Analysis and Design of Deep Neural Networks

Ahmad Kalhor
Associate Professor
School of Electrical and Computer Engineering
University of Tehran
Spring 2022

Part1

Design of DNNs, by analyzing the well known layers, blocks, modules, and architectures

1. Structure analysis in DNNs

1.1 Layers, Blocks and Modules

- Fully Connected layers and blocks
- Convolution Layers-Blocks-Modules
- Recurrent Layers-Modules
- Attention Layers-Modules
- Pooling Layers
- Normalization Layers

1.2 Architectures

- CNNs
- Region Based CNNs (R-CNNs)
- CNNs for Segmentation
- CNNs for Segmentation
- Transformers

1.1 Layers, Blocks and Modules

1.1.1. Fully Connected layers and blocks

1.1.2 Convolution Layers-Blocks-Modules

1.1.3 Recurrent Layers –Modules

1.1.4 Attention Layers –Modules

1.1.5 Pooling Layers

1.1.6 Normalizing Layers

1.1.1 Fully Connected layers and their blocks

Ideal to make partitions, maps and encoded/decoded data from a set of distinct inputs.

- One FC layer
- A block of two FC layers
- A block of three FC layers
- A block of more than three FC layers

One FC layer

Definition

- $\boldsymbol{y} = \boldsymbol{f}(\mathrm{W}\boldsymbol{x} + \mathbf{b}),\ \mathrm{W} = [\boldsymbol{w}_{ji}]\ \mathrm{b} = [b_j]$
- $\boldsymbol{x} \epsilon R^n \quad \boldsymbol{y} \epsilon R^m \quad i \in \{1, \cdots, n\}\ j \in \{1, \cdots, n\}$
- Activation function:
- $f \in \{sign, \mathrm{step}, \tanh, sigmoid, Relu, identity \cdots\}$

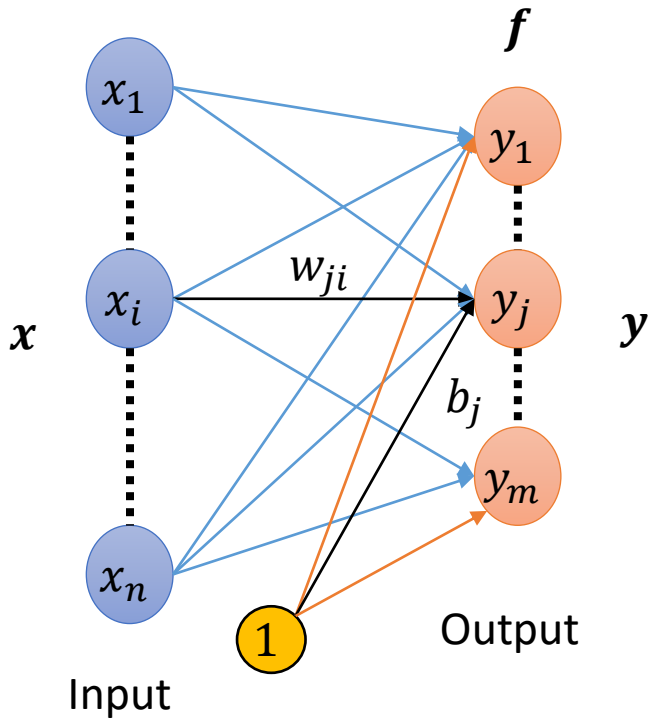
Functionality

(1) Forming an "n"dimensional hyper plane:

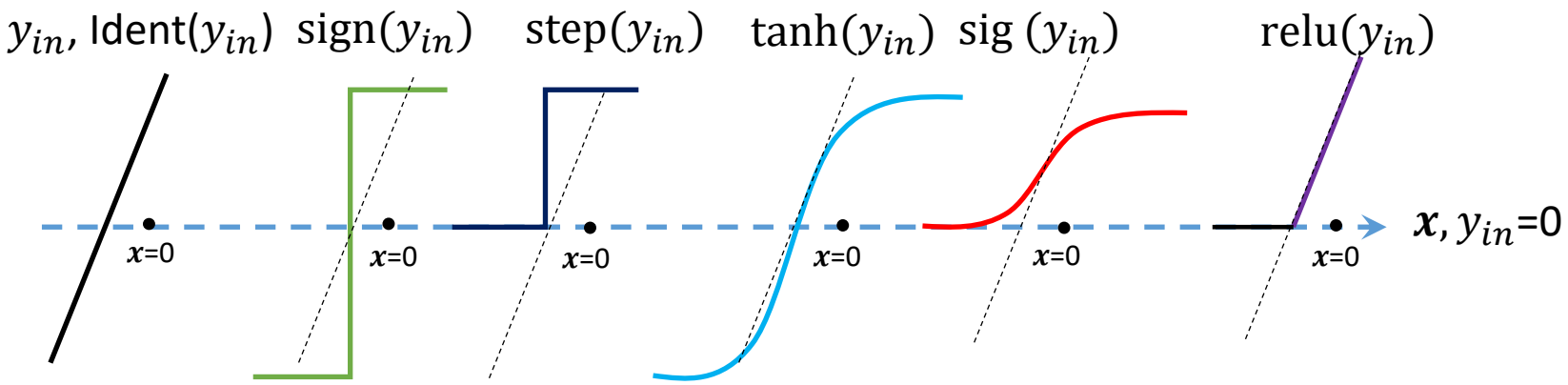
$$y_{in_j} = \mathrm{w_{j1}x_1} + \cdots + \mathrm{w_{jn}x_n} + \mathrm{b_j}$$

(2) Folding (hard/soft), Rectifying,.. on the hyper plane:

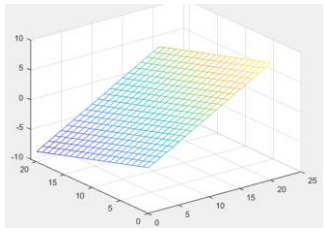
$$y_j = f(y_{in_j})$$



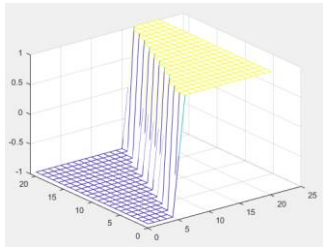
Example(1) for $n = 1$, $f(yin)$, $y_{in} = 2x + 1$ w:slope parameter , b: $(up \uparrow down \downarrow) shift\ parameter$



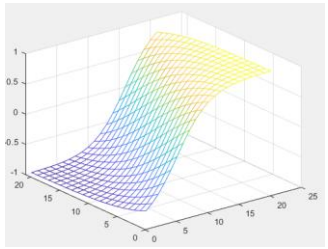
Example(2) for $n = 2$, $f(yin)$, $y_{in} = 2x_2 - x_1$+0



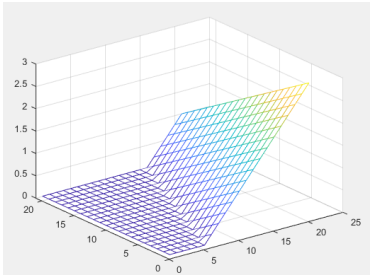
$y_{in}, ident(y_{in})$



sign(10y_{in})



tanh(y_{in})



Relu(y_{in})

❖ A “one FC layer” can transform the input space to a (hard or soft) folded or rectified hyper plane

Some Notes about one FC layer

1. One FC layer (with n inputs and m outputs) is formed from m neurons at output, where each neuron is connected to all n input units (fully connected).
2. Each input send a weighted signal to each neuron.
3. For each neuron, the summation of weighted inputs makes an independent hyper-plane just before activated by it.
4. The parameters of all hyper-planes are opted through a learning process.
5. For each neuron, the corresponding hyper-plane is hardly or softly folded or rectified just after activating by it.
6. Indeed, taking in inputs as a "n" dimensional vector, the FC layer provides "m" folded, or rectified hyper-planes at the output.
7. Assuming the inputs are binary or bipolar, and the activation functions make binary or bipolar values, a neuron in one FC layer can operate as a simple logic gate like “and”, “or, etc. (M&P neuron)
8. Assuming the activation functions of the neurons are identity, a FC layer operates as a linear transformer, which can approximate a linear regression model in a regression problem.
9. Assuming “r” independent linear or nonlinear correlations among inputs, the order of the formed hyper plan is “n-r”.

A block of two FC layers

Definition

- $\boldsymbol{y} = \boldsymbol{f}^2\left(\mathrm{W}^2 \underbrace{\boldsymbol{f}^{\mathbf{1}}\left(\mathrm{W}^1\boldsymbol{x} + \mathbf{b}^{\mathbf{1}}\right)}_{z} + \boldsymbol{b}^2\right)$
- $\boldsymbol{x} \epsilon R^n \boldsymbol{z} \epsilon R^{m_1} \boldsymbol{y} \epsilon R^m \quad i \in \{1, \cdots, n\}\ j \in \{1, \cdots, m_1\}\ k \in \{1, \cdots, m\}$
- Activation functions:
- $f^{1,2} \in \{sign, \text{step}, \tanh, sigmoid, Relu, identity \cdots\}$

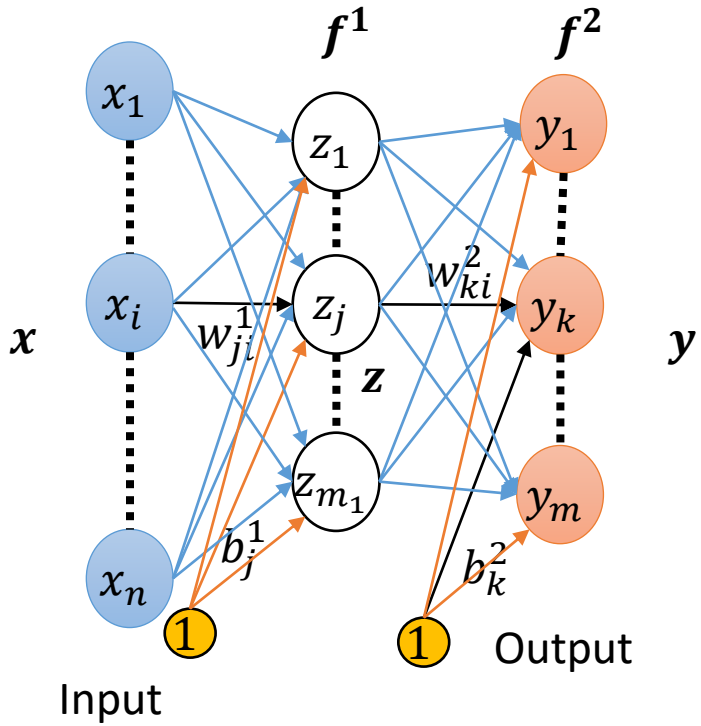
Overall Functionality

(1) A hyper plane: $z_{in_\mathrm{j}} = \mathrm{w}^1_{\mathrm{j1}}\mathrm{x}_1 + \cdots + \mathrm{w}^2_{\mathrm{jn}}\mathrm{x}_\mathrm{n} + \mathrm{b}^1_\mathrm{j}$

(2) Folding, Rectifying,.. on the hyper plane: $z_j = f^1(z_{in_\mathrm{j}})$

(3) A hyper plane: $y_{in_k} = \mathrm{w}^2_{\mathrm{k1}}\mathrm{z}_1 + \cdots + \mathrm{w}^2_{\mathrm{k}m_1}\mathrm{z}_{\mathrm{m}_1} + \mathrm{b}^2_\mathrm{k}$

(4) Folding, Rectifying,.. on the hyper plane: $y_k = f^2(y_{in_k})$



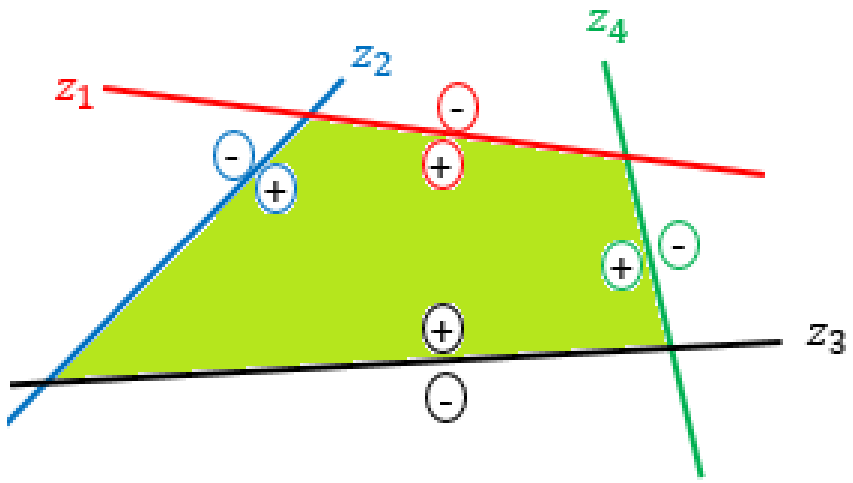
A Desired Functionality in classification /Regression problems

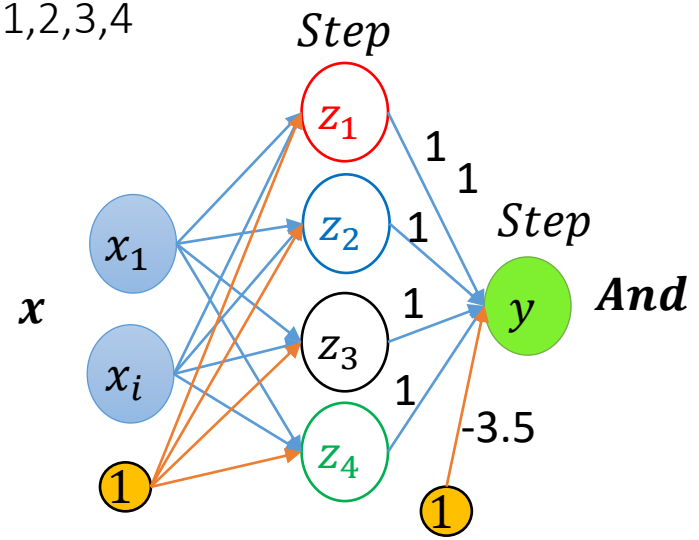
(1) A hyper plane: $z_{in_j} = w_{j1}^1 x_1 + \cdots + w_{jn}^2 x_n + b_j^1$

(2) Folding the hyper plane: $z_j = f^1(z_{in_j})$

(3) An "and" or "or logic gate for former folded hyper-planes is defined by "kth" neuron of last layer, by which "kth" convex hyper-polygon will be resulted.

Example for $n = 2, m_1 = 4, m=1$ $\quad z_j = w_{j1}^1 x_1 + w_{j2}^1 x_2 + b_j^1,$ $\quad$ j=1,2,3,4





- A block of two FC layers can transform the input space to multi convex hyper- polygon partitions
- Using soft activation functions, a block of two FC layers can transform the input space to multi soft convex hyper- polygon maps

A block of three FC layers

Definition

- $\boldsymbol{y} = f^3\left(\mathrm{W}^3\, \boldsymbol{f}^2\left(\mathrm{W}^2\, \underbrace{\boldsymbol{f}^{\mathbf{1}}\left(\mathrm{W}^1\boldsymbol{x} + \mathbf{b}^{\mathbf{1}}\right)}_{z} + \boldsymbol{b}^2\right)}_{u} + \boldsymbol{b}^3\right)$
- $\boldsymbol{x} \epsilon R^n \quad \boldsymbol{z} \epsilon R^{m_1} \quad \boldsymbol{u}\boldsymbol{z} \epsilon R^{m_2} \quad \boldsymbol{y} \epsilon R^m$
- $i \in \{1, \cdots, n\}\, j \in \{1, \cdots, m_1\}\, k \in \{1, \cdots, m_2\}\, l \in \{1, \cdots, m\}$
- Activation functions:
- $f^{1,2,3} \in \{sign, \text{step}, \tanh, sigmoid, Relu, idenity \cdots\}$

Overall Functionality

(1) A hyper plane: $z_{in_j} = \mathrm{w}^1_{j1}\mathrm{x}_1 + \cdots + \mathrm{w}^1_{jn}\mathrm{x}_n + \mathrm{b}^1_j$

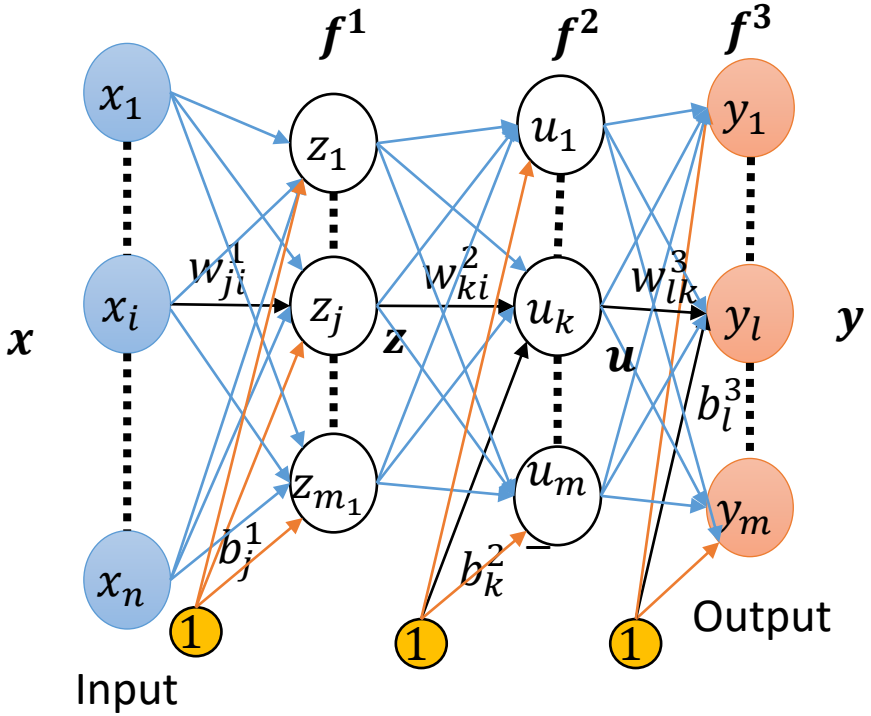
(2) Folding, Rectifying,.. on the hyper plane: $z_j = f^1(z_{in_j})$

(3) A hyper plane: $u_{in_k} = \mathrm{w}^2_{k1}\mathrm{z}_1 + \cdots + \mathrm{w}^2_{km_1}\mathrm{z}_{m_1} + \mathrm{b}^2_k$

(4) Folding, Rectifying,.. on the hyper plane: $u_k = f^2(y_{in_k})$

(5) A hyper plane: $y_{in_l} = \mathrm{w}^3_{l1}\mathrm{u}_1 + \cdots + \mathrm{w}^3_{lm_2}\mathrm{u}_2 + \mathrm{b}^3_l$

(6) Folding, Rectifying,.. on the hyper plane: $y_l = f^3(y_{in_l})$



A Desired Functionality in classification /Regression problems

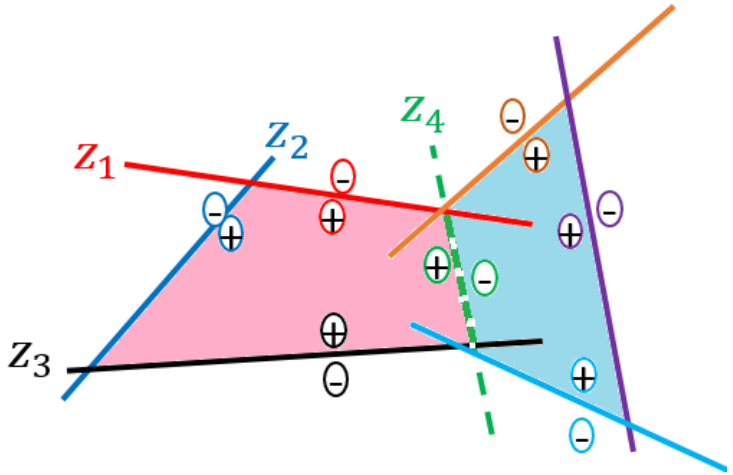
(1) A hyper plane: $z_{in_j} = w_{j1}^1 x_1 + \cdots + w_{jn}^1 x_n + b_j^1$

(2) Folding the hyper plane: $z_j = f^1(z_{in_j})$

(3) An "and" or "or logic gate for former folded hyper-planes is defined by "kth" neuron of second hidden layer 3, by which "kth" convex hyper-polygon will be resulted.

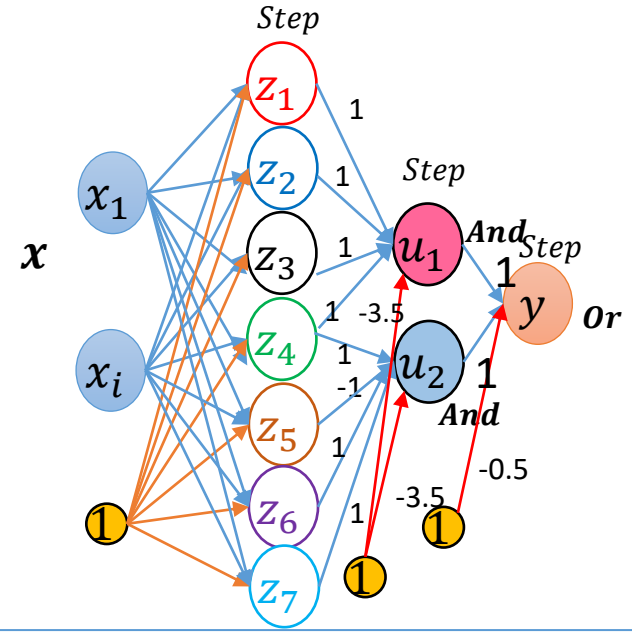
(4) An "and" or "or logic gate for former convex hyper-planes is defined by "lth" neuron of last layer, by which "lth" non-convex hyper-polygon will be resulted.

Example for $n = 2, m_1 = 7, m_2 = 2$ $m=1$



$$z_j = w_{j1}^1 x_1 + w_{j2}^1 x_2 + b_j^1, \quad \text{j=1,2,3,4,..,7}$$

$$u_k = w_{k1}^2 z_1 + w_{k2}^2 z_2 + b_k^2, \quad \text{k=1,2}$$

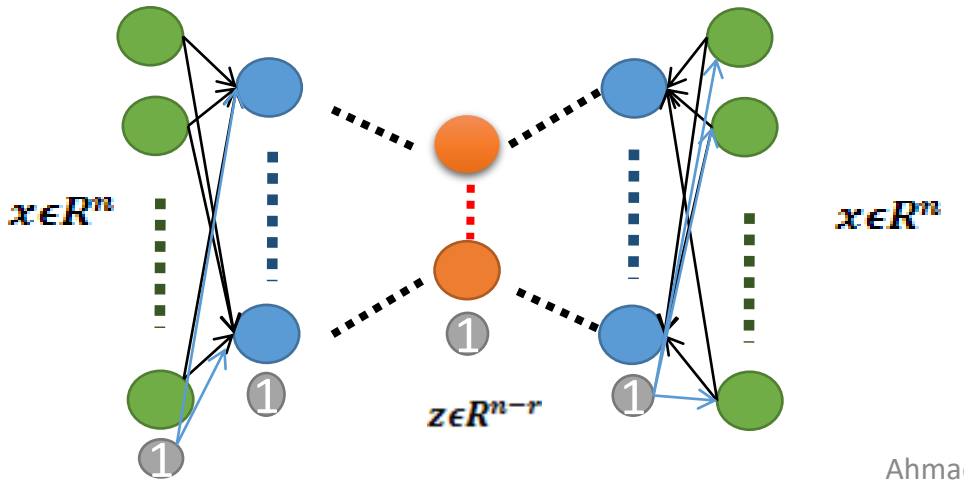


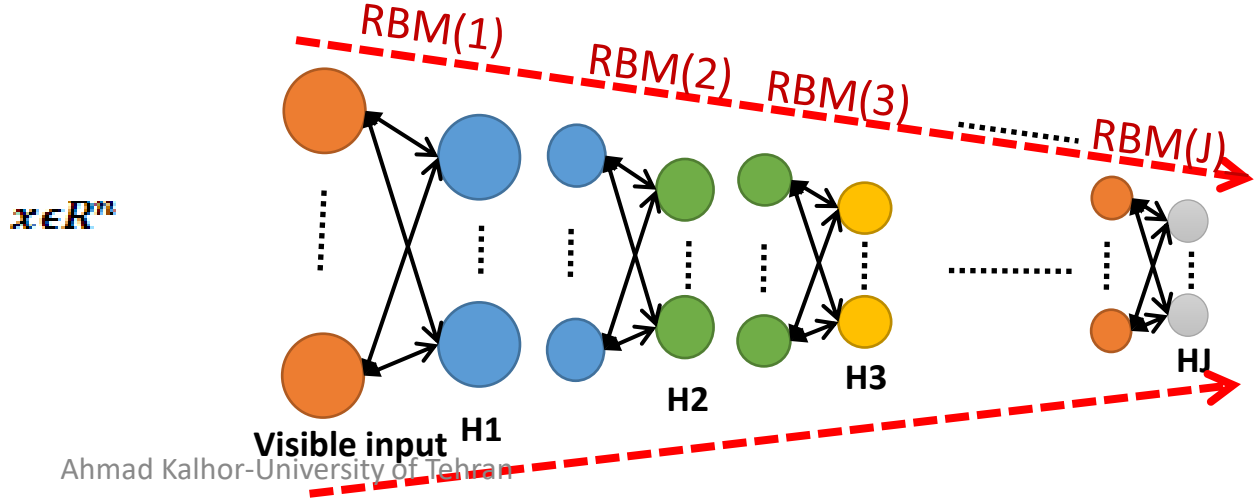
- A block of two FC layers can transform the input space to multi non-convex hyper- polygon partitions
- Using soft activation functions, a block of two FC layers can transform the input space to multi non-convex hyper- polygon volume maps

Some Notes about block of FC layers

1. A block of two or three FC layers are known as universal function approximator, through which any function by any desired accuracy can be approximated.
2. In comparison to a block of two FC layers, a block of three FC layers has better extrapolation (generalization) for unbounded regions and non-convex regions.

A block of FC layers with more than three layers are conventionally used in deep auto-encoders and cascaded restricted Boltzmann machines. In such blocks, each layer learns to encode or decode the taken data with a low change.





Some notes about FC layers

1. Fully connected layers are applied to a set of inputs reshaped as a vector; the spatial or temporal correlations among the inputs are not considered in its operation.
2. For high dimensional data, due to their massive wirings, they suffer from large memories, high computation load, and overfitting.
3. Although, they are appropriate for partitioning and mapping purposes, they are not ideal to remove disturbances, and filter and extract features from temporal, spatial , and multi modal signals.

