Module 13: Efficient DL and Cases with Complexity Issues

5LSM0: Convolutional neural networks for computer vision

Sander Klomp and Peter H.N. de With

Electrical Engineering / VCA research group

Module outline

1. Introduction and Motivation

Use case 1.1: Autonomous Driving (why we need Efficient Deep Learning)

- Overview, Perception and its challenges

Use case 1.2: Healthcare: early cancer detection in VLE signals

- Sensing principle and example solution concept

2. Efficient Deep Learning

- About "efficiency"
- Small neural networks
- Finding the right architecture
- Deep compression

Use Case 1: Autonomous Driving

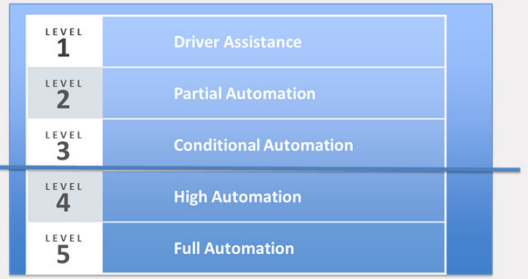
Overview

Source: These slides are adapted from Forrest Landola's keynote, EI2019, Burlingame, 2019

Autonomous driving: What is it?

for example,
PASSENGER CARS

for example,
ROBO-TAXIS



Where are we now?

Driver Assist versus Autonomous Vehicles

Level 1-3: Driver assistance

L1: Individual assistance functions

- Automatic emergency braking
- Lane keeping, lane assist
- Parking assist

L2: combining them

L3: vehicle takes over driving functions, but driver must be ready to take over

This can already benefit from CNNs!

