ATTENTION & NEURAL NETWORK ARCHITECTURES

Recap from last time

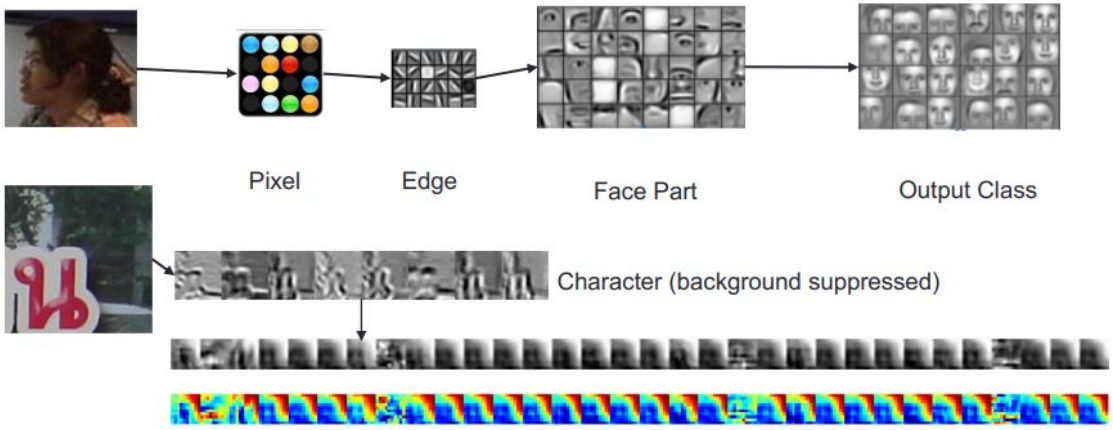
- DNN
- CNN
- RNN families (RNN, GRU, LSTM)
- Training & Optimizing NN

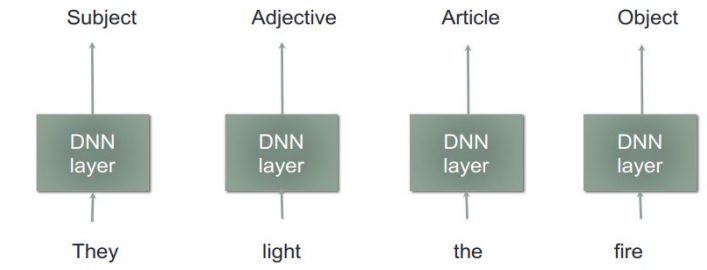
Structure of this lecture

- Attention + Transformer
- Deep Learning Architectures in Computer Vision

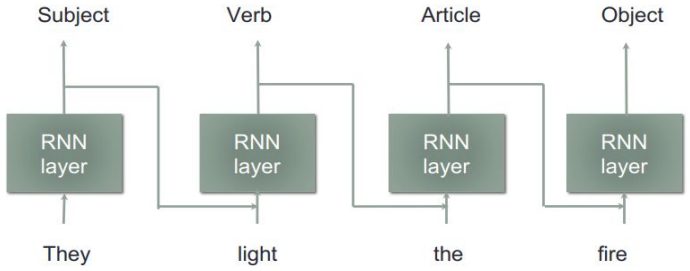
CNN and RNN recap

- CNN
  - Assume that **Spatial** information is useful
  - Convolutional layer has shift-invariant property
- RNN families (RNN, GRU, LSTM)
  - Assume that **Temporal** information is useful



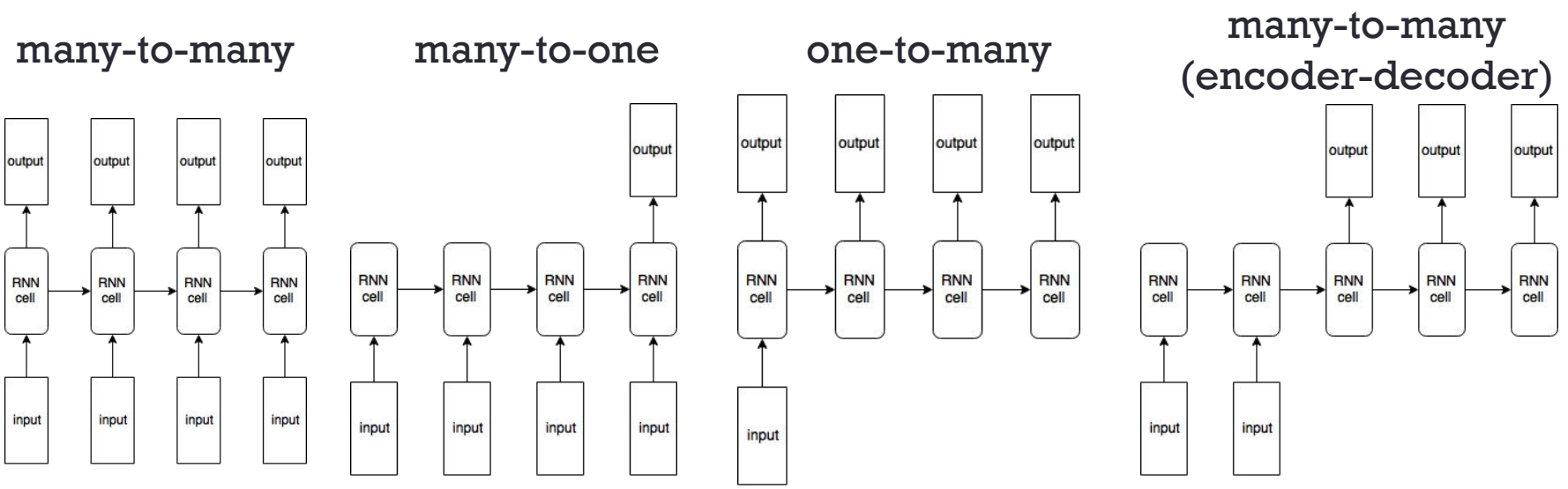


Problem 1: need a way to remember the past



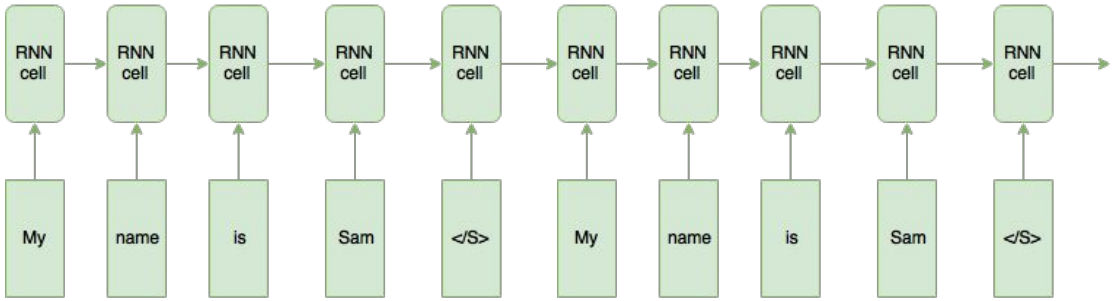
Output of the layer encodes something meaningful about the past

Different recurrent architectures



Drawbacks of CNN and RNN

- CNN
  - Shift-invariant property is actually a double edge sword
    - Does not make much sense in some tasks
      - Order Prediction (Given text “[A, B, C, D]” what is the third element?”)
  - Need a lot of layer to capture long range information
- RNN families (RNN, GRU, LSTM)
  - Poor parallelism
  - Exploding / Vanishing gradient
    - Not very of good at understanding very long sequence input.



…

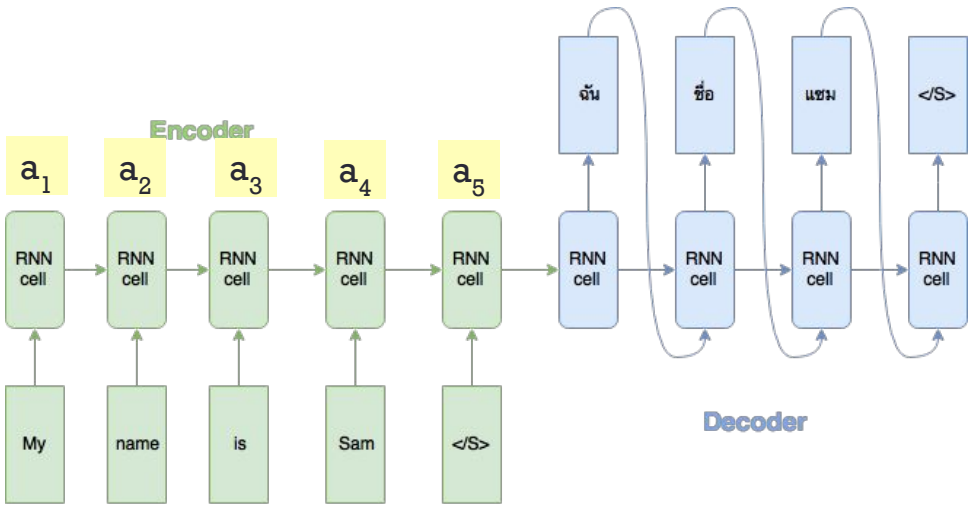
Shouldn’t there be some mechanism to mitigate this issue?

Attention Mechanism

Solution : A mechanism to capture long range information

Attention is commonly used in sequence-to-sequence model, it allows **the decoder part** of the network to focus/**attend** on a different part of **the encoder outputs** for every step of the decoder's own outputs.

This is what we want you to think about: How can information travel from one end to another in neural networks?



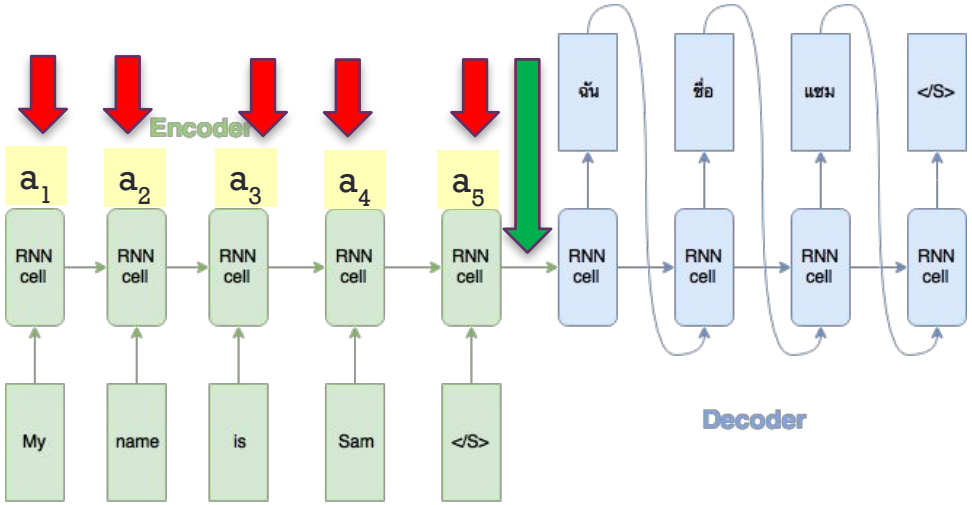
Machine Translation Problem: English to Thai

Attention Mechanism (cont.)

Why attention?

Main idea: We can use multiple vectors based on the length of the sentence instead of one.

Attention mechanism = Instead of encoding all the information into a fixed-length vector, the decoder gets to decide parts of the input source to pay attention. -> **Access to Global information**

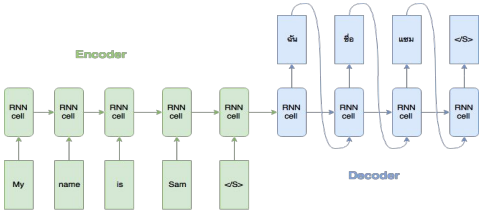


Machine Translation Problem: English to Thai

Graphical Example: English-to-Thai machine translation

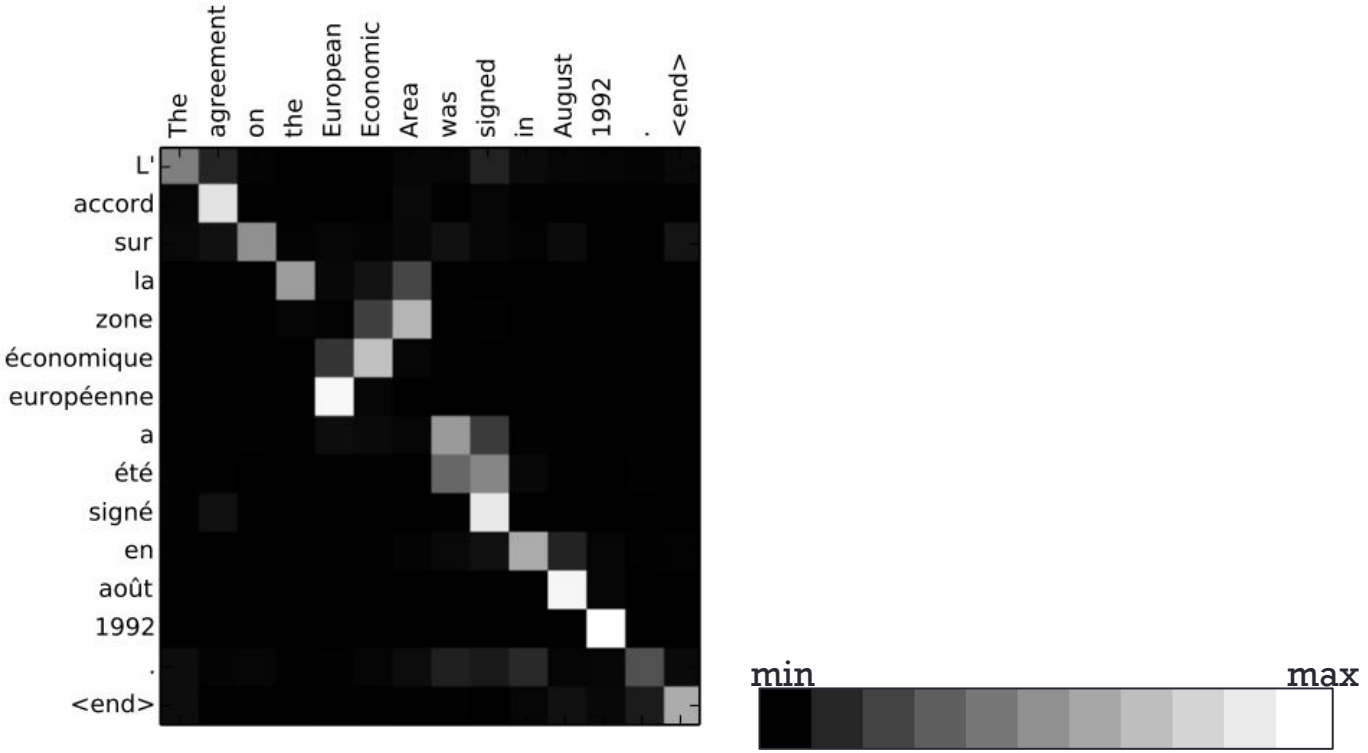
- This is a rough estimate of what might occur for English-to-Thai translation

	My	name	is	Sam	encoder
ฉัน					
ชื่อ					
แซม					
decoder					



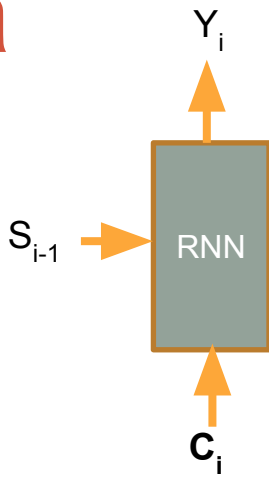
Machine Translation Problem: English to Thai

Graphical Example: English-to-French machine translation



Reference: Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." ICLR(2015).

Attention Mechanism: Basic Idea



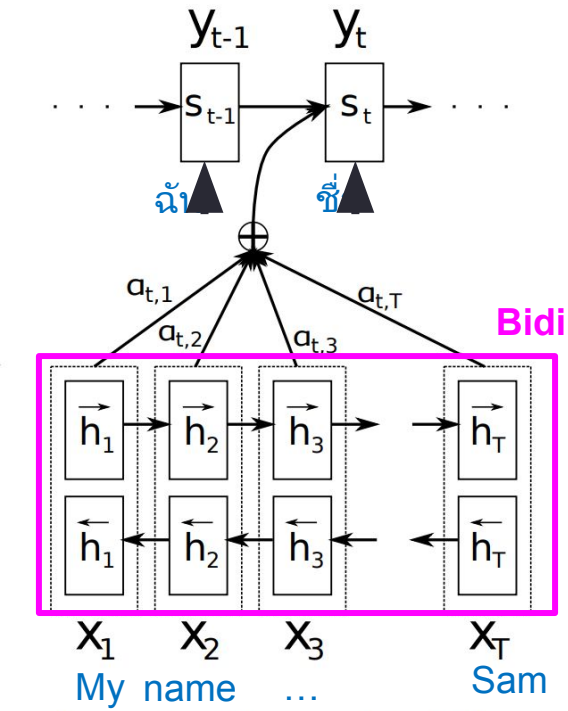
- Encode each word in the sequence into a vector
- When DECODING, perform a linear combination of these encoded vectors from the encoding step with their corresponding "attention weights".

$$\mathbf{c}_i = \sum_j a_{ij}\mathbf{h}_j$$

j = each encoder's input
i = each decoder's input

- A vector formed by this linear combination is called "**context vector**"
- Use context vectors as inputs for the decoding step

Reference: Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." ICLR(2015).



Decoder

Bidirectional LSTM

Encoder

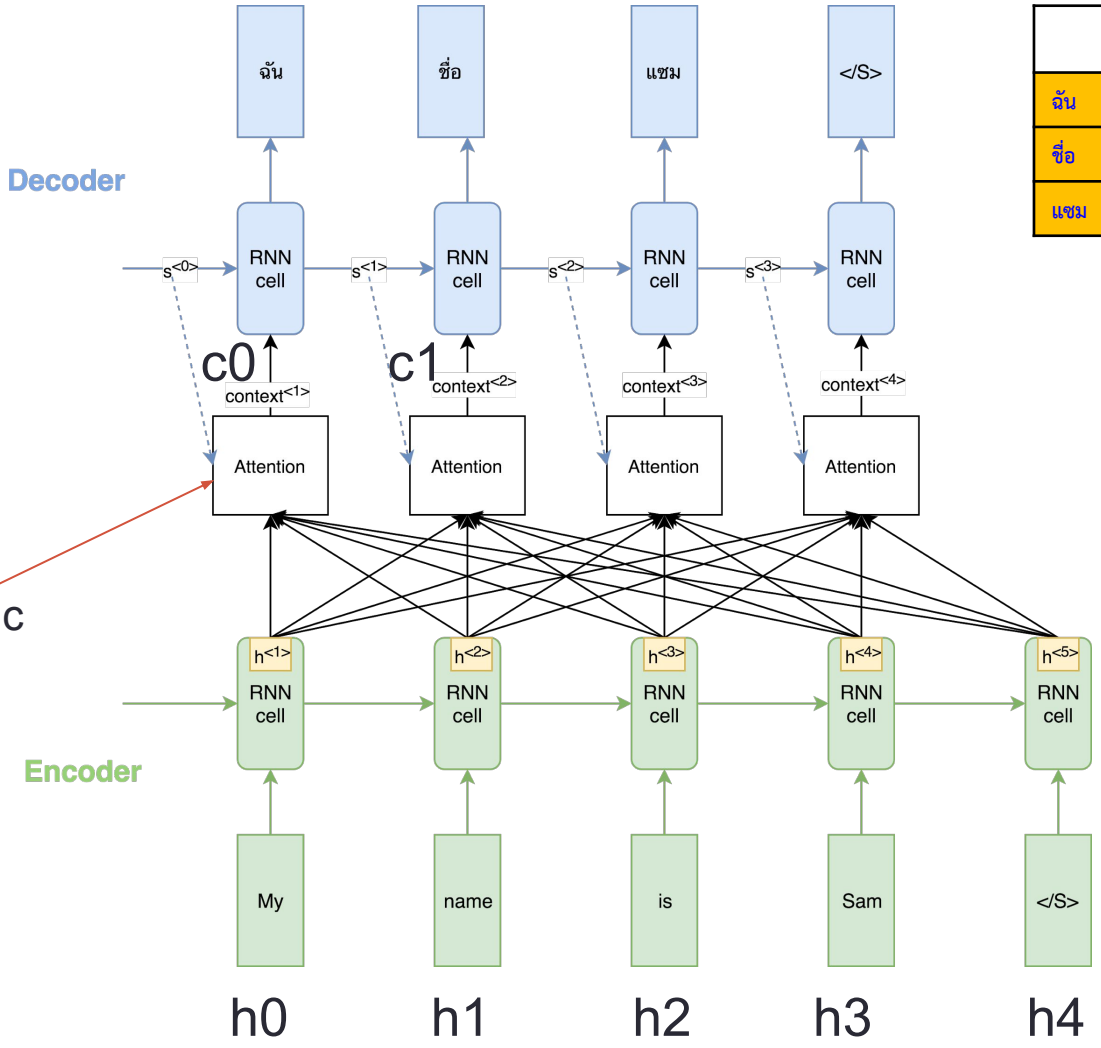
Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

source = encoder

target word = decoder

Reference: Bahdanau, D., Cho, K., & Bengio, Y.. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015

RNN and attention mechanism



Attention Mechanism (1): C_i

We want to calculate a context vector ***c*** based on hidden states $\mathbf{s}_0 \ldots \mathbf{s}_{m-1}$ that can be used with the current state $\mathbf{h}_j$ for prediction. The context vector $\mathbf{c}_i$ at position "i" is calculated as an average of the previous states weighted with the attention scores $\mathbf{a}_i$.

$$\mathbf{c}_i = \sum_j a_{ij} \mathbf{h}_j$$

context vector

attention score

encoder state at index j

i = decoder index
j = encoder index

$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

$$\frac{e^{f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j)}}{\sum_j e^{f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j)}}$$

previous hidden state/
decoder state

attention score(weight vector)
of encoder state at index j

encoder state at index j

Reference: http://ruder.io/deep-learning-nlp-best-practices/index.html#attention

Attention Mechanism (2): f_{att}

sum(a_ij) should be 1

$$\mathbf{c}_i = \sum_j a_{ij} \mathbf{h}_j$$

context vector

attention score

encoder state at index j

The attention function $f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j)$ calculates an unnormalized alignment score between the current hidden state $\mathbf{s}_{i-1}$ and the previous hidden state $\mathbf{h}_j$.

There are many variants of the attention function f_{att}.

i = decoder index
j = encoder index

$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

$$\frac{e^{f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j)}}{\sum_j e^{f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j)}}$$

previous hidden state/
decoder state

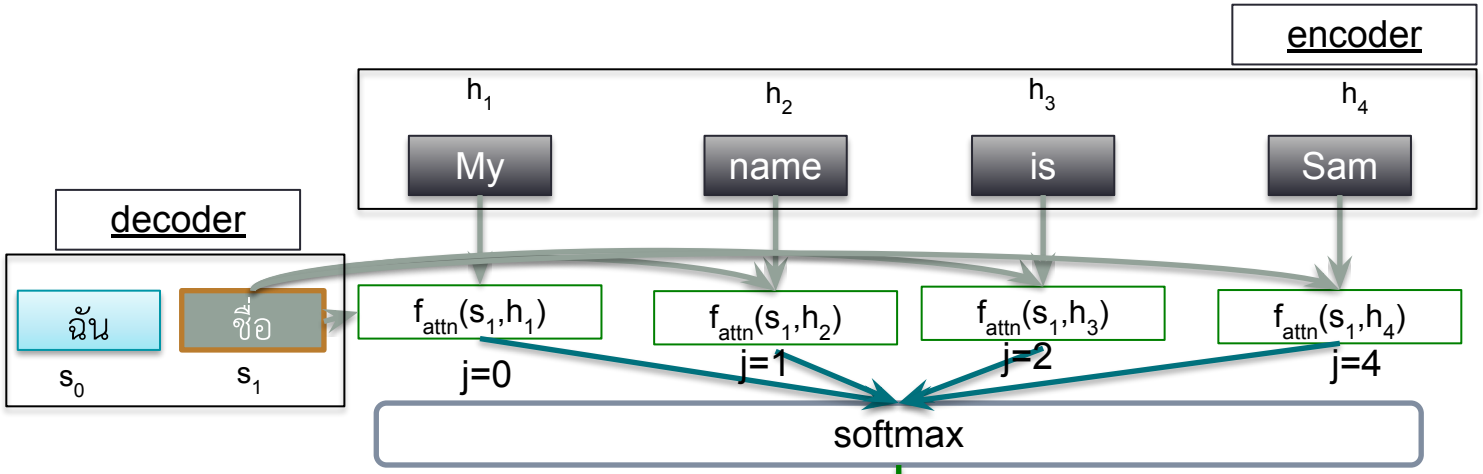
attention score(weight vector)
of encoder state at index j

encoder state at index j

Reference: http://ruder.io/deep-learning-nlp-best-practices/index.html#attention

Attention Calculation Example (1): Attention Scores

Now we want to predict แซม **(i=2)**



$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

$\mathbf{a}_2$=[$a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$]

i = decoder index
j = encoder index

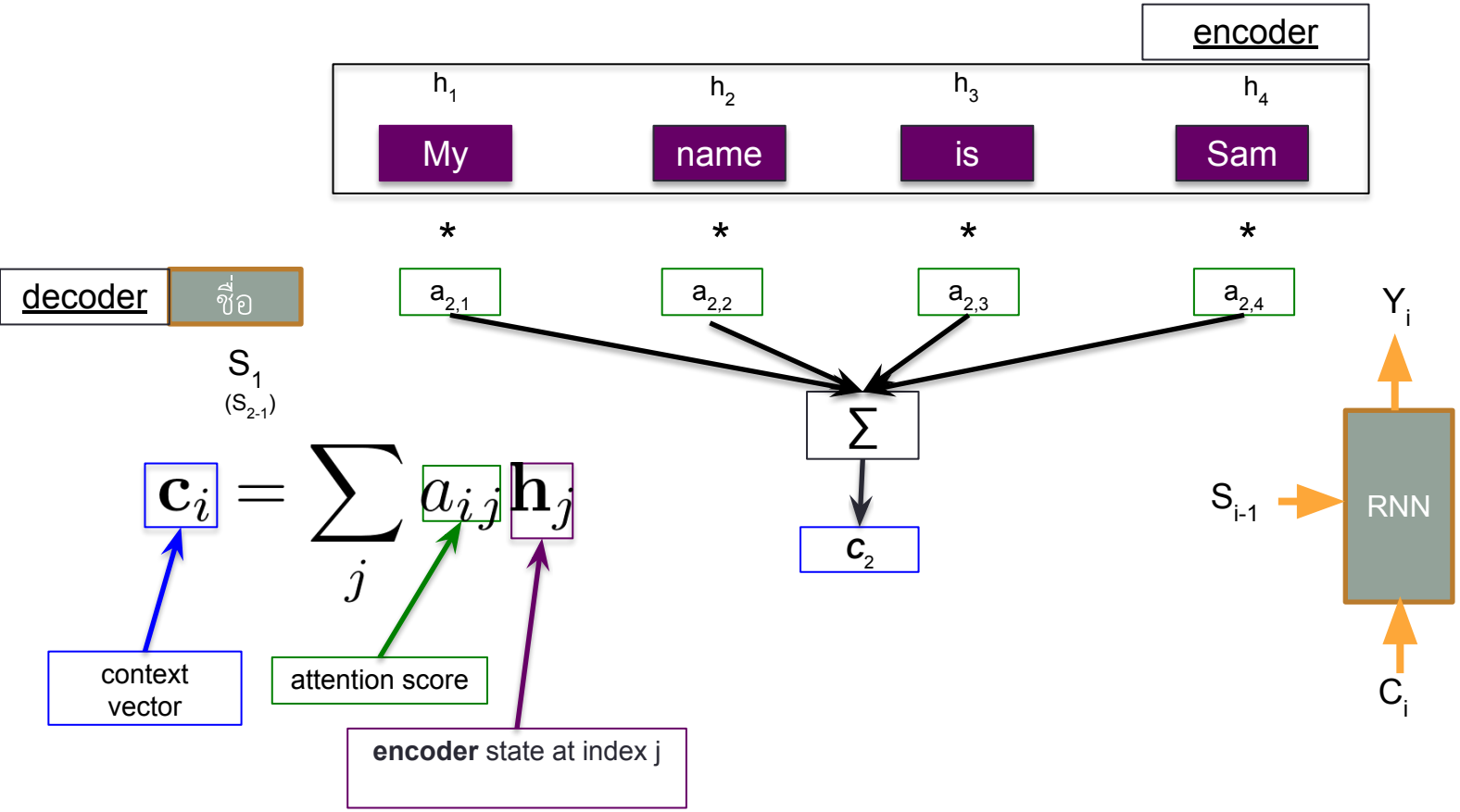
attention scores

previous hidden state/ decoder state

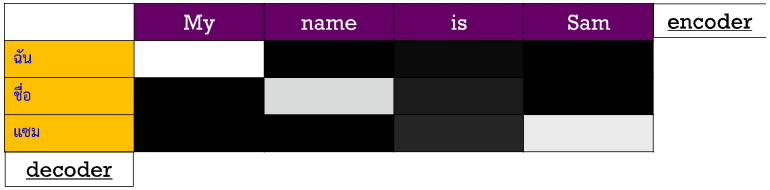
encoder state at index j

$$\frac{e^{f_{attn}(\mathbf{s}_{i-1},\mathbf{h}_j)}}{\sum_j e^{f_{attn}(\mathbf{s}_{i-1},\mathbf{h}_j)}}$$

Attention Calculation Example (2): Context Vector



$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$



Type of Attention mechanisms

(Remember that there are many variants of attention function f_{attn})

Additive attention: The original attention mechanism (Bahdanau et al., 2015) uses a one-hidden layer feed-forward network to calculate the attention alignment:

$$f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j) = v_a^T tanh(\mathbf{W}_a[\mathbf{s}_{i-1}; \mathbf{h}_j])$$

h_i = encoder index
s_j = decoder index

Multiplicative attention: Multiplicative attention (Luong et al., 2015) simplifies the attention operation by calculating the following function:

$$f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{s}_{i-1}^\top \mathbf{W}_a \mathbf{h}_j$$

Self-attention: Without any additional information, however, we can still extract relevant aspects from the sentence by allowing it to attend to itself using self-attention (Lin et al., 2017)

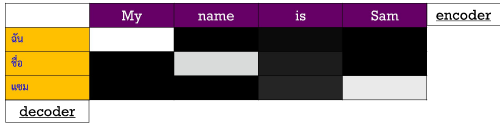
$$\mathbf{a} = softmax(\mathbf{w}_{s_2} tanh(\mathbf{W}_{s_1} \mathbf{H}^T))$$

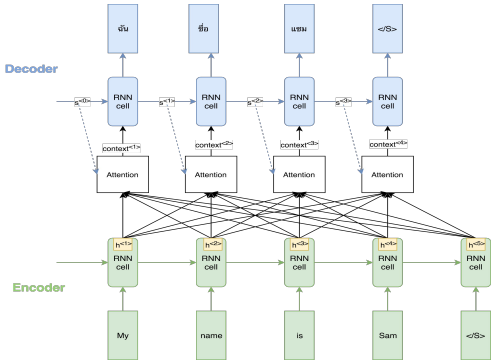
Key-value attention: key-value attention (Daniluk et al., 2017) is a recent attention variant that separates form from function by keeping separate vectors for the attention calculation.

Reference: http://ruder.io/deep-learning-nlp-best-practices/index.html#attention

$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

Additive Attention





- The original attention mechanism (Bahdanau et al., 2015) uses a one-hidden layer feed-forward network to calculate the attention alignment:

concatenation

$$f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j) = v_a^T tanh(\mathbf{W}_a[\mathbf{s}_{i-1}; \mathbf{h}_j])$$

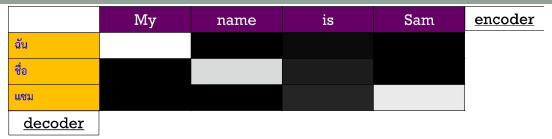
One-hidden layer (Dense)

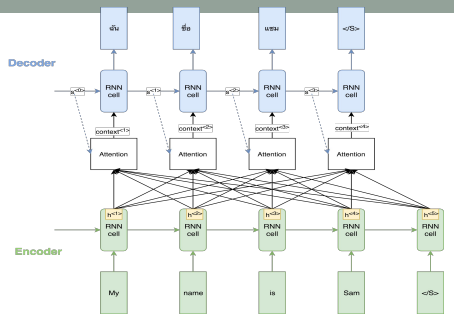
- Where $\mathbf{W}_a$ are learned attention parameters. Analogously, we can also use matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ to learn separate transformations for $\mathbf{s}_{i-1}$ and $\mathbf{h}_j$ respectively, which are then summed (hence the name additive):

$$f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j) = v_a^T tanh(\mathbf{W}_1\mathbf{s}_{i-1} + \mathbf{W}_2\mathbf{h}_j)$$

Reference: http://ruder.io/deep-learning-nlp-best-practices/index.html#attention

$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$





Multiplicative Attention

- Multiplicative attention (Luong et al., 2015) [16] simplifies the attention operation by calculating the following function:

$$f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{s}_{i-1}^{\top} \mathbf{W}_a \mathbf{h}_j$$

- Faster, more efficient than additive attention BUT additive attention performs better for larger dimensions
- One way to mitigate this is to scale f_{attn} by $\frac{1}{\sqrt{d_s}}$

d_s = #dimensions of hidden states in LSTM (context vector; latent factors)

  - Dot product of high dimensional vectors has high variance -> softmax is peaky -> small gradient -> harder to train

Additive attention

$$f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j) = v_a^T tanh(\mathbf{W}_a[\mathbf{s}_{i-1}; \mathbf{h}_j]) \longrightarrow f_{attn}(\mathbf{h}_i, \mathbf{h}_j) = v_a^T tanh(\mathbf{W}_a[\mathbf{h}_i; \mathbf{h}_j])$$

Self Attention (1)

- Without any additional information, we can still extract relevant aspects from the sentence by allowing it to attend to itself using self-attention (Lin et al., 2017)

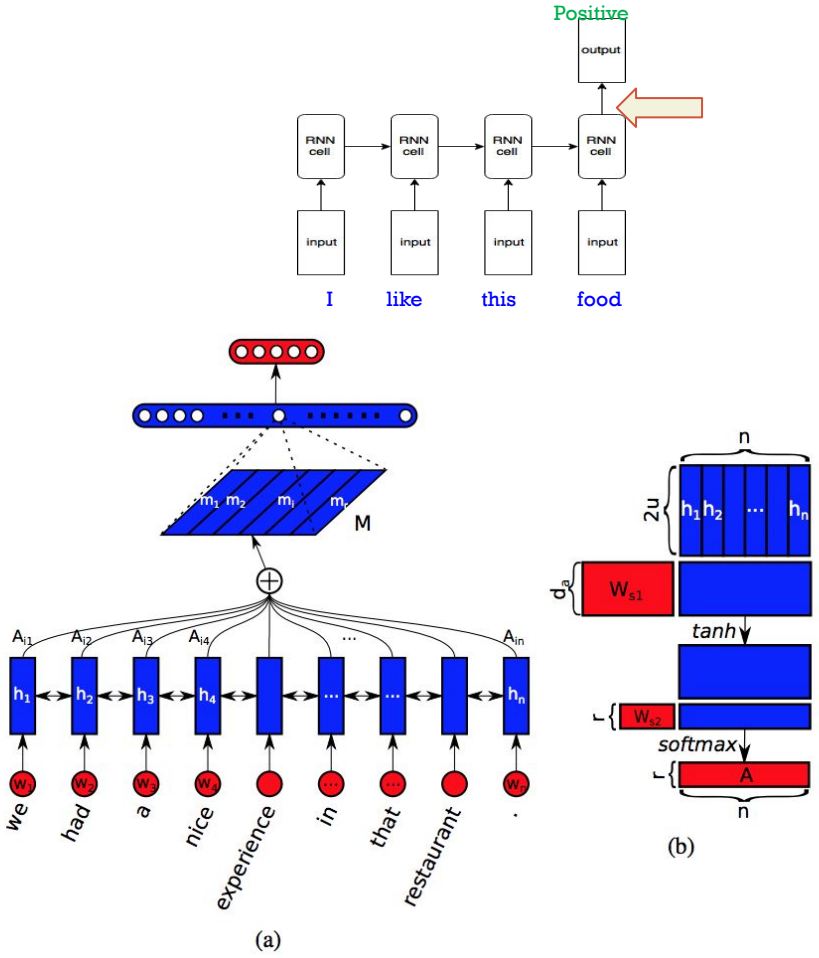
$$H = (\mathbf{h_1}, \mathbf{h_2}, \cdots \mathbf{h_n})$$

Fully connected layer

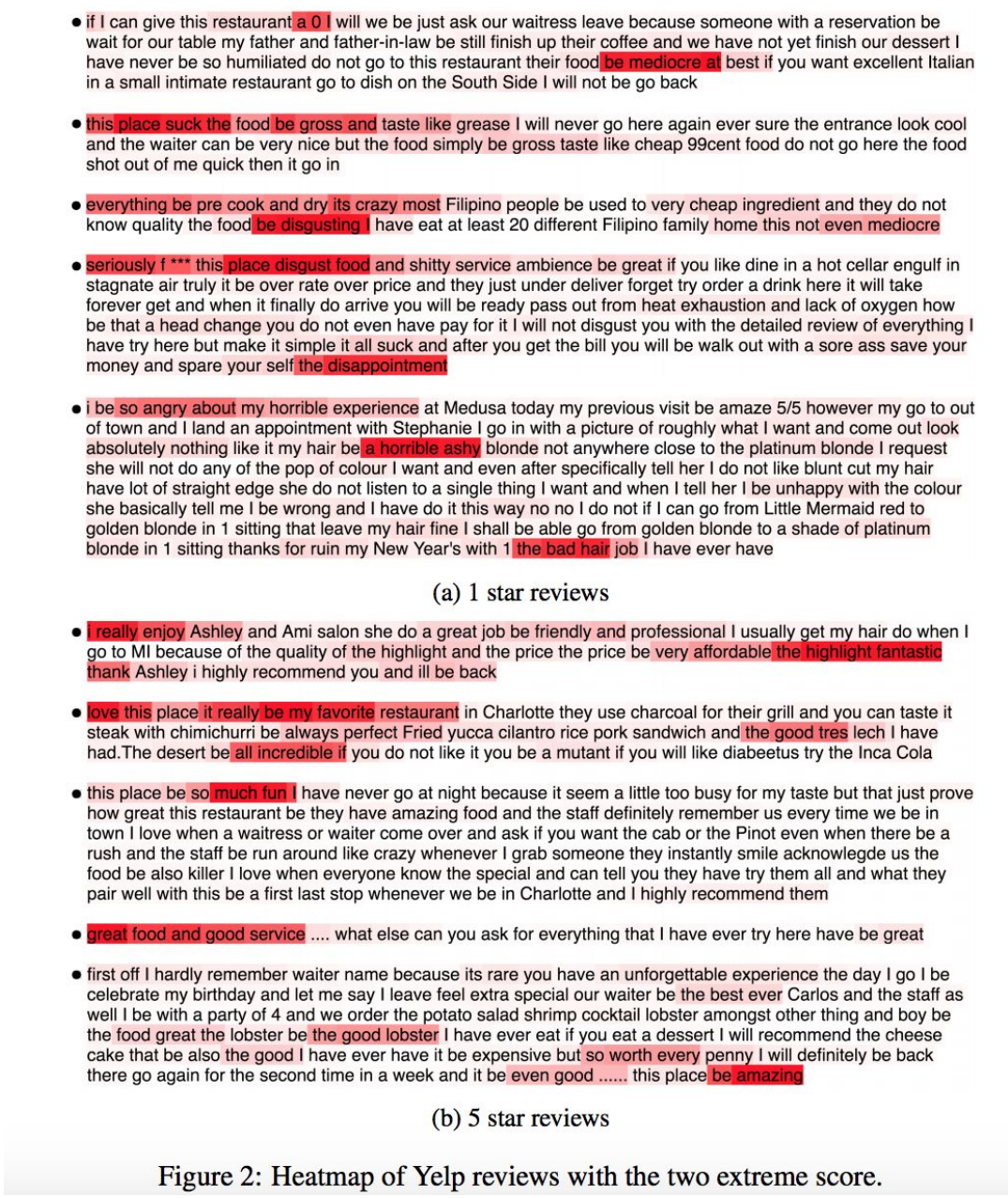
$$\mathbf{a} = softmax(\mathbf{w}_{s_2} tanh(\mathbf{W}_{s_1} \mathbf{H}^T)$$

One-hidden layer (Dense)

- w_{s1} is a weight matrix, w_{s2} is a vector of parameters. Note that these parameters are tuned by the neural networks.
- **The objective is to improve a quality of embedding vector by adding context information.**



Self-attention (2)



- if I can give this restaurant a 0 I will we be just ask our waitress leave because someone with a reservation be wait for our table my father and father-in-law be still finish up their coffee and we have not yet finish our dessert I have never be so humiliated do not go to this restaurant their food be mediocre at best if you want excellent Italian in a small intimate restaurant go to dish on the South Side I will not be go back

- this place suck the food be gross and taste like grease I will never go here again ever sure the entrance look cool and the waiter can be very nice but the food simply be gross taste like cheap 99cent food do not go here the food shot out of me quick then it go in

- everything be pre cook and dry its crazy most Filipino people be used to very cheap ingredient and they do not know quality the food be disgusting I have eat at least 20 different Filipino family home this not even mediocre

- seriously f *** this place disgust food and shitty service ambience be great if you like dine in a hot cellar engulf in stagnate air truly it be over rate over price and they just under deliver forget try order a drink here it will take forever get and when it finally do arrive you will be ready pass out from heat exhaustion and lack of oxygen how be that a head change you do not even have pay for it I will not disgust you with the detailed review of everything I have try here but make it simple it all suck and after you get the bill you will be walk out with a sore ass save your money and spare your self the disappointment

- i be so angry about my horrible experience at Medusa today my previous visit be amaze 5/5 however my go to out of town and I land an appointment with Stephanie I go in with a picture of roughly what I want and come out look absolutely nothing like it my hair be a horrible ashy blonde not anywhere close to the platinum blonde I request she will not do any of the pop of colour I want and even after specifically tell her I do not like blunt cut my hair have lot of straight edge she do not listen to a single thing I want and when I tell her I be unhappy with the colour she basically tell me I be wrong and I have do it this way no no I do not if I can go from Little Mermaid red to golden blonde in 1 sitting that leave my hair fine I shall be able go from golden blonde to a shade of platinum blonde in 1 sitting thanks for ruin my New Year's with 1 the bad hair job I have ever have

(a) 1 star reviews

- I really enjoy Ashley and Ami salon she do a great job be friendly and professional I usually get my hair do when I go to MI because of the quality of the highlight and the price the price be very affordable the highlight fantastic thank Ashley i highly recommend you and ill be back

- love this place it really be my favorite restaurant in Charlotte they use charcoal for their grill and you can taste it steak with chimichurri be always perfect Fried yucca cilantro rice pork sandwich and the good tres lech I have had.The desert be all incredible if you do not like it you be a mutant if you will like diabeetus try the Inca Cola

- this place be so much fun I have never go at night because it seem a little too busy for my taste but that just prove how great this restaurant be they have amazing food and the staff definitely remember us every time we be in town I love when a waitress or waiter come over and ask if you want the cab or the Pinot even when there be a rush and the staff be run around like crazy whenever I grab someone they instantly smile acknowlegde us the food be also killer I love when everyone know the special and can tell you they have try them all and what they pair well with this be a first last stop whenever we be in Charlotte and I highly recommend them

- great food and good service .... what else can you ask for everything that I have ever try here have be great

- first off I hardly remember waiter name because its rare you have an unforgettable experience the day I go I be celebrate my birthday and let me say I leave feel extra special our waiter be the best ever Carlos and the staff as well I be with a party of 4 and we order the potato salad shrimp cocktail lobster amongst other thing and boy be the food great the lobster be the good lobster I have ever eat if you eat a dessert I will recommend the cheese cake that be also the good I have ever have it be expensive but so worth every penny I will definitely be back there go again for the second time in a week and it be even good ..... this place be amazing

(b) 5 star reviews

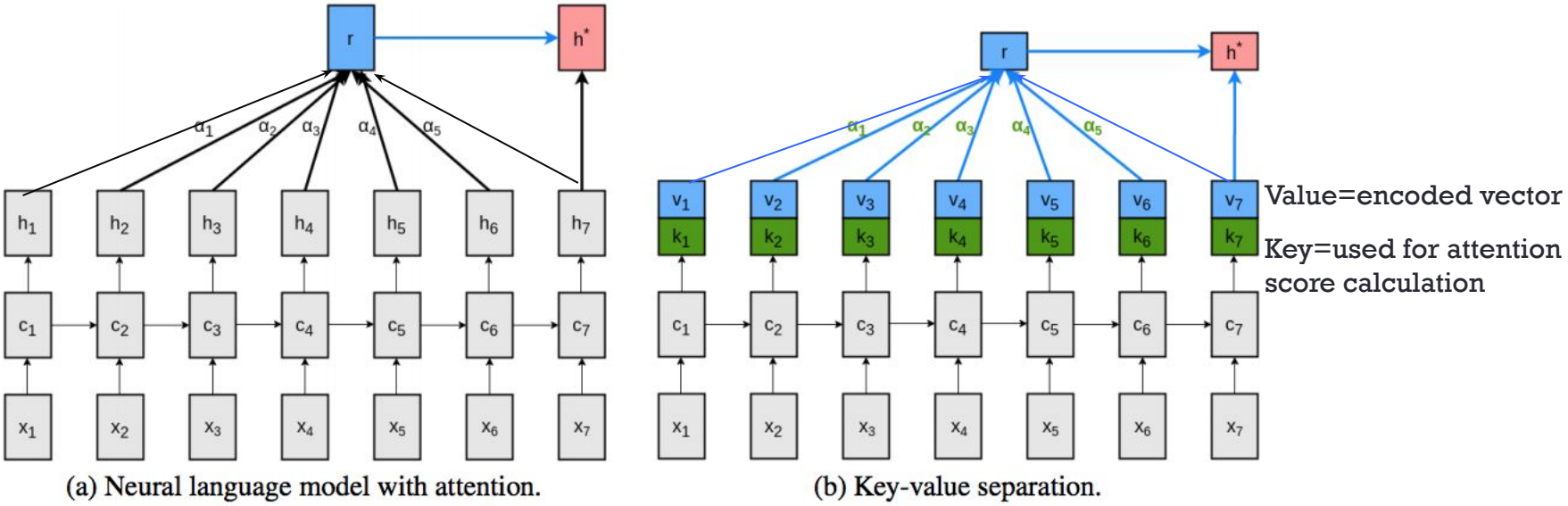
Figure 2: Heatmap of Yelp reviews with the two extreme score.

