http://bit.ly/aiconfny18cv

Introductions & Networking

Agenda

- PyTorch Overview
- Computer Vision Overview
- Getting started with PyTorch
- Break
- Computer Vision Models using PyTorch
  - Image Classification
  - Transfer Learning
  - Object Detection
  - Object Segmentation
- Computer Vision Datasets
- Deploying PyTorch Models
- PyTorch & Computer Vision Next

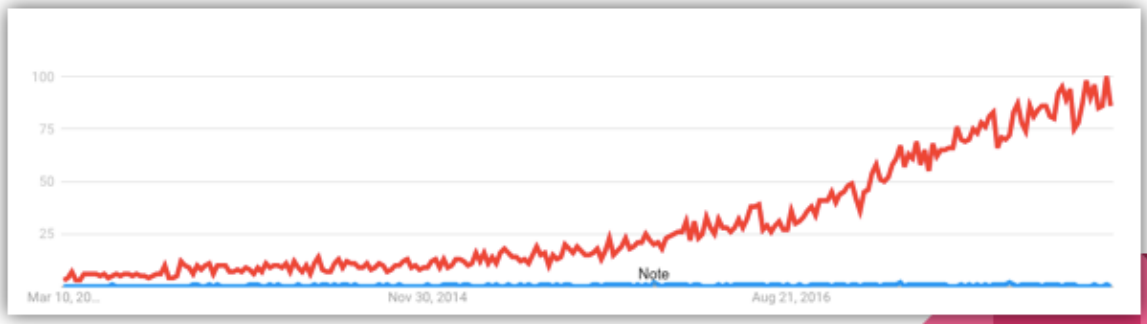
PyTorch Overview

http://bit.ly/aiconfny18cv

PyTorch Origins

- Torch – Lua
- Chainer
- HIPS Autograd

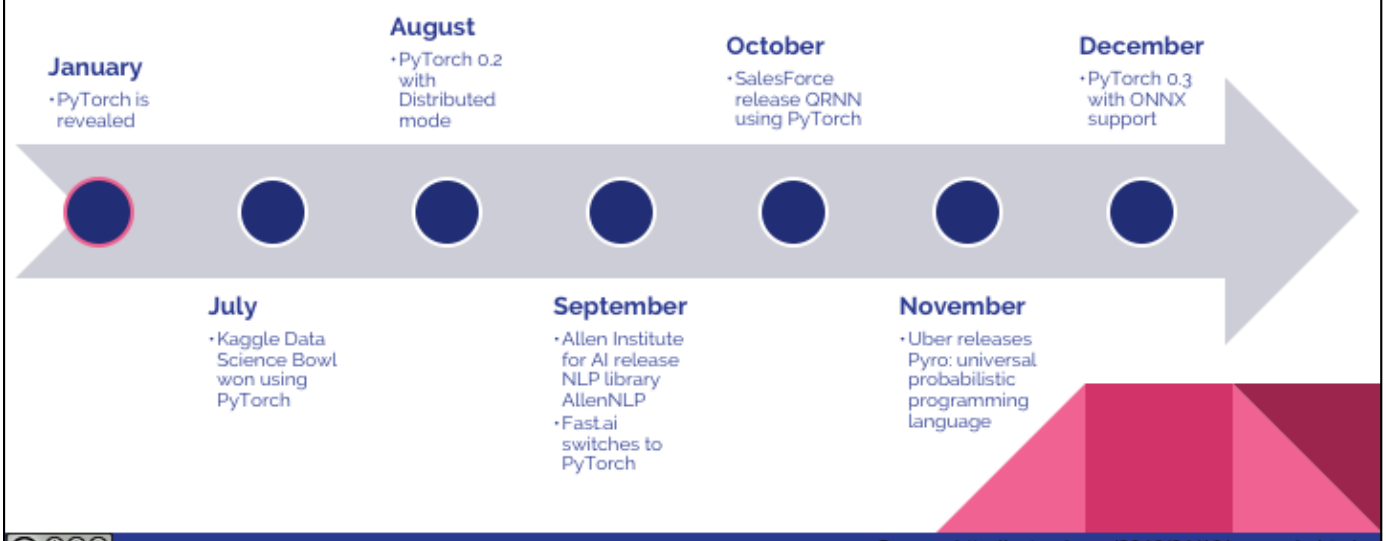
- Need for Dynamic execution library
- Popularity of Python in Machine Learning Community



Google Trends Last Five Years
Terms: Lua Machine Learning Python Machine Learning

http://bit.ly/aiconfny18cv

PyTorch – Year 2017 in Review



Source: http://pytorch.org/2018/01/19/a-year-in.html

PyTorch 2018

- April 2018: PyTorch 0.4
  - Tensors
    - Full support for advanced indexing
    - Fast Fourier Transforms
  - Neural Networks http://pytorch.org/2018/04/22/0_4_0-migration-guide.html
    - Trade-off memory for compute
    - bottleneck - a tool to identify hotspots in your code
  - torch.distributions
    - 24 basic probability distributions
    - Added cdf, variance, entropy, perplexity etc.
  - Distributed Training
    - Launcher utility for ease of use
    - NCCL2 backend
  - C++ Extensions
  - **Windows Support**
  - ONNX Improvements
    - RNN support

PyTorch – Community Overview

By the numbers

- 112,480 lines of Python code on github that import torch
- 4,959 repositories on Github that mention PyTorch in their name or description
- More than half a million downloads of PyTorch binaries. 651,916 to be precise.
- **5,400 users** wrote **21,500 posts** discussing 5,200 topics on PyTorch forums discuss.pytorch.org
- 131 mentions of PyTorch on Reddit's /r/machinelearning since the day of release. In the same period, TensorFlow was mentioned 255 times.

Innovation driver

- In the recent ICLR2018 conference submissions, PyTorch was mentioned in **87 papers**, compared to TensorFlow at 228 papers, Keras at 42 papers, Theano and Matlab at 32 papers.
- Francois Chollet's monthly arxiv.org mentions for frameworks had PyTorch at 72 mentions, with TensorFlow at 273 mentions, Keras at 100 mentions, Caffe at 94 mentions and Theano at 53 mentions.

Source: http://pytorch.org/2018/01/19/a-year-in.h

http://bit.ly/aiconfny18cv

Use Case Diversity: Research & Applications

PyTorch Comparison

- PyTorch dynamic computation allows flexibility of input and Python style debugging
- PyTorch best suited for research and prototyping, deployment via ONNX to other frameworks such as Caffe2, Apple CoreML, MXNet, Tensorflow
- Seasoned users of Python data stack (NumPy) can easily transition to PyTorch

Source: https://awni.github.io/pytorch-tensorflow/

Computer Vision Overview

http://bit.ly/aiconfny18cv

Computer Vision 101

- Image Properties
  - Pixels
  - Dimension
  - Channels
  - Color Depth

8-bit Image

Color Depth	Channel Range
1-Bit	0-1
8-Bit	0-255
16-Bit	0-65,535
24-Bit	0-16,777,215

One Pixel

Channel	Red	Green	Blue
Value	0-255	0-255	0-255

- Classification Image: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks
- Segmentation Image:https://www.cityscapes-dataset.com/examples/
- Detection Image: http://image-net.org/challenges/LSVRC/2015/ui/det.html

http://bit.ly/aiconfny18cv

Classification & Description

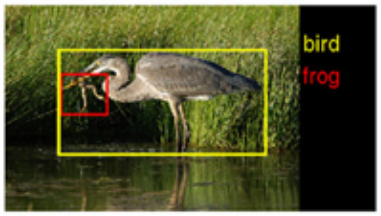
What is the man doing?

