Design

Arthur J. Redfern
arthur.redfern@utdallas.edu

Feb 13, 2019
Feb 18, 2019
Feb 20, 2019

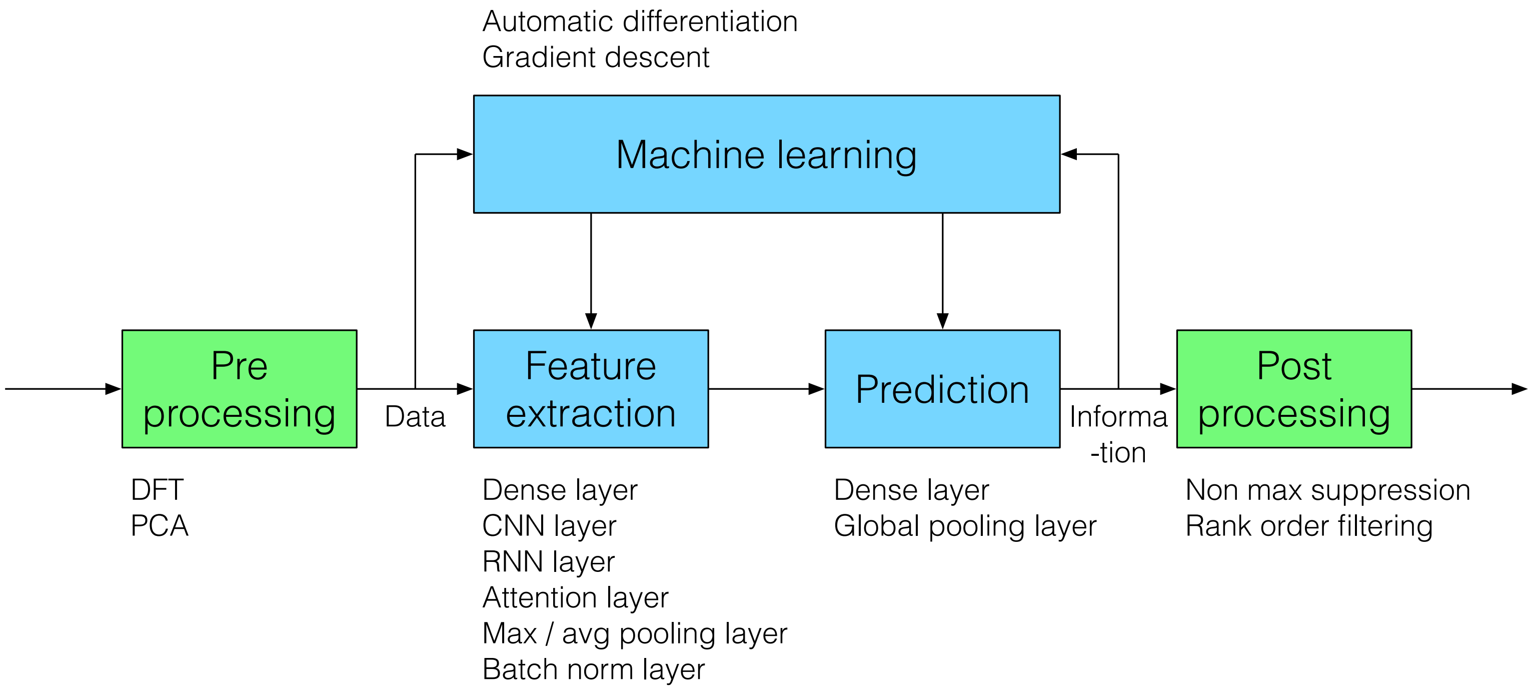
Outline

- Context
- Goal
- Preliminaries
- CNNs
- RNNs
- Attention
- References

Context

The First Part Of The Course

Provided the basic building block needed to design and train a xNN



The Second Part Of The Course

Where we are now

- The first part of the course focused on math and introduced the basic building blocks needed to build and train a xNN

- This is the start of the second part of the course
  - Use these building blocks to design a network to achieve a goal
    - Use an encoder decoder style architecture framework (with a focus on the encoder)
    - Review successful network designs and understand the keys behind them
  - Training to improve convergence and improve generalization through regularization
  - Implementations to make these networks run efficiently in practice which allows them to be used in all sorts of applications

- The third part of the course will then focus on applications: vision, speech, language, ...

Options For Organizing Course Parts 2 and 3

- Option A
  - A.1 Design
  - A.2 Training
  - A.3 Implementation
  - A.4 Vision, speech and language
- Option B
  - B.1 CNN design, training, implementation and vision
  - B.2 RNN design, training, implementation and speech
  - B.3 Attention design, training, implementation and language
- This course will use **option A** for the ordering of material
  - The choice is motivated by the overlap of network types to the subsequent training, implementation and application sections

Goal

Keep Your Goals Simple

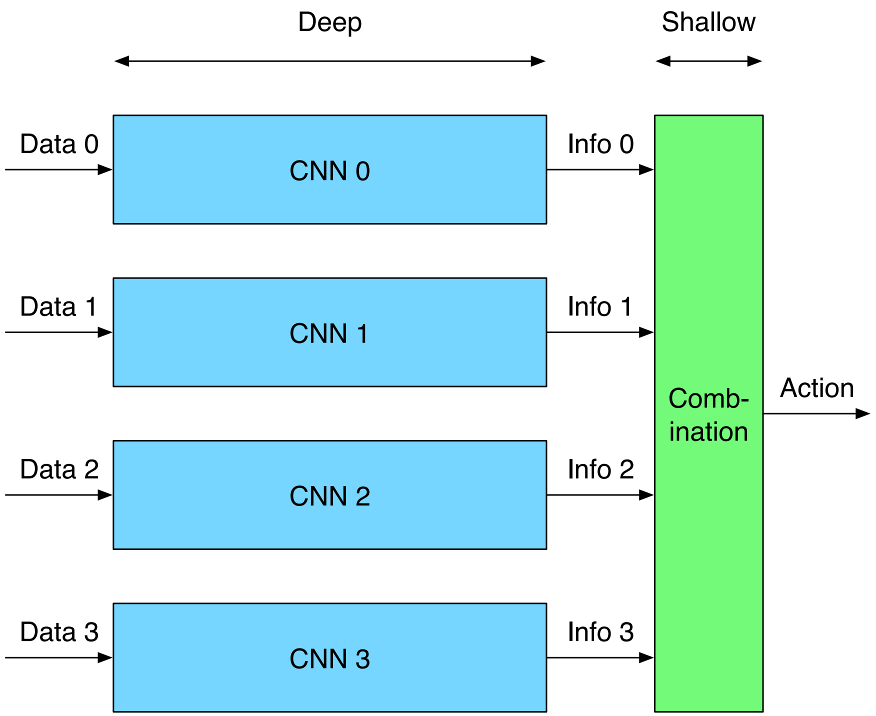
Both a deep network and general life comment

- You design a network to accomplish a goal
  - Never lose sight of this
  - View a xNN as a function you design that maps from data to information (or information to data)

- Keep the problems you address using deep networks like xNNs 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 xNN 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

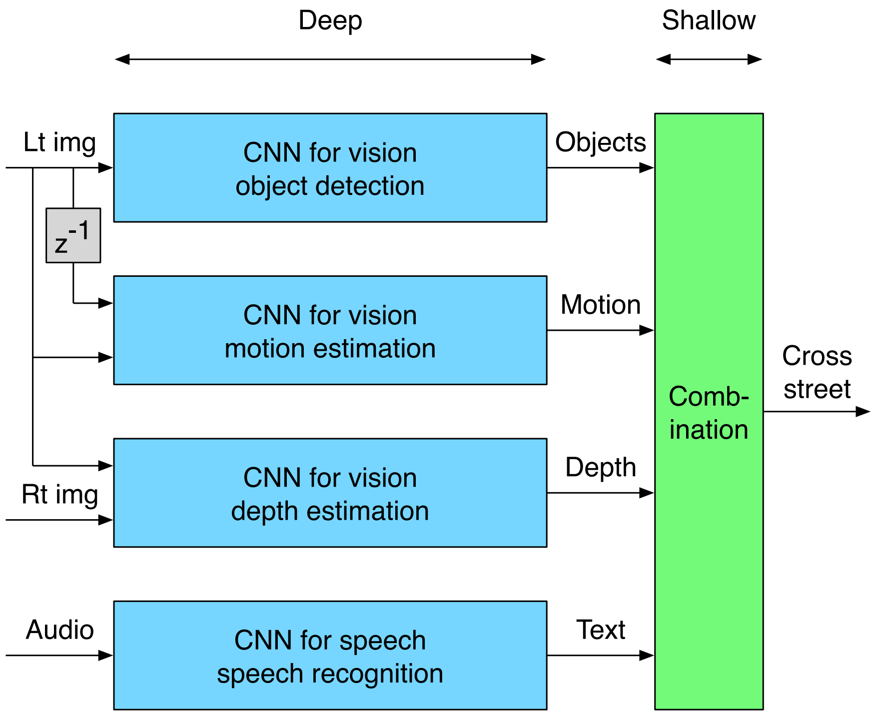
- How do you solve this problem as a human? An incomplete list includes ...

• 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	until across street
• 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 xNNs 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 xNNs
  - 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

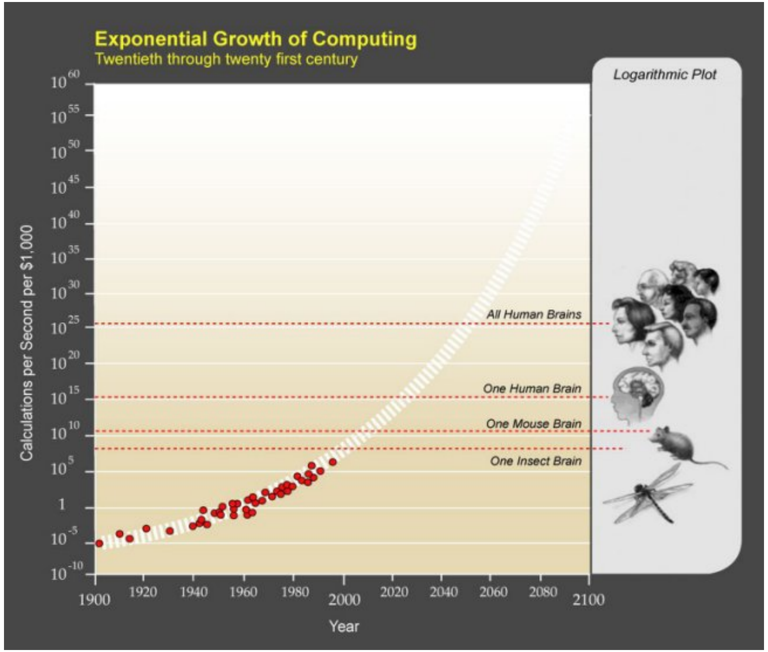
Preliminaries

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 (this is 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?



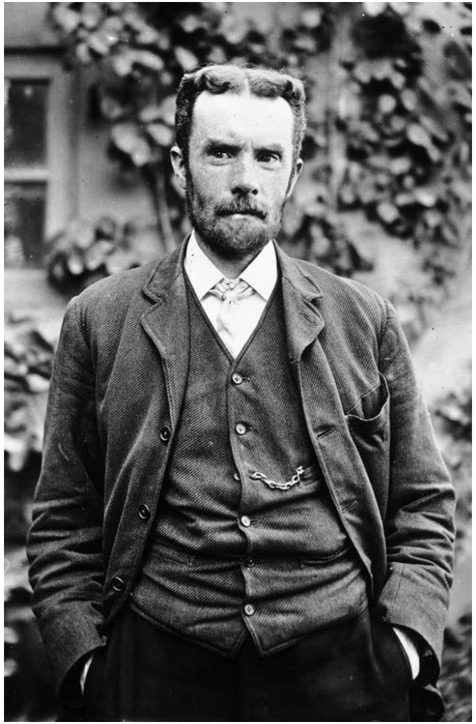
Theory Or Practice?

- Both (and that’s ok, it’s how science works)
- A frequent critique of deep learning is that no one knows why it works
  - 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 already 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?**"

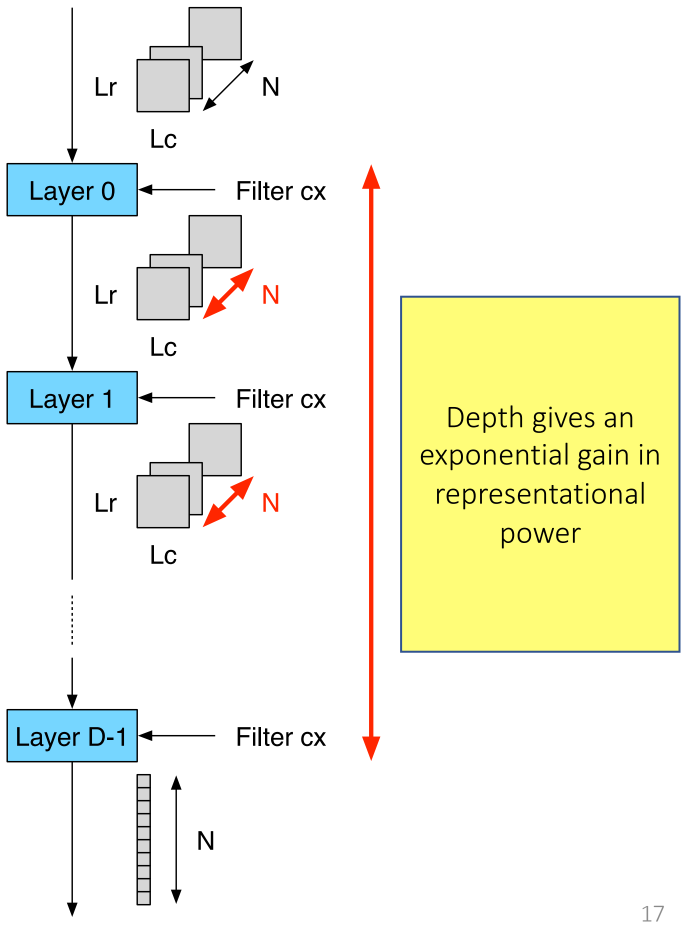
Quotes != proofs, but regardless, I like this



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

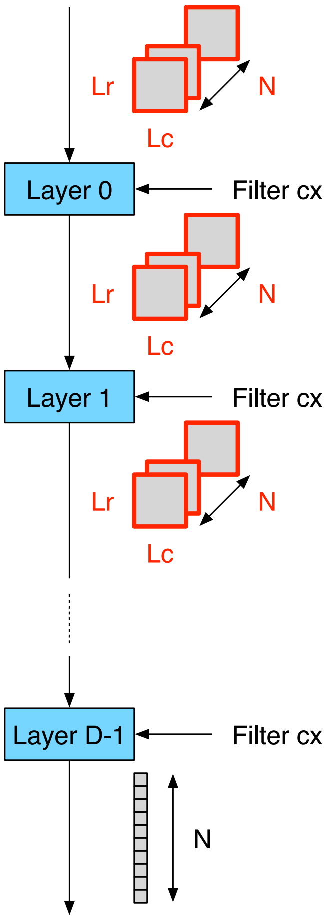
Network Size

- Network depth and width (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
  - The implementation lectures will look at this
  - For now, pay attention to compute at the beginning and memory at the end



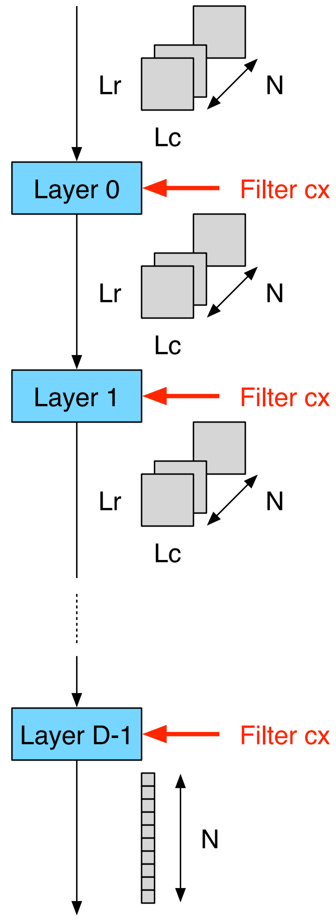
Feature Map Size

- 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

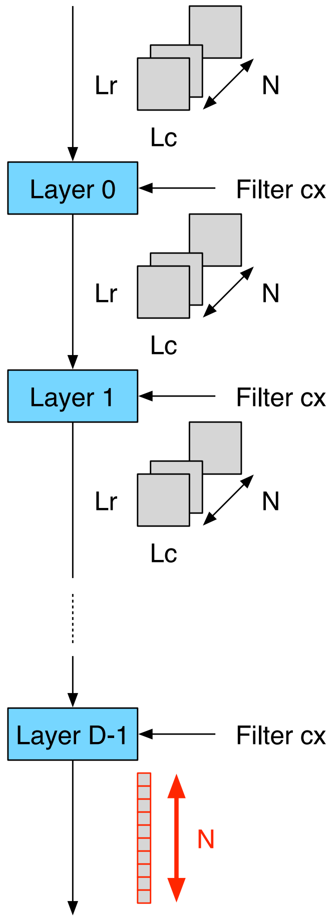


Networks Implement Functions

- A person’s 1st exposure to functions is typically f(x) that maps scalar x to scalar y
  - Ex: $y = f(x) = a x + b$
  - Ex: $y = f(x) = \sin(x)$
- xNNs also implement functions that map inputs to outputs
  - Will typically use scalars, vectors and tensors for inputs
    - Allows xNNs to handle all sorts of inputs like images, speech and language
  - Will typically use a pmf vector or multiple pmfs arranged as a tensor for outputs
    - Allows xNNs to handle all sorts of classification problems
- The input output mapping of the function (xNN) is determined by the choice of layers, their parameters and the structure in which they’re connected
  - The loss function is how to desired output of the function is defined
  - It’s used to modify the weights to make the output closer to the goal

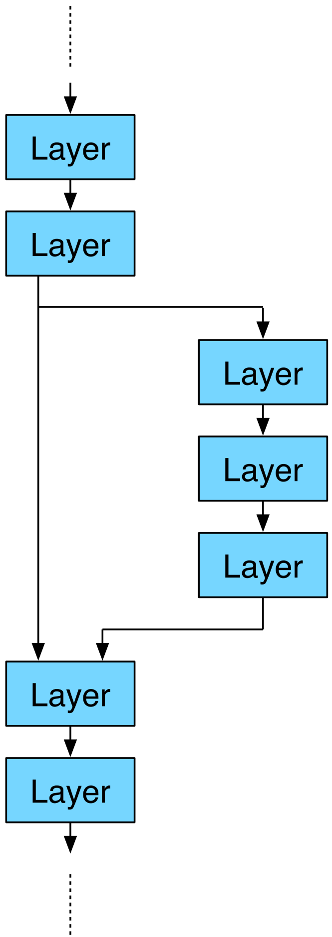
Problem Complexity

- 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)



CNNs

CNN Design Section Overview

- CNNs exploit spatial structure in data
  - Keep this in mind as you're thinking about applying them to a problem
- Overview
  - Encoder (tail and body) + decoder (head) design strategy with typical tail and head designs
  - A visual review of how layers combine features as a lead into network design
  - Network design framed as different building block structures between the tail and head
    - Serial
    - Parallel
    - Dense
    - Residual
  - Improving designs via ML based architecture search
  - Improving human understanding via visualization

CNNs: Strategy

A Framework For Thinking About CNNs

- Feature extraction and prediction
  - 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

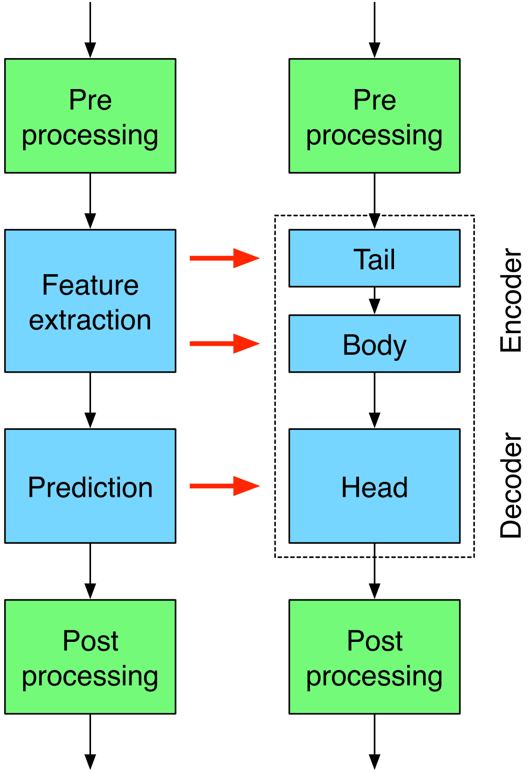
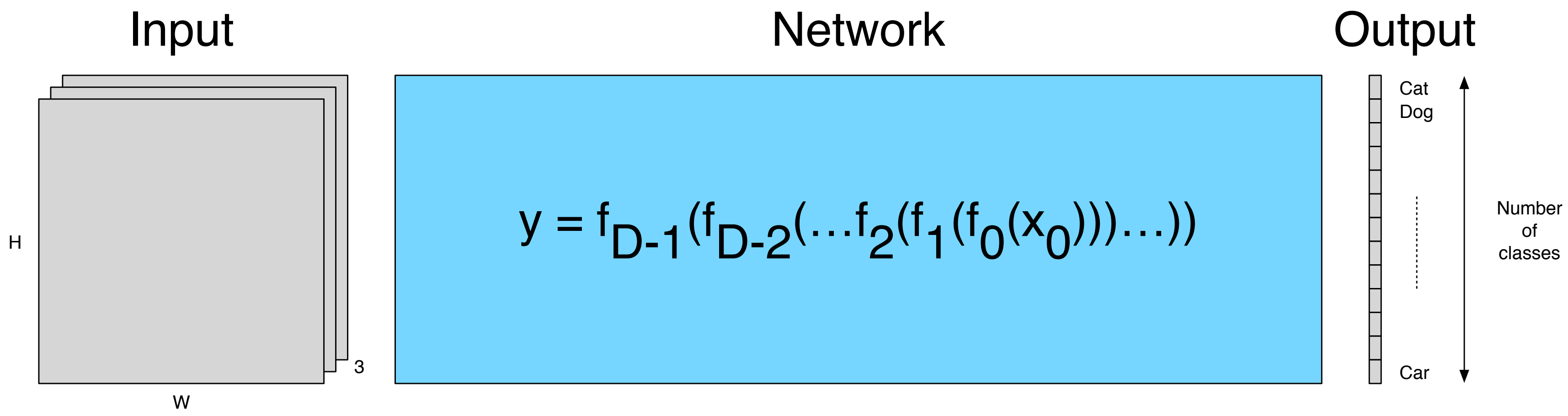


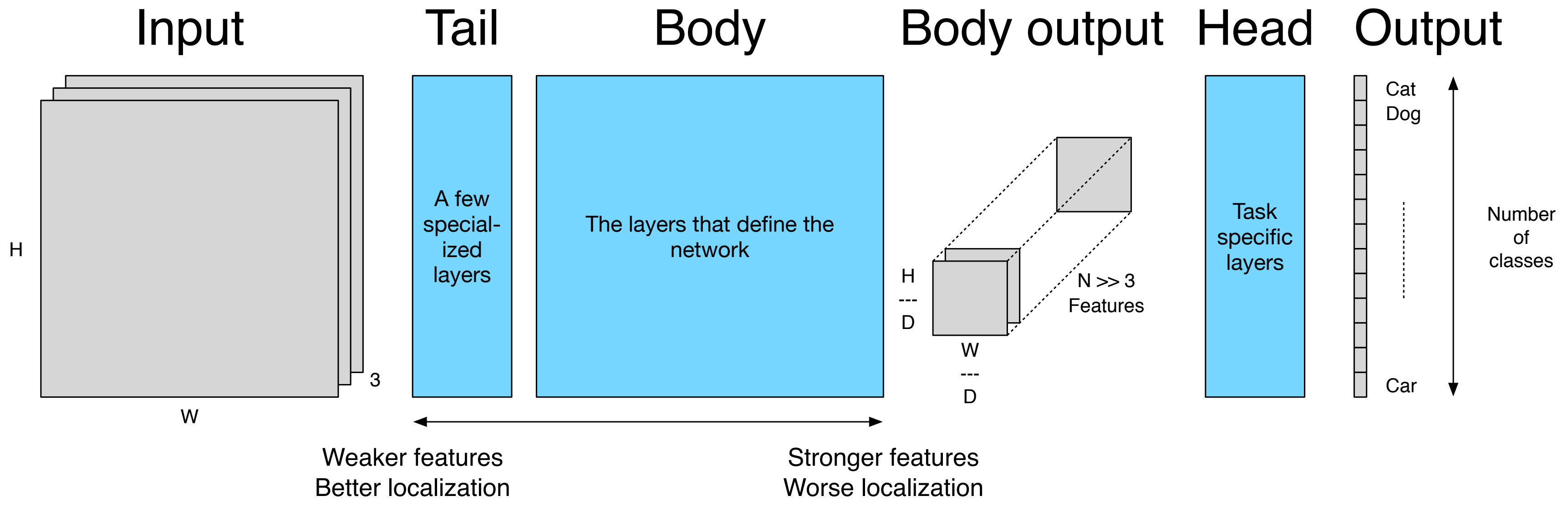
Image Classification As A Starting Point