$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

$$\mathbf{c}_i = \sum_j a_{ij}\mathbf{h}_j$$

Key-value attention (1)

Separate **h** in to **k**, **v** to specialize the neurons
k for learning attentions, **v** for sending information



(a) Neural language model with attention.

(b) Key-value separation.

Reference: Daniluk, M., Rockt, T., Welbl, J., & Riedel, S. (2017). Frustratingly Short Attention Spans in Neural Language Modeling. In ICLR 2017.

Additive attention

$$f_{attn}(\mathbf{s}_{i-1}, \mathbf{h}_j) = v_a^T tanh(\mathbf{W}_a[\mathbf{s}_{i-1}; \mathbf{h}_j])$$

Key-value attention (2)

previous outputs (memory) **with window L**

current output

k - output dimension
L - output length

key

value

$$\begin{bmatrix} \boldsymbol{k}_t \\ \boldsymbol{v}_t \end{bmatrix} = \boldsymbol{h}_t \quad \in \mathbb{R}^{2k}$$

$$\boldsymbol{M}_t = \tanh(\boldsymbol{W}^Y [\boldsymbol{k}_{t-L} \cdots \boldsymbol{k}_{t-1}] + (\boldsymbol{W}^h \boldsymbol{k}_t)\mathbf{1}^T) \quad \in \mathbb{R}^{k \times L}$$

Attention score

$$\boldsymbol{\alpha}_t = \text{softmax}(\boldsymbol{w}^T \boldsymbol{M}_t) \quad \in \mathbb{R}^{1 \times L}$$

Context vector (v_t not included)

$$\boldsymbol{r}_t = [\boldsymbol{v}_{t-L} \cdots \boldsymbol{v}_{t-1}]\boldsymbol{\alpha}^T \quad \in \mathbb{R}^k$$

The final representation

$$\boldsymbol{h}_t^* = \tanh(\boldsymbol{W}^r \boldsymbol{r}_t + \boldsymbol{W}^x \boldsymbol{v}_t) \quad \in \mathbb{R}^k$$

Context vector (v_t not included)

Original vector at index t

$$\mathbf{c}_i = \sum_j a_{ij} \mathbf{h}_j$$

Q K,V

$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

QKV attention

Q (Query), **K** (Key), **V** (value)
V for sending information
Q, K for interaction

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

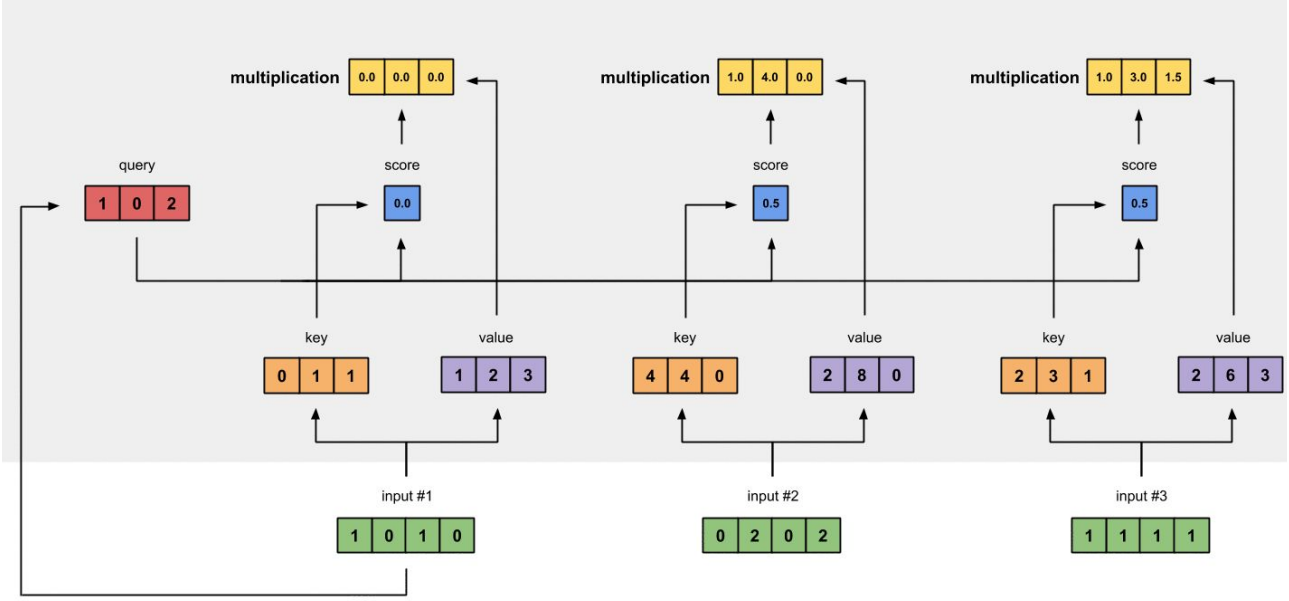
$$Q = W_Q s \quad K = W_K h \quad V = W_V h$$

$$a_{ij} = softmax(\frac{Q_i K_j^T}{\sqrt{d_k}})$$

$$y_i = \sum_j a_{ij} V_i$$

$$Q = W_Q h$$

For self-attention



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

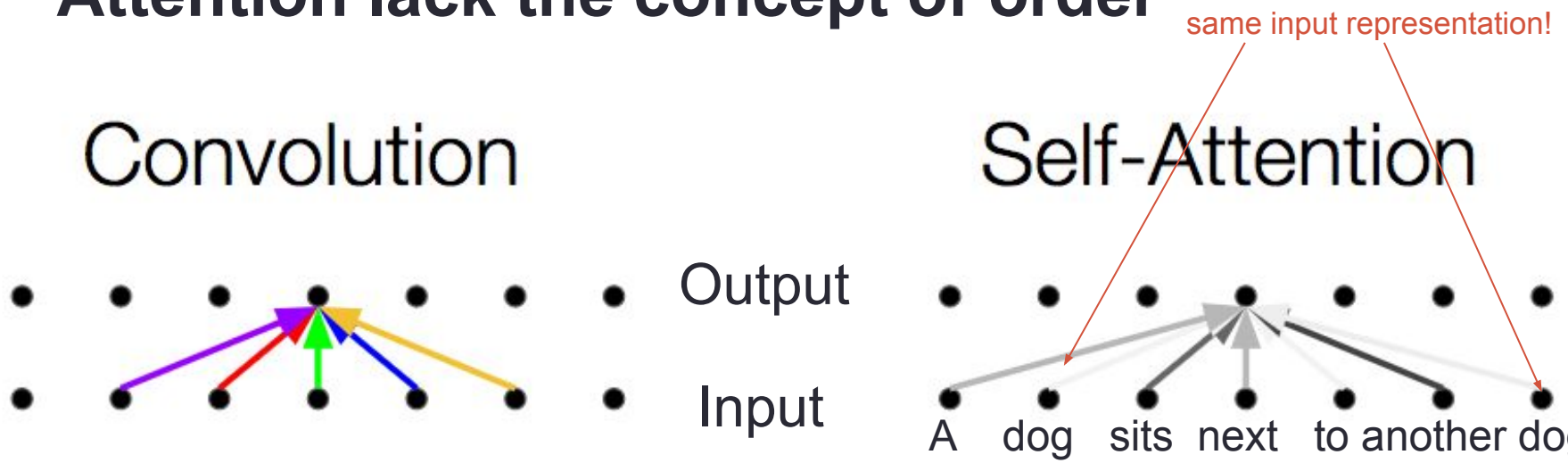
Convolution vs Self attention

Limited context vs Wide context

- Convolution has limited scope (filter size)
- Self-attention has **different learnable weight** for each output
- **Downside: O(n^2) (length^2) when computing attention weights**

Fixed weights vs weights changes according to input

Attention lack the concept of order



https://nlp.stanford.edu/seminar/details/lkaiser.pdf

Computational complexity

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

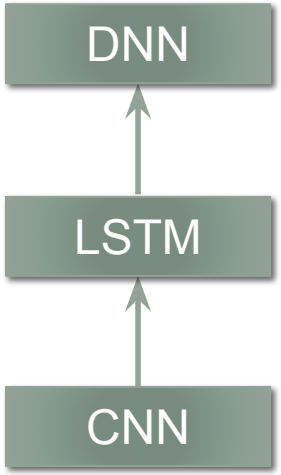
only O(k n d) for 1 filter

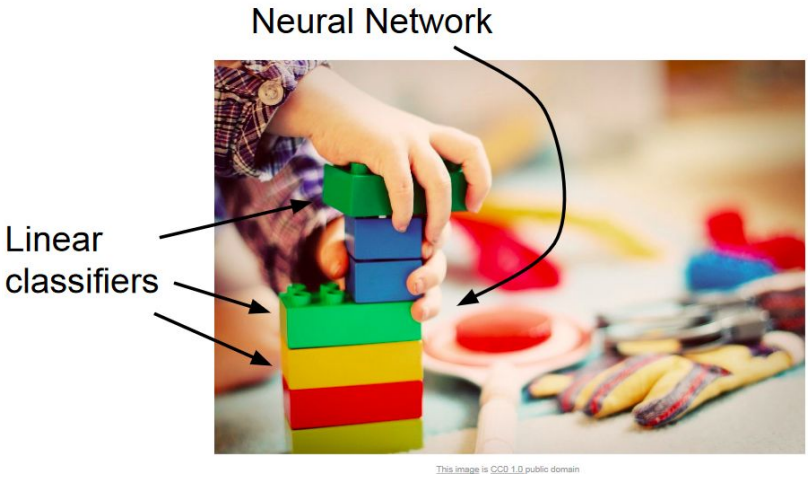
O(n/k) for standard convolution

Back to vision world

DNN Legos

- Typical models now consists of all 3 types
  - CNN: local structure in the feature. Used for feature learning.
  - RNN (GRU/LSTM) : remembering longer term structure or across time
  - DNN: Good for mapping features for classification. Usually used in final layers
  - Attention : remember long term structure / global information
- DNN structures encode inductive bias

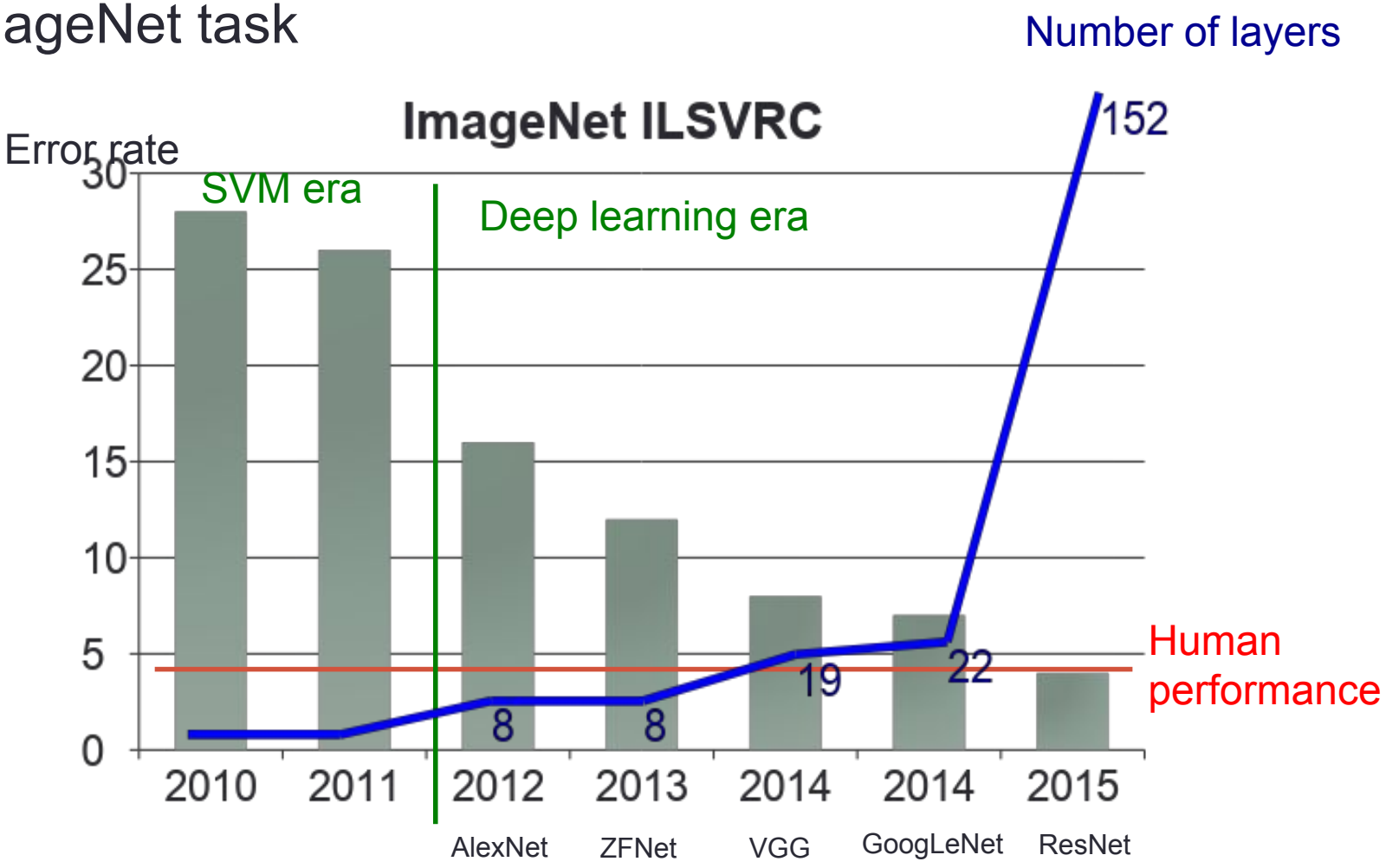




https://www.analyticsvidhya.com/blog/2017/08/10-advanced-deep-learning-architectures-data-scientists/

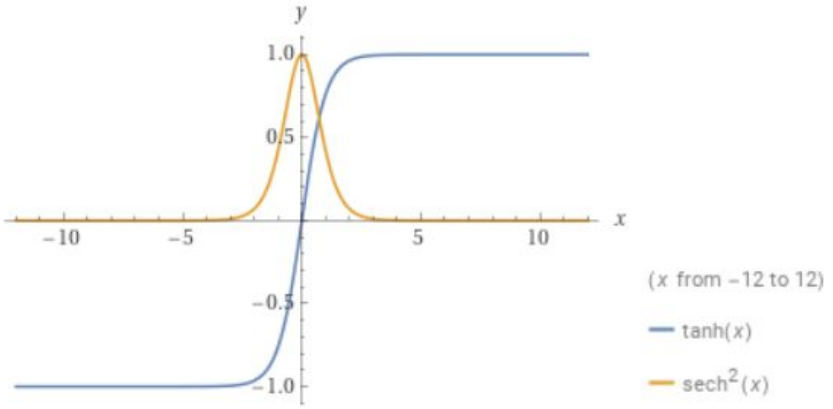
A brief history of imagenet architectures

- ImageNet task



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

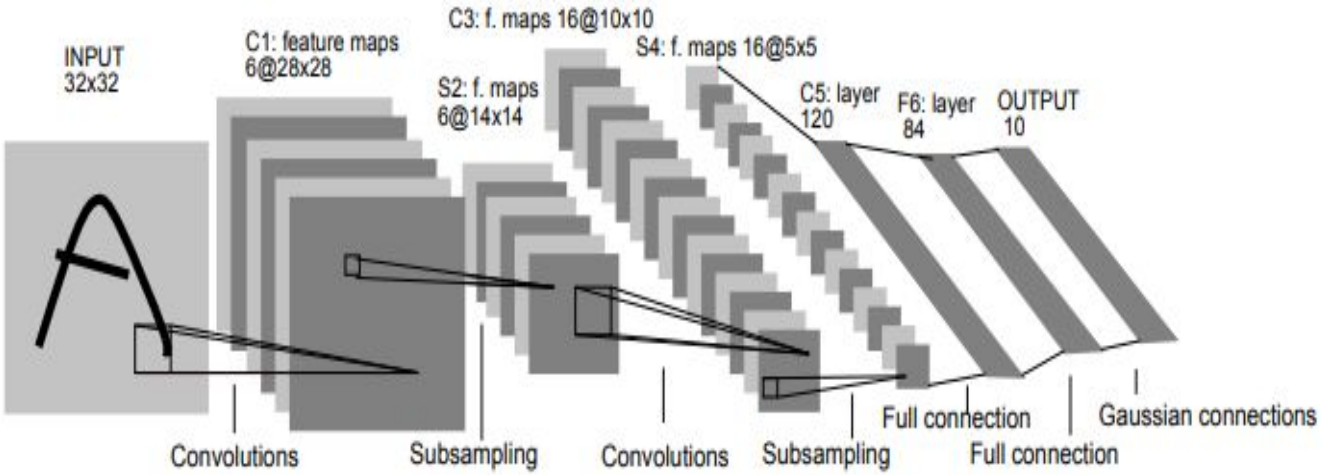
LeNet



Convolutions and poolings followed by fully connected layers

Tanh activations (zero-centering)

Ability to handle larger images limited by compute



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition,” 1998.

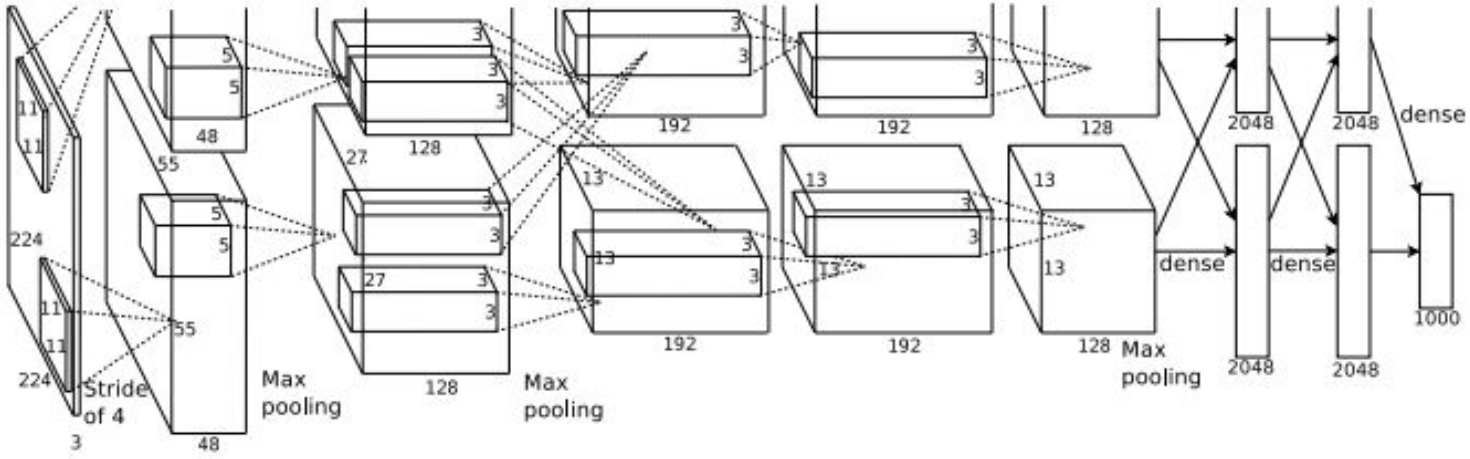
AlexNet

Doesn't make very much sense in computer vision

non-zero centering (potential gradient explosion)

Convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum

Two pipelines to fit into two GPUs (GTX 580, 3GB, 2010)



Alex Krizhevsky, et al. "ImageNet Classification with Deep Convolutional Neural Networks", 2012.

Fun facts

Apparently, Schmidhuber did it on GPU first (2011)

Flexible, High Performance Convolutional Neural Networks for Image Classification

Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, Jürgen Schmidhuber
IDSIA, USI and SUPSI
Galleria 2, 6928 Manno-Lugano, Switzerland
{dan,ueli,jonathan,luca,juergen}@idsia.ch

Abstract

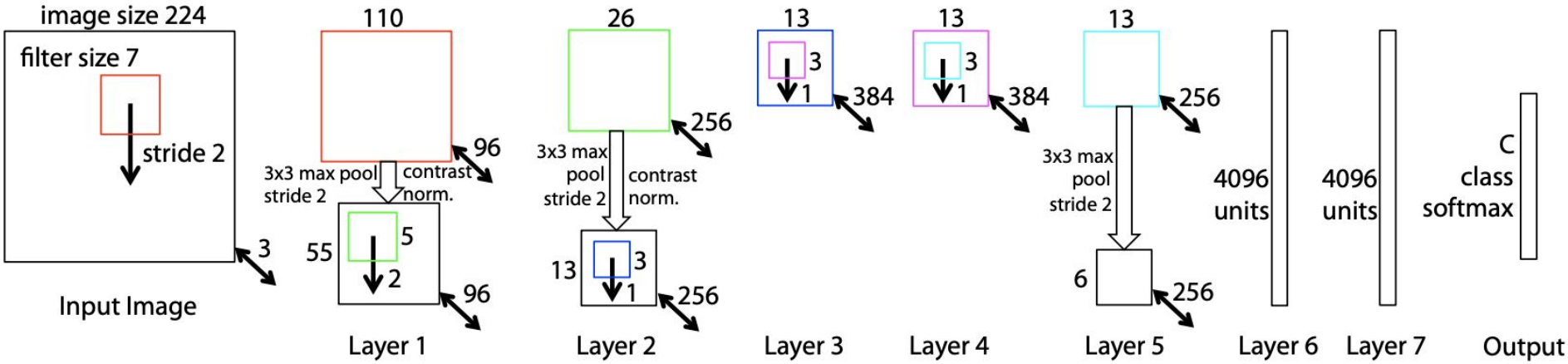
We present a fast, fully parameterizable GPU implementation of Convolutional Neural Network variants. Our feature extractors are neither carefully designed nor pre-wired, but rather learned in a supervised way. Our deep hierarchical architectures achieve the best published results on benchmarks for object classification (NORB, CIFAR10) and handwritten digit recognition (MNIST), with error rates of 2.53%, 19.51%, 0.35%, respectively. Deep nets trained by simple back-propagation perform better than more shallow ones. Learning is surprisingly rapid. NORB is completely trained within five epochs. Test error rates on MNIST drop to 2.42%, 0.97% and 0.48% after 1, 3 and 17 epochs, respectively.

(CNNs) [LeCun *et al.*, 1998; Behnke, 2003; Simard *e*
2003], whose weights (filters) are randomly initialized
changed in a supervised way using back-propagation (B

Despite the hardware progress of the past decades, co
tational speed is still a limiting factor for CNN architec
characterized by many building blocks typically set by
and error. To systematically test the impact of various a
tectures on classification performance, we present a fast
implementation on Graphics Processing Units (GPUs). P
ous GPU implementations of CNNs [Chellapilla *et al.*, 2
Uetz and Behnke, 2009; Strigl *et al.*, 2010] were hard-c
to satisfy GPU hardware constraints or use general pur
libraries, whereas our implementation is flexible and full
line (i.e., weight updates after each image). A notable ex
tion is [Jarrett *et al.*, 2009] who performed a thorough a
sis of the influence of all building blocks of a multistag
chitecture on recognition performance. Our implement

ZFNet

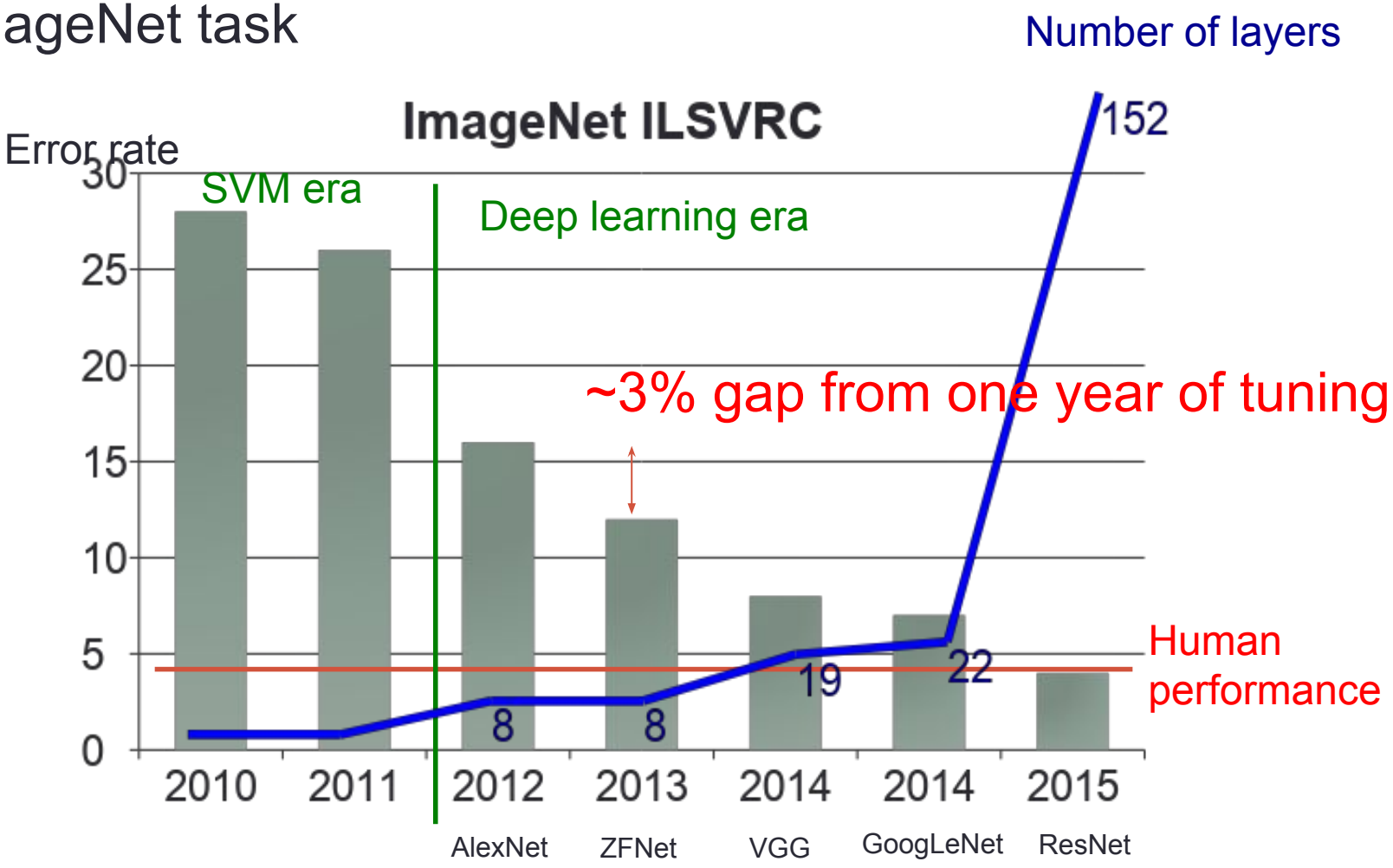
- Tweaking hyperparameters
- same GPU (GTX 580) but use only one



Matthew D. Zeiler, Rob Fergus, “Visualizing and Understanding Convolutional Networks,” 2013

A brief history of imagenet architectures

- ImageNet task



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

VGG

5x5 filter used 2.8x times more parameter than 3x3 one

Uniform 3x3 convolutional filters **(simplistic design)**

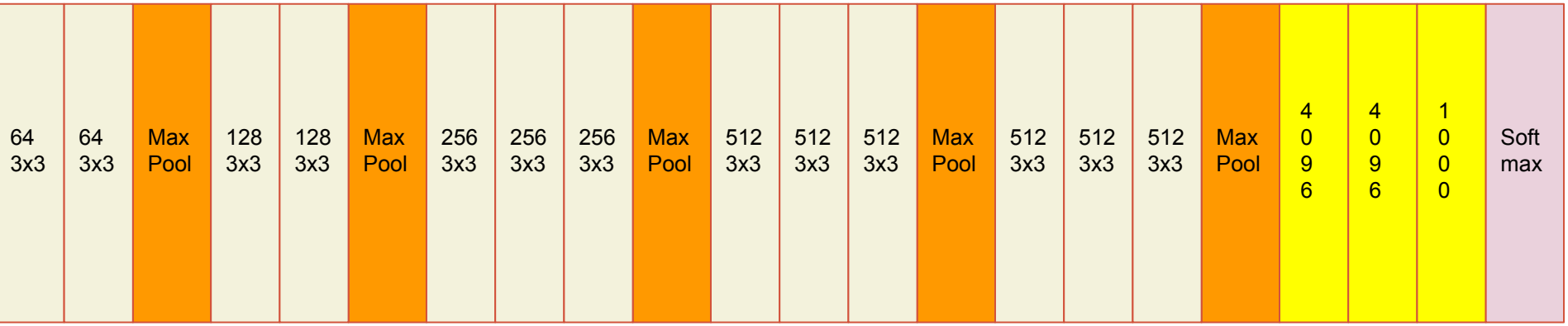
19 layers! Pushing the limits of conventional wisdom at that time.

- Initialization is very important! (Before good initializer like He, Glorot)

Used by many since pre-train weights are publically available

Trained using 4 GTX TITAN Black ! (6 GB each)

VGG16



Karen Simonyan, Andrew Zisserman"Very Deep Convolutional Networks for Large-Scale Image Recognition" 2014

AlexNet Recap

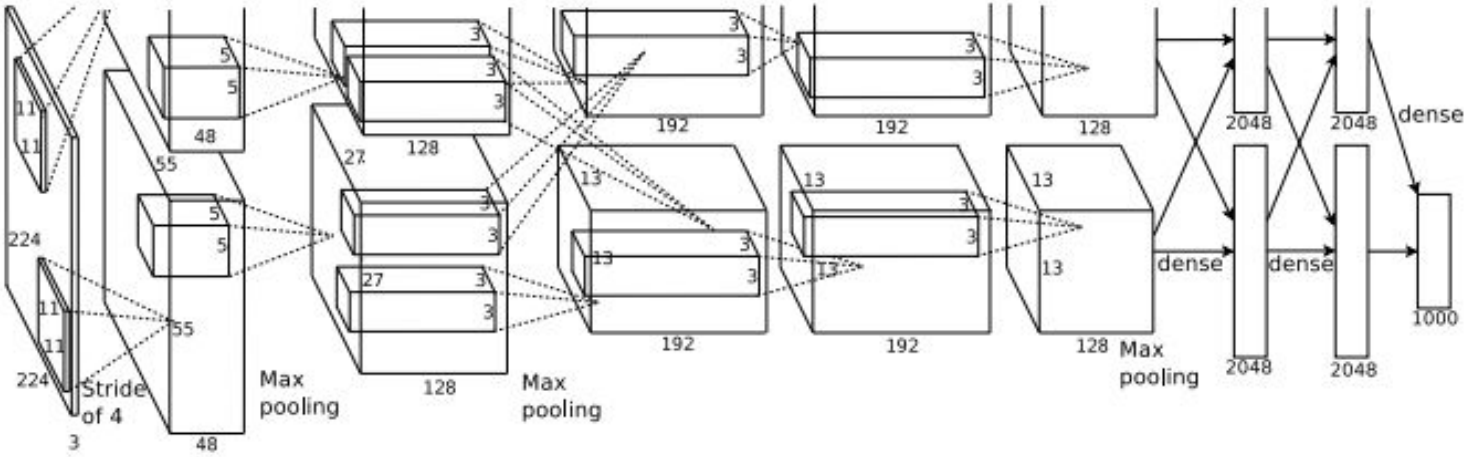
Replaced by Conv with strides

Doesn't make very much sense in computer vision

non-zero centering (potential gradient explosion)

Convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum

Two pipelines to fit into two GPUs (GTX 580, 3GB, 2010)

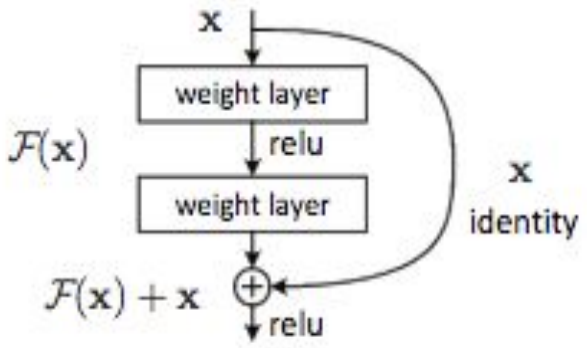


Alex Krizhevsky, et al. "ImageNet Classification with Deep Convolutional Neural Networks", 2012.