Level 4 – Level 5 Autonomy

Levels of Full Autonomy

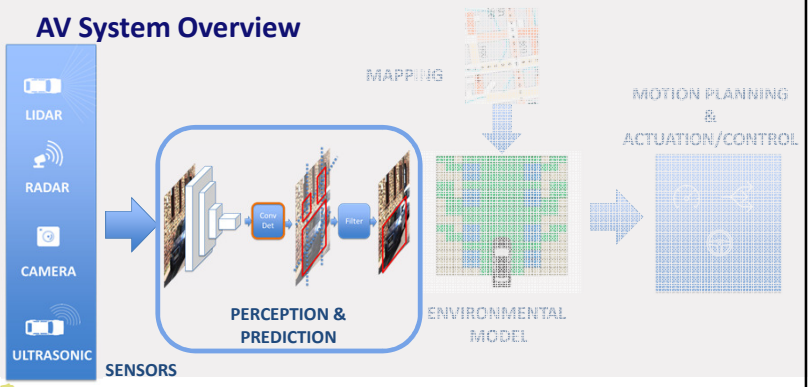
Level 4

- Full autonomy in constrained situations

Level 5

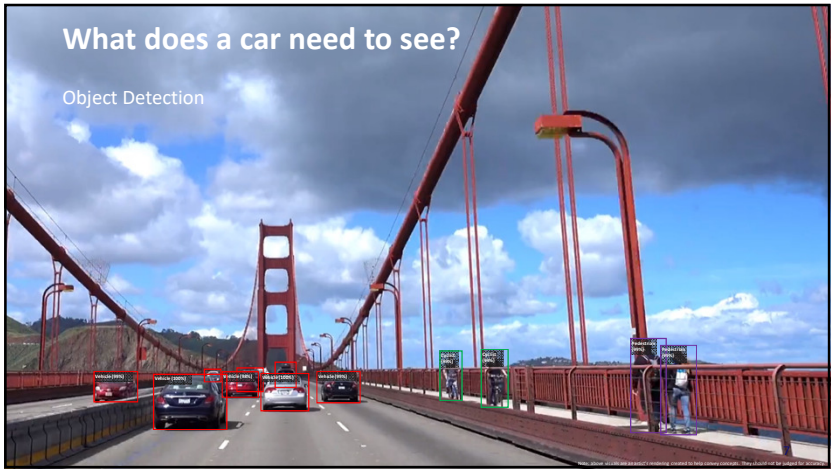
- Fully autonomous in all situations

AV System Overview



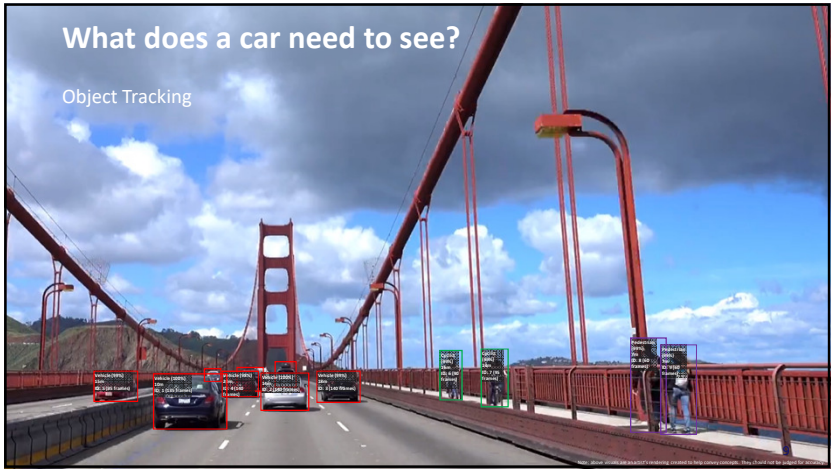
What does a car need to see?

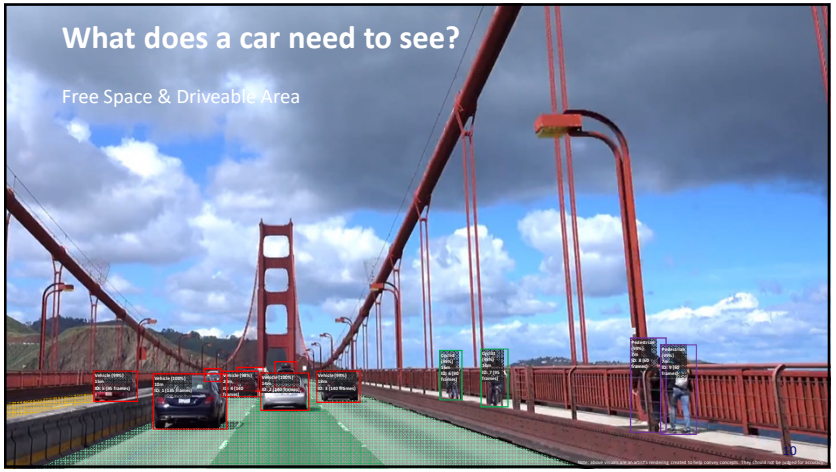
Object Detection

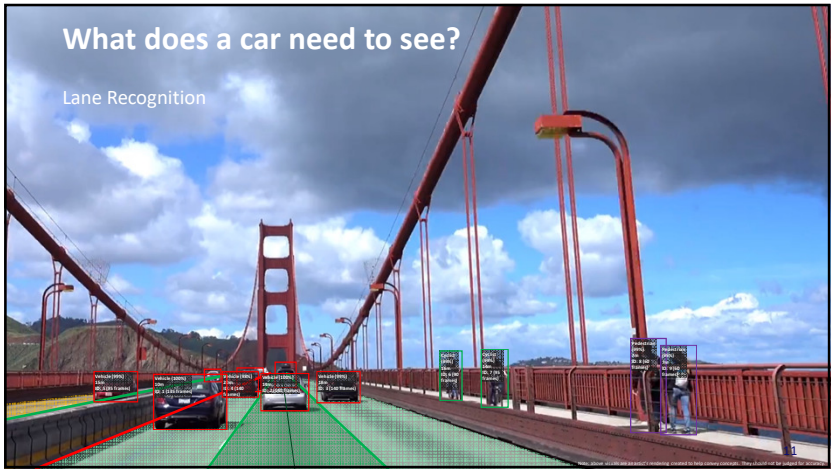


What does a car need to see?

Distance





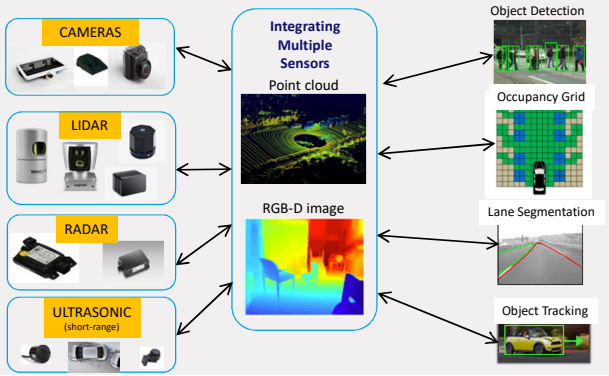


How does a car see? – A lot of cameras (and other sensors)

