Deep Learning Framework

A deep learning framework is an interface, library or a tool which allows us to build deep learning models more easily and quickly, without getting into the details of underlying algorithms. They provide a clear and concise way for defining models using a collection of pre-built and optimized components.

Topics Taken

May 9. Paper review due on the same day:

- Eleni Alexakis, Hadis Jami, Tony Okeke, Medical Imaging

- Nick Fioravanti, Justin Serwinski; Jalen Winfield, Deep Learning in Neural Networks with Neural Data Analysis: EEG

- Alexander Wang, A.I. Philosophy and Ethical Alignment

May 16. Paper review due on the same day:

- Josh Miller; Marc Mounzer; Kevin Chavez, Kasonde Chewe, Computational Neuroscience

- Christopher Campbell: Application in Biomedicine

Deep Learning Software

- CPU vs GPU
- Deep Learning Software
  - Caffe / Caffe2
  - Theano / TensorFlow
  - Torch / PyTorch

Deep Learning Code Resources

- TensorFlow: https://www.tensorflow.org/
- CUDA: http://docs.nvidia.com/cuda/cuda-c-programming-guide
- Matlab NN toolbox: https://www.mathworks.com/products/neural-network.html?requestedDomain=www.mathworks.com
- A big list of other Deep Learning Code Resources: http://deeplearning.net/software_links/

CPU (central processing unit)

vs

GPU (graphics processing unit)

CPU vs GPU

CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and "dumber"; great for parallel tasks

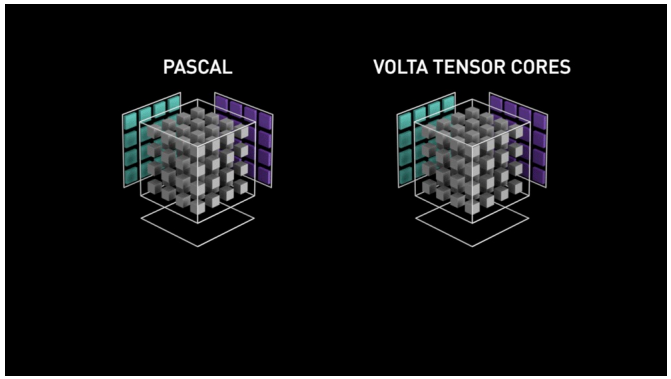
	# Cores	Clock Speed	Memory	Price
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.4 GHz	Shared with system	$339
CPU (Intel Core i7-6950X)	10 (20 threads with hyperthreading)	3.5 GHz	Shared with system	$1723
GPU (NVIDIA Titan Xp)	3840	1.6 GHz	12 GB GDDR5X	$1200
GPU (NVIDIA GTX 1070)	1920	1.68 GHz	8 GB GDDR5	$399

Note: 2018 benchmark

Why GPUs are more suited for Deep Learning?

Nvidia > AMD

	Accelerator	Architecture	FP32 CUDA Cores	FP64 Cores(excl. Tensor)	INT32 Cores	Boost Clock	Memory Clock	Memory Bus Width
2020/5	**A100**	Ampere	6912	3456	6912	1410 MHz	2.4Gbit/s HBM2	5120-bit
2017/5	**V100**	Volta	5120	2560	5120	1530 MHz	1.75Gbit/s HBM2	4096-bit
2016/4	**P100**	Pascal	3584	1792	N/A	1480 MHz	1.4Gbit/s HBM2	4096-bit



A big leap with Tensor cores! Back in the year 2018, Nvidia launched a new lineup of their GPUs i.e 2000 series. Also called RTX, these cards come with tensor cores that are dedicated to deep learning and based on Volta architecture.

https://www.analyticsvidhya.com/blog/2020/09/why-gpus-are-more-suited-for-deep-learning/?utm_source=feedburner&utm_medium=email&utm_campaign=Feed%3A+AnalyticsVidhya+%28Analytics+Vidhya%29

Programming GPUs

- CUDA (NVIDIA only)
  - Write C-like code that runs directly on the GPU
  - Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - Similar to CUDA, but runs on anything
  - Usually slower
- Udacity: Intro to Parallel Programming
  https://www.udacity.com/course/cs344
  - For deep learning just use existing libraries

Google Colaboratory

(Colab) GPU: **Tesla K80 --> T4 (Apr 2019) → P100 (Nov 2019) → V100 (pro, 2020)**
FREE

CPU (central processing unit)

vs

GPU (graphics processing unit)

How to load Google Drive:

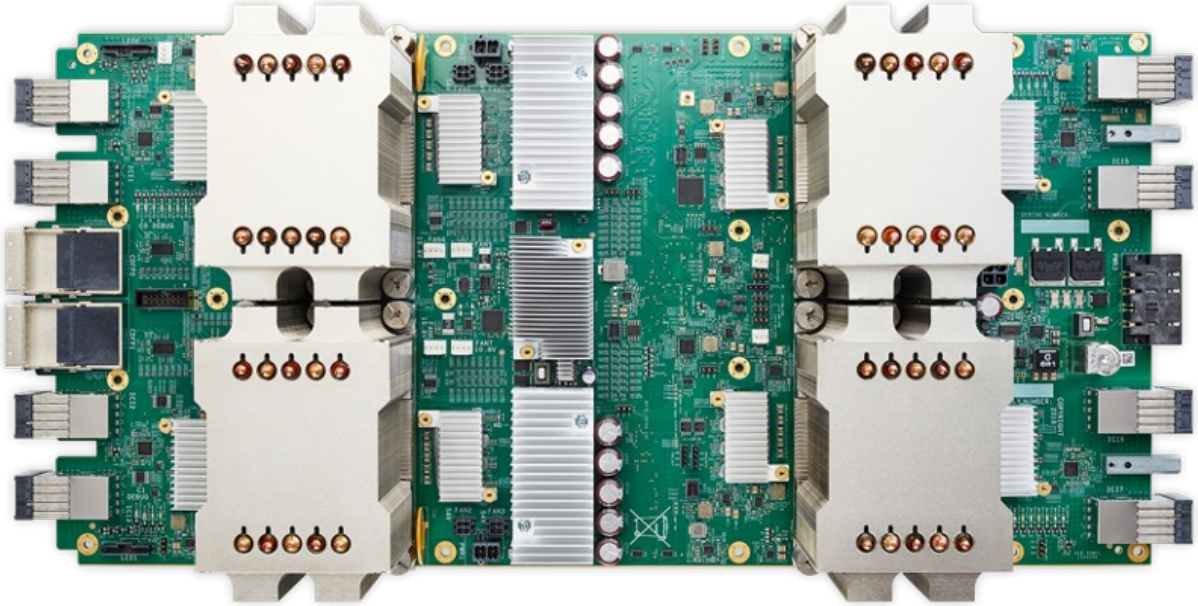
```
from google.colab import drive
drive.mount('/content/drive')
```

Colab Cloud TPU

- Google's Cloud TPU (Tensor Processing Unit) - Challenge to Nvidia's best GPU (Tesla V100):
  - 64GB vs 16 GB of memory, 180 vs 120 TFLOPS of performance

- Cloud TPU can train a ResNet-50 model to 75% accuracy in 24 hours (NVIDIA M40 GPU takes 14 days)
- Cloud TPUs at the rate of US$6.50/Cloud TPU/hour
- 15 to 30 times faster than contemporary GPUs and CPUs in inferencing,

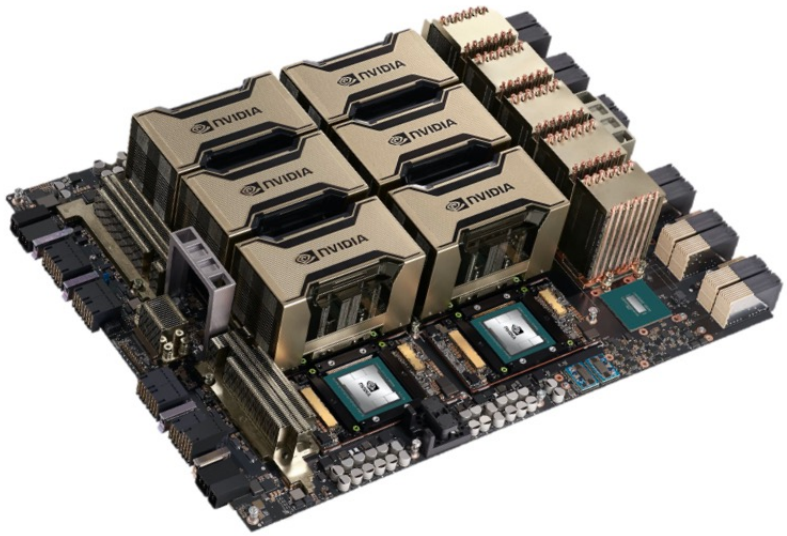
and delivered a 30–80 times improvement in TOPS/Watt measure. (as of 3/20/18)