1.1.2 Convolution Layers-Blocks-Modules*

Ideal to filter and encode/decode various, spatial, temporal and multi modal signals

- Simple Convolution
- Tiled Convolution
- Dilated Convolution
- Deconvolution (Transposed convolution)
- 1x1 Convolutions
- Flattened Convolutions
- Spatial and Cross-Channel convolutions
- Depth-wise Separable Convolutions
- Residual Blocks and types
- Grouped Convolutions
- Shuffled Grouped Convolutions

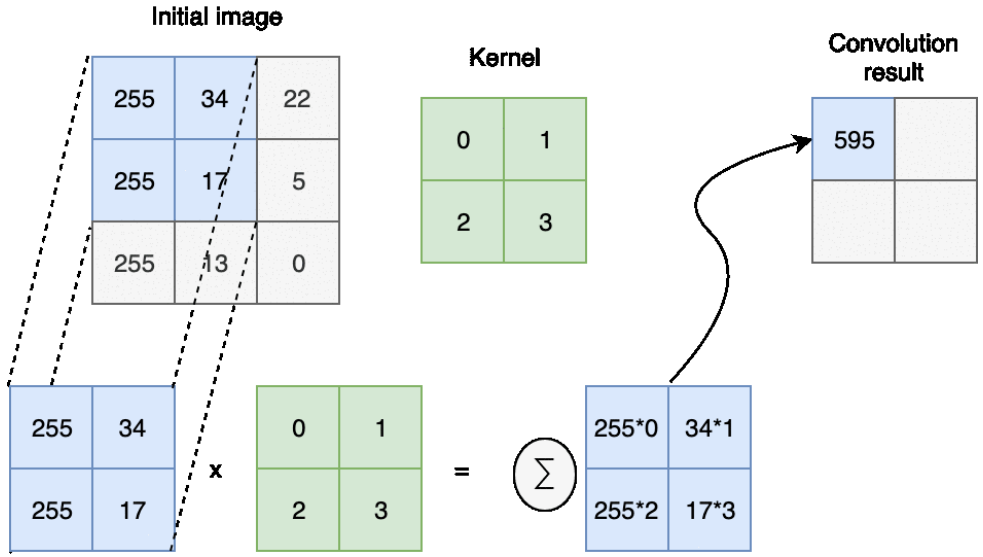
* Most of exampled are from https://ikhlestov.github.io/pages/machine-learning/convolutions-types/, Illarion Khlestov.

Simple Convolution

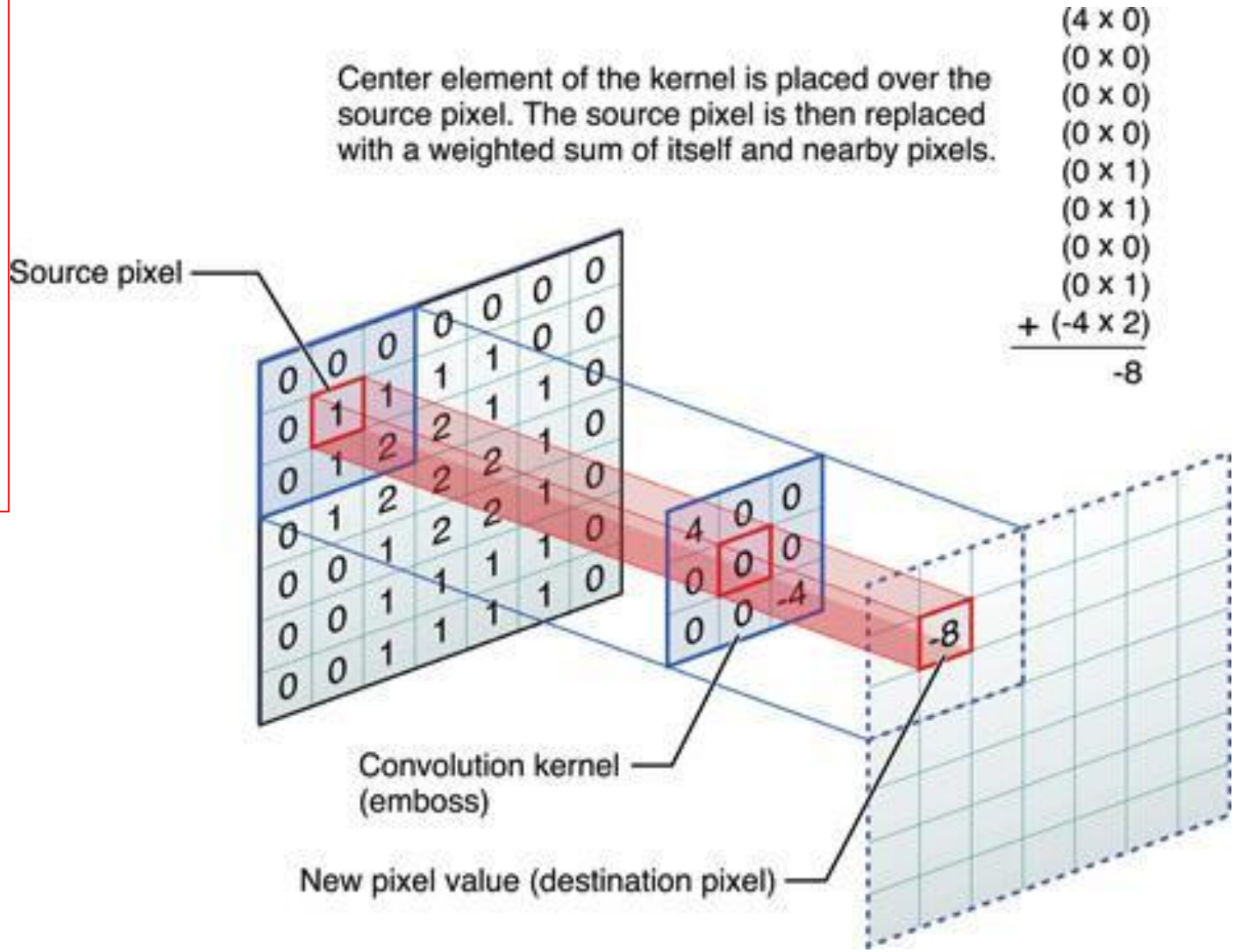
Take multiply dot products with same filter with some width/height shift.

Interesting because:

- Weights sharing and local connection
- it can capture and intensify all those patches which are adequately similar to the kernel and remove other ones by “relu”
- It can be interpreted as a filter that remove all patches which have weak linear correlation with the kernel



Example of convolution on two dimensional signal (image)

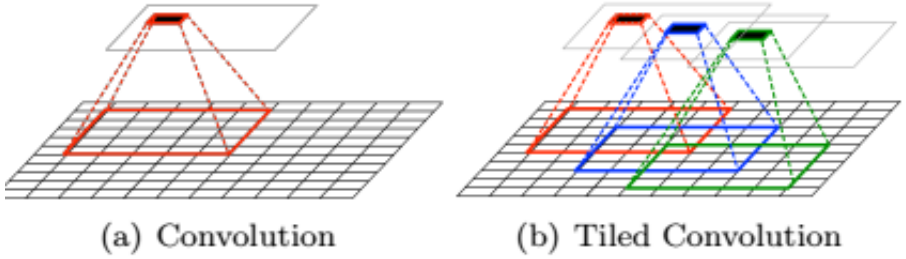


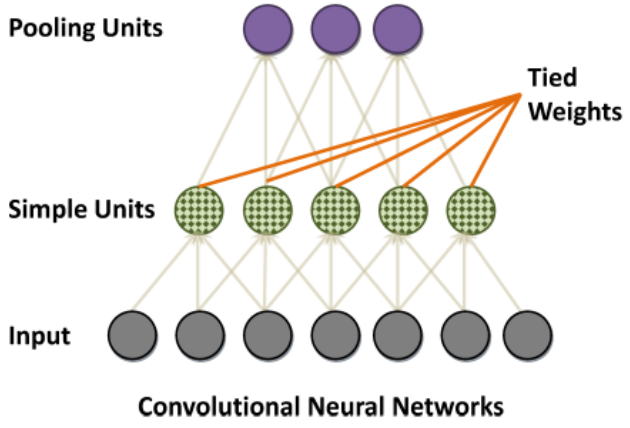
Tiled Convolution

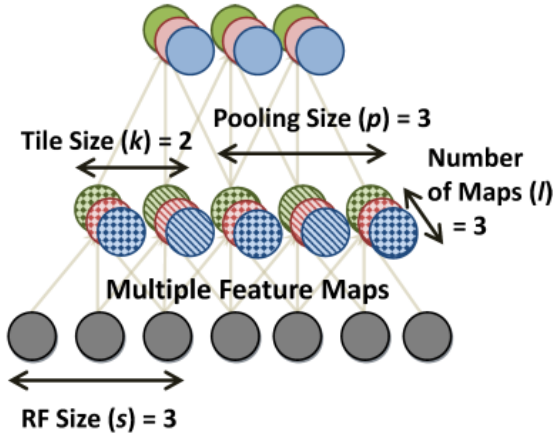
- It means that we can learn multiples feature maps rather than one feature map by considering several different filters
- In tiled convolution we can provide different kinds of invariance.
- Separate kernels are learned within the same layer

Example of Tiled convolution on one dimensional signal(time series)

Example of Tiled convolution on two dimensional signal







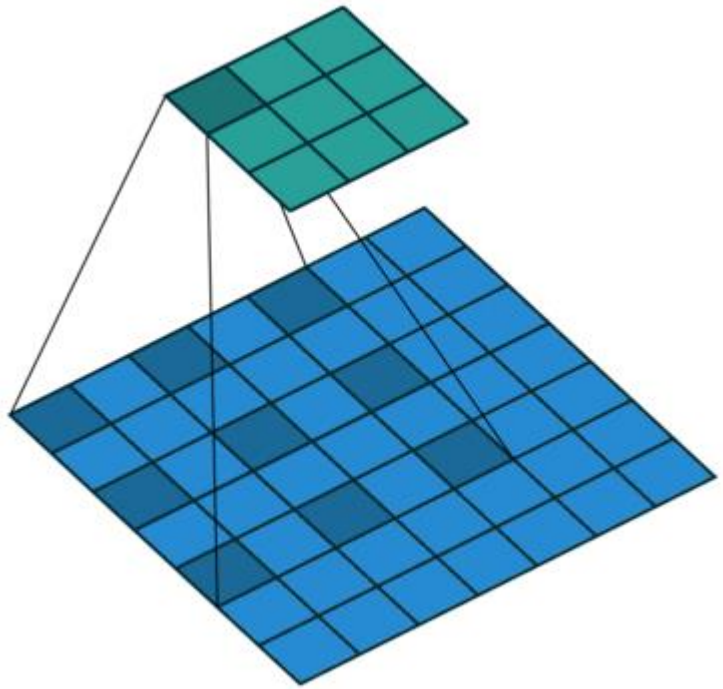
Some notes about the convolution layer

1. The convolution layer is a variant of the convolution operation used in LTI systems
2. A convolution layer actually transform an input (spatial, temporal, or multi modal) signal to several feature maps.
3. All units of feature maps just after convolution may be activated by applying an activation function like “relu”.
4. Using a filter, each feature map captures a certain feature from different local regions of the former signal.
5. Features, which are captured by filters, are actually frequently repeating sub-patterns within a signal.
6. The kernel size and the number of filters depend to the size and the number of the existing independent features, respectively.
7. Using stride=1 and padding, the size of a feature map is equal to the spatial size of the signal but using stride>1 (stride<1), the signal becomes down-sampled (up-sampled) by the convolution layer.
8. The resulting feature maps from a convolution layer, can be convolved again through a new convolution layer.
9. Through using a black of sequenced convolution layers (may come with activation functions, pooling and normalizing layers), actually redundancies and disturbances are removed and exclusive features for appropriate classification or regression problem will be appeared.

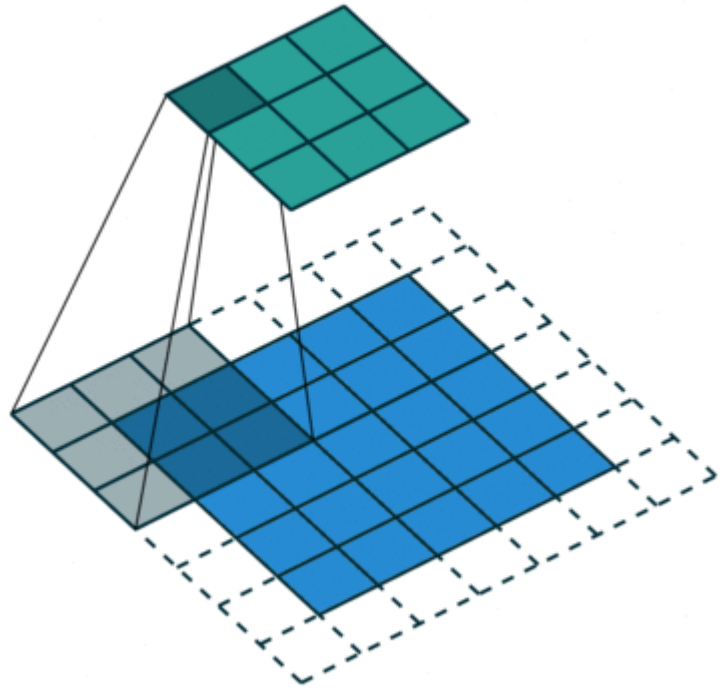
Dilated Convolution

To convolve and down-sample signals with very large spatial/temporal size

It is **a technique that expands the kernel (input) by inserting holes between the its consecutive elements**. In simpler terms, it is same as convolution but it involves pixel skipping, so as to cover a larger area of the input. ... In essence, normal convolution is just 1-dilated convolution



Dilated Convolution (l=2)

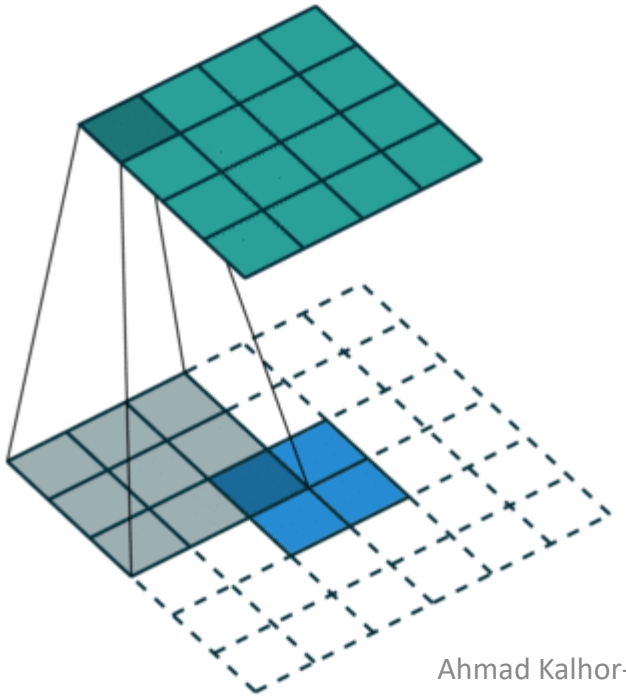


Standard Convolution (l=1)

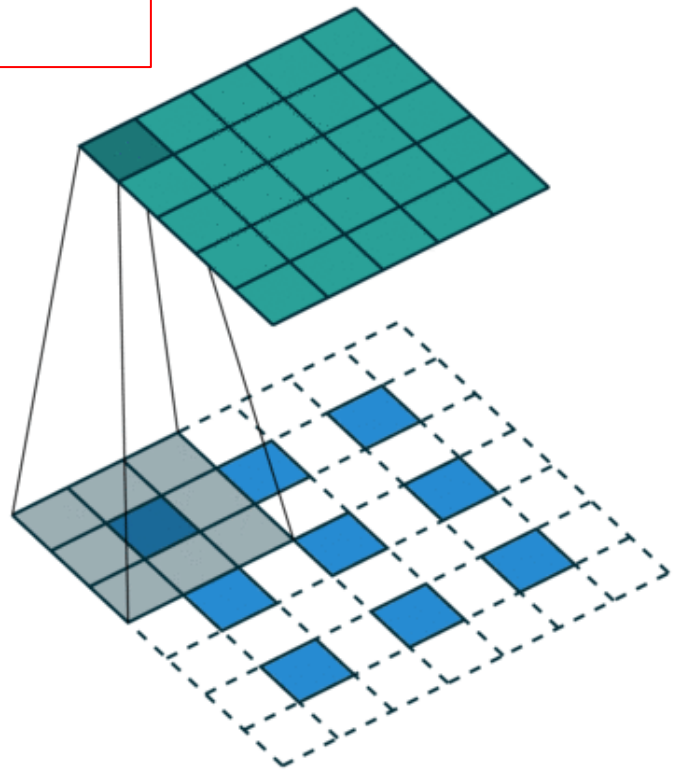
Deconvolution (Transposed Convolution)

Transposed convolution can be seen as the backward pass of a corresponding traditional convolution.
Transpose Convolution is a convolution layer which reverses the operation done by the corresponding convolution.
High pass Filter (corresponding Conv.)→ Low pass Filter (Tran. Conv.)
Smooth Filter (corresponding Conv.) → Difference Filter (Tran. Conv.)

Visually, for a transposed convolution with stride one and no padding, we just pad the original input (blue entries) with zeroes (white entries)).



In case of stride two and padding, the transposed convolution would look like this

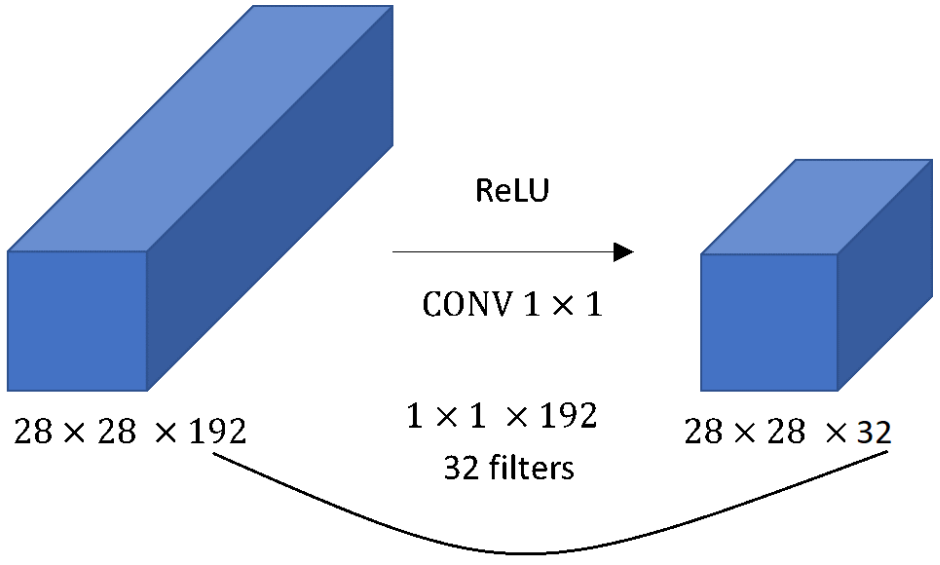


1x1 Convolutions

Initially 1x1 convolutions were proposed at Network-in-network(NiN). After they were highly used in GoogleNet architecture. Main features of such layers:

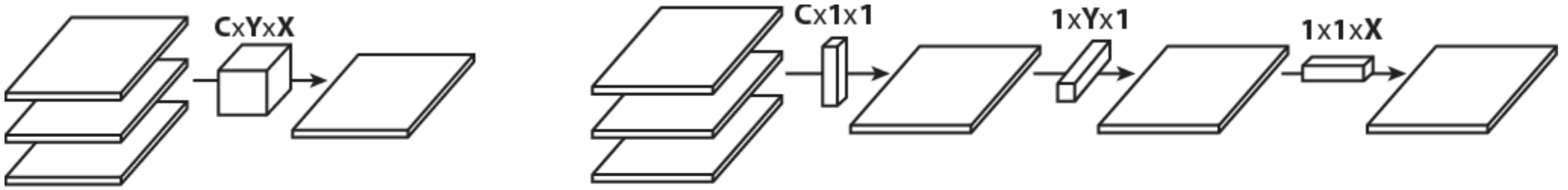
- Reduce or increase dimensionality
- Apply nonlinearity again after convolution
- Can be considered as “feature pooling”

They are used in such way: we have image with size 28x28x192, where 192 means features, and after applying 32 1x1 convolutions filters we will get images with 28x28x32 dimensions.



Flattened Convolutions

Were published in [Flattened Convolutional Neural Networks for Feedforward Acceleration](). Reason of usage same as 1x1 convs from NiN networks, but now not only features dimension set to 1, but also one of another dimensions: width or height.



(a) 3D convolution

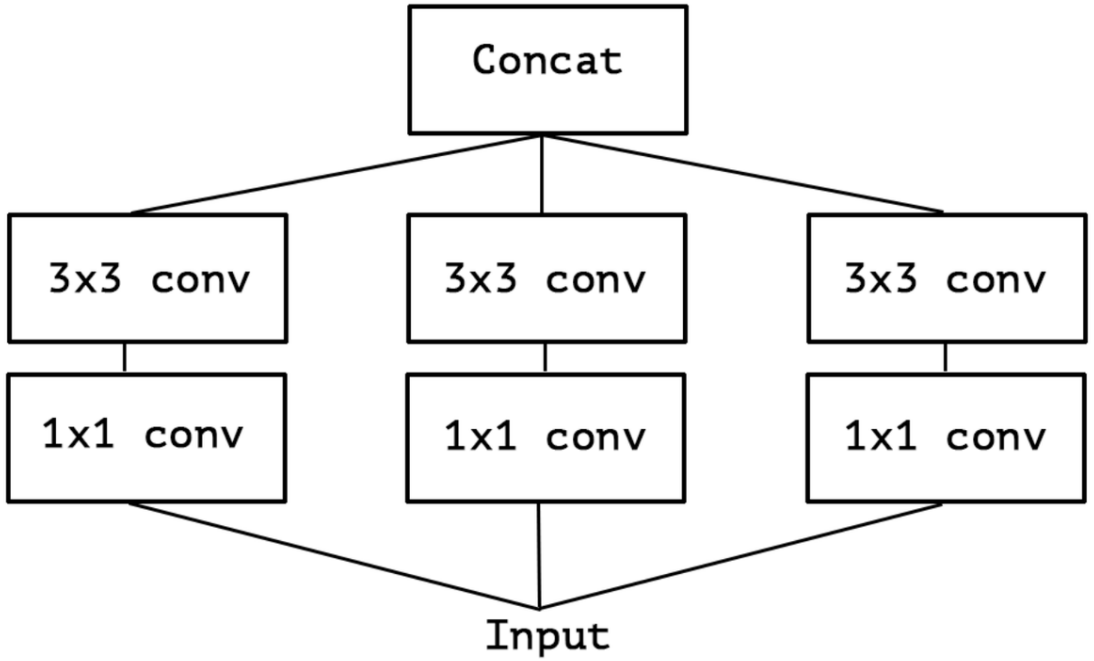
(b) 1D convolutions over different directions

The number of operations at each feature map is= a*a*(C*Y*X)

The number of operations at each feature map is= a*a*(C+Y+X)

Spatial and Cross-Channel convolutions

First this approach was widely used in **Inception network**. Main reason is to split operations for cross-channel correlations and at spatial correlations into a series of independently operations. Spatial convolutions means convolutions performed in **spatial dimensions** - **width** and **height**



- Grouping convolutions in independent parallel paths cause acceleration in computations and reduction in the required memory

Inception Module

Inception module is introduced by Szegedy *et al*. * which can be seen as a logical culmination of NIN. They use variable filter sizes to capture different visual patterns of different sizes, and approximate the optimal sparse structure by the inception modul

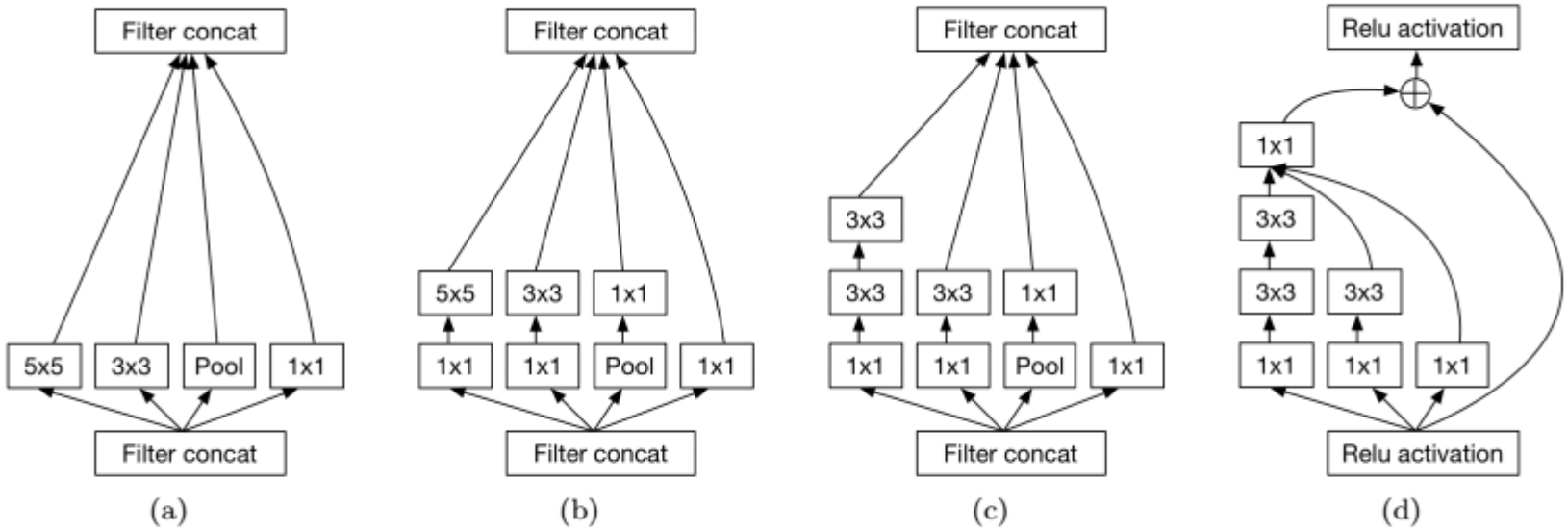


Figure 5: (a) Inception module, naive version. (b) The inception module used in [10]. (c) The improved inception module used in [41] where each 5×5 convolution is replaced by two 3×3 convolutions. (d) The Inception-ResNet-A module used in [42].

❖ Using parallel convolution paths with different kernel sizes increases the chance of better feature extraction

* C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015,
pp. 1–9.

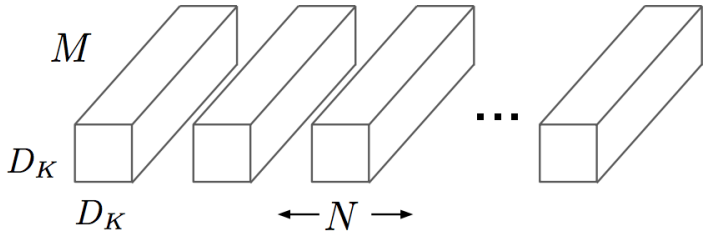
Depthwise Separable Convolutions

A lot about such convolutions published in the (Xception paper) or (MobileNet paper).
Consist of:

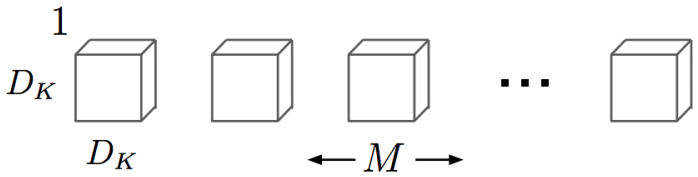
- **Depthwise convolution**, i.e. a spatial convolution performed independently over each channel of an input.
- **Pointwise convolution**, i.e. a 1x1 convolution, projecting the channels output by the depthwise convolution onto a new channel space.

