中国科学院自动化研究所
Institute of Automation
Chinese Academy of Sciences

2022

"Deep Learning Lecture"

Lecture 12：Applications in DL

Yan Huang

Center for Research on Intelligent Perception and Computing (CRIPAC)

National Laboratory of Pattern Recognition (NLPR)

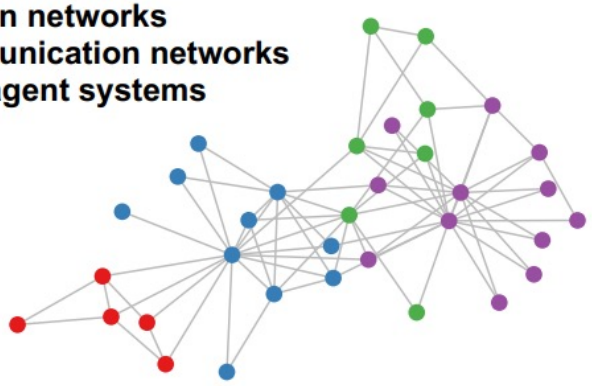
Institute of Automation, Chinese Academy of Science (CASIA)

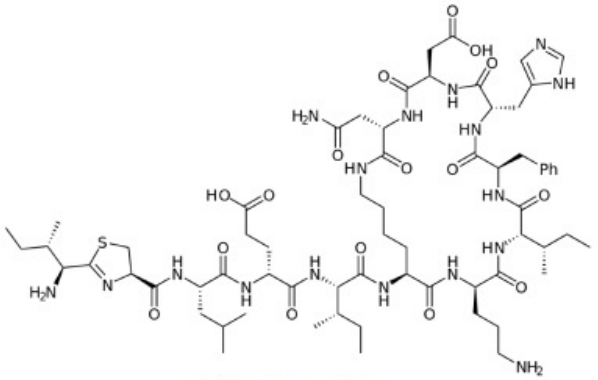
Outline

1. Course Review
2. DL for Computer Vision
3. DL Platforms

Graph-structured Data

Social networks
Citation networks
Communication networks
Multi-agent systems





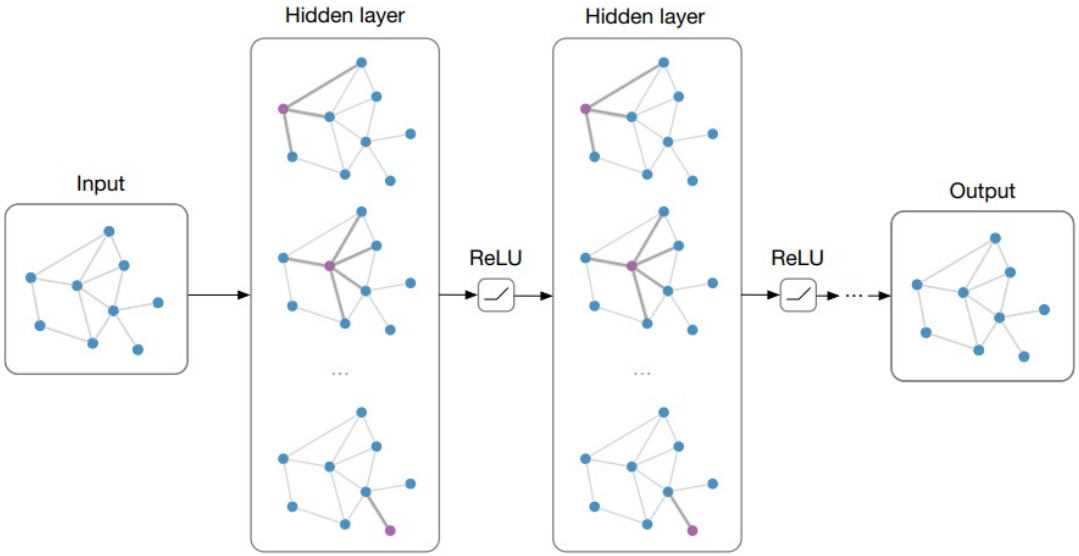
Molecules

Maps

A lot of real-world data does not live on grids

Standard DL architectures like CNNs and RNNs do **NOT** work here!

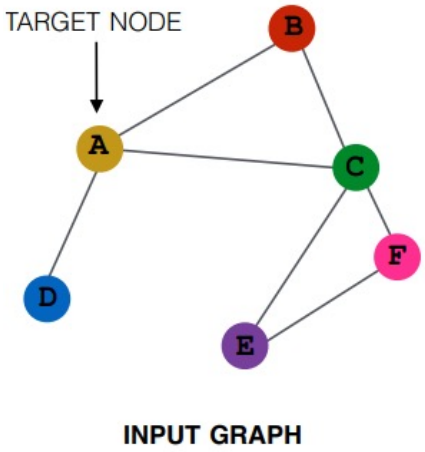
Graph Neural Networks



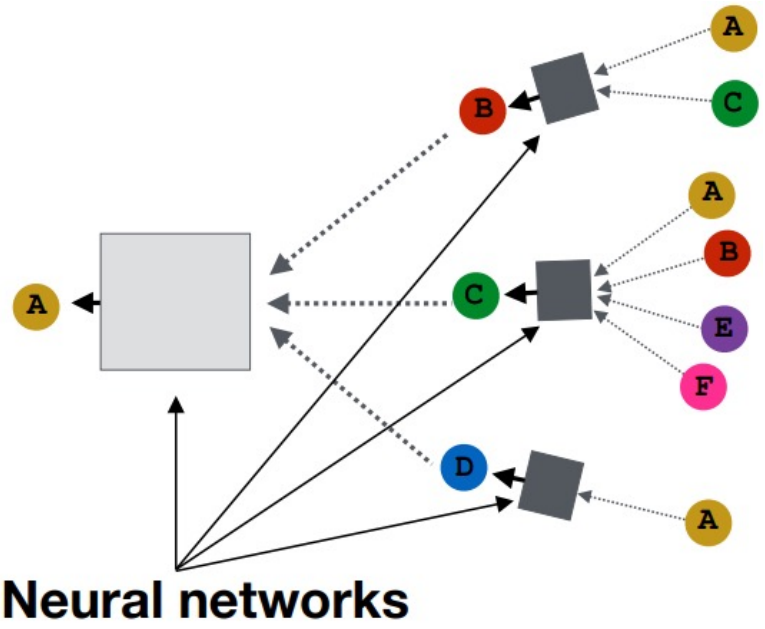
- Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$
- Adjacency matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix: $\mathbf{X} \in \mathbb{R}^{N \times F}$

Graph Neural Networks

- Nodes aggregate information from their neighbors using neural networks



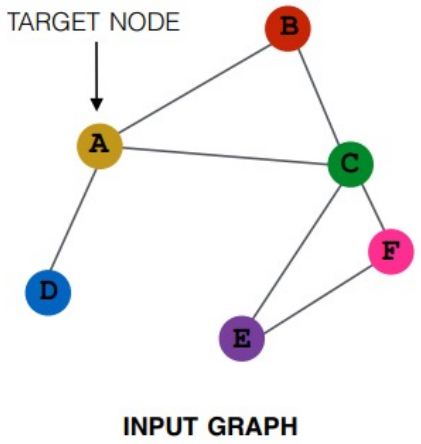
INPUT GRAPH

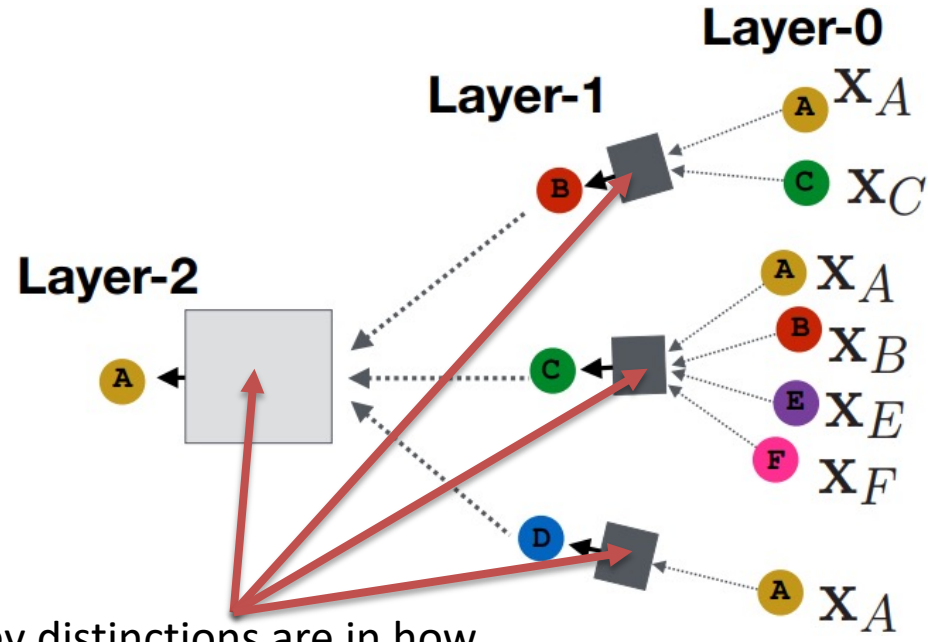


Neural networks

Graph Neural Networks

- Model can be of arbitrary depth
- Nodes have embeddings at each layer
- Layer-0 embedding of node u is its input feature, $i.e. x_u$.



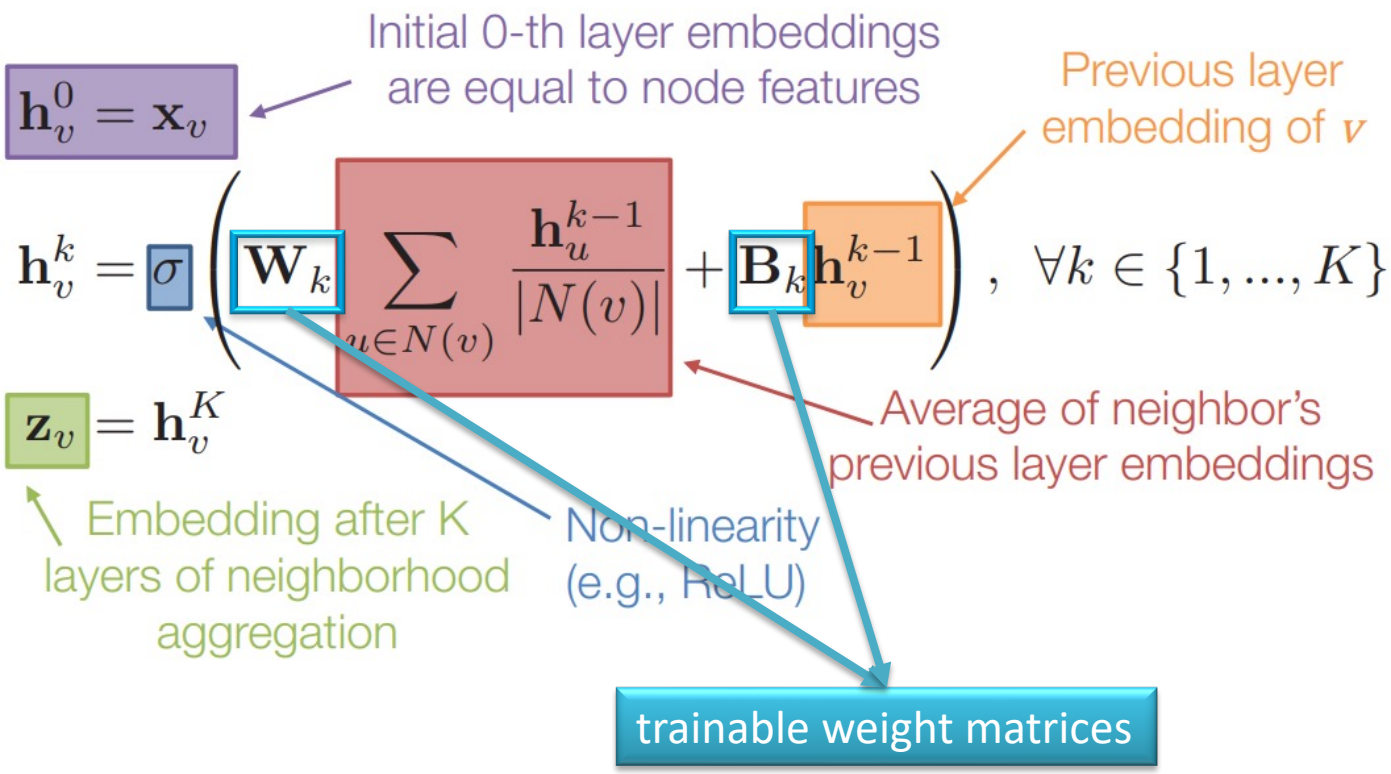


Key distinctions are in how different approaches aggregate information across the layers

Basic approach: Average information from neighbors and apply a neural network !

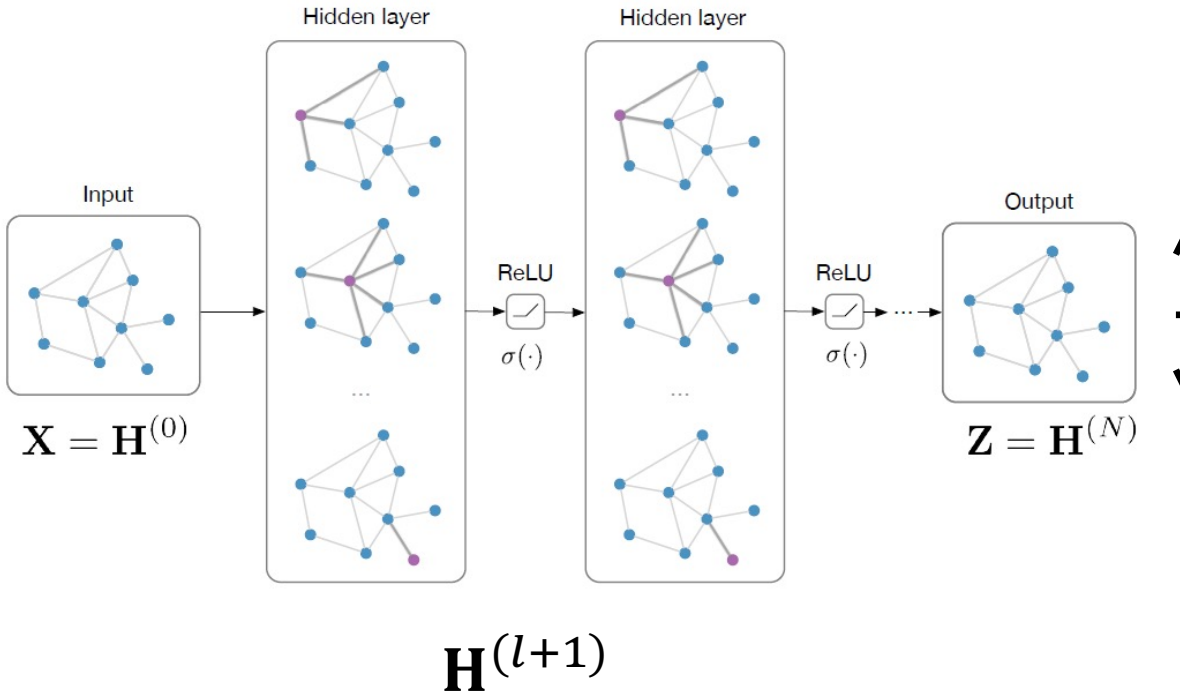
Graph Neural Networks

Basic approach: Average information from neighbors and apply a neural network !



Classification and link prediction

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\widehat{\mathbf{A}}$



Node classification:

$$softmax(\mathbf{Z_n})$$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$$softmax(\sum_n \mathbf{Z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

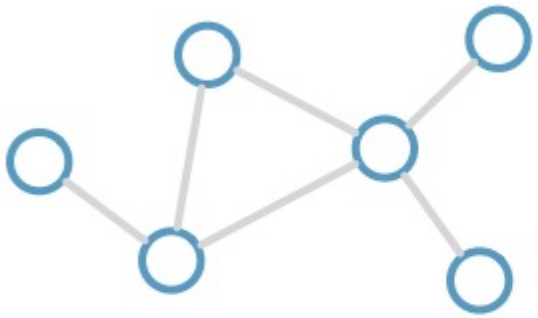
$$p(A_{ij}) = \sigma(\mathbf{z}_i^T \mathbf{z_j})$$

Kipf & Welling (NIPS BDL 2016)

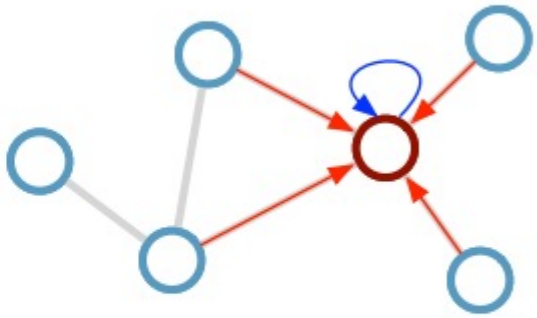
"Graph Auto-Encoders"

Graph Convolutional Networks

Consider this undirected graph:



Calculate update for node in red:



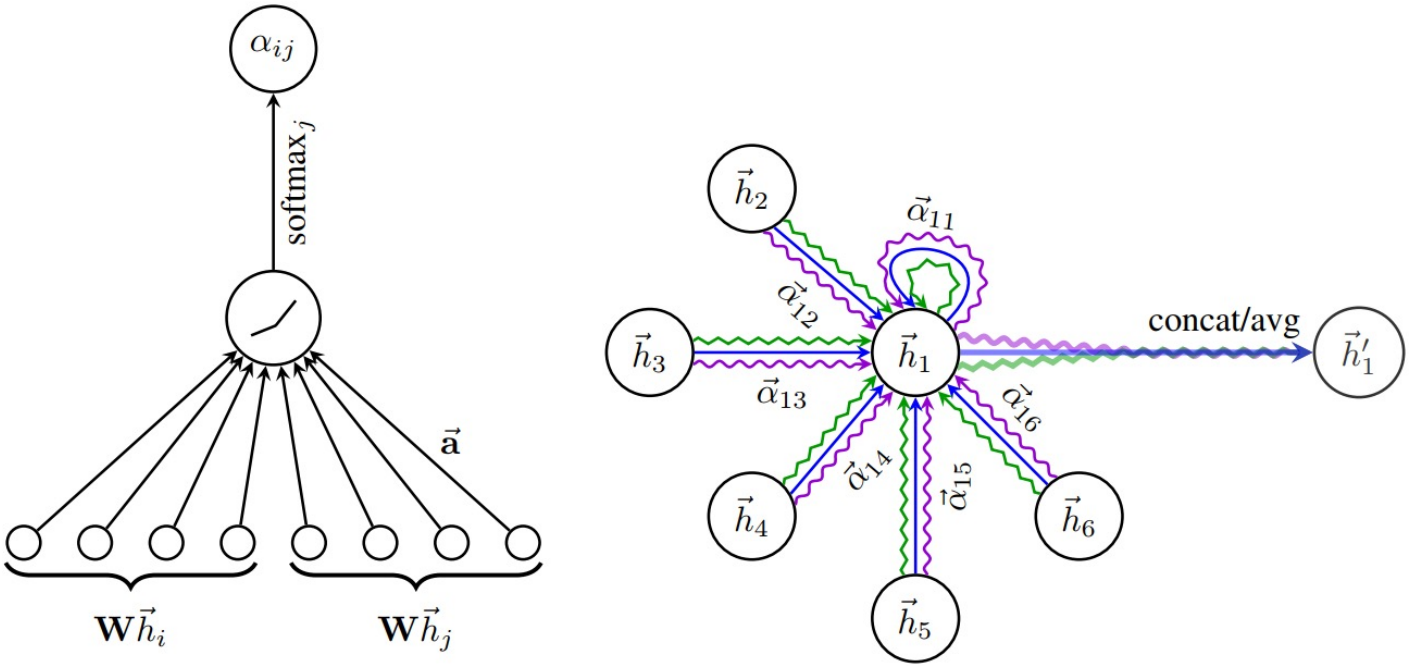
Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

GNN with Attention



$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

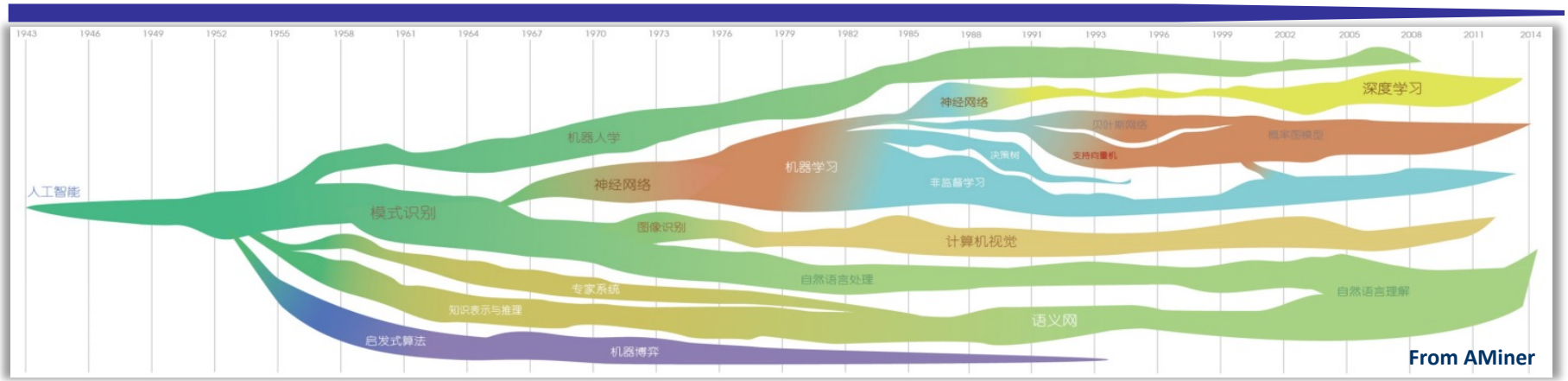
Average: $\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$ Concat: $\vec{h}'_i = \Big\|_{k=1}^{K}\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$