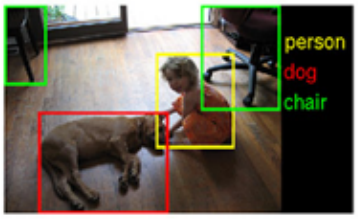
Submit

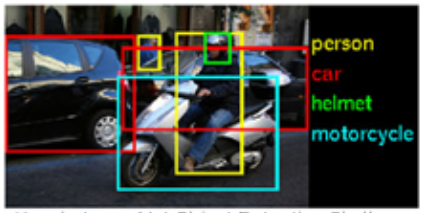
Predicted top-5 answers with confidence:

playing tennis	99.126%
tennis	0.553%
skateboarding	0.139%
serving	0.075%
jumping	0.033%

Localization







Source: Kaggle ImageNet Object Detection Challenge

Segmentation

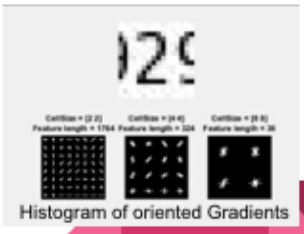
Source: The Cityscapes Dataset - Semantic, instance-wise, dense pixel annotations of 30 classes

http://bit.ly/aiconfny18cv

Computer Vision prior to Deep Learning

- Objectives
  - Extract generalized features (patterns) from images
  - Determine generalized descriptors for feature
  - Use generalized features to perform visual tasks based on descriptors

- What are features?
  - Edges
  - Shapes
  - Change in Colors/Intensity



https://www.mathworks.com/products/computer-vision/features.html

- Histogram of oriented gradients (HOG) feature extraction of image (top). Feature vectors of different sizes are created to represent the image by varying cell size.
- SURF: Speeded-up robust features (left), MSER: maximally stable extremal regions (center), and corner detection (right). Using the same image, three different feature types are detected and results plotted over the original image.

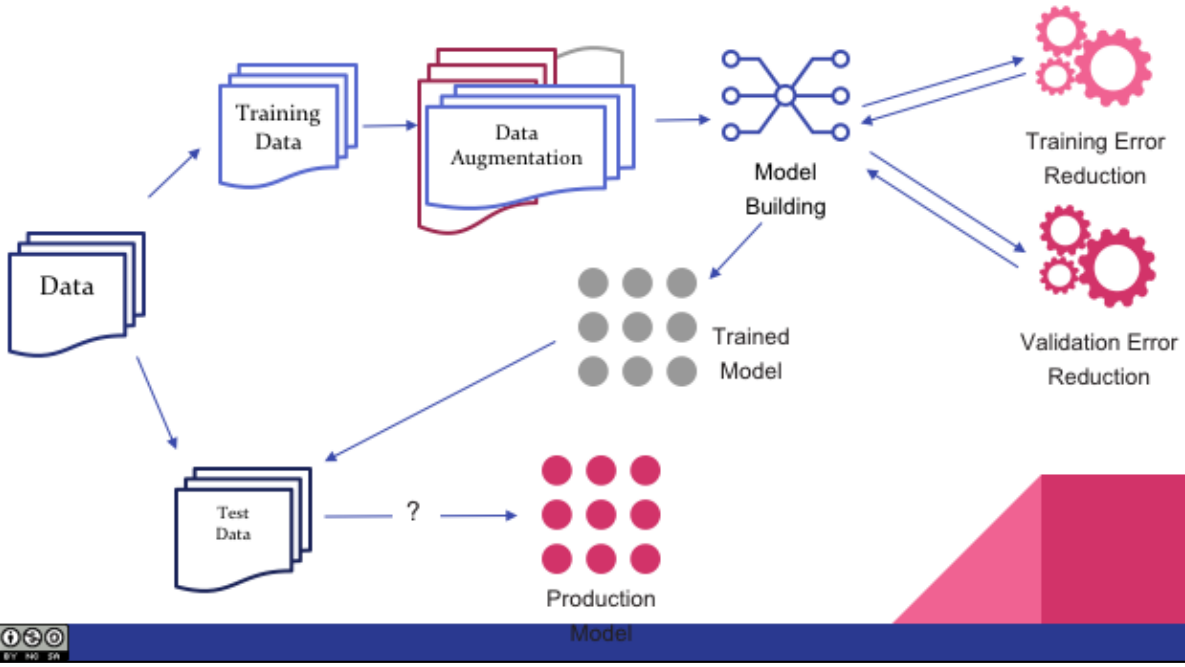
Traditional Machine Learning → Deep Learning

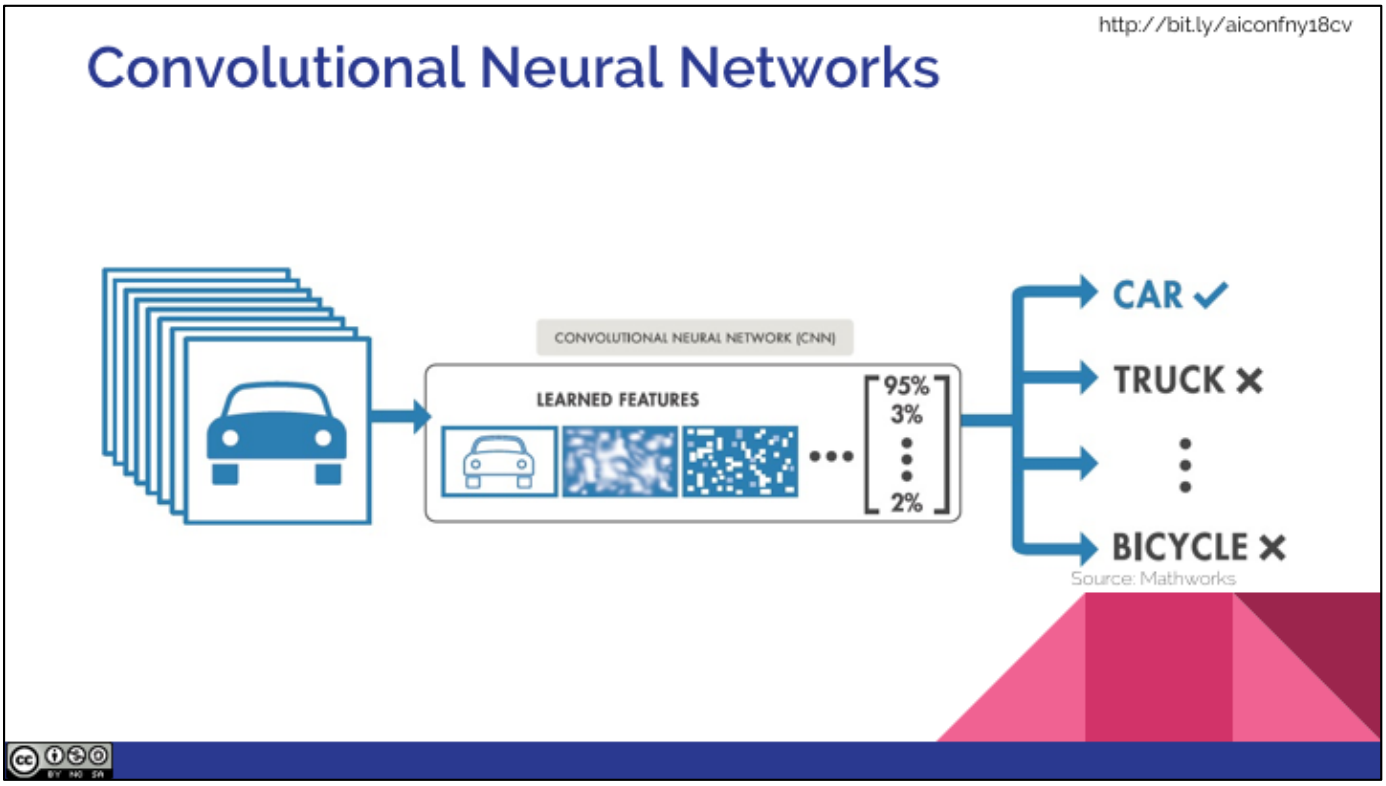
- Featuring engineering challenges
  - Human scaling: Time consuming to create rules for each object
  - Computational scaling: Translating human curated features to algorithms that machines can perform at scale is tedious
- Feature Learning
  - Design computational models to learn features
- Feature Learning Challenges
  - Need large datasets with annotations
  - Model architectures become complex both to design and run