Difference between Inception module and separable convolutions:

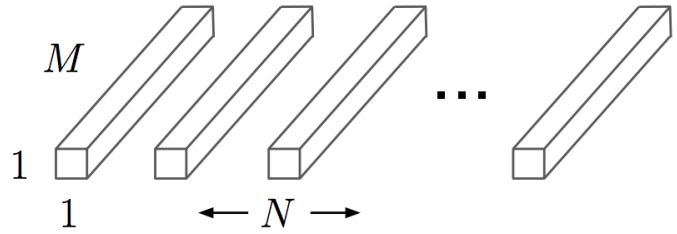
- Separable convolutions perform first channel-wise spatial convolution and then perform 1x1 convolution, whereas Inception performs the 1x1 convolution first.
- depthwise separable convolutions are usually implemented without non-linearities



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



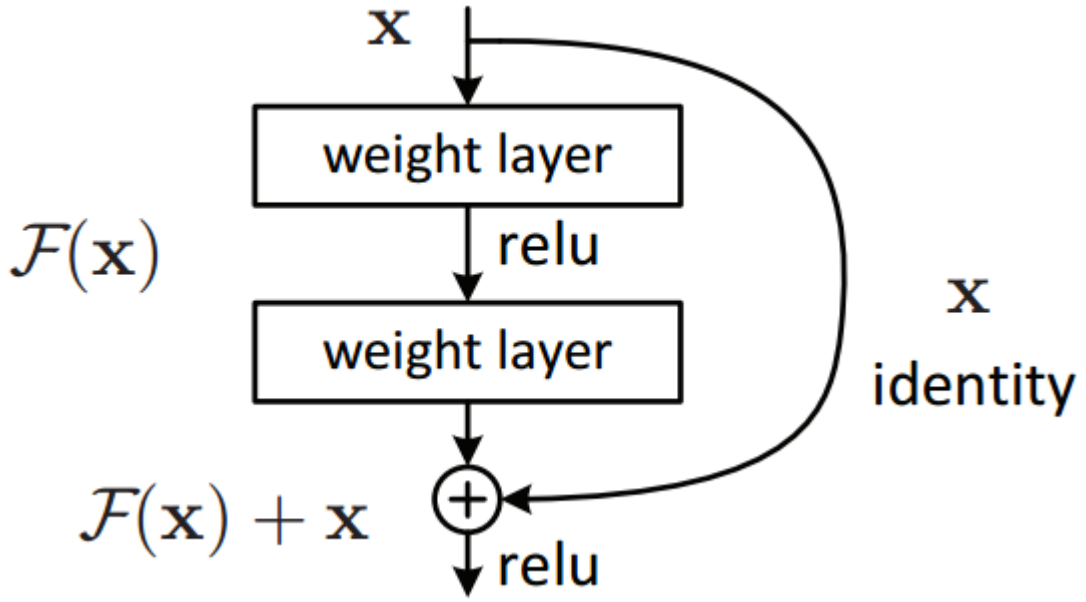
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

❖ Using Depthwise convolution decreases the computation load of 3D-convolution

Residual Blocks

To avoid missing information and provide a solution for vanishing gradient

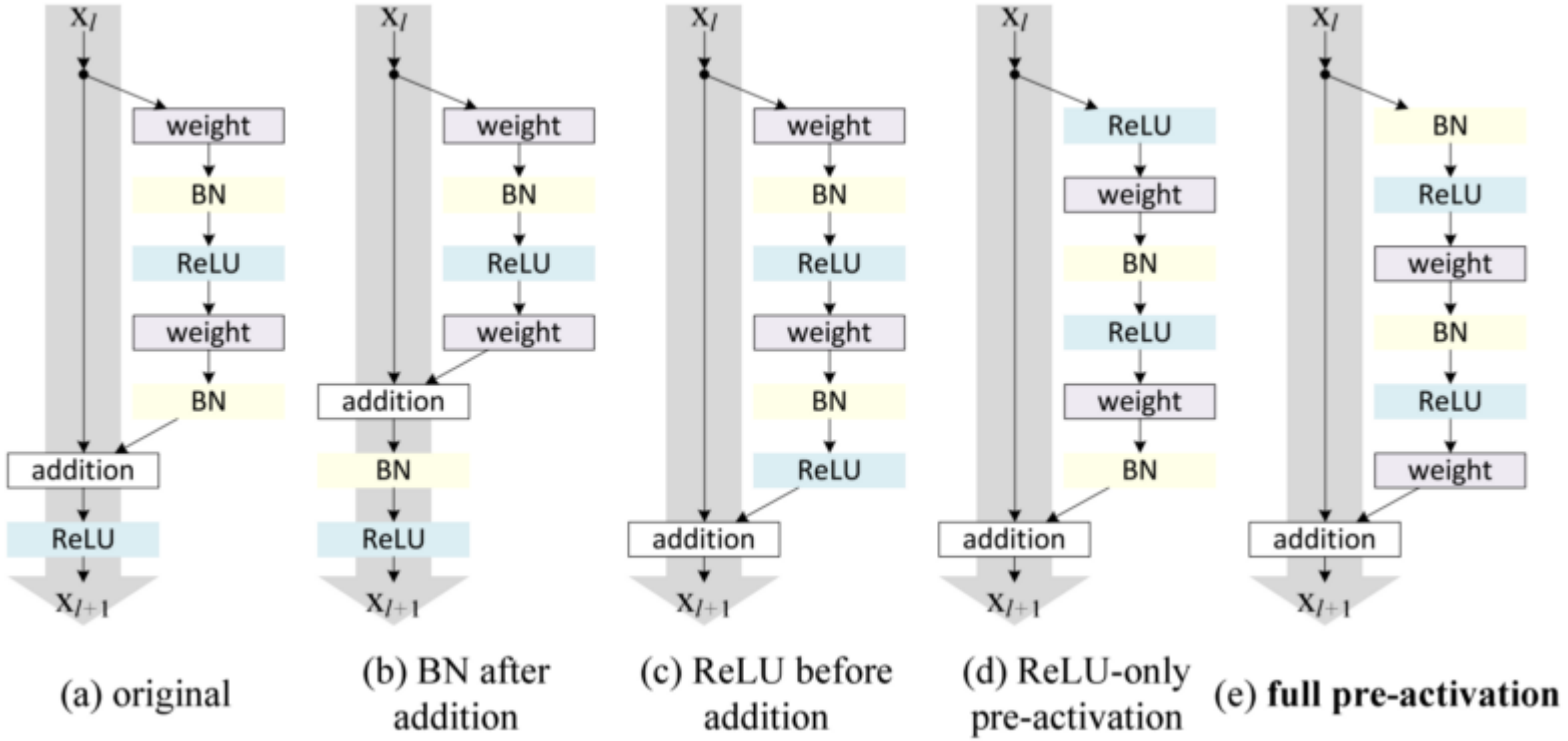
In traditional neural networks, each layer feeds into the next layer. In a network with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away. That's it. But understanding the intuition behind why it was required in the first place, why it is so important, and how similar it looks to some other state-of-the-art architectures is where we are going to focus on. There is more than one interpretation of why residual blocks are awesome and how & why they are one of the key ideas that can make a neural network show state-of-the-art performances on a wide range of tasks.



❖ Applying an identity path to convolution, residual block causes that features to be captured in a difference evolving learning manner without missing information.

Different forms of residual blocks

The image below shows how to arrange the residual block and identity connections for the optimal gradient flow. It has been observed that pre-activations with batch normalizations generally give the best results (i.e., the right-most residual block in the image below gives the most promising results).



Grouped Convolutions

Grouped convolutions were initial mentioned in AlexNet, and later reused in ResNeXt. Main motivation of such convolutions is to reduce computational complexity while dividing features on groups.

The image below shows multiple interpretations of a residual block.

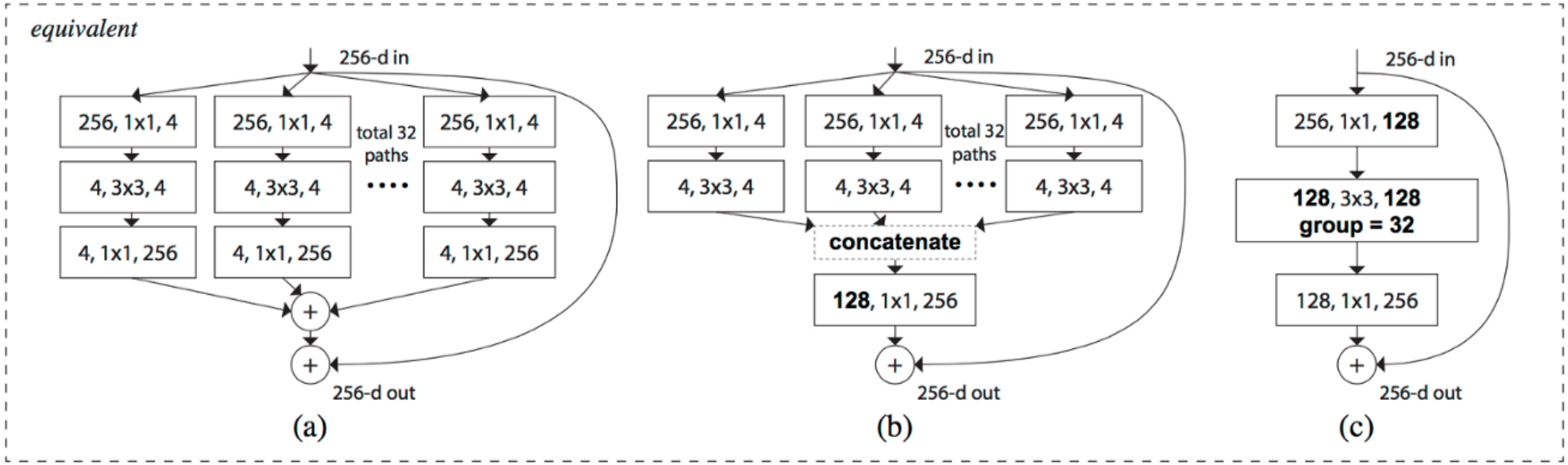


Figure 3. Equivalent building blocks of ResNeXt. **(a)**: Aggregated residual transformations, the same as Fig. 1 right. **(b)**: A block equivalent to (a), implemented as early concatenation. **(c)**: A block equivalent to (a,b), implemented as grouped convolutions [24]. Notations in **bold** text highlight the reformulation changes. A layer is denoted as (# input channels, filter size, # output channels).

Shuffled Grouped Convolutions

Shuffle Net proposed how to eliminate main side effect of the grouped convolutions that “outputs from a certain channel are only derived from a small fraction of input channels”.
They proposed shuffle channels in such way(layer with gg groups whose output has g×ng×n channels):

- reshape the output channel dimension into (g,n)(g,n)
- transpose output
- flatten output bac

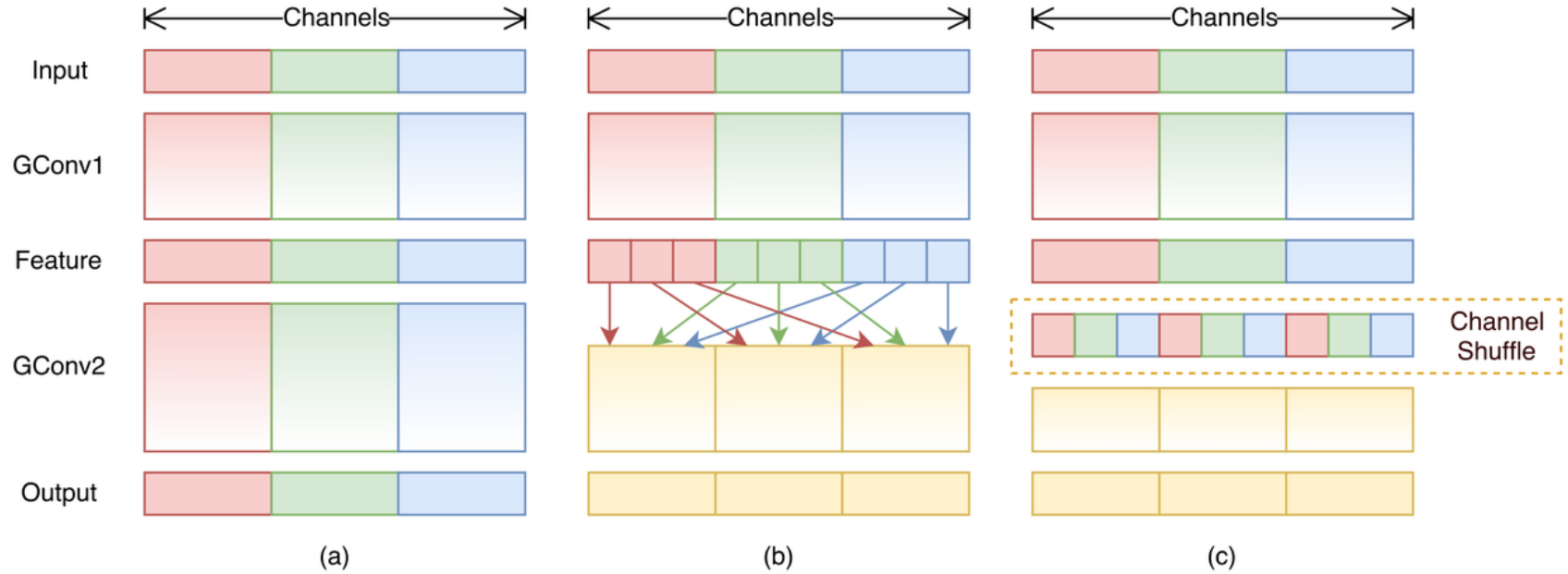


Figure 1: Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.

Some notes about convolution layers

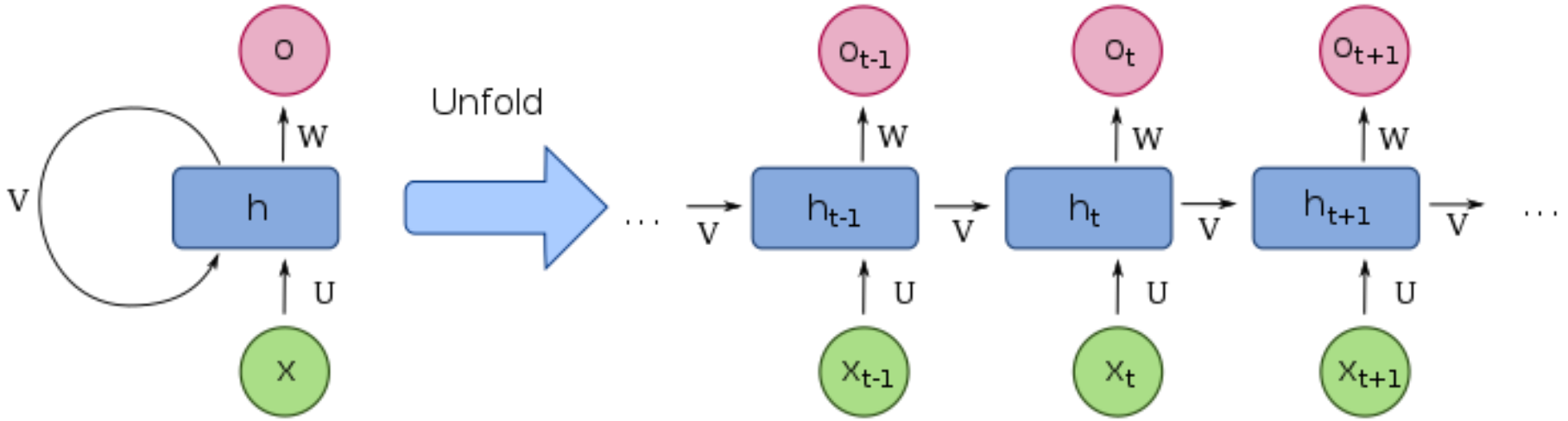
1. There are many extensions of simple convolutional layers.
2. Deconvolution layers are introduced as the backward pass of their corresponding convolution.
3. Residual blocks use a direct skip connection from inputs to outputs(as an identity function) to avoid missing information and to solve the problem of vanishing gradient. In addition, many other extended convolution layers use such skip connection in their structures.
4. Some convolutional layers such as "NiN" and flattened convolutions provide a considerable drop in computations by using point-wise and depth-wise convolution instead of using simple 3D convolution operations.
5. Some convolutional layers such as inception module by using some parallel convolutions with different resolutions have tried to provide better accuracy.
6. Grouped convolutions as well as shuffled grouped convolutions use several similar form and parallel convolution blocks (with an identity skip connection) in their structures. Each convolution block includes point-wise and depth-wise convolutions.

1.1.3 Recurrent Neural Networks

Ideal to retrieve short/long dependencies among the sequenced samples of a signals

- Simple RNN Module
- LSTM Module
- GRU Module
- Bidirectional RNN layer
- Bidirectional Deep RNNs
- TimeDistributed layer
- ConvLSTM layer

Simple RNN Module



$$\boldsymbol{h}_t = \mathbf{f}(\mathrm{V}\boldsymbol{h}_{t-1} + \mathrm{U}\boldsymbol{x}_t)$$

$$\boldsymbol{o}_t = \mathbf{g}(\mathrm{W}\boldsymbol{h}_t)$$

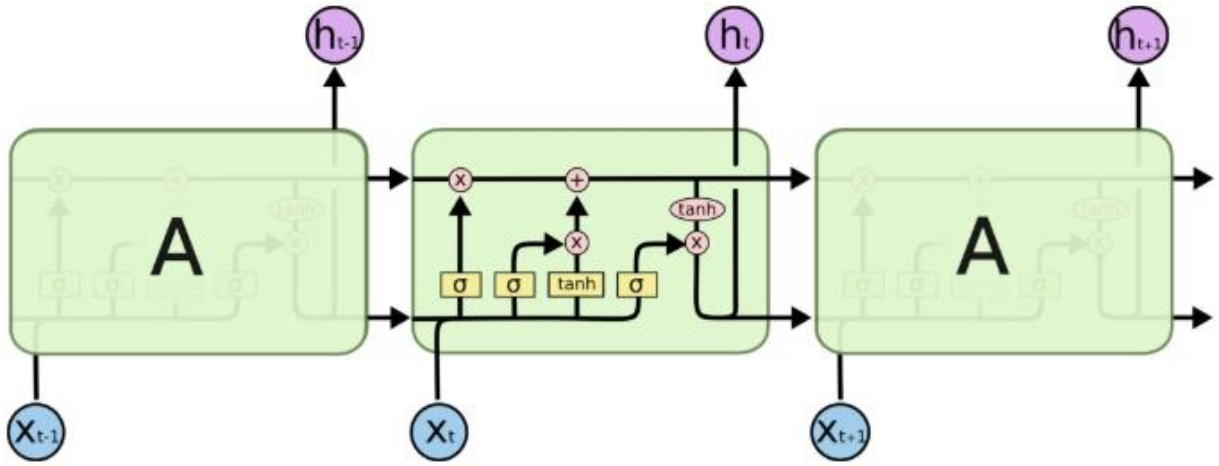
$$\boldsymbol{o}_t = \mathbf{g}\Big(\mathrm{W}\mathbf{f}\big(\mathrm{V}\mathbf{f}\big(\mathrm{V}\ldots(\mathrm{V}\mathbf{f}(\mathrm{V}\boldsymbol{h}_0 + \mathrm{U}\boldsymbol{x}_1)\ldots + \mathrm{U}\boldsymbol{x}_{t-1}\big)\big) + \mathrm{U}\boldsymbol{x}_t\big)\Big), t = 1,2,\cdots$$

Some notes

1. RNN actually make a set of nonlinear first order difference equations.
2. The current state $\boldsymbol{h}_t$ is affected by both exogenous input $\boldsymbol{x}_t$ and the former state $\boldsymbol{h}_{t-1}$.
3. The output $\boldsymbol{o}_t$ is a function of hidden state $\boldsymbol{h}_t$ at each time t .
4. Such equations can provide a memory from the short dependencies in a sequenced patterns
5. $Activation\ functions$: $f, g \epsilon \{Relu, tanh, sigm, sign, identity \ldots.\}$

LSTM (long short term memory)

Hochreitor & Shmidhuber 1997



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

$$\hat{y}_t = f(V.h_t + b_v)$$

Some notes:

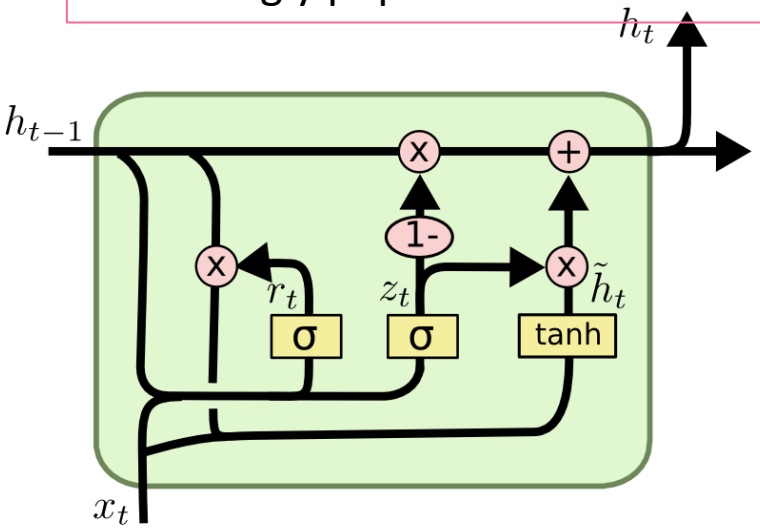
1. LSTM is a module of f*our interacted layers*.
2. A cell state is a key concept in a LSTM proving (packaging and carrying) all informative data required to save long/short dependencies among the sequenced patterns.
3. Using exogenous input and the hidden state, cell state interacts by three independent gates: forget, write, and read gates.
4. Some information on the cell state is forgotten by the forget gate.
5. Using exogenous input and the hidden state, a package of informative data is added to the cell state by write gate.
6. The hidden state is provided from the cell state by read gate.
7. The output is a function of the hidden state.

GRU (Gated Recurrent Unit)

Cho, et al. (2014)

Many extensions of the LSTM have been introduced. A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU.

GRU combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



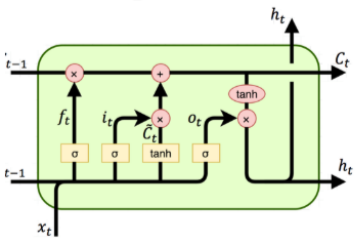
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

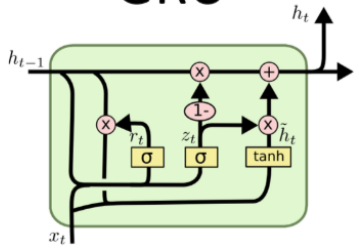
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM



GRU

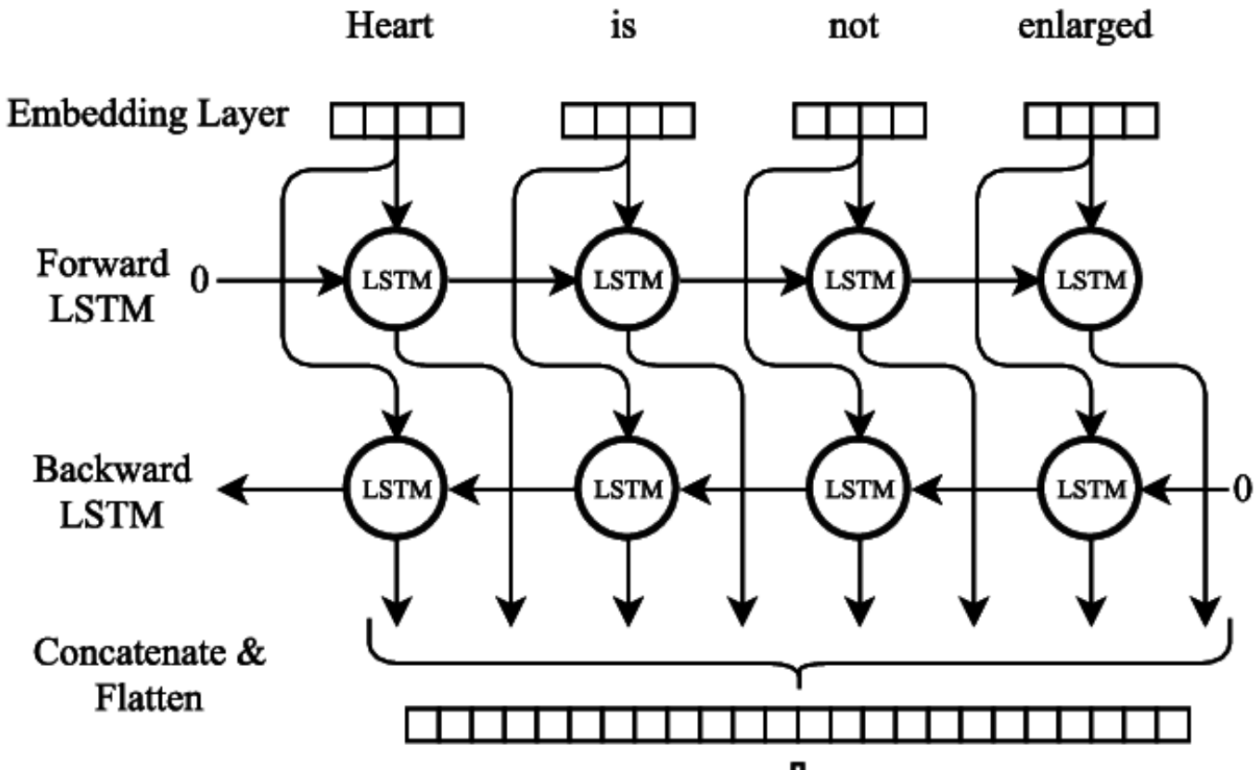


Bidirectional RNNs

A **Bidirectional LSTM**, or **biLSTM**, is a sequence processing model that consists of two LSTMs:

One taking the input in a forward direction, and the other in a backwards direction.
BiLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm (e.g. knowing what words immediately follow *and* precede a word in a sentence).

Image Source: Modelling Radiological Language with Bidirectional Long Short-Term Memory Networks, Cornegruta et al.