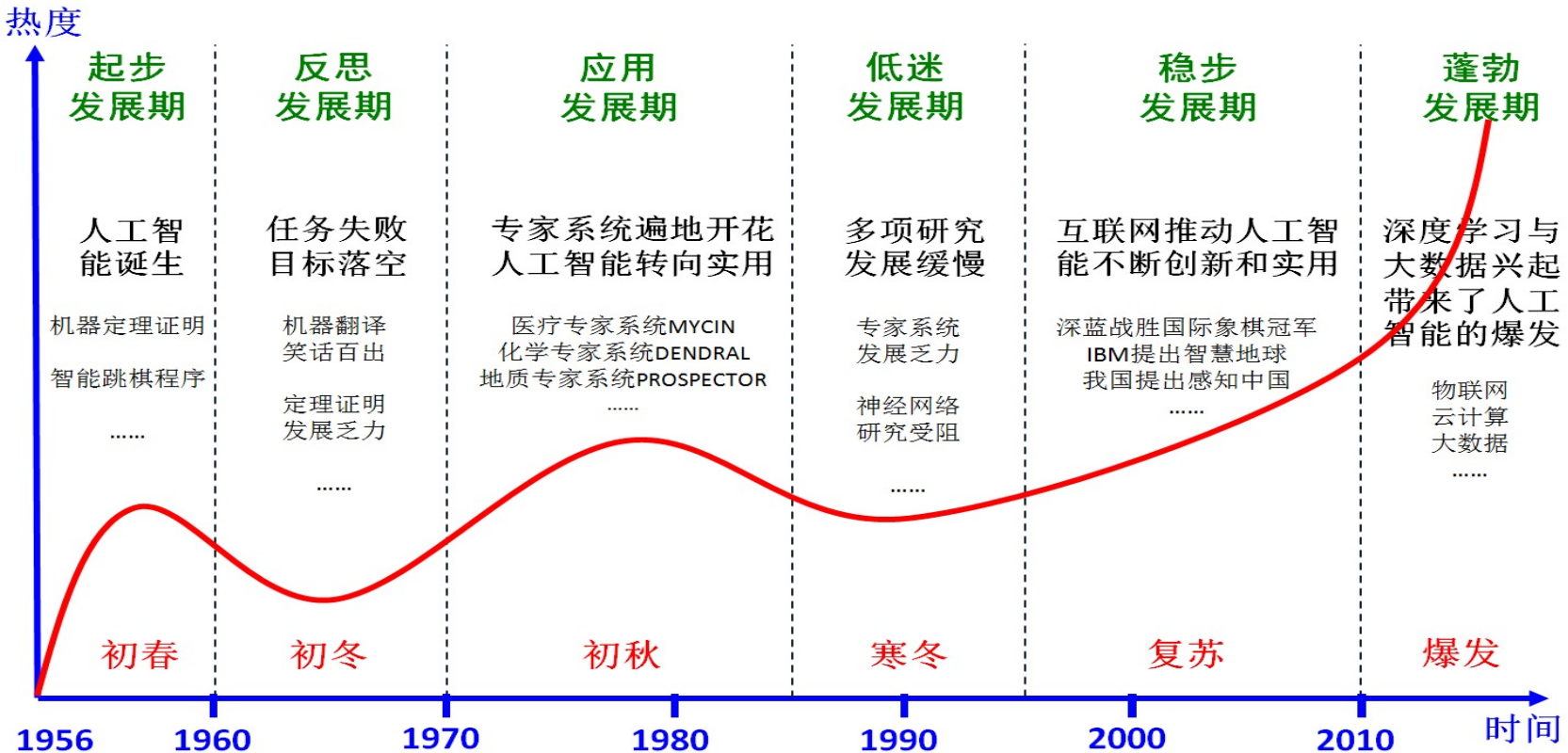
Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličkovićet al. (ICLR 2018)

Outline

1. Course Review
2. DL for Computer Vision
3. DL Platforms

人工智能的发展历程


1943
1946
1949
1952
1955
1958
1961
1964
1967
1970
1973
1976
1979
1982
1985
1988
1991
1993
1996
1999
2002
2005
2008
2011
2014
人工智能
机器人学
模式识别
神经网络
机器学习
图像识别
专家系统
知识表示与推理
启发式算法
机器博弈
神经网络
贝叶斯网络
概率图模型
决策树
支持向量机
非监督学习
深度学习
计算机视觉
自然语言处理
自然语言理解
语义网
From AMiner


热度
起步
发展期
人工智
能诞生
机器定理证明
智能跳棋程序
.....
反思
发展期
任务失败
目标落空
机器翻译
笑话百出
定理证明
发展乏力
.....
应用
发展期
专家系统遍地开花
人工智能转向实用
医疗专家系统MYCIN
化学专家系统DENDRAL
地质专家系统PROSPECTOR
.....
低迷
发展期
多项研究
发展缓慢
专家系统
发展乏力
神经网络
研究受阻
.....
稳步
发展期
互联网推动人工智
能不断创新和实用
深蓝战胜国际象棋冠军
IBM提出智慧地球
我国提出感知中国
.....
蓬勃
发展期
深度学习与
大数据兴起
带来了人工
智能的爆发
物联网
云计算
大数据
.....
初春
初冬
初秋
寒冬
复苏
爆发
1956
1960
1970
1980
1990
2000
2010
时间

Deep Neural Networks (DNN)

CNN: convolutional neural networks; RNN: recurrent neural networks; LSTM: long short term memory

- 1962 – neurobiological inspiration through simple/complex cell, *Hubel and Wiesel*
- 1970 – efficient error backpropagation, *Linnainmaa*
- 1979 – deep neocognitron, convolution, *Fukushima*
- 1987 – autoencoder, *Ballard*
- 1989 – backpropagation for CNN, *Lecun*
- 1991 – fundamental deep learning problem, *Hochreiter*
- 1991 – deep RNN, *Schmidhuber*
- 1997 – supervised LSTM-RNN, *Schmidhuber*

Deep Learning in Neural Networks: An Overview. Jurgen Schmidhuber, Technical Report 2014

Deep Neural Networks (DNN)

- **Two major drawbacks:**
  1. **large number of parameters →*high computational cost***
  2. **small training sets →*over-fitting problem***
- **Most researchers prefer:**
  1. ***hand-crafted features*, e.g., SIFT and LBP**
  2. ***"shallow" classification models*, e.g., SVM and Boosting**

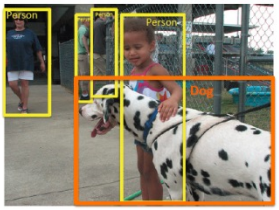
Two Recent Developments

Big Data

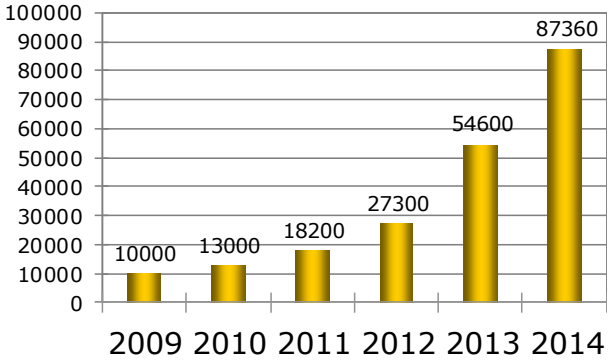
IMAGENET Large Scale Visual Recognition Challenge (ILSVRC) 2010-2013

~~20 object classes~~ ~~22,591 images~~

200 object classes	456,191 images	DET NEW
1000 object classes	1,431,167 images	CLS-LOC



http://image-net.org/challenges/LSVRC/{2010,2011,2012,2013}



Video surveillance data size (PB)

Cheap Computation

Features	Tesla K40	Tesla K20X	Tesla K20
Number and Type of GPU	1 Kepler GK110B	1 Kepler GK110	
Peak double precision floating point performance	1.43 Tflops	1.31 Tflops	1.17 Tflops
Peak single precision floating point performance	4.29 Tflops	3.95 Tflops	3.52 Tflops
Memory bandwidth (ECC off)	288 GB/sec	250 GB/sec	208 GB/sec
			5 GB
			2496

DNN can thus be fitted efficiently

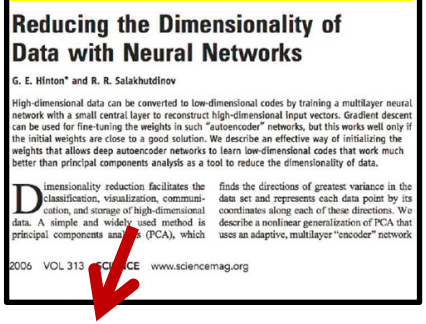
Pioneers (2006)

They lead deep learning through three major stages!

Rapid Development of AI Technology

Deep learning has achieved great success in computer vision, speech recognition and natural language understanding

Breakthrough



Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

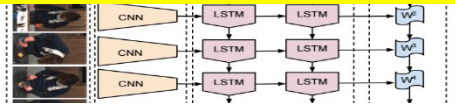
High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer "encoder" network

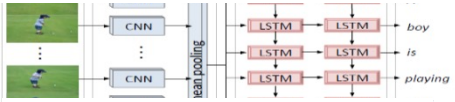
2006 VOL 313 SCIENCE www.sciencemag.org

ImageNet: 74% vs. 85%

Recurrent Neural Networks



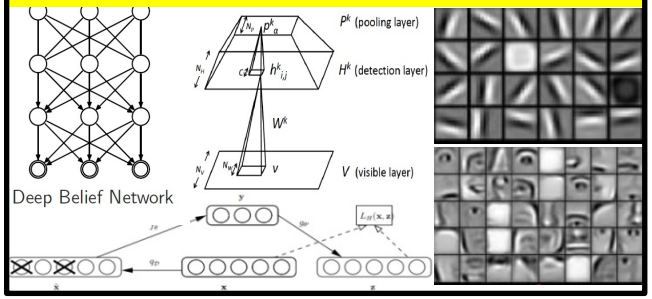
Activity recognition, CVPR2015



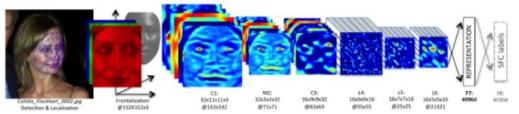
Video caption, CVPR2015

2006 2012 2015

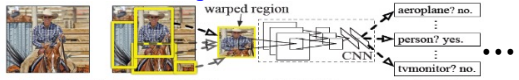
Representation Learning



Convolutional Neural Networks

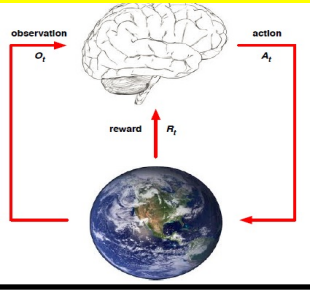


DeepFace, CVPR2014



RCNN for detection, CVPR2014

Reinforcement Learning



Advances in Computer Vision

Computer vision has made significant breakthrough on various tasks

Traditional shallow model
72% 2010
74% 2011

85% 2012
89% 2013
93% 2014
95% 2015
97% 2016 (better than human)

Object recognition/detection/segmentation

401 scene categories
83% 2015
91% 2016
94% 2017

Scene analysis and understanding

85.9% 2013
88.0% 2014
91.4% 2015
94.3% 2016
96.2% 2017

Action analysis and understanding

The man at bat readies to swing at the pitch while the umpire looks on.

CIDEr-D	date
1.123	2016-11-16
1.049	2016-10-25
1.042	2016-11-11

Language Description for image/video

Joint vision and language understanding

Big Visual Data Analysis

Large-Scale Dataset

Large-Scale Visual Computing

Platforms and Applications

Object Detection/ Segmentation

Object Recognition/ Retrieval

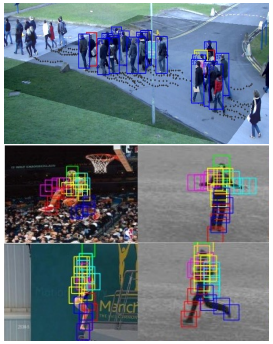
Motion/ Behavior Analysis

Scene Understanding

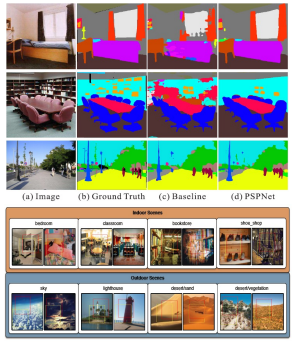
Vision by Language

Face/pedestrian detection/ segmentation

Face/gait recognition/ retrieval



Object tracking action recognition



Scene classification/ parsing

Visual captioning/ question answering

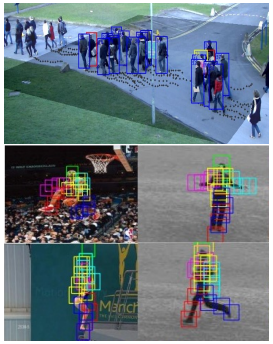
Big Visual Data Analysis

Large-Scale Dataset → **Large-Scale Visual Computing** → **Platforms and Applications**

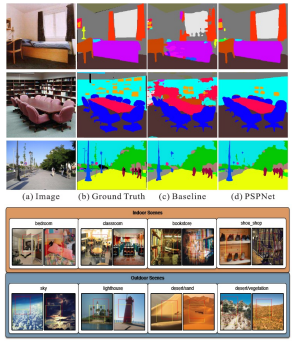
Object Detection/ Segmentation | **Object Recognition/ Retrieval** | **Motion/ Behavior Analysis** | **Scene Understanding** | **Vision by Language**

Face/pedestrian detection/ segmentation

Face/gait recognition/ retrieval



Object tracking action recognition

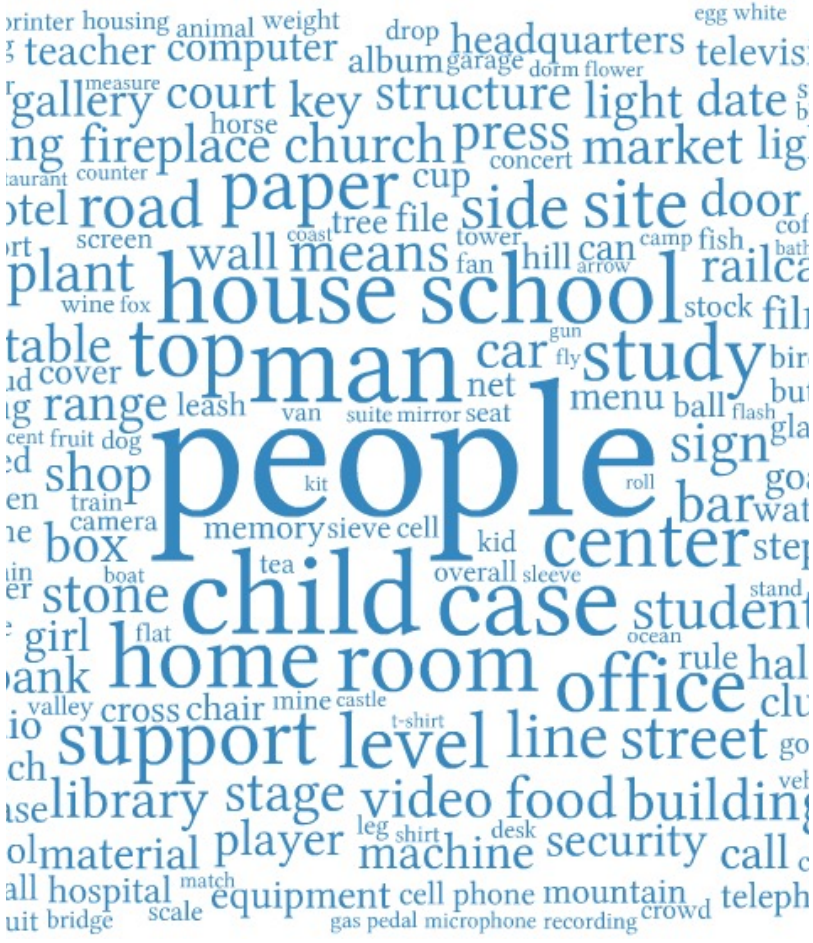


Scene classification/ parsing

Visual captioning/ question answering

ImageNet

IMAGENET



14,197,122 images, 21,841 synsets indexed

Object detection (DET)[top]

Task 1a: Object detection with provided training data

Ordered by number of categories won

Team name	Entry description	Number of object categories won	mean AP
CUImage	Ensemble of 6 models using provided data	109	0.662751
Hikvision	Ensemble A of 3 RPN and 6 FRCN models, mAP is 67 on val2	30	0.652704
Hikvision	Ensemble B of 3 RPN and 5 FRCN models, mean AP is 66.9, median AP is 69.3 on val2	18	0.652003
NUIST	submission_1	15	0.608752
NUIST	submission_2	9	0.607124

Object localization (LOC)[top]

Task 2a: Classification+localization with provided training data

Ordered by localization error

Team name	Entry description	Localization error	Classification error
Trimps-Soushen	Ensemble 3	0.077087	0.02991
Trimps-Soushen	Ensemble 4	0.077429	0.02991
Trimps-Soushen	Ensemble 2	0.077668	0.02991
Trimps-Soushen	Ensemble 1	0.079068	0.03144
Hikvision	Ensemble of 3 Faster R-CNN models for localization	0.087377	0.03711

Microsoft COCO

Image Classification/Object Detection/Semantic Segmentation/Image Captioning

- Object segmentation
- Recognition in Context
- Multiple objects per image
- More than 300,000 images
- More than 2 Million instances
- 80 object categories
- 5 captions per image
- Keypoints on 100,000 people

Challenges: Detections | Captions | Keypoints

	M1	M2	M3	M4	M5	date
Human[5]	0.638	0.675	4.836	3.428	0.352	2015-03-23
Google[4]	0.273	0.317	4.107	2.742	0.233	2015-05-29
MSR[11]	0.268	0.322	4.137	2.662	0.234	2015-04-08
Montreal/Toronto[10]	0.262	0.272	3.932	2.832	0.197	2015-05-14
MSR Captivator[12]	0.25	0.301	4.149	2.565	0.233	2015-05-28

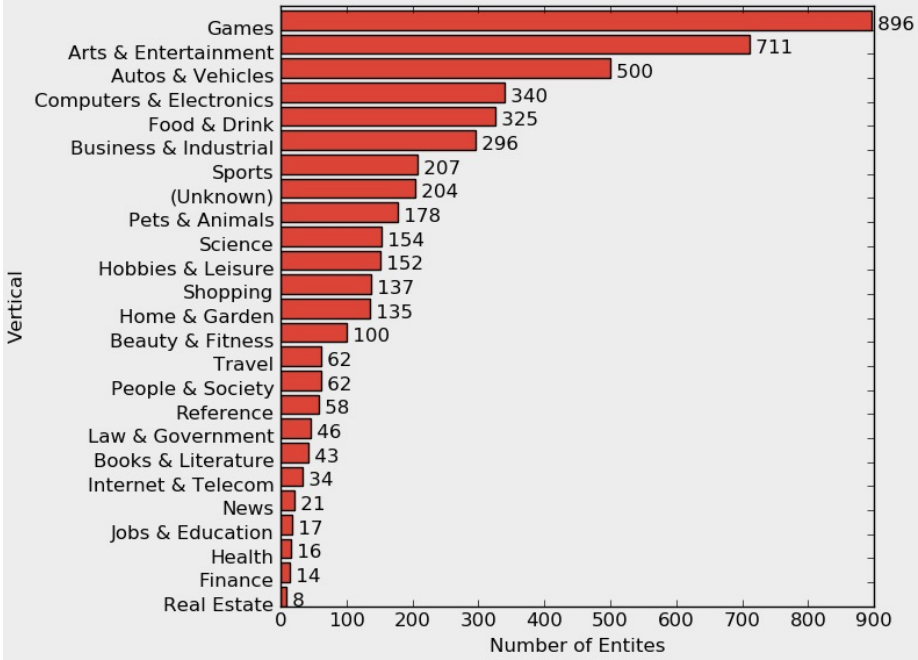
Dataset examples

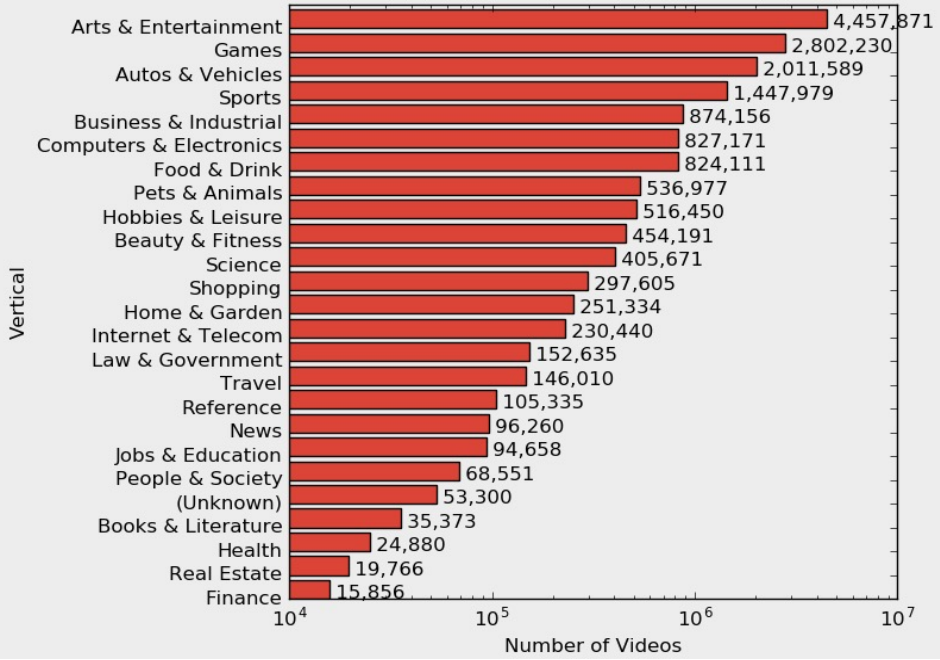
Google YouTube-8M

7 Million	450,000	3.2 Billion	4716	3.4
Video URLs	Hours of Video	Audio/Visual Features	Classes	Avg. Labels / Video

The videos are sampled uniformly to preserve the diverse distribution of popular content on YouTube, subject to a few constraints selected to ensure dataset quality and stability:

- Each video must be public and have at least 1000 views
- Each video must be between 120 and 500 seconds long
- Each video must be associated with at least one entity from our target vocabulary
- Adult & sensitive content is removed (as determined by automated classifiers)



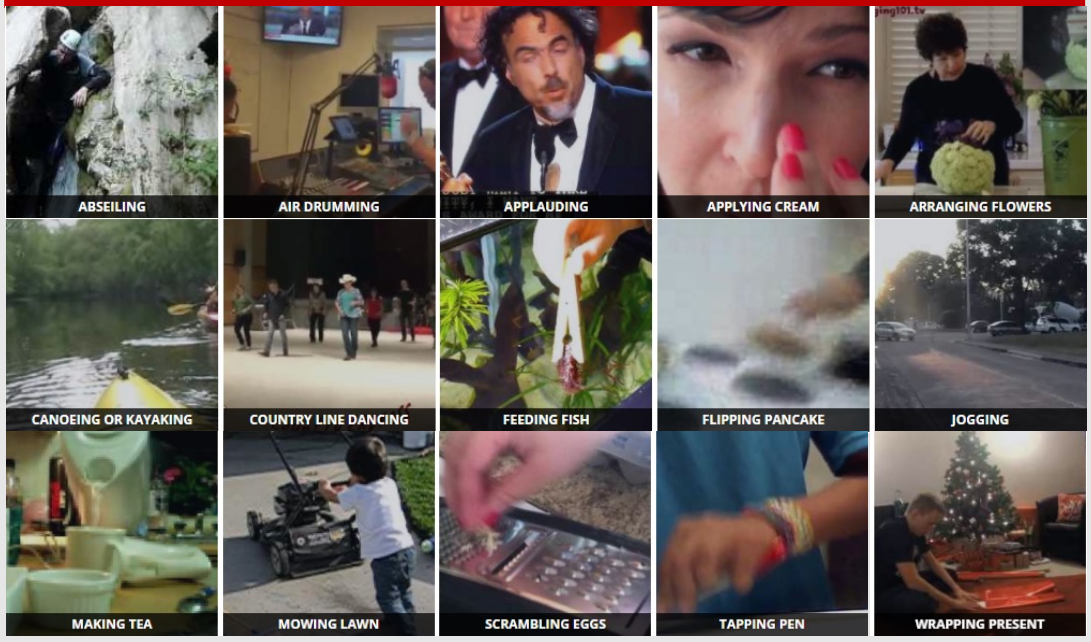


Kinetics

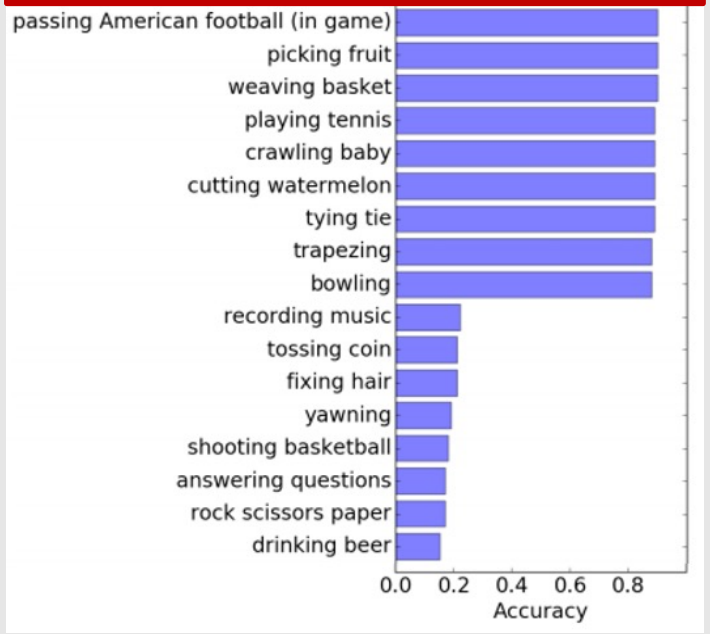
300,000	400	400~1150	10s
Video URLs	Classes	Clips / Class	Avg. Length / Clip

- The clips are sourced from YouTube videos
- The actions cover a broad range of classes including human-object interactions as well as human-human interactions
- There can be considerable camera motion/shake, illumination variations, shadows, background clutter, etc.