Cloud TPU

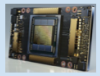
Courtesy of Google

New Compute Engine A2 VMs—first NVIDIA Ampere A100 GPUs in the cloud
Now available withNVIDIA H100 Tensor Core GPUs



https://cloud.google.com/blog/products/compute/announcing-google-cloud-a2-vm-family-based-on-nvidia-a100-gpu

Google Cloud GPU - product portfolio

Tesla GPUs	Training	Inference	Compute	Visualization	VM Family
A100 	●	○	●		A2
V100 	●	○	●		N1
P100 	○		●	○	N1
K80 	○		○		N1
T4 	○	●	○	●	N1
P4 		●		●	N1

NEW & Best Perf

High Growth & Low Cost

DPU - Data Processing Unit

CPU is for general purpose computing,
GPU is for accelerated computing and
DPU, which moves data around the data center, does data processing

The DPU can be used as a stand-alone embedded processor, but it's more often incorporated into a SmartNIC, a network interface controller that's used as a key component in a next generation server.

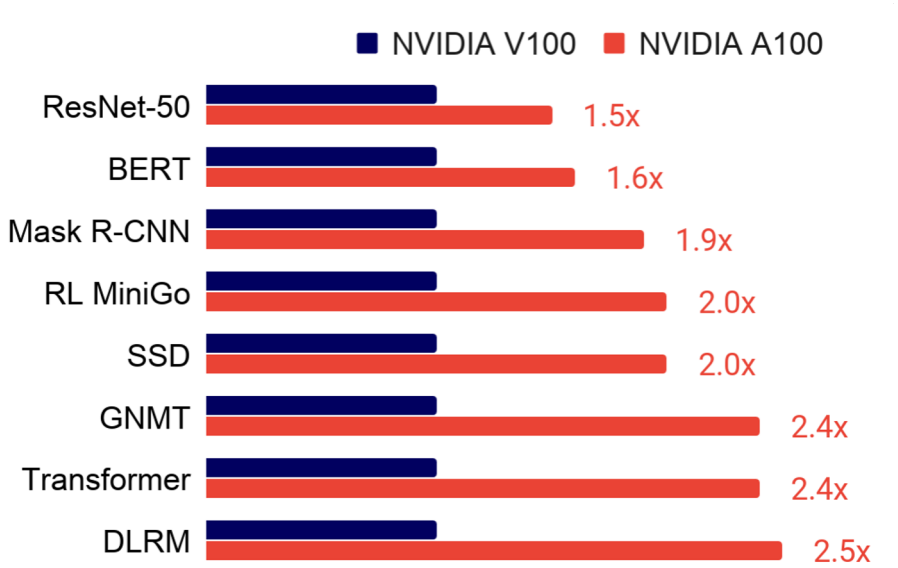
- Industry-standard, high-performance, software-programmable multi-core CPU
- High-performance network interface
- Flexible and programmable acceleration engines

NVIDIA, GTC 2020/10

Software-Defined
Hardware-Accelerated
In-Network Computing

NVIDIA will not rest either: Up to 2.5x training speedups with the new A100 GPU vs V100

- **The A100 GPU is NVIDIA's the first processor based on their new Ampere architecture. The company produced 4x performance gains on**

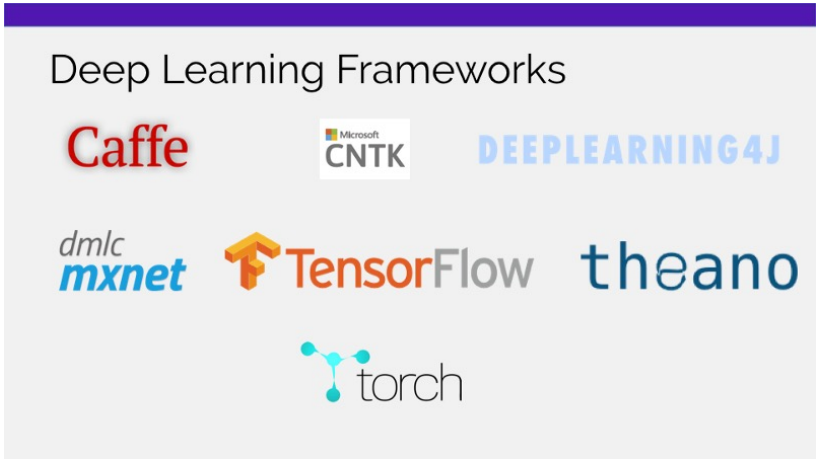


NVIDIA

Figure note: Per chip performance based on comparing performance at same scale and normalizing it to a single chip.

Deep Learning Frameworks

- Deep Learning requires a lot of computations. It typically involves neural network(s) with many nodes, and every node has many connections — which must be updated constantly during the learning. In other words, at each layer of the network, hundreds and thousands of identical artificial neurons perform the same computation. Therefore, the structure of a neural network fits very well with the kinds of computation that a GPU (Graphic Processing Unit) can efficiently perform — which are designed to compute, in parallel, the same instructions.
- As the deep learning and AI fields have been moving extremely fast in the last few years, we've also seen the introduction of many deep learning frameworks. Deep learning frameworks are created with the goal to run deep learning systems efficiently on GPUs.



https://heartbeat.fritz.ai/the-5-deep-learning-frameworks-every-serious-machine-learner-should-be-familiar-with-93f4d469d24c

TensorFlow (Google)

- TensorFlow was originally developed by researchers and engineers working on the Google Brain Team. Its purpose is geared towards deep neural networks and machine intelligence research. The library has officially been open-sourced on GitHub since late 2015. TensorFlow is extremely useful to do graph-based computations quickly.
- TensorFlow has been used widely in academic research and industrial applications. Some notable current uses include Deep Speech, RankBrain, SmartReply, and On-Device Computer Vision.
- www.tensorflow.org
- Keras: The High-Level Wrapper

PyTorch (Facebook)