Sensors

- Cameras
- LiDAR
- Radar
- Ultrasonic

Applying DNN to Integrated Sensor Data



P. Tomasello, et al. DSCnet: Replicating Lidar Point Clouds with Deep Sensor Cloning. arXiv, 2018.

Challenges

Solving just *perception* already results in significant challenges

- Getting enough training data
  - Generating useful data through simulation
  - Domain adaptation of simulation data to the real world
  - Utilizing sensor fusion (LiDAR, RADAR etc.)
- Accelerating training to cope with all this new data
- Handling high resolution images
- Getting more out of video (adding the temporal domain)
- Power and energy efficient nets
- Achieving 'efficient' inference

Challenges

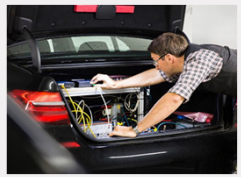
Solving just *perception* already results in significant challenges

- Getting enough training data
  - Generating useful data through simulation
  - Domain adaptation of simulation data to the real world
  - Utilizing sensor fusion (LiDAR, RADAR etc.)
- Accelerating training to cope with all this new data
- Handling high resolution images
- Getting more out of video (adding the temporal domain)
- Power and energy efficient nets
- **Achieving 'efficient' inference** Today's focus

Efficiency challenge: Computational cost

Perception is the most computationally intensive part of the software!

Audi
https://www.slashgear.com/man-vs-machine-my-rematch-against-audis-new-self-driving-rs-7-21415540/



BMW + Intel
https://newsroom.intel.com/news-releases/bmw-group-intel-mobileye-will-autonomous-test-vehicles-roads-second-half-2017/

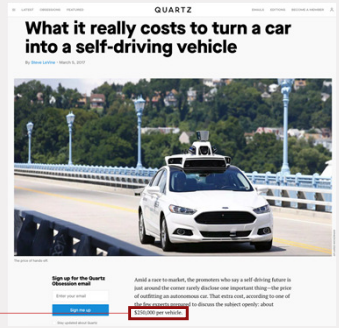
Ford
http://cwc.ucsd.edu/content/connected-cars-long-road-autonomous-vehicles

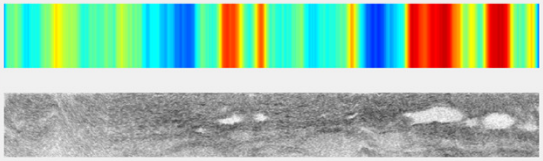
Computation is expensive

- $250.000 to make a vehicle autonomous
- Tens of thousands due to computational power required

In short:

- We want to process a huge amount of data
- "Server in the trunk" is not desirable
- **We need efficient CNN architectures!**





Use Case 2: Healthcare / Early detection of cancer with VLE

Overview

Source: Joost van der Putten, Slides adapted from PhD project on VLE learning 2019

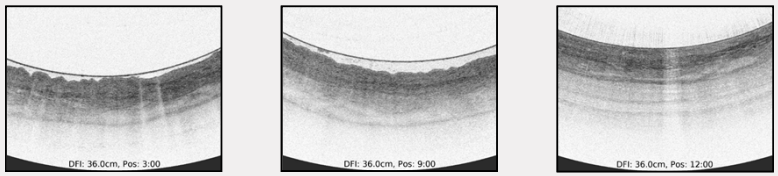
Volumetric Laser Endomicroscopy (VLE) Signal /

Algorithm for single-frame VLE snapshot classification

Prospectively gathered set of 111 snapshots (18 patients)

First focus on High-Grade Dysplasia (HGD) vs. Non-Dysplastic Baretts Esophagus (NDBE)

- **25 HGD**
- **86 NDBE**



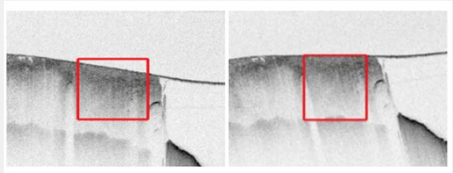
VLS Signal / Region of interest (ROI) segmentation

Previously

Cropping from whole image.

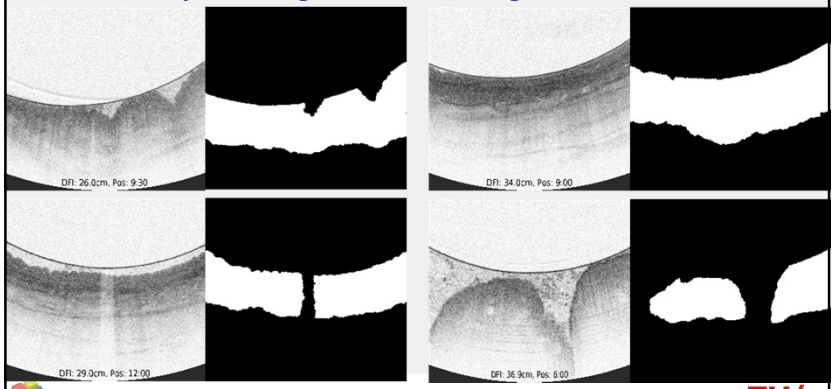
Rotate image for optimal usage of area.

