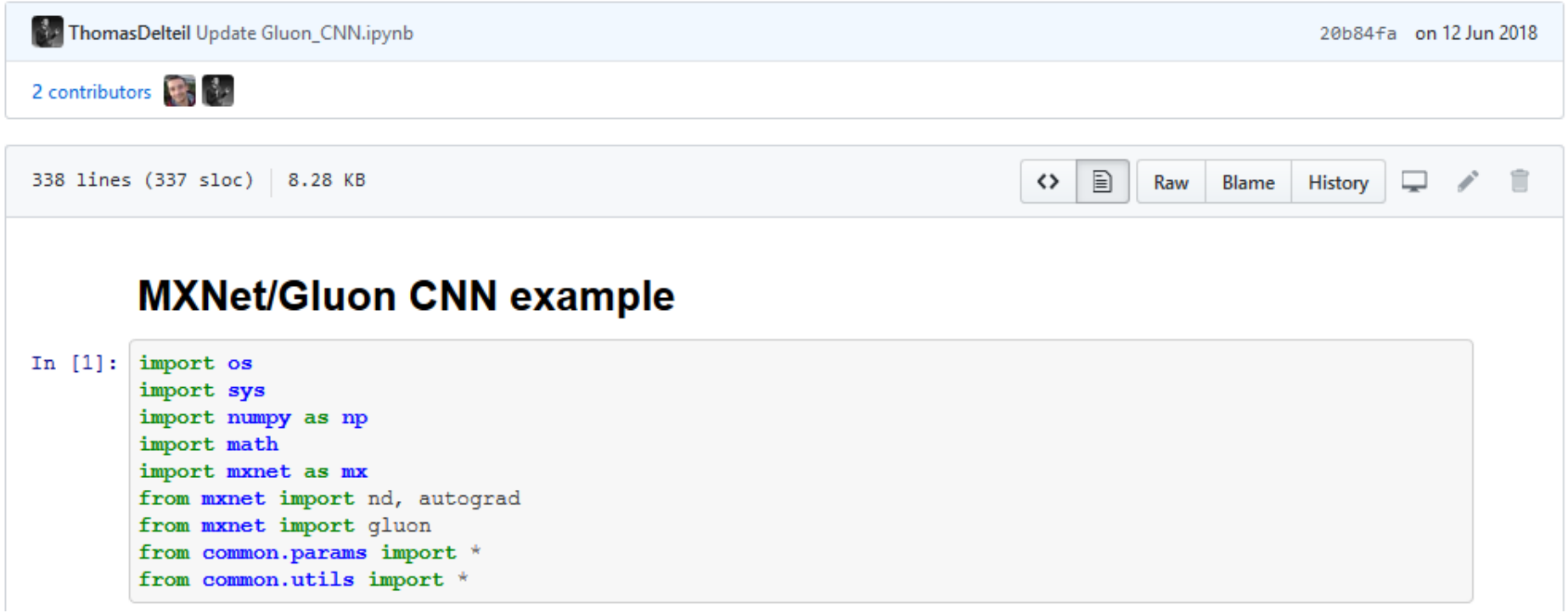
MXNet - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks



Caffe / Caffe2

Developers	Berkeley Vision and Learning Center
Initial release	
Stable Release	
Programming Languages	
Platform	Linux, macOS, Windows
License	BSD
Repository	
Website	

Caffe / Caffe2

Pros

- Good for feedforward networks and image processing
- fast, scalable, and lightweight
- Good for finetuning existing networks
- Train models without writing any code
- Python interface is pretty useful
- offers pre-trained models for building demo apps
- Can deploy without Python
- works well with other frameworks, like PyTorch, and it’s going to be merged into PyTorch 1.0

Cons

- Need to write C++ / CUDA for new GPU layers
- Not good for recurrent networks, or generative adversarial network
- Cumbersome for big networks (GoogLeNet, ResNet)
- Not really extensible
- Probably dying; slow development
- Not used as much in research anymore,
  - still popular for deploying models
- Limited community support

Caffe2 - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks

337 lines (336 sloc) | 9.68 KB

Raw Blame History

High-level Caffe2 Example

In [1]:
```
import os
import sys
import caffe2
import numpy as np
from caffe2.python import core, model_helper, workspace, visualize, brew, optimizer, utils
from caffe2.proto import caffe2_pb2
from common.params import *
from common.utils import *
```

Chainer

Developers	Community, Preferred Networks, Inc.
Initial release	June 2015
Stable Release	V5.4.0 / v6.0.0rc1
Programming Languages	Python
Platform	Cross-platform
License	MIT
Repository	https://github.com/chainer/chainer
Website	https://chainer.org/

Chainer

Pros

- much faster than other leading Python frameworks
- super flexible and intuitive
- Existing networks can be modified at runtime

Cons

- more difficult to debug
- community is relatively small

Chainer - Example

Example

by Ilia Karmanov / ilkarman

Deep Learning Framework Examples

https://github.com/ilkarman/DeepLearningFrameworks

Deeplearning4j

Developers	Various
Initial release	
Stable Release	1.0.0-beta3
Programming Languages	Java
Platform	Cross-platform
License	Apache License 2.0
Repository	https://github.com/deeplearning4j/deeplearning4j
Website	https://deeplearning4j.org/

Deeplearning4j

Pros

- commercial-grade, open-source framework
- written mainly for Java and Scala
- robust, flexible and effective
- can process huge amounts of data without sacrificing speed
- works with Apache Hadoop and Spark, on top of distributed CPUs or GPUs
- documentation is really good
- has a community version and an enterprise version

Cons

- Java is not very popular among machine learning projects
- framework itself cannot rely on growing codebases
- costs of development for your project may be much higher

How to choose the right Deep Learning framework? - Revisited

- Depends on the task to solve
  - Highly experimental?
    - Flexibility, Debugging, visualization
    - PyTorch
  - Only Experimenting?
    - Flexibility, Less overhead, comprehensiveness
    - Keras, PyTorch
  - Putting into production?
    - Good model export/import
    - Tensorflow, (keras), MxNet, deeplearnring4j
  - Large-scale deployment?
    - Good cpu optimization, support for Sparq or similar
    - Tensorflow, MxNet, deeplearning4j, CNTK

How to choose the right Deep Learning framework? - Revisited

- Depends on the complexity of the models
  - Fine-tune pre-trained models?
    - Keras, PyTorch
  - Use adapt well-known architectures?
    - Keras, PyTorch
  - Small straigt forward models?
    - Keras, PyTorch
  - Complex architectures, custom loss functions or metrics, multi-task learning, etc.?
    - PyTorch
  - Optimze for speed
    - MxNet, Chainer, CNTK

DEEP LEARNING FRAMEWORK COMPAIRISON

Overview

Benchmarking State-of-the-Art Deep Learning Software Tools

Shaohuai Shi, Qiang Wang, Pengfei Xu, Xiaowen Chu
Department of Computer Science, Hong Kong Baptist University
{*csshshi, qiangwang, pengfeixu, chxw*}*@comp.hkbu.edu.hk*

Benchmarks of deep learning software tools (2016)

TABLE 6. THE CONDUCTED EXPERIMENTS WITH THE COMBINATION OF NETWORKS, TOOLS AND HARDWARE.