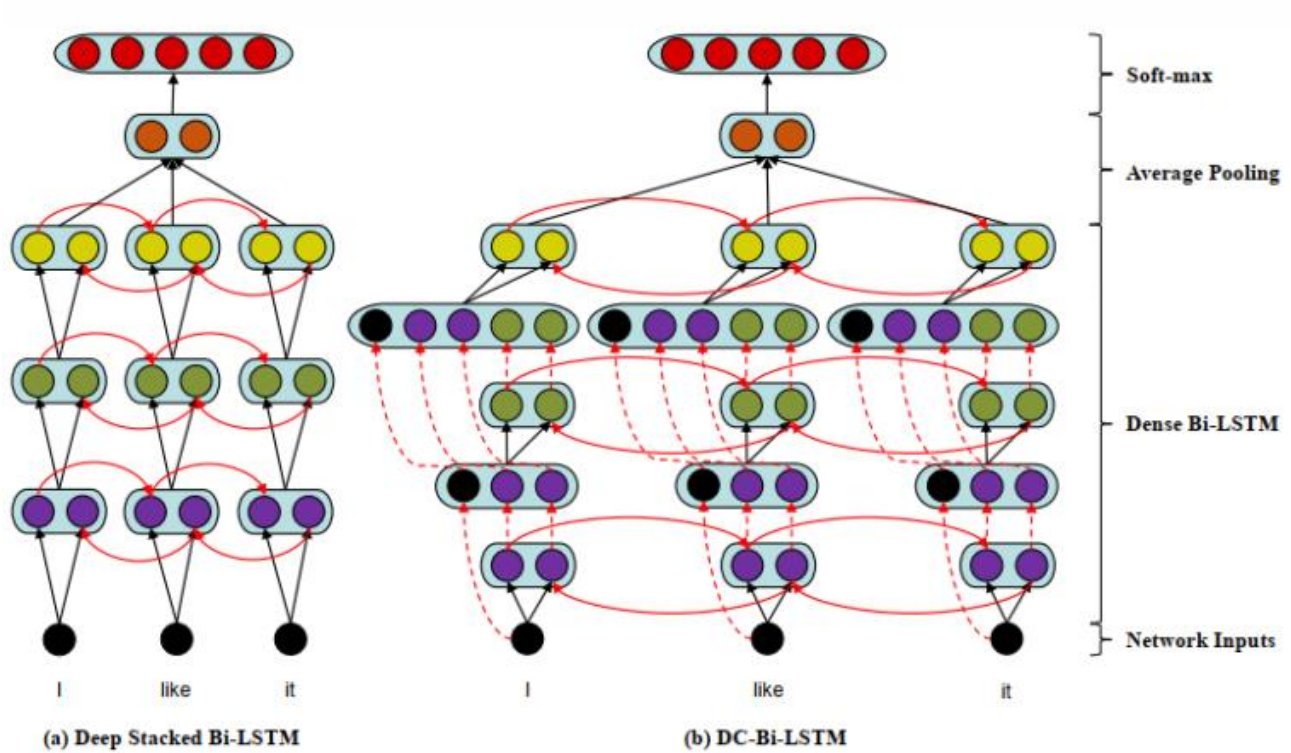


It can be used in categorization
Missed words prediction and so on.

Deep (Bidirectional) RNNs

A block of one or more than one LSTM or Bi-LSTM layers

- **Deep (Bidirectional) RNNs** are similar to Bidirectional RNNs, only that we now have multiple layers per time step. In practice this gives us a higher learning capacity (but we also need a lot of training data)



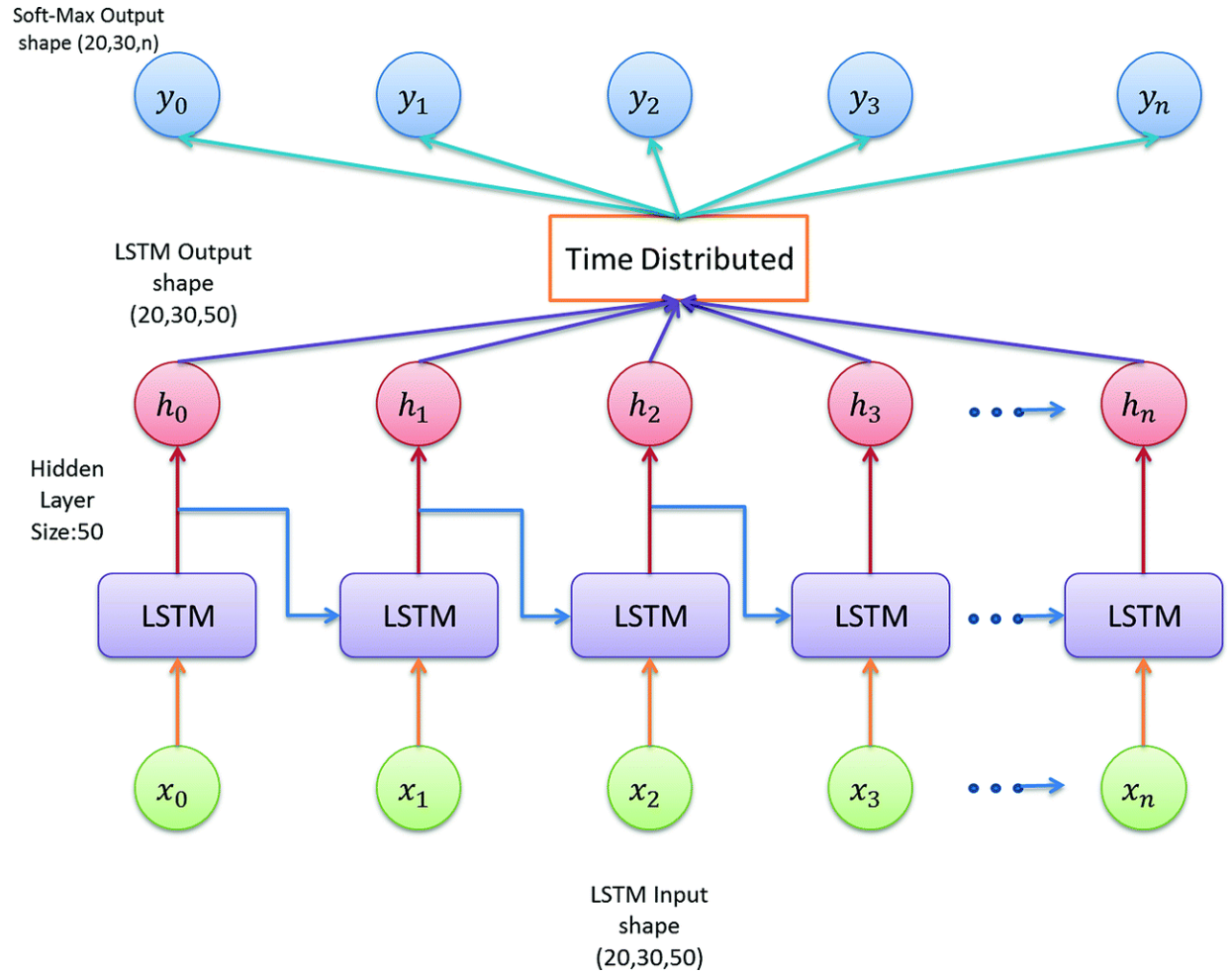
Deep Stacked Bi-LSTM

Densely Connected Bidirectional

Time distributed Layer

to reduce the required memory in processing multi sequenced samples(images) to predict the target

Time Distributed layer is very useful to work with time series data or video frames.
It allows **to use a layer for each input**. That means that instead of having several input "models", we can use "one model" applied to each input. Then GRU or LSTM can help to manage the data in "time".



Time step: **the number of past samples of a sequence which are used in a LSTM to predict the targets.**

ConvLSTM

Xingjian Shi Zhourong, et al (2015)

- To provide memory with sequential images, one approach is using **ConvLSTM** layers.
- **ConvLSTM** is a Recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations. As a result, the data that flows through the ConvLSTM cells keeps the input dimension (3D in our case) instead of being just a 1D vector with features.
- Using shared weights, convolution provide more appropriate processing and less computations in comparison to matrix product.

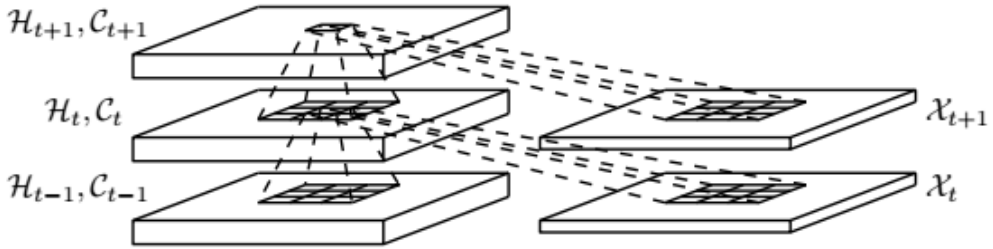
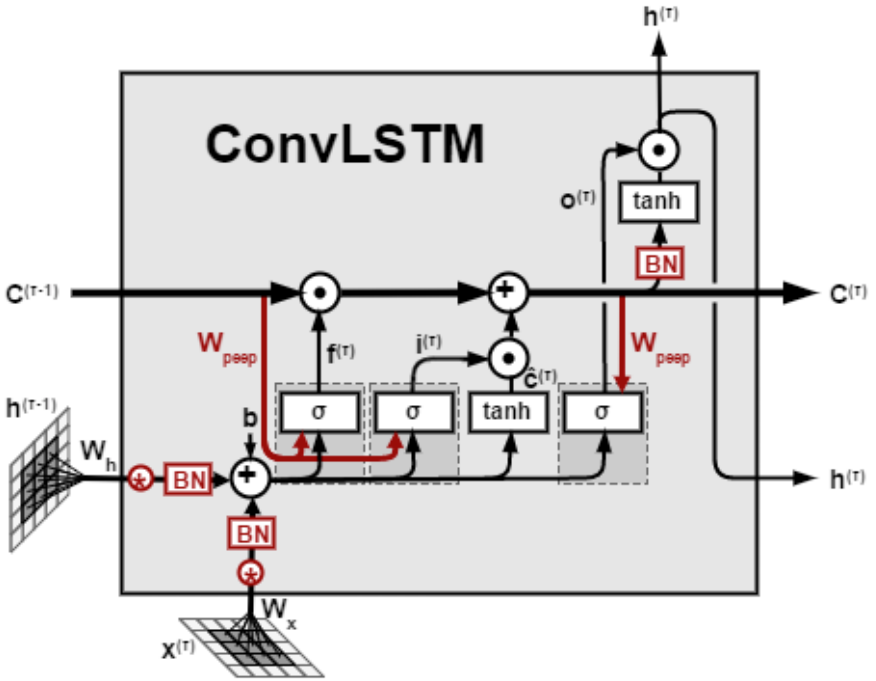


Figure 2: Inner structure of ConvLSTM

$$i_t = \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f)$$
$$\mathcal{C}_t = f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o)$$
$$\mathcal{H}_t = o_t \circ \tanh(\mathcal{C}_t)$$



where '$*$' denotes the convolution operator and '$\circ$', as before, denotes the Hadamard product (**element-wise product**)

ConvLSTM 2D,3D

1.1.4 Attention Layers-Modules

An improving mechanism for RNNs by applying an interface connecting the encoder and decoder

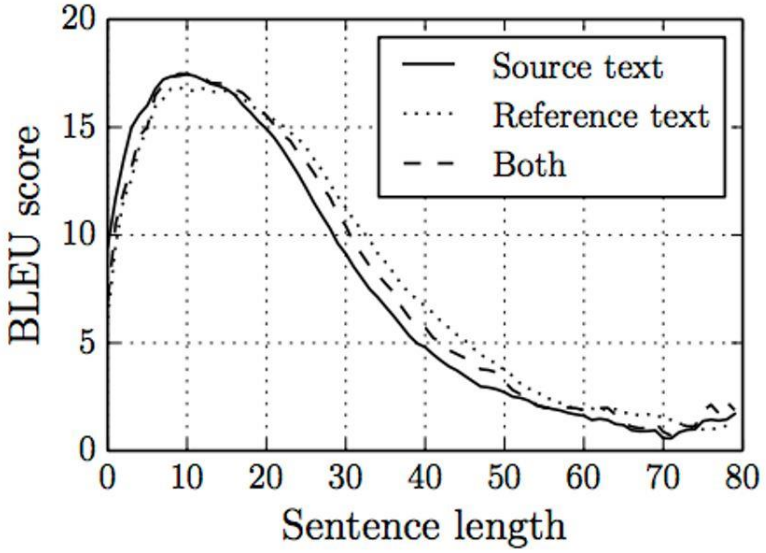
- In translation machines, LSTM/GRU as a seq2seq model can not predict successful translation for a sentence, when its length increases.

What is attention layer in Deep Learning?

- Every RNN/LSTM can be interpreted as an Encoder and Decoder.
  - Encoder encodes the input samples of a chronological signal to hidden state.
  - Decoder decodes the hidden state to the output.

Attention is an interface connecting the encoder and decoder that provides the decoder with information from every encoder hidden state.

With this framework, the model is able to selectively focus on valuable parts of the input sequence and hence, learn the association between them.



BLEU:
Bilingual Evaluation Understudy

*Neural Machine Translation by Jointly Learning to Align and Translate

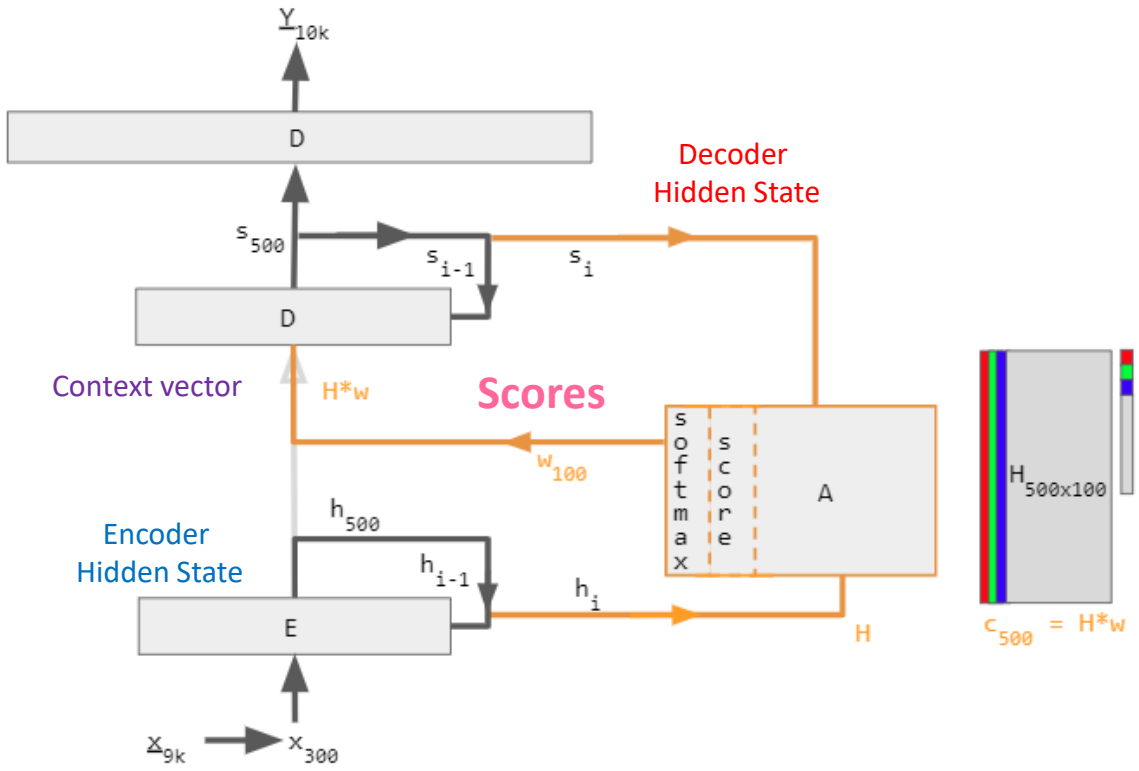
Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio

A language translation example

To build a machine that translates English-to-French (see the shown diagram), one starts with an Encoder-Decoder and grafts an attention unit to it. In the simplest case such as the shown example, the attention unit is just lots of dot products of recurrent layer states and does not need training. In practice, the attention unit consists of 3 fully connected neural network layers that needs to be trained. The 3 layers are called Query, Key, and Value.

Encoder-Decoder with attention. This diagram uses specific values to relieve an already cluttered notation alphabet soup. The left part (in black) is the Encoder-Decoder, the middle part (in orange) is the attention unit, and the right part (in grey & colors) is the computed data. Grey regions in H matrix and w vector are zero values.

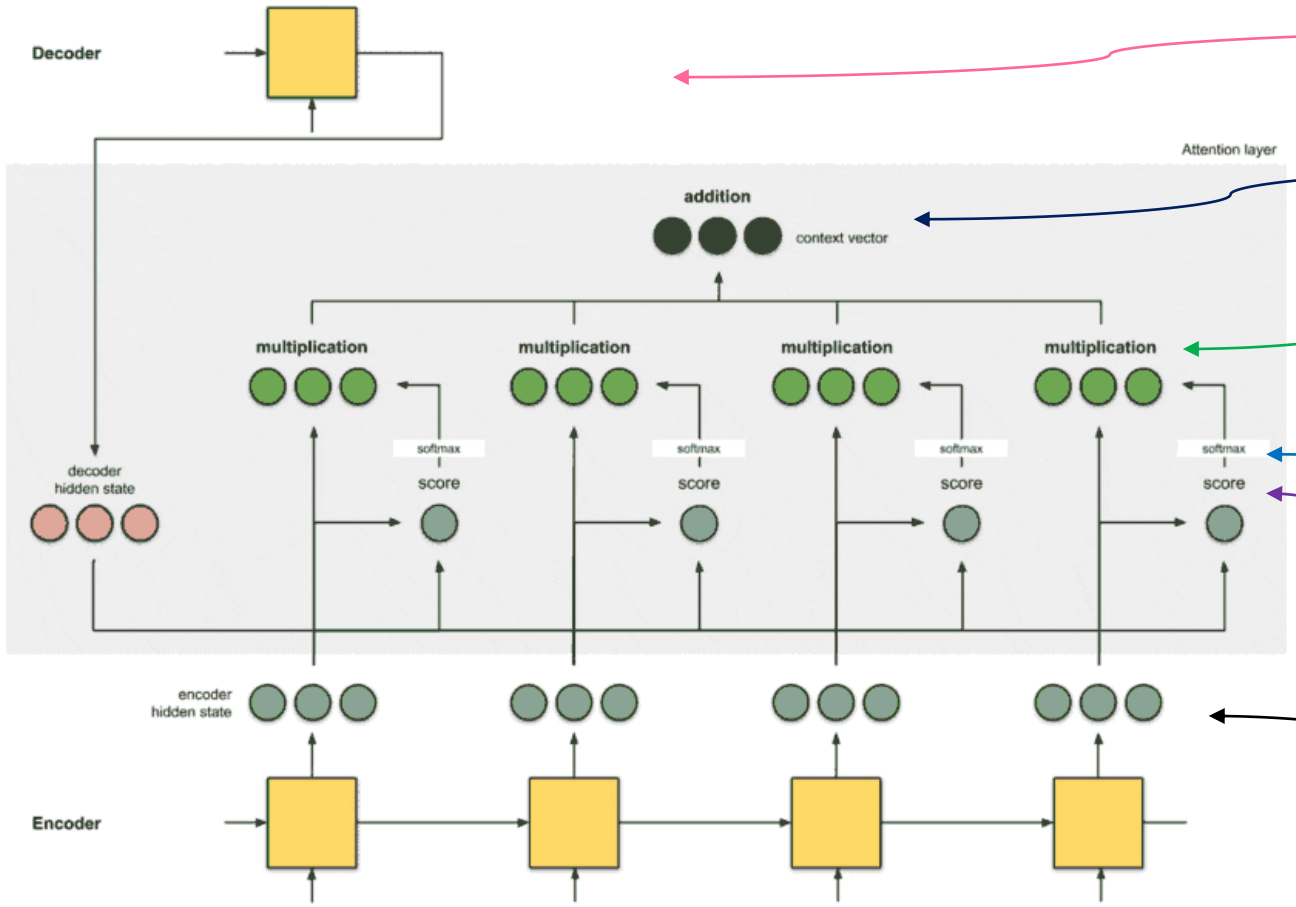
Numerical subscripts are examples of vector sizes. Lettered subscripts i and i-1 indicate time step.



label	description
100	max sentence length
300	embedding size (word dimension)
500	length of hidden vector
9k, 10k	dictionary size of input & output languages respectively.
x, Y	9k and 10k 1-hot dictionary vectors. x → x implemented as a lookup table rather than vector multiplication. Y is the 1-hot maximizer of the linear Decoder layer D; that is, it takes the argmax of D's linear layer output.
x	300-long word embedding vector. The vectors are usually pre-calculated from other projects such as GloVe or Word2Vec.
h	500-long encoder hidden vector. At each point in time, this vector summarizes all the preceding words before it. The final h can be viewed as a "sentence" vector, or a thought vector as Hinton calls it.
s	500-long decoder hidden state vector.
E	500 neuron RNN encoder. 500 outputs. Input count is 800–300 from source embedding + 500 from recurrent connections. The encoder feeds directly into the decoder only to initialize it, but not thereafter; hence, that direct connection is shown very faintly.
D	2-layer decoder. The recurrent layer has 500 neurons and the fully connected linear layer has 10k neurons (the size of the target vocabulary).[7] The linear layer alone has 5 million (500 * 10k) weights -- ~10 times more weights than the recurrent layer.
score	100-long alignment score
w	100-long vector attention weight. These are "soft" weights which changes during the forward pass, in contrast to "hard" neuronal weights that change during the learning phase.
A	Attention module — this can be a dot product of recurrent states, or the Query-Key-Value fully connected layers. The output is a 100-long vector w.
H	500x100. 100 hidden vectors h concatenated into a matrix
c	500-long context vector = H * w. c is a linear combination of h vectors weighted by w.

Unfolded attention layer

The below figure demonstrates an Encoder-Decoder architecture with an attention layer (All time steps).



5-Feed the context vector into the decoder.

4-Sum the alignment vectors (as context vector)

3- Multiply each encoder hidden state by its softmax score.

2-Run all the scores through a softmax layer

1-Obtain a score for every encoder hidden state.

0-Prepare hidden states.

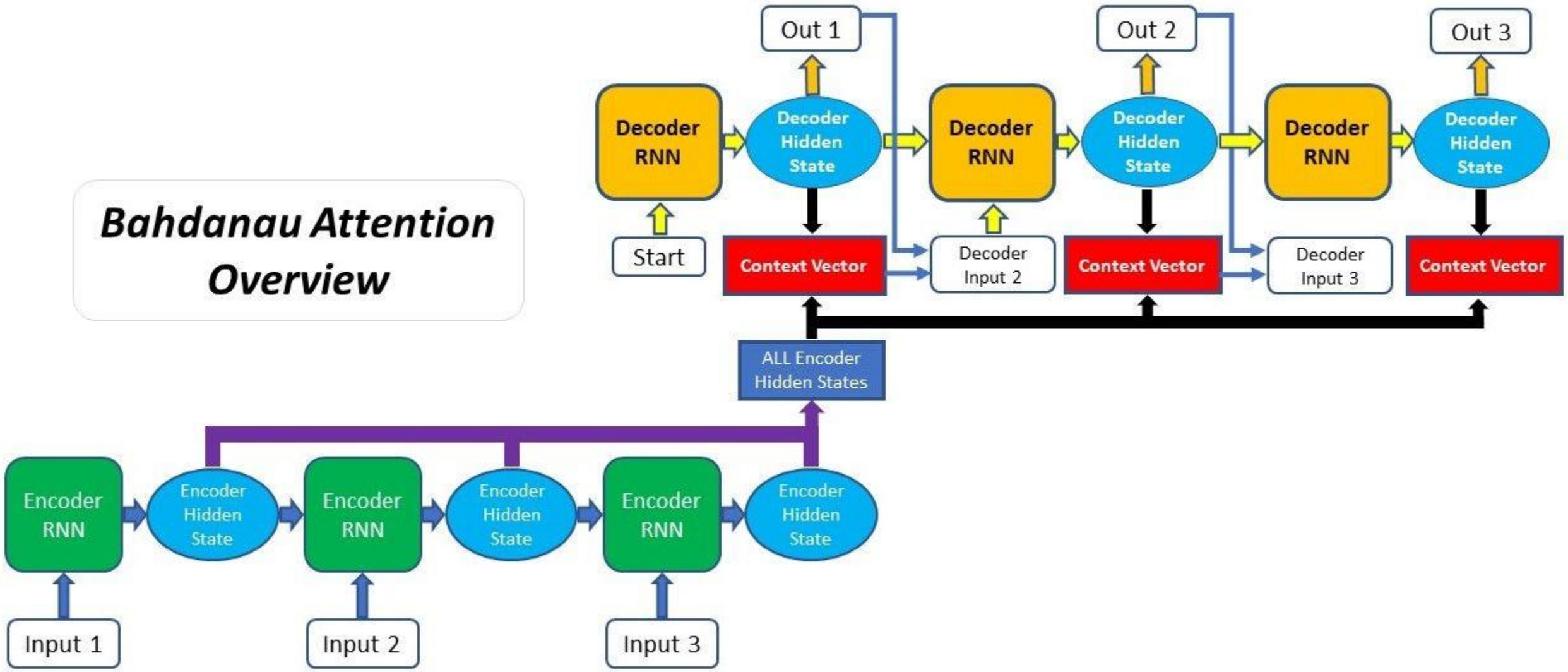
***Encoder-Decoder Recurrent Neural Network Models for Neural Machine Translation**
by **Jason Brownlee** on January 1, 2018 in **Deep Learning for Natural Language Processing**

The implementations of an attention layer can be broken down into 4 steps.