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

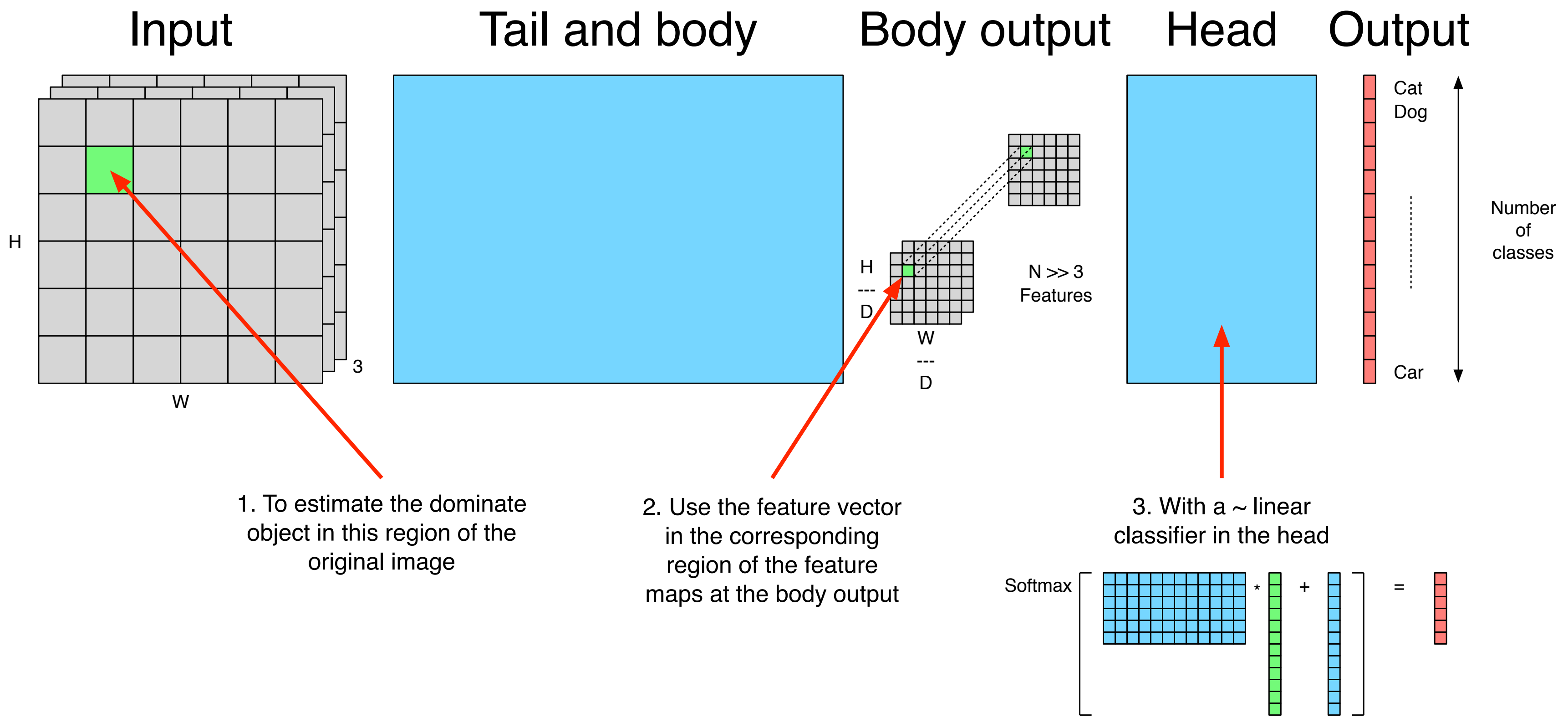


Tail Body And Head Decomposition

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

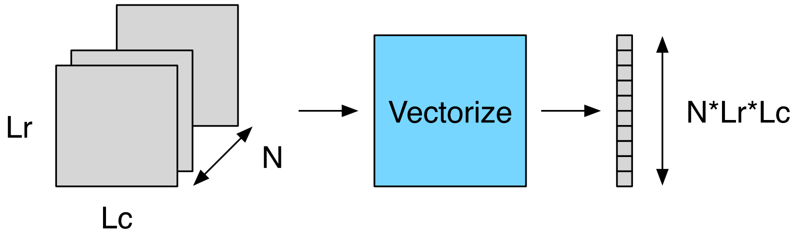


Conceptually How The Head Works

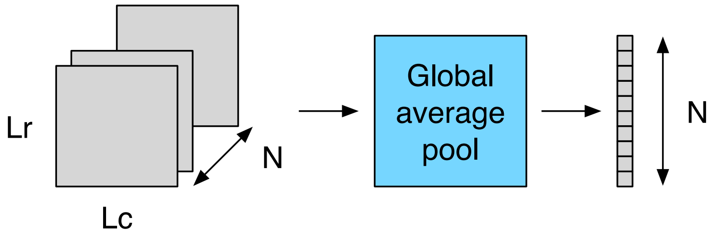


Feature Map To Feature Vector Transform

- 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)

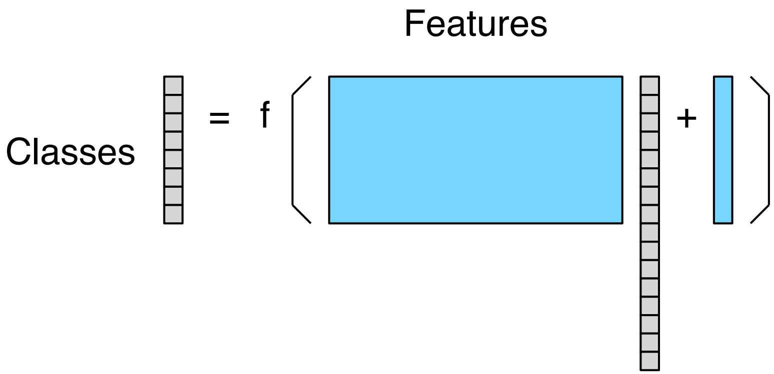
The linear classifier requires an input vector of a fixed size; convolutional layers in the body work on feature maps of ~ arbitrary size; this needs to connect them

Linear Classifier

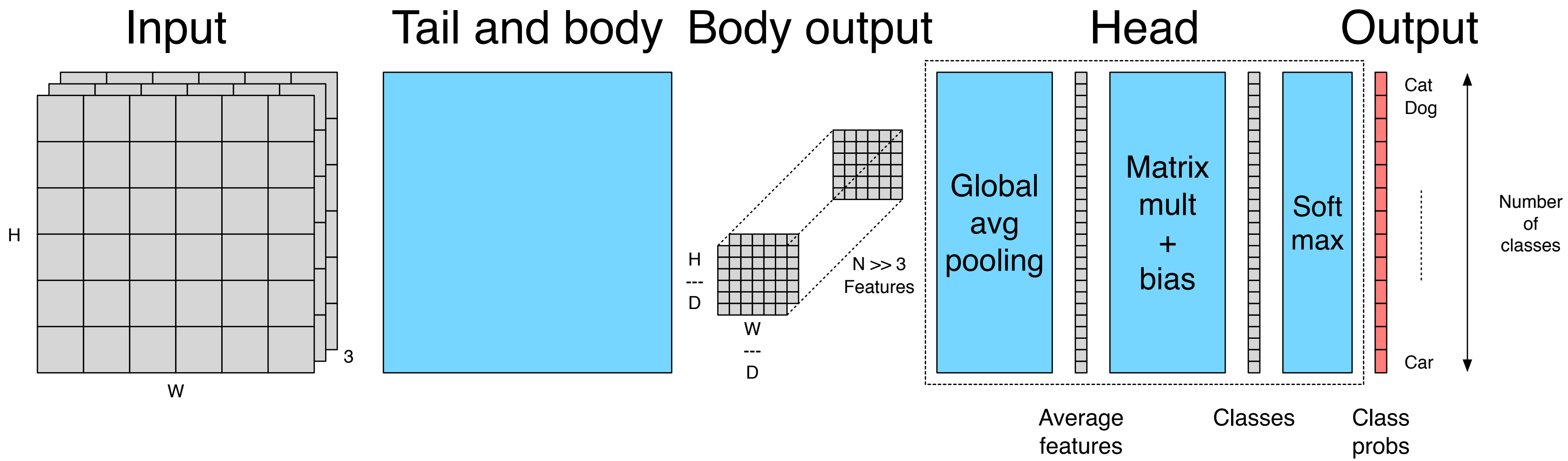
- Linear classifier
  - + softmax to convert to probabilities (training)
  - + arg max to select max output as class (testing)

- 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 Classification Head Design



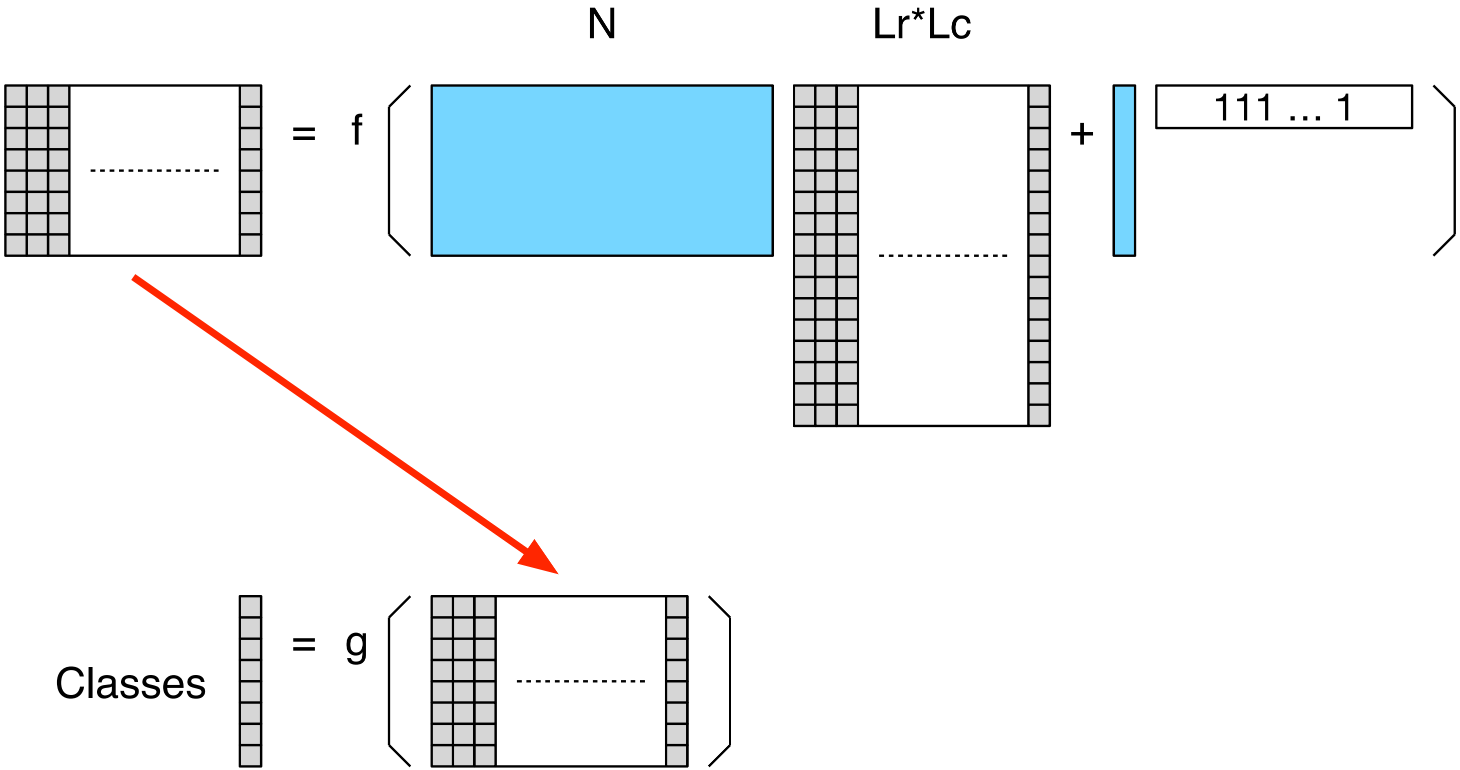
2 Classification Head Design Ideas

- Maybe already done by someone else (I don't know)?
  Maybe doesn't work as well (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

Classification Head Design Idea 1

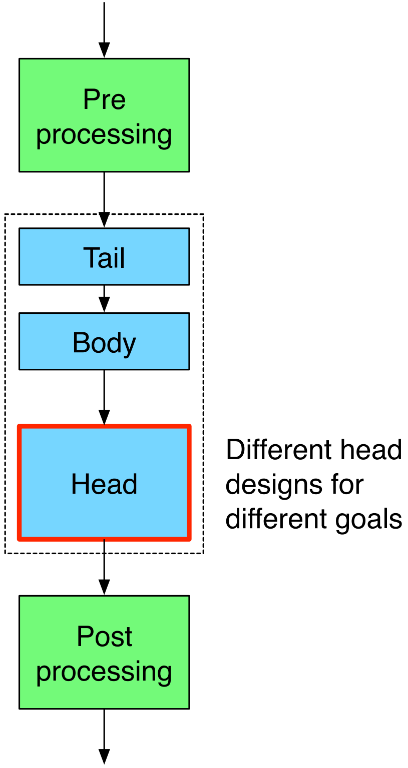


In The Near Future

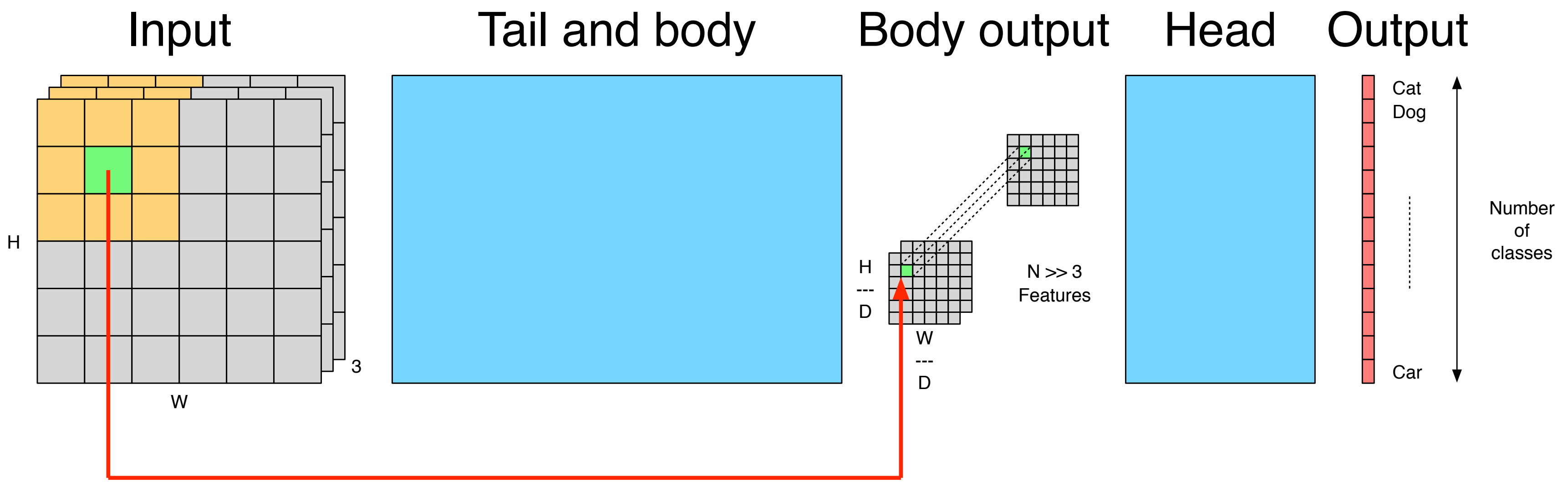
- Specifically, when we look at different applications in more detail

- We're going to look at different head designs (decoders) 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



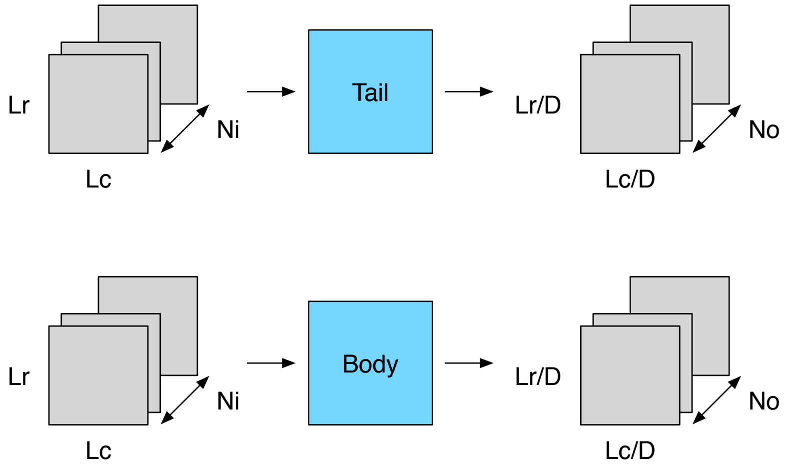
Conceptually How The Tail And Body Work



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

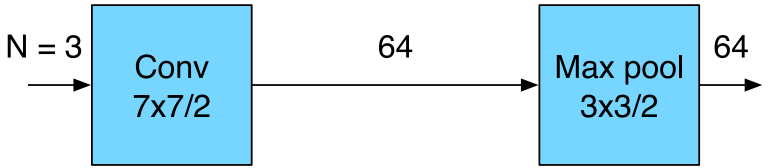
Why Distinguish Between The Tail And Body?

- 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

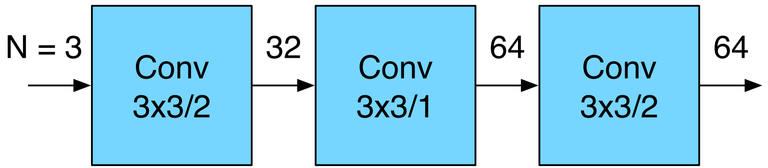


Tail: Data To Weak Features

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

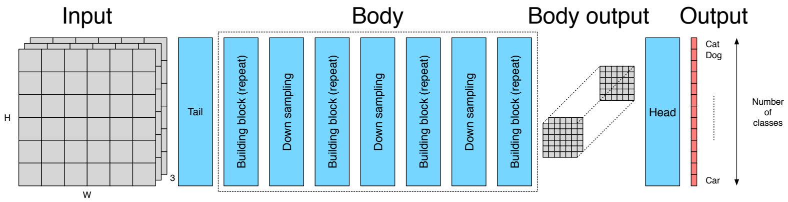


- Features
  - High compute
  - High feature map memory
  - Low parameter memory
  - Lots of spatial redundancy



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

Body: Weak Features To Strong Features

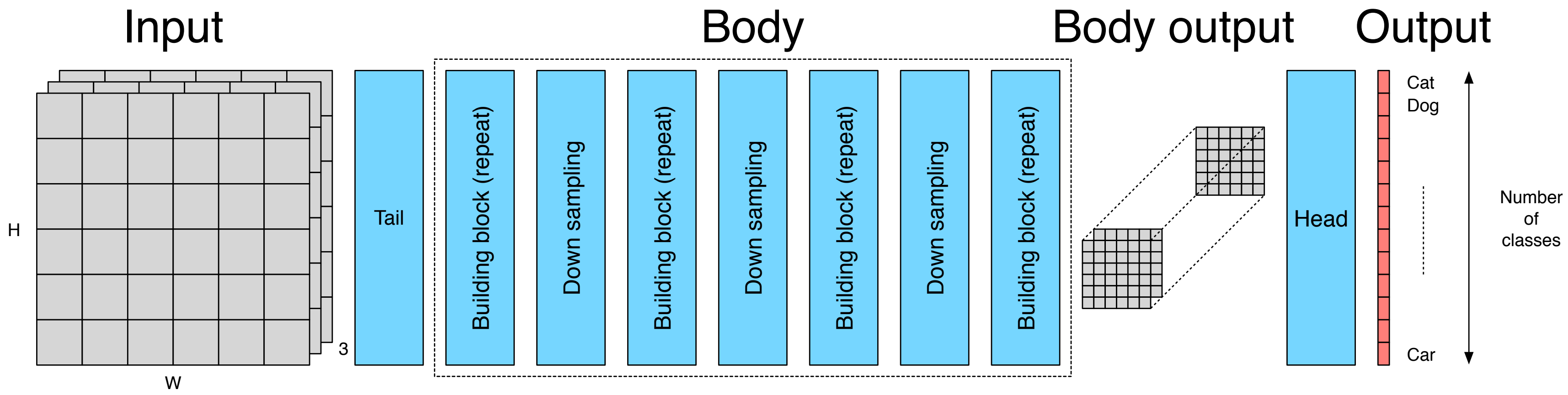


- Building blocks we'll see in network design followed by down sampling (strided convolution or pooling)
- Interpretation
  - Lower level features are more class agnostic
    - Less channels earlier in the network
  - Deeper features are more class specific
    - More channels later in the network
- 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

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)

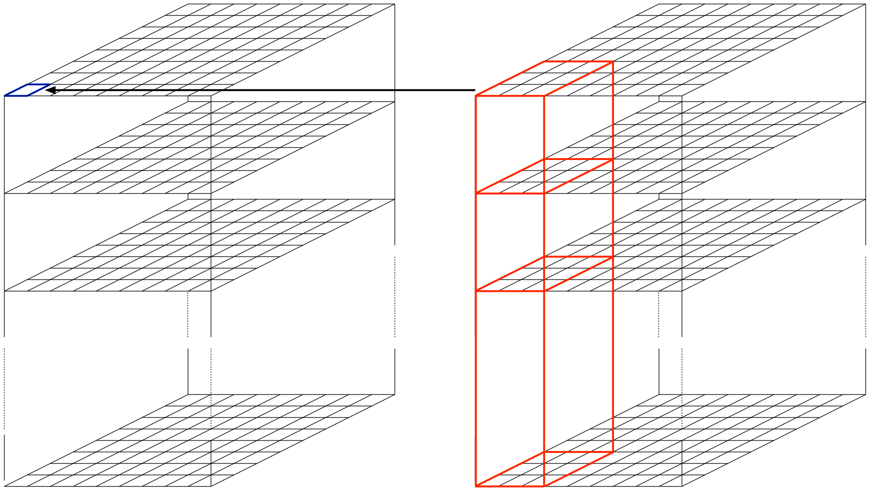
CNNs: Layers

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
  - Rearrangement
  - Training

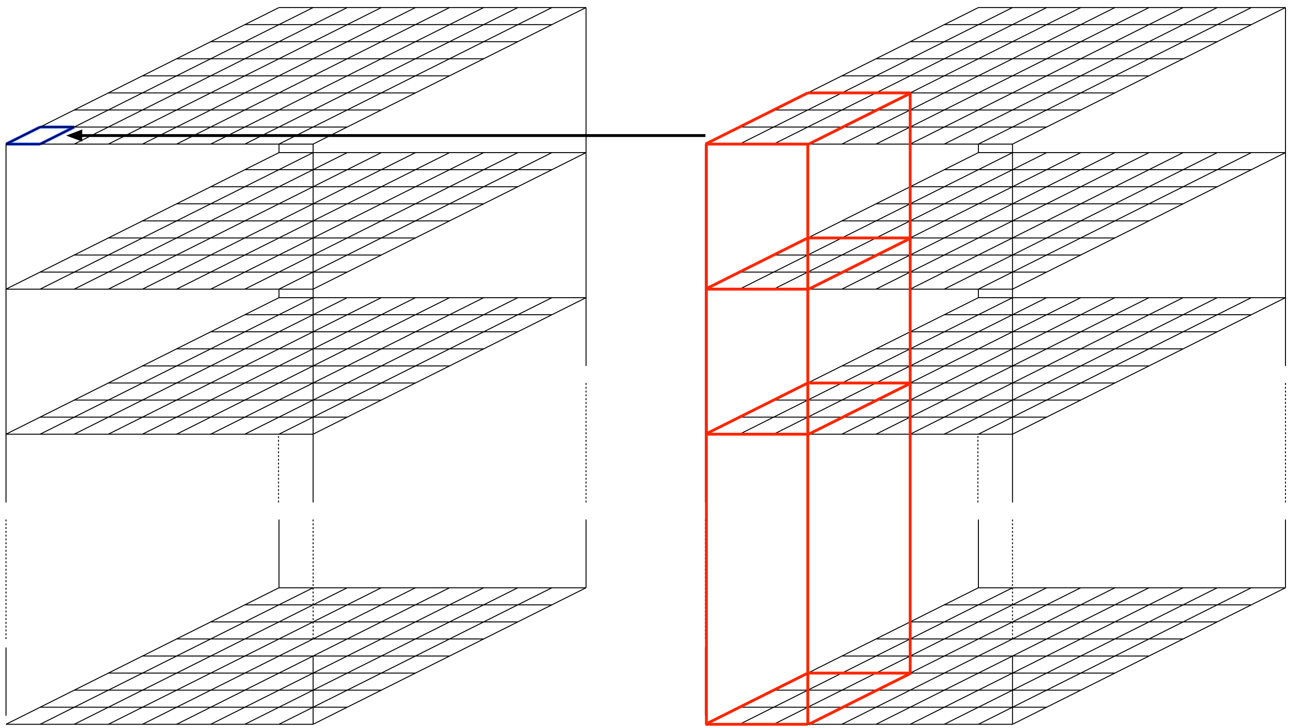
Channel And Space

- 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)



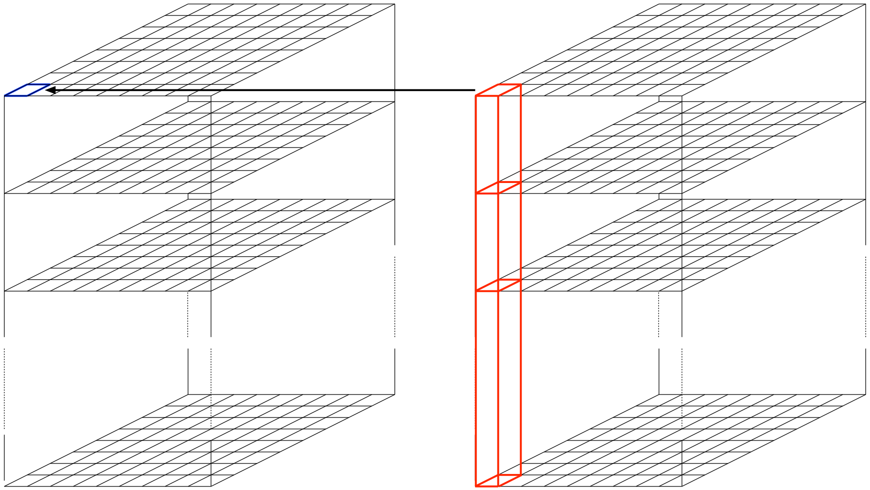
Spatial invariance of CNN style 2D convolution allows for memory and compute reduction but is also plausible with respect to feature extraction from many images

Channel And Space

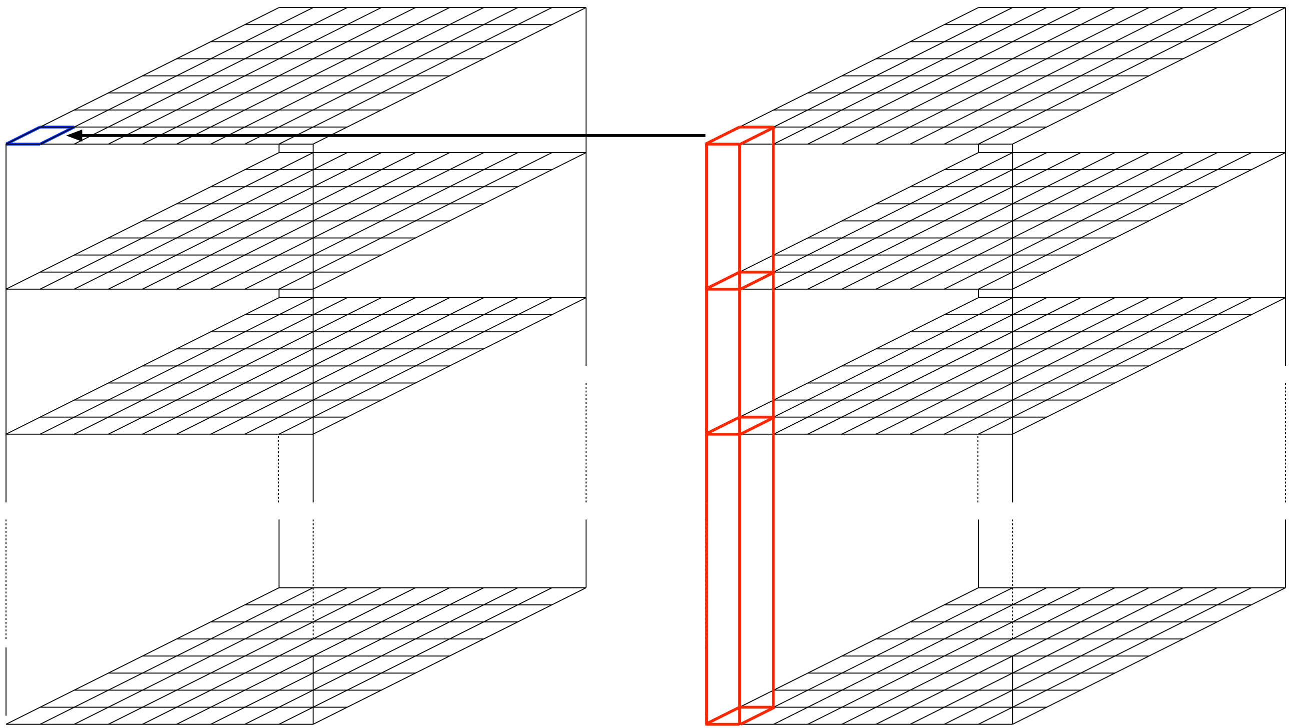


Channel

- 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

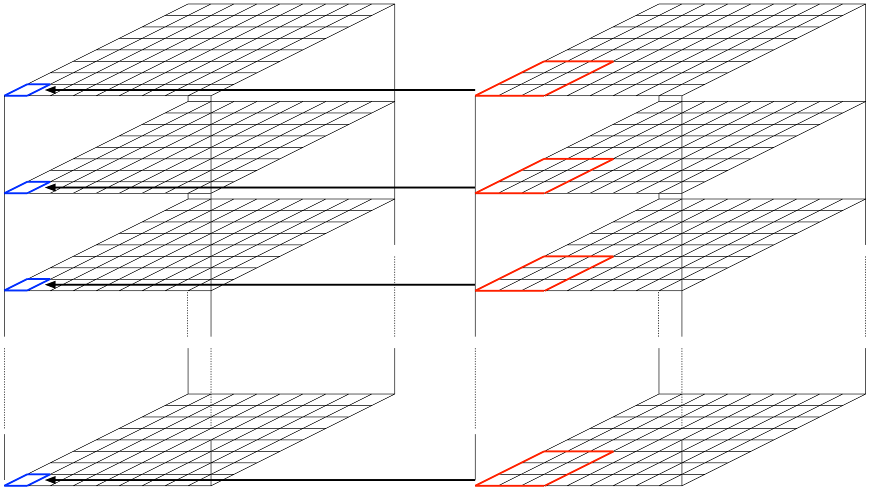


Space

- CNN style 2D convolution that's fully grouped (standard 2D convolution, depth wise separable convolution)