Network	Tool	Hardware						
		CPU		Single GPU			Multiple GPUs	
		i7-3820	E5-2630v3	GTX 980	GTX 1080	K80	2 GPUs	4 GPUs
FCN-S	Caffe	✓	✓	✓	✓	✓	✗	✗
	CNTK	✓	✓	✓	✓	✓	✗	✗
	MXNet	✓	✓	✓	✓	✓	✗	✗
	TensorFlow	✓	✓	✓	✓	✓	✗	✗
	Torch	✓	✓	✓	✓	✓	✗	✗
AlexNet-S	Caffe	✓	✓	✓	✓	✓	✗	✗
	CNTK	✓	✓	✓	✓	✓	✗	✗
	MXNet	✓	✓	✓	✓	✓	✗	✗
	TensorFlow	✓	✓	✓	✓	✓	✗	✗
	Torch	✓	✓	✓	✓	✓	✗	✗
ResNet-50	Caffe	✓	✓	✓	✓	✓	✗	✗
	CNTK	✗	✗	✓	✓	✓	✗	✗
	MXNet	✓	✓	✓	✓	✓	✗	✗
	TensorFlow	✓	✓	✓	✓	✓	✗	✗
	Torch	✓	✓	✓	✓	✓	✗	✗
FCN-R	Caffe	✓	✓	✓	✓	✓	✓	✓
	CNTK	✓	✓	✓	✓	✓	✓	✓
	MXNet	✓	✓	✓	✓	✓	✓	✓
	TensorFlow	✓	✓	✓	✓	✓	✓	✓
	Torch	✓	✓	✓	✓	✓	✓	✓
AlexNet-R	Caffe	✓	✓	✓	✓	✓	✓	✓
	CNTK	✓	✓	✓	✓	✓	✓	✓
	MXNet	✓	✓	✓	✓	✓	✓	✓
	TensorFlow	✓	✓	✓	✓	✓	✓	✓
	Torch	✓	✓	✓	✓	✓	✓	✓
ResNet-56	Caffe	✓	✓	✓	✓	✓	✓	✓
	CNTK	✗	✗	✓	✓	✓	✓	✓
	MXNet	✓	✓	✓	✓	✓	✓	✓
	TensorFlow	✗	✗	✓	✓	✓	✓	✓
	Torch	✓	✓	✓	✓	✓	✓	✓
LSTM	Caffe	✗	✗	✗	✗	✗	✗	✗
	CNTK	✓	✓	✓	✓	✓	✗	✗
	MXNet	✗	✗	✓	✓	✓	✗	✗
	TensorFlow	✓	✓	✓	✓	✓	✗	✗
	Torch	✓	✓	✓	✓	✓	✗	✗

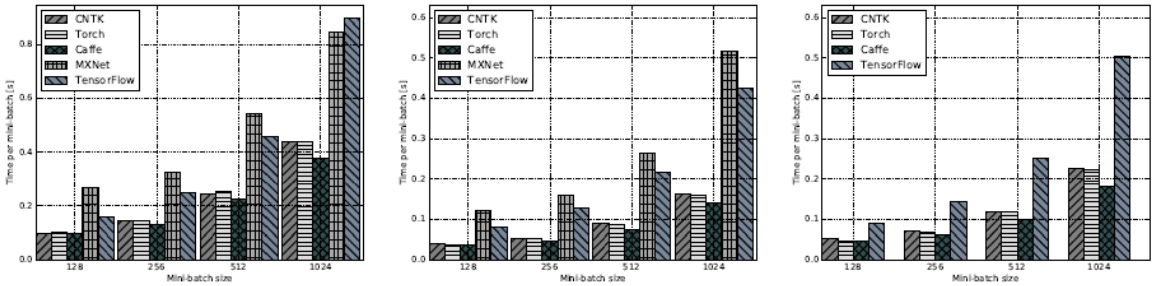
TABLE 7. COMPARATIVE EXPERIMENT RESULTS (TIME PER MINI-BATCH IN SECOND)