Labor intensive and will never incorporate all available data.



Idea: Algorithm for automatic ROI (tissue) segmentation

VLE examples and gold standard segmentations

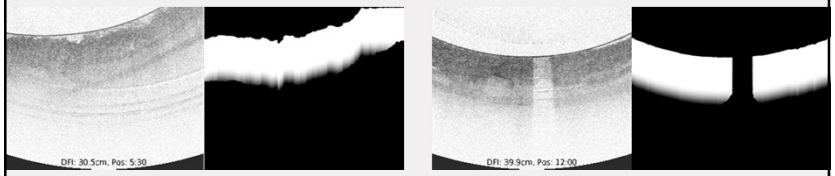


VLE Signal / Undefined lower boundary in various scans

Lower regions of the image have less signal

Ground-truth depth is somewhat arbitrary, however, **upper boundary is clear**

From previous work: most valuable information is found in **approximately the top 200 pixels** of the tissue



VLE segmentation / U-net and custom loss function

U-net is a widely used CNN for medical segmentation problems (Ronneberger *et al.*, 2015)

Multi-scale approach

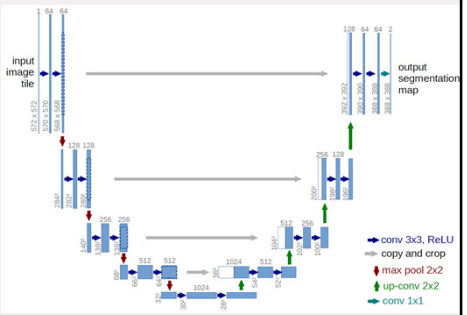
End-to-end learning

A custom loss function:

Less penalty for misclassifying lower regions

Normal penalty for the top 200 pixels

Harsh penalty for misclassifying area immediately above ROI



VLE Signal / ROI segmentation results

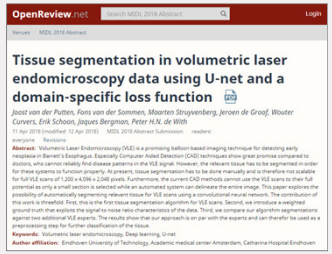
Compared results to gold standard segmentations of three assessors

Quantitatively compared segmentations assessor segmentations

System *within* inter-observer variability

Submitted results to MIDL 2018, Amsterdam

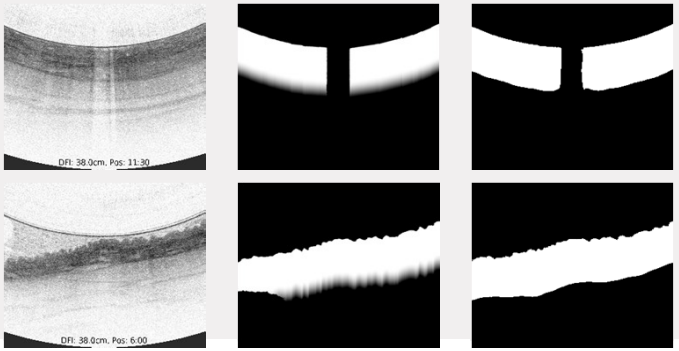
DICE scores	System vs. human		Human vs. human	
	Basic model	**Weighted model**	**Assessor 2**	**Assessor 3**
Assessor 1	0.95	0.97	0.96	0.96
Assessor 2	0.95	0.97	-	0.96
Assessor 3	0.95	0.97	-	-


OpenReview.net

Tissue segmentation in volumetric laser endomicroscopy data using U-net and a domain-specific loss function

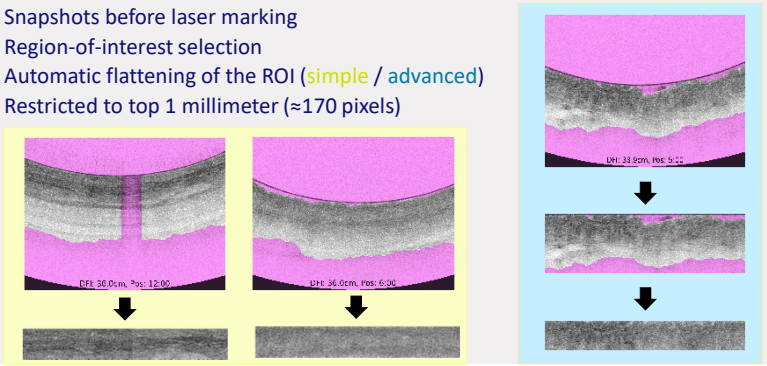
VLE Signal / ROI segmentation – Visual results

