

Design

Arthur J. Redfern
arthur.redfern@utdallas.edu

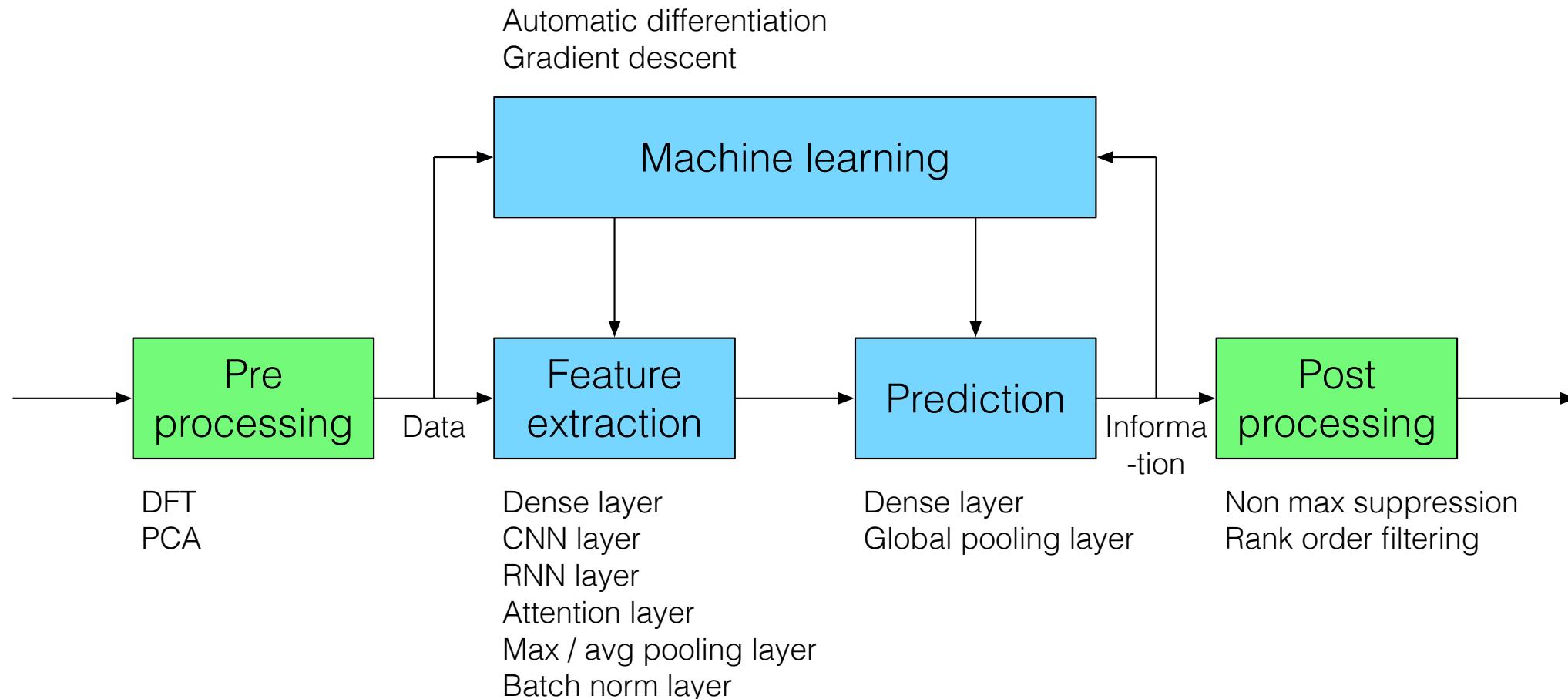
Outline

- Context
- Goals
- Preliminaries
- CNNs
- RNNs
- Transformers
- References

Context

The First Part Of The Course

Provided the basic building blocks 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)
 - The 3rd part of the course will focus on the decoder
 - Review successful network designs and understand the keys behind them
 - Training to improve convergence and improve generalization
 - 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, language, speech, ...

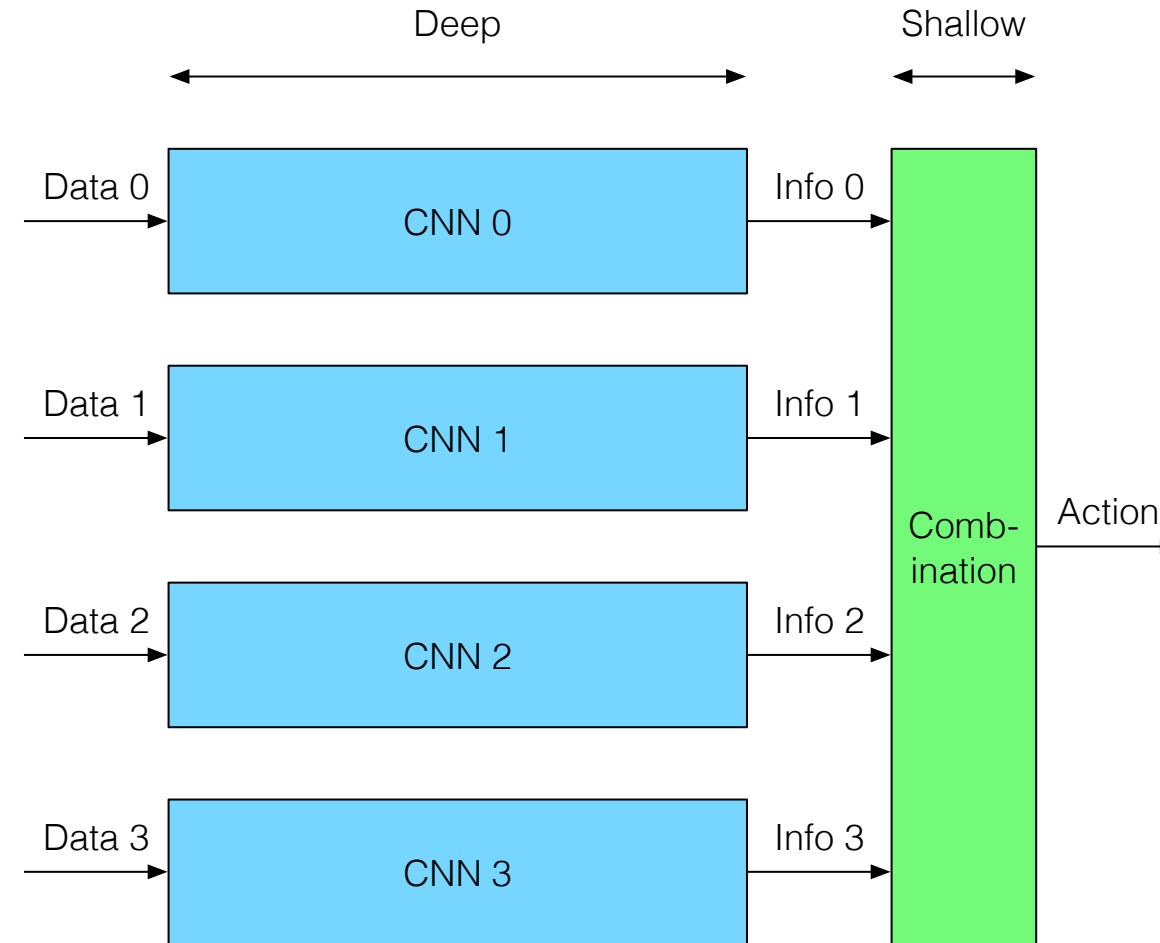
Goals

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
- You probably want to 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



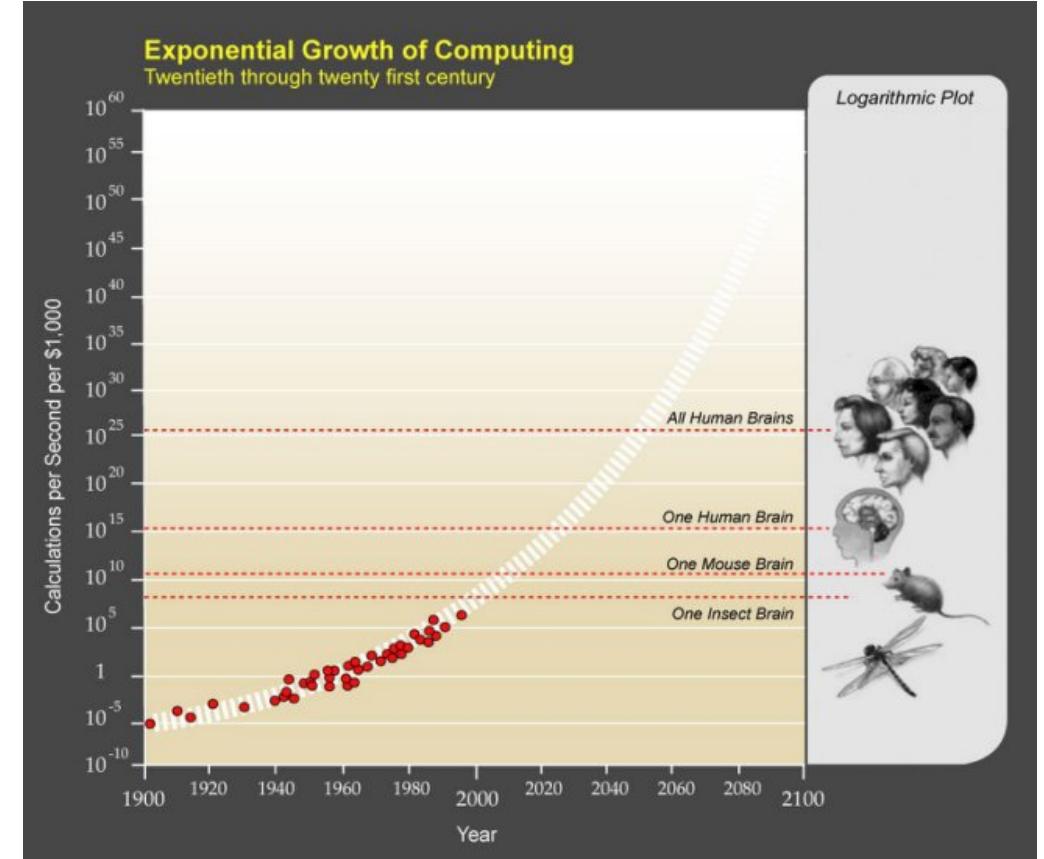
This Semester And This Class

- 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?



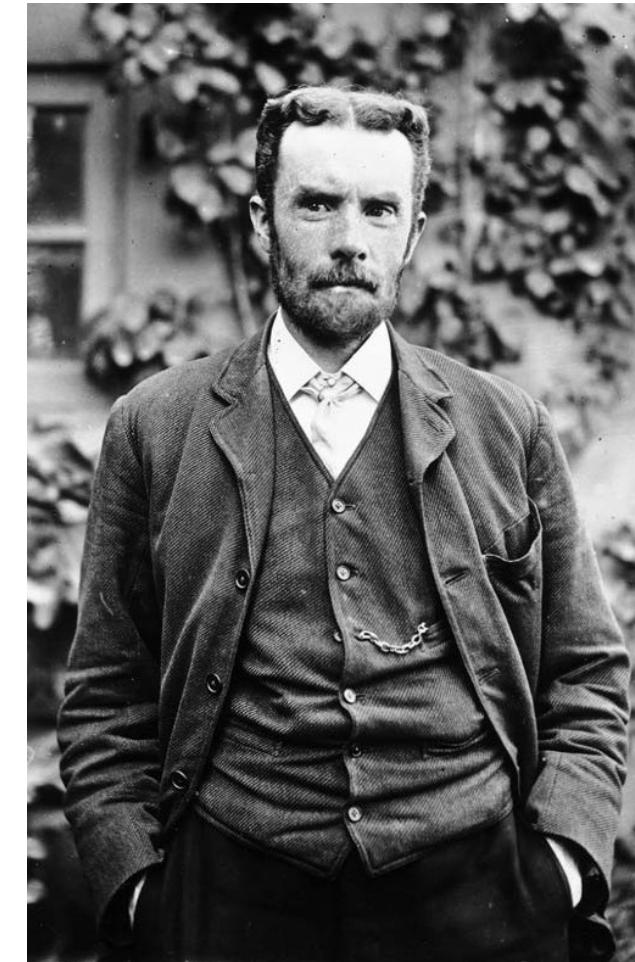
Theory Or Practice?

- 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 (\rightarrow 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?”

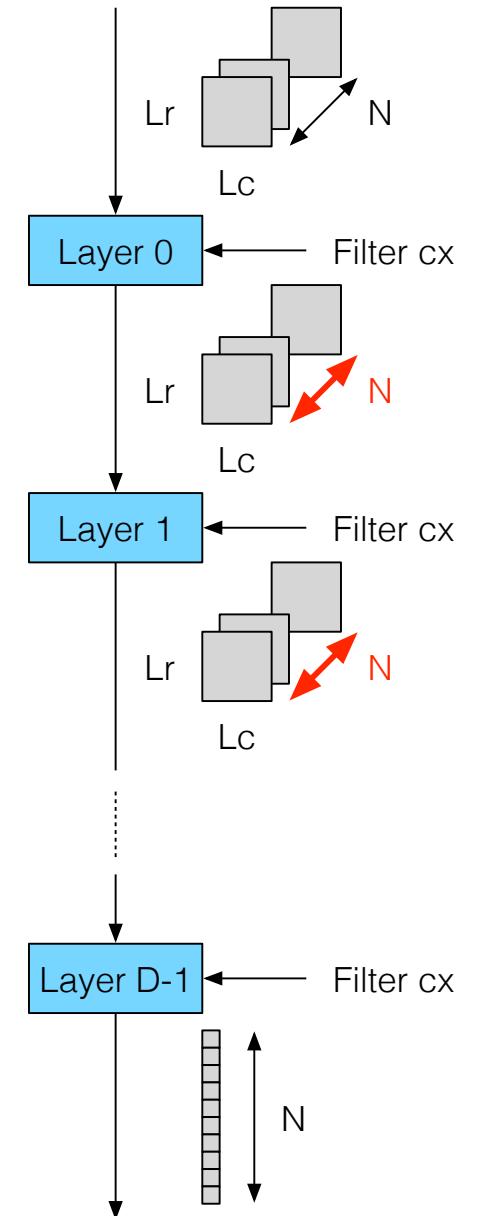
Quotes != proofs, but regardless, I like this



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 function 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 (speed)
 - 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

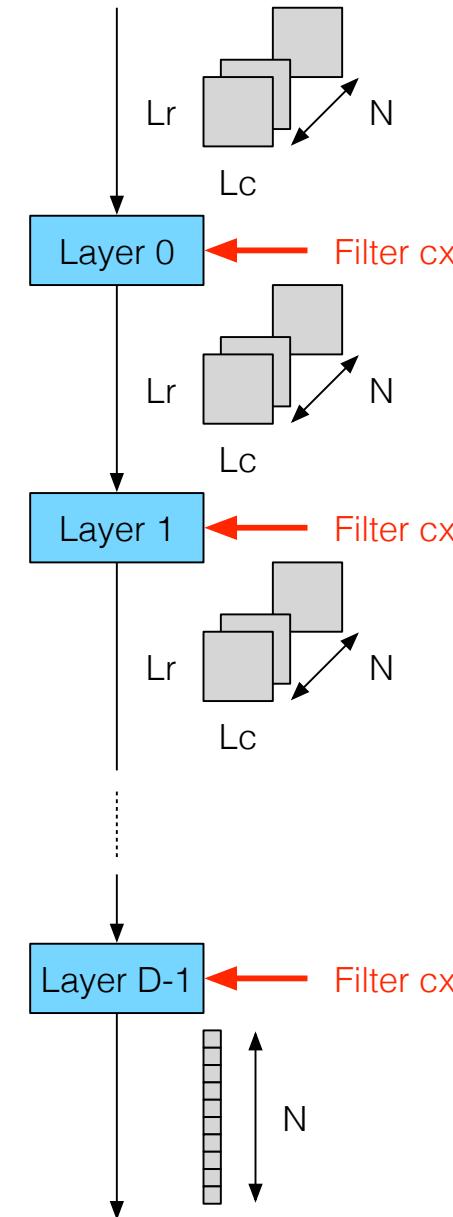


Depth gives an exponential gain in representational power, though there appears to be a limit to the ideal depth based on problem complexity

Do deeper convolutional networks perform better?
<https://arxiv.org/abs/2010.09610>

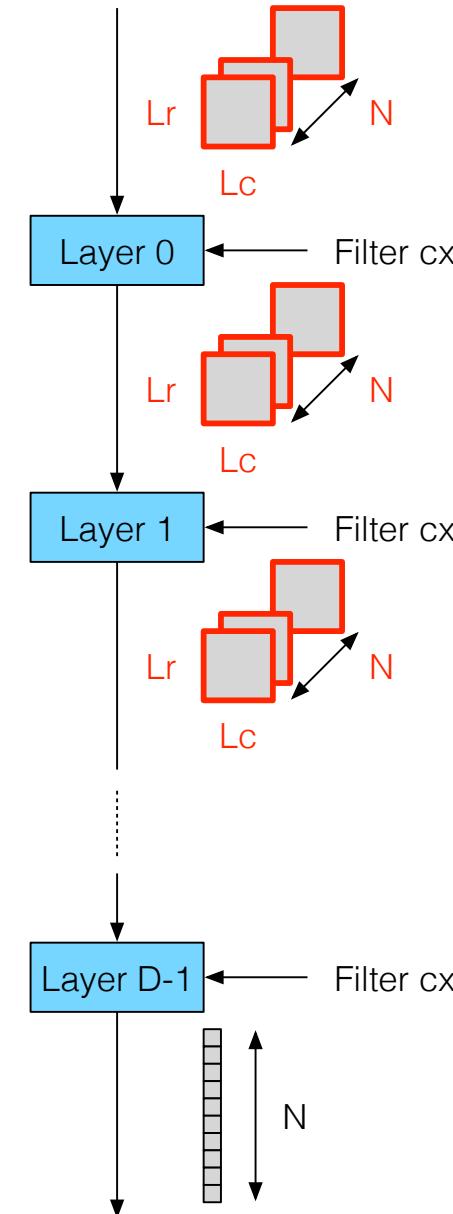
Network Size

- Filter coefficients ($F_r \times F_c \times N_i \times N_o$ and bias N_o per linear transformation)
 - Together with the network structure contain all knowledge 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 knowledge, so there's a balance



Data And Information Size

- Feature maps ($N_{i/o} \times L_r \times L_c$ 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

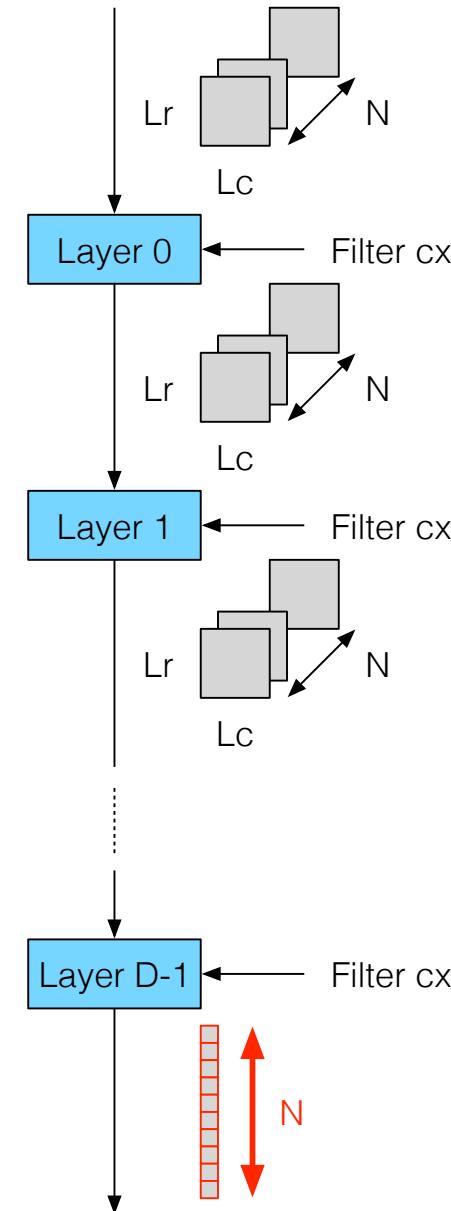


Networks Implement Functions

- 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, language and speech
 - 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 the desired output of the function is determined
 - 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 function approximation proof from the calculus lecture
 - Lot's of problems we care about are really pretty complex



CNNs

CNN Design Section Overview

- CNNs exploit local spatial structure in data
 - Keep this in mind as you're thinking about applying them to a problem
- Overview
 - Tail
 - Body
 - Serial, parallel, dense, residual, conditional building blocks
 - ML based architecture search
 - Head

A Framework For Thinking About CNNs

- Feature extraction and prediction
 - Tail feature extraction (optional)
 - Body feature extraction
 - Head prediction
- 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

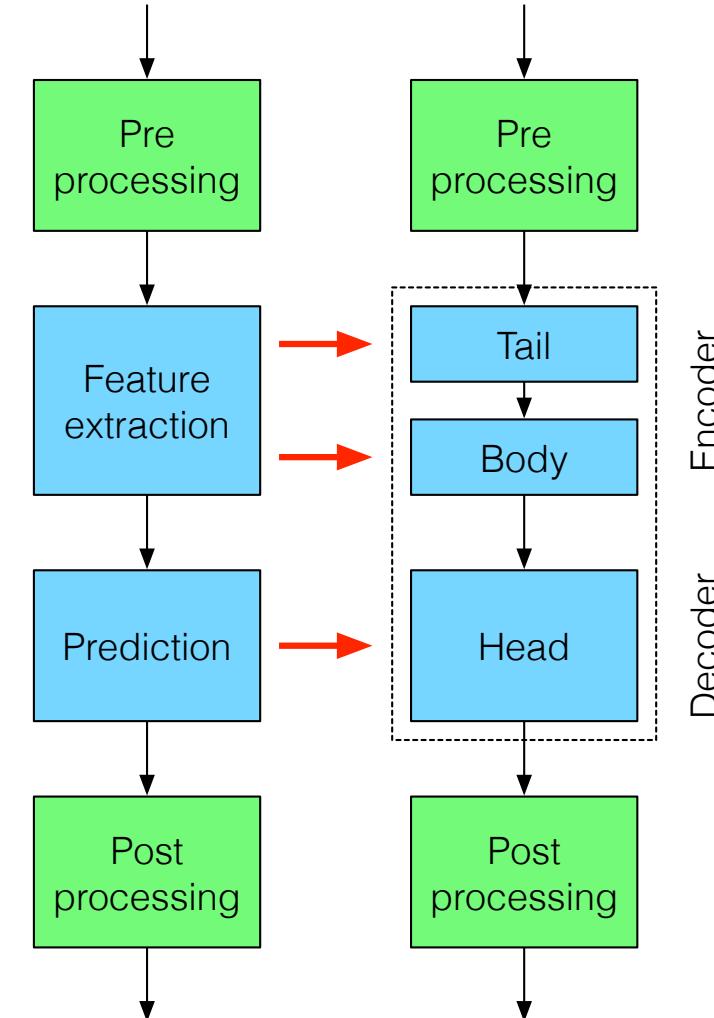
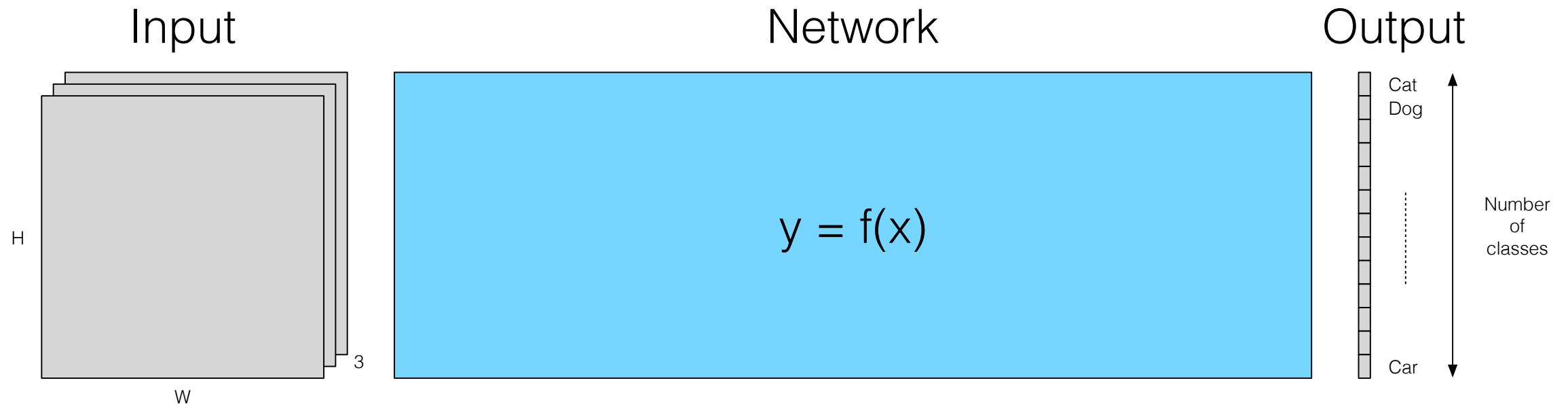


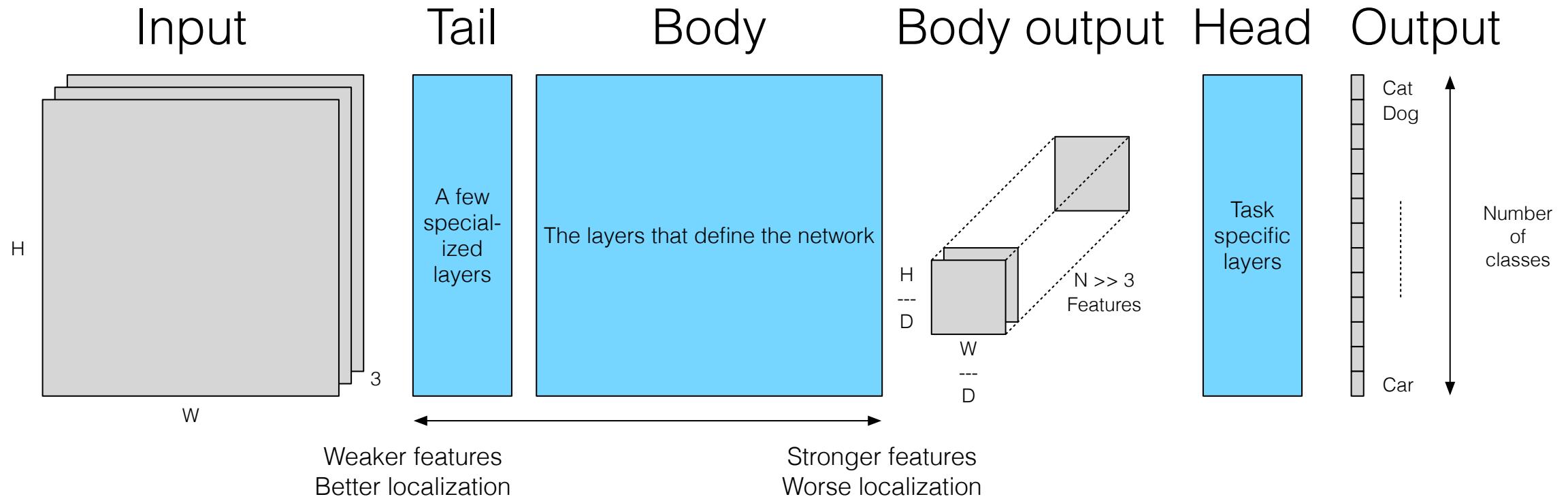
Image Classification As A Starting Point

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



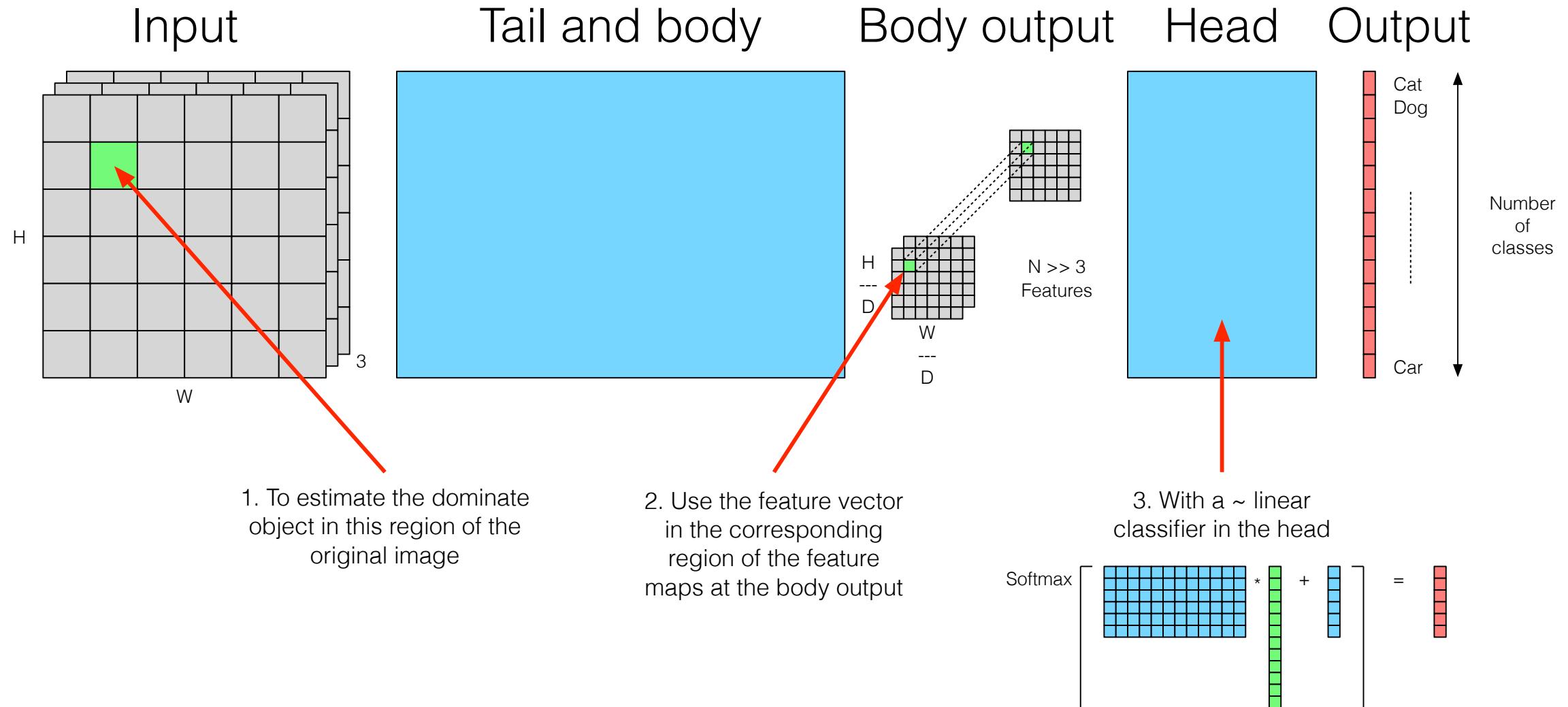
Tail Body And Head Decomposition

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



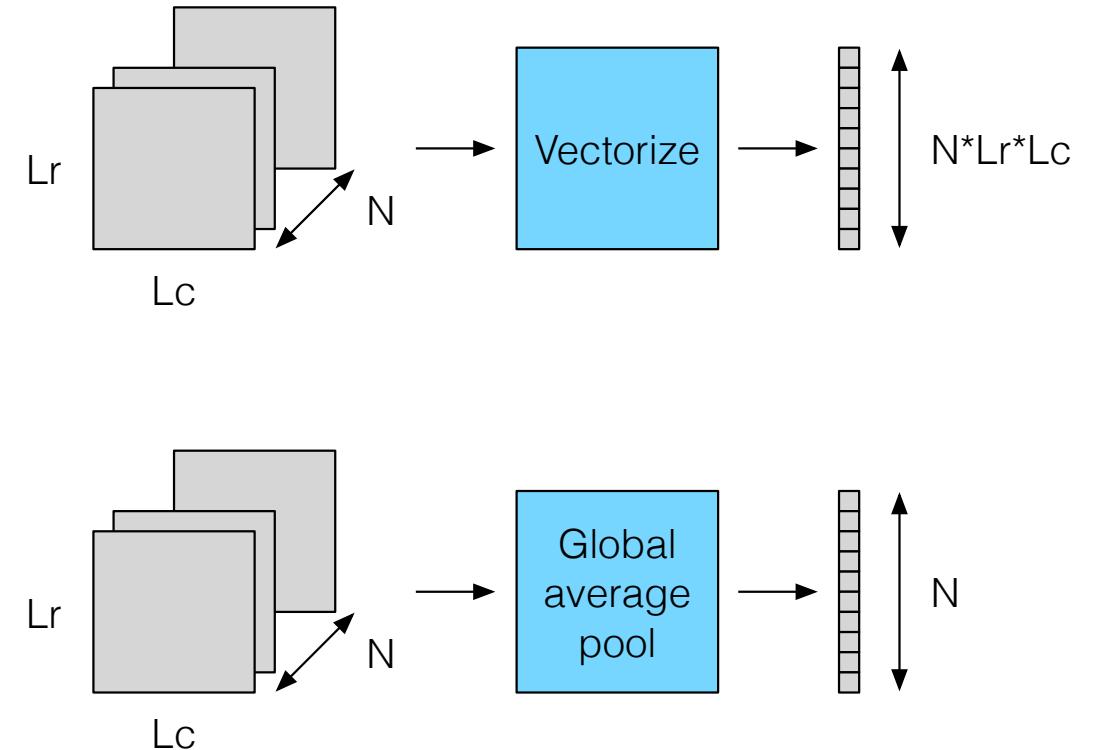
CNNs – Head (Decoder)

Conceptually How The Head Works



Feature Map To Feature Vector Transform

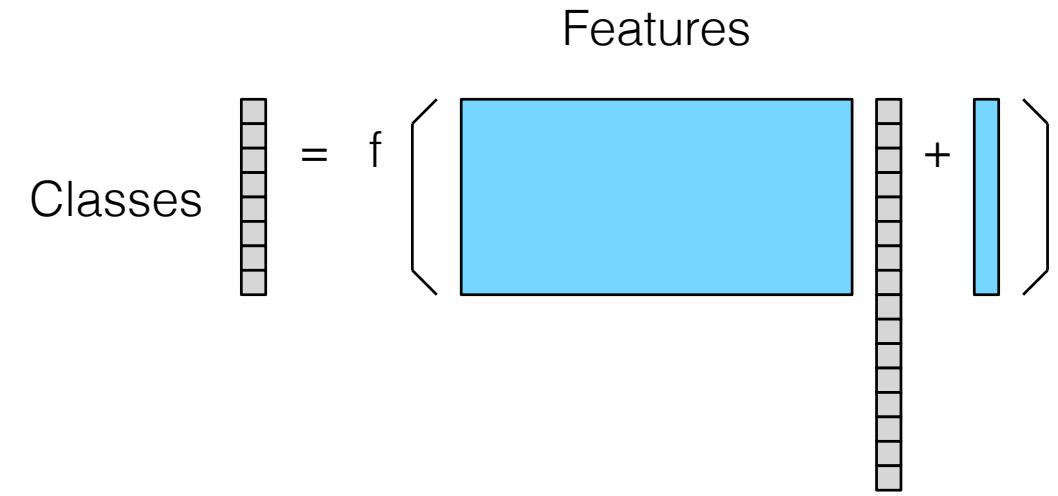
- 3D tensor to 1D vector transformation
 - Output of convolutional layers is $N \times L_r \times L_c$
 - Input to linear classifier is $N \times 1$
 - 2 common methods for transforming between the output of the convolutional layers to the input of the linear classifier
- Vectorize
 - Historically first
 - 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
- Global average pool
 - Most popular right now
 - Average features together for all pixels matching the requirements of image classification (1 label for whole image)
 - 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
 - With softmax to convert to probabilities (training)
 - With 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

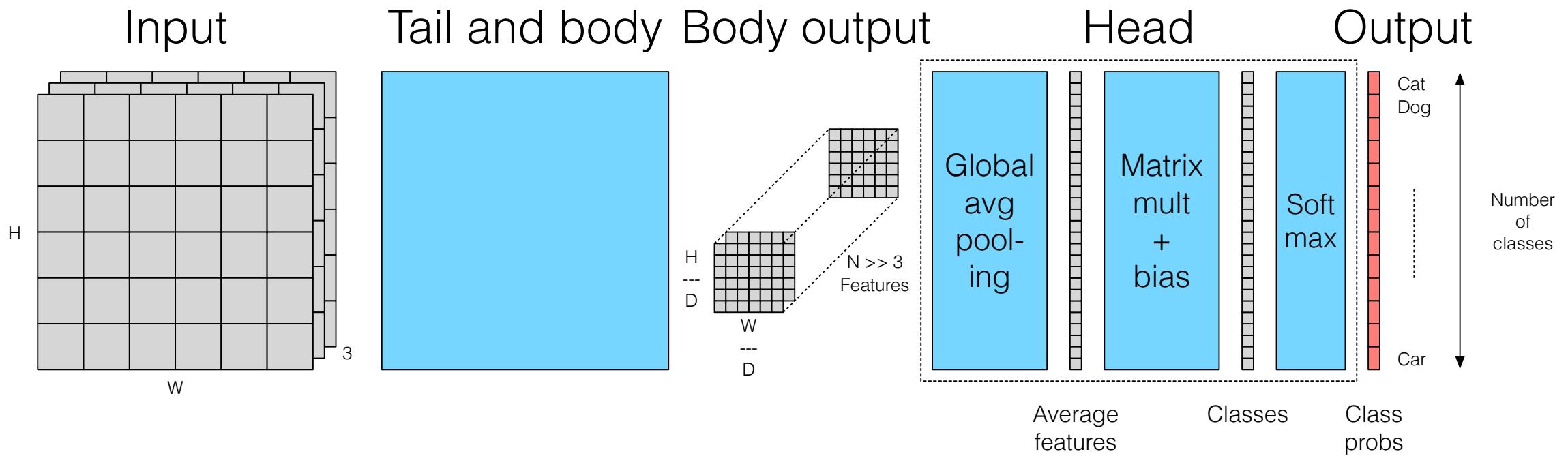


A downside of building efficient network tails and bodies is that the final classifier contributes to a higher fraction of the total parameters; for some commentary on this see

Do we need fully connected output layers in convolutional networks?

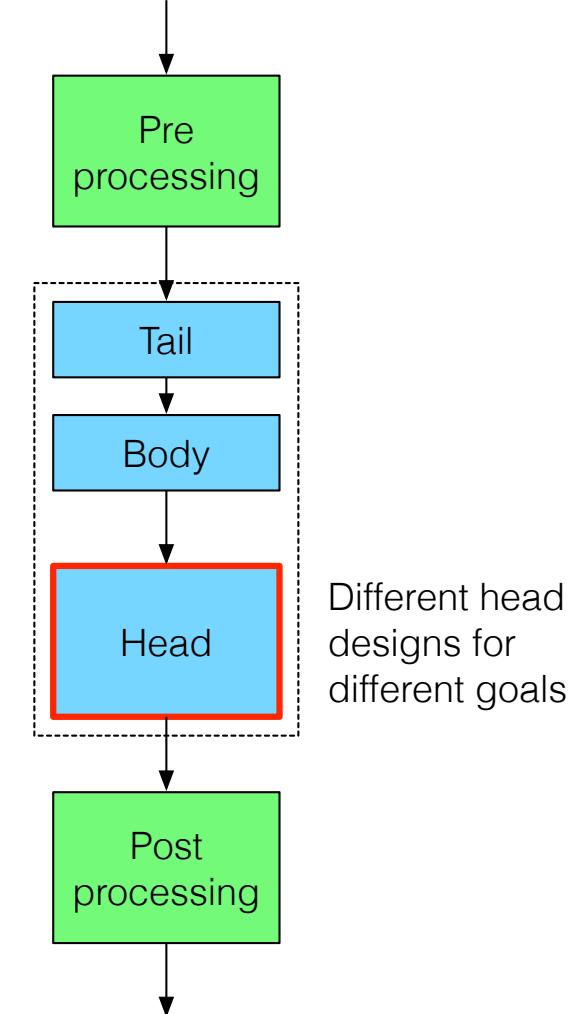
<https://arxiv.org/abs/2004.13587>

Putting It All Together



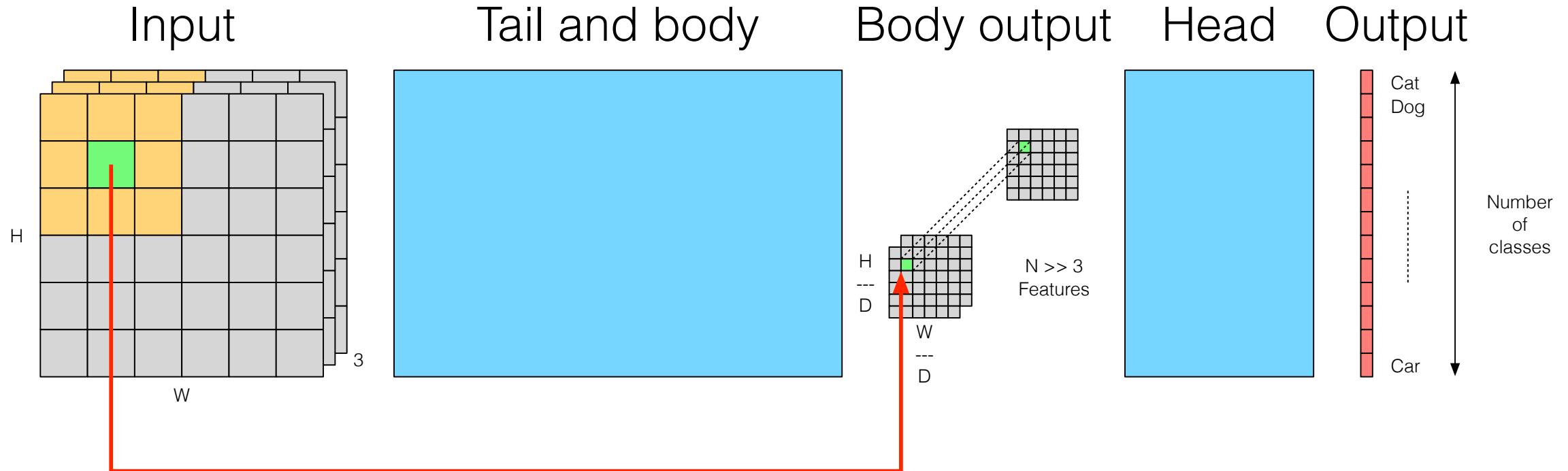
In The Near Future (App Part Of Course)

- 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



CNNs – Tail (Encoder Pt 1)

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

Receptive Field Size

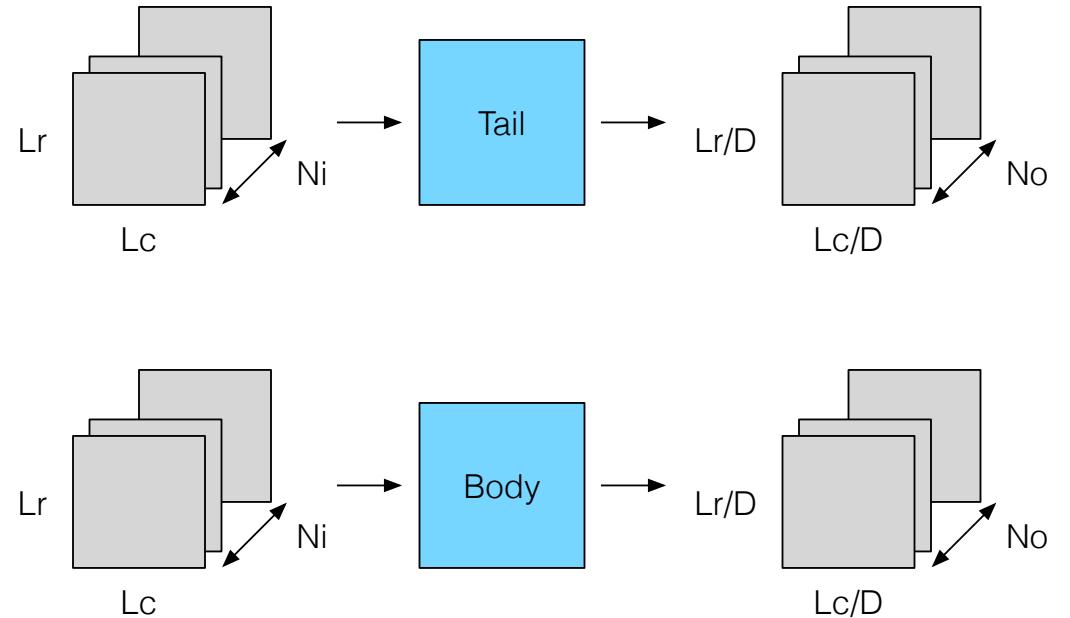
- Receptive field size at the head
 - How to compute in theory
 - But effectively smaller in practice (convolving a bunch of filters gives a \sim Gaussian shape)
- What does the classifier have to work with when making a decision
 - Idea of looking at the input through a screen of that size
 - Objects 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) then subtract $S - 1$ (to remove the outer edge)
- You can find the receptive field size at any place in the network by starting with a receptive field size of 1 at that location and using the above procedure to work backwards; this is 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)

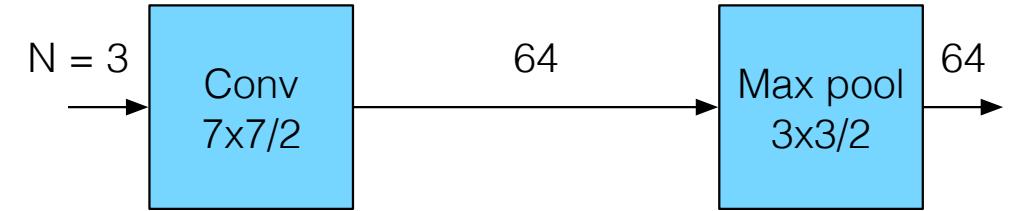
Why Distinguish Between The Tail And Body?