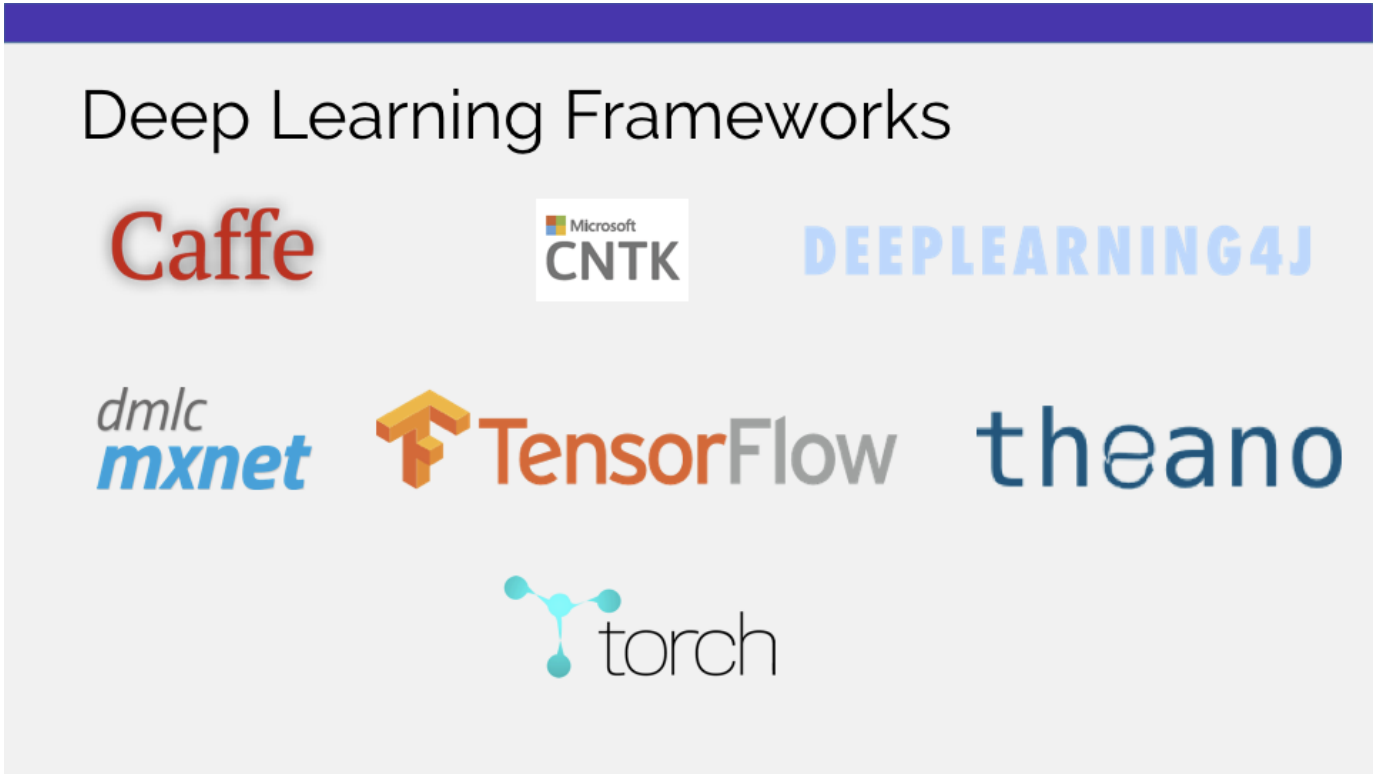
PYTORCH

- PyTorch is a relatively new deep learning framework that is quickly becoming popular among researchers. The Facebook AI Research team developed it to address challenges in the adoption of its predecessor library, Torch. Due to the low popularity of the programming language Lua, Torch can never experience the growth that Google's TensorFlow has. Thus, PyTorch adopted the native Python imperative programming style, which is already familiar to many researchers, developers, and data scientists. It also supports dynamic computation graphs, a feature that makes it attractive to researchers and engineers working with time-series and natural language processing data.
- The best adoption so far has come from Uber, which has built Pyro — a universal probabilistic programming language using PyTorch as its backend. PyTorch's dynamic ability to perform differentiation and construct gradients is extremely valuable for random operations in a probabilistic model.
- http://pytorch.org

Which Deep Learning Frameworks Should You Use?

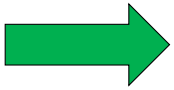
- Ease of use (in terms of architecture and speed), GPU support, availability of tutorials and training materials, neural network modeling capability, and languages supported are all important considerations when choosing which one is best for you.
- TensorFlow and PyTorch emerge as the preferred frameworks of most deep learning practitioners. While both frameworks use Python, there are a couple of differences between them:
  - PyTorch has a cleaner interface and is easier to use, especially for beginners. Writing code (for the most part) feels intuitive, instead of fighting against the library. TensorFlow, on the other hand, is much more cumbersome with so many small, obscure libraries.
  - However, TensorFlow comes with much more support and a very large, vibrant, and helpful community. This means that there are more online courses, code tutorials, docs, and blog posts for TensorFlow than for PyTorch.
  - TensorFlow is more scalable and is very compatible with distributed execution. It supports everything from single GPUs to massive systems which involve heavy distributed reinforcement learning with real-time trials and errors.
  - Most importantly, TensorFlow uses static computational graph, while PyTorch uses dynamic computational graph. The dynamic graph-based approach gives easier debuggability and more processing power for complex architecture, such as dynamic neural networks. The static graph-based approach gives easier deployment to mobile, easier deployment to more exotic architectures, and the ability to do compiler techniques ahead of time. For that reason, PyTorch is better for rapid prototyping for hobbyists and small-scale projects, while TensorFlow is better for large-scale deployments, especially when cross-platform and embedded deployment are considerations.

Deep Learning Frameworks



Deep Learning Frameworks

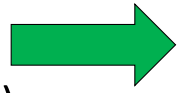
Caffe
(UC Berkeley)



Caffe2
(Facebook)

CNTK
(Microsoft)

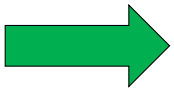
Torch
(NYU / Facebook)



PyTorch
(Facebook)

Paddle
(Baidu)

Theano
(U Montreal)



TensorFlow
(Google)

MXNet
(Amazon)
main framework of
choice at AWS

- Goodbye Theano, announced in Nov. 2017 by Youshua Bengio
- Caffe2 and PyTorch merged on Mar. 30, 2018

Deep Learning Frameworks

Deep Learning Framework	Release Year	Written in which language?	CUDA supported?	Does it have pretrained models?
TensorFlow	2015	C++, Python	Yes	Yes
Keras	2015	Python	Yes	Yes
PyTorch	2016	Python, C	Yes	Yes
Caffe	2013	C++	Yes	Yes
Deeplearning4j	2014	C++, Java	Yes	Yes

Why Deep Learning Frameworks

- Easily build big computational graphs
- Easily compute gradients in computational graphs
- Run it all efficiently on GPU (wrap cuDNN, cuBLAS, etc)

TensorFlow

TensorFlow

Three main components:

- The TensorFlow API, written in C++, contains the API to define the models and train the models with data. It also has a user-friendly Python interface.
- TensorBoard is a visualization toolkit to help with analyzing, visualizing, and debugging TensorFlow graphs
- TensorFlow Serving is a flexible, high-performance serving system used to deploy pre-trained machine learning models in production. Also written in C++ and accessible with a Python interface, Serving is able to switch from old to new models instantaneously

TensorFlow has been used widely in academic research and industrial applications. Some notable current uses include Deep Speech, RankBrain, SmartReply, and On-Device Computer Vision.

Models, tutorials of TensorFlow at this GitHub repo.

TensorFlow: Neural Net

Running example: Train a two-layer ReLU network on random data with L2 loss

TensorFlow: Neural Net

```
import numpy as np
import tensorflow as tf
```

(Assume imports at the top of each snipppet)

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: Neural Net

First **define** computational graph

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])
```

Then **run** the graph many times

```
with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: Neural Net

Create **placeholders** for input x, weights w1 and w2, and targets y

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: Neural Net

Forward pass: compute prediction for y and loss (L2 distance between y and y_pred)

No computation happens here - just building the graph!

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: Neural Net

Tell TensorFlow to compute loss of gradient with respect to w1 and w2.

Again no computation here - just building the graph

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: Neural Net

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Now done building our graph, so we enter a **session** so we can actually run the graph

TensorFlow: Neural Net

Create numpy arrays that will fill in the placeholders above

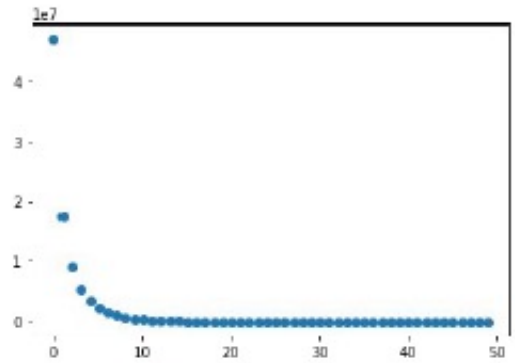

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

TensorFlow: Neural Net

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Run the graph: feed in the numpy arrays for x, y, w1, and w2; get numpy arrays for loss, grad_w1, and grad_w2

TensorFlow: Neural Net



Train the network: Run the graph over and over, use gradient to update weights

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    learning_rate = 1e-5
    for t in range(50):
        out = sess.run([loss, grad_w1, grad_w2],
                       feed_dict=values)
        loss_val, grad_w1_val, grad_w2_val = out
        values[w1] -= learning_rate * grad_w1_val
        values[w2] -= learning_rate * grad_w2_val
```

Keras: The High-Level Wrapper

Deep Learning frameworks operate at 2 levels of abstractions:

Low Level — where mathematical operations and neural network primitives are implemented (TensorFlow, Theano, PyTorch etc.) and

High Level — where low level primitives are used to implement neural network abstractions, such as models and layers (Keras).

Keras is a wrapper over its backend libraries, which can be TensorFlow or Theano — meaning that if you're using Keras with TensorFlow backend, you're running TensorFlow code.

Keras: High-Level Wrapper

Keras is a layer on top of TensorFlow, makes common things easy to do

(Also supports Theano backend)

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High-Level Wrapper

Define model object as a sequence of layers