Snapshot | Weighted gold standard | System prediction



Data pre-processing

Snapshots before laser marking
Region-of-interest selection
Automatic flattening of the ROI (simple / advanced)
Restricted to top 1 millimeter (≈170 pixels)



Conventional CAD methods - 1

*F. van der Sommen et al. "Predictive features for early cancer detection in Barrett's esophagus using volumetric laser endomicroscopy." *Computerized Medical Imaging and Graphics*(2018).

Evaluated most promising features from earlier ex-vivo work*

Optimal hyperparameters also from ex-vivo experiments*

Various widely-used (conventional) classification methods

SVM, Random Forest, AdaBoost, Naive Bayes, etc.

Three validation experiments

Leave-one-out cross-validation (LOOCV) on unbalanced set (25 HGD vs 86 NDBE)

4-fold cross-validation (4-fold CV) on unbalanced set (25 HGD vs 86NDBE)

(and other tests beyond the scope here)

Conventional – 2 / LOOCV on unbalanced data set

Relatively low sensitivity due to low number of positives (#pos)
Operating point can be changed by changing the cut-off value / threshold

		Default operating points	
	AUC	Sensitivity	Specificity
Linear SVM	93.7	52.0	98.8
Random Forest	92.0	68.0	95.4
K-Nearest Neighbors	90.7	40.0	97.7
Naive Bayes	**93.4**	**88.0**	**82.6**
Discriminant Analysis	89.7	72.0	83.7
Non-linear SVM	85.2	36.0	97.7
Neural Network	89.0	64.0	90.1
Adaptive Boosting	89.3	72.0	89.5

Conventional – 3 / 4-fold Cross-Validation on unbalanced data set

- Less data: slightly lower scores and higher variability
- Largest effect on sensitivity (very low #pos)

		Default operating points	
	AUC	Sensitivity	Specificity
Linear SVM	91.2	52.0	97.7
Random Forest	94.8	0.00	1.00
K-Nearest Neighbors	92.8	32.0	84.6
Naive Bayes	**92.0**	**84.0**	**81.4**
Discriminant Analysis	87.7	68.0	84.9
Non-linear SVM	85.8	36.0	98.8
Neural Network	89.2	60.0	94.1
Adaptive Boosting	90.1	64.0	93.0

End-to-end Deep Learning for VLE classification

Useful information is found mainly vertically

Surface intensity

Layering

Calculate a dysplasia score for each A-line

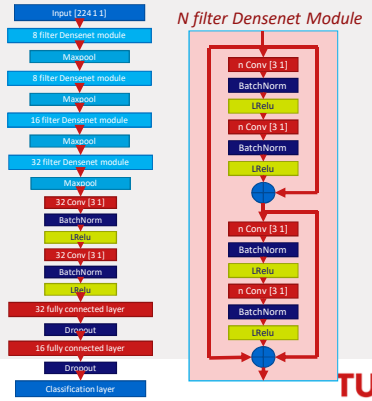
Sufficient data for end-to-end learning!

Convolutional neural network with 1-D filters

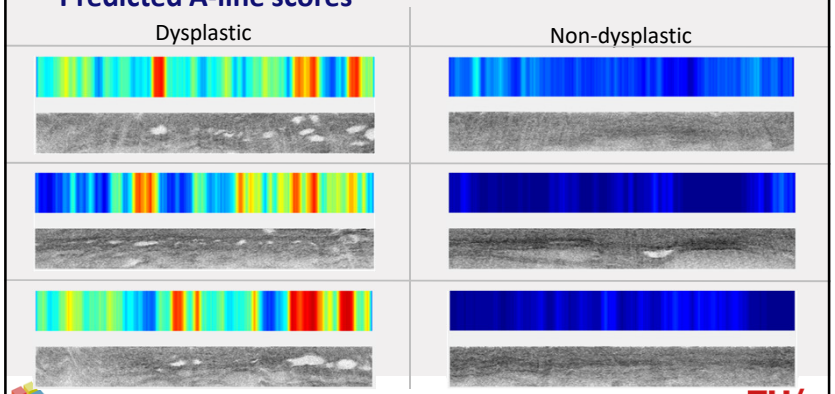
DenseNet (Huang *et al.*, 2017)

Reduced number of parameters!