Convolutional Neural Networks

http://bit.ly/aiconfny18cv

Deep (Supervised Machine) Learning Basics

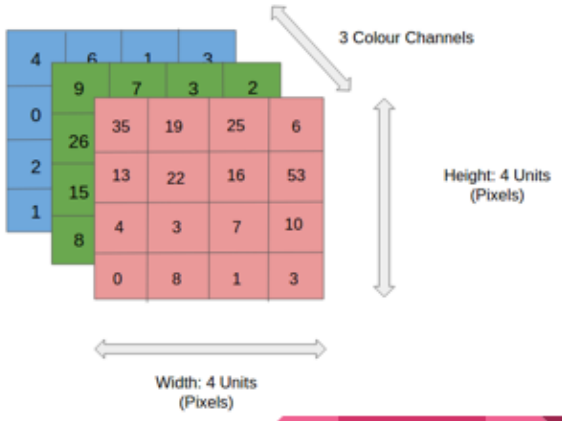




- https://www.mathworks.com/discovery/convolutional-neural-network.html

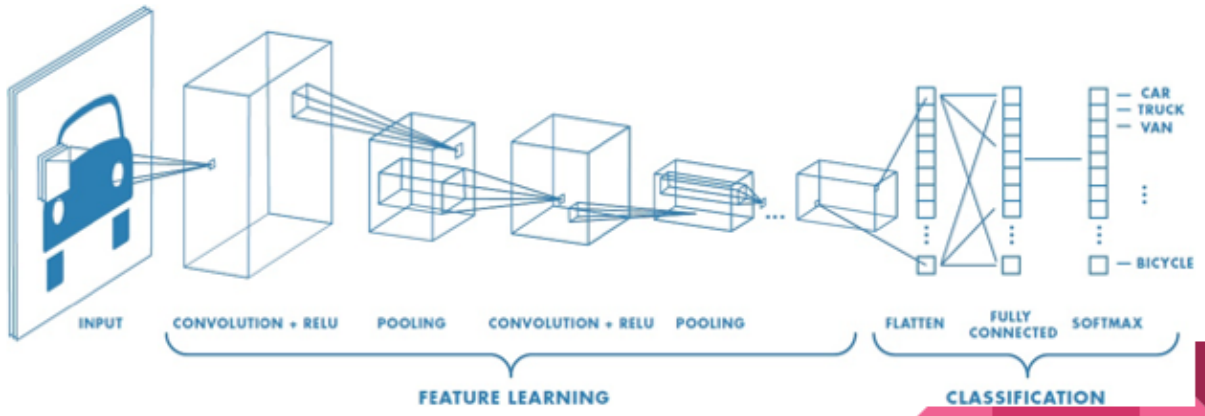
Convolutional Neural Network Properties

- Learning Low Level Features from High Dimensional Data
- Must Have Data Properties:
  - Shift Invariant
  - Correlated
- Simply put: Changing column order should not change meaning of Data
- Naturally Suitable Data Types:
  - Images, Audio, Video
  - Time Series



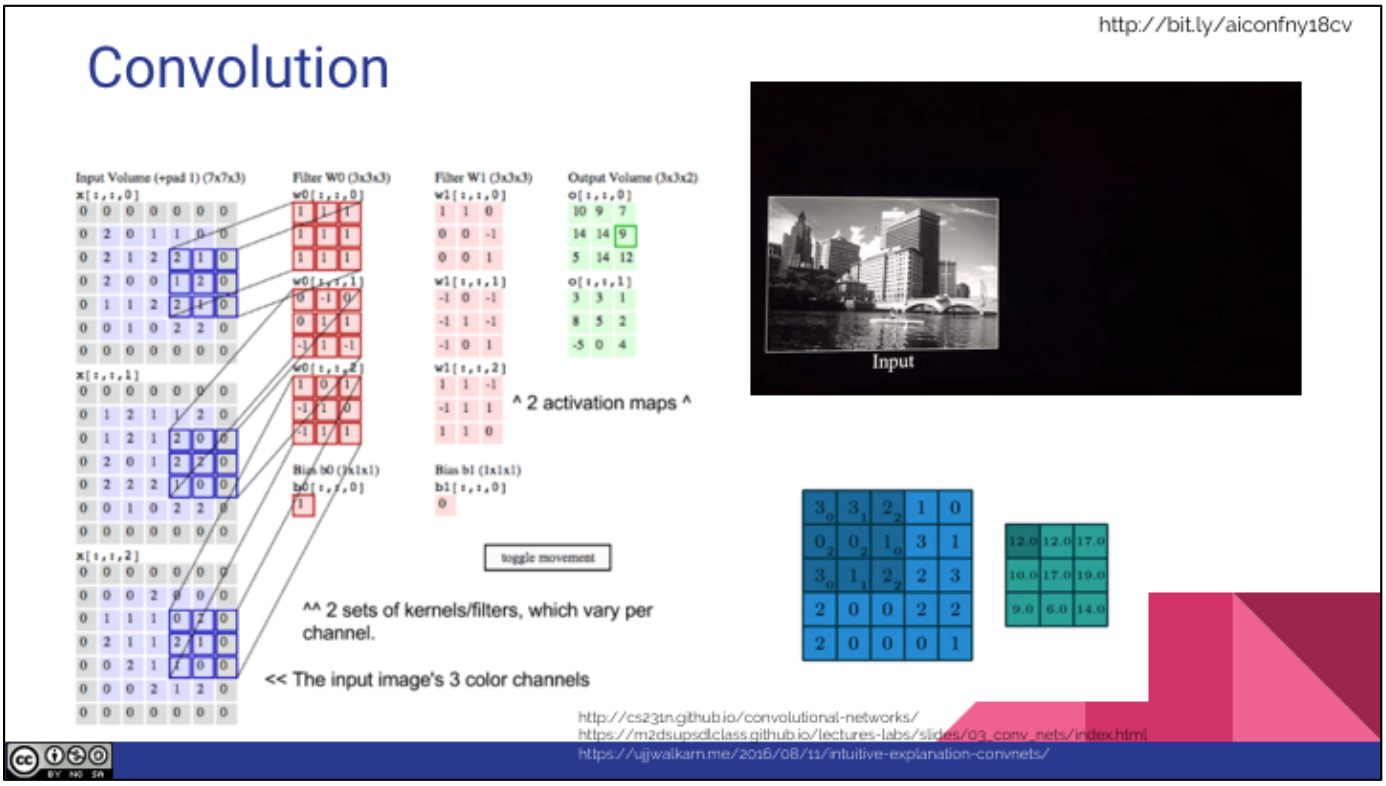
http://bit.ly/aiconfny18cv

Convolutional Neural Networks



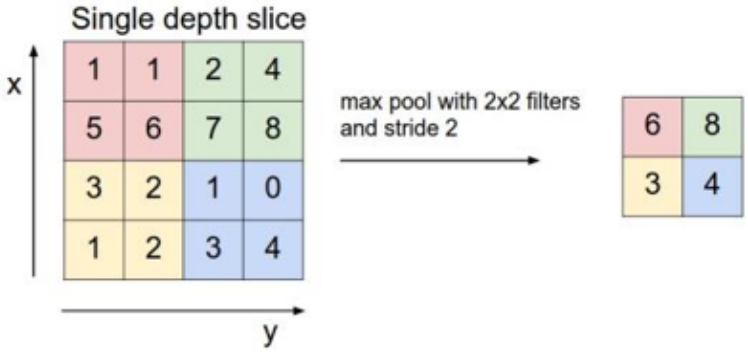
Source: Mathworks

- https://www.mathworks.com/discovery/convolutional-neural-network.html



- http://cs231n.github.io/convolutional-networks/
- https://m2dsupsdlclass.github.io/lectures-labs/slides/03_conv_nets/index.html
- https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