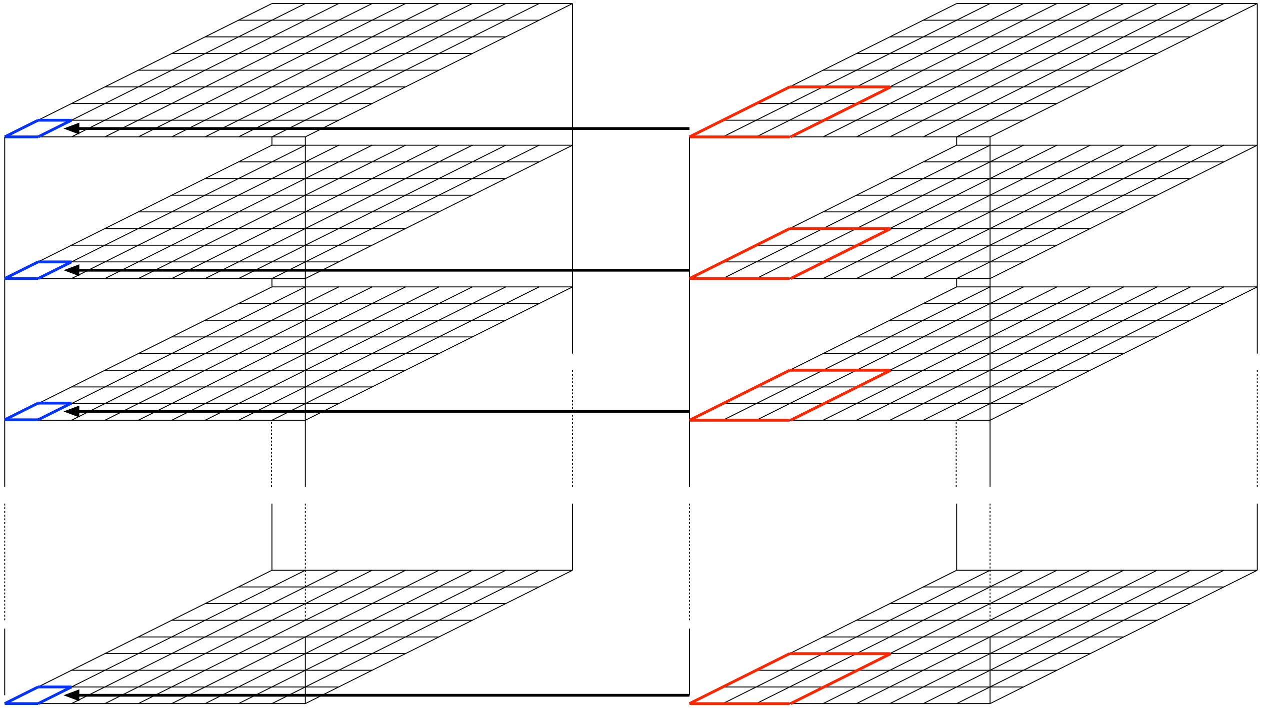
- Decimation
  - Max pooling
  - Avg pooling
  - Global average pooling
  - Spatial pyramid pooling

- Interpolation
  - Nearest neighbor
  - Bilinear



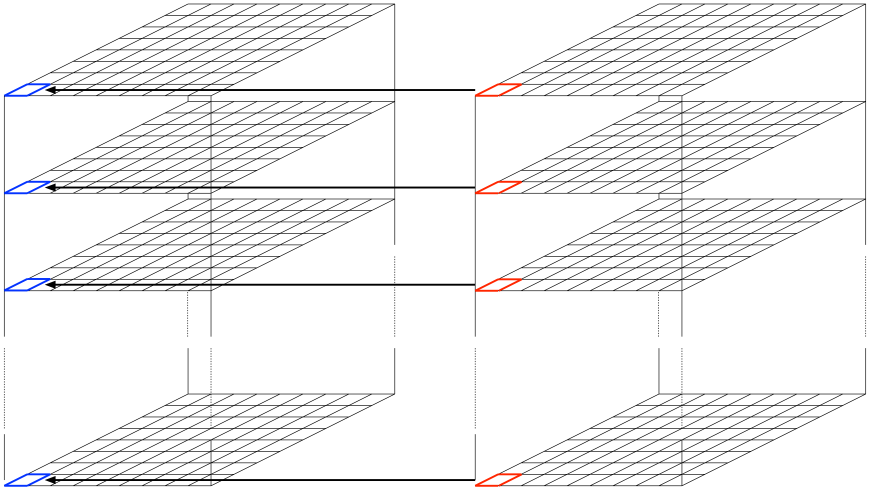
Pooling, as shown in the above figure, allows for the aggregation of larger regions of input features in the generation of output features (increasing receptive field size)

Space

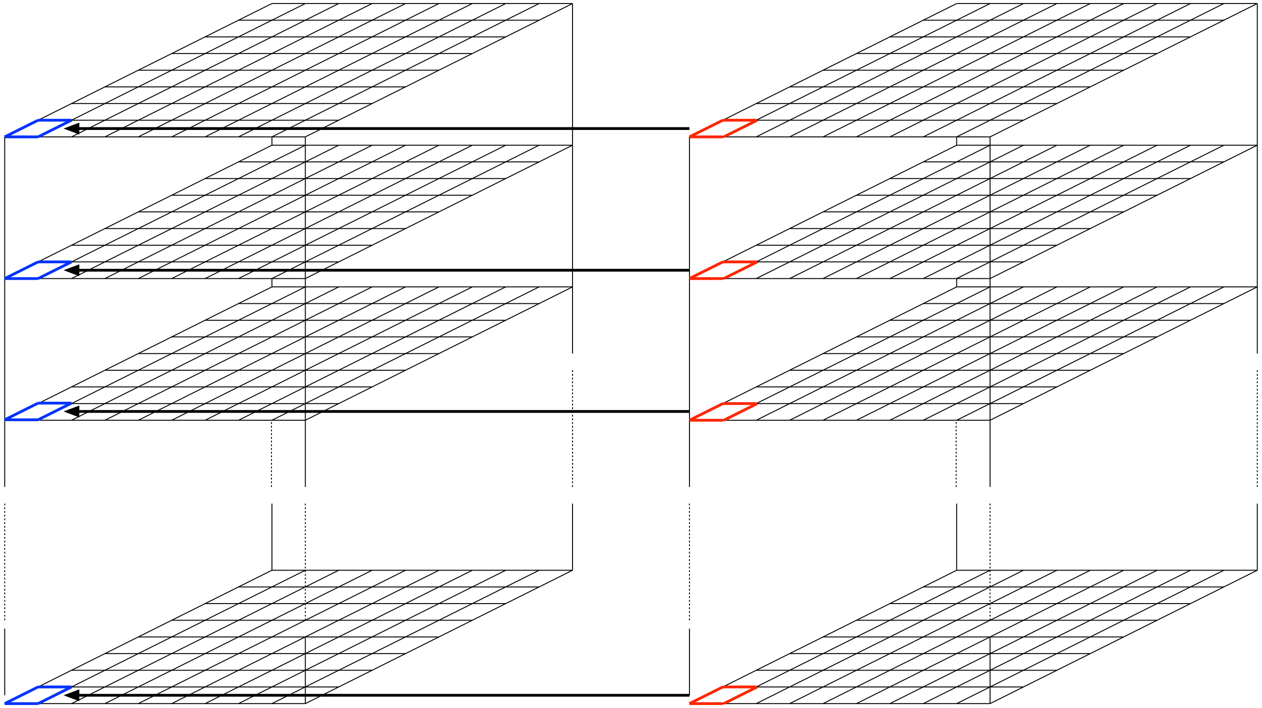


Element

- 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, ...

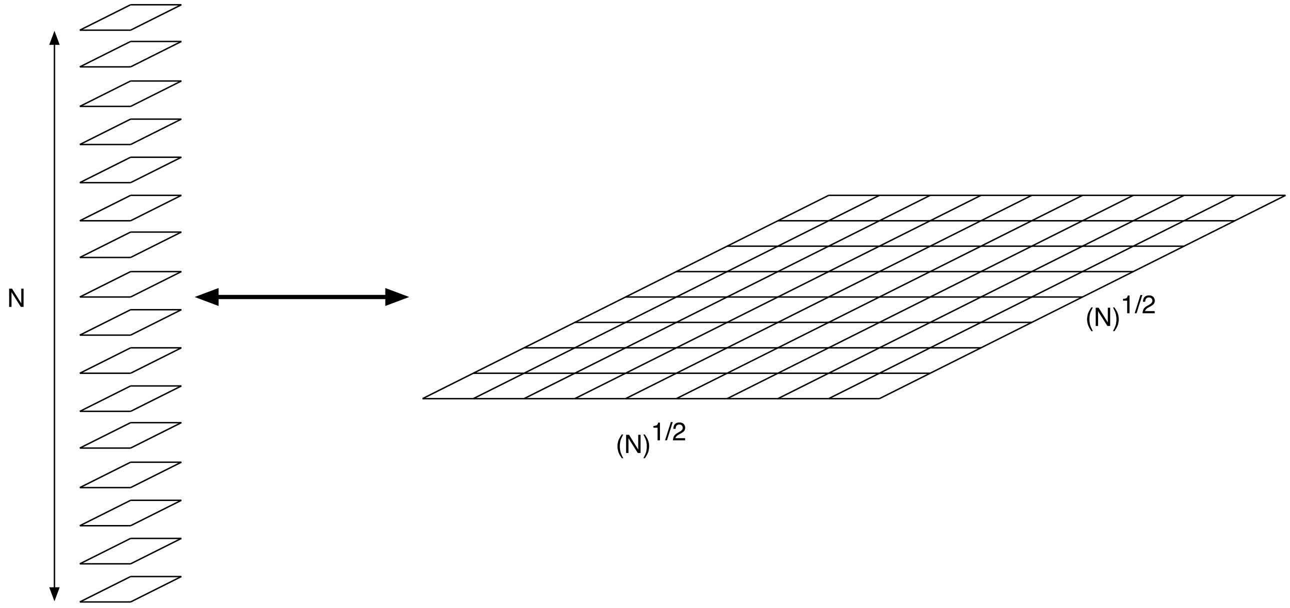


Element



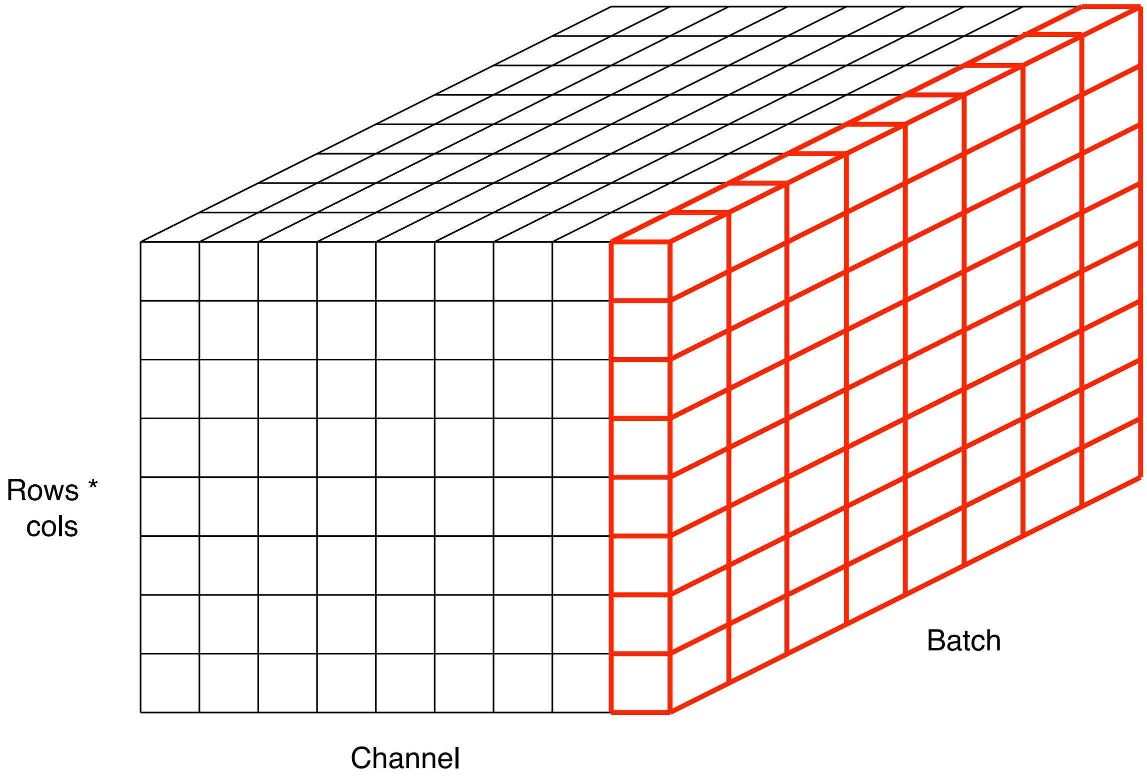
Rearrangement

- 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



CNNs: Networks

Networks Overview

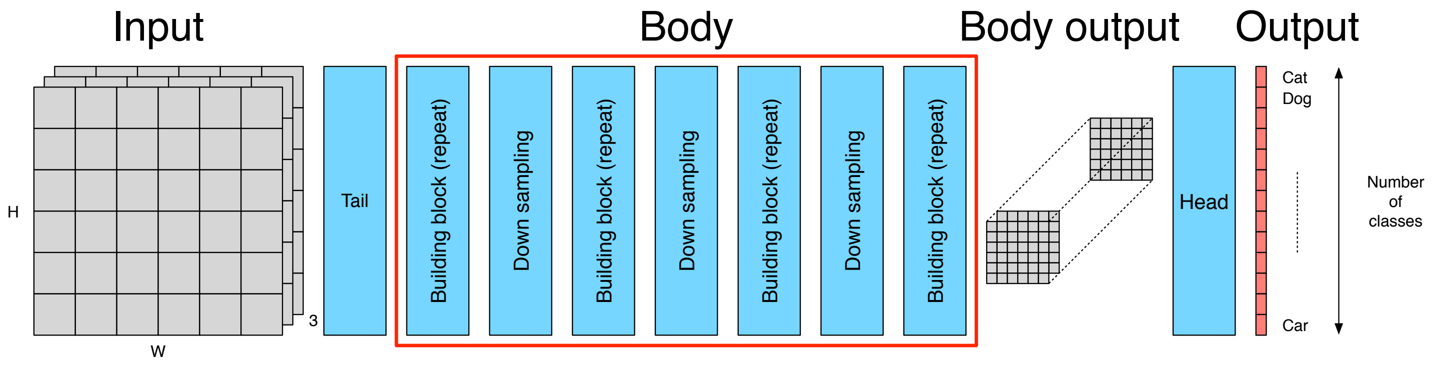
- Illustrated in the context of image classification ...

- 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

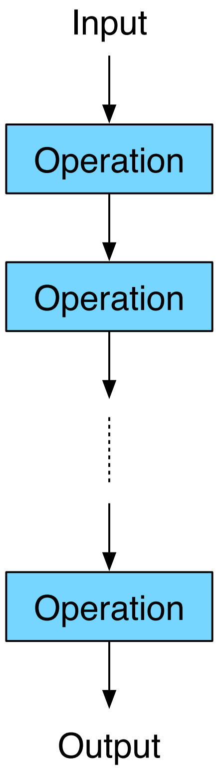
Organization

- 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
  - Serial
  - Parallel
  - Dense
  - Residual
- Will also briefly look at different optimization methods for network design

Serial

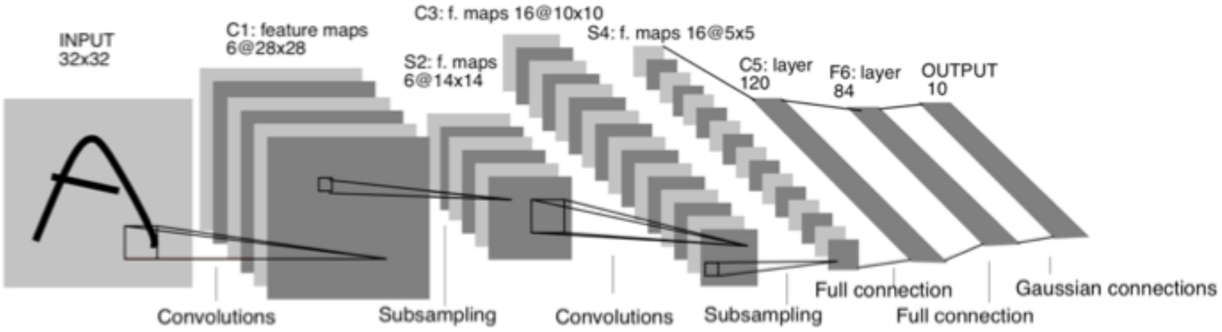
- A sequential set of operations from input to output
  - The historical starting point and how we've thought about network design so far

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



Serial: 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

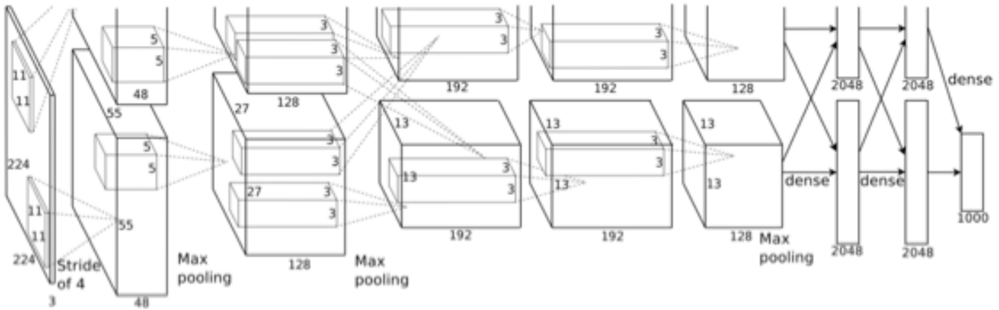


Notes
- While the code approach to creating an error to optimize is out of favor, maybe it's worth thinking about when there are many similar categories and how it potentially can be used with / relates to a hierarchical head design, creating multiple targets for a single category, ...

Figure from original paper

Serial: 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

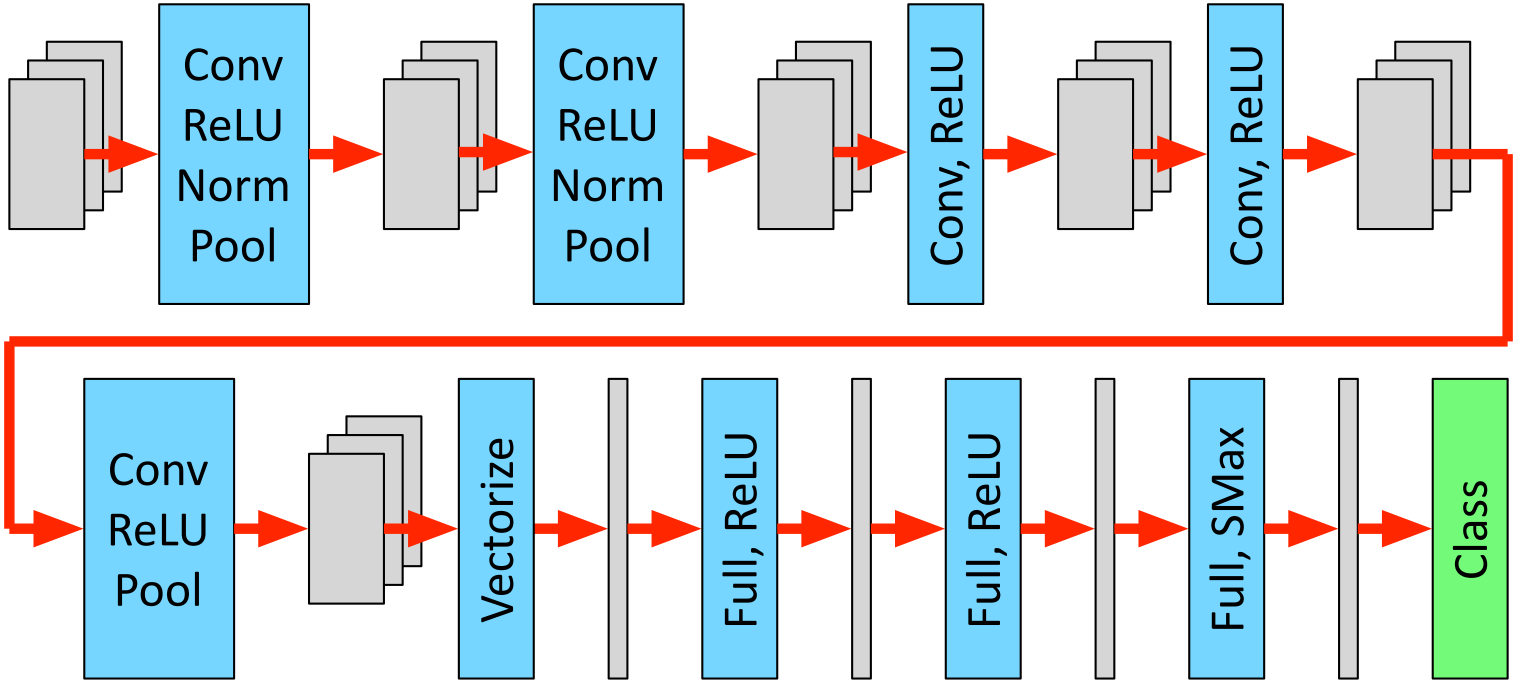
- 5 convolutional and 3 fully connected layers
- ReLU nonlinearity (vs tanh or sigmoid) and 3x3/2 max pooling (vs 2x2/2)
- Grouping (on 2 GPUs) to allow for larger models
- Data augmentation and dropout
- CaffeNet like AlexNet but flip flop LRN and pooling, remove grouping

Visualization

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

Figure from original paper

Serial: AlexNet / CaffeNet



Serial: 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

Serial: 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

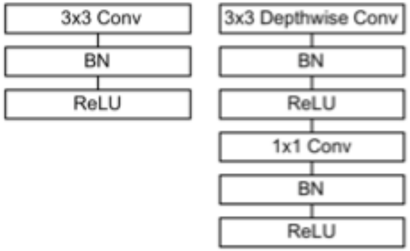
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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

Serial: 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× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× 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	$1 \times 1 \times 1000$

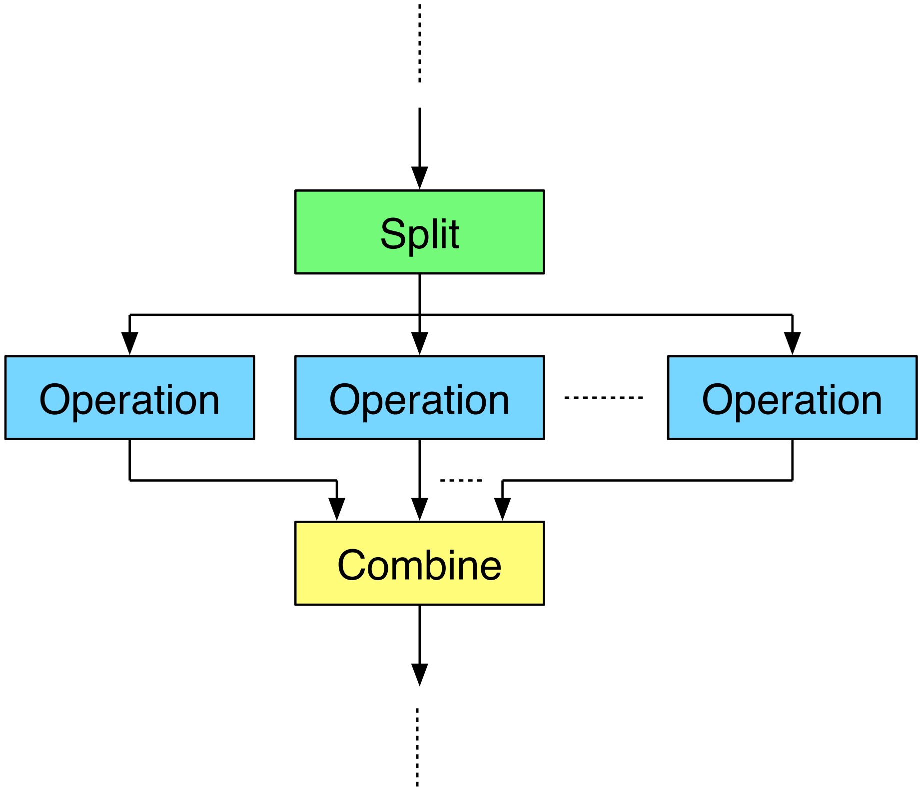
Notes
- CNN style 2D 3x3 convolution replaced with standard 2D 3x3 convolution and CNN style 1x1 convolution (matrix matrix mult)
- 3x3 mixes across space, 1x1 mixes across channel

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

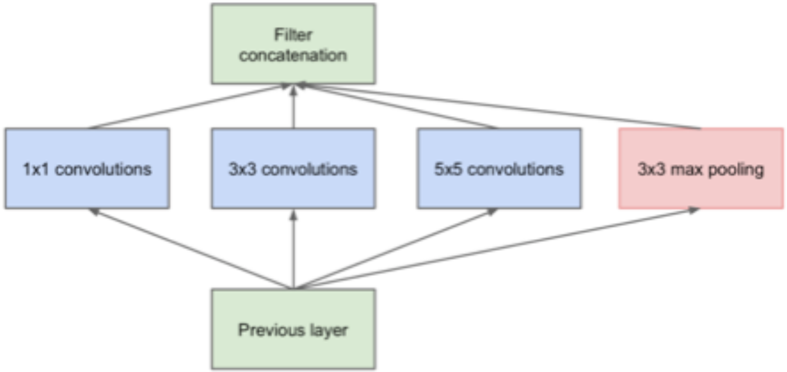
- A simple guide to the versions of the inception network
  - https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202
- Going deeper with convolutions
  - https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf
- Rethinking the inception architecture for computer 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