Vanishing/Exploding gradient

- Backprop introduces many multiplications down chain
- The gradient value gets smaller and smaller
  - The deeper the network the smaller the gradient in the lower layers
  - Lower layers changes too slowly (or not at all)
  - Hard to train very deep networks (>6 layers)
- The opposite can also be true. The gradient explodes from repeated multiplication
  - Put a maximum value for the gradient (Gradient clipping)

- How to deal with this?
  - Residual connection
  - Gradient injection



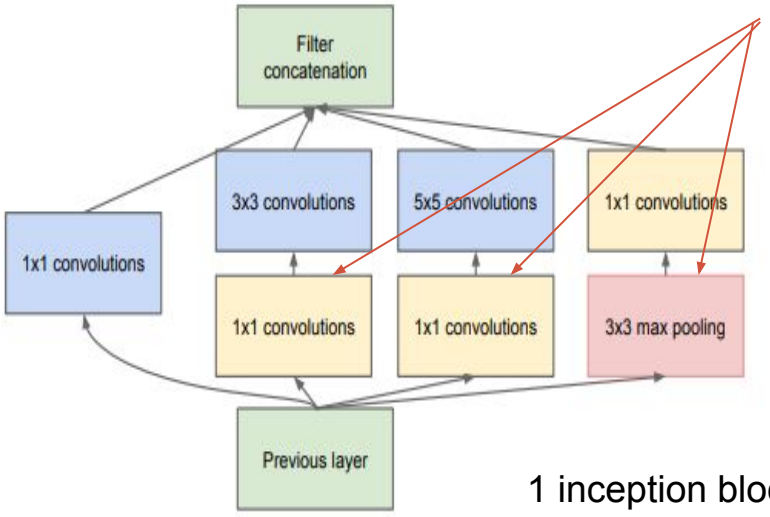
https://arxiv.org/abs/1512.03385

GoogLeNet (Inception v1)

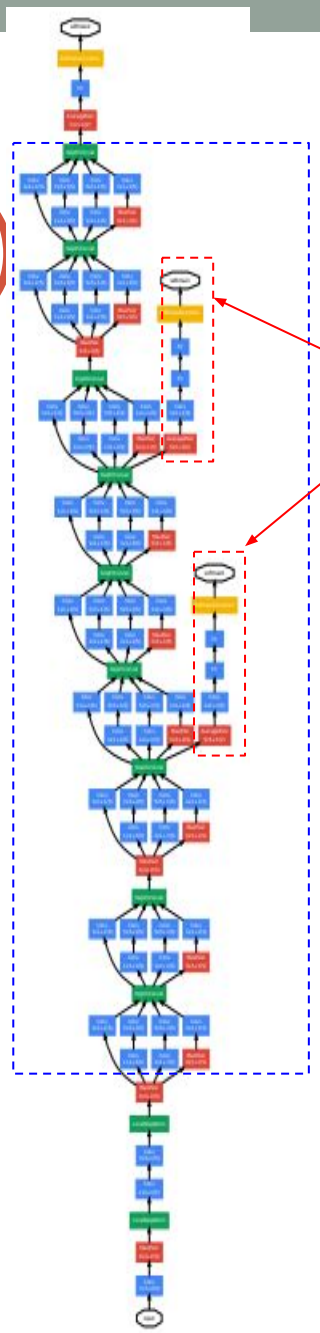
Multiple filter sizes per layer (objects come in different scales)

Dimensionality reduction via 1x1 convolution

Multiple softmax losses to help the gradient problem

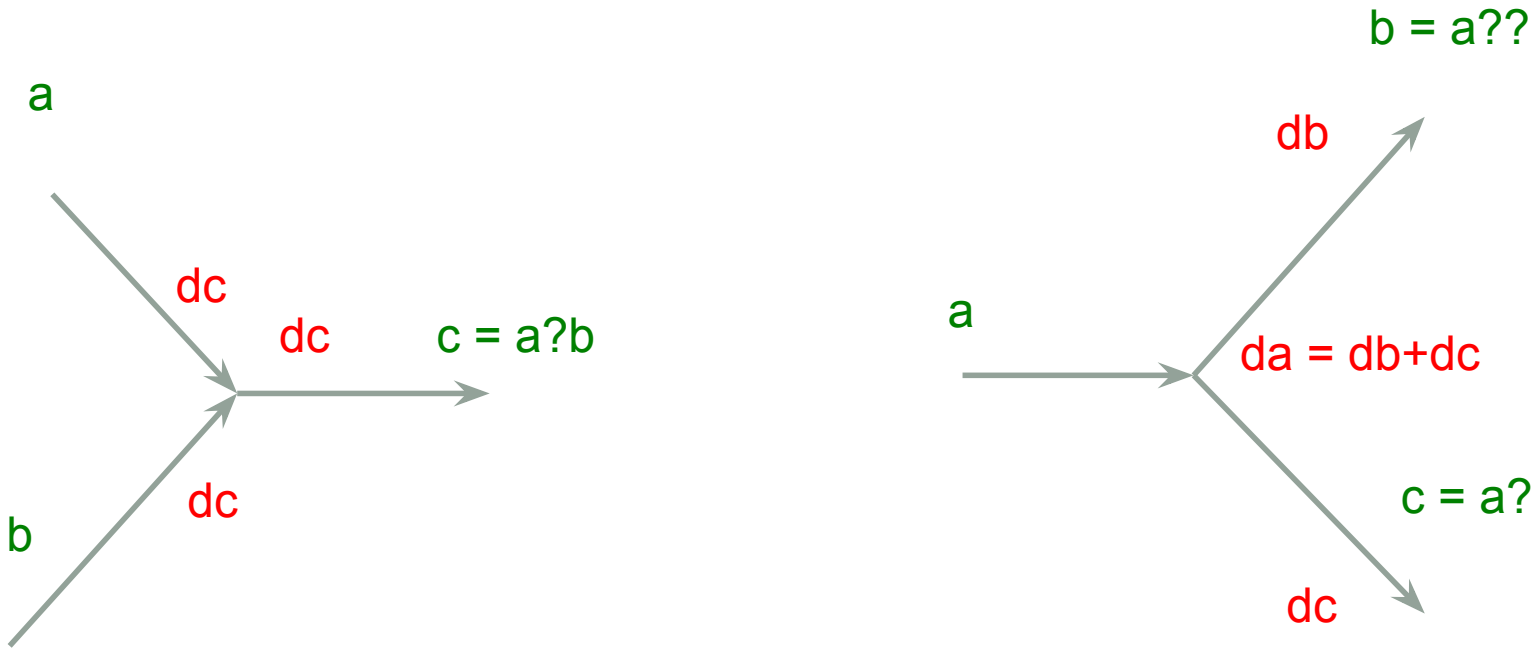


1 inception block



Christian Szegedy, et al. “Going Deeper with Convolutions,” 2014

Gradient flow at forks

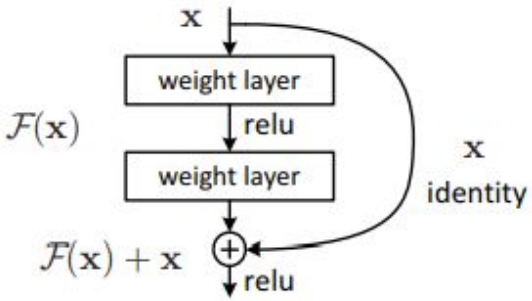


Forward and backward pass acts differently at forks

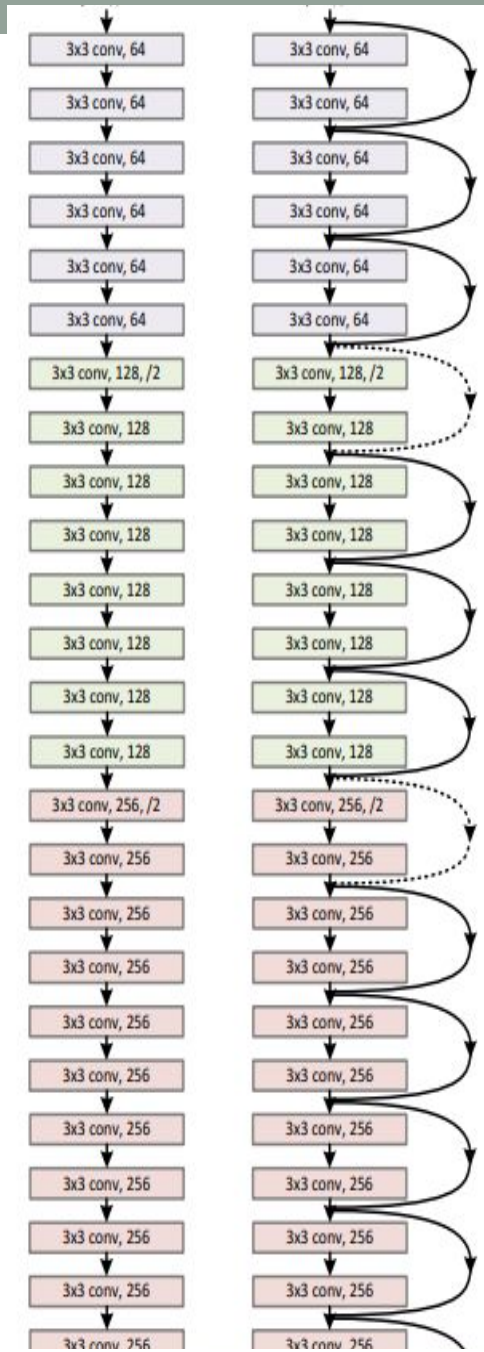
ResNet (Residual Network)

Batch norm

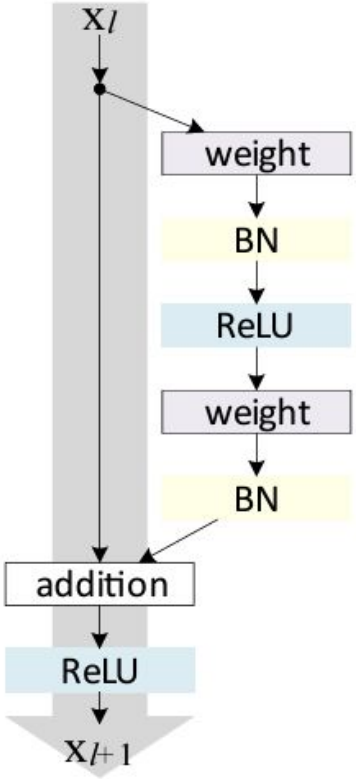
Extra “skip connections” to reduce the vanishing gradient problem



$$y = F(x) + x \rightarrow F(x) = y - x$$



Need extra operations to make the size the same



(a) original

Kaiming He, et al. “Deep Residual Learning for Image Recognition” 2015

Batch normalization

- Recent technique for (implicit) regularization
- Normalize every mini-batch at various batch norm layers to standard Gaussian (different from global normalization of the inputs)
- Place batch norm layers before (or after?) non-linearities
- Faster training (put numbers in expected range – can use larger learning rates) and better generalizations

For each mini-batch that goes through batch norm

1. Normalize by the mean and variance of the mini-batch for each dimension
2. Shift and scale by learnable parameters

$$\hat{x} = \frac{x - \mu_b}{\sigma_b}$$

$$y = \alpha\hat{x} + \beta$$

Replaces dropout in some networks

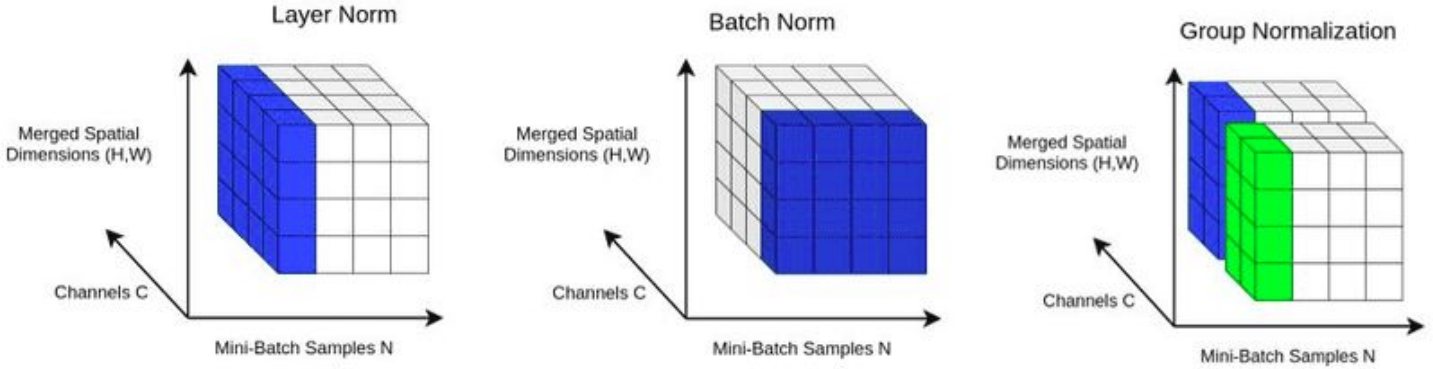
https://arxiv.org/abs/1502.03167

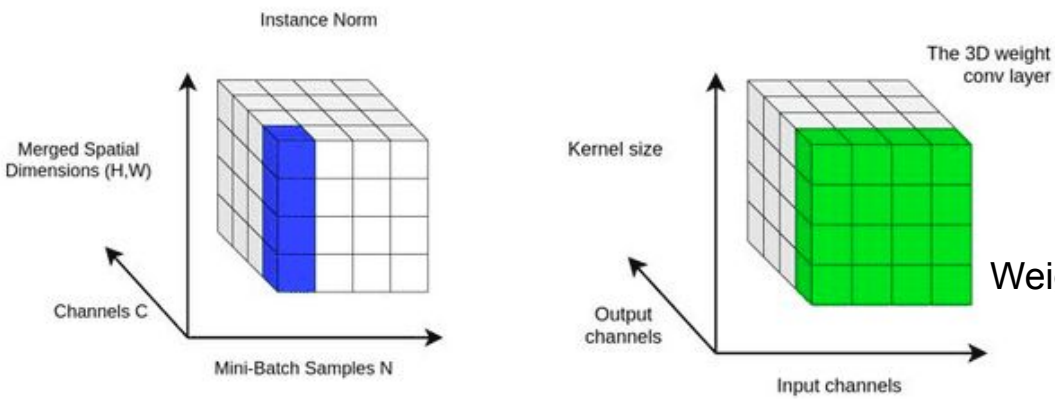
Mean changes every minibatch – a form of noise (regularization)

Other normalizations

- Other normalizations are out there

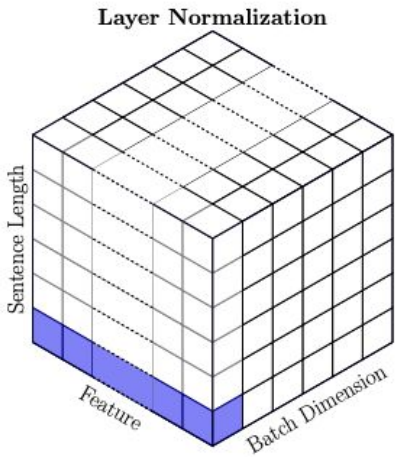
NLP and CV layer norm are not the same (In NLP, layer norm is applied separately for each element in sequence)





Weight norm

Layer norm in transformers

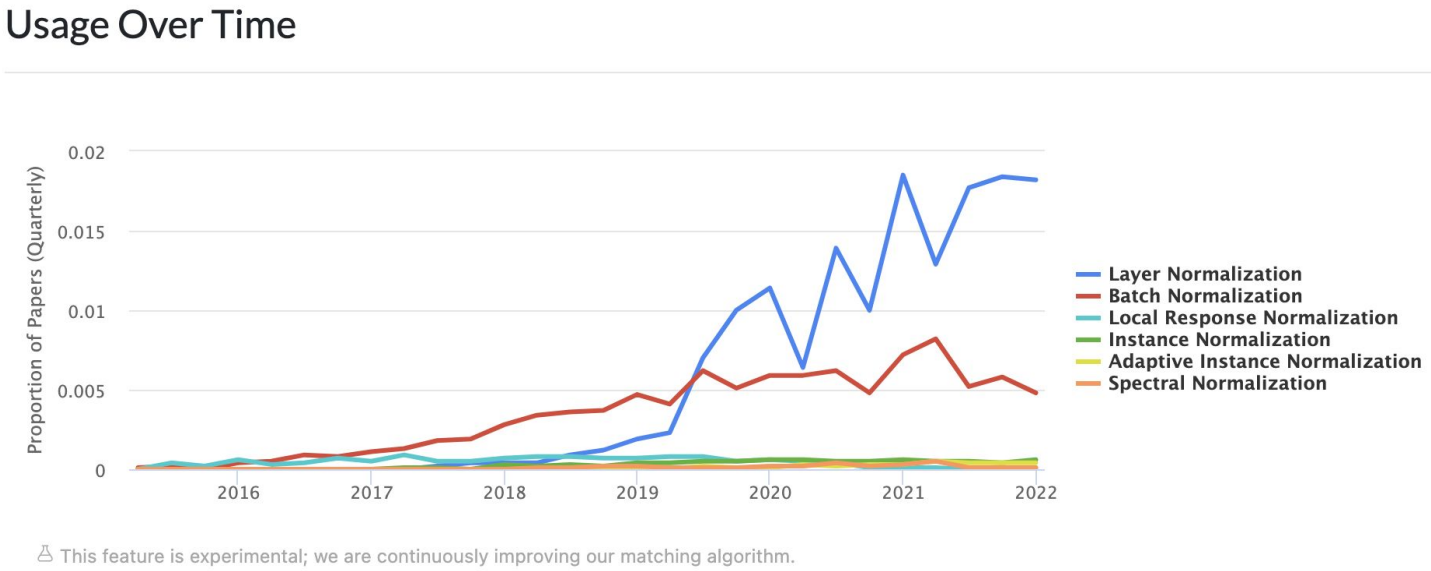


https://proceedings.mlr.press/v119/shen20e/shen20e.pdf

https://theaisummer.com/normalization/

Other normalizations

Trend of normalization used in papers



NLP – layer norm
Image – Batch norm/layer norm
Small batches - Group norm (batch norm is not reliable in small batches)

https://paperswithcode.com/method/layer-normalization

Training and inference of batch norm

BATCHNORM2D

CLASS `torch.nn.BatchNorm2d(`*`num_features`*, *`eps=1e-05`*, *`momentum=0.1`*, *`affine=True`*, *`track_running_stats=True`*, *`device=None`*, *`dtype=None`*`)` [SOURCE]

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

The mean and standard-deviation are calculated per-dimension over the mini-batches and γ and β are learnable parameter vectors of size *C* (where *C* is the input size). By default, the elements of γ are set to 1 and the elements of β are set to 0. The standard-deviation is calculated via the biased estimator, equivalent to *torch.var(input, unbiased=False)*.

Also by default, during training this layer keeps running estimates of its computed mean and variance, which are then used for normalization during evaluation. The running estimates are kept with a default `momentum` of 0.1.

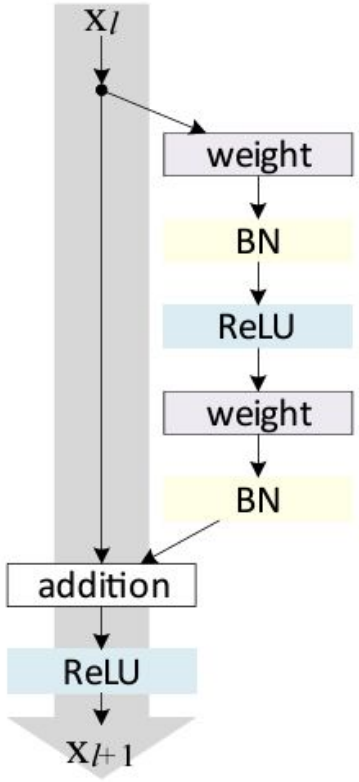
If `track_running_stats` is set to `False`, this layer then does not keep running estimates, and batch statistics are instead used during evaluation time as well.

https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

Standardized Architecture

CNN architectures are more standardized after ResNet

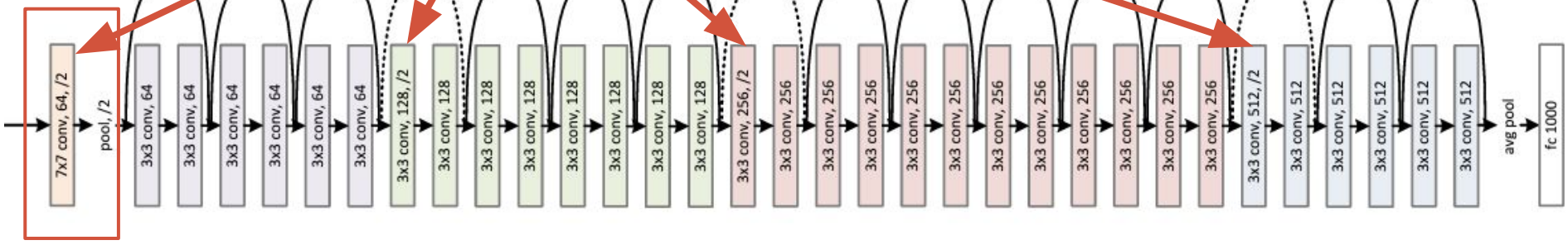
- Big stem to quickly downsample
- small filter (3x3 or 1x1)
- Five stages model (downsampling factor of 32) with weighted pooling
- Global average pooling at the end
- Conv-BN-Relu block



(a) original

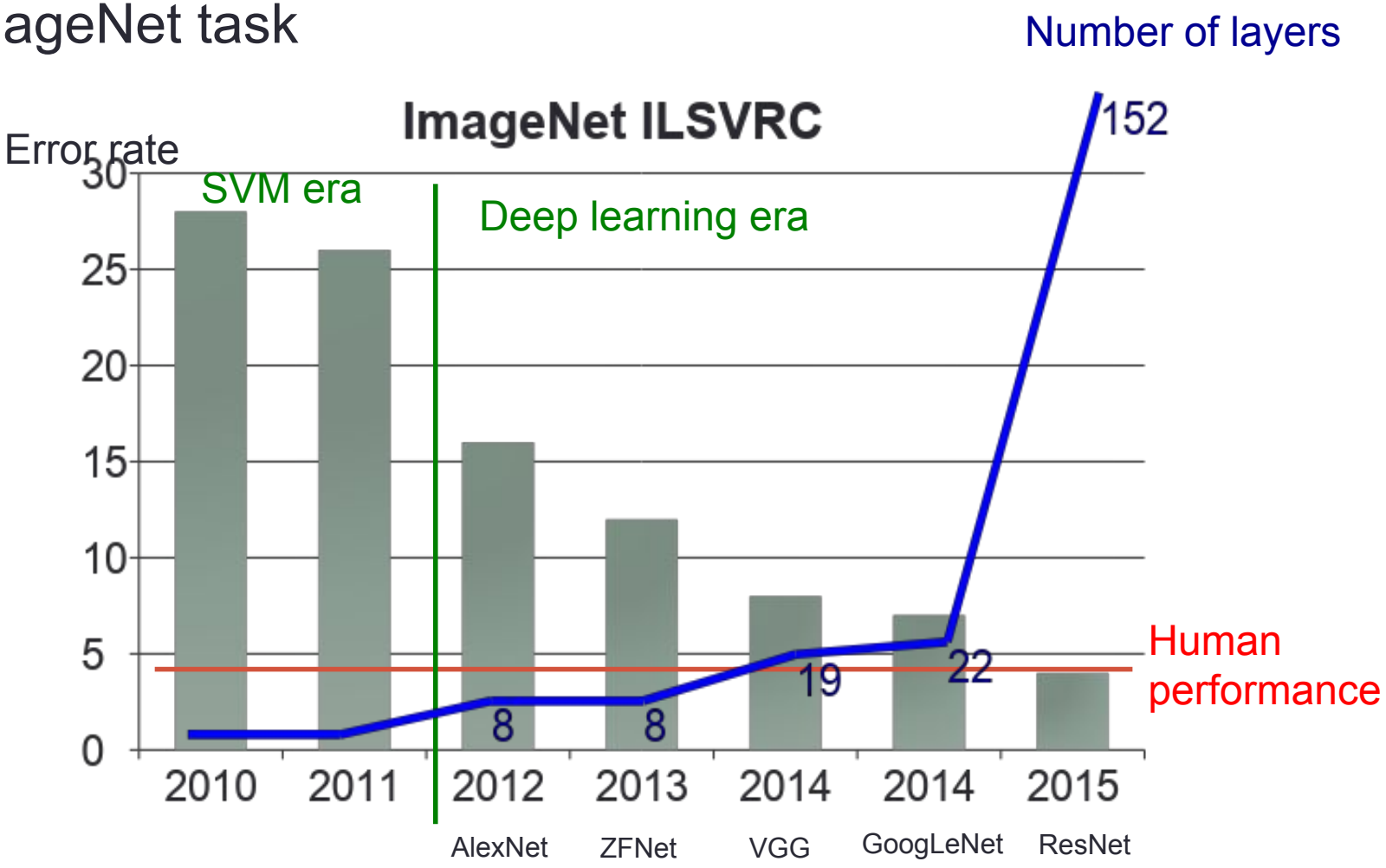
Convolution with strides

Stem



A brief history of imagenet architectures

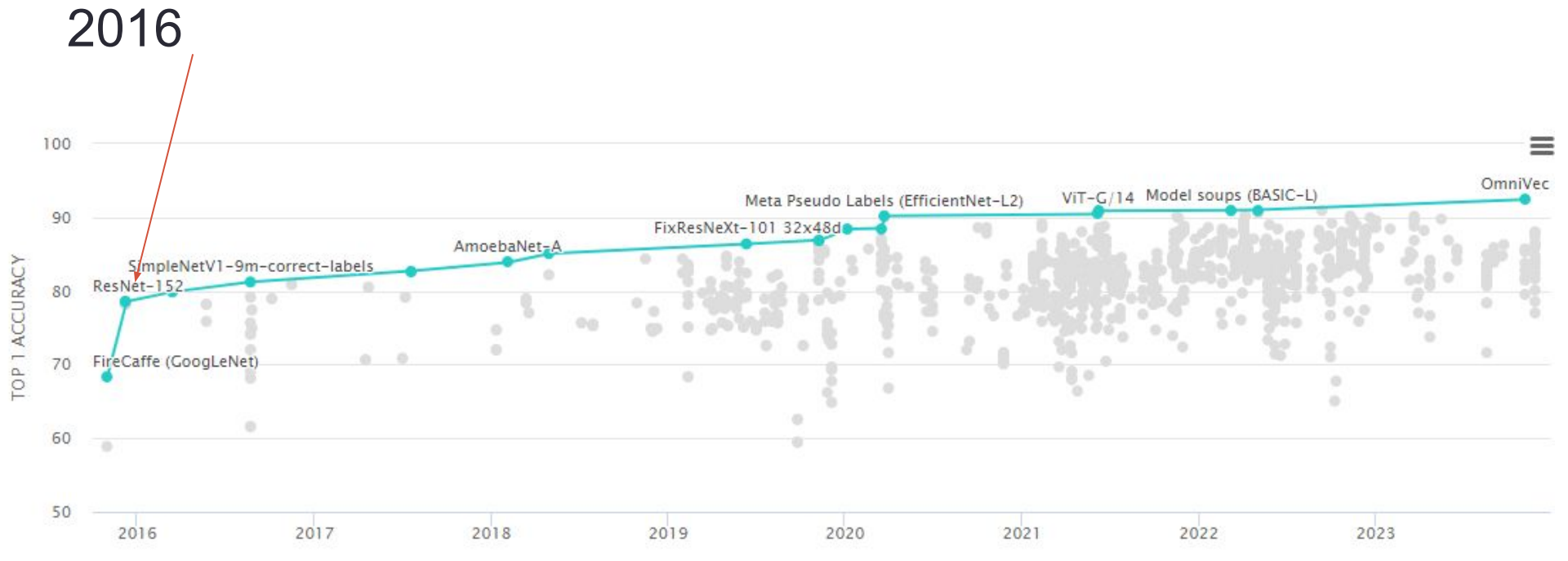
- ImageNet task



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Trends after ResNet

- More efficient convolutional block design
- Wider networks, deeper models, denser connections
- Global Context + Transformer architecture
- Better representation learning (self-supervision + multimodality)



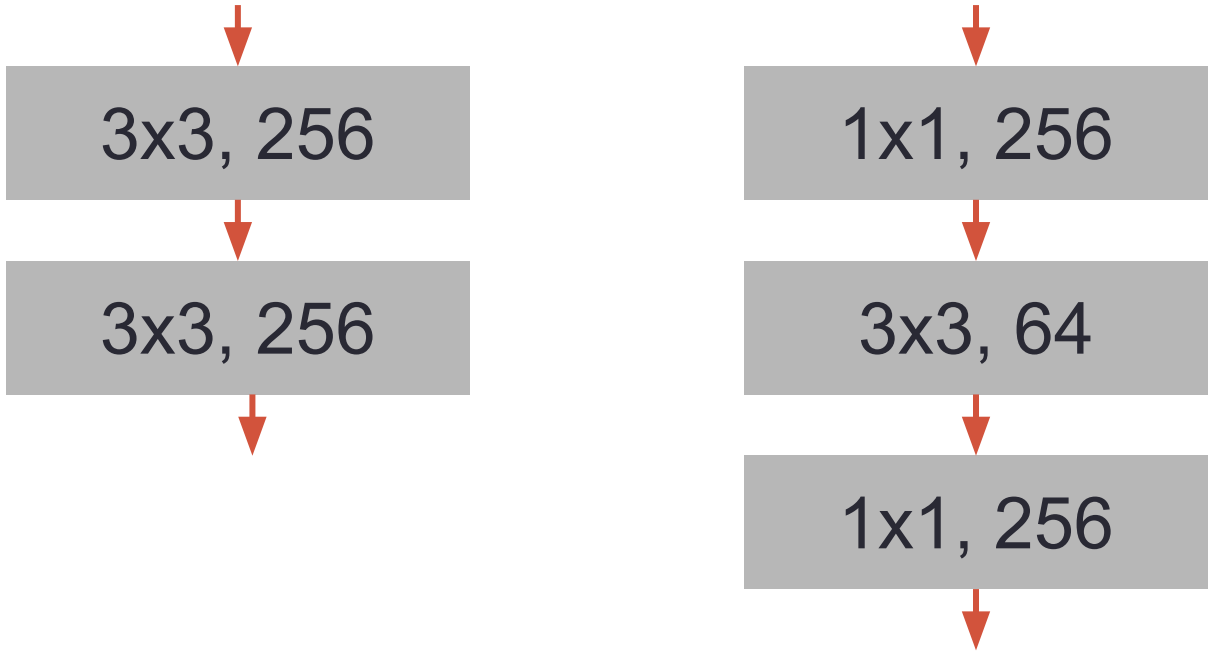
Convolutional block design

- 1x1 convolution
- factorized convolution
- group convolution

More efficient convolution block

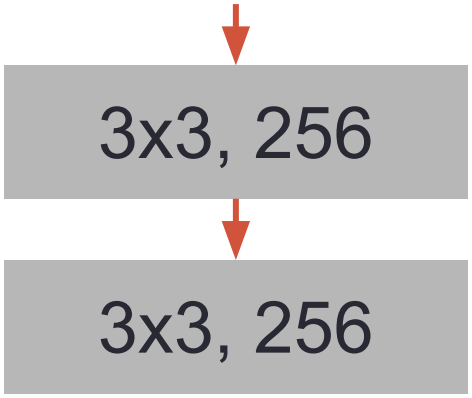
Consider Input tensor of size (W x H x 64)

Q: which block has more parameters?

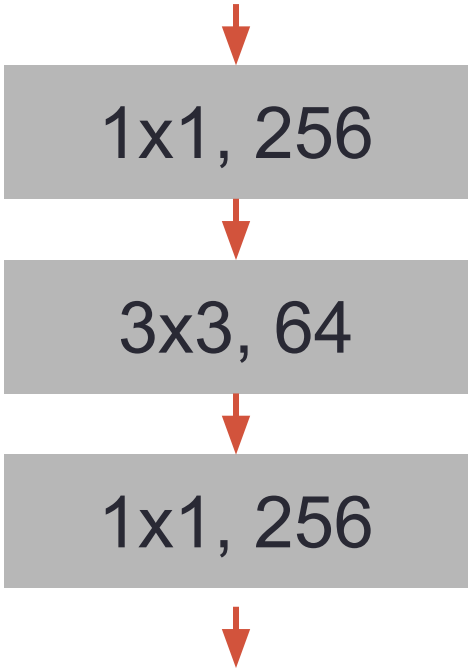


More efficient convolution block

Parameters : 3x3x64x256 +
3x3x256x256
=737,280



Parameters : 1x1x64x256 +
3x3x256x64 +
1x1x64x256
= 180,224



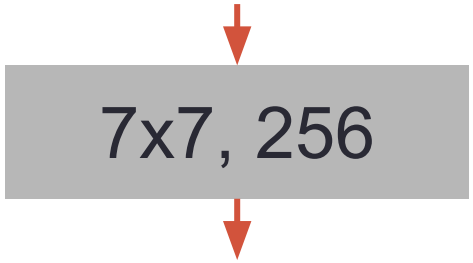
The one on the right use 4 times less parameters

Better Tradeoff!

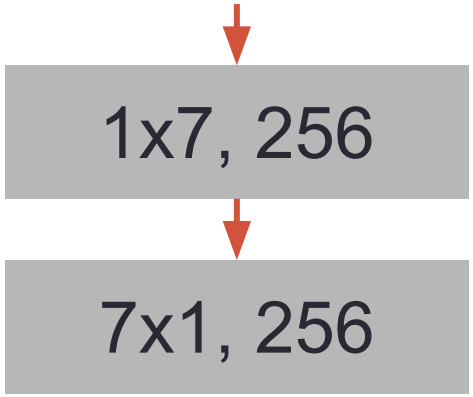
Bottleneck block

Factorized convolution

- k x k convolution vs 1 x k then k x1 convolution
- Not efficient on small filter



Parameters : 7x7x64x256
=802,816



Parameters : 1x7x64x256 +
7x1x256x256
= 573,440

Consider Input tensor of size (W x H x 64)

Inception v2+v3

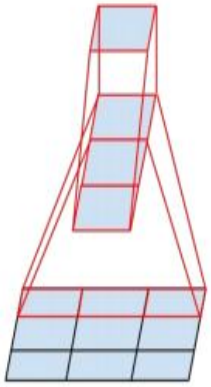
Implement 5x5 with two 3x3s

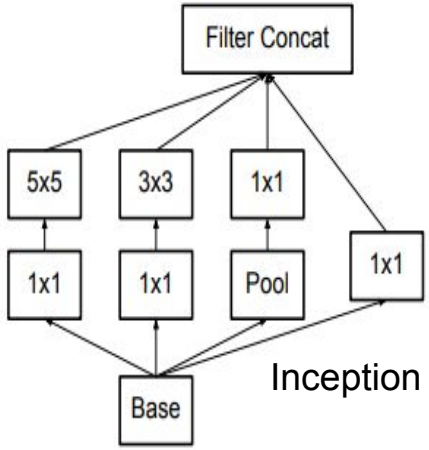
Factorized convolution

3x3 -> 3x1 and 1x3

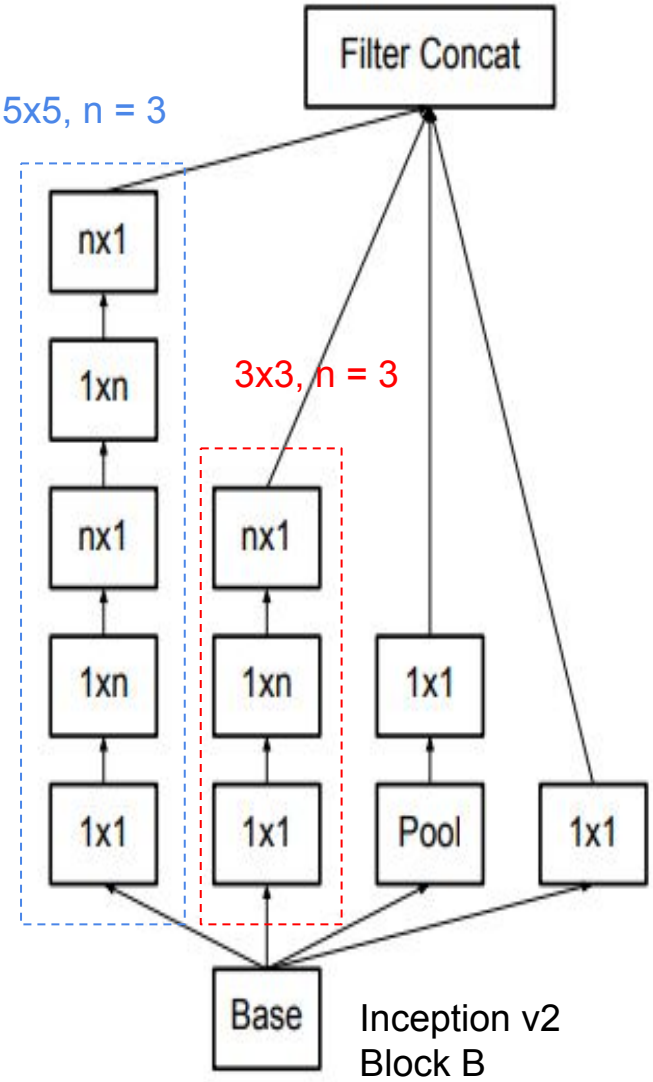
3 types of inception blocks

RMSprop, Batch norm, label smoothing





Inception v1



Inception v2
Block B

Christian Szegedy, “Rethinking the Inception Architecture for Computer Vision,” 2015

Group convolutions

Magic!

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

- ResNext reintroduce concept of group convolutions
- split-transform-merge strategy
- Original : 3x3 filter with d = 256
- Modified : 64 groups of 3x3 filters with d = 4

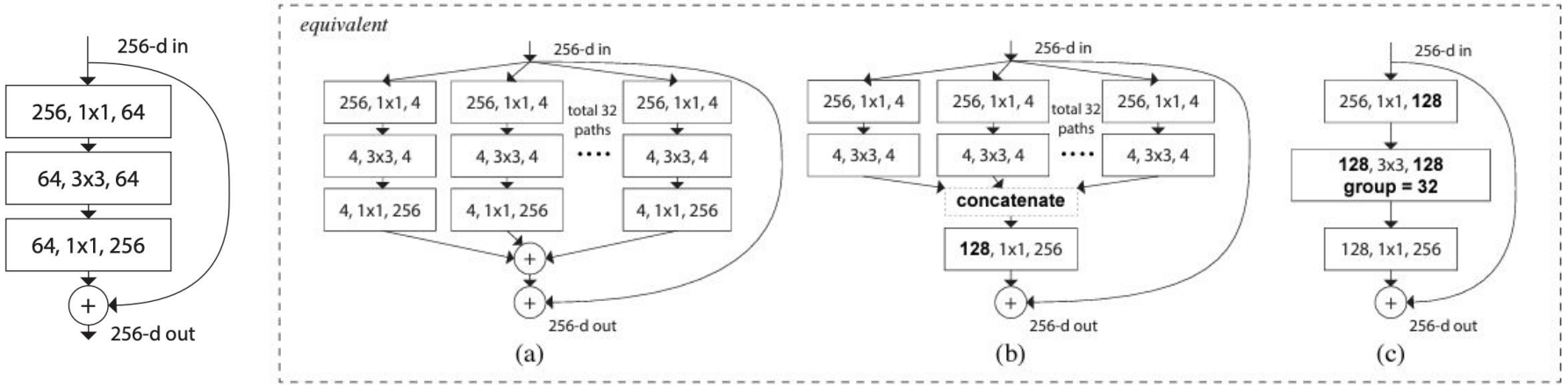


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

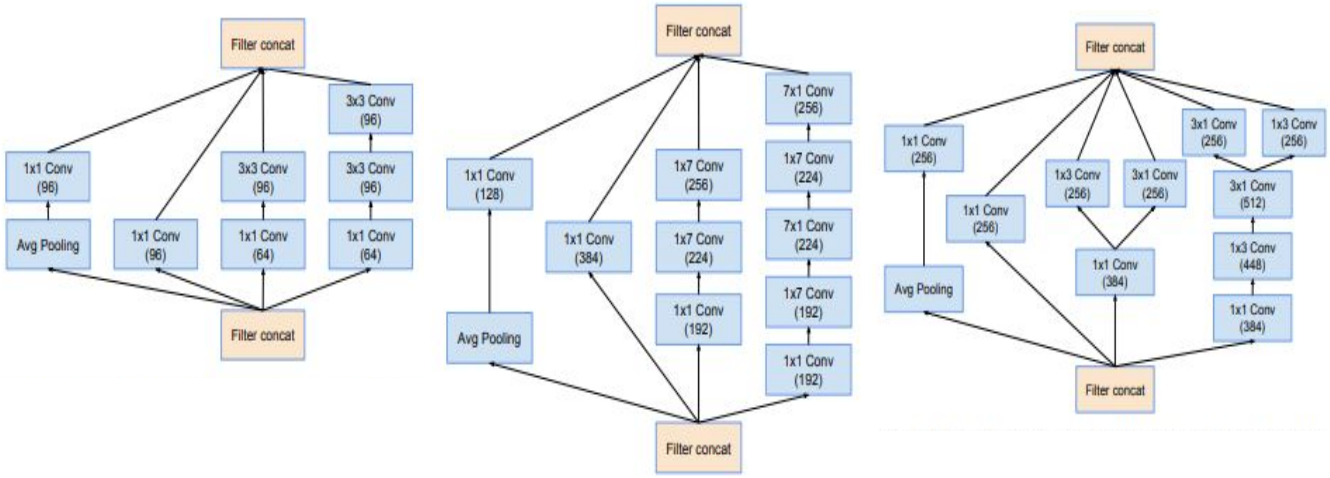
Aggregated Residual Transformations for Deep Neural Networks (2016)

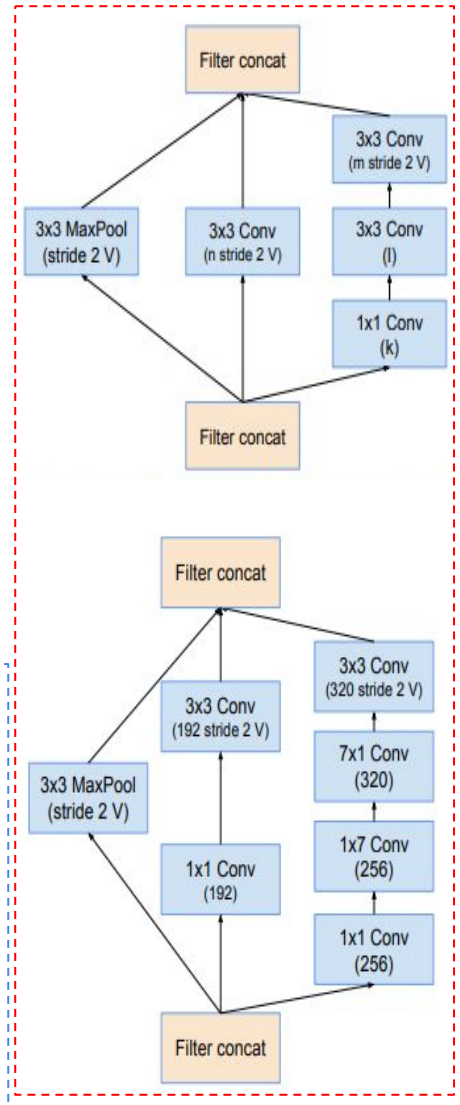
Inception v4

Same three types of inception blocks from v3

Add two types of reduction blocks for reducing the size of the grid (super pooling blocks)

Inception blocks





Reduction blocks

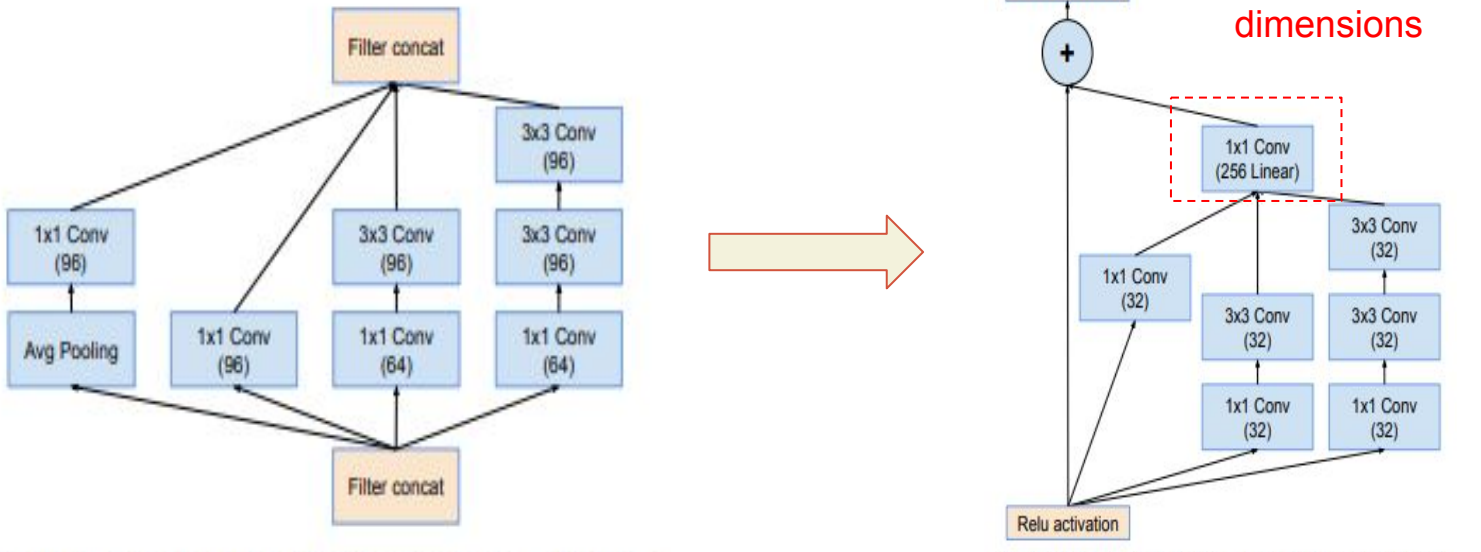
Christian Szegedy, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning” 2016.

