Design

Arthur J. Redfern
axr180074@utdallas.edu
Sep 17, 2018
Sep 19, 2018
Sep 24, 2018

Outline

- Motivation
- Goal
- Preliminaries
- Strategy
- Layers
- Networks
- Visualization
- References

Motivation

The Previous Lectures Covered A Lot

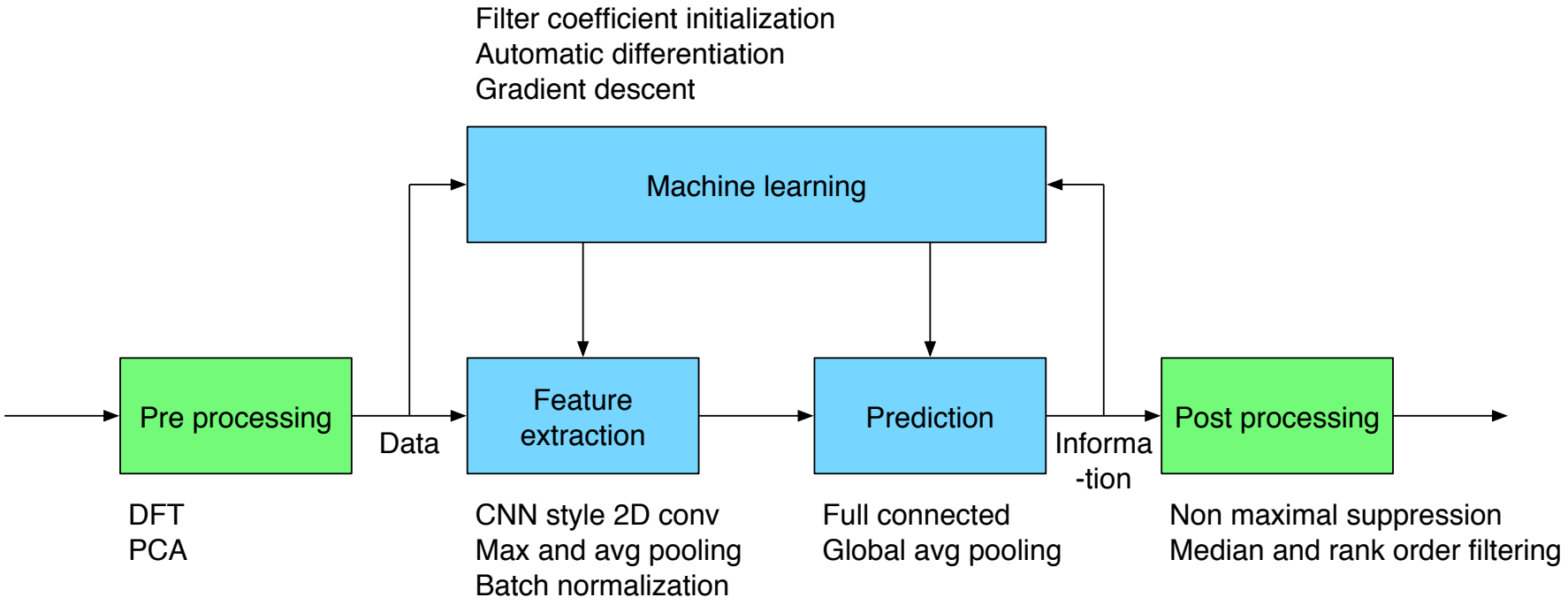
- We've briefly looked at a lot of different parts of basic math
  - 1 intro lecture where a framework for information extraction was introduced including pre processing, feature extraction, prediction and post processing
  - 2 lectures on linear algebra covering sets, fields, vectors, matrices, tensors, functions, vector spaces, normed vector spaces, inner product spaces, matrix vector multiplication, matrix matrix multiplication, CNN style 2D convolution, DFTs and PCA
  - 2 lectures on calculus covering derivatives, sub derivatives, partial derivatives, gradients, Jacobians, chain rule, critical points, gradient descent, automatic differentiation with reverse more accumulation and universal approximation
  - 2 lectures on probability covering probability spaces, events, random variables, expected value, normalization, law of large numbers, central limit theorem, random processes, stationarity, time averages, ergodicity, entropy, mutual information, Kullback Leibler divergence, data processing inequality, compression, Huffman coding and arithmetic coding
  - 1/2 of a lecture on algorithms covering comparison sorts, sequential merge sort, parallel merge sort, pooling layers, median and rank order filtering and non maximal suppression

~ 30 lectures / semester, 7.5 = 25% allocated to math; remaining ~ 25% CNNs, ~ 25% implementation, ~ 25% application; minus tests and projects

Now We Start To Put The Pieces Together

- Introduction: flow of pre processing, information extraction, prediction and post processing
- Linear algebra: CNN style 2D convolutional layers, fully connected layers, pre processing methods
- Algorithms: pooling layers, post processing methods
- Probability: initialization, information in feature maps and filter coefficients, batch normalization, error functions
- Calculus: filter coefficient estimation, approximation

Now We Start To Put The Pieces Together



But Realize That There's (A Lot) More

- A basic amount of material from linear algebra, calculus and probability was needed such everything hangs together

- But it's possible to go much deeper and broader in each of these topics
  - We'll do some of that selectively in subsequent lectures
  - And you're well setup to do more in the future based on your own interests
  - Starting from a strong base makes learning new material a whole lot better

- Summary: the diagram on the previous slide is a nice start but there's a lot more for us to consider together and for you to consider individually

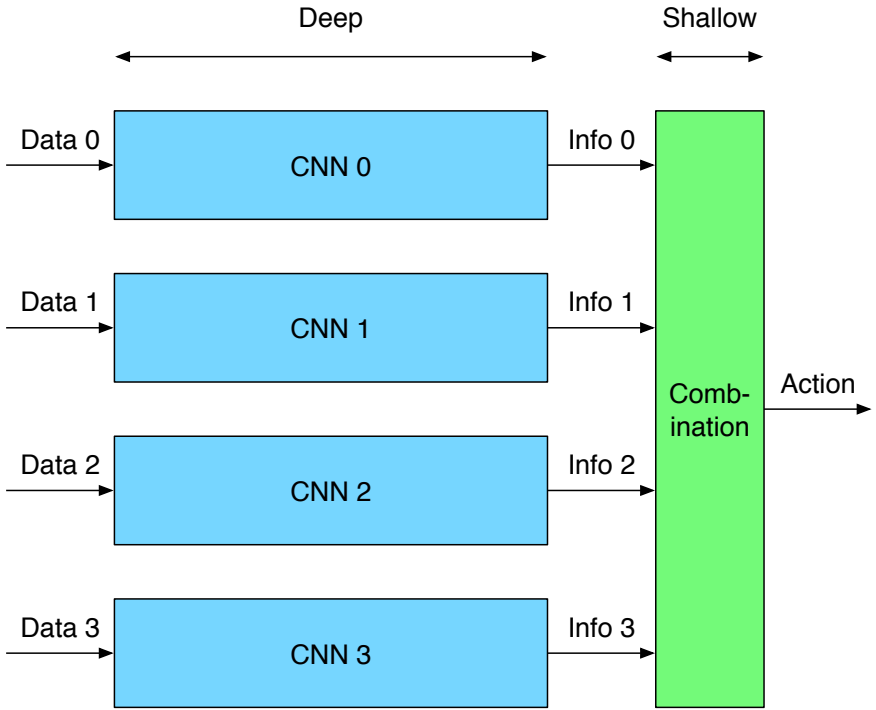
Goal

Keep Your Goals Simple

Both a deep network and general life comment

- You design a network to accomplish a goal
  - Never lose sight of this
- Keep the problems you address using deep networks like CNNs simple
  - Deep networks work best when there's a whole lot of labeled data to train on
  - The simpler the task the more data you have related to that task
  - A simple combinatorics argument illustrates this
- Handle complex problems via a shallow combination of simple problems that were solved with deep networks
  - A smaller shallow structure likely requires less data to train
  - A CNN may or may not be used for this

Keep Your Goals Simple



Example: Crossing A Busy Street

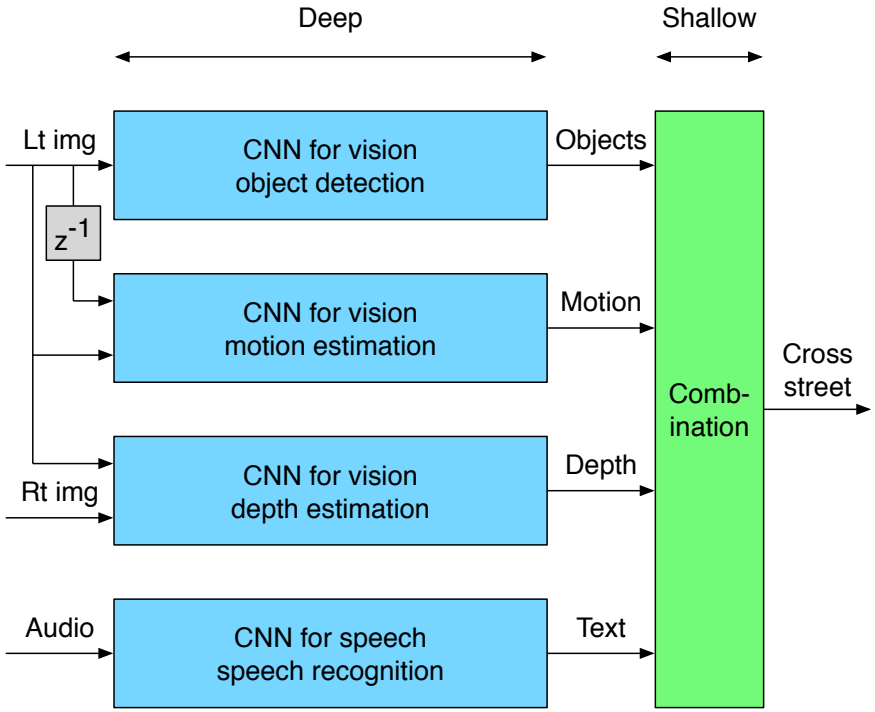
- Goal: get from 1 side of the street to the other side in a safe, efficient and law abiding way

- How do you solve this problem as a human? An incomplete list includes ...
  - Pre processing — look up from mobile phone
  - Vision — recognition of many objects in the scene, approximate distances, approximate motion
  - Sound — recognition of many sounds in the scene, approximate localization
  - Modeling — prediction of object paths vs desirable object separation for different scenarios
  - Risk analysis — from a personal health and legal perspective
  - Action — decide whether to start crossing via some trajectory at some speed looking some way
  - Repeat
  - Post processing — look down at mobile phone

https://www.freeimageslive.co.uk/files/images008/shibuya_busy_people.jpg

Example: Crossing A Busy Street

- How do you solve this problem as a computer?
  - Use CNNs to solve “simple” vision and speech problems, maybe part of the modeling prediction problem, maybe part of the action selection problem
  - Probably use other methods for risk analysis
  - Probably put the individual pieces together with a shallow structure



This Semester And This Class

- This semester will focus on small simple problems solved via CNNs
  - It's not that solving complex problems with shallow structures is not important (it is)
  - It's just that simple problems are a pre requisite and / or subset of complex problems
  - Simple problems will also be sufficiently difficult to solve that we won't get too bored

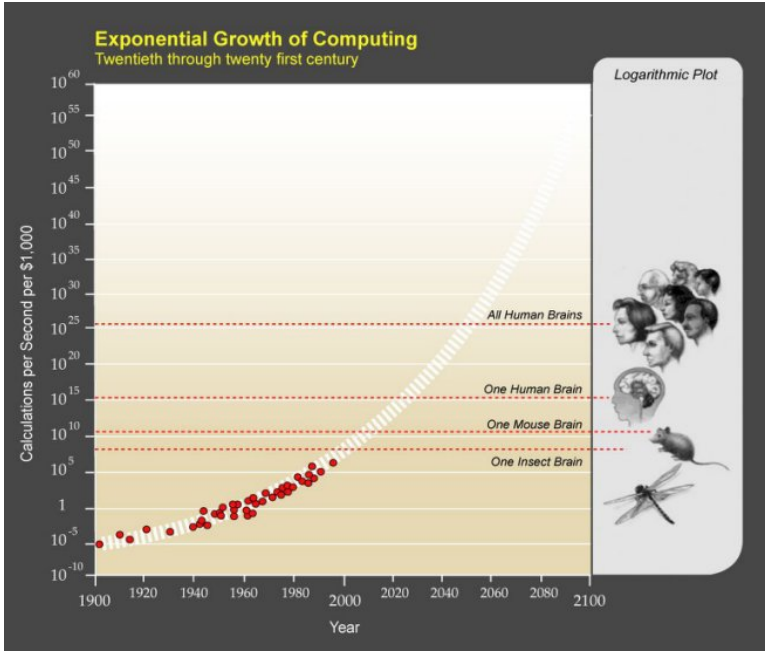
- It's nice to have a default problem to think about new ideas in the context of
  - Will use image classification as our starting point in this lecture
  - Will later generalize in a natural way to more problems in subsequent lectures on applications

Sound Like A Good Plan?

Preliminaries

Inspiration

- It's ok to look to nature for inspiration for basic principles that work
- But don't get too hung up on biology and replicating brain structure
  - If you believe in physics, the brain is an existence proof of what can be done in 3 pounds of material and 20 W of power (20 % of an average human's 100 W power budget)
  - If you believe in evolution, why build replica of the current human brain, why not build the better brain that people will have 1M years from now?
- Why have aliens never contacted us?



Theory Or Practice?

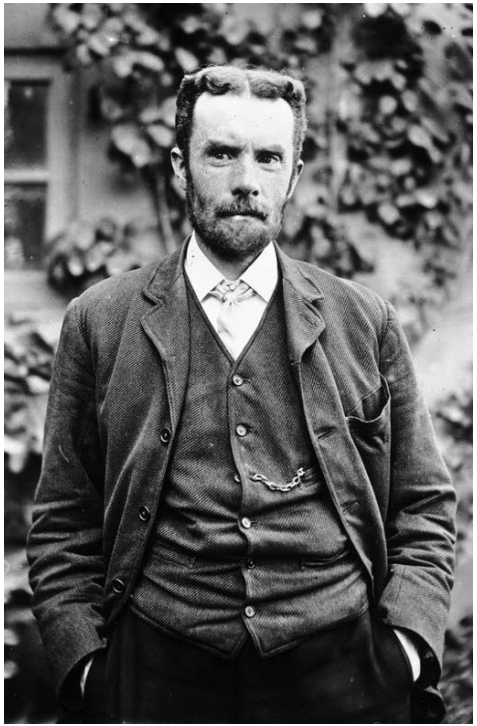
- Both (and that's ok, it's how science works)

- A frequent critique of deep learning and CNNs is that no one knows why they work
  - This is way too general a statement
  - This is not correct
  - There are strong theoretical underpinnings and frameworks for understanding what's going on
  - Is every single detail known about every single thing? No. But what scientific field would answer yes?
  - We've seen some theory, realize that much more exists

- There's a place in science for experimentation and trying new things
  - It's a reasonable way to make progress
  - Experimentation works best when it's guided by a combination of theory and intuition
  - Successes (→ practice) are nice in that they allow focused work on theory to explain

Oliver Heavyside

- "Mathematics is of two kinds, Rigorous and Physical. The former is Narrow: the latter Bold and Broad. To have to stop to formulate rigorous demonstrations would put a stop to most physico-mathematical inquiries. **Am I to refuse to eat because I do not fully understand the mechanism of digestion?**"



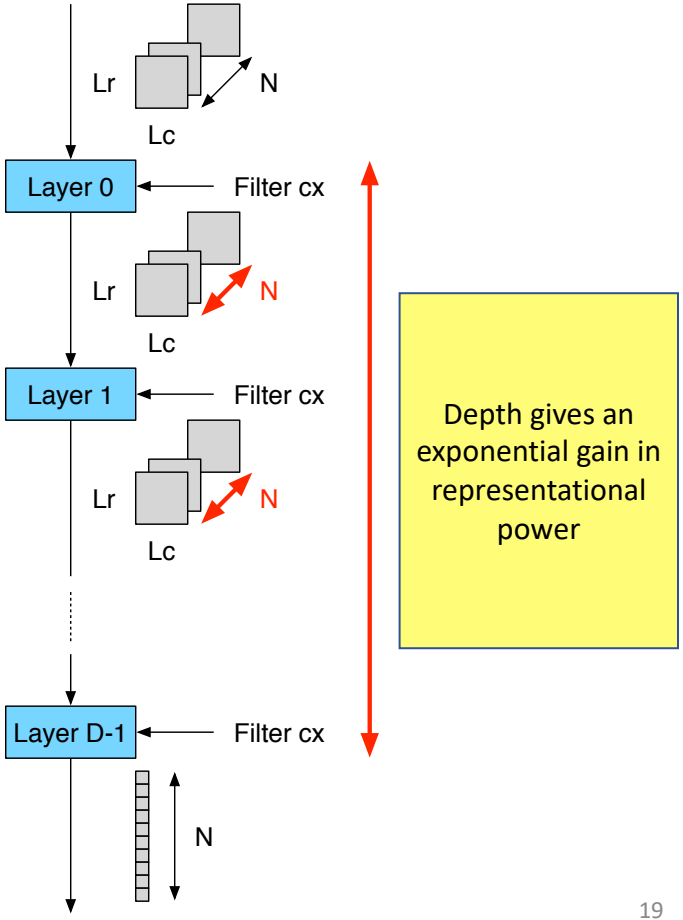
https://en.wikipedia.org/wiki/Oliver_Heaviside#/media/File:Oheaviside.jpg

Quotes != proofs, but regardless, I like this 1

Network Size

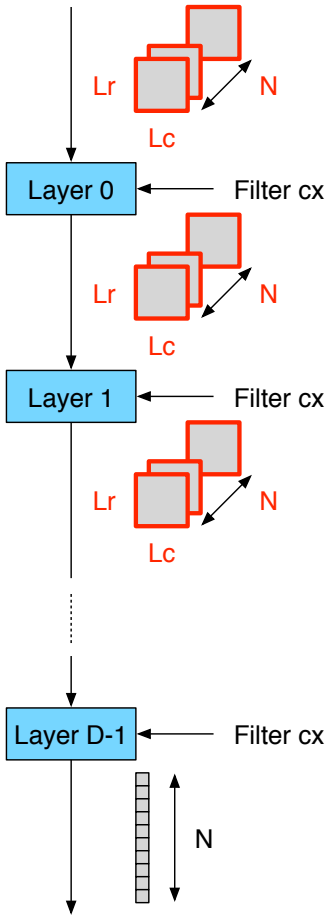
- Network depth and breadth (D, $N_{i/o}$)
  - Deeper and wider networks can approximate more complex functions
  - Remember the universal approximation proof from the calculus lecture
  - Deeper and wider networks are enabled by more data and better hardware
  - Do you want a smaller brain or bigger brain?
  - How do you sell a house?