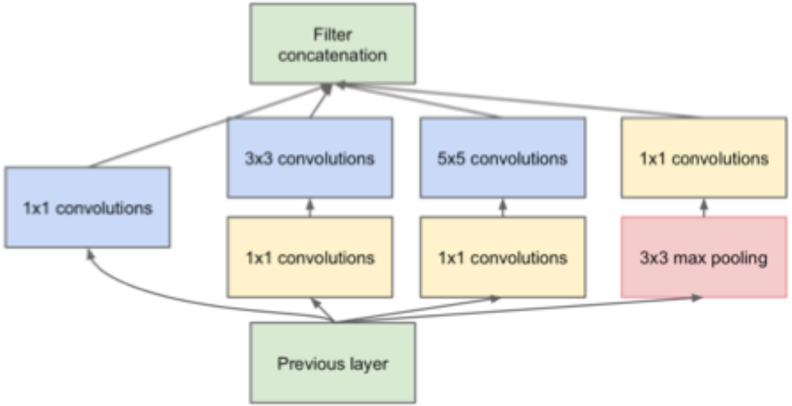
- https://dgschwend.github.io/netscope/#/preset/googlenet
- https://dgschwend.github.io/netscope/#/preset/inceptionv3
- https://dgschwend.github.io/netscope/#/preset/inceptionv4

Parallel: GoogLeNet

Parallel paths (not used by GoogLeNet)



Parallel paths with dimensionality reduction



Figures from original paper

Parallel: GoogLeNet

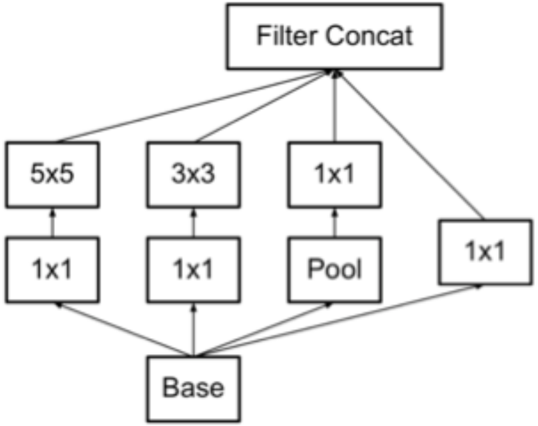
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	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

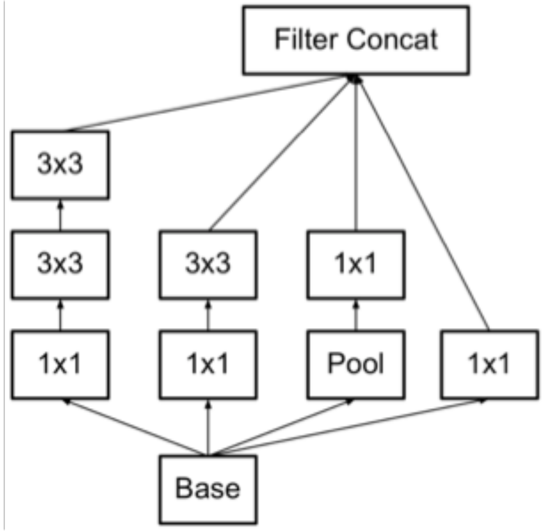
Parallel: Inception V2 And V3

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

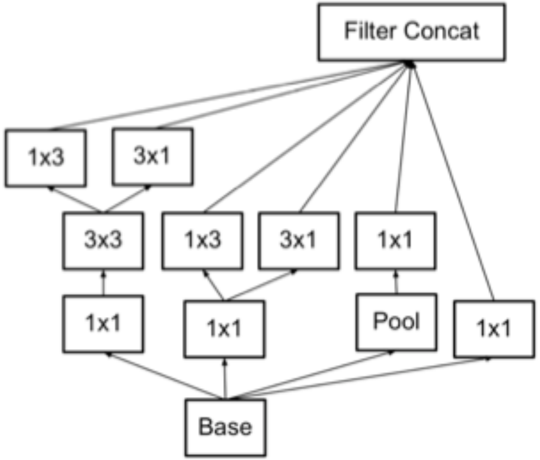
Inception V1



“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

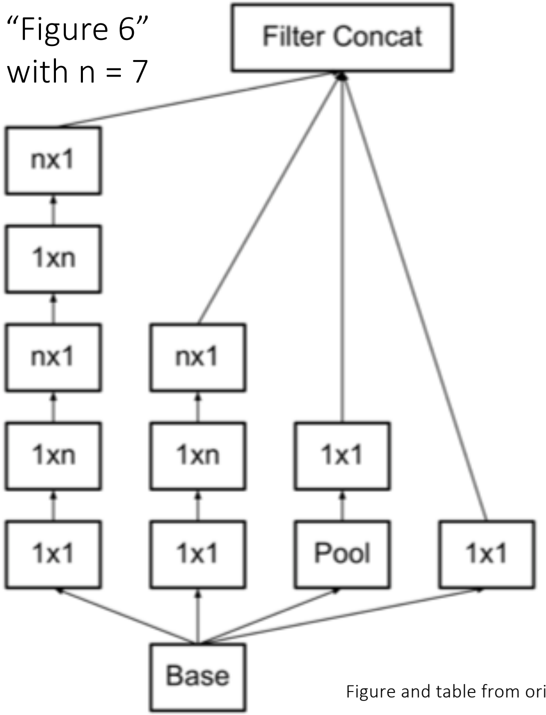
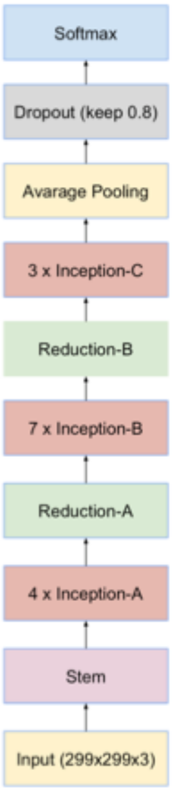
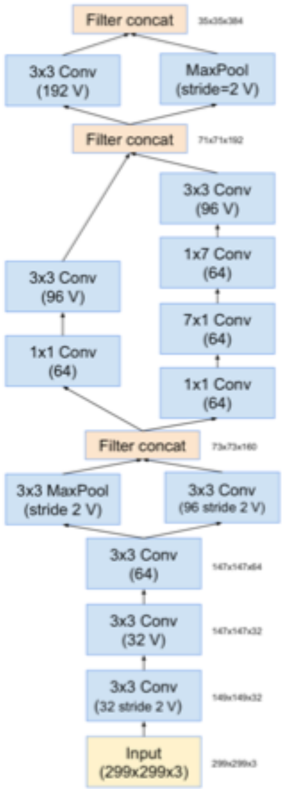


Figure and table from original paper

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

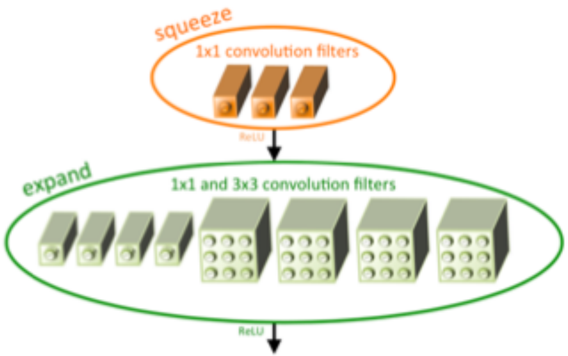




Figures from original paper

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)

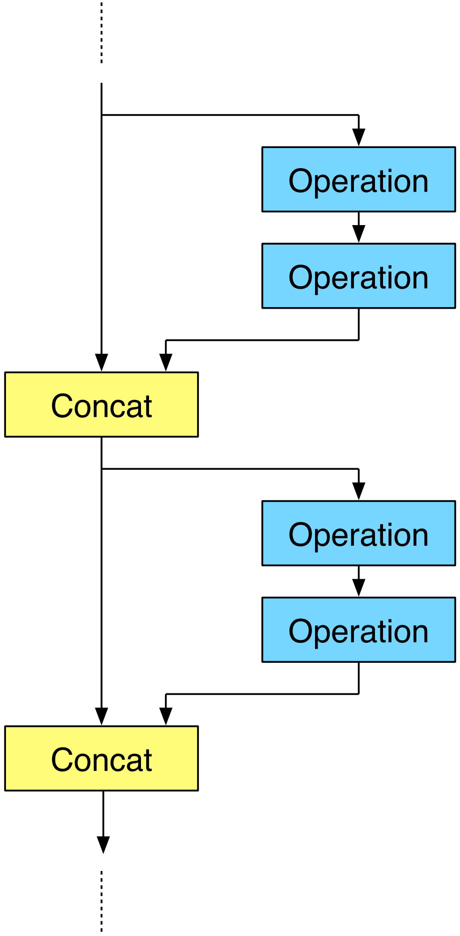
Question
- How small can a network be to solve a problem to a given level of accuracy?

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

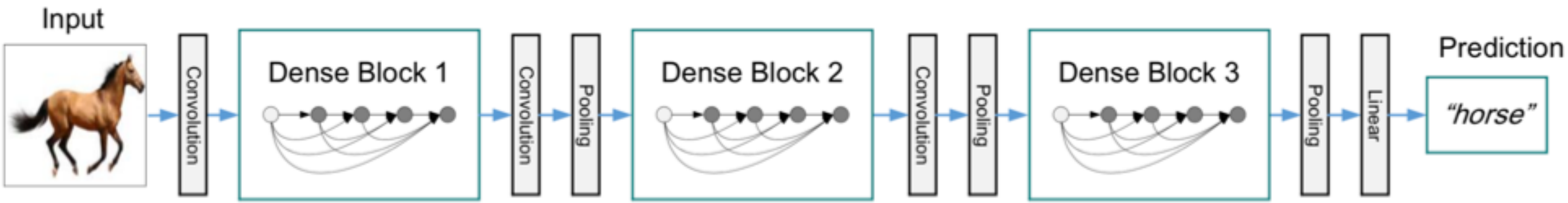
Table from original paper

Dense

- Per building block
  - Input split into 2 paths: direct and transformation
  - 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
  $x_d = h([x_{d-1}, x_{d-2}, \ldots, x_0])$
- 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

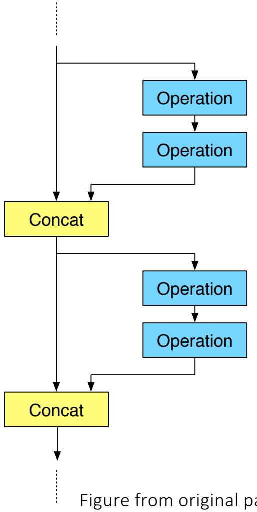


Figure from original paper

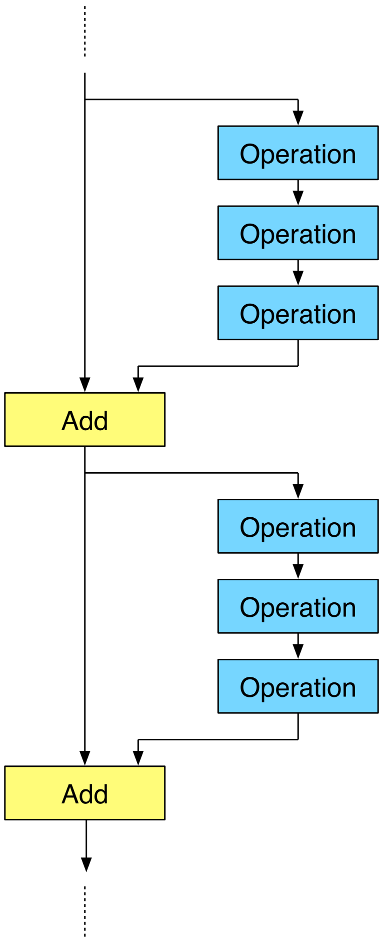
Dense: DenseNet

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	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table from original paper

Residual

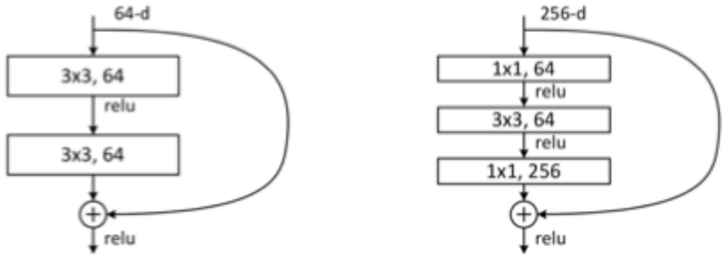
- Per building block
  - Input split into 2 paths
  - Direct and operations (residual)
  - Add 2 paths together to get output
- Output is input + perturbation
  - Clean information flow allows for the training of very deep networks
  - Many options for perturbations (a general concept that can be added 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



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

Figure from original paper

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:

$\partial e/\partial \mathbf{x}_{d} = (\partial \mathbf{x}_{d+1}/\partial \mathbf{x}_{d})\ (\partial e/\partial \mathbf{x}_{d+1})$	Chain rule
$= (\partial \mathbf{f}_{d}/\partial \mathbf{x}_{d})\ (\partial e/\partial \mathbf{x}_{d+1})$	Sub f_d for x_{d+1}
$= (\partial \mathbf{x}_{d}/\partial \mathbf{x}_{d} + \partial \mathbf{h}_{d}/\partial \mathbf{x}_{d})\ (\partial e/\partial \mathbf{x}_{d+1})$	Sub $f_d = x_d + h_d(x_d)$
$= (1 + \partial \mathbf{h}_{d}/\partial \mathbf{x}_{d})\ (\partial e/\partial \mathbf{x}_{d+1})$	Note $\partial \mathbf{x}_{d}/\partial \mathbf{x}_{d} = 1$
$= \partial e/\partial \mathbf{x}_{d+1} + (\partial \mathbf{h}_{d}/\partial \mathbf{x}_{d})\ (\partial e/\partial \mathbf{x}_{d+1})$	Linear operator

Residual: ResNet

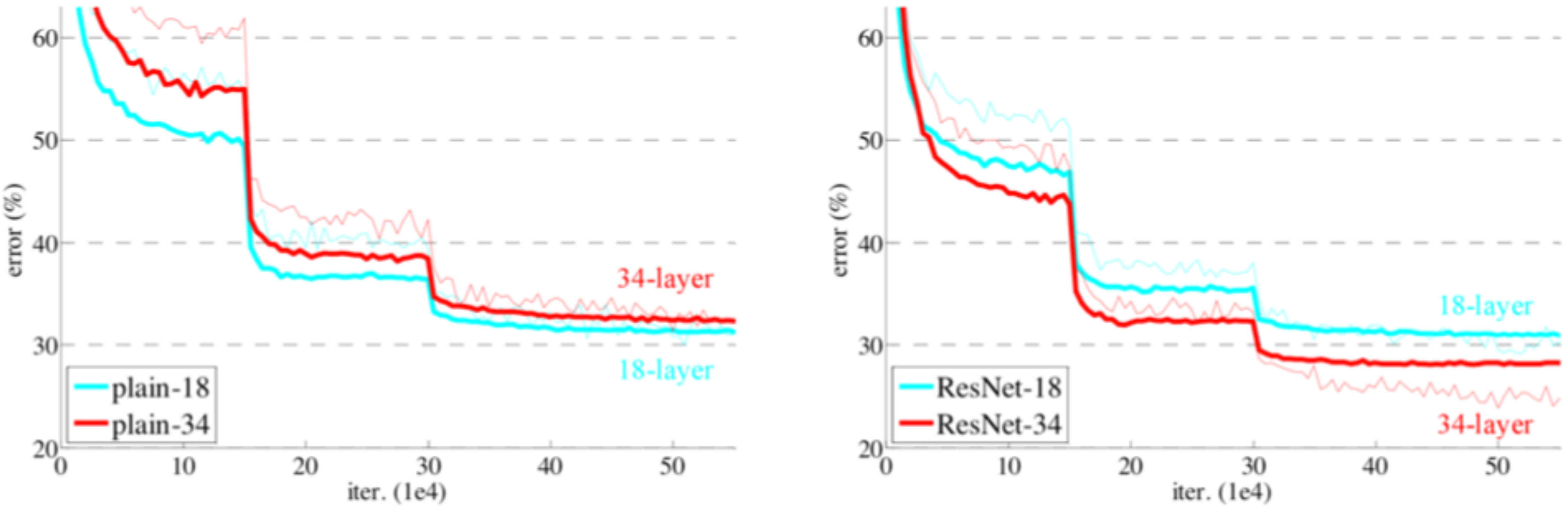


Figure from original paper

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	[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	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

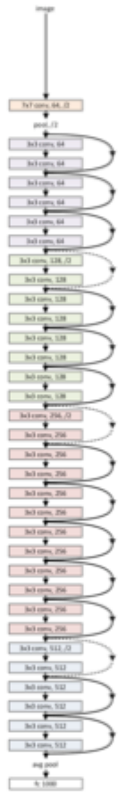
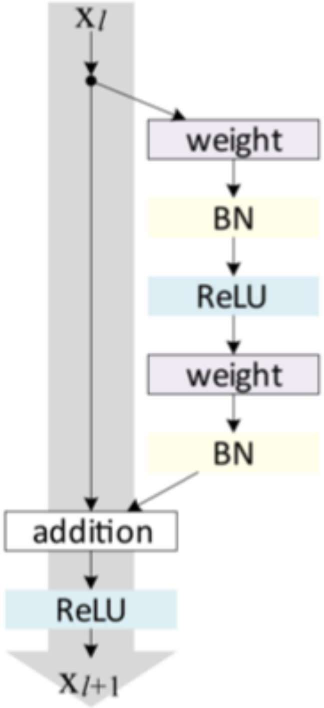


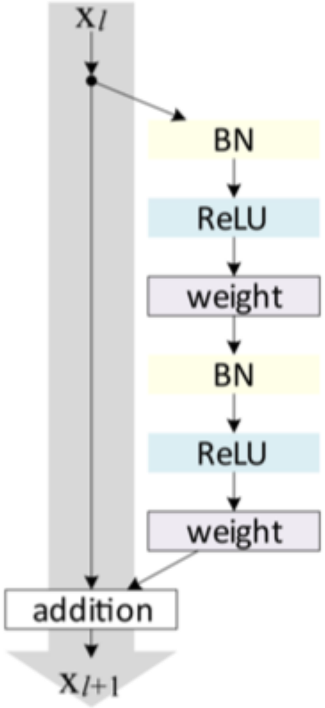
Figure and table from original paper

Residual: ResNet

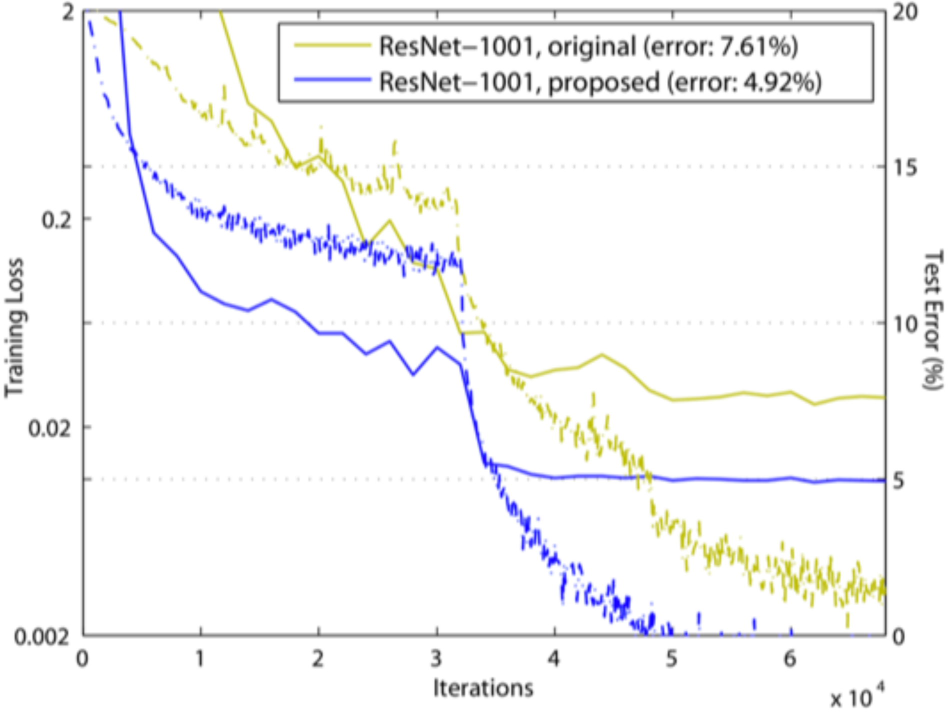
Original



Paper 2



CIFAR 10 Accuracy



Figures from original paper

Residual: ResNeXt / MobileNet V2 / Exception

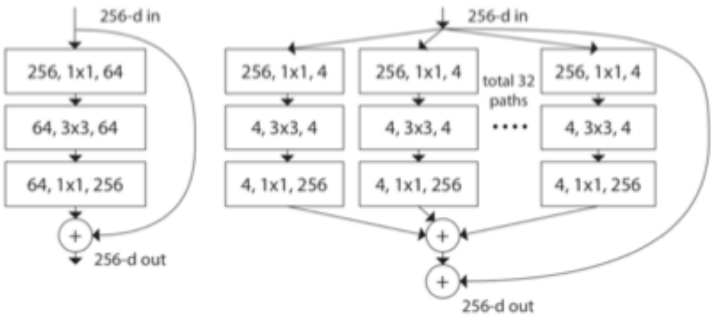
- Combine 2 good ideas
  - Residual connections are good for enabling deep networks
  - Separable filter structures are good for reducing compute

- Strategy
  - Use separable filter structures for the operations in the residual path

- Examples
  - ResNeXt
  - MobileNet V2
  - Exception

Residual: ResNeXt

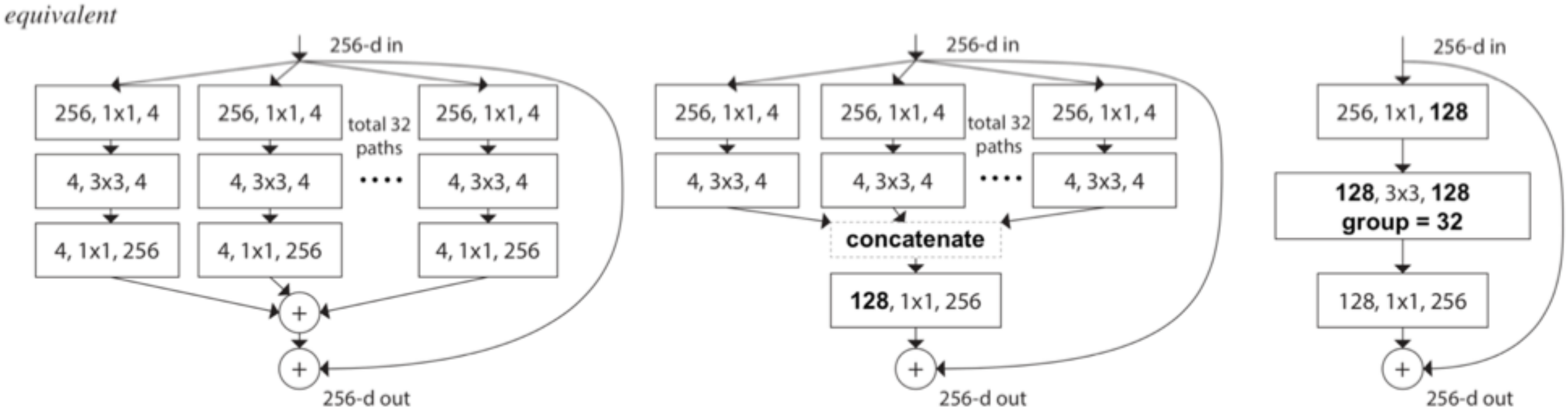
- Design
  - Replace 3x3 convolutions in the residual bottleneck with highly grouped 3x3 convolutions
- Aggregated residual transformations for deep neural networks
  - https://arxiv.org/abs/1611.05431



stage	output	ResNet-50	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
		3×3 max pool, stride 2	3×3 max pool, stride 2
conv2	56×56	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×6
conv5	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**$\times 10^6$	**25.0**$\times 10^6$
FLOPs		**4.1**$\times 10^9$	**4.2**$\times 10^9$

Figure from original paper

Residual: ResNeXt



Residual: MobileNet V2

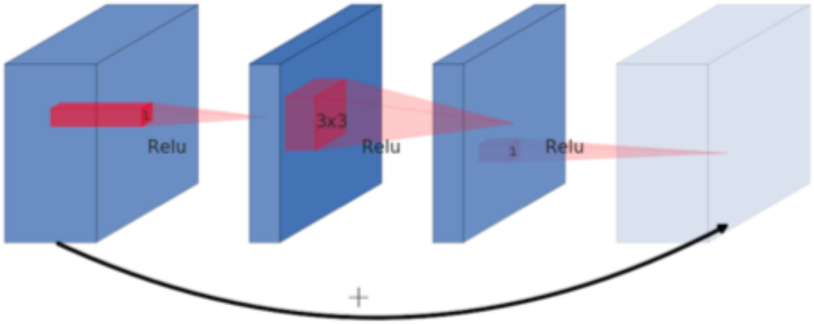
- Design
  - “Inverted” residuals that expand the number of channels
  - Previous designs had input / outputs with a larger number of channels and residual with less channels (a bottleneck)
  - MobileNet V2 flips this and keeps a smaller number of channels and residuals with more channels
  - Output of last convolution before summation has no nonlinearity applied
- MobileNetV2: inverted residuals and linear bottlenecks
  - https://arxiv.org/abs/1801.04381
  - http://machinethink.net/blog/mobilenet-v2/https://github.com/RuiminChen/Caffe-MobileNetV2-ReLU6

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table from original paper

Residual: MobileNet V2

Standard bottleneck



Inverted bottleneck

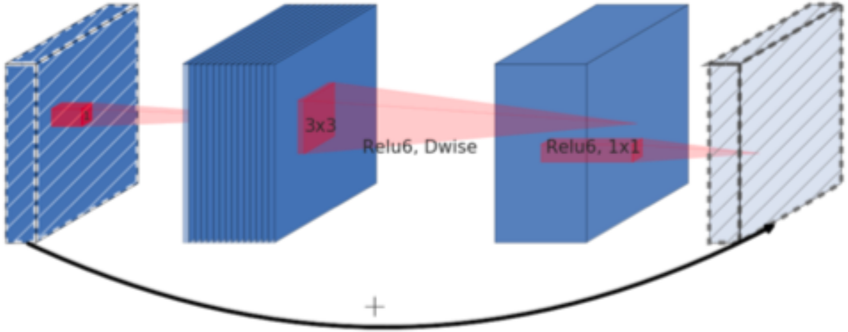


Figure from original paper

Residual: Exception

- Design
  - Inspired by Inception, ResNet and separable convolutions
  - Regular structure
  - Note the final network width (wide)

- Xception: deep learning with depthwise separable convolutions
  - https://arxiv.org/abs/1610.02357