Inception-ResNet v1-v2

Introduce residual connections into inception blocks

Poolings changed to additions, 1x1 added to keep dimensions for the residual

Similar idea to ResNeXt

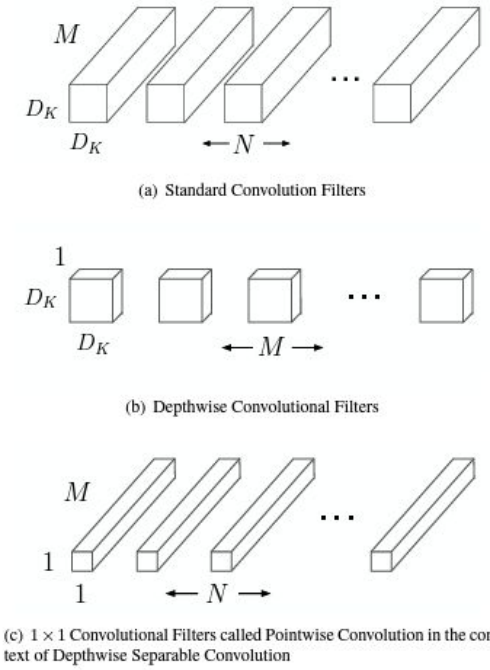


Christian Szegedy, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning” 2016.

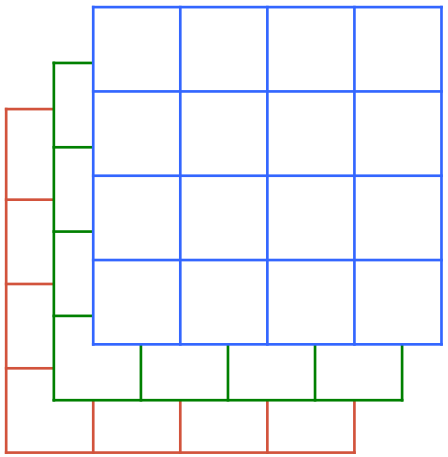
Depthwise separable convolution

Depthwise separable convolution: two-step convolution

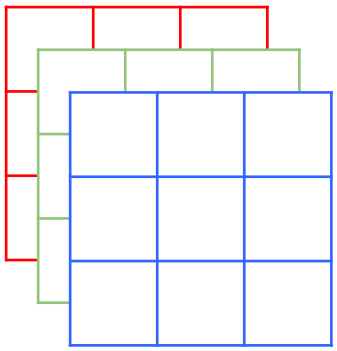
1. Depthwise convolution
   - Could be seen as an extreme version of group convolution (number of groups = number of channels)
2. 1x1 convolution



input



One regular convolution filter



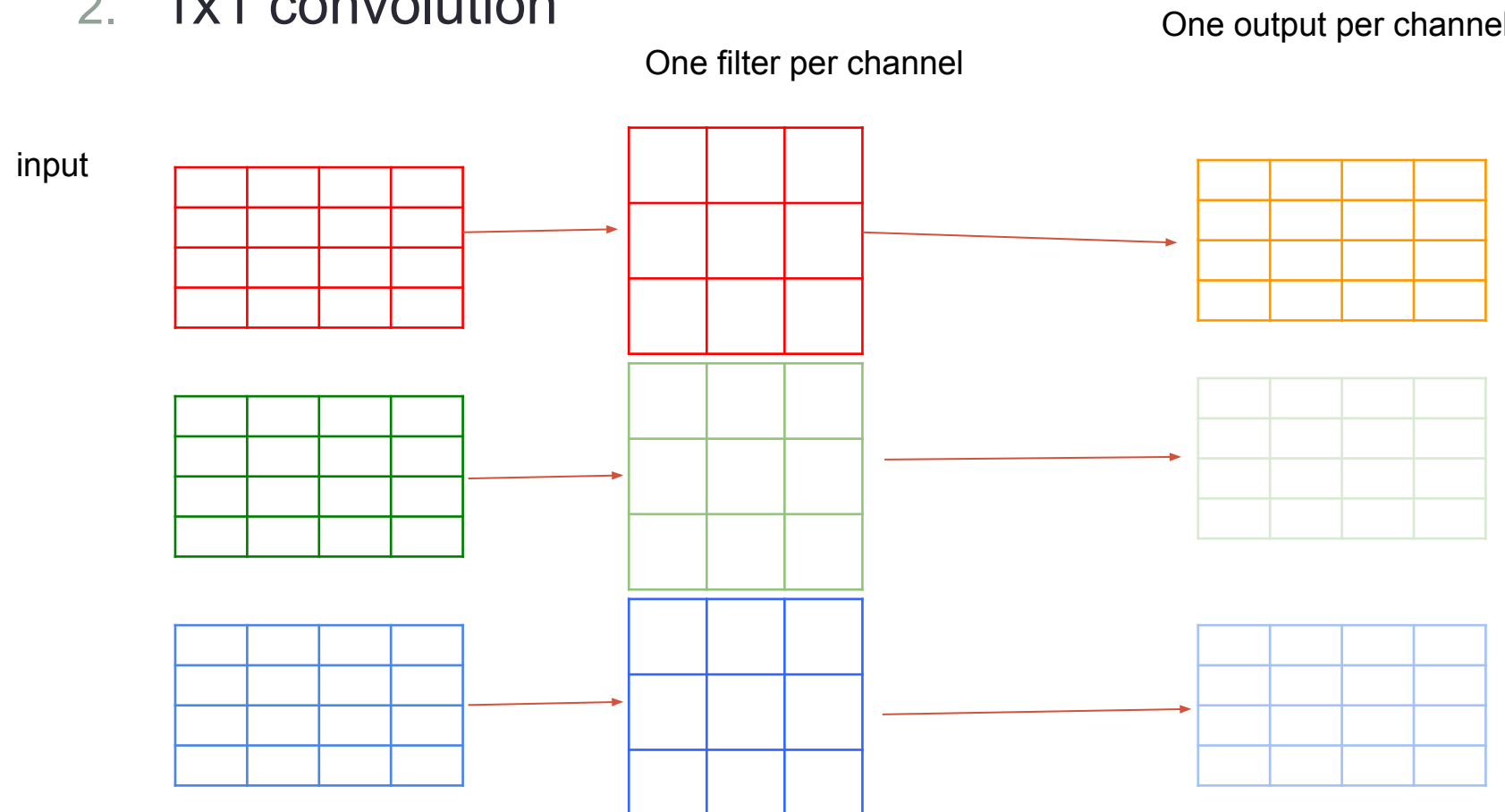
Typical convolution 3x3 filter is 3x3xinput channel

Depthwise convolution 3x3 filter is 3x3x1
1 filter per input channel

Depthwise separable convolution

Depthwise separable convolution: two-step convolution

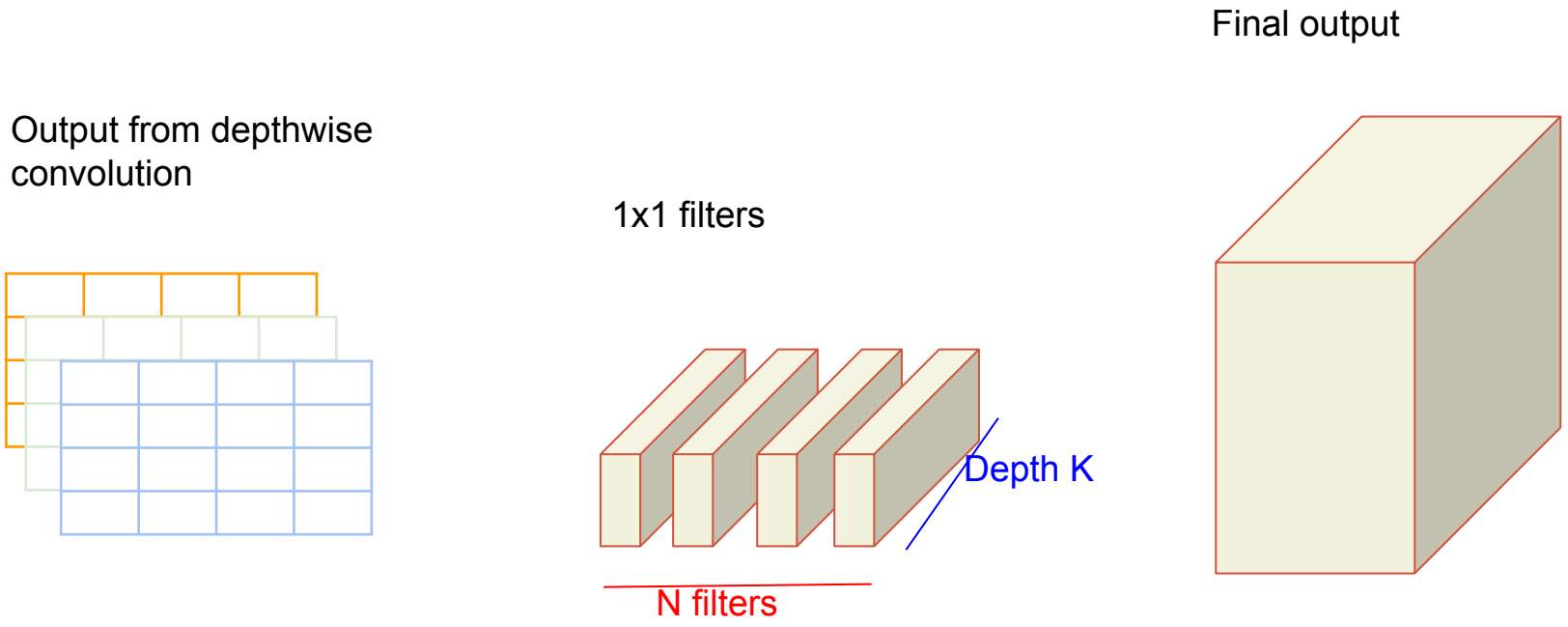
1. Depthwise convolution (learn from each channel, spatial info)
2. 1x1 convolution



Depthwise separable convolution

Depthwise separable convolution: two-step convolution

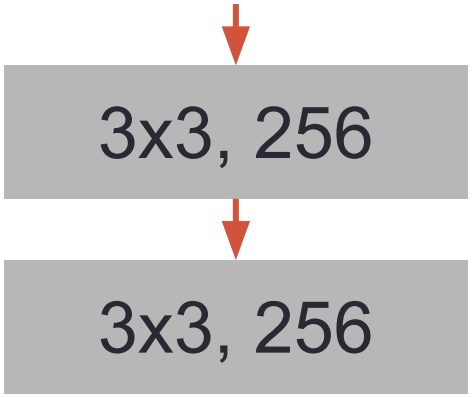
1. Depthwise convolution
2. 1x1 convolution (combine each channel, feature info)



10 3x3 depthwise separable convoltuion will have 3x3x3 + 3x10 = 57 parameters

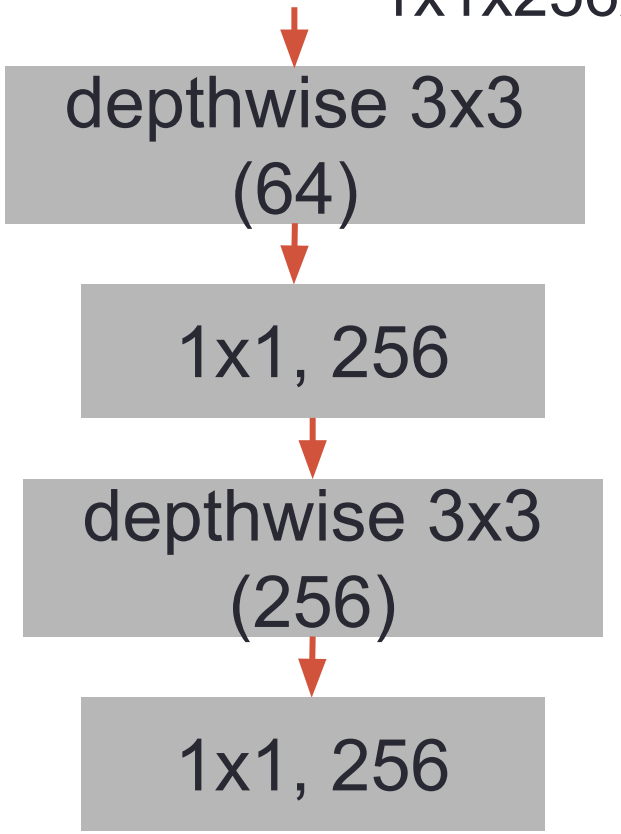
More efficient convolution block (2)

Parameters : 3x3x64x256 +
3x3x256x256
=737,280



Consider Input tensor of size (W x H x 64)

Parameters : 3x3x64 +
1x1x64x256 +
3x3x256 +
1x1x256x256 = 84,800



The one on the right use 9 times less parameters!

Even Better Tradeoff!

Xception

Replace convolutions in inception with depthwise separable convolutions

Smaller model

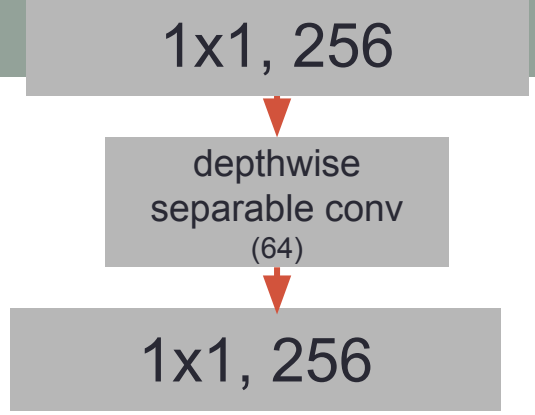
Faster compute

Comparable with Inception v3 while much faster

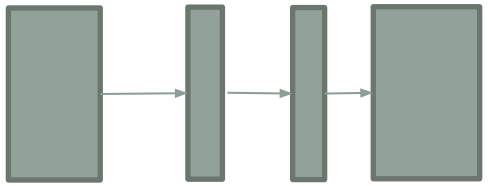
Used in MobileNet and other models

François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions” 2016.
Andrew G. Howard, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” 2017.

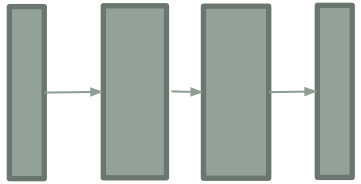
Inverted bottleneck



- A bottleneck block
  - reduce channels using 1x1
  - Regular conv on fewer channels
  - Increase channel using 1x1



- Inverted bottleneck
  - Expand bottleneck instead of shrink)
  - Increase channels using 1x1
  - Regular conv on more channels
  - Reduce channels using 1x1



- Bottleneck combines information between channels
- With residual connections, inverted bottleneck uses less runtime memory during inference (DAG computation graph)

https://arxiv.org/abs/1801.04381

Mobile inverted Bottleneck

- Use depthwise convo instead of conv (MobileNetv2)
- Add Squeeze and Excitation (SE) (EfficientNetv1)
- Can fused 1x1 with depthwise (EfficientNetv2)

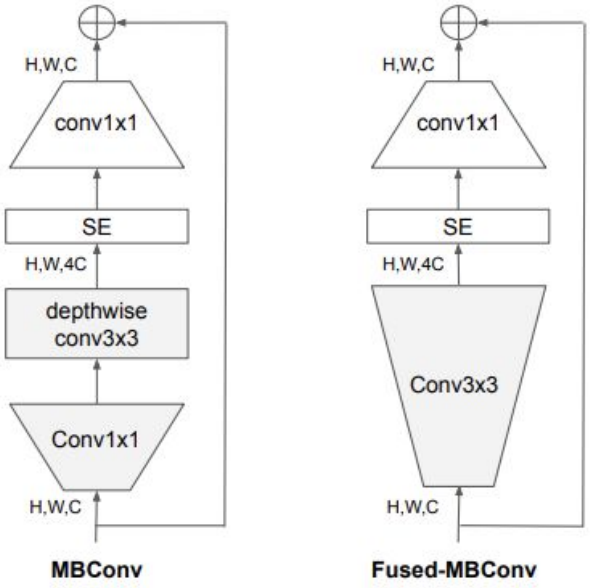
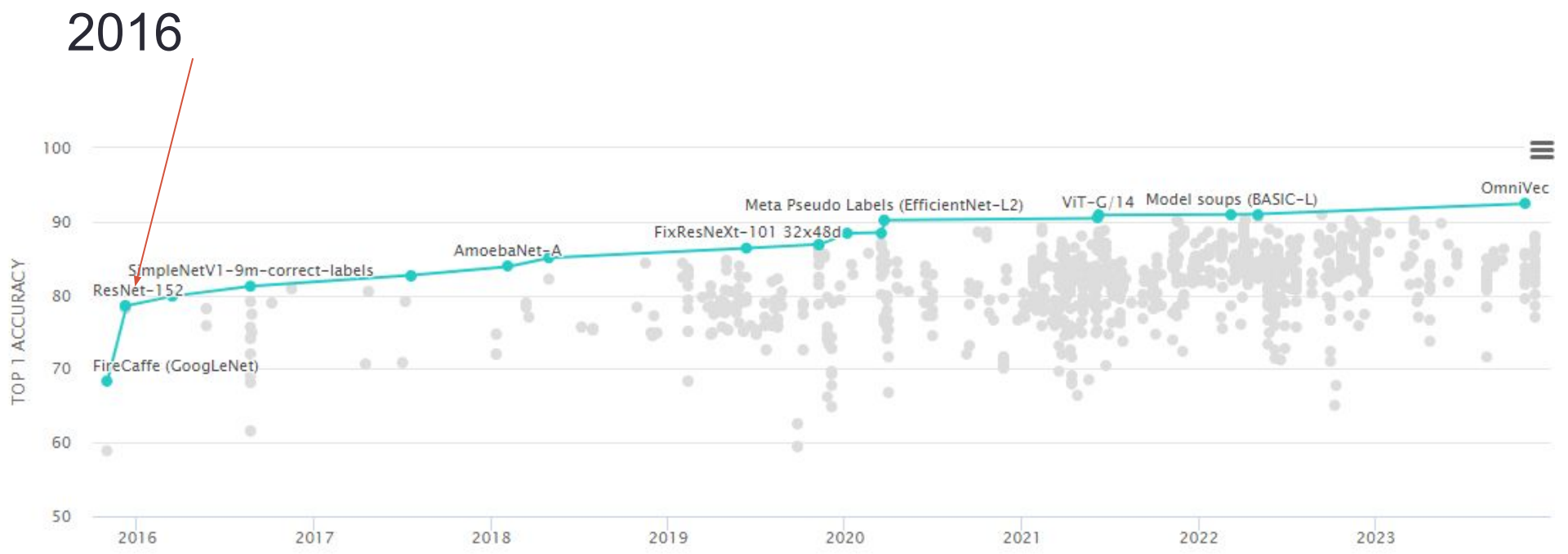


Figure 2. Structure of MBConv and Fused-MBConv.

Trends after ResNet

- More efficient convolutional block design
- **Wider networks, deeper models, denser connections**
- Global Context + Transformer architecture
- Better representation learning (self-supervision + multimodality)



WideResNet

- Find which N, k works the best
- The too-deep but wide network outperforms the deeper networks

group name	output size	block type = $B(3,3)$
conv1	32×32	$[3\times3,\ 16]$
conv2	32×32	$\begin{bmatrix} 3\times3,\ 16\times k \\ 3\times3,\ 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3,\ 32\times k \\ 3\times3,\ 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3,\ 64\times k \\ 3\times3,\ 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Abstract

Deep residual networks were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, and so training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. To tackle these problems, in this paper we conduct a detailed experimental study on the architecture of ResNet blocks, based on which we propose a novel architecture where we decrease depth and increase width of residual networks. We call the resulting network structures wide residual networks (WRNs) and show that these are far superior over their commonly used thin and very deep counterparts. For example, we demonstrate that even a simple 16-layer-deep wide residual network outperforms in accuracy and efficiency all previous deep residual networks, including thousand-layer-deep networks, achieving new state-of-the-art results on CIFAR, SVHN, COCO, and significant improvements on ImageNet. Our code and models are available at https://github.com/szagoruyko/wide-residual-networks.

Wide Residual Networks (2016)

ResNetv2

Q : What happens if we goes deeper than 200 layers?

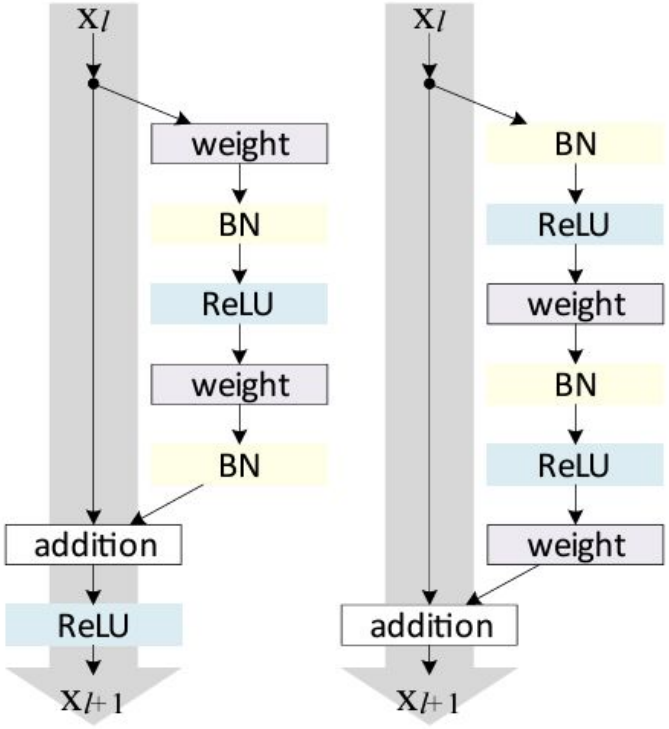
A : The performance gets better with some modifications though the return is diminished.

dataset	network	[redacted]	pre-activation unit
CIFAR-10	ResNet-110 (1layer skip)		8.91
	ResNet-110		6.37
	ResNet-164		5.46
	ResNet-1001		4.92
CIFAR-100	ResNet-164		24.33
	ResNet-1001		22.71

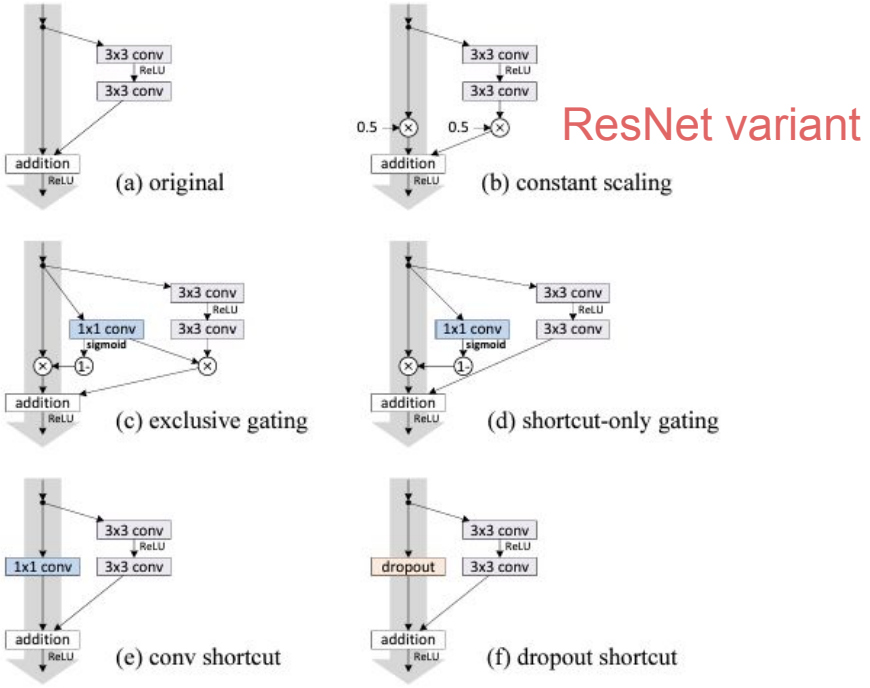
Kaiming He, et al., Identity Mappings in Deep Residual Networks (2016)

ResNetv2

- Conclude on how should skipped connections be added



(a) original (b) proposed



ResNet variant

Figure 2. Various types of shortcut connections used in Table 1. The grey arrows indicate the easiest paths for the information to propagate. The shortcut connections in (b-f) are impeded by different components. For simplifying illustrations we do not display the BN layers, which are adopted right after the weight layers for all units here.

Kaiming He, et al., Identity Mappings in Deep Residual Networks (2016)

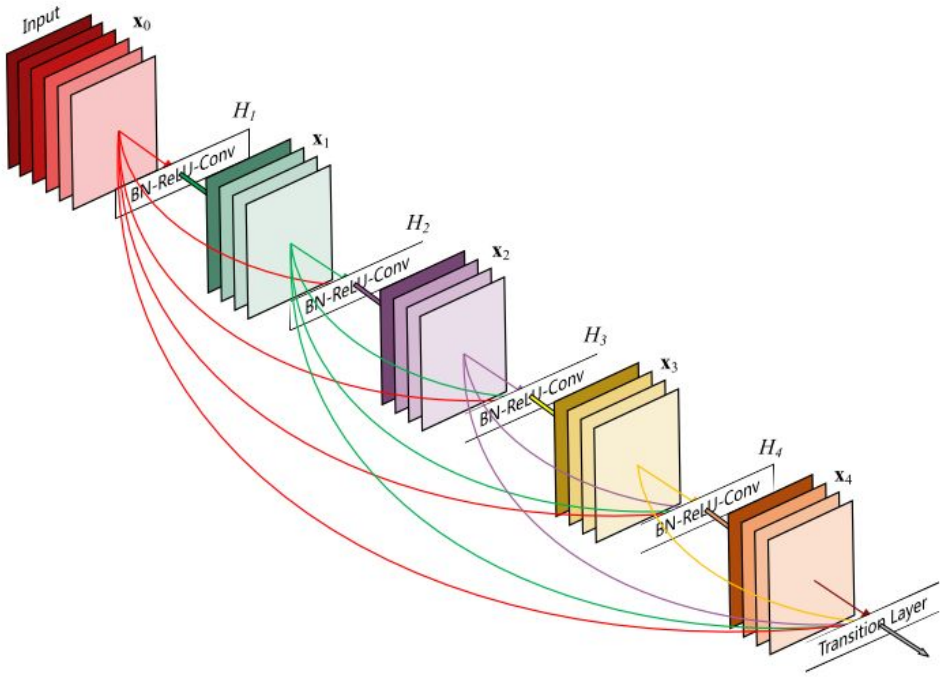
DenseNet

Instead of adding the residuals, concatenates the feature maps from previous layers

Densely connect multiple layers

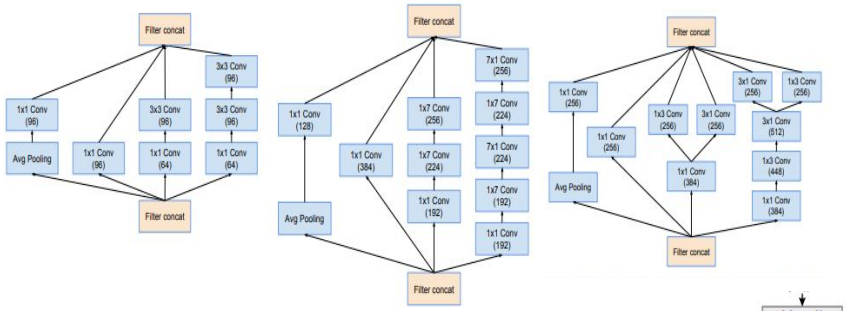
Multi-scale feature maps

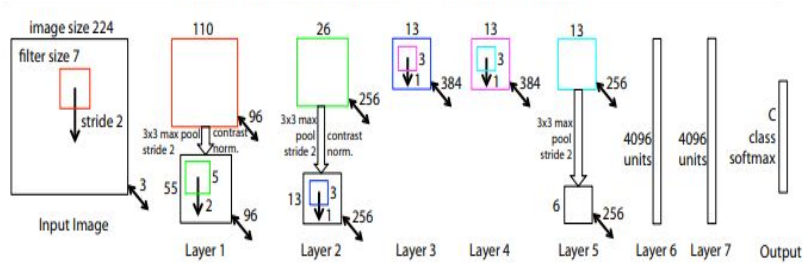
Smaller model due to smaller number of channels

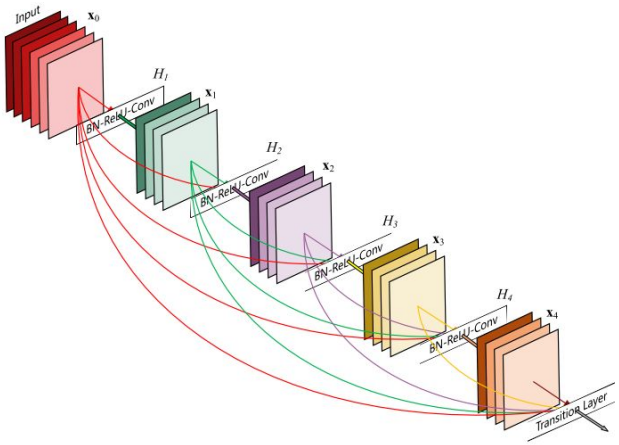


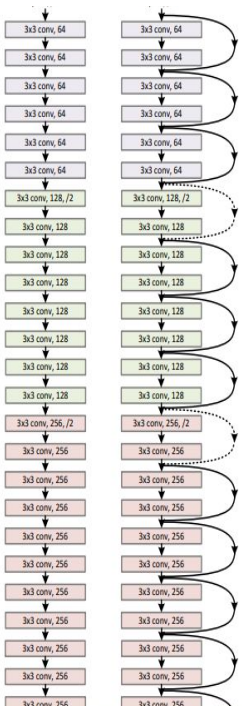
Densely Connected Convolutional Networks 2016

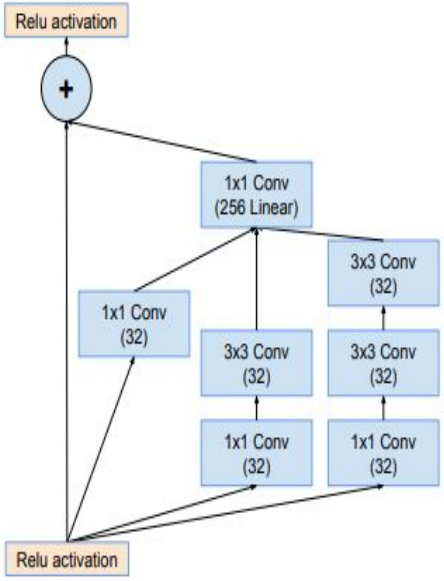
Do people keep tweaking network architectures until the end of time?









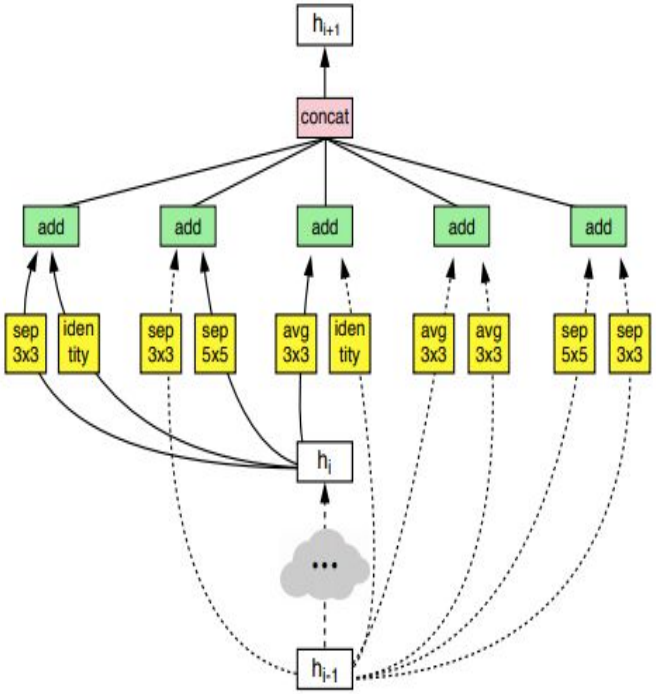


NasNet

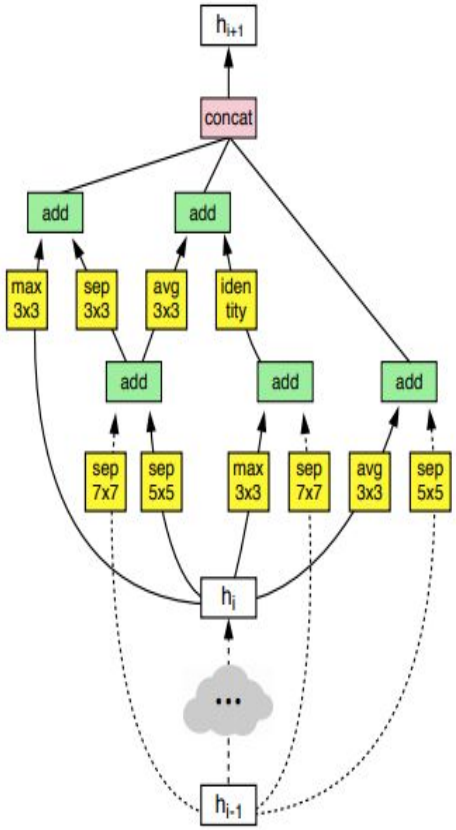
Neural Architecture Search Network

Use reinforcement learning to search for the best network configuration

Lowers compute (FLOPs) while maintaining high accuracy



Inception block



Reduction block

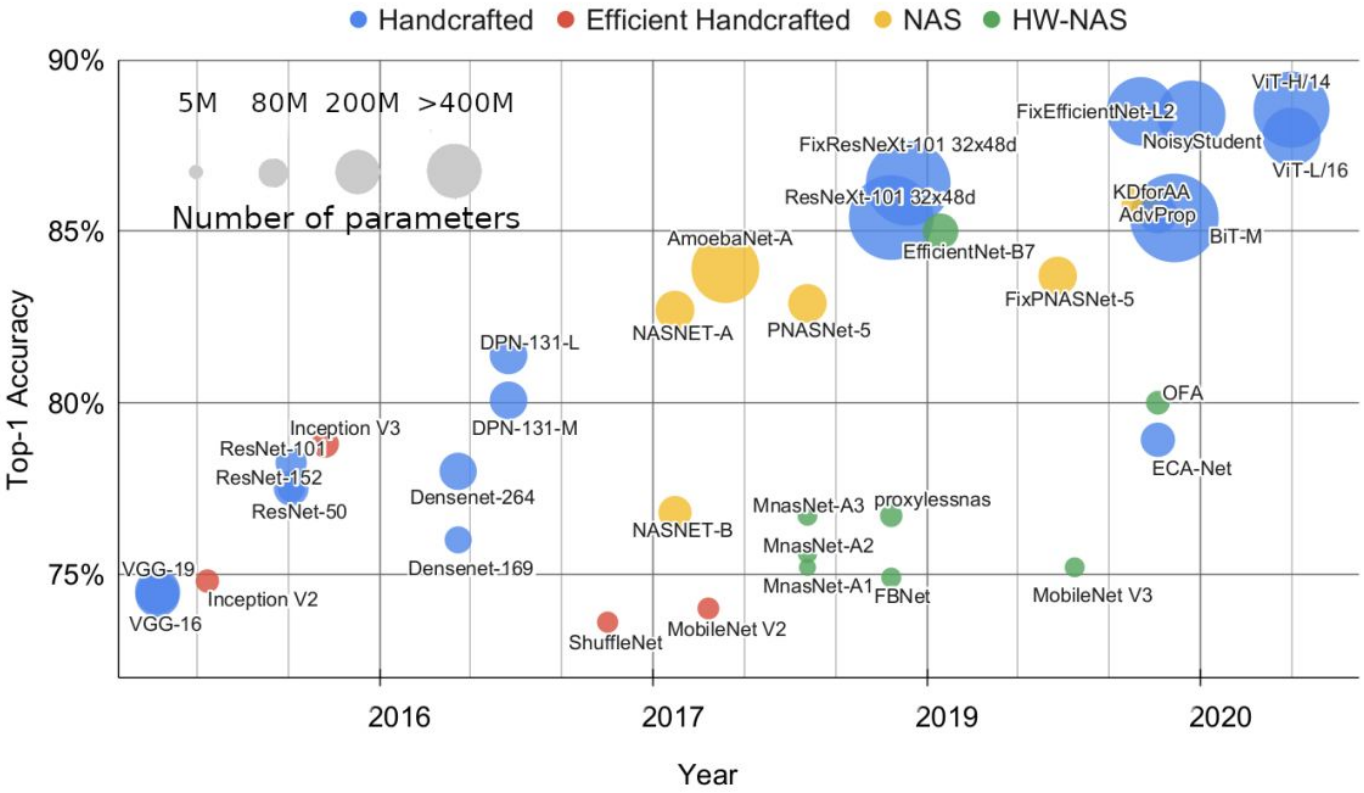
Barret Zoph, et al. “Learning Transferable Architectures for Scalable Image Recognition,” 2017.