		Desktop CPU (Threads used)				Server CPU (Threads used)						Single GPU		
		1	2	4	8	1	2	4	8	16	32	G980	G1080	K80
FCN-S	Caffe	1.324	0.790	**0.578**	15.444	1.355	0.997	0.745	**0.573**	0.608	1.130	0.041	**0.030**	0.071
	CNTK	1.227	0.660	**0.435**	-	1.340	0.909	0.634	0.488	**0.441**	1.000	0.045	**0.033**	0.074
	TF	7.062	4.789	2.648	**1.938**	9.571	6.569	3.399	1.710	0.946	**0.630**	0.060	**0.048**	0.109
	MXNet	4.621	2.607	2.162	**1.831**	5.824	3.356	2.395	2.040	**1.945**	2.670	-	**0.106**	0.216
	Torch	1.329	0.710	**0.423**	-	1.279	1.131	0.595	0.433	**0.382**	1.034	0.040	**0.031**	0.070
AlexNet-S	Caffe	1.606	0.999	**0.719**	-	1.533	1.045	**0.797**	0.850	0.903	1.124	0.034	**0.021**	0.073
	CNTK	3.761	1.974	**1.276**	-	3.852	2.600	1.567	1.347	**1.168**	1.579	0.045	**0.032**	0.091
	TF	6.525	2.936	1.749	**1.535**	5.741	4.216	2.202	1.160	**0.701**	0.962	0.059	**0.042**	0.130
	MXNet	2.977	2.340	2.250	**2.163**	3.518	3.203	2.926	2.828	**2.827**	2.887	0.020	**0.014**	0.042
	Torch	4.645	2.429	**1.424**	-	4.336	2.468	1.543	1.248	**1.090**	1.214	0.033	**0.023**	0.070
RenNet-50	Caffe	11.554	7.671	**5.652**	-	10.643	8.600	6.723	**6.019**	6.654	8.220	-	**0.254**	0.766
	CNTK	-	-	-	-	-	-	-	-	-	-	0.240	**0.168**	0.638
	TF	23.905	16.435	10.206	**7.816**	29.960	21.846	11.512	6.294	**4.130**	4.351	0.327	**0.227**	0.702
	MXNet	48.000	46.154	44.444	**43.243**	57.831	57.143	54.545	54.545	**53.333**	55.172	0.207	**0.136**	0.449
	Torch	13.178	7.500	**4.736**	4.948	12.807	8.391	5.471	4.164	**3.683**	4.422	0.208	**0.144**	0.523
FCN-R	Caffe	2.476	1.499	**1.149**	-	2.282	1.748	1.403	1.211	1.127	**1.127**	0.025	**0.017**	0.055
	CNTK	1.845	0.970	0.661	**0.571**	1.592	0.857	0.501	0.323	**0.252**	0.280	0.025	**0.017**	0.053
	TF	2.647	1.913	1.157	**0.919**	3.410	2.541	1.297	0.661	0.361	**0.325**	0.033	**0.020**	0.063
	MXNet	1.914	1.072	0.719	**0.702**	1.609	1.065	0.731	0.534	0.451	**0.447**	0.029	**0.019**	0.060
	Torch	1.670	0.926	**0.565**	0.611	1.379	0.915	0.662	0.440	0.402	**0.366**	0.025	**0.016**	0.051
AlexNet-R	Caffe	3.558	2.587	**2.157**	2.963	4.270	3.514	3.381	**3.364**	4.139	4.930	0.041	**0.027**	0.137
	CNTK	9.956	7.263	**5.519**	6.015	9.381	6.078	4.984	**4.765**	6.256	6.199	0.045	**0.031**	0.108
	TF	4.535	3.225	1.911	**1.565**	6.124	4.229	2.200	1.396	1.036	**0.971**	**0.227**	0.317	0.385
	MXNet	13.401	12.305	12.278	**11.950**	17.994	17.128	16.764	**16.471**	17.471	17.770	0.060	**0.032**	0.122
	Torch	5.352	3.866	**3.162**	3.259	6.554	5.288	4.365	**3.940**	4.157	4.165	0.069	**0.043**	0.141
RenNet-56	Caffe	6.741	5.451	**4.989**	6.691	7.513	**6.119**	6.232	6.689	7.313	9.302	-	**0.116**	0.378
	CNTK	-	-	-	-	-	-	-	-	-	-	0.206	**0.138**	0.562
	TF	-	-	-	-	-	-	-	-	-	-	0.225	**0.152**	0.523
	MXNet	34.409	31.255	**30.069**	31.388	44.878	43.775	**42.299**	42.965	43.854	44.367	0.105	**0.074**	0.270
	Torch	5.758	3.222	**2.368**	2.475	8.691	4.965	3.040	**2.560**	2.575	2.811	0.150	**0.101**	0.301
LSTM	Caffe	-	-	-	-	-	-	-	-	-	-	-	-	-
	CNTK	0.186	0.120	**0.090**	0.118	0.211	0.139	0.117	0.114	**0.114**	0.198	0.018	**0.017**	0.043
	TF	4.662	3.385	1.935	**1.532**	6.449	4.351	2.238	1.183	0.702	**0.598**	0.133	**0.065**	0.140
	MXNet	-	-	-	-	-	-	-	-	-	-	0.089	**0.079**	0.149
	Torch	6.921	3.831	**2.682**	3.127	7.471	4.641	3.580	**3.260**	5.148	5.851	0.399	**0.324**	0.560

Note: The mini-batch sizes for FCN-S, AlexNet-S, ResNet-50, FCN-R, AlexNet-R, ResNet-56 and LSTM are 64, 16, 16, 1024, 1024, 128 and 128 respectively.

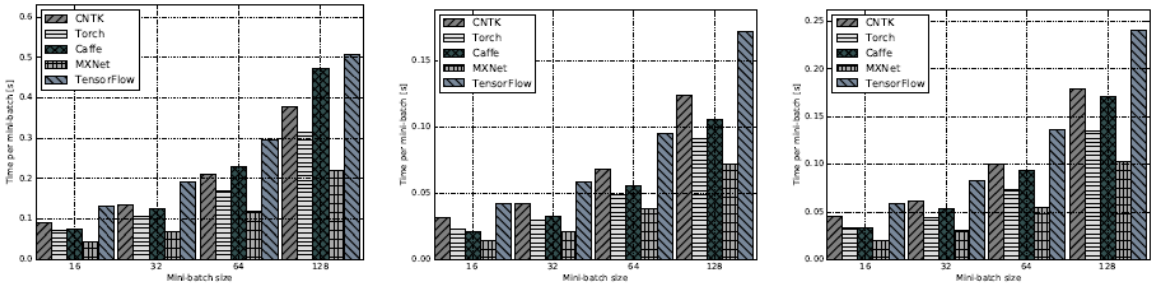
Shi, S., Wang, Q., Xu, P., & Chu, X. (2016, November). Benchmarking state-of-the-art deep learning software tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)* (pp. 99-104). IEEE.

Overview



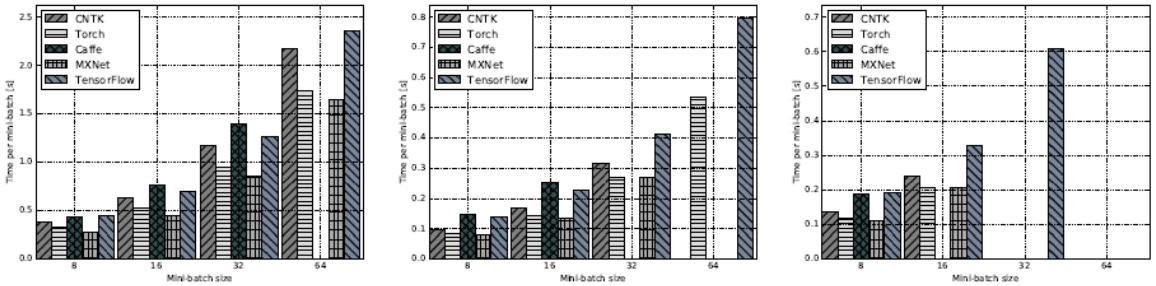
(a) Results on Tesla K80. (b) Results on GTX1080. (c) Results on GTX980.