- 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 at all?
 - There are a few common / simple 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

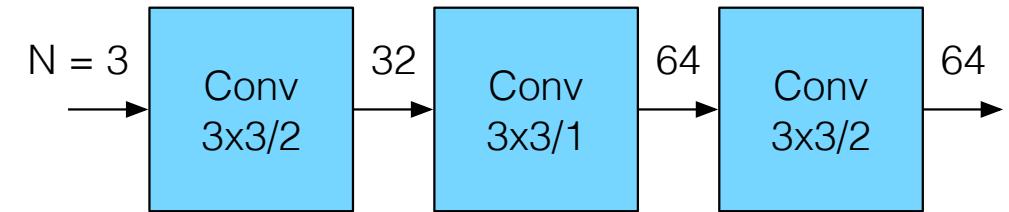


The Tail Maps Data To Weak Features

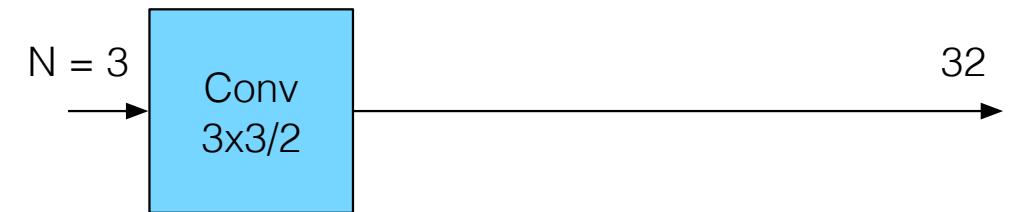
- Initial data to feature encoding
 - Optional (can be thought of as part of the body)



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



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

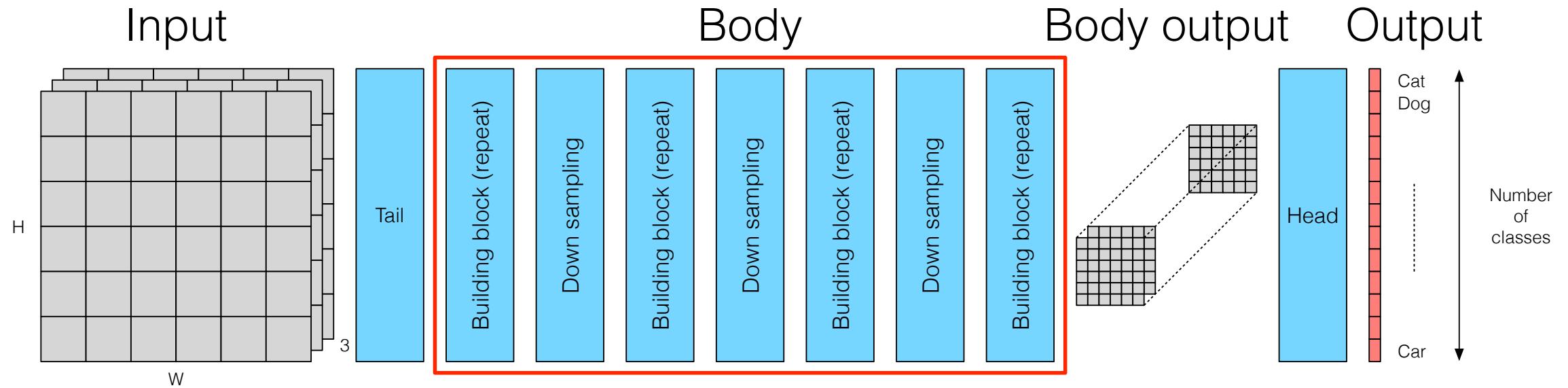


CNNs – Body (Encoder Pt 2)

So Far ...

- We've seen some standard tail designs
 - The start of feature extraction / encoding
 - Ex: $7 \times 7 / 2$ conv $\rightarrow 3 \times 3 / 2$ max pool
- We've seen some standard head designs
 - Prediction / decoding
 - Ex: global avg pool \rightarrow fully connected layer \rightarrow soft or arg max
 - Will look at more in the context of different applications

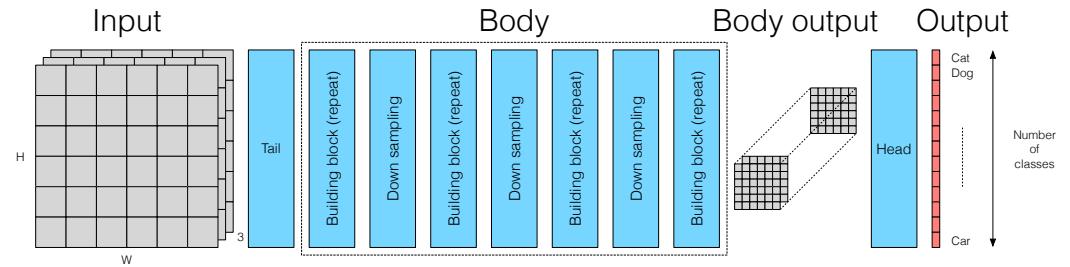
We'll Now Focus On The Body



- The network body (encoder part 2)
 - The main portion of feature extraction / encoding
 - The body is effectively what defines / names the network
 - Relative to the tail and head, 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

The Body Maps Weak Features To Strong Features

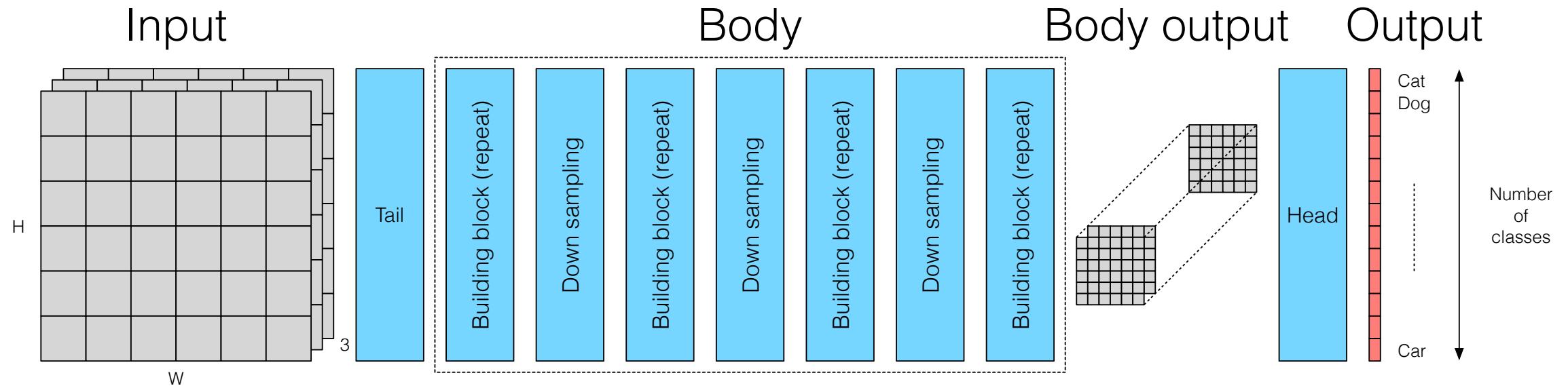
- Building blocks we'll see in network design followed by down sampling (strided convolution in the building block or pooling)
- Interpretation
 - Lower level features are more class agnostic
 - Less channels earlier in the network
 - Deeper features are more class specific
 - More channels later in 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 (though some recent work increases by ~ 2.5)
- Result is the data volume shrinks by 2
 - Data before = channels \times rows \times cols
 - Data after = $2^* \text{channels} \times \text{rows}/2 \times \text{cols}/2$
= data before / 2
 - Since compute in convolutional layers is proportional to the number of input channels * the number of output channels the amount of compute tends to stay the same

Generic ImageNet Classification Structure

For ImageNet classification it's common to have 5 levels of down sampling before the global average pool: ~ 1 or 2 in the tail and ~ 4 or 3 in the body



Different Building Blocks

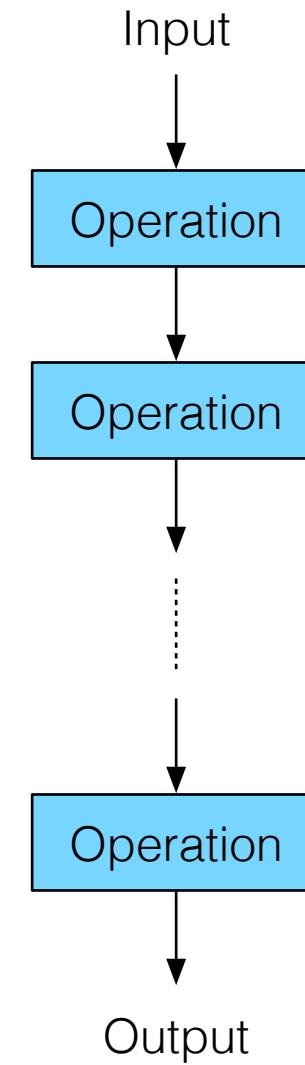
- There are many many different network body designs
- This section will look at some of the more historically important and successful designs grouped together based on their building block structure
 - Serial
 - Parallel
 - Dense
 - Residual
 - Input dependent
- Will also briefly look at different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

CNNs – Body (Encoder Pt 2)

– Serial Building Blocks

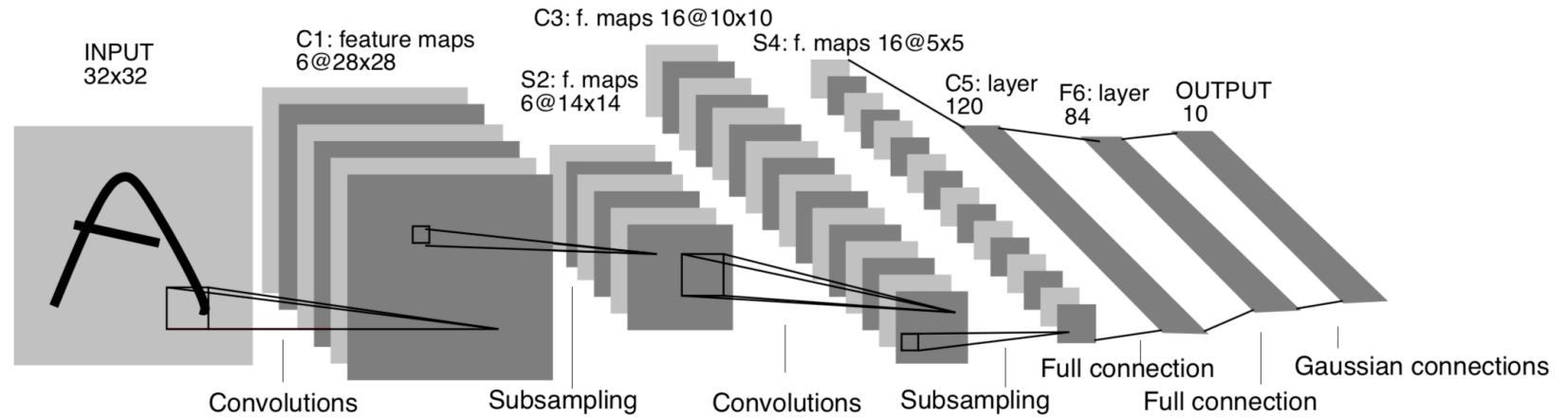
Serial Building Blocks

- A sequential set of operations from input to output
 - The historical starting point and how we've thought about network design so far
- Highlighting a few innovations
 - LeNet
 - Starting point
 - Narrow width shallow network
 - Many different conv sizes
 - AlexNet / CaffeNet
 - Medium width network
 - VGG
 - Medium depth network
 - All 3x3 convs
 - MobileNets V1
 - Depth wise separable conv



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>

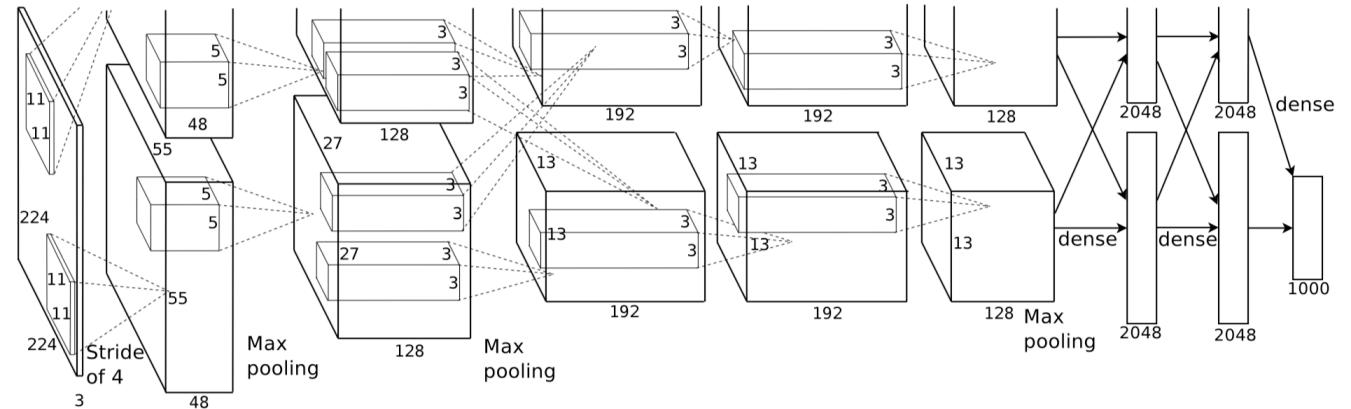


Notes

- 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,
- ...

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



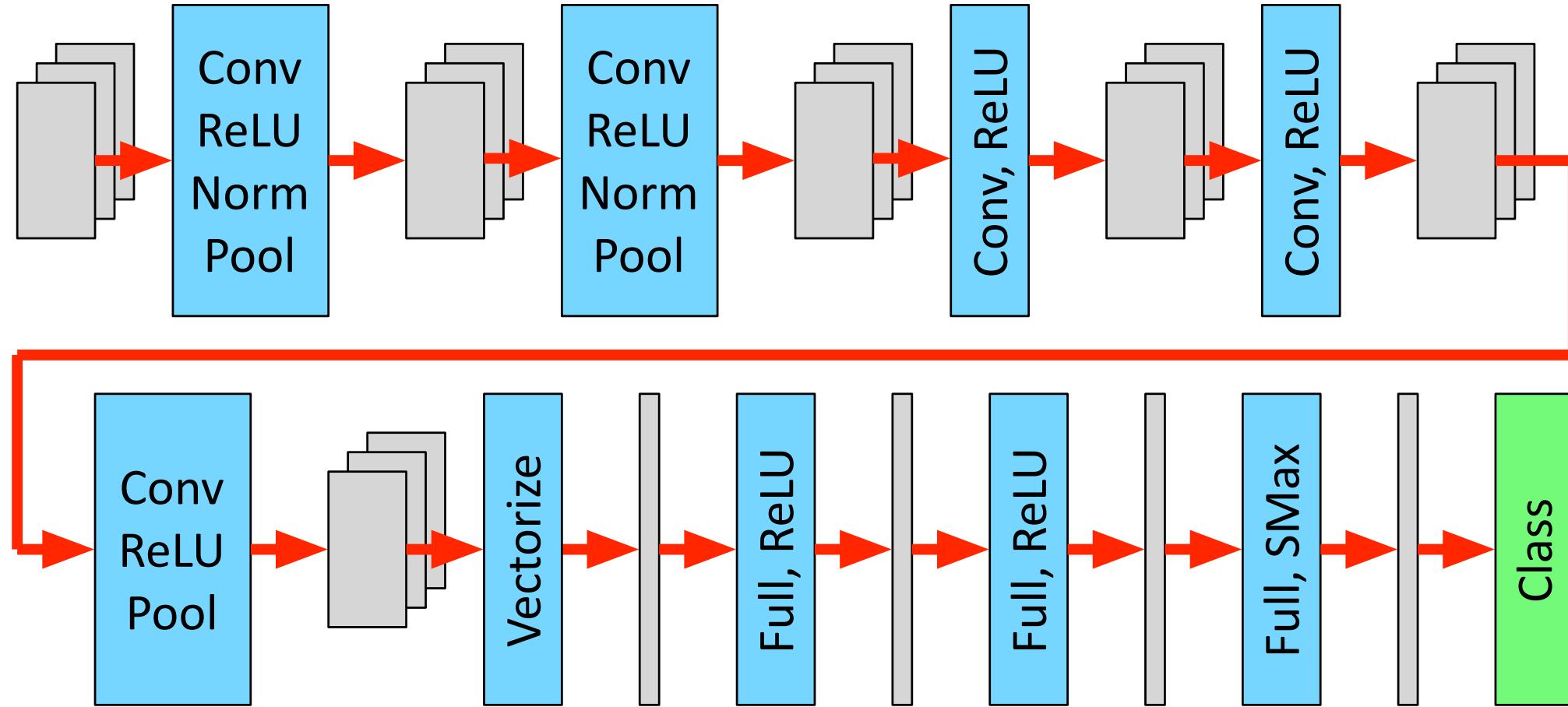
Notes

- 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>

AlexNet / CaffeNet



AlexNet / CaffeNet

#	Type	B	Ni	M	P	F	S	No	MACs	In	Map	Parameter	Out	Map	All	Mem	Op
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
0	Inpt	1	3	224	0	0	0	0	0	0	0	0	0	150528	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				

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

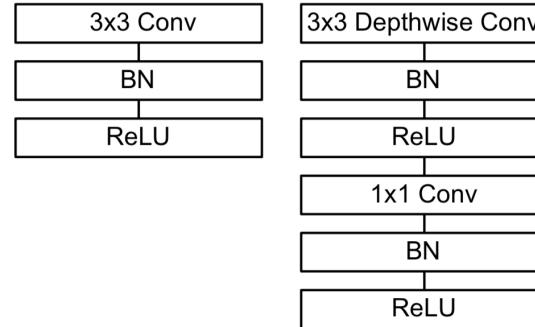
- <https://dgschwend.github.io/netscope/#/preset/vgg-16>

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
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
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
maxpool					
FC-4096	FC-4096	FC-1000	soft-max		

MobileNet V1

Nota
bene

- 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 (lower arithmetic intensity)
 - 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>



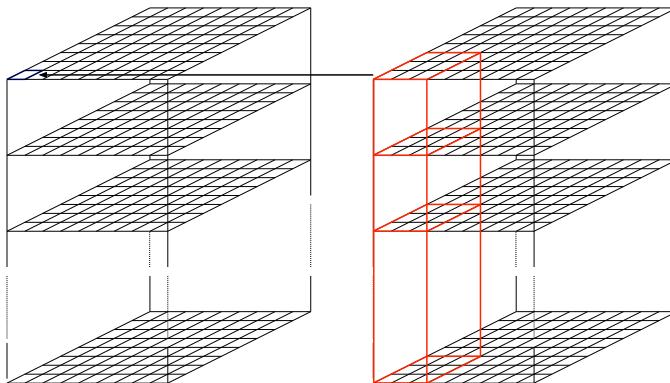
Notes

- 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

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32 \text{ dw}$	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64 \text{ dw}$	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128 \text{ dw}$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128 \text{ dw}$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256 \text{ dw}$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256 \text{ dw}$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512 \text{ dw}$	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512 \text{ dw}$	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024 \text{ dw}$	$7 \times 7 \times 1024$
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	$1 \times 1 \times 1000$

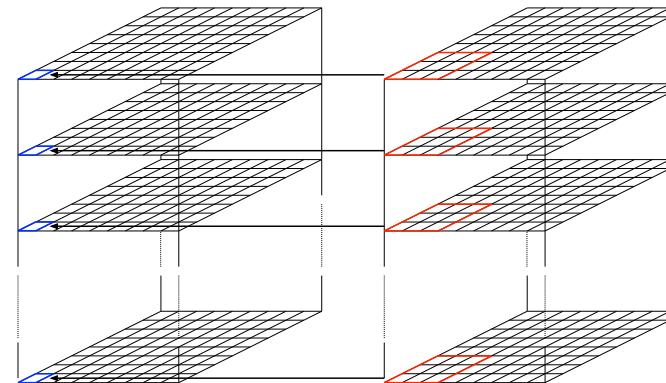
MobileNet V1

CNN style 2D convolution

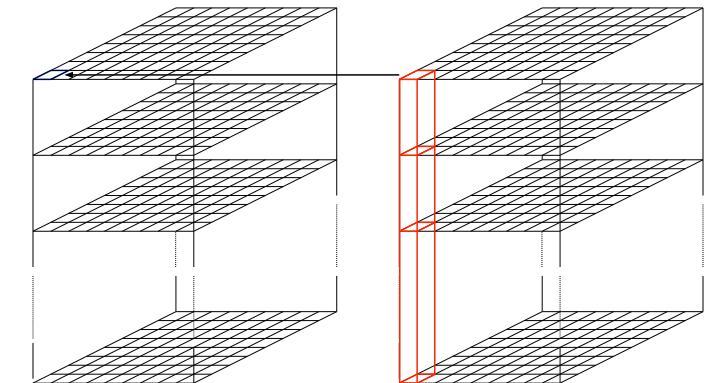


Spatial and channel mixing

Depth wise separable convolution



Spatial mixing

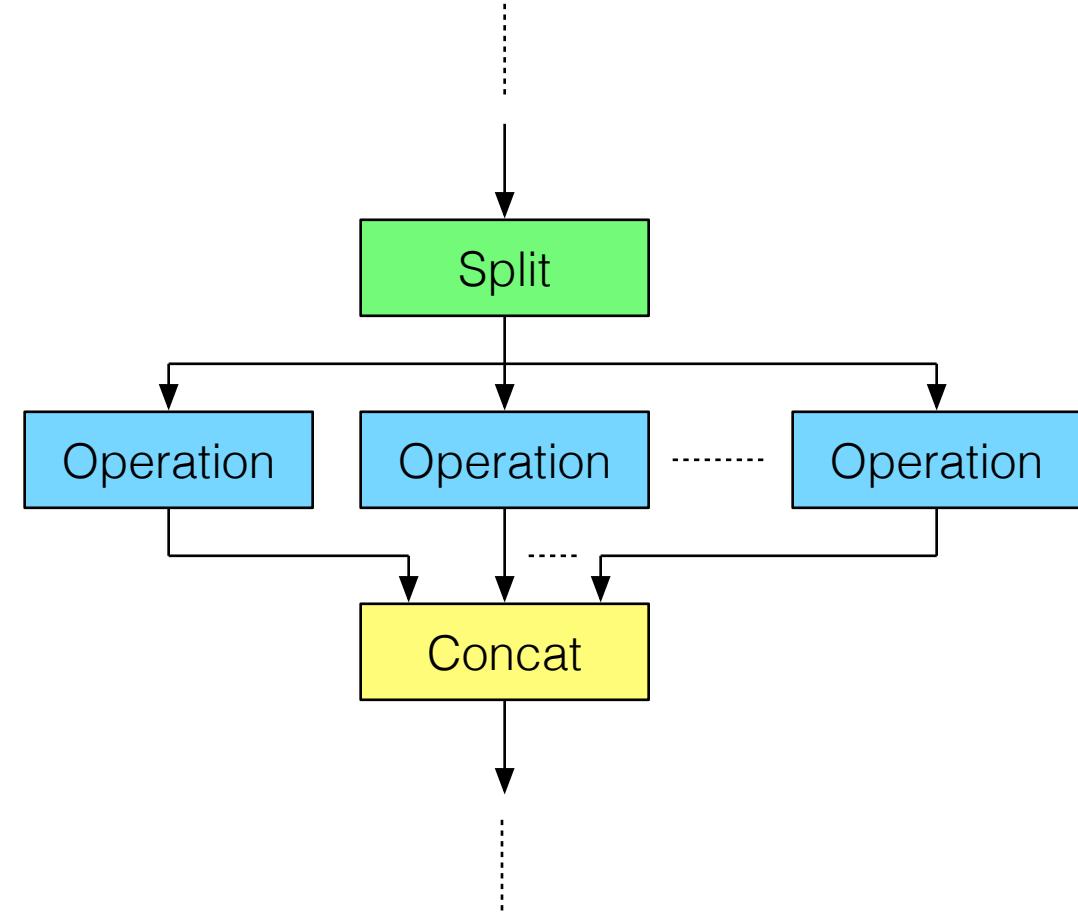


Channel mixing

CNNs – Body (Encoder Pt 2) – Parallel Building Blocks

Parallel Building Blocks

- Per building block
 - Input split
 - Parallel structure
 - Output concat
- Examples
 - GoogLeNet*
 - Inception V2, V3 and V4



*My personal favorite network name

Commentary

- Parallel structures were more relevant / popular when network depths were limited because of challenges with convergence during training
 - This is before residual connections got popular
 - This is before (or right around) when batch norm got popular
- Parallel structures like this feel less relevant now that it's possible to train arbitrarily deep networks and they've somewhat fallen out of favor
- However, they're still important to understand
 - From a motivation perspective
 - From a historical perspective

It's Easy To Get Lost In The Details

- When you do, come back to this slide and remember these general strategies
- Output features
 - Are each created from all input features
 - Some are created from a 1×1 spatial region, some from a 3×3 spatial region, some from a 5×5 spatial region, ...
 - The number of features created from a particular spatial region was determined via optimization
 - Taking into account both performance and accuracy
 - Larger convs are broken up into the serial processing of smaller convs to reduce MACs and params
 - Just like VGG
 - 5×5 : $3 \times 3, 3 \times 3$
 - 7×7 : $3 \times 3, 3 \times 3, 3 \times 3$
 - Larger convs are also broken up into the serial processing of row conv and col conv to reduce MACs and params
 - This isn't very popular in recent network designs, it's now much more common is to process channels and spatial components separately like MobileNetV1
 - 5×5 : $1 \times 5, 5 \times 1$
 - 7×7 : $1 \times 7, 7 \times 1$
 - 1×1 convolutions are sometimes used to reduce the number of channels before larger spatial operations
 - Allows mixing of channels as needed for the larger conv
 - Allows for the reduction of MACs and params

GoogLeNet / Inception V2, V3 And V4

- Design

- Parallel paths with different receptive field sizes for information to flow through a layer
- Different output features are created via different size spatial mixings (1x1, 3x3, 5x5, ...)
- Referred to as inception modules (think network in a network, but not NiN)
- 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

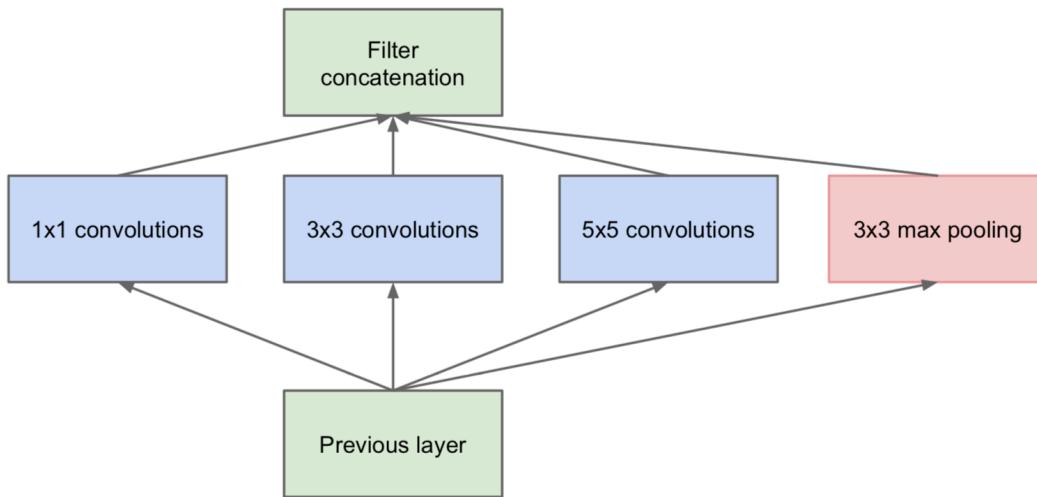
- A simple guide to the versions of the inception network
 - <https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>
- Going deeper with convolutions
 - <https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf>
- Rethinking the inception architecture for computer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

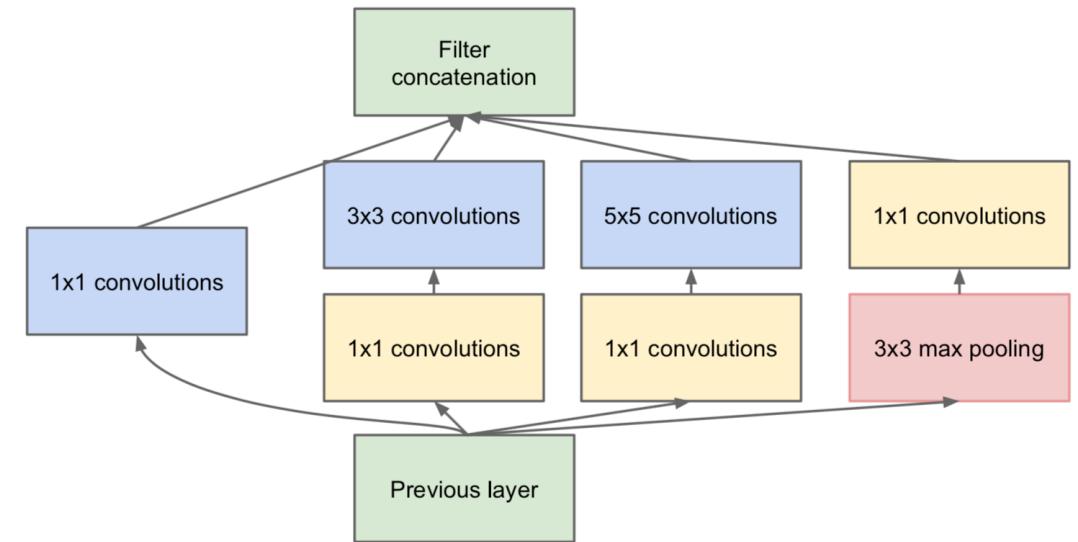
- <https://dgschwend.github.io/netscope/#/preset/googlenet>
- <https://dgschwend.github.io/netscope/#/preset/inceptionv3>
- <https://dgschwend.github.io/netscope/#/preset/inceptionv4>

GoogLeNet

Parallel paths (not used by GoogLeNet)

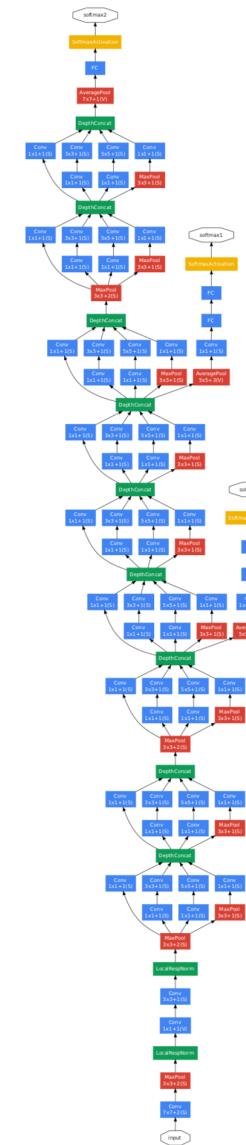


Parallel paths with dimensionality reduction



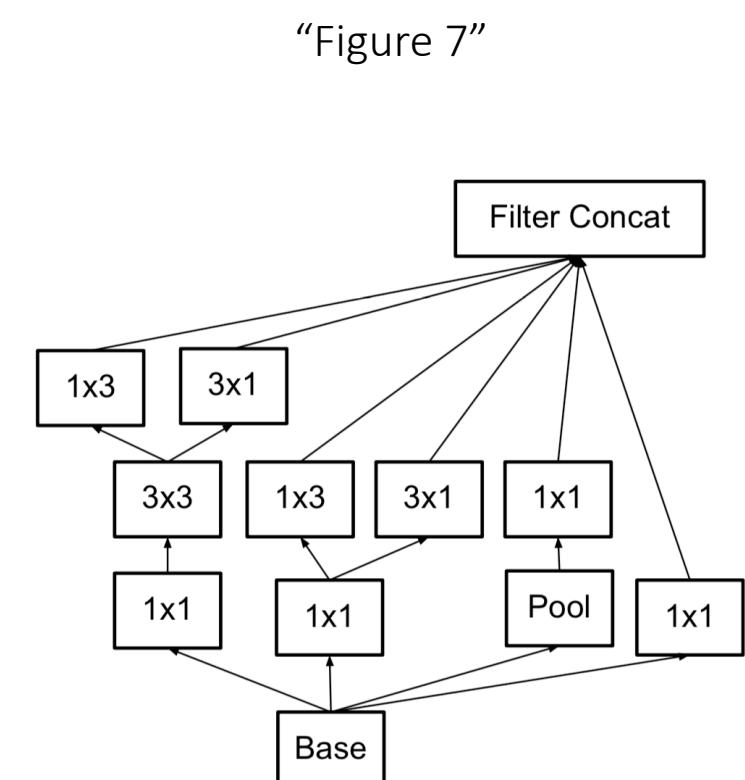
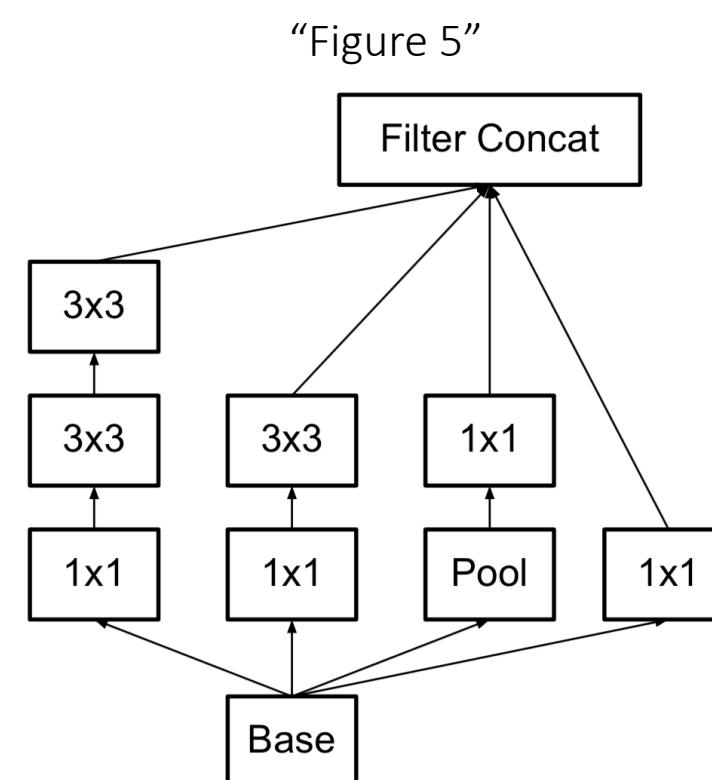
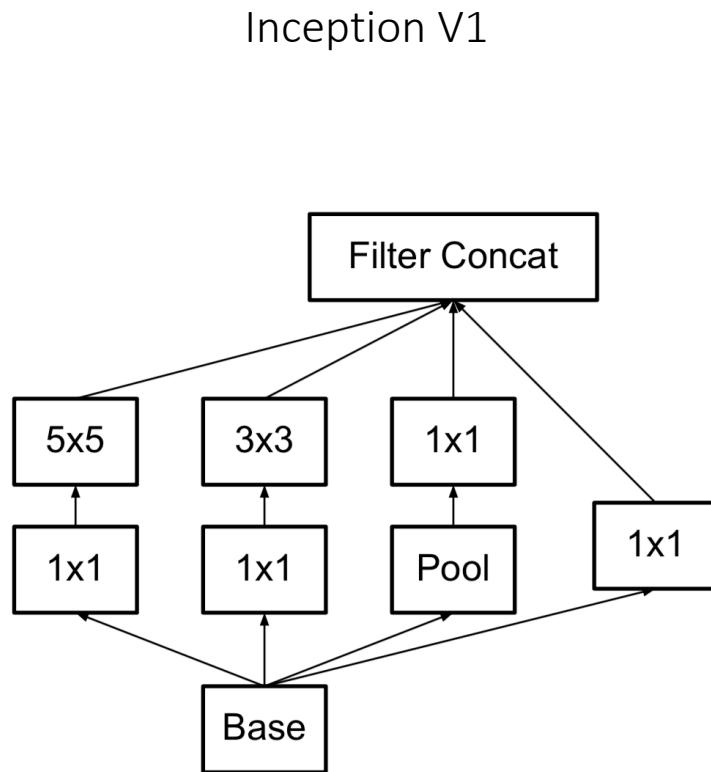
GoogLeNet

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								



Inception V2 And V3

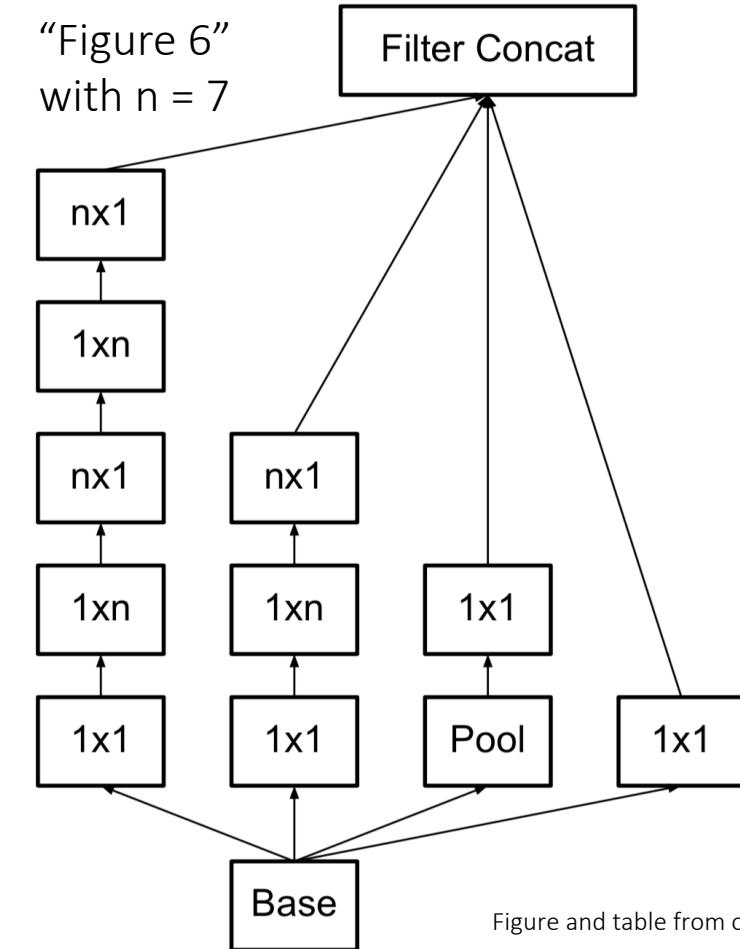
Inception V2 factors 5x5 filters to a stack of two 3x3 filters, replaces some 3x3 with 1x3 and 3x1; Inception V3 adds a module with a 7x7 filter replaced by three 3x3 filters and modifies the training to use a RMSprop solver with label smoothing and aux heads



Inception V2 And V3

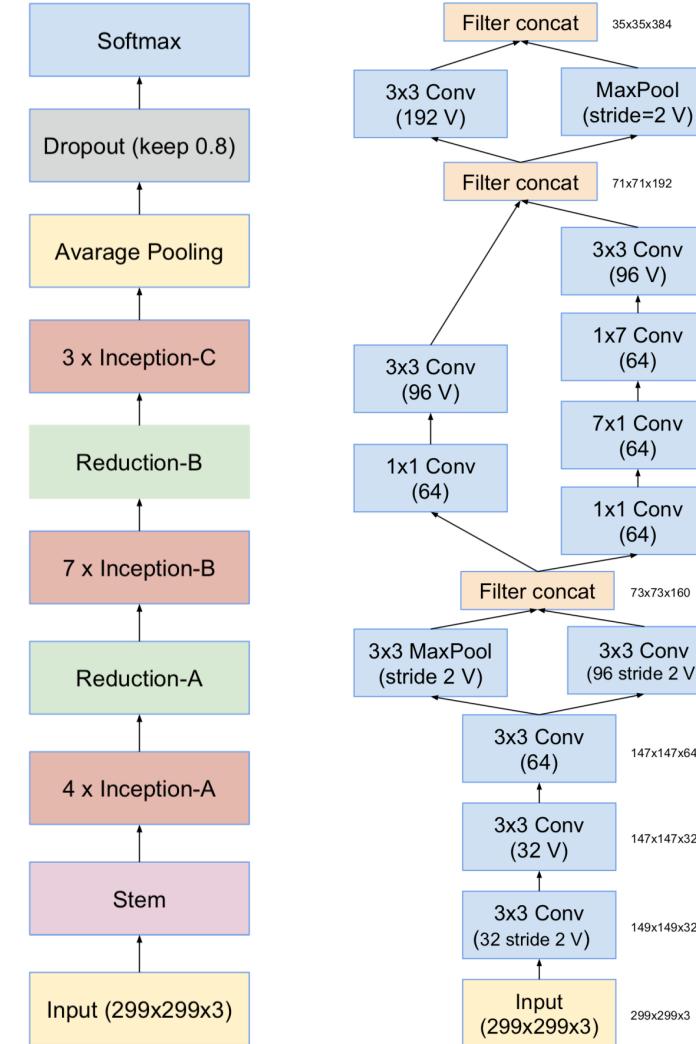
Inception V2 factors 5x5 filters to a stack of two 3x3 filters, replaces some 3x3 with 1x3 and 3x1; Inception V3 adds a module with a 7x7 filter replaced by three 3x3 filters and modifies the training to use a RMSprop solver with label smoothing and aux heads

type	patch size/stride or remarks	input size
conv	$3 \times 3 / 2$	$299 \times 299 \times 3$
conv	$3 \times 3 / 1$	$149 \times 149 \times 32$
conv padded	$3 \times 3 / 1$	$147 \times 147 \times 32$
pool	$3 \times 3 / 2$	$147 \times 147 \times 64$
conv	$3 \times 3 / 1$	$73 \times 73 \times 64$
conv	$3 \times 3 / 2$	$71 \times 71 \times 80$
conv	$3 \times 3 / 1$	$35 \times 35 \times 192$
3×Inception	As in figure 5	$35 \times 35 \times 288$
5×Inception	As in figure 6	$17 \times 17 \times 768$
2×Inception	As in figure 7	$8 \times 8 \times 1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$



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

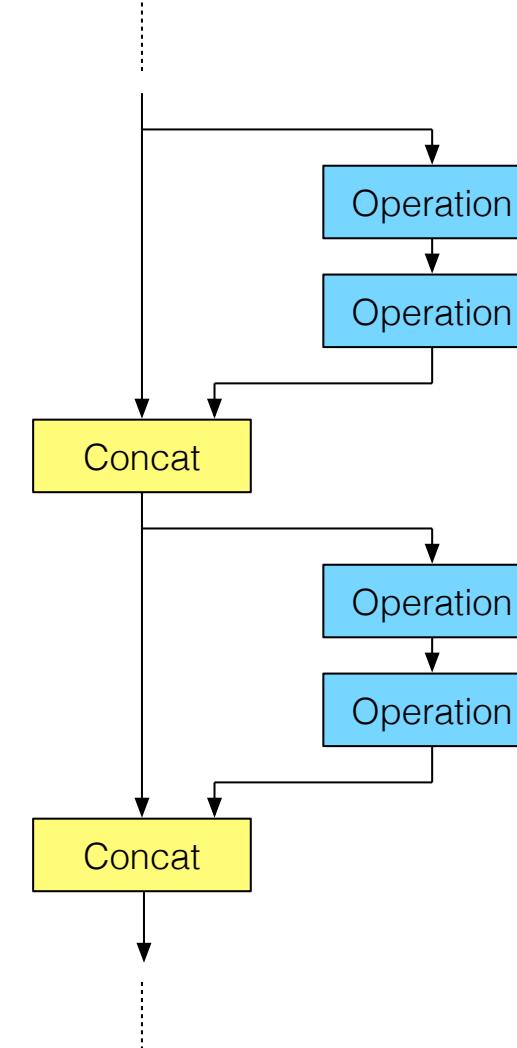


CNNs – Body (Encoder Pt 2)

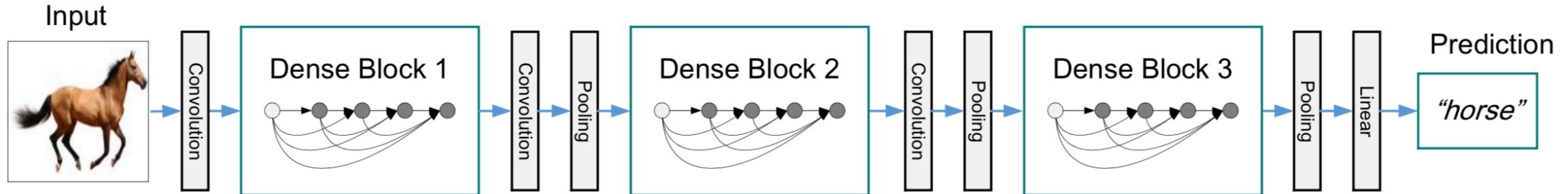
- Dense Building Blocks

Dense Building Blocks

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



DenseNet

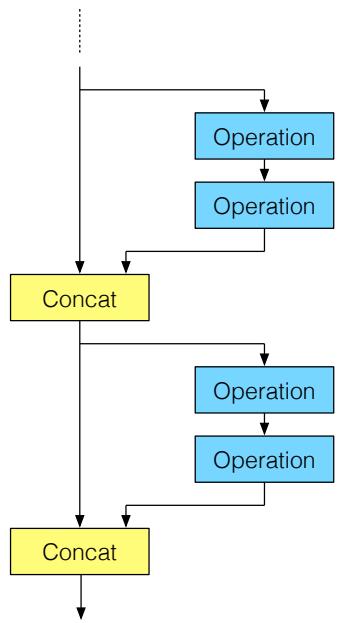


- Feed forward style dense connection
- Dense blocks use CNN style 1×1 convolution to reduce channels followed by CNN style 3×3 convolution to mix across channels and space
- Number of channels added per the above operation is referred to as the growth rate of the network (networks can get wide fast)
- 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

$$x_d = h([x_{d-1}, x_{d-2}, \dots, x_0])$$
- 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>



DenseNet

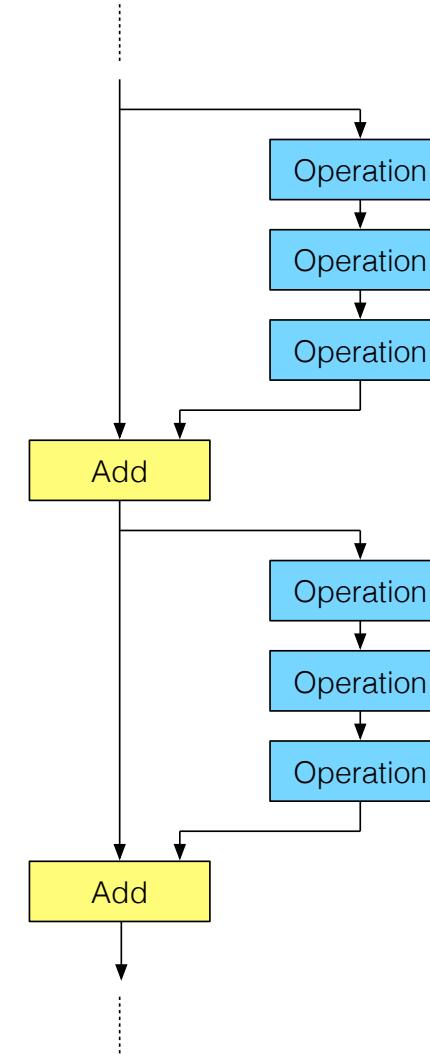
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112			7×7 conv, stride 2	
Pooling	56×56			3×3 max pool, stride 2	
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
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	

CNNs – Body (Encoder Pt 2)

– Residual Building Blocks

Residual Building Blocks

- Per building block
 - Input split into 2 paths
 - Direct (identity) 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
 - ...