```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High-Level Wrapper

Define optimizer object

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High-Level Wrapper

Build the model, specify loss function

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Keras: High-Level Wrapper

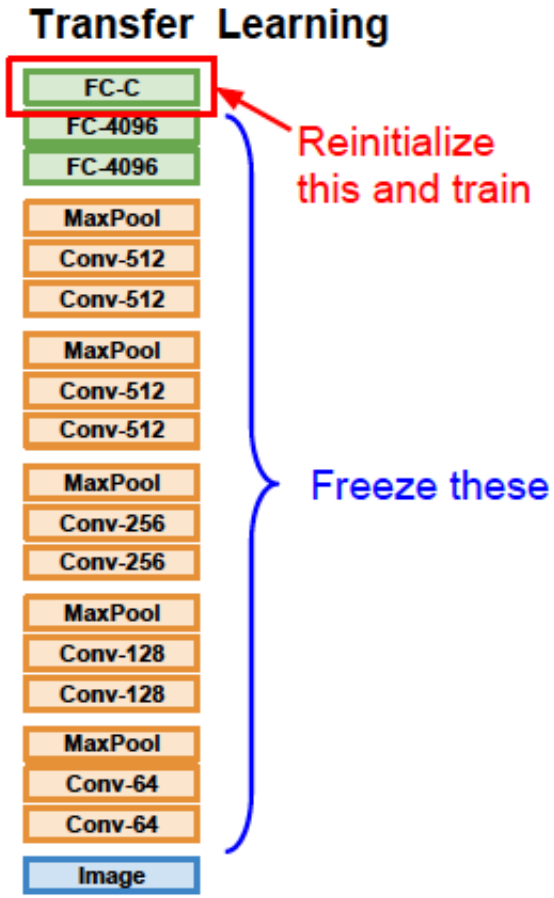
```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

N, D, H = 64, 1000, 100

model = Sequential()
model.add(Dense(input_dim=D, output_dim=H))
model.add(Activation('relu'))
model.add(Dense(input_dim=H, output_dim=D))

optimizer = SGD(lr=1e0)
model.compile(loss='mean_squared_error',
              optimizer=optimizer)

x = np.random.randn(N, D)
y = np.random.randn(N, D)
history = model.fit(x, y, nb_epoch=50,
                    batch_size=N, verbose=0)
```

Train the model with a single line!

TensorFlow: Pretrained Models

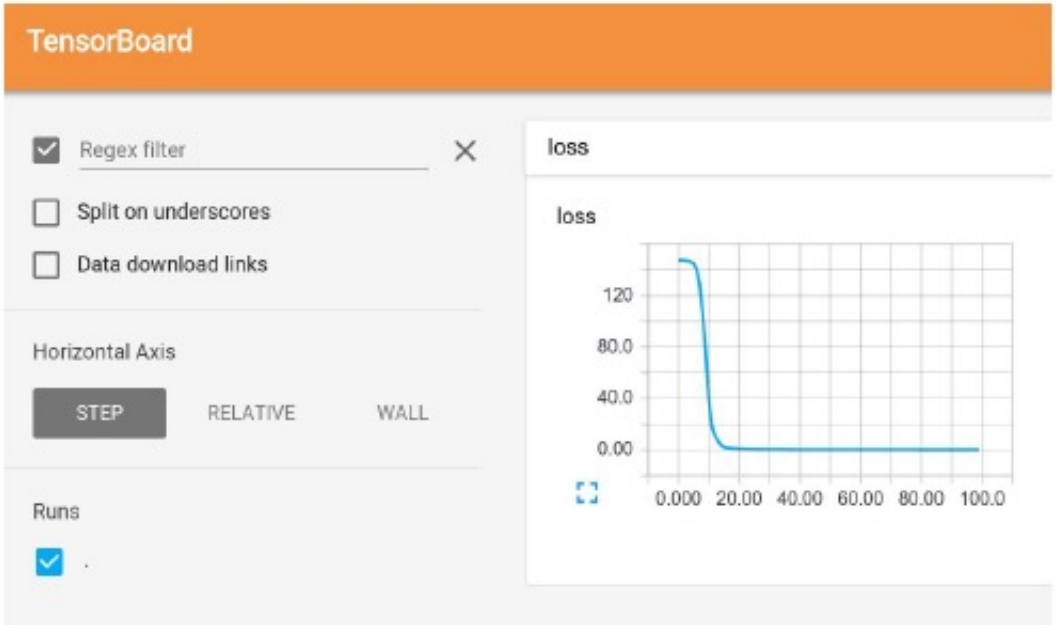
TF-Slim: (https://github.com/tensorflow/models/tree/master/slim/nets)

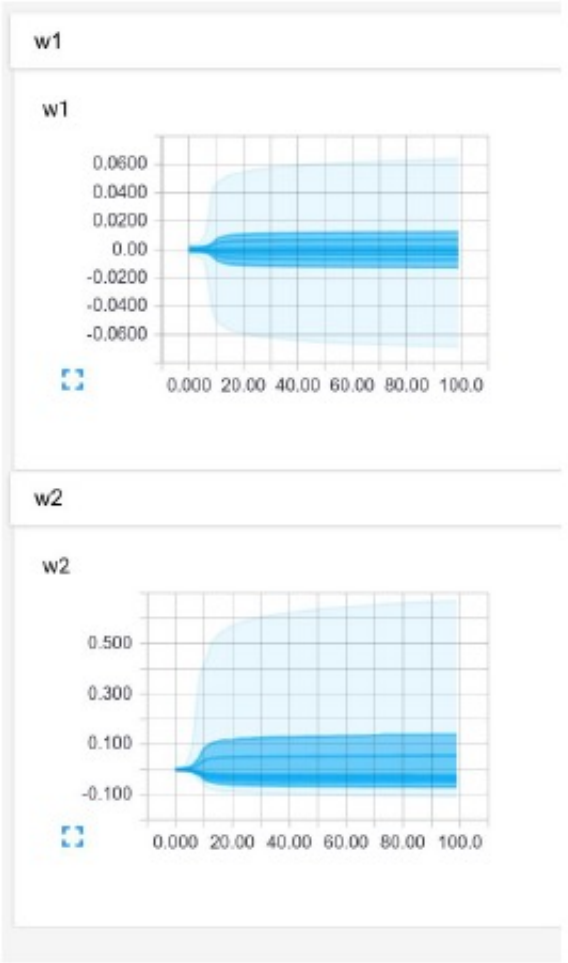
Keras: (https://github.com/fchollet/deep-learning-models)