NAS doesn't know com sys arch

- ★ Response Time
  - Elapsed Time - Wall clock
  - CPU Time - Only CPU time (without the wait time) ... we will revisit this later.
  - From O.S. class, we use: user time, real time, sys time.
- ★ Throughput
  - Tasks per time unit
- ★ Units for commercialization (Not a real performance)
  - MIPS
    - Millions of instructions per second
  - FLOPS
    - Floating Point Operations per second

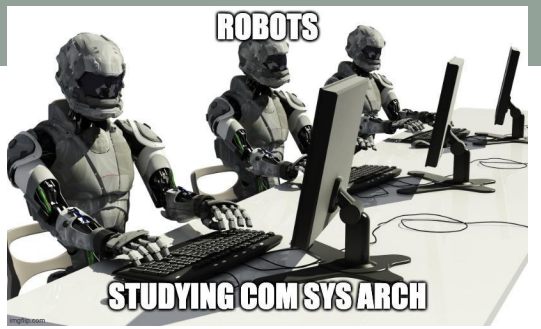
$$Elapse\ Time = CPU\ Time + I/O\ Time + Memory\ Time$$

- FLOPs is not a good measure of performance



https://arxiv.org/abs/2101.09336

EfficientNetv1, v2



- Architecture/hardware aware NAS + handcrafted building blocks (from mobilenetV2)
- Tradeoff between width, depth, resolution

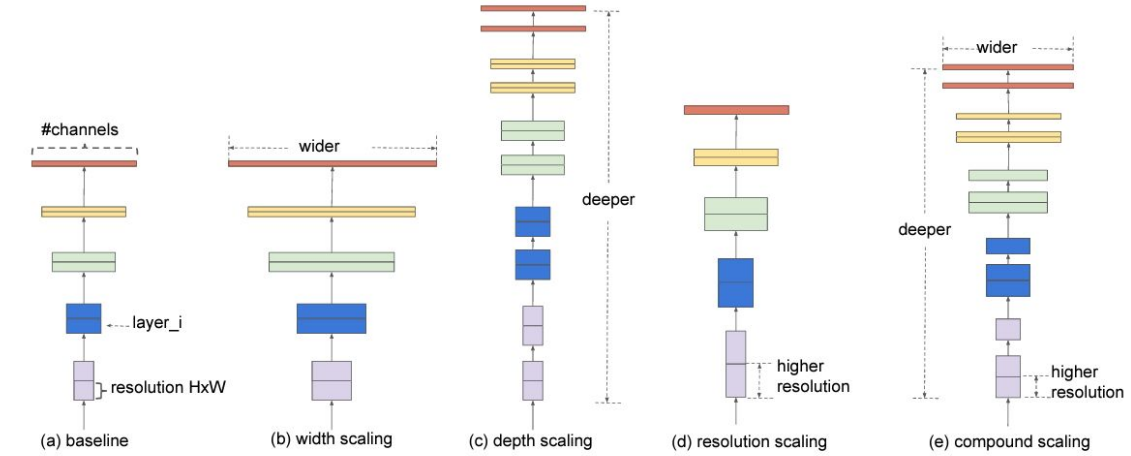


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

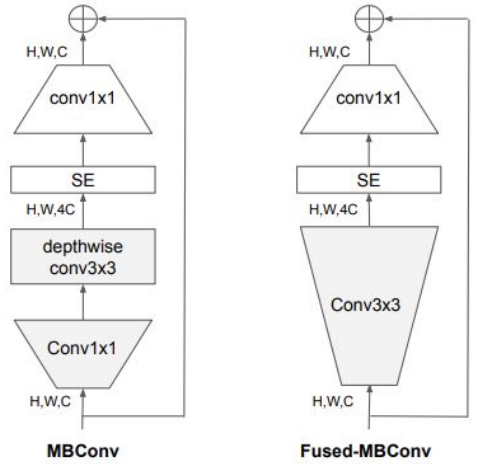
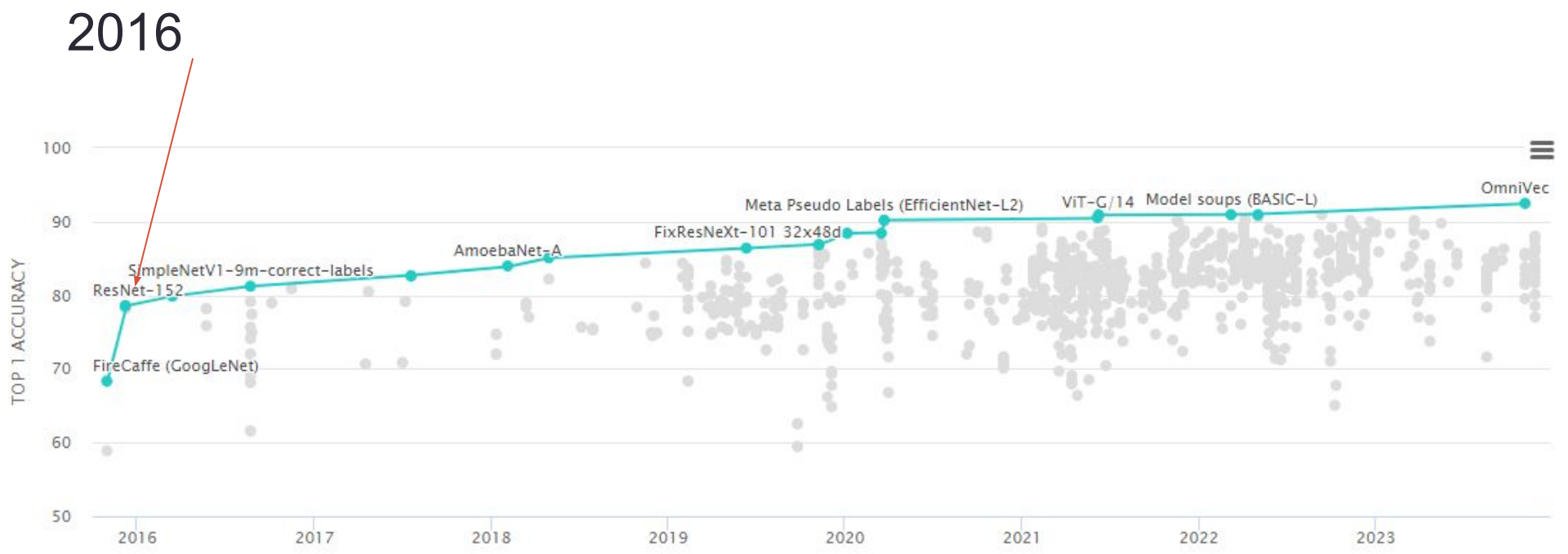


Figure 2. Structure of MBConv and Fused-MBConv.

https://arxiv.org/abs/1905.11946
https://arxiv.org/abs/2104.00298

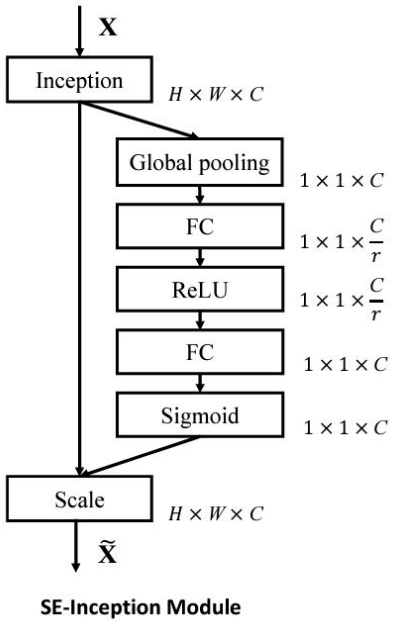
Trends after ResNet

- More efficient convolutional block design
- Wider networks, deeper models, denser connections
- **Global Context + Transformer architecture**
- Better representation learning (self-supervision + multimodality)

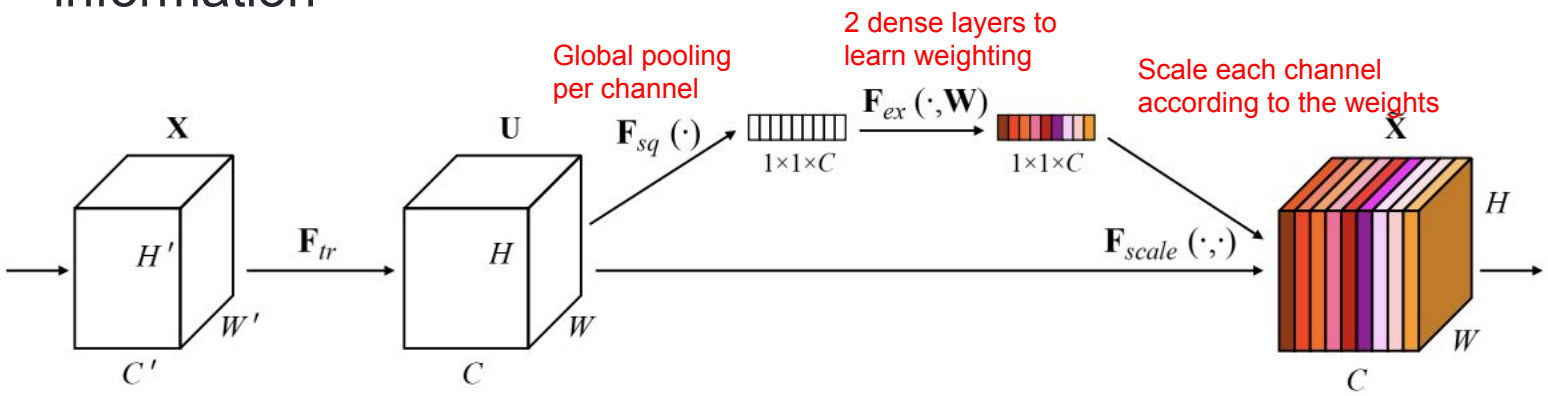


Squeeze and excitation

- Maybe each channel should be weighted differently?
  - A picture with wave patterns should look for fish features
  - A picture with grass patterns should look for cow features
- "Attention" on channel info
  - CBAM extends this work by weighting spatial information



SE-Inception Module



Squeeze-and-Excitation Networks (2017), CBAM: Convolutional Block Attention Module (2018)

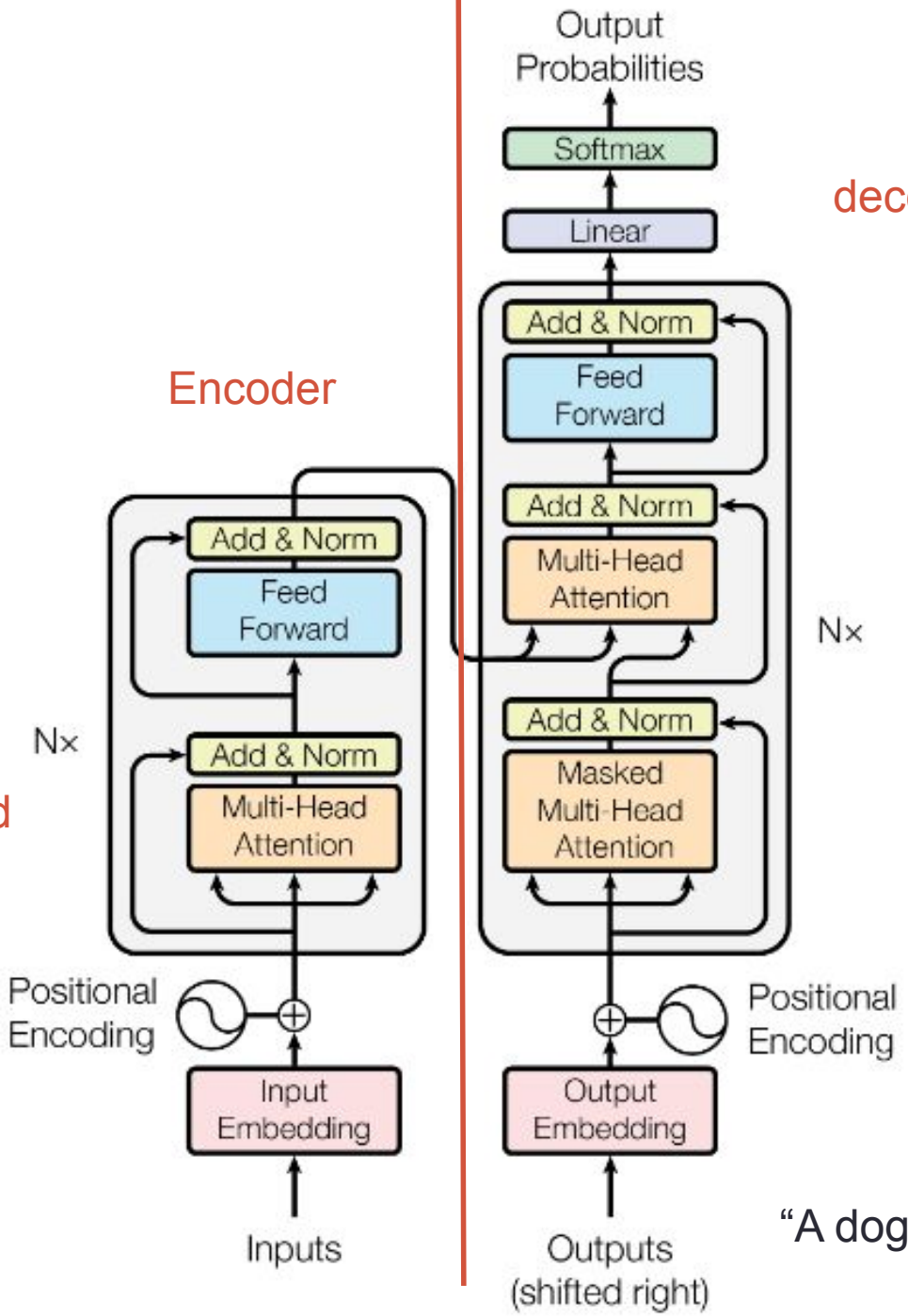
Transformer

Attention is all you need

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

https://arxiv.org/pdf/1706.03762.pdf



Encoder

decoder

To eliminate GRU which remembers time, encode position instead

"A dog sits next to another dog"

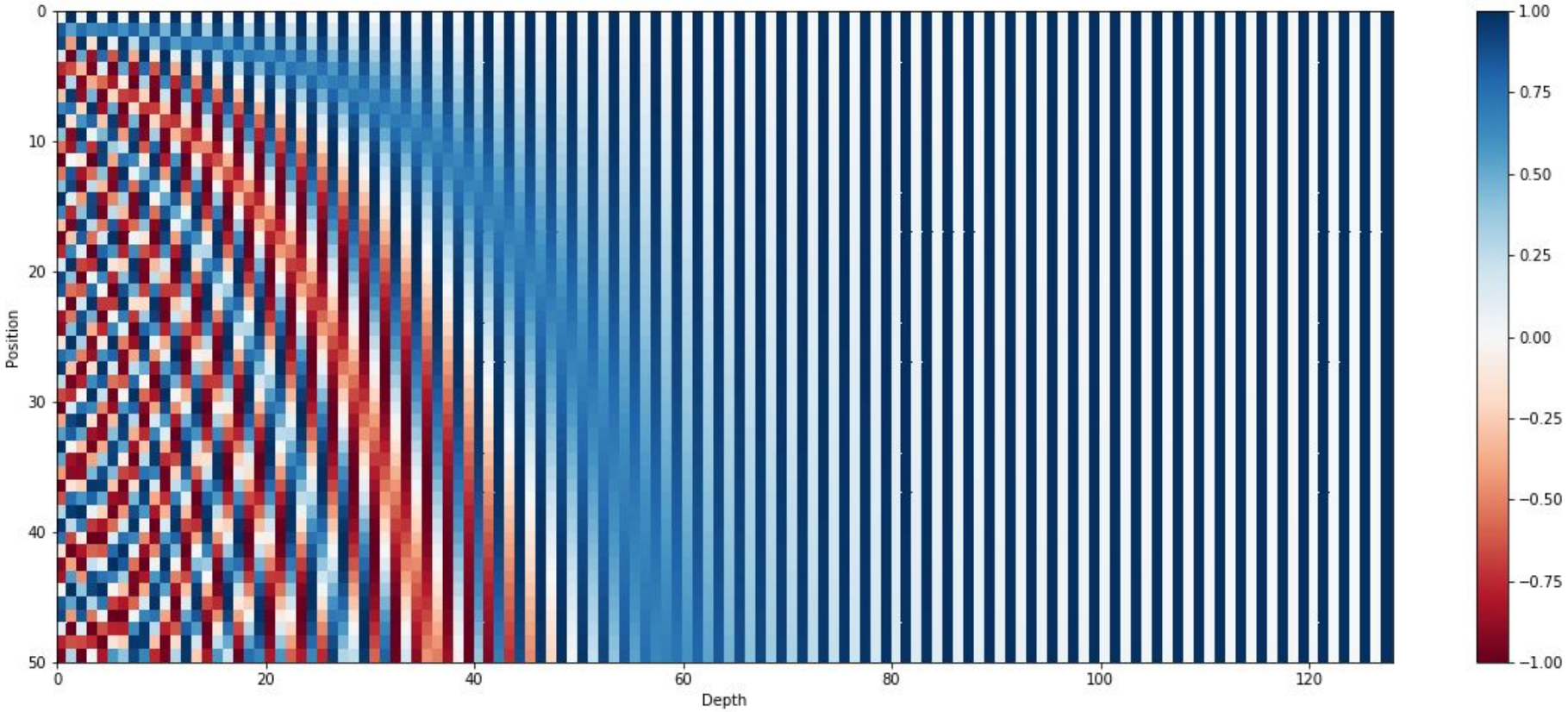
Positional encoding

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

Matrix M can be used to change phase

$$M.\begin{bmatrix}\sin(\omega_k.t)\\ \cos(\omega_k.t)\end{bmatrix} = \begin{bmatrix}\sin(\omega_k.(t+\phi))\\ \cos(\omega_k.(t+\phi))\end{bmatrix}$$

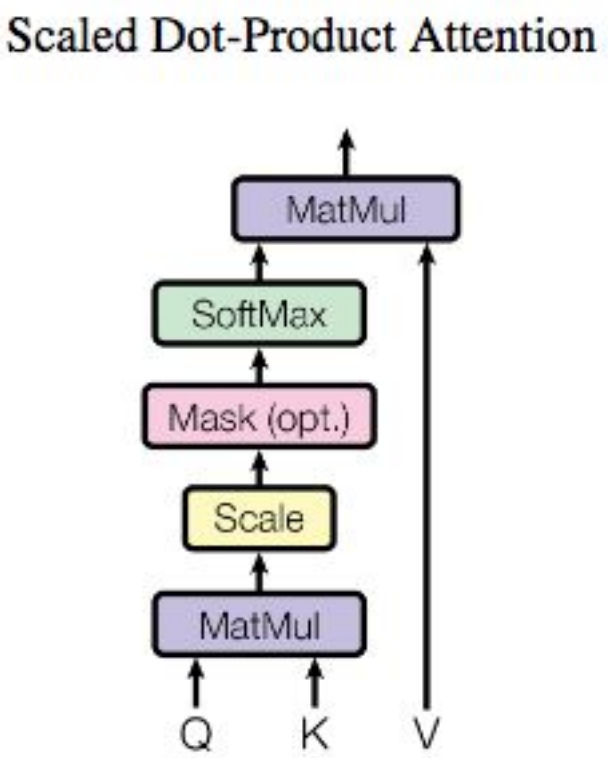


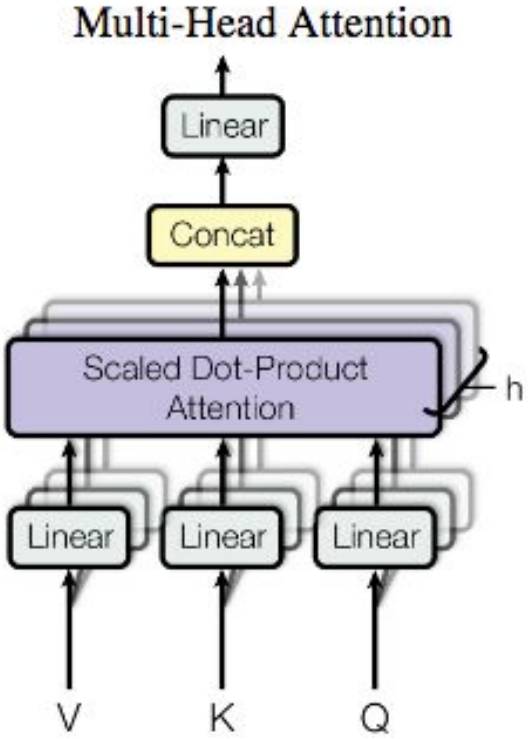
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Multi-head attention

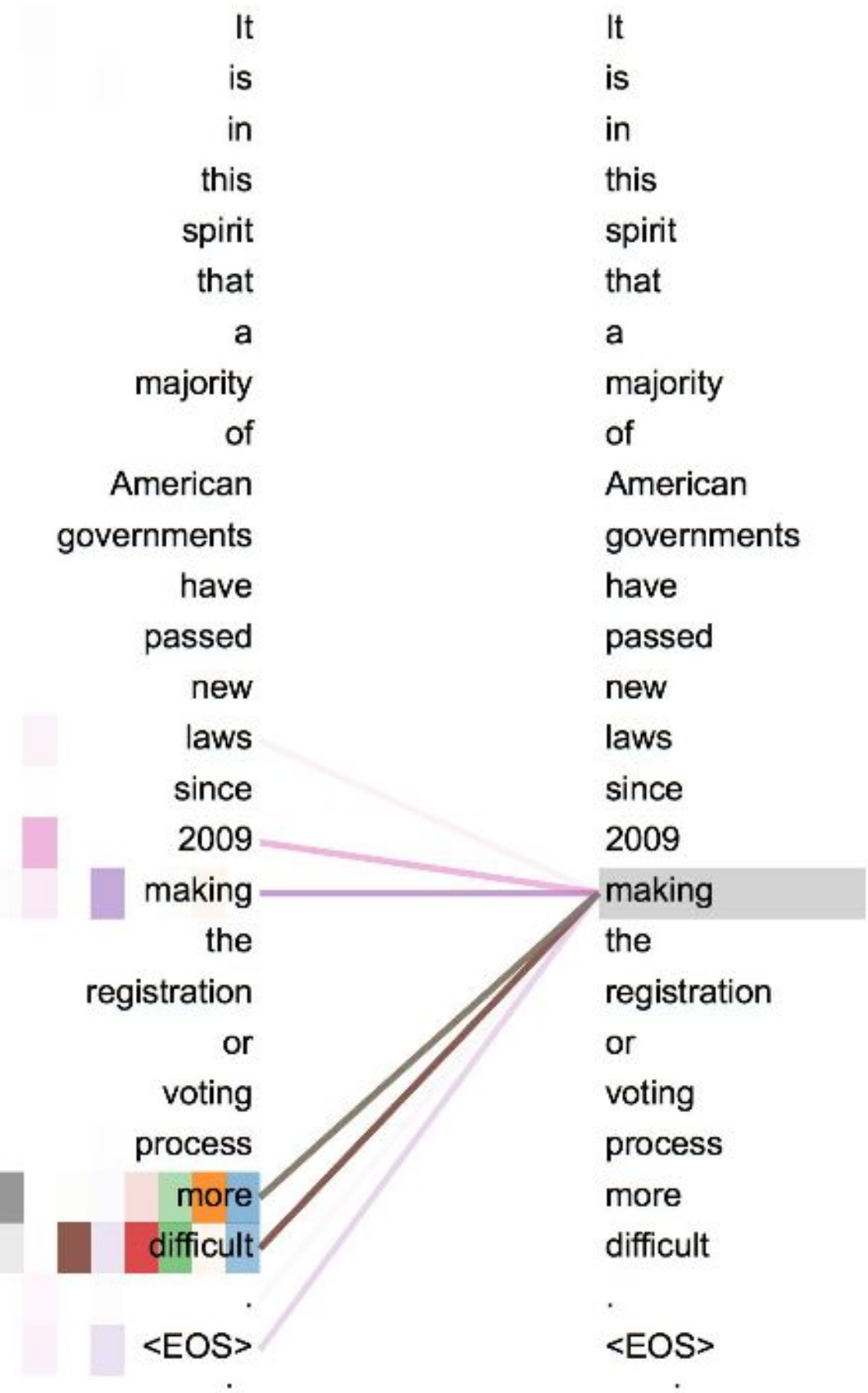
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

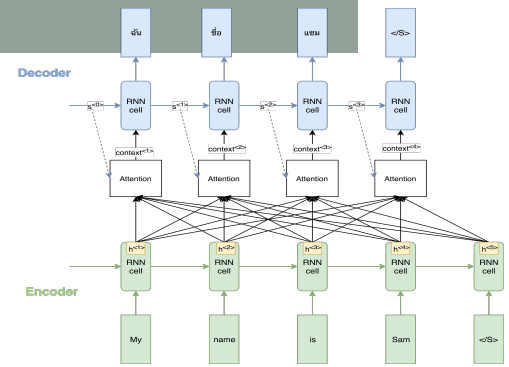
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$





Multi-head visualization

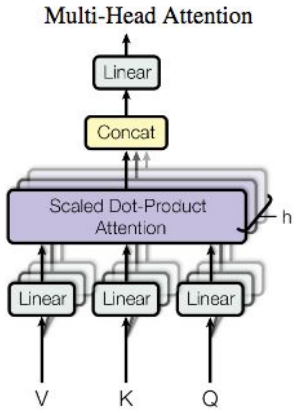


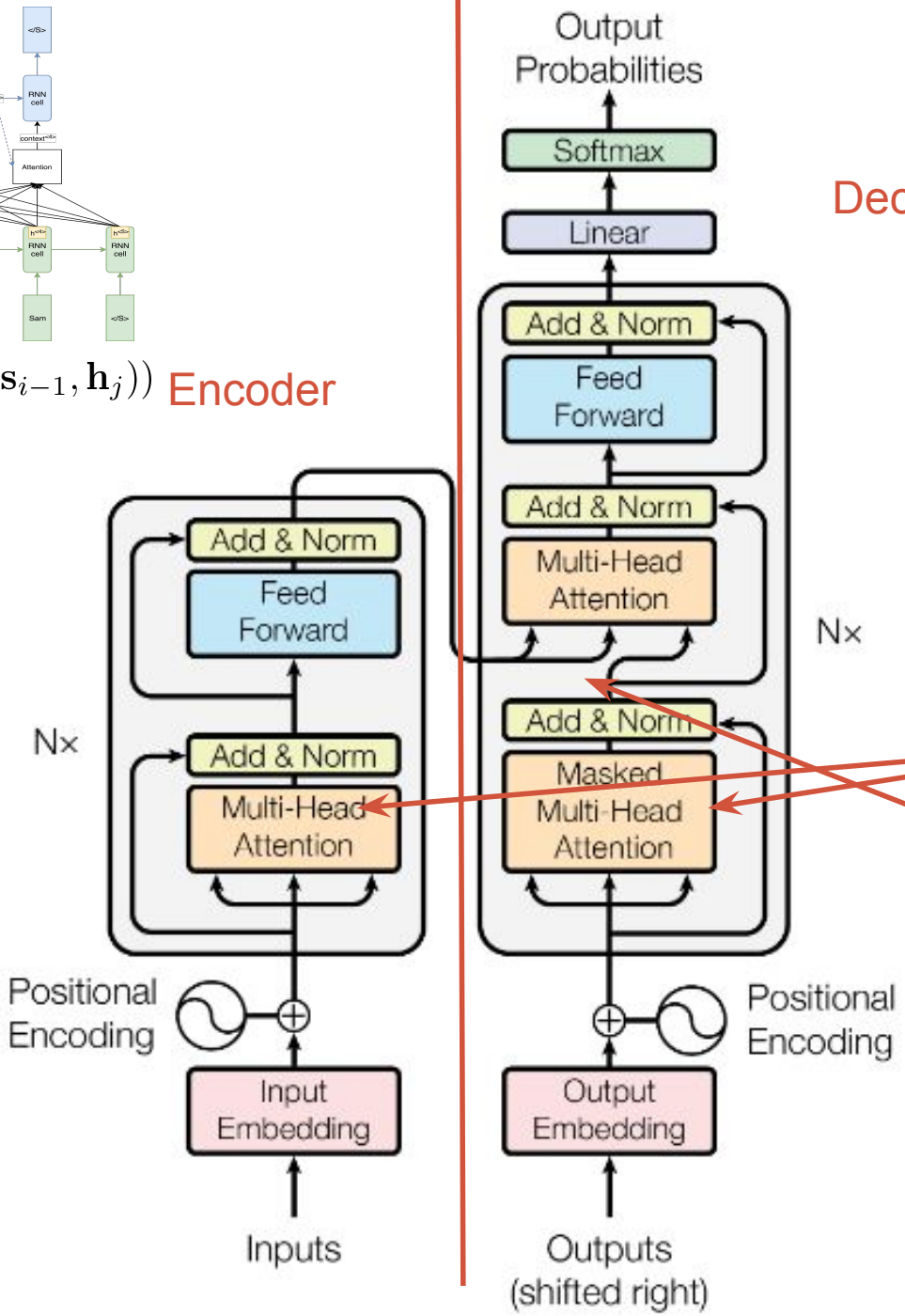


$$a_{ij} = softmax(f_{att}(\mathbf{s}_{i-1}, \mathbf{h}_j))$$

Encoder

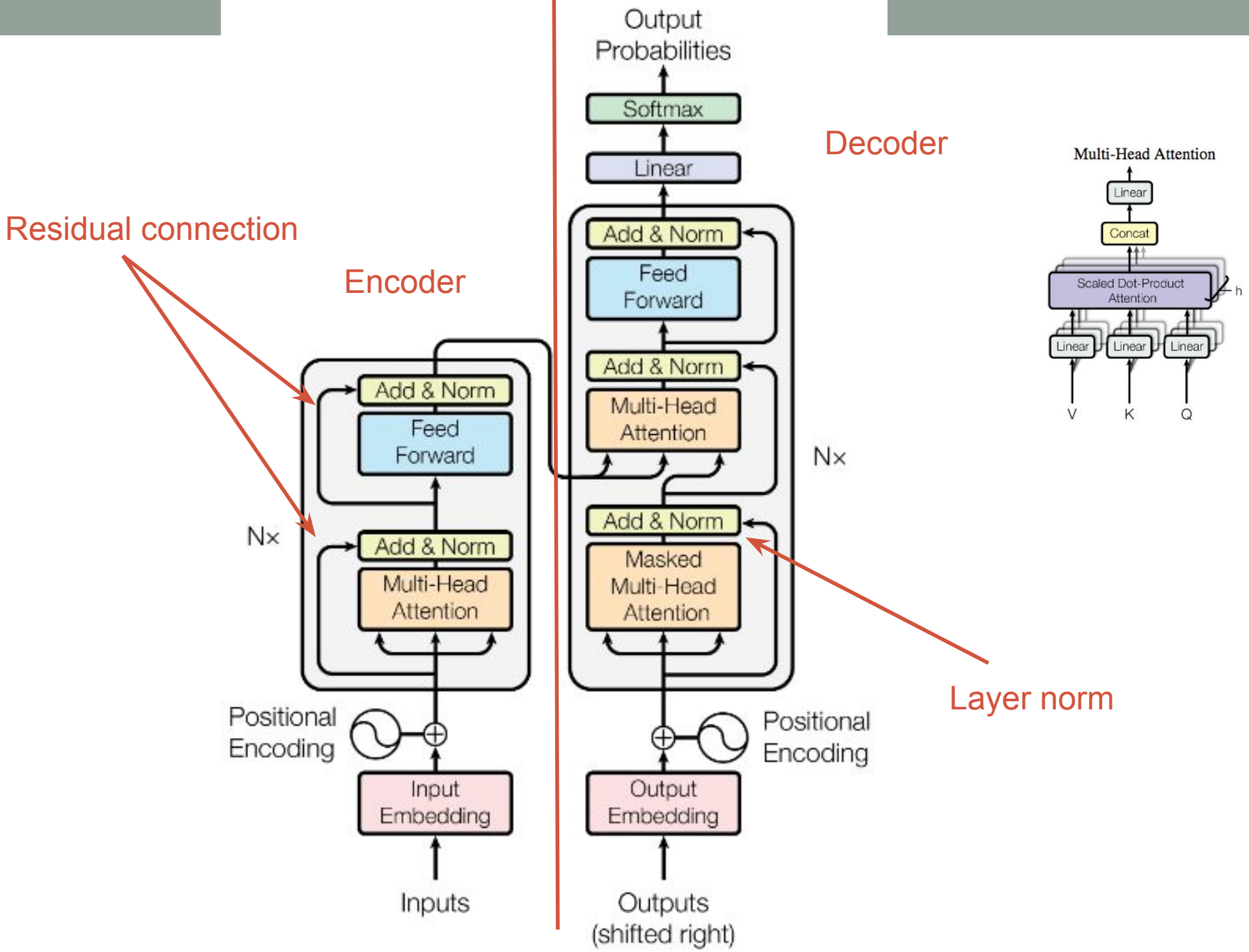
Decoder





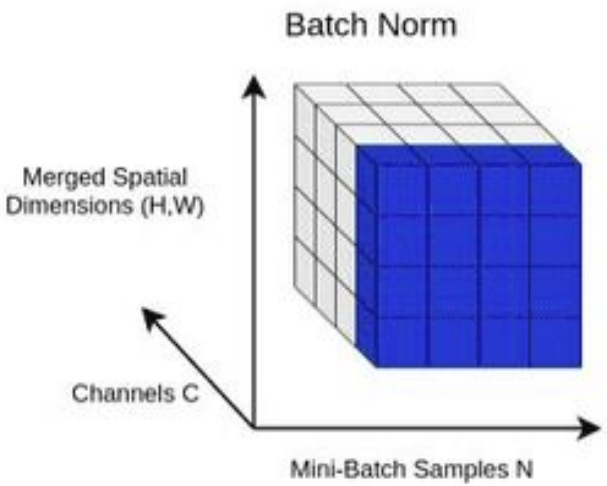
Self attention

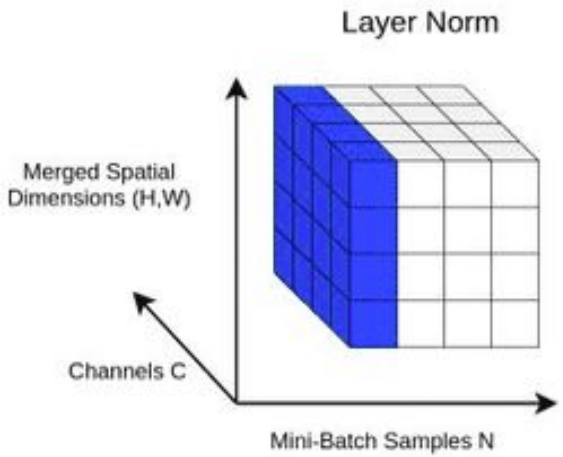
Encoder gives Value and Key
The decoder gives Query

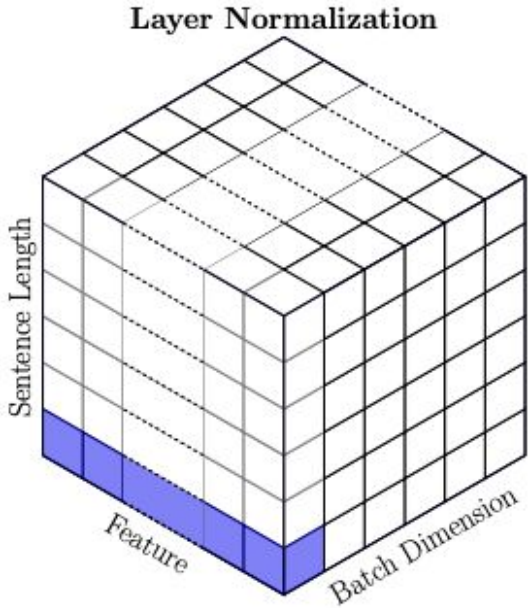

Residual connection
Encoder
Decoder
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Layer norm
Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

Layer Normalization

- Layer Normalization makes more sense than Batch Normalization in NLP
  - Sentence length for each input could be varied
  - Synchronization problem
  - Attention forward at each time-step independently (not sequential)

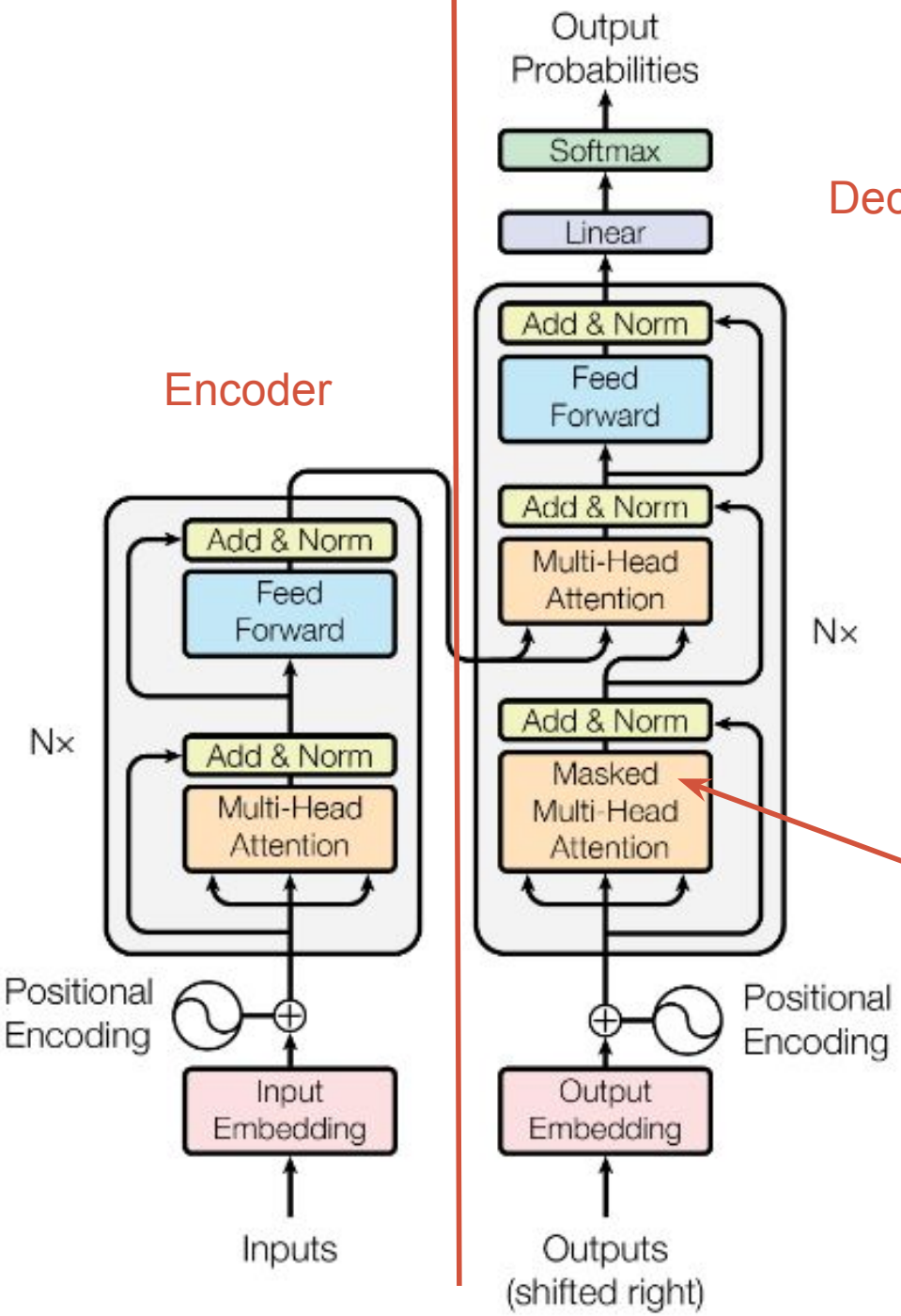






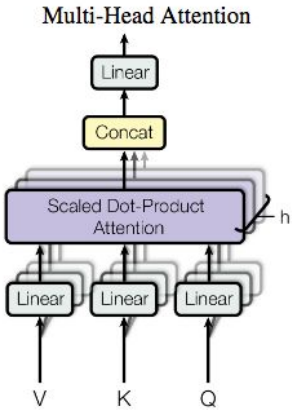
Layer norm in transformers

https://theaisummer.com/normalization/



Encoder

Decoder



Prevent looking ahead (cheating)

Without masking, attention can access global information.

a_ij=0 on future position.

MT results

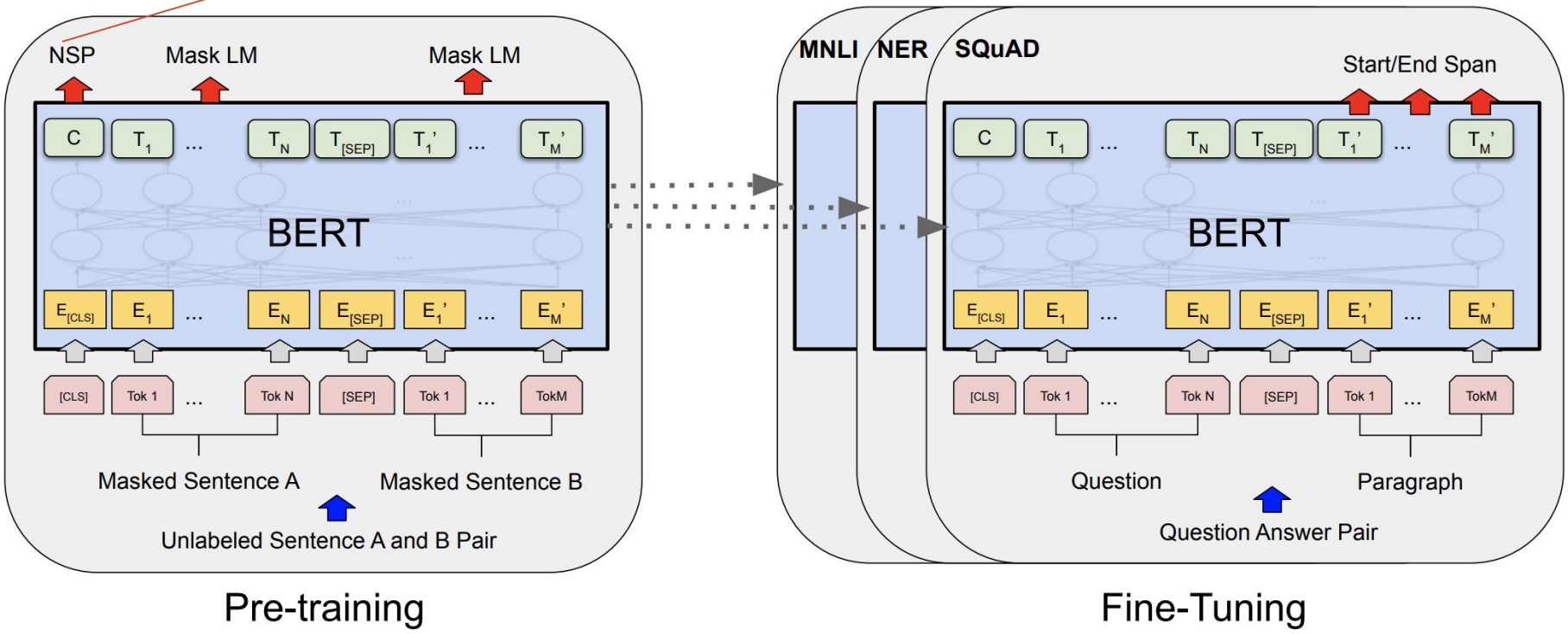
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Can use for other tasks, like ASR, parsing, etc.