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

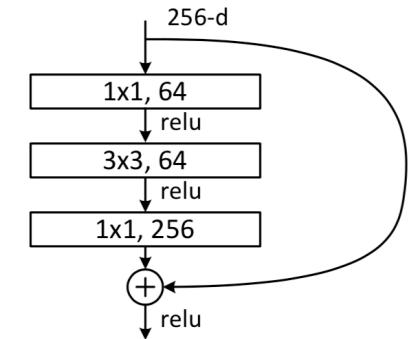
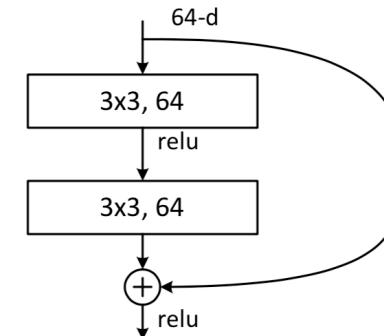
- Residual block

- Nota bene →
- Standard layer: $x_{d+1} = f_d(x_d)$
 - Residual block: $x_{d+1} = f_d(x_d) + h_d(x_d)$
 - 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>



Notes

- Increases depth very nicely
- Need to handle increases to the number of channels specially (3 methods presented, small modifications by others later)
- Some technicalities for keeping the residual path as identity like as possible (see paper 2, though not as popular now)
- 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>

ResNet

- The error gradient back propagates in identity + perturbation form too

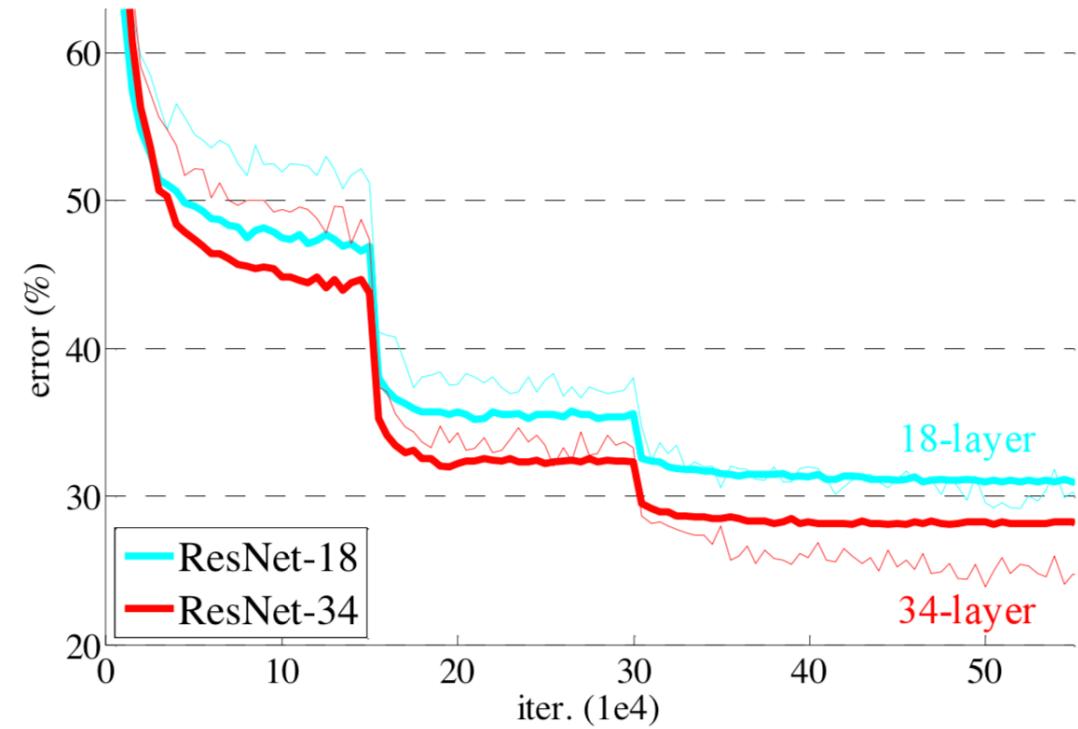
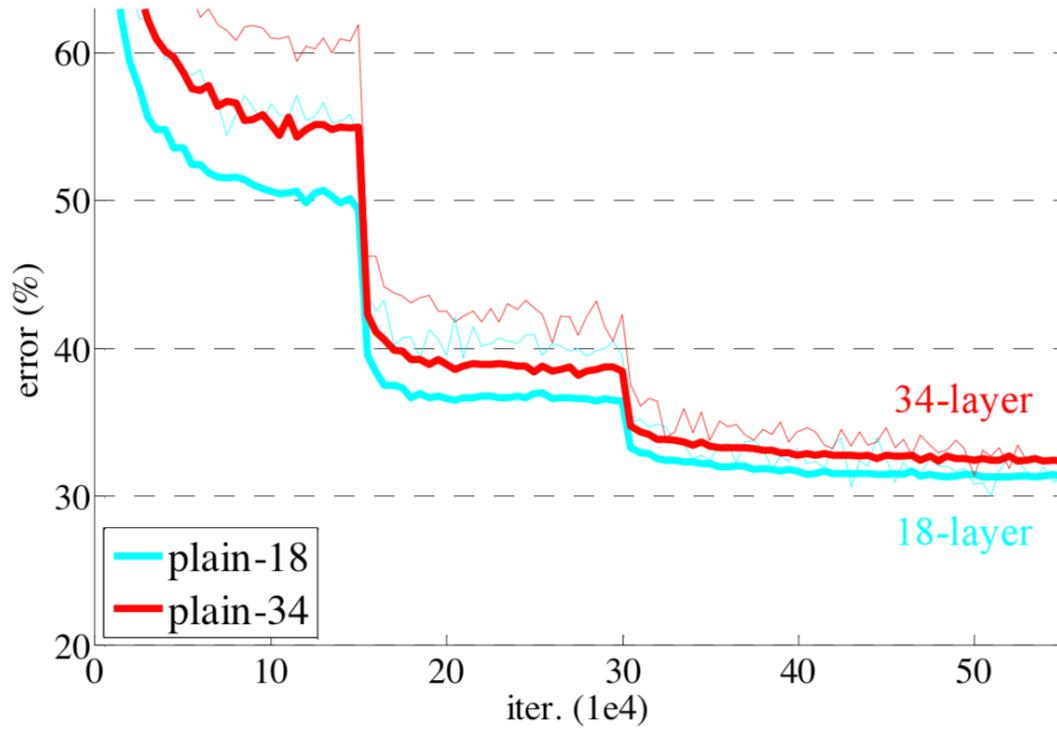
- Given: $\frac{\partial e}{\partial \mathbf{x}_{d+1}}$

- Find: $\frac{\partial e}{\partial \mathbf{x}_d}$

- Solution:

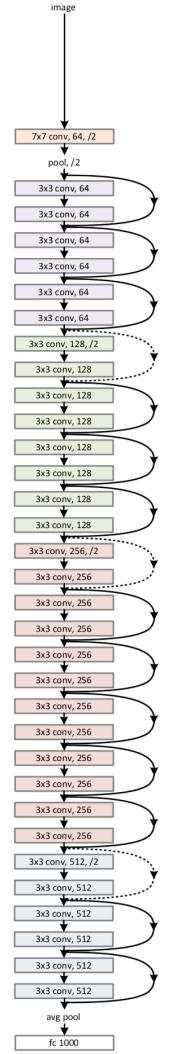
$$\begin{aligned}
 \frac{\partial e}{\partial \mathbf{x}_d} &= (\frac{\partial \mathbf{x}_{d+1}}{\partial \mathbf{x}_d}) (\frac{\partial e}{\partial \mathbf{x}_{d+1}}) && \text{Chain rule} \\
 &= (\frac{\partial f_d}{\partial \mathbf{x}_d}) (\frac{\partial e}{\partial \mathbf{x}_{d+1}}) && \text{Sub } f_d \text{ for } \mathbf{x}_{d+1} \\
 &= (\frac{\partial \mathbf{x}_d}{\partial \mathbf{x}_d} + \frac{\partial h_d}{\partial \mathbf{x}_d}) (\frac{\partial e}{\partial \mathbf{x}_{d+1}}) && \text{Sub } f_d = \mathbf{x}_d + h_d(\mathbf{x}_d) \\
 &= (1 + \frac{\partial h_d}{\partial \mathbf{x}_d}) (\frac{\partial e}{\partial \mathbf{x}_{d+1}}) && \text{Note } \frac{\partial \mathbf{x}_d}{\partial \mathbf{x}_d} = 1 \\
 &= \frac{\partial e}{\partial \mathbf{x}_{d+1}} + (\frac{\partial h_d}{\partial \mathbf{x}_d}) (\frac{\partial e}{\partial \mathbf{x}_{d+1}}) && \text{Linear operator}
 \end{aligned}$$

ResNet



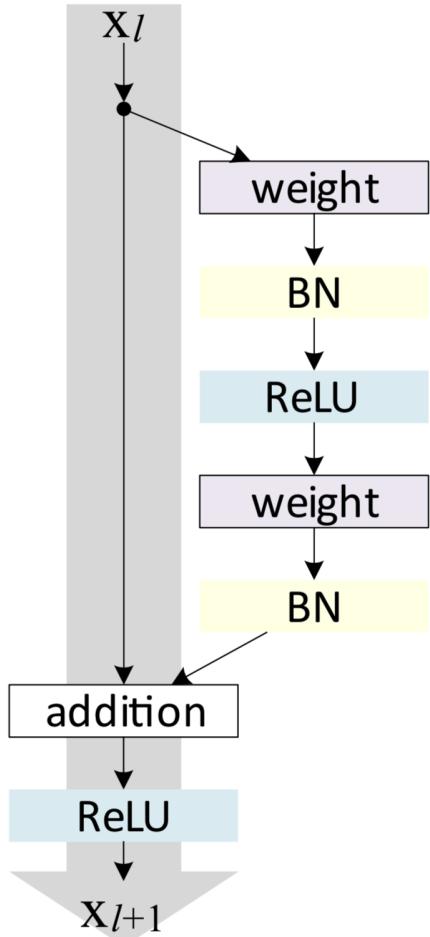
ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	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

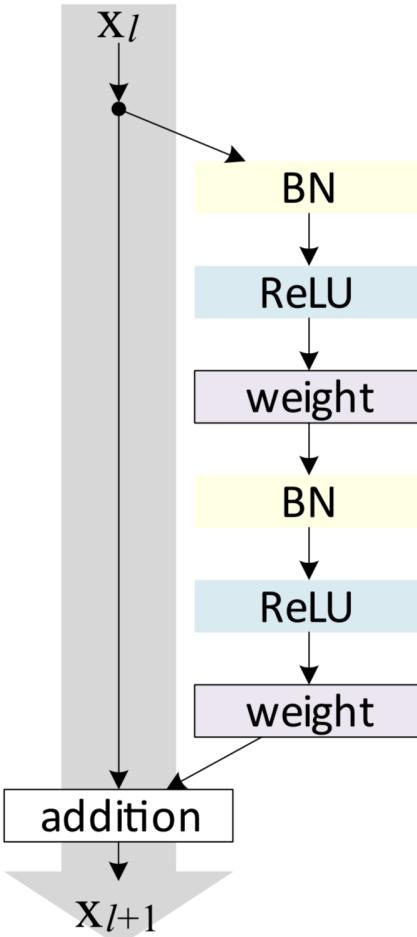


ResNet

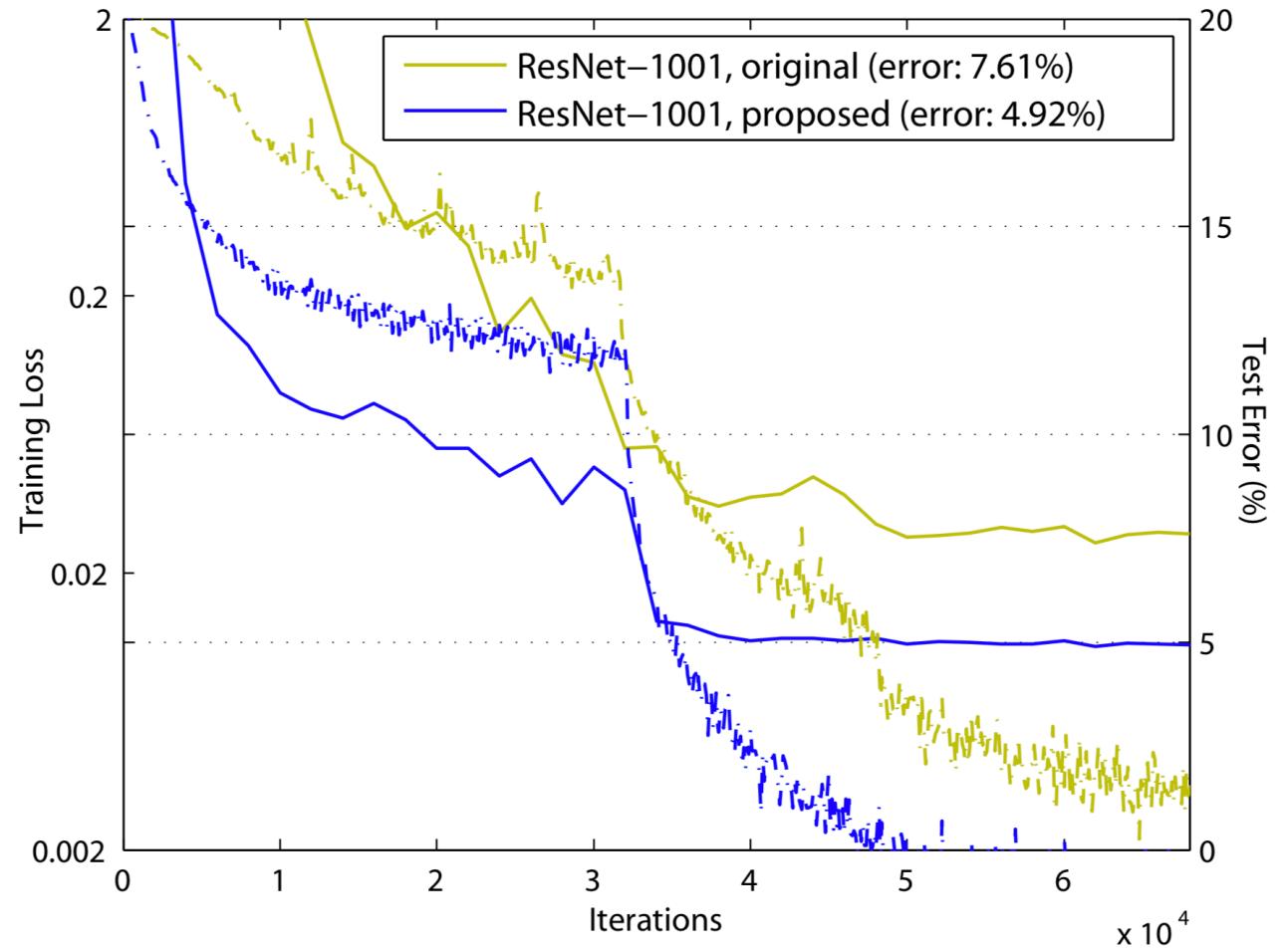
Original



Paper 2



CIFAR 10 Accuracy

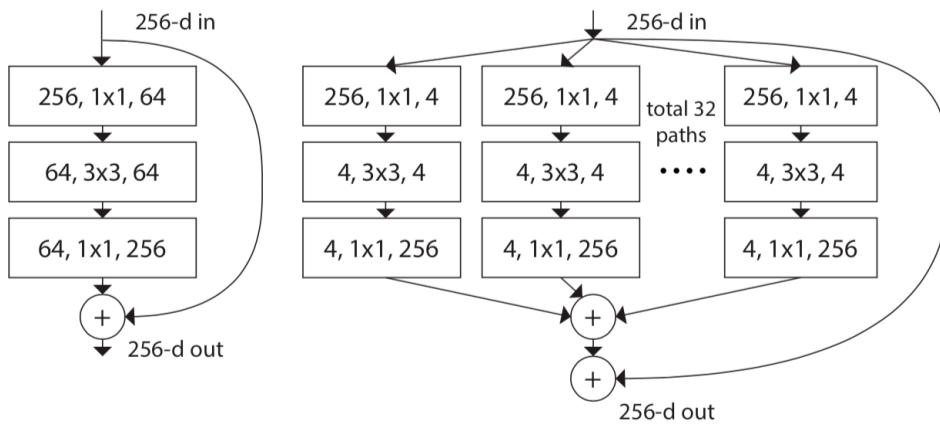


Residual + Separable Filters

- 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
 - RegNetX/Y

ResNeXt

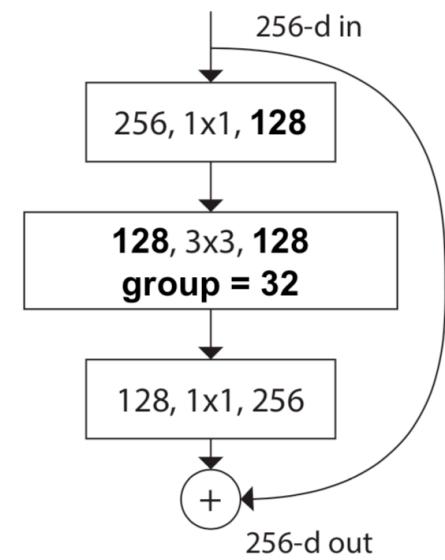
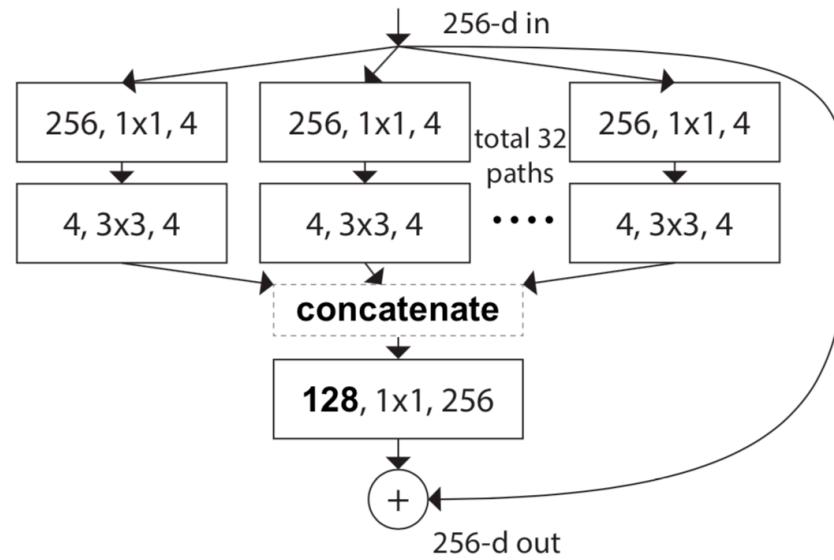
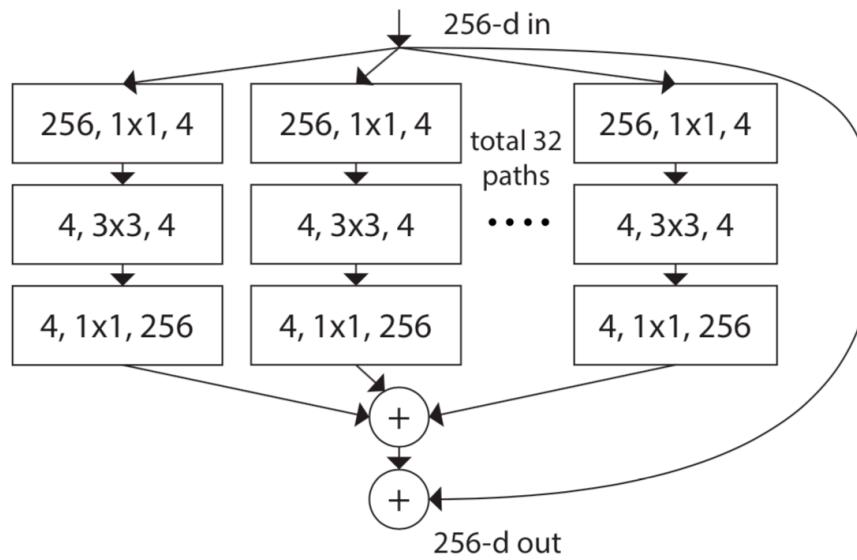
- Design
 - Replace 3x3 convolutions in the residual bottleneck with highly grouped 3x3 convolutions
- Aggregated residual transformations for deep neural networks
 - <https://arxiv.org/abs/1611.05431>



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

ResNeXt

equivalent



A Few Additional Papers

- ResNeSt: split-attention networks
 - <https://arxiv.org/abs/2004.08955>
- ReXNet: diminishing representational bottleneck on convolutional neural network
 - <https://arxiv.org/abs/2007.00992>
 - <https://github.com/clovaai/rexnet>
- Rethinking bottleneck structure for efficient mobile network design
 - <https://arxiv.org/abs/2007.02269>
- ResNet or DenseNet? introducing dense shortcuts to ResNet
 - <https://arxiv.org/abs/2010.12496>

CNNs – Body (Encoder Pt 2) – Input Dependent Building Blocks

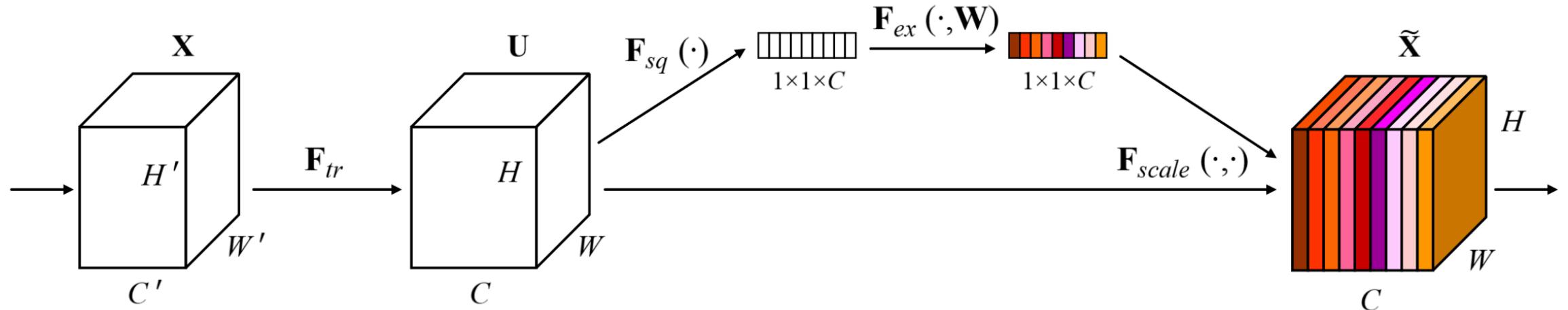
Input Dependent Building Blocks

- All of the building blocks seen so far have 1 thing in common: they're independent of the input (i.e., the function that the network implements is not dependent on the input)
- We've seen other network building blocks that are input dependent and have positive properties (e.g., the self attention operator from the linear algebra slides)
- We can add input dependent building blocks to previous CNN building blocks to achieve higher levels of accuracy at a given level of complexity
 - Some variants use feature maps to create weights to re weight feature maps
 - Some variants use feature maps to create weights for the CNN style 2D conv operation
 - Some variants use feature maps to modify the nonlinearity
 - Some variants use information from the whole feature map to implicitly increase the receptive field size earlier on in the network

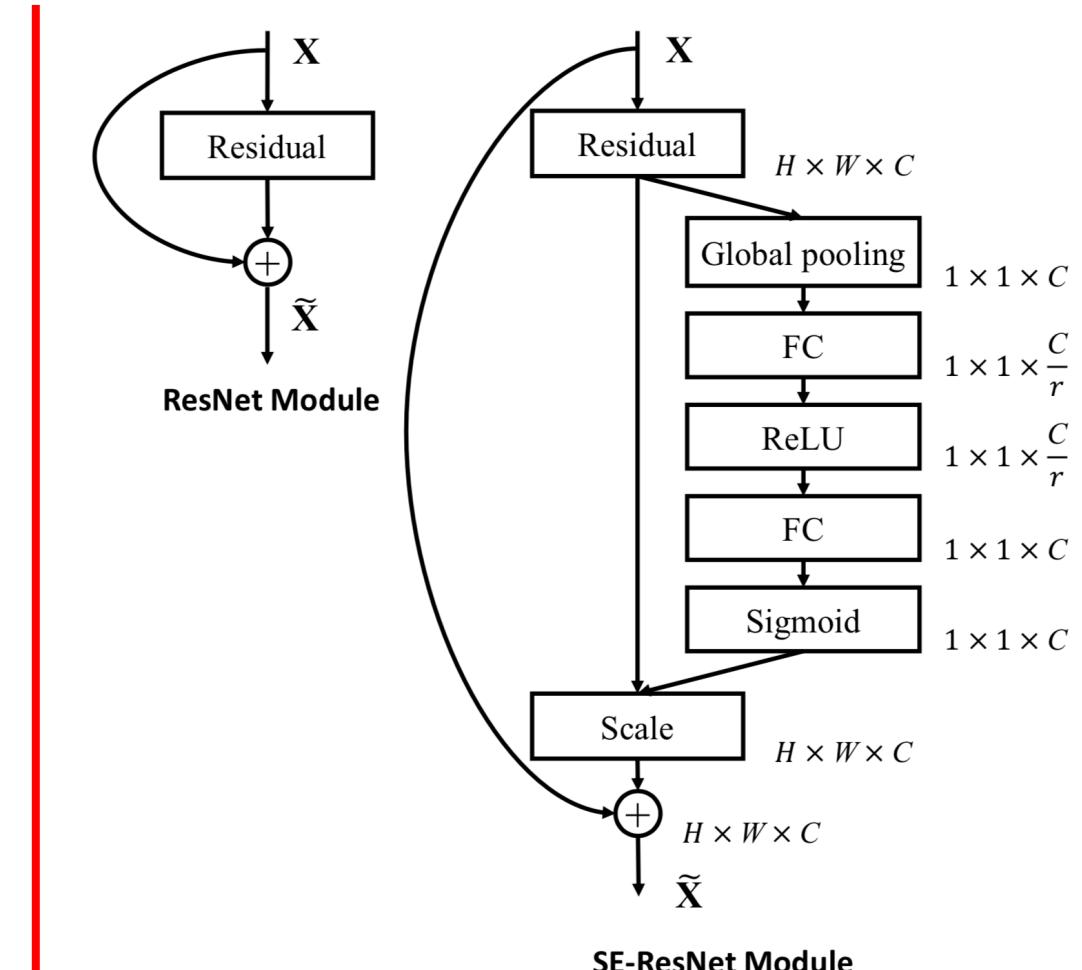
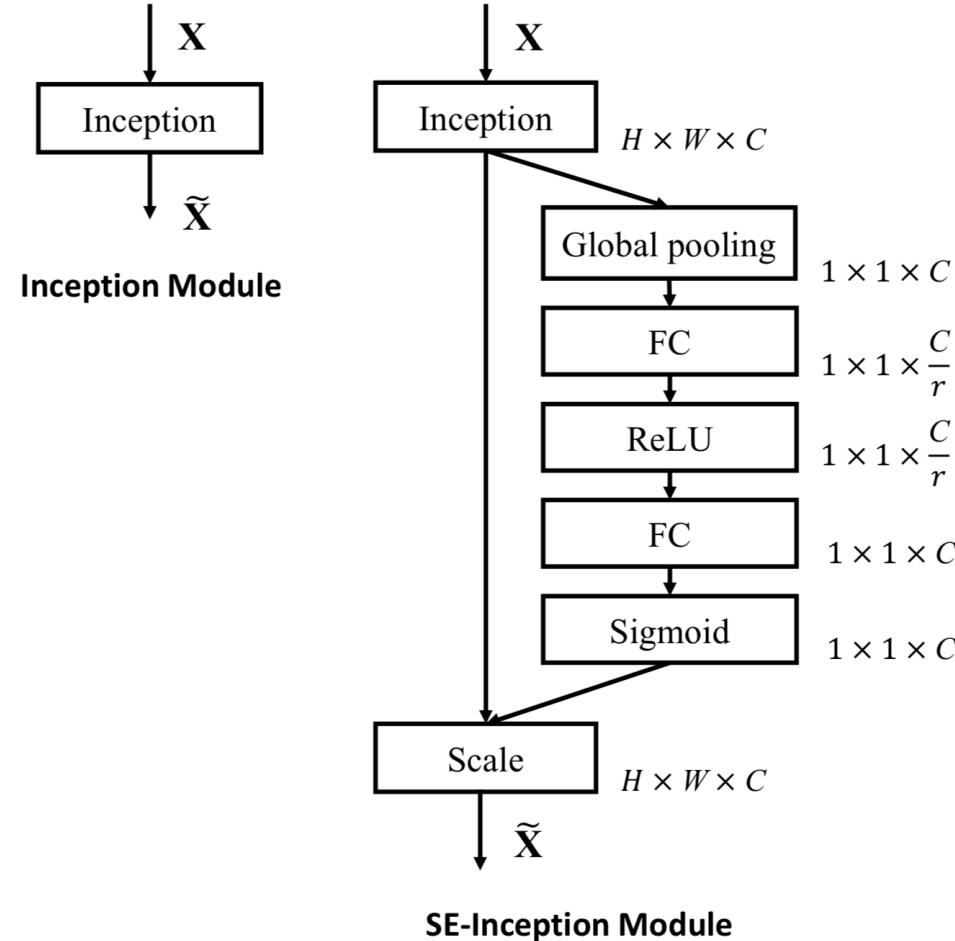
Squeeze And Excitation Networks

- 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
 - Feature re calibration has some similarities with attention
 - 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
- Squeeze-and-excitation networks
 - <https://arxiv.org/abs/1709.01507>

Squeeze And Excitation Networks



Squeeze And Excitation Networks



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	
	$\begin{bmatrix} \text{conv, } 1 \times 1, 64 \\ \text{conv, } 3 \times 3, 64 \\ \text{conv, } 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 64 \\ \text{conv, } 3 \times 3, 64 \\ \text{conv, } 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 128 \\ \text{conv, } 3 \times 3, 128 \\ \text{conv, } 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28×28	$\begin{bmatrix} \text{conv, } 1 \times 1, 128 \\ \text{conv, } 3 \times 3, 128 \\ \text{conv, } 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv, } 1 \times 1, 128 \\ \text{conv, } 3 \times 3, 128 \\ \text{conv, } 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv, } 1 \times 1, 256 \\ \text{conv, } 3 \times 3, 256 \\ \text{conv, } 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14×14	$\begin{bmatrix} \text{conv, } 1 \times 1, 256 \\ \text{conv, } 3 \times 3, 256 \\ \text{conv, } 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv, } 1 \times 1, 256 \\ \text{conv, } 3 \times 3, 256 \\ \text{conv, } 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv, } 1 \times 1, 512 \\ \text{conv, } 3 \times 3, 512 \\ \text{conv, } 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv, } 1 \times 1, 512 \\ \text{conv, } 3 \times 3, 512 \\ \text{conv, } 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 512 \\ \text{conv, } 3 \times 3, 512 \\ \text{conv, } 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 1024 \\ \text{conv, } 3 \times 3, 1024 \\ \text{conv, } 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1×1	global average pool, 1000-d fc , softmax		

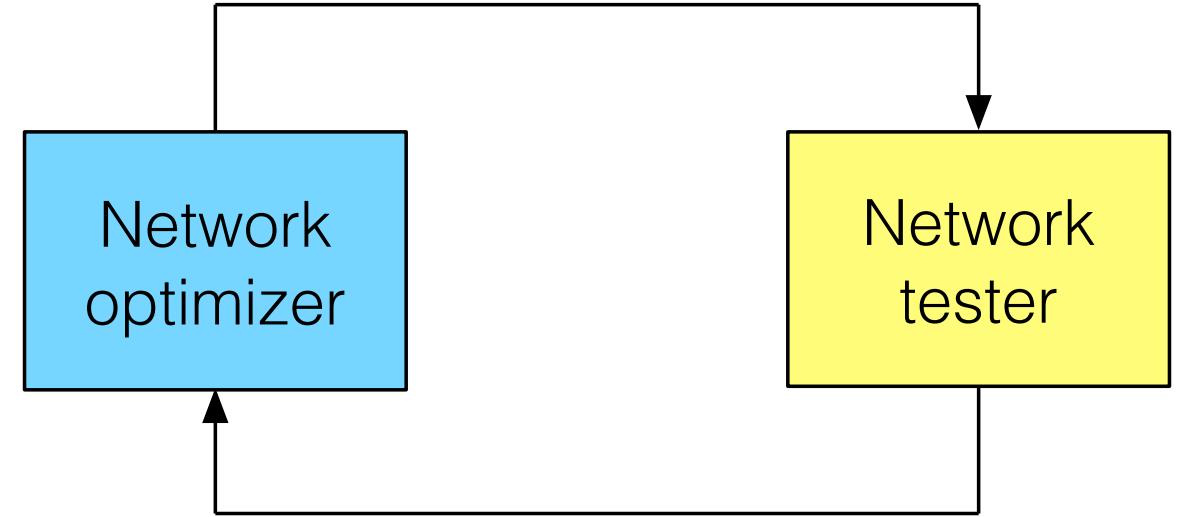
Additional Examples

- Squeeze-and-excitation networks
 - <https://arxiv.org/abs/1709.01507>
- Competitive inner-imaging squeeze and excitation for residual network
 - <https://arxiv.org/abs/1807.08920>
- Accuracy booster: performance boosting using feature map re-calibration
 - <https://arxiv.org/abs/1903.04407>
- CondConv: conditionally parameterized convolutions for efficient inference
 - <https://arxiv.org/abs/1904.04971>
- Conditional convolutions for instance segmentation
 - <https://arxiv.org/abs/2003.05664>
- Dynamic region-aware convolution
 - <https://arxiv.org/abs/2003.12243>
- DyNet: dynamic convolution for accelerating convolutional neural networks
 - <https://arxiv.org/abs/2004.10694>
- WeightNet: revisiting the design space of weight networks
 - <https://arxiv.org/abs/2007.11823>

CNNs – Body (Encoder Pt 2) – Architecture Search

Optimized Architecture Search

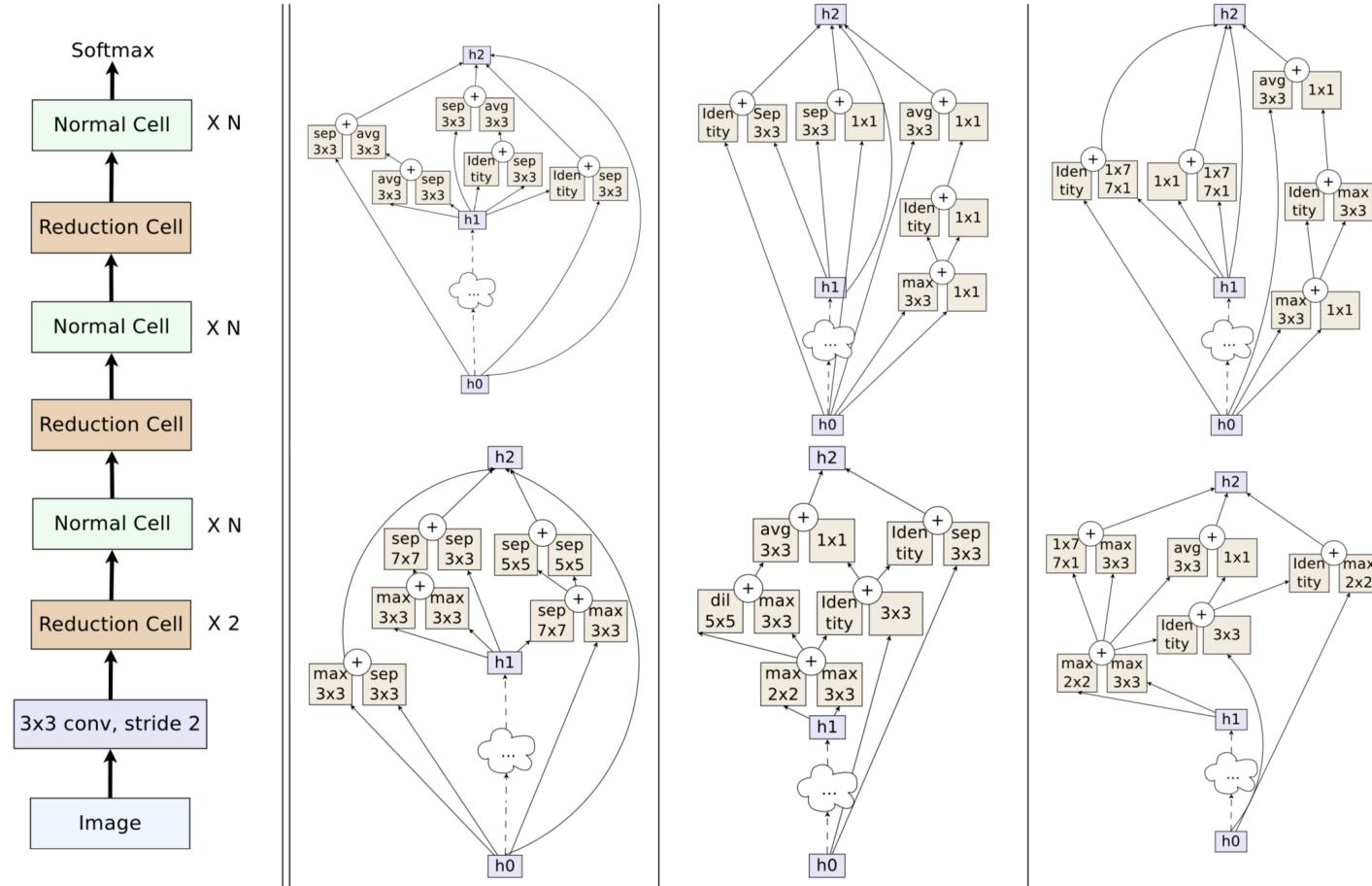
- Using ML to find new architectures
 - Interesting from an exploration perspective
 - Likely a mix of human curated building blocks and modifications with machine determined combinations and sizes
- Also known as neural architecture search
 - It feels like once a basic collection of network design techniques becomes known, most of the action shifts to neural architecture search with various criteria (accuracy, performance, ...)
- 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)



Components Of The Network Optimizer

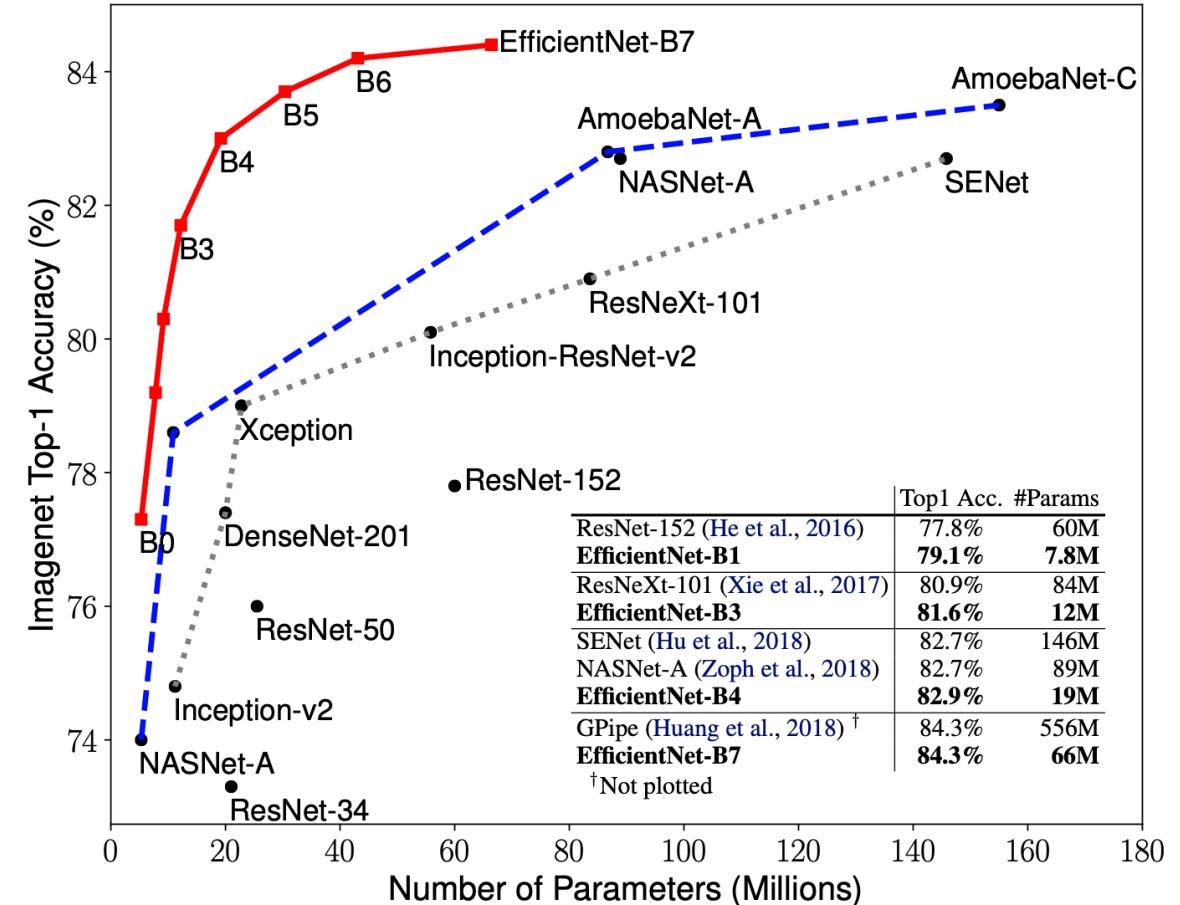
- Search algorithm
 - Ideally an optimization algorithm that works in a sequential (improvement) batch (of multiple models tested at a time) manner
 - Reinforcement learning variants are common
 - Search space
 - Early papers tended to have a very large search space
 - Recent trend towards network design spaces
 - Create a good parameterized building block
 - Create a distribution of models based on varying parameters within the building block
 - Find rules of thumb for parameter combinations that enable high accuracy models at a variety of complexity levels
- On network design spaces for visual recognition
 - <https://arxiv.org/abs/1905.13214>
 - Designing network design spaces
 - <https://arxiv.org/abs/2003.13678>
 - <https://github.com/facebookresearch/pycls>
 - EfficientNet: rethinking model scaling for convolutional neural networks
 - <https://arxiv.org/abs/1905.11946>
 - <https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet>

Example Large Search Space Building Blocks

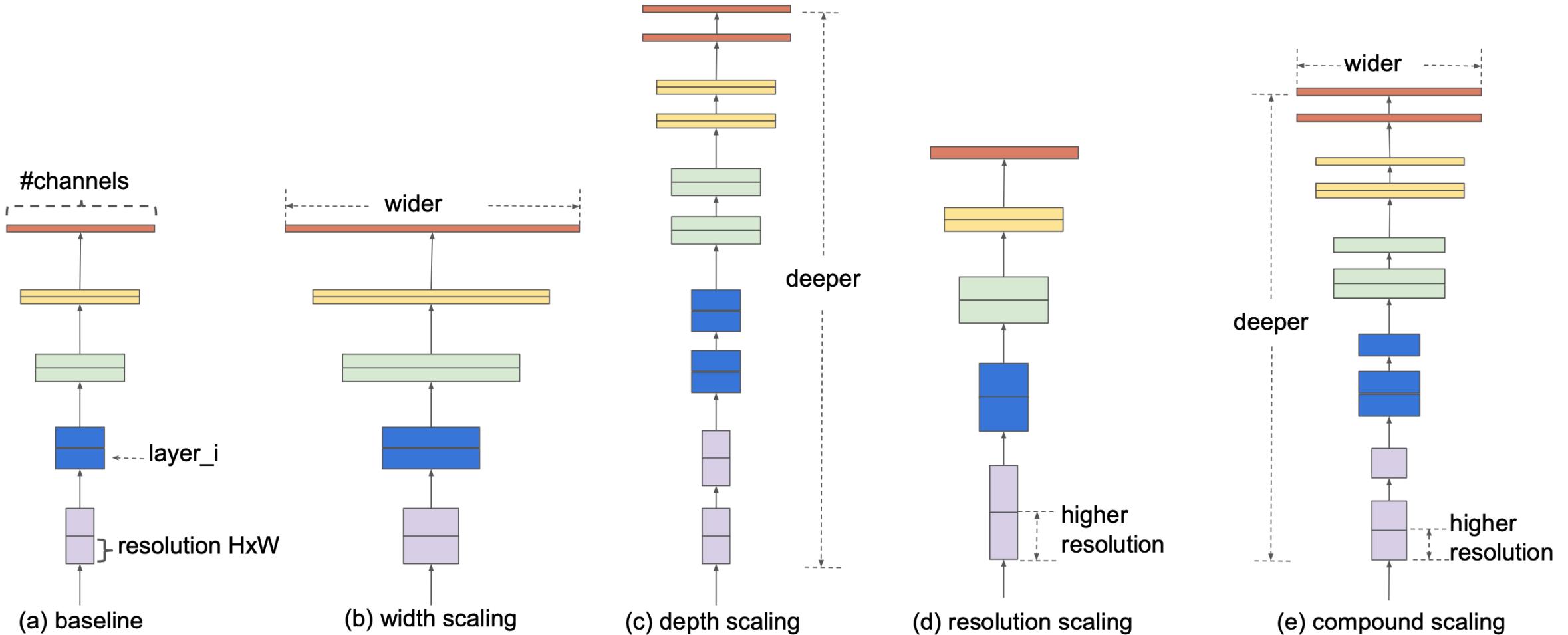


EfficientNet Design Strategy

- Uses neural architecture search within a relatively large search space to create a baseline network
- Then uses a network design space strategy to scale the depth, width and resolution to create a family of networks
- Also does an excellent job with training
 - More on that in the training section
- EfficientNet: rethinking model scaling for convolutional neural networks
 - <https://arxiv.org/abs/1905.11946>