- Performance
  - A drawback of a deeper and wider network is increased complexity / reduced performance
  - Will look at this in detail later
  - For now, pay attention to compute at the beginning and memory at the end



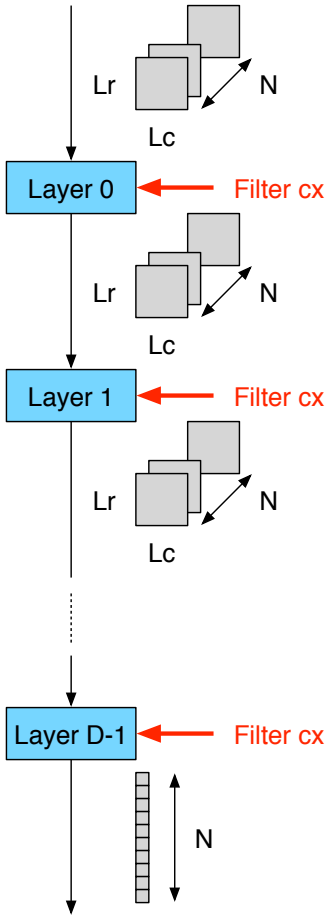
Feature Map Size

- Feature maps ($N_{i/o}$ x L_r x L_c per layer)
  - Feature maps encode information in the testing data
  - The information you're trying to extract is diffused within the feature maps
  - Need to track feature map data size as it flows through the network to make sure that there are no information killing bottlenecks
  - Remember the data processing inequality from the probability / information theory lecture



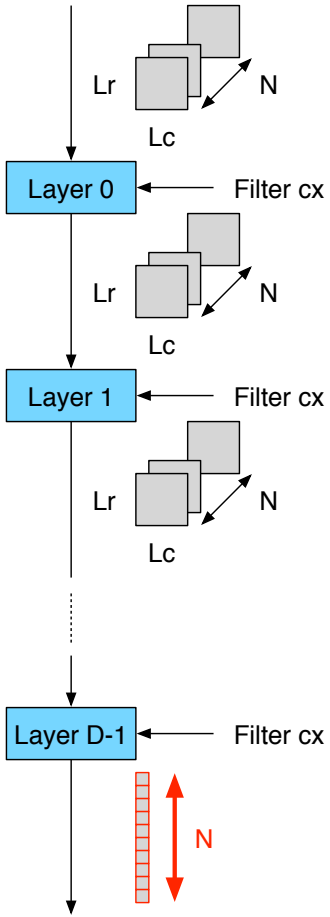
Filter Coefficient Size

- Filter coefficients (F_r x F_c x N_i x N_o and bias N_o per linear transformation)
  - Together with the network structure contain all information extracted from training data that will be applied to testing data
  - Remember automatic differentiation and gradient descent from the calculus lecture
  - Would like to have as few as possible because of performance reasons
  - But less filter coefficients tends to mean less extracted information, so there's a balance



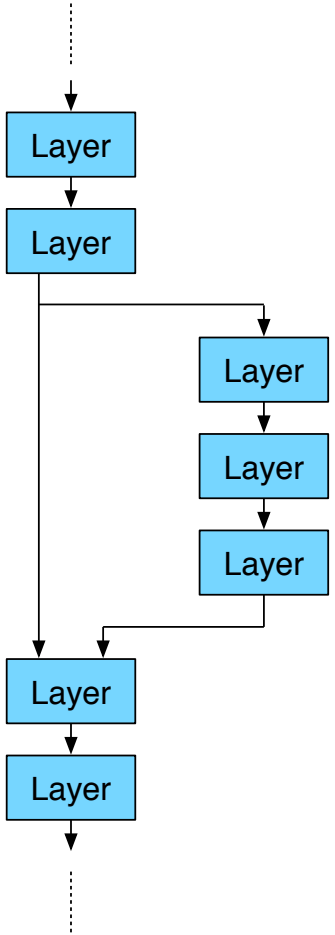
Problem Complexity

- It's important to understand how complex the problem is you're trying to solve
  - Classification to 2 classes, 1000 classes, regression to a real number, …
  - Easily separable classes or easily confused classes?
  - The simpler the function needed to solve the problem the simpler the network that can be used to solve it (and the converse is also true)
  - Remember the universal approximation proof from the calculus lecture
  - Lot's of problems we care about are really pretty complex



Graph Specification

- Implicitly or explicitly, computational graphs are used for high level network descriptions (other algorithms too)
- Graph edges are memory and nodes are compute
  - Not shown are practicalities that we'll care about during the implementation including feature map location and data movement, instruction location and data movement, computation partitioning, computation algorithm selection, ...
  - We'll deal with this in the implementation section
- Graphs are used for training and testing
  - We've already seen this with automatic differentiation with reverse mode accumulation
  - Typically just need to specify the forward graph, the backward graph and weight update graph can be auto generated (with a few other differences between training and testing)



Strategy

A Framework For Thinking About Networks

- Feature extraction and prediction (the focus of these slides using CNNs)
  - Tail feature extraction (optional) | encoder
  - Body feature extraction | encoder
  - Head prediction | decoder
- Will discuss ~ application specific components of the data to information extraction problem later in the context of applications
  - Pre processing
  - Post processing
- Will also discuss training, evaluation, performance, implementation and more applications later
  - For now just focusing on design basics
  - But realize that everything is coupled

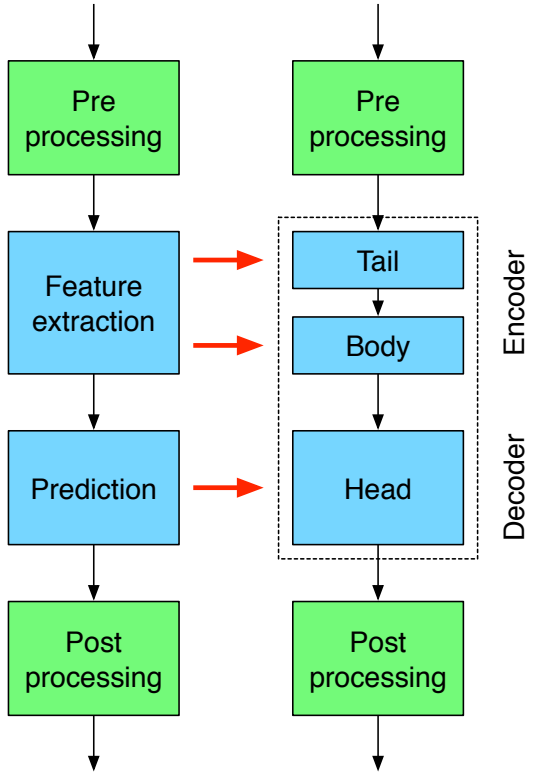
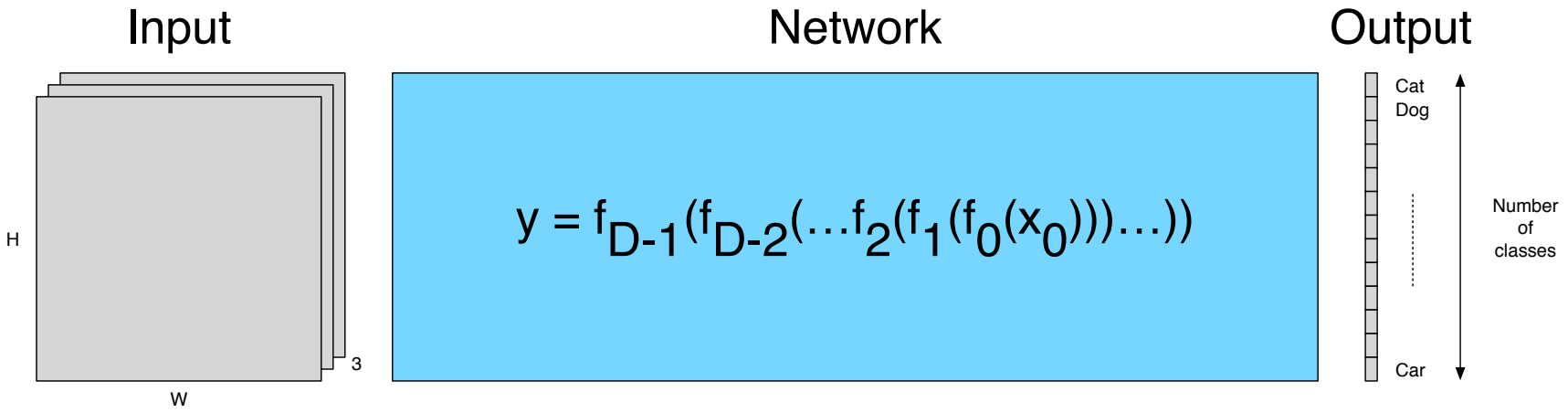


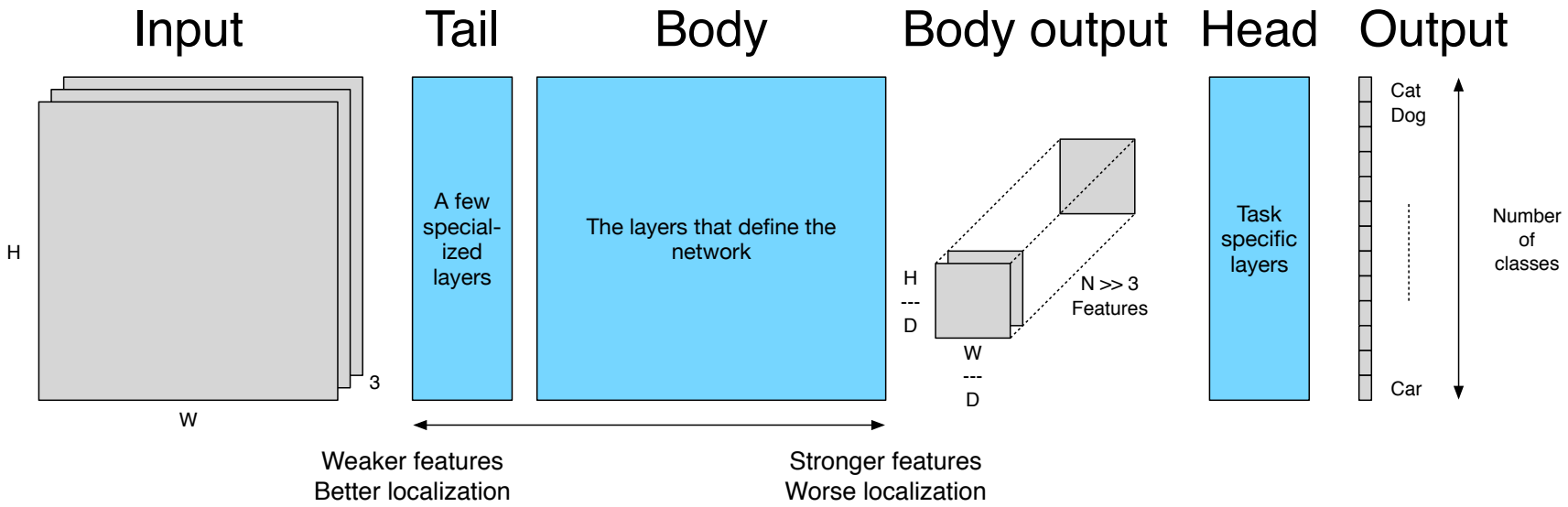
Image Classification As A Starting Point

A classification network maps an input image to an output vector; the largest element of the output vector is the predicted class

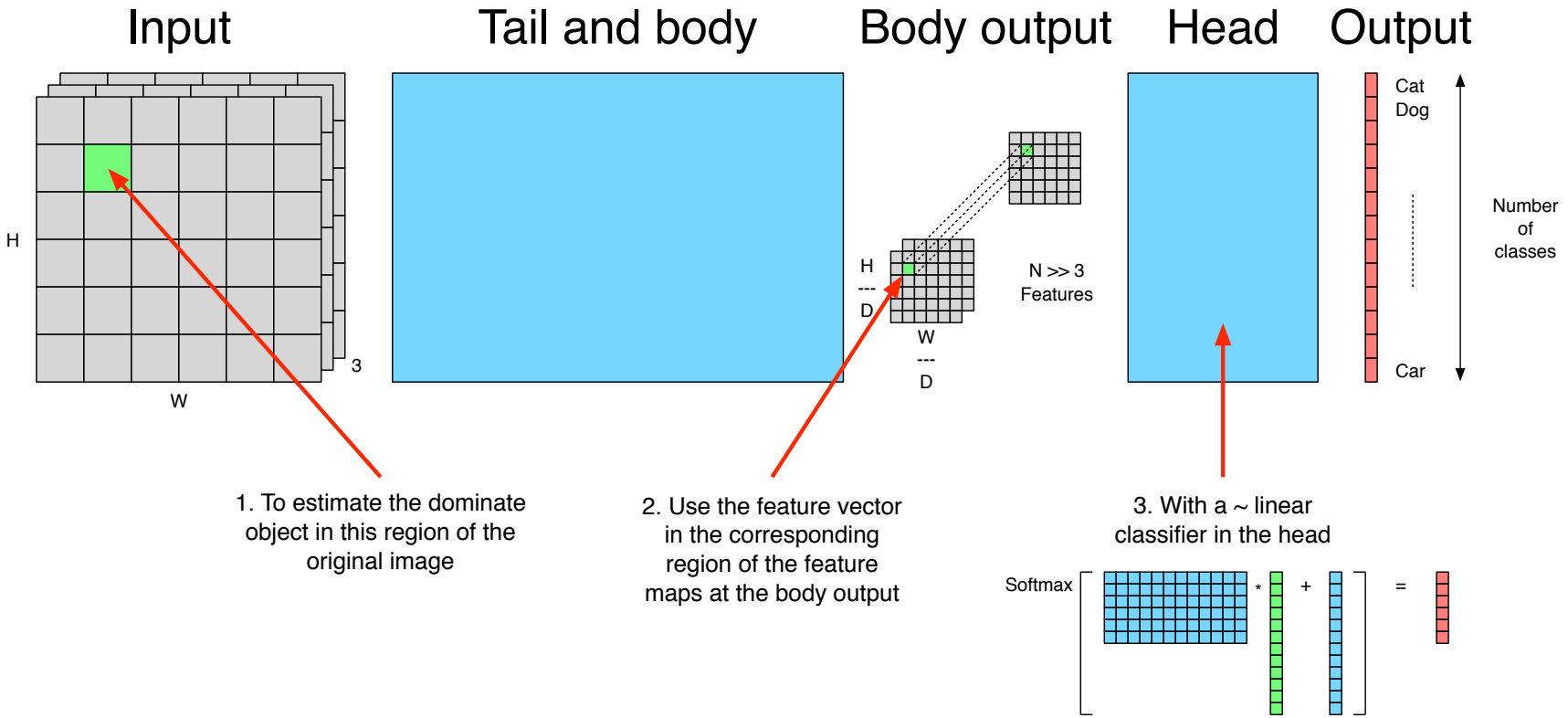


Tail Body And Head Decomposition

Modern networks decompose into a tail, body and head; the tail and body encode / extract features and the head decodes / makes predictions