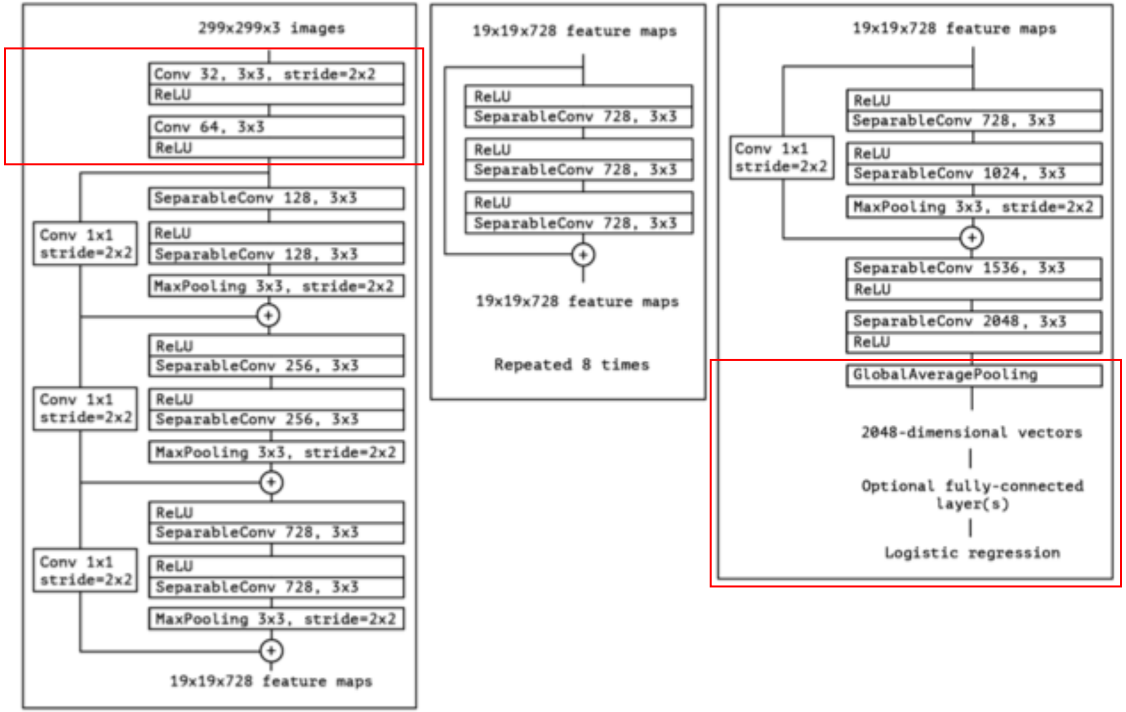


Figure from original paper

Residual: Exception

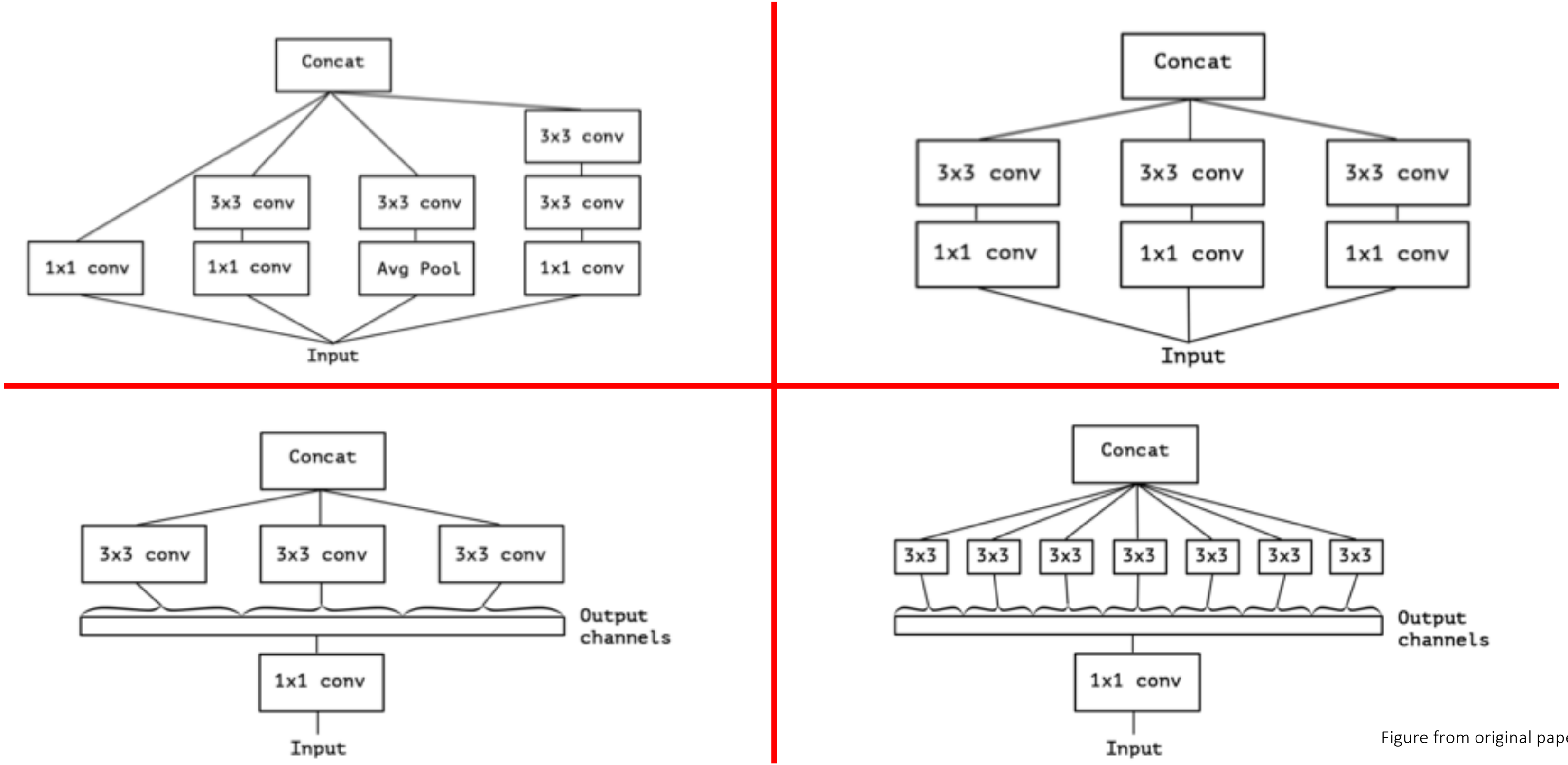


Figure from original paper

Residual: ShuffleNet V1

- Design
  - Use shuffling of channels to allow 1x1 convolutions with grouping to save computations while allowing mixing of features

- ShuffleNet: an extremely efficient convolutional neural network for mobile devices
  - https://arxiv.org/abs/1707.01083

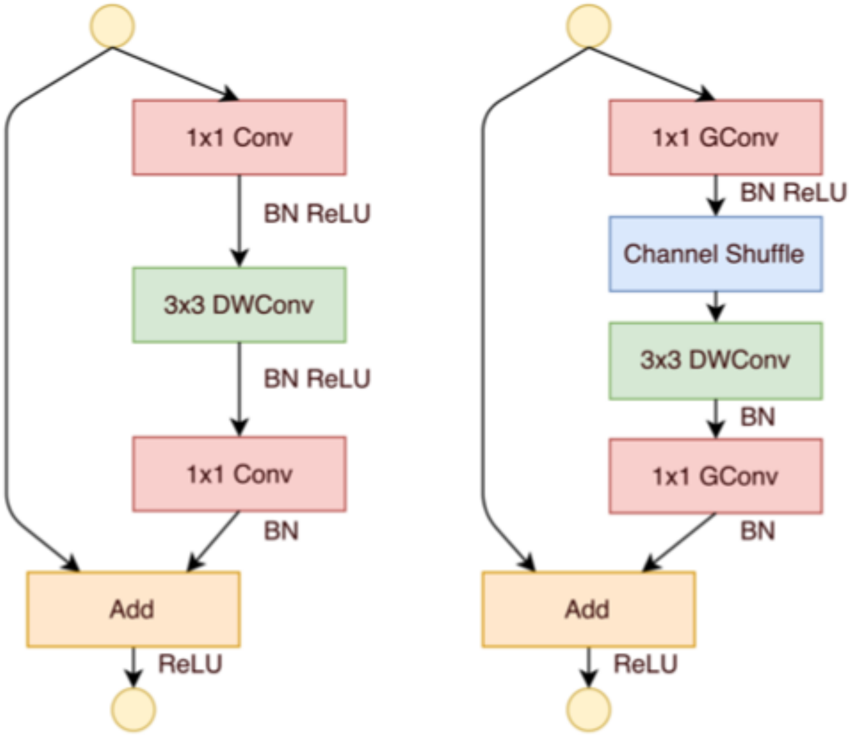


Figure from original paper

Residual: ShuffleNet V1

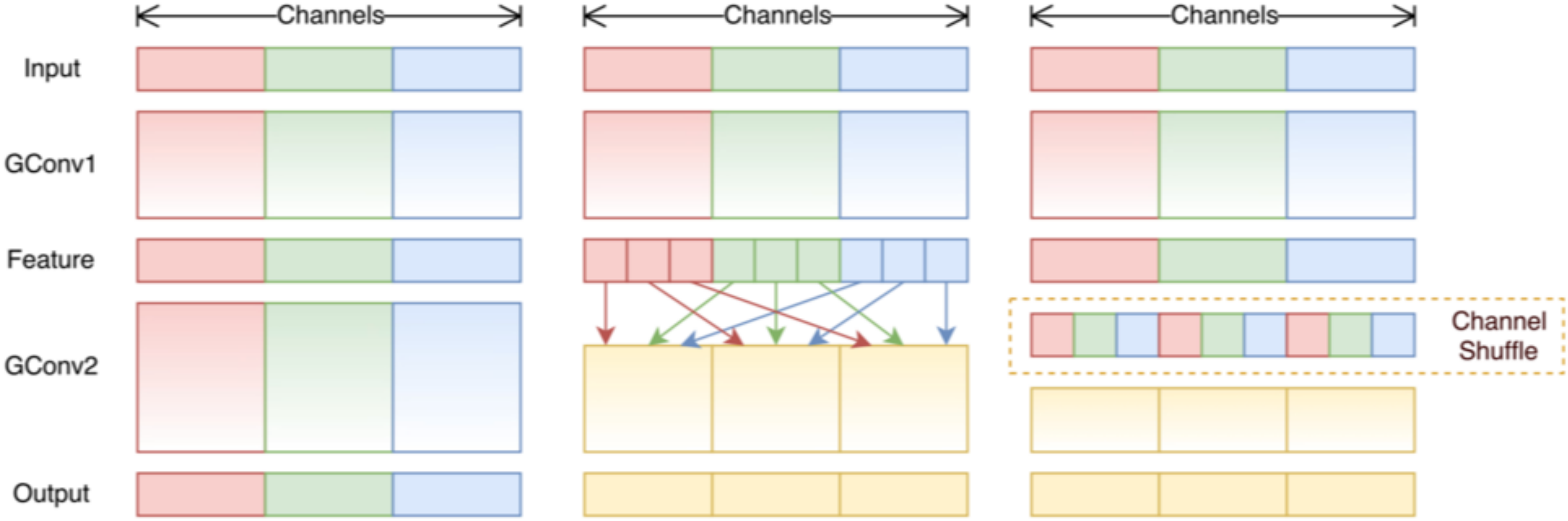


Figure from original paper

Residual: ShuffleNet V1

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

Table from original paper

Residual: ShiftNet

- Design
  - Replaces 3x3 (or other) size convolutions with different shifts for each channel
  - The previous or subsequent 1x1 will then do mixing and some spatial filtering implicitly
  - Note that this is similar to a CNN style 2D convolution with a FxF filter being equivalent to CNN style 2D convolution with F^2 1x1 filters
- Shift: a zero FLOP, zero parameter alternative to spatial convolutions
  - https://arxiv.org/abs/1711.08141

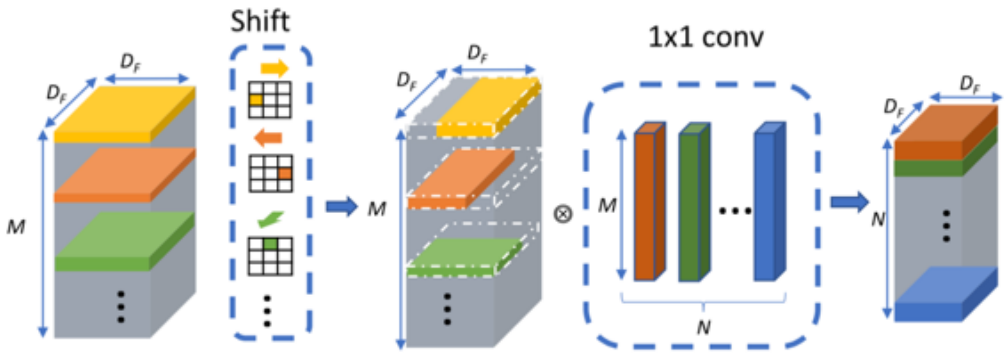
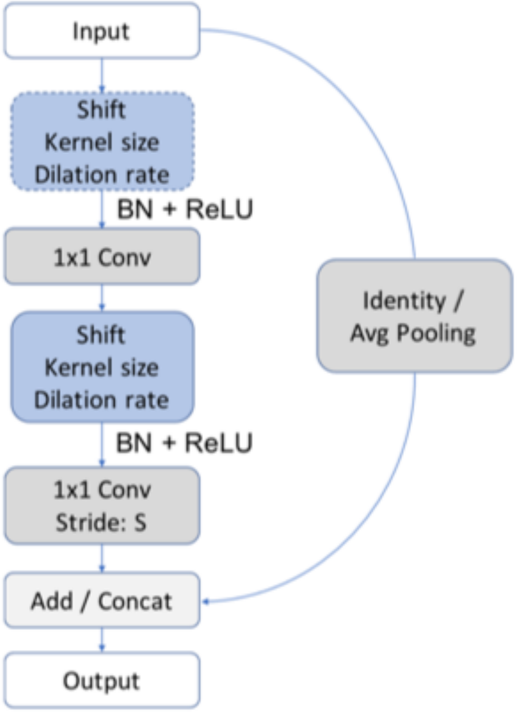


Figure from original paper

Residual: ShiftNet

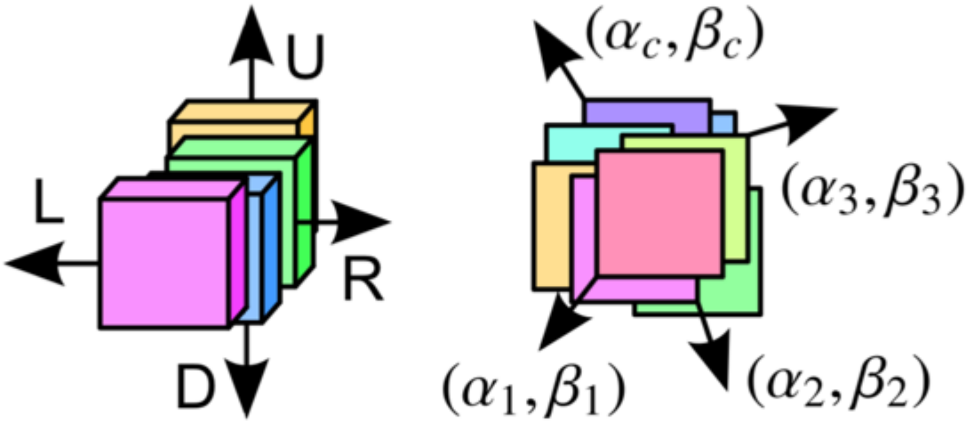


Group	Type/ Stride	Kernel	$\mathcal{E}$	Output Channel	Repeat
-	Conv /s2	7×7	-	32	1
1	CSC / s2	5×5	4	64	1
	CSC / s1	5×5	4		4
2	CSC / s2	5×5	4	128	1
	CSC / s1	5×5	3		5
3	CSC / s2	3×3	3	256	1
	CSC / s1	3×3	2		6
4	CSC / s2	3×3	2	512	1
	CSC / s1	3×3	1		2
-	Avg Pool	7×7	-	512	1
-	FC	-	-	1k	1

Figure and table from original paper

Residual: Adaptive Shift Layer

- Design
  - Uses a learnable "adaptive" shift parameter that apples a different shift to each channel
  - Has some similarities with dilated convolution (which has been shown to work well in automated architecture search)
- Constructing fast network through deconstruction of convolution
  - https://arxiv.org/abs/1806.07370



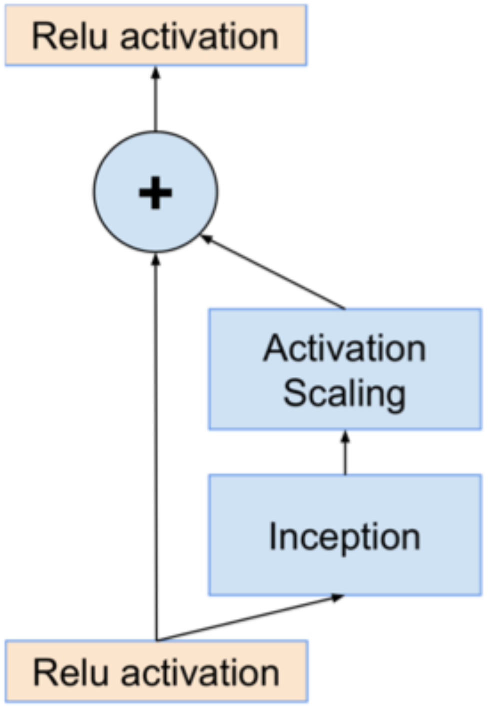
(a) Grouped Shift (b) ASL

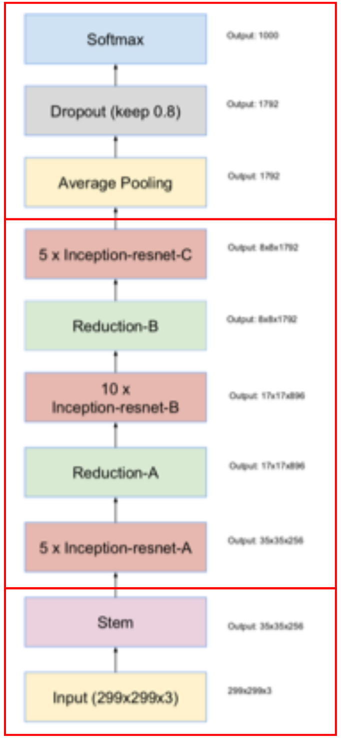
Figure from original paper

Residual: Inception ResNet V1 And V2

- Design
  - Combine Inception V4 with residual connections

- Inception-v4, inception-resnet and the impact of residual connections on learning
  - https://arxiv.org/abs/1602.07261





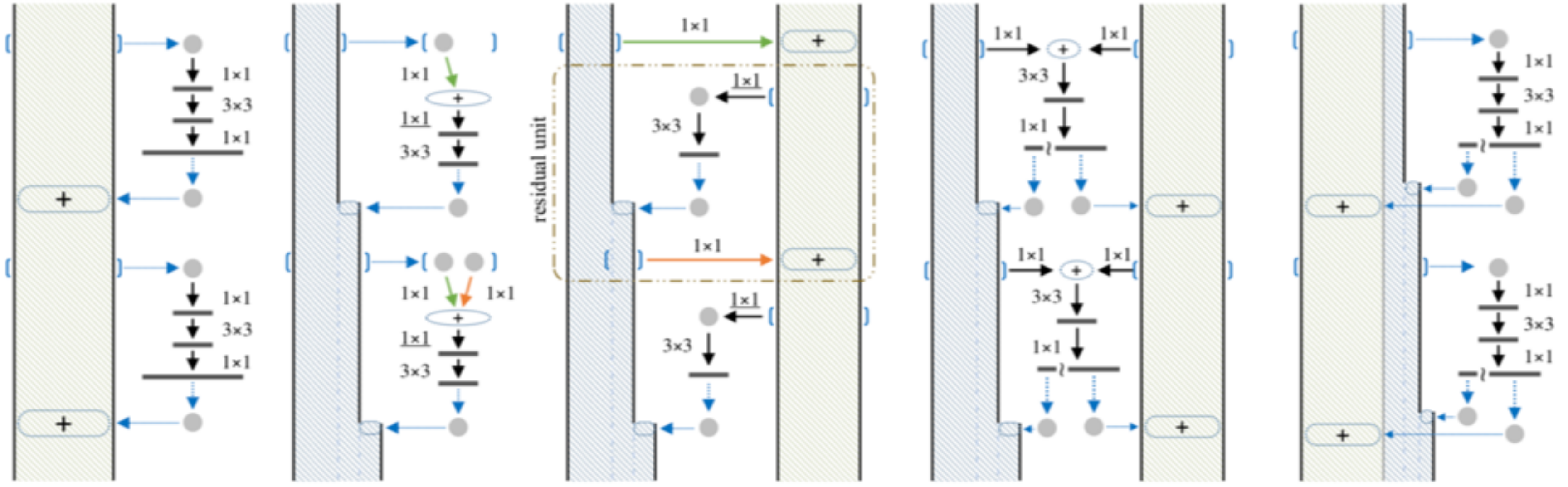
Visualization
- https://dgschwend.github.io/netscope/#/preset/inceptionv4_resnet

Figures from original paper

Residual: ResNet DenseNet (Parallel)

- Design
  - Parallel combination of ResNet and DenseNet blocks with shared computation
- Dual path networks
  - https://arxiv.org/abs/1707.01629

Residual: ResNet DenseNet (Parallel)



(a) Residual Network (b) Densely Connected Network (c) Densely Connected Network (with shared connections) (d) Dual Path Architecture (e) DPN

Figure from original paper

Residual: ResNet DenseNet (Parallel)

stage	output	DenseNet-161 (k=48)	ResNeXt-101 (32×4d)	ResNeXt-101 (64×4d)	DPN-92 (32×3d)	DPN-98 (40×4d)
conv1	112x112	7 × 7, 96, stride 2	7 × 7, 64, stride 2	7 × 7, 64, stride 2	7 × 7, 64, stride 2	7 × 7, 96, stride 2
		3 × 3 max pool, stride 2	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2	3 × 3 max pool, stride 2
conv2	56x56	[1×1, 192; 3×3, 48] × 6	[1×1, 128; 3×3, 128, G=32; 1×1, 256] × 3	[1×1, 256; 3×3, 256, G=64; 1×1, 256] × 3	[1×1, 96; 3×3, 96, G=32; 1×1, 256 (+16)] × 3	[1×1, 160; 3×3, 160, G=40; 1×1, 256 (+16)] × 3
conv3	28×28	[1×1, 192; 3×3, 48] × 12	[1×1, 256; 3×3, 256, G=32; 1×1, 512] × 4	[1×1, 512; 3×3, 512, G=64; 1×1, 512] × 4	[1×1, 192; 3×3, 192, G=32; 1×1, 512 (+32)] × 4	[1×1, 320; 3×3, 320, G=40; 1×1, 512 (+32)] × 6
conv4	14×14	[1×1, 192; 3×3, 48] × 36	[1×1, 512; 3×3, 512, G=32; 1×1, 1024] × 23	[1×1, 1024; 3×3, 1024, G=64; 1×1, 1024] × 23	[1×1, 384; 3×3, 384, G=32; 1×1, 1024 (+24)] × 20	[1×1, 640; 3×3, 640, G=40; 1×1, 1024 (+32)] × 20
conv5	7×7	[1×1, 192; 3×3, 48] × 24	[1×1, 1024; 3×3, 1024, G=32; 1×1, 2048] × 3	[1×1, 2048; 3×3, 2048, G=64; 1×1, 2048] × 3	[1×1, 768; 3×3, 768, G=32; 1×1, 2048 (+128)] × 3	[1×1, 1280; 3×3, 1280, G=40; 1×1, 2048 (+128)] × 3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params		**28.9** $\times 10^6$	**44.3** $\times 10^6$	**83.7** $\times 10^6$	**37.8** $\times 10^6$	**61.7** $\times 10^6$
FLOPs		**7.7** $\times 10^9$	**8.0** $\times 10^9$	**15.5** $\times 10^9$	**6.5** $\times 10^9$	**11.7** $\times 10^9$

Table from original paper

Residual: Squeeze And Excitation Networks

- Notes
  - Aggregate information over the full feature map to determine a re weighting of features based on the full image (feature re calibration)
  - A quasi way to increase the receptive field size
  - Can be added to existing networks as a parallel path
  - Early on excites features in a class agnostic manner
  - Later on responds differently to different classes
  - Maybe less important later on since features are more class specific

Observation

- Top performing architectures (found from optimization) tend to use dilated convolution which increases receptive field size
- See some similarities here of the importance of increasing receptive field size

- Squeeze-and-excitation networks
  - https://arxiv.org/abs/1709.01507

Residual: Squeeze And Excitation Networks

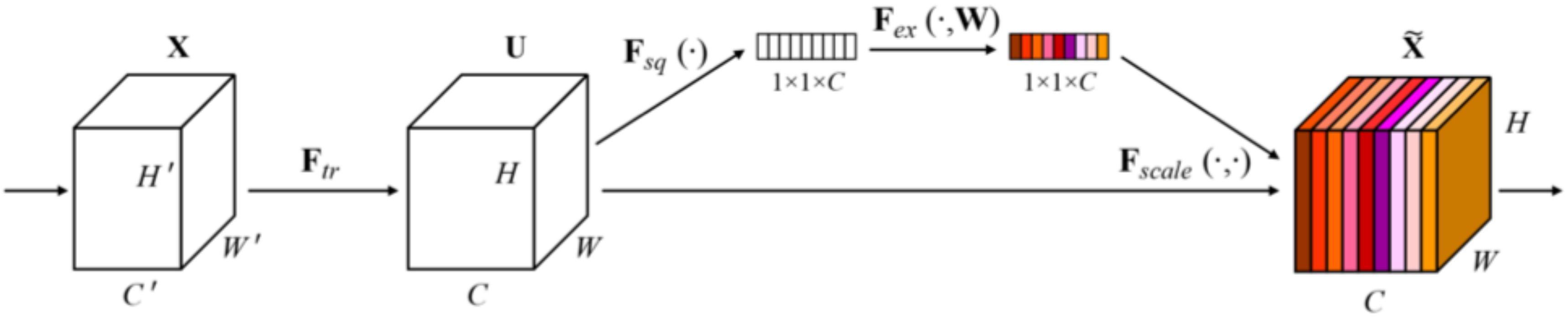
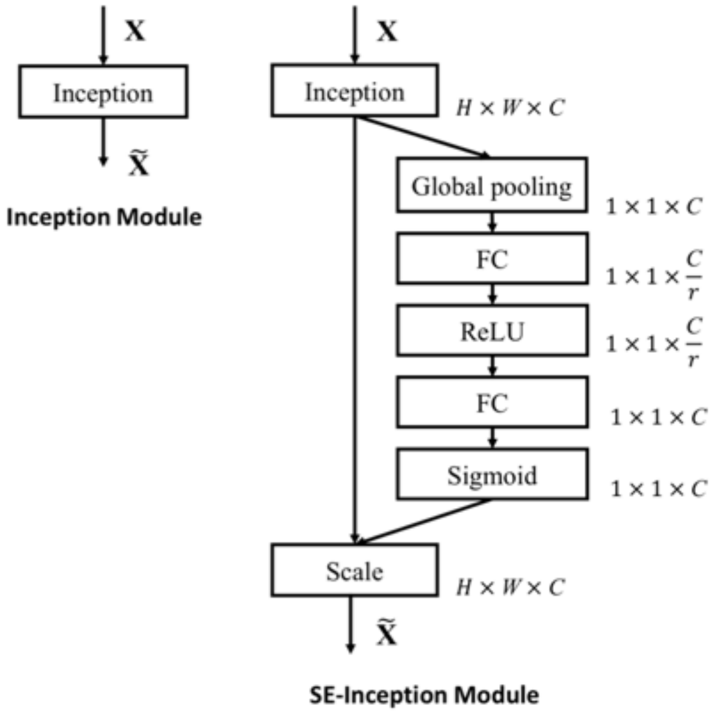
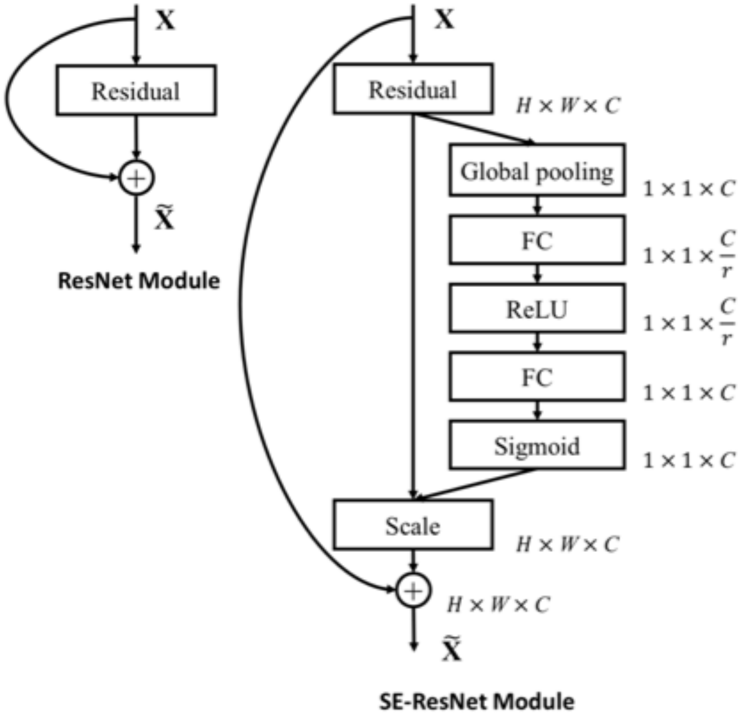