For each time step:

- **Step 0: Prepare hidden states.**
- First, prepare all the available encoder hidden states (green) and the first decoder hidden state (red). In our example, we have 4 encoder hidden states and the current decoder hidden state. (Note: the last consolidated encoder hidden state is fed as input to the first time step of the decoder. The output of this first time step of the decoder is called the first decoder hidden state.)
- **Step 1: Obtain a score for every encoder hidden state.**
- A score (scalar) is obtained by a score function (also known as alignment score function or alignment model). In this example, the score function is a dot product between the decoder and encoder hidden states.
- **Step 2: Run all the scores through a softmax layer.**
- We put the scores to a softmax layer so that the softmax scores (scalar) add up to 1. These softmax scores represent the attention distribution.
- **Step 3: Multiply each encoder hidden state by its softmax score.**
- By multiplying each encoder hidden state with its softmax score (scalar), we obtain the alignment vector or the annotation vector. This is exactly the mechanism where alignment takes place.
- **Step 4: Sum the alignment vectors.**
- The alignment vectors are summed up to produce the context vector. A context vector is an aggregated information of the alignment vectors from the previous step.
- **Step 5: Feed the context vector into the decoder.**

Bahdanau Attention Overview



*Neural Machine Translation by Jointly Learning to Align and Translate

Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio

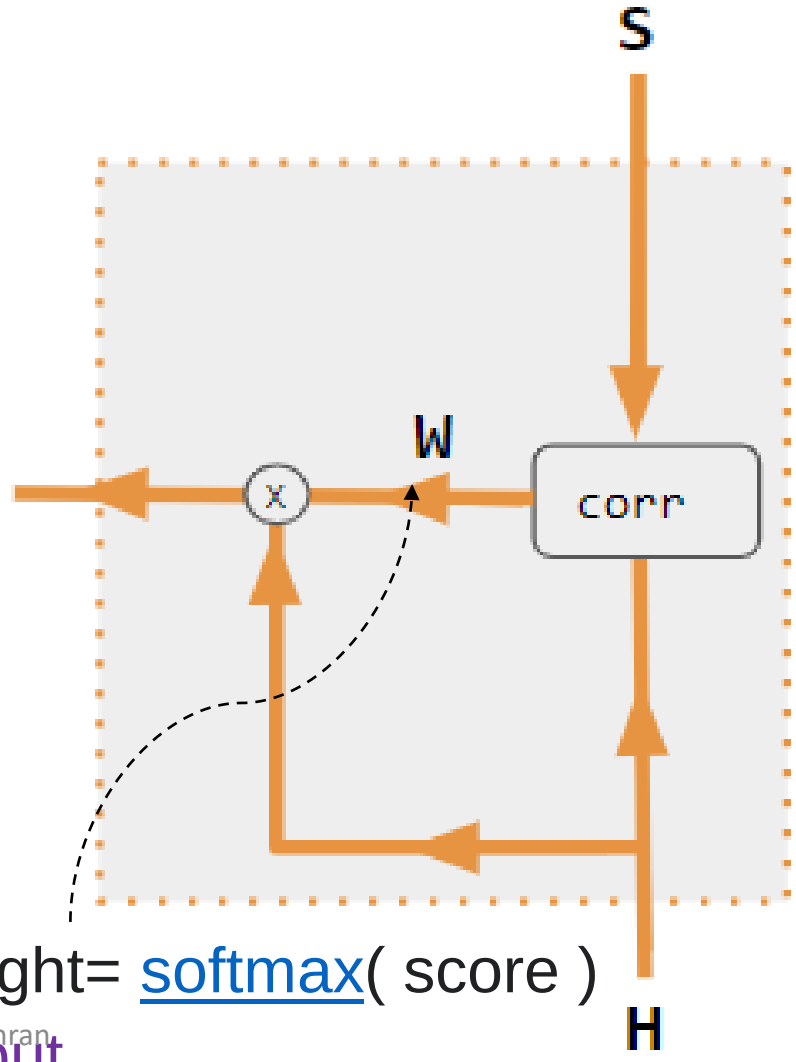
Attention variants

1. encoder-decoder dot product
2. encoder-decoder QKV (Query-Key-Value)
3. encoder-only dot product
4. encoder-only QKV
5. Pytorch tutorial

1. encoder-decoder dot product

s = decoder input to Attention
=Decoder hidden state

Both Encoder & Decoder are needed to calculate Attention.



c = context vector = H*w

w = attention weight= softmax(score)

H = encoder output

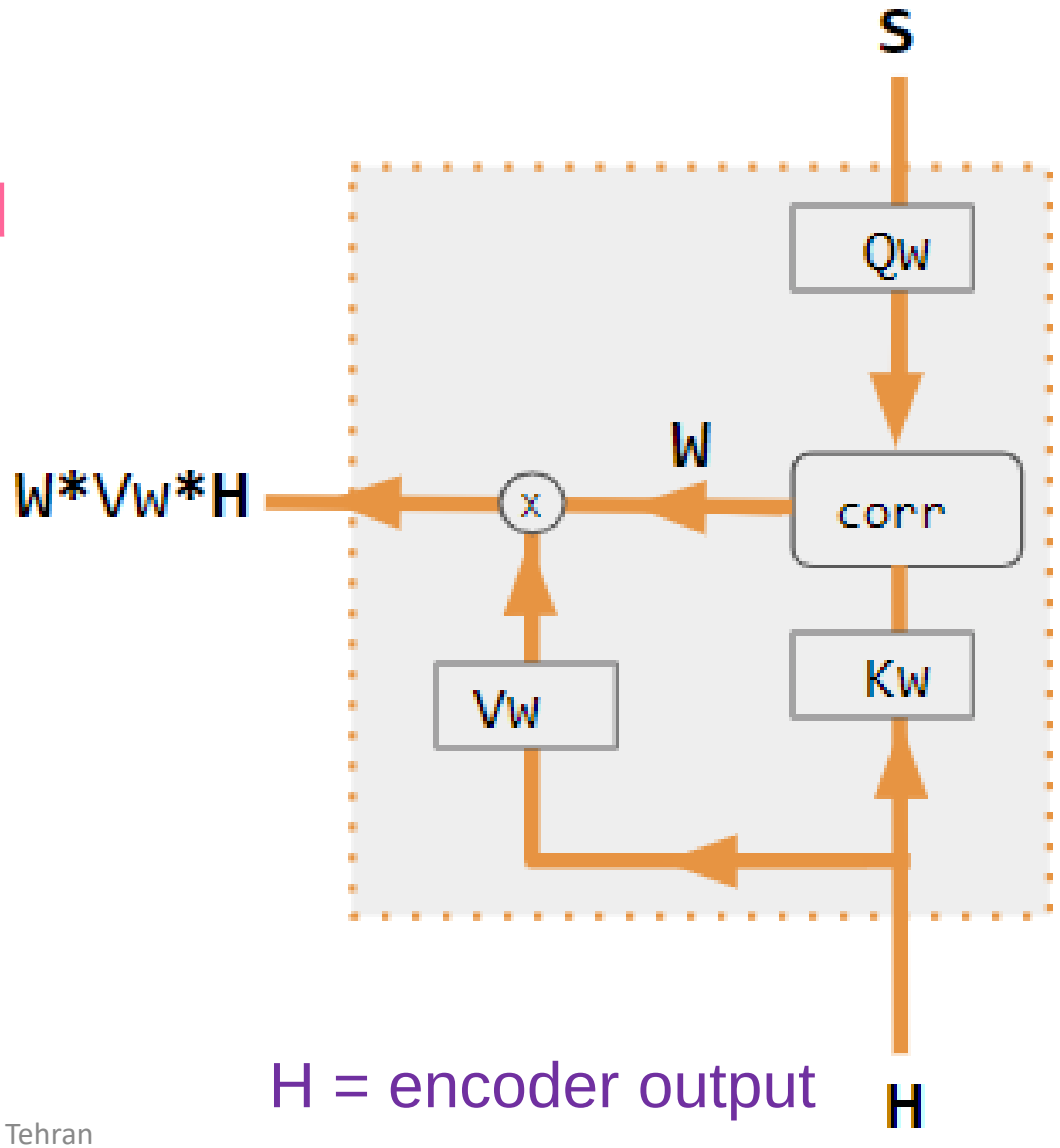
1.*Luong, Minh-Thang (2015-09-20). "Effective Approaches to Attention-based Neural Machine Translation". arXiv:1508.04025v5 [cs.CL].*

2. encoder-decoder QKV

s = decoder input to Attention

Both Encoder & Decoder are needed to calculate Attention.

c = context vector

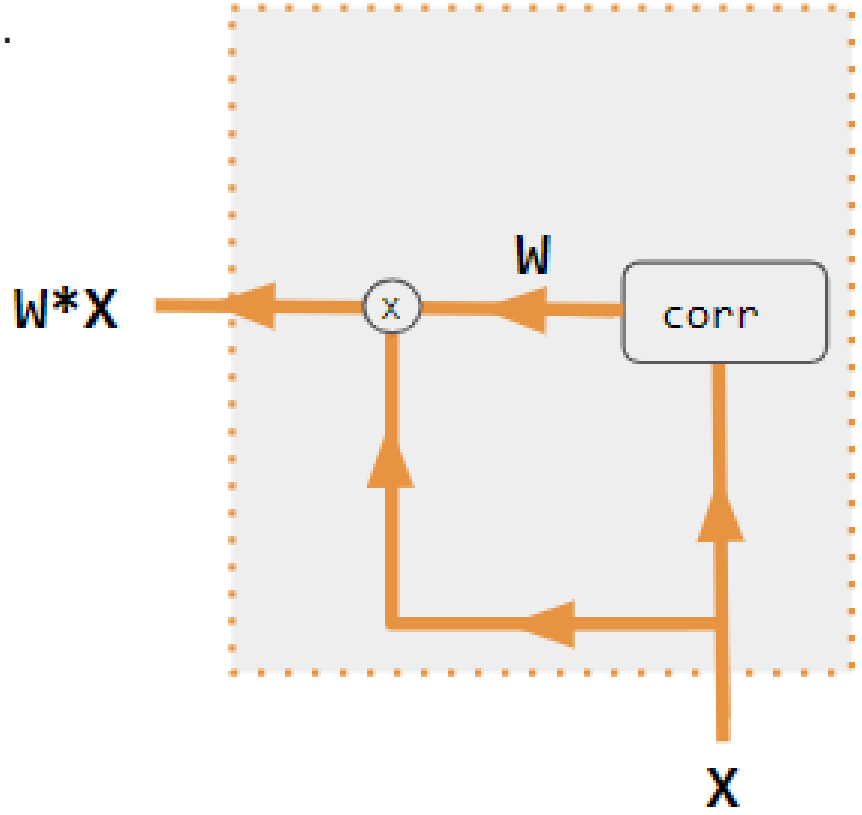


H = encoder output

1. *Neil Rhodes (2021). CS 152 NN—27: Attention: Keys, Queries, & Values. Event occurs at 06:30. Retrieved 2021-12-22.*

3. encoder-only dot product

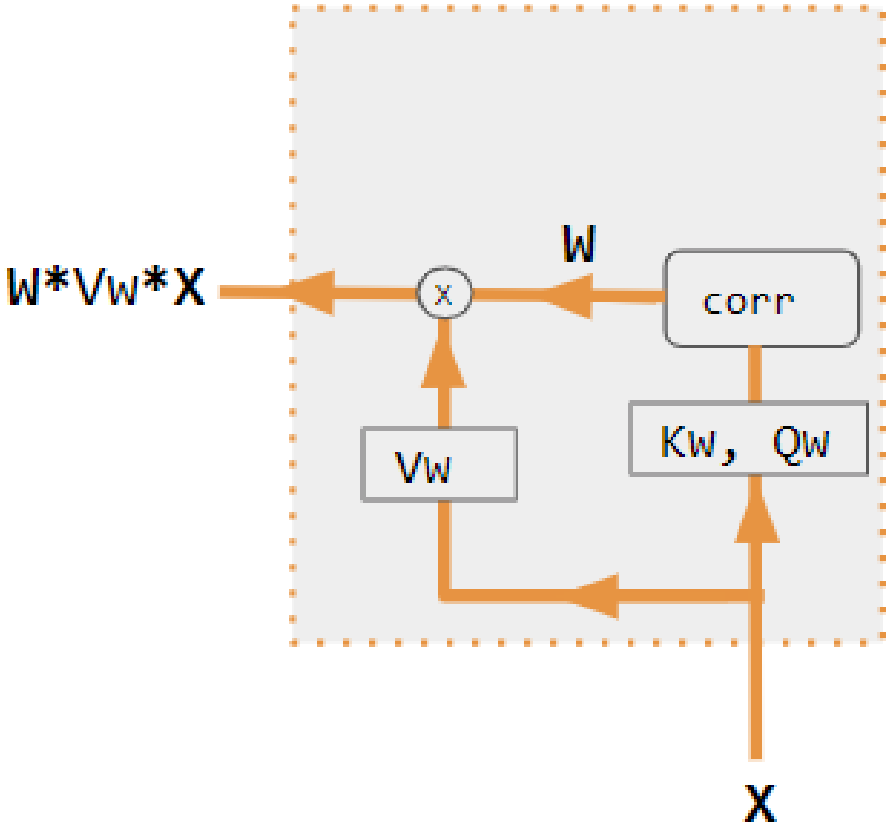
Decoder is NOT used to calculate Attention. With only 1 input into corr, W is an auto-correlation of dot products.
$w_{ij} = x_i * x_j$ [10]



1. *Alfredo Canziani & Yann Lecun (2021). NYU Deep Learning course, Spring 2020. Event occurs at 05:30. Retrieved 2021-12-22.*

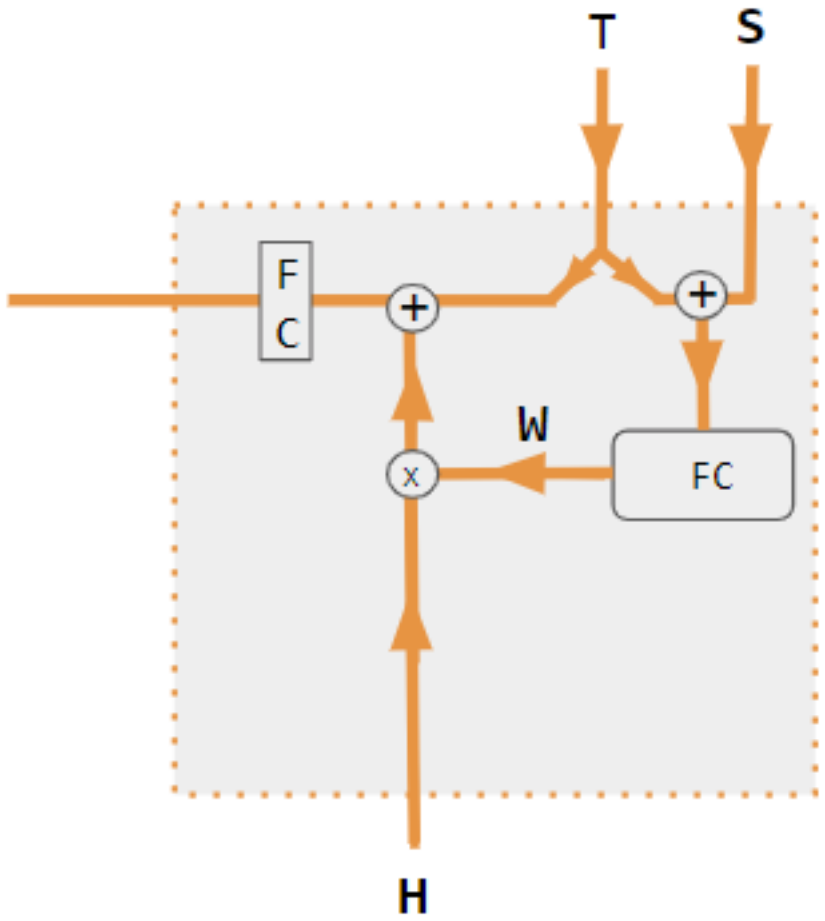
4. encoder-only QKV

Decoder is NOT used to calculate Attention.[11]



1. *Alfredo Canziani & Yann Lecun (2021). NYU Deep Learning course, Spring 2020. Event occurs at 20:15. Retrieved 2021-12-22.*

5. Pytorch tutorial



A FC layer is used to calculate Attention instead of dot product correlation.

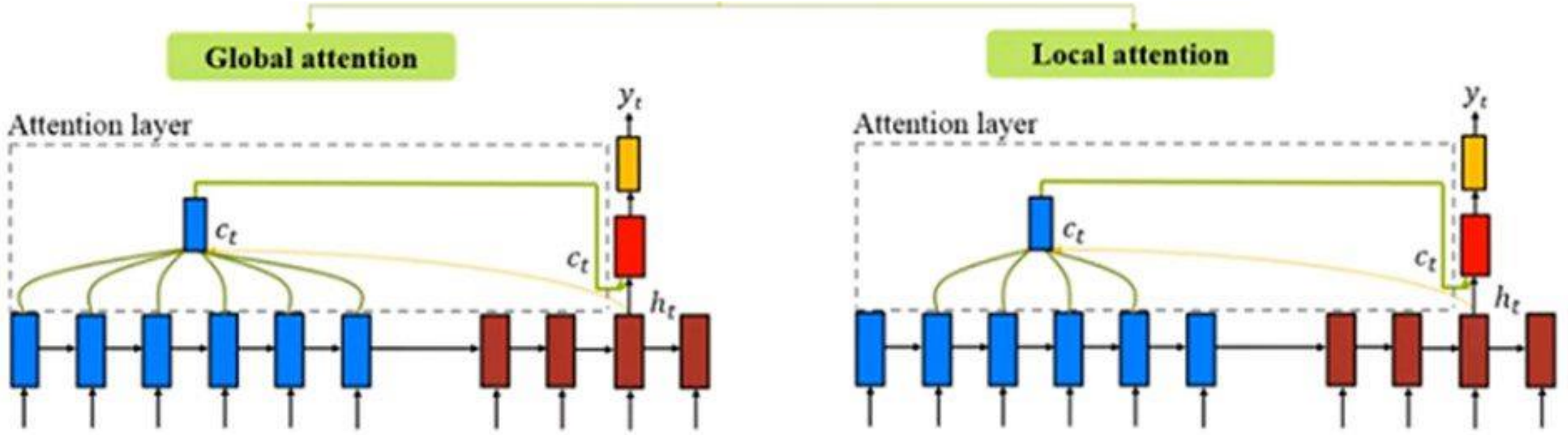
S = decoder hidden state, T = target word embedding.

H = encoder hidden state, X = input word embedding

1. *Robertson, Sean. "NLP From Scratch: Translation With a Sequence To Sequence Network and Attention". pytorch.org. Retrieved 2021-12-22.*

Types of attention

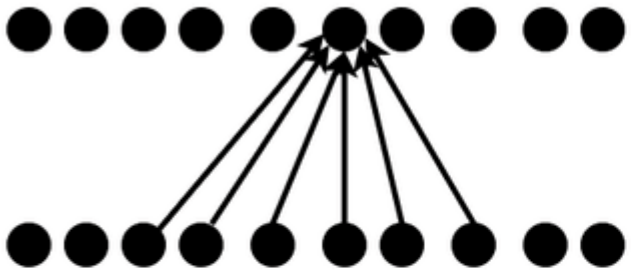
Depending on how many source states contribute while deriving the attention vector (α), there can be three types of attention mechanisms:



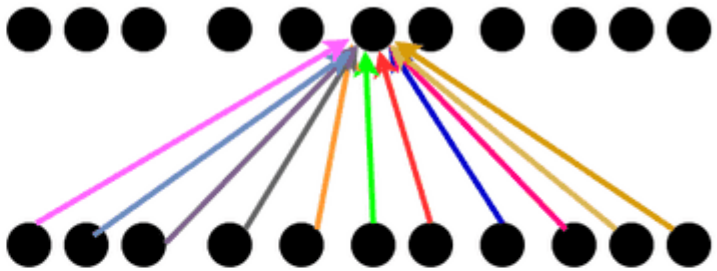
- **Global Attention**: When attention is placed on all source states. In global attention, we require as many weights as the source sentence length.
- **Local Attention**: When attention is placed on a few source states.
- **Hard Attention**: When attention is placed on only one source state.

You can check out my Kaggle notebook or GitHub repo to implement NMT with the attention mechanism using TensorFlow.

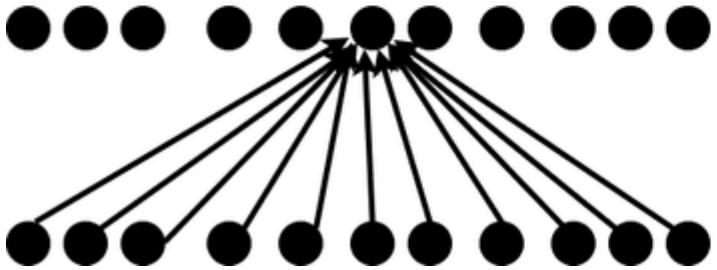
Convolution



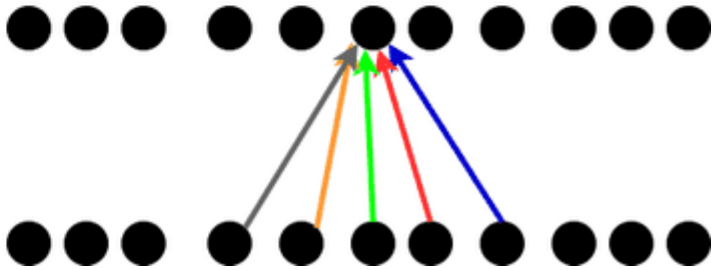
Global attention



Fully Connected layer



Local attention



1.1.5 Pooling layers

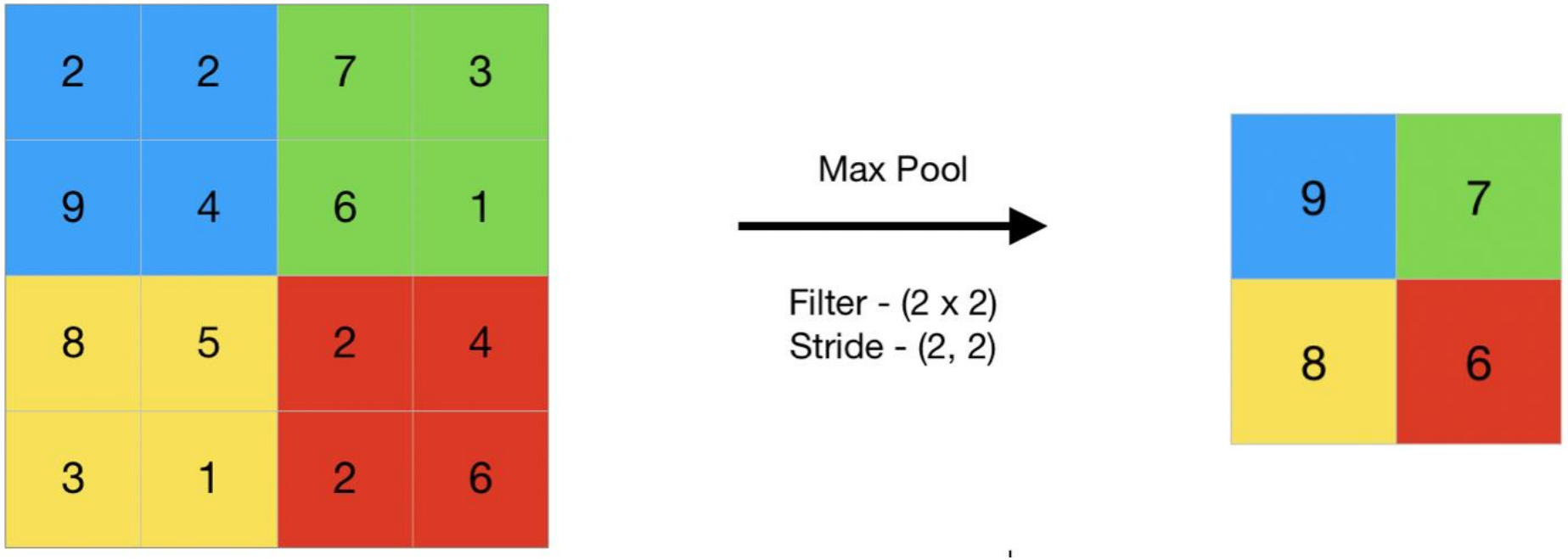
to reduce the spatial dimensions of the feature maps(subsampling) and to provide small spatial invariances

- Max pooling
- Min pooling
- Average pooling
- Global Pooling

Other pooling extensions

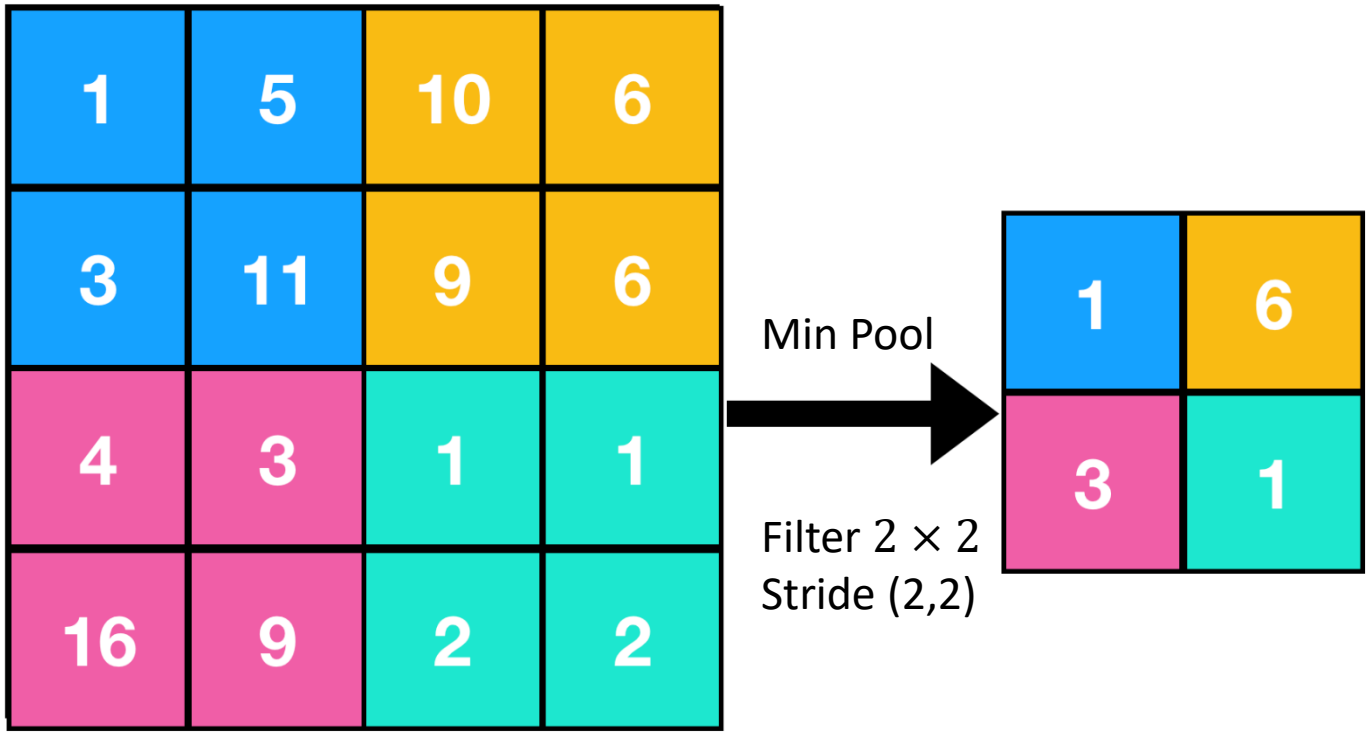
Mixed Pooling
Lp pooling
Stochastic pooling
Spatial Pyramid Pooling
Region of Interest Pooling
Multi-scale order-less pooling (MOP)
Super-pixel Pooling
PCA Networks
Compact Bilinear Pooling
Lead Asymmetric Pooling (LAP)
Edge-aware Pyramid Pooling
Spectral Pooling
Row-Wise Max-Pooling
Intermap Pooling
Per-pixel Pyramid Pooling
Rank-based Average Pooling
Weighted Pooling
Genetic-Based Pooling

Max pooling: The maximum pixel value of the batch is selected



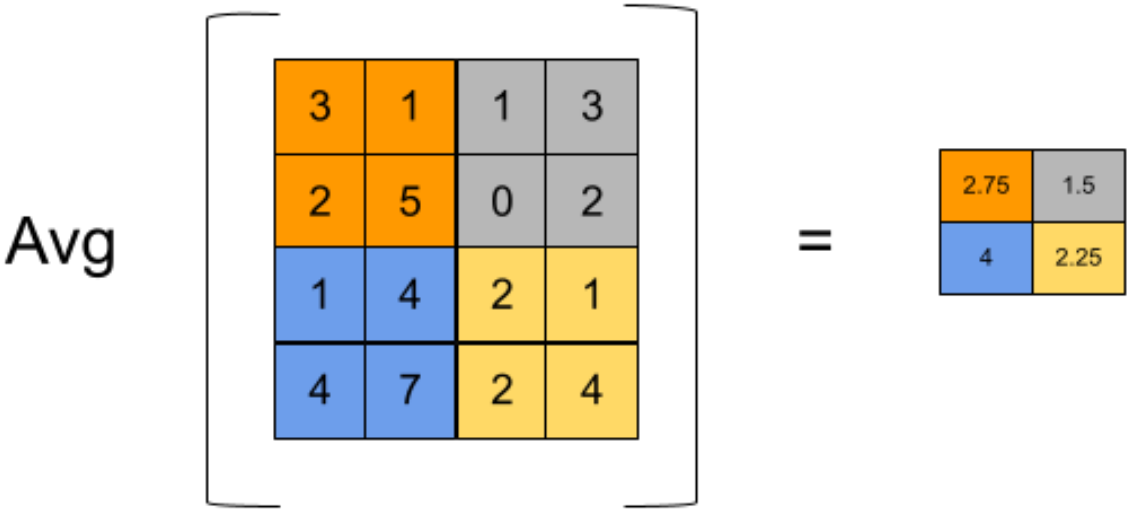
- Through Max Pooling, among units of a patch at each feature map, choose one which has the maximum similarity to the filter and ignore other ones.