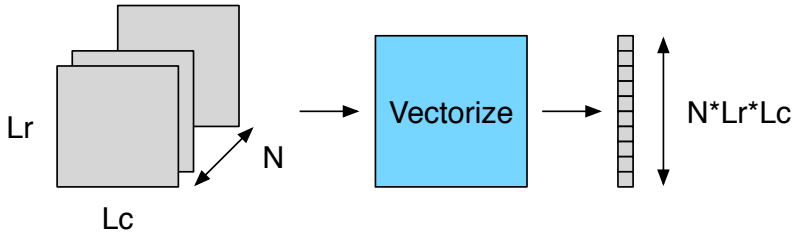


Conceptually How The Head Works

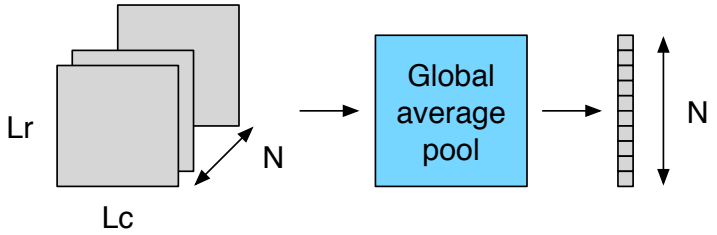


Feature Map To Feature Vector Transform

- 3D tensor to 1D vector transformation
  - Output of convolutional layers is N x Lr x Lc
  - Input to linear classifier is N x 1
  - 2 common methods for transforming between the output of the convolutional layers to the input of the linear classifier



- Vectorize
  - Historically first
  - Allows features for all pixels to be combined arbitrarily at the expense of more complexity and memory
  - Likely a lot of redundancy in the subsequent classifier
    - Perhaps why dropout is frequently used for this case



- Global average pool
  - Most popular right now
  - Average features together for all pixels
  - Less complexity and memory (math subset of vectorize)

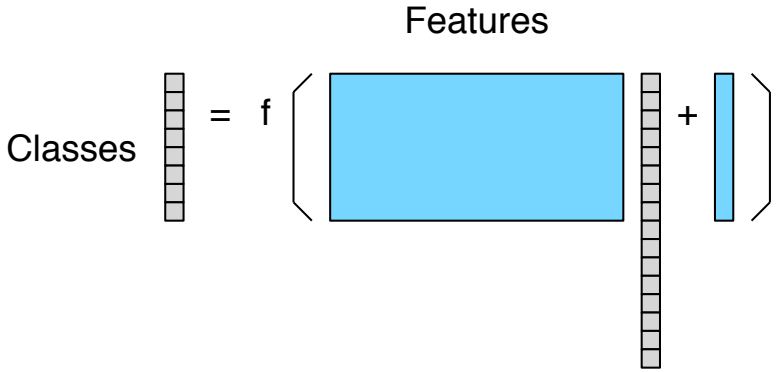
The linear classifier requires an input vector of a fixed size; convolutional layers in the body work on feature maps of ~ arbitrary size; this needs to connect them

Linear Classifier

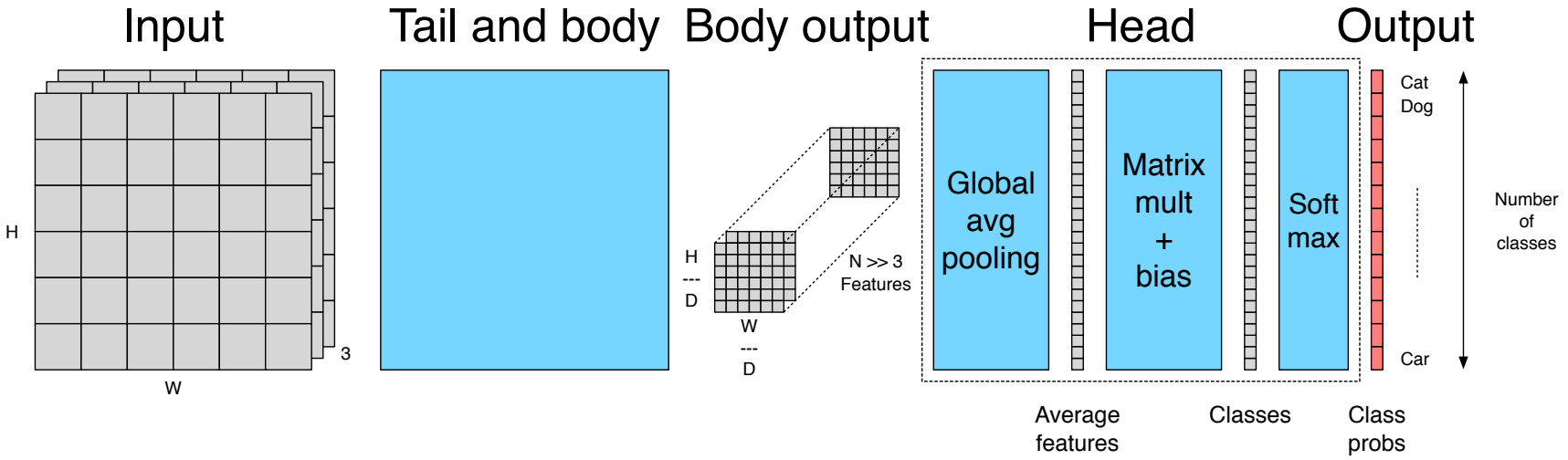
- Linear classifier
  - + softmax to convert to probabilities (training)
  - + arg max to select max output as class (testing)

- Implicit assumption is that the output classes are linearly separable in terms of the input features to the linear classifier (the job of the tail and body)

- The linear classifier works best when the number of features is same order of magnitude or larger than the number of classes
  - See the linear algebra notes for outline of proof
  - Intuition is that more features add robustness to the estimation



Example Head Design



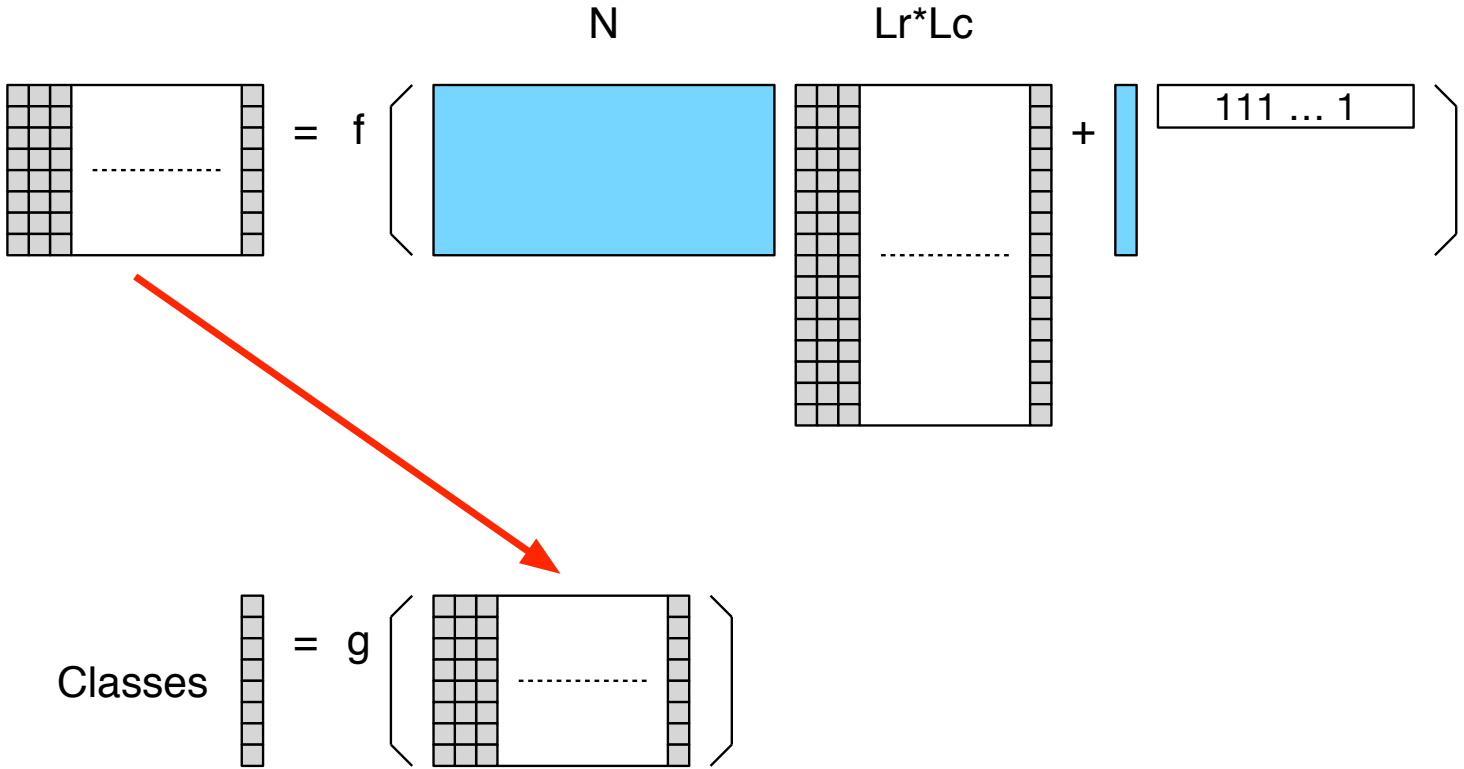
2 Head Design Ideas

- Maybe already done by someone else (I don't know)?
  Maybe doesn't work as well (I don't know)?
  Maybe would make a nice project (I don't know)? ...

- An idea that sits in between vectorize and global average pool
  - Idea is to estimate the class of each final feature map pixel (which corresponds to a region in the original image) then combine these classes to estimate the dominant class of the full image
  - Mechanics: create vectors for each pixel, process them all with the same classifier then combine classifier outputs (different options for this, could be as simple as picking the most common, could be trained which allows class combination and location to come into play; SqueezeNet V1 did this where the combination is averaging, is there a better method?)
  - Related: could potentially create better classification error functions with training data with localized labels
  - Similar memory and compute to global average pool (matrix matrix vs matrix vector which is memory bound)

- An idea that is an enhancement of global average pooling followed by a linear classifier
  - Different regions in the image potentially have different levels of importance for predicting the dominant class in the image (maybe edges are less important and the center is more important?)
  - Mechanics: weight the features of the final feature map different spatially via a pointwise multiplication before the global average pooling, potentially learn the optimal weighting

Head Design Idea 1

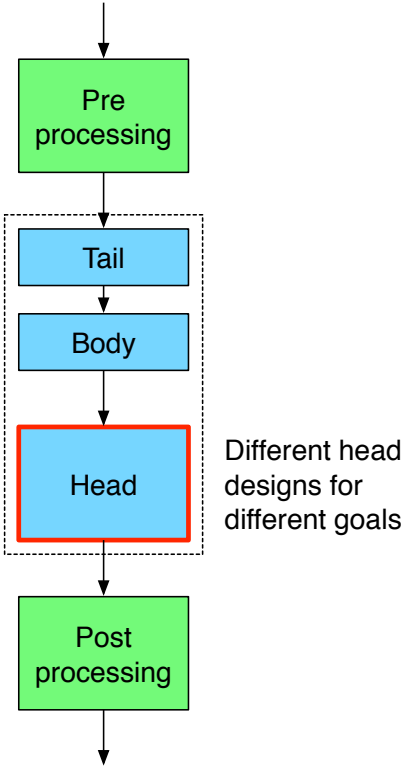


In The Near Future

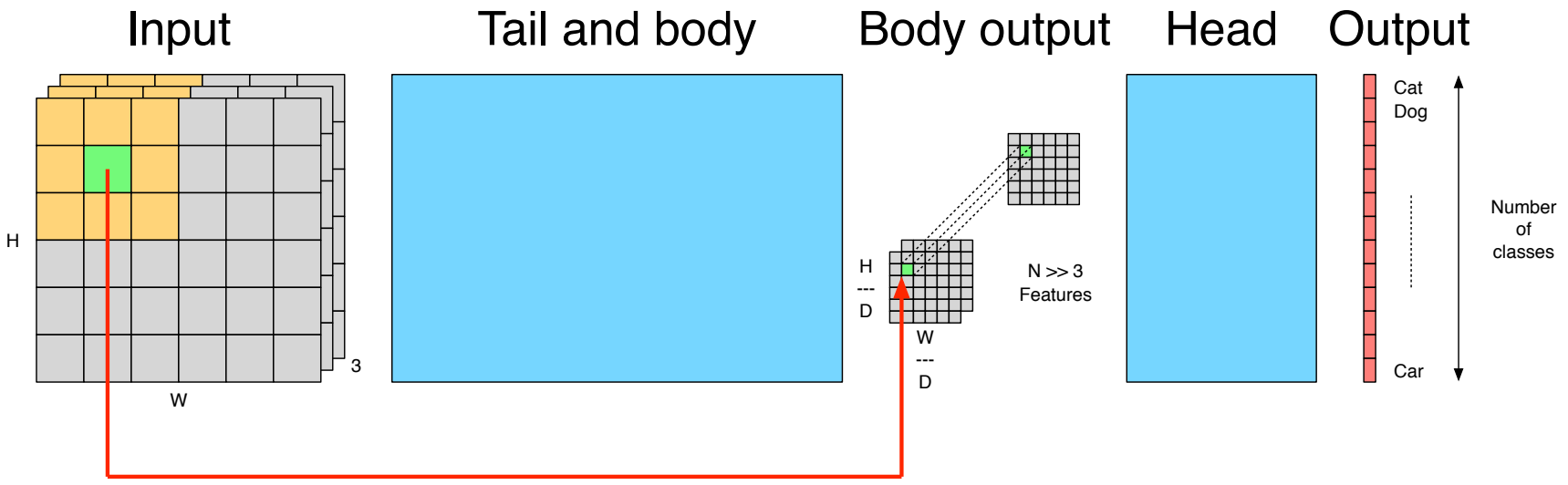
- Specifically, when we look at different applications in more detail

- We're going to look at different head designs to accomplish different goals
  - Multiple object detection
  - Segmentation
  - Depth
  - Motion
  - Character estimation
  - ...

- Note that more than 1 head can be attached to the same tail and body
  - Implicitly, the same features are used for more than 1 task



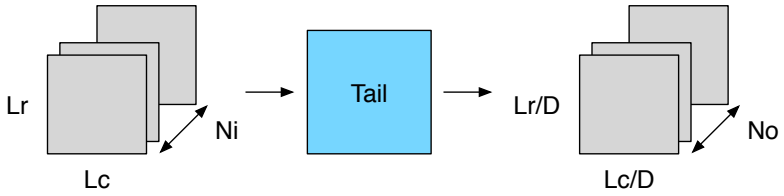
Conceptually How The Tail And Body Work



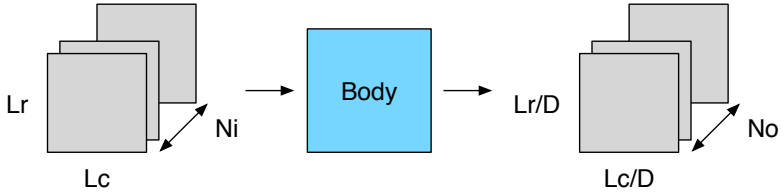
The tail and body transform regions from an image to feature vectors; the receptive field size shown in orange is the area around the region of interest that the body can use to create the feature vector

Why Distinguish Between The Tail And Body?