Figure 9. The performance comparison of FCN-S on GPU platforms.



(a) Results on Tesla K80. (b) Results on GTX1080. (c) Results on GTX980.

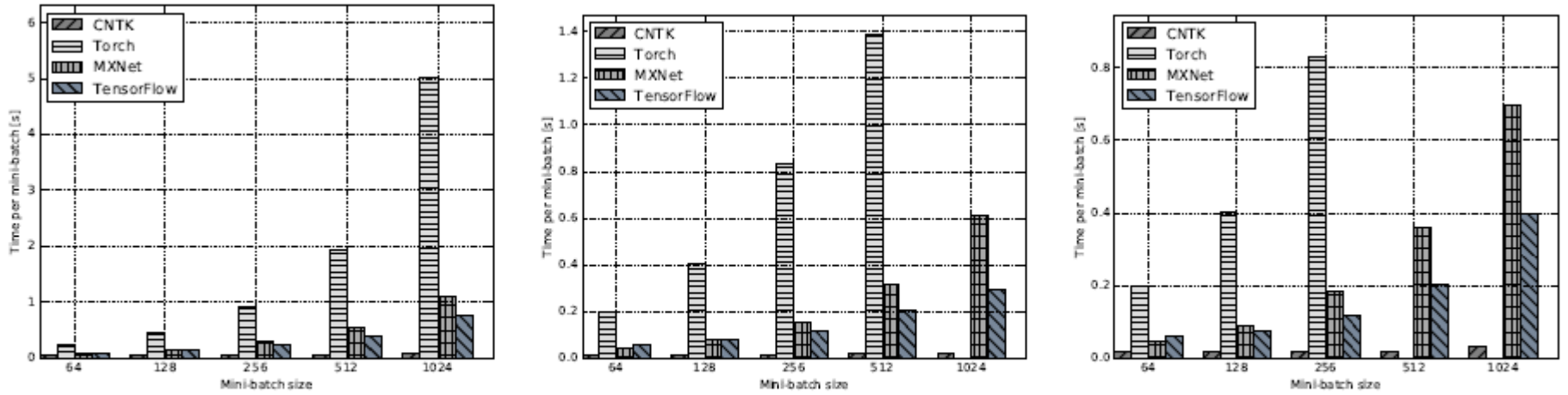
Figure 10. The performance comparison of AlexNet-S on GPU platforms.



(a) Results on Tesla K80. (b) Results on GTX1080. (c) Results on GTX980.

Figure 11. The performance comparison of ResNet-50 on GPU platforms.

Overview

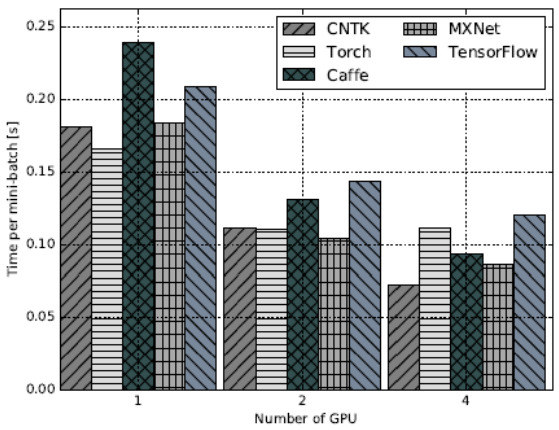


(a) Results on Tesla K80. (b) Results on GTX1080. (c) Results on GTX980.

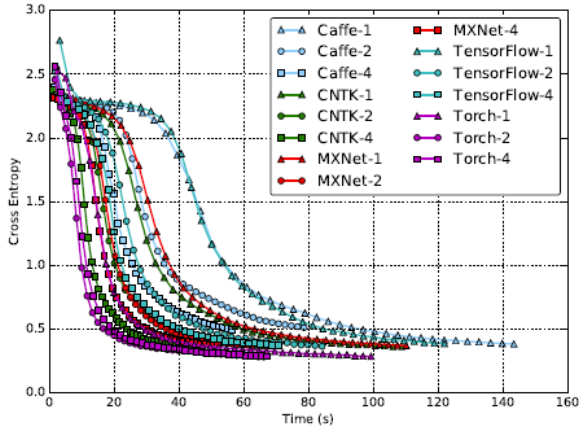
Figure 15. The performance comparison of LSTM on GPU platforms.

Shi, S., Wang, Q., Xu, P., & Chu, X. (2016, November). Benchmarking state-of-the-art deep learning software tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)* (pp. 99-104). IEEE.

Overview

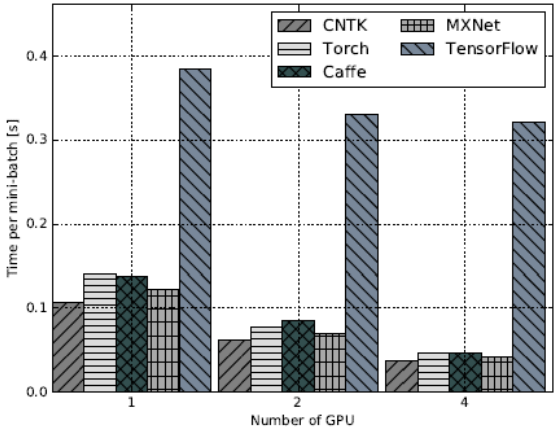


(a) Performance comparison of FCN-R on mulit-GPU platform.

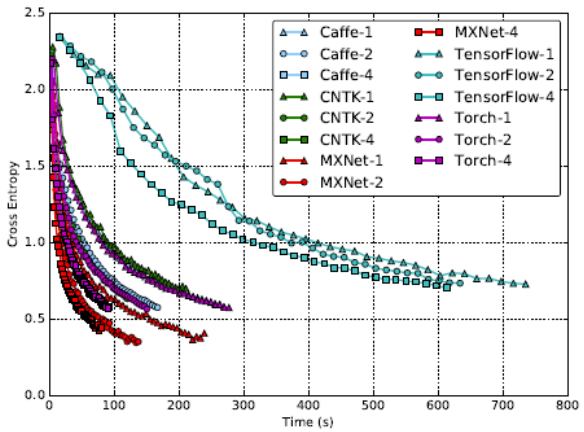


(b) Convergent speed on multi-GPU platform.

Figure 16. Results of FCN-R with a mini-batch size of 4096 on multiple GPUs. (The suffix number of the item in the legend represents the number of GPU used by that tool.)