Dataset examples



Partial simple/hard classes



Visual Question Answering (VQA)

Balanced Real Images/Balanced Binary Abstract Scenes/Abstract Scenes

- 265,016 images (COCO and abstract scenes)
- At least 3 questions (5.4 questions on average) per image
- 10 ground truth answers per question
- 3 plausible (but likely incorrect) answers per question
- Automatic evaluation metric
- Open-ended and multiple-choice tasks

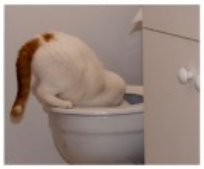
VQA Real Image Challenge (Open-Ended) 2017
Organized by: VQA Team

Rank	Participant Team	yes/no	number	other	overall
1	**HDU-USYD-UNCC**	86.65	51.13	61.75	70.92
2	**DeepSearch**	86.21	48.82	61.58	70.40
3	**Adelaide-Teney ACRV MSR**	86.60	48.64	61.15	70.34
4	**VQA-E**	85.74	48.18	60.12	69.44
5	**Tohoku CV**	85.54	49.00	58.99	68.91

Is the TV on?

yes  no 

How many pets are present?

2  1 

What time of day is it?

night  noon 

Does the man have a foot in the air?

yes  no 

What sign is this?

handicap  one way 

Is the computer a laptop or a desktop?

desktop  laptop 

Are any benches occupied?

no  yes 

What color are the wall tiles?

blue  brown 

Big Visual Data Analysis

Large-Scale Dataset

Large-Scale Visual Computing

Platforms and Applications

Object Detection/ Segmentation

Object Recognition/ Retrieval

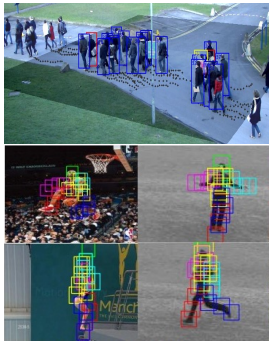
Motion/ Behavior Analysis

Scene Understanding

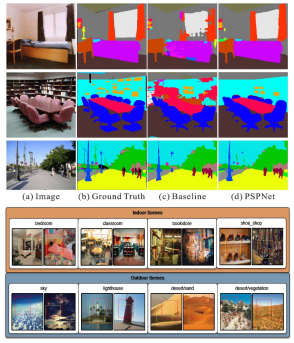
Vision by Language

Face/pedestrian detection/ segmentation

Face/gait recognition/ retrieval



Object tracking action recognition

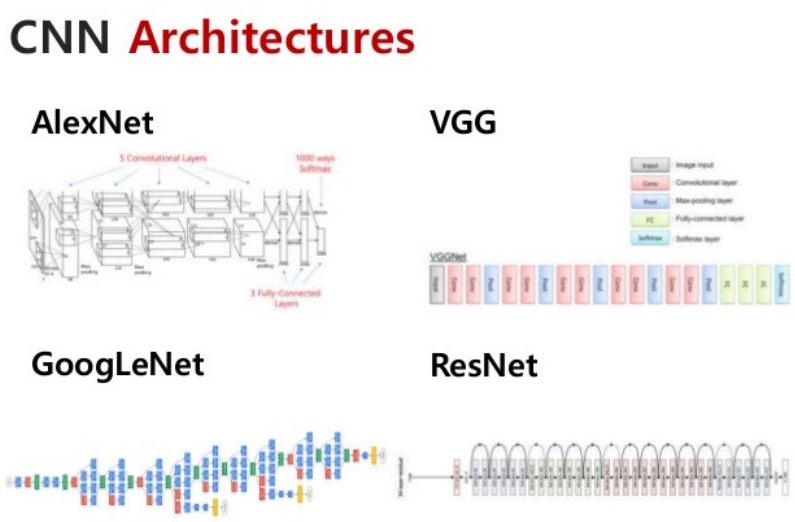


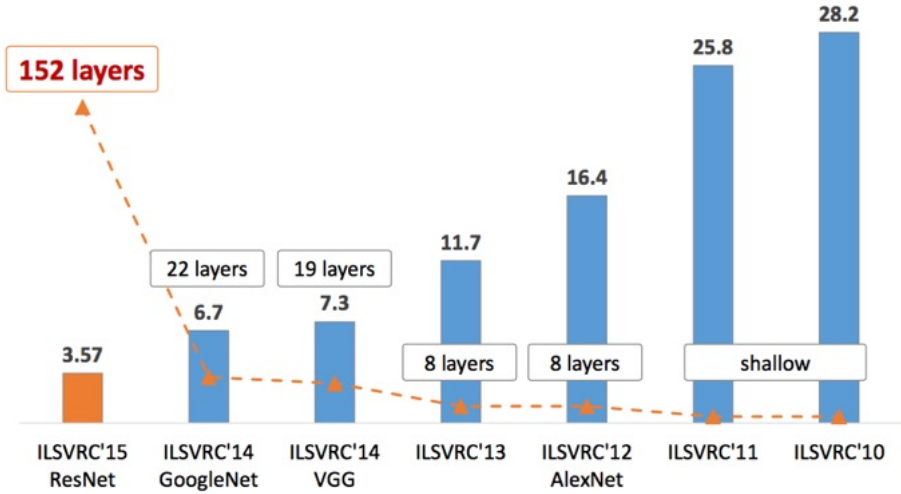
Scene classification/ parsing

Visual captioning/ question answering

Image Classification

Deep neural networks have surpassed human on the accuracy of image classification in ImageNet



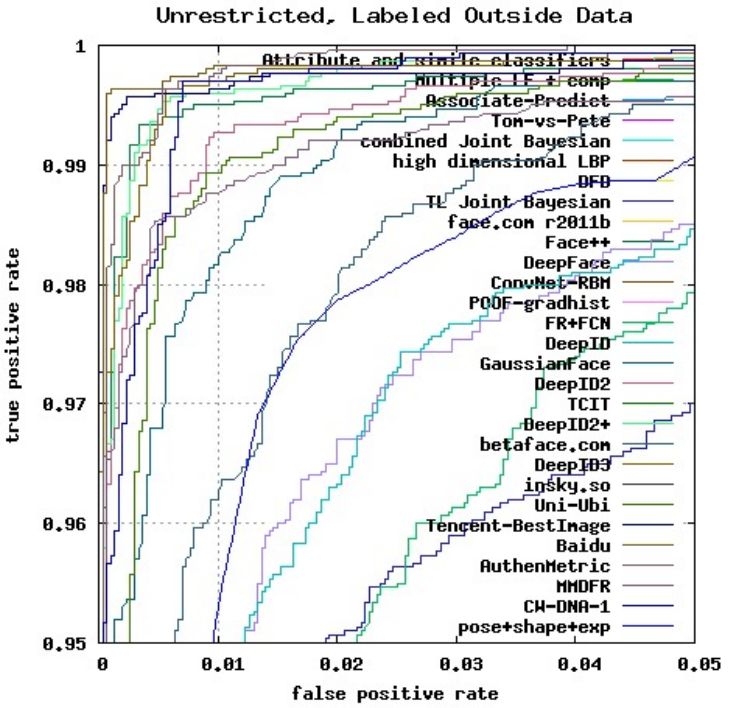


Year	Winner	Classification Error
2012	AlexNet	0.153
2013	Clarifai	0.11197
2014	GoogLeNet	0.06656
2015	MSRA	0.03567
2016	CUImage	0.02991

Face Recognition

- **LFW Dataset (Labeled Faces in the Wild)**

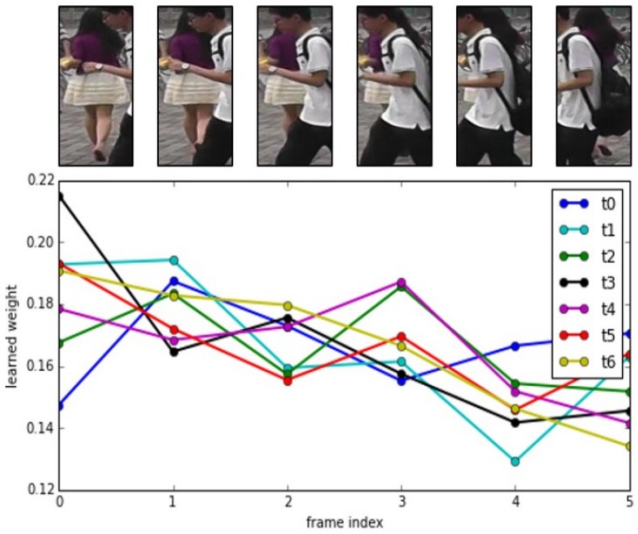
Method/Team	Accuracy and Standard Error
DeepID	0.9745 ± 0.0026
DeepID2	0.9915 ± 0.0013
DeepID2+	0.9947 ± 0.0012
Face++	0.9950 ± 0.0036
DeepID3	0.9953 ± 0.0010
FaceNet	0.9963 ± 0.0009
Baidu	0.9977 ± 0.0006
YouTu	0.9980 ± 0.0023



Spatial-temporal Recurrent Networks for Video-based Person Re-identification

Task: **re-identify person in videos captured by different cameras**
Solution: exploit **temporal attention scheme** to selectively learn discriminate features from key frames

Selectively attend to key frames



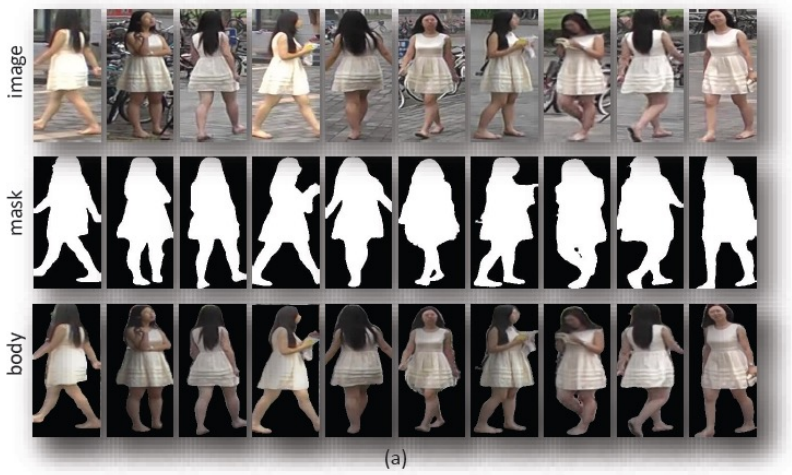
Retrieve persons with most similar appearance

Zhou et al., See the Forest for the Trees: Joint Spatial and Temporal Recurrent Neural Networks for Video-based Person Re-identification, CVPR, 2017.

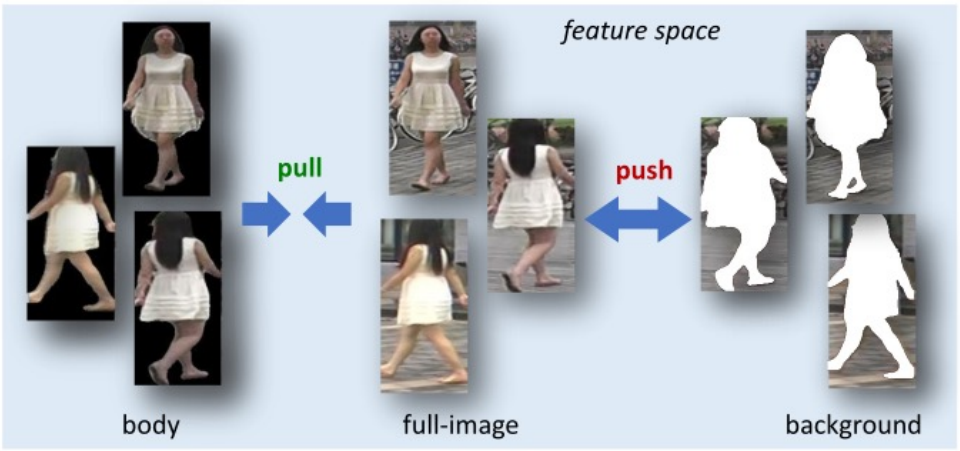
Mask-guided Contrastive Attention Model for Person Re-Identification

Task: **re-identify foreground persons captured by different cameras**
Solution: exploit **mask-guided contrastive attention model** to learn discriminative and robust features invariant to background clutters

Introduce masks as extra inputs



Region-level triplet loss restrains the features



Song et al., Mask-guided Contrastive Attention Model for Person Re-Identification, CVPR, 2018.

Summary

- Along with the success of image classification in ImageNet, large amounts of recognition tasks have achieved human-level performance in restricted condition, e.g., face recognition

- However, there still remains much work to be done in real unrestricted condition, particularly in big visual data

Big Visual Data Analysis

Large-Scale Dataset

Large-Scale Visual Computing

Platforms and Applications

Object Detection/ Segmentation

Object Recognition/ Retrieval

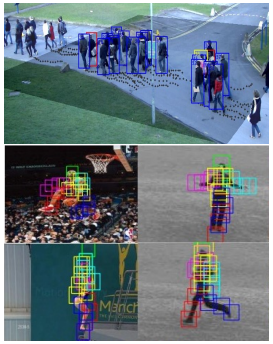
Motion/ Behavior Analysis

Scene Understanding

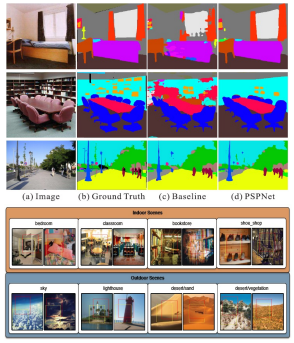
Vision by Language

Face/pedestrian detection/ segmentation

Face/gait recognition/ retrieval



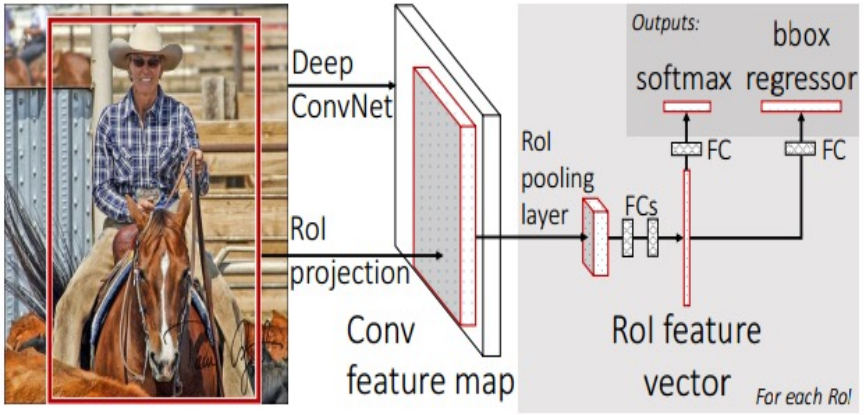
Object tracking action recognition



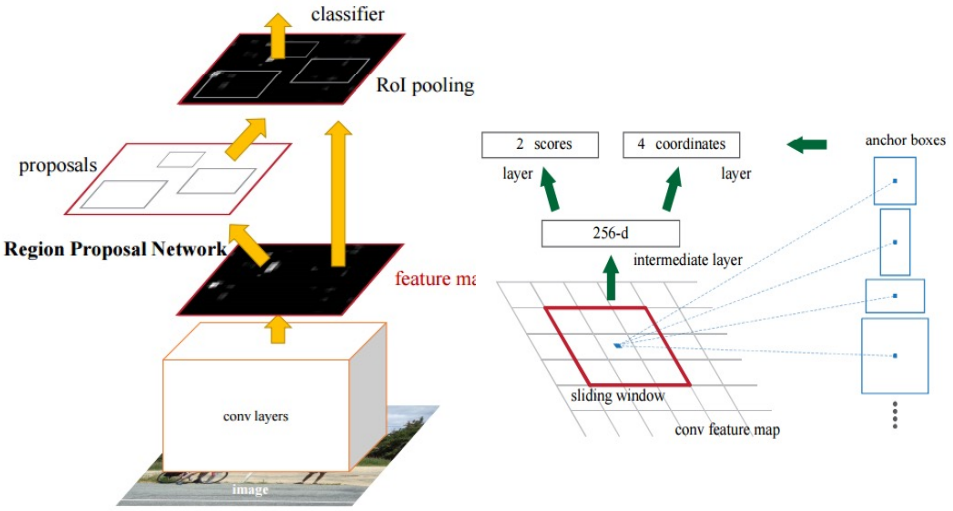
Scene classification/ parsing

Visual captioning/ question answering

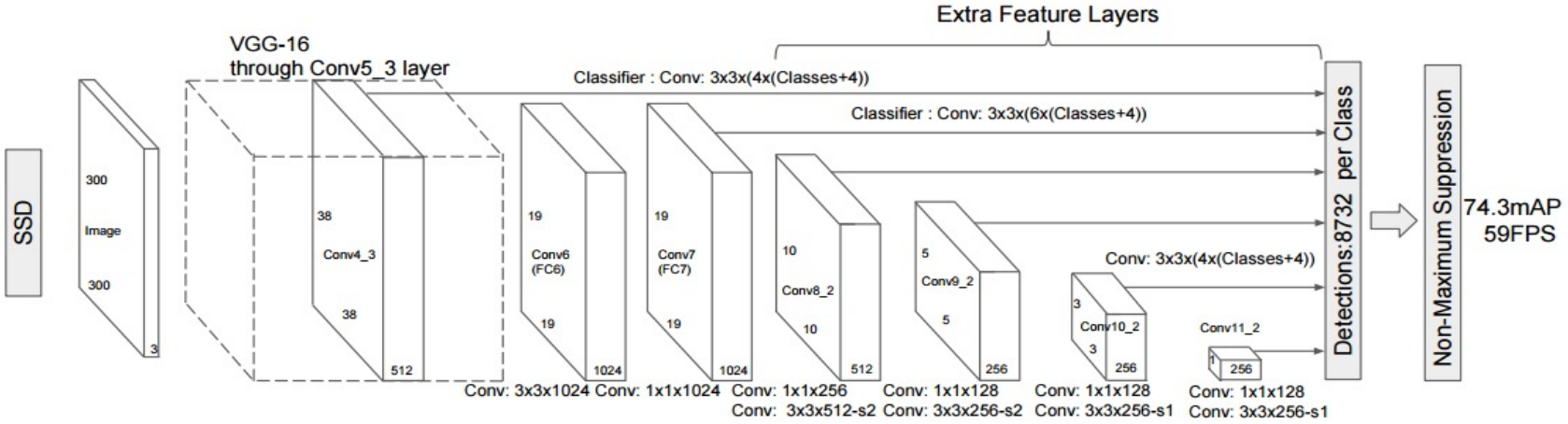
Object Detection



R. Girshick, Fast R-CNN: Fast Region-based Convolutional Networks for object detection, ICCV, 2015.

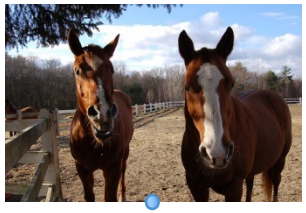


S. Ren et al., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, NIPS, 2015.



W. Liu et al., SSD: Single Shot MultiBox Detector, ECCV, 2016.

Weakly Supervised Object Detection



Positive Images

Negative Images

Input Images

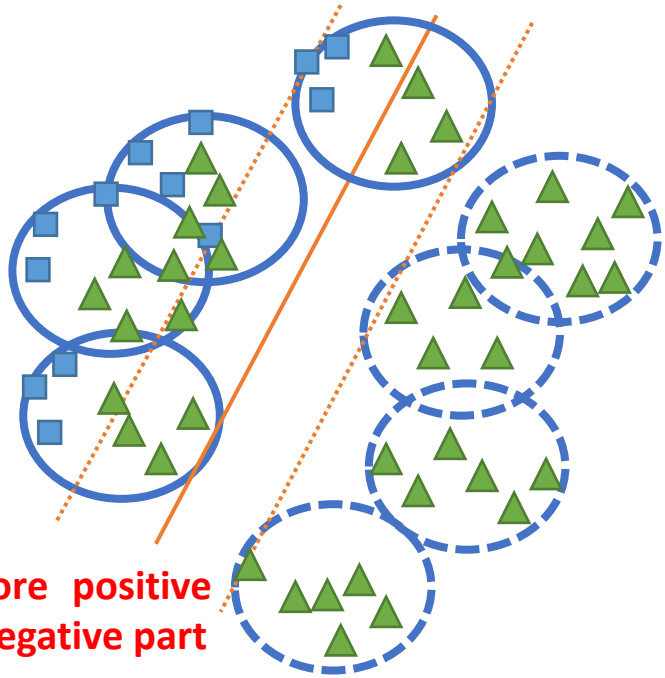
Region Proposals

Representations

MILinear Mining

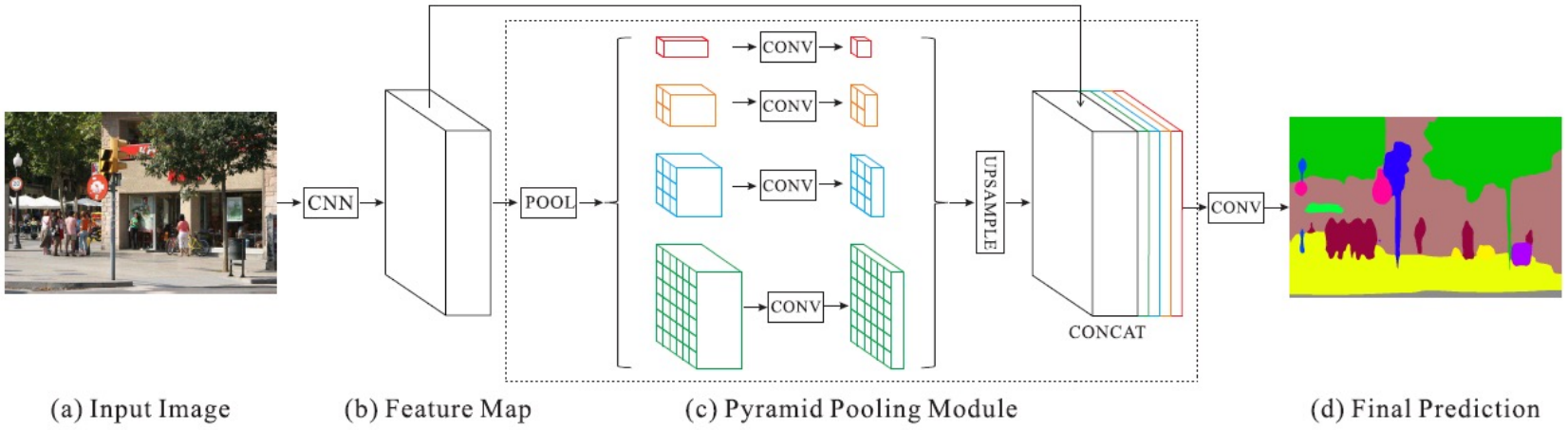
Novelty: To reduce ambiguity in positive images, we present a bag-splitting algorithm that iteratively generates new negative bags from positive ones