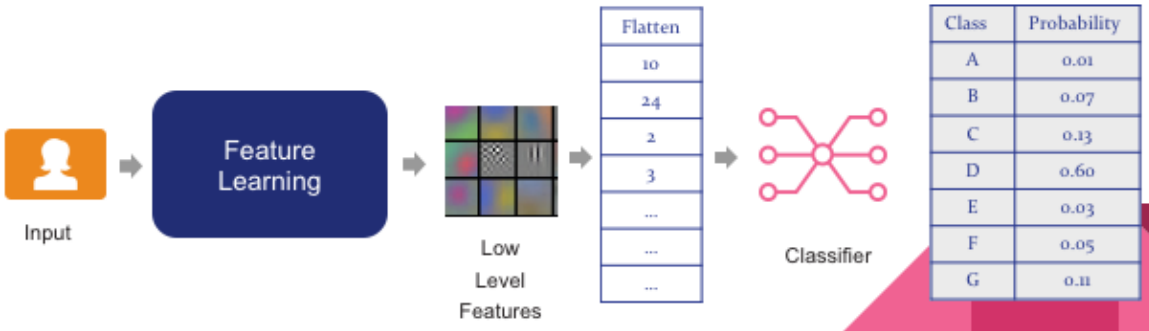
Pooling



Source: Stanford University CS 231 Course:
http://cs231n.github.io/convolutional-networks/#pool

Fully Connected

- Traditional Machine Learning: Feature Engineering
- Deep Learning: Feature Learning
- Learned Features are fed to traditional classifier such as Multivariate Logistic Regression (Softmax)



Loss Reduction

Class	Probability	Truth	Loss/ Error
A	0.01	0	0
B	0.07	0	0
C	0.13	0	0
D	0.60	1	0.22
E	0.03	0	0
F	0.05	0	0
G	0.11	0	0

Cross Entropy Loss

One-Hot Encoded Vector of Class Label

Back Propagation

- Account for loss across the weights and biases in the network
- Use cost function provided by Back Propagation to optimize the weights and biases to reduce loss
- Variety of Stochastic Gradient Descent techniques used for optimization

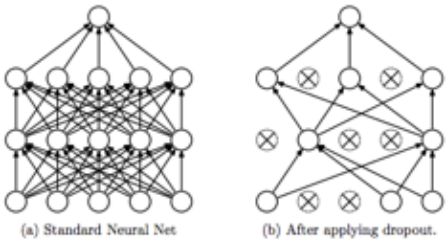
Overview of Gradient Descent Techniques:
http://ruder.io/optimizing-gradient-descent/

https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/

Generalization

Dropout

Randomly remove learned nodes from network



(a) Standard Neural Net (b) After applying dropout.

Dropout: A Simple Way to Prevent Neural Networks from Overfitting
http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Batch Normalization

- During training input data shifts as weights and parameter adjust values
- To prevent internal covariate shift, normalize each batch by mean and variance

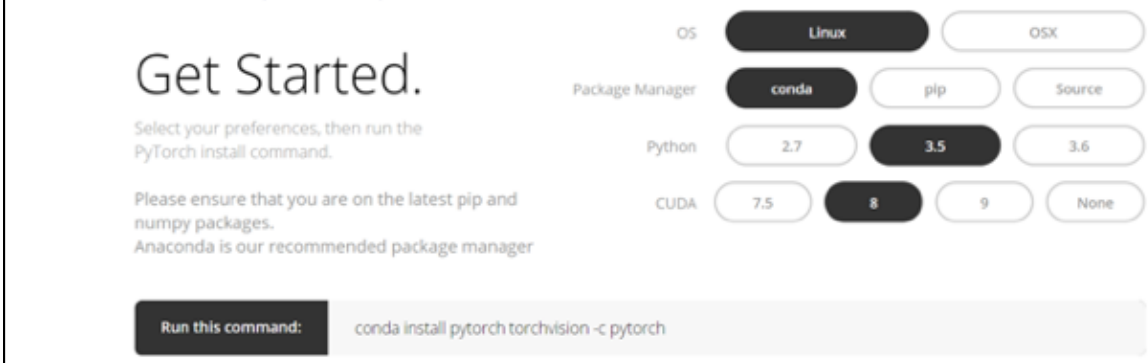
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
https://arxiv.org/abs/1502.03167

Getting started with PyTorch

http://bit.ly/aiconfny18cv

Installation

http://pytorch.org/



http://bit.ly/aiconfny18cv

PyTorch Basics Jupyter Notebook

- Using Jupyter Notebook, please open *pytorch_basics.ipynb* from the tutorial repository on Github.

Automatic Differentiation

Automatic Differentiation

Chain rule for today's computational needs:

1. Technique to determine derivative of function within a function
2. Complex functions can be written as compositions of simple functions

- PyTorch `autograd` package: Provides automatic differentiation on all tensor operations

torch.Tensor

- Central class for autograd
  - Wraps a tensor and records the history use `requires_grad=True`
  - Supports all tensor operations
- Use .backward
  - Gradients computed automatically

Need more here

PyTorch Autograd Notebook

- Using Jupyter Notebook, please open *autograd_tutorial.ipynb* from the tutorial repository on Github.

PyTorch
Neural Net & Optimization
Package Overview

PyTorch's nn module

- Containers
- Convolution Layers
- Pooling Layers
- Padding Layers
- Non-linear Activations (weighted sum + nonlinearity)
- Non-linear Activations (Other)
- Normalization layers
- Recurrent layers
- Linear layers
- Dropout layers
- Sparse layers
- Distance functions
- Loss functions
- Vision layers
- DataParallel layers (multi-GPU, distributed)
- Utilities

http://pytorch.org/docs/stable/nn.html

http://bit.ly/aiconfny18cv

PyTorch's optim algorithms

Algorithms		Algorithms	
adadelta	Adaptive Delta	sgd	Stochastic Gradient Descent
adagrad	Adaptive Subgradients	asgd	Averaged Stochastic Gradient Descent
adam	Stochastic Optimization	lbfgs	Limited BFGS
adamax	Adam based on Infinity Norm	rprop	Resilient Backpropagation
sparse_adam	Adam suitable for sparse tensors	rmsprop	Root Mean Squared Propagation

http://pytorch.org/docs/stable/optim.html

Adam Adaptive Moment Estimation
BFGS - used for unconstrained nonlinear optimization problems

TorchVision

PyTorch Torchvision

- Datasets
  - MNIST
  - Fashion-MNIST
  - EMNIST
  - COCO
  - LSUN
  - Imagenet-12
  - CIFAR
  - STL10
  - SVHN
  - PhotoTour
- Utilities
  - make_grid
  - save_image
- Models
  - Alexnet
  - VGG
  - ResNet
  - SqueezeNet
  - DenseNet
  - Inception v3
- Transforms

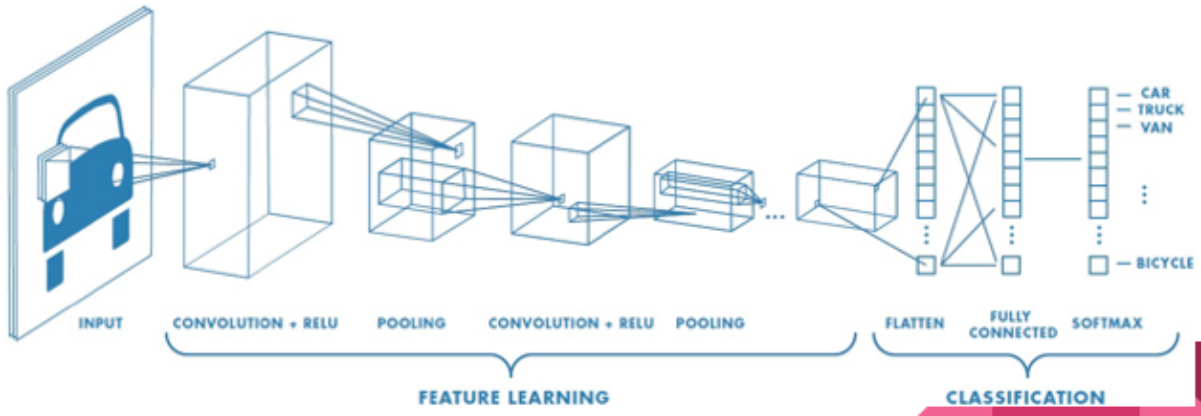
http://pytorch.org/docs/stable/torchvision/