Min pooling: The minimum pixel value of the batch is selected.



❖ It is mostly used when the image has a light background since min pooling will select darker pixels

Average pooling: The average value of all the pixels in the batch is selected.



Filter 2×2
Stride (2,2)

- Assuming all units of a patch are almost equal in information but with additive mean-zero noise, average pooling performs as a smooth function increasing the signal to noise ratio.

Global Pooling Layers

- Instead of down sampling patches of the input feature map, global pooling down samples the entire feature map to a single value.
- This would be the same as setting the *pool_size* to the size of the input feature map.
- Global pooling can be used in a model to aggressively summarize the presence of a feature in an image.
- It is also sometimes used in models as an alternative to using a fully connected layer to transition from feature maps to an output prediction for the model (**fully convolutional network like U-net**).
- Both global average pooling and global max pooling are defined.

1.1.6 Normalizing Layers

to speed up and stabilize the learning process

- Batch Normalization
- Weight Normalization
- Layer Normalization
- Group/Instance Normalization
- Weight Standarization

*Nilesh Vijayrania
Intrigued about Deep learning and all things ML.
Dec 10, 2020

Batch Normalization(BN)

A layer for standardization of inputs at each layer

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

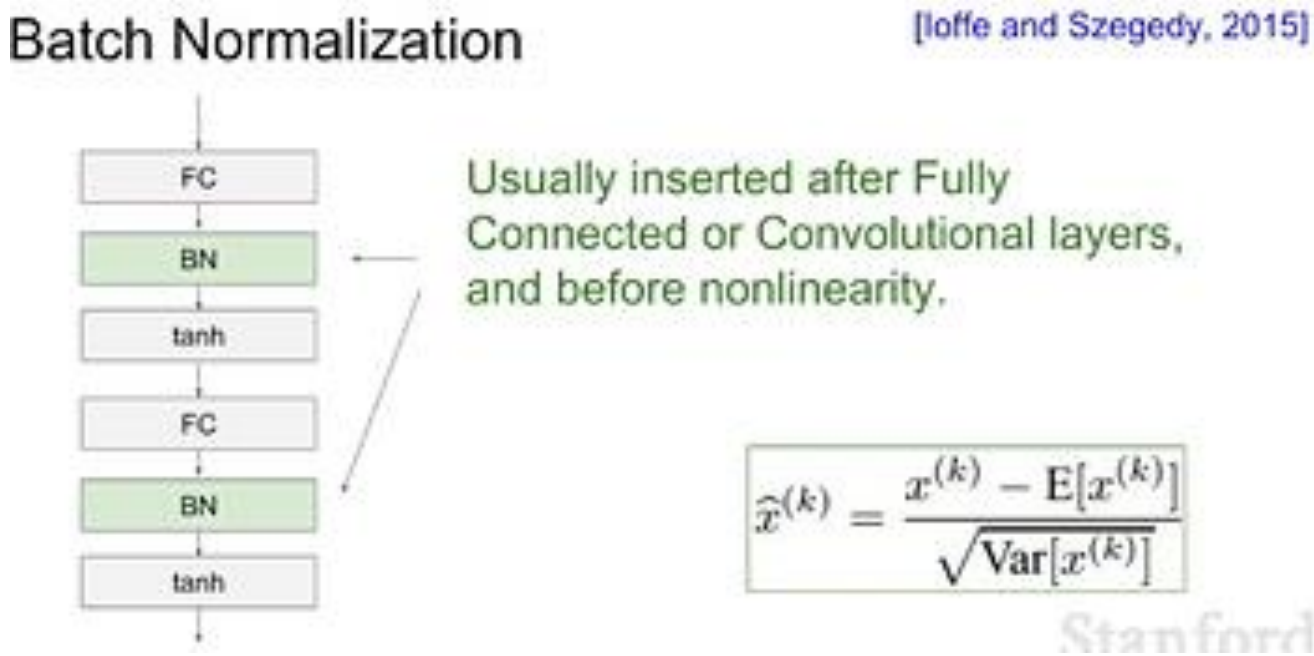
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.



The question is how BN helps NN training? since the layers are stacked one after the other, the data distribution of input to any particular layer changes too much due to slight update in weights of earlier layer, and hence the current gradient might produce suboptimal signals for the network. But BN restricts the distribution of the input data to any particular in the network, which helps the network to produce better gradients for weights update. Hence BN often provides a much stable and accelerated training regime

Weight Normalization(WN)

to decouple the length from the direction of the weight vector (Salimans, Tim, and Durk P. Kingma, 2016)

The authors of the Weight Normalization paper suggested using two parameters g(for length of the weight vector) and v(the direction of the weight vector) instead of w for gradient descent in the following manner.

$$\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}$$

- In weight normalization, the norm of the applied weights is constant (unitary) but the direction of the weights is free.
- Weight Normalization speeds up the training similar to batch normalization and unlike BN, it is applicable to RNNs as well.
- But the training of deep networks with Weight Normalization is significantly less stable compared to Batch Normalization and hence it is not widely used in practice.

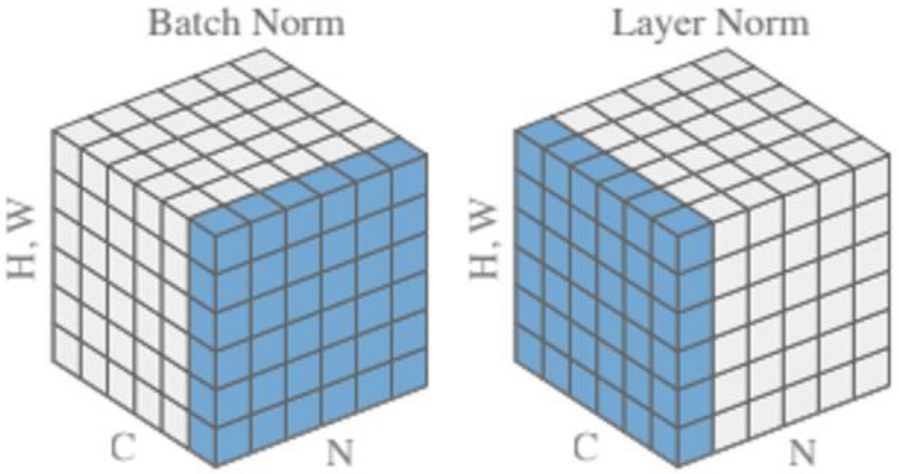
A Comparison between Batch Norm. and Wight Norm.

	BatchNorm: explicit normalization	WeightNorm: implicit normalization
Formula	$o_j = \frac{Wx-\mu_B}{\sigma_B^2}$	$o_j = \frac{Wx}{\|W\|_F}$
Goal	$\|o\|_2 \approx \text{const}$	$\|o\|_2 \approx \|x\|_2$
Assumptions	Batch is big enough	W is close to orthogonal matrix

Layer Normalization(LN)

Normalize a long feature maps rather than examples

- Layer Normalization normalizes the activations along the feature direction instead of mini-batch direction.
- This overcomes the cons of BN by removing the dependency on batches and makes it easier to apply for RNNs as well.
- In essence, Layer Normalization normalizes each feature of the activations to zero mean and unit variance.
- In batch normalization, **input values of the same neuron for all the data in the mini-batch are normalized**. Whereas in layer normalization, input values for all neurons in the same layer are normalized for each data sample.



(N, C, H, W)

N: number of data samples in Mini Batch

C: number of feature maps (channels)

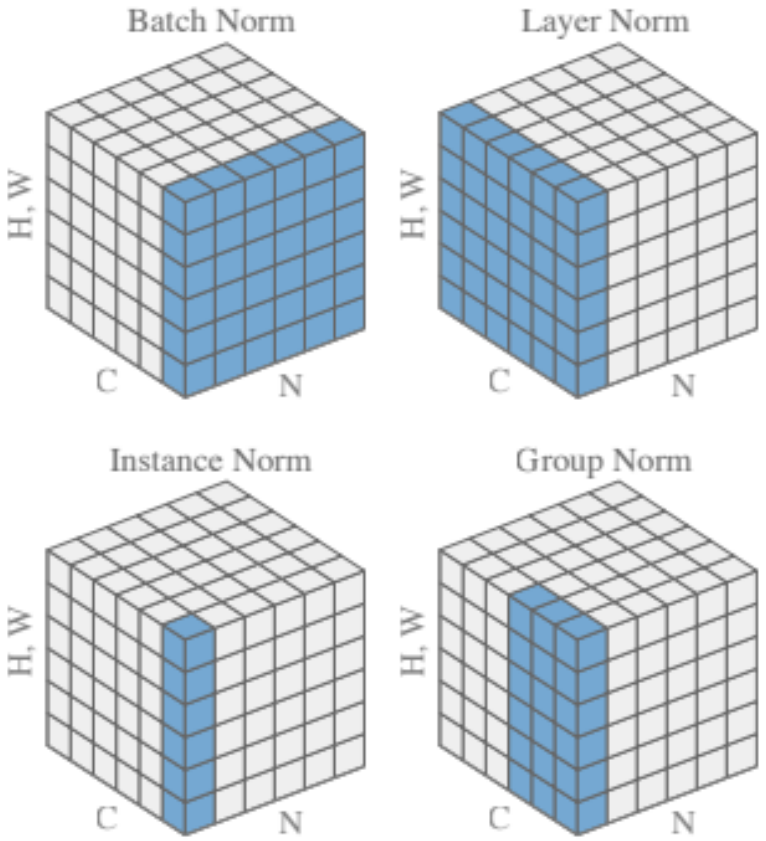
H,W: spatial size of feature maps

* Geoffrey Hinton et al. 2016

Group/Instance Normalization(GN)

normalize along a certain groups of feature maps and normalizes each group separately

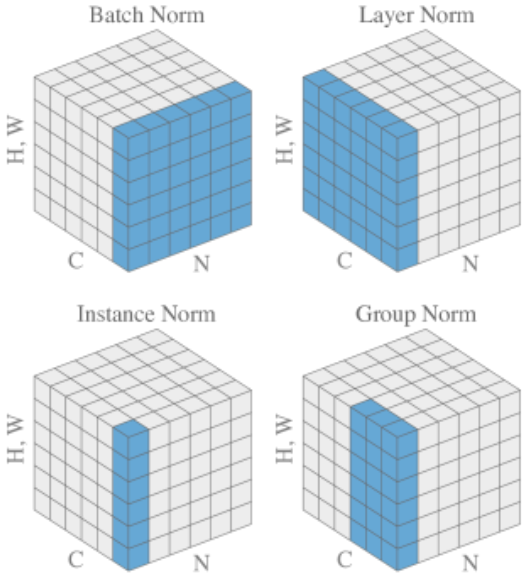
- Similar to layer Normalization, Group Normalization is also applied along the feature direction but unlike LN, it divides the features into certain groups and normalizes each group separately.
- In practice, Group normalization performs better than layer normalization.
- its parameter *num_groups* is tuned as a hyperparameter.
- In a case, when we choose one feature map to normalize, we have an instance normalization.

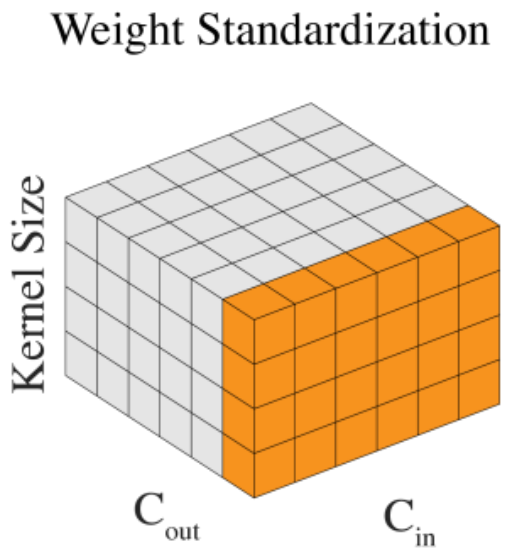


Weight standardization

to batch normalization of weights at a layer instead of data

Weight Standardization is transforming the weights of any layer to have zero mean and unit variance.
This layer could be a convolution layer, RNN layer or linear layer, etc.
For any given layer with shape(N, *) where * represents 1 or more dimensions, weight standardization, transforms the weights along the * dimension(s).





$$\hat{\boldsymbol{W}} = \left[\hat{\boldsymbol{W}}_{i,j} \mid \hat{\boldsymbol{W}}_{i,j} = \frac{\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}}}{\sigma_{\boldsymbol{W}_{i,\cdot}} + \epsilon}\right]$$

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x}$$

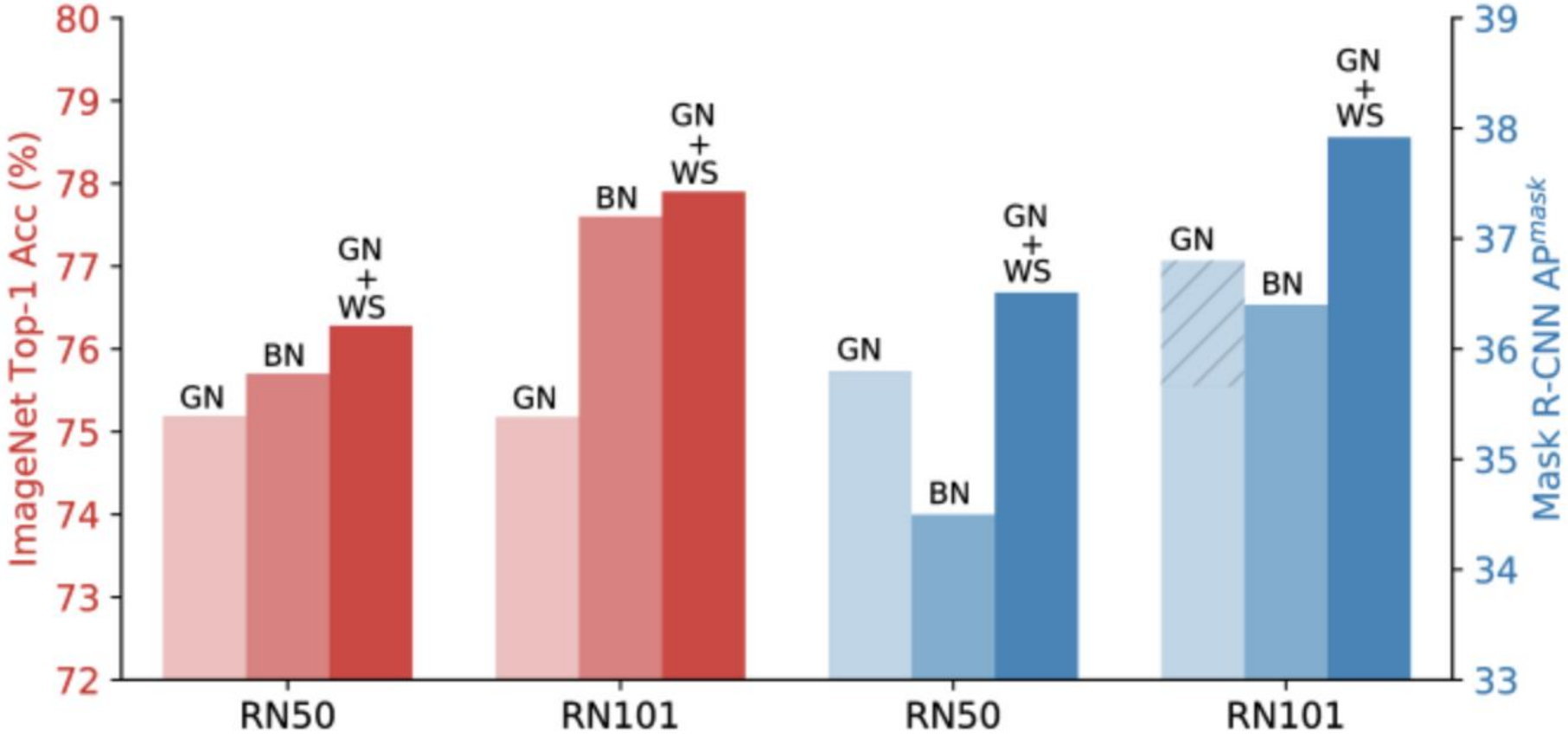
$$\mu_{\boldsymbol{W}_{i,\cdot}} = \frac{1}{I}\sum_{j=1}^{I} \boldsymbol{W}_{i,j}, \quad \sigma_{\boldsymbol{W}_{i,\cdot}} = \sqrt{\frac{1}{I}\sum_{i=1}^{I}(\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}})^2}$$

C_{in}: number of input channels
C_{out}: number of output channels
I: corresponds to the number of input channels within the kernel region of each output channel.

*Qiao, Siyuan, et al. 2019

Comparison of different normalization layers*

in Resnet50 and Resnet101 on ImageNet classification and MS COCO



*Siyuan Qiao et al. Weight Standardization. GN+WS effect on classification and object detection task[4]

1.2 Architectures

1.2.1 CNNs

1.2.2 Region based CNNs (R-CNNs)

1.2.3 CNNs for Segmentation

1.2.4 Transformers

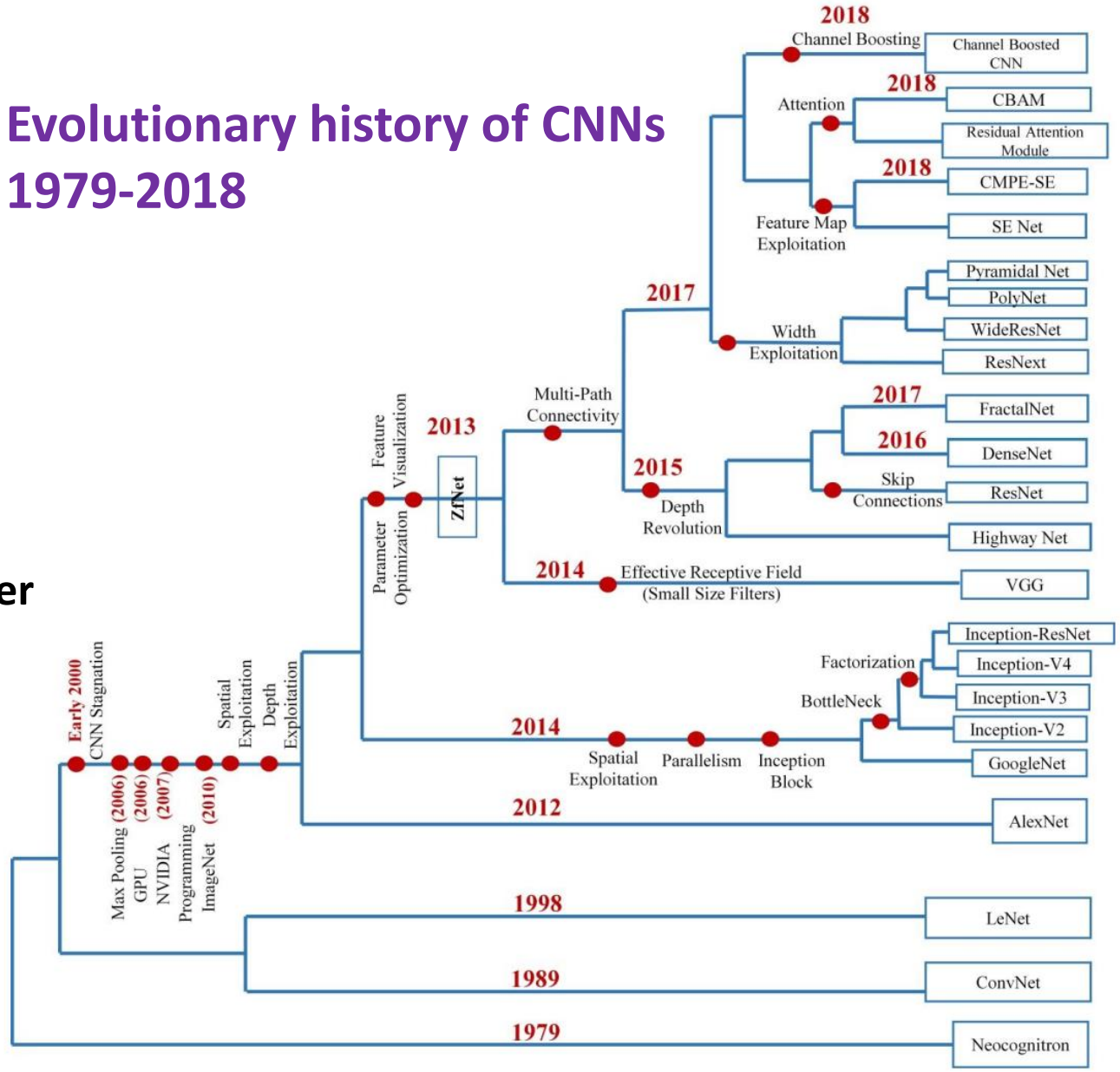
1.2.1 CNNs

Neural Networks which use convolution layers to filter and extract features from signals to solve a classification or regression problem.

In 1989, LeCuN et al. proposed the first multilayered CNN named ConvNet, whose origin rooted in Fukushima's Neocognitron (Fukushima and Miyake 1982; Fukushima 1988). **Japanese handwritten character recognition and other pattern recognition tasks**

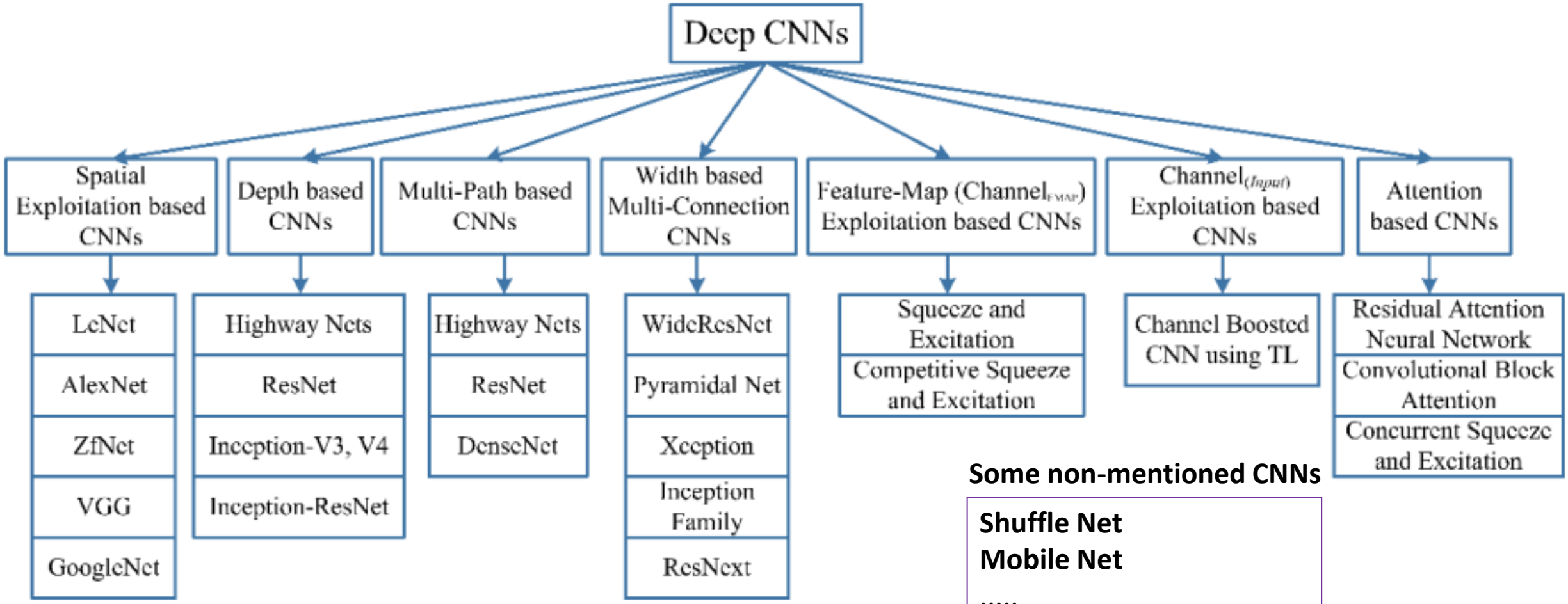
In 1998, LeCuN proposed an improved version of ConvNet, which was famously known as LeNet-5, and it started the use of CNN in classifying characters in a document recognition related applications (LeCun et al. 1995, 1998).