- The distinction is somewhat arbitrary
  - Both take input feature maps (an RGB input is 3 feature maps) and generate output feature maps
  - Features get stronger after each transformation



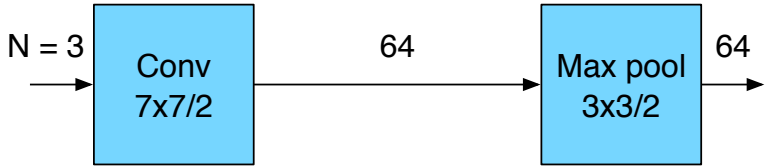
- So why make the distinction?
  - There are a few common tail designs that most people use
  - There are a lot of different building blocks for the body that effectively define the network name



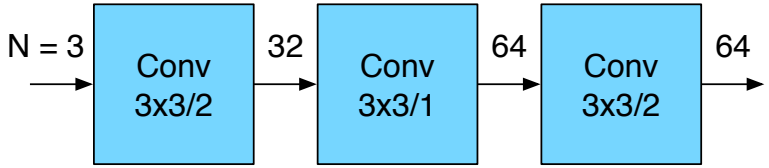
- Plan
  - We'll discuss a few tail designs in this section
  - We'll discuss body designs in the network section

Tail: Data To Weak Features

- Initial data to feature mapping
  - Optional (can be thought of as part of the body)



- Features
  - High compute
  - High feature map memory
  - Low parameter memory
  - Lots of spatial redundancy



- Common design components
  - Aggressive down sampling (but don't lose useful information)
  - Aggressive increase in the number of channels

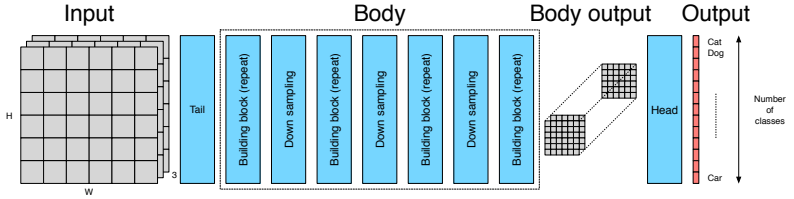
Project idea: look at the transformations in trained tail designs, look for ways of saving compute (e.g., I believe a paper in the past exploited symmetry to 1/2 compute)

Body: Weak Features To Strong Features

- Building blocks we'll see in network design followed by down sampling (strided convolution or pooling)

- Interpretation
  - Lower level features are more class agnostic
    - Less channels earlier in the network
  - Deeper features are more class specific
    - More channels later in the network

- Feature map changes
  - Gradual memory reduction
  - Loss of spatial resolution
  - Increase of channel / feature dimension and strength



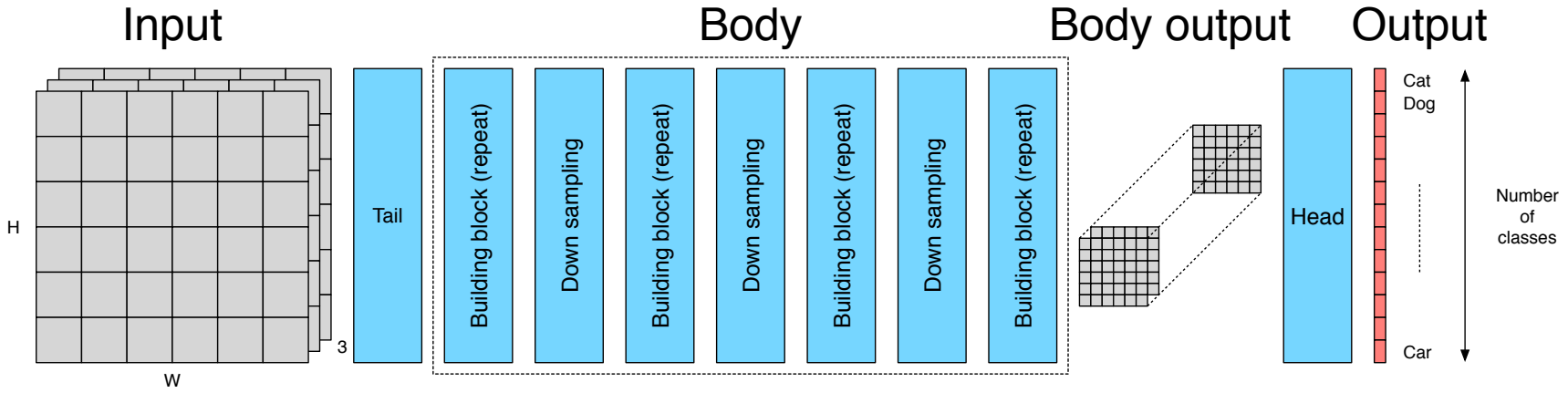
- Around down sampling stages it's common to
  - Reduce the rows and cols both by 2
  - Increase the channels by 2

- Result is the data volume shrinks by 2
  - Data before = channels x rows x cols
  - Data after = 2*channels x rows/2 x cols/2
    = data before / 2
  - Effectively reduces compute by a factor of 2

- Project idea: investigate ordering of channel increase and down sampling for classification and segmentation

Body: Weak Features To Strong Features

For ImageNet classification it's common to have 5 levels of down sampling before the global average pool: ~ 2 in the tail and ~ 3 in the body



Receptive Field Size

- Receptive field size at the head
  - How to compute in theory
  - But effectively smaller in practice (convolving a bunch of filters gives a ~ Gaussian shape)
- What does the classifier have to work with when making a decision
  - Idea of looking at the input through a screen of that size
  - Objects of different classes can have different sizes, objects of the same class can have different sizes
  - Implications on changing the input resolution

Calculation

- Start at the output last convolutional layer and initialize the receptive field size to 1
- Move backwards through the network to the very beginning
- Every time you go through a filter increase the receptive field size by adding F – 1 (the length of the filter – 1); observe that 1x1 filters do not increase the receptive field size
- Every time you go backwards through a down sampling operation multiply the receptive field size by S (the down sampling factor)
- You can find the receptive field size at any place in the network; later useful when looking at skip connections
- Once at the very beginning you have the receptive field size with respect to the input
- Note: this is the maximum theoretical size, and the actual size will likely be much smaller (think a Gaussian like shape centered in the receptive field span that collects energy non uniformly)

Layers

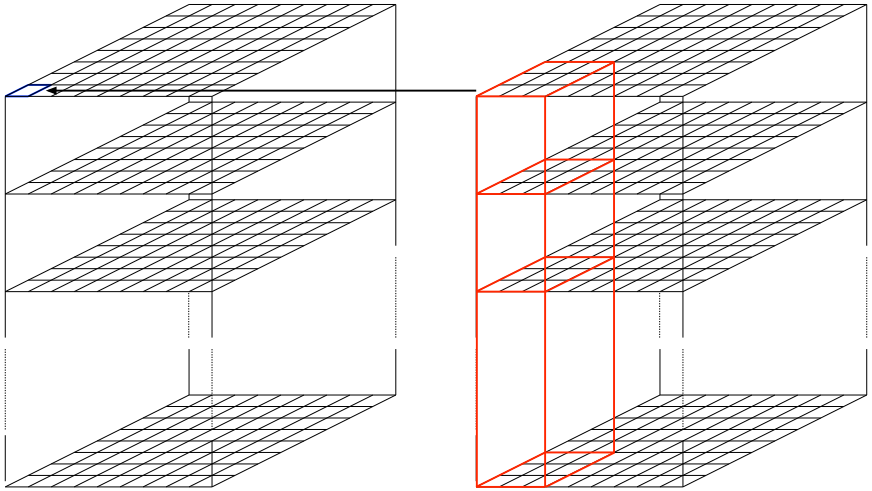
Layer Overview

- Layers map input feature maps to outputs feature maps
  - This section will look at the individual operations
  - The network section will combine multiple layers together into building blocks

- The key is to understand the dimensions in which layers combine information when mapping from input to output
  - Channel and space
  - Channel
  - Space
  - Element
  - Time
  - Rearrangement
  - Training

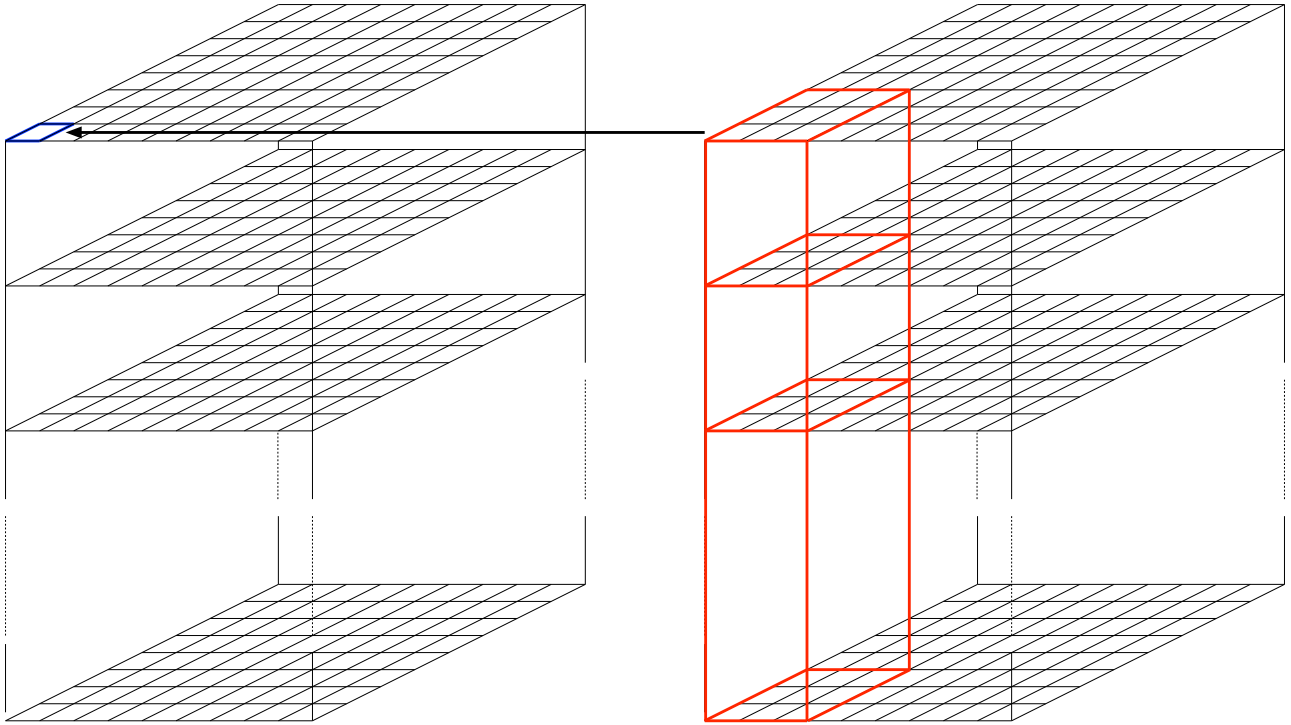
Channel And Space

- CNN style 2D convolution layer
  - Filter
    - Tensor up sampling
    - Matrix creation (nothing to do)
  - Input feature map
    - Tensor up sampling
    - Tensor 0 padding
    - Matrix create (Toeplitz style)
  - Matrix reduction from filter up sampling
    - Remove cols from filter matrix from up sampling
    - Remove associated rows from the input matrix
  - Matrix reduction from output down sampling
    - Remove cols from input matrix
  - Output feature map
    - Matrix create via filter matrix * input matrix + bias
    - Tensor creation (nothing to do)



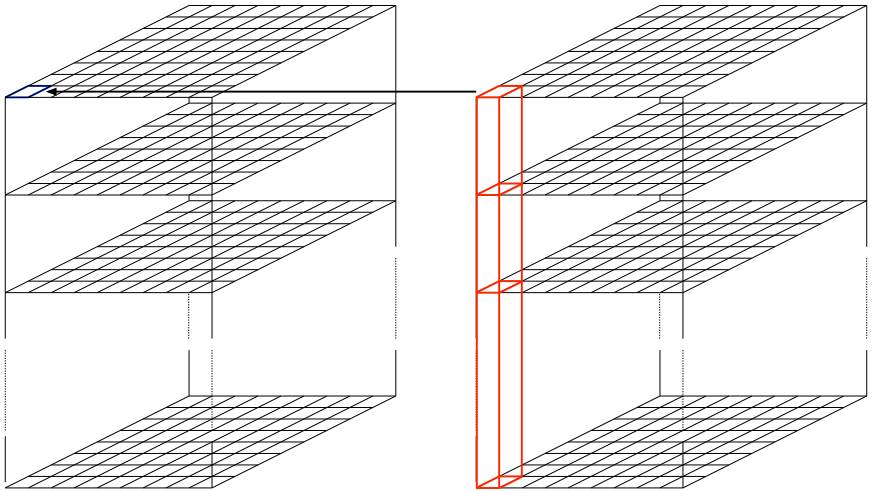
Spatial invariance of CNN style 2D convolution allows for memory and compute reduction but is also plausible with respect to feature extraction from many images

Channel And Space

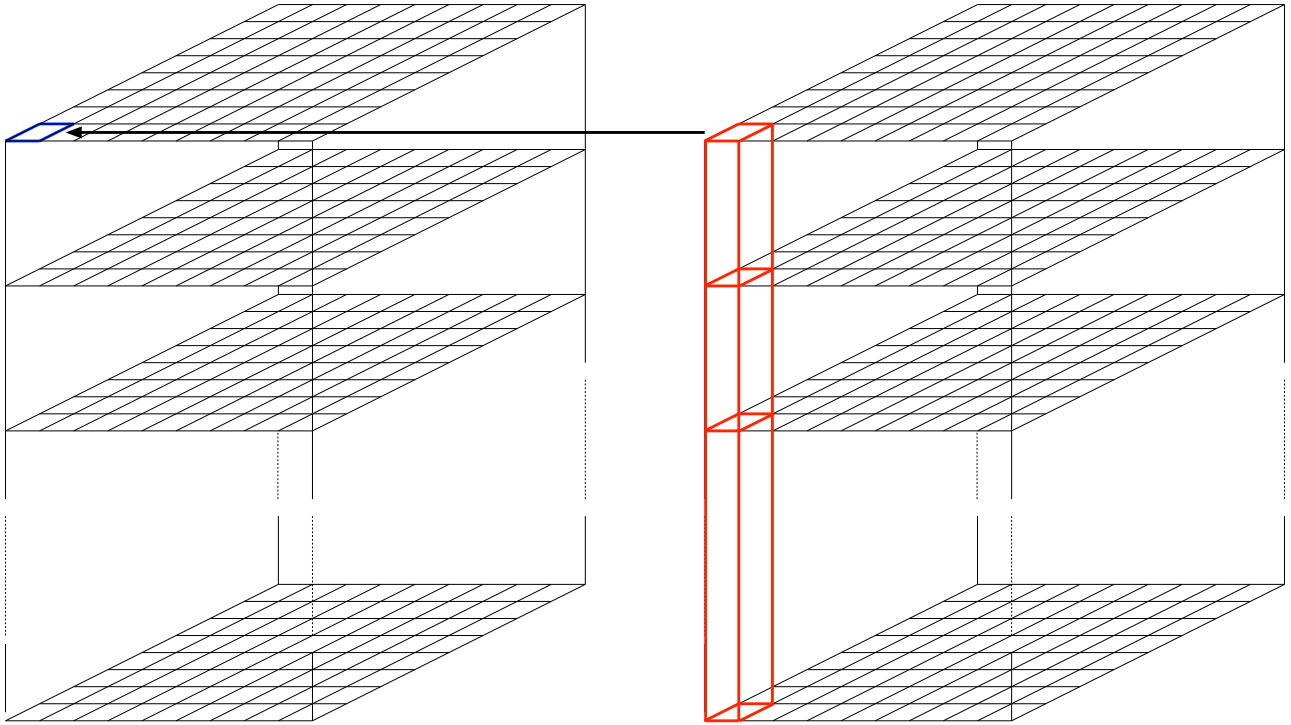


Channel

- CNN style 2D convolution with Fr = Fc = 1 (matrix matrix multiplication)
- Fully connected layer for a Ni x 1 x 1 input (or batch of Ni x 1 x 1 inputs)
- Local response normalization
  - Famously used in AlexNet / CaffeNet
  - Typically not used now (input normalization via pre processing helps some)