Break

Anatomy of a PyTorch Model

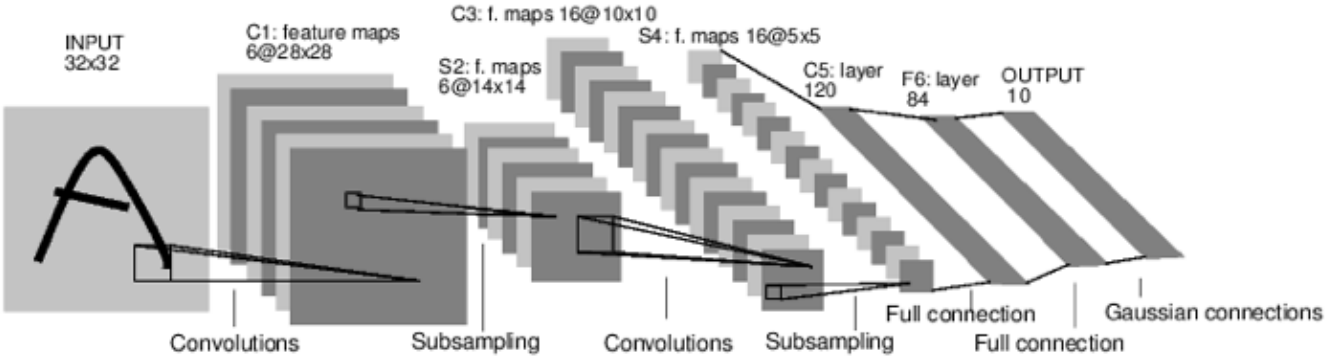
http://bit.ly/aiconfny18cv

Image Classification



Source: Mathworks

Image Classification



Source: https://awni.github.io/pytorch-tensorflow/

Typical PyTorch CNN Training Code

1. Imports
2. Declare Hyperparameters
3. Define Train & Test data
4. Define the network
5. Define the Loss Function
6. Define the Backward Propagation using Automatic Differentiation
7. Define Optimizer
8. Define Training method
9. Define Accuracy calculation
10. Calculate Accuracy on Test data

Define the network

```
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features


net = Net()
print(net)
```

```
Net(
  (conv1): Conv2d (1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d (6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120)
  (fc2): Linear(in_features=120, out_features=84)
  (fc3): Linear(in_features=84, out_features=10)
)
```

Define the network

LeNet Image Input

```
input = Variable(torch.randn(1, 1, 32, 32))
out = net(input)
print(out)
```

```
Variable containing:
-0.0573 -0.0559 -0.1204  0.1070 -0.0275 -0.0118  0.0264  0.0515 -0.0257  0.0483
[torch.FloatTensor of size 1x10]
```

```
net.zero_grad()
out.backward(torch.randn(1, 10))
```

Always zero out the gradients they are accumulated

The Loss Function and Back Propagation

Example loss calculation

```
output = net(input)
target = Variable(torch.arange(1, 11))
criterion = nn.MSELoss()

loss = criterion(output, target)
print(loss)
```

```
Variable containing:
 38.3962
[torch.FloatTensor of size 1]
```

```
net.zero_grad()     # zeroes the gradient buffers of all parameters

print('conv1.bias.grad before backward')
print(net.conv1.bias.grad)

loss.backward()

print('conv1.bias.grad after backward')
print(net.conv1.bias.grad)
```

```
conv1.bias.grad before backward
Variable containing:
 0
 0
 0
 0
 0
 0
[torch.FloatTensor of size 6]

conv1.bias.grad after backward
Variable containing:
1.00000e-02 *
 -6.1643
 -2.5201
 -4.5238
  6.8188
 -0.2024
  3.4750
[torch.FloatTensor of size 6]
```

Updating the weights

```
import torch.optim as optim

# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

Training

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # wrap them in Variable
        inputs, labels = Variable(inputs), Variable(labels)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.data[0]
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.220
[1,  4000] loss: 1.887
[1,  6000] loss: 1.676
[1,  8000] loss: 1.578
[1, 10000] loss: 1.511
[1, 12000] loss: 1.462
[2,  2000] loss: 1.404
[2,  4000] loss: 1.373
[2,  6000] loss: 1.373
[2,  8000] loss: 1.325
[2, 10000] loss: 1.315
[2, 12000] loss: 1.267
Finished Training
```

Accuracy

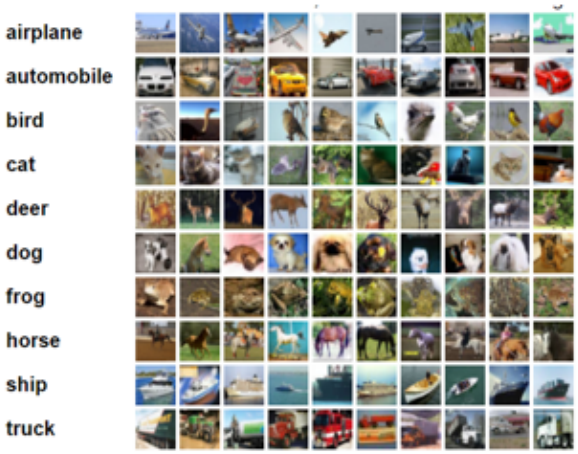
```
correct = 0
total = 0
for data in testloader:
    images, labels = data
    outputs = net(Variable(images))
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 55 %
```

Computer Vision Models

PyTorch CIFAR10 Tutorial

- Ten classes
- Dataset:
  - 60,000 32x32 color images in 10 classes
  - 6000 images per class.
  - 50,000 training images
  - 10,000 test images

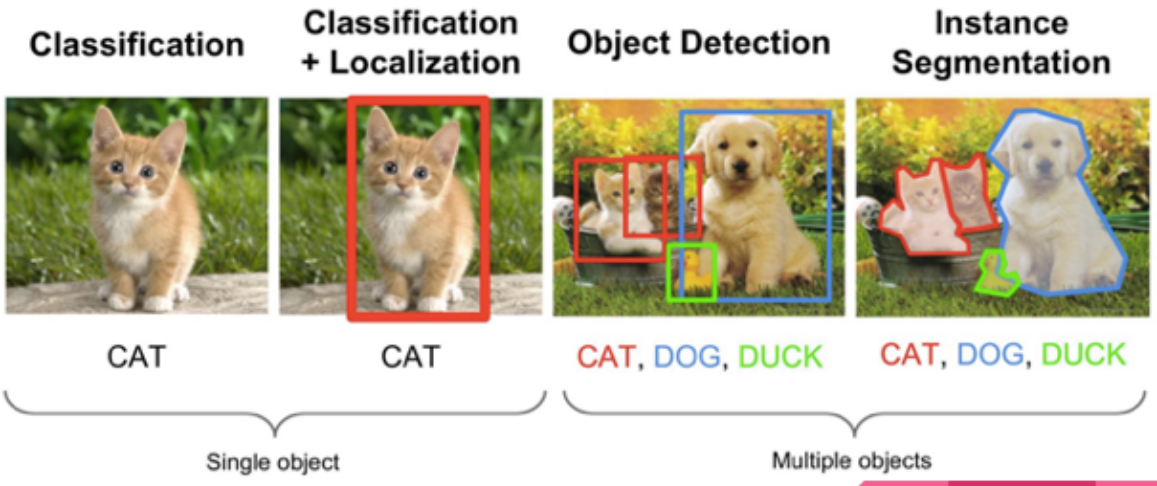


https://www.cs.toronto.edu/~kriz/cifar.html

PyTorch CIFAR10 Tutorial

- Using Jupyter Notebook, please open *cifar10_tutorial.ipynb* from the tutorial repository on Github.