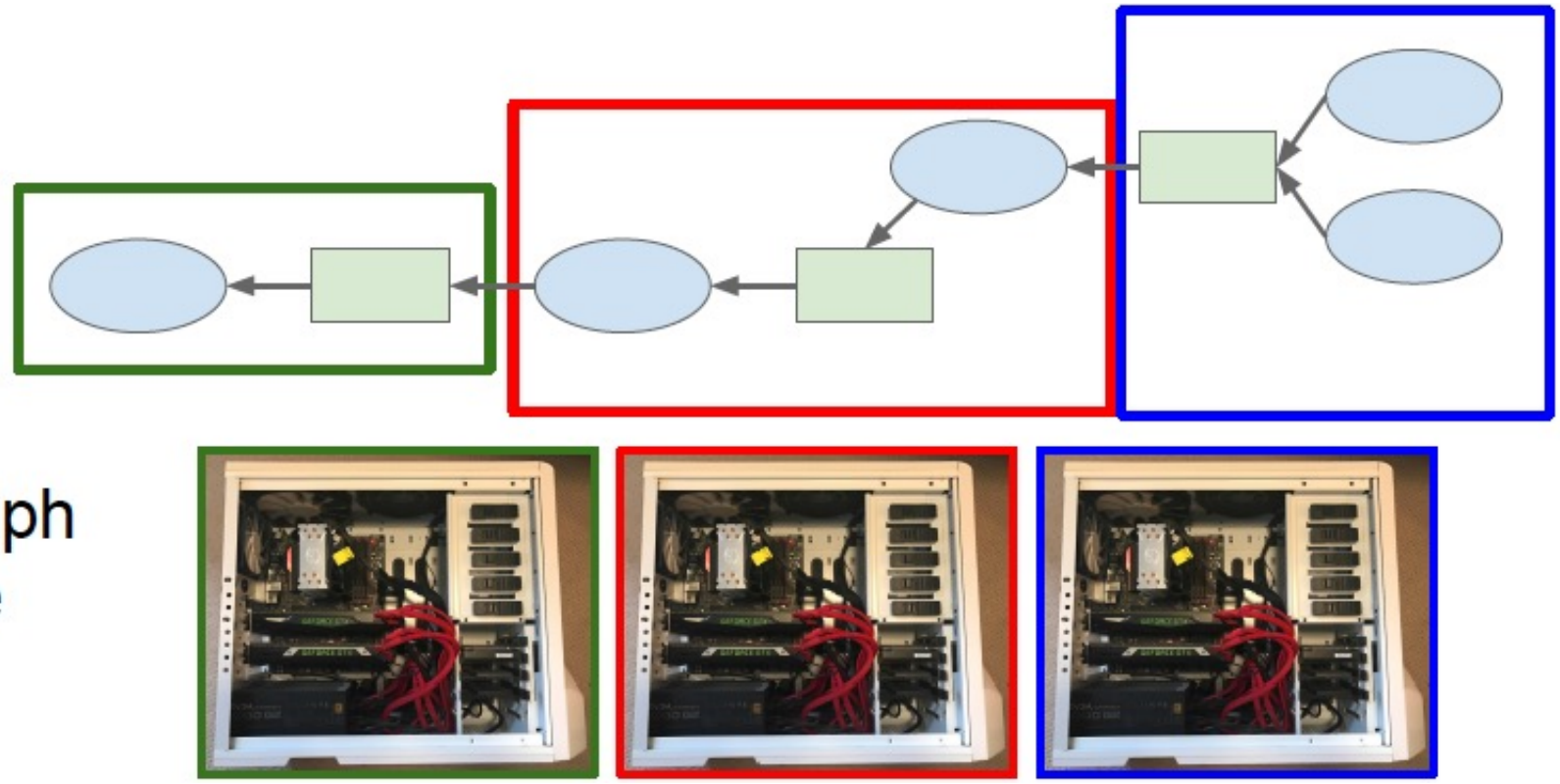
TensorFlow: Tensorboard

Add logging to code to record loss, stats, etc
Run server and get pretty graphs!





TensorFlow: Distributed Version



Split one graph over multiple machines!

https://www.tensorflow.org/deploy/distributed

Google:
TensorFlow

"One framework to rule them all"

Facebook:
PyTorch +Caffe2

Research Production

TensorFlow 2.0 out on 9/30/2019
TensorFlow 2.0 Alpha out on 3/4/2019!
(tf.keras as high-level API and Eager execution by default)

PyTorch 0.4 out on 4/24/2018 (1st taught).
PyTorch 1.0 stable release 12/2018!

Sum up

TensorFlow is a safe bet for most projects. Not perfect but has huge community, wide usage. Maybe pair with high-level wrapper (Keras, Sonnet, etc)

PyTorch is best for research. However still new, there can be rough patches.

Use TensorFlow for one graph over many machines

Consider TensorFlow for production deployment

Consider TensorFlow for mobile

TensorFlow vs. PyTorch

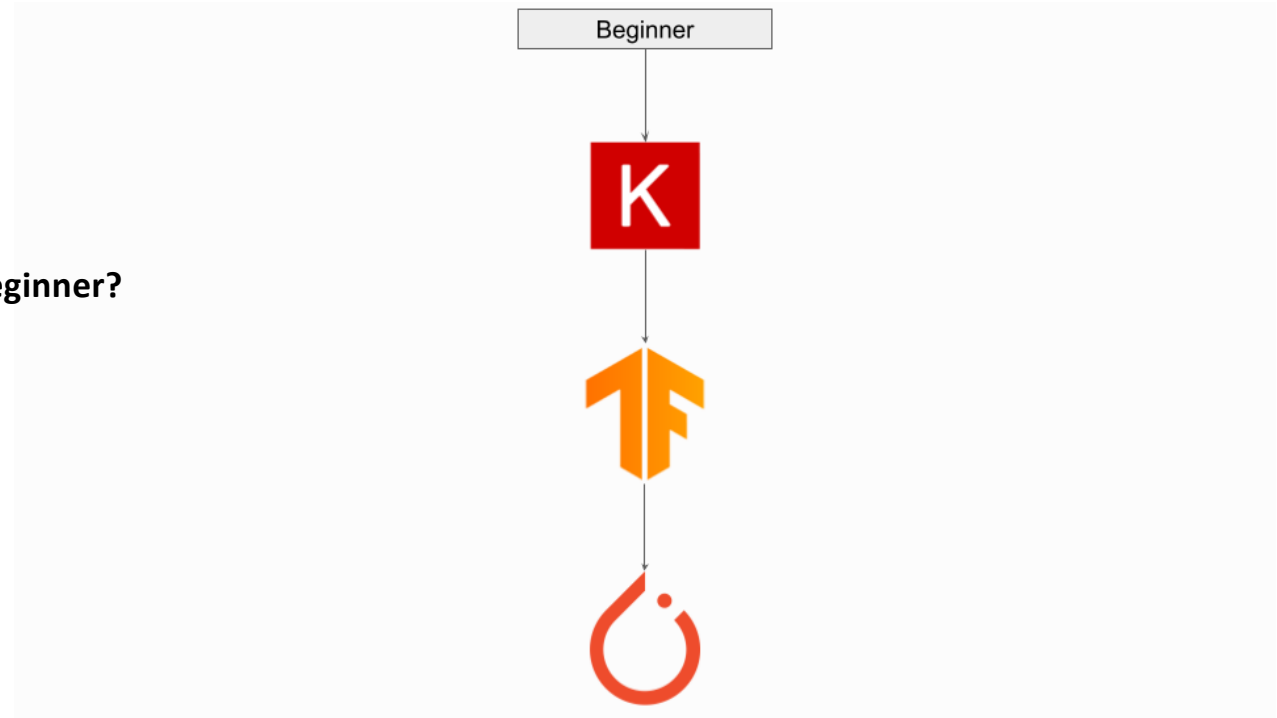
- TensorFlow: **"Define-and-Run"**, in which one would define conditions and iterations in the graph structure, then run it.

- PyTorch: **"Define-by-Run"**, in which graph structure is defined on-the-fly during forward computation. Imperative programming

- **Static computational graph (TensorFlow) vs. Dynamic computational graph (PyTorch):** The dynamic graph-based approach gives easier debuggability and more processing power for complex architecture, such as dynamic neural networks. The static graph-based approach gives easier deployment to mobile, easier deployment to more exotic architectures, and the ability to do compiler techniques ahead of time.

TensorFlow vs. PyTorch

What if I'm a Total Beginner?