BERT

Full transformer trained on a lot of data
Masked LM to learn bi-directional LM
Next sentence prediction to learn discourse

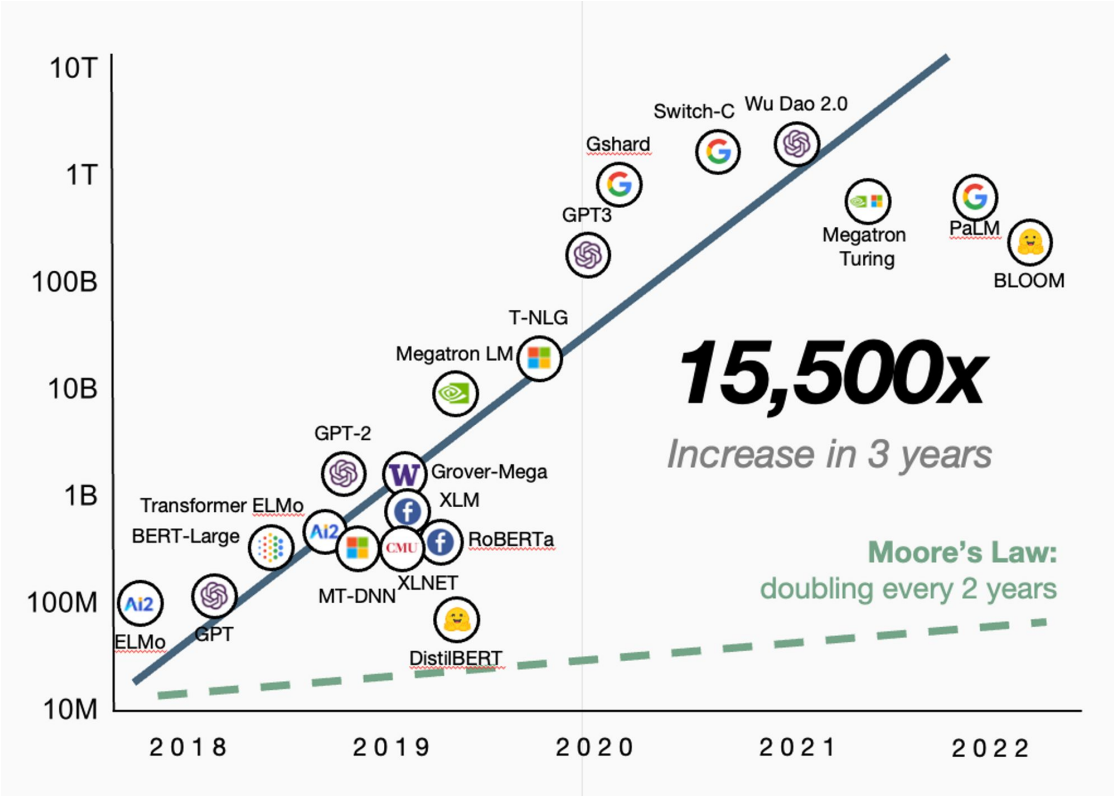


https://arxiv.org/abs/1810.04805

Recent trends in transformers

- Larger models

  Possible by more efficient architectures, more data, more compute, smarter distributed training



https://www.dingran.me/from-transformer-to-llm/

Tricks to make better transformers

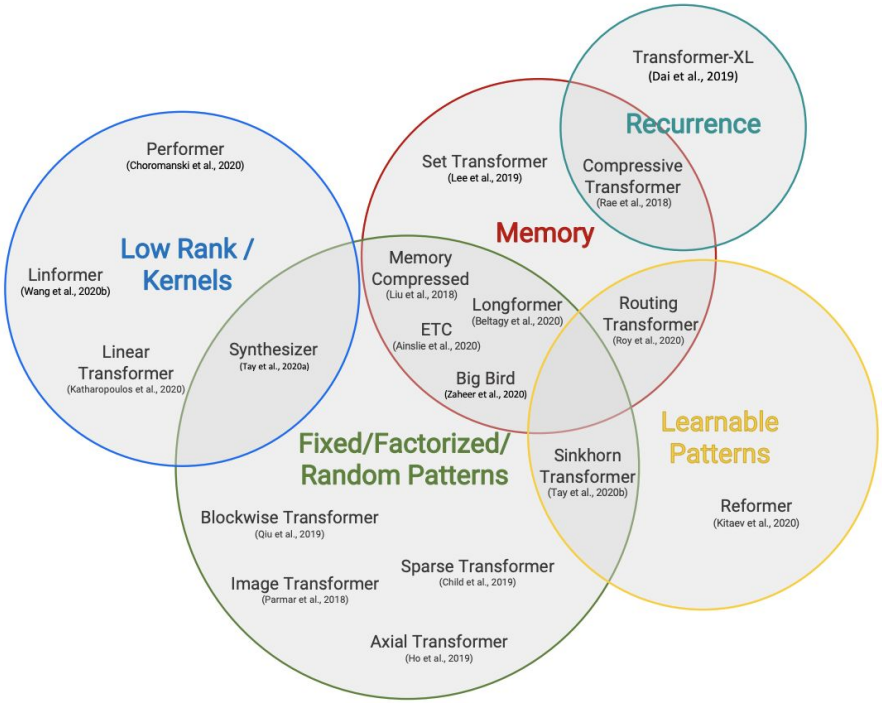


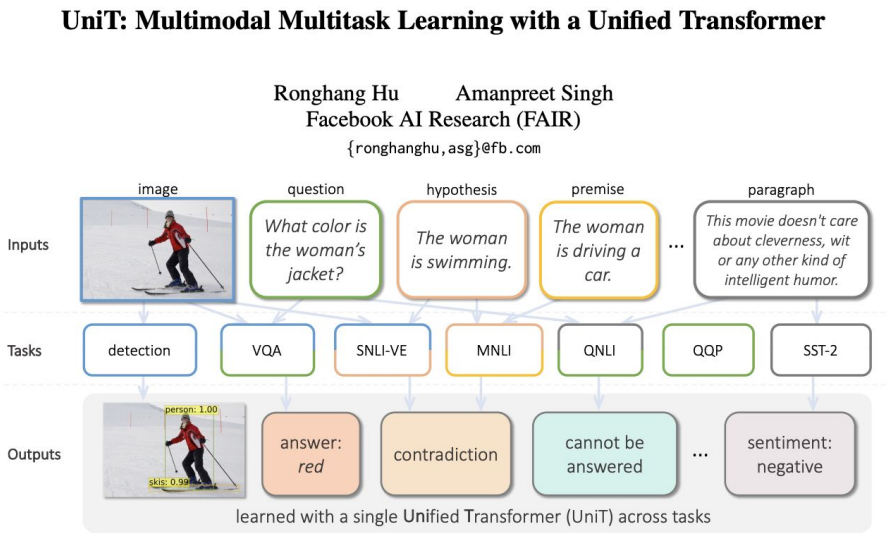
Figure 2: Taxonomy of Efficient Transformer Architectures.

Efficient Transformers: A Survey, 2020 https://arxiv.org/pdf/2009.06732.pdf

Many types of data could be seen as sequential.

Trends

- Transformer is expensive
- Transformer replaces CNN-based architecture
- Transformer is all you need



UniT: Multimodal Multitask Learning with a Unified Transformer, 2021

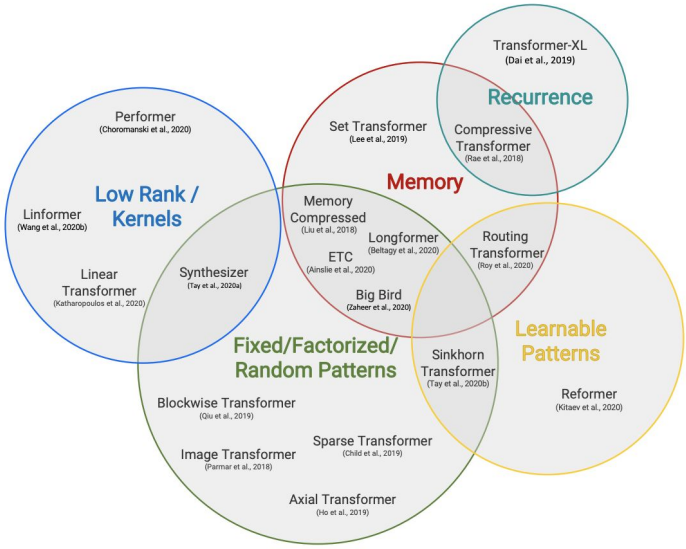


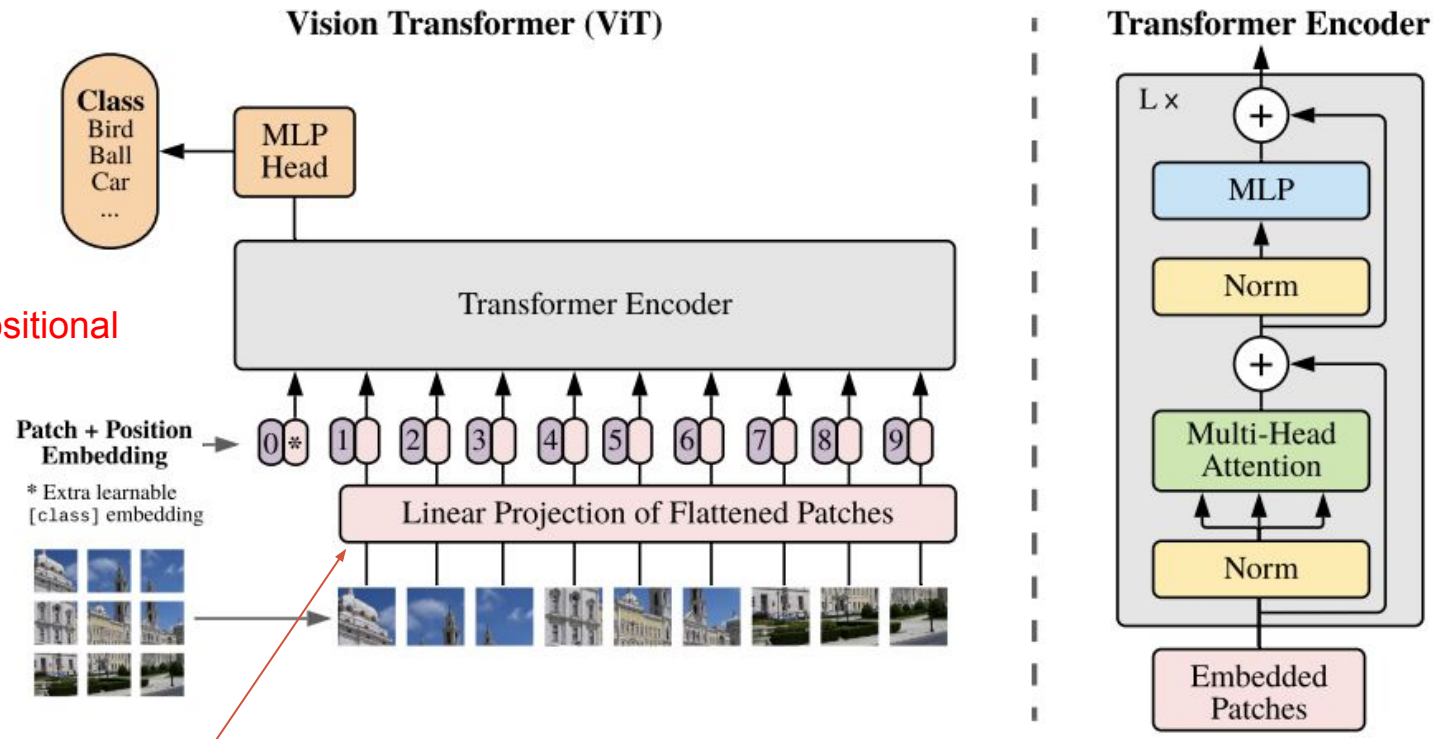
Figure 2: Taxonomy of Efficient Transformer Architectures.

Efficient Transformers: A Survey, 2020

Back to vision world

Vision transformer

The paper explains this model using less than one page



Learnable positional embedding

4x4 non-overlapping CNN filter

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE
https://arxiv.org/pdf/2010.11929v1.pdf

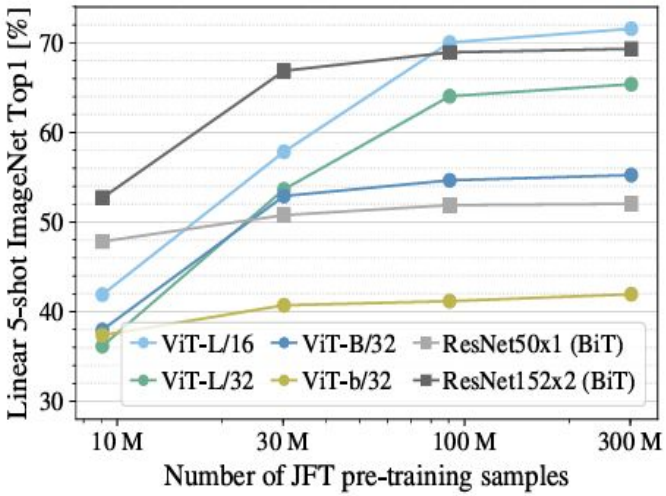
Pros and Cons of ViT

Cons (Less severe after 2021):

- Horrible performance & efficiency w/o good pre-training
  - Also very unstable in some tasks w/o good pre-training
- Low flexibility
- Resource hungry
  - High parameter count / Slow training time
  - Not a fully sequential model

Pros :

- Well benefit from good pre-training
- Good for multi-modality training
- Simplistic design



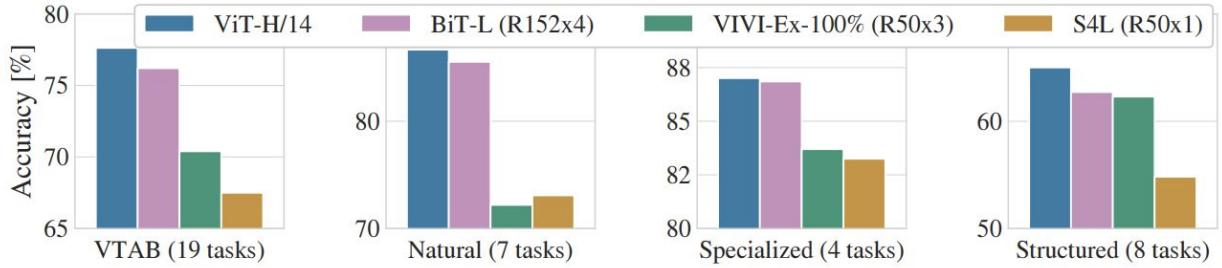


Figure 2: Breakdown of VTAB performance in *Natural*, *Specialized*, and *Structured* task groups.

Only wins if data is VERY large

Does not work well for object detection and semantic segmentation as backbones

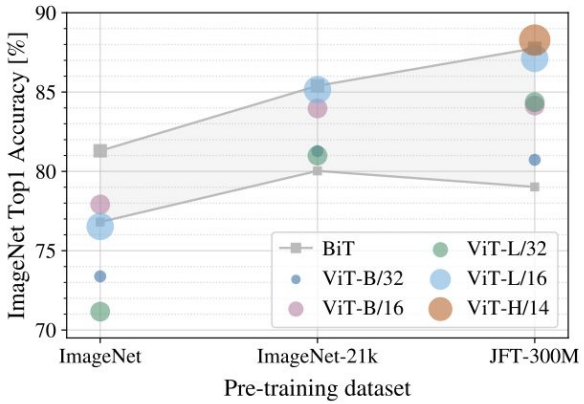


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

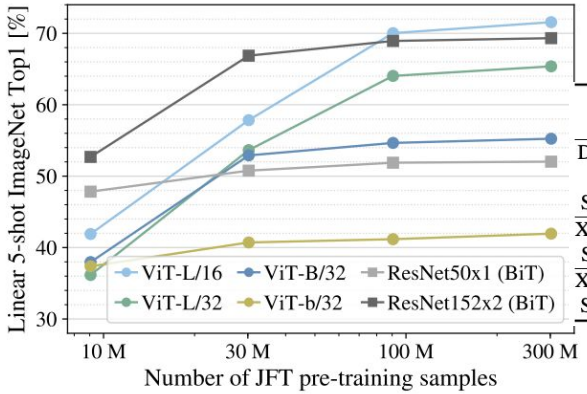


Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT, which performs better with larger pre-training. ViT-b is ViT-B with all hidden dimensions halved.

(b) Various backbones w. Cascade Mask R-CNN

	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	param	FLOPs	FPS
DeiT-S†	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	**50.5**	**69.3**	**54.9**	**43.7**	**66.6**	**47.1**	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	**51.8**	**70.4**	**56.3**	**44.7**	**67.9**	**48.5**	107M	838G	12.0
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	**51.9**	**70.9**	**56.5**	**45.0**	**68.4**	**48.7**	145M	982G	11.6

https://arxiv.org/pdf/2103.14030.pdf

Swin Transformer

- So…we're dealing with images
  - Different scale
  - Translation equivalence (object detection)
    - Translation invariance – move input, same output
    - Translation equivariance – move input, move output same amount

Hierarchical Vision Transformer using Shifted Windows (2021)

Swin Transformer

- Multi-scale
- Local attention (conv-like) in a local window
- Weights across local windows are shared
  - Translation equivariance

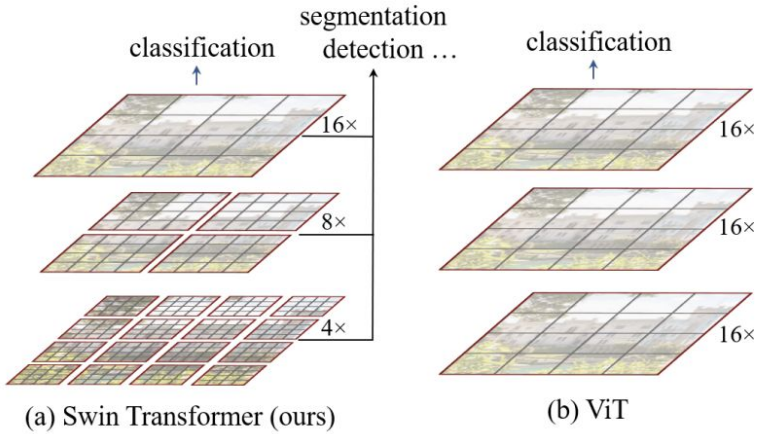
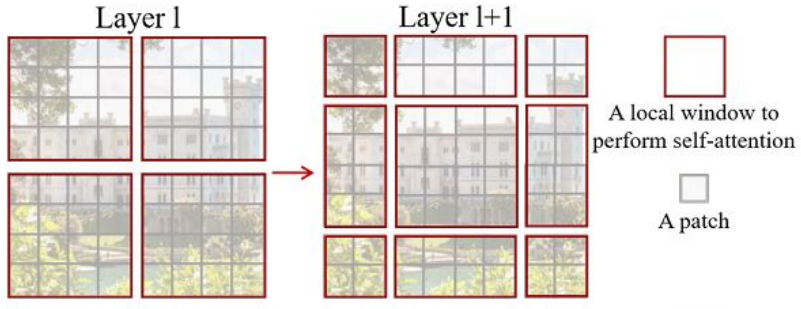


Figure 1. (a) The proposed Swin Transformer builds hierarchical feature maps by merging image patches (shown in gray) in deeper layers and has linear computation complexity to input image size due to computation of self-attention only within each local window (shown in red). It can thus serve as a general-purpose backbone for both image classification and dense recognition tasks. (b) In contrast, previous vision Transformers [20] produce feature maps of a single low resolution and have quadratic computation complexity to input image size due to computation of self-attention globally.

(b) Various backbones w. Cascade Mask R-CNN

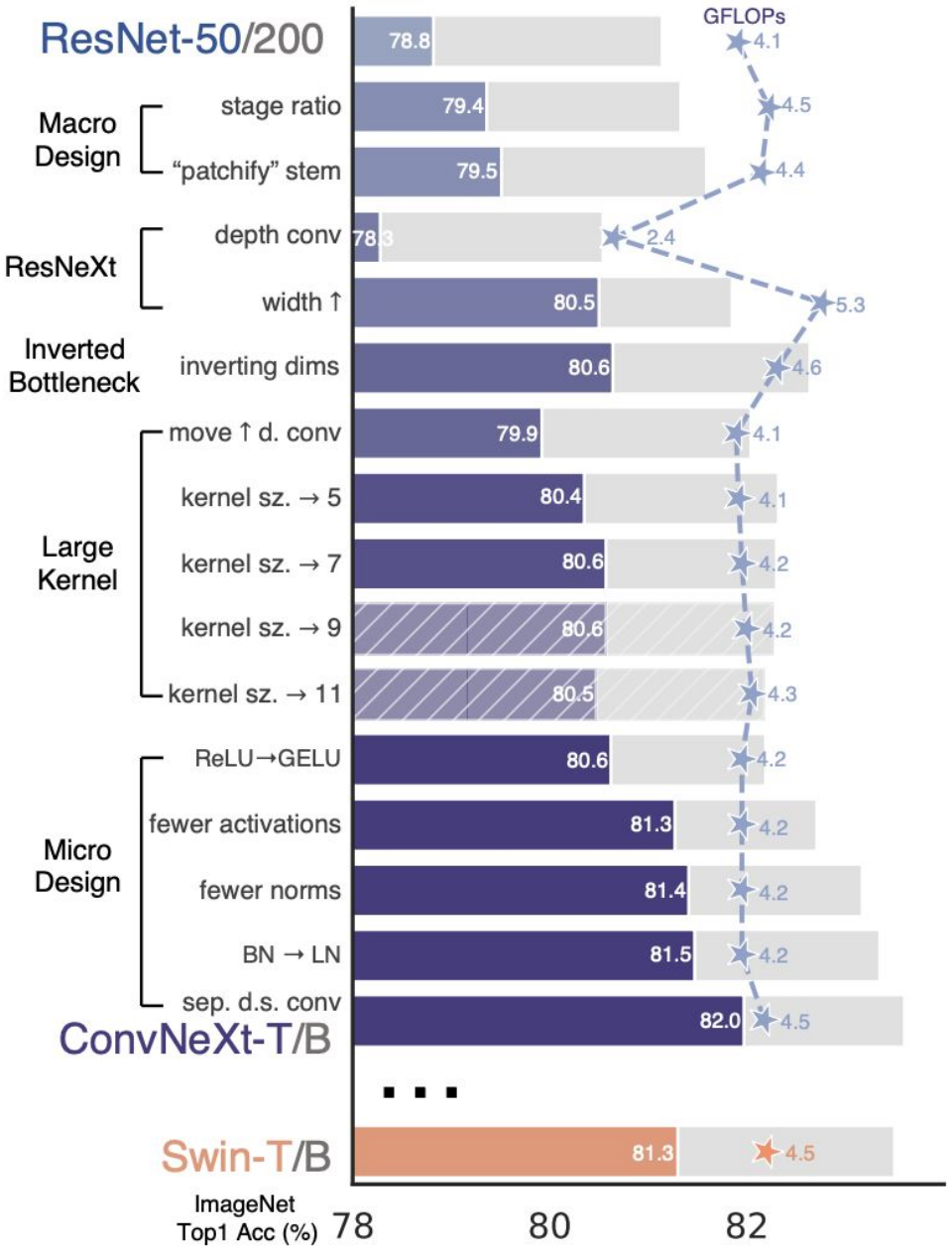
	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	param	FLOPs	FPS
DeiT-S†	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	**50.5**	**69.3**	**54.9**	**43.7**	**66.6**	**47.1**	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	**51.8**	**70.4**	**56.3**	**44.7**	**67.9**	**48.5**	107M	838G	12.0
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	**51.9**	**70.9**	**56.5**	**45.0**	**68.4**	**48.7**	145M	982G	11.6



Shift window in the next layer

ConvNext

- To get the most recent hype see ConvNext
- Bag of tricks++
- Highly recommended as a review paper

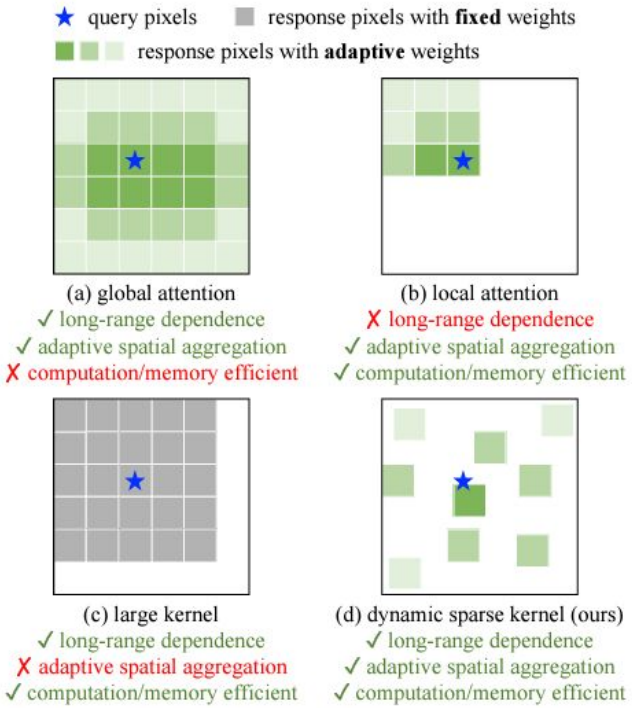


A ConvNet for the 2020s (2022)

After 2022

- CNN embraces more ViT design
  - Still lose when pretrained though
  - The return of deformable convolution
  - Super large depthwise convolution kernel (31x31)
- Finding ways to create large CNN

The above view naturally lead to a question: what if we use *a few large* instead of *many small* kernels to conventional CNNs? Is large kernel or the way of building large recep-



Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs (2022)
InternImage: Exploring Large-Scale Vision Foundation Models with Deformable Convolutions (2023)

CNN vs ViT

- Without pretraining, CNN outperforms ViT models
- With good pertaining, ViT models win but the gap is getting smaller

huge CNN still use only 1B parameters

method	type	scale	#params	#FLOPs	acc (%)
DeiT-S [58]	T	224^2	22M	5G	79.9
PVT-S [10]	T	224^2	25M	4G	79.8
Swin-T [2]	T	224^2	29M	5G	81.3
CoAtNet-0 [20]	T	224^2	25M	4G	81.6
CSwin-T [12]	T	224^2	23M	4G	82.7
PVTv2-B2 [11]	T	224^2	25M	4G	82.0
DeiT III-S [64]	T	224^2	22M	5G	81.4
SwinV2-T/8 [16]	T	256^2	28M	6G	81.8
Focal-T [65]	T	224^2	29M	5G	82.2
ConvNeXt-T [21]	C	224^2	29M	5G	82.1
ConvNeXt-T-dcls [66]	C	224^2	29M	5G	82.5
SLaK-T [29]	C	224^2	30M	5G	82.5
HorNet-T [43]	C	224^2	23M	4G	83.0
InternImage-T (ours)	C	224^2	30M	5G	83.5
PVT-L [10]	T	224^2	61M	10G	81.7
Swin-S [2]	T	224^2	50M	9G	83.0
CoAtNet-1 [20]	T	224^2	42M	8G	83.3
PVTv2-B4 [11]	T	224^2	63M	10G	83.6
SwinV2-S/8 [16]	T	256^2	50M	12G	83.7
ConvNeXt-S [21]	C	224^2	50M	9G	83.1
ConvNeXt-S-dcls [66]	C	224^2	50M	10G	83.7
SLaK-S [29]	C	224^2	55M	10G	83.8
HorNet-S [43]	C	224^2	50M	9G	84.0
InternImage-S (ours)	C	224^2	50M	8G	84.2
DeiT-B [58]	T	224^2	87M	18G	83.1
Swin-B [2]	T	224^2	88M	15G	83.5
CoAtNet-2 [20]	T	224^2	75M	16G	84.1
PVTv2-B5 [11]	T	224^2	82M	12G	83.8
DeiT III-B [64]	T	224^2	87M	18G	83.8
SwinV2-B/8 [16]	T	256^2	88M	20G	84.2
RepLKNet-31B [22]	C	224^2	79M	15G	83.5
ConvNeXt-B [21]	C	224^2	88M	15G	83.8
ConvNeXt-B-dcls [66]	C	224^2	89M	17G	84.1
SLaK-B [29]	C	224^2	95M	17G	84.0
HorNet-B [43]	C	224^2	88M	16G	84.3
InternImage-B (ours)	C	224^2	97M	16G	84.9
Swin-L‡ [2]	T	384^2	197M	104G	87.3
CoAtNet-3‡ [20]	T	384^2	168M	107G	87.6
CoAtNet-4‡ [20]	T	384^2	275M	190G	87.9
DeiT III-L‡ [64]	T	384^2	304M	191G	87.7
SwinV2-L/24‡ [16]	T	384^2	197M	115G	87.6
RepLKNet-31L‡ [22]	C	384^2	172M	96G	86.6
HorNet-L‡ [43]	C	384^2	202M	102G	87.7
ConvNeXt-L‡ [21]	C	384^2	198M	101G	87.5
ConvNeXt-XL‡ [21]	C	384^2	350M	179G	87.8
InternImage-L‡ (ours)	C	384^2	223M	108G	87.7
InternImage-XL‡ (ours)	C	384^2	335M	163G	88.0
ViT-G/14# [30]	T	518^2	1.84B	5160G	90.5
CoAtNet-6# [20]	T	512^2	1.47B	1521G	90.5
CoAtNet-7# [20]	T	512^2	2.44B	2586G	90.9
Florence-CoSwin-H# [59]	T	–	893M	–	90.0
SwinV2-G# [16]	T	640^2	3.00B	–	90.2
RepLKNet-XL# [22]	C	384^2	335M	129G	87.8
BiT-L-ResNet152x4# [67]	C	480^2	928M	–	87.5
InternImage-H# (ours)	C	224^2	1.08B	188G	88.9
InternImage-H# (ours)	C	640^2	1.08B	1478G	89.6

Table 2. **Image classification performance on the ImageNet validation set**. "type" refers to model type, where "T" and "C" denote transformer and CNN, respectively. "scale" is the input scale. "acc" is the top-1 accuracy. "‡" indicates the model is pre-trained on ImageNet-22K [31]. "#" indicates pretraining on extra large-scale private dataset such as JFT-300M [68], FLD-900M [59], or the joint public dataset in this work.

MLP-Mixer

- MLP returns!
- Attention is overrated, embrace MLP!

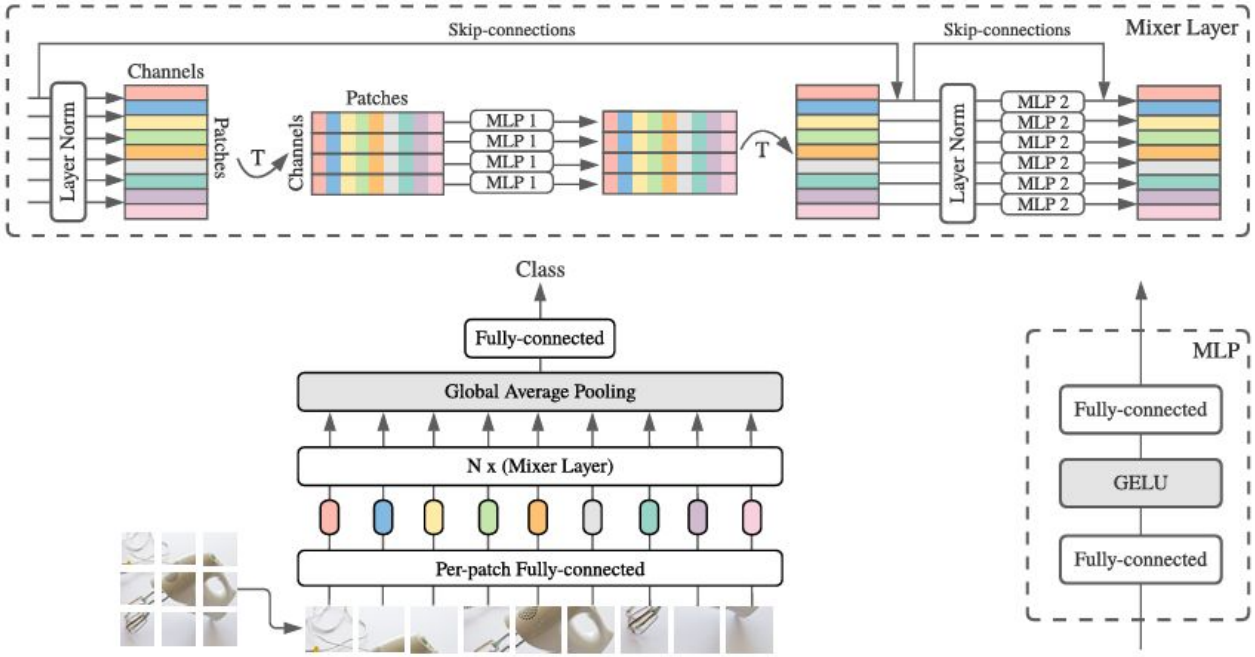
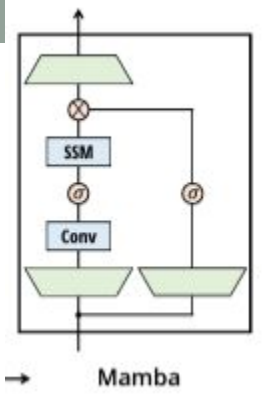


Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, and layer norm on the channels.

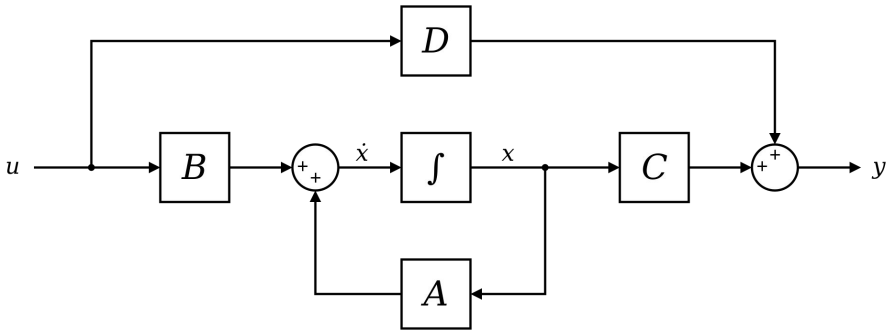
MLP-Mixer: An all-MLP Architecture for Vision (2021)

S4 + Mamba



- RNN+CNN strike backs!
- Structured State Space (S4) sequence model
- By constraining matrix A, the linear system could now remember the past!
- A lot of optimization to make it very fast

Linear systems



$$\overline{\boldsymbol{K}} = (C\overline{\boldsymbol{B}}, C\overline{\boldsymbol{AB}}, \dots, C\overline{\boldsymbol{A}}^{k}\overline{\boldsymbol{B}}, \dots)$$

$$y = x * \overline{\boldsymbol{K}}$$

Input u(t), Output : y(t)

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$
$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$

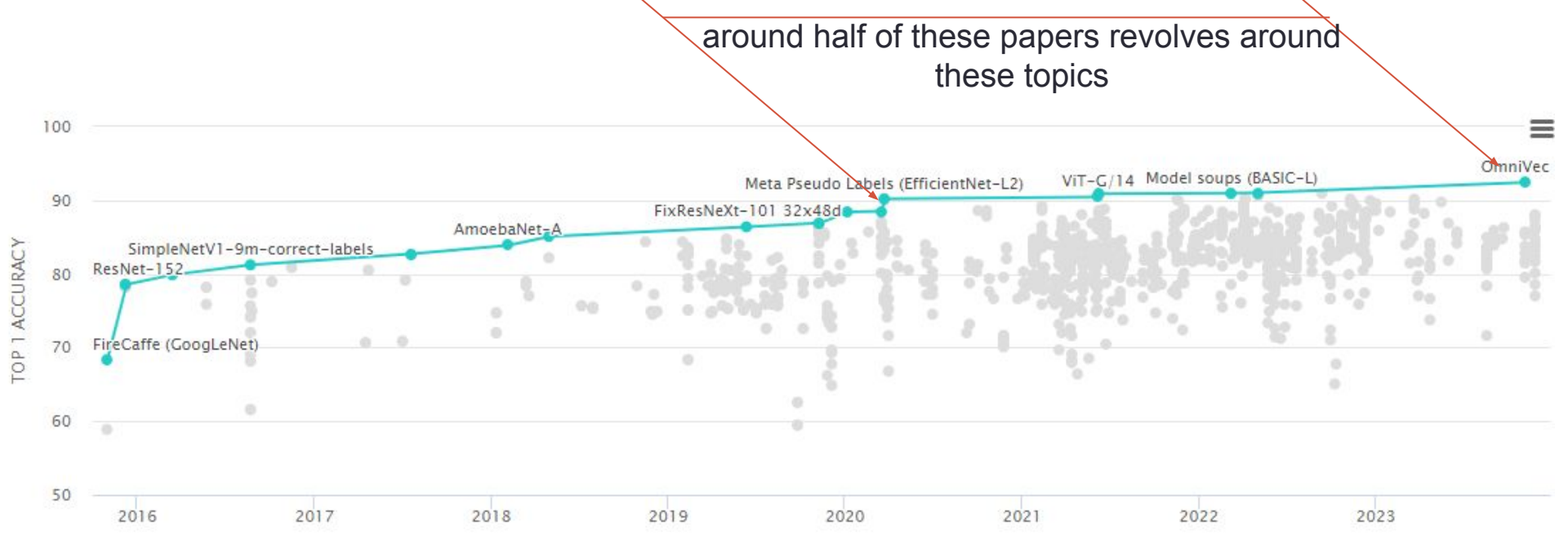
RNN Input x, Output : y

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Efficiently Modeling Long Sequences with Structured State Spaces (2022)
Mamba: Linear-Time Sequence Modeling with Selective State Spaces (2023)

Trends after ResNet

- More efficient convolutional block design
- Wider networks, deeper models, denser connections
- Global Context + Transformer architecture
- **Better representation learning (self-supervision / semi-supervision / multi-modality)**



Object classifiers summary

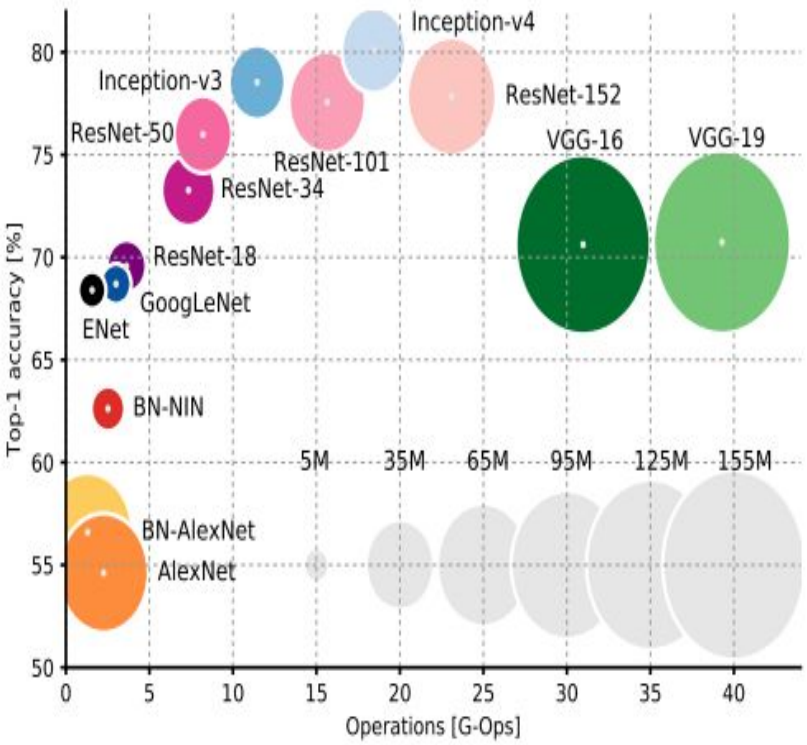
Most successful tricks:

residual, batch norms (layer norms), multi-path, data augmentation

These object classifiers are often called backbones and used by other models

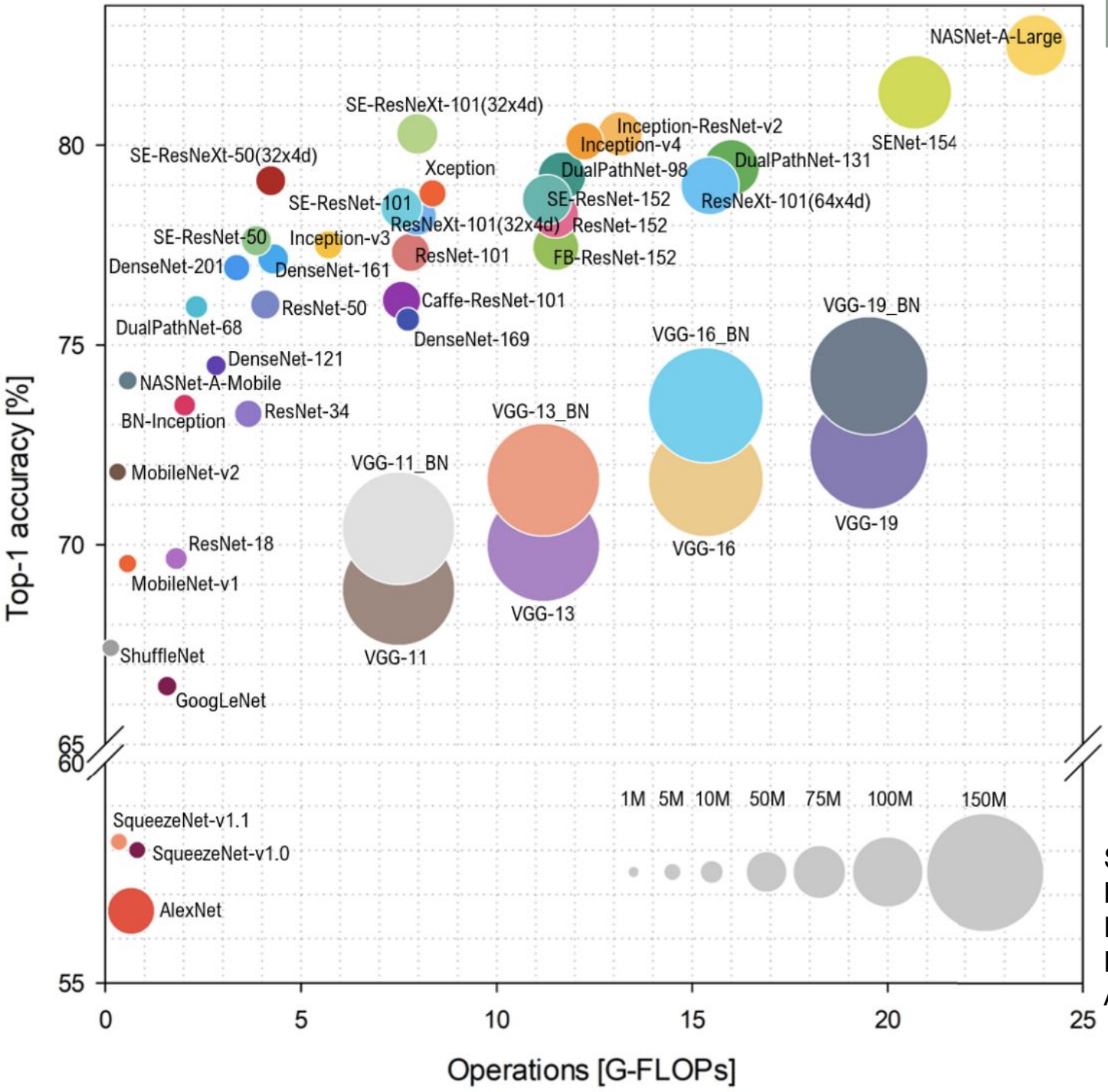
- Feature Pyramid is a neck
- Output layer is the head

Comparing accuracy, model size, transferability and compute.