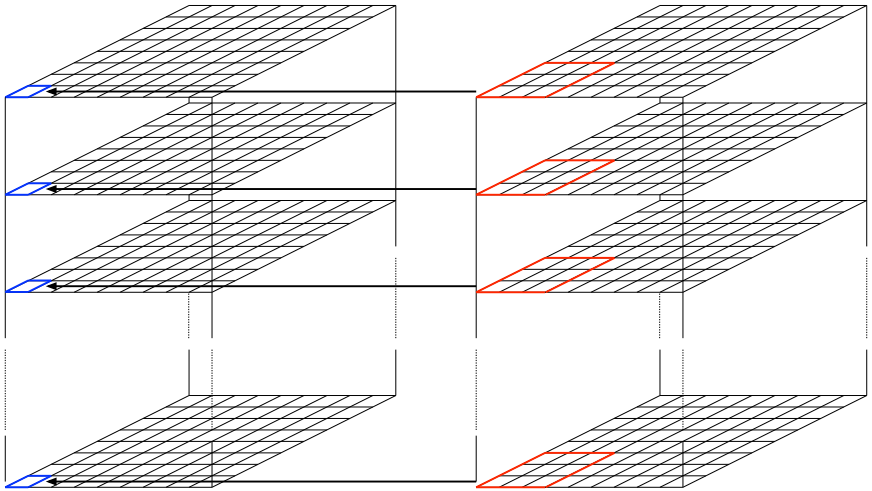


Channel



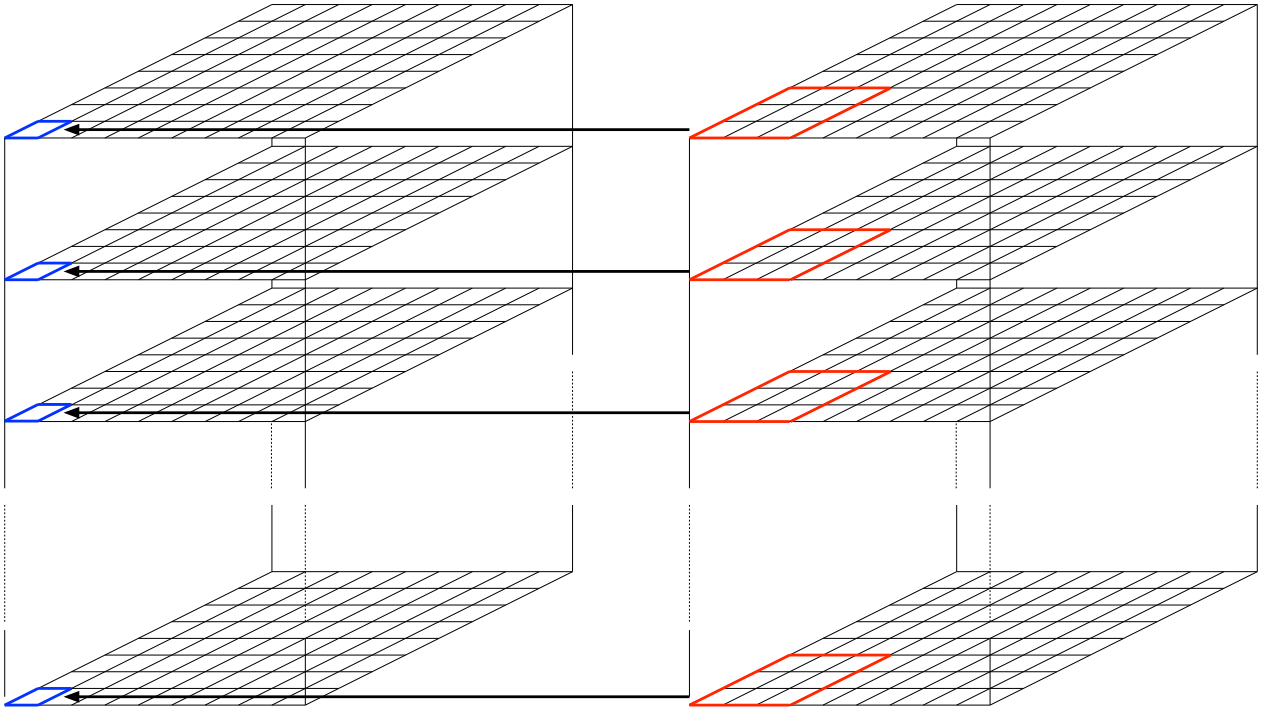
Space

- CNN style 2D convolution that's fully grouped (standard 2D convolution, depth wise separable convolution)
- Decimation
  - Max pooling
  - Avg pooling
  - Global average pooling
  - Spatial pyramid pooling
- Interpolation
  - Nearest neighbor
  - Bilinear



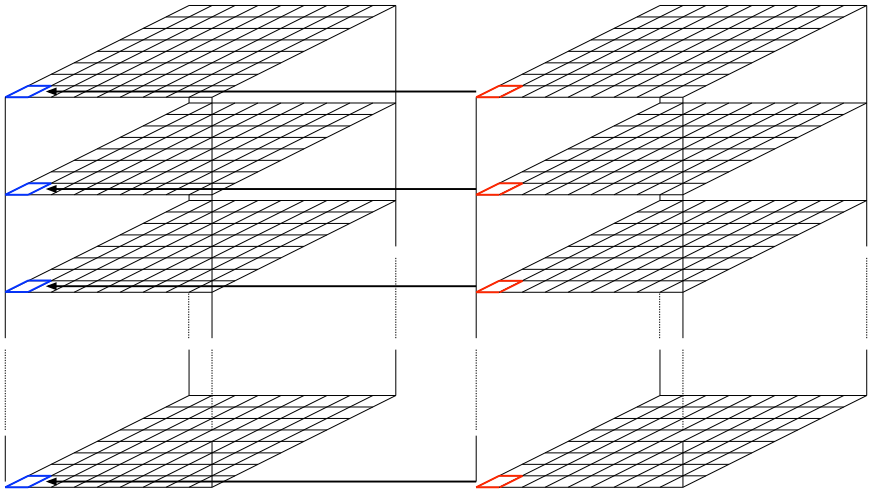
Pooling, as shown in the above figure, allows for the aggregation of larger regions of input features in the generation of output features (increasing receptive field size)

Space

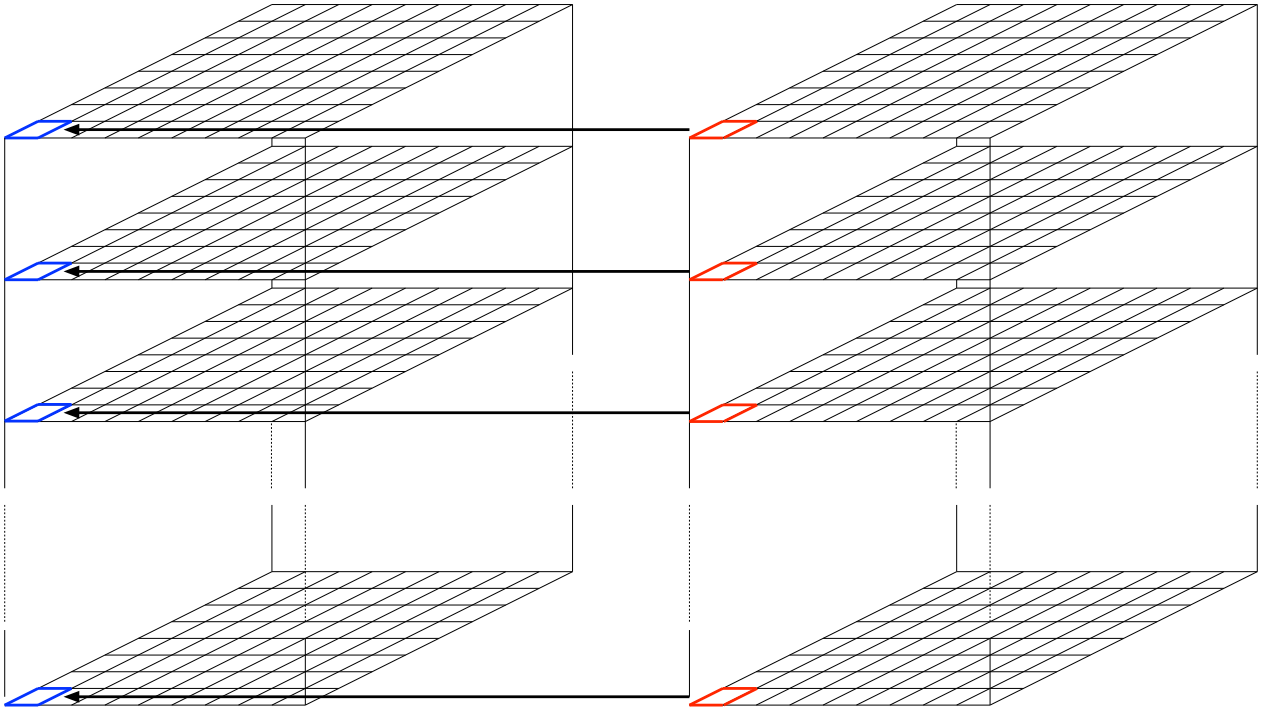


Element

- Nonlinearities
  - ReLU, ReLU6, PReLU, ReLU with a small negative slope, sigmoid, tanh, ...
- 1 input 1 output linear
  - Scale
  - Offset
- 2 input 1 output math
  - Add, multiply, max, ...



Element

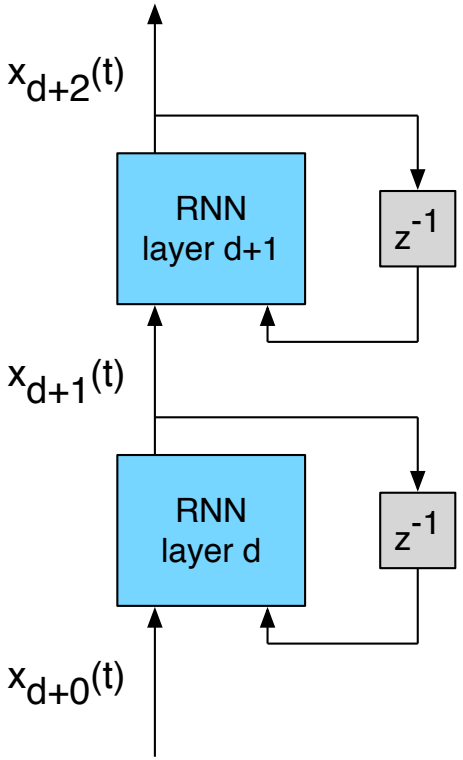


Time

- Will come back to these in the context of speech and language

- Examples (many variants exist)
  - RNN
  - LSTM

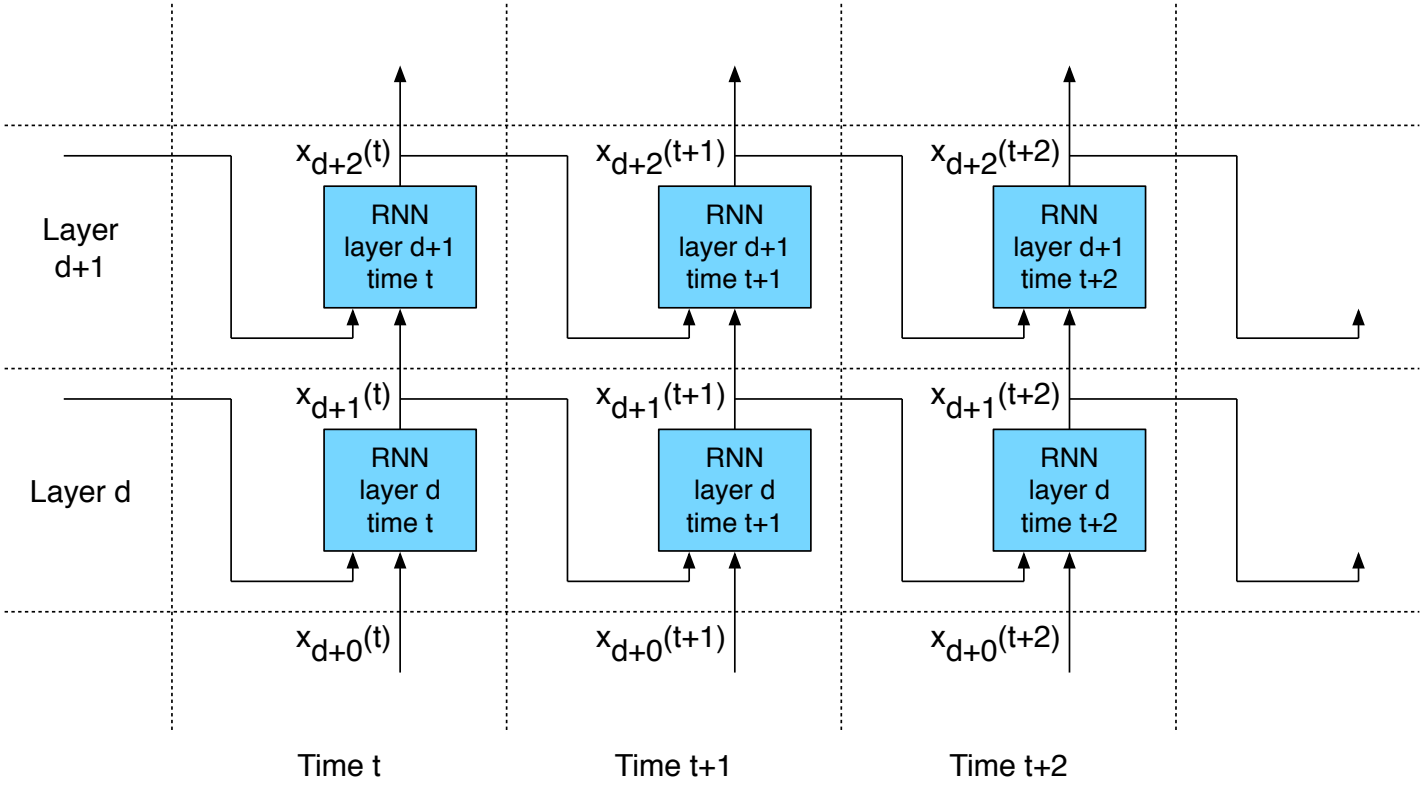
- An example deep RNN
  - $\mathbf{x}_{d+1}(t) = f_{d+0}(\mathbf{G}_{d+0}\,\mathbf{x}_{d+1}(t-1) + \mathbf{H}_{d+0}\,\mathbf{x}_{d+0}(t) + \mathbf{v}_{d+0})$
  - $\mathbf{x}_{d+2}(t) = f_{d+1}(\mathbf{G}_{d+1}\,\mathbf{x}_{d+2}(t-1) + \mathbf{H}_{d+1}\,\mathbf{x}_{d+1}(t) + \mathbf{v}_{d+1})$
  - ...



Question to myself, maybe a project idea (?): investigate the structure of G for different applications and at different places in the network; how much mixing is there across features?

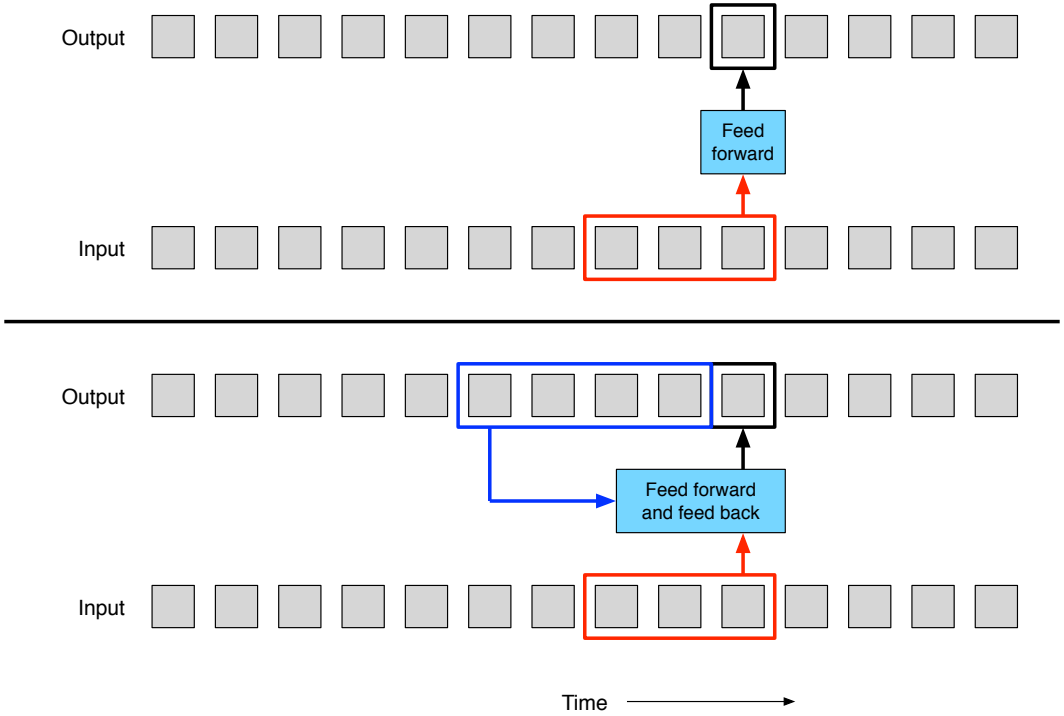
Time

The RNN on the previous slide unwrapped in time



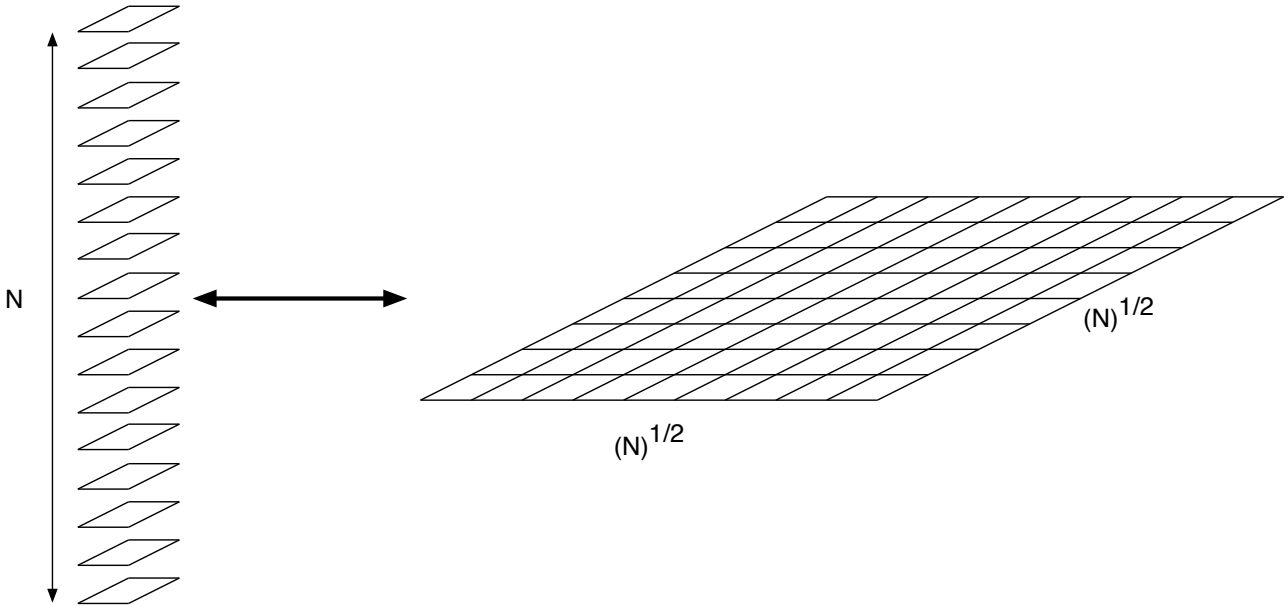
Time

Note commonalities between fully connected layers and FIR filtering and RNN layers and IIR filtering