http://bit.ly/aiconfny18cv



https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

Object Detection/Localization in PyTorch

- Review: https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318
- SSD: https://towardsdatascience.com/learning-note-single-shot-multibox-detector-with-pytorch-part-1-38185e84bd79
- Faster R-CNN: https://github.com/jwyang/faster-rcnn.pytorch
- YOLO v3: https://github.com/ayooshkathuria/pytorch-yolo-v3

Source: https://awni.github.io/pytorch-tensorflow/

http://bit.ly/aiconfny18cv

Image Segmentation

- Evolution from Classification → Detection → Segmentation
- Goal: Pixel level identification
- Metrics: Jaccard Index

$$IoU = \frac{True\ Positive}{True\ Positive + False\ Positive + False\ Negative}$$

- Datasets:
  - PASCAL VOC
  - MS COCO
  - Cityscapes
  - ADE20K
  - Imagenet
  - KITTI
- Resources:
  - https://meetshah1995.github.io/semantic-segmentation/deep-learning/pytorch/visdom/2017/06/01/semantic-segmentation-over-the-years.html

Source: https://awni.github.io/pytorch-tensorflow/

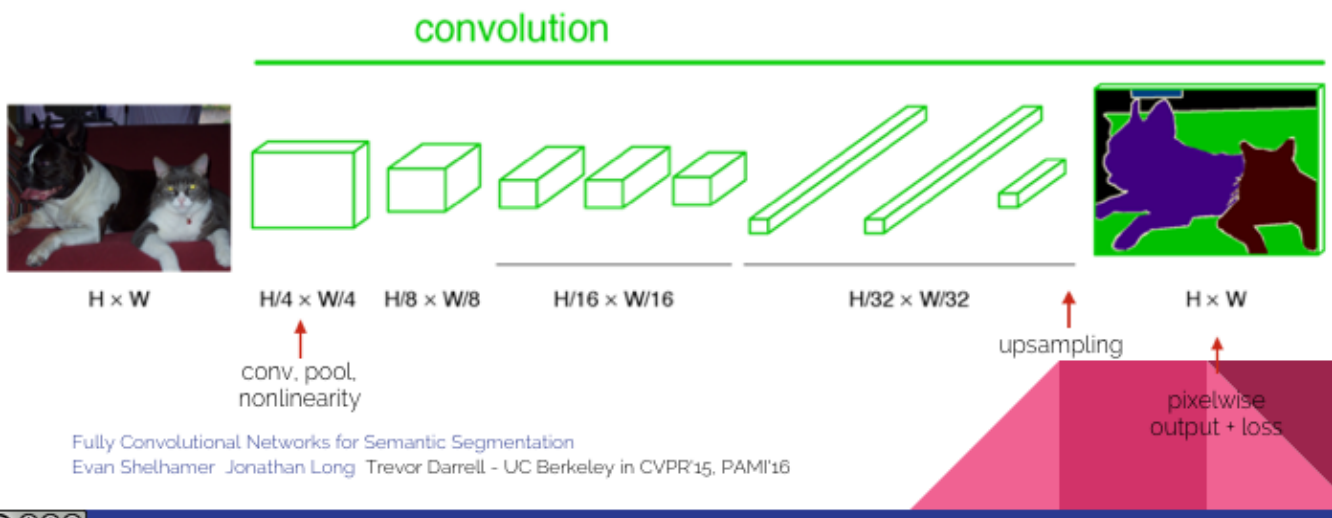
- Source Cityscape Dataset: To assess performance, we rely on the standard Jaccard Index, commonly known as the PASCAL VOC intersection-over-union metric IoU=TP/(TP+FP+FN), where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole test set.
- Owing to the two semantic granularities, i.e. classes and categories, we report two separate

mean
performance scores: $IoU_{category}$ and Io U_{class}.

- In either case, pixels labeled as void do not contribute to the score.

Fully Convolutional Networks

Segmentation Objective: Predict Pixel Label

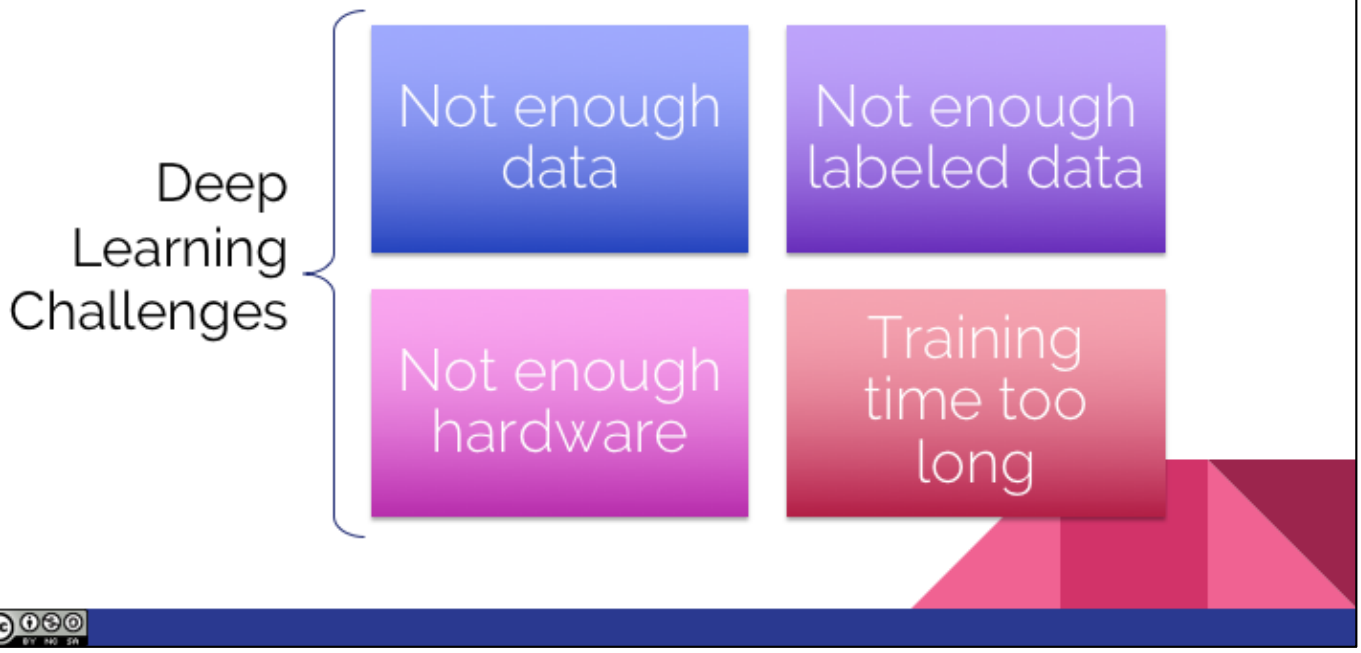


Fully Convolutional Networks for Semantic Segmentation
Evan Shelhamer Jonathan Long Trevor Darrell - UC Berkeley in CVPR'15, PAMI'16

PyTorch implementation: https://github.com/wkentaro/pytorch-fcn

http://bit.ly/aiconfny18cv

Why Transfer Learning?



http://bit.ly/aiconfny18cv

Transfer Learning

Domain to Domain Learning

- Learn Features, Weights, Biases in one domain
- Find domain with similar data characteristics and distribution
- Apply model in new domain

Retraining

- Freeze layers in a "good" model up to tunable depth of feature extraction but before final prediction layer
- Remove final layer(s)
- Relearn final and classification layers using your own data

Bonus: Combine Domain to Domain and Retraining

http://bit.ly/aiconfny18cv

PyTorch Transfer Learning Tutorial

- Using Jupyter Notebook, please open *transfer_learning_tutorial.ipynb* from the tutorial repository on Github.