High number of parameters
!=
High memory utilization

Alfredo Canziani, et al. “An analysis of deep neural network models for practical applications,” 2017



Simone Bianco, et al., Benchmark Analysis of Representative Deep Neural Network Architectures, 2019

FLOP and FPS are different

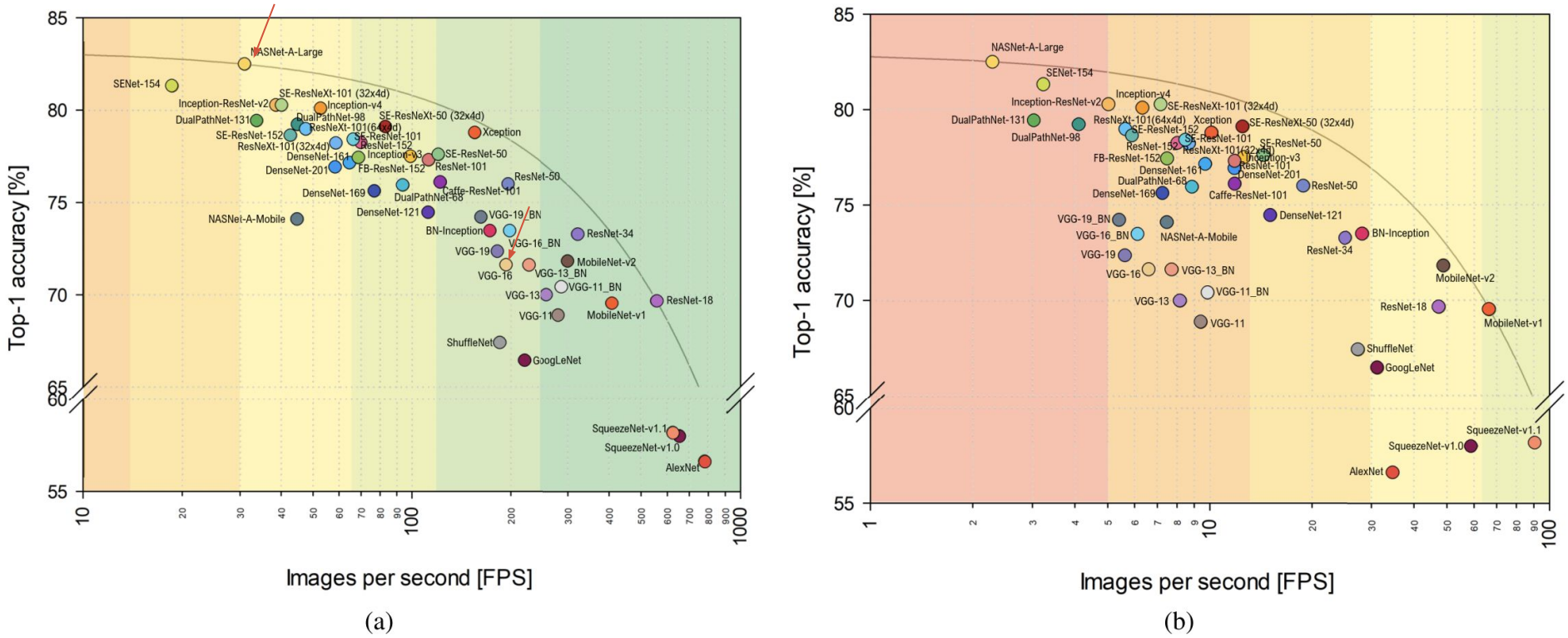
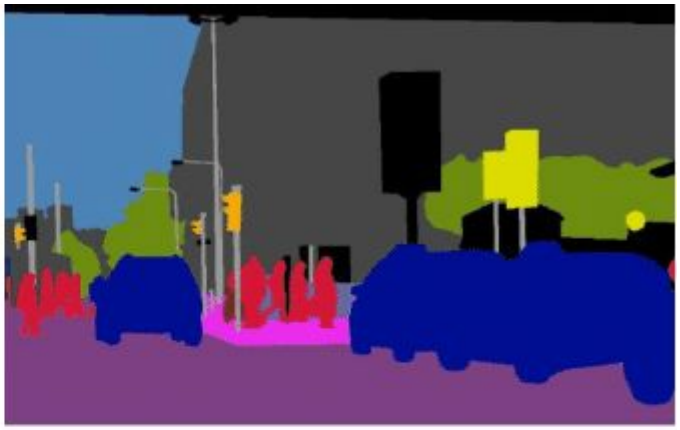


FIGURE 3. **Top-1 accuracy *vs.* number of images processed per second (with batch size 1) using the Titan Xp (a) and Jetson TX1 (b).**

Simone Bianco, et al., Benchmark Analysis of Representative Deep Neural Network Architectures, 2019

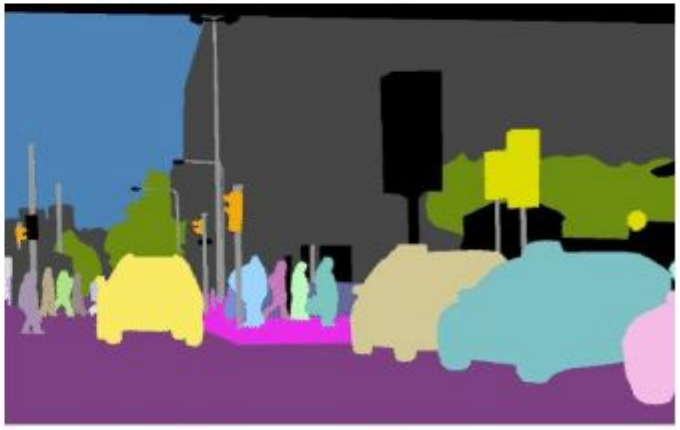
Common computer vision task

(a) Image



(b) Semantic Segmentation

(c) Instance Segmentation



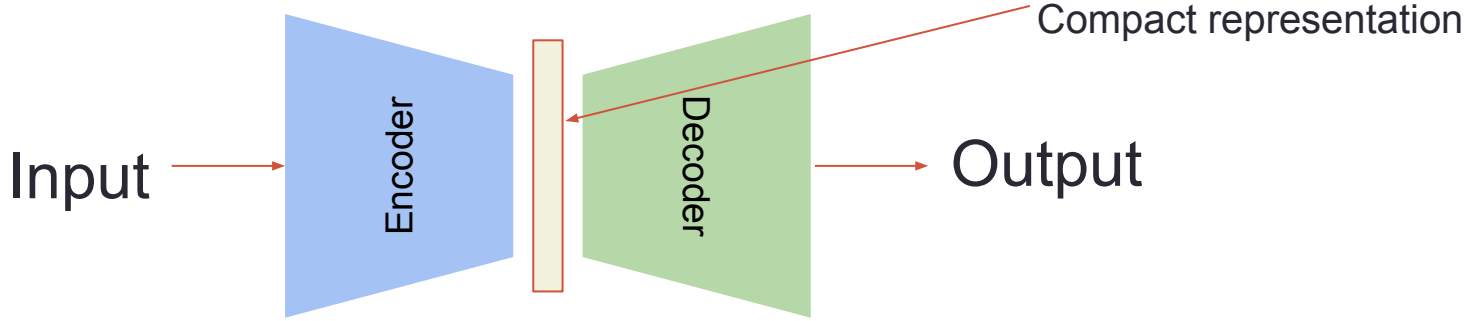
(d) Panoptic Segmentation

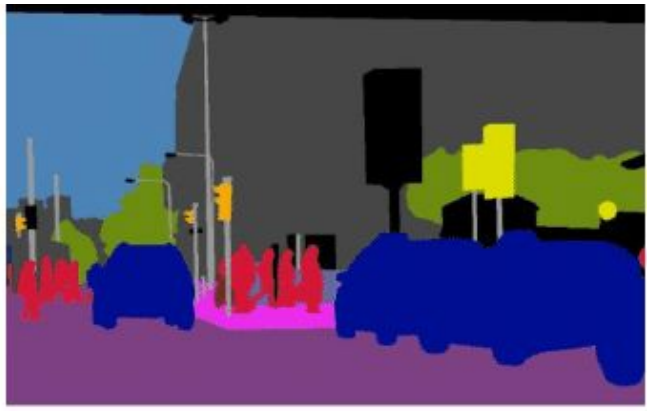
+Object detection

A Survey on Deep Learning for Localization and Mapping: Towards the Age of Spatial Machine Intelligence

Semantic segmentation

- Input : H x W x C
- Output : H x W x (n_class + 1) ;
- Usually treated as binary classification task on pixel level
- Usually learned using encoder-decoder architecture





Unet

- A typical architecture for segmentation task
- Have a skip connection to preserve resolution

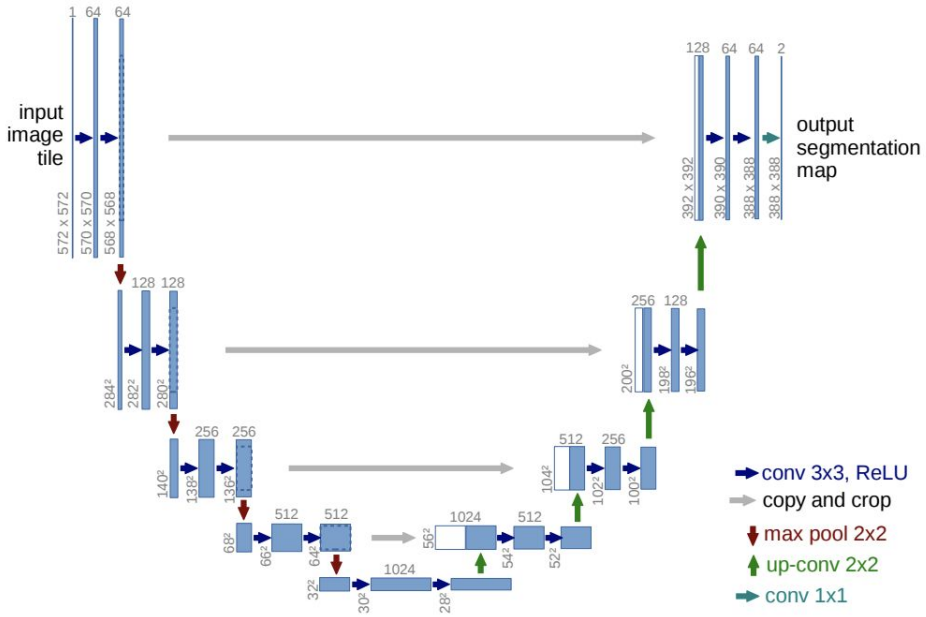
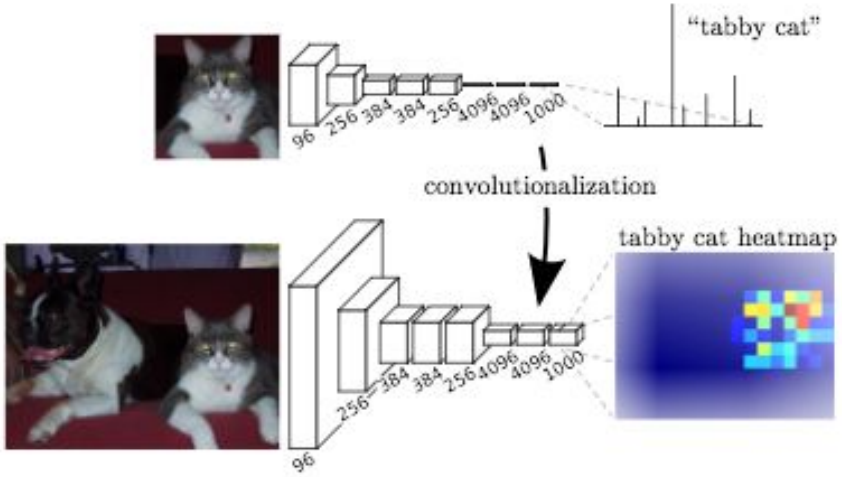


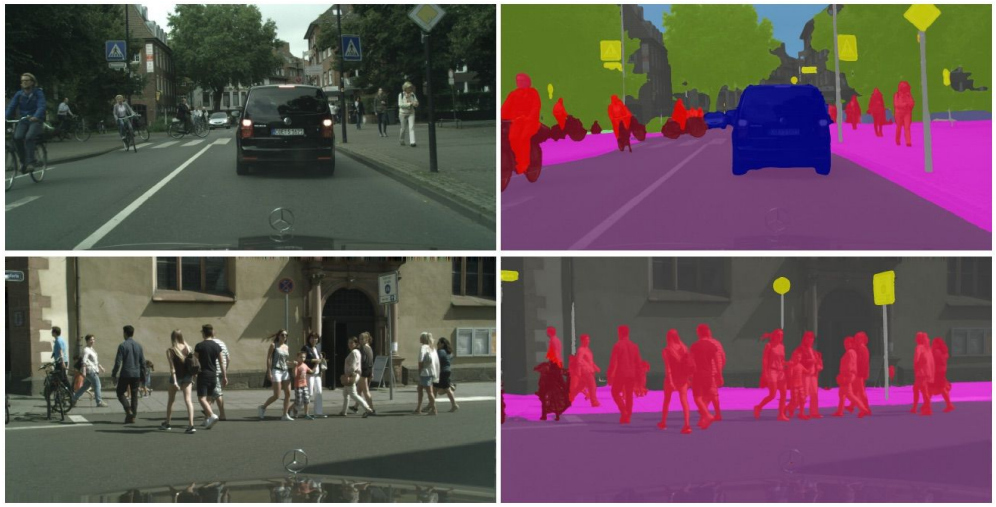
Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

U-Net: Convolutional Networks for Biomedical Image Segmentation (2015)

Sometimes you want to increase the size of your feature map

Image segmentation



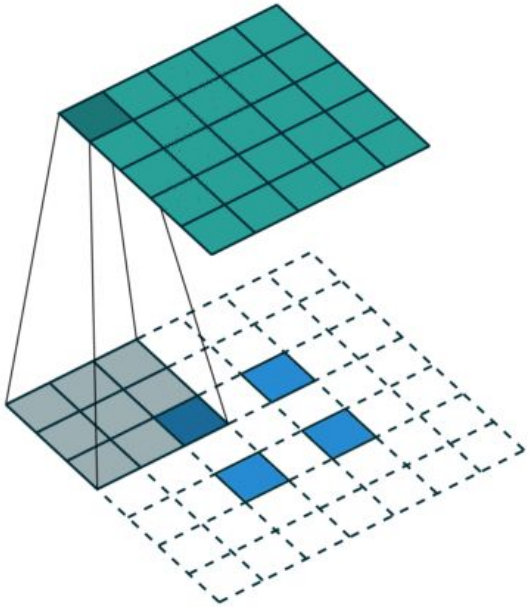


2 main approaches to upsample
De-convolution
resize (unpooling) +convolution

https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf
http://vladlen.info/publications/feature-space-optimization-for-semantic-video-segmentation/

De-convolution (Upsampling)

- 3x3 de-convolution filter, stride 2, pad 1



Other names because this name sucks (for me)
- Convolution transpose, upconvolution, backward strided convolution

https://medium.com/geekculture/deconvolution-and-how-it-works-and-its-importance-80ce8f137e1b

Upsampling notes

Deconvolution filter size should be a multiple of the stride to avoid checkerboard artifacts

Unpooling can be replaced with regular image resizing techniques (interpolation)

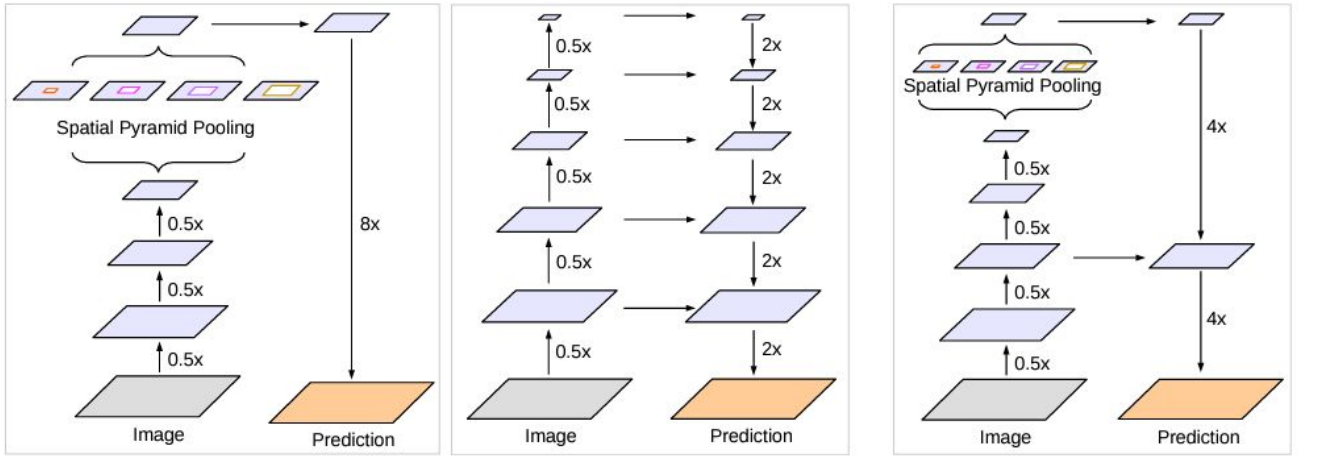
Image using deconv

Image using resize upsampling

https://distill.pub/2016/deconv-checkerboard/

Deeplab / V2 / V3 / V3+

- Similar to Unet but mainly focus on natural image
- Predictions are more rough



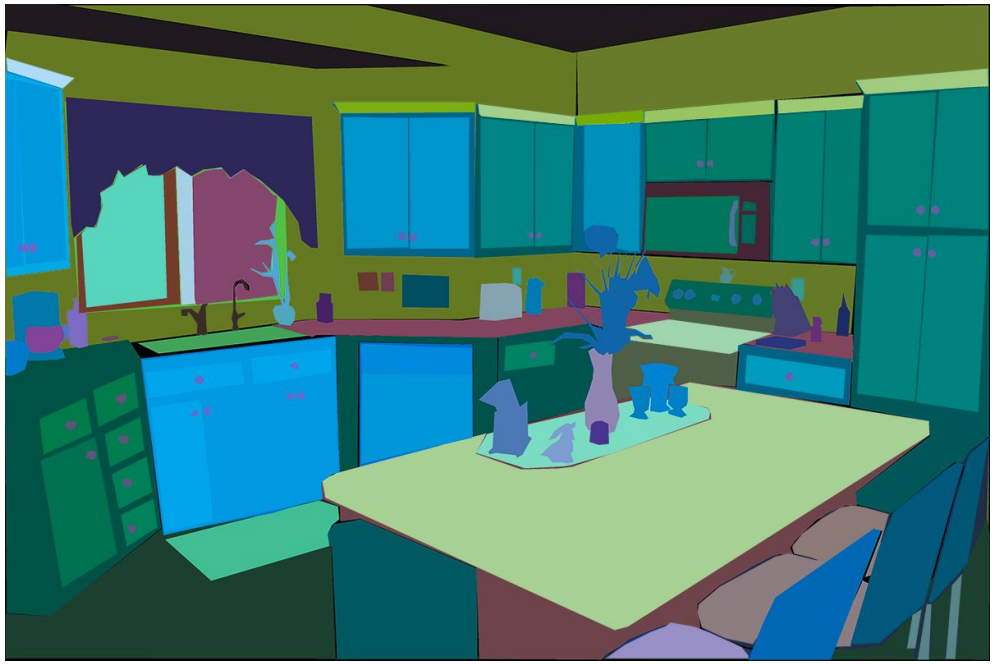
(a) Spatial Pyramid Pooling (b) Encoder-Decoder (c) Encoder-Decoder with Atrous Conv

Fig. 1. We improve DeepLabv3, which employs the spatial pyramid pooling module (a), with the encoder-decoder structure (b). The proposed model, DeepLabv3+, contains rich semantic information from the encoder module, while the detailed object boundaries are recovered by the simple yet effective decoder module. The encoder module allows us to extract features at an arbitrary resolution by applying atrous convolution.

Rethinking Atrous Convolution for Semantic Image Segmentation (2017)

ADE20K

Harder semantic segmentation task



Dataset stats

The current version of the dataset contains:

- 27,574 images (25,574 for training and 2,000 for testing) spanning 365 different scenes.
- 707,868 unique objects from 3,688 categories, along with their WordNet definition and hierarchy.
- 193,238 annotated object parts and parts of parts.

https://github.com/CSAILVision/ADE20K

Guide on choosing the architecture

- Is it solved before? -> Copy!
- Is it similar to some standard vision/NLP task? -> copy with modification
- Nothing close?
  - What would a human do?
  - What kind of inductive bias should we put in?
    - What kind of invariance does the data has?
    - What kind of noise/augmentation would work?
- Start simple then add extra
- Overfit then think of ways to regularize

Implementation

- Codes usually written as blocks

https://github.com/milesial/Pytorch-UNet/

```python
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=False):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        factor = 2 if bilinear else 1
        self.down4 = Down(512, 1024 // factor)
        self.up1 = Up(1024, 512 // factor, bilinear)
        self.up2 = Up(512, 256 // factor, bilinear)
        self.up3 = Up(256, 128 // factor, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.outc = OutConv(64, n_classes)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
```

```python
class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bilinear, use the normal convolutions to reduce the number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        # if you have padding issues, see
        # https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-Buggy/commit/0e854509c2cea854
        # https://github.com/xiaopeng-liao/Pytorch-UNet/commit/8ebac70e633bac59fc22bb5195e513d5832
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)
```

Other stuff to look for

Layers

Dilated convolution

https://towardsdatascience.com/understanding-2d-dilated-convolution-operation-with-examples-in-numpy-and-tensorflow-with-d376b3972b25

Deformable convolution

https://arxiv.org/abs/1703.06211

Necks

Feature Pyramid Networks (multi-scale backbone)

https://arxiv.org/abs/1612.03144

CNN Summary

Building blocks

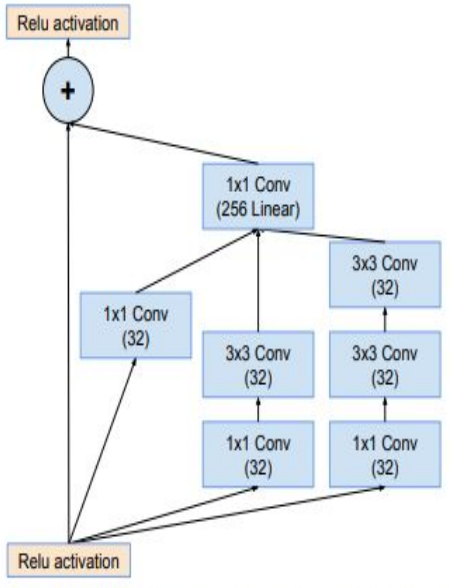
Matched filters and pooling

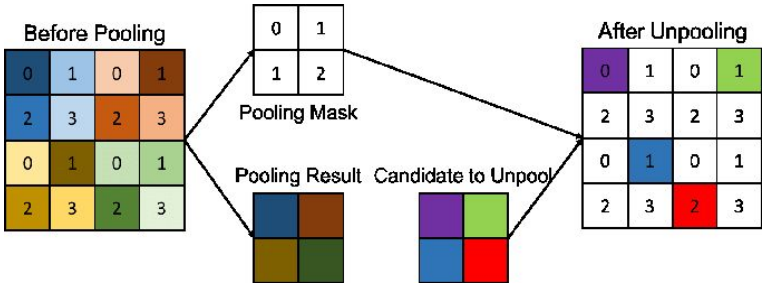
Filter factorization

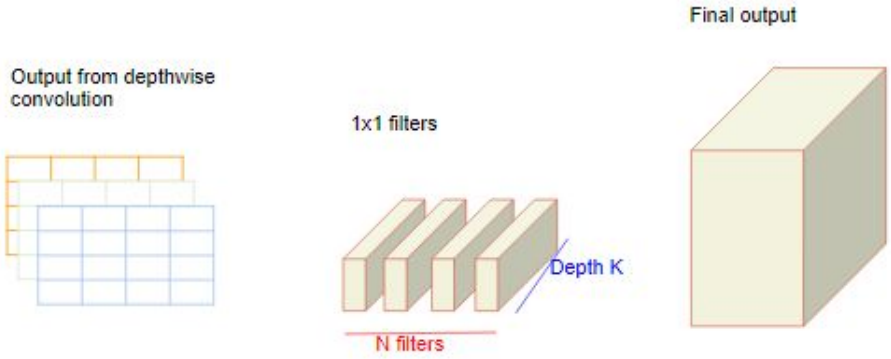
- 5x5 -> two 3x3
- depthwise vs matrix factorization

1x1 convolution

Deconvolution and upsampling

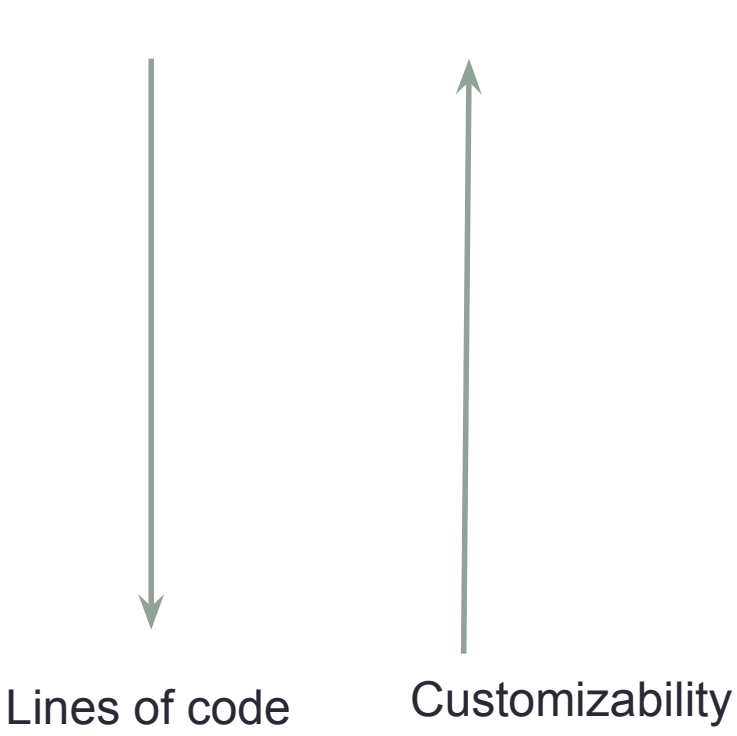


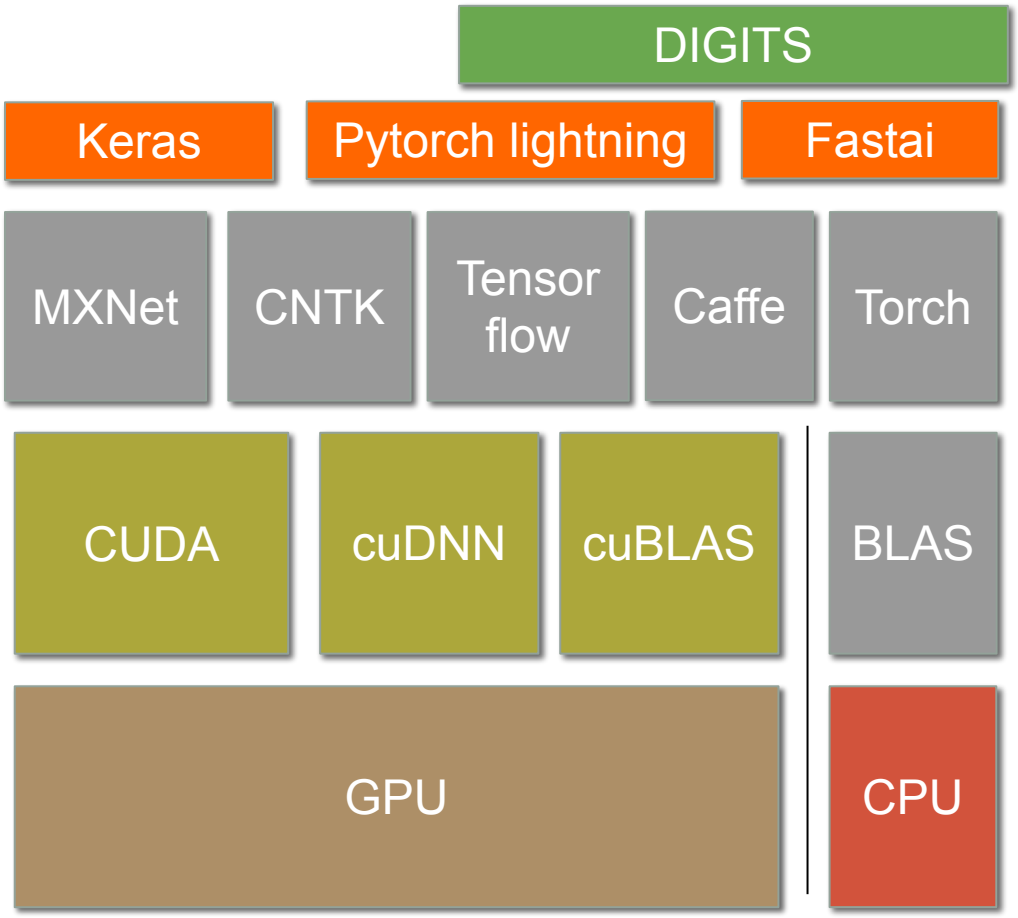




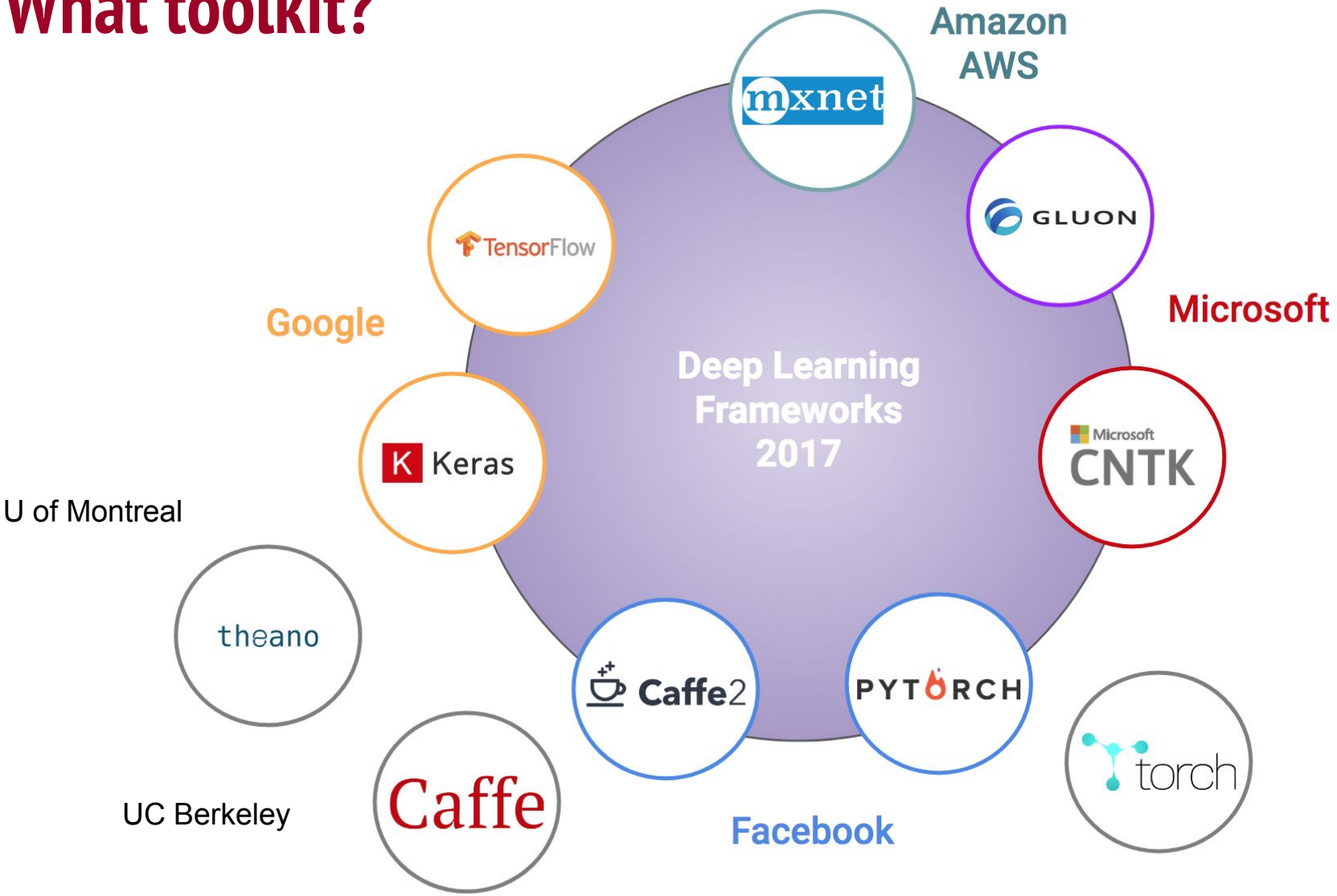
What toolkit

Tradeoff between customizability and ease of use



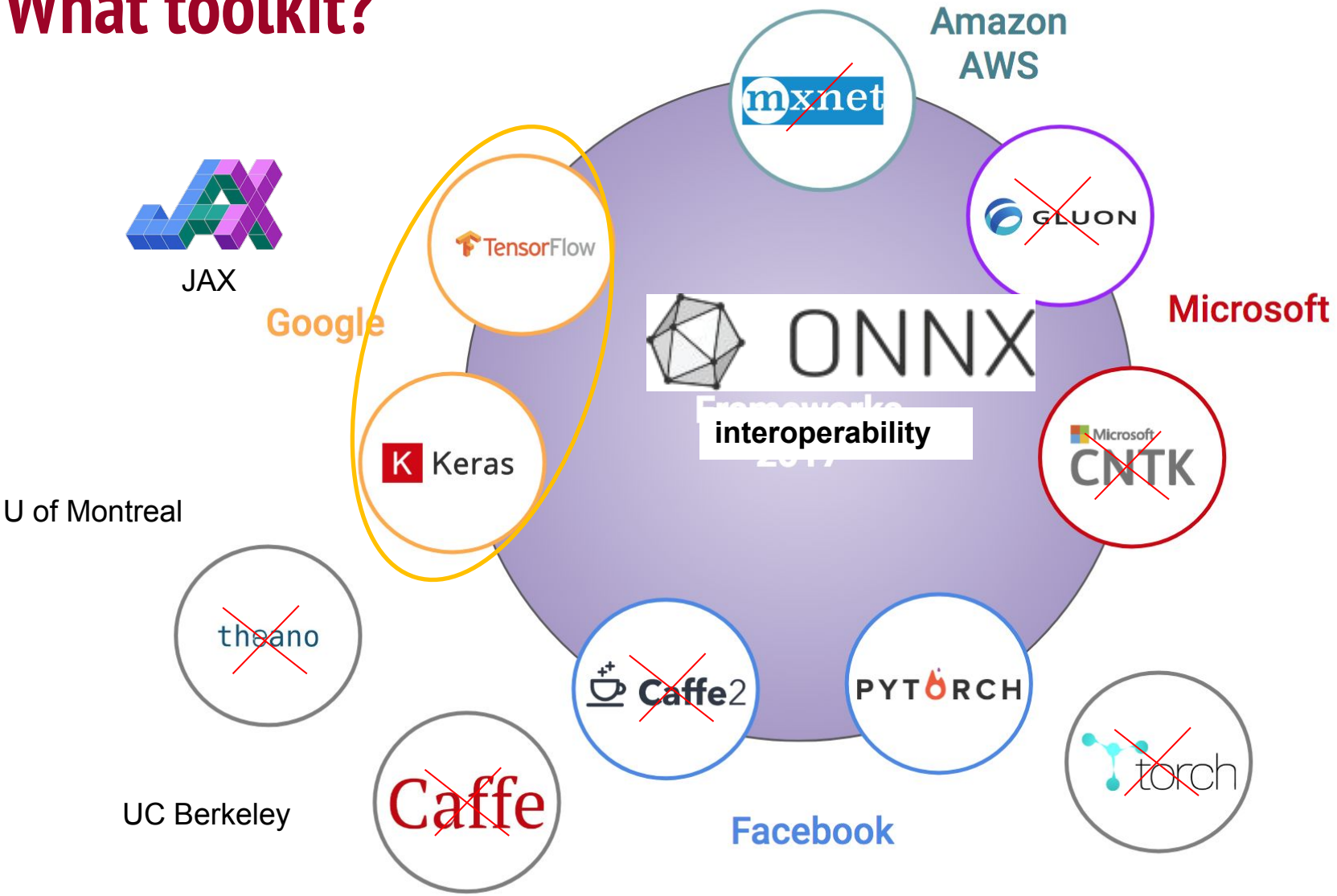


What toolkit?



https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

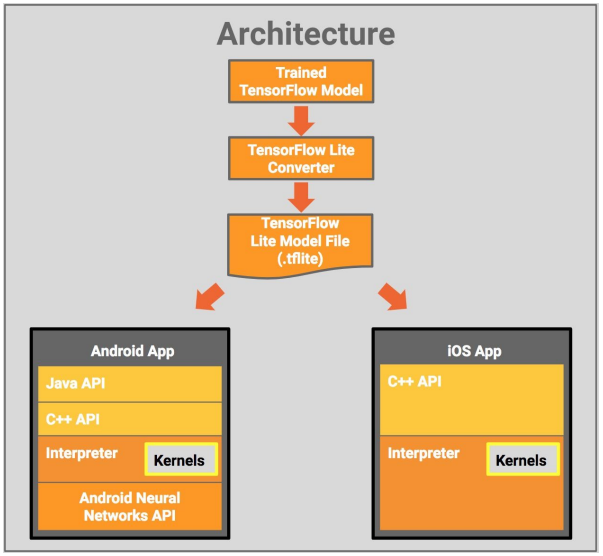
What toolkit?