EfficientNet Uses Compound Scaling



EfficientNet–B0 Baseline Structure

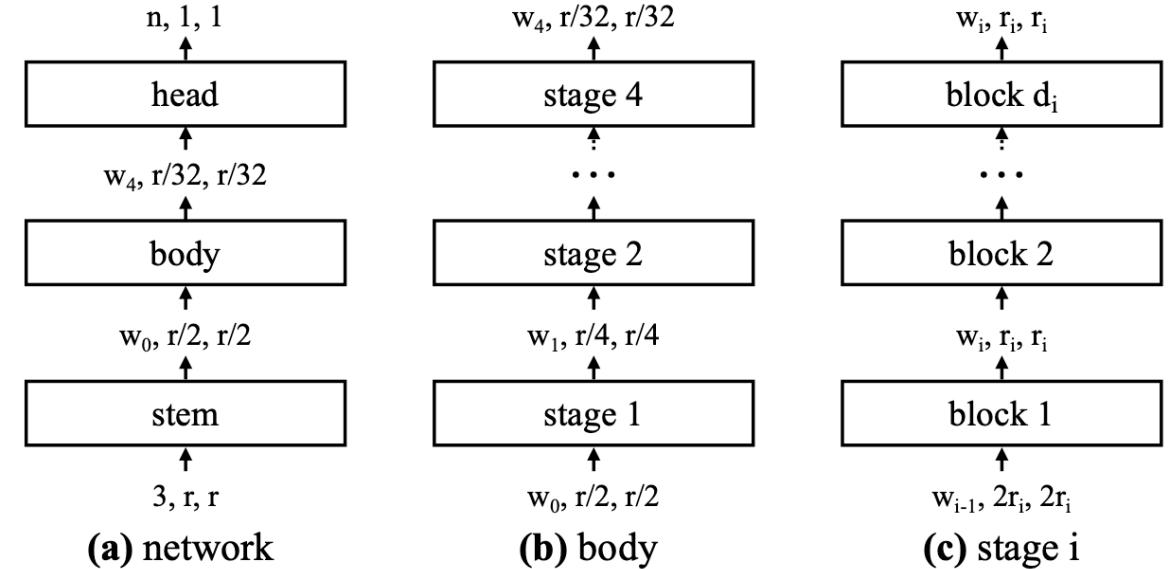
Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels \hat{C}_i	#Layers \hat{L}_i
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

EfficientNet Illustrates That Training Matters

	B0	B1	B2	B3	B4	B5	B6	B7	B8	L2-475	L2
Baseline preprocessing	76.7% (ckpt)	78.7% (ckpt)	79.8% (ckpt)	81.1% (ckpt)	82.5% (ckpt)	83.1% (ckpt)					
AutoAugment (AA)	77.1% (ckpt)	79.1% (ckpt)	80.1% (ckpt)	81.6% (ckpt)	82.9% (ckpt)	83.6% (ckpt)	84.0% (ckpt)	84.3% (ckpt)			
RandAugment (RA)						83.7% (ckpt)		84.7% (ckpt)			
AdvProp + AA	77.6% (ckpt)	79.6% (ckpt)	80.5% (ckpt)	81.9% (ckpt)	83.3% (ckpt)	84.3% (ckpt)	84.8% (ckpt)	85.2% (ckpt)	85.5% (ckpt)		
NoisyStudent + RA	78.8% (ckpt)	81.5% (ckpt)	82.4% (ckpt)	84.1% (ckpt)	85.3% (ckpt)	86.1% (ckpt)	86.4% (ckpt)	86.9% (ckpt)	-	88.2% (ckpt)	88.4% (ckpt)

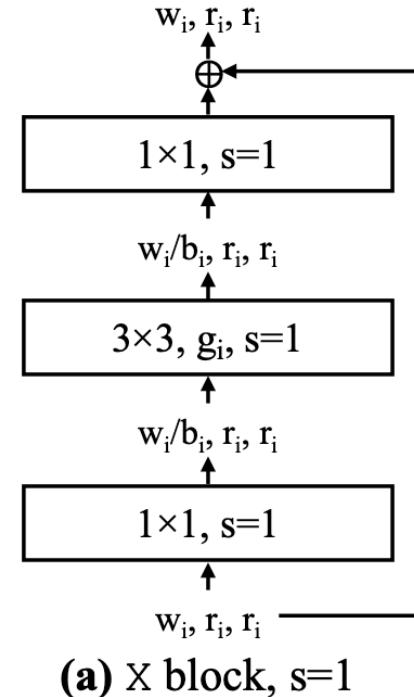
RegNetX/Y Design Strategy

- Simple stem
 - 3x3 conv from 3 input to 32 output channels
- Body is composed of 1-3-1 residual blocks with grouping
 - 1st block for a level has down sampling
 - Parameterized width ($\sim 2.5x$ increase per level)
 - Parameterized group size in the 3x3
 - RegNetY includes SE blocks, RegNetX does not
 - Gives about a 0.5% – 1.5% accuracy boost
- Simple head
 - Global avg pool
 - Fully connected layer

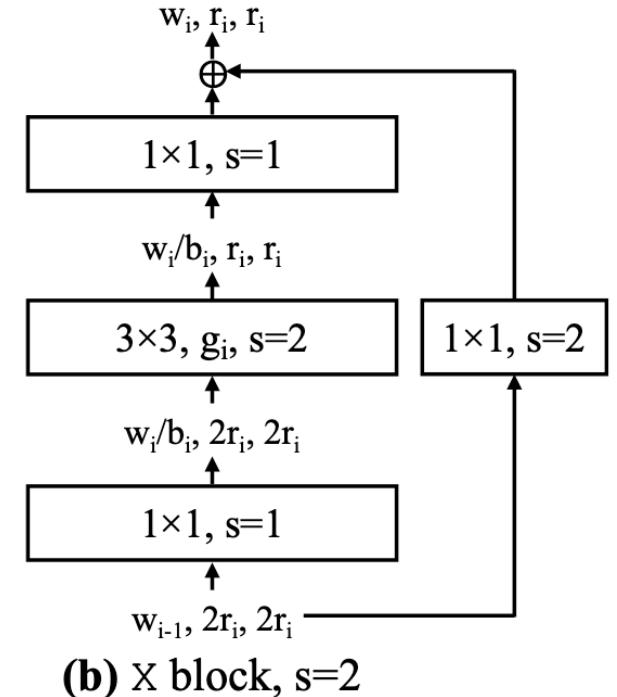


RegNetX/Y Design Strategy

- Simple stem
 - 3x3 conv from 3 input to 32 output channels
- Body is composed of 1-3-1 residual blocks with grouping
 - 1st block for a level has down sampling
 - Parameterized width ($\sim 2.5x$ increase per level)
 - Parameterized group size in the 3x3
 - RegNetY includes SE blocks, RegNetX does not
 - Gives about a 0.5% – 1.5% accuracy boost
- Simple head
 - Global avg pool
 - Fully connected layer



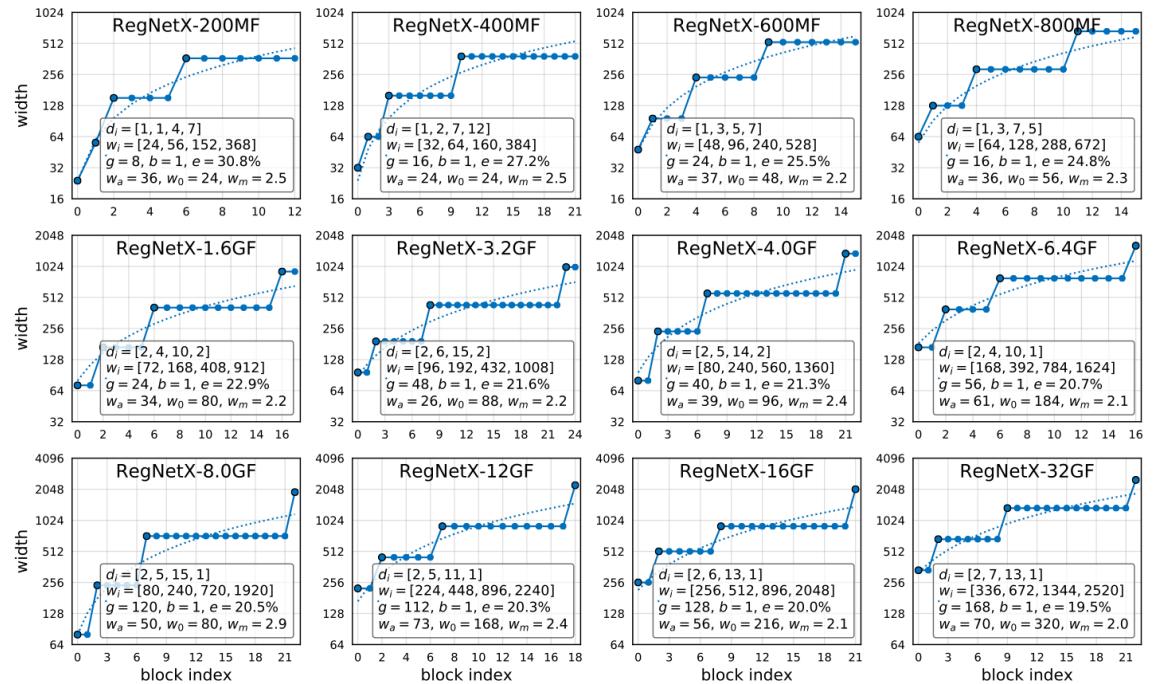
(a) X block, s=1



(b) X block, s=2

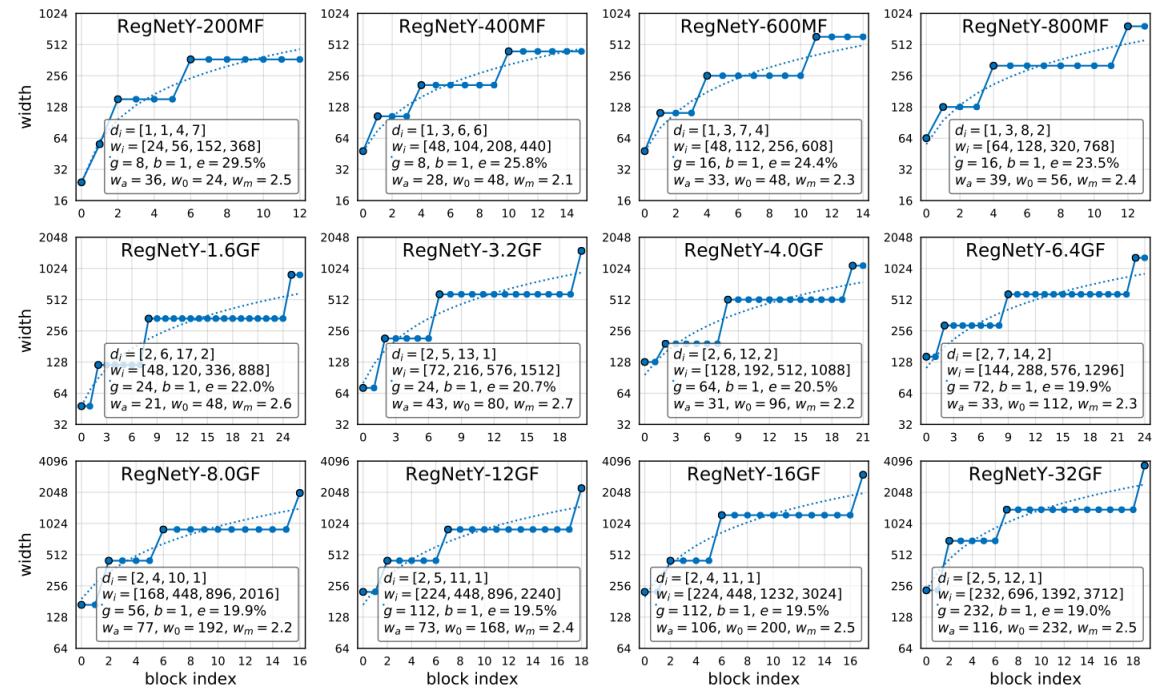
RegNetX Example Models

	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	error (top-1)
REGNETX-200MF	0.2	2.7	2.2	1024	10	2.8	31.1 ± 0.09
REGNETX-400MF	0.4	5.2	3.1	1024	15	3.9	27.3 ± 0.15
REGNETX-600MF	0.6	6.2	4.0	1024	17	4.4	25.9 ± 0.03
REGNETX-800MF	0.8	7.3	5.1	1024	21	5.7	24.8 ± 0.09
REGNETX-1.6GF	1.6	9.2	7.9	1024	33	8.7	23.0 ± 0.13
REGNETX-3.2GF	3.2	15.3	11.4	512	57	14.3	21.7 ± 0.08
REGNETX-4.0GF	4.0	22.1	12.2	512	69	17.1	21.4 ± 0.19
REGNETX-6.4GF	6.5	26.2	16.4	512	92	23.5	20.8 ± 0.07
REGNETX-8.0GF	8.0	39.6	14.1	512	94	22.6	20.7 ± 0.07
REGNETX-12GF	12.1	46.1	21.4	512	137	32.9	20.3 ± 0.04
REGNETX-16GF	15.9	54.3	25.5	512	168	39.7	20.0 ± 0.11
REGNETX-32GF	31.7	107.8	36.3	256	318	76.9	19.5 ± 0.12



RegNetY Example Models

	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	error (top-1)
REGNEY-200MF	0.2	3.2	2.2	1024	11	3.1	29.6 ± 0.11
REGNEY-400MF	0.4	4.3	3.9	1024	19	5.1	25.9 ± 0.16
REGNEY-600MF	0.6	6.1	4.3	1024	19	5.2	24.5 ± 0.07
REGNEY-800MF	0.8	6.3	5.2	1024	22	6.0	23.7 ± 0.03
REGNEY-1.6GF	1.6	11.2	8.0	1024	39	10.1	22.0 ± 0.08
REGNEY-3.2GF	3.2	19.4	11.3	512	67	16.5	21.0 ± 0.05
REGNEY-4.0GF	4.0	20.6	12.3	512	68	16.8	20.6 ± 0.08
REGNEY-6.4GF	6.4	30.6	16.4	512	104	26.1	20.1 ± 0.04
REGNEY-8.0GF	8.0	39.2	18.0	512	113	28.1	20.1 ± 0.09
REGNEY-12GF	12.1	51.8	21.4	512	150	36.0	19.7 ± 0.06
REGNEY-16GF	15.9	83.6	23.0	512	189	45.6	19.6 ± 0.16
REGNEY-32GF	32.3	145.0	30.3	256	319	76.0	19.0 ± 0.12



RegNetX/Y Comparisons

Mobile regime

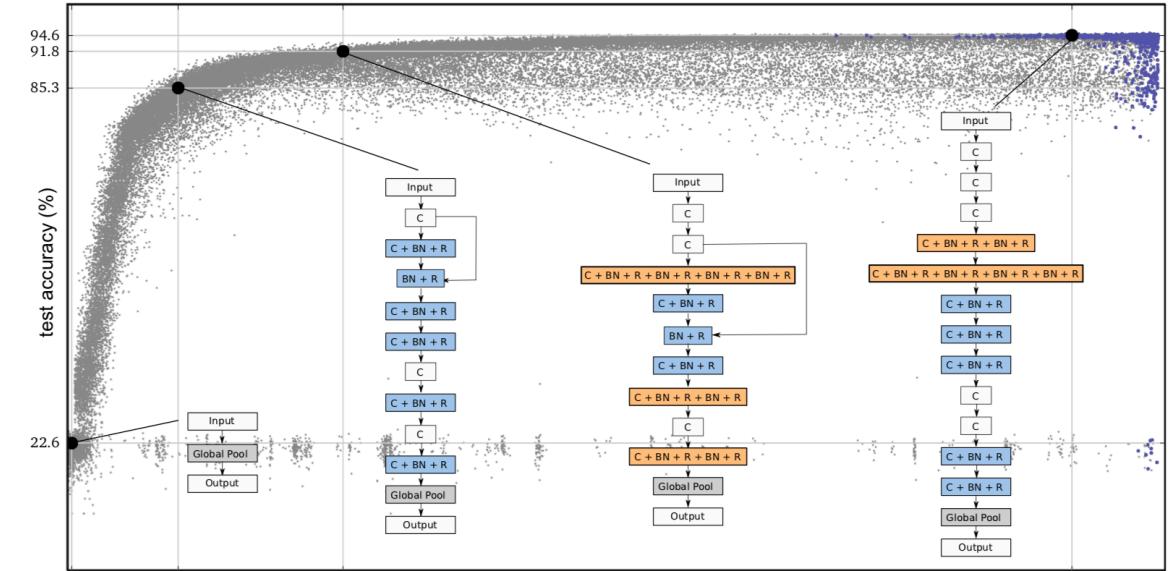
	flops (B)	params (M)	top-1 error
MOBILENET [9]	0.57	4.2	29.4
MOBILENET-V2 [25]	0.59	6.9	25.3
SHUFFLENET [33]	0.52	-	26.3
SHUFFLENET-V2 [19]	0.59	-	25.1
NASNET-A [35]	0.56	5.3	26.0
AMOEBA-NET-C [23]	0.57	6.4	24.3
PNASNET-5 [17]	0.59	5.1	25.8
DARTS [18]	0.57	4.7	26.7
REGNETX-600MF	0.60	6.2	25.9 ± 0.03
REGNETY-600MF	0.60	6.1	24.5 ± 0.07

EfficientNet (using standard training)

	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	top-1 error ours \pm std [orig]
EFFICIENTNET-B0	0.4	5.3	6.7	256	34	11.7	24.9 ± 0.03 [23.7]
REGNETY-400MF	0.4	4.3	3.9	1024	19	5.1	25.9 ± 0.16
EFFICIENTNET-B1	0.7	7.8	10.9	256	52	15.6	24.1 ± 0.16 [21.2]
REGNETY-600MF	0.6	6.1	4.3	1024	19	5.2	24.5 ± 0.07
EFFICIENTNET-B2	1.0	9.2	13.8	256	68	18.4	23.4 ± 0.06 [20.2]
REGNETY-800MF	0.8	6.3	5.2	1024	22	6.0	23.7 ± 0.03
EFFICIENTNET-B3	1.8	12.0	23.8	256	114	32.1	22.5 ± 0.05 [18.9]
REGNETY-1.6GF	1.6	11.2	8.0	1024	39	10.1	22.0 ± 0.08
EFFICIENTNET-B4	4.2	19.0	48.5	128	240	65.1	21.2 ± 0.06 [17.4]
REGNETY-4.0GF	4.0	20.6	12.3	512	68	16.8	20.6 ± 0.08
EFFICIENTNET-B5	9.9	30.0	98.9	64	504	135.1	21.5 ± 0.11 [16.7]
REGNETY-8.0GF	8.0	39.2	18.0	512	113	28.1	20.1 ± 0.09

A Few Final Thoughts

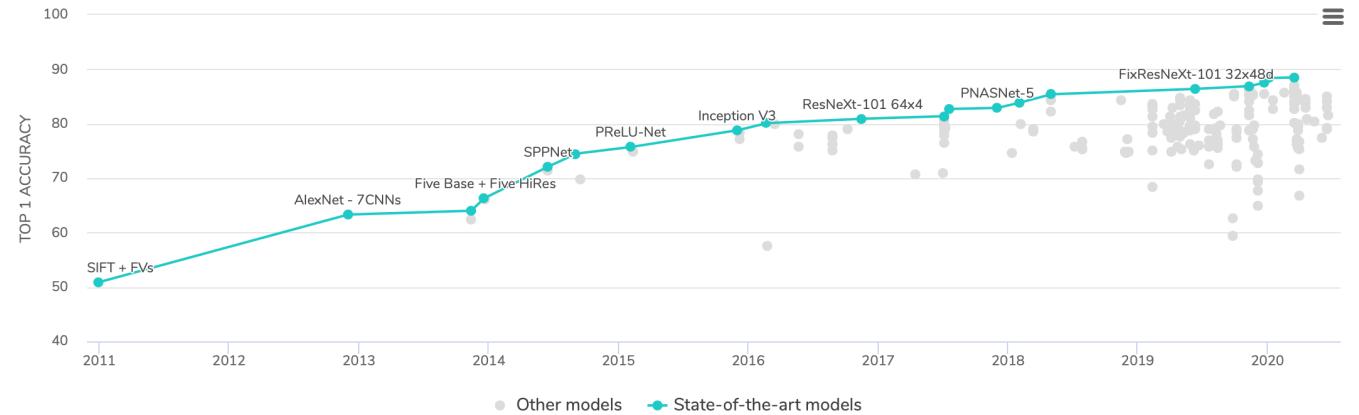
- Round tripping: while originally devised as a method for designing networks that (sort of) doesn't require humans, neural architecture search actually can be used to give humans ideas on what matters most in network design
 - Makes sense as it can be viewed as a really really large set of trials with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



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



I used to put a list of key networks and their top 1/5 accuracy on ImageNet but there's a problem – a small revolution in training has enabled significant accuracy gains from the values originally reported in the papers; trying to separate out what is an architectural improvement vs an artifact of less good training is becoming more difficult

On top of this, complexity for networks varies widely, both theoretical and in practice when mapped to different architectures

But I would like to highlight the Papers With Code web site as a good way of tracking state of the art

CNNs – Visualization

What Feature Maps Look Like

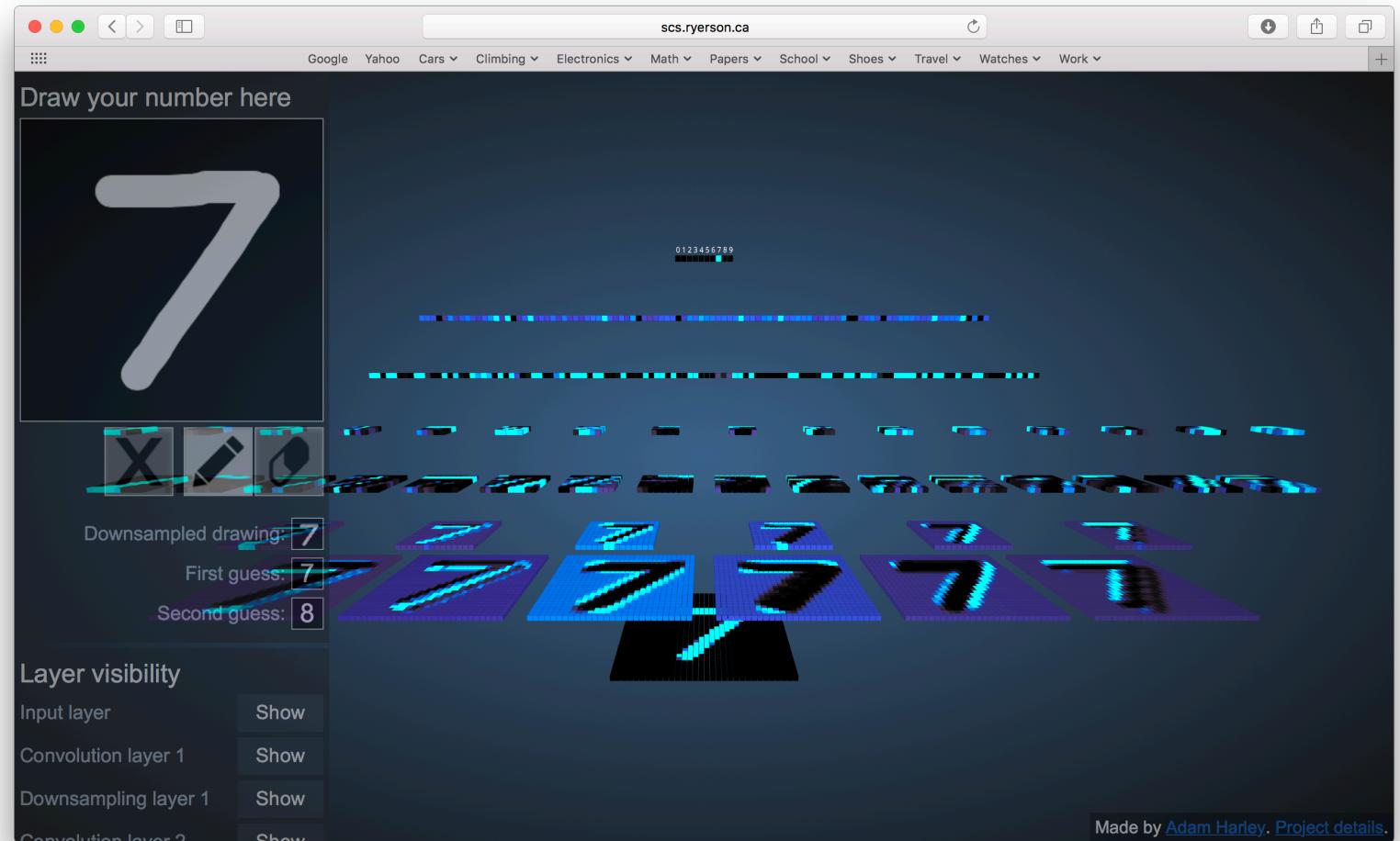
- In general ...
 - Smooth regions and edges at the beginning (image space)
 - Spikey at the end (feature space)
- In general ...
 - Ok to clip some in the beginning (many important parts)
 - Don't want to clip at the end (want to find the max)

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>
 - What does it mean to understand a neural network?
 - <https://arxiv.org/abs/1907.06374>
 - 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>

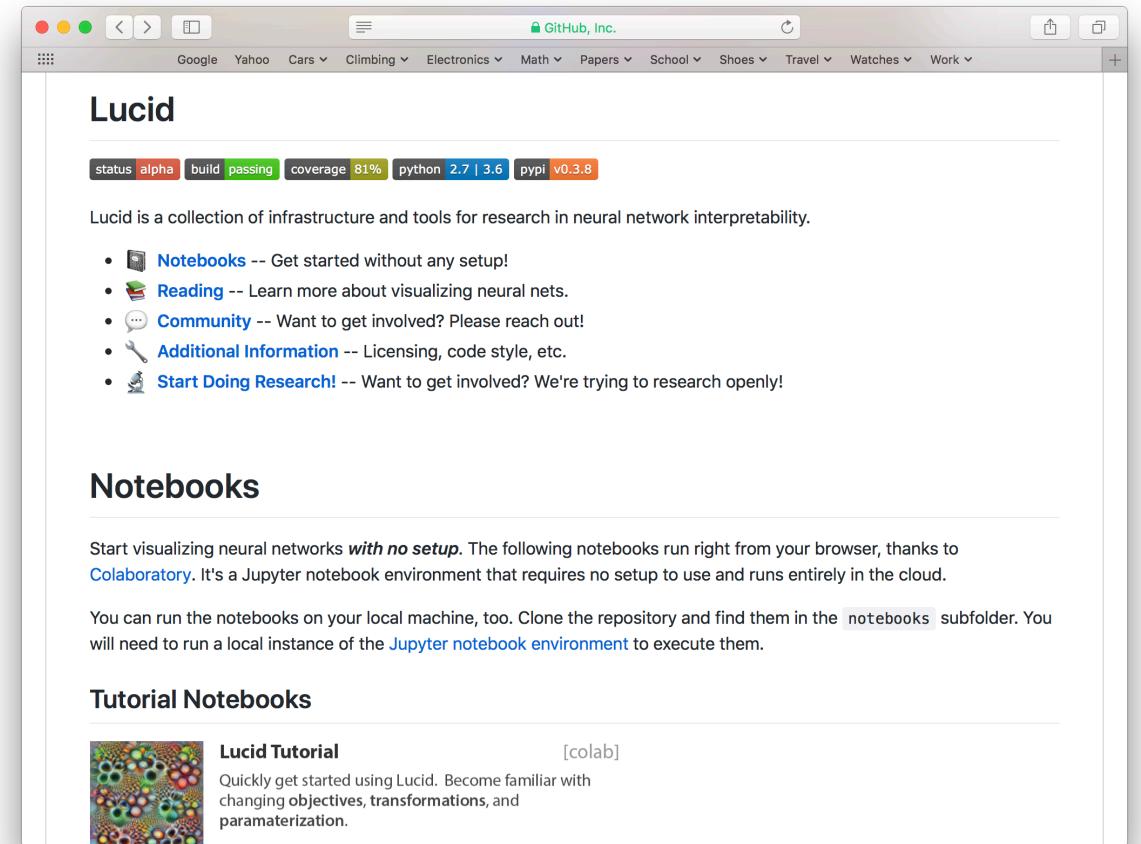
Interactive Demo

- 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/>



For More Info On Visualization

- Visualization for machine learning
 - https://media.neurips.cc/Conferences/NIPS2018/Slides/Visualization_for_ML.pdf
 - This is a generally useful tutorial on visualization
- Exploring neural networks with activation atlases
 - <https://distill.pub/2019/activation-atlas/>

RNNs

Quick Comment

- Ok, so you know how at the beginning of this lecture we said it's nice to have a default problem to think about with respect to network design and that we were going to use image classification as that default problem?
- The RNN section is an exception to this
- But if you want a default problem to think about in the RNN section, think about predicting the next word from a previous sequence of words
- We'll discuss some other problems in the language and speech sections

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

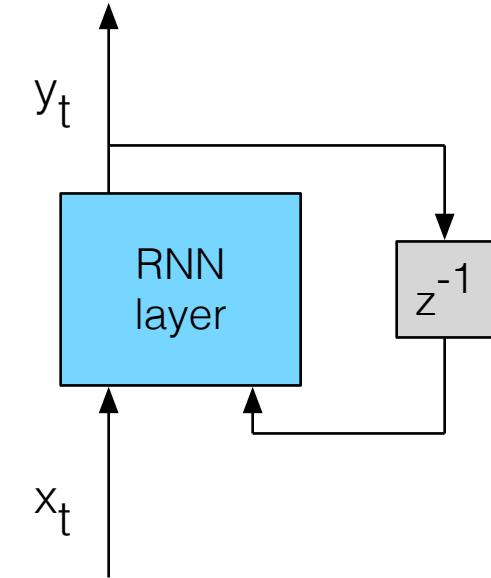
Side note: feed forward neural networks are universal function approximators but not Turing complete; RNNs, via their memory state, are also Turing complete

On the computational power of neural nets

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

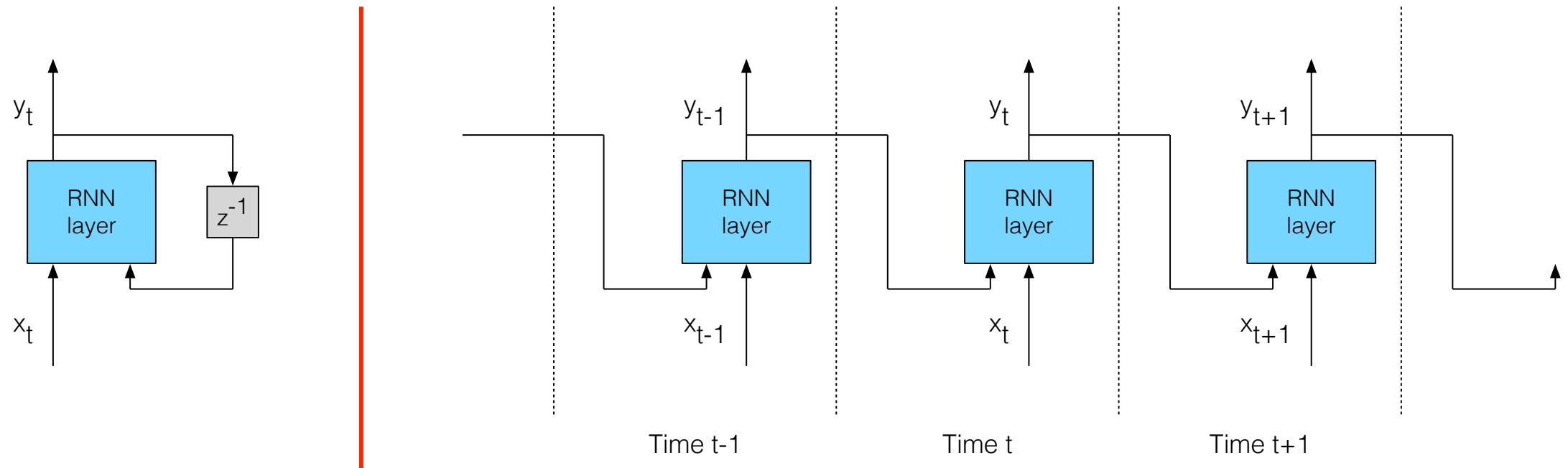
Layers: RNN

- RNN layer
 - $y_t^T = f(x_t^T H + y_{t-1}^T G + v^T)$
 - t is the current time step, $t - 1$ is the previous time step
 - $x_t^T H$ is the multiplication of the $1 \times K$ input vector x_t^T with the $K \times N$ matrix H
 - $y_{t-1}^T G$ is the multiplication of the $1 \times N$ previous output vector y_{t-1}^T with the $N \times N$ matrix G
 - v^T is a $1 \times N$ bias vector
 - f is a pointwise nonlinearity
 - y_t^T is the $1 \times N$ current output vector



Layers: RNN

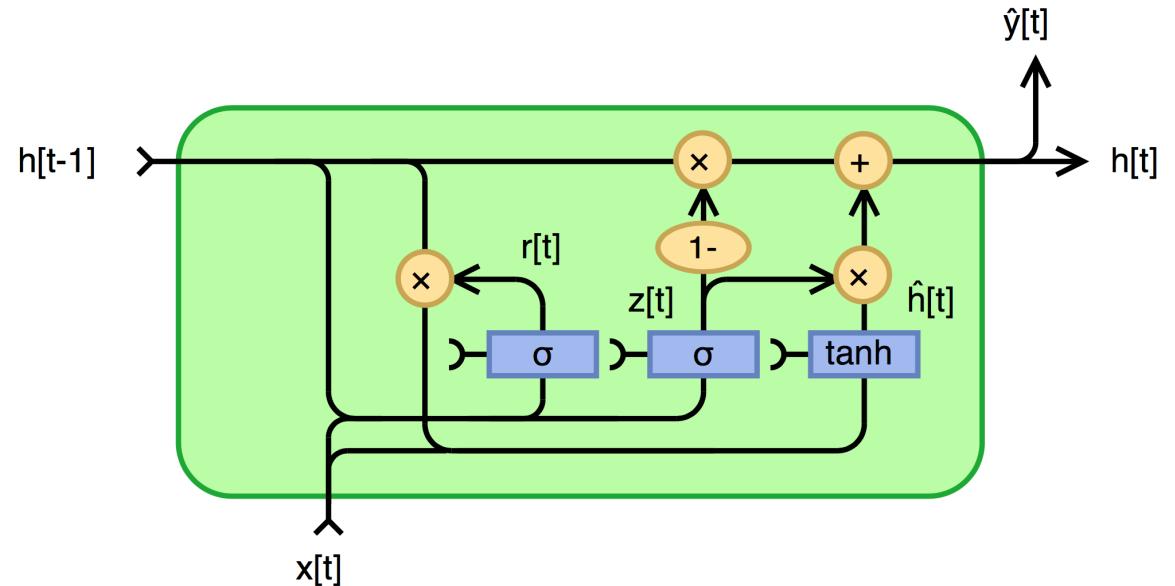
A RNN layer illustrated with feedback (left) and unwrapped in time (right)



Layers: GRU

Gated recurrent units

- Update gate
 - $u_t^T = \sigma_g (x_t^T H_u + y_{t-1}^T G_u + v_u^T)$
 - Parameters H_u , G_u and v_u^T
 - σ_g is a sigmoid, constrains gate to (0, 1)
- Reset gate
 - $r_t^T = \sigma_g (x_t^T H_r + y_{t-1}^T G_r + v_r^T)$
 - Parameters H_r , G_r and v_r^T
 - σ_g is a sigmoid, constrains gate to (0, 1)
- Output
 - $y_t^T = [u_t^T \odot y_{t-1}^T] + [(1 - u_t^T) \odot \sigma_h (x_t^T H_y + (r_t^T \odot y_{t-1}^T) G_y + v_y^T)]$
 - Parameters H_y , G_y and v_y^T
 - σ_h is a hyperbolic tan, constrains update term to (-1, 1)
 - \odot is the Hadamard (point wise) product



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

Layers: LSTM

Long short term memory

- Forget gate
 - $f_t^T = \sigma_g (x_t^T H_f + y_{t-1}^T G_f + v_f^T)$
 - Parameters H_f , G_f and v_f^T
 - σ_g is a sigmoid
 - How much to keep of the current state
- Input gate
 - $i_t^T = \sigma_g (x_t^T H_i + y_{t-1}^T G_i + v_i^T)$
 - Parameters H_i , G_i and v_i^T
 - σ_g is a sigmoid
 - How much to keep of the current input
- Output gate
 - $o_t^T = \sigma_g (x_t^T H_o + y_{t-1}^T G_o + v_o^T)$
 - Parameters H_o , G_o and v_o^T
 - σ_g is a sigmoid
 - How much to expose of the output

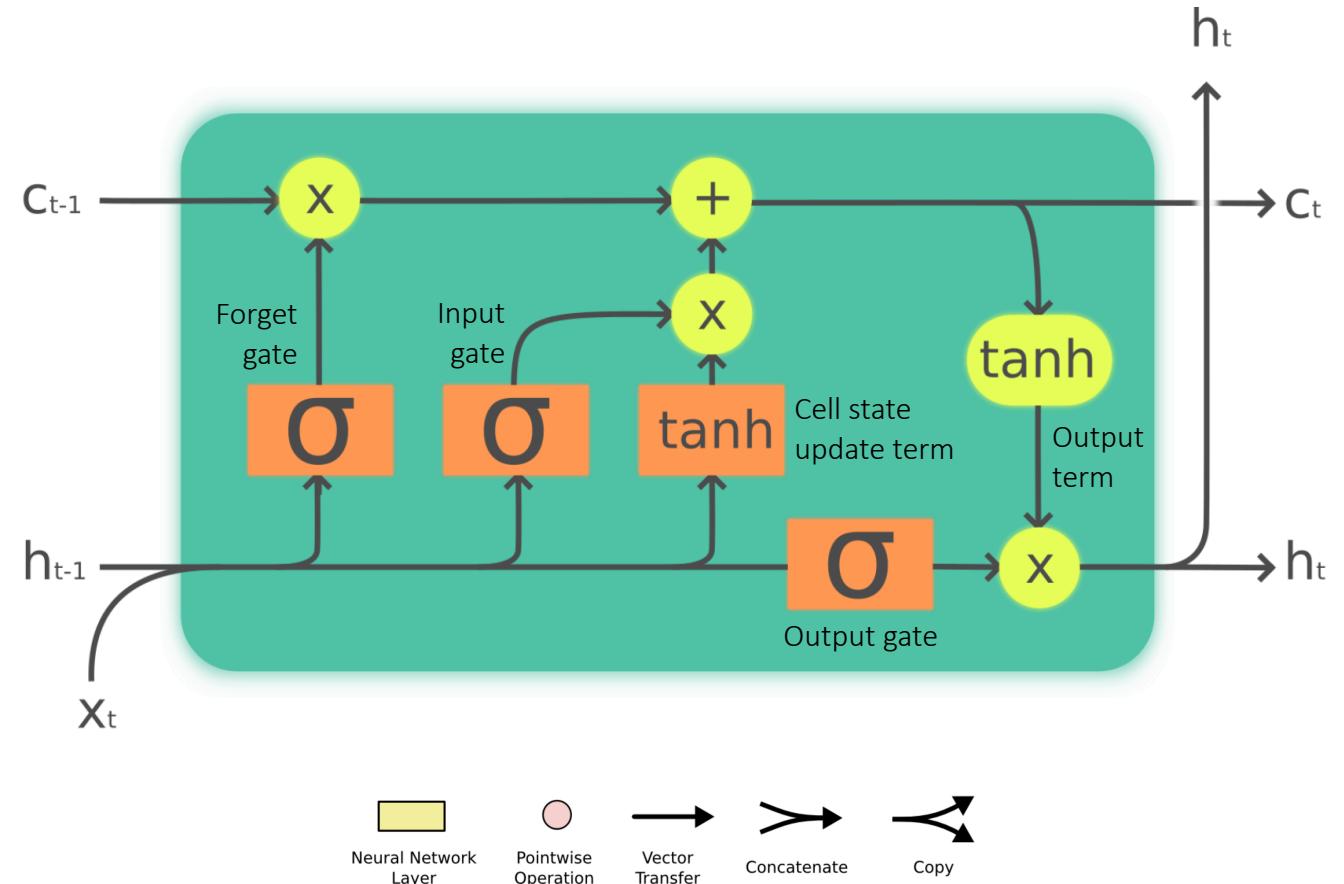


Figure from https://en.wikipedia.org/wiki/Long_short-term_memory 109

Layers: LSTM

Long short term memory

- Cell state update

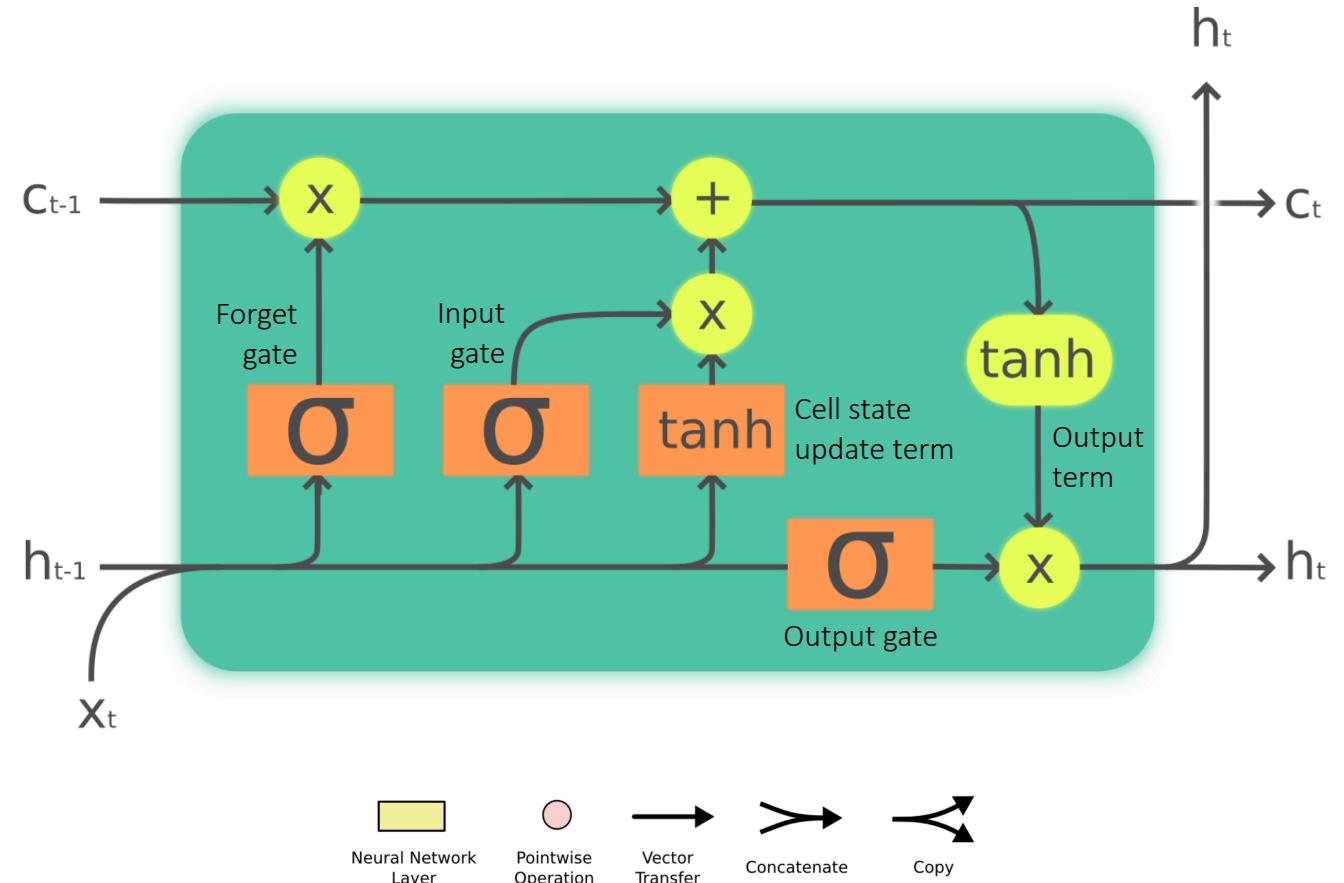
- $c_t^T = f_t^T \odot c_{t-1}^T + i_t^T \odot \sigma_h(x_t^T H_c + y_{t-1}^T G_c + v_c^T)$
- Parameters W_c , U_c and b_c
- σ_h is a hyperbolic tangent

- Output

- $y_t^T = o_t^T \odot \sigma_h(c_t^T)$
- σ_h is a hyperbolic tangent

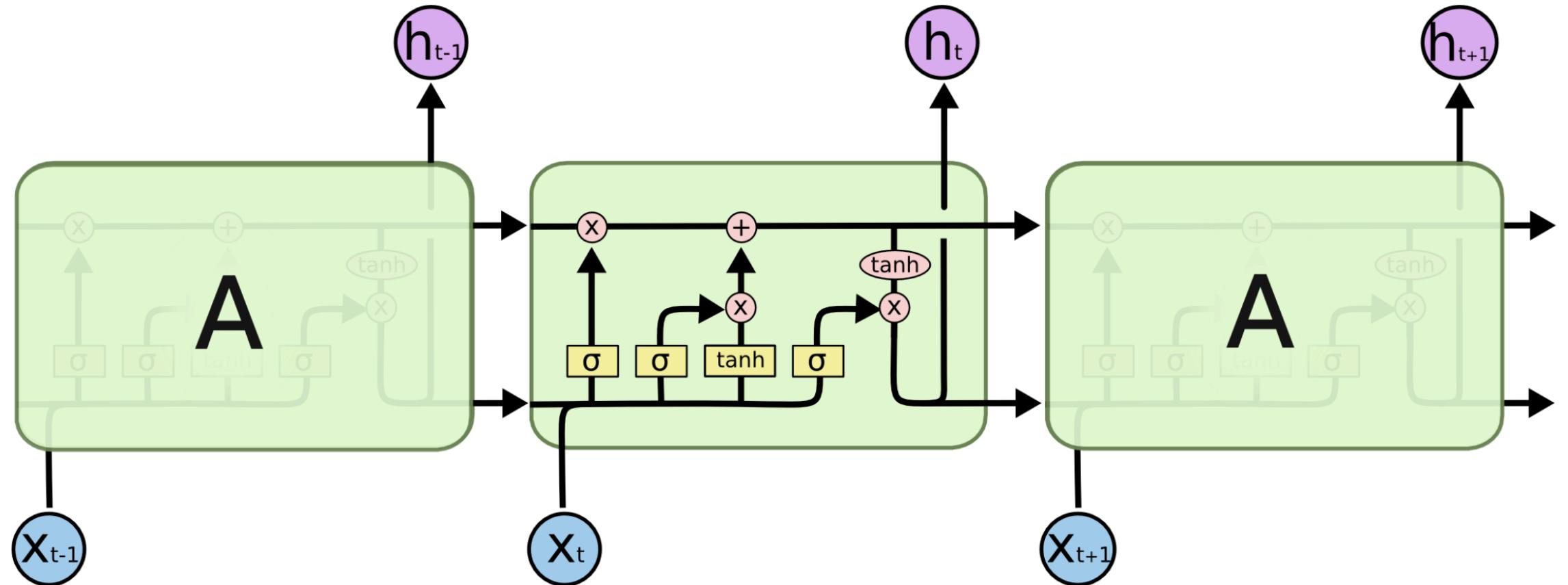
- Nota bene

- The RNN and GRU cells used the output as the cell state
- The LSTM cell has a separate output and cell states