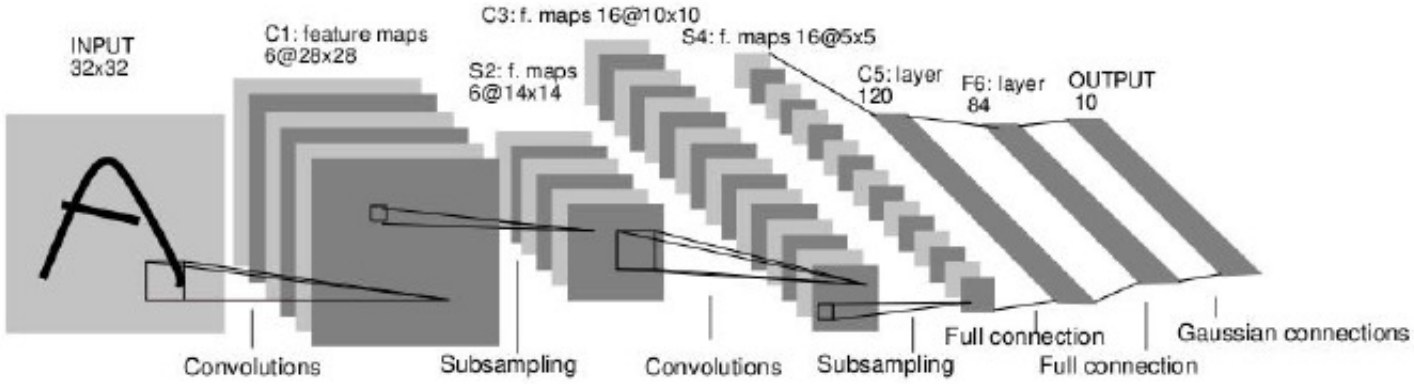


CNN Architectures

- AlexNet
- VGG
- GoogLeNet
- ResNet
- DenseNet
- Many more …

- SqueezeNet & MobileNet for iOS

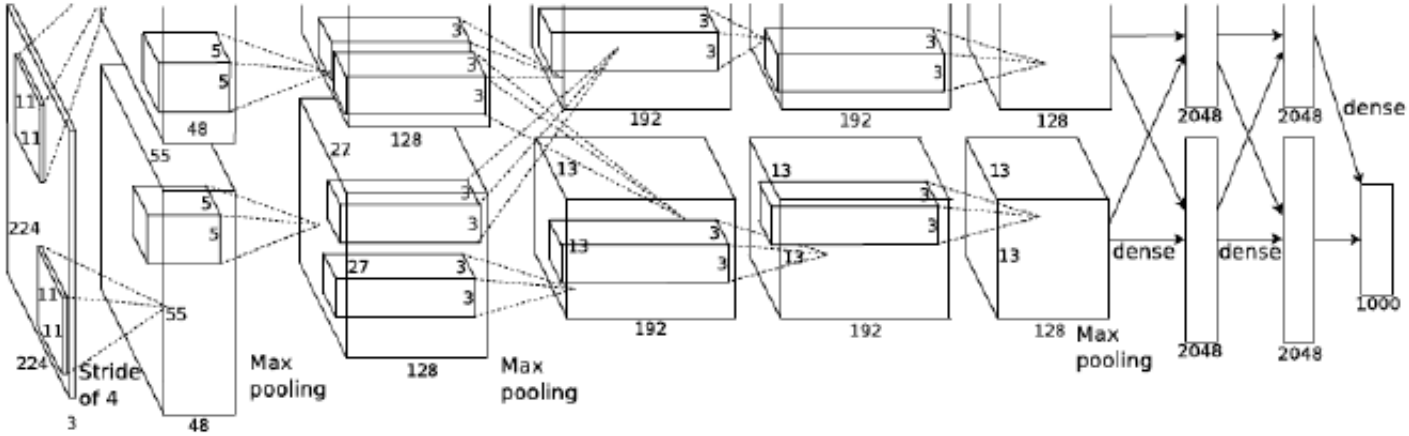
LeNet-5



- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner,
Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

AlexNet: ILSVRC 2012 winner



Similar framework to LeNet but:

- Max pooling, ReLU nonlinearity
- More data and bigger model (7 hidden layers, 650K units, 60M params)
- GPU implementation (50x speedup over CPU)
- Trained on two GPUs for a week
- Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Clarifai: Refinement of AlexNet
ILSVRC 2013 winner

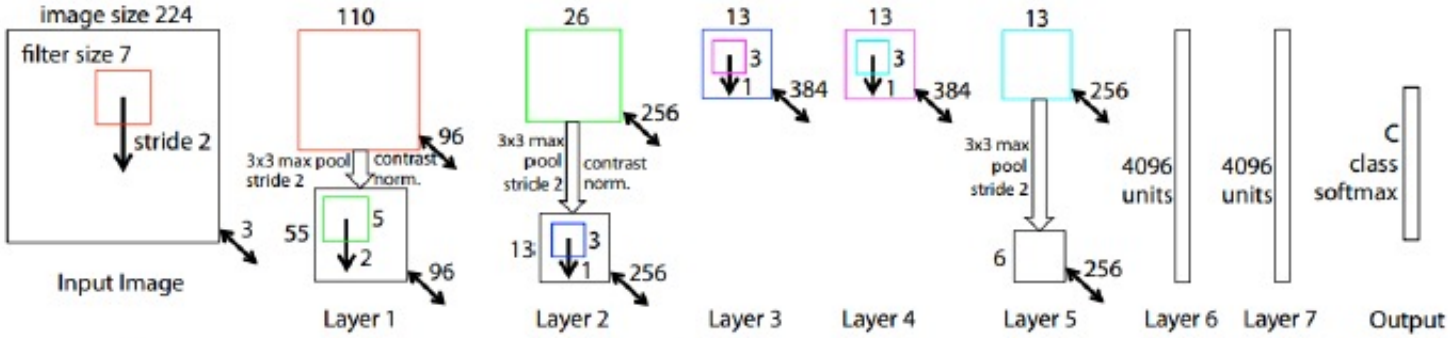


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C-way softmax function, C being the number of classes. All filters and feature maps are square in shape.

M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks,
ECCV 2014 (Best Paper Award winner)

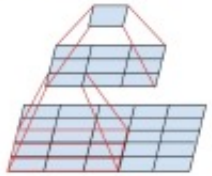
VGGNet: ILSVRC 2014 2nd place

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

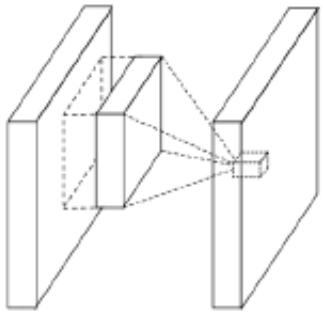
- Sequence of deeper networks trained progressively
- Large receptive fields replaced by successive layers of 3x3 convolutions (with ReLU in between)



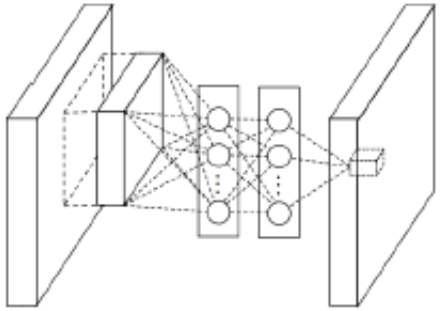
- One 7x7 conv layer with C feature maps needs $49C^2$ weights, three 3x3 conv layers need only $27C^2$ weights
- Experimented with 1x1 convolutions