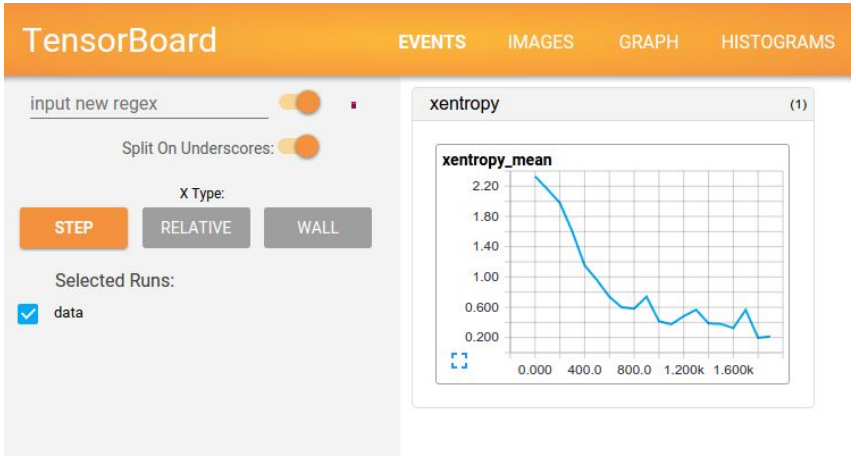


https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
  - Javaruntime support
- Best tools: increasingly pytorch
- Community: increasingly pytorch
- In the end you should know some tf and pytorch

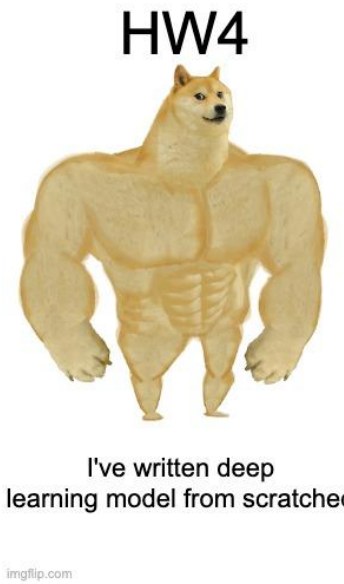




Pytorch steps

- Setting up dataloader
  - Gives minibatch
- Define a network
  - Init weights
  - Define computation graph
- Setup optimization method
  - Pick LR scheduler
  - Pick optimizer
- Training loop
  - Forward (compute Loss)
  - Backward (compute gradient and apply gradient)

- Let’s demo



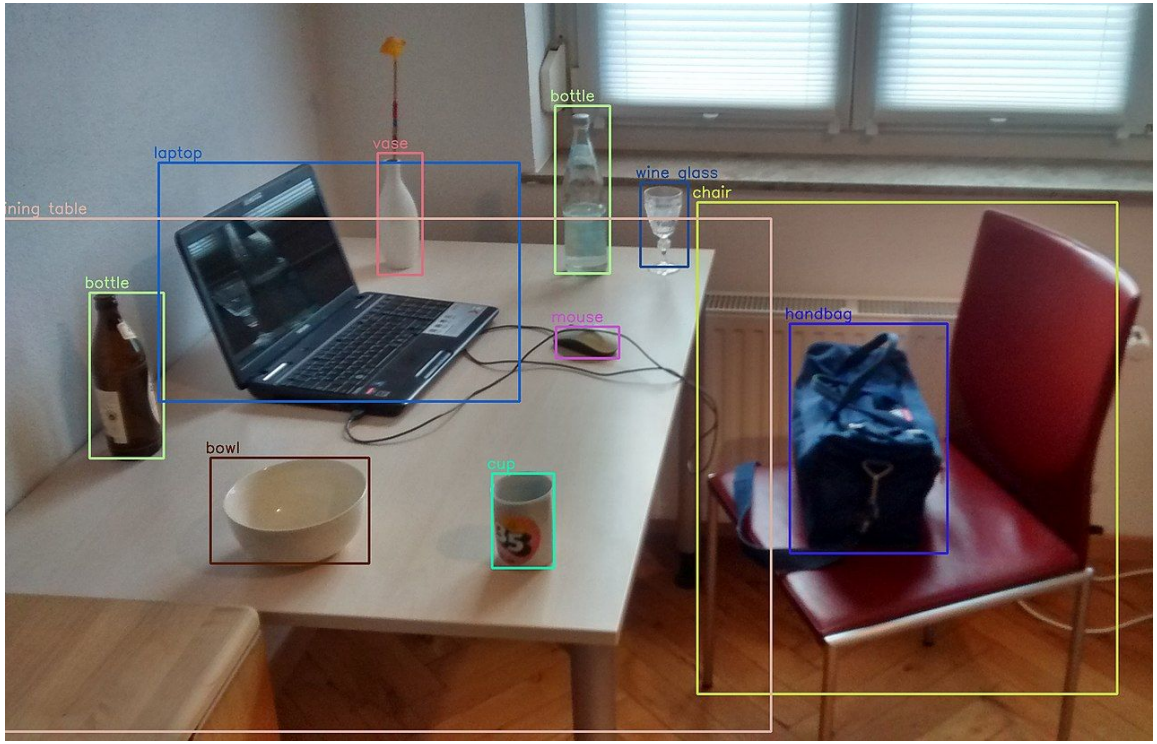
Debugging guide

- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/guide3/Debugging_PyTorch.html
  has list of common errors and best practices
- http://karpathy.github.io/2019/04/25/recipe/
  has guide for end-to-end model building (start simple and go more advance)

Appendix

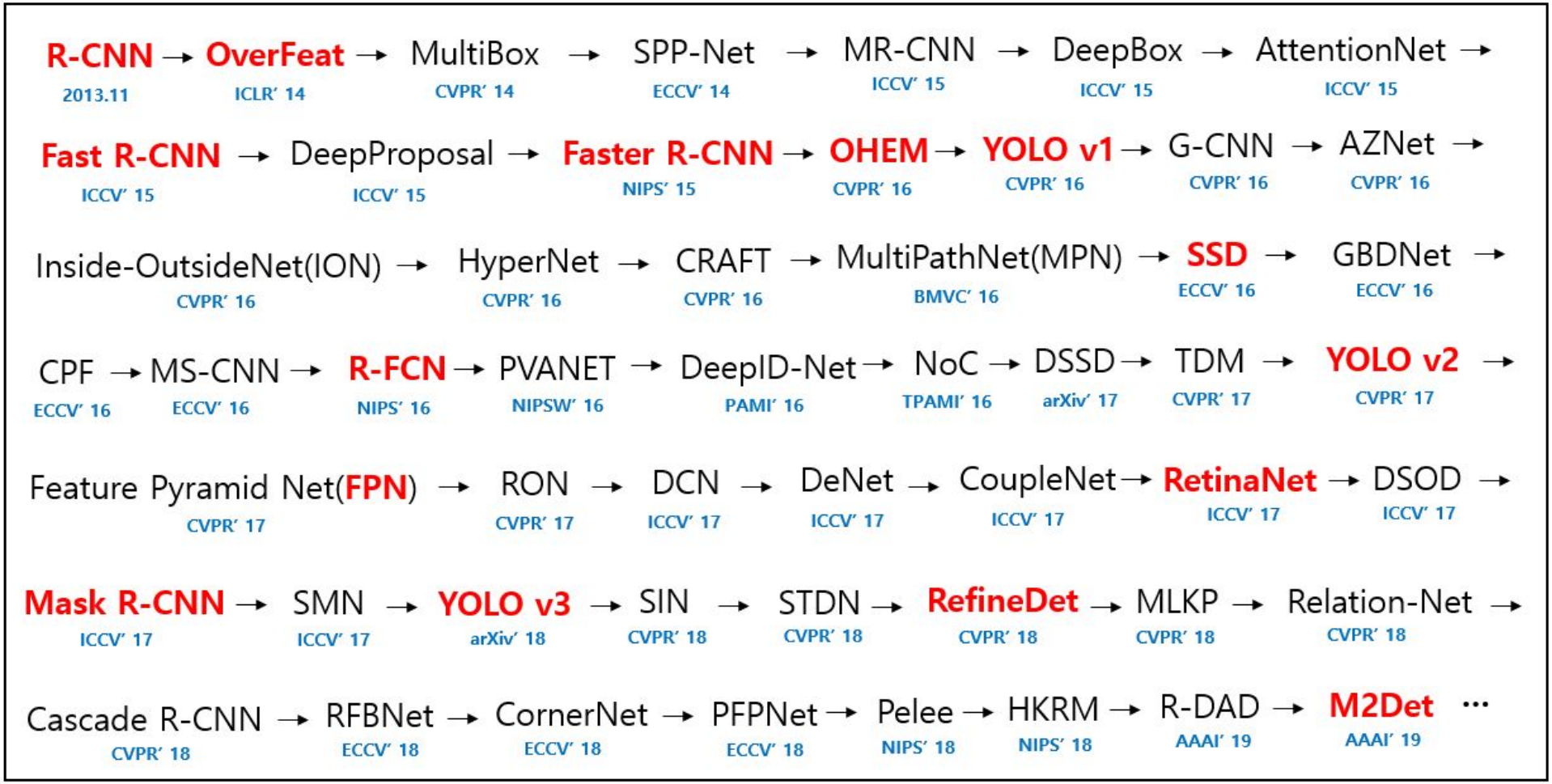
Object Detection

- Input : H x W x C
- Output : { (x, y, w, h, n_class +1 or n_class) }



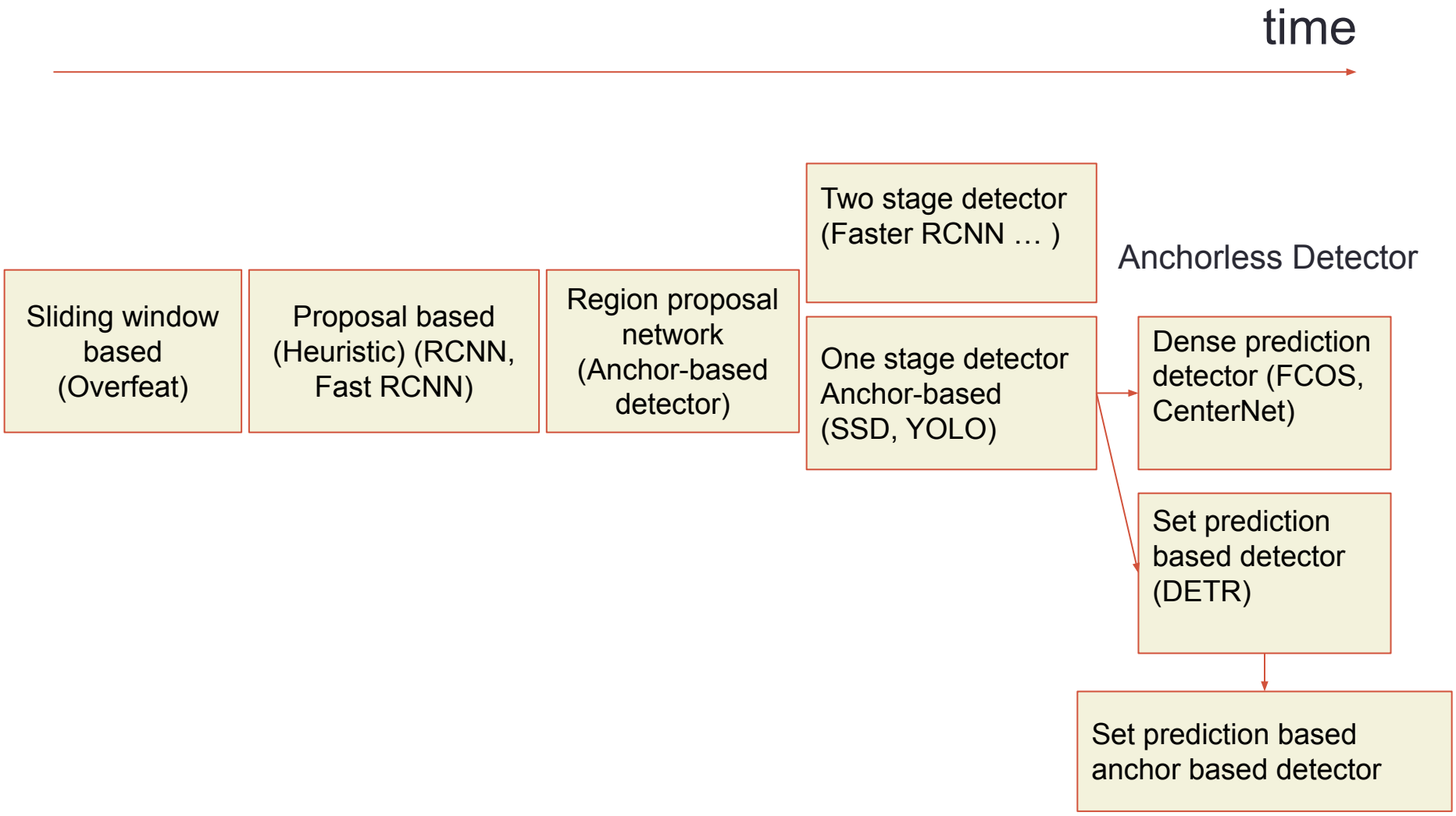
https://en.wikipedia.org/wiki/Object_detection#/media/File:Detected-with-YOLO--Schreibtisch-mit-Objekten.jpg

Object Detection History



https://media.licdn.com/dms/image/C4E12AQHGBblzH3QHLA/article-cover_image-shrink_720_1280/0/1626487694785?e=1714003200&v=beta&t=bTzkGDTHiYfQdkbLiB8LyK6XMeNdY3EwfuxvWJfEmW0

Object Detection History



The goods old days

Haar cascade / Viola Jones Detector (~2001)

https://www.youtube.com/watch?v=hPCTwxF0qf4&ab_channel=AnkurDivekar

Rapid Object Detection using a Boosted Cascade of Simple Features (2001)

Sliding window based detector

- Follow old traditional detector (Harr) but used CNN for prediction instead
- Also predict object boundary (shift)

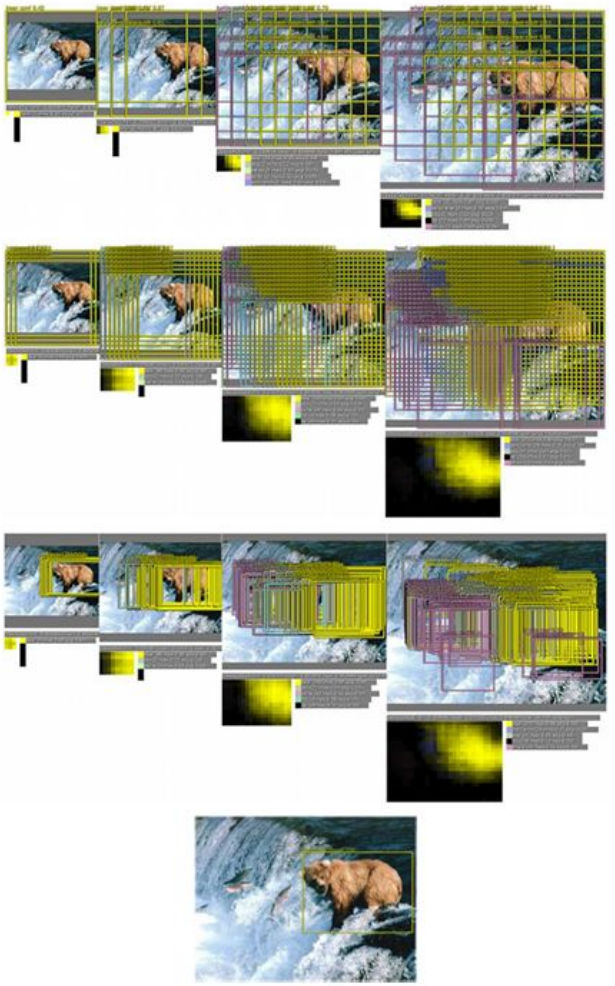
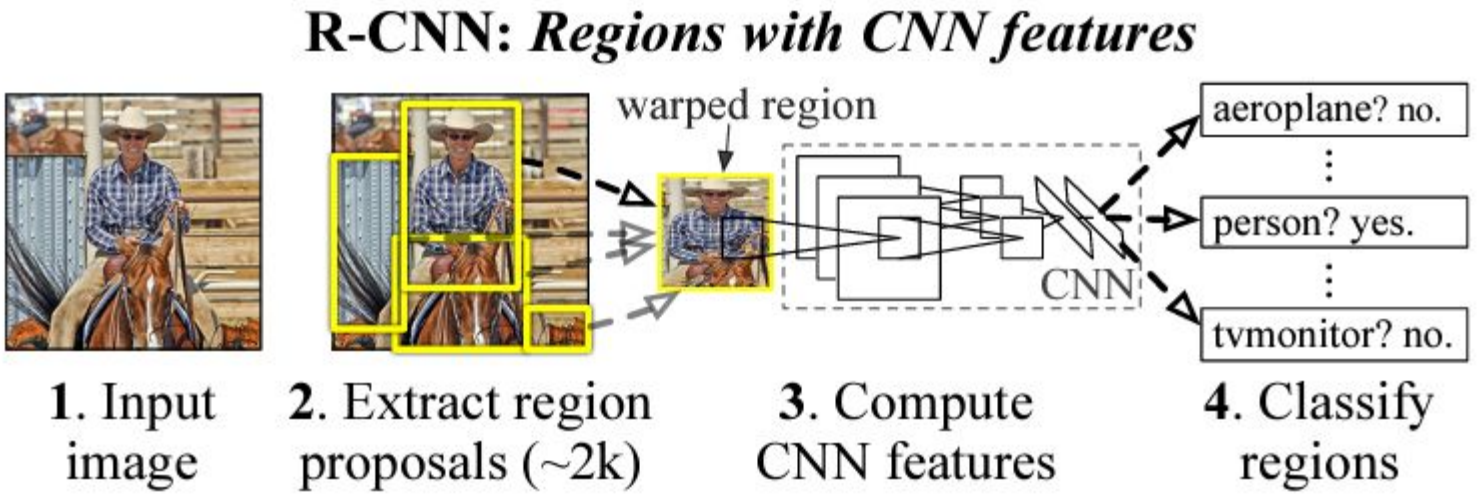


Figure 6: **Localization/Detection pipeline.** The raw classifier/detector outputs a class and a confidence for each location (1st diagram). The resolution of these predictions can be increased using the method described in section 3.3 (2nd diagram). The regression then predicts the location scale of the object with respect to each window (3rd diagram). These bounding boxes are then merge and accumulated to a small number of objects (4th diagram).

OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks (2013)

Proposal based detector (R-CNN)

- Instead of sliding through the whole image, we should only focus on area that are more likely to contain objects
- Region proposals are generated through heuristics (Selective search)
- Apply CNN to every proposal to classify whether it is object or background (still very slow)



Rich feature hierarchies for accurate object detection and semantic segmentation (2014)

Selective search

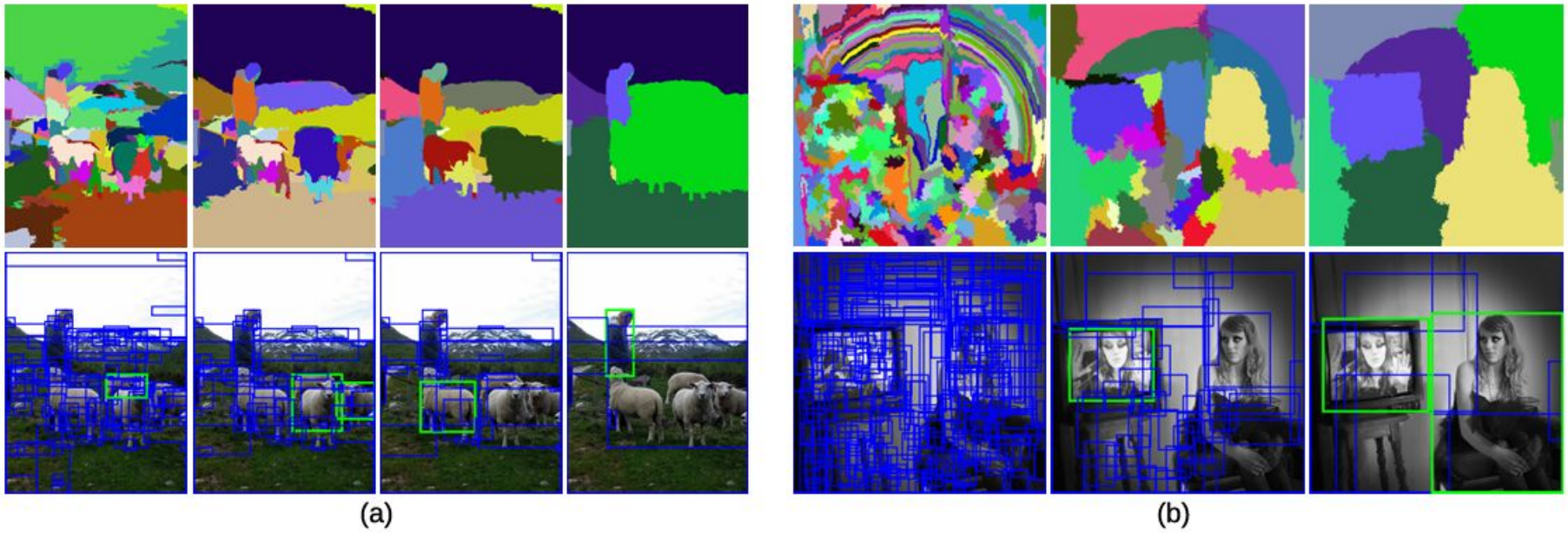


Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

Selective Search for Object Recognition (2013)

Fast R-CNN

- Instead of performing inference on every proposal, we perform inference on the whole image only once
- Feature are extracted using ROI pooling to obtain a fixed-size feature map
- It still requires heuristic to generate proposal

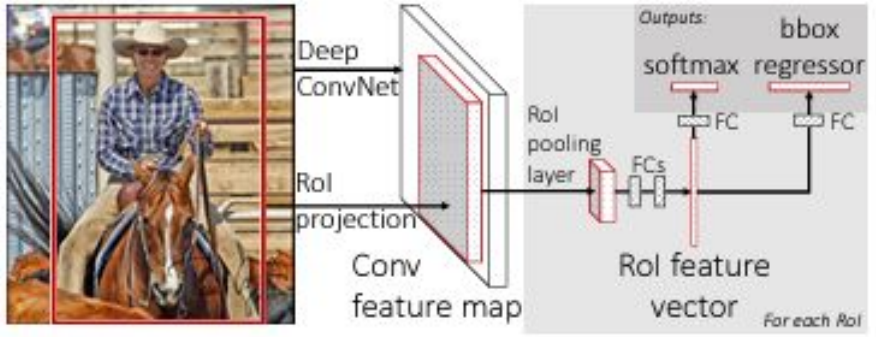
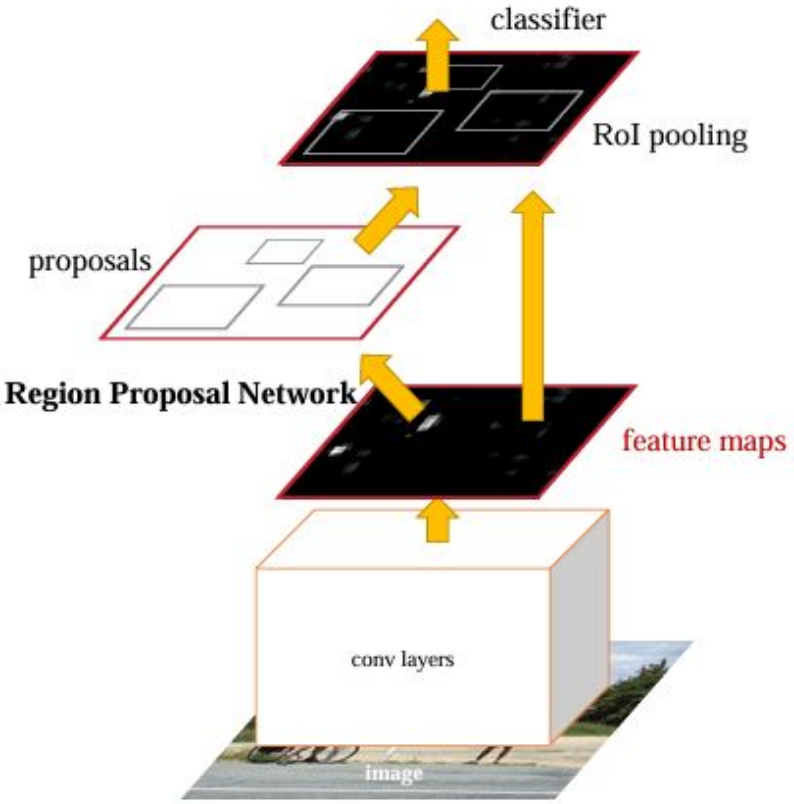


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

Fast R-CNN (2015)

Faster R-CNN (Anchor-based detector)



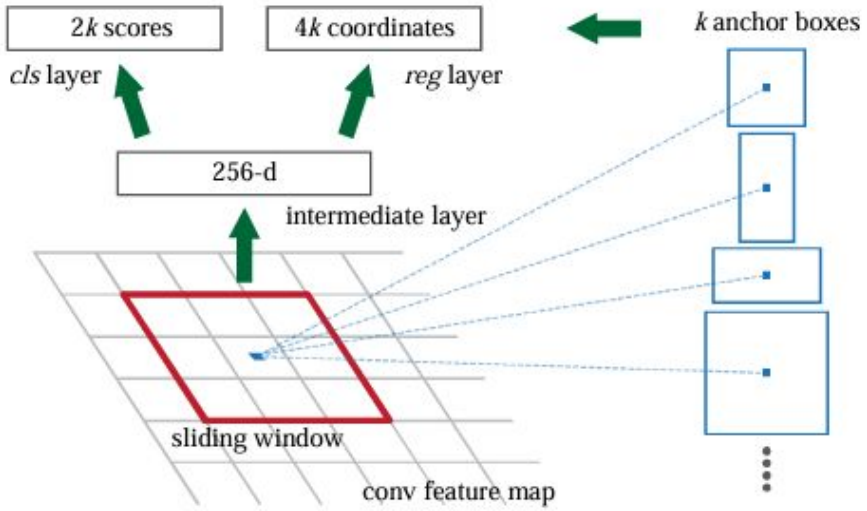


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (2016)

Mask R-CNN (Instance segmentation)

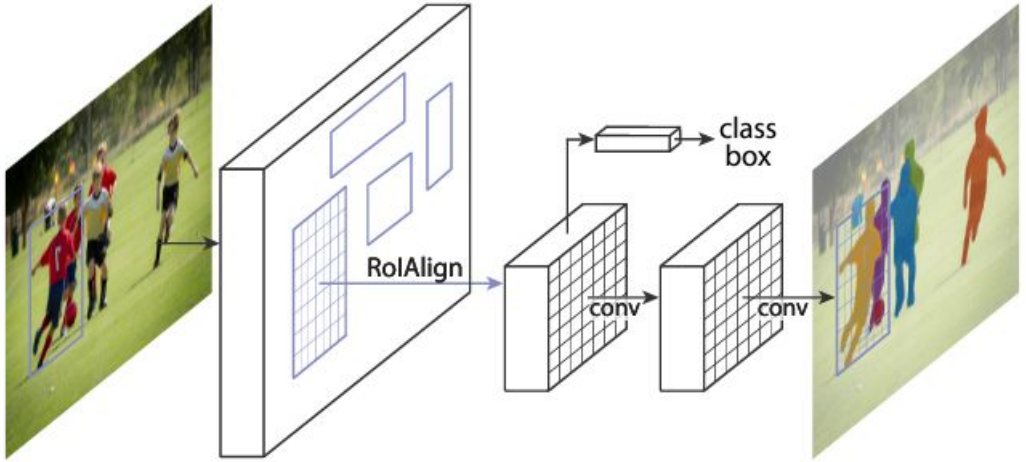


Figure 1. The **Mask R-CNN** framework for instance segmentation.

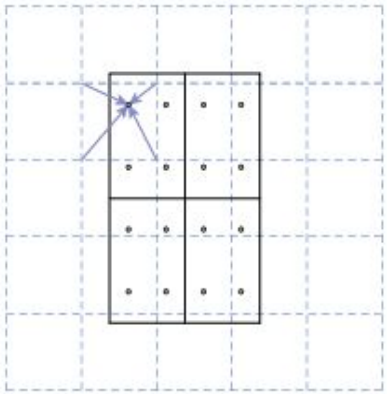
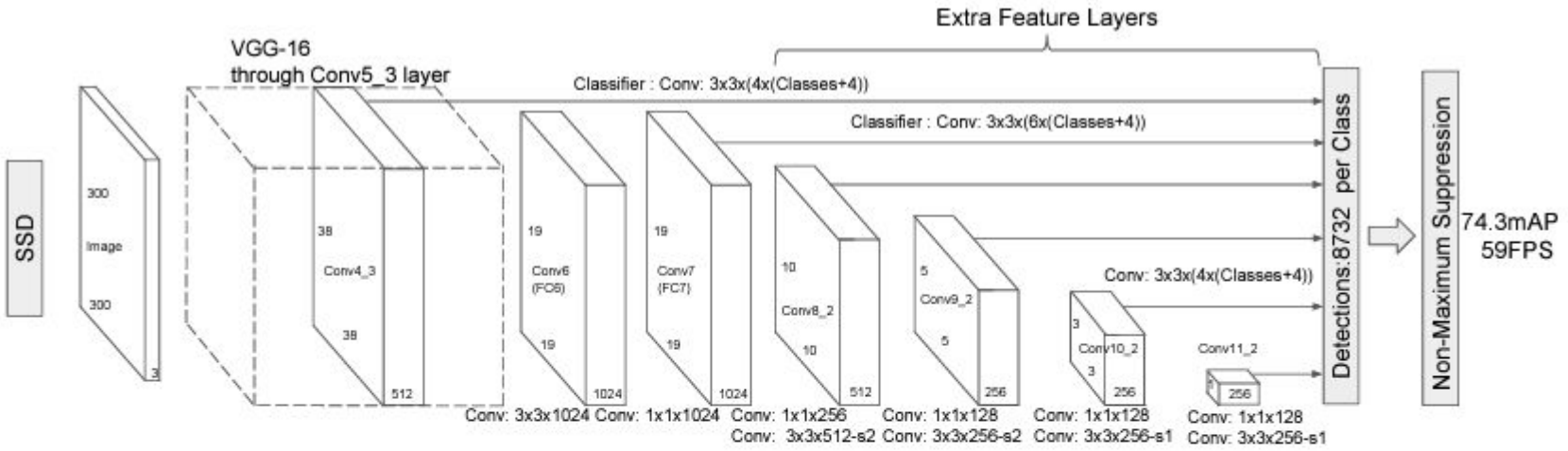


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

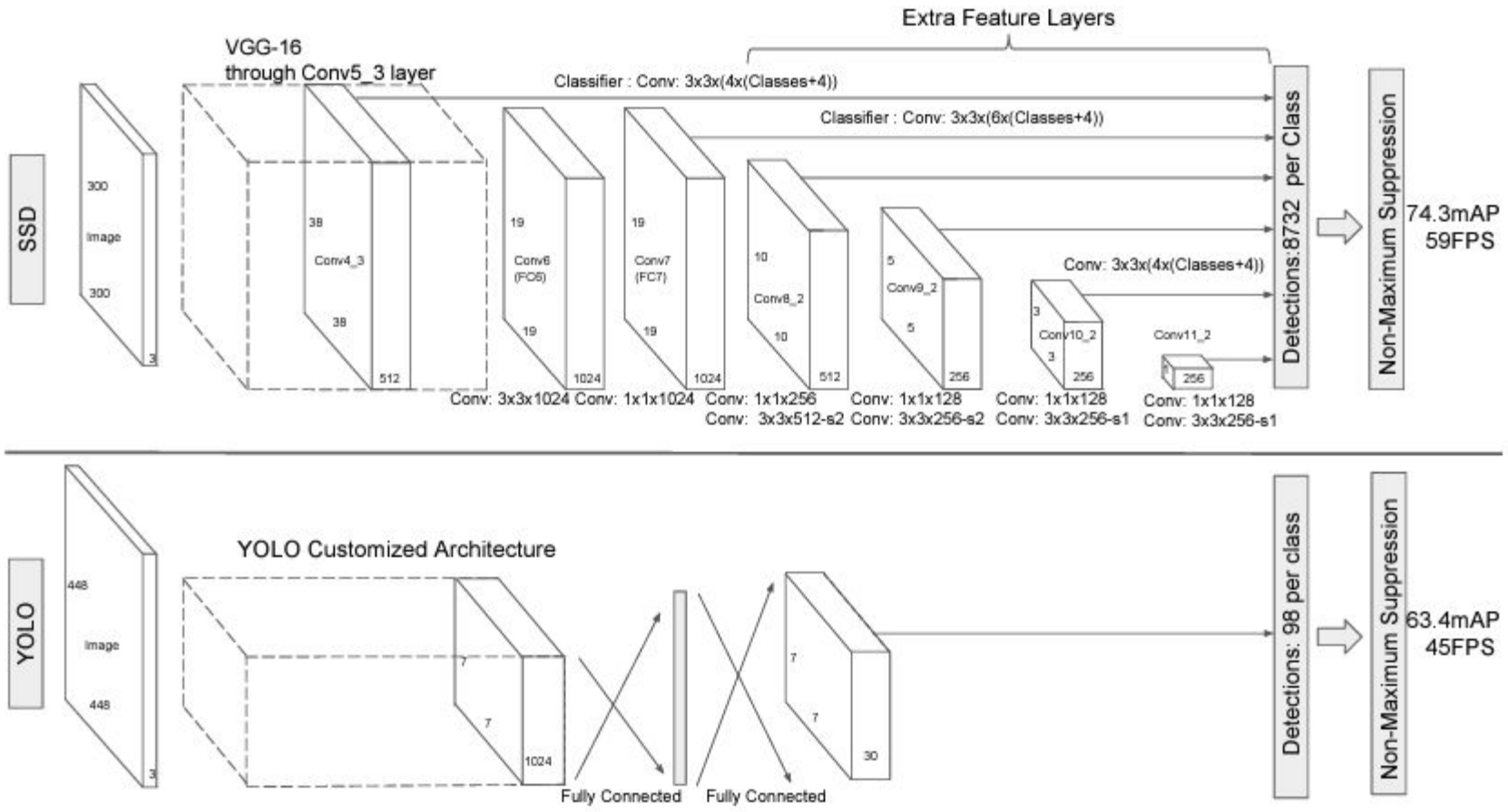
Mask R-CNN (2017)

SSD



SSD: Single Shot MultiBox Detector (2016)

SSD / YOLO


SSD
300
Image
300
3
VGG-16
through Conv5_3 layer
38
Conv4_3
38
512
19
Conv6
(FC6)
19
1024
19
Conv7
(FC7)
19
1024
10
Conv8_2
10
512
5
Conv9_2
5
256
3
Conv10_2
3
256
Conv11_2
1
256
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256
Conv: 3x3x512-s2
Conv: 1x1x128
Conv: 3x3x256-s2
Conv: 1x1x128
Conv: 3x3x256-s1
Conv: 1x1x128
Conv: 3x3x256-s1
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP
59FPS
YOLO
448
Image
448
3
YOLO Customized Architecture
7
7
1024
Fully Connected
Fully Connected
7
7
30
Detections: 98 per class
Non-Maximum Suppression
63.4mAP
45FPS

YOLOv2

Bunch of tricks combined together

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

YOLO9000: Better, Faster, Stronger (2016)

Feature Pyramid Network (FPN)

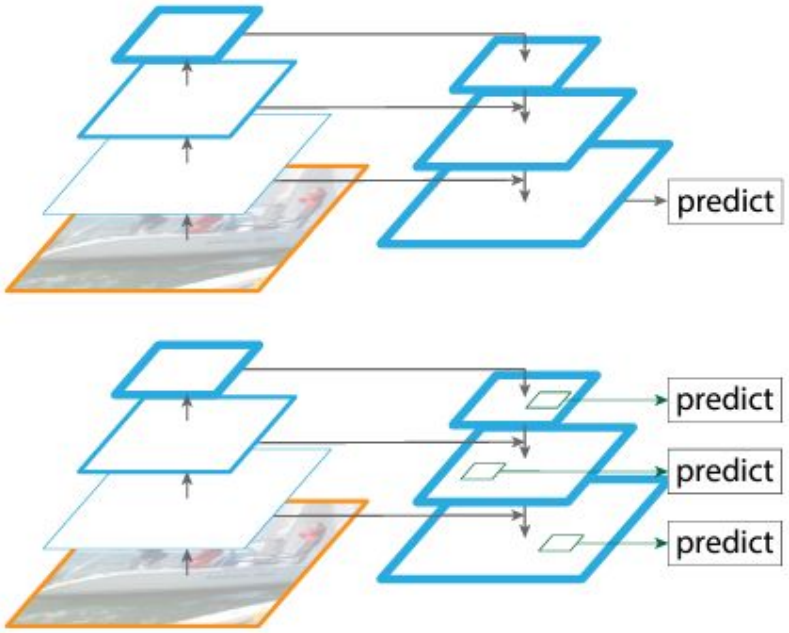


Figure 2. Top: a top-down architecture with skip connections, where predictions are made on the finest level (*e.g.*, [28]). Bottom: our model that has a similar structure but leverages it as a *feature pyramid*, with predictions made independently at all levels.

Feature Pyramid Networks for Object Detection (2017)

FPN War

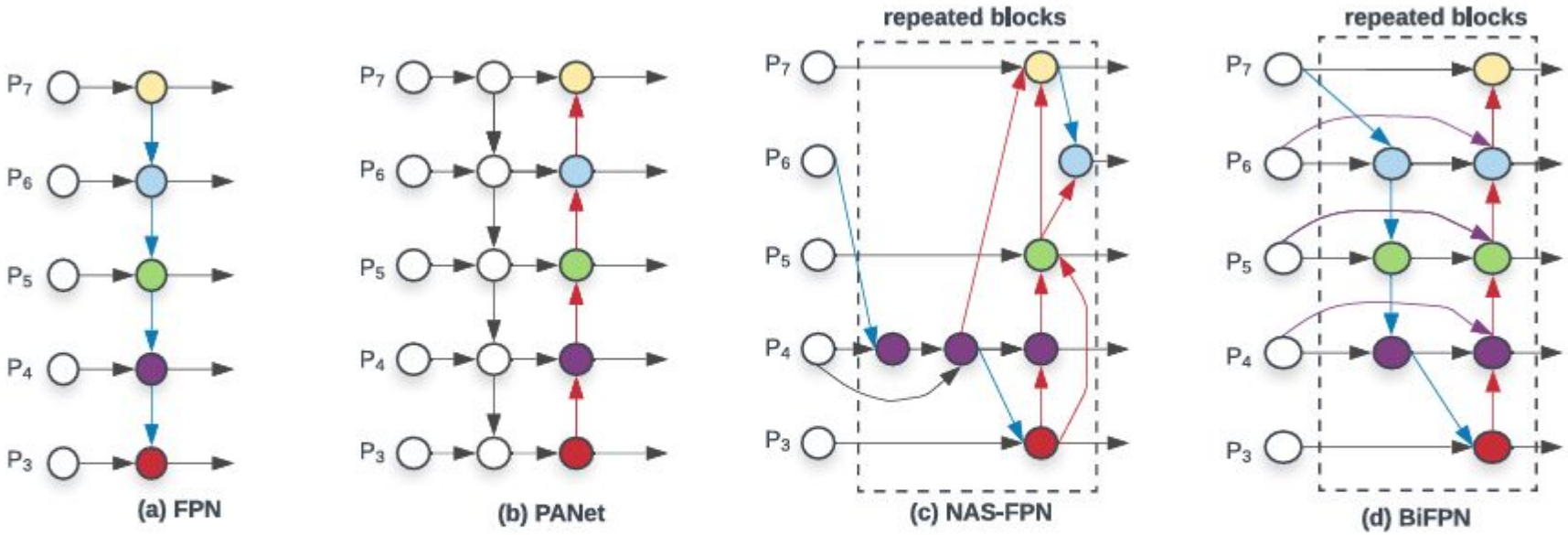


Figure 2: **Feature network design –** (a) FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

YOLOv3

YOLOv2 + FPN

RetinaNet

Mitigate the problem caused by FPN

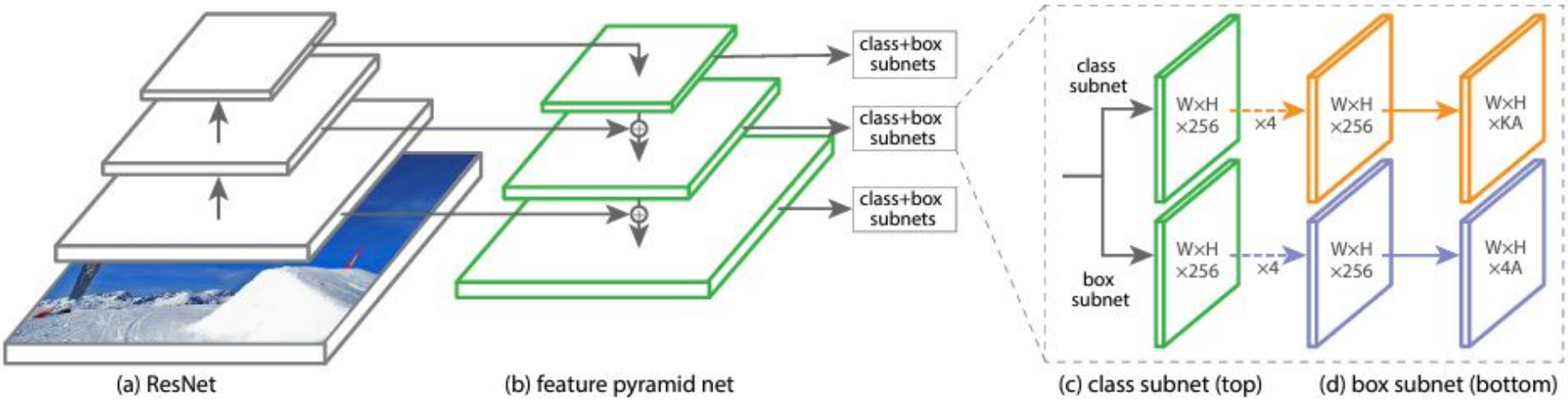


Figure 3. The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [20] while running at faster speeds.

Focal Loss for Dense Object Detection (2018)

FCOS (Anchorless detector)

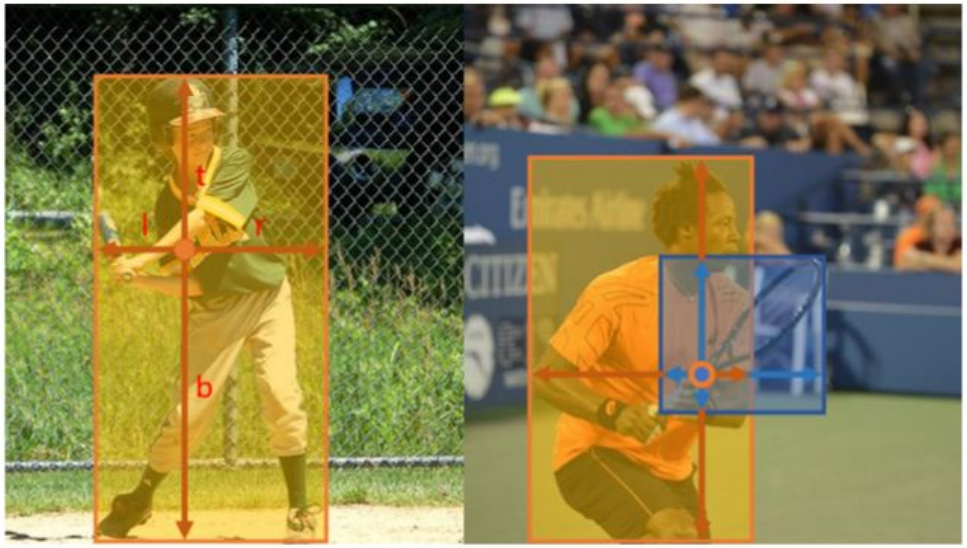


Figure 1 – As shown in the left image, FCOS works by predicting a 4D vector (l, t, r, b) encoding the location of a bounding box at each foreground pixel (supervised by ground-truth bounding box information during training). The right plot shows that when a location residing in multiple bounding boxes, it can be ambiguous in terms of which bounding box this location should regress.

FCOS: Fully Convolutional One-Stage Object Detection (2019)

CenterNet

keypoint heatmap [C]

local offset [2]

object size [2]

Objects as Points (2019)

DETR
(Set prediction based detector)

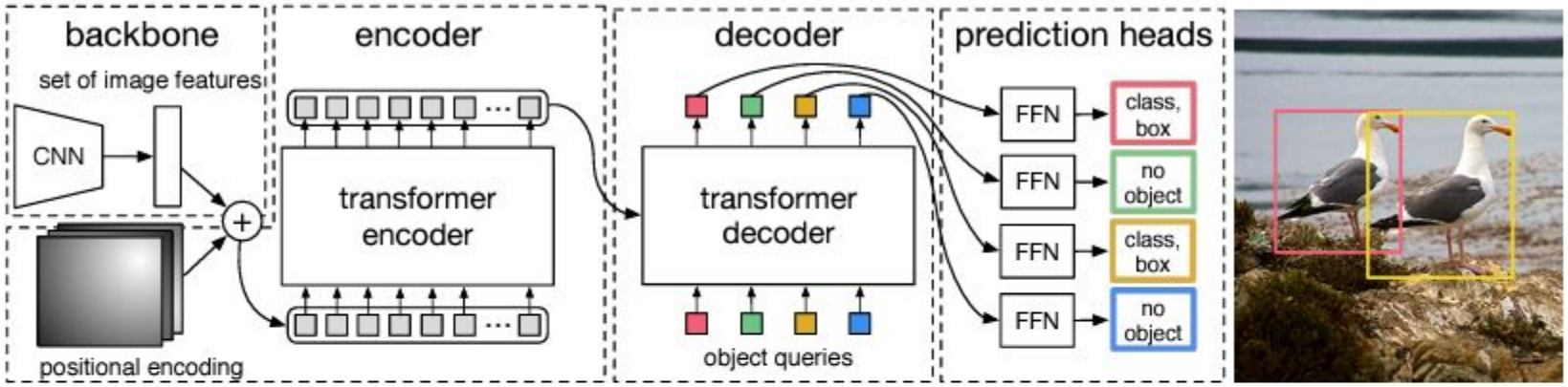


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

End-to-End Object Detection with Transformers (2020)