Figure from original paper

Residual: Squeeze And Excitation Networks





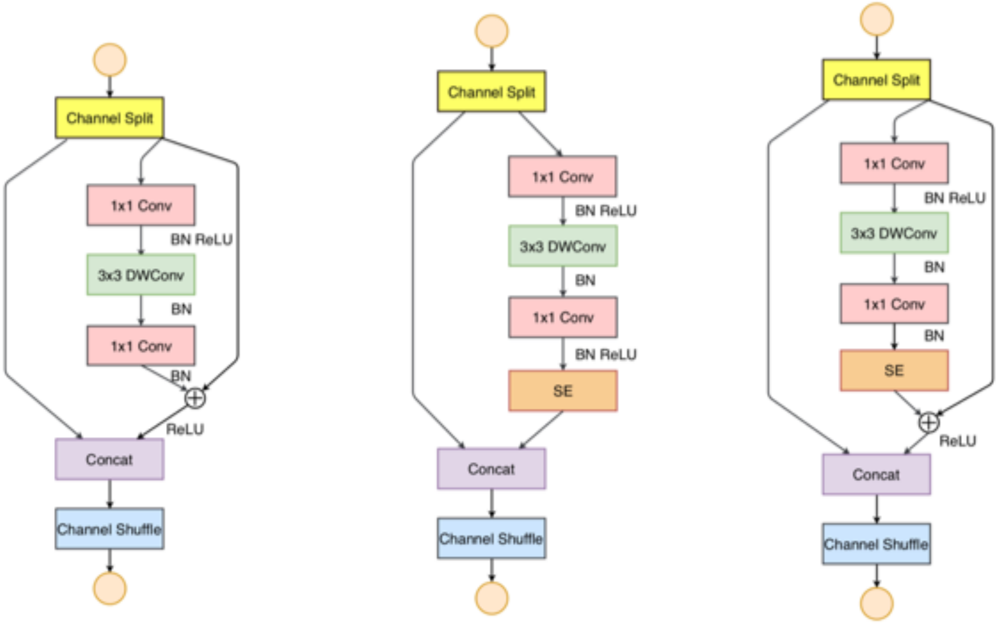
Residual: Squeeze And Excitation Networks

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 ($32 \times 4d$)
112×112	conv, 7×7, 64, stride 2		
56×56	max pool, 3×3, stride 2		
	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \\ \text{conv}, 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \\ \text{conv}, 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 & \\ \text{conv}, 3 \times 3, 128 & C = 32 \\ \text{conv}, 1 \times 1, 256 & \\ fc, [16, 256] & \end{bmatrix} \times 3$
28×28	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \\ \text{conv}, 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \\ \text{conv}, 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 & \\ \text{conv}, 3 \times 3, 256 & C = 32 \\ \text{conv}, 1 \times 1, 512 & \\ fc, [32, 512] & \end{bmatrix} \times 4$
14×14	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \\ \text{conv}, 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \\ \text{conv}, 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 & \\ \text{conv}, 3 \times 3, 512 & C = 32 \\ \text{conv}, 1 \times 1, 1024 & \\ fc, [64, 1024] & \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \\ \text{conv}, 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \\ \text{conv}, 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 1024 & \\ \text{conv}, 3 \times 3, 1024 & C = 32 \\ \text{conv}, 1 \times 1, 2048 & \\ fc, [128, 2048] & \end{bmatrix} \times 3$
1×1	global average pool, 1000-d fc, softmax		

Figure from original paper

Residual: ShuffleNet V2

- Improvements to ShuffleNet V1 based on empirical performance studies on CPU and GPU architectures
  - Setting Ni = No improves memory performance
  - Optimizing the grouping parameter
  - Minimizing the number of parallel paths
  - Elementwise operations cannot be ignored
- ShuffleNet V2: practical guidelines for efficient CNN architecture design
  - https://arxiv.org/abs/1807.11164
- Note: we will look at performance in much more detail in the implementation lectures



ShuffleNet V2 building blocks

Figure from original paper

Residual: ShuffleNet V2

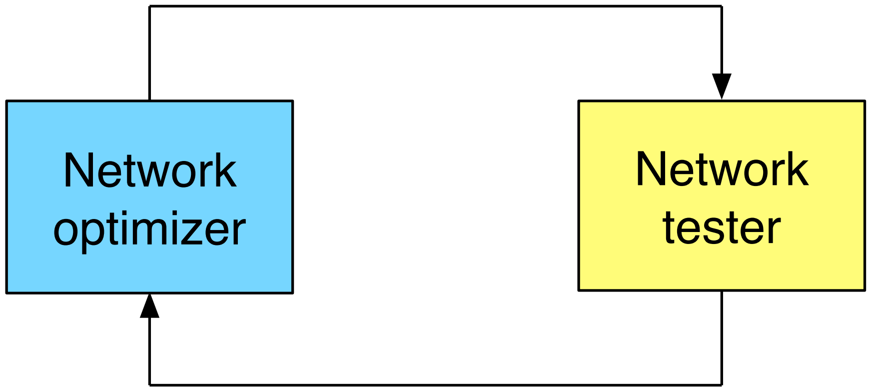
layer	output size	ShuffleNet v1-50 (group=3)	ShuffleNet v2-50	Resnet-50	SE-ShuffleNet v2-164
conv1_x	112×112	3×3, 64, stride 2	3×3, 64, stride 2	7×7, 64, stride 2	3×3, 64, stride 2 3×3, 64 3×3, 128
conv2_x	56×56	3×3 max pool, stride 2			
		[1×1, 360 3×3, 360 1×1, 360] ×3	[1×1, 244 3×3, 244 1×1, 244] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 340 3×3, 340 1×1, 340] ×10
conv3_x	28×28	[1×1, 720 3×3, 720 1×1, 720] ×4	[1×1, 488 3×3, 488 1×1, 488] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 680 3×3, 680 1×1, 680] ×10
conv4_x	14×14	[1×1, 1440 3×3, 1440 1×1, 1440] ×6	[1×1, 976 3×3, 976 1×1, 976] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 1360 3×3, 1360 1×1, 1360] ×23
conv5_x	7×7	[1×1, 2880 3×3, 2880 1×1, 2880] ×3	[1×1, 1952 3×3, 1952 1×1, 1952] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 2720 3×3, 2720 1×1, 2720] ×10
conv6	7×7	-	1×1, 2048	-	1×1, 2048
	1×1	average pool, 1000-d fc, softmax			
FLOPs		2.3G	2.3G	3.8G	12.7G

Figure from original paper

Optimized Architecture Search

- Using ML to find new architectures
  - Interesting from an exploration perspective
  - Likely a mix of human curated building blocks and modifications with machine determined combinations and applications

- Note that it needs to avoid 2 levels of overfitting
  - 1 is the standard for a given network trained on part of a data set and evaluated on another part of the data set (standard danger)
  - 2 is for the network architecture itself (new danger)



Optimized Architecture Search

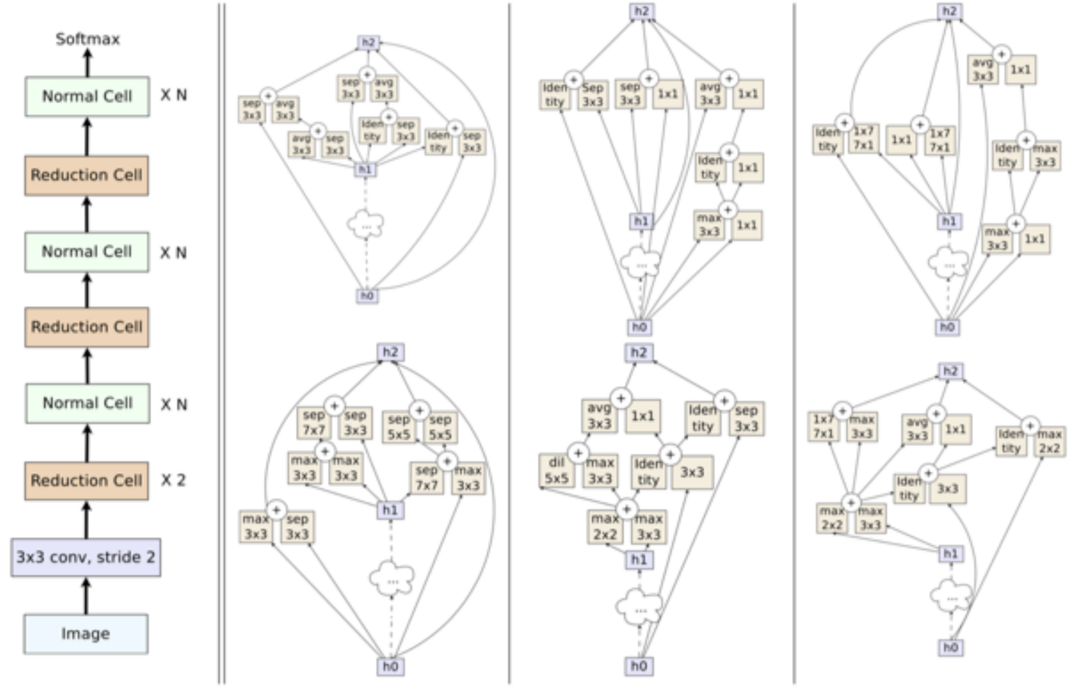
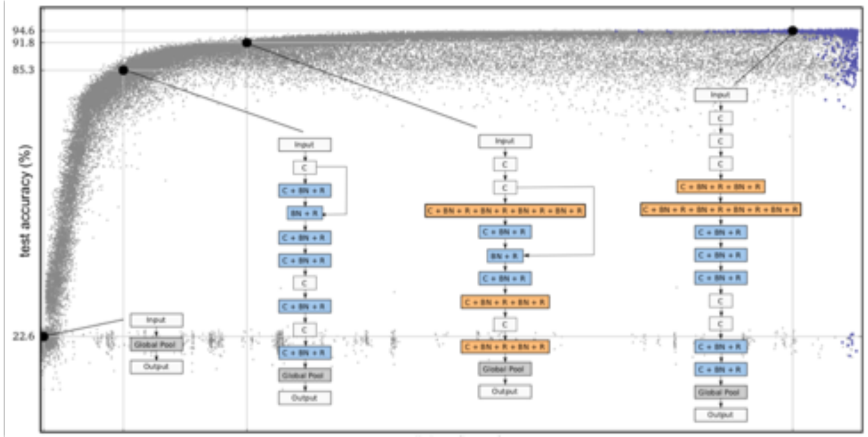


Figure from original paper

Optimized Architecture Search

- Round tripping: while originally devised as a method for designing networks that (sort of) doesn't require humans, neural architecture search actually can be used to give humans ideas on what matters most in network design
  - Makes sense as it can be viewed as a really really large set of trials with varying parameters
  - Ex: a more extensive use of Atrous convolutions even in classification problems



- For balance on the topic it's worthwhile to read
  - An opinionated introduction to AutoML and neural architecture search
    - https://www.fast.ai/2018/07/16/auto-ml2/
  - Google's AutoML: cutting through the hype
    - https://www.fast.ai/2018/07/23/auto-ml-3/

Figure from https://arxiv.org/abs/1703.01041

Optimized Architecture Search

- For an up to date list of references see
  - https://www.ml4aad.org/automl/literature-on-neural-architecture-search/

- Neural architecture search with reinforcement learning
  - https://arxiv.org/abs/1611.01578
- Large-scale evolution of image classifiers
  - https://arxiv.org/abs/1703.01041
- Learning transferable architectures for scalable image recognition
  - https://arxiv.org/abs/1707.07012
- Searching for activation functions
  - https://arxiv.org/abs/1710.05941
- Progressive neural architecture search
  - https://arxiv.org/abs/1712.00559
  - https://cs.jhu.edu/~cxliu/slides/pnas-talk-eccv.pdf
- Regularized evolution for image classifier architecture search
  - https://arxiv.org/abs/1802.01548

Optimized Architecture Search

- Efficient neural architecture search via parameter sharing
  - https://arxiv.org/abs/1802.03268
- MnasNet: platform-aware neural architecture search for mobile
  - https://arxiv.org/abs/1807.11626
- DARTS: differentiable architecture search
  - https://arxiv.org/abs/1806.09055
- Faster discovery of neural architectures by searching for paths in a large model
  - https://openreview.net/pdf?id=ByQZjx-0-
- PPP-net: platform-aware progressive search for Pareto-optimal neural architectures
  - https://openreview.net/pdf?id=B1NT3TAIM
- AdaNet: adaptive structural learning of artificial neural networks
  - http://proceedings.mlr.press/v70/cortes17a/cortes17a.pdf
  - https://github.com/tensorflow/adanet
- Neural architect: a multi-objective neural architecture search with performance prediction
  - http://www.sysml.cc/doc/94.pdf

Accuracy On ImageNet Classification

- Caveats
  - Just because a network is good at ImageNet classification doesn't mean that it will be good at a different task
  - Just because a network is better than another network at ImageNet classification doesn't mean that it will be better at a different task
  - So think of this as a starting point for thinking about network accuracy but not the last word
  - Really important to think about the particular problem you're interested in and what is the optimal network for extracting information

Network	Top 1	Top 5	Param Mem	Feature Mem	MACs
AlexNet	56.90	80.10			
CaffeNet	57.10	80.10			
VGG 16	71.59	90.38			
VGG 19	72.70	91.00			
ResNet 50	76.15	92.86			
ResNet 101	77.37	93.56			
ResNet 152	78.31	94.06			
ResNeXt 50	77.80	93.38			
ResNeXt 101	78.80	94.05			
DenseNet 121	74.91	92.19			
GoogLeNet	68.70	88.90			
Inception V4	80.00	95.00			
MobileNet V1	70.90	89.90			
MobileNet V2	71.80	91.00			

Personal Notes For Vision Optimized CNNs

- Think of these as a reasonable starting point and some priorities to keep in mind when doing network design, not absolutes (different problems can take very different solutions)

- For data plan on
  - Pre training on ImageNet using 3 x 224 x 224 images
  - Transfer learning, head replacement
  - More training on target data set which we'll take to be 3 x 512 x 1024 images

- Fix the tail (for all) and head for the initial training
  - Tail: data normalization, 64 x 3 x 7 x 7 / 2 conv + BN + ReLU, 3 x 3 / 2 max pool
  - Head (1st): global avg pool, 1000 x N full + bias + soft max
design for N > 1000 such that ≥ features per class

Personal Notes For Vision Optimized CNNs

- Determine the number of down sampling stages
  - Target a feature size of 6 – 8 x 6 – 8 or larger before global avg pooling
  - 5 down sampling stages take ImageNet inputs to N x 7 x 7
  - 5 down sampling stages take 3 x 512 x 1024 inputs to N x 16 x 32
  - Expand channel by 2 then down sample so no excess information bottleneck

- Example structure (with 5 levels of down sampling)
  - Tail level 0 to 2
  - Building block level 2 — Repeat a few times (high compute)
  - Down sampling level 2 to 3
  - Building block level 3 — Repeat a few times
  - Down sampling level 3 to 4
  - Building block level 4 — Repeat many times (good balance of compute and memory, good receptive field)
  - Down sampling level 4 to 5
  - Building block level 5 — Repeat a few times (high memory)

Personal Notes For Vision Optimized CNNs

- Building block
  - ResNeXt isn't a bad place to start, optimize grouping in bottleneck for target architecture
  - Use padding with convolutions and pooling layers to keep input and output feature sizes integer multiples
    - This will help in later designs with more complex heads
    - Note that this also applies to the tail design
  - Use batch norm basically after all convolutions (then merge into conv as much as possible after training)
  - At concat and add nodes make sure inputs have same ranges (both all positive or both positive and negative)
  - Track receptive field size and compare to objects of interest in original image
  - Track feature map memory, filter weight memory, operations (matrix, vector and scalar) and optimize for target architecture
  - If you can afford the computation and the subsequent network tasks requires localization consider atrous convolution

CNNs: Visualization

What Feature Maps Look Like

- In general ...
  - Smooth regions and edges at the beginning (image space)
  - Spikey at the end (feature space)

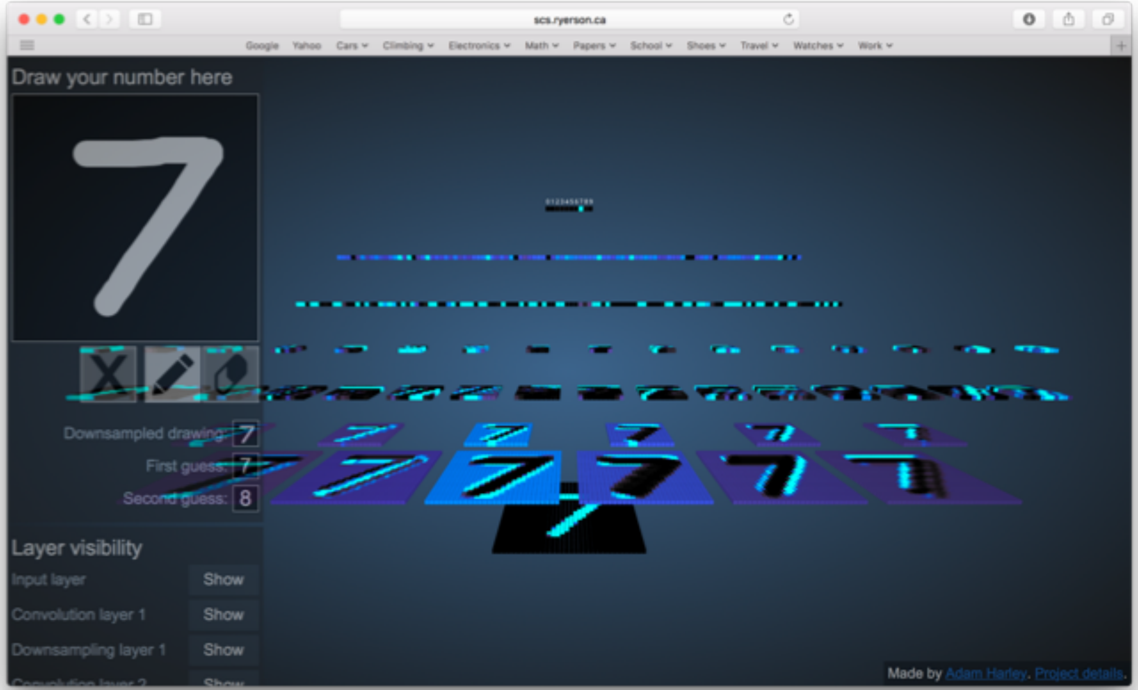
- In general ...
  - Ok to clip some in the beginning (many important parts)
  - Don’t want to clip at the end (want to find the max)

Human Understanding

- Lots of work on increasing human understanding of how the network makes decisions
  - Visualization deep inside convolutional networks: visualising image classification models and saliency maps
    - https://arxiv.org/abs/1312.6034
  - Understanding intermediate layers using linear classifier probes
    - https://arxiv.org/abs/1610.01644
  - Feature visualization
    - https://distill.pub/2017/feature-visualization/
  - The building blocks of interpretability
    - https://distill.pub/2018/building-blocks/
  - Visualization for machine learning
    - https://media.neurips.cc/Conferences/NIPS2018/Slides/Visualization_for_ML.pdf
  - Under the hood
    - https://fleuret.org/ee559/ee559-slides-8-1-looking-at-parameters.pdf
    - https://fleuret.org/ee559/ee559-slides-8-2-looking-at-activations.pdf
    - https://fleuret.org/ee559/ee559-slides-8-3-visualizing-in-input.pdf
    - https://fleuret.org/ee559/ee559-slides-8-4-optimizing-inputs.pdf

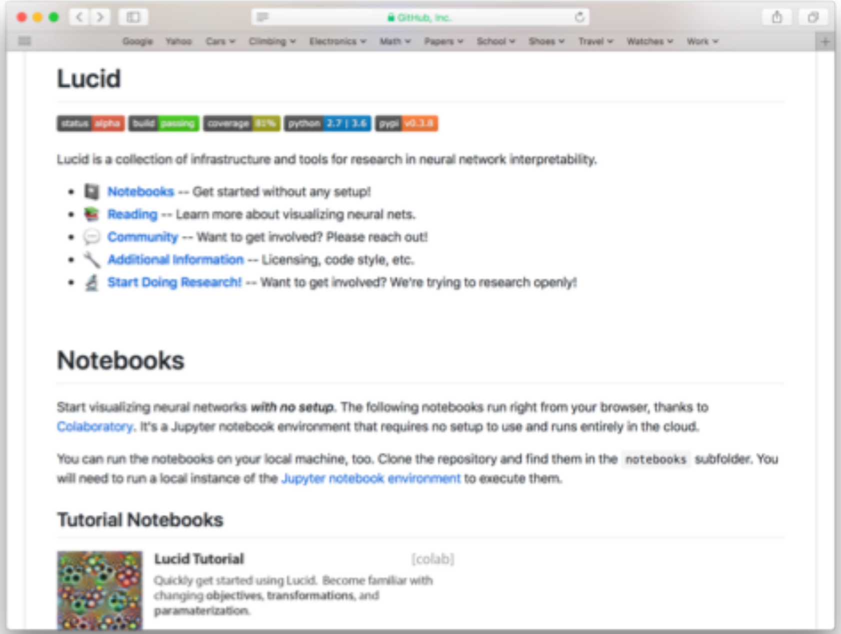
Interactive Demo

- For a nice interactive MNIST network visualization demo see
  - http://scs.ryerson.ca/~aharley/vis/conv/



Lucid

- Tools for neural network interpretability
  - https://github.com/tensorflow/lucid
- Built in model zoo of common CNN architectures
- Built in notebooks that run in Colab for the techniques described in Distill articles
  - Feature visualization
    - https://distill.pub/2017/feature-visualization/
  - The building blocks of interpretability
    - https://distill.pub/2018/building-blocks/
  - Differentiable image parameterizations
    - https://distill.pub/2018/differentiable-parameterizations/



RNNs

Strategy

- RNNs exploit sequential structure in data
  - Keep this in mind as you're thinking about applying them to a problem
- Overview
  - Layers
    - Basic, GRU and LSTM
  - Networks
    - Single layer, stacked, stacked with residual, stacked with pyramidal
    - Uni and bi directional
    - Sparsely gated mixtures of experts
  - Mappings
    - 1 input to many outputs, many inputs to 1 output and many inputs to many outputs

A likely bit of confusion that these slides will not improve (sorry)

- RNN is sometimes used to refer to a specific recurrent layer structure
- RNN is sometimes used to refer to a recurrent layer in general (RNN, GRU, LSTM, ...)
- In general, not something to stress about

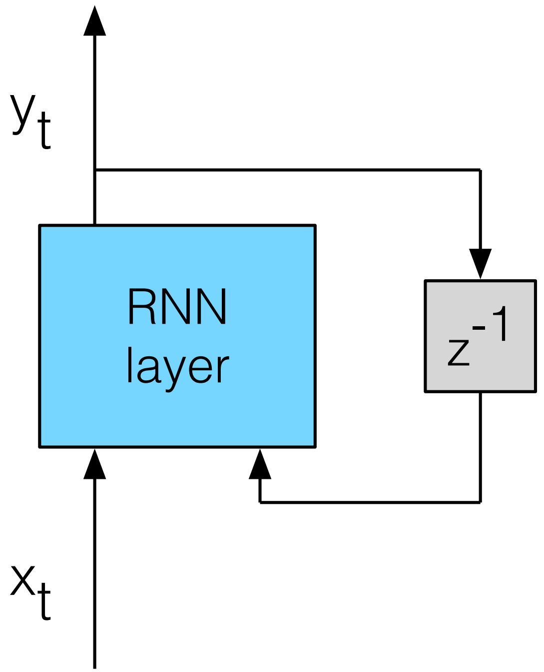
Side note: feed forward neural networks are universal approximators but not Turing complete; RNNs, via their memory state, are also Turing complete

On the computational power of neural nets

https://binds.cs.umass.edu/papers/1992_Siegelmann_COLT.pdf

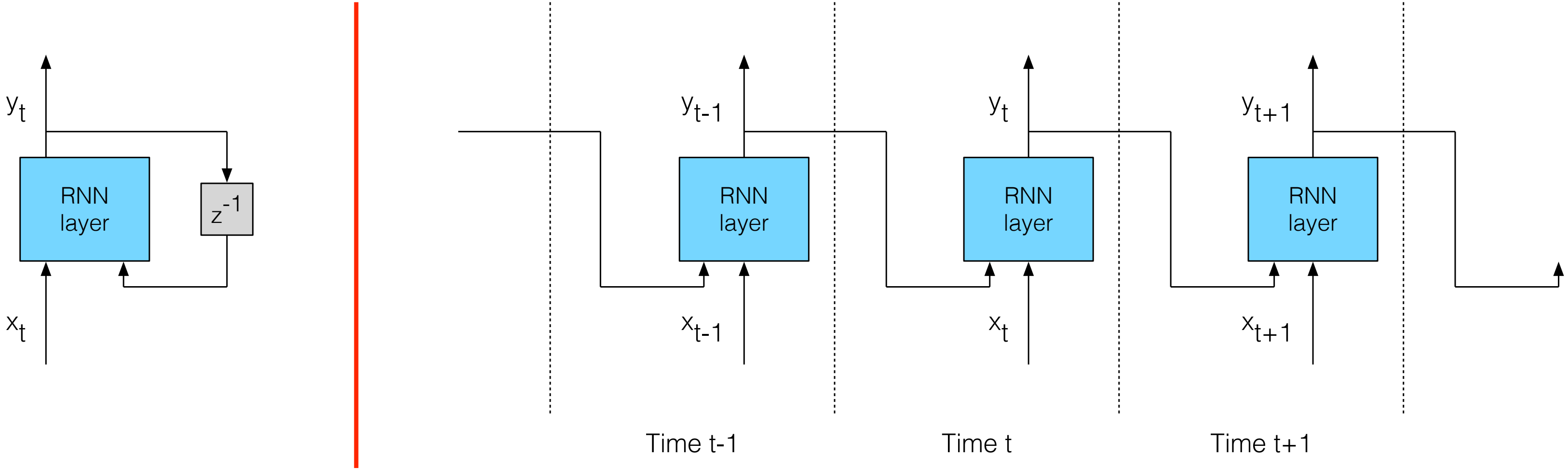
Layers: RNN

- RNN layer
  - $\mathbf{y}_t = f(\mathbf{H}\,\mathbf{x}_t + \mathbf{G}\,\mathbf{y}_{t-1} + \mathbf{v})$
  - t is the current time step, t – 1 is the previous time step
  - $\mathbf{H}\,\mathbf{x}_t$ is multiplication of a M x K matrix **H** with a K x 1 input vector $\mathbf{x}_t$
  - $\mathbf{G}\,\mathbf{y}_{t-1}$ is multiplication of a M x M matrix **G** with a M x 1 previous output vector $\mathbf{y}_{t-1}$
  - **v** is a M x 1 bias vector
  - f is a pointwise nonlinearity as in the case of a densely connected layer
  - $\mathbf{y}_t$ is a M x 1 current output vector