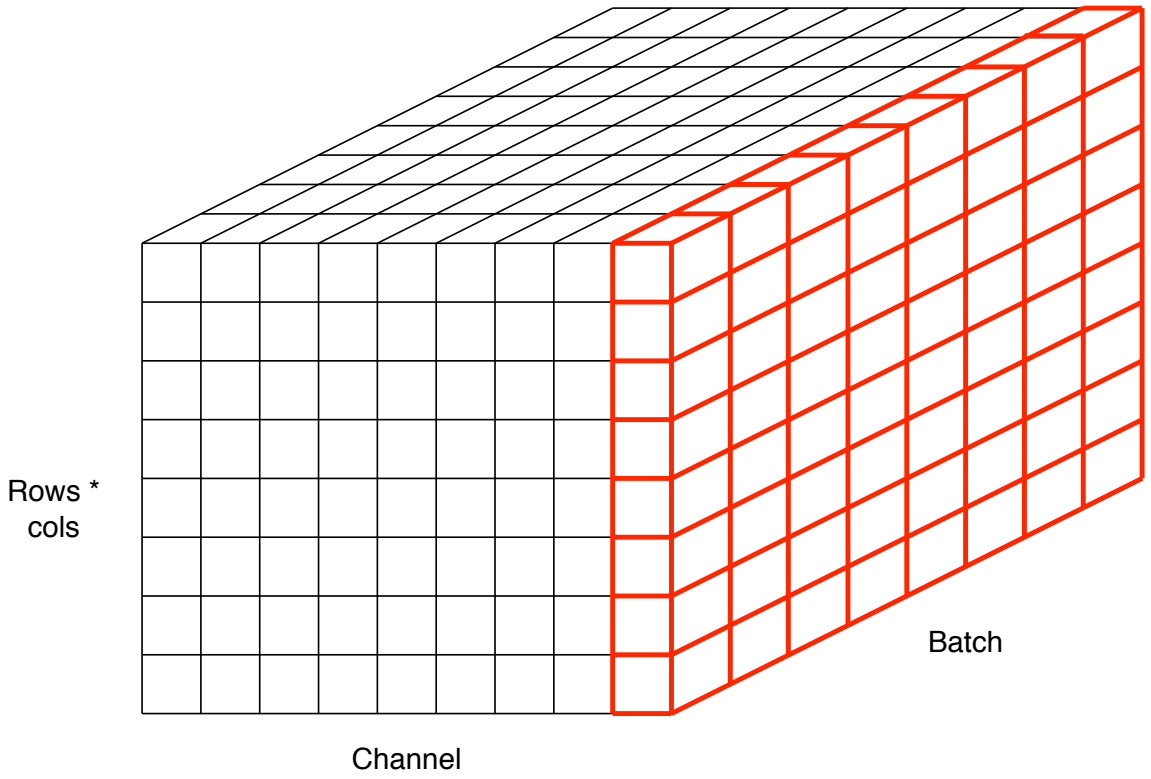
Rearrangement

- Reshape
  - Channel to space
  - Space to channel
- Concatenate
  - Channel
  - Space
- Split
  - Channel
  - Space



Training

- Will come back to these in the context of training
- Normalization
  - Batch
  - Group
- Dropout
- Loss
  - Lp
  - Softmax + cross entropy



Networks

Networks Overview

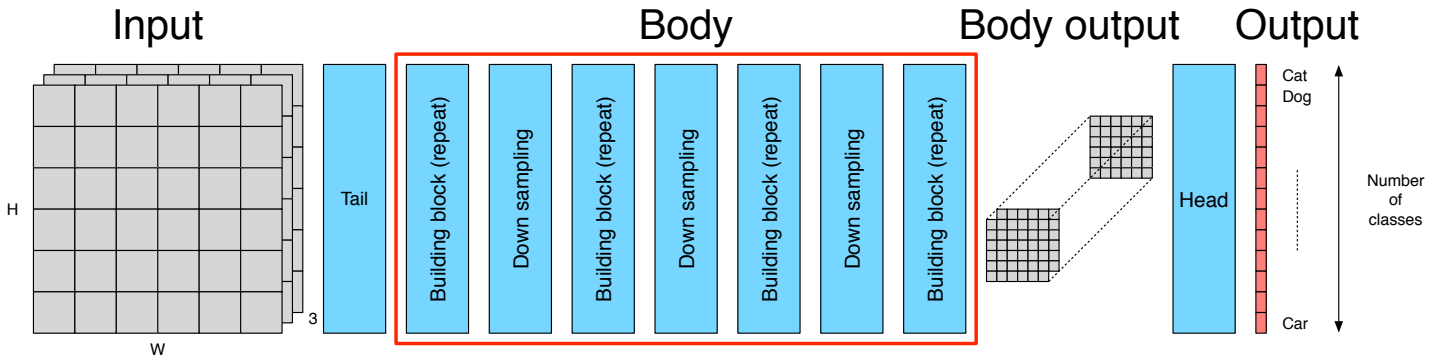
- Illustrated in the context of image classification ...

- We've seen some standard tail designs
  - The start of feature extraction / encoding
  - Ex: 7x7/2 conv, 3x3/2 max pool
  - Will look at more in the context of different applications

- We've seen some standard head designs
  - Prediction / decoding
  - Ex: global avg pool, fully connected layer, soft or arg max
  - Will look at more in the context of different applications

- We've seen a bunch of layers that combine information in different ways

Networks Overview



- Networks will now focus on some popular body designs
  - The main portion of feature extraction / encoding
  - The body is effectively what defines / names the network
  - Traditionally, there's a lot more variety in the body design
  - Frequently going to put together layers to form common building blocks
  - Frequently going to replicate the building blocks at different network depths

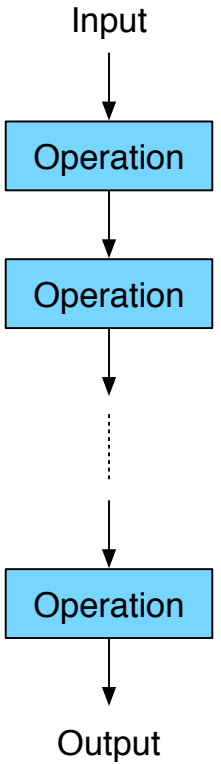
Organization

- There are many many different network body designs
- This section will look at some of the more historically important and successful designs grouped together based on their building block structure
  - Chain
  - Parallel
  - Dense
  - Residual
- Will also briefly look at different optimization methods for network design

Chains

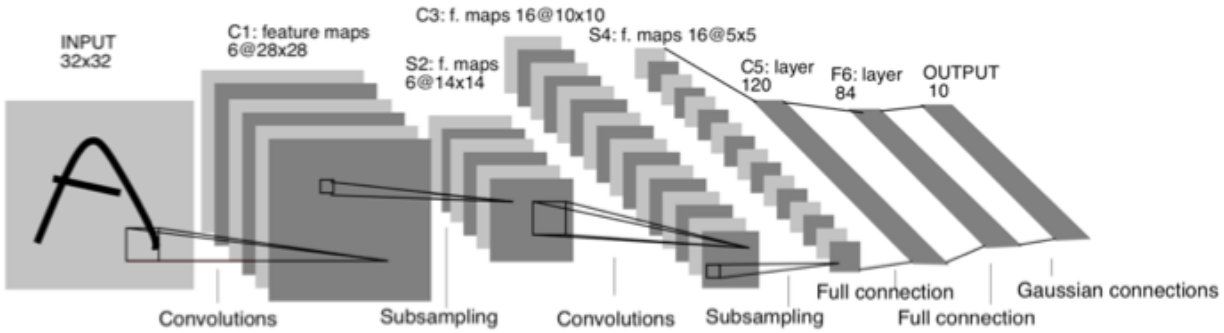
- A sequential set of operations from input to output
  - The historical starting point and how we've thought about network design so far

- Examples
  - LeNet
  - AlexNet / CaffeNet
  - VGG
  - MobileNets V1



Chains – LeNet

- Included here for historical reasons
  - Significant for core CNN design ideas still used today
- Not many layers
  - Ok because input (MNIST) is small and the problem is not overly complex
- Different head design
  - Uses vectorization
  - Predicts code vs 1 hot
- Gradient-based learning applied to document recognition
  - http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

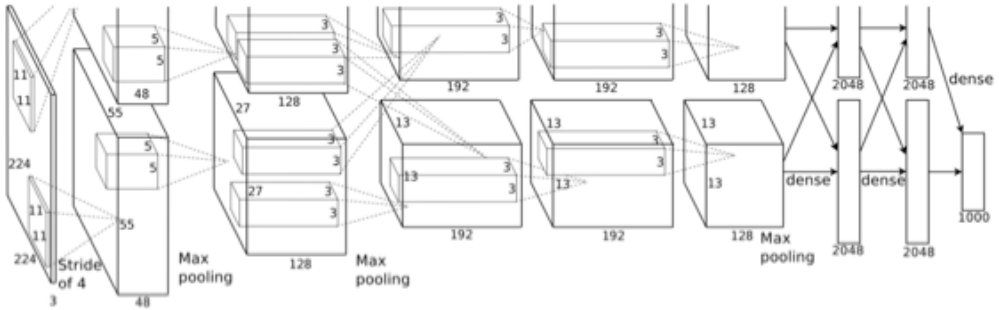


Notes
- While the code approach to creating an error to optimize is out of favor, maybe it's worth thinking about when there are many similar categories and how it potentially can be used with / relates to a hierarchical head design, creating multiple targets for a single category, ...

Figure from original paper

Chains – AlexNet / CaffeNet

- Included here for historical reasons
  - Significant for winning the ILSVRC in 2012 which helped re spark the use of CNNs for vision and all sorts of other problems
- Issues
  - Many free parameters in larger filters
  - Data locality for local response norm
  - Memory in fully connected layers
  - Real time implementation is implicitly a memory bandwidth test
- ImageNet classification with deep convolutional neural networks
  - https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- CaffeNet
  - https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet



Notes

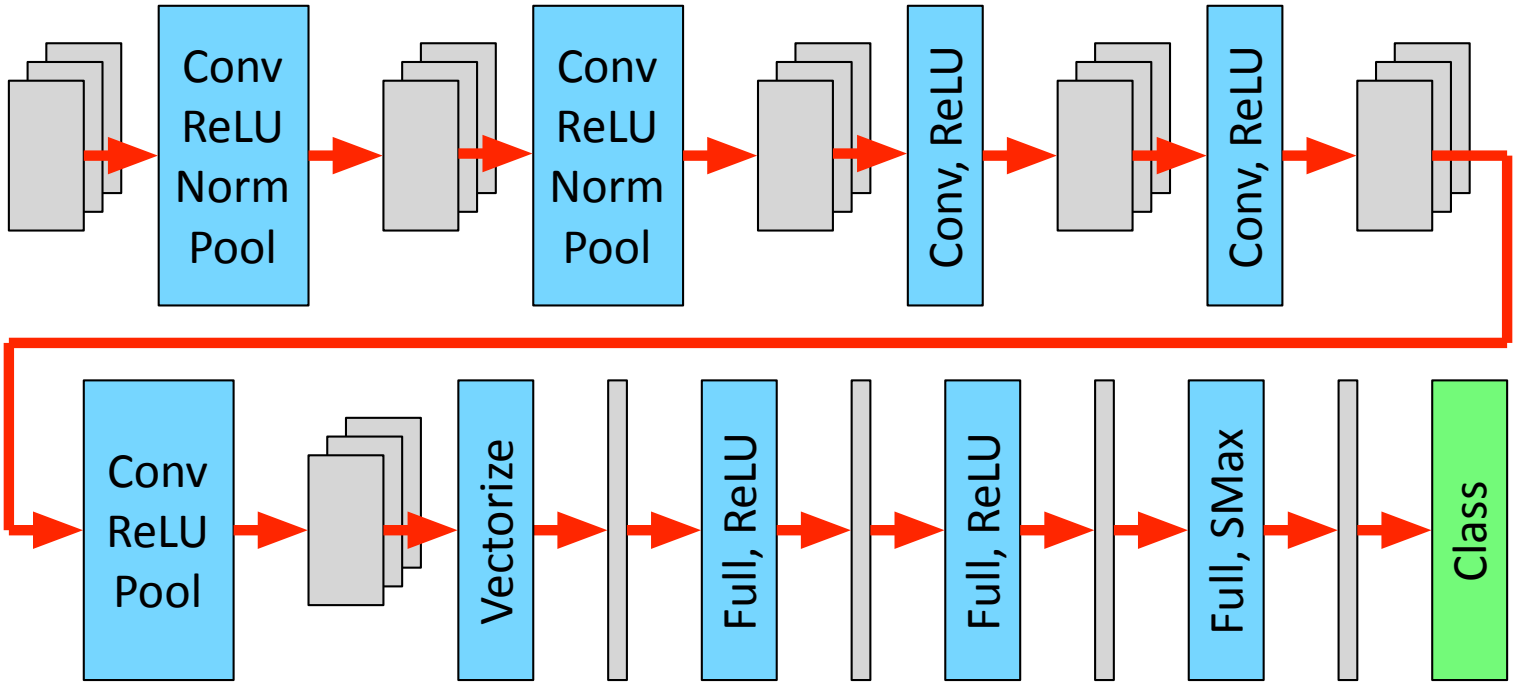
- 5 convolutional and 3 fully connected layers
- ReLU nonlinearity (vs tanh or sigmoid) and 3x3/2 max pooling (vs 2x2/2)
- Grouping (on 2 GPUs) to allow for larger models
- Data augmentation and dropout
- CaffeNet like AlexNet but flip flop LRN and pooling, remove grouping

Visualization

- https://dgschwend.github.io/netscope/#/preset/alexnet
- https://dgschwend.github.io/netscope/#/preset/caffenet

Figure from original paper

Chains – AlexNet / CaffeNet



Chains – AlexNet / CaffeNet

#	Type	B	Ni	M	P	F	S	No	MACs	In Map	Parameter	Out Map	All Mem
0	Inpt	1	3	224	0	0	0	0	0	0	0	150528	150528
1	Conv	1	3	224	3	11	4	96	105705600	150528	325248	290400	766176
1	Nonl	1	96	55	0	0	0	0	290400	290400	0	290400	580800
1	Norm	1	96	55	0	5	0	0	2613600	290400	0	290400	580800
1	Pool	1	96	55	0	3	2	0	629856	290400	0	69984	360384
2	Conv	1	96	27	4	5	1	256	448084224	69984	801024	186624	1057632
2	Nonl	1	256	27	0	0	0	0	186624	186624	0	186624	373248
2	Norm	1	256	27	0	5	0	0	1679616	186624	0	186624	373248
2	Pool	1	256	27	0	3	2	0	389376	186624	0	43264	229888
3	Conv	1	256	13	2	3	1	384	149585280	43264	949632	64896	1057792
3	Nonl	1	384	13	0	0	0	0	64896	64896	0	64896	129792
4	Conv	1	384	13	2	3	1	384	224345472	64896	1392000	64896	1521792
4	Nonl	1	384	13	0	0	0	0	64896	64896	0	64896	129792
5	Conv	1	384	13	2	3	1	256	149563648	64896	928000	43264	1036160
5	Nonl	1	256	13	0	0	0	0	43264	43264	0	43264	86528
5	Pool	1	256	13	0	3	2	0	82944	43264	0	9216	52480
6	Full	1	9216	1	0	1	1	4096	37752832	9216	37752832	4096	37766144
6	Nonl	1	4096	1	0	0	0	0	4096	4096	0	4096	8192
7	Full	1	4096	1	0	1	1	4096	16781312	4096	16781312	4096	16789504
7	Nonl	1	4096	1	0	0	0	0	4096	4096	0	4096	8192
8	Full	1	4096	1	0	1	1	1000	4097000	4096	4097000	1000	4102096
8	Nonl	1	1000	1	0	0	0	0	1000	1000	0	1000	2000