Among other benefits..



Predicted A-line scores



Healthcare Use Case 2 / Early cancer detection take aways...

Considering the OCT signal characteristics: noisy and buried patterns

Accurate segmentation required for finding the ROI suitable signal

Deep Learning compares to the best-scoring conventional methods

But:….adapt method to the nature of the scanning to reduce complexity!

Post-processing is also required, this is not discussed here …

About computational cost

Hardware evolution

The computation problem

- Won't rapid GPU advancements just solve the computation problem for us?

Platform	Computation (TOP/s)	Year
NVIDIA K20 [1]	3.50 (32-bit float)	2012
NVIDIA V100 [2]	112 (16-bit float)	2018
Next-gen: 20 TOP/W	2500*	2020 (est.)

* Assuming half the power is spent on computation, and the other half is spent on memory and other devices.

[1] https://www.nvidia.com/content/PDF/kepler/Tesla-K20-Passive-BD-06455-001-v05.pdf
[2] http://www.nvidia.com/content/PDF/Volta-Datasheet.pdf (PCIe version)

The computation problem

- Won't rapid GPU advancements just solve the computation problem for us?

Platform	Computation (TOP/s)	Memory Bandwidth (TB/s)	Computation-to-bandwidth ratio	Year
NVIDIA K20 [1]	3.50 (32-bit float)	0.208 (GDDR5)	17	2012
NVIDIA V100 [2]	112 (16-bit float)	0.900 (HBM2)	124	2018
Next-gen: 20 TOP/W	2500*	1.800 (HBM3)[3]	1389	2020 (est.)

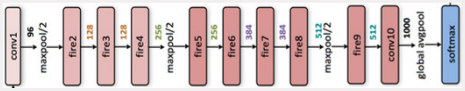
- **No!**
- A new bottleneck: memory
- Defining "efficient" as only computations is not enough

[1] https://www.nvidia.com/content/PDF/kepler/Tesla-K20-Passive-BD-06455-001-v05.pdf
[2] http://www.nvidia.com/content/PDF/Volta-Datasheet.pdf (PCIe version)
[3] https://www.eteknix.com/gddr6-hbm3-details-emerge/

In case you still aren't convinced

- Besides future-proofing against the memory bottleneck…

- Small neural networks:
  - are more feasible for embedded implementations
  - provide freedom from cloud servers (privacy, low latency)
  - allow quick over-the-air updates for mobile devices
  - train faster in distributed training scenarios (that have a communication bottleneck)
  - reduce energy cost



Small Neural Networks

And the definition of efficiency

Redefining efficiency

squeeze *(verb)*: to make an AI system use less resources using whatever means necessary

Redefining efficiency

Memory Footprint and Bandwidth | Computational Operations | Power and Energy | Time

squeeze *(verb)*: to make an AI system use less resources using whatever means necessary

Redefining efficiency

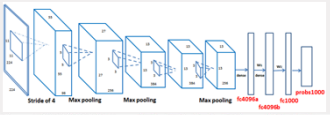
Memory Footprint and Bandwidth | Computational Operations | Power and Energy | Time

squeeze *(verb)*: to make an AI system use less resources using whatever means necessary

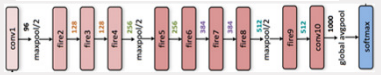
New DNN Models | Application-specific Quantization and Pruning | Superior Implementations | Differentiated Data and Training Strategies

Squeezenet preview

AlexNet [1]



SqueezeNet [2]



CNN	Top-5 Accuracy ImageNet	Model Parameters	Model Size
AlexNet[1]	80.3%	60M	243MB
SqueezeNet[2]	80.3%	1.2M	4.8MB

compresses to 500KB

- Model size: 500x compressed
- Could even fit in L2 cache instead of RAM

Finding the right architecture

Slides based mostly on the paper:
Small Neural Nets Are Beautiful: Enabling Embedded Systems with Small Deep Neural Network Architectures by F. Iandola and K. Keutzer

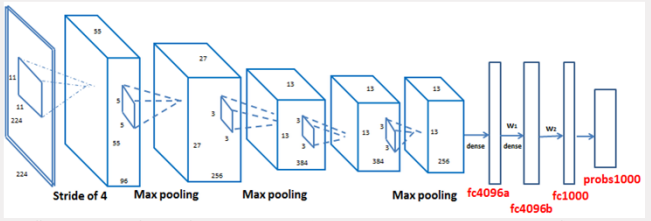
A quick recap on important CNN terms

- Layer: a function applied to its input. Parameters may or may not be learned by training
  - E.g. convolution, activation function, interpolation, pooling, fully connected
- Convolutional filter dimension: filters have a spatial size (width x height) and a depth (usually equal to the input channels)
- Activation map: The output tensor of a layer, for CNNs usually: width x height x channels

What can we do to make this network smaller?