Layers: LSTM

Long short term memory

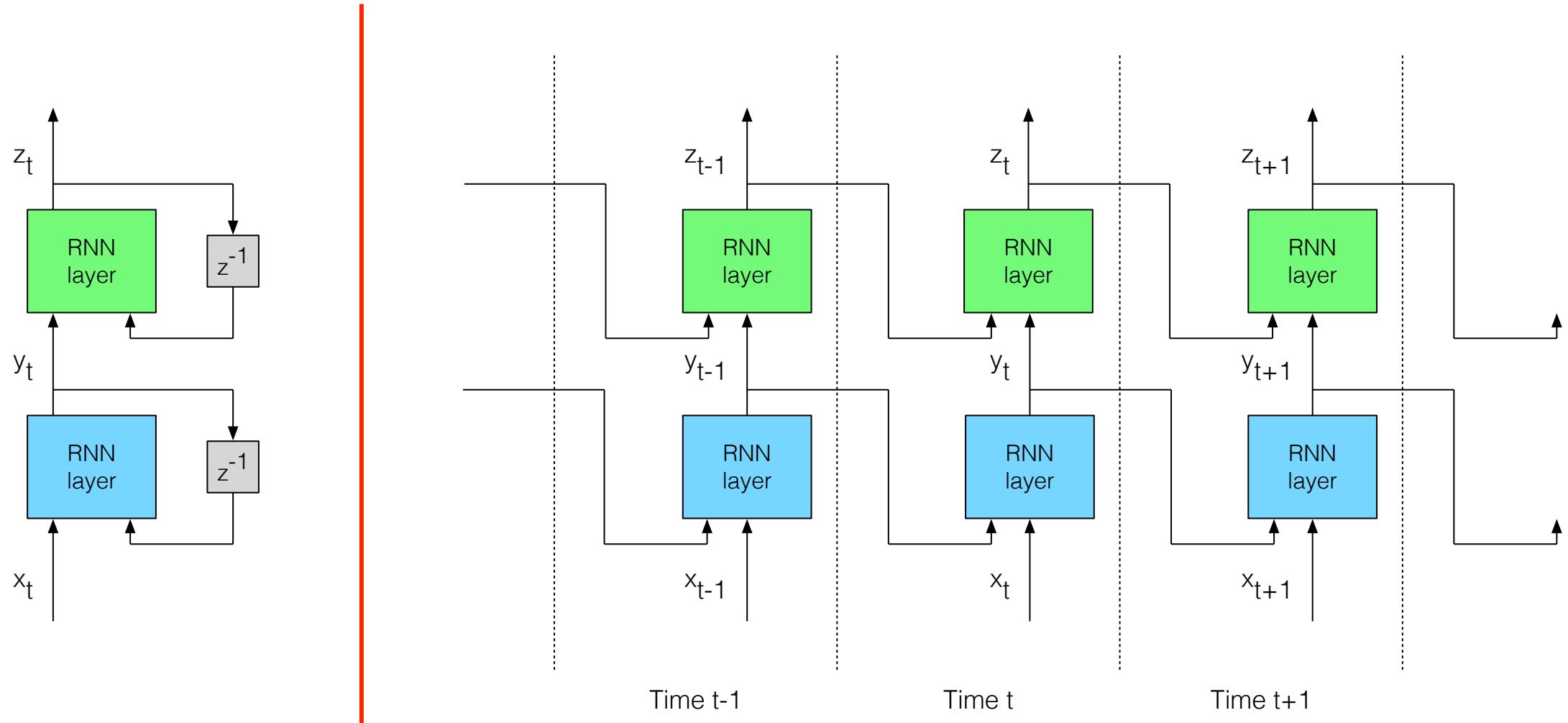


Legend:
Yellow square: Neural Network Layer
Pink circle: Pointwise Operation
Black arrow: Vector Transfer
Double-headed arrow: Concatenate
Copy icon: Copy

Figure from <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> 111

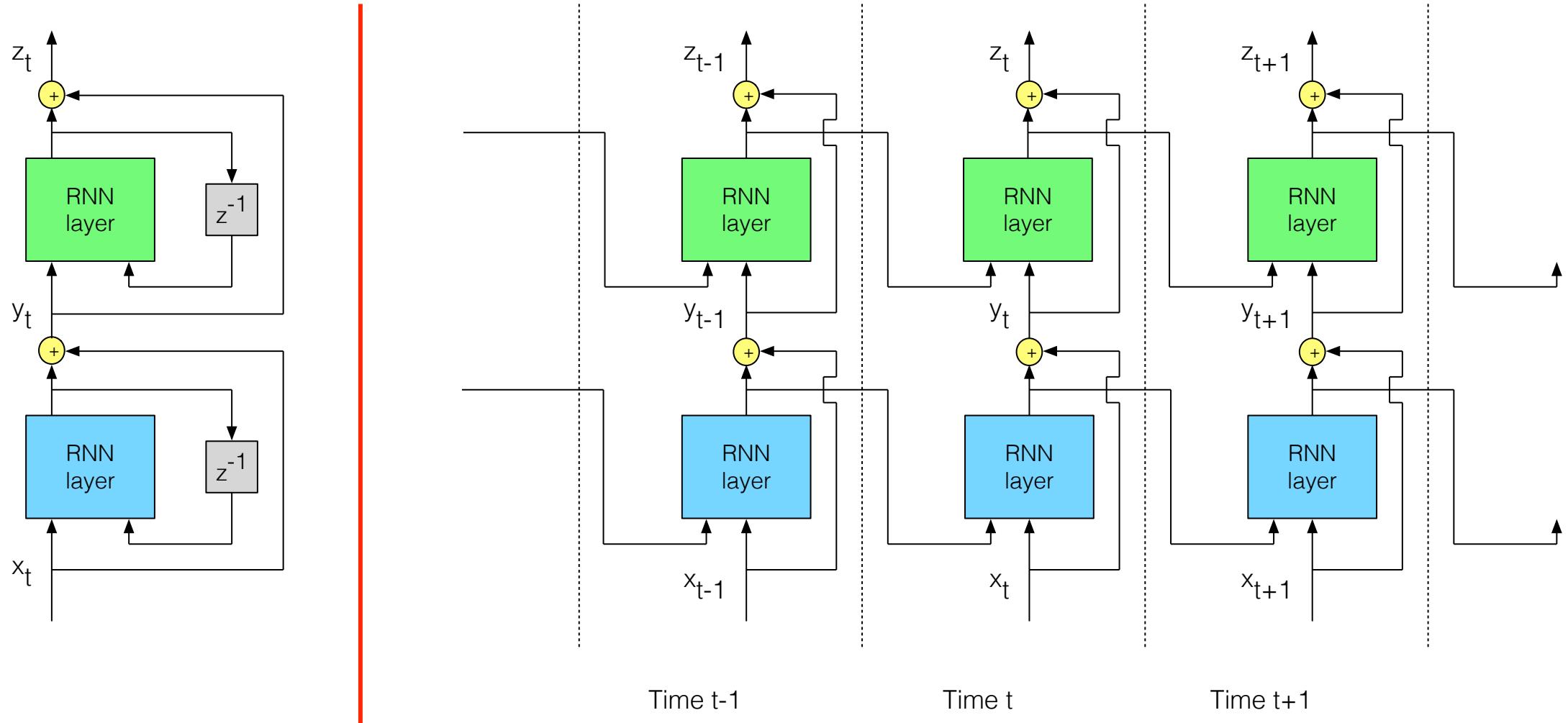
Networks: Stacked

RNNs exploit sequential structure in time (frame) to map from weaker features to stronger features; figure shown for a RNN or GRU cell (LSTM would have a separate output and cell state connection)



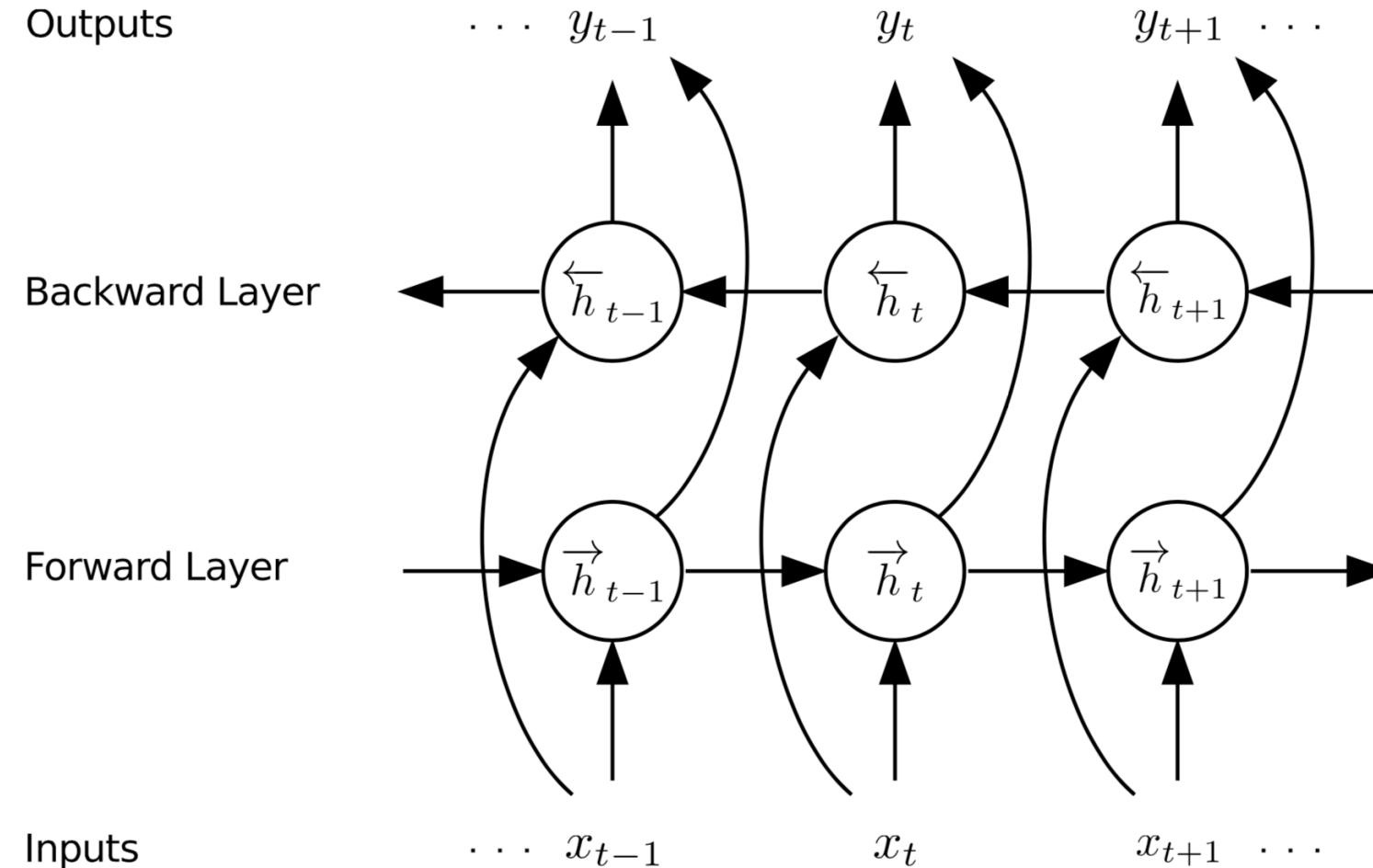
Networks: Residual

This would typically be part of a stacked RNN design; figure shown for a RNN or GRU cell (LSTM would have a separate output and cell state connection)



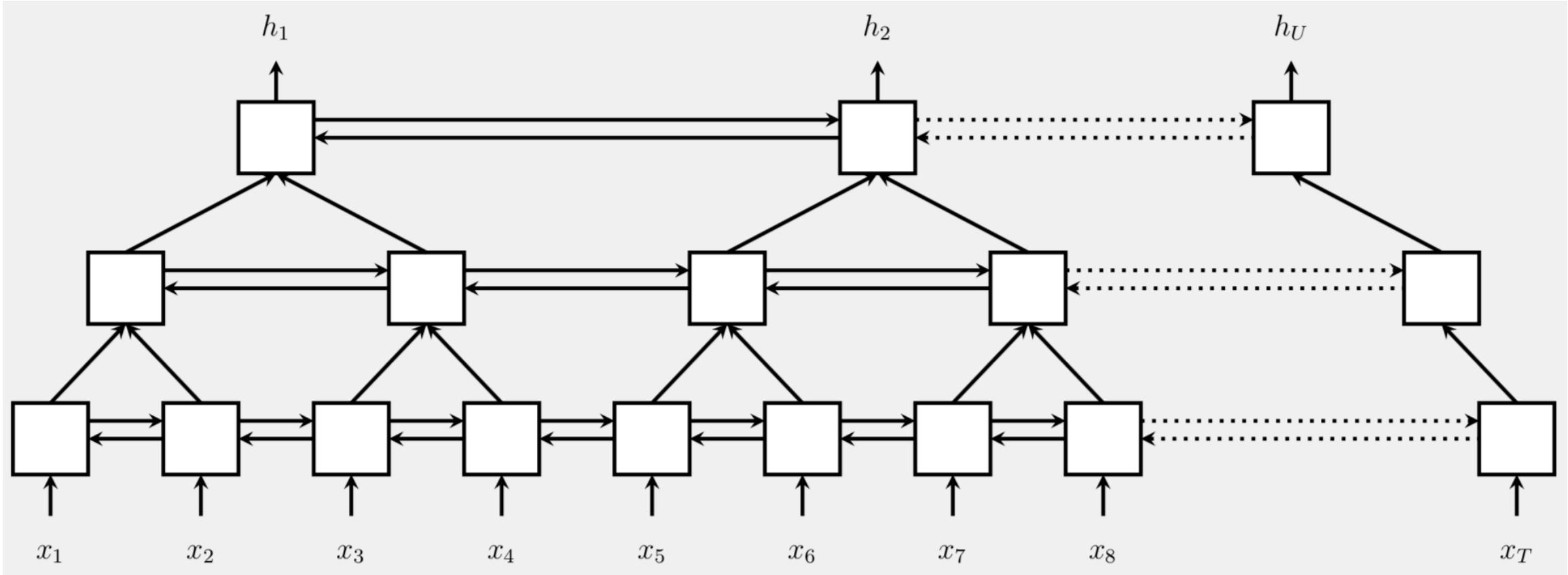
Networks: Bi Directional

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



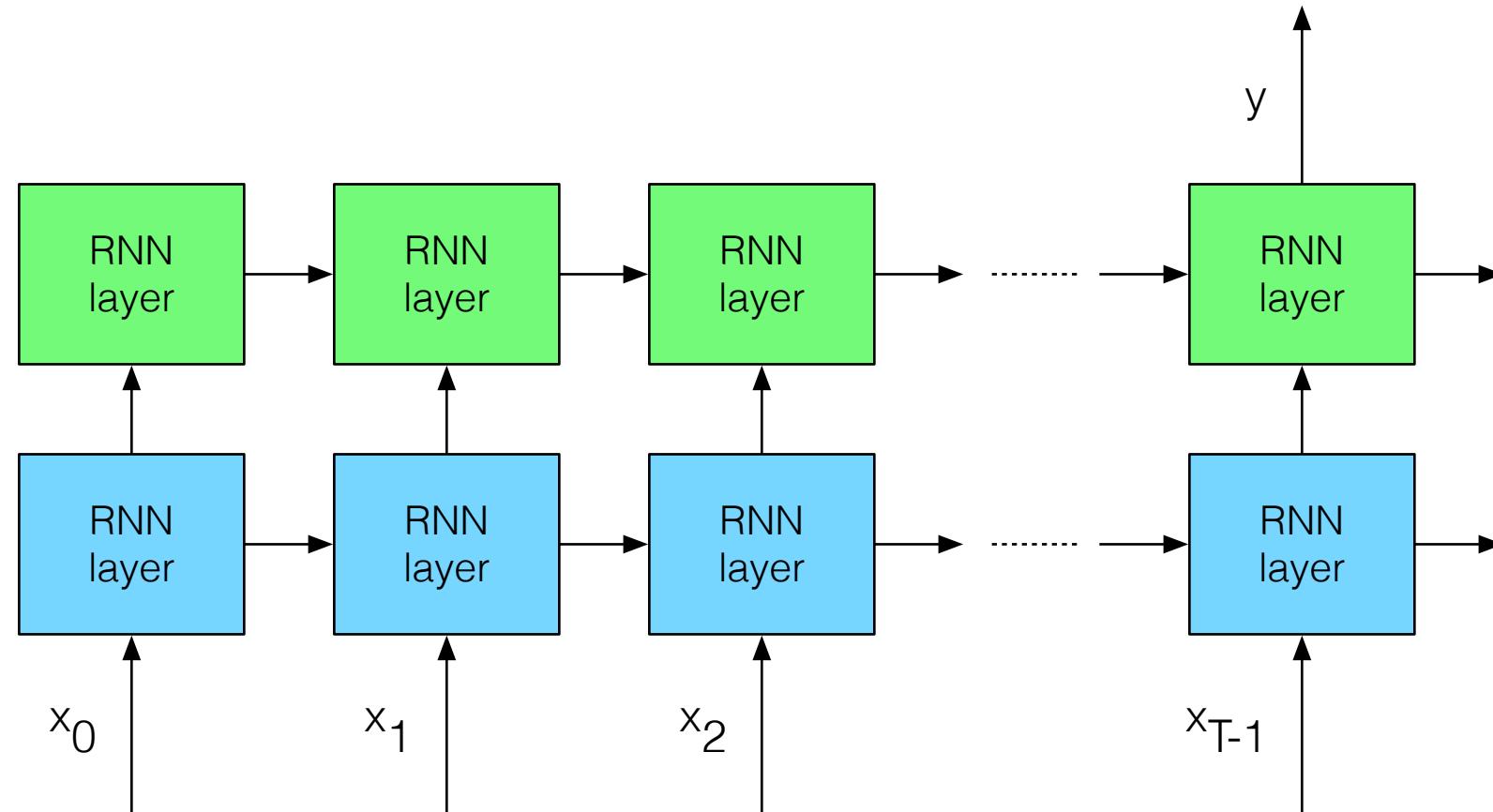
Networks: Pyramidal

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



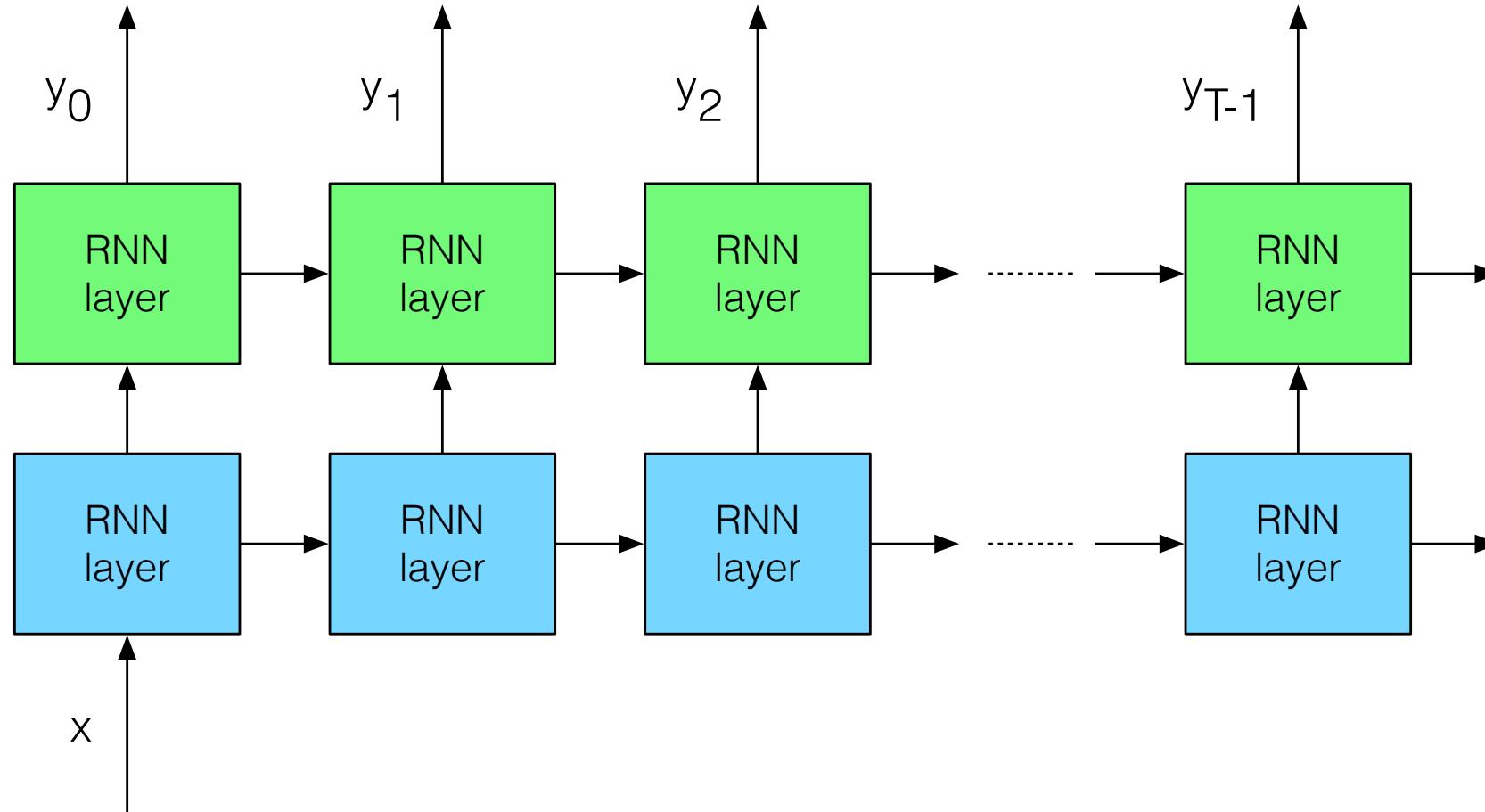
Mappings: Many Inputs To 1 Output

Illustrated here with a stacked 2 layer uni directional RNN



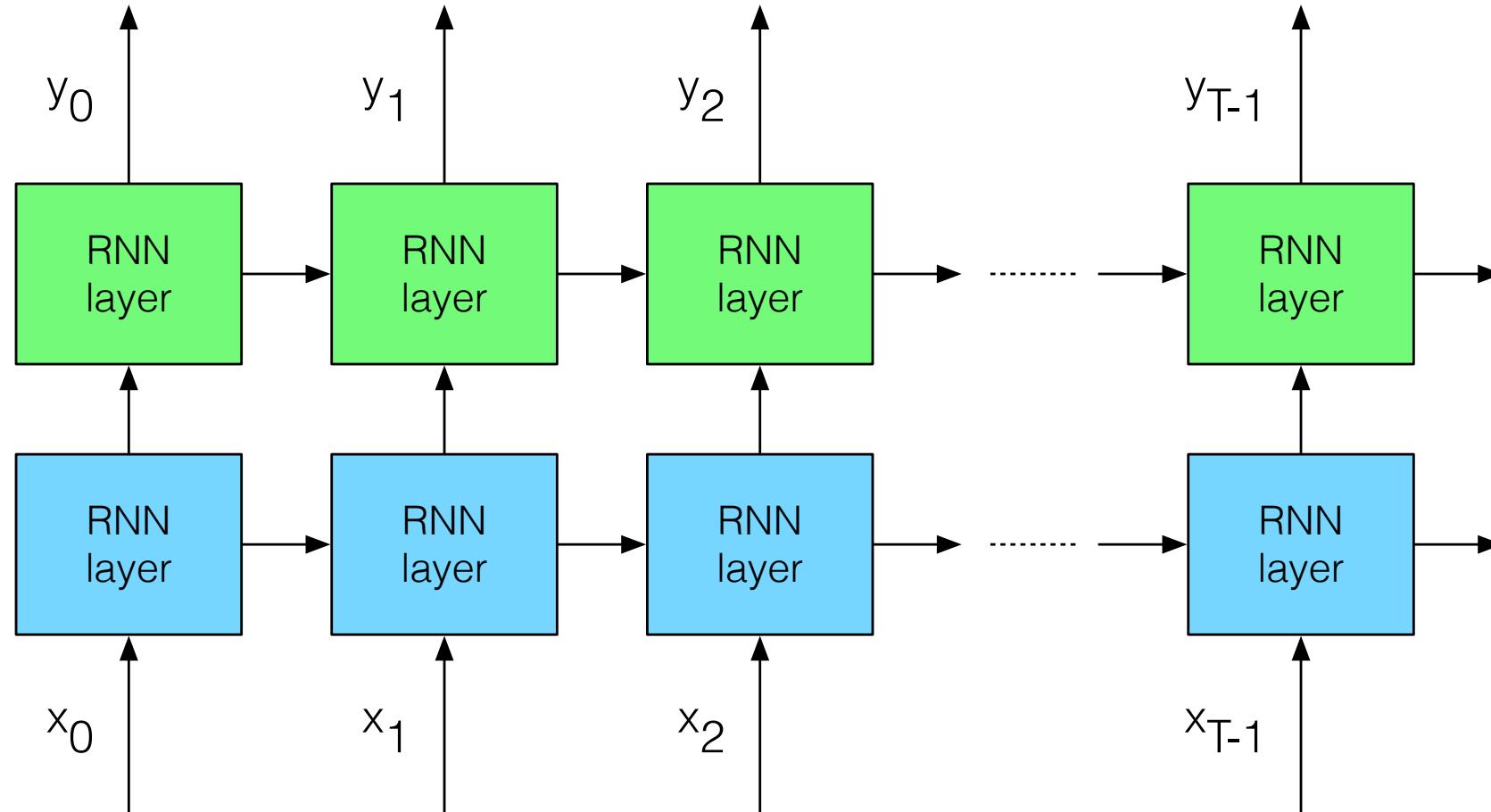
Mappings: 1 Input To Many Outputs

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>
- The unreasonable effectiveness of the forget gate
 - <https://arxiv.org/abs/1804.04849>
- 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>

Transformers

Transformers Design Section Overview

- Self attention plays a key role in the design of transformers
 - Standard self attention operation
 - Linear complexity self attention operation
- Transformer building blocks
 - Standard(-ish) transformer
 - Neural architecture search variants
- Image classification networks
 - Operator replacement
 - Patch embedding

... in xNN design



Self Attention Operation Reminder

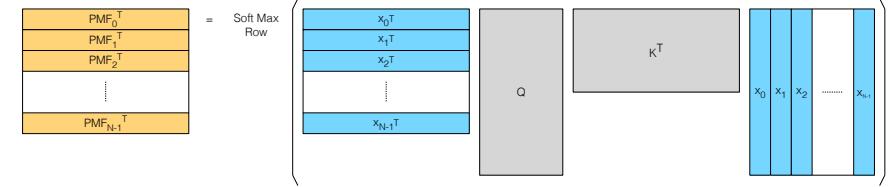
- Multi headed self attention

- Intra feature vector mixing via value V_i and output O_i matrices
- Inter feature vector mixing via the input dependent softmax() term which can be thought of as the self attention matrix A_i^T
- Trained parameter matrices query Q_i , key K_i , value V_i , output O_i and vector bias b are independent of the sequence length N
- Complexity is quadratic in the sequence length ($N \times N$ matrix multiplies a $N \times d_m$ matrix for a total of $N^2 d_m$ MACs)
- $Z^T = \text{SelfAttention} (X^T)$

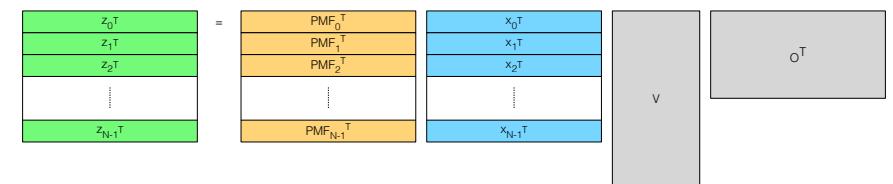
$$\begin{aligned}
 &= \sum_{i=0}^{\text{heads}-1} \text{softmax} (X^T Q_i K_i^T X / \text{scale}) X^T V_i O_i^T + 1 b^T \\
 &\quad \text{applied per row} \quad (N \times dm)(dm \times dk)(dk \times dm)(dm \times N) \quad (N \times dm)(dm \times dv)(dv \times dm) \quad (N \times 1)(1 \times dm)
 \end{aligned}$$

$$= \sum_{i=0}^{\text{heads}-1} A_i^T X^T V_i O_i^T + 1 b^T$$

1. Create self attention matrix A_i^T



2. Mix across (via A_i^T) and within (via $V_i O_i^T$) feature vectors then add bias 1 b^T (not shown)



Self Attention Op With Linear Complexity

- This is the multi headed self attention operation as defined in the Linformer
 - E_i and F_i are used to make compute linear in sequence length N via abstracting the inner matrix dimension from the sequence length ($N \times k$ matrix multiplies a $k \times d_m$ matrix for a total of Nkd_m MACs)
 - E_i and F_i are trained for a specific sequence length N making the structure more cumbersome to apply to variable length sequences
 - Setting $E = E_i = F_i$ has little impact on accuracy (empirically observed)
 - $Z^T = s(X^T)$

$$= \sum_{i=0}^{\text{heads-1}} \text{softmax} \left(\frac{X^T Q_i K_i^T X E_i^T}{\text{scale}} \right) F_i X^T V_i O_i^T + 1 b$$

applied per row $N \times dm \ dm \times dk \ dk \times dm \ dm \times N \ N \times k$
 $k \times N \ N \times dm \ dm \times dv \ dv \times dm \ N \times 1 \ 1 \times dm$

Note that many other complexity reducing variants of self attention exist

Transformer Network Building Block

- Transformer network building block
 - Pre norm variant shown, repeated for blocks $l = 1, \dots, L$
 - Other variants exist (original paper used post norm, no bias self attention)
 - Transformer network building block design is an active area of research
 - Identity + residual with layer normalization and self attention

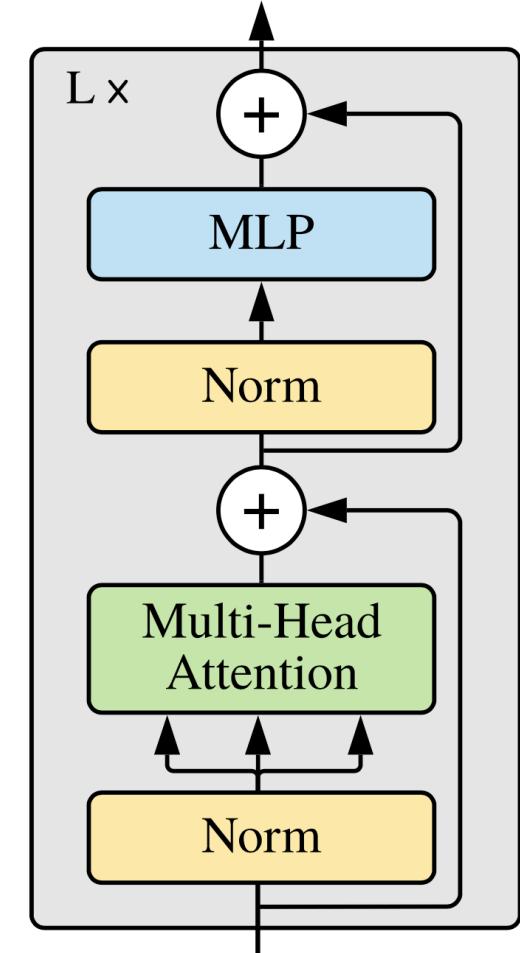
$$Y_l^T = X_{l-1}^T + d(s_l(g(X_{l-1}^T)))$$

N x dm dropout self attn norm N x dm
 N x dm dropout self attn norm N x dm

- Identity + residual with (repeat 2x) {layer normalization, linear layer and dropout}

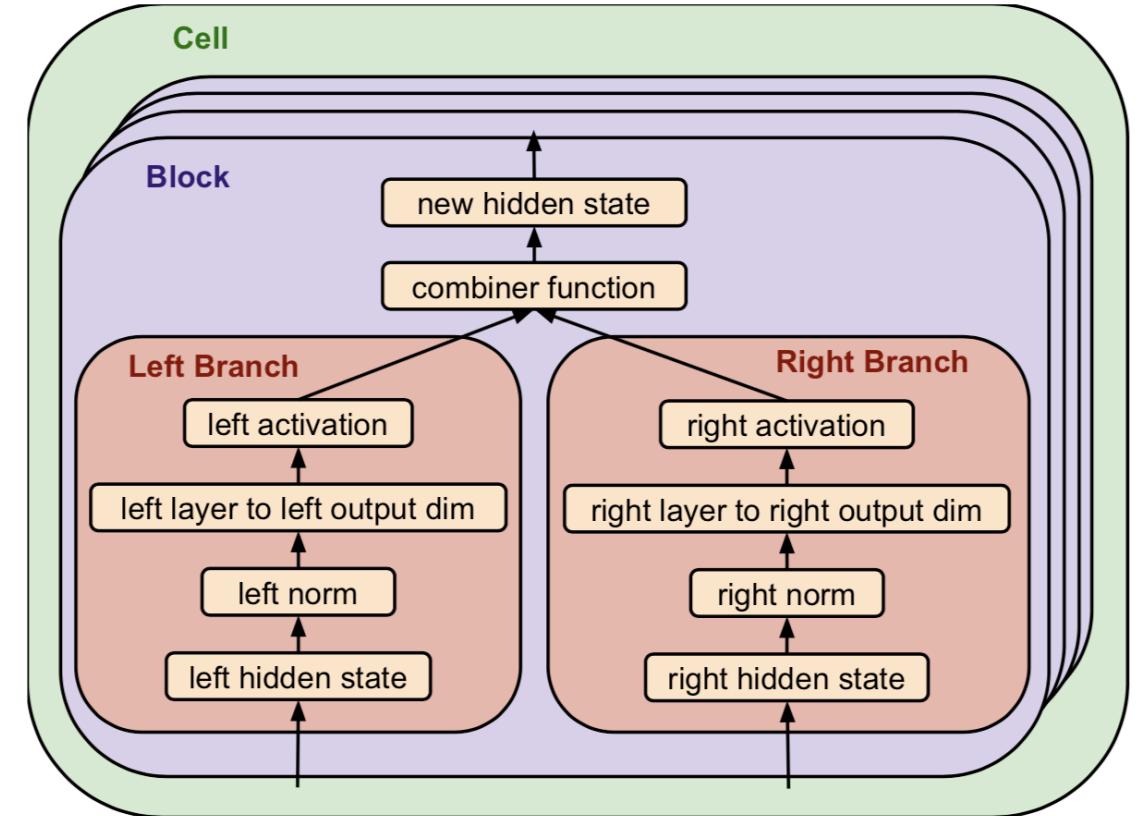
$$X_l^T = Y_l^T + d(d(f_{\text{GELU}}(g(Y_l^T) H_{l,A} + 1 b_{l,A}^T)) H_{l,B} + 1 b_{l,B}^T)$$

N x dm dropout dropout nonlinearity norm N x dm dm x df N x 1 1 x df df x dm N x 1 1 x dm
 N x dm dropout dropout nonlinearity norm N x dm dm x df N x 1 1 x df df x dm N x 1 1 x dm



Ex NAS Designed Transformer Building Blocks

- Note: expect a lot of emerging activity in this area
- Pay less attention with lightweight and dynamic convolutions
 - <https://arxiv.org/abs/1901.10430>
- The evolved transformer
 - Continuing the theme of using optimization to improve network design, training, ...
 - <https://arxiv.org/abs/1901.11117>



Ex NAS Designed Transformer Building Blocks

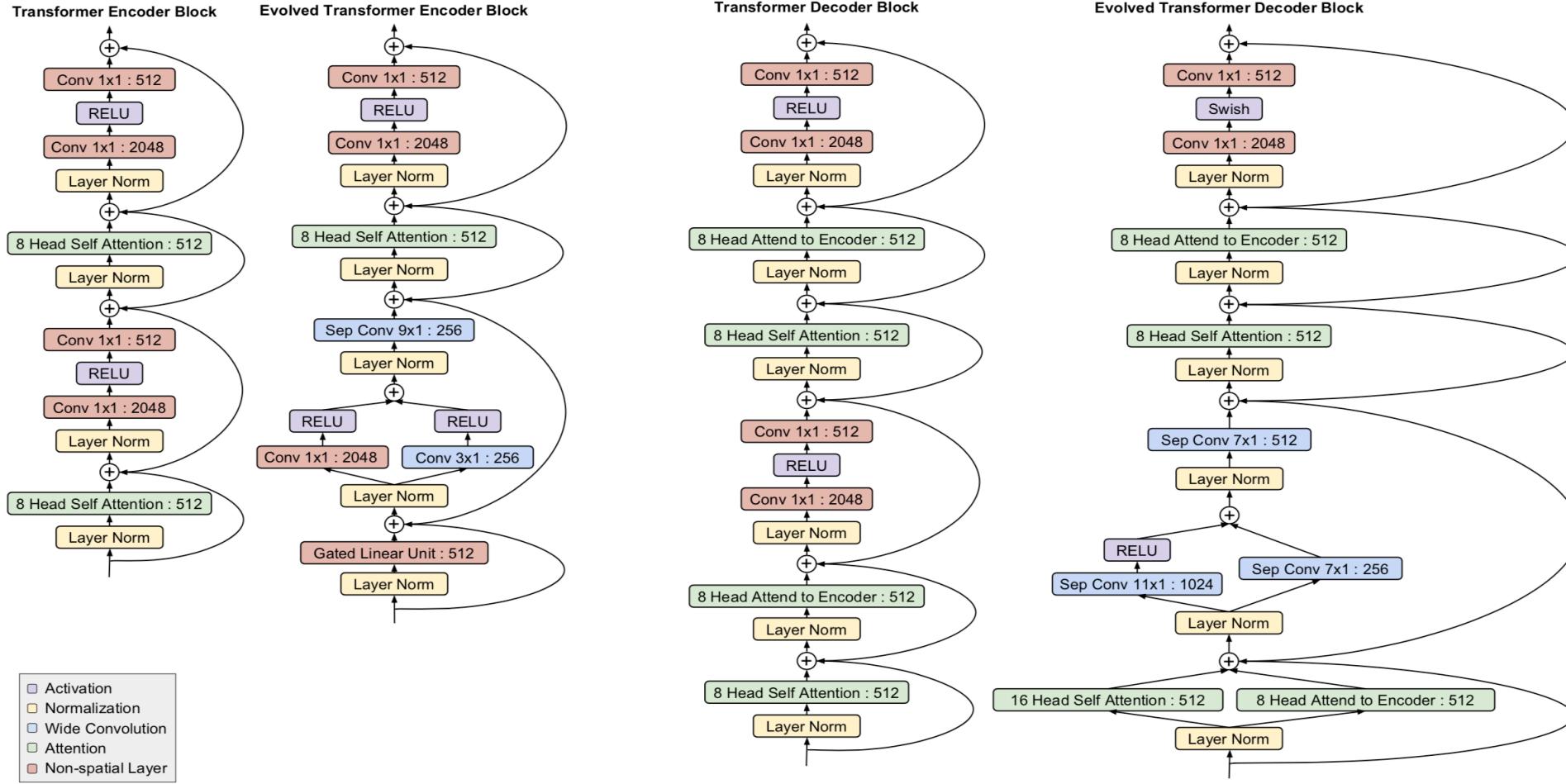


Image Classification Network Design

- Can we apply self attention to vision problems like image classification?
 - Data dependent filter coefficients
 - Receptive field benefits
 - High capacity architecture that can benefit from self supervised pre training on larger datasets
- Possibility 1: pixel based convolution operation replacement
 - Start from a common CNN architecture
 - Replace CNN style 2D convolution with self attention
- Possibility 2: patch based processing
 - Patch embedding → self attention encoder → decoder / classifier

1: Pixel Based Convolution Op Replacement

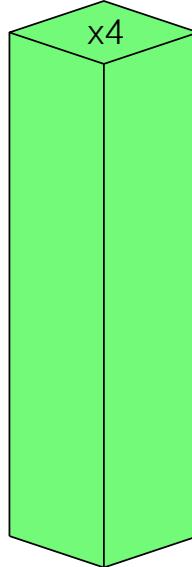
- Basic setup
 - X and Y are each $C \times H \times W$ feature maps
 - s is a to be defined self attention operation
 - Want $Y = s(X)$
- Self attention for convolution op replacement
 - Let $x_{i,j} = X(:, i, j)$ and $y_{i,j} = Y(:, i, j)$ be column vectors
 - Let $N(i, j)$ be a neighborhood (e.g., 3×3) around i, j with N elements indexed $0, \dots, N - 1$
 - Let $X_{N(i,j)} = [x_0, \dots, x_{N-1}]$ be a matrix formed by stacking all vectors $x_{i,j}$ with $i, j \in N(i, j)$ shoulder to shoulder
 - Let trainable matrices Q and K be $C \times d_m$ and V be $C \times C$
 - $y_{i,j}^T = \text{softmax}(\mathbf{x}_{i,j}^T Q K^T X_{N(i,j)} / \text{scale}) X_{N(i,j)}^T V + b^T$
 - Repeat this for every location i, j in the feature map

A few more comments and details

- Effectively doing attention based inter vector mixing followed by V matrix intra vector mixing
- The V based matrix multiplication is of size $C \times C$; to reduce computation grouping of channels is used (just like CNN style 2D convolution)
- As defined on this slide, the weighted combination of inputs used to create each output has no dependence on position (just similarity)
- To add a positional dependence a fixed or learned position vector p^T can be added to $x_{i,j}^T$ and a matrix P^T can be added to $X_{N(i,j)}^T$ inside the softmax operation
- Other variants are possible

Goal: Map Input Neighborhood To Output

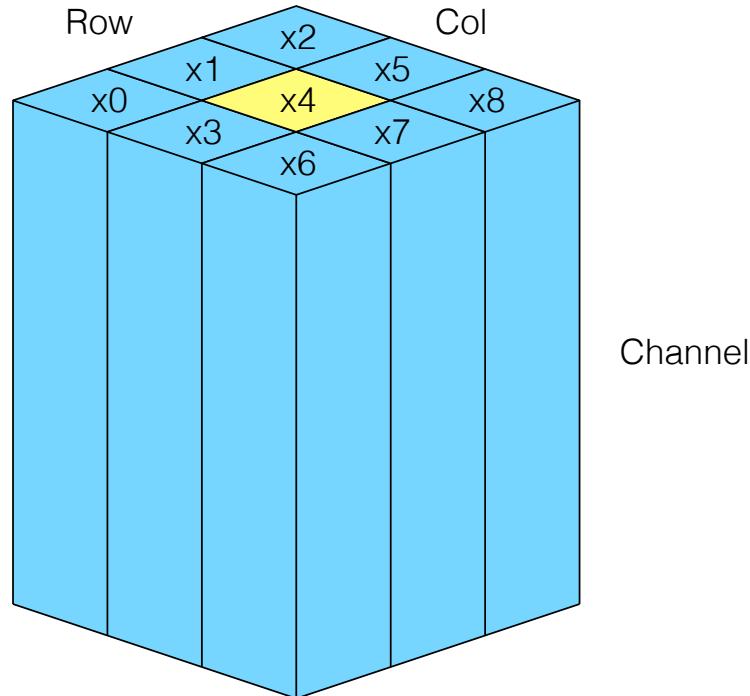
Repeat this for each output location i,j in the original feature map



Self attn conv op replacement



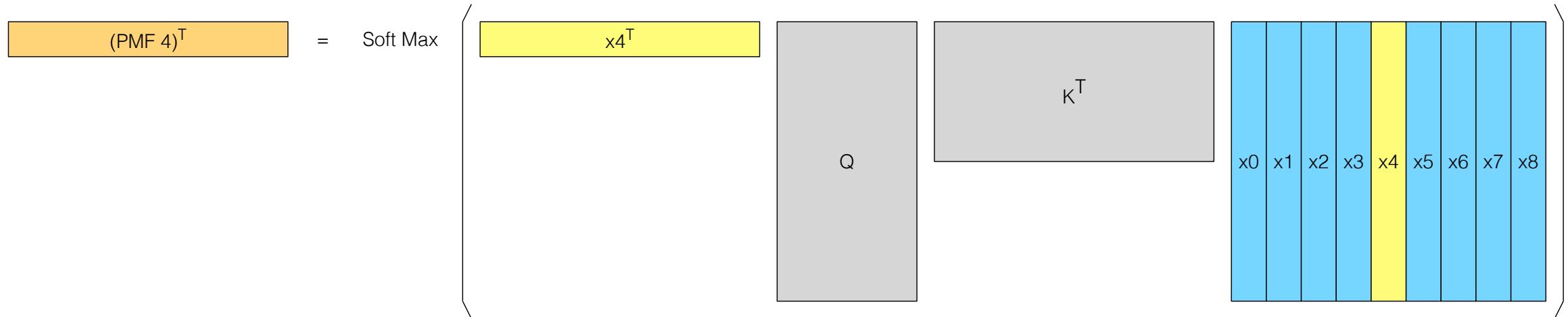
A single output vector is shown; the process is repeated for all locations in the input feature maps to generate full output feature maps



Example given for a $C \times 3 \times 3$ neighborhood in a larger input feature maps; arbitrary size and shape neighborhoods are possible

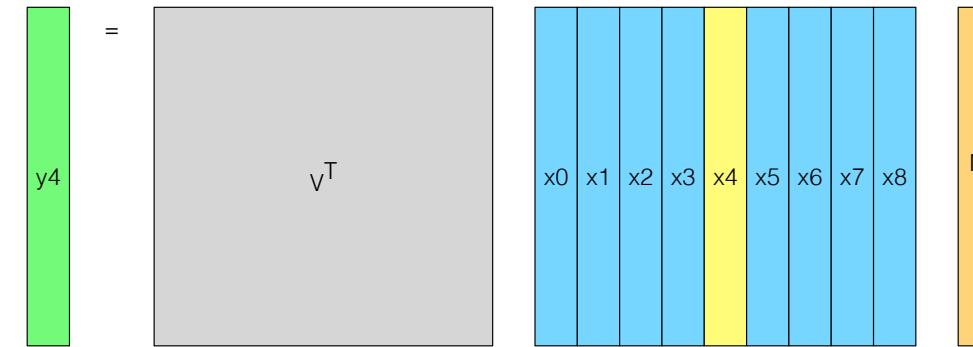
A: Create A PMF For Mixing Across Vectors