Evolutionary history of CNNs 1979-2018



*Assifullah khan, et al. 2020

Taxonomy of deep CNN architectures showing seven different categories



Some non-mentioned CNNs

Shuffle Net
Mobile Net
.....
Efficient Net
Transformer-CNN

(1) Spatial Exploitation based CNNs

CNNs which improve themselves by exploring different levels of neighborhood (spatial) correlation among pixels of an input image or units of feature maps by employing different filter sizes

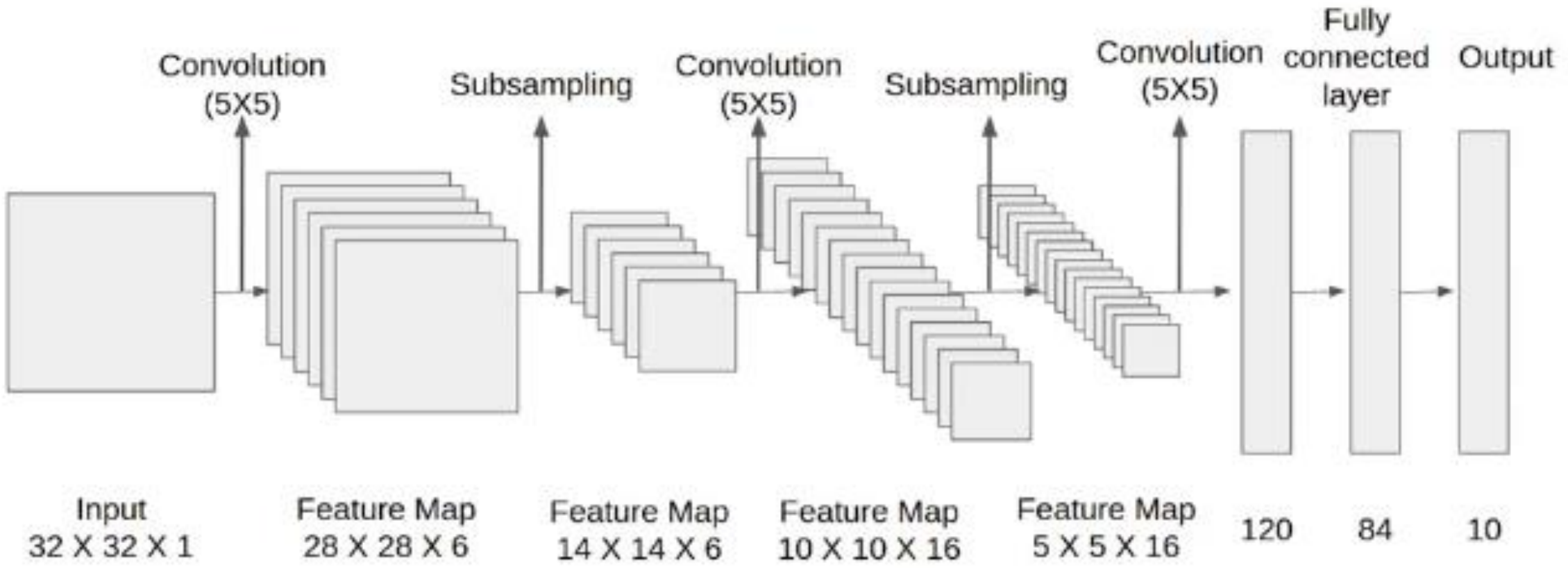
CNNs: **LeNet** **AlexNet** **ZFNet** **VGG** **GoogleNet**

.

Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
LeNet	1998	- First popular CNN architecture	0.060 M	[dist]MNIST: 0.8 MNIST: 0.95	5	Spatial Exploitation	(LeCun et al. 1995)
AlexNet	2012	- Deeper and wider than the LeNet - Uses Relu, dropout and overlap Pooling - GPUs NVIDIA GTX 580	60 M	ImageNet: 16.4	8	Spatial Exploitation	(Krizhevsky et al. 2012)
ZfNet	2014	-Visualization of intermediate layers	60 M	ImageNet: 11.7	8	Spatial Exploitation	(Zeiler and Fergus 2013)
VGG	2014	- Homogenous topology - Uses small size kernels	138 M	ImageNet: 7.3	19	Spatial Exploitation	(Simonyan and Zisserman 2015)
GoogLeNet	2015	- Introduced block concept - Split transform and merge idea	4 M	ImageNet: 6.7	22	Spatial Exploitation	(Szegedy et al. 2015)

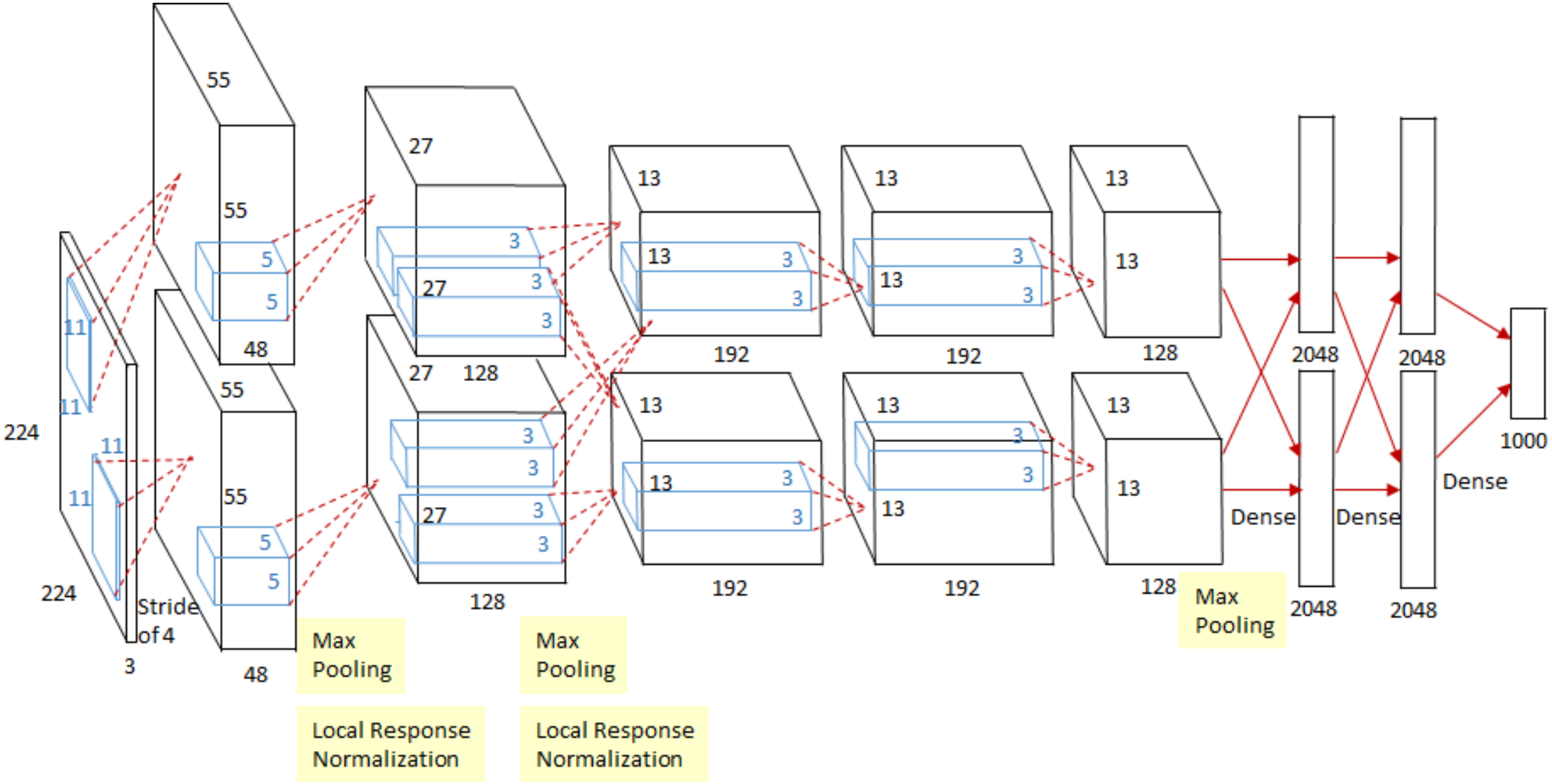
Lenet-5

First popular CNN architecture



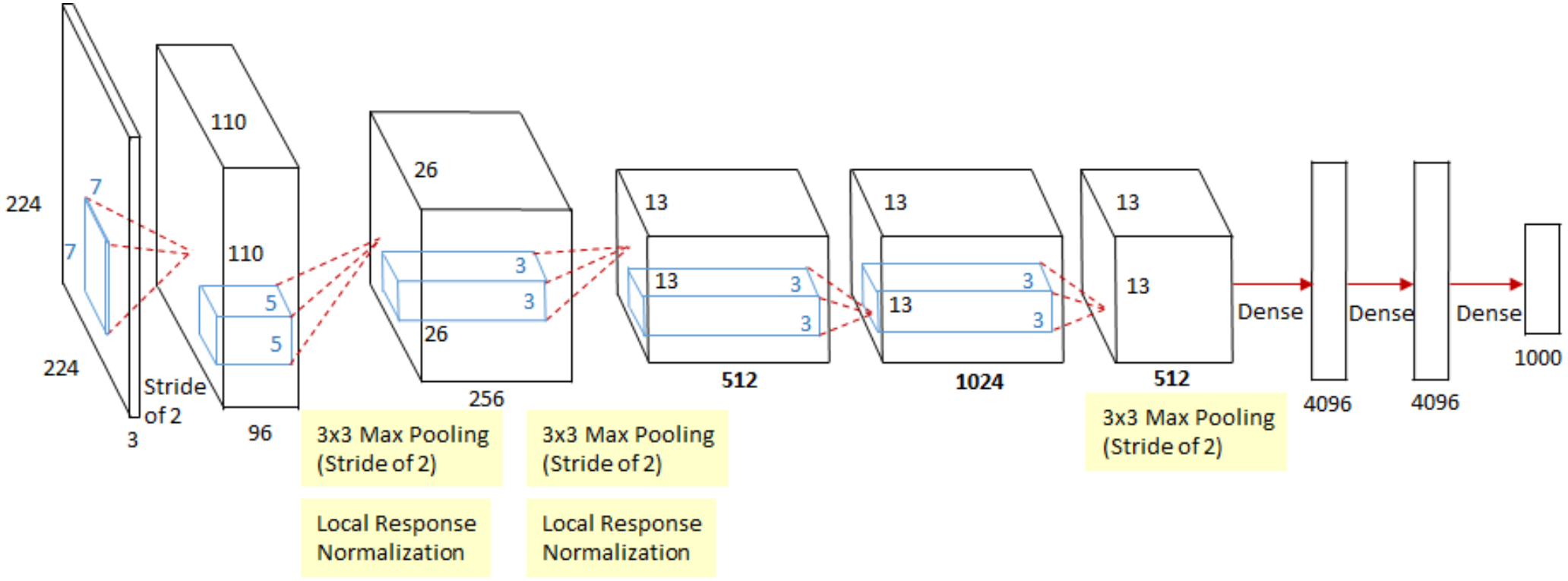
AlexNet

- Deeper and wider than the LeNet
- Uses Relu, dropout and overlap Pooling



ZFNET

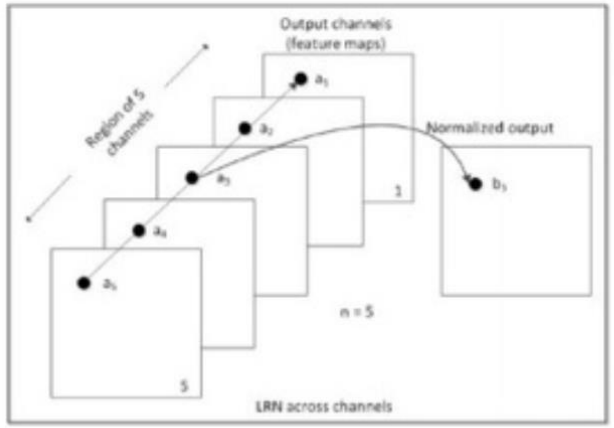
Visualization of intermediate layers

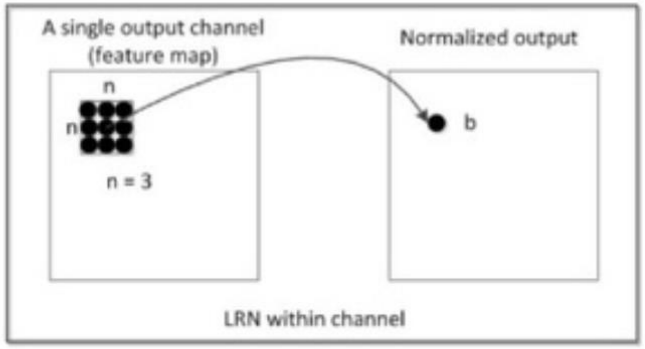


Local Response Normalization (utilized in AlexNET and ZFNet)

Local Response Normalization is a normalization layer that implements the idea of lateral inhibition.
Lateral inhibition is a concept in neurobiology that refers to the phenomenon of an excited neuron inhibiting its neighbours: this leads to a peak in the form of a local maximum, creating contrast in that area and increasing sensory perception. In practice, we can either normalize within the same channel or normalize across channels when we apply LRN to convolutional neural networks.

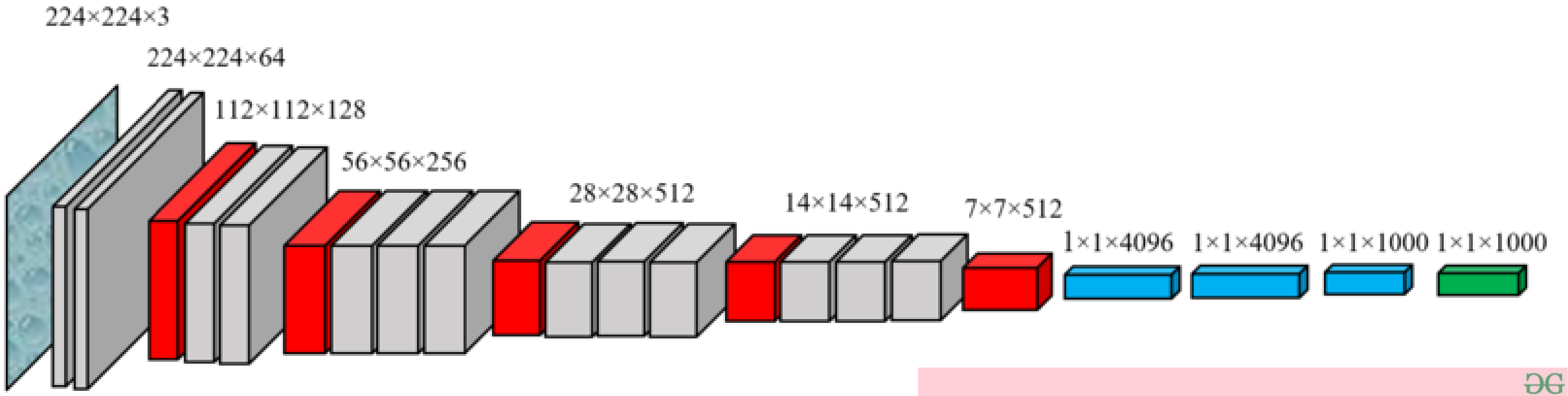
- Tries to mimic the inhibition scheme in the brain

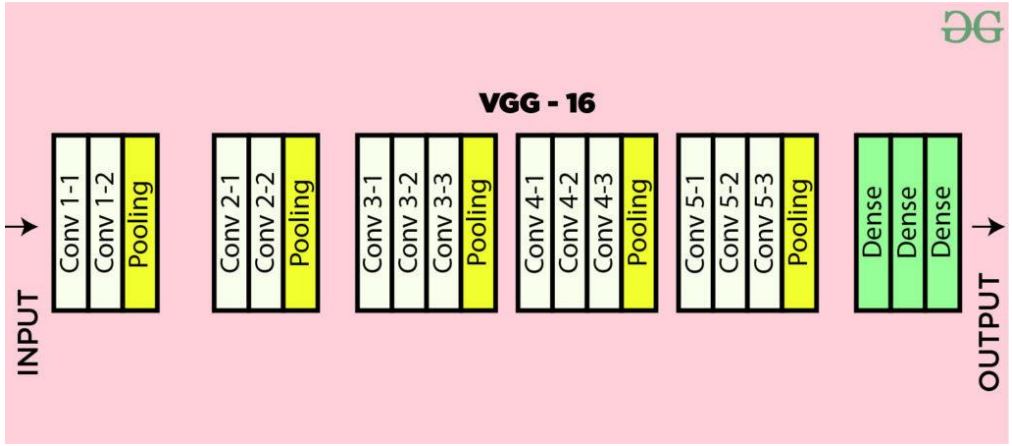




VGG

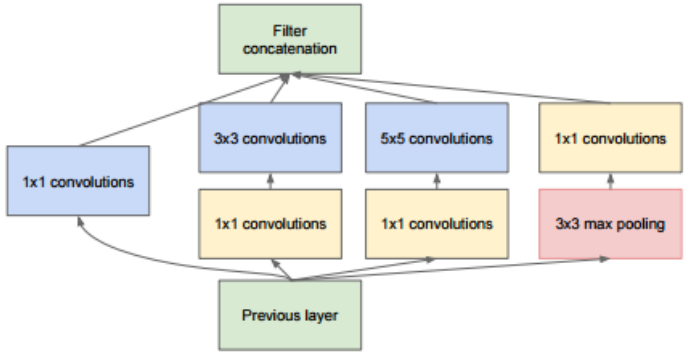
- Homogenous topology
- Uses small size kernels





GoogLeNet

- Introduced block concept
- Split transform and merge idea



(b) Inception module with dimension reductions

Auxiliary Classifiers are type of architectural component that seek to improve the convergence of very deep networks.
They are classifier heads we attach to layers before the end of the network.
The motivation is to push useful gradients to the lower layers to make them immediately useful and improve the convergence during training by combatting the vanishing gradient problem.
They are notably used in the Inception family of convolutional neural networks.

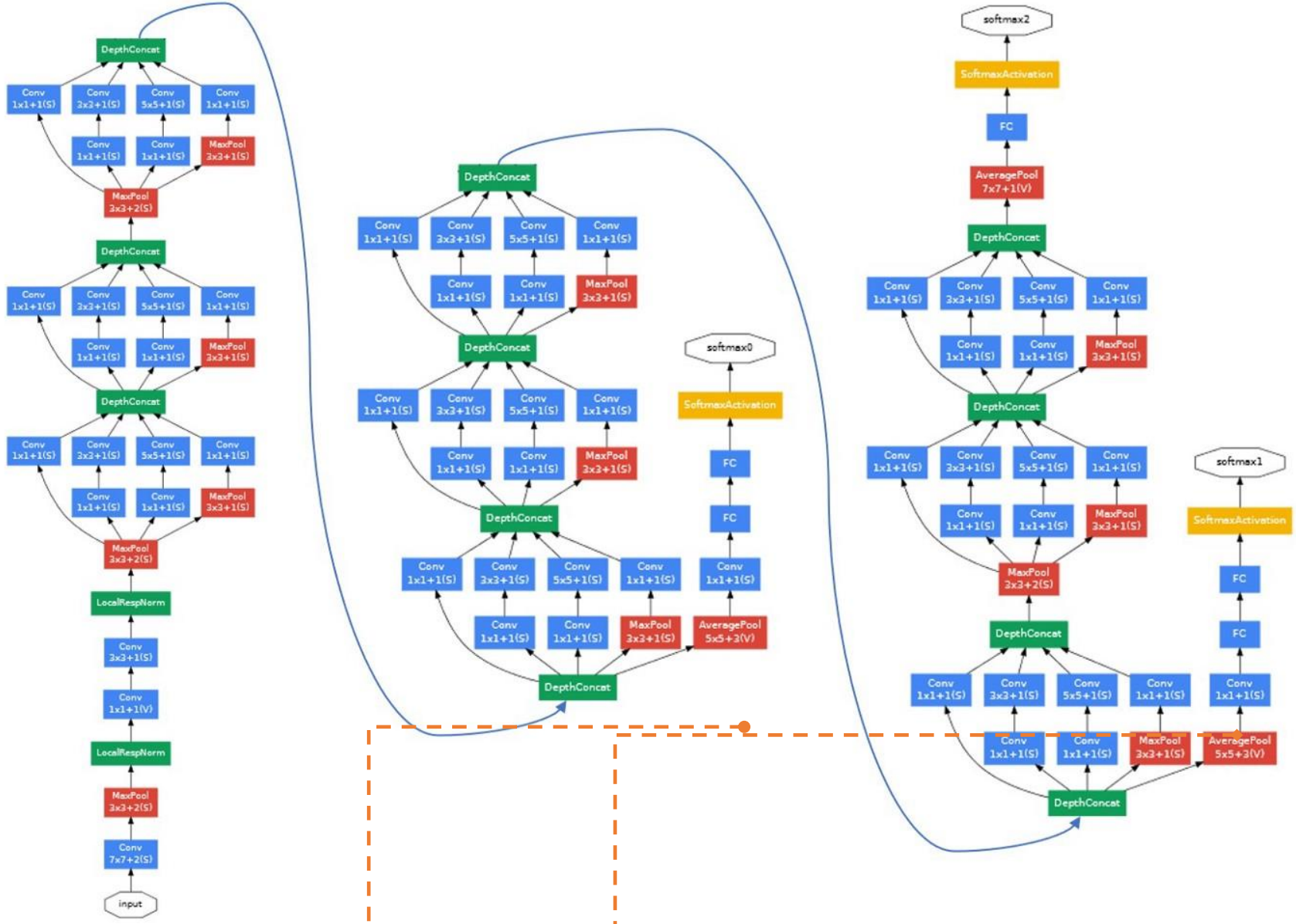


Table 5a Major challenges associated with implementation of Spatial exploitation based CNN architectures.

Spatial Exploitation	As convolutional operation considers the neighborhood (correlation) of input pixels, therefore different levels of correlation can be explored by using different filter sizes.	
Architecture	**Strength**	**Gaps**
LeNet	• Exploited spatial correlation to reduce the computation and number of parameters • Automatic learning of feature hierarchies	• Poor scaling to diverse classes of images • Large size filters • Low level feature extraction
AlexNet	• Low, mid and high-level feature extraction using large and small size filters on initial (5x5 and 11x11) and last layers (3x3) • Give an idea of deep and wide CNN architecture • Introduced regularization in CNN • Started parallel use of GPUs as an accelerator to deal with complex architectures	• Inactive neurons in the first and second layers • Aliasing artifacts in the learned feature-maps due to large filter size
ZfNet	• Introduced the idea of parameter tuning by visualizing the output of intermediate layers • Reduced both the filter size and stride in the first two layers of AlexNet	• Extra information processing is required for visualization
VGG	• Proposed an idea of effective receptive field • Gave the idea of simple and homogenous topology	• Use of computationally expensive fully connected layers
GoogLeNet	• Introduced the idea of using Mutiscale Filters within the layers • Gave a new idea of split, transform, and merge • Reduce the number of parameters by using bottleneck layer, global average-pooling at last layer and Sparse Connections • Use of auxiliary classifiers to improve the convergence rate	• Tedious parameter customization due to heterogeneous topology • May lose the useful information due to representational bottleneck

(2) Depth based CNNs

CNNs which improve themselves by increasing the depth to utilize more cascaded filters and to achieve better feature representation

CNNs: Inception-V3, V4 Inception-ResNet ResNet Highway Networks

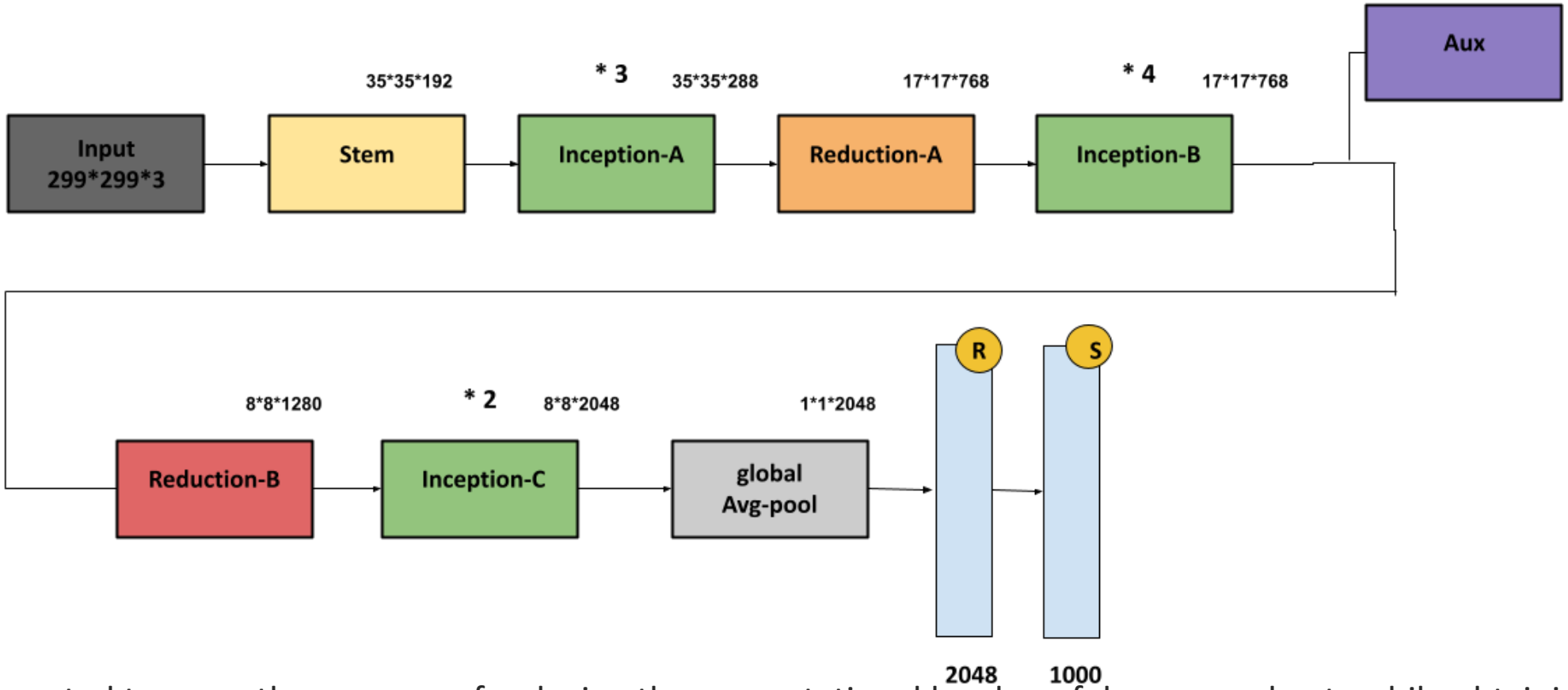
Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6	159	Depth + Width	(Szegedy et al. 2016b)
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76	19	Depth + Multi-Path	(Srivastava et al. 2015a)
Inception-V4	2016	- Split transform and merge idea Uses asymmetric filters	35 M	ImageNet: 4.01	70	Depth +Width	(Szegedy et al. 2016a)
Inception-ResNet	2016	- Uses split transform merge idea and residual links	55.8M	ImageNet: 3.52	572	Depth + Width + Multi-Path	(Szegedy et al. 2016a)
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M	ImageNet: 3.6 CIFAR-10: 6.43	152 110	Depth + Multi-Path	(He et al. 2015a)

Inception V3

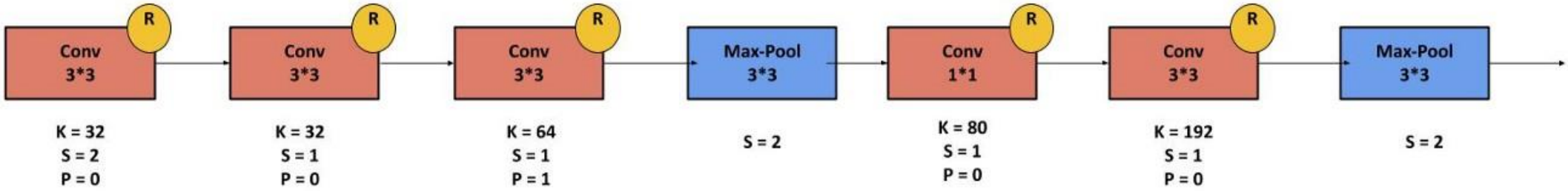
- Handles the problem of a representational bottleneck

-Using inception modules, Inceptions decease the representation size and avoid information missing

- Replace large size filters with small filters

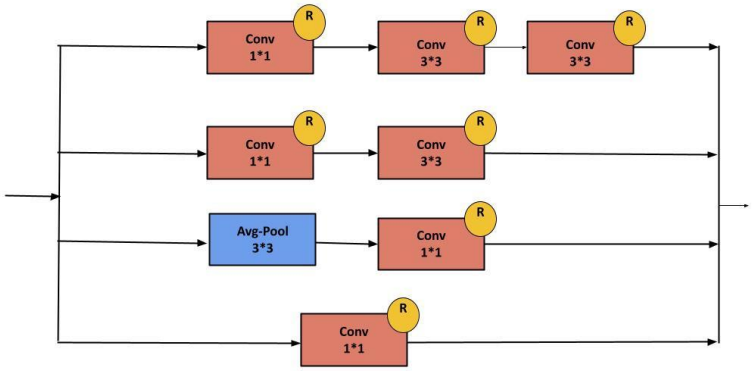


Inception is created to serve the purpose of reducing the computational burden of deep neural nets while obtaining state-of-art performance.