(a) Performance comparison of AlexNet-R on mulit-GPU platform.



(b) Convergent speed on multi-GPU platform.

Figure 17. Results of AlexNet-R with a mini-batch size of 1024 on multiple GPUs. (The suffix number of the item in the legend represents the number of GPU used by that tool.)

Shi, S., Wang, Q., Xu, P., & Chu, X. (2016, November). Benchmarking state-of-the-art deep learning software tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)* (pp. 99-104). IEEE.

DeepSpark

- Pros
  - Deploy DNNs on Spark
  - Train DNNs on Spark
  - Use computing power of Spark Cluster
- Cons
  - Difficult to set-up
  - Especially difficult to set-up heterogenouus environments (CPU + GPU)

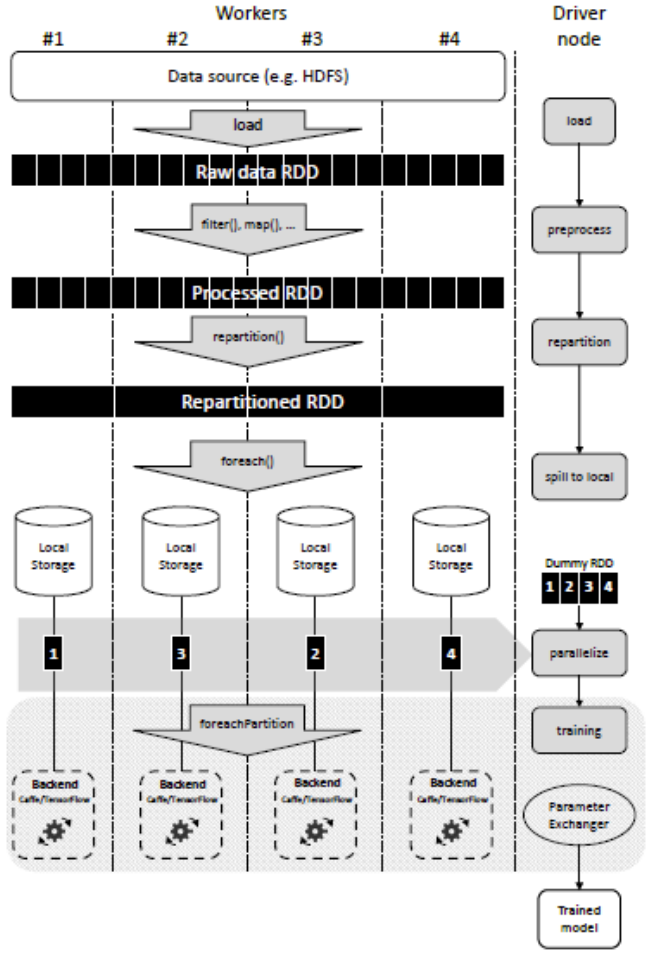


Figure 1: DeepSpark training workflow

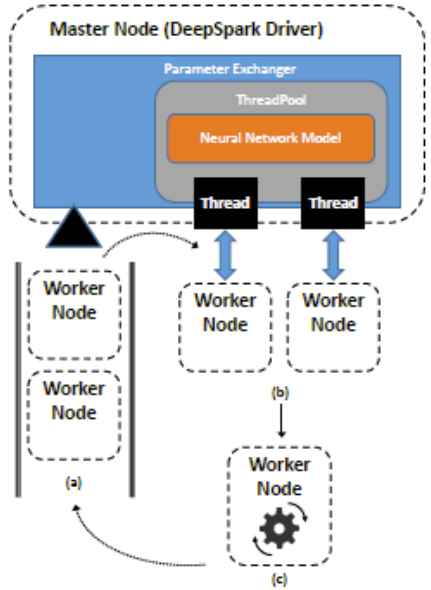


Figure 2: Learning process with parameter exchanger. (a) Worker nodes that want to exchange the parameters are waiting in a queue until an available thread appears. (b) Exchanger threads take care of the worker request. In this example, there are two exchanger threads in the pool. (c) After exchange, a worker node performs its SGD process during the communication period τ.

Kim, H., Park, J., Jang, J., & Yoon, S. (2016). Deepspark: A spark-based distributed deep learning framework for commodity clusters. *arXiv preprint arXiv:1602.08191*.

Deep Learning Frameworks on small scales / embedded devices

Table 1: The overview of the combinations of hardware and software packages.

	MacBook Pro	Intel FogNode	NVIDIA Jetson TX2	Raspberry Pi	Nexus 6P
TensorFlow	√	√	√	√	×
Caffe2	√	√	√	×	×
MXNet	√	√	×	×	×
PyTorch	√	√	√	×	×
TensorFlow Lite	×	×	×	×	√

Table 2: The experimental setup of hardware.

	CPU	Frequency	Cores	Memory	OS
MacBook Pro	Intel Core i5	2.7GHz	4	8GB	macOS 10.13.2
FogNode	Inter Xeon E3-1275 v5	3.6GHz	4	32GB	Linux 4.13.0-32-generic
Jetson TX2	ARMv8 + NVIDIA Pascal GPU	2GHz	6	8GB	Linux 4.4.38-tegra
Raspberry Pi	ARMv71	0.9GHz	4	1GB	Linux rasberrypi 4.4.34
Nexus 6P	Qualcomm Snapdragon 810	2GHz	8	2.6GB	Android 8.0 3.10.73

Zhang, X., Wang, Y., & Shi, W. (2018). pcamp: Performance comparison of machine learning packages on the edges. In *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

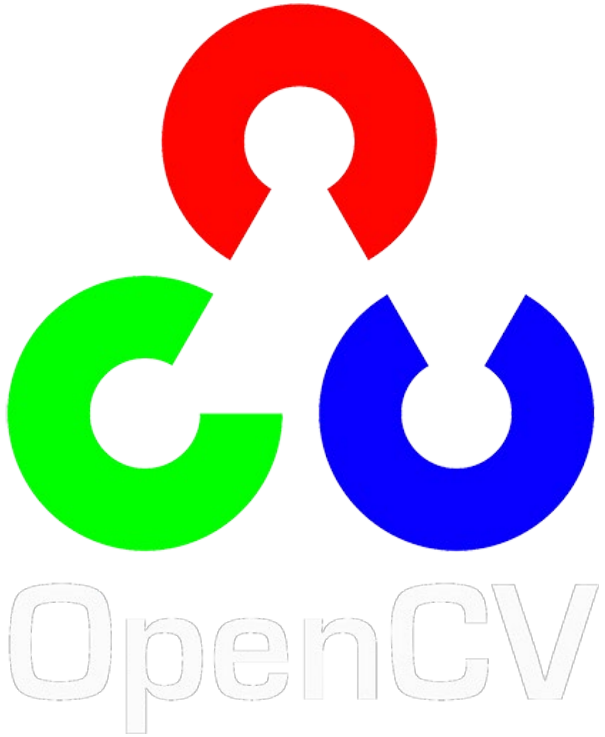
Task Specific Frameworks

FFG
Forschung wirkt.