Where "smaller" means: takes less memory to save the parameters

(For now, let's ignore whether or not it will mess up the performance…)



https://www.researchgate.net/publication/300412100_Deep_Learning_for_Image_Retrieval_What_Works_and_What_Doesn%27t/figures?lo=1

A quick recap on important CNN terms

- Layer: a function applied to its input. Parameters may or may not be learned by training
  - E.g. convolution, activation function, interpolation, pooling, fully connected
- Convolutional filter dimension: filters have a spatial size (width x height) and a depth (usually equal to the input channels)
- Activations: The output tensors of a layer, for CNNs usually: width x height x channels

Network efficiency boosters

- Layers:
  - Replacing fully connected layers with convolution layers
- Convolution filters have a spatial size and a depth
  - Depthwise convolutions and shuffle operations
  - Filter stacking and spatial kernel reduction
- Activations: width x height x channels
  - Evenly-spaced downsampling
  - Channel reduction
- Lower-level optimizations
  - Deep compression

CNN Architecture Search: ≥50x

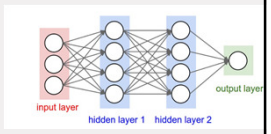
Can also do both

Model Compression: 10x

Images: Prof. Warren Gross (McGill Univ.)

Replacing fully connected layers by convolutions

- Fully connected (FC) layers often contain a LOT of parameters
  - Recap exercise: AlexNet and VGG both contain an FC layer of
    - *Input channels = 4096*
    - *Output channels = 4096*
    - *Width x Height = 1x1*
    - *How many parameters does this layer take?* 4096x4096 ~ 17M



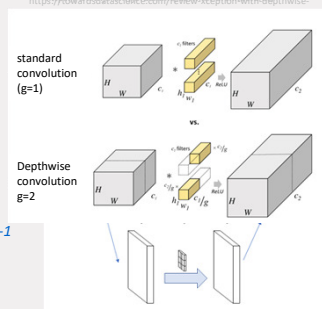
- Large channel counts are often needed for good fully-connected layer performance
- Instead just use a few convolutions on fewer channels earlier in the network
  - *An example from SqueezeNet: How many parameters does a 3x3 convolution have for 512 input channels and 64 output channels on a 14x14 activation map?* 3x3x512x64 ~ 30k
  - Only a few of these layers will already compensate for removing the above FC layer!

Changing convolution filters:

Depthwise Convolution

Apparently #channels is a problem.
So why not just use part of the channels per filter?

- **"Depthwise convolution"**
  (or "group convolutions" or "filter groups")
- Adds additional dimension g, where each filter has C/g channels (C=input channels)
  - *E.g. if $g = 2$ half of the filters are applied to channels 0 to $C/2$-1 and the other half to $C/2$ to C*
  - *The parameters saved factor is maximal for $g = C$ (1 filter per channel, saves a factor C in parameters)*



Example where g=C

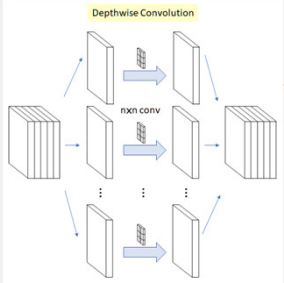
Problem of depthwise convolution?

Changing convolution filters:

Depthwise Convolution

- Problem: no information exchange over channels!
  - *Worst case: entire network of depthwise convolutions. This acts like multiple completely independent networks*
- Two possible solutions:
  - *Place normal 1x1 convolution afterwards to combine over channels → Introduces some parameters again, but good performance*
  - *Add random shuffle operations after every few layers → 0 new learned parameters, but worse performance*

https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568



Example where $g=C$

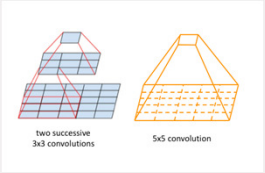
Changing convolution filters:

Stacking smaller filters

What you have already seen: multiple smaller convolutions achieve same receptive field as a larger one

- Slightly fewer parameters
- Often even improved performance

https://www.jeremyjordan.me/convnet-architectures/



Filter stacking

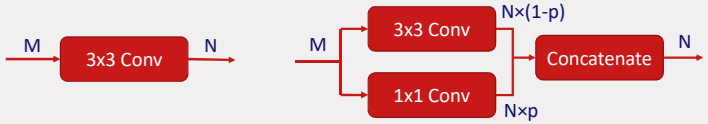
Why does stacking smaller filters save parameters?