Working with Computer Vision Datasets

Working with video

Extract Frames and Compose Video using FFMPEG
https://www.ffmpeg.org/

Extract `ffmpeg -i <video.file> image%03d.jpg`

Compose `ffmpeg -framerate 24 -i img%03d.jpg <output_video.file>`

Data Augmentation

- `torchvisions.transforms`
  - `ColorJitter`
  - `Grayscale`
  - `LinearTransformation`
  - `Pad`
  - `RandomSizedCrop`
  - `RandomAffine`
  - `RandomApply`
  - `RandomChoice`
  - `RandomGrayscale`
  - `RandomHorizontalFlip`
  - `RandomOrder`
  - `RandomResizedCrop`
  - `RandomRotation`
  - `RandomSizedCrop`
  - `RandomVerticalFlip`

Image Source: Deep Image: Scaling up Image Recognition (Baidu)
https://arxiv.org/abs/1501.02876

Data Annotation


```
{"coco_url":"http://mscoco.org/images/405628",
"date_captured":"2013-11-21 00:26:09",
"file_name":"COCO_train2014_000000405628.jpg",
"flickr_url":"http://farm5.staticflickr.com/4130/5133418813_61f33c
9633_z.jpg",
"height":426,
"id":405628,
"license":5,
"width":640}
```

```
{"area":7581.427150000003,
"bbox":[95.28,146.43,131.92,142.0],
"category_id":1,"id":188293,
"image_id":405628,
"iscrowd":0,
"segmentation":[[219.87,177.58,224.45,172.08,225.37,157.42,218.95
,146.43,208.88,149.18,208.88,149.18,202.46,151.92,198.8,153.76,19
8.8,153.76,196.97,154.67,184.14,153.76,183.23,153.76,158.49,157.4
2,133.75,157.42,95.28,186.74,105.35,183.99,153.91,164.75,169.48,1
74.83,169.48,197.73,135.59,231.63,131.92,235.29,133.75,247.2,152.
99,247.2,155.74,247.2,175.9,241.71,179.56,228.88,184.14,278.35,19
5.14,280.18,194.22,288.43,200.63,282.93,198.8,271.02,224.45,227.0
5,224.45,227.05,227.2,196.82,219.87,183.99]]},
```

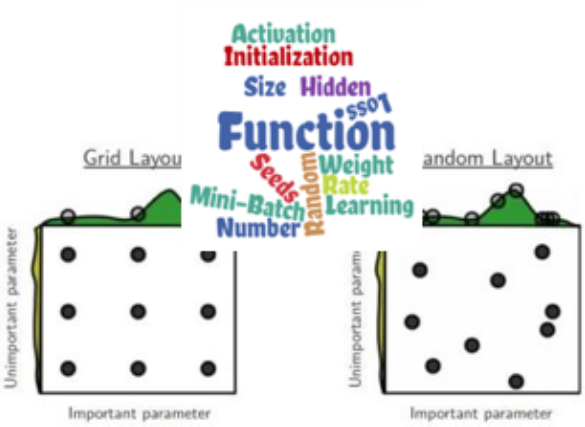

Data Annotation

	BOUNDING BOX	DOTS	POLYGON	SEMANTIC SEGMENTATION
COST	Least expensive	More expensive	More expensive	Most expensive
TIME COMMITMENT	Lowest	Medium	Medium	Highest
PRECISION	Good	Great	Great	Excellent
INSTANCE-BASED (OUTPUT CONTAINS DISCRETE OBJECTS)	Yes	Yes	Yes	No
POSSIBLE TO LABEL SEVERAL OBJECTS?	Yes	Yes	Yes	Yes
POSSIBLE TO LABEL SEVERAL CLASSES OF OBJECT IN A SINGLE JOB?	No	No	No	Yes
OUTPUT*	X,Y coordinate, width and length of each box	Series of x,y coordinates	Series of x,y coordinates, with shapes resolving	Coded RGB pixels as an image

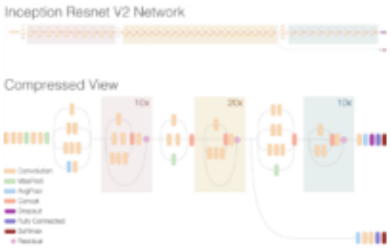
Source: CrowdFlower WHAT WE LEARNED LABELING 1 MILLION IMAGES

http://bit.ly/aiconfny18cv

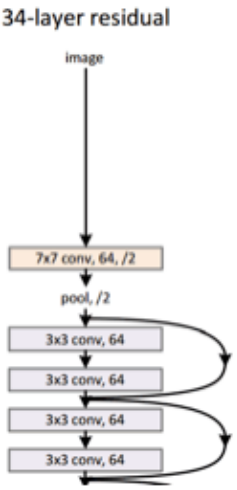
Training Challenges



Hyper-Parameter Optimization



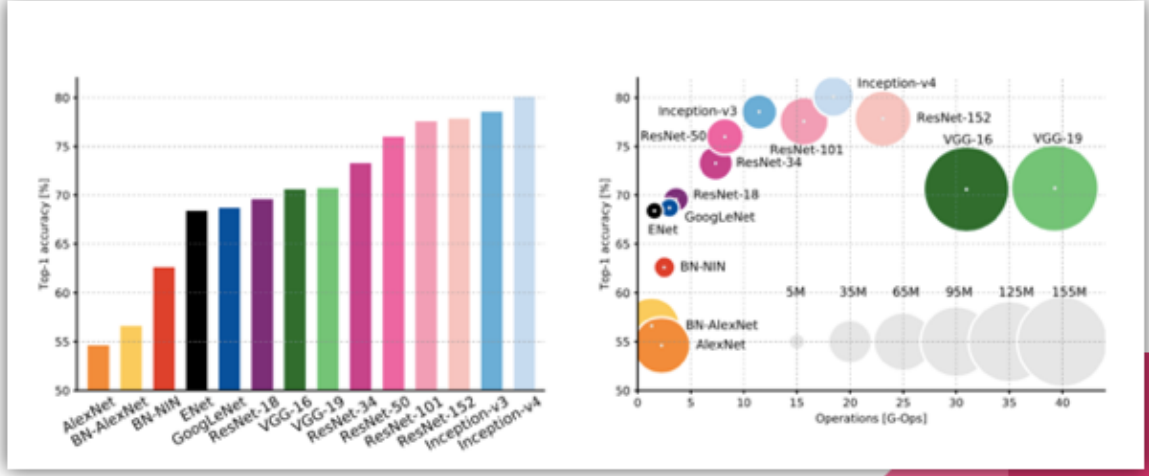
Google Inception v3



Microsoft Resnet

https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html
http://colinraffel.com/wiki/neural_network_hyperparameters

Analysis from training Computer Vision Models



An Analysis of Deep Neural Network Models for Practical Applications https://arxiv.org/abs/1605.07678

Training Computer Vision Models: Time & Cost

Batch Size	epochs	Top-1 Accuracy	hardware	cost ($)	time
256	100	58.7%	8-core CPU + K20 GPU	3,000	144h
512	100	58.8%	1 DGX station	129,000	6h 10m
4096	100	58.4%	1 DGX station	129,000	2h 19m
32K	100	58.5%	512 KNLs	1.2 million	24m

Table 7: The speed and hardware cost for training AlexNet.
For batch size=32K, we changed local response norm in AlexNet to batch norm.

Batch Size	epochs	Top-1 Accuracy	hardware	cost ($)	time
256	90	73.0%	1 DGX station	129,000	21h
8192	90	72.7%	1 DGX station	129,000	21h
8192	90	72.7%	32 DGX stations	4.1 million	1h
32K	90	72.4%	512 KNLs	1.2 million	1h

Table 8: The speed and hardware cost for training ResNet50. We did not use data augmentation.