Reminder: $\mathbf{a}^T \mathbf{b} = \langle \mathbf{a}, \mathbf{b} \rangle = \sum a_i b_i = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos \Theta_{ab}$ ← is dependent on how similar is the direction of the 2 vectors

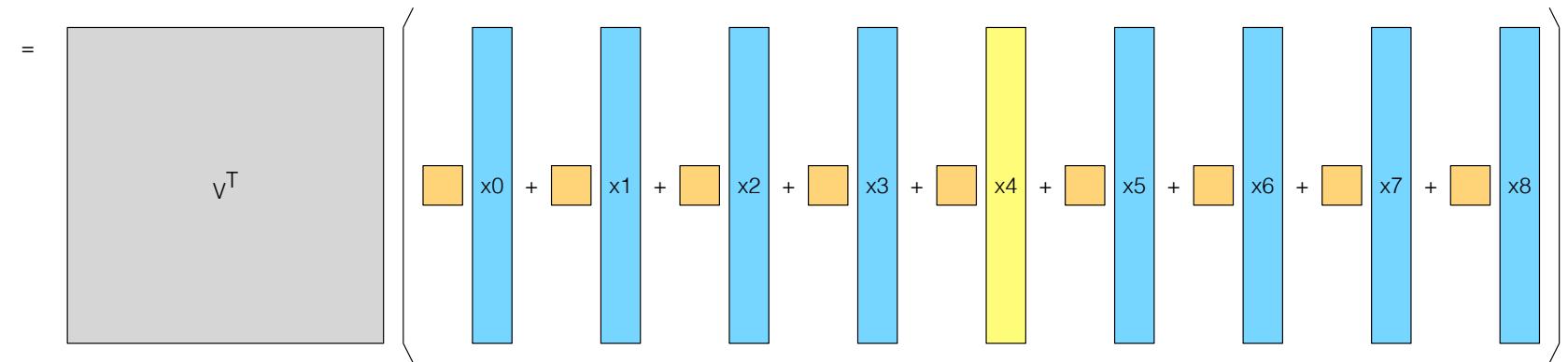


B: Mix Across And Within Feature Vectors

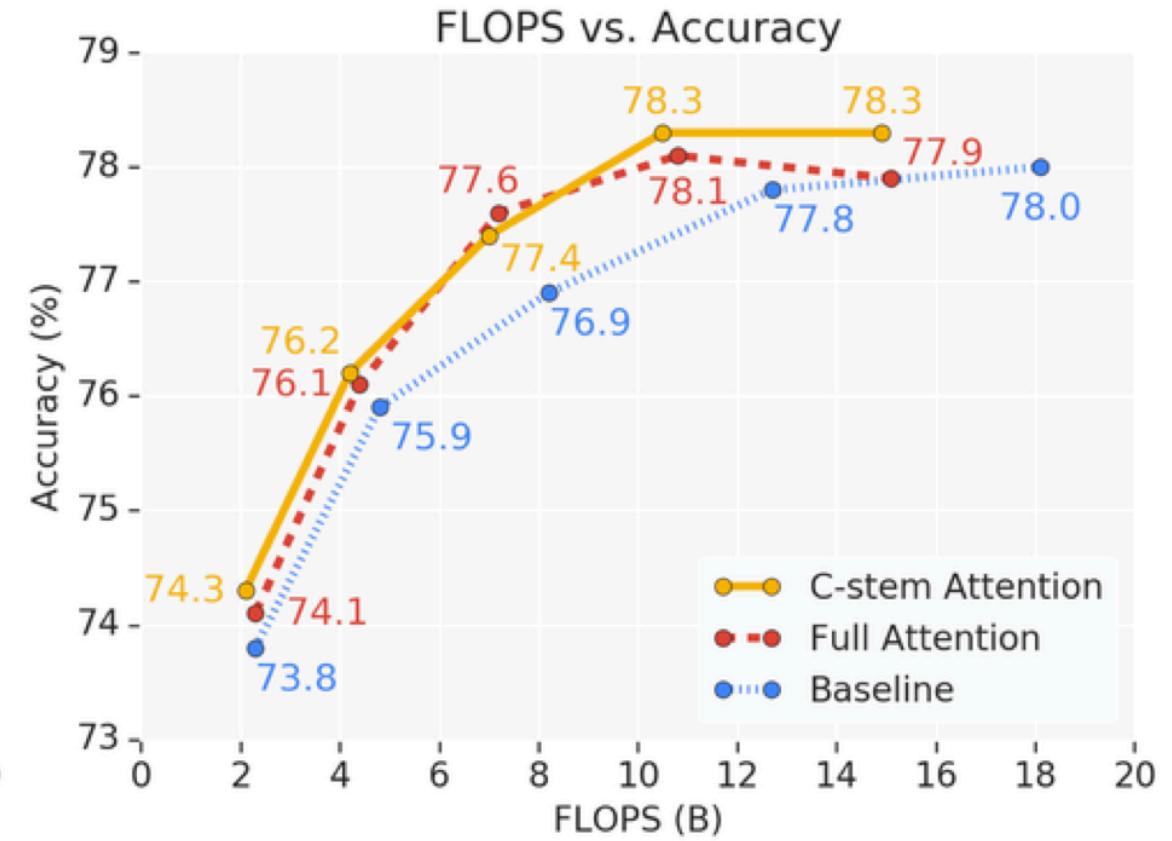
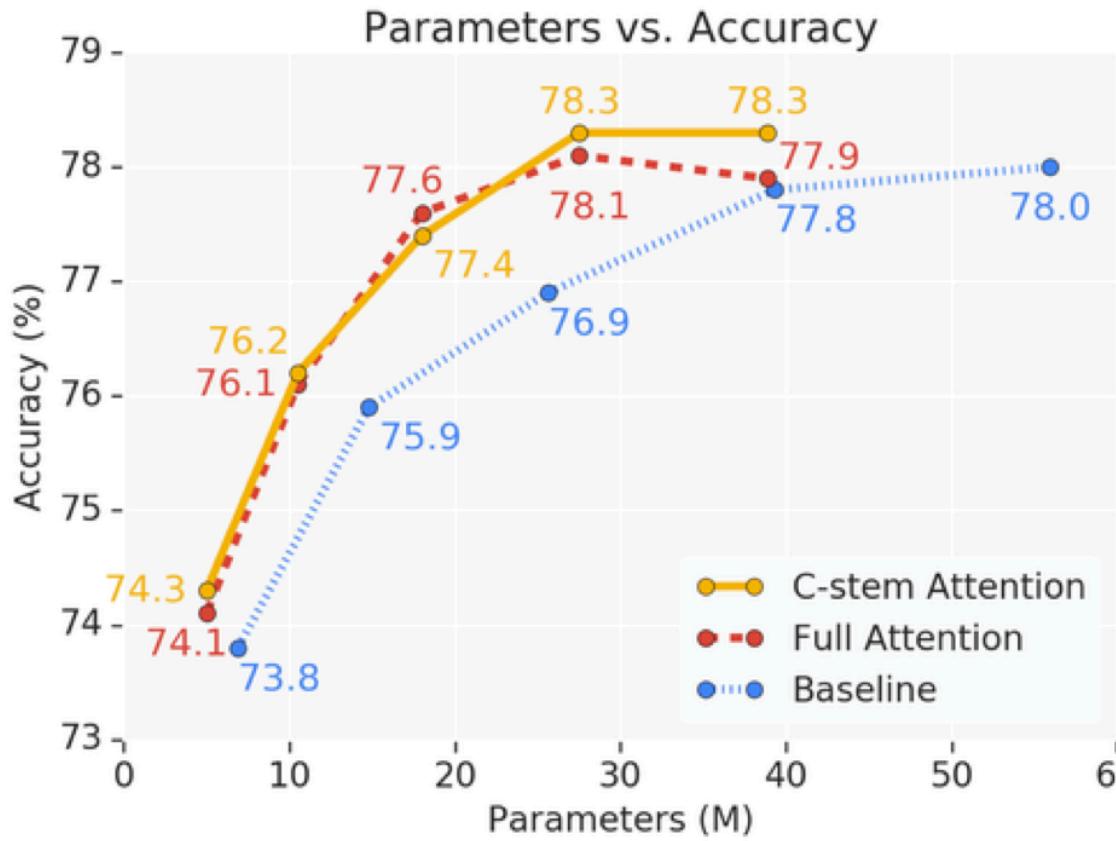
- Attention based inter vector mixing is effectively a data dependent spatial mixing



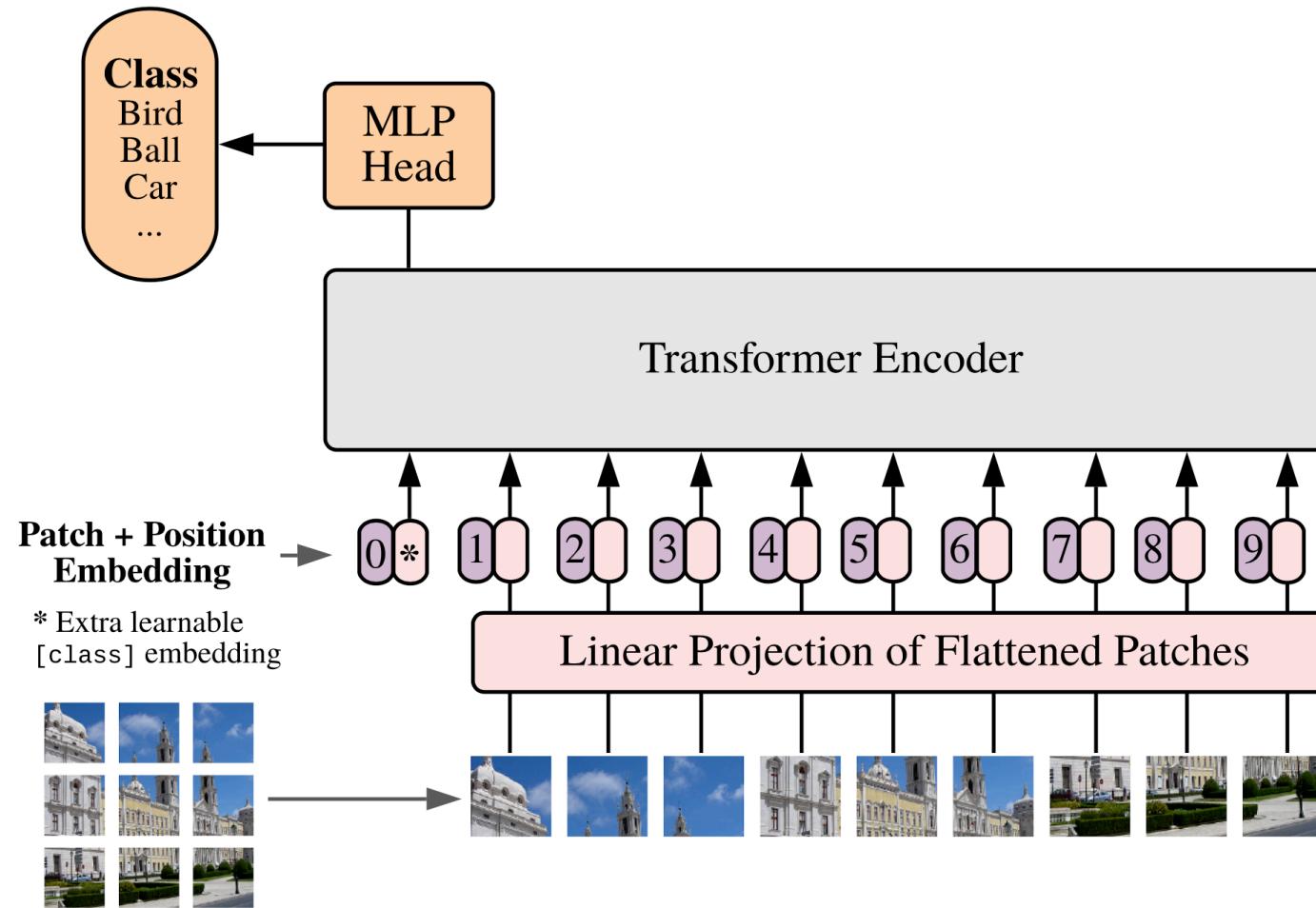
- V matrix intra vector mixing is effectively a data independent channel mixing



Results

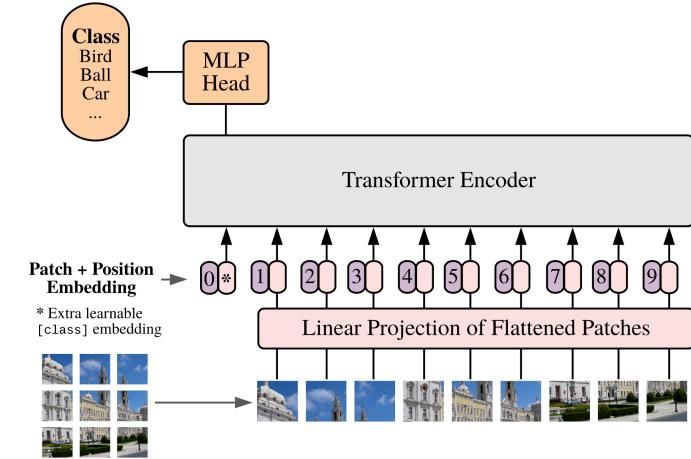


2: Image Patch Based Self Attention



Embedding

- Definitions
 - Classification token $\mathbf{p}_{\text{class}}^T$ is of size $1 \times d_m$
 - Patches $\mathbf{P}_1, \dots, \mathbf{P}_{N-1}$ are each of size $C \times P \times P$ and $N - 1$ of them tile the full original image
 - Patch projection matrix \mathbf{W}_{pro} is of size $P^2C \times d_m$
 - Patch position matrix $\mathbf{W}_{\text{pos}}^T$ is of size $N \times d_m$



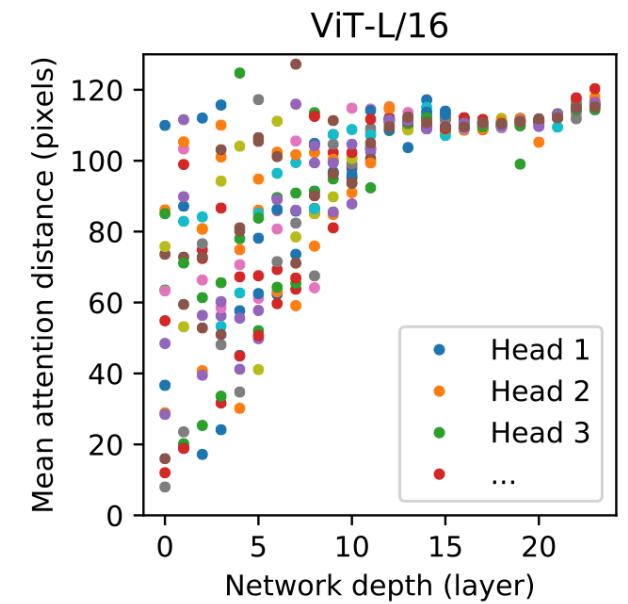
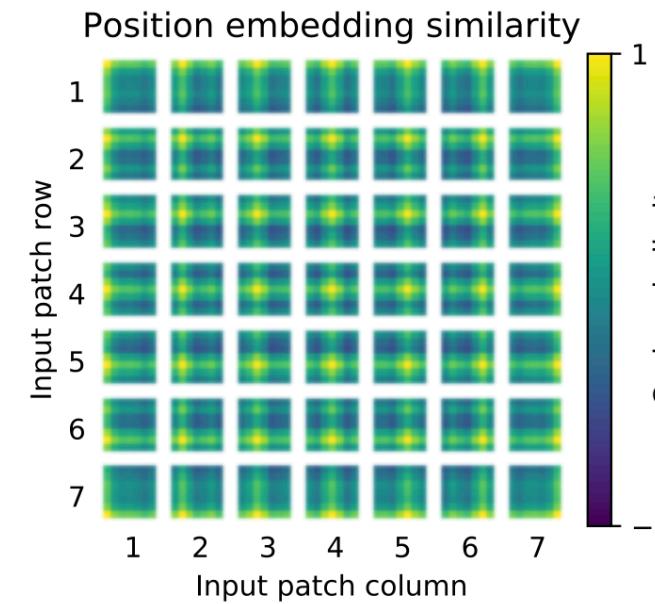
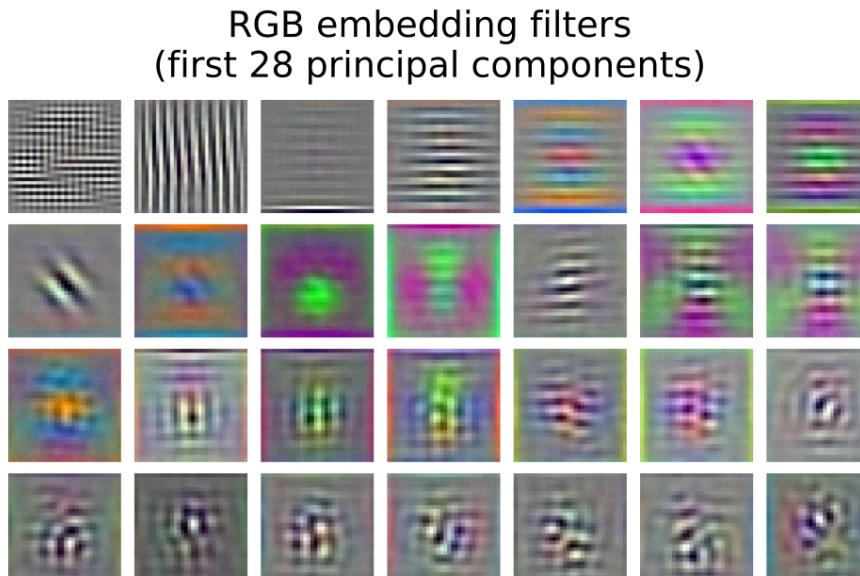
- Image embedding into a sequence of vectors stacked on top of each other to form a matrix

$$\mathbf{X}_0^T = [\mathbf{p}_{\text{class}}^T \quad | \quad \text{vec}(\mathbf{P}_1)^T \mathbf{W}_{\text{pro}} \quad | \quad \dots \quad | \quad \text{vec}(\mathbf{P}_{N-1})^T \mathbf{W}_{\text{pro}}]^T$$

Alternative embedding strategy

- Pass the input image through the encoder portion of a CNN to generate feature maps of size $d_m \times H \times W$
- Each of the HW vectors of size $d_m \times 1$ becomes a row of $\mathbf{X}_0^T - \mathbf{W}_{\text{pos}}^T$

Embedding And Effective Receptive Field

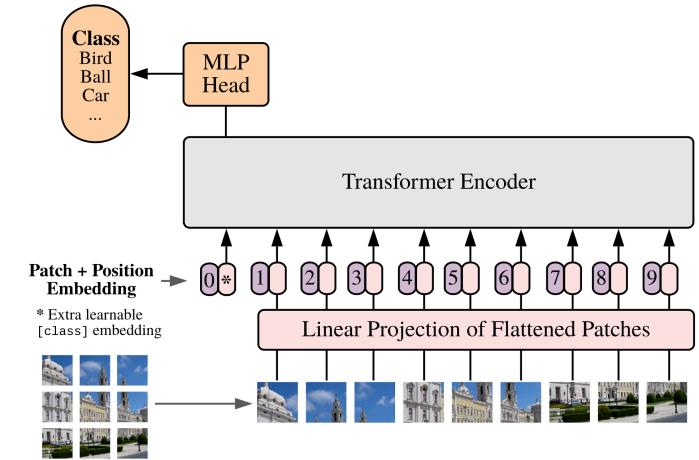


Decoder / Classifier

- The 1st output vector corresponding to the classifier token position is used as the input to the classifier
 - No dependence on the input sequence length
- Classifier
 - (Repeat 2x) {layer normalization, linear layer and dropout}

$$\mathbf{X}_{L+1}^T = d_? (d (f_{\text{GELU}} (g (\mathbf{X}_L(:, 0)^T) \mathbf{H}_{L+1,A} + \mathbf{b}_{L+1,A}^T)) \mathbf{H}_{L+1,B} + \mathbf{b}_{L+1,B}^T)$$

dropout dropout nonlinearity norm 1 x dm dm x df 1 x df df x classes 1 x classes
 $\mathbf{X}_L(:, 0)$ $\mathbf{H}_{L+1,A}$ $\mathbf{b}_{L+1,A}^T$ $\mathbf{H}_{L+1,B}$ $\mathbf{b}_{L+1,B}^T$



Results

	Ours (ViT-H/14)	Ours (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.36	87.61 ± 0.03	87.54 ± 0.02	88.4/ 88.5 *
ImageNet ReAL	90.77	90.24 ± 0.03	90.54	90.55
CIFAR-10	99.50 ± 0.06	99.42 ± 0.03	99.37 ± 0.06	—
CIFAR-100	94.55 ± 0.04	93.90 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	97.56 ± 0.03	97.32 ± 0.11	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	99.74 ± 0.00	99.63 ± 0.03	—
VTAB (19 tasks)	77.16 ± 0.29	75.91 ± 0.18	76.29 ± 1.70	—
TPUv3-days	2.5k	0.68k	9.9k	12.3k

Table 2: Comparison with state of the art on popular image classification datasets benchmarks. Vision Transformer models pre-trained on the JFT300M dataset often match or outperform ResNet-based baselines while taking substantially less computational resources to pre-train. *Slightly improved 88.5% result reported in Touvron et al. (2020).

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

More 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>

A Final Note

- 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

Backup

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 Attention design, training, implementation and language
 - B.3 RNN design, training, implementation and speech
- 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

Example: Crossing A Busy Street

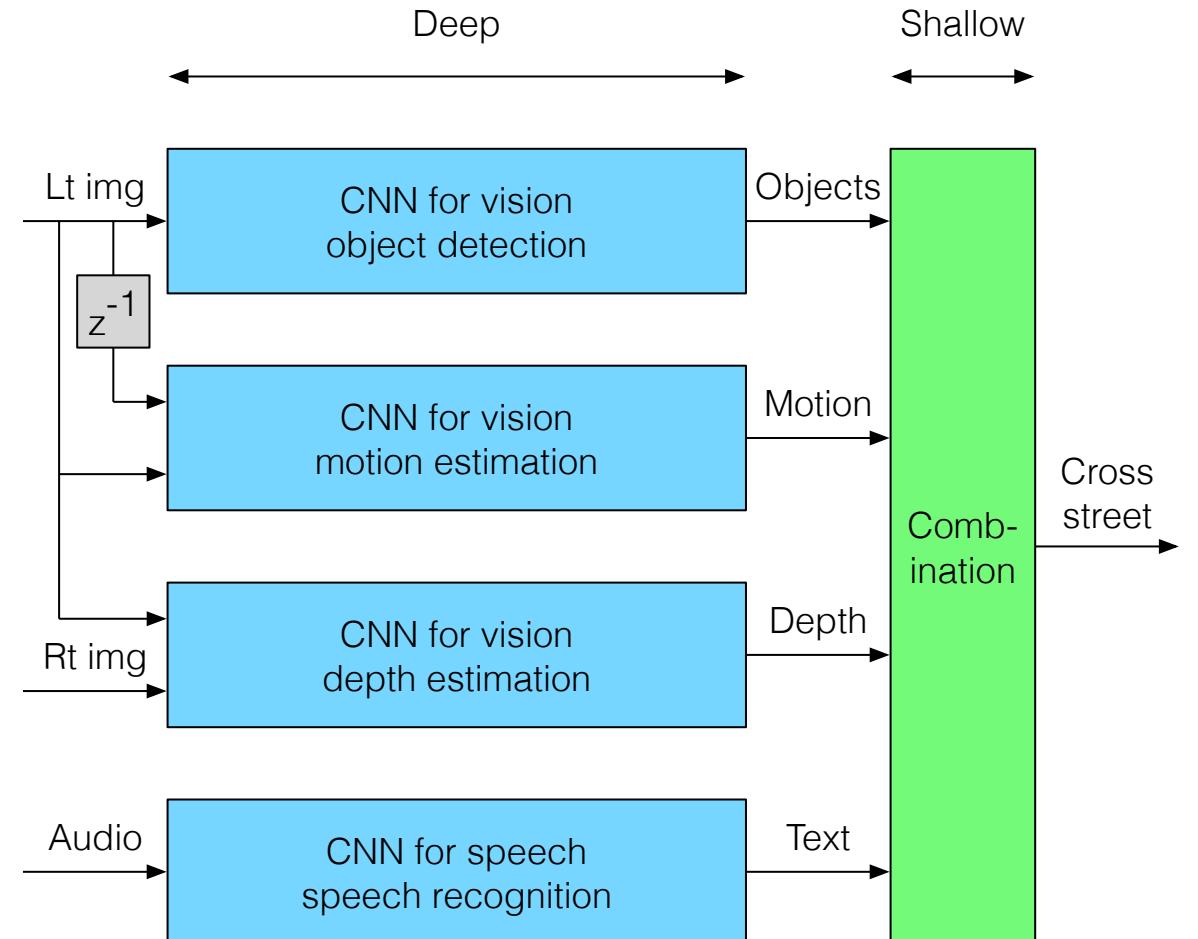
- Goal: get from 1 side of the street to the other side in a safe, efficient and law abiding way
- How do you solve this problem as a human? An incomplete list includes ...
 - Pre processing look up from mobile phone
 - Vision recognition of many objects in the scene, approximate distances, approximate 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

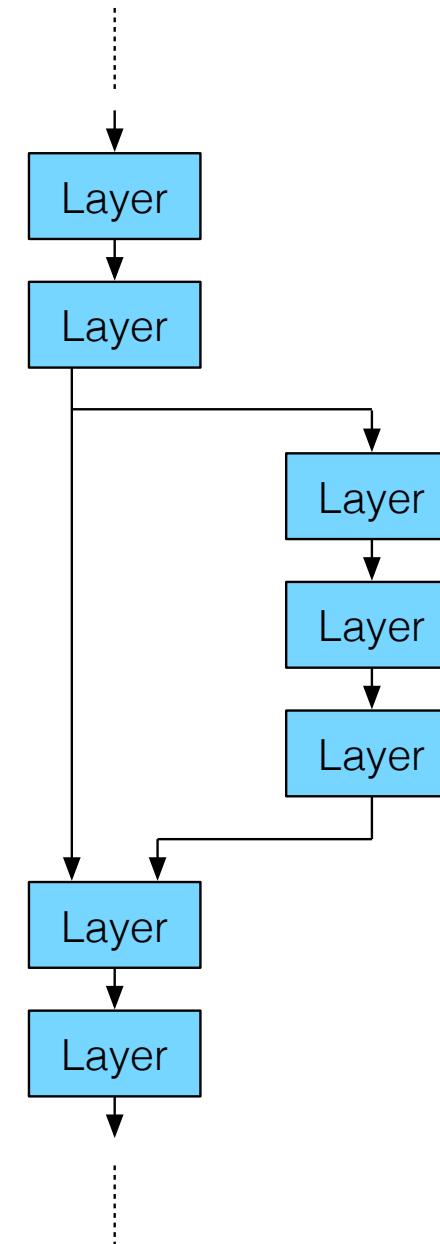
Example: Crossing A Busy Street

- 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



Graph Specification

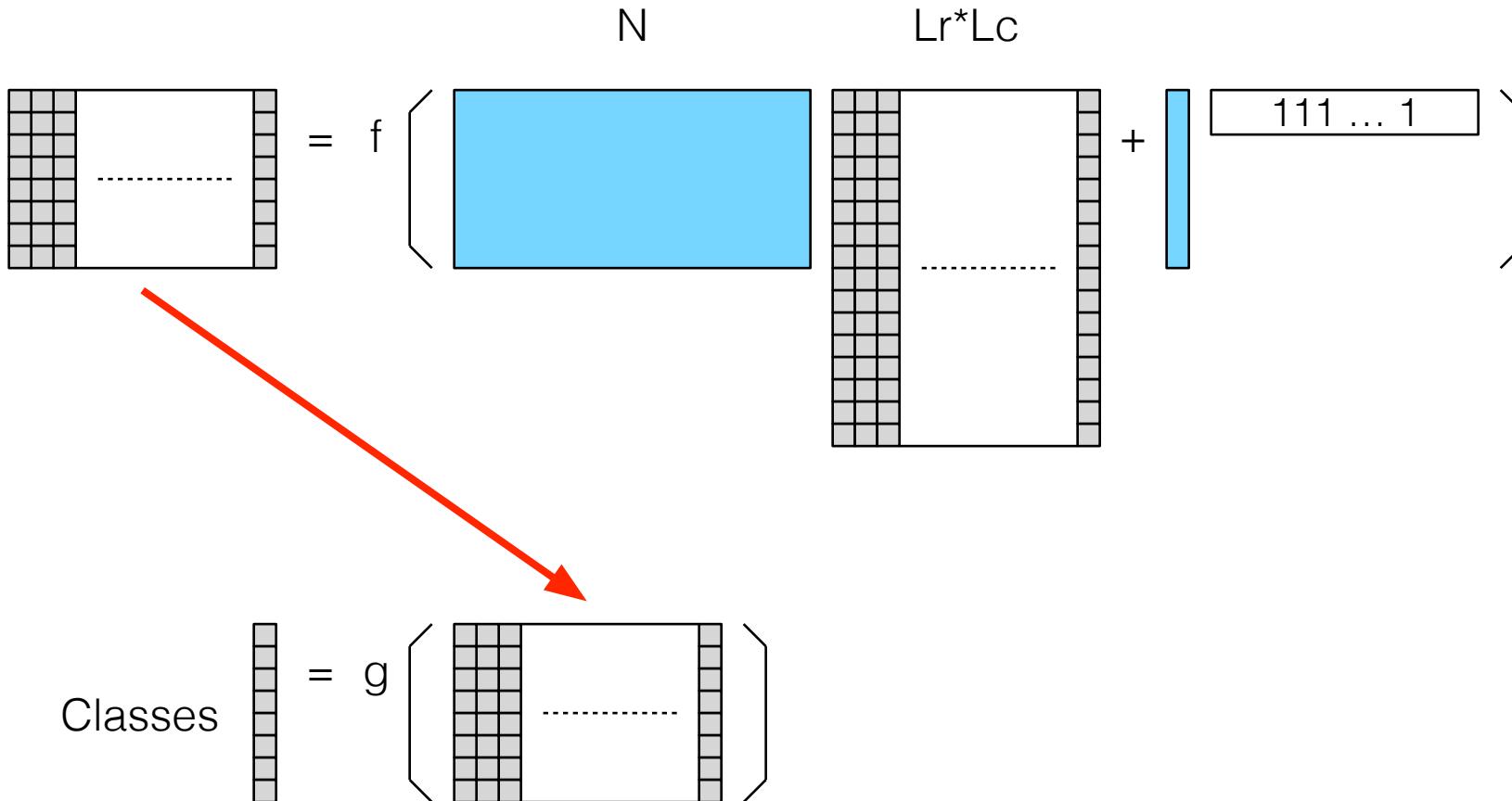
- Implicitly or explicitly, computational graphs are used for high level network descriptions (other algorithms too)
- Graph edges are memory / dependencies and nodes are compute
 - Not shown are practicalities that we'll care about during the implementation including feature map location and data movement, 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)



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

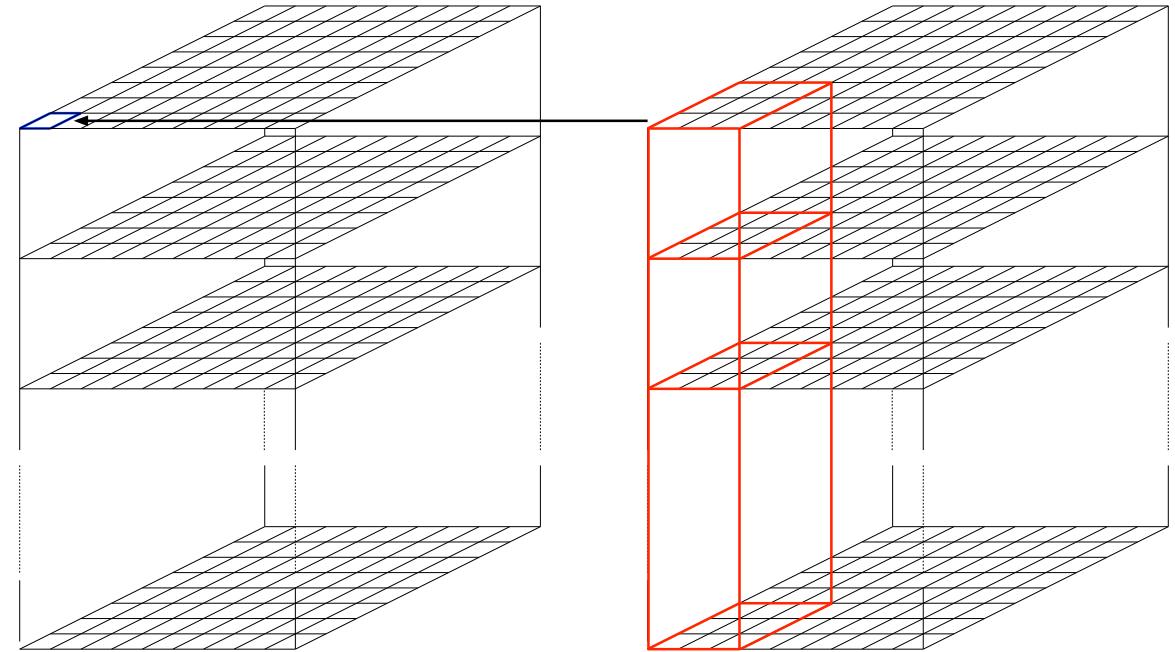


Layer Overview (We've Already Seen These)

- Layers map input feature maps to output 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

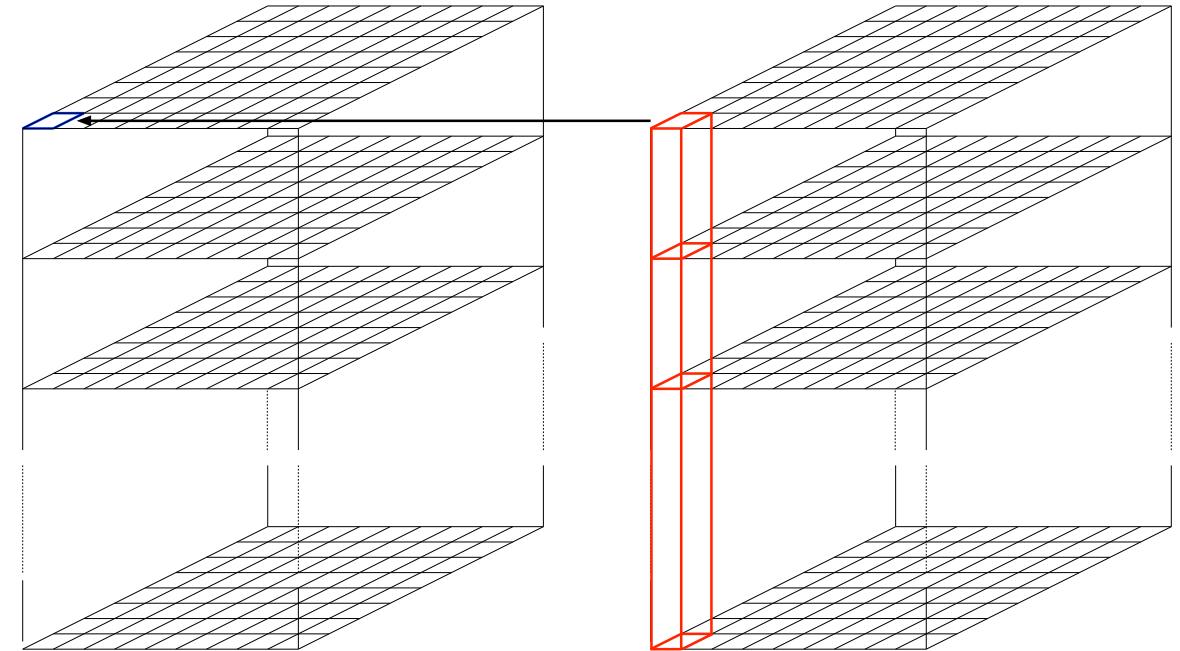
- CNN style 2D convolution layer
 - Filter
 - Tensor up sampling
 - Matrix creation (nothing to do)
 - Input feature map
 - Tensor up sampling
 - Tensor 0 padding
 - Matrix create (Toeplitz style)
 - Matrix reduction from filter up sampling
 - Remove cols from filter matrix from up sampling
 - Remove 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

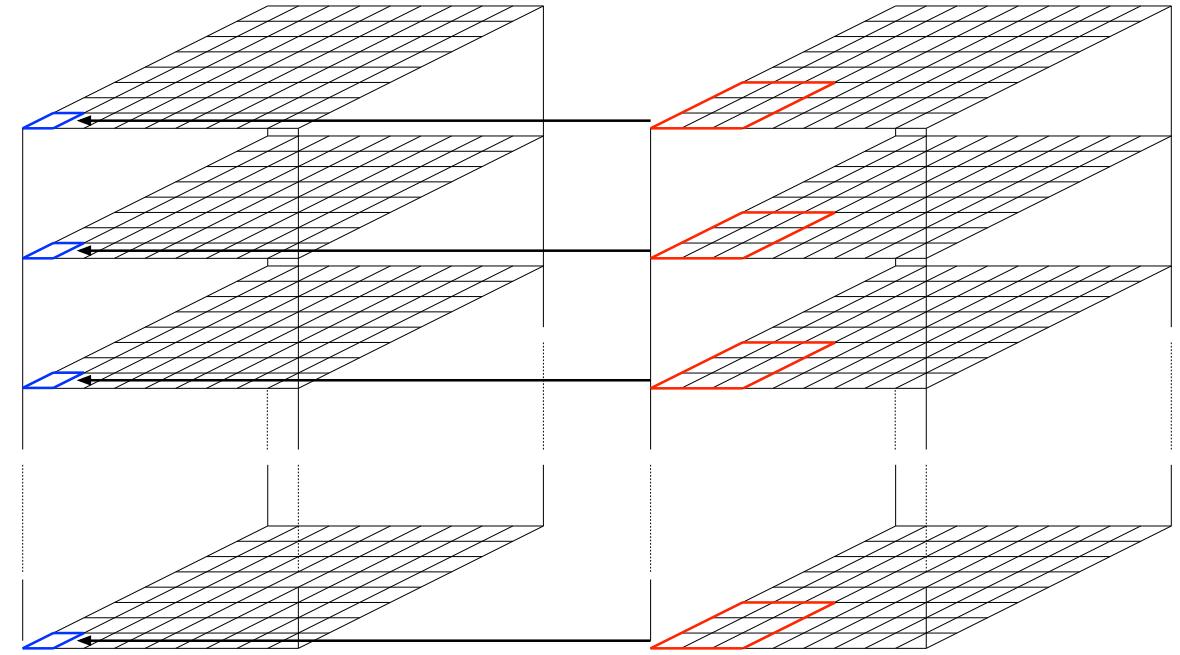
Channel

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



Space

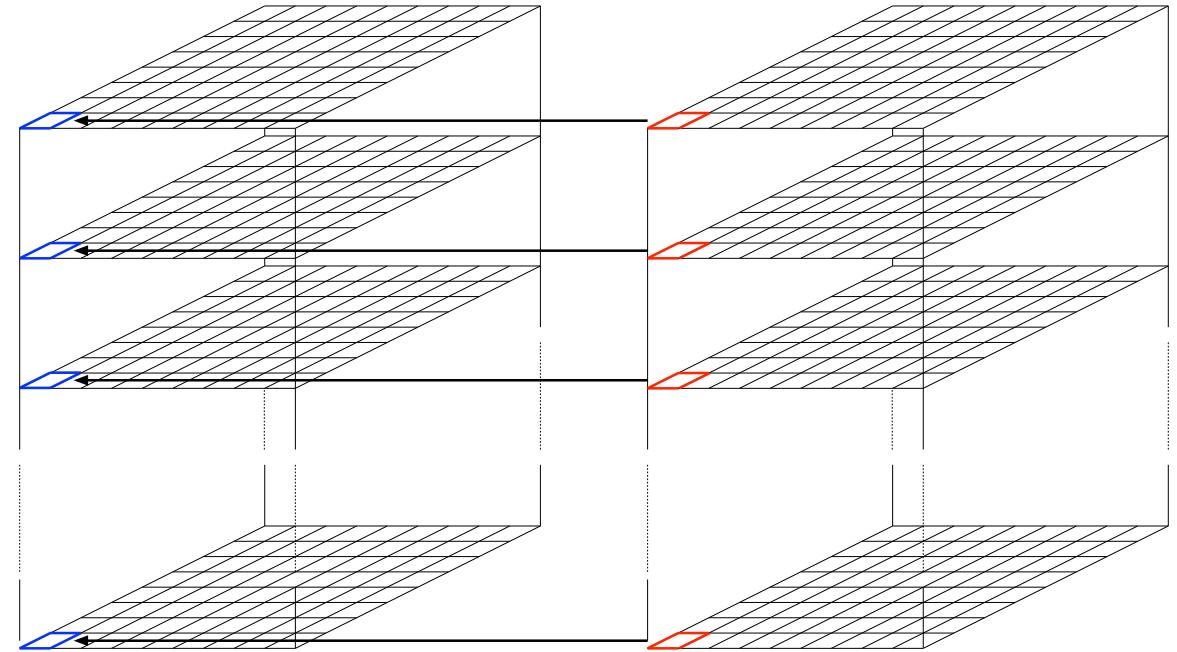
- CNN style 2D convolution that's fully grouped (standard 2D convolution, depth wise separable convolution)
- Decimation
 - Max pooling
 - Avg pooling
 - Global average pooling
 - Spatial pyramid pooling
- Interpolation
 - Nearest neighbor
 - Bilinear



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)

Element

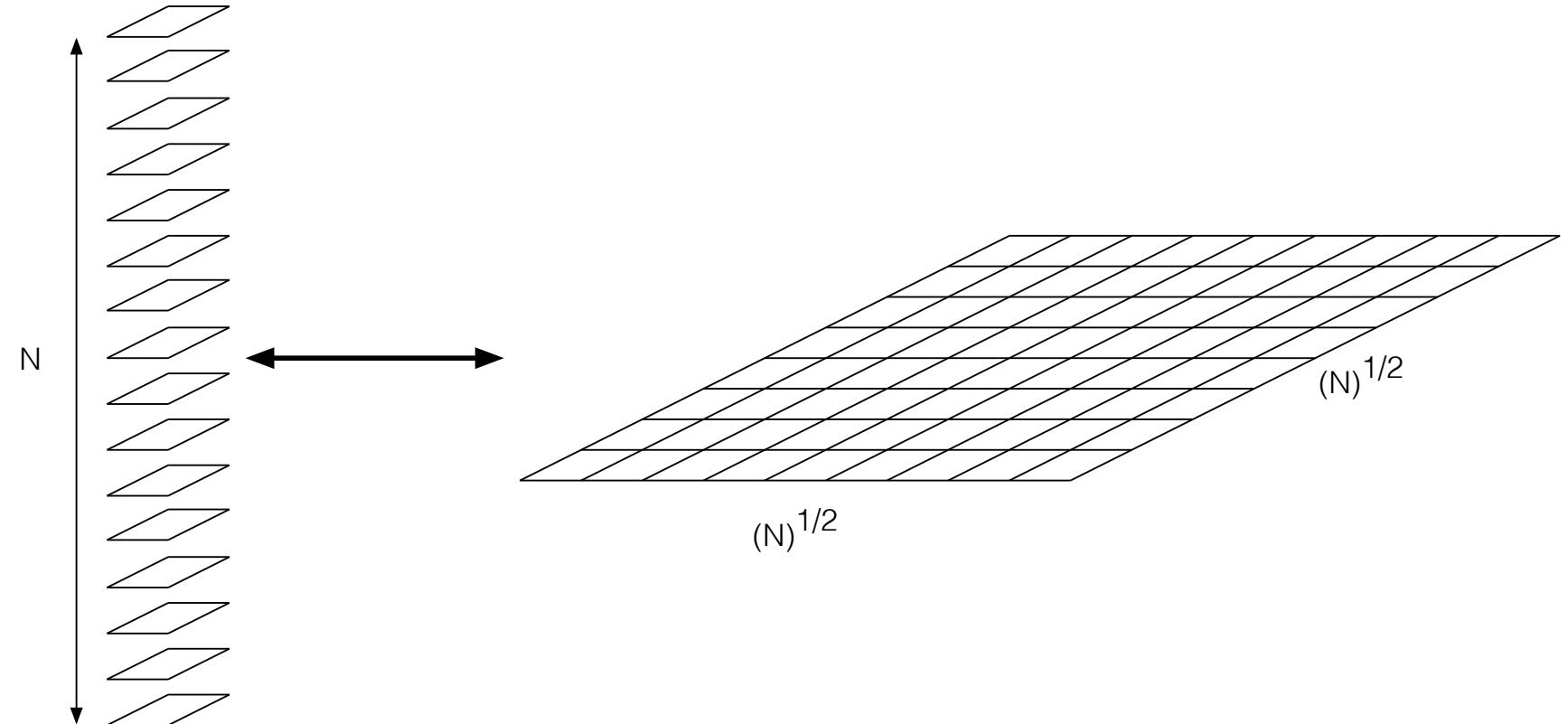
- Nonlinearities
 - ReLU, ReLU6, PReLU, ReLU with a small negative slope, sigmoid, tanh, ...
- 1 input 1 output linear
 - Scale
 - Offset
- 2 input 1 output math
 - Add, multiply, max, ...