Changing convolution filters:

Spatial kernel reduction

Better yet: not all convolutions need to be of size ≥3x3.

- Replace up to factor p of 3x3 convolutions by 1x1 convolutions
- Up to P = 50% of 3x3 convolution filters may be replaced by 1x1 without performance loss!
  - *Saves memory up to factor 3x3/(3x3*0.5+1x1*0.5) = 1.8*

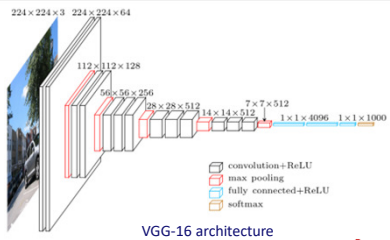


What this looks like for N filters (on an input map of M channels)

Managing Activation Map sizes

- Downsampling
  - Reduce height and width, but usually increase depth
  - No impact on number of parameters for convolution layers
  - **"Evenly-spaced downsampling"** is a good default
    - *Downsampling too early hurts accuracy*
    - *Downsampling too late increases computational cost*

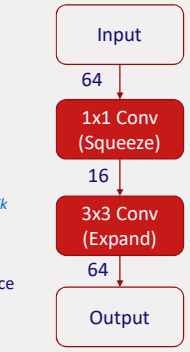
https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/



VGG-16 architecture

Managing Activation Map sizes

- "Channel reduction" (or "bottleneck layers")
  - Squeeze input into fewer channels using 1x1 convolution
  - Then follow with the 3x3 convolution to expand again
  - Why does this save parameters?
    - *Example: 3x3 convolution with 64 in/output channels = 3*3*64*64 = 37k*
    - *Example architecture on the right: 1*1*64*16 + 3*3*16*64 = 10k*
- This turns out to have only a minor negative effect on performance



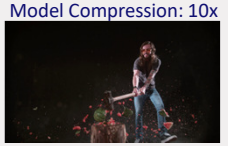
Simplified bottleneck structure

Remember the memory gain:

CNN Architecture Search: ≥50x

What we have done so far

Model Compression: 10x



Up next (very briefly)

Images: Prof. Warren Gross (McGill Univ.)

Model Compression

Slides based on the paper:
Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding by Han et al.

Deep compression

- What can we do when we have already optimized our architecture?
  → Optimize at a lower level
- Some common options
  - Weight quantization
  - Pruning
  - Weight coding

Weight quantization

- Normally weights are 32-bit floating point numbers
  - This is often more accurate than needed
- Smaller precision Reduces both memory AND computation time significantly!
  - 16-bit floating point is an easy 50% reduction
  - Taking it to the extreme: binary networks. 32x memory savings, up to 60x faster on IC's
    - *Although GPUs do not have the hardware to exploit this…*

1.2367…	0.7623…	0.0285…
2.1209…	0.0184…	0.0395…
0.0195…	0.9664…	0.0218…

32-bit floating point
overkill

1.23	0.762	0.0285
2.12	0.0184	0.0395
0.0195	0.966	0.0218

16-bit floating point
probably good enough

1	1	0
1	0	0
0	1	0

1-bit binary value
loss of performance

For illustration purposes. This isn't truly what fp16 looks like

Pruning and Huffman coding

- Set convolution weights that are close to zero exactly to zero, and "freeze" them
- Retrain the network with its remaining weights
  - Generally this recovers all accuracy of the original network
- In AlexNet and VGG this results in about 90% of the weights being set to 0!
- Huffman coding can efficiently store weights that are non-uniformly distributed

1.23	0.76	0.02
2.12	0.01	0.03
0.01	0.96	0.02

3x3 convolution weights

1.23	0.76	0
2.12	0	0
0	0.96	0

Pruned and frozen

1.19	0.74	0
2.14	0	0
0	1.01	0

Retrained

Summary

- **Cases**
  1. Autonomous driving: Showed the importance of real-time computation
  2. Healthcare: Reduce complexity by exploiting image acquisition method
- **Designing memory-efficient deep convolutional architectures**
  - Replacing fully connected layers by convolutions
  - Depth-wise convolutions
  - Stacking filters and spatial kernel reduction
  - Evenly spaced down-sampling
  - Channel reduction
- **Deep compression: compressing convolution weights**
  - Pruning + Huffman coding
  - Weight quantization