Chains – VGG

- Included here for historical reasons
  - No local response norm
  - Stacked 3x3 filters
  - Family of networks
- Issues
  - Fully connected layers still too big
  - Real time implementation is implicitly a memory bandwidth test
- Very deep convolutional networks for large-scale image recognition
  - https://arxiv.org/abs/1409.1556

Notes

- 2 stacked 3x3 conv layers have a receptive field size of 5 with only 18 parameters (vs 25 for a 5x5 filter); additionally, they have 2 nonlinearities (vs 1 for a 5x5 filter)
- 3 stacked 3x3 conv layers have a receptive field size of 7 with only 27 parameters (vs 49 for a 7x7 filter); additionally, they have 3 nonlinearities (vs 1 for a 7x7 filter)

Visualization

- https://dgschwend.github.io/netscope/#/preset/vgg-16

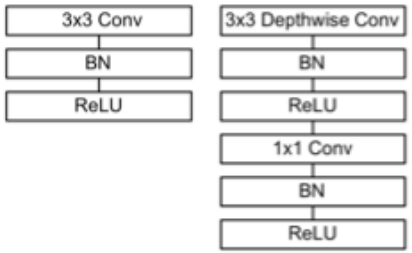
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

Table from original paper

Chains – MobileNet V1

- Design
  - Improves efficiency using a cascade of 3x3 spatial and 1x1 channel (vs standard 3x3); less parameters and compute with 1 extra nonlinearity takes previous stacked 3x3 idea to next level
  - Global pool then single fully connected layer for head
- Problems
  - Hidden downside is that there's less reuse of feature maps in the computation
  - Good for a particular vector architecture, inefficient for an optimized matrix architecture
- MobileNets: efficient convolutional neural networks for mobile vision applications
  - https://arxiv.org/abs/1704.04861

Nota bene →



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

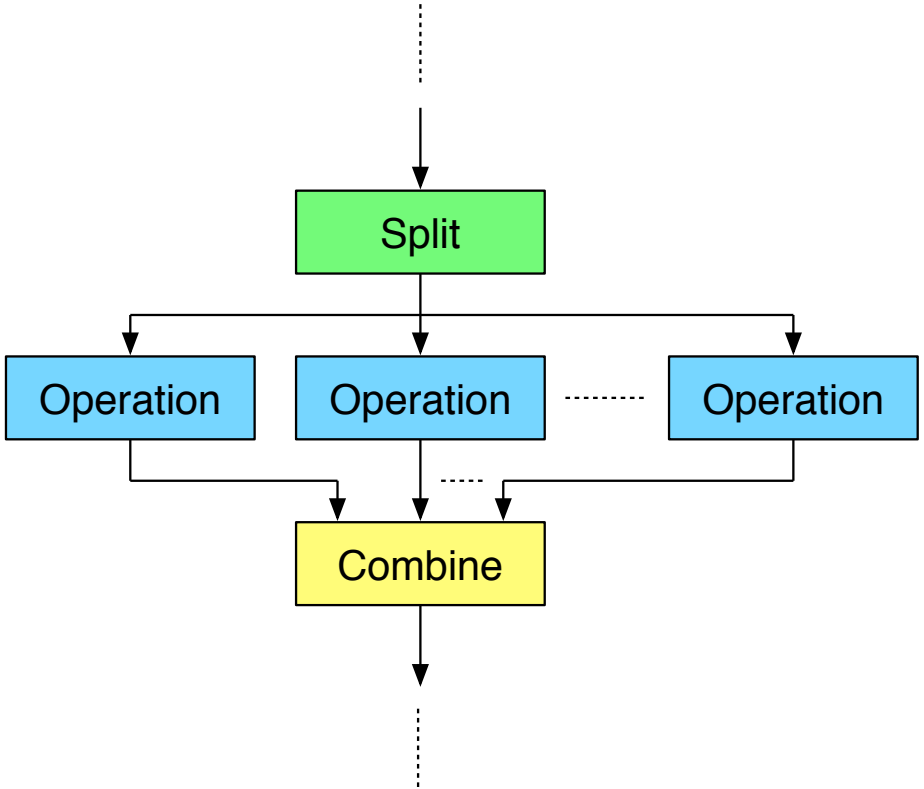
Notes

- CNN style 2D 3x3 convolution replaced with standard 2D 3x3 convolution and CNN style 1x1 convolution (matrix matrix mult)
- 3x3 mixes across space, 1x1 mixes across channel

Figure and table from original paper

Parallel

- Per building block
  - Input split
  - Parallel structure
  - Output combine
- Examples
  - GoogLeNet*
  - Inception V3 and V4
  - SqueezeNet



*My personal favorite network name

Parallel – GoogLeNet / Inception V2, V3 And V4

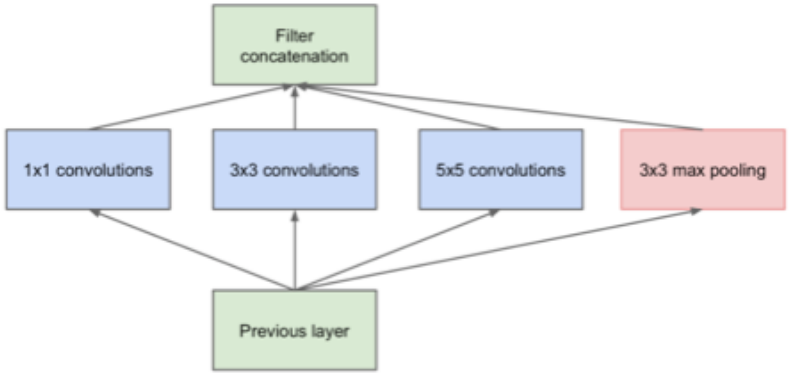
- Design
  - Parallel paths with different receptive field sizes for information to flow through a layer
  - Referred to as inception modules (think network in a network)
  - Different inception module designs are possible, both for different networks and within a network
  - This feels like a bit of a jumping off point for thinking about ML optimized designs
- Comments
  - Personally I prefer a slightly more regular structure, but math is not dependent on my personal preferences
  - These networks perform excellently and are reasonably well suited to implementation

Notes
- A simple guide to the versions of the inception network
  - https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202
- Going deeper with convolutions
  - https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf
- Rethinking the inception architecture for computer vision
  - https://arxiv.org/abs/1512.00567
- Inception-v4, inception-resnet and the impact of residual connections on learning
  - https://arxiv.org/abs/1602.07261

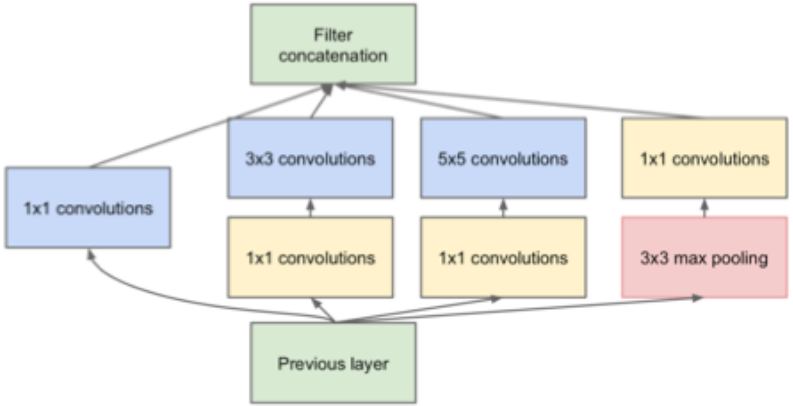
Visualization
- https://dgschwend.github.io/netscope/#/preset/googlenet
- https://dgschwend.github.io/netscope/#/preset/inceptionv3
- https://dgschwend.github.io/netscope/#/preset/inceptionv4

Parallel – GoogLeNet

Parallel paths (not used by GoogLeNet)



Parallel paths with dimensionality reduction



Figures from original paper

Parallel – GoogLeNet

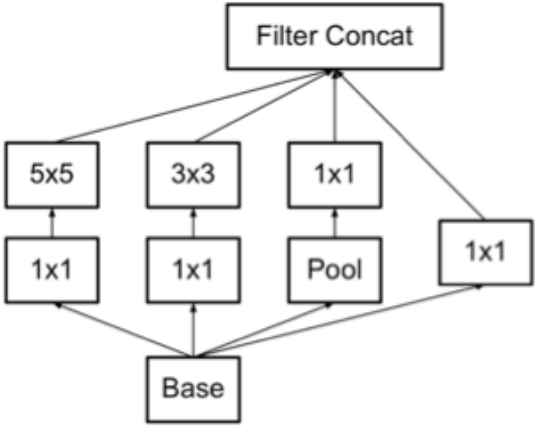
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure and table from original paper

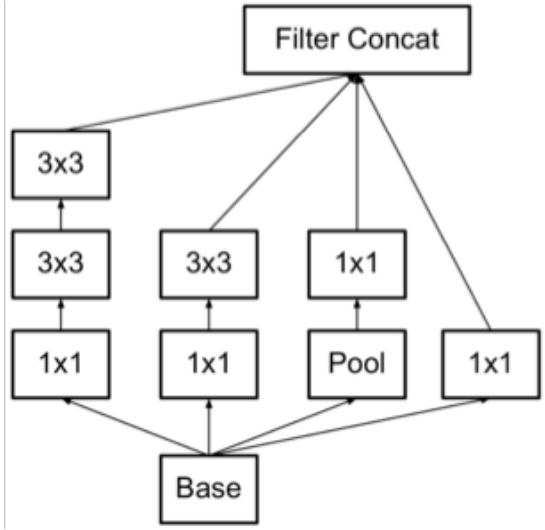
Parallel – Inception V2 And V3

Inception V2 factors 5x5 filters to a stack of two 3x3 filters, replaces some 3x3 with 1x3 and 3x1; Inception V3 adds a module with a 7x7 filter replaced by three 3x3 filters and modifies the training to use a RMSprop solver with label smoothing and aux heads

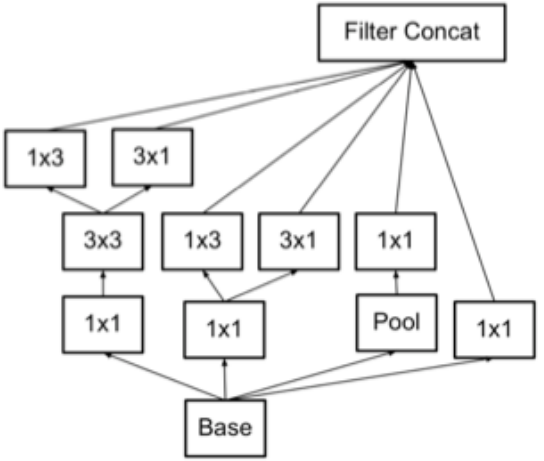
Inception V1



"Figure 5"



"Figure 7"



Figures from original paper

Parallel – Inception V2 And V3

Inception V2 factors 5x5 filters to a stack of two 3x3 filters, replaces some 3x3 with 1x3 and 3x1; Inception V3 adds a module with a 7x7 filter replaced by three 3x3 filters and modifies the training to use a RMSprop solver with label smoothing and aux heads

type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

"Figure 6" with n = 7

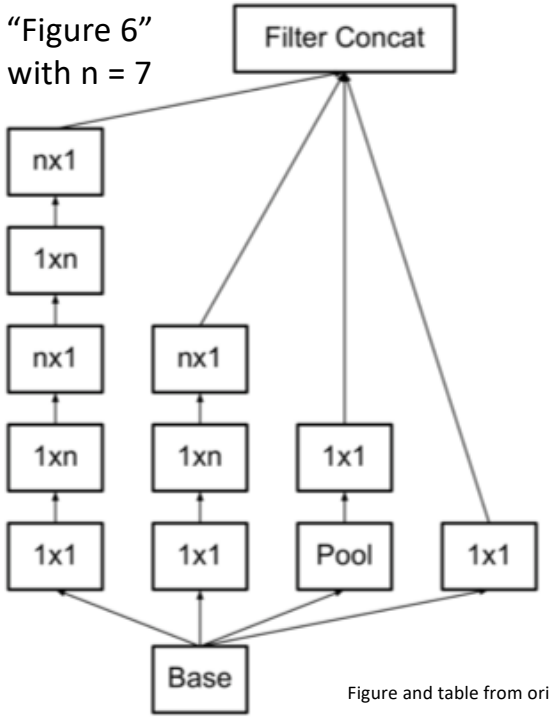
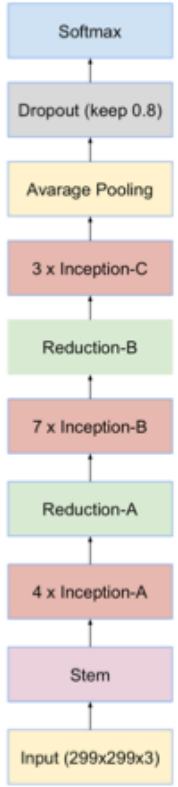
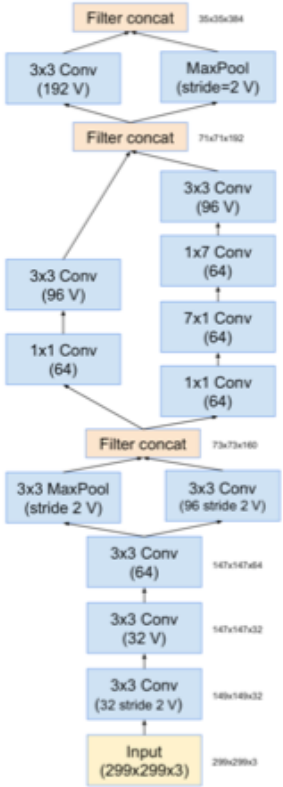


Figure and table from original paper

Parallel – Inception V4

- Custom
  - 1 tail block (stem)
  - 3 different inception blocks
  - 2 different reduction blocks
  - 1 standard head
- Commentary
  - Humans are good at working with regular structures
  - Computers are fine with optimizing towards irregular structures
  - It feels like this network sits somewhere in between

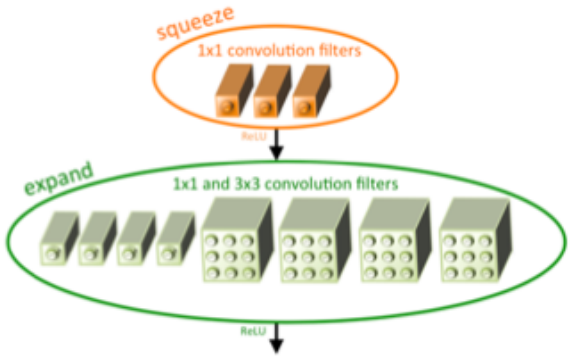




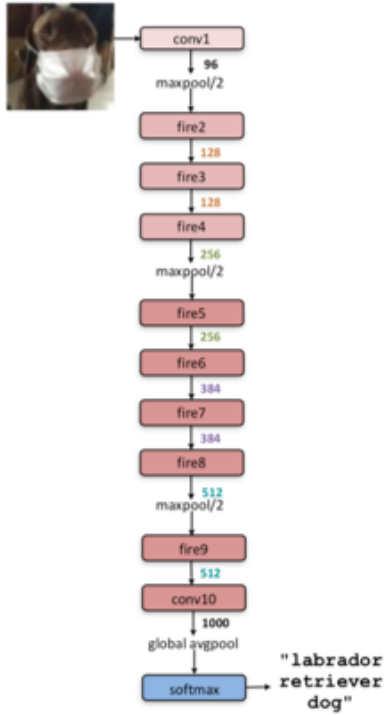
Figures from original paper

Parallel – SqueezeNet

- Included here for historical reasons
  - 1 of the first networks designed with practical implementation constraints in mind
  - Includes a number of features that are common in later designs: parallel structure, separable filters, reduced precision, sparsity, one 1x1 convolution in head, …
- SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size
  - https://arxiv.org/abs/1602.07360



Notes
- Network is very small
- Claims of size are correct (and significant at the time), but AlexNet baseline is inefficient relative to modern designs
- 1x1 filters in the squeeze portion create an information bottleneck, the 1x1 and 3x3 filters in the expand module allow different receptive field sizes



Figures from original paper

Parallel – SqueezeNet

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations		parameters				compression info				1,248,424 (total)	**421,098** (total)

Question

- How small can a network be to solve a problem to a given level of accuracy?
- Maybe a nice project?