Conv. Nets

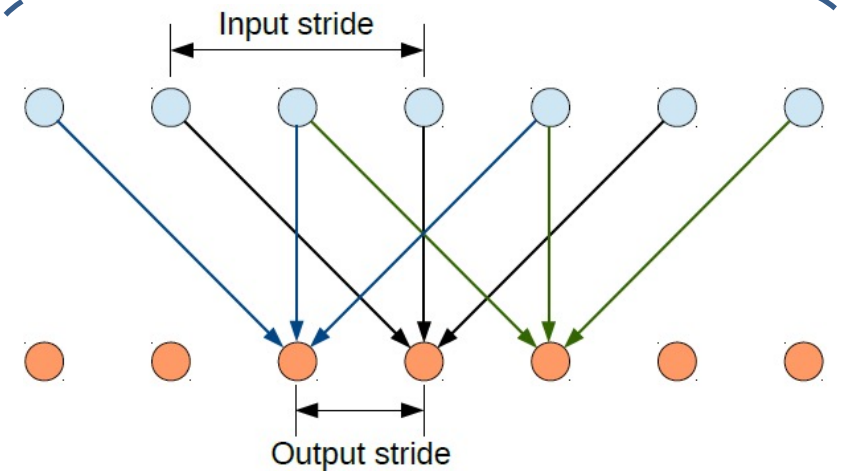


W. Ren et al., Weakly Supervised Large Scale Object Localization with Multiple Instance Learning and Bag Splitting, TPAMI, 2016.

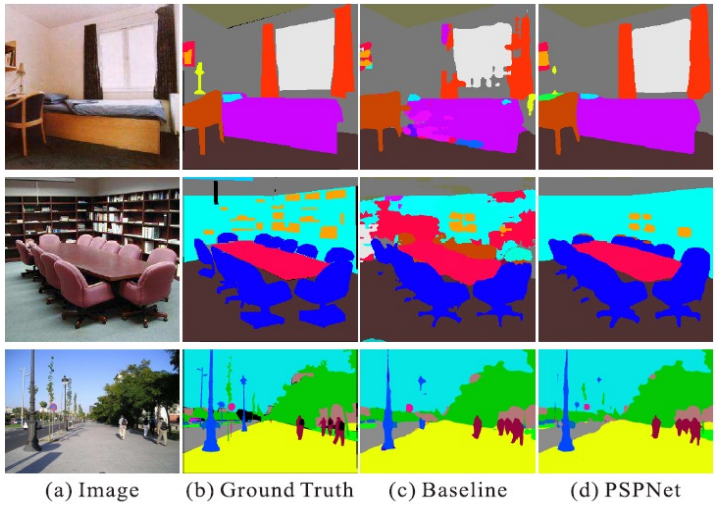
Image Labelling/Segmentation



Network structure



Dilated convolution



Inference visualization

H. Zhao et al., Pyramid Scene Parsing Network, arXiv:1612.01105.

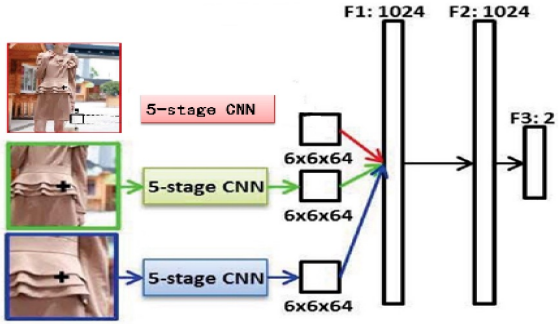
Multi-scale CNN for Object Segmentation

Task: segment foreground object from videos

Solution: propose "pixel-patch-image" multi-scale CNN to model context information

Pixel-level

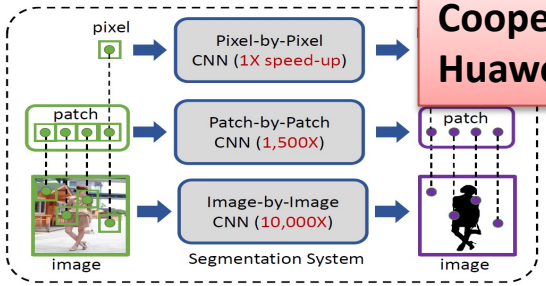
Internet Contest for Cloud & Mobile Computing on Human Segmentation, Champion



Team	Accuracy (%)
Second place	78.17
Third place	76.00
Forth place	75.95
Ours	**86.83**

Patch-level

Accuracy slightly drops, but the speed is 1000+ faster



Cooperate with Huawei, Samsung

Table 1. Performance comparison of different structures.

Methods	Resolution	ACC(%)	MRE	Speed
Pixel-by-Pixel [13]	100*100	**86.83**	- -	30s
Simple-seg-net	48*48	62.70	147.1	< **1ms**
Alex-seg-net	48*48	82.12	96.2	1ms
	112*112	80.20	490.8	2ms
VGG-seg-net	48*48	**83.57**	**94.8**	1ms

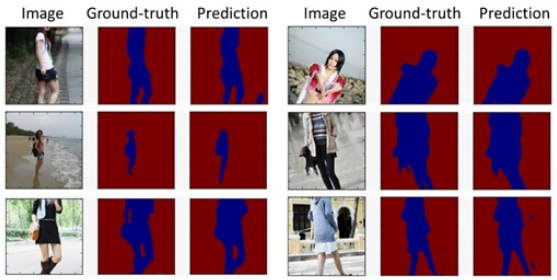
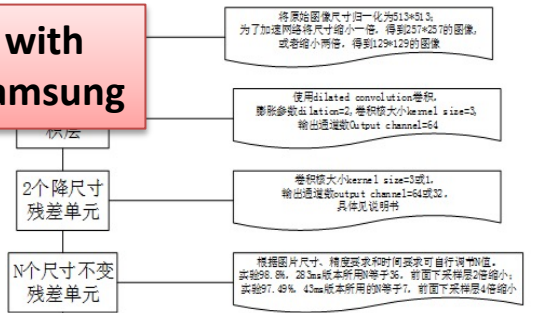


Image-level

Large improvements on both accuracy and speed



	ACC	Speed /CPU
FCN-32s	97.5%	~7000
50model	97.5%	45
500model	98.8%	283

Summary

- Object detection and segmentation have great potential applications, e.g., self-driving
- Supervised object detection and segmentation have made great progress by virtue of deep neural networks
- Weakly supervised object detection and segmentation still need to be further investigated

Big Visual Data Analysis

Large-Scale Dataset

Large-Scale Visual Computing

Platforms and Applications

Object Detection/ Segmentation

Object Recognition/ Retrieval

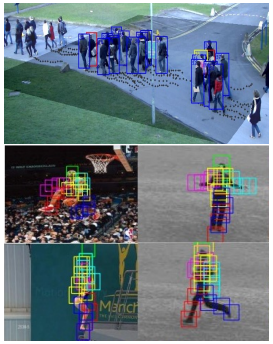
Motion/ Behavior Analysis

Scene Understanding

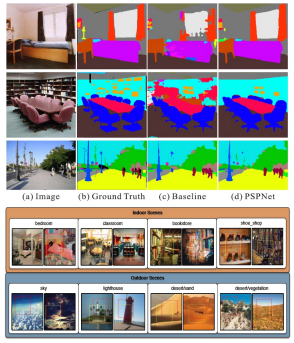
Vision by Language

Face/pedestrian detection/ segmentation

Face/gait recognition/ retrieval



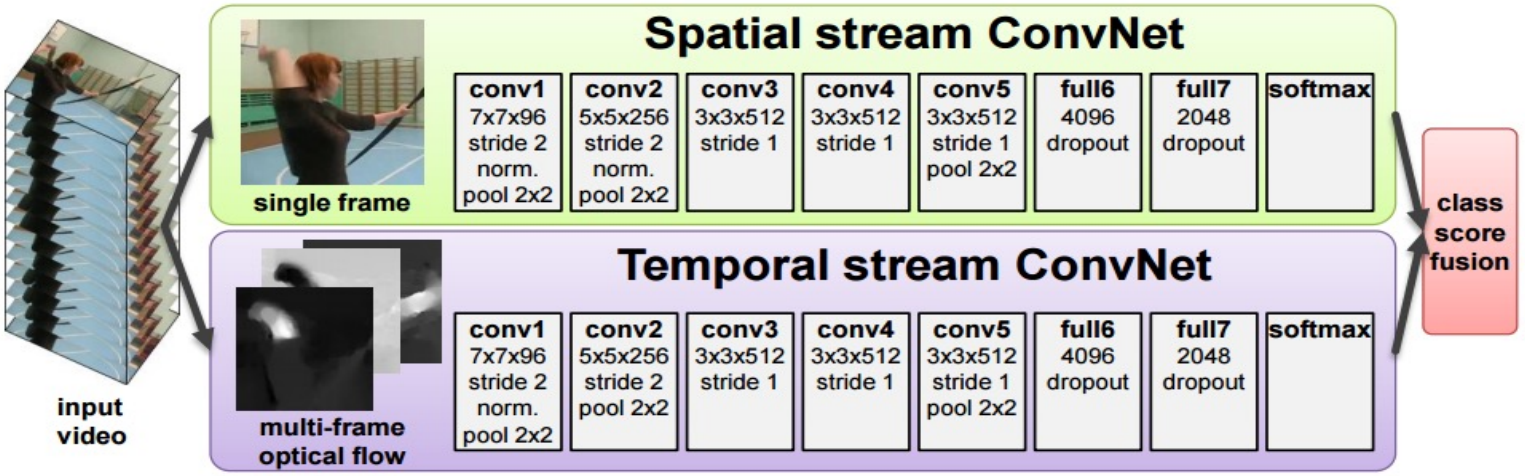
Object tracking action recognition



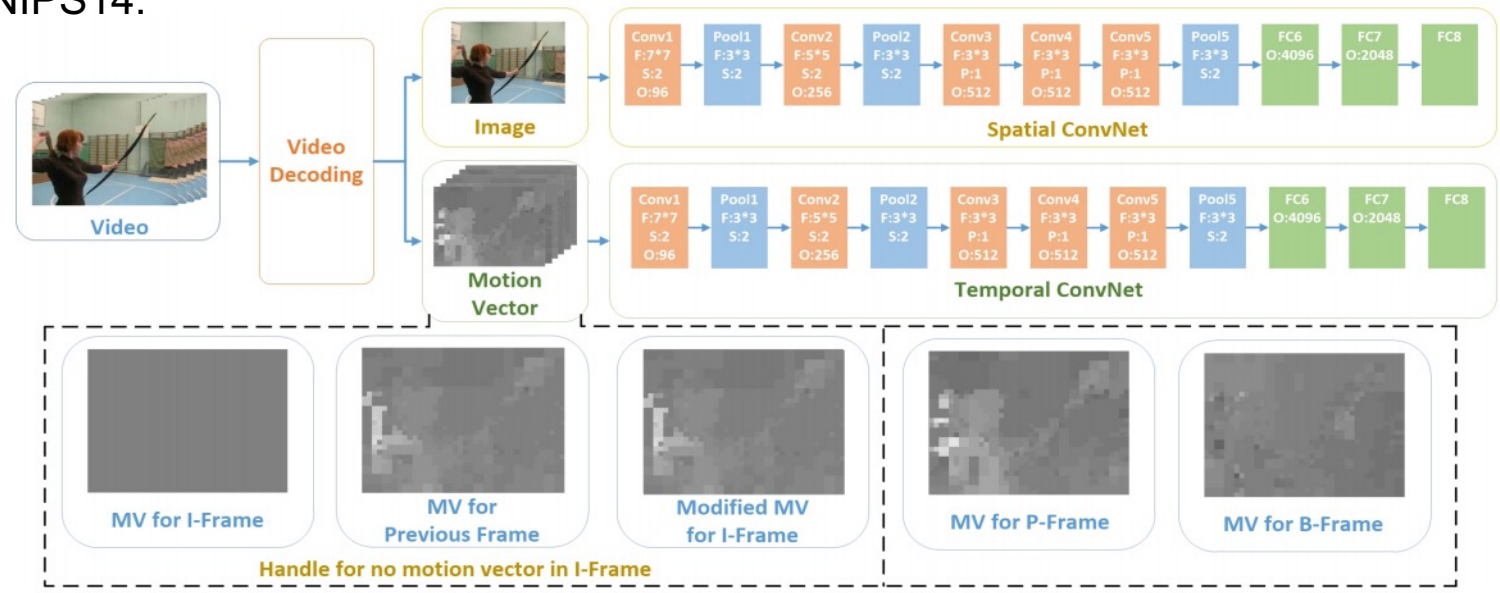
Scene classification/ parsing

Visual captioning/ question answering

Two-Stream Action Recognition



Simonyan and Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. NIPS14.



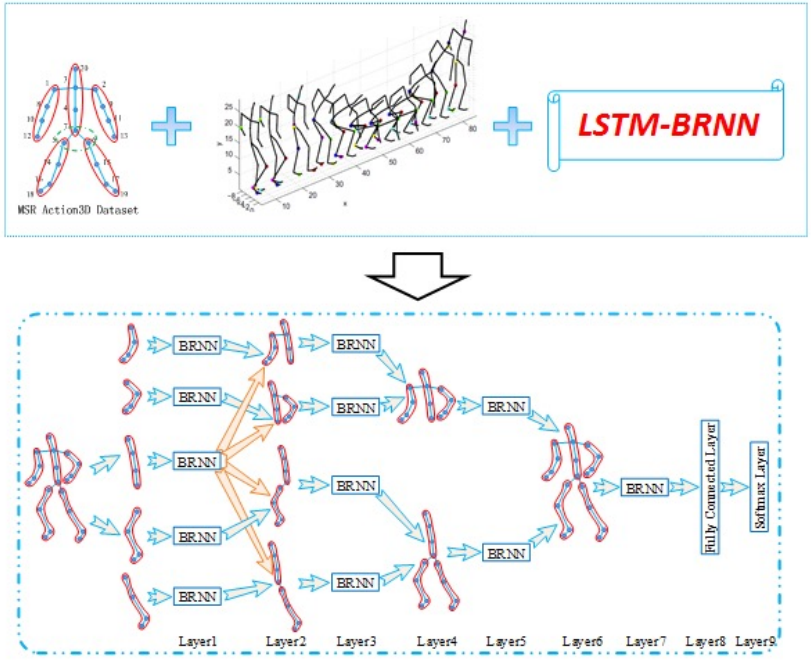
Zhang et al. Real-time Action Recognition with Enhanced Motion Vector CNNs, CVPR,2016.

Hierarchical/Two-stream Recurrent Networks for Skeleton based Action Recognition

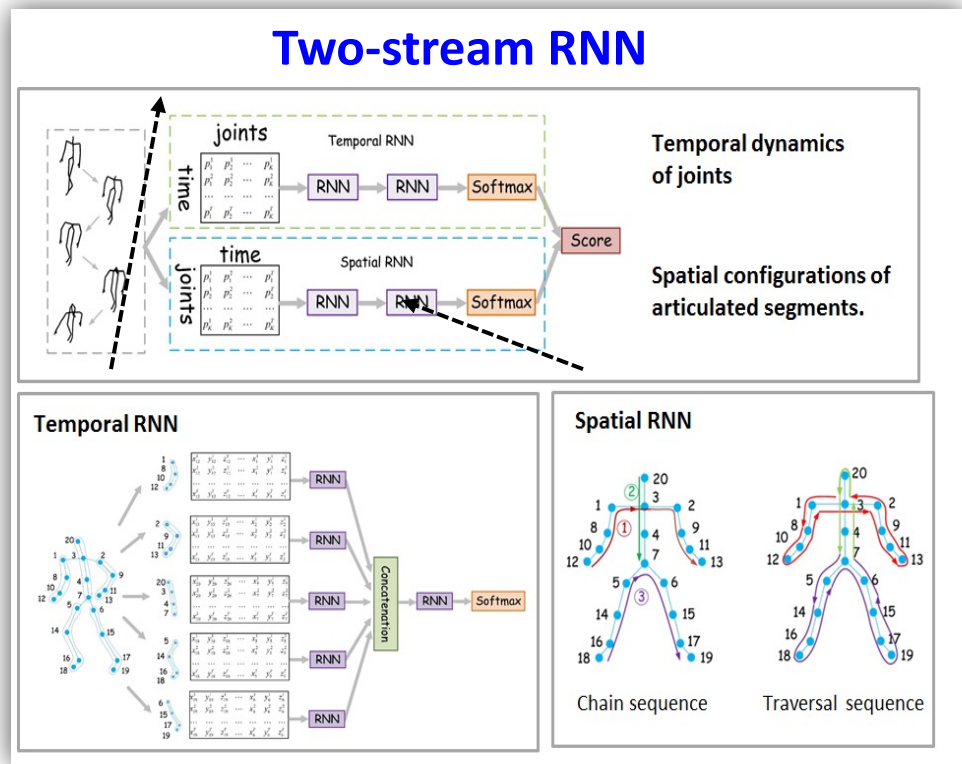
Task: recognize human action patterns in videos

Solution: propose hierarchical/two-stream recurrent networks to model both spatial static and temporal dynamical properties in human skeleton sequences

Hierarchical RNN



Two-stream RNN



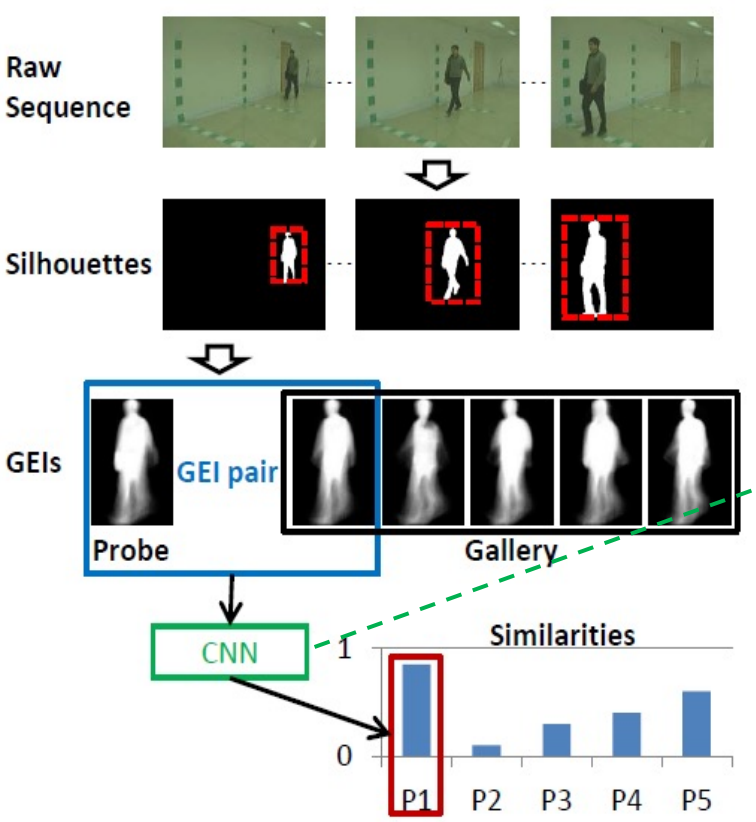
Wang et al., Modeling Temporal Dynamics and Spatial Configurations of Actions Using Two-Stream Recurrent Neural Networks, CVPR, 2017.

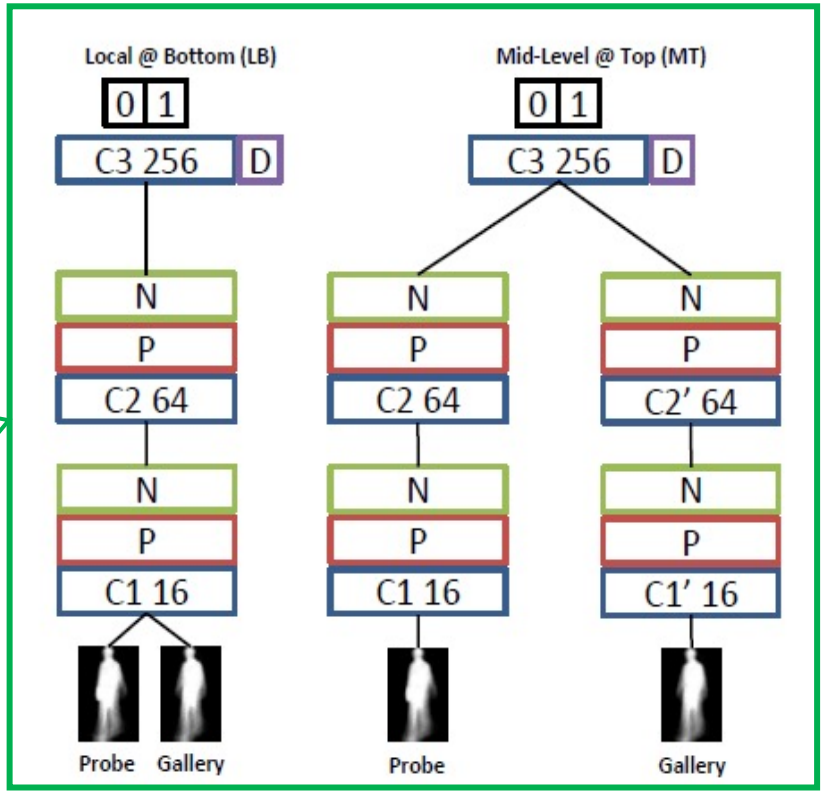
Du et al., Representation Learning of Temporal Dynamics for Skeleton-Based Action Recognition, TIP, 2016.