ImageNet Training in 24 Minutes https://arxiv.org/abs/1709.05011

Deploying PyTorch Models

Deploying PyTorch Models: ONNX

- ONNX: Open Neural Network eXchange
  - Framework fragmentation: Project goal to allow researchers and developers to port models from one framework to another allowing interoperability
  - Currently supports some of Caffe2, PyTorch, Microsoft Cognitive Toolkit, Apache MXNet
- Currently PyTorch only supports model export not import
- Resources:
  - ONNX: https://onnx.ai/ & https://github.com/onnx/tutorials
  - PyTorch: http://pytorch.org/docs/master/onnx.html

PyTorch AlexNet model to ONNX

```
from torch.autograd import Variable
import torch.onnx
import torchvision

dummy_input = Variable(torch.randn(10, 3, 224, 224)).cuda()
model = torchvision.models.alexnet(pretrained=True).cuda()

# providing these is optional, but makes working with the
# converted model nicer.
input_names = [ "learned_%d" % i for i in range(16) ] + [ "actual_input_1" ]
output_names = [ "output1" ]

torch.onnx.export(model, dummy_input,
          "alexnet.proto", verbose=True,
          input_names=input_names, output_names=output_names)
```


http://pytorch.org/docs/stable/onnx.html#example-end-to-end-alexnet-from-pytorch-to-caffe2

Readable Model

```
    # All parameters are encoded explicitly as inputs.  By convention,
    # learned parameters (ala nn.Module.state_dict) are first, and the
    # actual inputs are last.
    graph(%learned_0 : Float(10, 3, 224, 224)
          %learned_1 : Float(64, 3, 11, 11)
          # The definition sites of all variables are annotated with type
          # information, specifying the type and size of tensors.
          # For example, %learned_2 is a 192 x 64 x 5 x 5 tensor of floats.
          %learned_2 : Float(64)
          %learned_3 : Float(192, 64, 5, 5)
          # ---- omitted for brevity ----
          %learned_14 : Float(4096)
          %learned_15 : Float(1000, 4096)
          %actual_input_1 : Float(1000)) {
      # Every statement consists of some output tensors (and their types),
      # the operator to be run (with its attributes, e.g., kernels, strides,
      # etc.), its input tensors (%learned_0, %learned_1, %learned_2)
      %17 : Float(10, 64, 55, 55) = Conv[dilations=[1, 1], group=1, kernel_shape=[11, 11], pads=[2, 2, 2, 2], strides=[4, 4]](%learned_0, %learned_1, %learned_2), scope:
AlexNet/Sequential[features]/Conv2d[0]
      %18 : Float(10, 64, 55, 55) = Relu(%17), scope: AlexNet/Sequential[features]/ReLU[1]
      %19 : Float(10, 64, 27, 27) = MaxPool[kernel_shape=[3, 3], pads=[0, 0, 0, 0], strides=[2, 2]](%18), scope: AlexNet/Sequential[features]/MaxPool2d[2]
      # ---- omitted for brevity ----
      %29 : Float(10, 256, 6, 6) = MaxPool[kernel_shape=[3, 3], pads=[0, 0, 0, 0], strides=[2, 2]](%28), scope: AlexNet/Sequential[features]/MaxPool2d[12]
      %30 : Float(10, 9216) = Flatten[axis=1](%29), scope: AlexNet
      # UNKNOWN_TYPE: sometimes type information is not known.  We hope to eliminate
      # all such cases in a later release.
      %31 : Float(10, 9216), %32 : UNKNOWN_TYPE = Dropout[is_test=1, ratio=0.5](%30), scope: AlexNet/Sequential[classifier]/Dropout[0]
      %33 : Float(10, 4096) = Gemm[alpha=1, beta=1, broadcast=1, transB=1](%31, %learned_11, %learned_12), scope: AlexNet/Sequential[classifier]/Linear[1]
      # ---- omitted for brevity ----
      %output1 : Float(10, 1000) = Gemm[alpha=1, beta=1, broadcast=1, transB=1](%38, %learned_15, %actual_input_1), scope: AlexNet/Sequential[classifier]/Linear[6]
      # Finally, a network returns some tensors
      return (%output1);
    }
```


http://pytorch.org/docs/stable/onnx.html#example-end-to-end-alexnet-from-pytorch-to-caffe2

PyTorch AlexNet model to ONNX

```
import onnx
# Load the ONNX model

model = onnx.load("alexnet.proto")
# Check that the IR is well formed
onnx.checker.check_model(model)

# Print a human readable representation of the graph
onnx.helper.printable_graph(model.graph)

import caffe2.python.onnx.backend as backend
import numpy as np

rep = backend.prepare(model, device="CUDA:0") # or "CPU"
# For the Caffe2 backend:
#     rep.predict_net is the Caffe2 protobuf for the network
#     rep.workspace is the Caffe2 workspace for the network
#       (see the class caffe2.python.onnx.backend.Workspace)
outputs = rep.run(np.random.randn(10, 3, 224, 224).astype(np.float32))
# To run networks with more than one input, pass a tuple
# rather than a single numpy ndarray.

print(outputs[0])
```


http://pytorch.org/docs/stable/onnx.html#example-end-to-end-alexnet-from-pytorch-to-caffe2

PyTorch & Computer Vision Next

What is next for PyTorch?

- Tensor Comprehensions now available in PyTorch speed up innovation in Deep Learning by allowing researchers to scale new ideas to large models

```
def avgpool(float(B, C, H, W) input) -> (output) {{
    output(b, c, h, w) += input(b, c, h * {sH} + kh, w * {sW} + kw)
                          where kh in 0:{kH}, kw in 0:{kW}
}}
```

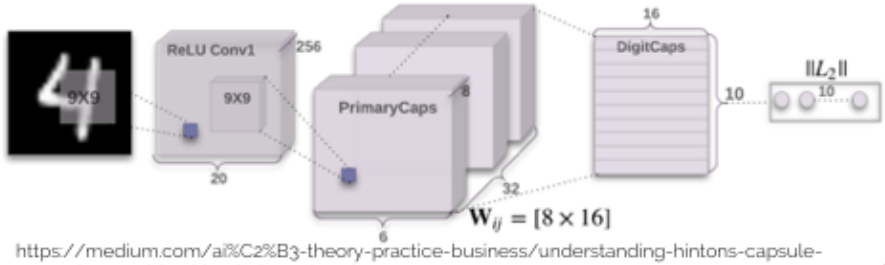
Tensor Comprehension for 2D Average Pooling

https://research.fb.com/announcing-tensor-comprehensions/

Computer Vision *Next*

- Generative Adversarial Networks (GANs)
  - Deep Convolution Generative Adversarial Networks (https://github.com/pytorch/examples/tree/master/dcgan)
- Capsule Networks

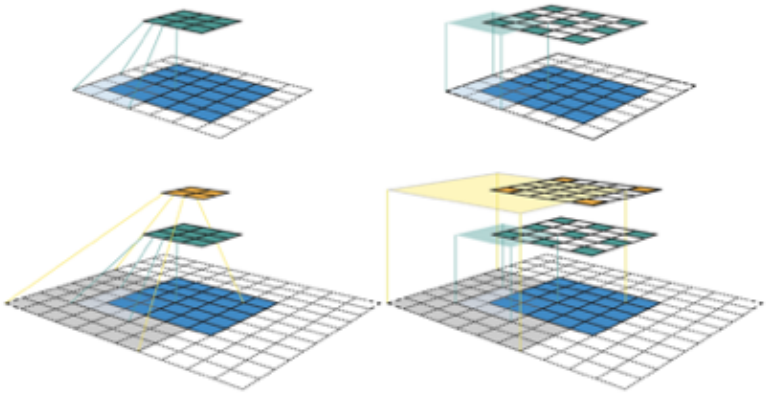
  Dynamic Routing Between Capsules (https://arxiv.org/abs/1710.09829)



https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b

Features and Receptive Field

- A guide to receptive field arithmetic for Convolutional Neural Networks



https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

PyTorch: A flexible approach for computer vision models

Discussion