K. Simonyan and A. Zisserman,
Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

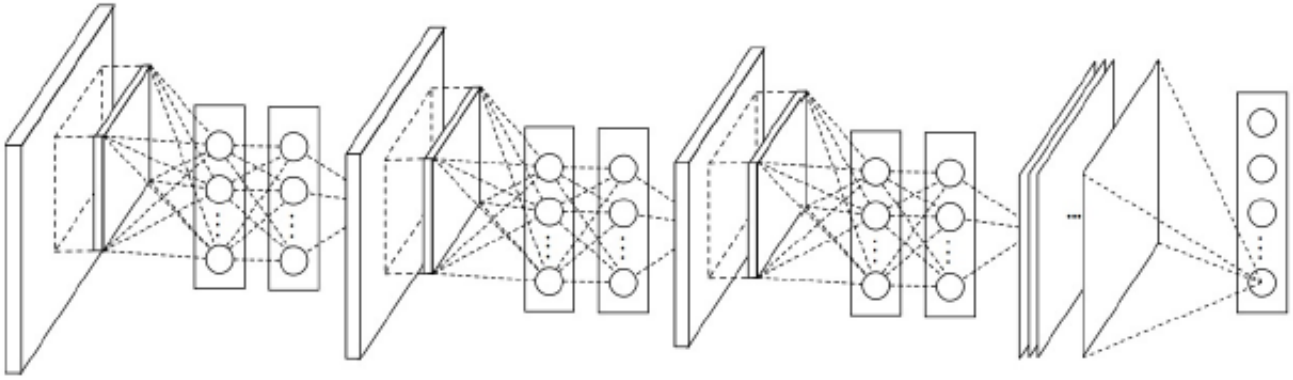
Network in network



(a) Linear convolution layer



(b) Mlpconv layer



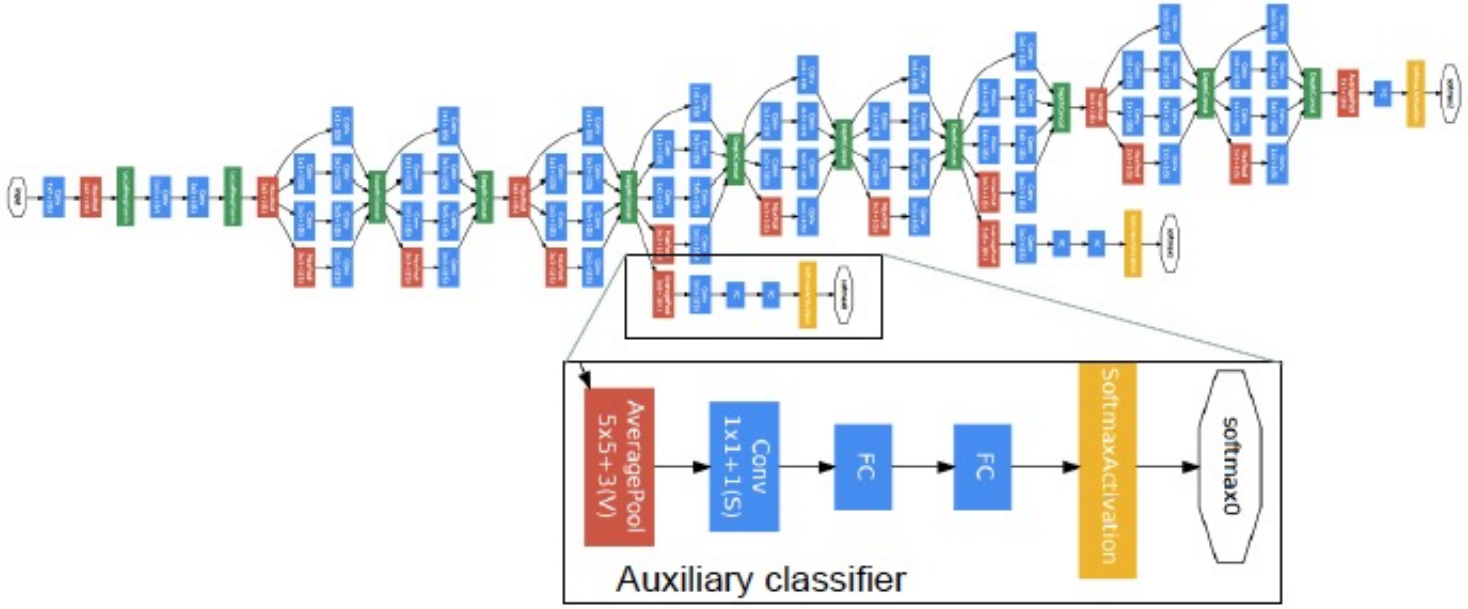
M. Lin, Q. Chen, and S. Yan, Network in network, ICLR 2014

GoogLeNet: ILSVRC 2014 winner

http://knowyourmeme.com/memes/we-need-to-go-deeper

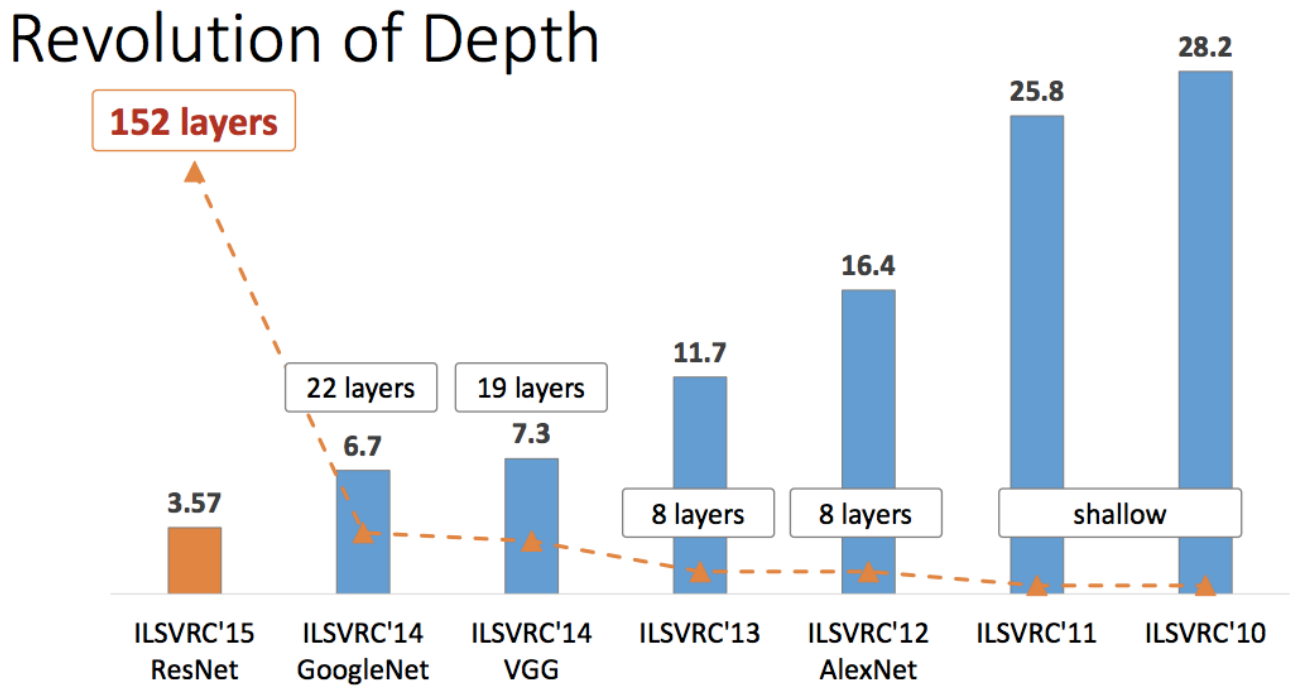
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

GoogLeNet (22-layer)



C. Szegedy et al., Going deeper with convolutions, CVPR 2015

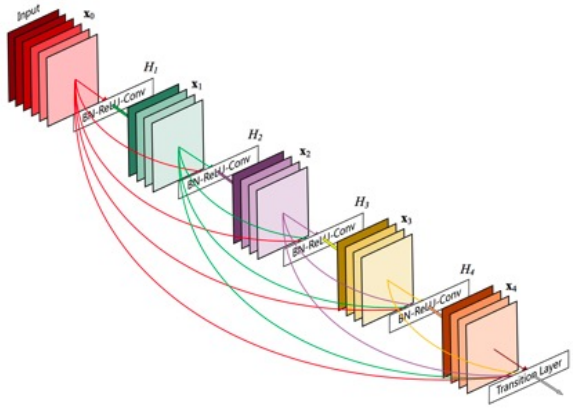
ResNet: ILSVRC 2015 winner



ResNet: ILSVRC 2015 winner – 152 layers

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

DenseNet: CVPR 2017, Best Paper – 250 layers



Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, Densely Connected Convolutional Networks, CVPR 2017, Best Paper Award