Du et al., Hierarchical recurrent neural network for skeleton based action recognition, CVPR, 2015.

Multi-Channel CNNs for Gait Recognition

Task: **model walking styles for person identification in videos**
Solution: propose **multi-channel CNNs** to capture the appearance and motion variations





Proposed architectures

Wu et al., A Comprehensive Study on Cross-view Gait based Human Identification with Deep CNNs, TPAMI, 2017.

Multi-Channel CNNs for Gait Recognition

CASIA-B: the most difficult gait dataset

Our accuracy 90% outperforms previous best accuracy 83%

Gallery NM #1-4	0°-180°				36°-144°		
Probe NM #5-6	0°	54°	90°	126°	54°	90°	126°
SVR [34]	–	28	29	34	35	44	45
TSVD [33]	–	39	33	42	49	50	54
CMCC [12]	46.3	52.4	48.3	56.9	-	-	-
ViDP [27]	–	59.1	50.2	57.5	83.5	76.7	80.7
Ours	**54.8**	**77.8**	**64.9**	**76.1**	**90.8**	**85.8**	**90.4**

OULP: the largest gait dataset

Our accuracy 98% outperforms previous best accuracy 87%

Probe angle	Gallery angle	Identical angle	
	Mean	Ours	NN [28]
55°	91.6 ± 0.2	98.8 ± 0.1	84.7
65°	92.3 ± 0.1	98.9 ± 0.2	86.6
75°	92.4 ± 0.1	98.9 ± 0.0	86.9
85°	94.8 ± 0.3	98.9 ± 0.1	85.7

Large-scale remote gait recognition

CASIA-HT: current largest dataset

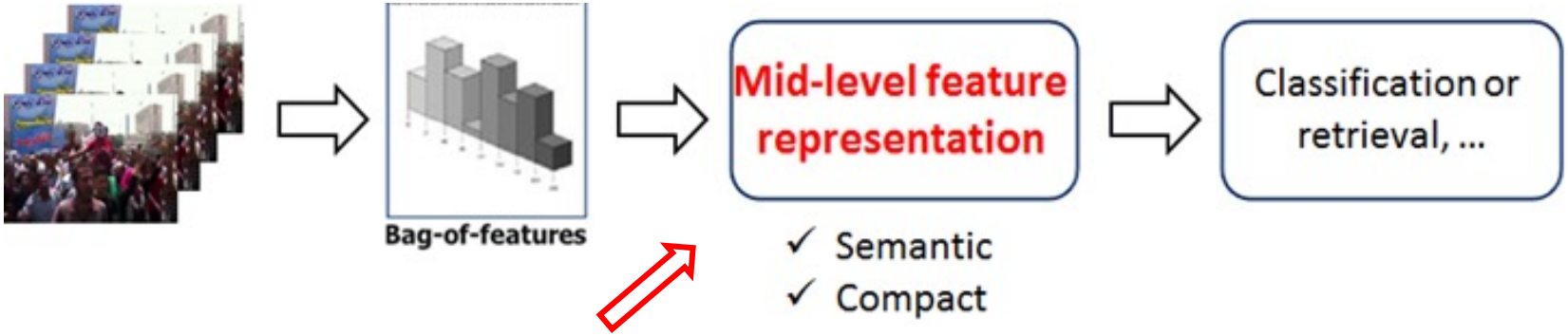
Industrialization

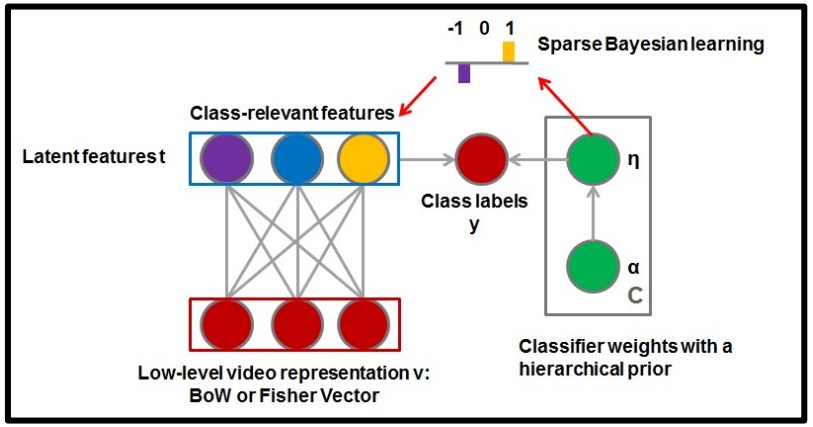
Watrix technology co., LTD

http://www.watrix.cc

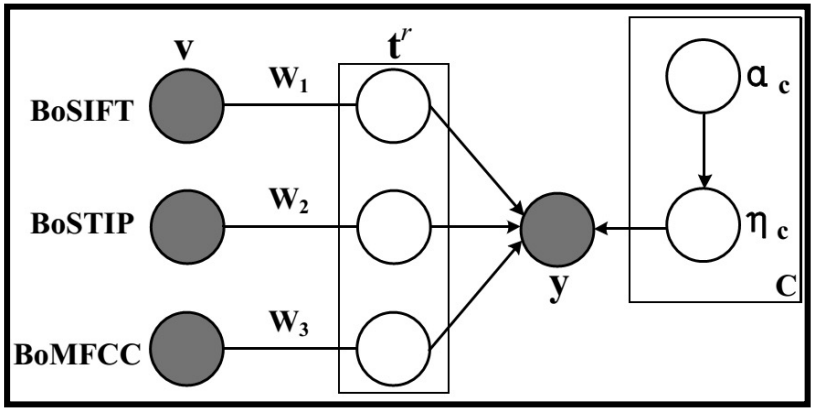
Class-relevance RBM for Event Recognition

Task: **recognize complex group action and event in videos**
Solution: propose **class-relevance RBM** to jointly learn discriminate mid-level features and sparse classifiers





Multimodal extension

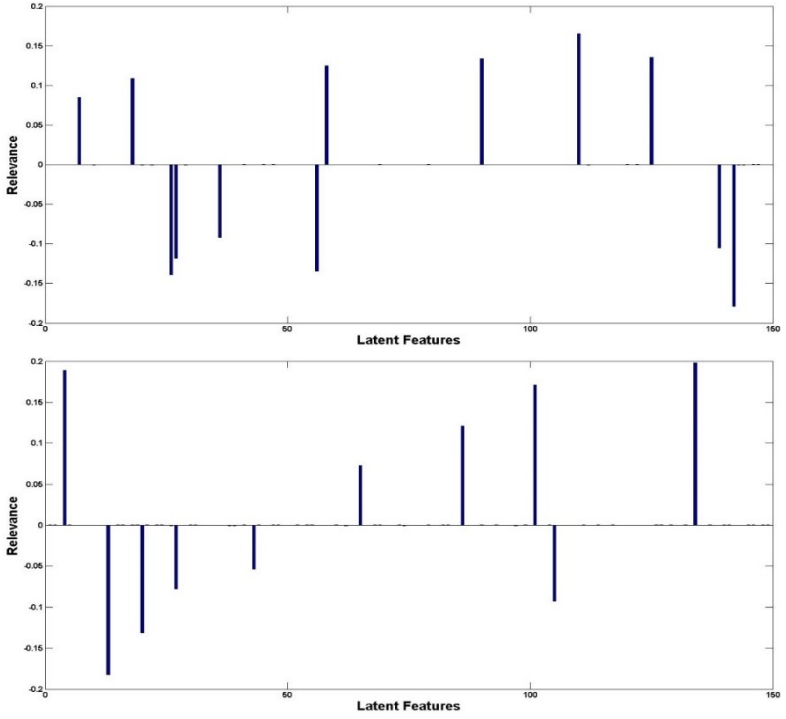


Zhao et al., Learning Relevance Restricted Boltzmann Machine for Unstructured Group Activity and Event Understand-ing, IJCV, 2016.
Zhao et al., Relevance Topic Model for Unstructured Social Group Activity Recognition, NIPS, 2013.

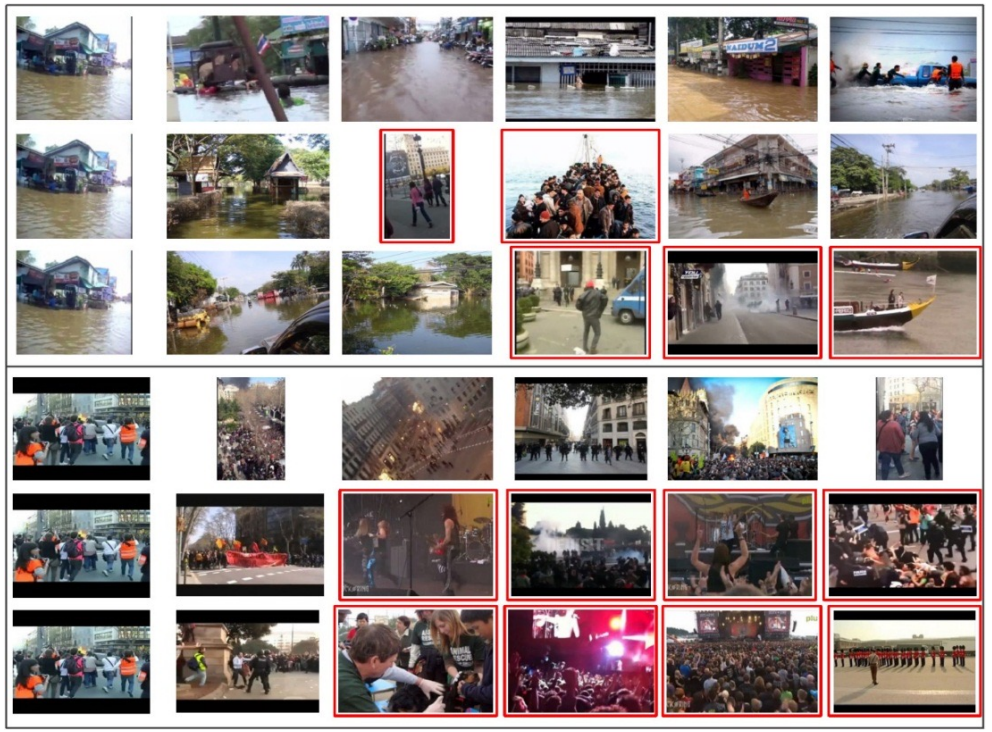
Class-relevance RBM for Event Recognition

Visualization of interested points extracted by our model

Learned class-relevance sparse features



Comparison of retrieved results



Summary

- Action recognition is well investigated in short videos, but there still remain many problems unsolved in long videos
- Online real-time action detection and recognition for robotics should be paid much attention
- Group event analysis is under great demands and need to be further improved

Big Visual Data Analysis

Large-Scale Dataset

Large-Scale Visual Computing

Platforms and Applications

Object Detection/ Segmentation

Object Recognition/ Retrieval

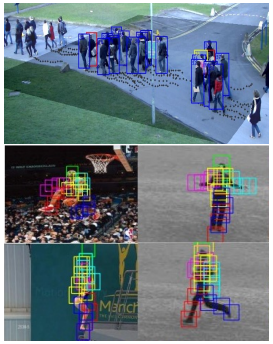
Motion/ Behavior Analysis

Scene Understanding

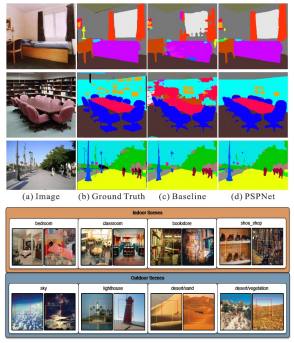
Vision by Language

Face/pedestrian detection/ segmentation

Face/gait recognition/ retrieval



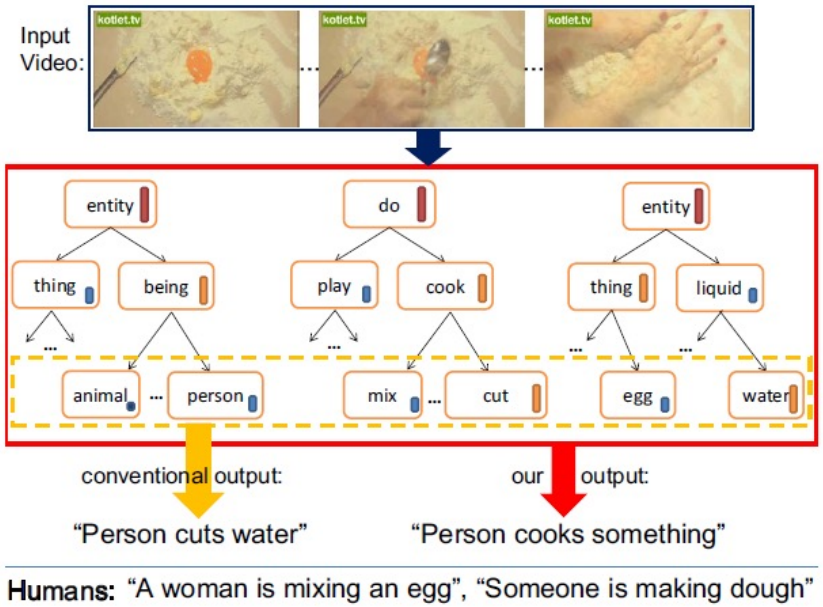
Object tracking action recognition



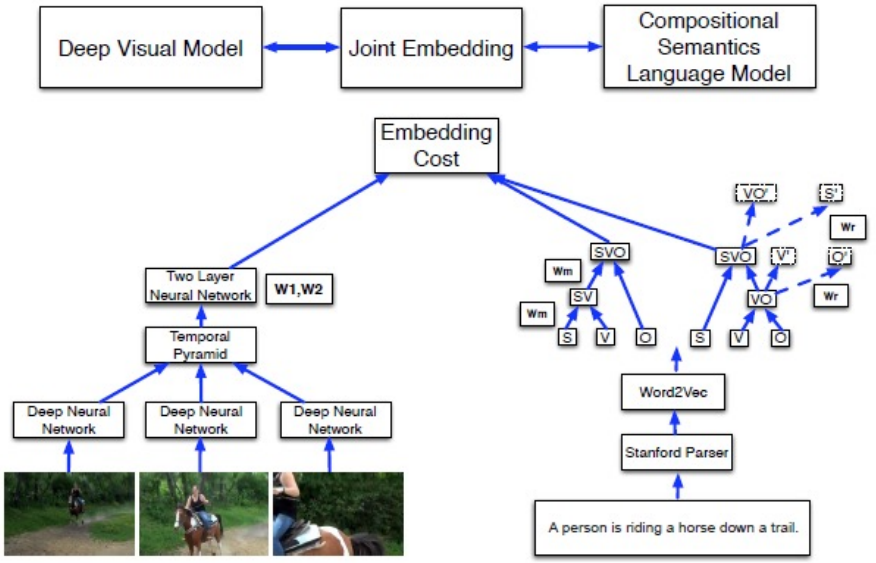
Scene classification/ parsing

Visual captioning/ question answering

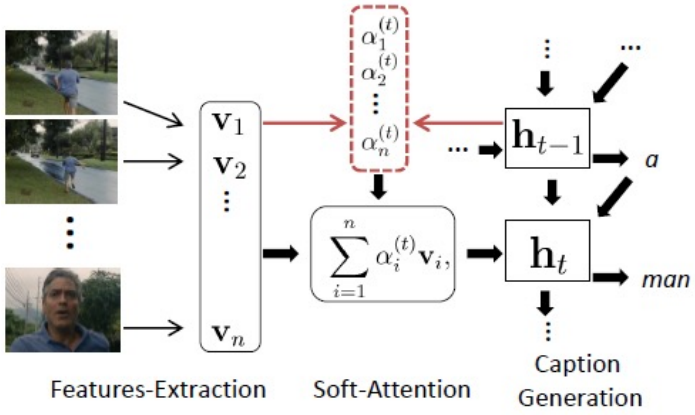
Visual Captioning

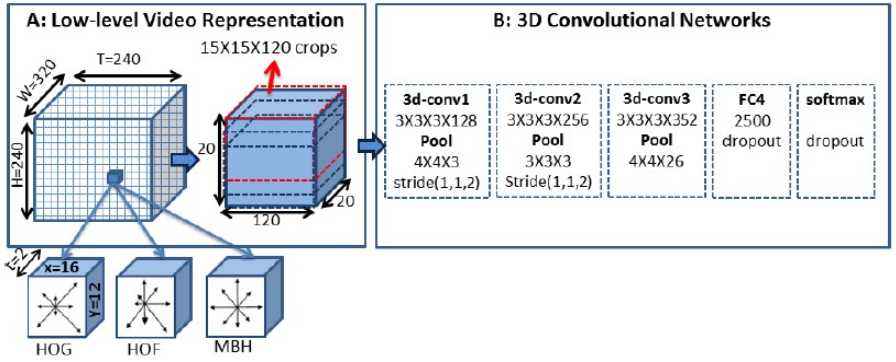


Guadarrama et al., Youtube2text:Recognizing and describing arbitrary activities, ICCV, 2013.



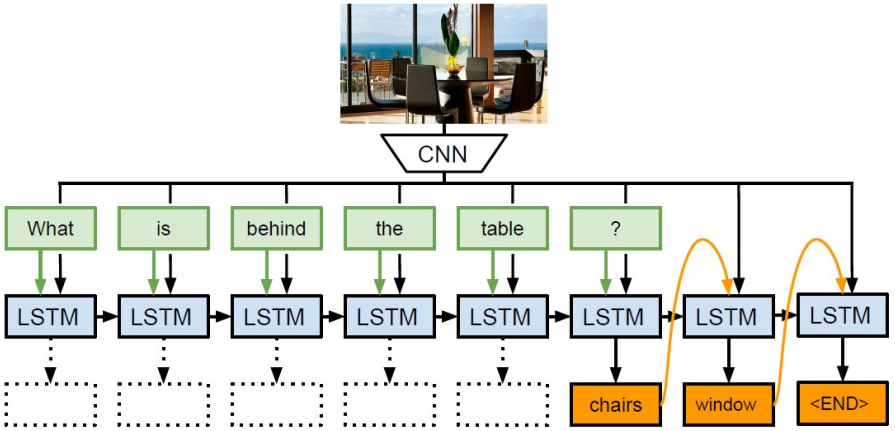
Krishnamoorthy et al., Generating Natural-Language Video Descriptions Using Text-Mined Knowledge, AAAI, 2013.



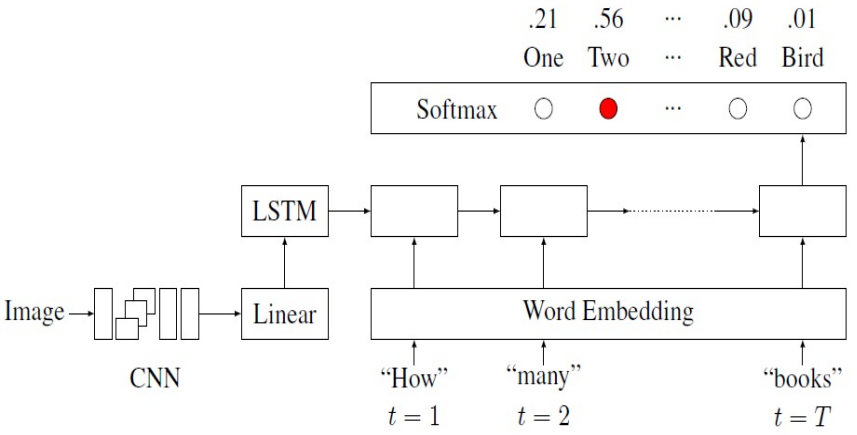


Yao et al., Describing Videos by Exploiting Temporal Structure, ICCV, 2015.

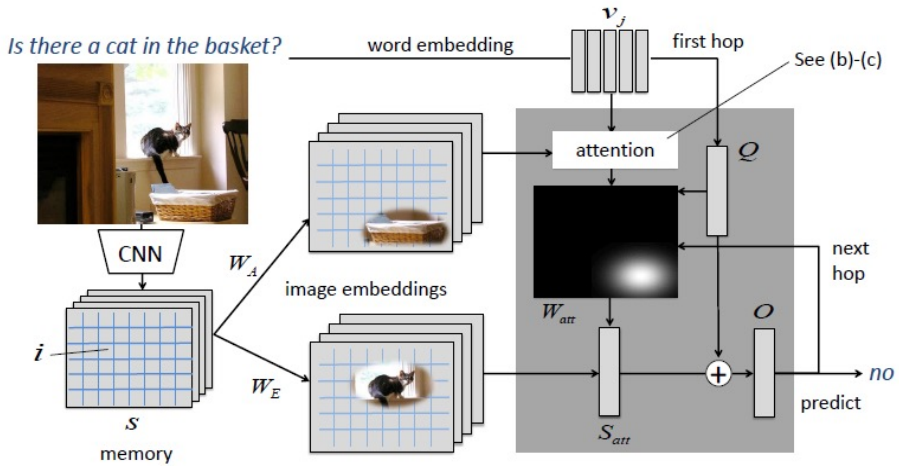
Visual Question Answering



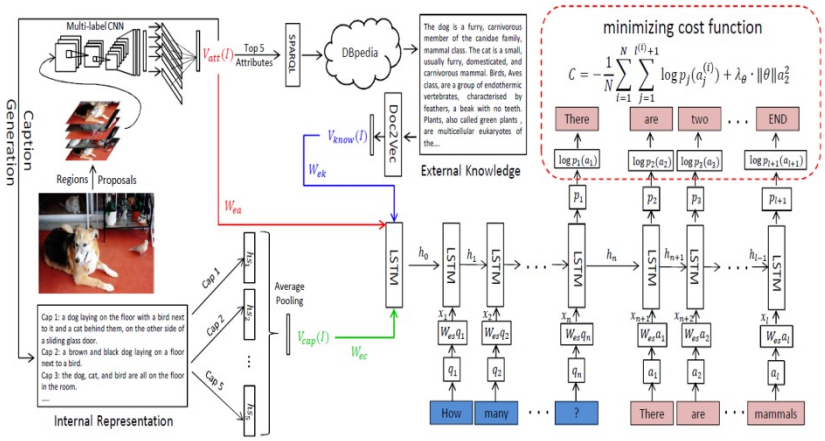
Malinowski et al., Ask Your Neurons: A Neural-based Approach to Answering Questions about Images, ICCV, 2013.



Ren et al., Exploring Models and Data for Image Question Answering, NIPS, 2013.