For a survey paper on different nonlinearities see Activation functions: comparison of trends in practice and research for deep learning (<https://arxiv.org/abs/1811.03378>)

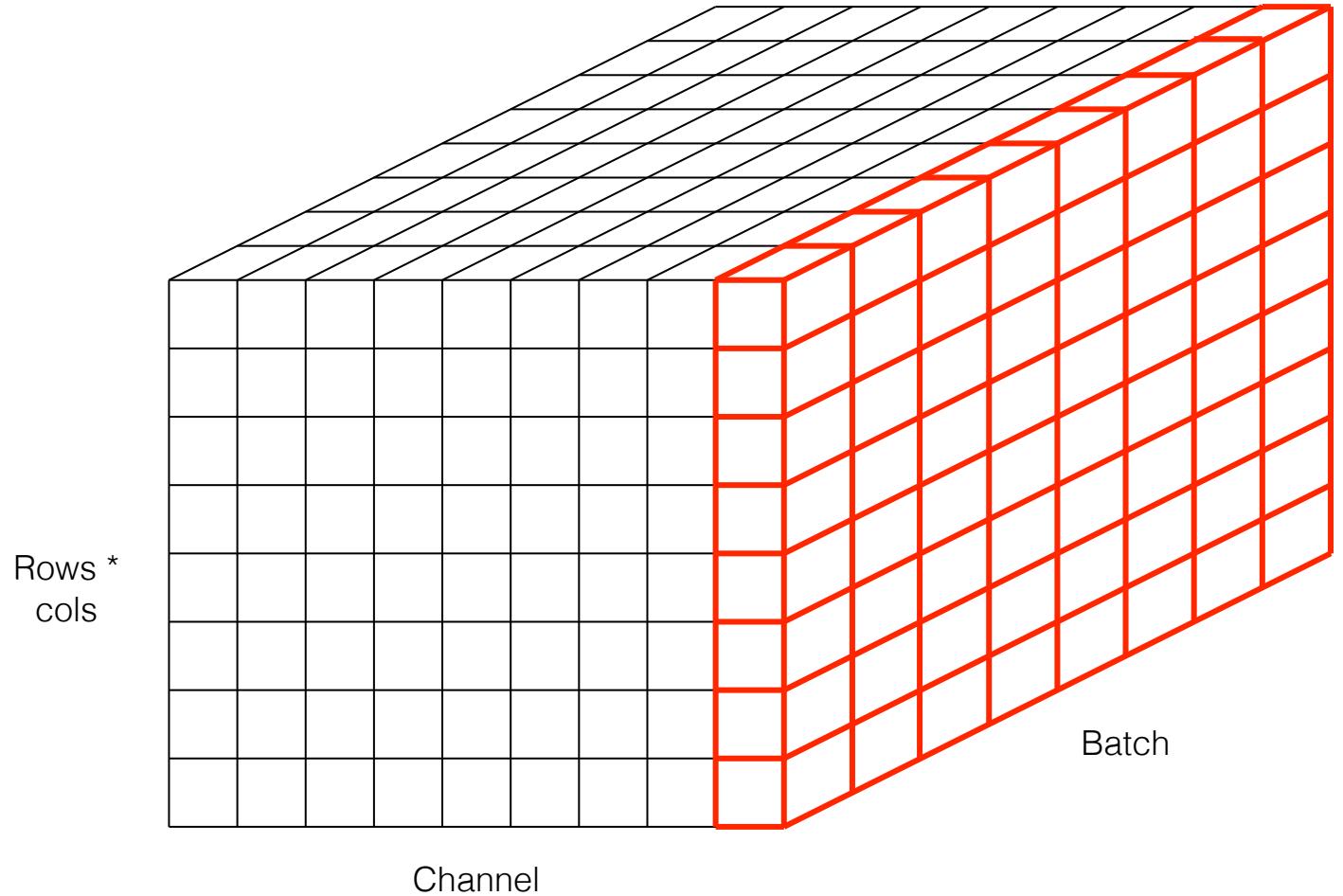
Rearrangement

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

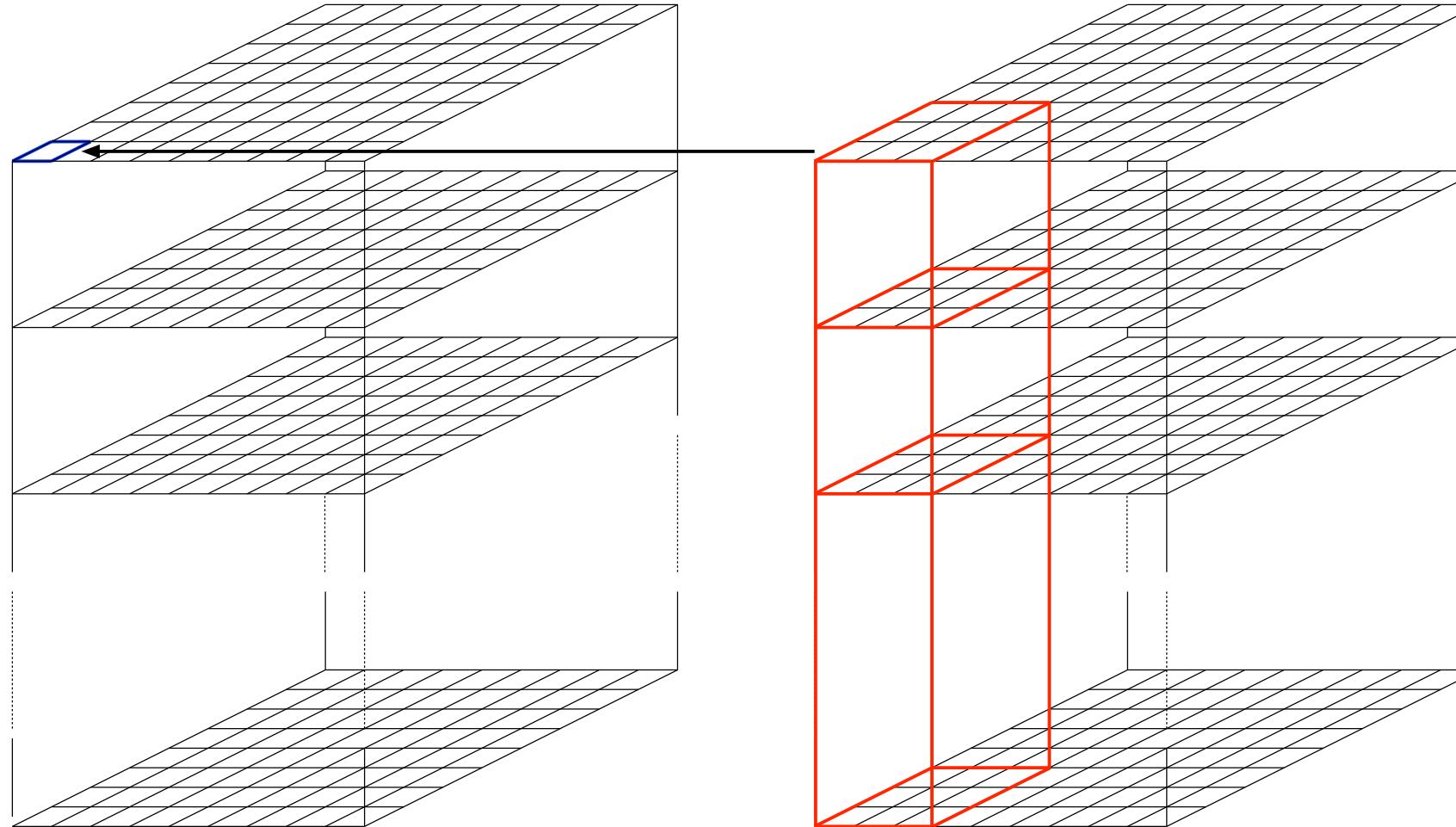


Training

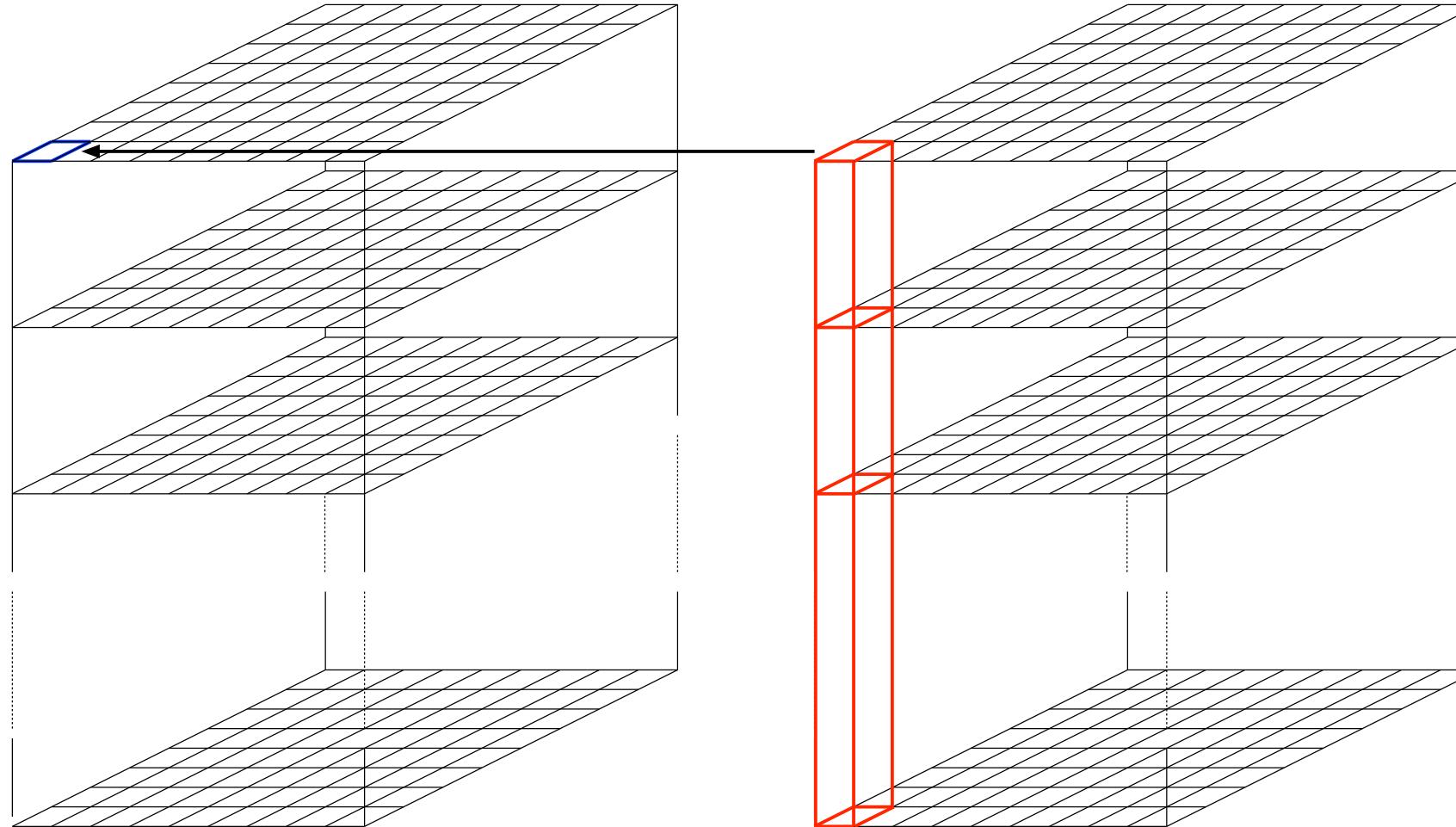
- Will come back to these in the context of training
- Normalization
 - Batch
 - Group
- Dropout
- Loss
 - L_p
 - Softmax + cross entropy



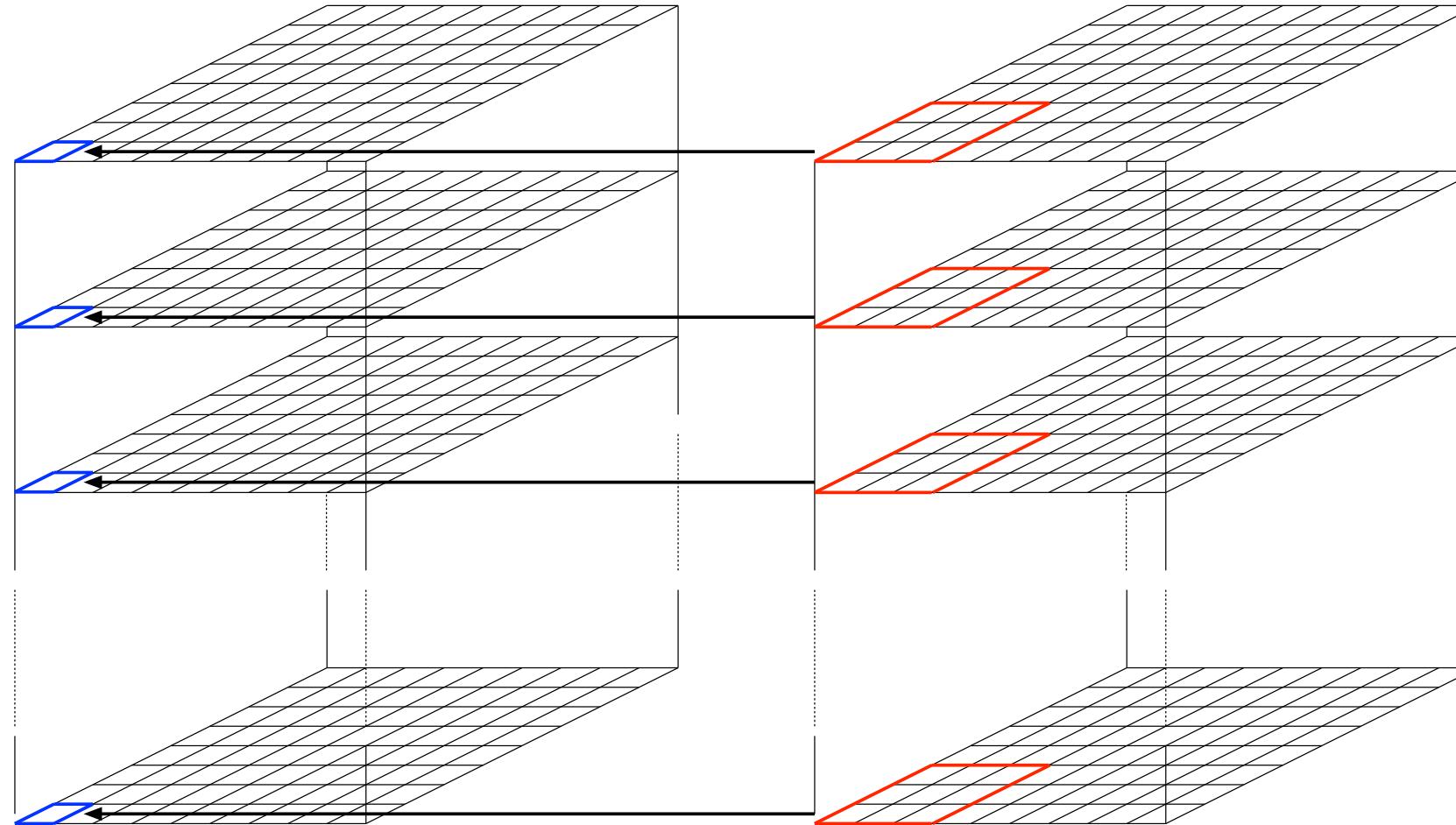
Channel And Space



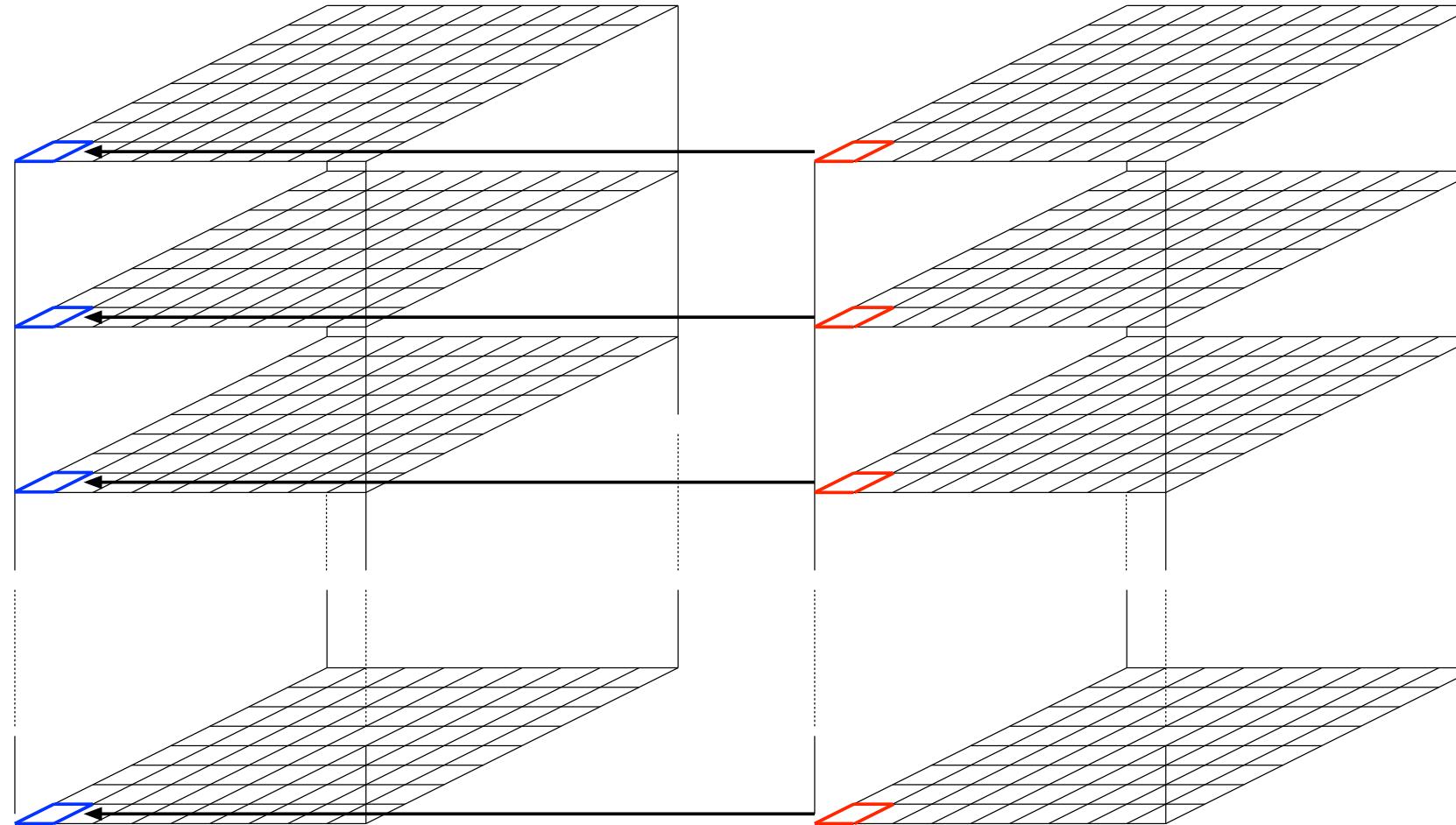
Channel



Space

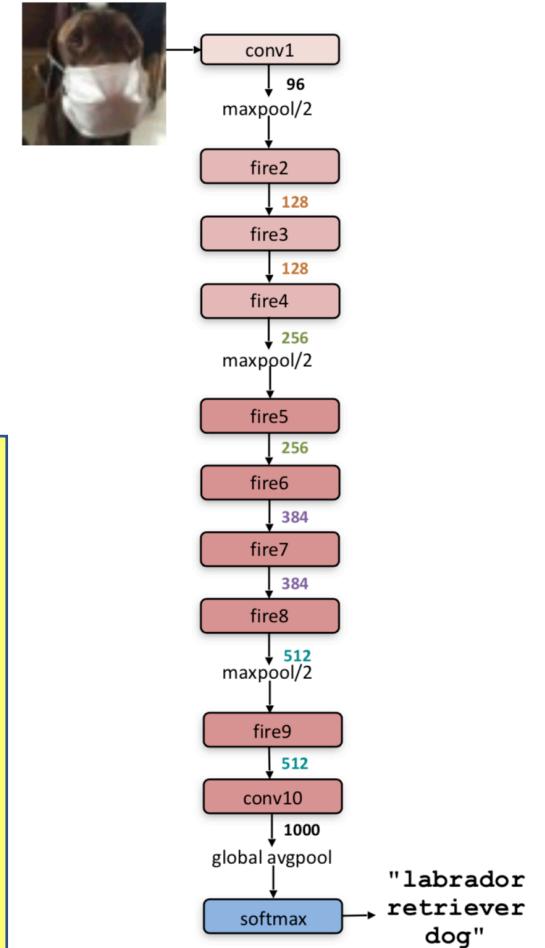
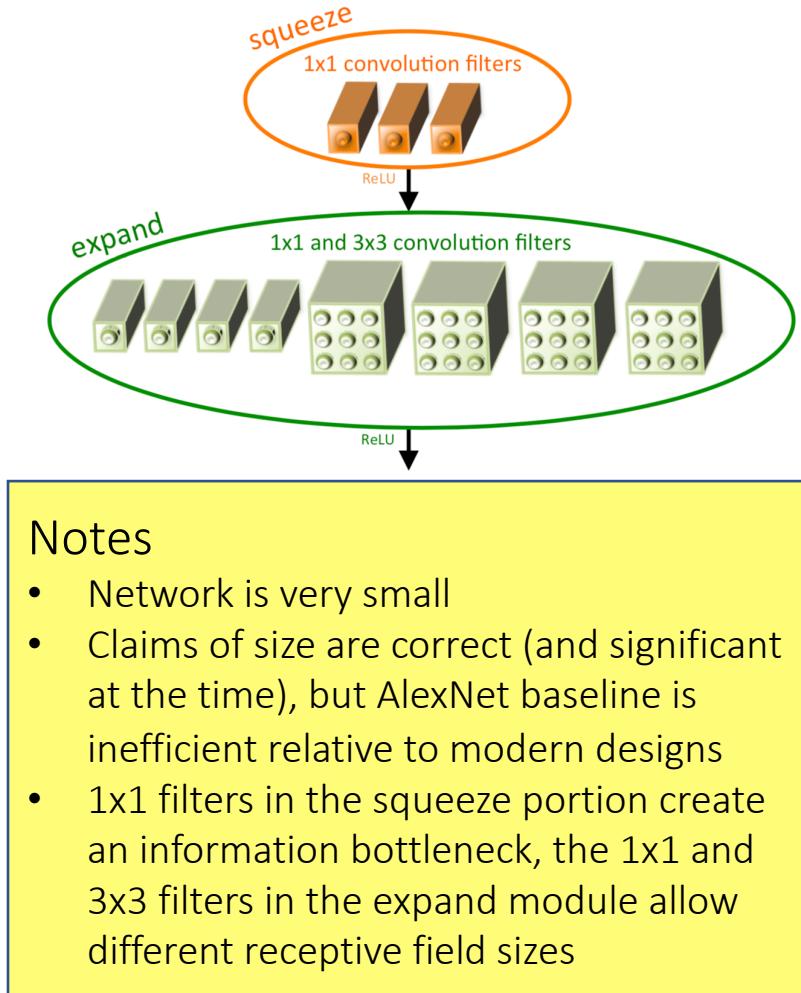


Element



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>



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									
											1,248,424 (total)	421,098 (total)

Question

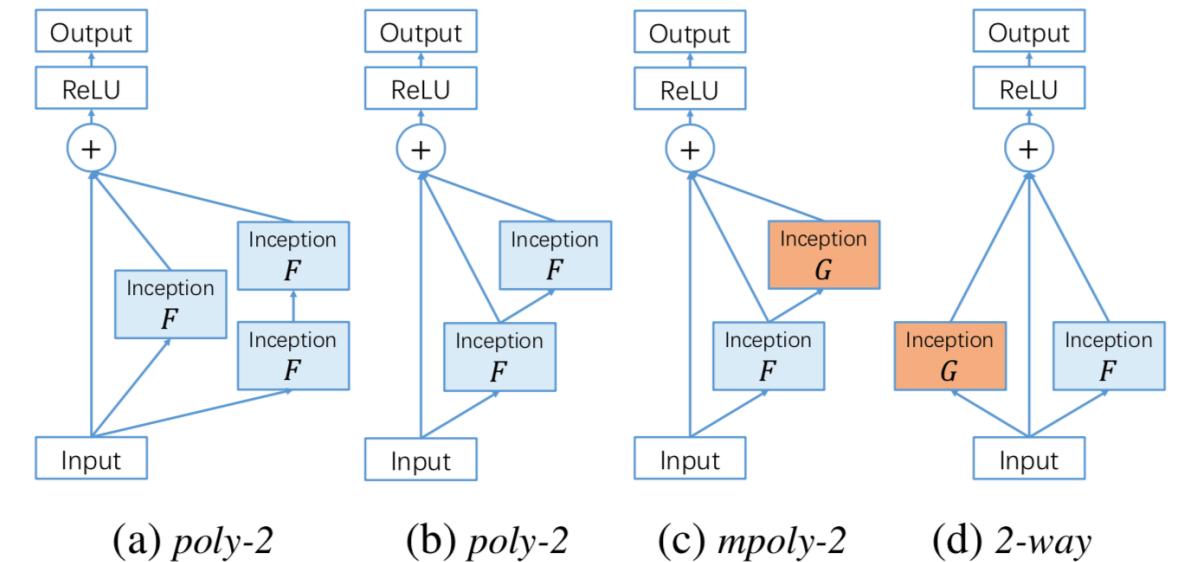
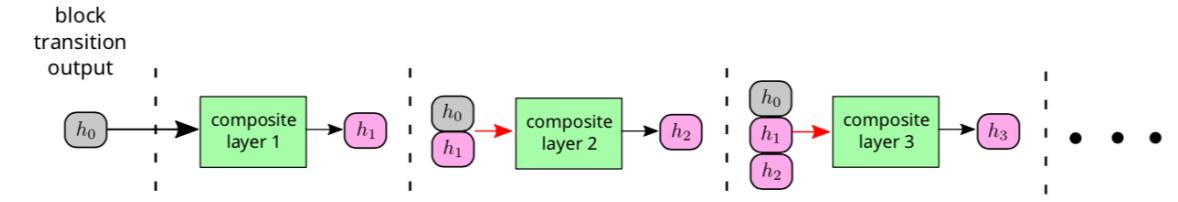
- How small can a network be to solve a problem to a given level of accuracy?

Visualization

- <https://dgschwend.github.io/netscope/#/present/squeezeenet>
- https://dgschwend.github.io/netscope/#/present/squeezeenet_v11

Dense: DelugeNets And PolyNet

- DelugeNets: deep networks with efficient and flexible cross-layer information inflows
 - <https://arxiv.org/abs/1611.05552>
 - Some similarities to DenseNet
- PolyNet: a pursuit of structural diversity in very deep networks
 - <https://arxiv.org/abs/1611.05725>
 - Some similarities to DenseNet combined with Inception ResNet (which we'll see in a bit)

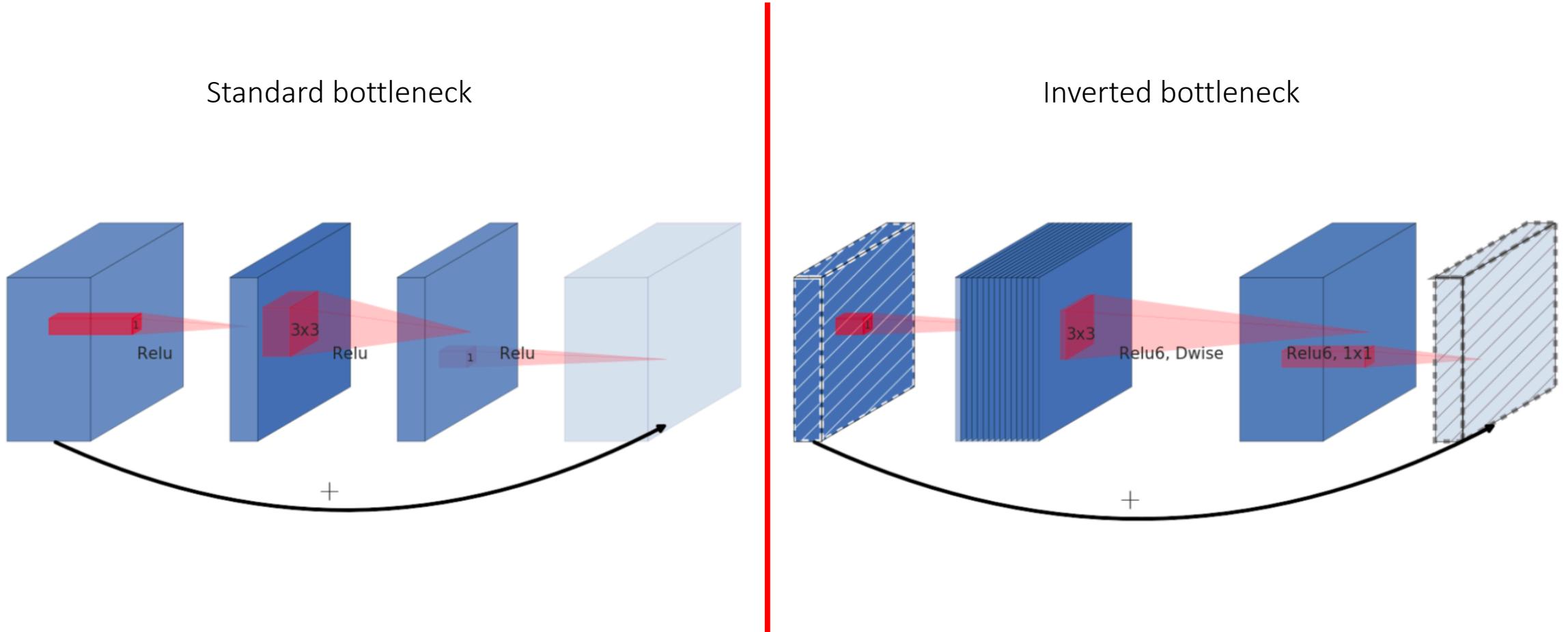


Residual: MobileNet V2

- 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>

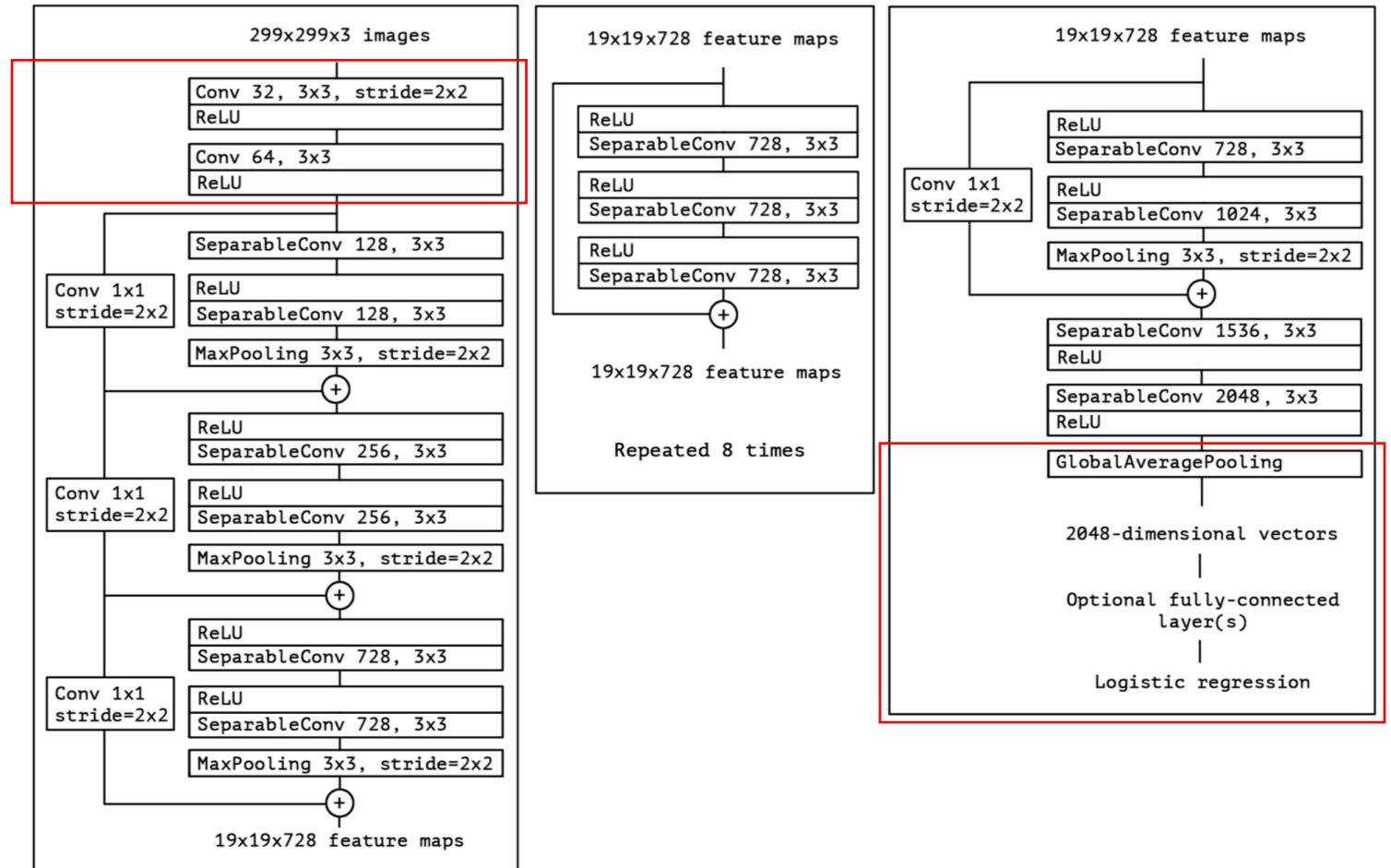
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck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

Residual: MobileNet V2

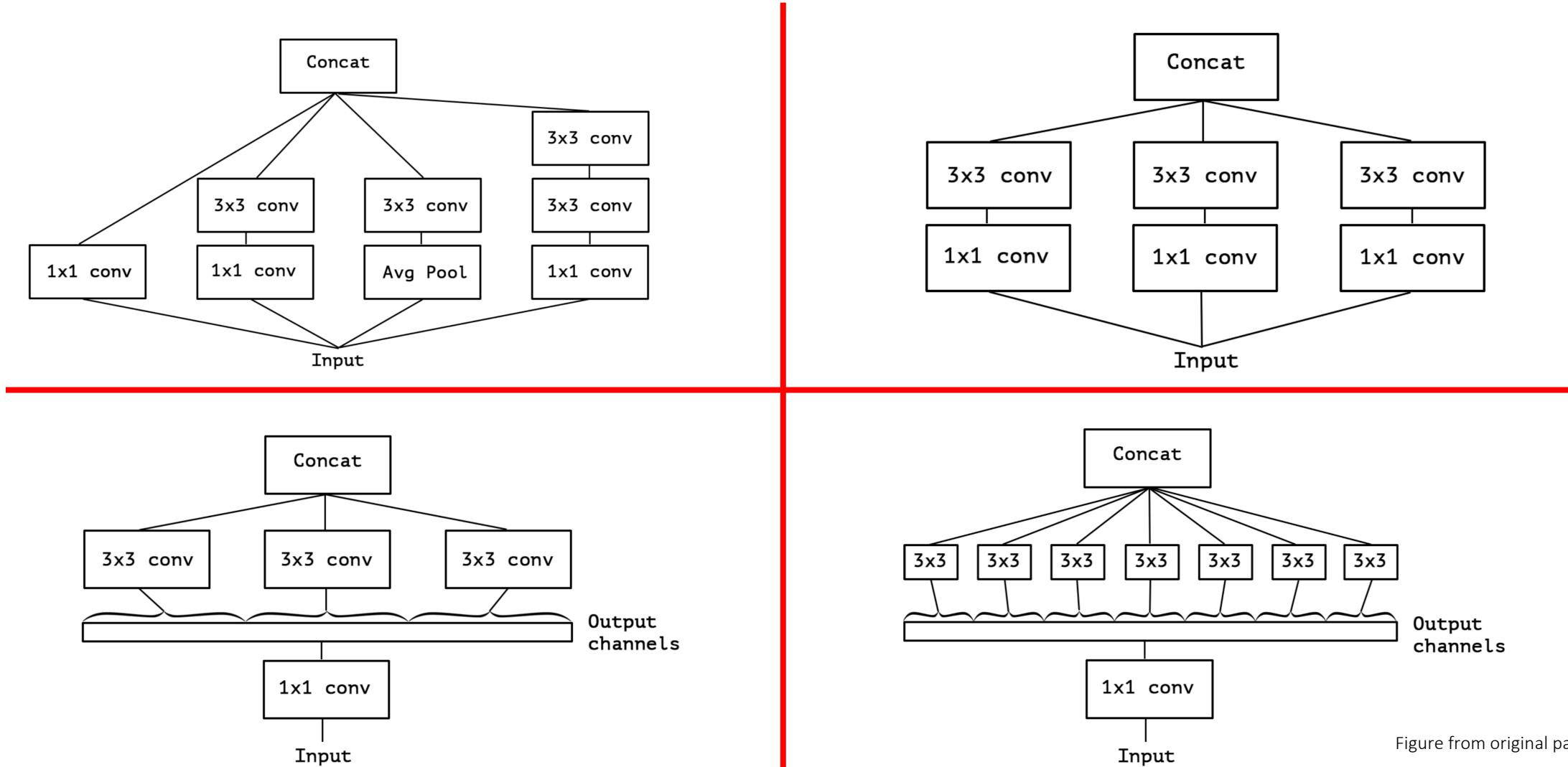


Residual: Exception

- 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>

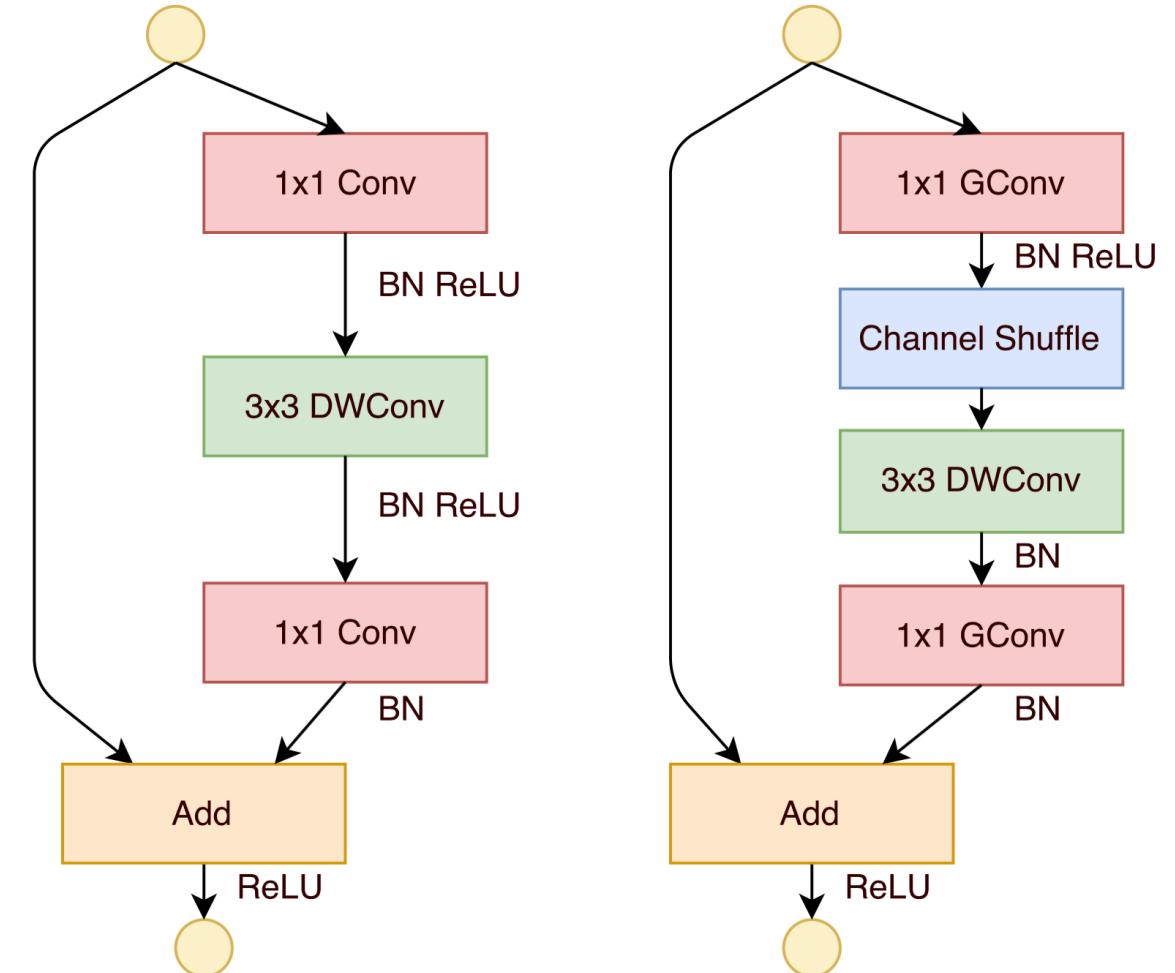


Residual: Exception

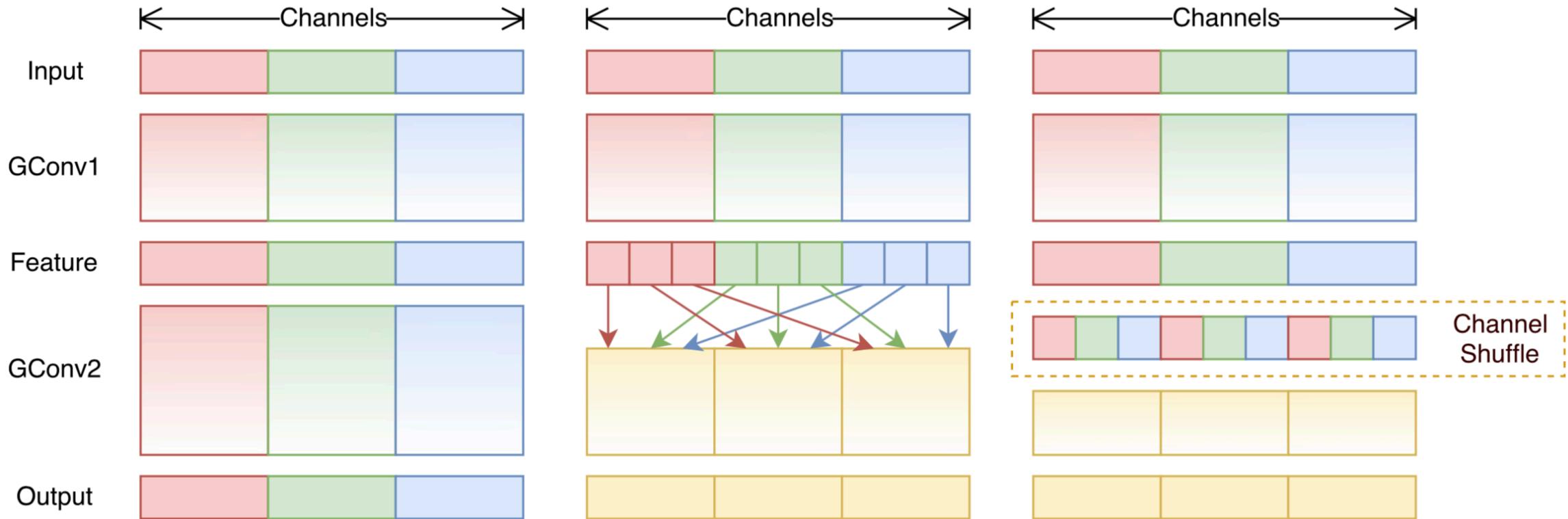


Residual: ShuffleNet V1

- 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>



Residual: ShuffleNet V1

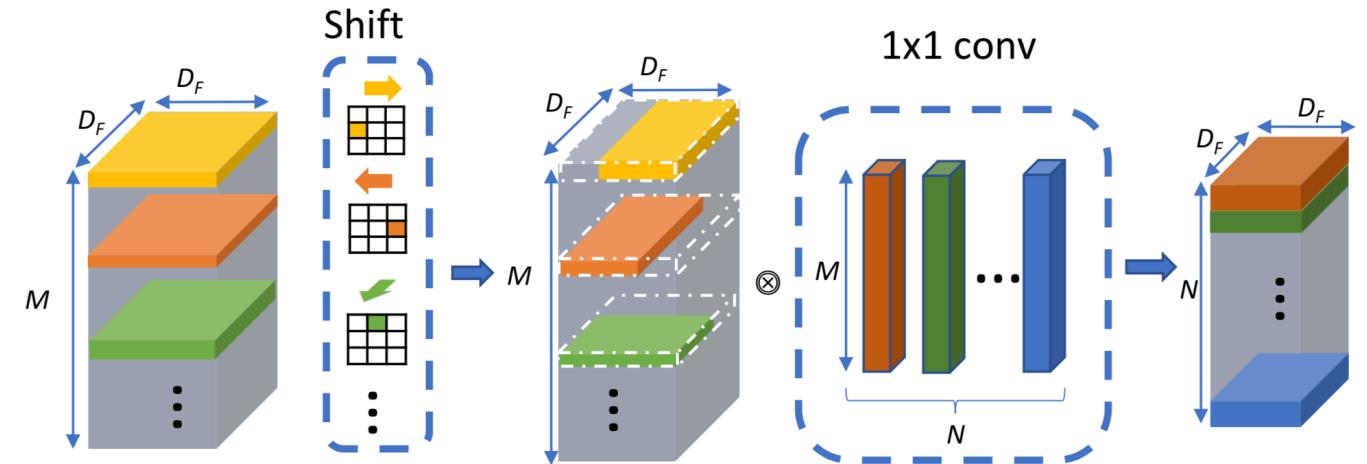


Residual: ShuffleNet V1

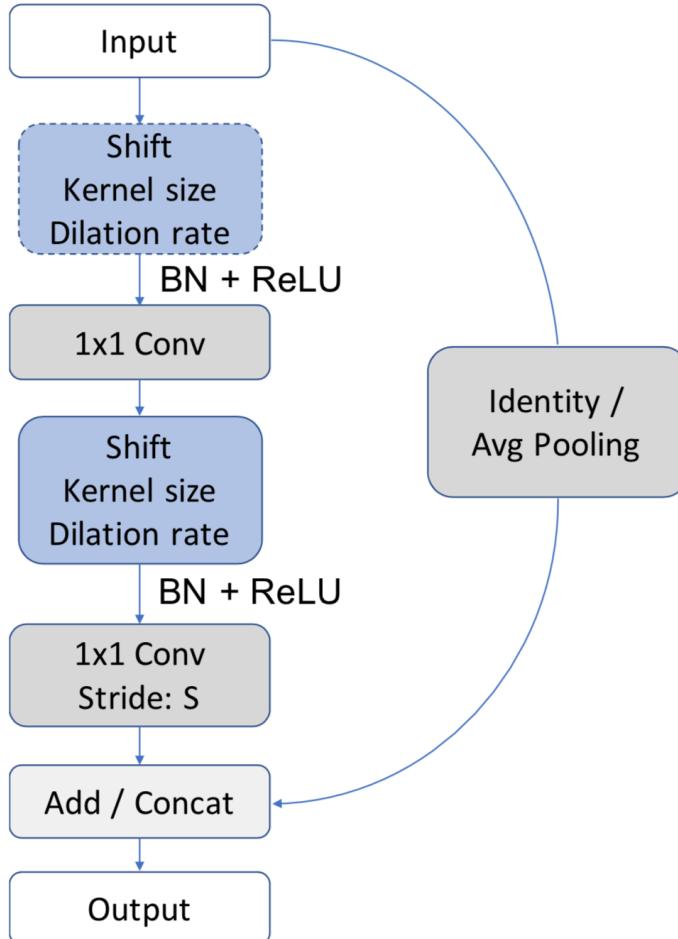
Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	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