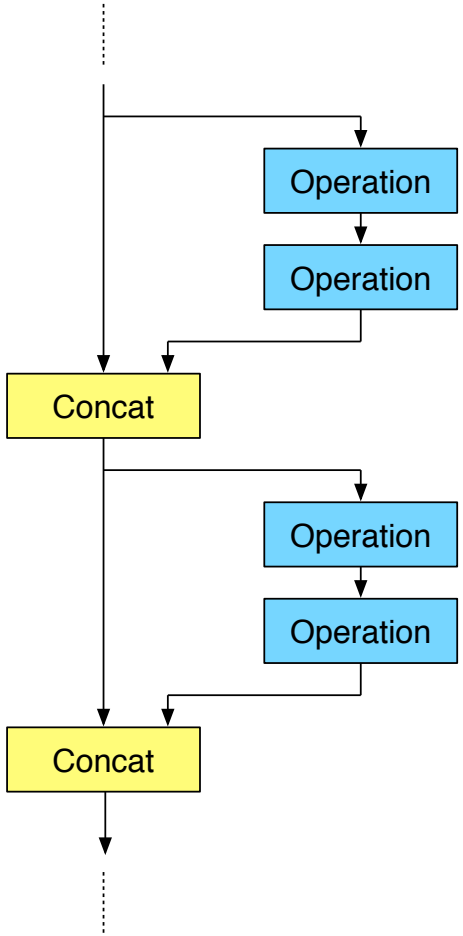
Visualization

- https://dgschwend.github.io/netscope/#/preset/squeezenet
- https://dgschwend.github.io/netscope/#/preset/squeezenet_v11

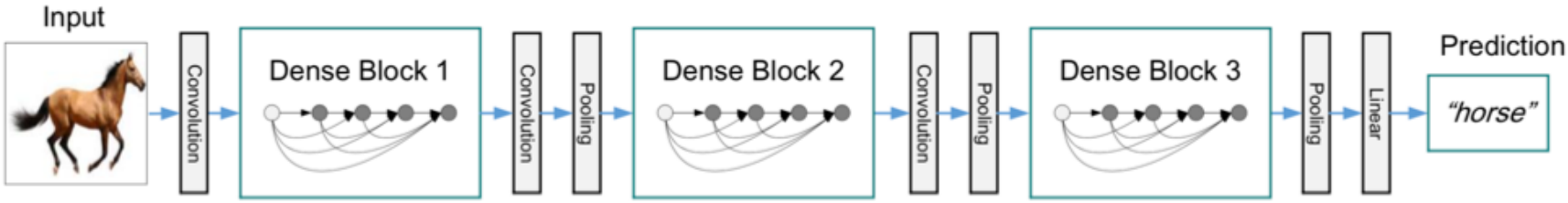
Table from original paper

Dense

- Per building block
  - Input split into 2 paths
  - Direct and operations
  - Concatenate
  - Repeat
- Adds width to the network in the form of features at different depths (applied nonlinearities)
- Examples
  - DenseNet



Dense – DenseNet



- Feed forward style dense connection
- Dense blocks use CNN style 1x1 convolution to reduce channels followed by CNN style 3x3 convolution to mix across channels and space
- Number of channels added per the above operation is referred to as the growth rate of the network (networks can get wide fast)
- Densely connected convolutional networks
  - https://arxiv.org/abs/1608.06993

Notes

- Layer d output is a function of earlier feature maps within the dense block
  $\mathbf{x}_d = h([\mathbf{x}_{d-1}, \mathbf{x}_{d-2}, ..., \mathbf{x}_0])$
- It feels like there are some similarities in philosophy to stacked networks and the Volterra series
- Other dense connection structures are possible
  - Multi-scale dense networks for resource efficient image classification
  - https://arxiv.org/abs/1703.09844

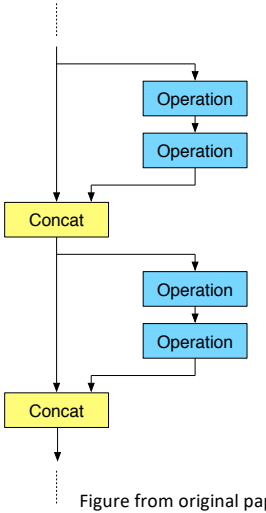


Figure from original paper

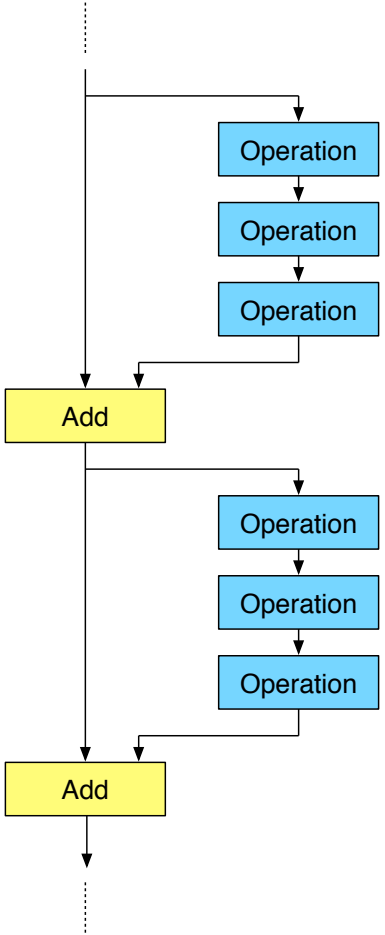
Dense – DenseNet

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table from original paper

Residual

- Per building block
  - Input split into 2 paths
  - Direct and operations (residual)
  - Add 2 paths together to get output
- Output is input + perturbation
  - Clean information flow allows for the training of very deep networks
  - Many options for perturbations (a general concept that can be added to many networks composed of building blocks)
- Examples
  - ResNet
  - ResNeXt / MobileNet V2 / Xception
  - ShuffleNet and ShiftNet
  - ResNet Inception and ResNet DenseNet
  - Squeeze and excitation networks



Residual – ResNet

- Observation
  - Depth in networks is limited by training
  - By designing a residual block that allows the gradient to flow cleanly between layers much deeper networks are enabled
- Residual block
  - Standard layer: $x_{d+1} = f_d(x_d)$
  - Residual block: $x_{d+1} = f_d(x_d) = x_d + h_d(x_d)$
  - Options for the perturbation $h_d(x_d)$ presented in the paper
    - 3x3 conv, 3x3 conv
    - 1x1 conv, 3x3 conv, 1x1 conv (bottleneck)
- Deep residual learning for image recognition
  - https://arxiv.org/abs/1512.03385
- Identity mappings in deep residual networks
  - https://arxiv.org/abs/1603.05027
- Residual connections encourage iterative inference
  - https://research.fb.com/wp-content/uploads/2018/03/residual-connections-encourage.pdf

Nota bene

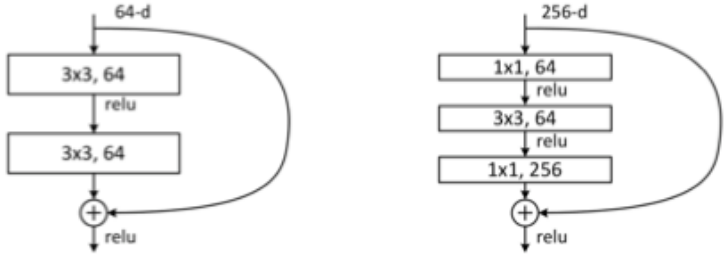


Figure from original paper

Notes
- Increases depth very nicely
- Need to handle increases to the number of channels specially (3 methods presented)
- Some technicalities for keeping the residual path as identity like as possible (see paper 2)
- Want both branches at add points to be both positive and negative, drawback is that this potentially decreases compressibility of feature maps

Visualization
- https://dgschwend.github.io/netscope/#/preset/resnet-50
- https://dgschwend.github.io/netscope/#/preset/resnet-152

Residual – ResNet

- The error gradient back propagates in identity + perturbation form too
  - Given: $\partial e/\partial \mathbf{x}_{d+1}$
  - Find: $\partial e/\partial \mathbf{x}_{d}$
  - Solution:

$$\partial e/\partial \mathbf{x}_d = (\partial \mathbf{x}_{d+1}/\partial \mathbf{x}_d)\,(\partial e/\partial \mathbf{x}_{d+1}) \quad \text{Chain rule}$$

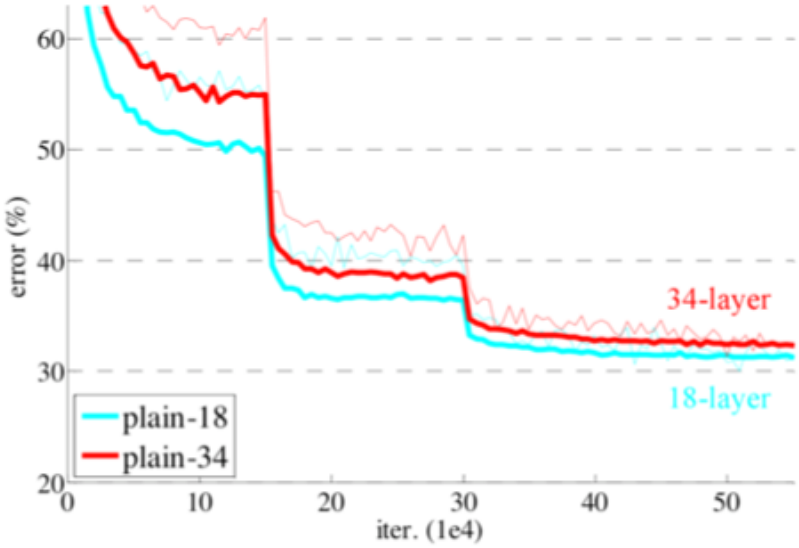
$$= (\partial \mathbf{f}_d/\partial \mathbf{x}_d)\,(\partial e/\partial \mathbf{x}_{d+1}) \quad \text{Sub } \mathbf{f}_d \text{ for } \mathbf{x}_{d+1}$$

$$= (\partial \mathbf{x}_d/\partial \mathbf{x}_d + \partial \mathbf{h}_d/\partial \mathbf{x}_d)\,(\partial e/\partial \mathbf{x}_{d+1}) \quad \text{Sub } \mathbf{f}_d = \mathbf{x}_d + h_d(\mathbf{x}_d)$$

$$= (1 + \partial \mathbf{h}_d/\partial \mathbf{x}_d)\,(\partial e/\partial \mathbf{x}_{d+1}) \quad \text{Note } \partial \mathbf{x}_d/\partial \mathbf{x}_d = 1$$

$$= \partial e/\partial \mathbf{x}_{d+1} + (\partial \mathbf{h}_d/\partial \mathbf{x}_d)\,(\partial e/\partial \mathbf{x}_{d+1}) \quad \text{Linear operator}$$

Residual – ResNet



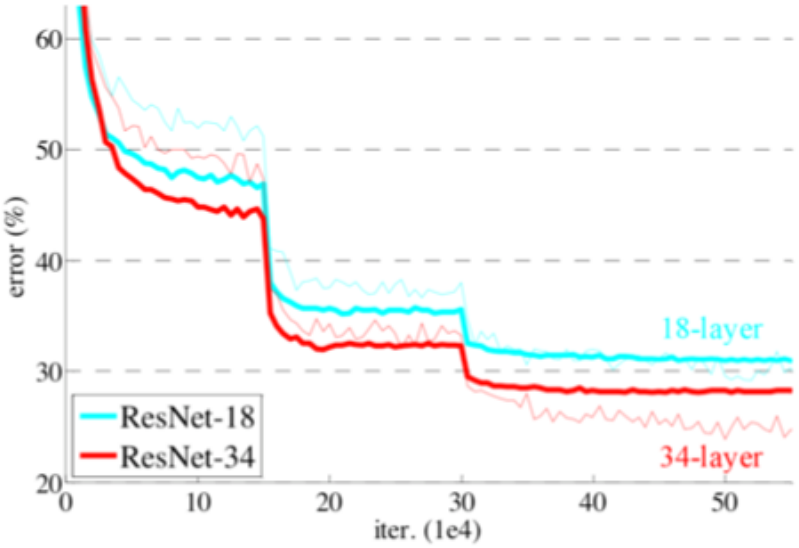


Figure from original paper

Residual – ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

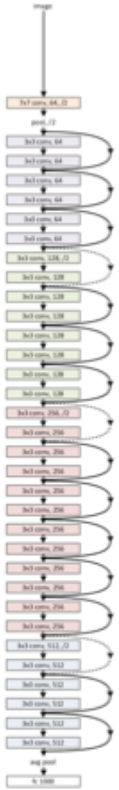
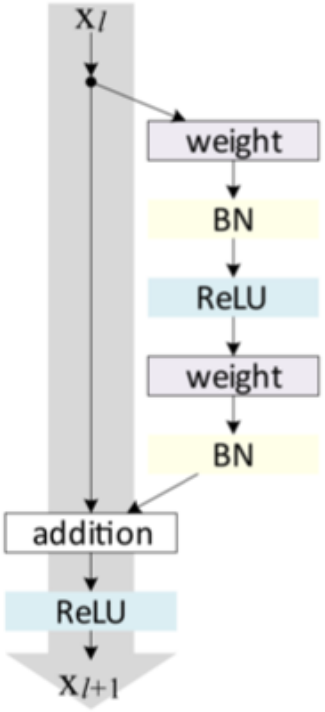


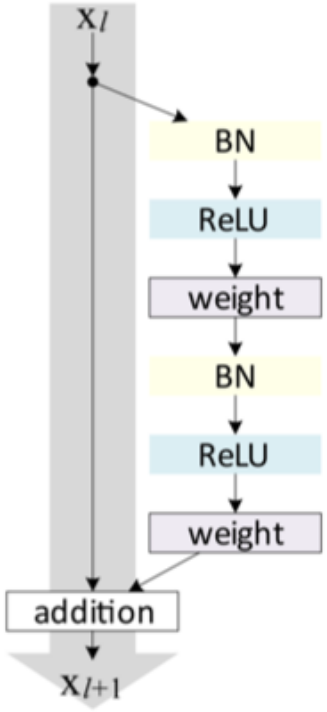
Figure and table from original paper

Residual – ResNet

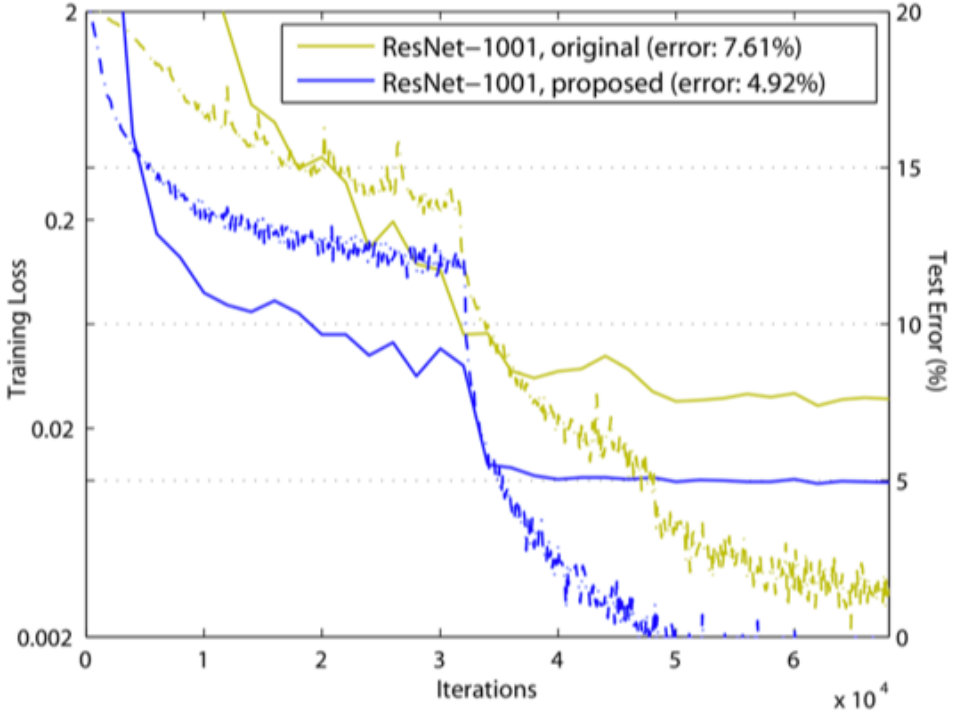
Original



Paper 2



CIFAR 10 Accuracy



Figures from original paper

References

List

- References for individual networks are listed on the slide on which they occur

- Mathematics of deep learning
  - https://arxiv.org/pdf/1712.04741.pdf
  - http://www.vision.jhu.edu/tutorials/ICCV17-Tutorial-Math-Deep-Learning-Intro-Rene.pdf
- On the number of linear regions of deep neural networks
  - https://arxiv.org/abs/1402.1869
- An introduction to deep learning (lecture 1)
  - http://www.cs.toronto.edu/~ranzato/files/ranzato_deeplearn17_lec1_vision.pdf
- Image classification with deep learning
  - http://www.cs.toronto.edu/~ranzato/files/ranzato_CNN_stanford2015.pdf
- Batch normalization: accelerating deep network training by reducing internal covariate shift
  - https://arxiv.org/abs/1502.03167
- Group normalization
  - https://arxiv.org/abs/1803.08494

List

- Improving neural networks by preventing co-adaptation of feature detectors
  - https://arxiv.org/abs/1207.0580
- VGG convolutional neural networks practical
  - http://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html
- Deep learning
  - http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf
- Deep learning for AI
  - http://www.iro.umontreal.ca/~bengioy/talks/IJCAI2018-DLtutorial.html
- Netscope
  - https://dgschwend.github.io/netscope/quickstart.html