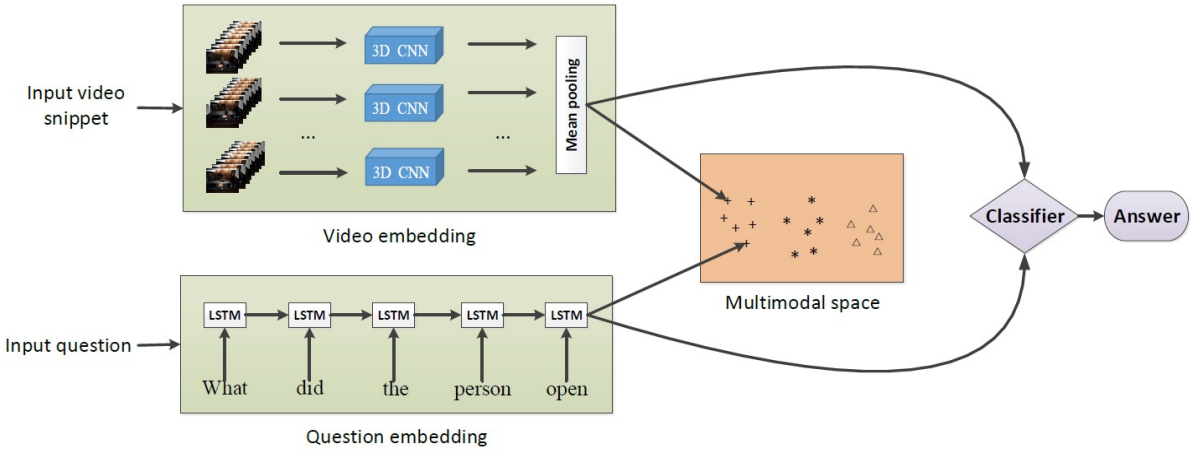
Xu et al., Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering, ECCV, 2016.



Wu et al., Ask, Ask Me Anything: Free-form Visual Question Answering Based on Knowledge from External Sources, CVPR, 2016.

Long Video Question Answering

Task: **interactively understand very long videos given questions**
Solution: segment a long video into multiples clips, find the most similar clip to the given question, and then predict the answer



Map the video clip and question in the common space

W. Wang et al., Long Video Question Answering: A retrieval based method, PR, 2020.

Long Video Question Answering

Recognize the location of objects

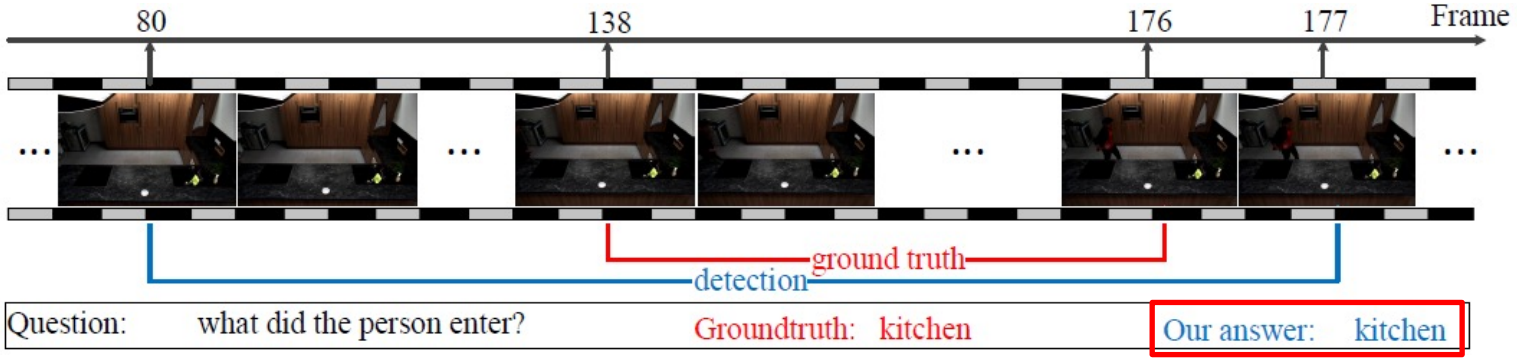
f \ q	Object	Number	Color	Location	Total
8	**14.85%**	**41.53%**	45.88%	**37.89%**	**19.59%**
16	14.50%	40.21%	**47.06%**	37.46%	19.19%
32	14.55%	30.69%	47.05%	34.79%	18.34%
64	14.50%	35.71%	45.88%	35.07%	18.62%

Achieve better recognition accuracy

f	8	16	32	64
IoU	4.54%	7.77%	4.82%	**8.29%**

f	8	16	32	64
Baseline	14.66%	14.71%	14.73%	14.56%
Ours	**19.59%**	**19.19%**	**18.34%**	**18.62%**

Examples of long video question answering



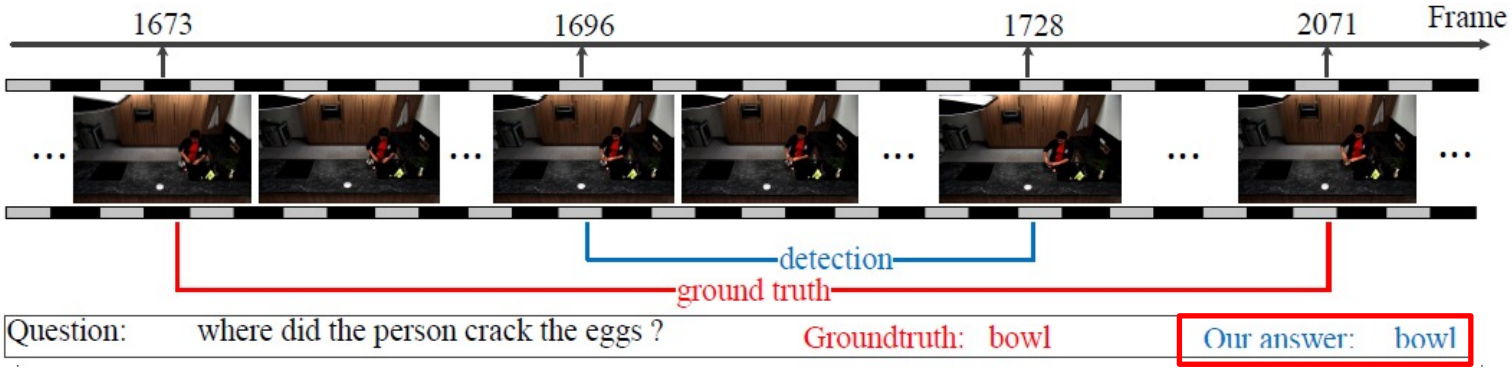
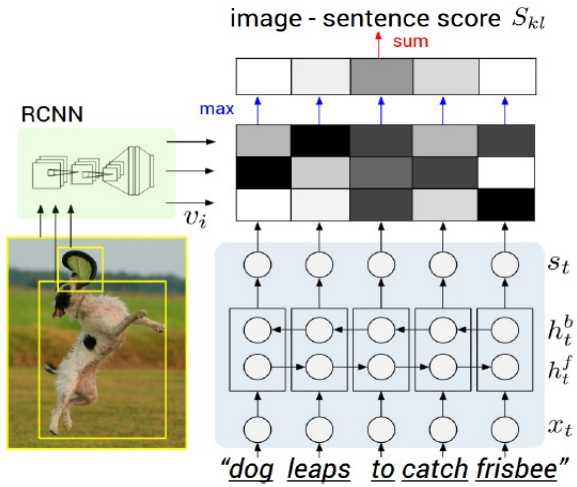
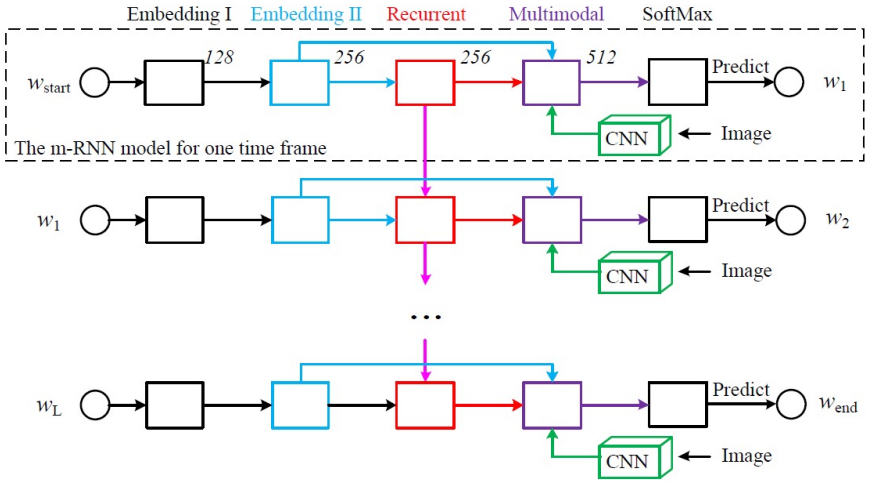


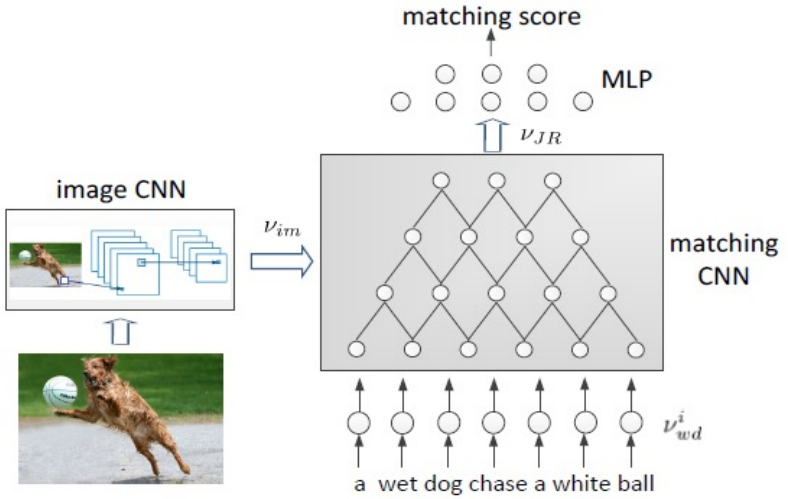
Image and Sentence Matching



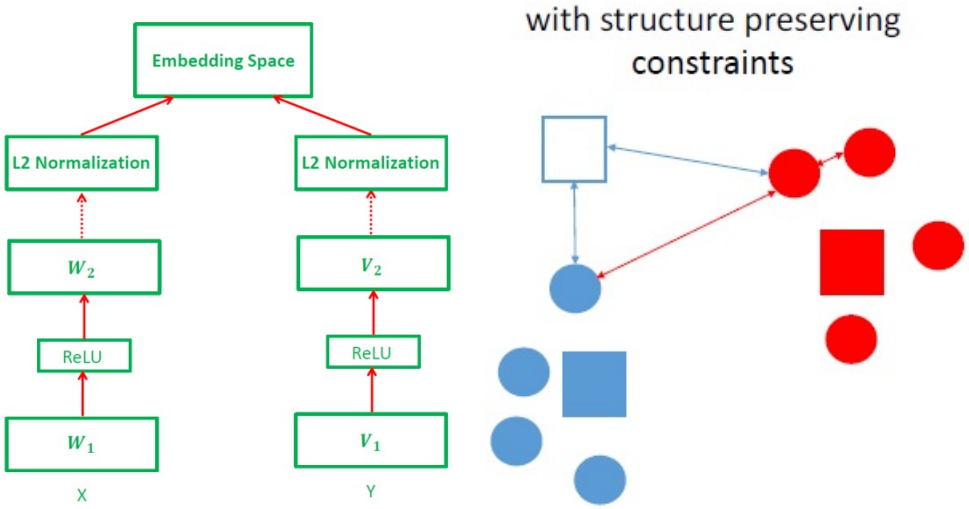
Karpathy et al., Deep Visual-Semantic Alignments for Generating Image Descriptions, CVPR, 2016.



Mao et al., Deep Captioning with Multimodal Recurrent Neural Networks, ICLR, 2015.



Ma et al., Multimodal Convolutional Neural Networks for Matching Image and Sentence, ICCV, 2015.

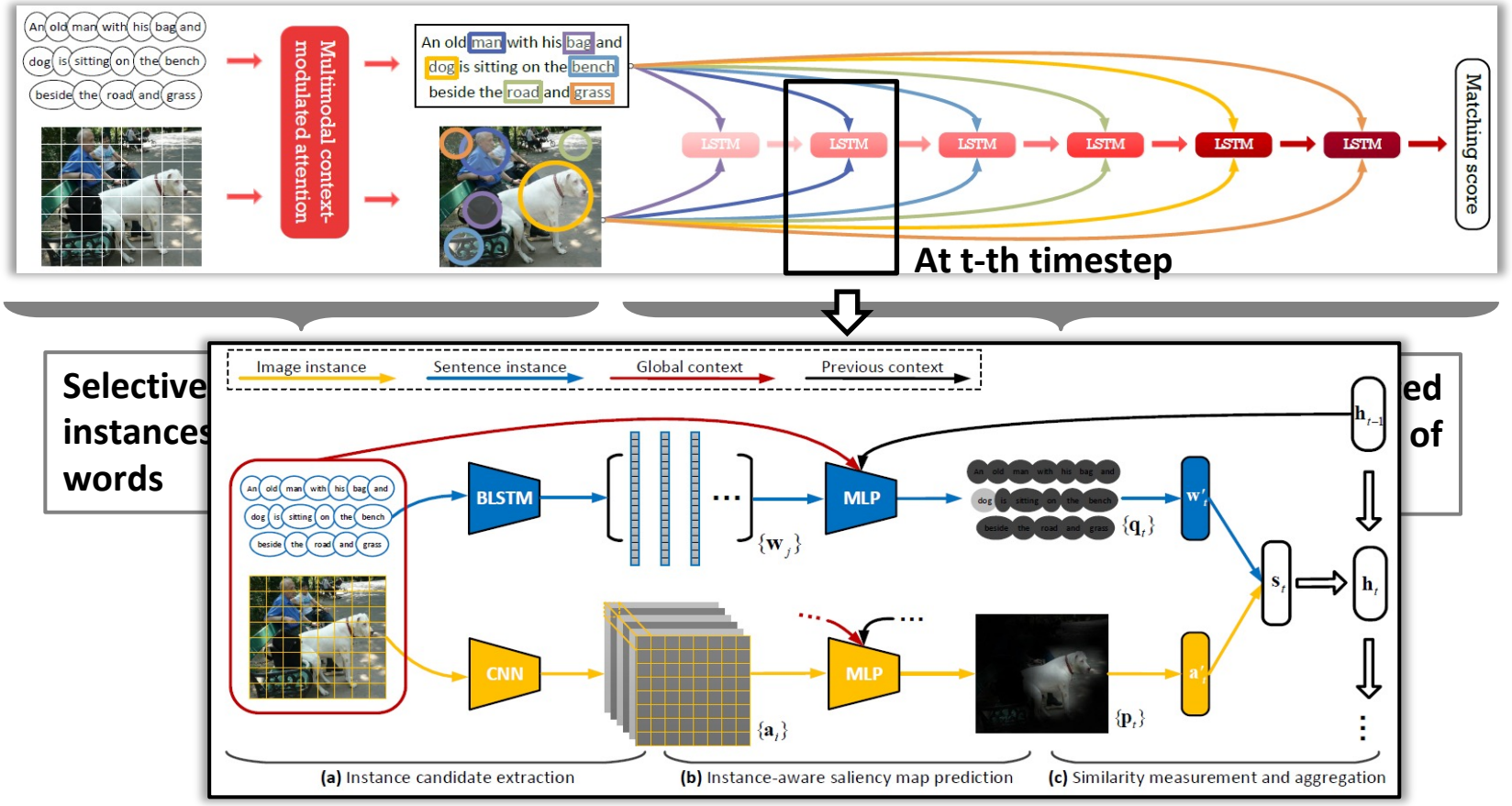


Wang et al., Learning Deep Structure-Preserving Image-Text Embeddings, CVPR, 2016.

Selective Multimodal LSTM for Instance-aware Visual-semantic Matching

Task: **match salient objects and words in image and sentence**

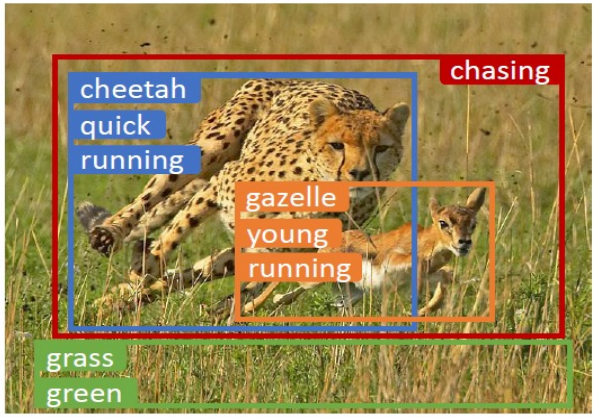
Solution: take image and sentence for example, propose **selective multimodal LSTM** to selectively attend to salient instances



Y. Huang et al., Instance-aware Image and Sentence Matching with Selective Multimodal LSTM, CVPR, 2017.

Learning Semantic Concepts and Order for Image and Sentence Matching

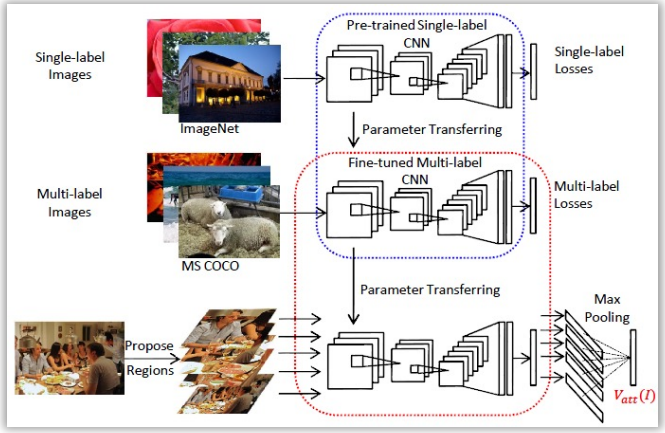
Task: **learn semantic concepts and organize them in a semantic order**
Solution: use multi-label CNN to predict semantic concepts, and context-gated sentence generation scheme for semantic order learning



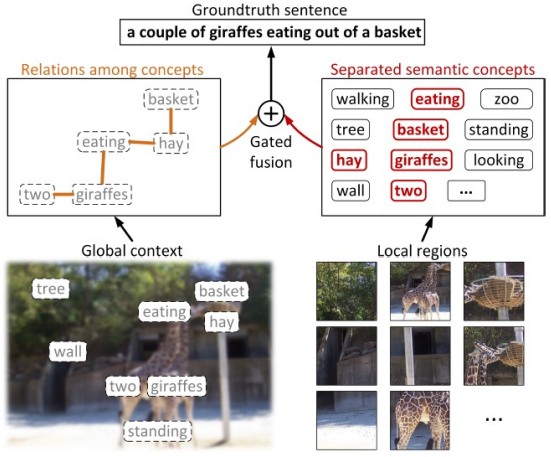
- **Semantic concepts:**

Objects:	Properties:	Actions:
cheetah	quick	chasing
gazelle	young	running
grass	green	running

- **Semantic order:**
  cheetah chasing gazelle grass
  ...
- **Matched sentence:**
  A quick cheetah is chasing a young gazelle on grass.



multi-label CNN



context-gated sentence generation

Y. Huang et al., Learning Semantic Concepts and Order for Image and Sentence Matching, CVPR, 2018.

Summary

- Joint vision and language understanding has achieved impressive results, and there still exist much space to explore

- Due to the huge visual-semantic gap, deep visual analysis is the core of visual captioning/question answering

- Modeling cognitive mechanisms, e.g., attention and memory, is the future direction for vision and language