Layers: RNN

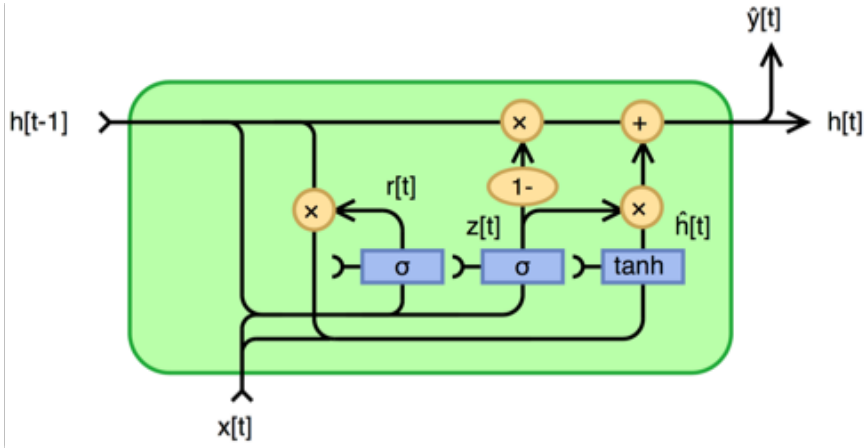
A RNN layer illustrated with feedback (left) and unwrapped in time (right)



Layers: GRU

Gated recurrent units

- Update gate
  - $z_t = \sigma_g (W_z x_t + U_z h_{t-1} + b_z)$
  - Parameters W_z, U_z and b_z
  - σ_g is a sigmoid, constrains gate to (0, 1)
- Reset gate
  - $r_t = \sigma_g (W_r x_t + U_r h_{t-1} + b_r)$
  - Parameters W_r, U_r and b_r
  - σ_g is a sigmoid, constrains gate to (0, 1)
- Output
  - $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \sigma_h (W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$
  - Parameters W_r, U_r and b_r
  - σ_h is a hyperbolic tan, constrains update term to (0, 1)
  - $\odot$ is the Hadamard (point wise) product



- If the reset gate is close to 0 then the previous state information is dropped from the update term
- If the update gate is close to 1 then the previous state is propagated cleanly through the unit

Layers: LSTM

Long short term memory

- Forget gate
  - $f_t = \sigma_g (W_f x_t + U_f h_{t-1} + b_f)$
  - Parameters W_f, U_f and b_f
  - σ_g is a sigmoid
  - How much to keep of the current state
- Input gate
  - $i_t = \sigma_g (W_i x_t + U_i h_{t-1} + b_i)$
  - Parameters W_i, U_i and b_i
  - σ_g is a sigmoid
  - How much to keep of the current input
- Output gate
  - $o_t = \sigma_g (W_o x_t + U_o h_{t-1} + b_o)$
  - Parameters W_o, U_o and b_o
  - σ_g is a sigmoid
  - How much to expose of the output

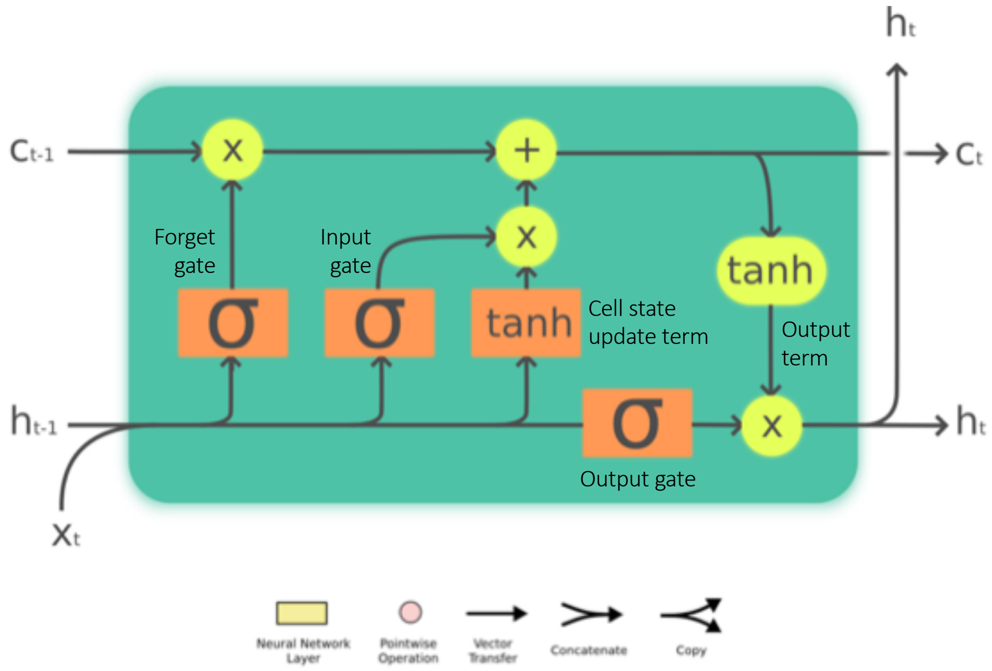


Figure from https://en.wikipedia.org/wiki/Long_short-term_memory

Layers: LSTM

Long short term memory

- Cell state update
  - $c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c (W_c x_t + U_c h_{t-1} + b_c)$
  - Parameters W_c, U_c and b_c
  - σ_c is a hyperbolic tangent
- Output
  - $h_t = o_t \odot \sigma_h(c_t)$
  - σ_h is a hyperbolic tangent
- Nota bene
  - The RNN and GRU cells used the output as the cell state
  - The LSTM cell has a separate output and cell states

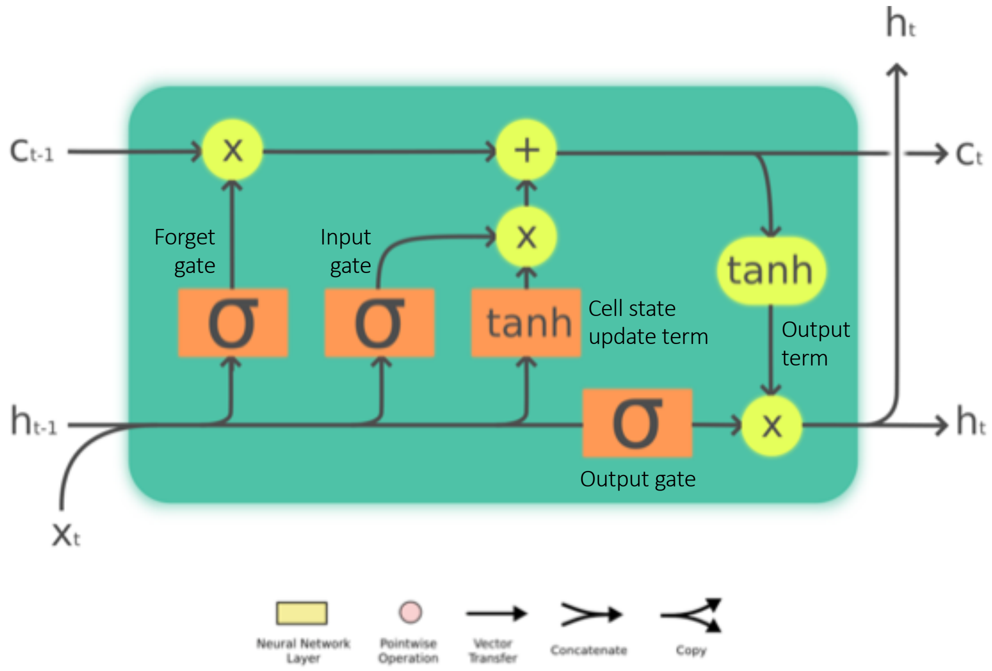


Figure from https://en.wikipedia.org/wiki/Long_short-term_memory

Layers: LSTM

Long short term memory

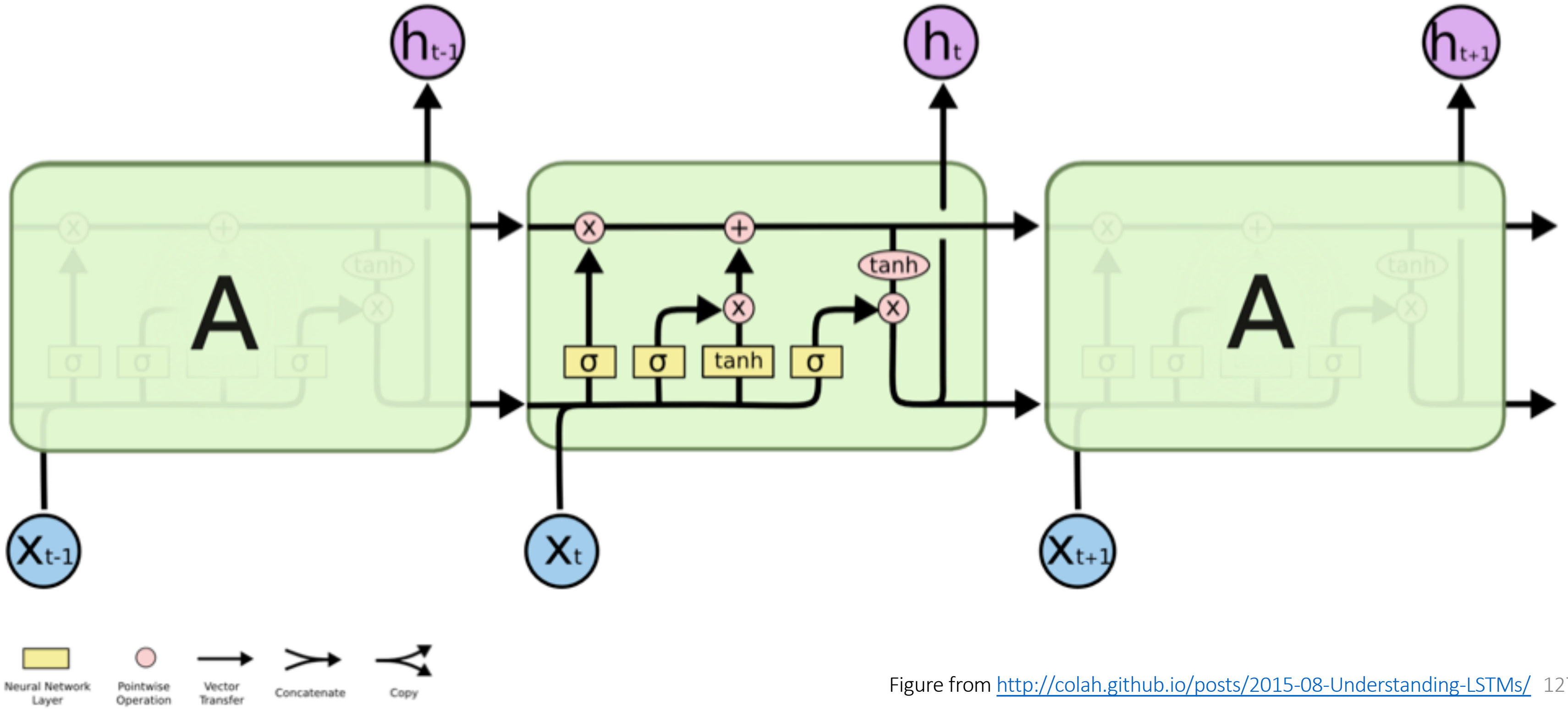
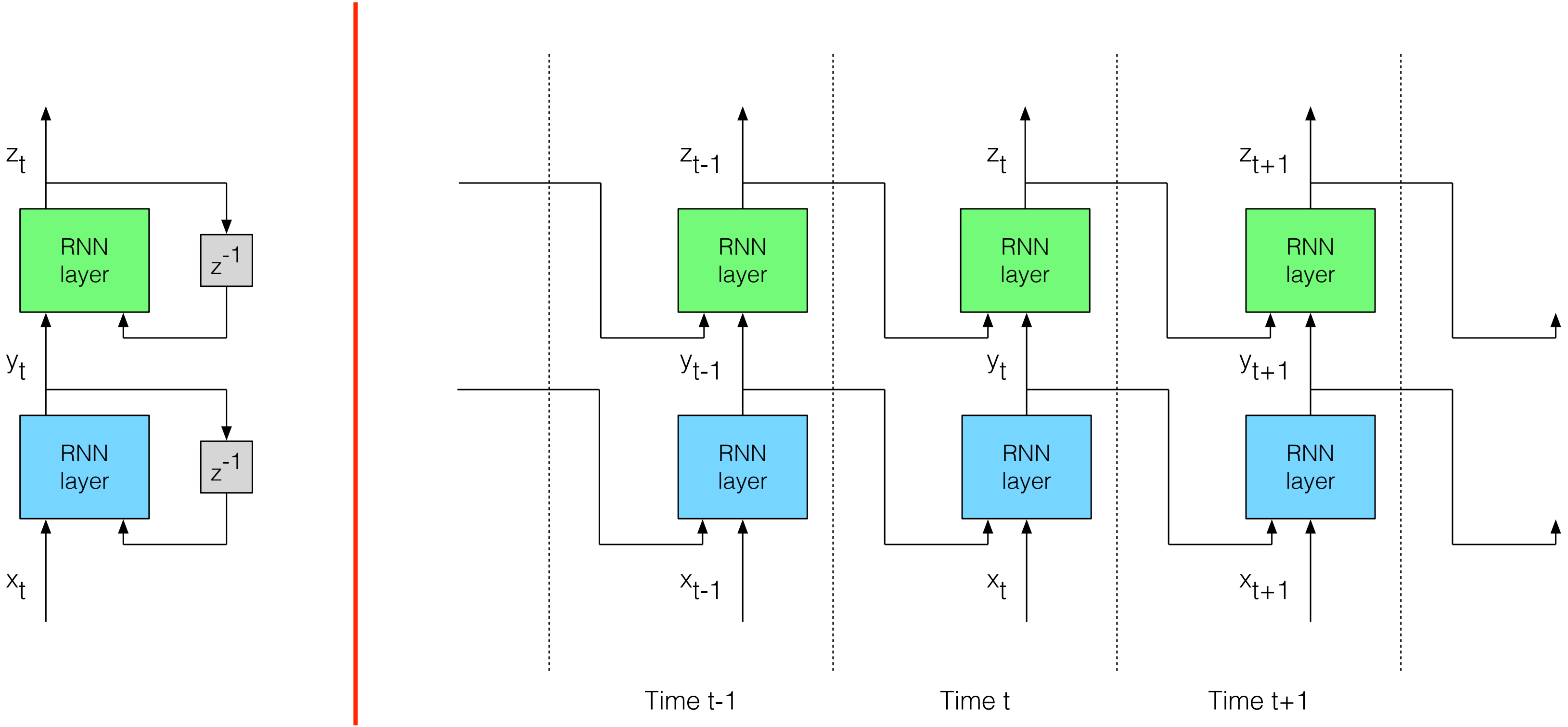


Figure from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

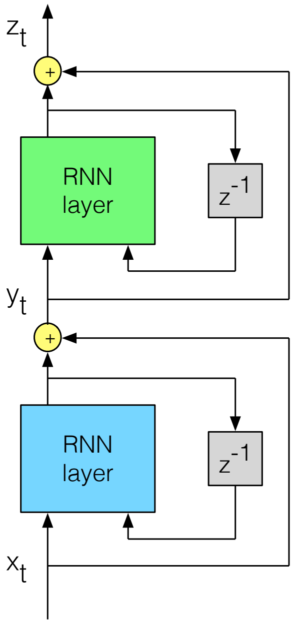
Networks: Stacked

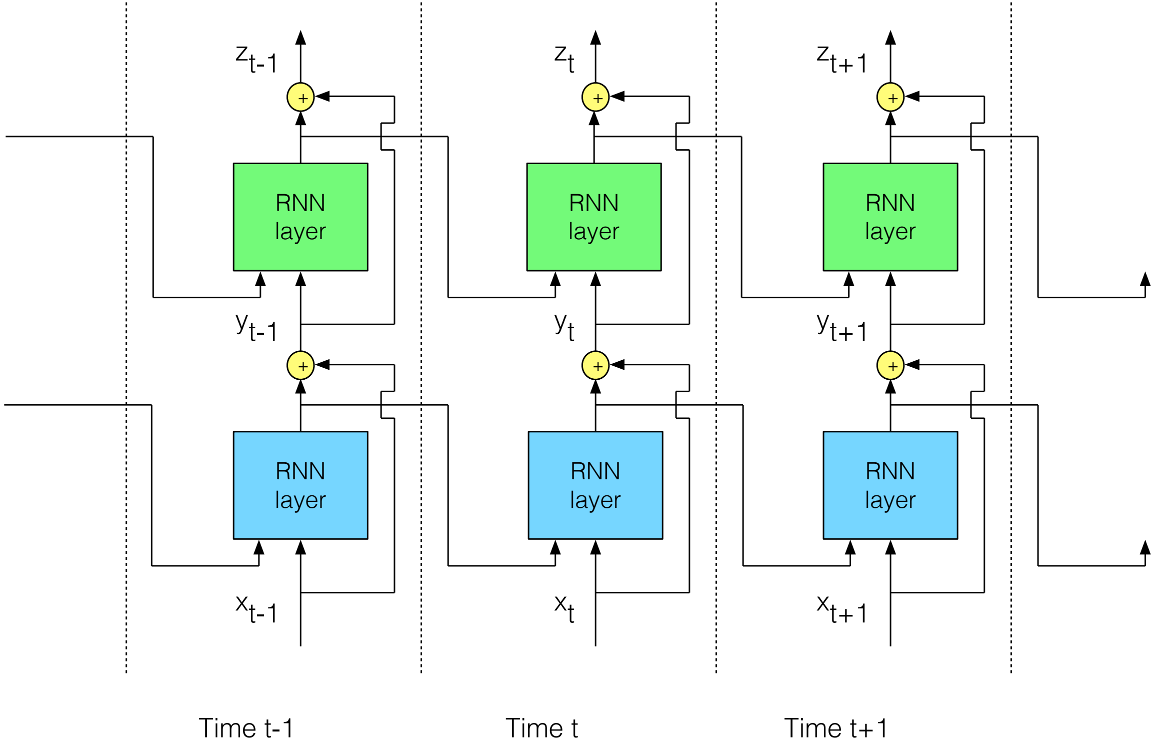
RNNs exploit sequential structure in time (frame) to map from weaker features to stronger features; figure shown for a RNN or GRU cell



Networks: Residual

This would typically be part of a stacked RNN design; figure shown for a RNN or GRU cell





Networks: Bi Directional

Bi directional nets have separate forward and backward RNNs and variants for a layer that are combined (concatenate or add) to produce the layer output

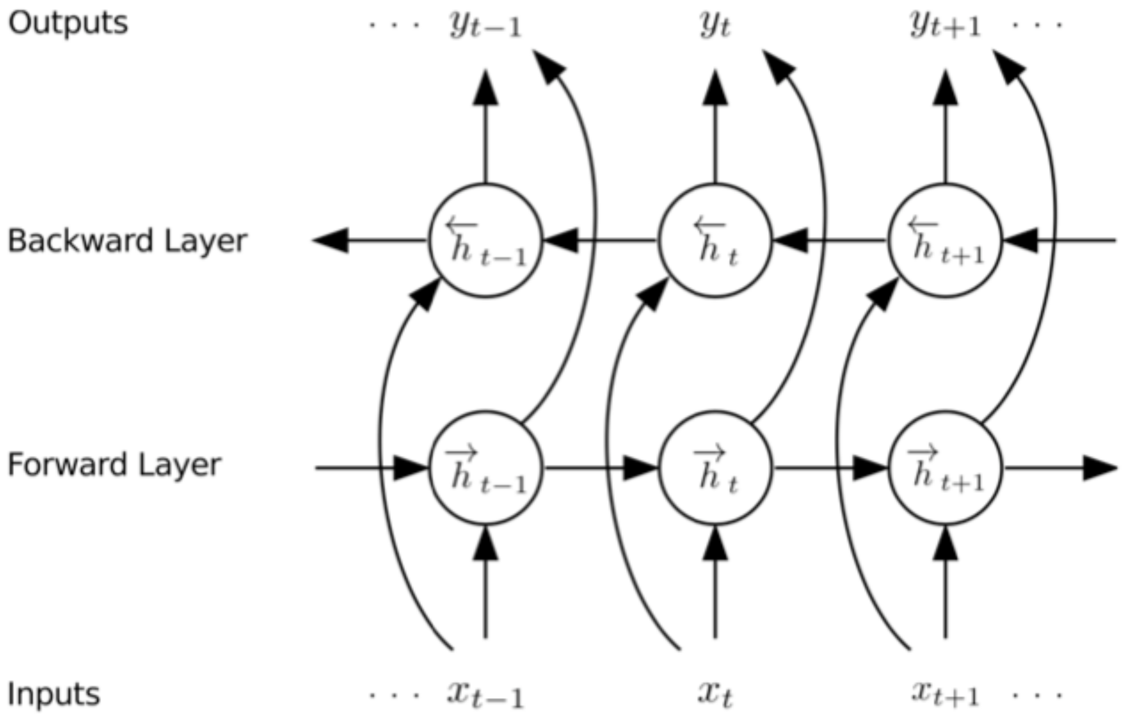


Figure from https://arxiv.org/abs/1303.5778

Networks: Pyramidal

Pyramidal networks aggregate multiple outputs from the previous level to reduce the sequence length the decoder has to deal with

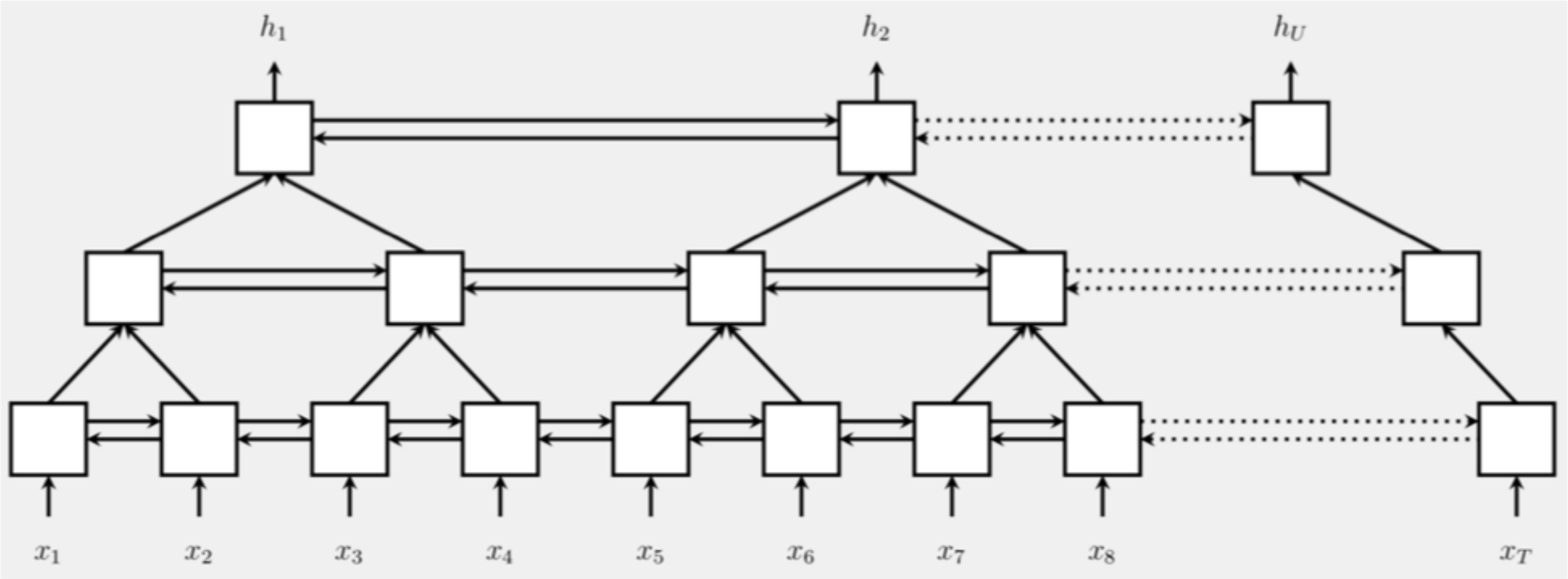


Figure from https://arxiv.org/abs/1508.01211

Networks: Sparsely Gated Mixtures Of Experts

Multiple feed forward networks between layers in a stacked RNN with a gating mechanism to select a small subset at a time (a mixture of experts)