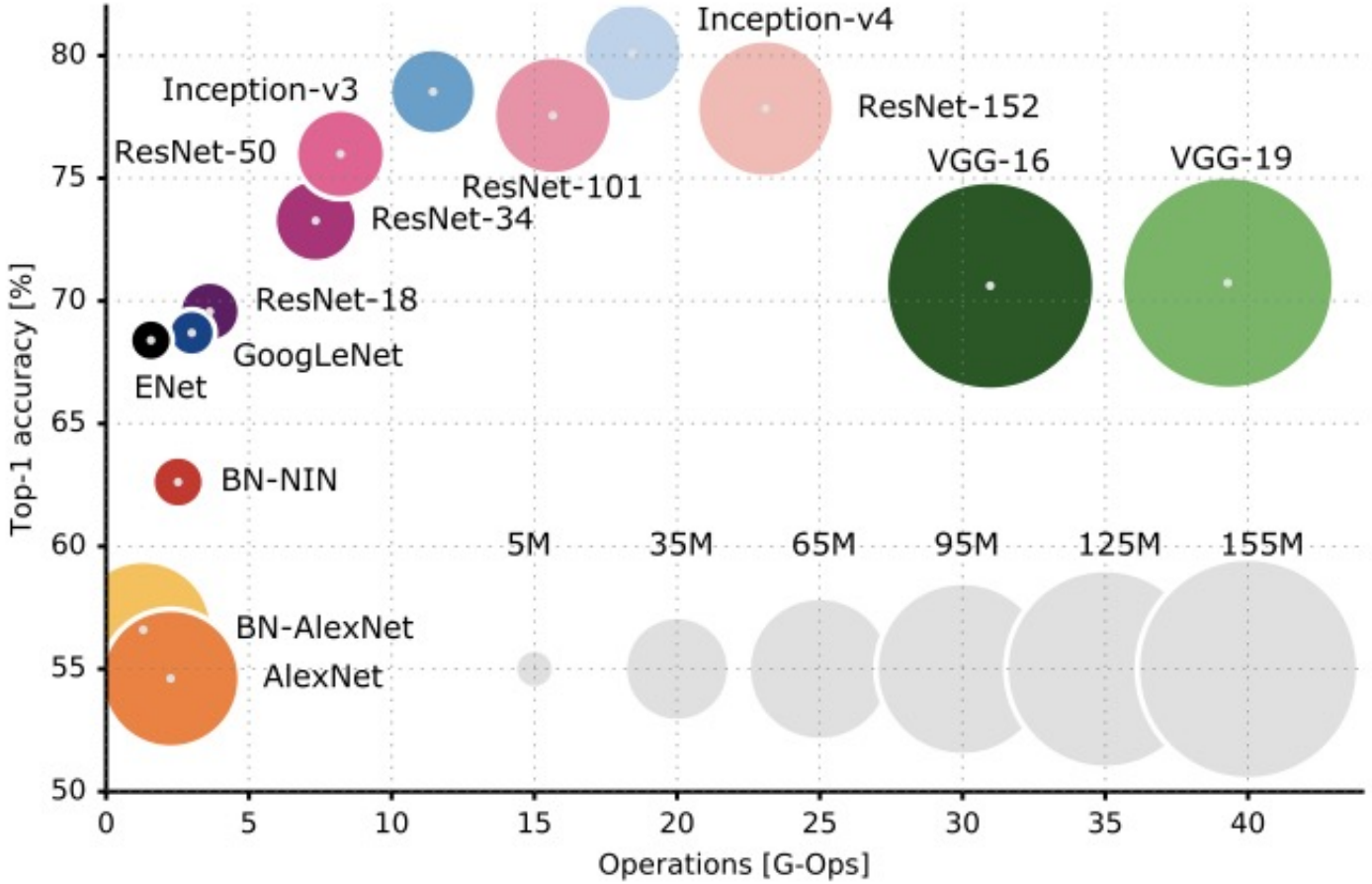
Each layer is receiving a “collective knowledge” from all preceding layers.

Summary: ILSVRC 2012-2015

Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (AlexNet, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
ResNet (152 layers)	2015	1st	3.57%	
Human expert*			5.1%	

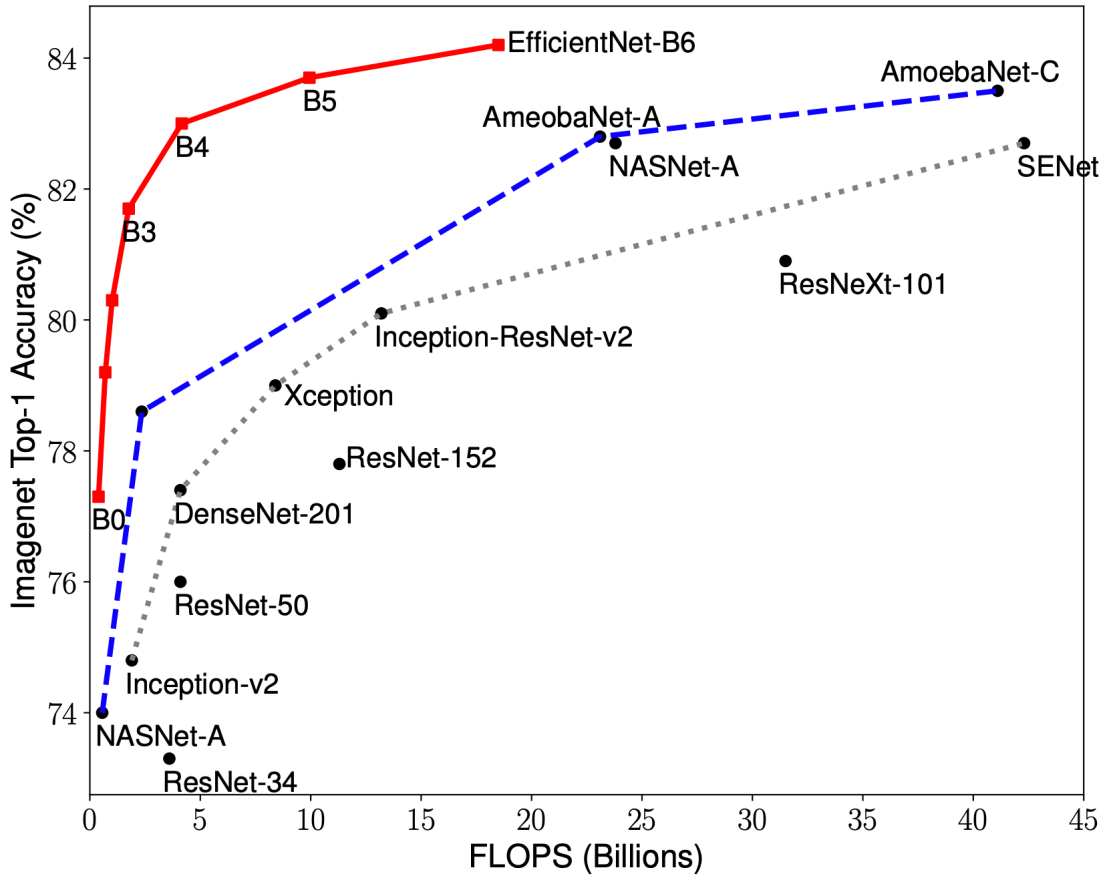
http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

Accuracy vs. efficiency



https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

Accuracy vs. efficiency



Pre-Trained Models for Image Classification

- VGG-16
- ResNet50
- Inceptionv3
- EfficientNet

Tan and Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, arXiv:1905.11946, 2020

Demo: Train a two-layer ReLU network on random data with L2 loss with TensorFlow, Keras and PyTorch

~~tensorflow101.ipynb (run it in Colab)~~
keras101.ipynb
pytorch101new.ipynb

Google Colab GPU
g.co/colab

The GPU used in the backend is **Tesla K80, P4, P100**
The 12-hour limit is for a continuous assignment of VM

Colab Pro ($9.99/month, cancel anytime), announced 2/2020

- Faster GPU: T4 or P100
- Longer runtimes: < 24-hour
- More memory: High-memory VMs (2x regular)

colab vm info

python v=3.6.3 (python3)
tensorflow v=1.6.0rc1
2 CPU VM with 1 optional GPU for free
tf device=/device:GPU:0
model name : Intel(R) Xeon(R) CPU @ 2.30GHz
model name : Intel(R) Xeon(R) CPU @ 2.30GHz
MemTotal: 13341960 kB (13GB ram)
MemFree: 4275200 kB
MemAvailable: 11659380 kB

Which version of python you are using?
(first select ' Python 2' in Notebook Settings; for 'Python 3' – no output)

```
!pip show python
```

Which version of tensorflow you are using?

```
!pip show tensorflow
```

Is GPU Working? (first select 'GPU' in Notebook Settings)

```
import tensorflow as tf
tf.test.gpu_device_name()
```

Which GPU Am I Using?

```
import torch
torch.cuda.get_device_name()
gpu_info = !nvidia-smi

from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

What about RAM?

```
!cat /proc/meminfo
```

What about CPU?

```
!cat /proc/cpuinfo
```