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>



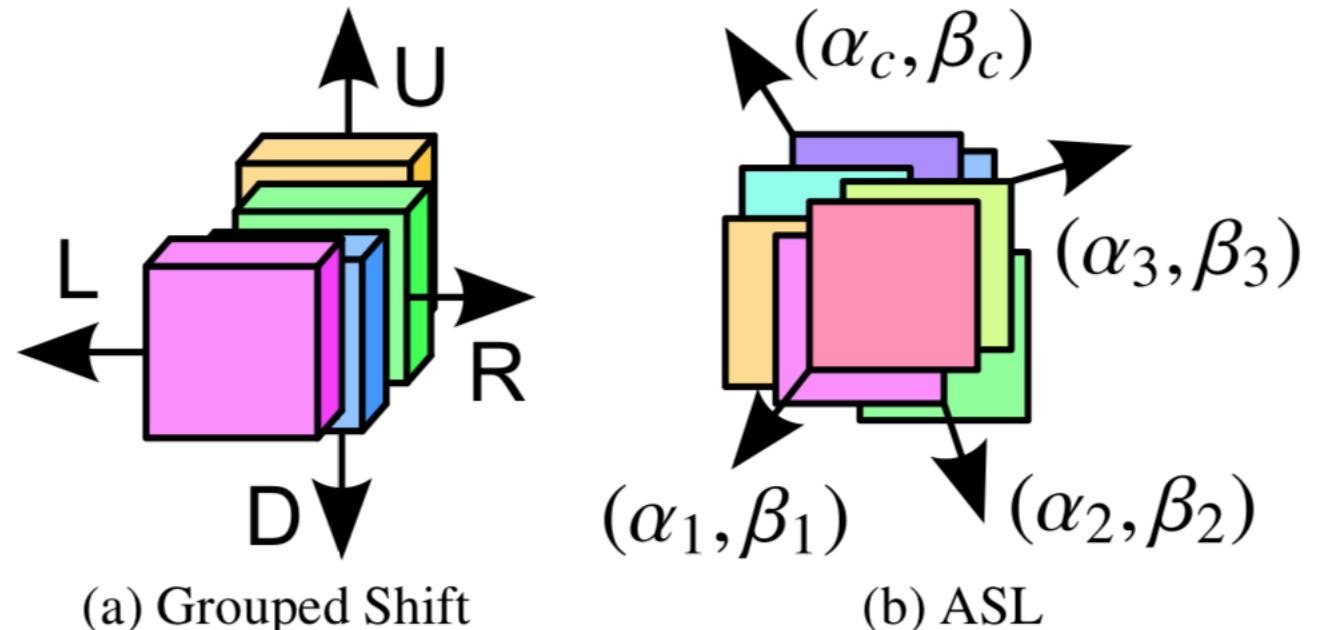
Residual: ShiftNet



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

Residual: Adaptive Shift Layer

- Design
 - Uses a learnable “adaptive” shift parameter that applies 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>

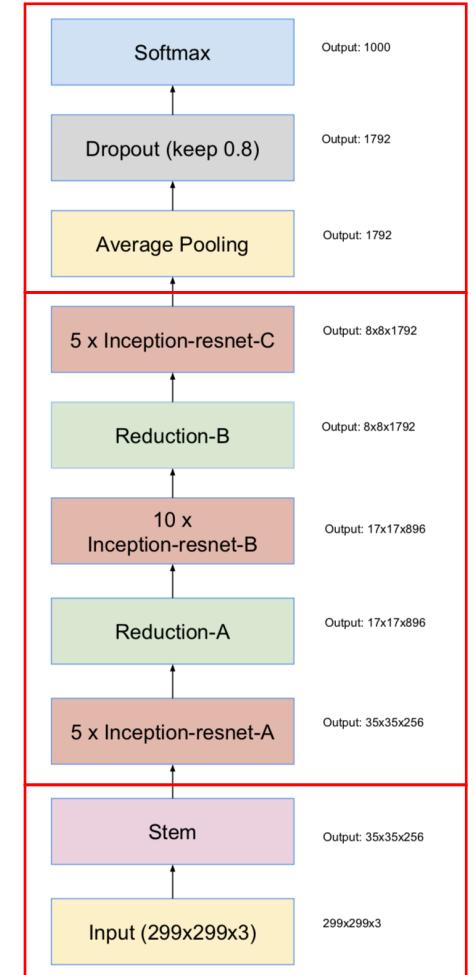
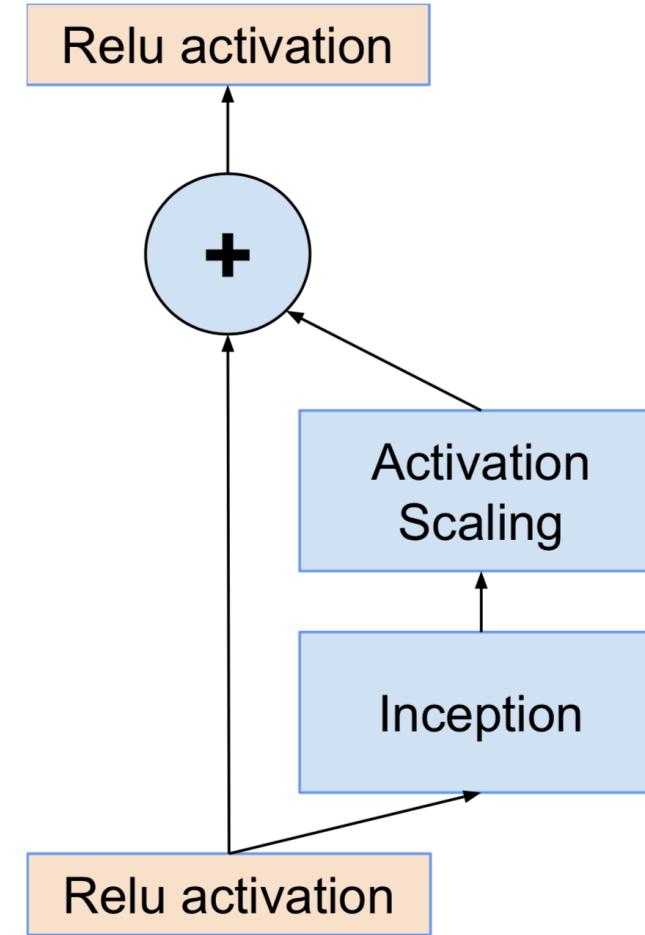


Residual: Inception ResNet V1 And V2

- 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

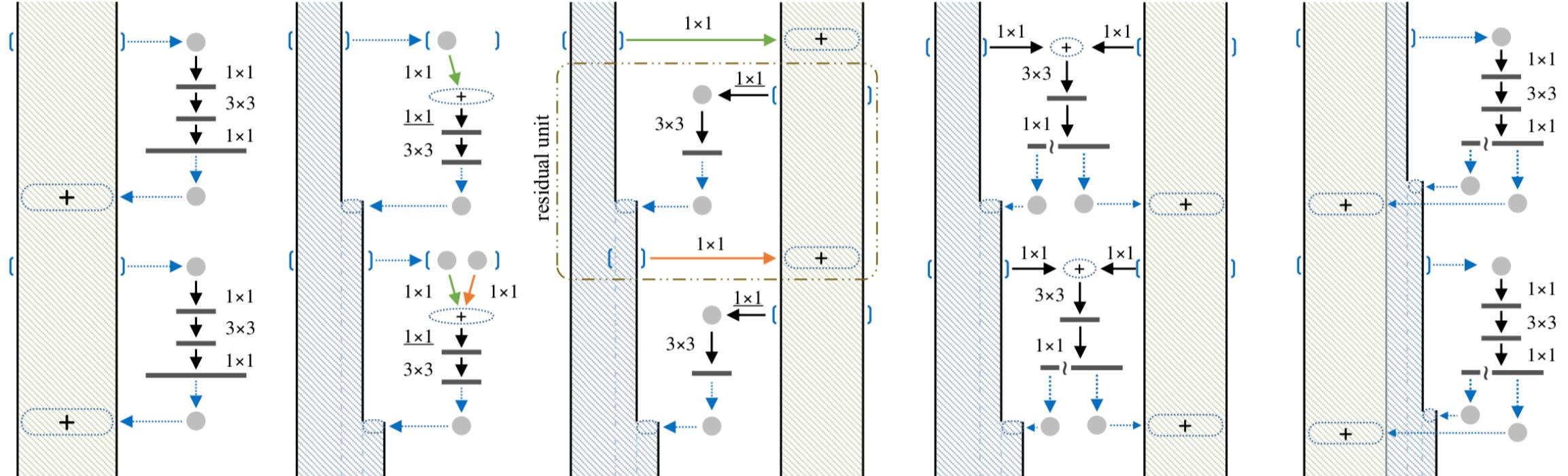
- https://dgschwend.github.io/netscope/#/preset/inceptionv4_resnet



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

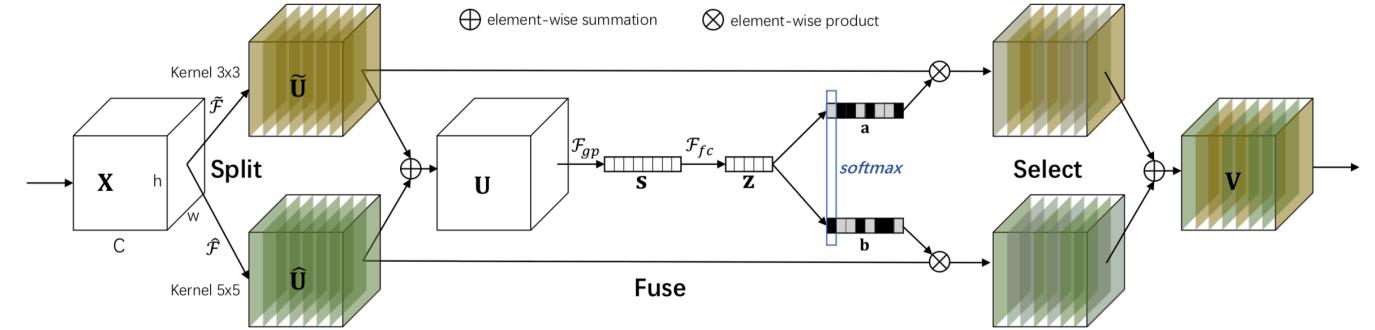
(e) DPN

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	$7 \times 7, 96$, stride 2 3×3 max pool, stride 2	$7 \times 7, 64$, stride 2 3×3 max pool, stride 2	$7 \times 7, 64$, stride 2 3×3 max pool, stride 2	$7 \times 7, 64$, stride 2 3×3 max pool, stride 2	$7 \times 7, 96$, stride 2 3×3 max pool, stride 2
conv2	56x56	$\begin{bmatrix} 1 \times 1, 192 \\ 3 \times 3, 48 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128, G=32 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256, G=64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 96 \\ 3 \times 3, 96, G=32 \\ 1 \times 1, 256 (+16) \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 160 \\ 3 \times 3, 160, G=40 \\ 1 \times 1, 256 (+16) \end{bmatrix} \times 3$
conv3	28x28	$\begin{bmatrix} 1 \times 1, 192 \\ 3 \times 3, 48 \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256, G=32 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512, G=64 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 192 \\ 3 \times 3, 192, G=32 \\ 1 \times 1, 512 (+32) \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 320 \\ 3 \times 3, 320, G=40 \\ 1 \times 1, 512 (+32) \end{bmatrix} \times 6$
conv4	14x14	$\begin{bmatrix} 1 \times 1, 192 \\ 3 \times 3, 48 \end{bmatrix} \times 36$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512, G=32 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024, G=64 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 384 \\ 3 \times 3, 384, G=32 \\ 1 \times 1, 1024 (+24) \end{bmatrix} \times 20$	$\begin{bmatrix} 1 \times 1, 640 \\ 3 \times 3, 640, G=40 \\ 1 \times 1, 1024 (+32) \end{bmatrix} \times 20$
conv5	7x7	$\begin{bmatrix} 1 \times 1, 192 \\ 3 \times 3, 48 \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024, G=32 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 2048 \\ 3 \times 3, 2048, G=64 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 768 \\ 3 \times 3, 768, G=32 \\ 1 \times 1, 2048 (+128) \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 1280 \\ 3 \times 3, 1280, G=40 \\ 1 \times 1, 2048 (+128) \end{bmatrix} \times 3$
	1x1	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		28.9×10^6	44.3×10^6	83.7×10^6	37.8×10^6	61.7×10^6
FLOPs		7.7×10^9	8.0×10^9	15.5×10^9	6.5×10^9	11.7×10^9

Residual: Selective Kernel Networks

- Notes
 - Combines elements of a residual path, parallel filters and squeeze and excite style weighting (now applied to weight different branches)
- Selective kernel networks
 - <https://arxiv.org/abs/1903.06586>



Residual: Selective Kernel Networks

- Notes
 - Combines elements of a residual path, parallel filters and squeeze and excite style weighting (now applied to weight different branches)
- Selective kernel networks
 - <https://arxiv.org/abs/1903.06586>

Output	ResNeXt-50 (32×4d)	SENet-50	SKNet-50
112 × 112		$7 \times 7, 64, \text{stride } 2$	
56 × 56		$3 \times 3 \text{ max pool, stride } 2$	
56 × 56	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128, G = 32 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128, G = 32 \\ 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 128 \\ SK[M = 2, G = 32, r = 16], 128 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
28 × 28	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256, G = 32 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256, G = 32 \\ 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 256 \\ SK[M = 2, G = 32, r = 16], 256 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
14 × 14	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512, G = 32 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512, G = 32 \\ 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 512 \\ SK[M = 2, G = 32, r = 16], 512 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
7 × 7	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024, G = 32 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024, G = 32 \\ 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 1024 \\ SK[M = 2, G = 32, r = 16], 1024 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
1 × 1	$7 \times 7 \text{ global average pool, 1000-d } fc, \text{softmax}$		
#P	25.0M	27.7M	27.5M
GFLOPs	4.24	4.25	4.47

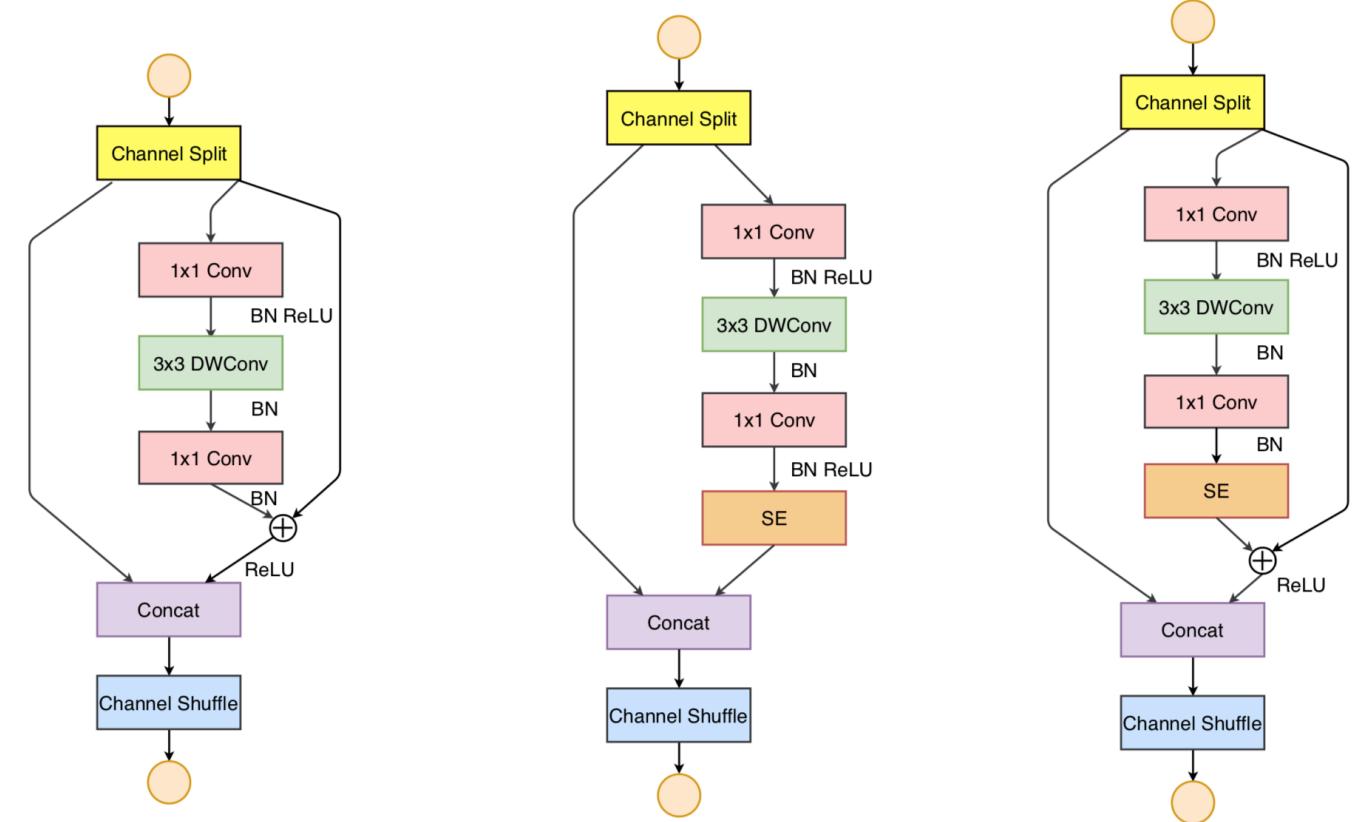
Residual: Selective Kernel Networks

- Notes
 - Combines elements of a residual path, parallel filters and squeeze and excite style weighting (now applied to weight different branches)
- Selective kernel networks
 - <https://arxiv.org/abs/1903.06586>

	top-1 err (%)		#P	GFLOPs
	224×	320×		
ResNeXt-50	22.23	21.05	25.0M	4.24
AttentionNeXt-56 [44]	21.76	–	31.9M	6.32
InceptionV3 [43]	–	21.20	27.1M	5.73
ResNeXt-50 + BAM [32]	21.70	20.15	25.4M	4.31
ResNeXt-50 + CBAM [45]	21.40	20.38	27.7M	4.25
SENet-50 [12]	21.12	19.71	27.7M	4.25
SKNet-50 (ours)	20.79	19.32	27.5M	4.47
ResNeXt-101	21.11	19.86	44.3M	7.99
Attention-92 [44]	–	19.50	51.3M	10.43
DPN-92 [5]	20.70	19.30	37.7M	6.50
DPN-98 [5]	20.20	18.90	61.6M	11.70
InceptionV4 [41]	–	20.00	42.0M	12.31
Inception-ResNetV2 [41]	–	19.90	55.0M	13.22
ResNeXt-101 + BAM [32]	20.67	19.15	44.6M	8.05
ResNeXt-101 + CBAM [45]	20.60	19.42	49.2M	8.00
SENet-101 [12]	20.58	18.61	49.2M	8.00
SKNet-101 (ours)	20.19	18.40	48.9M	8.46

Residual: ShuffleNet V2

- Improvements to ShuffleNet V1 based on empirical performance studies on CPU and GPU architectures
 - Setting $N_i = N$ 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

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
					3×3 max pool, stride 2
conv2_x	56×56	$\begin{bmatrix} 1\times1, 360 \\ 3\times3, 360 \\ 1\times1, 360 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 244 \\ 3\times3, 244 \\ 1\times1, 244 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 340 \\ 3\times3, 340 \\ 1\times1, 340 \end{bmatrix} \times 10$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 720 \\ 3\times3, 720 \\ 1\times1, 720 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 488 \\ 3\times3, 488 \\ 1\times1, 488 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 680 \\ 3\times3, 680 \\ 1\times1, 680 \end{bmatrix} \times 10$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 1440 \\ 3\times3, 1440 \\ 1\times1, 1440 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 976 \\ 3\times3, 976 \\ 1\times1, 976 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 1360 \\ 3\times3, 1360 \\ 1\times1, 1360 \end{bmatrix} \times 23$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 2880 \\ 3\times3, 2880 \\ 1\times1, 2880 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1952 \\ 3\times3, 1952 \\ 1\times1, 1952 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 2720 \\ 3\times3, 2720 \\ 1\times1, 2720 \end{bmatrix} \times 10$
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

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>
- Resource-efficient neural architect
 - <https://arxiv.org/abs/1806.07912>
- DARTS: differentiable architecture search
 - <https://arxiv.org/abs/1806.09055>
- MnasNet: platform-aware neural architecture search for mobile
 - <https://arxiv.org/abs/1807.11626>
- MnasNet: towards automating the design of mobile machine learning models
 - <https://ai.googleblog.com/2018/08/mnasnet-towards-automating-design-of.html>
- ProxylessNAS: direct neural architecture search on target task and hardware
 - <https://arxiv.org/abs/1812.00332>
- SNAS: stochastic neural architecture search
 - <https://arxiv.org/abs/1812.09926>
- Exploring randomly wired neural networks for image recognition
 - <https://arxiv.org/abs/1904.01569>

Optimized Architecture Search

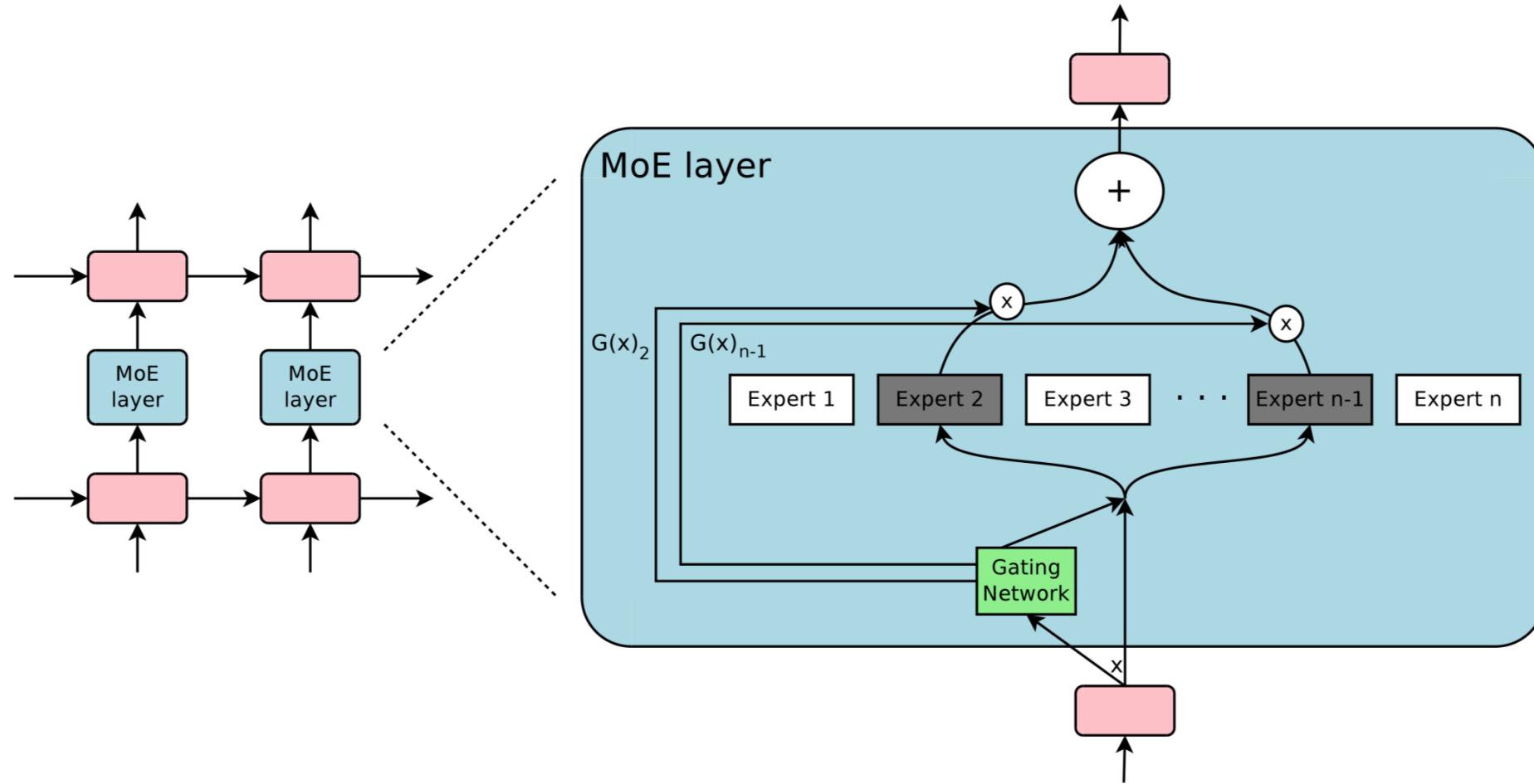
- Searching for MobileNetV3
 - <https://arxiv.org/abs/1905.02244>
- EfficientNet: rethinking model scaling for convolutional neural networks
 - <https://arxiv.org/abs/1905.11946>
- EfficientNet: improving accuracy and efficiency through AutoML and model scaling
 - <https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html>
- EfficientNets
 - <https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet>
- NAS evaluation is frustratingly hard
 - <https://arxiv.org/abs/1912.12522>

Optimized Architecture Search

- 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>

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)



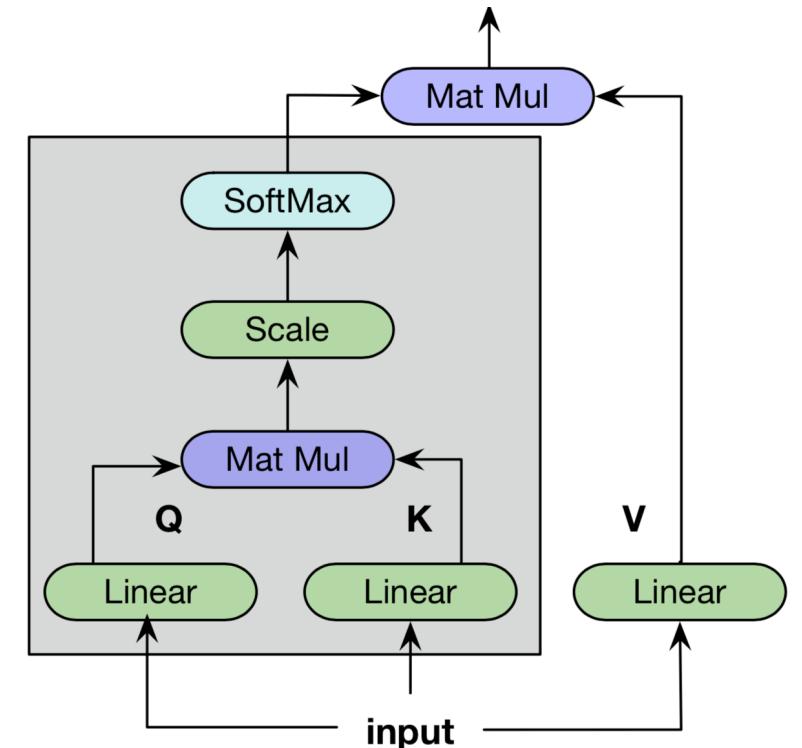
Strategy

- 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 to map from weaker features to stronger features
 - Visualization

Self Attention Layer

- Input X^T is a $M \times K$ matrix composed of M input vectors with K features per vector
- Compute query, key and value matrices (Ws are weights)
 - Query: $X_{q,i}^T = X^T W_{q,i}$, $(M \times P) = (M \times K) (K \times P)$
 - Key: $X_{k,i}^T = X^T W_{k,i}$, $(M \times P) = (M \times K) (K \times P)$
 - Value: $X_{v,i}^T = X^T W_{v,i}$, $(M \times L) = (M \times K) (K \times L)$
- Single headed output Y_i^T is a $M \times L$ matrix composed of M output vectors with L features per vector
 - $Y_i^T = \text{softmax}_{\text{row}}(X_{q,i}^T X_{k,i} / P^{1/2}) X_{v,i}^T$
 $= \text{softmax}_{\text{row}}(X^T W_{q,i} W_{k,i}^T X / P^{1/2}) X^T W_{v,i}$
 $= A_i^T X^T W_{v,i}$

Single headed self attention layer



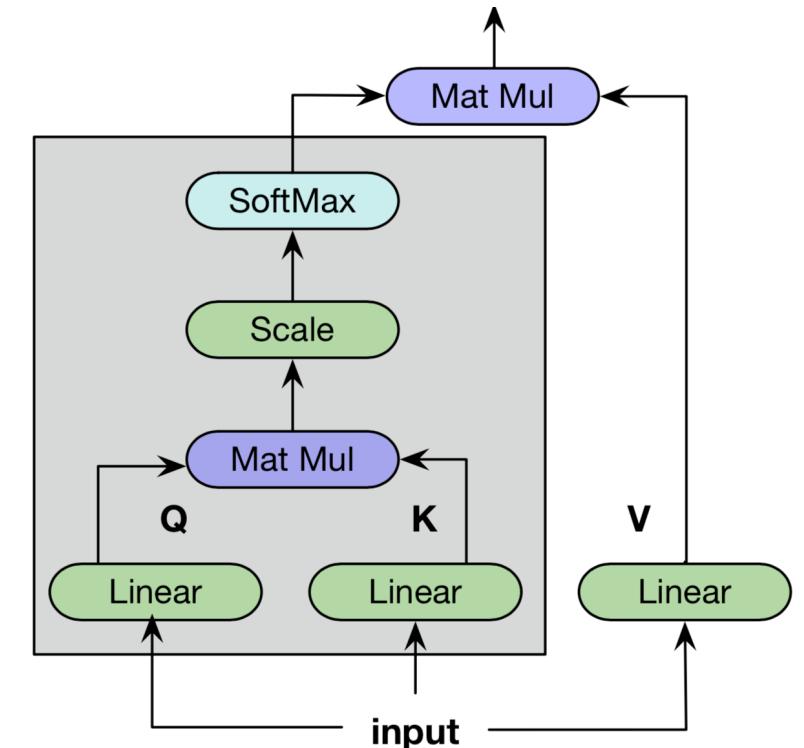
When applied to a matrix the function softmax_{row()} is applied separately per row

Figure from <https://arxiv.org/abs/1901.10430> 189

Self Attention Layer

- Single headed self attention notes
 - Think of A_i^T as 1 pmf per row
 - $A_i^T X^T$ will effectively be M different pmf weighted mixings of the input vectors
 - If P isn't $< 1/2 K$ then it seems appropriate to replace $W_{q,i} W_{k,i}^T$ with a $K \times K$ weight matrix $W_{qk,i}$; choosing $P \ll 1/2 K$ implies creating a low dimension projection to determine the attention distribution
 - The $W_{v,i}$ weight matrix allows the mixing of features within a row of X^T to create new features and / or change the number of features in that row
 - The result is a per head mapping from M input vectors with K features per input vector to M output vectors with L features per output vector

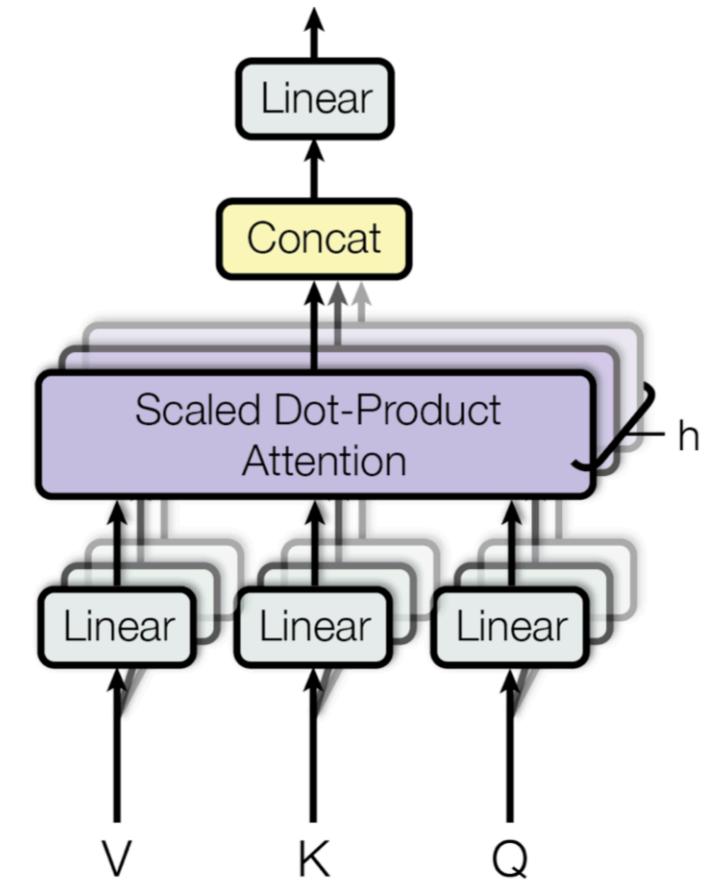
Single headed self attention layer



Self Attention Layer

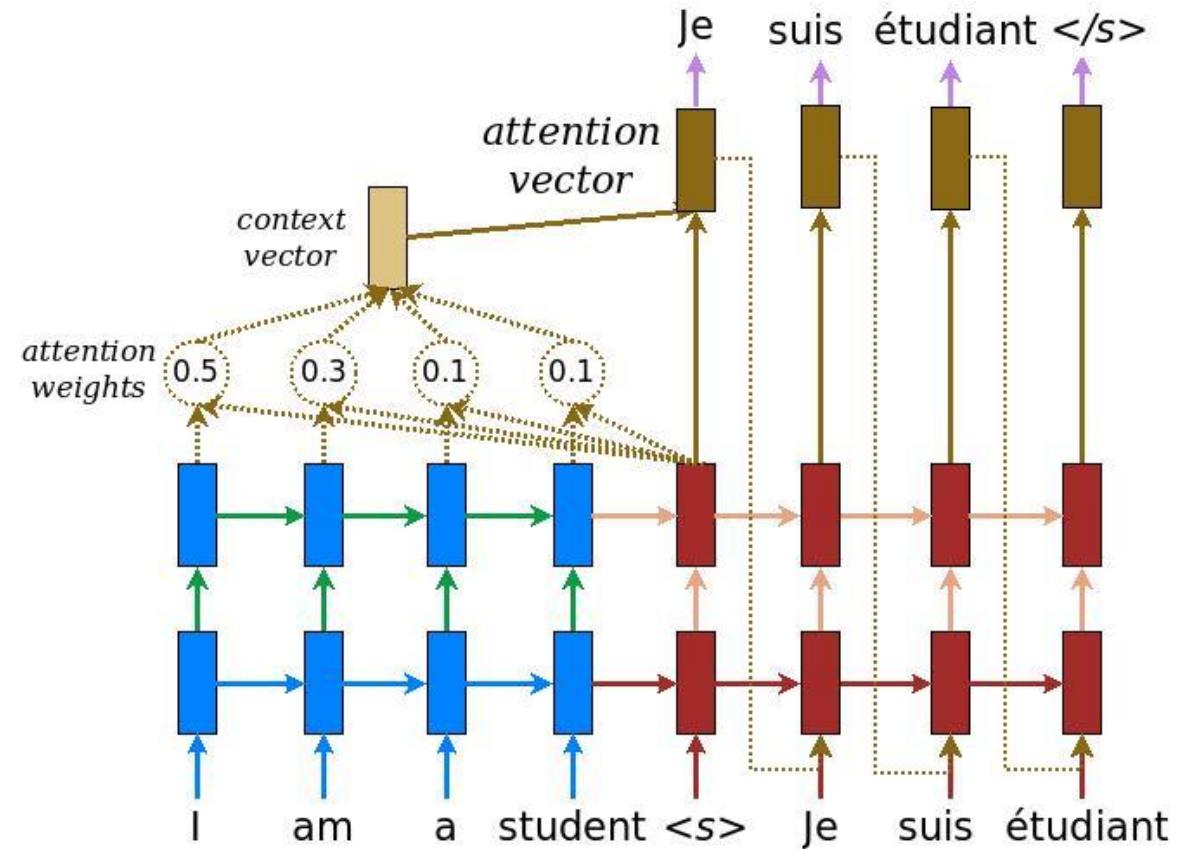
- In multi headed self attention, the outputs of multiple single headed self attention transforms fed by the same input are concatenated together and transformed to create the $M \times N$ output Y^T composed of M output vectors with N feature per vector
 - $W_{o,i}$ is a $L \times N$ weight matrix
 - $$\begin{aligned} Y^T &= \sum_i Y_i^T W_{o,i} & , i = 0, \dots, H - 1 \\ &= \sum_i A_i^T X^T W_{v,i} W_{o,i} & , i = 0, \dots, H - 1 \end{aligned}$$
- Multi headed self attention notes
 - If $KL + LN < KN$ then it seems appropriate to combine $W_{v,i}$ $W_{o,i}$ into a weight matrix $W_{vo,i}$ of size $K \times N$
 - The result is an overall mapping from M input vectors with K features per input vector to M output vectors with N features per output vector

Multi headed self attention layer concatenates outputs of single headed self attention and multiplies by an additional weight matrix



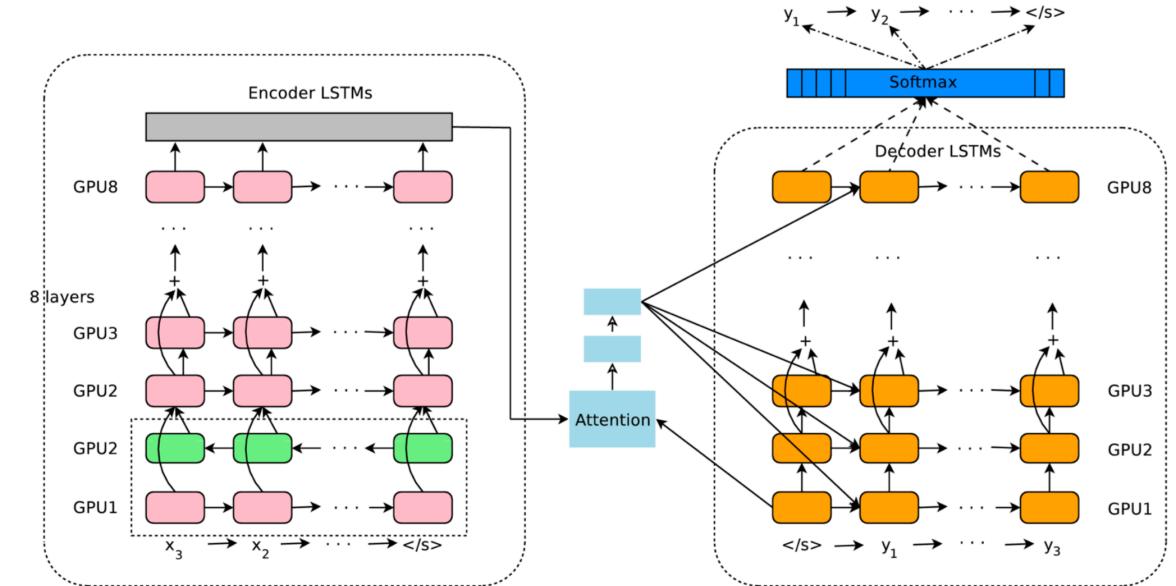
Encoder – Attention – Decoder Layer

- Encoder – attention – decoder layer
 - X^T is a $M \times K$ input, z_d^T is a $1 \times P$ query at position d and $W_{k,0}^T$ is a $P \times K$ trainable weight matrix
 - $a_{0,d}^T = \text{softmax}(z_d^T W_{k,0}^T X / P^{1/2})$ is a $1 \times M$ attention probability mass function for position d (this is the generalized dot product variant, other possibilities exist)
 - $y_{0,d}^T = a_{0,d}^T X^T$ is a $1 \times K$ attention output; the output row vector is a pmf weighted sum of the input row vectors



Encoder – Attention – Decoder Layer

- This is just scratching the surface of attention and there are many possibilities
 - Different scoring functions used to generate attention distributions
 - Different input windows (global / soft, local / hard fixed and adaptive)
 - Different choices of query and binding location for the output



- Aside: RPN and RoI align layers can be thought of as attention mechanisms in object detection and object segmentation networks

Encoder – Attention – Decoder Layer

- Attention intuition
 - Depending on what you're interested in you focus on different things
 - Different parts of an image for captioning
 - Different parts of a sound waveform for speech to text transduction
 - Different parts of a sentence for language to language translation
 - Attention provides a way of doing this
 - The output is a weighted combination of inputs based on a query (what you're interested in)
- A note on computation
 - Transposing matrices is bad, so instead compute $\mathbf{a}_{0,d}^T = (\text{softmax}(\mathbf{X}^T \mathbf{W}_{k,0} \mathbf{z}_d / P^{1/2}))^T$
 - \mathbf{X}^T is the format that the input data is already in
 - Directly store and update $\mathbf{W}_{k,0}$ so no transpose is needed there
 - Transposing the result is transposing a vector so that is just bookkeeping

Encoder – Attention – Decoder Layer

- Note that you can “derive” encoder – attention – decoder from self attention
 - Replace the $M \times P$ input queries with a $1 \times P$ query from the decoder: $X^T W_{q,0} \rightarrow z_d^T$
 - This allows the number of attention outputs D ($d = 0, \dots, D - 1$) to be decoupled from the number of input vectors M , a difference from self attention
 - Simplify
 - Let the number of output features $N =$ the number of input features K
 - Let $L = N$ such that there’s no reduced rank projection
 - Let $W_{v,0} W_{o,0} = I_K$ as any additional projection can be handled by the decoder
 - Historically, though, self attention came from / was motivated by attention
- Question
 - Does it make sense to consider a multi headed version of attention (similar to the multi headed version of self attention)? Should the multiple heads be concatenated instead of added?

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
 - $\text{Sin}()$ and $\text{cos}()$ 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 $y = \text{LayerNorm}(x + \text{Part1or2}(x))$ 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

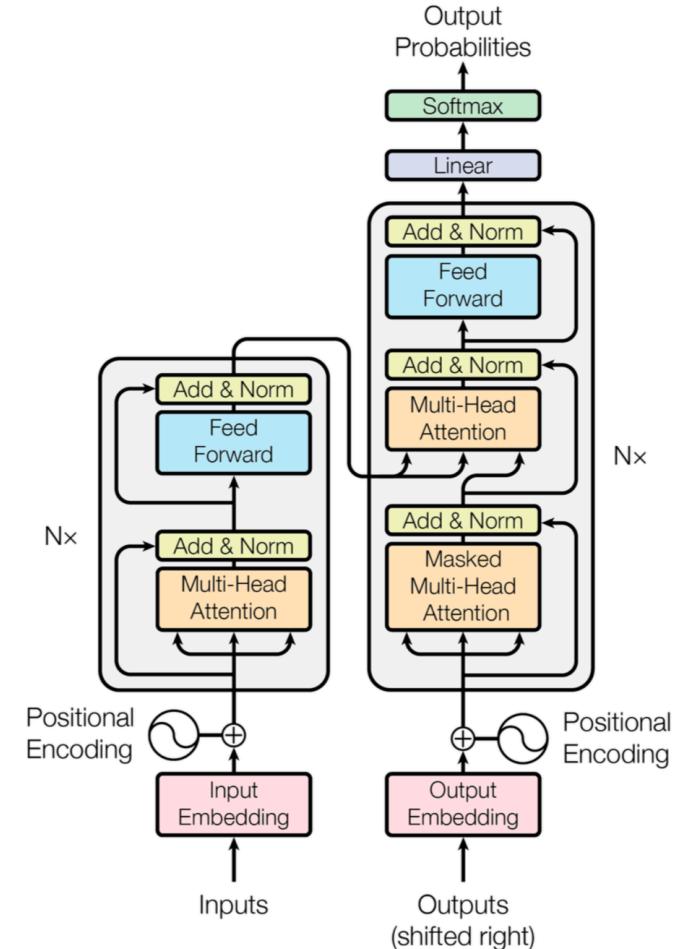


Figure from <https://arxiv.org/abs/1706.03762> 196

Some Recent Results

- Learning multi-attention convolutional neural network for fine-grained image recognition
 - http://openaccess.thecvf.com/content_ICCV_2017/papers/Zheng_Learning_Multi-Attention_Convolutional_ICCV_2017_paper.pdf
- Generative modeling with sparse transformers
 - <https://openai.com/blog/sparse-transformer/>
- Sparse attention
 - https://github.com/openai/sparse_attention
- Language modeling with gated convolutional networks
 - <https://arxiv.org/abs/1612.08083>
- Residual attention network for image classification
 - <https://arxiv.org/abs/1704.06904>
- Area attention
 - <https://arxiv.org/abs/1810.10126>
- A2-nets: double attention networks
 - <https://arxiv.org/abs/1810.11579>

Some Recent Results

- Attention augmented convolutional networks
 - <https://arxiv.org/abs/1904.09925>
- Generating long sequences with sparse transformers
 - <https://arxiv.org/abs/1904.10509>
- Adaptive attention span in transformers
 - <https://arxiv.org/abs/1905.07799>
- Augmenting self-attention with persistent memory
 - <https://arxiv.org/abs/1907.01470>
- Large memory layers with product keys
 - <https://arxiv.org/abs/1907.05242>

Some Recent Results

- Depth-adaptive transformer
 - <https://arxiv.org/abs/1910.10073>
- On the relationship between self-attention and convolutional layers
 - <https://arxiv.org/abs/1911.03584>
- Synthesizer: rethinking self-attention in transformer models
 - <https://arxiv.org/abs/2005.00743>
- Reformer: the efficient transformer
 - <https://arxiv.org/abs/2001.04451>

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
- Improving neural networks by preventing co-adaptation of feature detectors
 - <https://arxiv.org/abs/1207.0580>
- On the number of linear regions of deep neural networks
 - <https://arxiv.org/abs/1402.1869>
- Batch normalization: accelerating deep network training by reducing internal covariate shift
 - <https://arxiv.org/abs/1502.03167>
- 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>
- The history began from AlexNet: a comprehensive survey on deep learning approaches
 - <https://arxiv.org/abs/1803.01164>
- Group normalization
 - <https://arxiv.org/abs/1803.08494>

List

- 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
- 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>