Big Visual Data Analysis

Large-Scale Dataset

Large-Scale Visual Computing

Platforms and Applications

Object Detection/ Segmentation

Object Recognition/ Retrieval

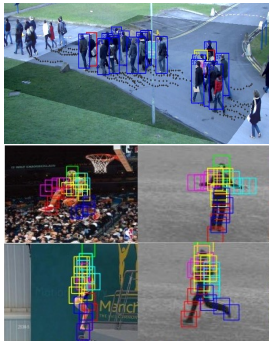
Motion/ Behavior Analysis

Scene Understanding

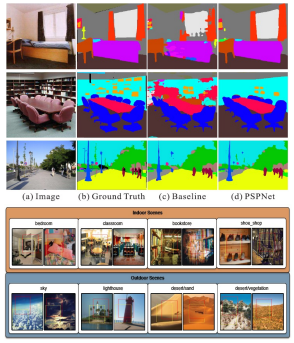
Vision by Language

Face/pedestrian detection/ segmentation

Face/gait recognition/ retrieval



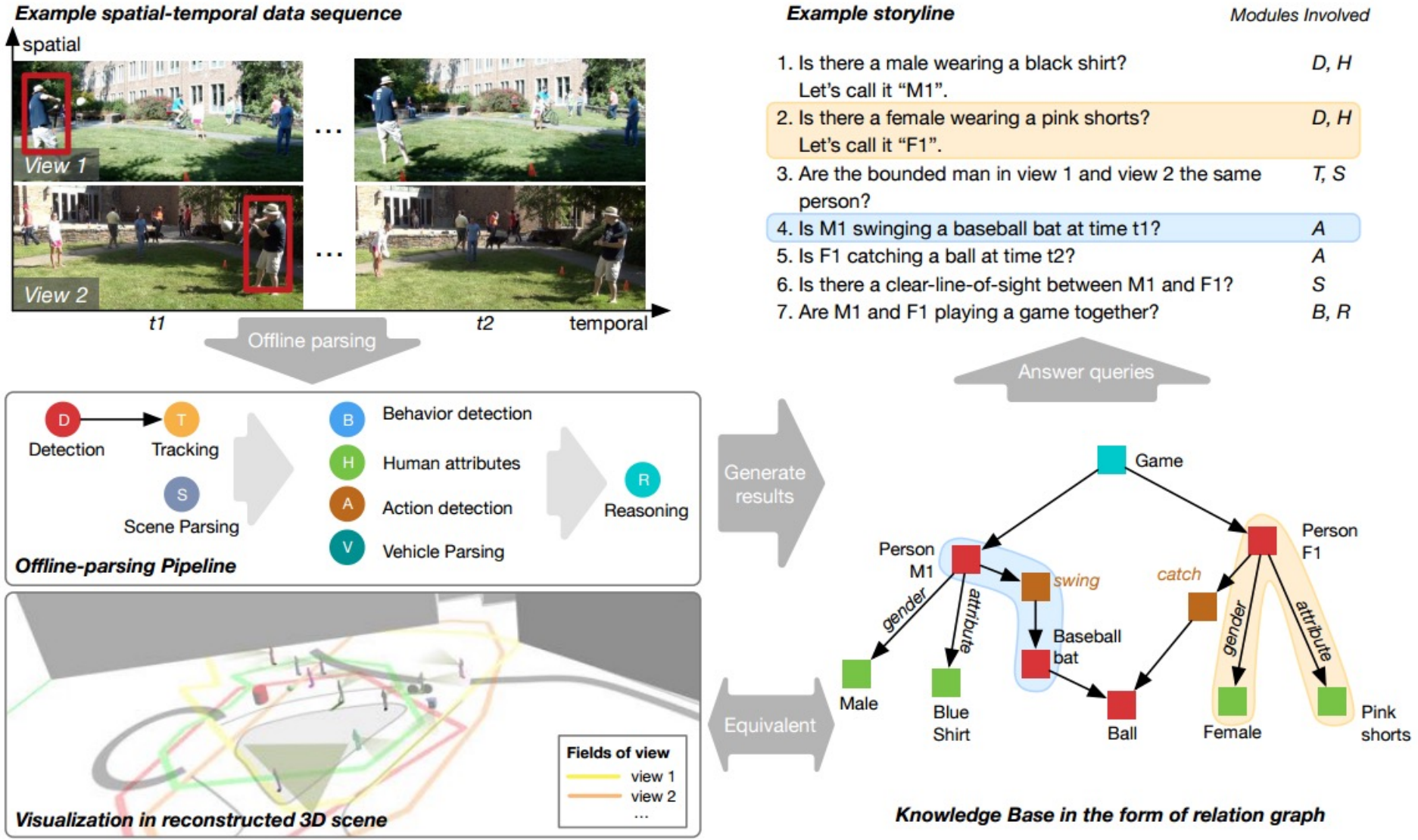
Object tracking action recognition



Scene classification/ parsing

Visual captioning/ question answering

Visual Turing Test



Q. Hang et al., A Restricted Visual Turing Test for Deep Scene and Event Understanding, arXiv 1512.01715.

ISEE: Intelligent Scene Evaluation and Exploration

=

Big Visual Data

+

System

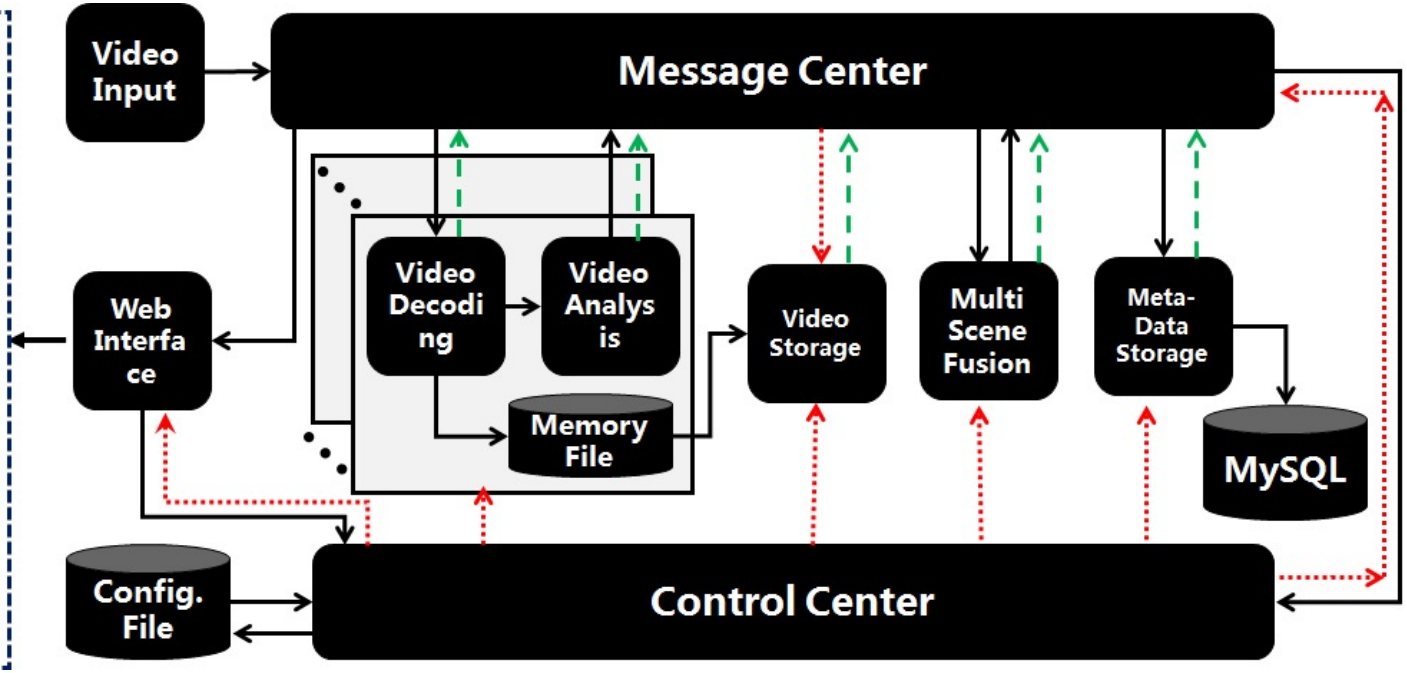
+

Algorithms

Object, Event Retrieval

Interesting Pattern Discovery

Scene Situation Evaluation

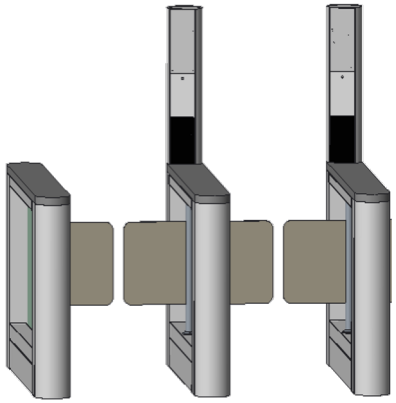


SIR: Smart Identity Recognition

- Very first commercialized long-distance iris and face recognition system in China.
- Core techs are independently developed by CASIA
- High stability and high precision for large-scale person identification

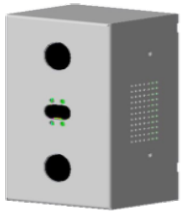
SIR001A Column

SIR002A Column

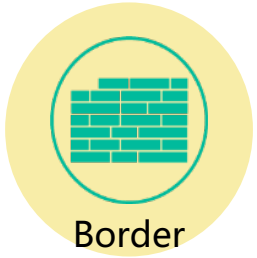


SIR002B E-Gate

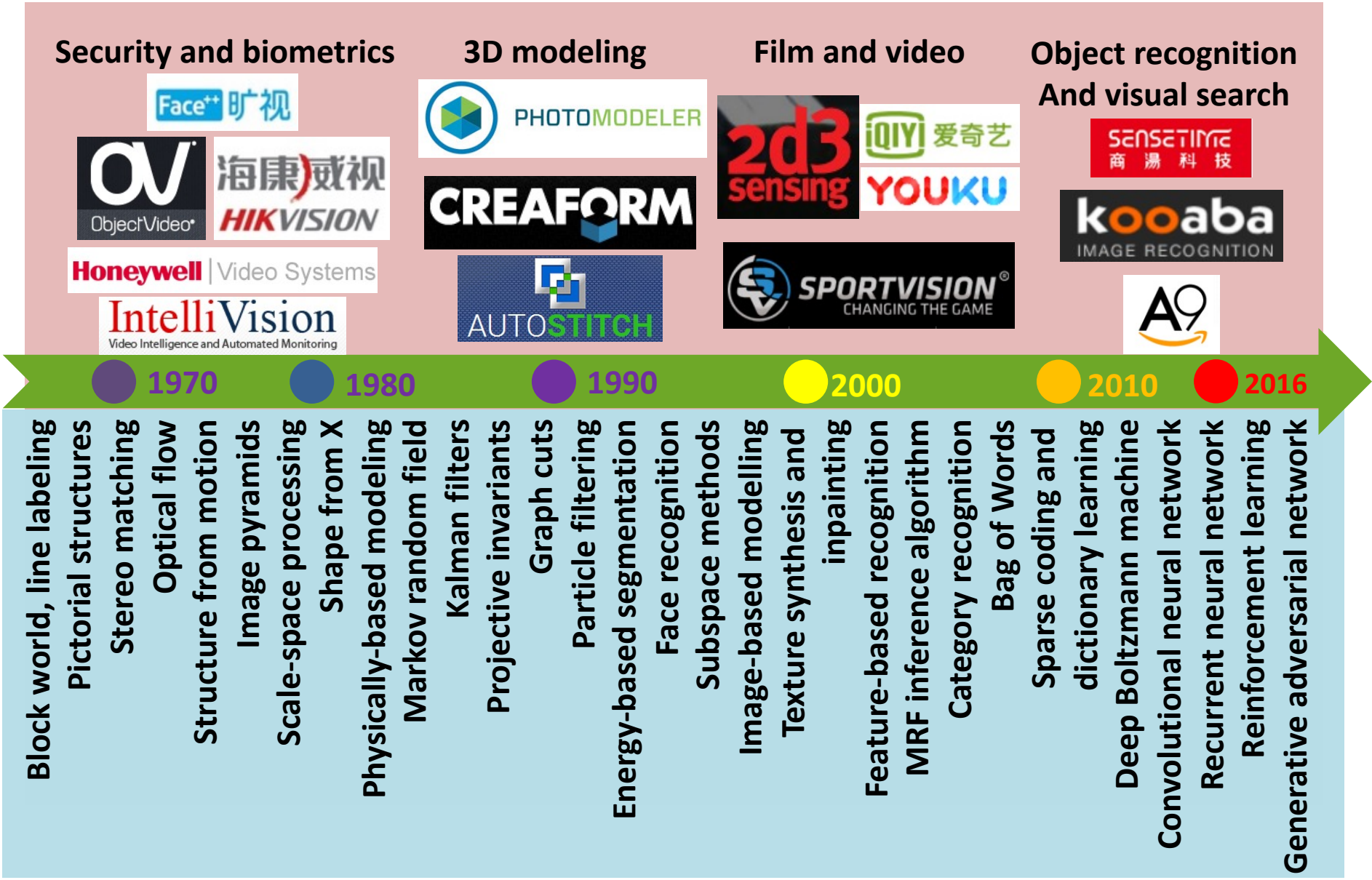
SIR003A Portable



SIR002 Module



Computer Vision Companies



Outline

1. Course Review
2. DL for Computer Vision
3. DL Platforms

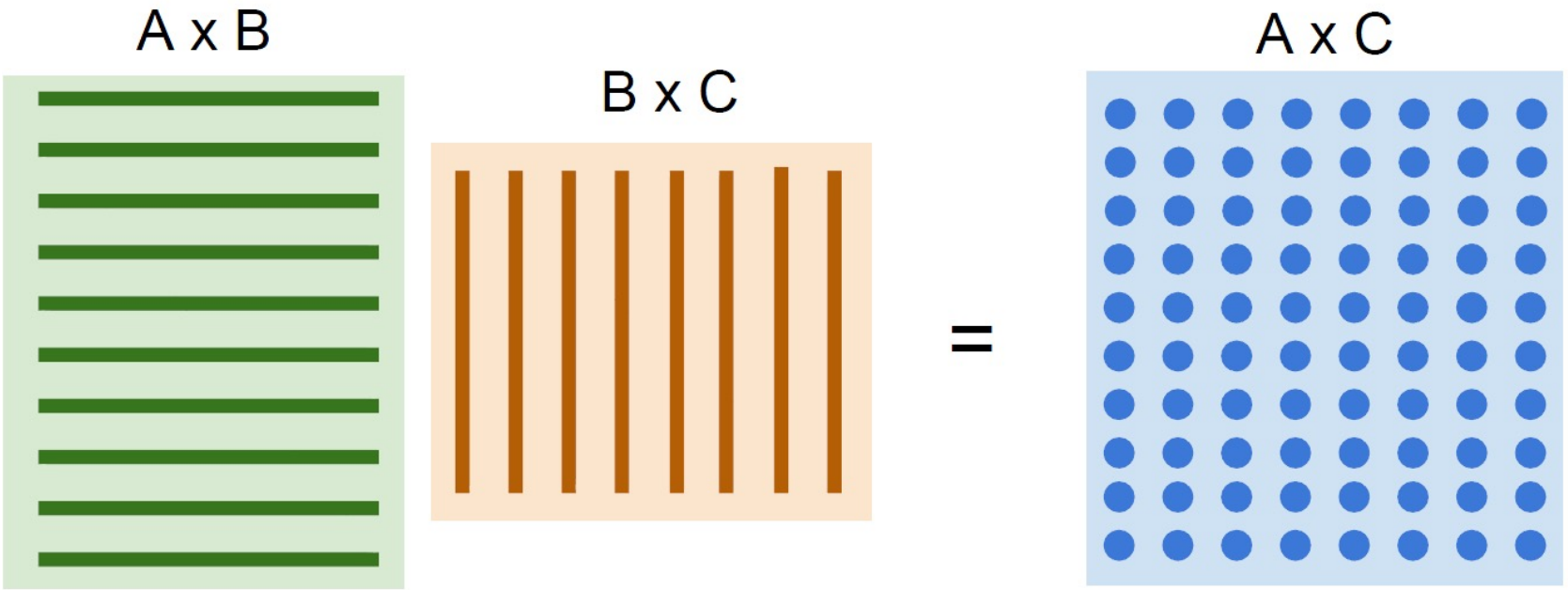
CPU vs GPU

	# 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

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

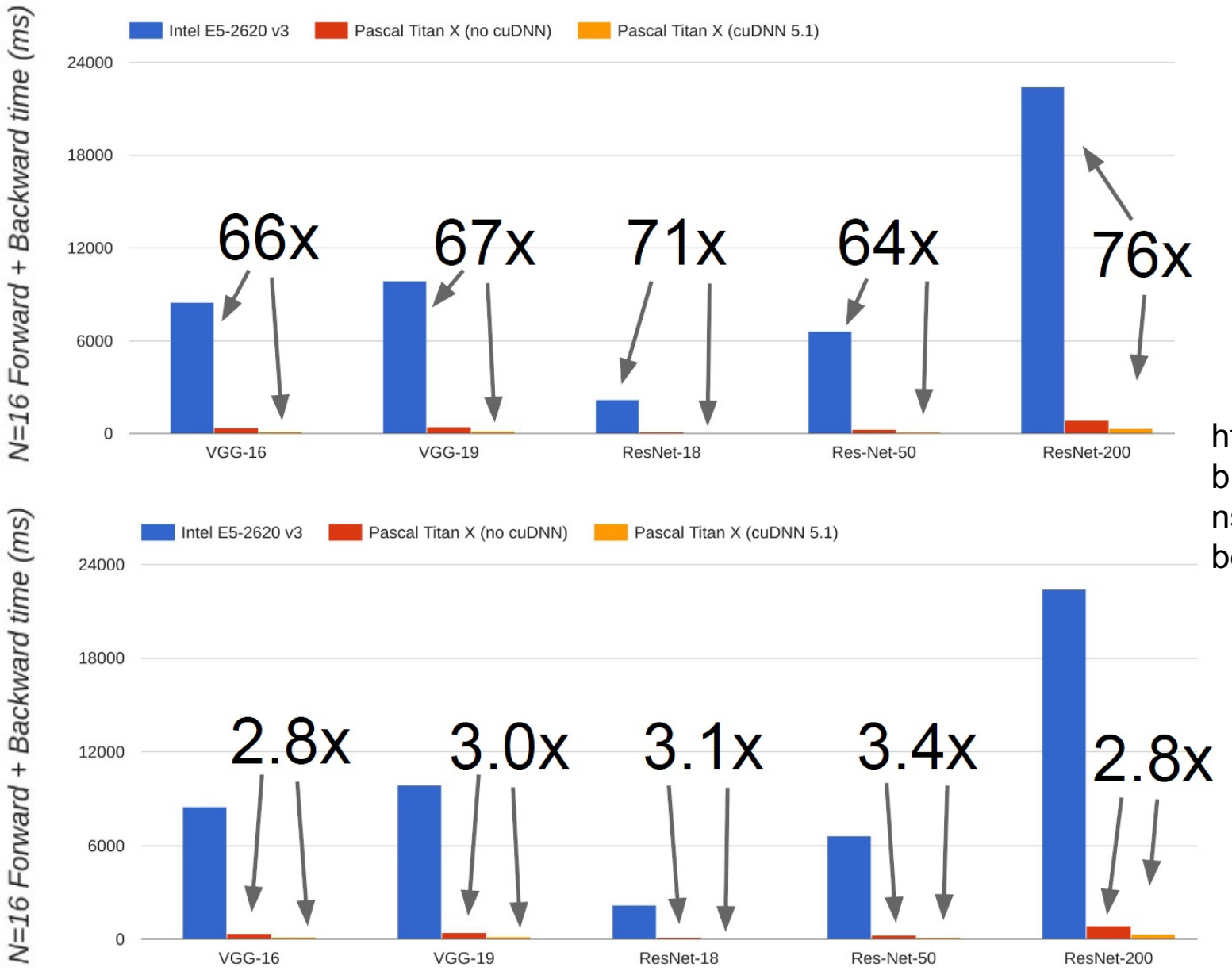
Example: Matrix Multiplication



Spot the GPUs!
(graphics processing unit)

This image is in the public domain

Example: Matrix Multiplication



https://github.com/jcjohnson/cnn-benchmarks

Deep Learning Frameworks

A bit about these

Caffe
(UC Berkeley)

→ Caffe2
(Facebook)

Torch
(NYU / Facebook)

→ PyTorch
(Facebook)

Theano
(U Montreal)

→ TensorFlow
(Google)

Mostly these

Paddle
(Baidu)

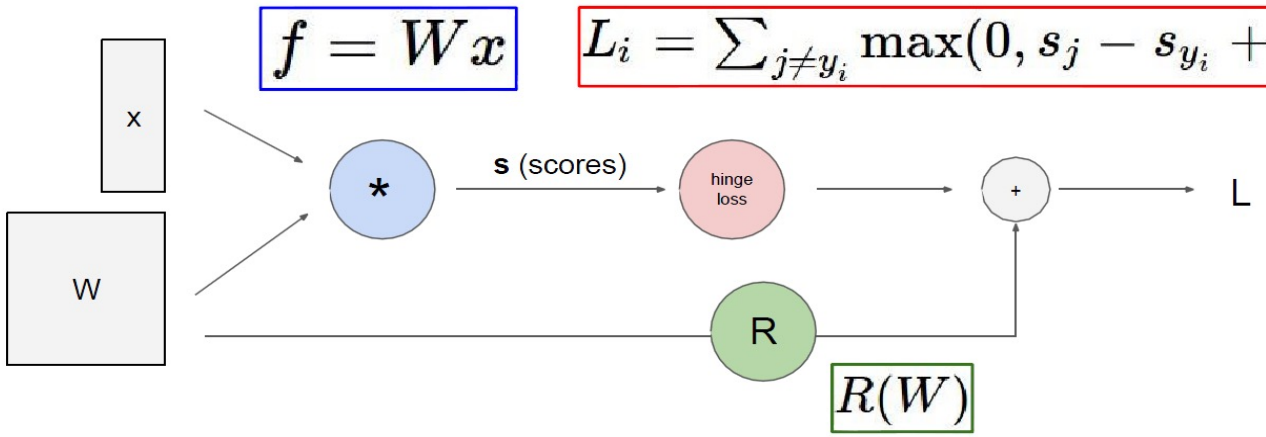
CNTK
(Microsoft)

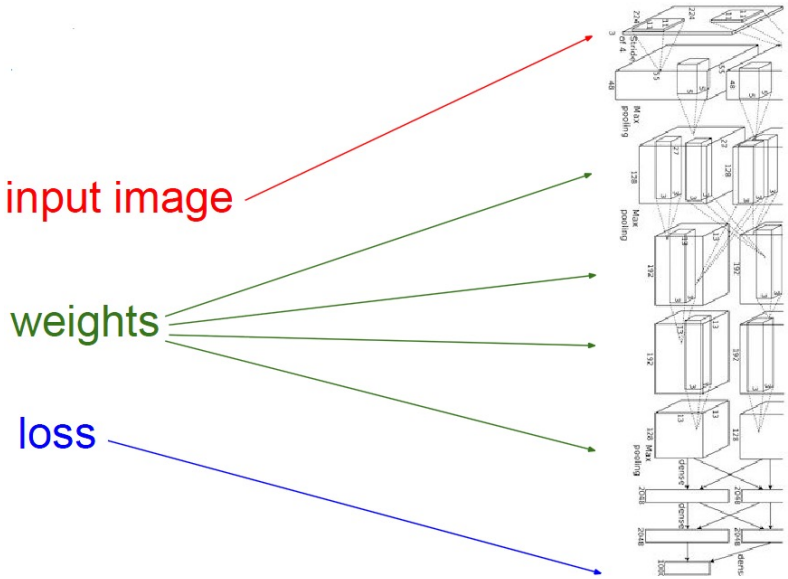
MXNet
(Amazon)

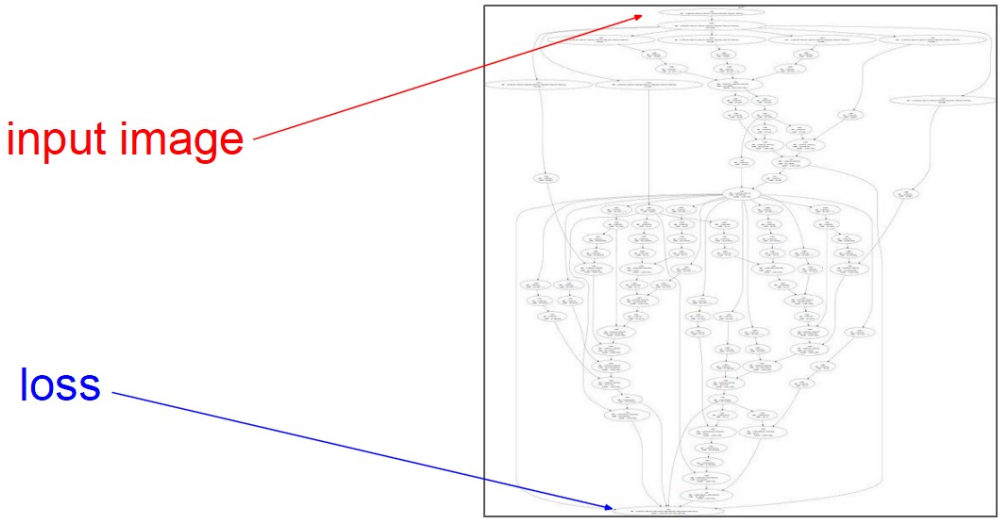
Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

And others...