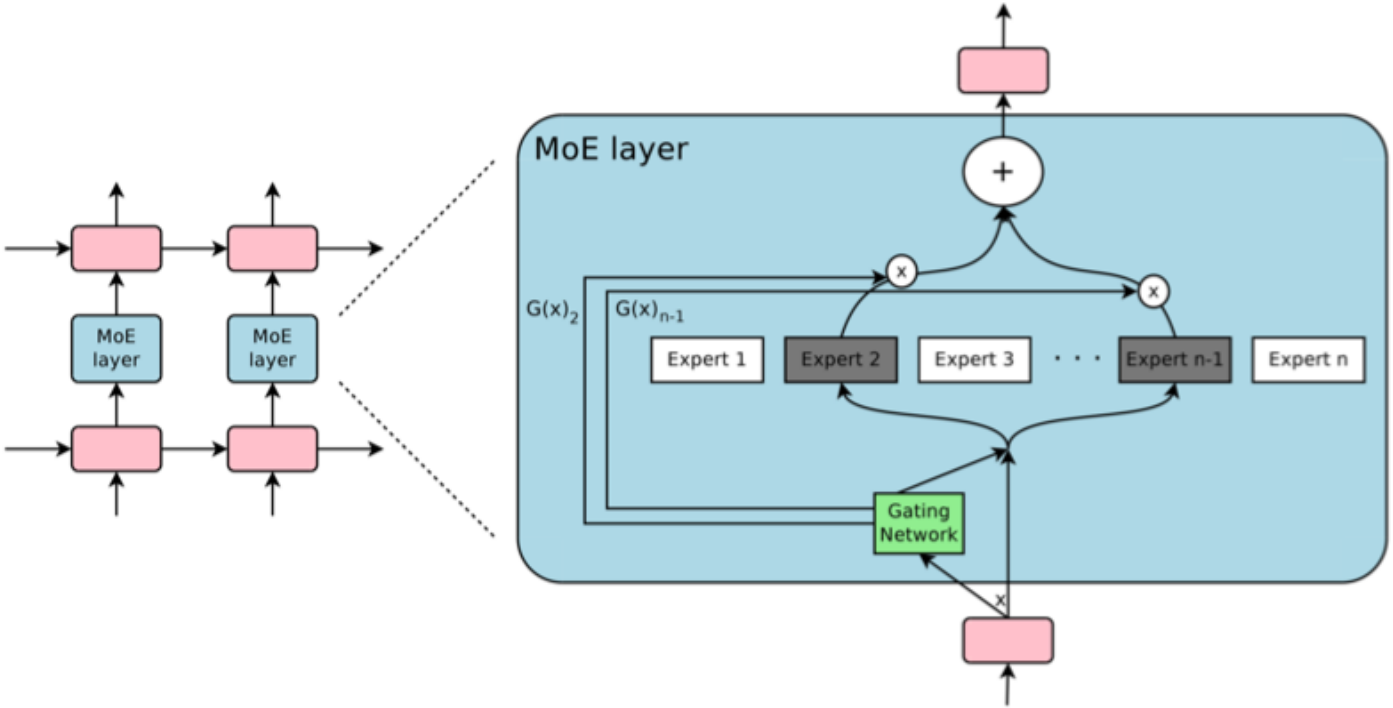
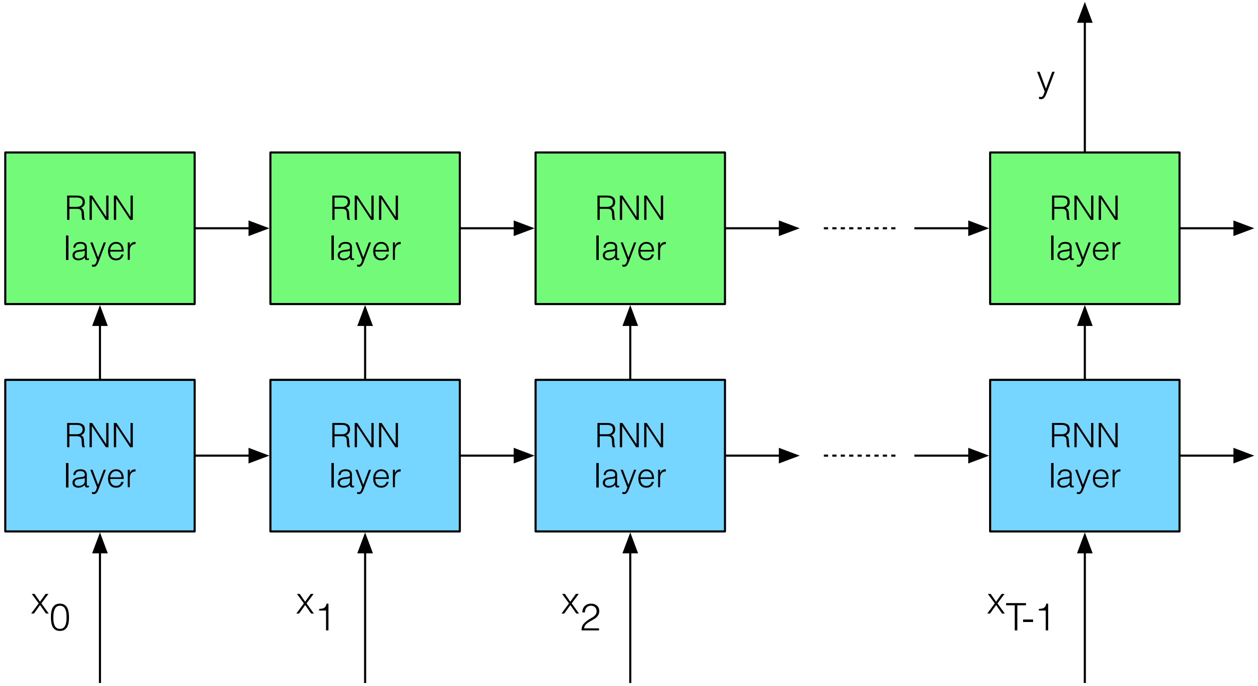


Figure from https://arxiv.org/abs/1701.06538

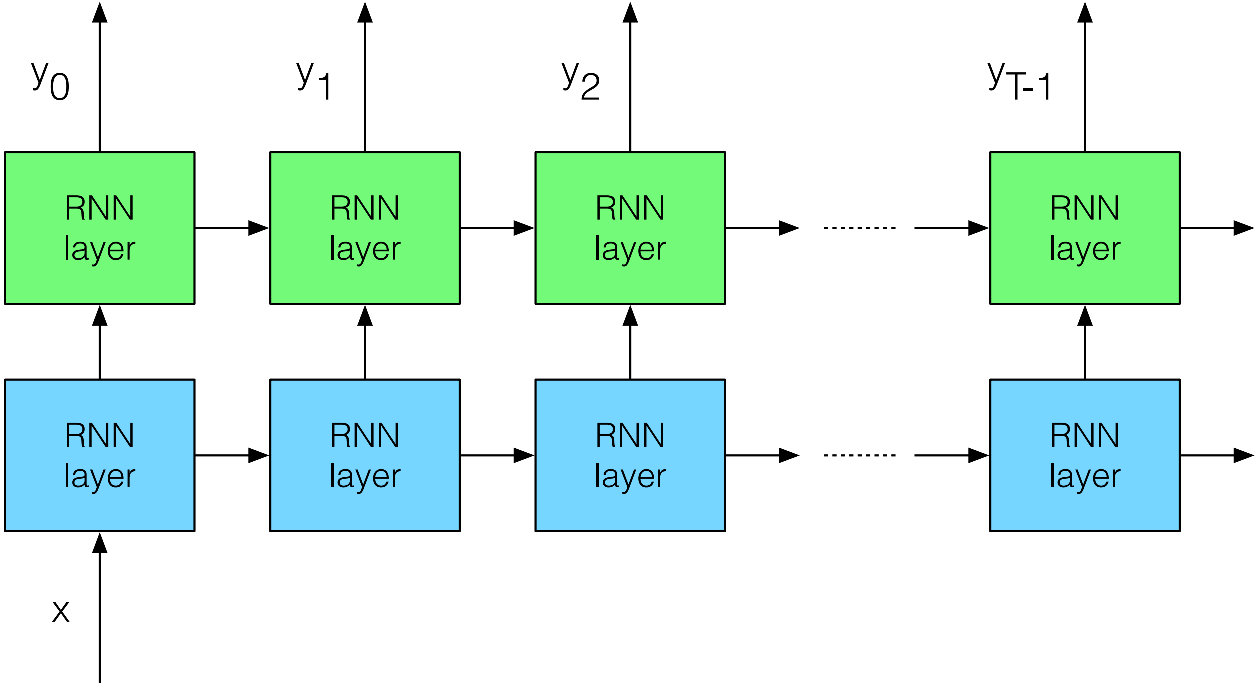
Mappings: Many Inputs To 1 Output

Illustrated here with a stacked 2 layer uni directional RNN



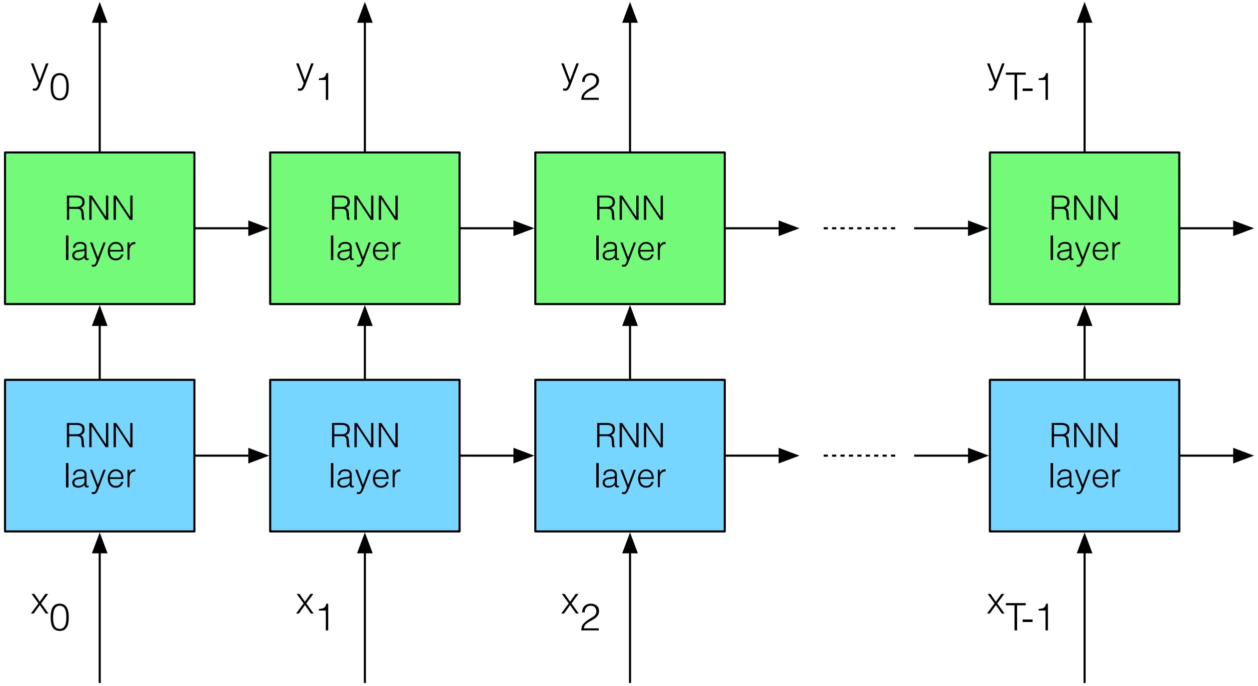
Mappings: 1 Input To Many Outputs

Illustrated here with a stacked 2 layer uni directional RNN



Mappings: Many Inputs To Many Outputs

Illustrated here with a stacked 2 layer uni directional RNN



A Brief Pause

- So far we've seen ...
  - A menu of different cell types with different memory properties
  - A number of different ways of connecting these cells into networks
  - All sorts of different input output mappings
  - (and there are many more different options not covered here)

- How do you design a RNN architecture for solving a problem with sequential structure?

- Think back to the beginning of this set of slides
  - You design a network to achieve a goal (never lose sight of this)
  - What is the goal of the problem you're trying to solve?
  - For this problem what's the best architecture to transform inputs to features to classes?

Visualization

- Visualization for RNNs usually applies to looking at alignments in sequence to sequence models with RNNs used for the encoder and decoder

- We'll look at this in the appropriate application sections

RNN Related References

- A critical review of recurrent neural networks for sequence learning
  - https://arxiv.org/abs/1506.00019
- Recurrent neural networks
  - https://zsc.github.io/megvii-pku-dl-course/slides/Lecture9%20Recurrent%20Neural%20Networks.pdf
- Understanding LSTM networks
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs
- Attention and augmented recurrent neural networks
  - https://distill.pub/2016/augmented-rnns

Attention

Strategy

- Attention exploits localized structure in data
  - Keep this in mind as you're thinking about applying them to a problem
  - Some additional motivation for using attention
    - Accuracy – attention allows the use of specific input features that matter based on the current output feature being generated (not forcing a neighborhood (CNN) or sequential (RNN))
    - Computation – attention allows the computation of output features based only on a subset of input features; additionally, it can frequently be formulated as a dense matrix matrix multiplication

- Outline
  - Common use 1: encoder – attention – decoder network structures
  - Common use 2: self attention feature encoding
  - Visualization

Encoder – Attention – Decoder

- Attention layer
  - **A** is a K x N encoder feature matrix and $\mathbf{b}_m$ is a K x 1 decoder key at position m
  - $\mathbf{s}_m^T = \mathbf{b}_m^T \mathbf{A}$ is a 1 x N attention score for position m
  - $\boldsymbol{\alpha}_m = \text{softmax}(\mathbf{s}_m)$ is a N x 1 attention distribution for position m
  - $\mathbf{c}_m = \mathbf{A}\,\boldsymbol{\alpha}_m$ is a K x 1 attention output
- The output $\mathbf{c}_m$ is a linear combination of encoded features appropriate for the decoder at position m

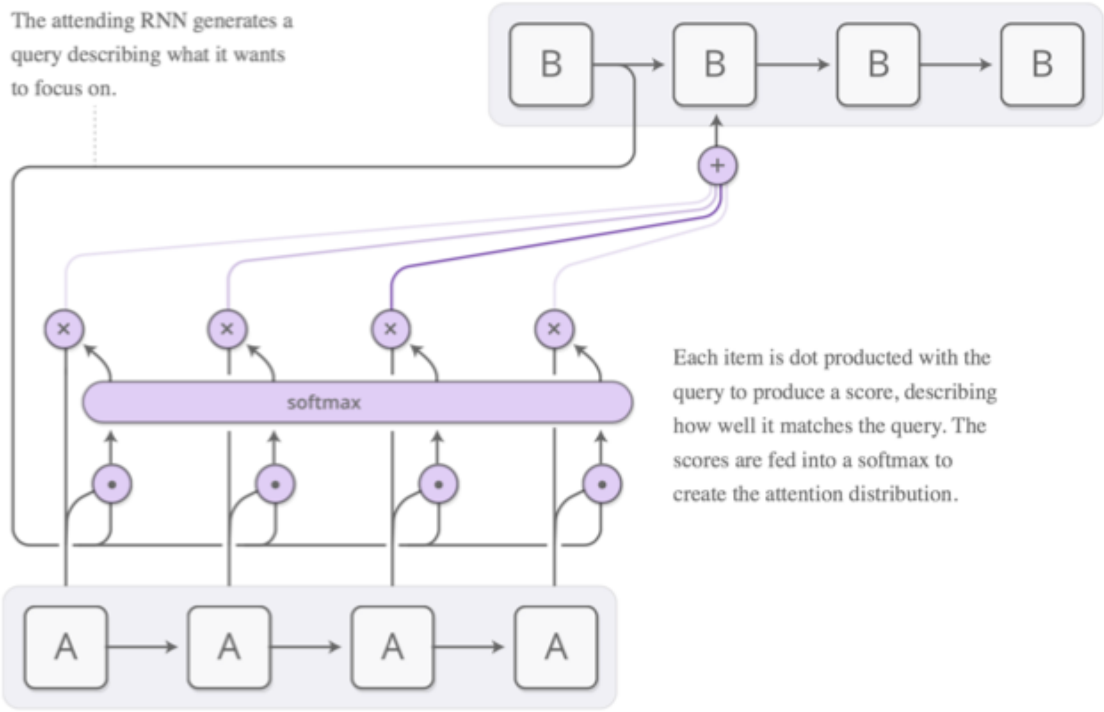


Figure from https://distill.pub/2016/augmented-rnns/

A Few Variations

- Soft (global) vs hard (~ local) attention
  - Soft (global) is ideal but requires more compute
  - Hard (optimized window) is nice from a complexity view but not differential
  - Local (fixed window) is a like hard but differential
- Score functions
  - The previous example used a dot product to create an attention score
  - There are other possibilities
- Attention output vector use location
  - Early vs late binding in the decoder

Example score functions

- Cosine: $s_m(k) = \text{cosine}(\mathbf{b}_m, \mathbf{A}(:, k))$
- Concat: $s_m(k) = \mathbf{v}_a^T \tanh(\mathbf{W}_a\ [\mathbf{b}_m; \mathbf{A}(:, k)])$
- Location: $s_m(k) = \mathbf{W}_a\ \mathbf{b}_m$
- Dot product: $s_m(k) = \mathbf{b}_m\ \mathbf{A}(:, k)$
- Generalized: $s_m(k) = \mathbf{b}_m\ \mathbf{W}_a\ \mathbf{A}(:, k) / K^{1/2}$

Note: soft max is applied to the score to determine the attention distribution

Table adapted from https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Self Attention

- Thought train
  - The encoder attention decoder architecture used attention to determine an output $\mathbf{c}_m$ that's a combination of encoded features appropriate for the decoder at position m
  - However, attention can also be used in place of an RNN to transform from weaker features to stronger features in an encoder (or decoder) using single and multi head variants
  - It can also be combined with other network design strategies (e.g., residual layers) and building blocks (normalization, dense layers)
  - This was made famous in the Transformer architecture from the paper Attention is all you need

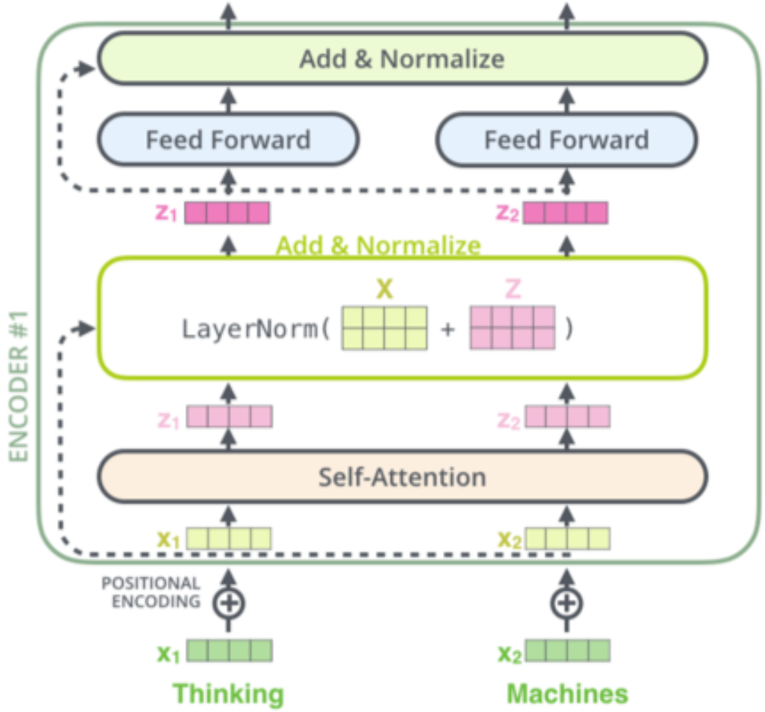


Figure from http://jalammar.github.io/illustrated-transformer/ based on https://arxiv.org/abs/1706.03762

The Transformer

No sequential dependency unlike RNNs, full context unlike CNNs

- Positional encoding
  - No sequential recurrent connections to enforce sequential structure in the output so a positional encoding is added
  - Sin() and cos() vectors with different frequencies are added to the inputs
- Encoder
  - 6 identical layers with 2 parts each
  - Part 1 is a multi head self attention mechanism
  - Part 2 is a fully connected layer
  - Each part includes normalization as y = LayerNorm(x + Part1or2(x)) and dropout
- Decoder
  - 6 identical layers with 3 parts each
  - Part 1 is a multi head self attention mechanism identical to encoder part 1
  - Part 2 is a multi head attention mechanism applied to the encoder output
  - Part 3 is a fully connected layer identical to encoder part 2
  - Each part includes normalization and dropout as in the encoder

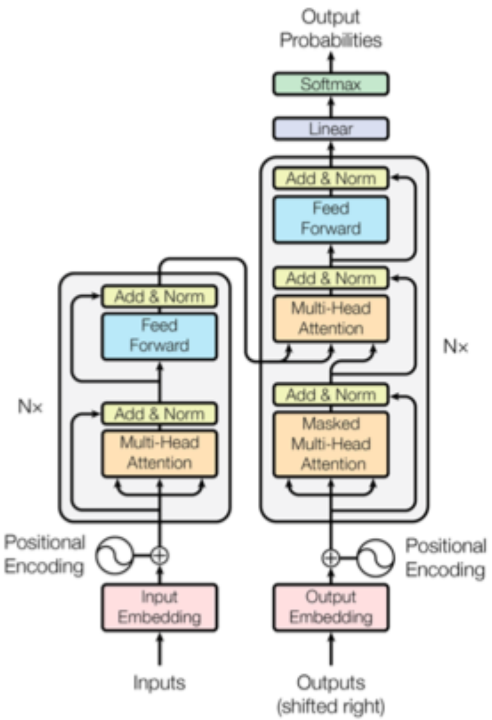


Figure from https://arxiv.org/abs/1706.03762

The Transformer

No sequential dependency unlike RNNs, full context unlike CNNs

Note that **X** is transposed here relative to the description in the linear algebra slides in the encoder – attention – decoder formulation (input feature vectors are in rows here)

- Query key value based attention mechanism
  - Single and multi head variants
  - Think of multi head as giving the network more degrees of freedom in learning localized information and passing it from 1 layer to the next

- Multi head attention equations (**W**s are learned weights)
  - Query, key and value matrices (attention input **X**)

    $\mathbf{Q}_i = \mathbf{X}\,\mathbf{W}_i^Q$

    $\mathbf{K}_i = \mathbf{X}\,\mathbf{W}_i^K$

    $\mathbf{V}_i = \mathbf{X}\,\mathbf{W}_i^V$
  - Attention($\mathbf{Q}_i$, $\mathbf{K}_i$, $\mathbf{V}_i$) = softmax($\mathbf{Q}_i\,\mathbf{K}_i^T$ / sqrt(d_k)) $\mathbf{V}_i$ = $\mathbf{head}_i$
  - Output = Concat($\mathbf{head}_1$, ..., $\mathbf{head}_h$) $\mathbf{W}^O$

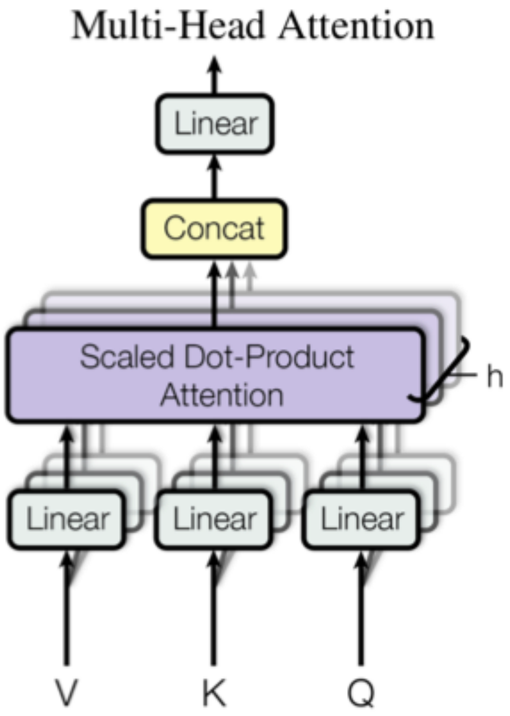


Figure from https://arxiv.org/abs/1706.03762

The Transformer

A visualization to help with some of the information on the previous slides

Note that **X** is transposed here relative to the description in the linear algebra slides in the encoder – attention – decoder formulation (input feature vectors are in rows here)

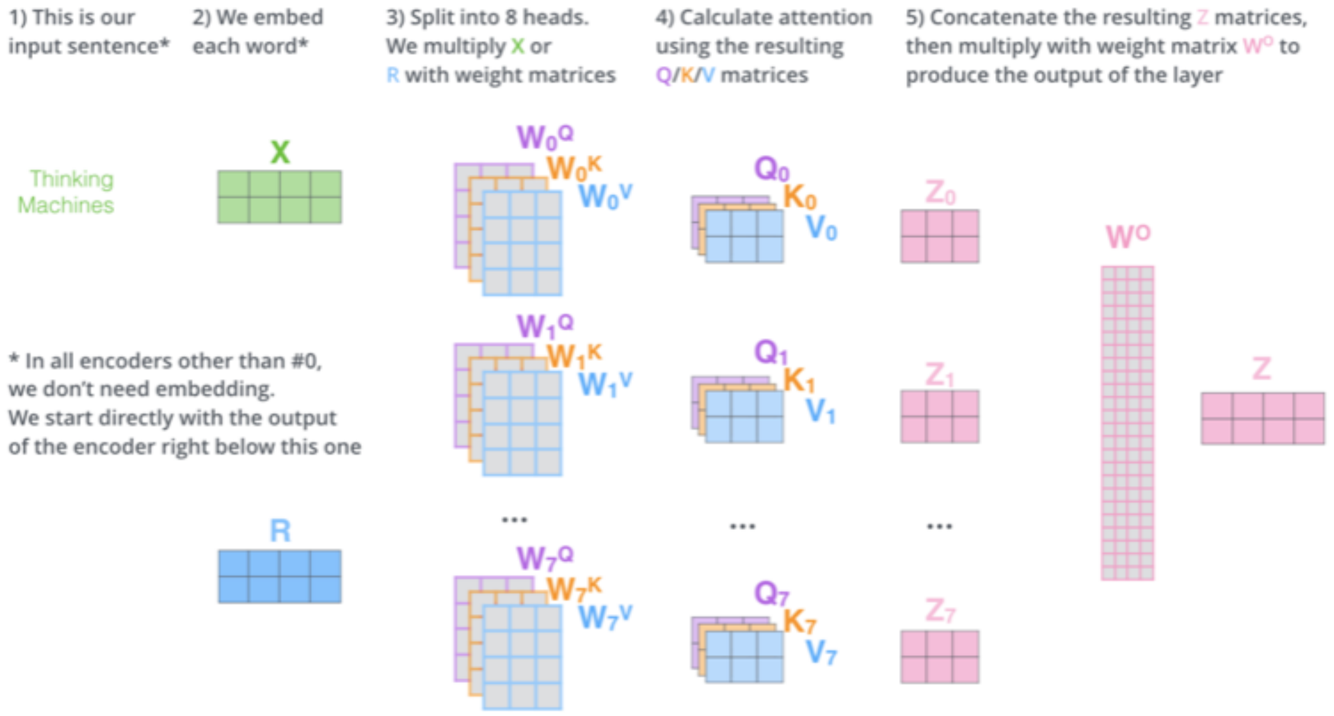


Figure from http://jalammar.github.io/illustrated-transformer/

Visualization

- Visualization for attention usually applies to looking at alignments in sequence to sequence models with attention or RNNs used for the encoder and decoder and an additional attention mechanism in between
- We'll look at this in the appropriate application sections

Combinations of Attention With xNNs

- The previously described Transformer combined attention and dense layers

- One model to learn them all
  - https://arxiv.org/abs/1706.05137
  - Combines attention and CNNs to make a general architecture capable of many tasks

- Universal transformers
  - https://arxiv.org/abs/1807.03819
  - Combines attention and RNNs to make a universal model of computation

- Neural Turing machines
  - Combines attention and an external memory
  - https://arxiv.org/abs/1410.5401
  - http://www.robots.ox.ac.uk/~tvg/publications/talks/NeuralTuringMachines.pdf

An Additional Take Away

- It feels like we're still in the early days of figuring out how to incorporate attention into different designs and applications
- It's important to understand the motivation and realize that there will be continual developments in this area

Attention Related References

- Neural machine translation by jointly learning to align and translate
  - https://arxiv.org/abs/1409.0473
- Neural Turing machines
  - https://arxiv.org/abs/1410.5401
  - http://www.robots.ox.ac.uk/~tvg/publications/talks/NeuralTuringMachines.pdf
- Show, attend and tell: neural image caption generation with visual attention
  - https://arxiv.org/abs/1502.03044
- End-to-end memory networks
  - https://arxiv.org/abs/1503.08895
- Pointer networks
  - https://arxiv.org/abs/1506.03134
- Effective approaches to attention-based neural machine translation
  - https://arxiv.org/abs/1508.04025
- Long short-term memory-networks for machine reading
  - https://arxiv.org/abs/1601.06733

Attention Related References

- A structured self-attentive sentence embedding
  - https://arxiv.org/abs/1703.03130
- Massive exploration of neural machine translation architectures
  - https://arxiv.org/abs/1703.03906
- Attention is all you need
  - https://arxiv.org/abs/1706.03762
- One model to learn them all
  - https://arxiv.org/abs/1706.05137
- A simple neural attentive meta-learner
  - https://arxiv.org/abs/1707.03141
- Self-attention generative adversarial networks
  - https://arxiv.org/abs/1805.08318
- Universal transformers
  - https://arxiv.org/abs/1807.03819
- Implementing neural Turing machines
  - https://arxiv.org/abs/1807.08518

Attention Related References

- Attention and augmented recurrent neural networks
  - https://distill.pub/2016/augmented-rnns
- The illustrated transformer
  - http://jalammar.github.io/illustrated-transformer/
- Attention in neural networks and how to use it
  - http://akosiorek.github.io/ml/2017/10/14/visual-attention.html
- Attention? Attention!
  - https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

References

List

- References for individual networks are listed on the slide on which they occur, the references listed here are more general and apply to many networks

- 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