IMAGE PROCESSING

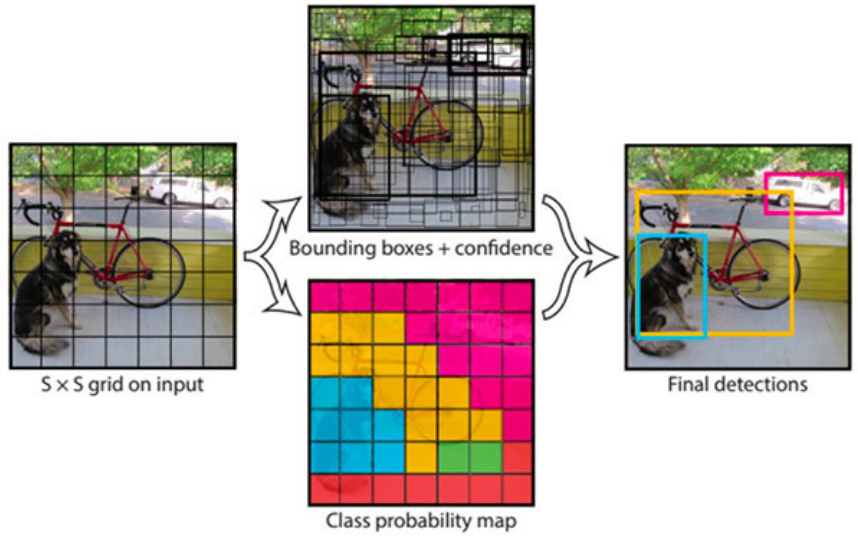
OpenCV

- Swiss Army Knife for Image and Video-Processing
- Huge collection of state-of-the-art algorithms and models
- Huge collection of image and video features
- Large Machine Learning module
  - Video Analysis
  - Object Detection
- Deep Learning module
- Camera Calibration
- Computational Photography
- Image Stitching (e.g. Panorama)
- Android, iOS Support



OpenCV

- YOLO
  - Well known object recognition model
  - YOLOv3 in OpenCV 3.4.2
  - Pros
    - Easy integration in OpenCV code
    - Optimized CPU version (9x faster)
    - Python support (original YOLO in C)



https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

Pil/Pillow

- Not as bloated as OpenCV
- Sufficient for most tasks
  - Especially if images are not extensively pre-processed
- Good for
  - Image I/O
  - Geometric transformations
  - Simple enhancements
- Integrates better into Jupyter
- Cons
  - Images are not represented as numpy arrays
  - Have to be converted to and from

Create JPEG thumbnails

```
from __future__ import print_function
import os, sys
from PIL import Image

size = (128, 128)

for infile in sys.argv[1:]:
    outfile = os.path.splitext(infile)[0] + ".thumbnail"
    if infile != outfile:
        try:
            im = Image.open(infile)
            im.thumbnail(size)
            im.save(outfile, "JPEG")
        except IOError:
            print("cannot create thumbnail for", infile)
```

Enhancing images

```
from PIL import ImageEnhance

enh = ImageEnhance.Contrast(im)
enh.enhance(1.3).show("30% more contrast")
```

scikit-image

- Large collection of image processing algorithms
- Integrates with OpenCV
- Images stored as numpy arrays
- Integrates into scikit and scipy stack

Conversion between color and gray values

Converting an RGB image to a grayscale image is reali

```
>>> from skimage.color import rgb2gray
>>> from skimage import data
>>> img = data.astronaut()
>>> img_gray = rgb2gray(img)
```

Conversion from RGBA to RGB - Removing alpha channel through alpha blending

Converting an RGBA image to an RGB image by alpha blending it with a background is realized with `rgba2rgb()`

```
>>> from skimage.color import rgba2rgb
>>> from skimage import data
>>> img_rgba = data.logo()
>>> img_rgb = rgba2rgb(img_rgba)
```

Numpy

- Most basic image transformations can be solved with numpy
- **Common tasks in image processing**:
  - Input/Output, displaying images
  - Basic manipulations: cropping, flipping, rotating, …
  - Image filtering: denoising, sharpening
  - Registration

```
>>> face = misc.face(gray=True)
>>> face[0, 40]
127
>>> # Slicing
>>> face[10:13, 20:23]
array([[141, 153, 145],
       [133, 134, 125],
       [ 96,  92,  94]], dtype=uint8)
>>> face[100:120] = 255
>>>
>>> lx, ly = face.shape
>>> X, Y = np.ogrid[0:lx, 0:ly]
>>> mask = (X - lx / 2) ** 2 + (Y - ly / 2) ** 2 > lx * ly / 4
>>> # Masks
>>> face[mask] = 0
>>> # Fancy indexing
>>> face[range(400), range(400)] = 255
```

Images are arrays: use the whole `numpy` machinery.

0	1	2
3	4	5
6	7	8

AUDIO PROCESSING

librosa

- Most comprehensive Audio library in Python
- Generally, Audio is always problematic
  - Independant from programming language and OS
  - Librosa helped a lot
- Easy Audio I/O
  - Implicit decoding/encoding in various formats (e.g. .mp3, .ogg, .flacc, etc.)
- Easy Audio transformations
  - samplerate, mono/stereo
- Spectral Transformations / Feature Extraction
  - Mel-Transformations (standard in Deep Learning for Audio)
  - Beat Tracking, Tempo estimation, etc.
- Requires a Backend
  - ffmpeg, sox, libmad, mpg123, mpg321

ffmpeg

- Most valuable tool in multimedia processing (Swiss Army knife)
- Nothing you can‘t solve with it
  - Converting (exotic) video formats
  - Extracting / removing audio from videos
  - Converting (exotic) audio formats
  - Slicing videos
    - Often video-files are too big to be processed with a Python library
  - Concatenating video/audio
- Batch processing

TEXT PROCESSING

Natural Language Toolkit (nltk)