Computational Graphs





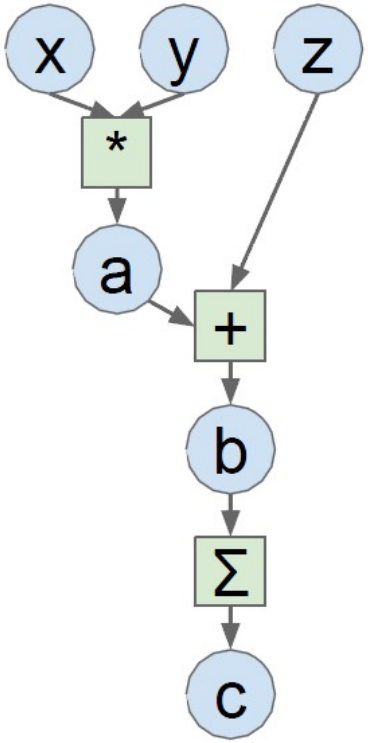


Computational Graphs

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

Numpy

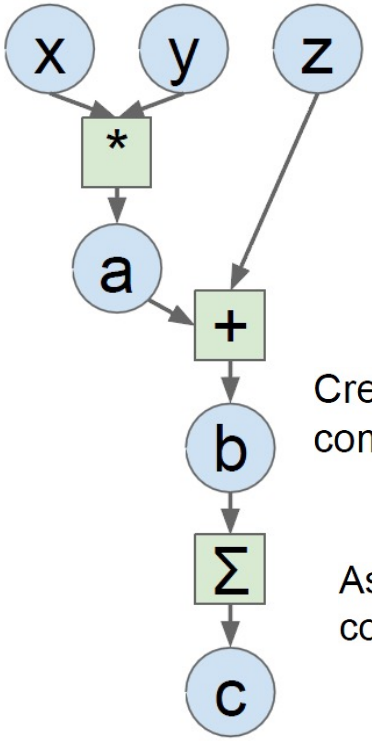
```
import numpy as np
np.random.seed(0)

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
```



Problems:

- Can't run on GPU
- Have to compute our own gradients

Computational Graphs



TensorFlow

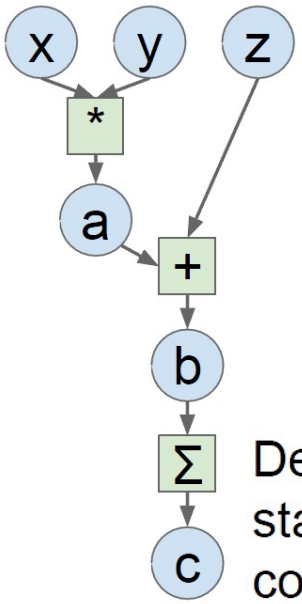
Tell TensorFlow to run on **CPU**

Create forward computational graph

Ask TensorFlow to compute gradients

```
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
```

Computational Graphs



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

Forward pass looks just like numpy

Calling c.backward() computes all gradients

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)
```

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

```
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)
```

TensorFlow: Neural Net

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

```
import numpy as np
import tensorflow as tf

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

```
import numpy as np
import tensorflow as tf

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

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

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

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

Create numpy arrays that will fill in the placeholders above

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

```
import numpy as np
import tensorflow as tf

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
```

Problem: copying weights between CPU / GPU each step

TensorFlow: Neural Net

Change w1 and w2 from **placeholder** (fed on each call) to **Variable** (persists in the graph between calls)

Can use an **optimizer** to compute gradients and update weights

Remember to execute the output of the optimizer!

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((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 * diff, axis=1))

optimizer = tf.train.GradientDescentOptimizer(1e-5)
updates = optimizer.minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    values = {x: np.random.randn(N, D),
              y: np.random.randn(N, D),}
    losses = []
    for t in range(50):
        loss_val, _ = sess.run([loss, updates],
                               feed_dict=values)
```

Cannot use predefined common loss and layer

Keras: High-Level Wrapper

Keras is a layer on top of TensorFlow, makes common things easy to do, also supports Theano backend

Define model object as a sequence of layers

Define optimizer object

Build the model, specify loss function

Train the model with a single line!

```
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)
```

Theano

TensorFlow is similar in many ways to Theano (earlier framework from Montreal)

```
import theano
import theano.tensor as T

# Batch size, input dim, hidden dim, num classes
N, D, H, C = 64, 1000, 100, 10

x = T.matrix('x')
y = T.vector('y', dtype='int64')
w1 = T.matrix('w1')
w2 = T.matrix('w2')

# Forward pass: Compute scores
a = x.dot(w1)
a_relu = T.nnet.relu(a)
scores = a_relu.dot(w2)

# Forward pass: compute softmax loss
probs = T.nnet.softmax(scores)
loss = T.nnet.categorical_crossentropy(probs, y).mean()

# Backward pass: compute gradients
dw1, dw2 = T.grad(loss, [w1, w2])

f = theano.function(
      inputs=[x, y, w1, w2],
      outputs=[loss, scores, dw1, dw2],
    )
```

Define symbolic variables (similar to TensorFlow placeholder)

Forward pass: compute predictions and loss

Ask Theano to compute **gradients** for us

Compile a **function** that computes loss, scores, and gradients from data and weights

Theano

```
# Run the function
xx = np.random.randn(N, D)
yy = np.random.randint(C, size=N)
ww1 = 1e-2 * np.random.randn(D, H)
ww2 = 1e-2 * np.random.randn(H, C)

learning_rate = 1e-1
for t in xrange(50):
  loss, scores, dww1, dww2 = f(xx, yy, ww1, ww2)
  print loss
  ww1 -= learning_rate * dww1
  ww2 -= learning_rate * dww2
```

Run the function many times to train the network

```
import theano
import theano.tensor as T

# Batch size, input dim, hidden dim, num classes
N, D, H, C = 64, 1000, 100, 10

x = T.matrix('x')
y = T.vector('y', dtype='int64')
w1 = T.matrix('w1')
w2 = T.matrix('w2')

# Forward pass: Compute scores
a = x.dot(w1)
a_relu = T.nnet.relu(a)
scores = a_relu.dot(w2)

# Forward pass: compute softmax loss
probs = T.nnet.softmax(scores)
loss = T.nnet.categorical_crossentropy(probs, y).mean()

# Backward pass: compute gradients
dw1, dw2 = T.grad(loss, [w1, w2])

f = theano.function(
      inputs=[x, y, w1, w2],
      outputs=[loss, scores, dw1, dw2],
    )
```

PyTorch

Three Levels of Abstraction	TensorFlow equivalent
Tensor: Imperative ndarray, but runs on GPU	Numpy array
Variable: Node in a computational graph; stores data and gradient	Tensor, Variable, Placeholder
Module: A neural network layer; may store state or learnable weights	tf.layers, or TFSlim, or TFLearn, or Sonnet, or ….

PyTorch: Tensors

PyTorch Tensors are just like numpy arrays, but they can run on GPU.

To run on GPU, just cast tensors to a cuda datatype!

Create random tensors for data and weights

Forward pass: compute predictions and loss

Backward pass: manually compute gradients

Gradient descent step on weights

```
import torch

dtype = torch.cuda.FloatTensor

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

PyTorch: Autograd

A PyTorch **Variable** is a node in a computational graph

- x.data is a Tensor
- x.grad is a Variable of gradients (same shape as x.data)
- x.grad.data is a Tensor of gradients

We will not want gradients (of loss) with respect to data
Do want gradients with respect to weights

Forward pass looks exactly the same as the Tensor version, but everything is a variable now

Compute gradient of loss with respect to w1 and w2 (zero out grads first)

Make gradient step on weights

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in), requires_grad=False)
y = Variable(torch.randn(N, D_out), requires_grad=False)
w1 = Variable(torch.randn(D_in, H), requires_grad=True)
w2 = Variable(torch.randn(H, D_out), requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    if w1.grad: w1.grad.data.zero_()
    if w2.grad: w2.grad.data.zero_()
    loss.backward()

    w1.data -= learning_rate * w1.grad.data
    w2.data -= learning_rate * w2.grad.data
```

PyTorch: nn

Higher-level wrapper for working with neural nets

Define our model as a sequence of layers

nn also defines common loss functions

Forward pass: feed data to model, and prediction to loss function

Backward pass: compute all gradients

Make gradient step on each model parameter

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    model.zero_grad()
    loss.backward()

    for param in model.parameters():
        param.data -= learning_rate * param.grad.data
```

PyTorch: Pretrained Models

```
import torch
import torchvision

alexnet = torchvision.models.alexnet(pretrained=True)
vgg16 = torchvision.models.vgg16(pretrained=True)
resnet101 = torchvision.models.resnet101(pretrained=True)
```

Super easy to use pretrained models with torchvision
https://github.com/pytorch/vision

Torch vs PyTorch

Torch
(-) Lua
(-) No autograd
(+) More stable
(+) Lots of existing code
(0) Fast

PyTorch
(+) Python
(+) Autograd
(-) Newer, still changing
(-) Less existing code
(0) Fast

Conclusion: Probably use PyTorch for new projects

Caffe

- Core written in C++
- Has Python and MATLAB bindings
- Good for training or finetuning feedforward classification models
- Often no need to write code!
- Not used as much in research anymore, still popular for deploying models

Training / Finetuning

No need to write code!

1. Convert data (run a script)
2. Define net (edit prototxt)
3. Define solver (edit prototxt)
4. Train (with pretrained weights) (run a script)

No Need to Write Code

Step 1: Convert Data

- DataLayer reading from LMDB is the easiest
- Create LMDB using *convert_imageset*
- Need text file where each line is:
  "[path/to/image.jpeg] [label]"
- Create HDF5 file yourself using h5py
- Use Python interface

Step 2: Define Network (prototxt)

```
name: "LogisticRegressionNet"
layers {
  top: "data"
  top: "label"
  name: "data"
  type: HDF5_DATA
  hdf5_data_param {
    source: "examples/hdf5_classification/data/train.txt"
    batch_size: 10
  }
  include {
    phase: TRAIN
  }
}
```

```
inner_product_param {
  num_output: 2
  weight_filler {
    type: "gaussian"
    std: 0.01
  }
  bias_filler {
    type: "constant"
    value: 0
  }
}
```

```
layers {
  bottom: "data"
  top: "fc1"
  name: "fc1"
  type: INNER_PRODUCT
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
```

No Need to Write Code

Step 3: Define Solver (prototxt)

- Write a prototxt file defining a SolverParameter
- If finetuning, copy existing solver.prototxt file
  - Change net to be your net
  - Change napshot_prefix to your output
  - Reduce base learning rate (divide by 100)
  - Maybe change max_iter and snapshot

```
net: "models/bvlc_alexnet/train_val.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 100000
display: 20
max_iter: 450000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/bvlc_alexnet/caffe_alexnet_train"
solver_mode: GPU
```

No Need to Write Code

Step 4: Train

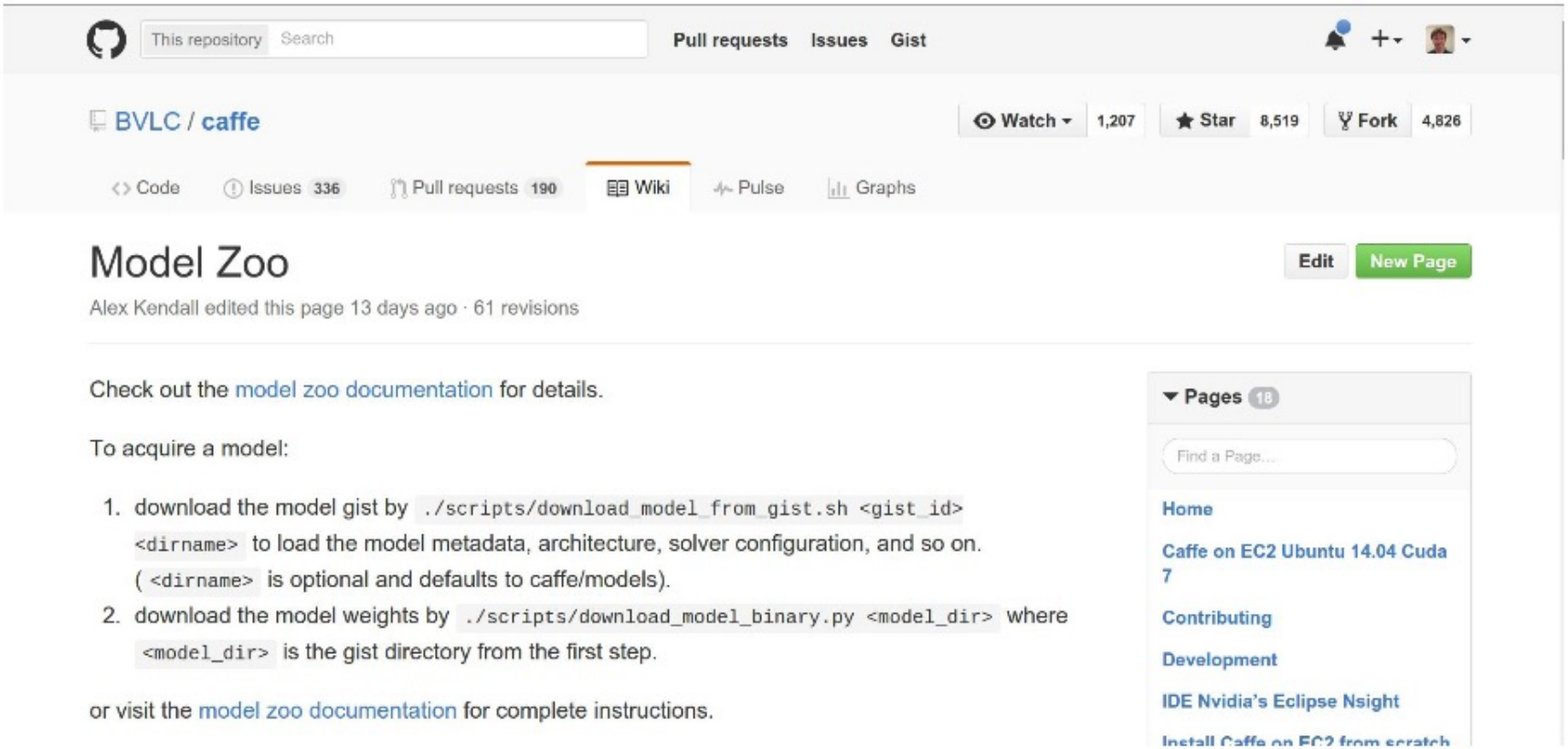
```
./build/tools/caffe train \
  -gpu 0 \
  -model path/to/trainval.prototxt \
  -solver path/to/solver.prototxt \
  -weights path/to/pretrained_weights.caffemodel
```

`-gpu -1` for CPU-only

`-gpu all` for multi-gpu

Caffe Model Zoo

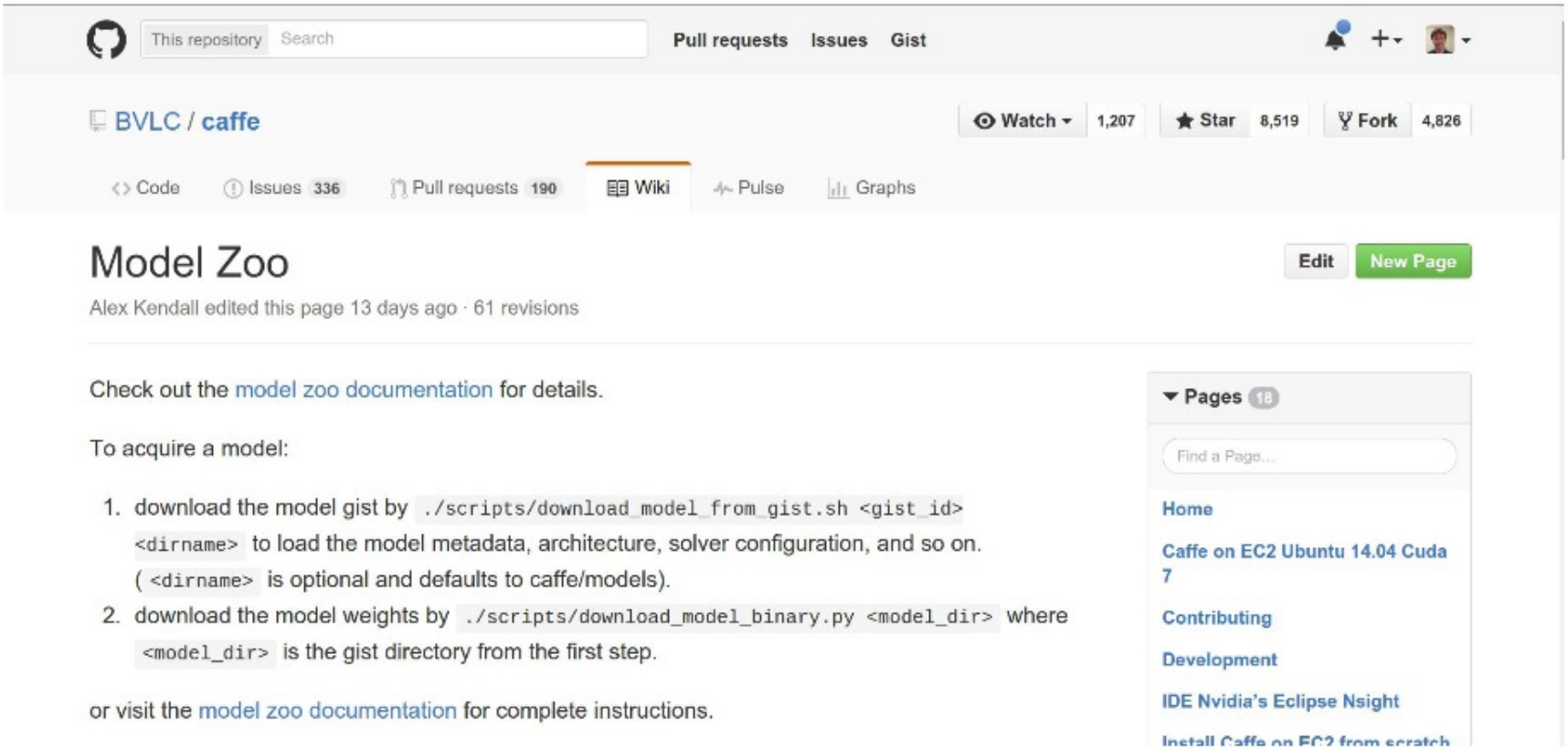
AlexNet, VGG, GoogLeNet, ResNet, plus others



https://github.com/BVLC/caffe/wiki/Model-Zoo

Caffe Model Zoo

AlexNet, VGG, GoogLeNet, ResNet, plus others



https://github.com/BVLC/caffe/wiki/Model-Zoo

Caffe Pros / Cons

(+) Good for feedforward networks

(+) Good for finetuning existing networks

(+) Train models without writing any code!

(+) Python interface is pretty useful!

(+) Can deploy without Python

(-) Need to write C++ / CUDA for new GPU layers

(-) Not good for recurrent networks

(-) Cumbersome for big networks (GoogLeNet, ResNet)

Summary

- **TensorFlow** is a safe bet for most projects, not perfect but wide usage, maybe pair with high-level wrapper (Keras, etc)
- **Theano** is already dated
- **PyTorch** is the best for research
- Use **TensorFlow** for one graph over many machines
- Consider **Caffe** or **TensorFlow** for production deployment
- Consider **TensorFlow** or **Caffe2** for mobile

References-CNN

1. **(Image Recognition) AlexNet** : Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012
2. **(Action Recognition)**: Shuiwang Ji, Wei Xu, Ming Yang, Kai Yu, 3D Convolutional Neural Networks for Human Action Recognition, ICML, 2010
3. **(Detection) R-CNN**: Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, CVPR, 2014
4. **(Detection) Faster R-CNN**: Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, arXiv:1506.01497
5. **(Segmentation) F-CNN**: Jonathan Long, Evan Shelhamer, Trevor Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR, 2015
6. **(Tracking)** Chao Ma, Jia-Bin Huang, Xiaokang Yang and Ming-Hsuan Yang, Hierarchical Convolutional Features for Visual Tracking, ICCV, 2015
7. **(Super-resolution)** Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang, Learning a Deep Convolutional Network for Image Super-Resolution, ECCV, 2014

References-CNN

8. **(Edge Detection)**: Gedas Bertasius, Jianbo Shi, Lorenzo Torresani, DeepEdge: A Multi-Scale Bifurcated Deep Network for Top-Down Contour Detection, CVPR, 2015

9. **(Face Recognition)**: Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf, DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR, 2014

10. **(Question Answering)**: Hyeonwoo Noh, Paul Hongsuck Seo, and Bohyung Han, Image Question Answering using Convolutional Neural Network with Dynamic Parameter Prediction, arXiv:1511.05765

11. **(Video Caption)**: Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, Trevor Darrell, Long-term Recurrent Convolutional Networks for Visual Recognition and Description, CVPR, 2015

12. **(Text Classification)**: Xiang Zhang, Junbo Zhao, Yann LeCun, Character-level Convolutional Networks for Text Classification, NIPS, 2015

13. **(Retrieval)**: Fang Zhao, Yongzhen Huang, Liang Wang, Tieniu Tan, Deep Semantic Ranking Based Hashing for Multi-Label Image Retrieval, CVPR, 2015

References-RNN

1. (**Object Recognition**) Ming Liang and Xiaolin Hu, Recurrent Convolutional Neural Network for Object Recognition, CVPR, 2015
2. (**Speech Recognition**) Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio, Attention-Based Models for Speech Recognition, NIPS, 2015
3. (**Machine Translation**) Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR, 2015
4. (**Question Answering**) Karl M. Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom, Teaching Machines to Read and Comprehend, NIPS, 2015
5. (**Scene Labeling**) Pedro Pinheiro and Ronan Collobert, Recurrent Convolutional Neural Networks for Scene Labeling, ICML, 2014
6. (**Tracking**) Quan Gan, Qipeng Guo, Zheng Zhang, and Kyunghyun Cho, First Step toward Model-Free, Anonymous Object Tracking with Recurrent Neural Networks, arXiv: 1511.06425
7. (**Segmentation**) Francesco Visin, Kyle Kastner, Aaron Courville, Yoshua Bengio, Matteo Matteucci, and Kyunghyun Cho, ReSeg: A Recurrent Neural Network for Object Segmentation, arXiv:1511.07053

References–RNN

8. (**Language Modeling**) Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, and Richard S. Zemel, Skip-Thought Vectors, NIPS, 2015

9. (**Image Caption**) Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention, ICML, 2015

10. (**Video Caption**) Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell, Long-term Recurrent Convolutional Networks for Visual Recognition and Description, CVPR, 2015

11. (**Question Answering**) Mengye Ren, Ryan Kiros, and Richard Zemel, Exploring Models and Data for Image Question Answering, NIPS, 2015

12. (**Robotics**) Hongyuan Mei, Mohit Bansal, and Matthew R. Walter, Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences, arXiv: 1506.04089

13. (**Video Super-resolution**) Yan Huang, Wei Wang, and Liang Wang, Bidirectional Recurrent Convolutional Networks for Multi-Frame Super-Resolution, NIPS, 2015

Acknowledgement

Some of the materials in these slides are from:

- Shubhendu Trivedi and Risi Kondor, University of Chicago, Deep Learning Course
- Hung-yi Lee, National Taiwan University, Machine Learning and having it Deep and Structured course
- Xiaogang Wang, The Chinese University of Hong Kong, Deep Learning Course
- Fei-Fei Li, Standord University, CS231n Convolutional Neural Networks for Visual Recognition course
- Thomas Kipf, University of Cambridge, CompBio Seminar
- Semi-Supervised Classification with Graph Convolutional Networks. T. N. Kipf, M. Welling, ICLR 2017
- Deep Learning for Network Biology : snap.stanford.edu/deepnetbio-ismb
- Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Questions?

Thank You !