- Most well known and comprehensive Python library
- Excellent basic text processing / pre-processing
  - Tokenizing, stemming
- Good Natural Language Processing
  - Traditional approaches to
    - Named Entitiy Recognition (NER)
    - Part of Speech Tagging (POS)
- Supports various languages

Some simple things you can do with NLTK

Tokenize and tag some text:

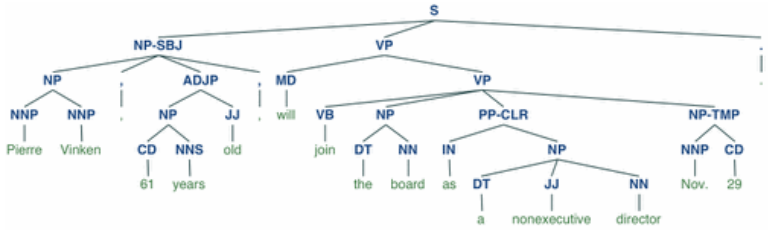
```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

Identify named entities:

```
>>> entities = nltk.chunk.ne_chunk(tagged)
>>> entities
Tree('S', [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'),
           ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'),
      Tree('PERSON', [('Arthur', 'NNP')]),
           ('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'),
           ('very', 'RB'), ('good', 'JJ'), ('.', '.')])
```

Display a parse tree:

```
>>> from nltk.corpus import treebank
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> t.draw()
```



FAKULTÄT FÜR !NFORMATIK

gensim

- Performant Text Processing Library
- Widely adopted
- Good documentation
- One of the first implementations of word2vec
- Comprehensive word embedding handling
- Implements various Text Retrieval Approaches
  - Latent Semantic Indexing (LSI)
  - Latent Dirichlet Allocation (LDA)
- Embeddings can be used in DL Framworks
  - Keras, tensorflow

```
sentences = [['first', 'sentence'], ['second', 'sentence']]
# train word2vec on the two sentences
model = gensim.models.Word2Vec(sentences, min_count=1)
```

```
2017-12-21 16:25:19,117 : INFO : collecting all words and their counts
2017-12-21 16:25:19,119 : INFO : PROGRESS: at sentence #0, processed 0 words, keeping 0 word types
2017-12-21 16:25:19,120 : INFO : collected 3 word types from a corpus of 4 raw words and 2 sentences
2017-12-21 16:25:19,120 : INFO : Loading a fresh vocabulary
2017-12-21 16:25:19,121 : INFO : min_count=1 retains 3 unique words (100% of original 3, drops 0)
2017-12-21 16:25:19,122 : INFO : min_count=1 leaves 4 word corpus (100% of original 4, drops 0)
2017-12-21 16:25:19,123 : INFO : deleting the raw counts dictionary of 3 items
2017-12-21 16:25:19,124 : INFO : sample=0.001 downsamples 3 most-common words
2017-12-21 16:25:19,124 : INFO : downsampling leaves estimated 0 word corpus (5.7% of prior 4)
2017-12-21 16:25:19,125 : INFO : estimated required memory for 3 words and 100 dimensions: 3900 bytes
2017-12-21 16:25:19,126 : INFO : resetting layer weights
2017-12-21 16:25:19,127 : INFO : training model with 3 workers on 3 vocabulary and 100 features, using s
g=0 hs=0 sample=0.001 negative=5 window=5
2017-12-21 16:25:19,130 : INFO : worker thread finished; awaiting finish of 2 more threads
2017-12-21 16:25:19,130 : INFO : worker thread finished; awaiting finish of 1 more threads
2017-12-21 16:25:19,131 : INFO : worker thread finished; awaiting finish of 0 more threads
2017-12-21 16:25:19,131 : INFO : training on 20 raw words (0 effective words) took 0.0s, 0 effective wor
ds/s
2017-12-21 16:25:19,132 : WARNING : under 10 jobs per worker: consider setting a smaller `batch_words' f
or smoother alpha decay
```

flair

- By Zalando Research
- Powerful NLP Library
- On-top of state-of-the-art
- Pre-trained models/embeddings
  - on large corpora
  - Various (european) languages
    - English
    - German (!)
    - Dutch, Polish, French, Italian, …

Comparison with State-of-the-Art

Flair outperforms the previous best methods on a range of NLP tasks:

Task	Language	Dataset	Flair	Previous best
Named Entity Recognition	English	Conll-03	**93.18** (F1)	*92.22 (Peters et al., 2018)*
Named Entity Recognition	English	Ontonotes	**89.3** (F1)	*86.28 (Chiu et al., 2016)*
Emerging Entity Detection	English	WNUT-17	**49.49** (F1)	*45.55 (Aguilar et al., 2018)*
Part-of-Speech tagging	English	WSJ	**97.85**	*97.64 (Choi, 2016)*
Chunking	English	Conll-2000	**96.72** (F1)	*96.36 (Peters et al., 2017)*
Named Entity Recognition	German	Conll-03	**88.27** (F1)	*78.76 (Lample et al., 2016)*
Named Entity Recognition	German	Germeval	**84.65** (F1)	*79.08 (Hänig et al, 2014)*
Named Entity Recognition	Dutch	Conll-03	**90.44** (F1)	*81.74 (Lample et al., 2016)*
Named Entity Recognition	Polish	PolEval-2018	**86.6** (F1) (Borchmann et al., 2018)	*85.1 (PolDeepNer)*

flair

- Simple API
  - Embed words
  - Stack multiple embeddings
    - Word
    - Charachter
    - Forward, backward
  - Document Embeddings
    - Pooling
    - RNNs

```
from flair.data import Sentence
from flair.models import SequenceTagger

# make a sentence
sentence = Sentence('I love Berlin .')

# load the NER tagger
tagger = SequenceTagger.load('ner')

# run NER over sentence
tagger.predict(sentence)
```

```
from flair.embeddings import WordEmbeddings, CharacterEmbeddings

# init standard GloVe embedding
glove_embedding = WordEmbeddings('glove')

# init standard character embeddings
character_embeddings = CharacterEmbeddings()
```

```
from flair.embeddings import StackedEmbeddings

# now create the StackedEmbedding object that combines all embeddings
stacked_embeddings = StackedEmbeddings(
    embeddings=[glove_embedding, character_embeddings])
```

flair

- Python implementation of
  - BERT
  - ELMo
- Cons
  - No API Documentation
  - Instead, a set of well made Jupyter Notebooks with structured examples

ID	Language	Embedding
'bert-base-uncased'	English	12-layer, 768-hidden, 12-heads, 110M parameters
'bert-large-uncased'	English	24-layer, 1024-hidden, 16-heads, 340M parameters
'bert-base-cased'	English	12-layer, 768-hidden, 12-heads , 110M parameters
'bert-large-cased'	English	24-layer, 1024-hidden, 16-heads, 340M parameters
'bert-base-multilingual-cased'	104 languages	12-layer, 768-hidden, 12-heads, 110M parameters
'bert-base-chinese'	Chinese Simplified and Traditional	12-layer, 768-hidden, 12-heads, 110M parameters

ID	Language	Embedding
'small'	English	1024-hidden, 1 layer, 14.6M parameters
'medium'	English	2048-hidden, 1 layer, 28.0M parameters
'original'	English	4096-hidden, 2 layers, 93.6M parameters
'pt'	Portuguese	
'pubmed'	English biomedical data	more information

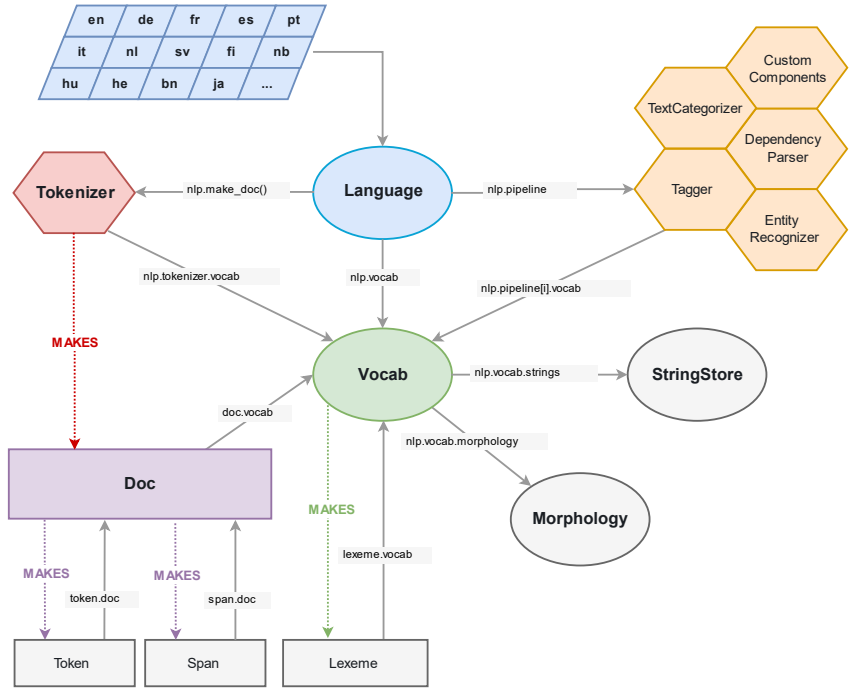
Tutorials

We provide a set of quick tutorials to get you started with the library:

- Tutorial 1: Basics
- Tutorial 2: Tagging your Text
- Tutorial 3: Using Word Embeddings
- Tutorial 4: Using BERT, ELMo, and Flair Embeddings
- Tutorial 5: Using Document Embeddings
- Tutorial 6: Loading your own Corpus
- Tutorial 7: Training your own Models
- Tutorial 8: Optimizing your own Models
- Tutorial 9: Training your own Flair Embeddings

spacy

- Well known NLP Python library
- Supports
  - Tokenization, POS, NER, classification
  - sentiment analysis, dependency parsing
  - word vectors



Hardware

Main Issues

- Choice of GPU
  - Which? how many?
- Embedding Hardware
  - CPU
  - RAM
  - Storage
  - Speed of Bus-system

GPUs

- Which GPUs / Numer of GPUs?
  - Depends on the task
    - usually more GPUs beat stronger GPUs
      - Option to run multi-gpu models
      - Parallelization of experiments / tasks
    - Size of the model
  - Consumer Hardware often sufficient
  - High-End GPUs (e.g. Tesla V100)
    - Single/double precission (16/32 bit)
      - faster computation
    - NV-Link
      - Faster bus to connect multiple GPUs
    - Multiple addressable GPUs on single graphic card

CPUs

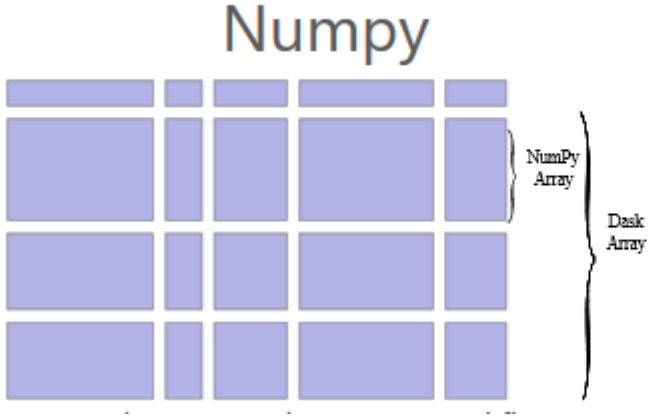
- Don‘t forget about the CPU!
- Not all tasks are processed on the GPU
  - Pre-processing
  - Data managemnet
  - Parallelization (data I/O)
- Weak CPU might lead to weak GPU utilization

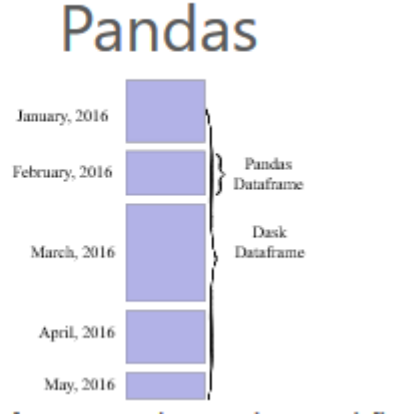
RAM

- Important for fast I/O handling
- Pre-loading all data / large chunks into RAM
- Multiple GPUs / parallel experiments require a lot of RAM

Out-of-Core / Out-of-RAM Handling

- Preparing data within a single Numpy/hdf5 file
- Loading entire file into memory not possible
- Approaches
  - Chunking
  - Memory mapping
  - Dask
    - Numpy, Pandas Wrapper
    - Huge community
    - Good support





HDD / SSD

- Fast data I/O is key
- Data should be located on fast SSDs

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Thank You!

Alexander Schindler
Department of Information Systems and Engineering
Vienna University of Technology
http://www.ifs.tuwien.ac.at/~schindler

FFG
Forschung wirkt.