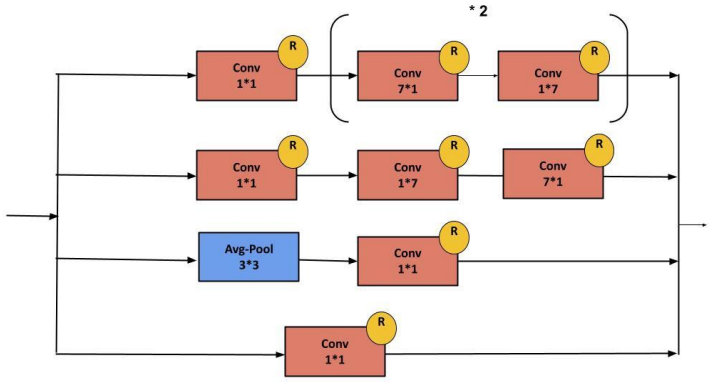


STEM

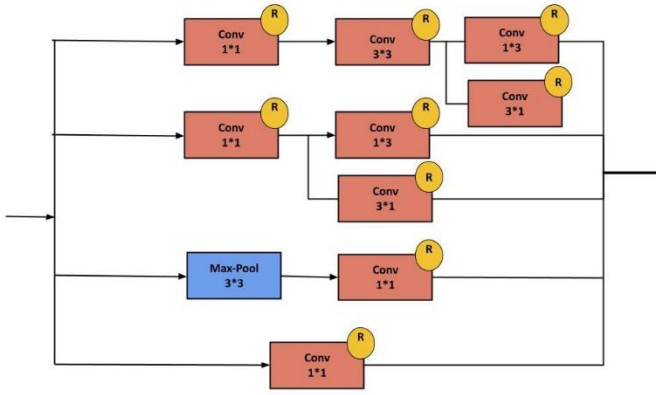
The Stem is a particular convolutional network module before the Inception-resnet blocks



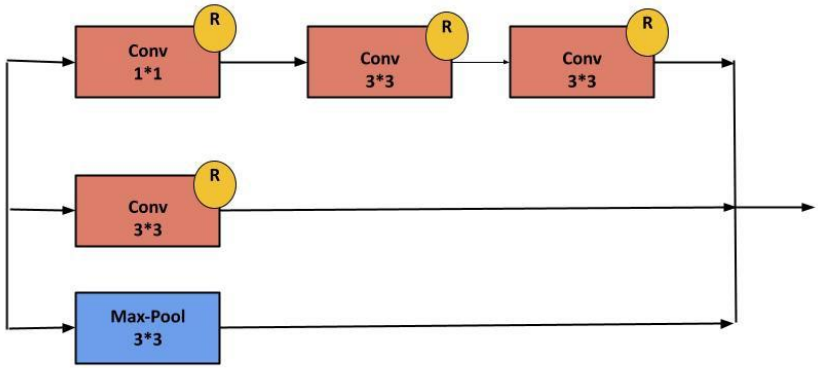
Inception-A Block :



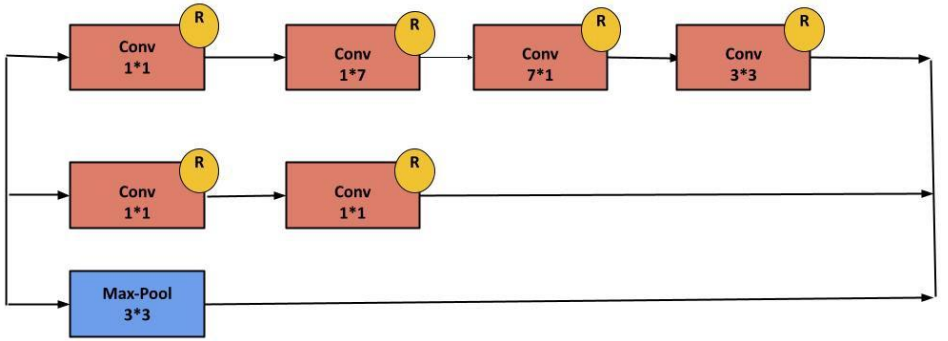
Inception-B Block :



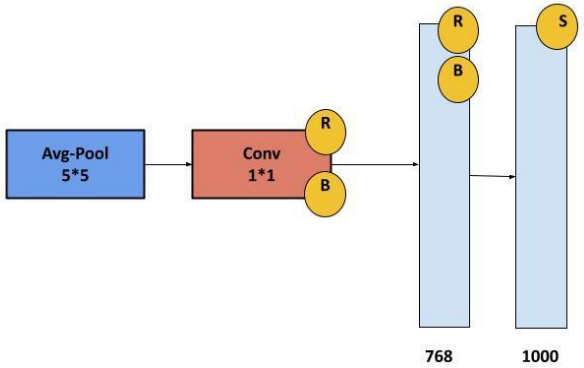
Inception-C Block :



Reduction-A Block :



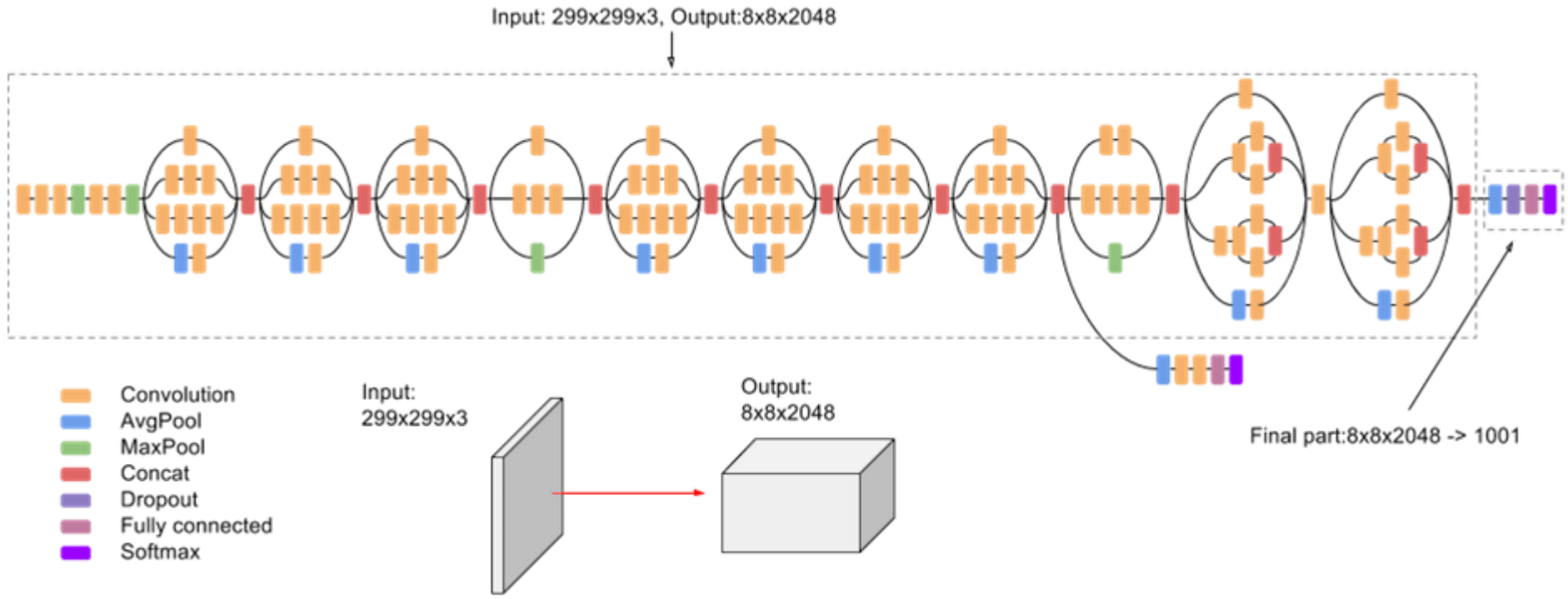
Reduction-B Block :



Auxiliary Classifier Block :

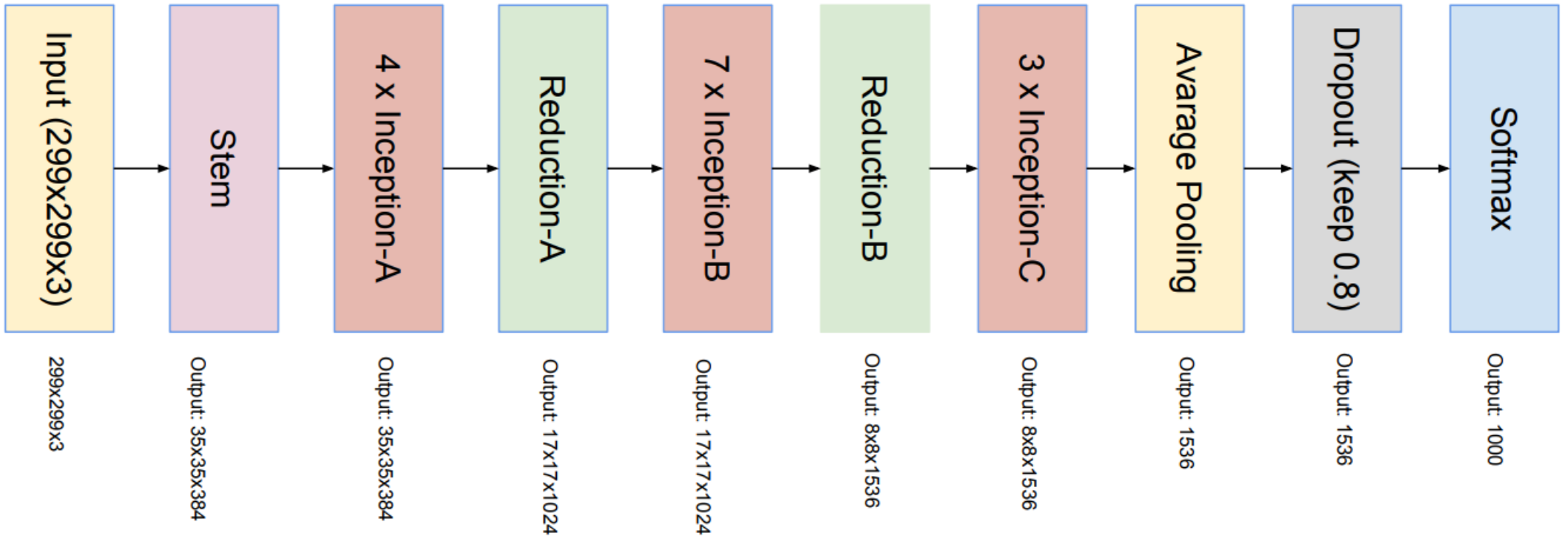
Auxiliary Classifiers are **type of architectural component that seek to improve the convergence of very deep networks**. They are classifier heads we attach to layers before the end of the network.

Inception 3

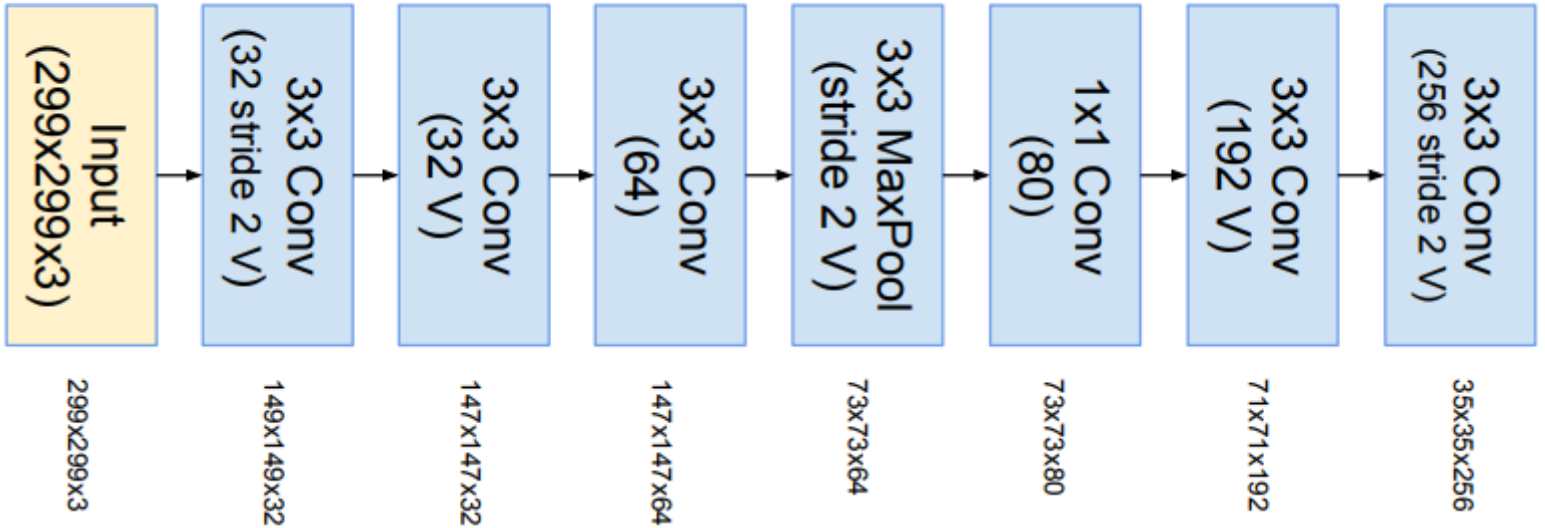

Input: 299x299x3, Output:8x8x2048
Final part:8x8x2048 -> 1001
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax
Input:
299x299x3
Output:
8x8x2048

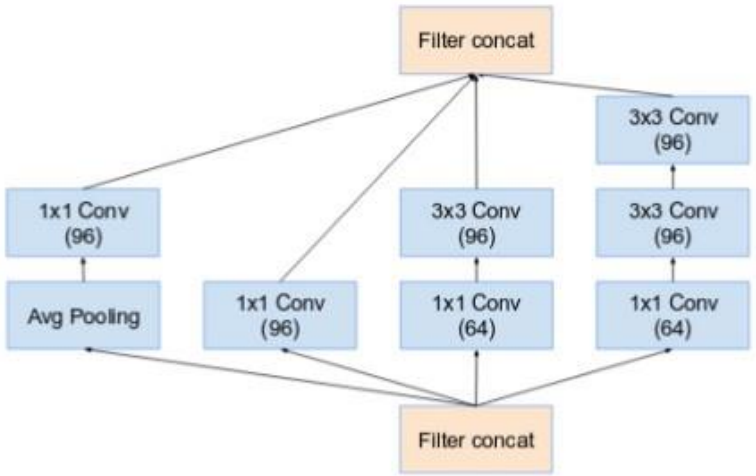
Inception 4

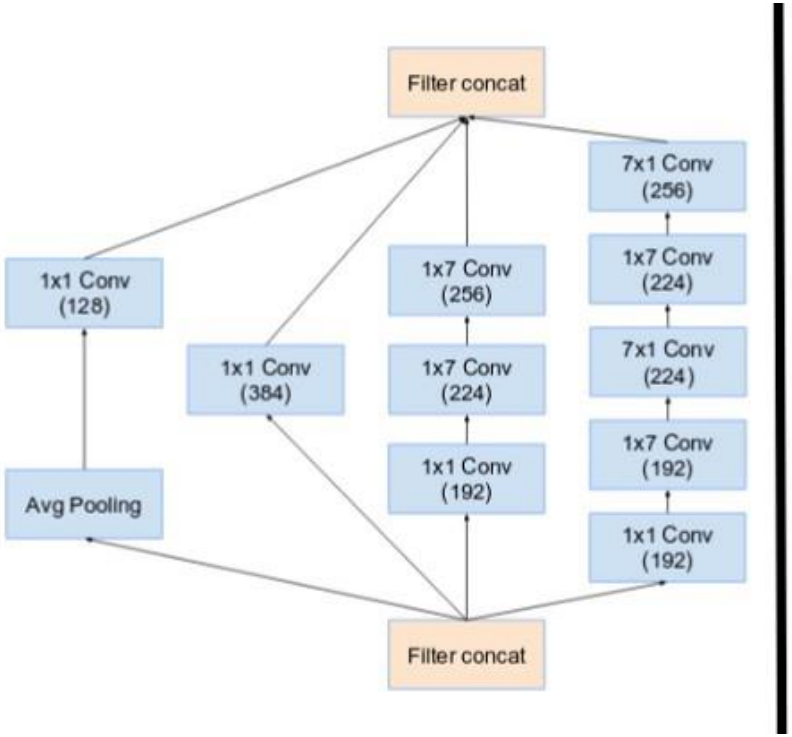
- Deep hierarchies features, Multi level feature representation
 -inception-v4 use more inception modules than Inception-v3.
- Split transform and merge idea Uses asymmetric filters

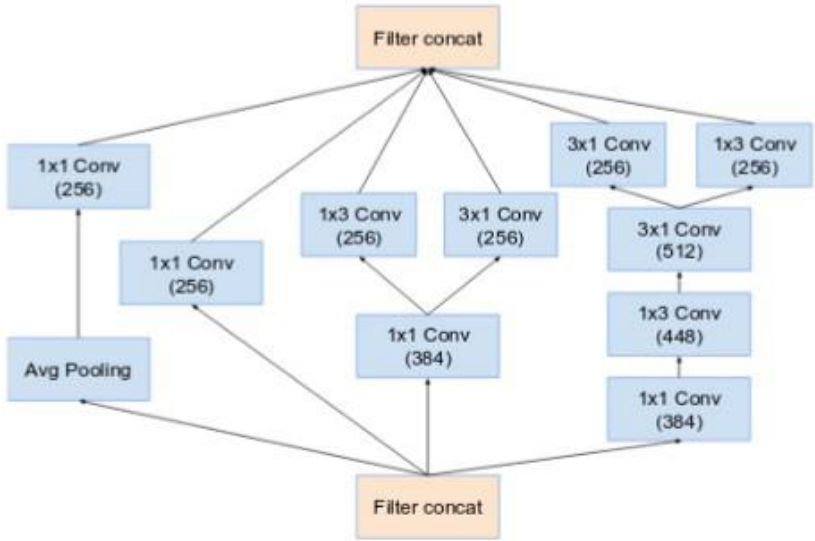


STEM

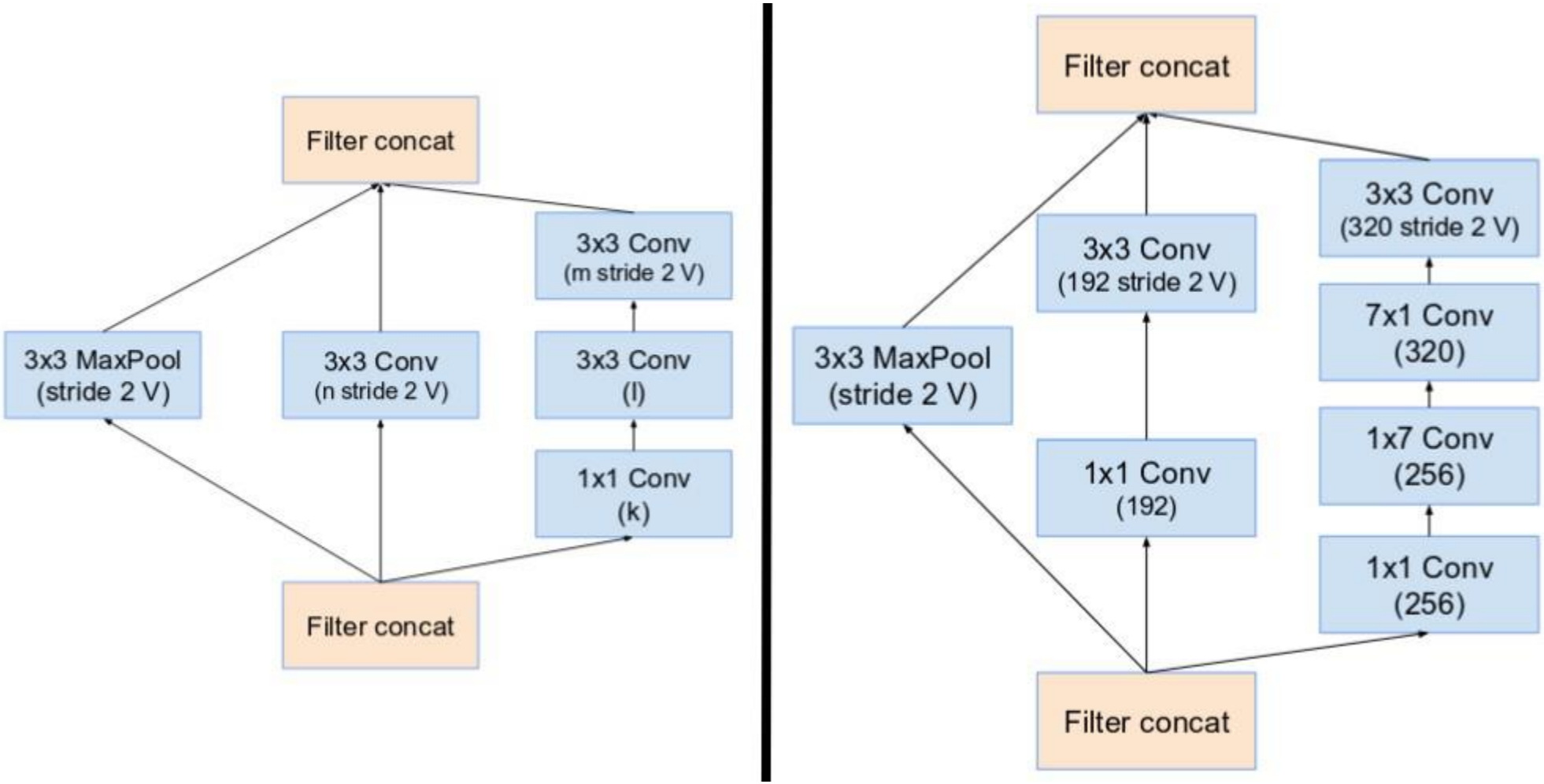

Input
(299x299x3)
299x299x3
3x3 Conv
(32 stride 2 V)
149x149x32
3x3 Conv
(32 V)
147x147x32
3x3 Conv
(64)
147x147x64
3x3 MaxPool
(stride 2 V)
73x73x64
1x1 Conv
(80)
73x73x80
3x3 Conv
(192 V)
71x71x192
3x3 Conv
(256 stride 2 V)
35x35x256







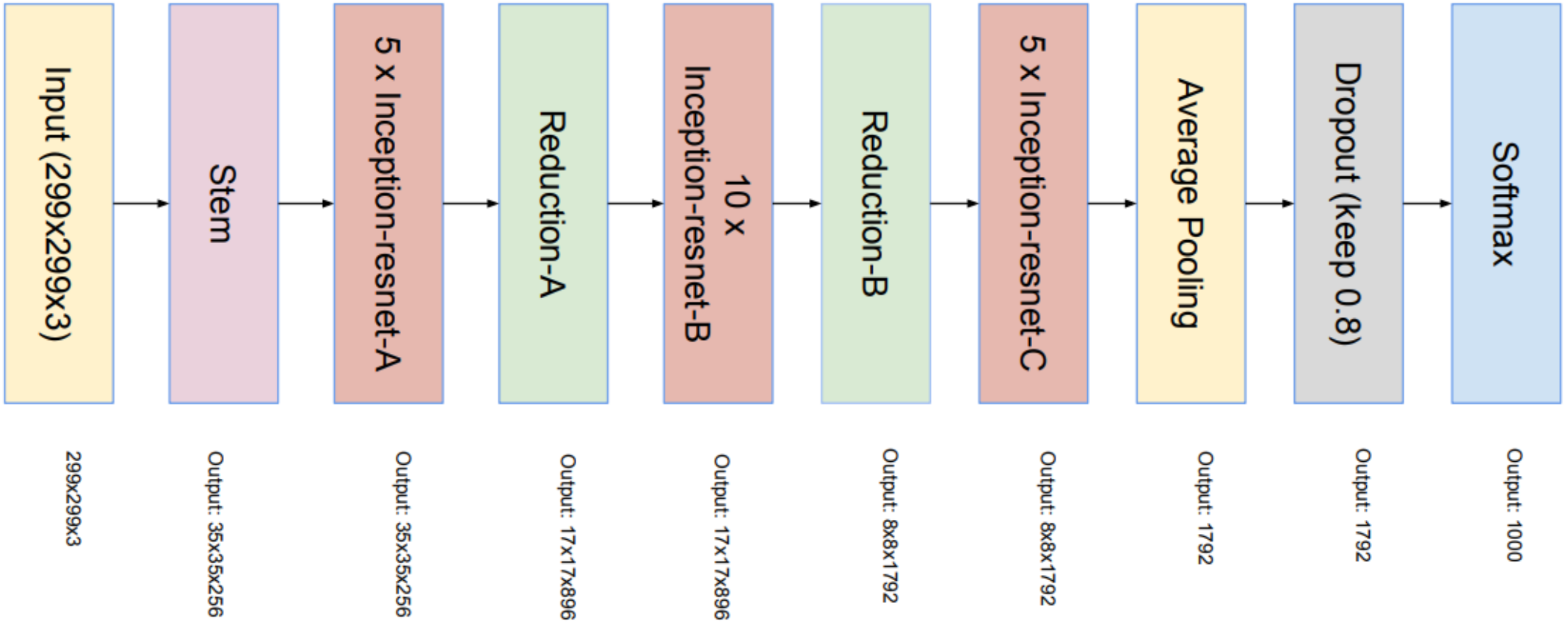
Inception Modules A, B, C of Inception-v4



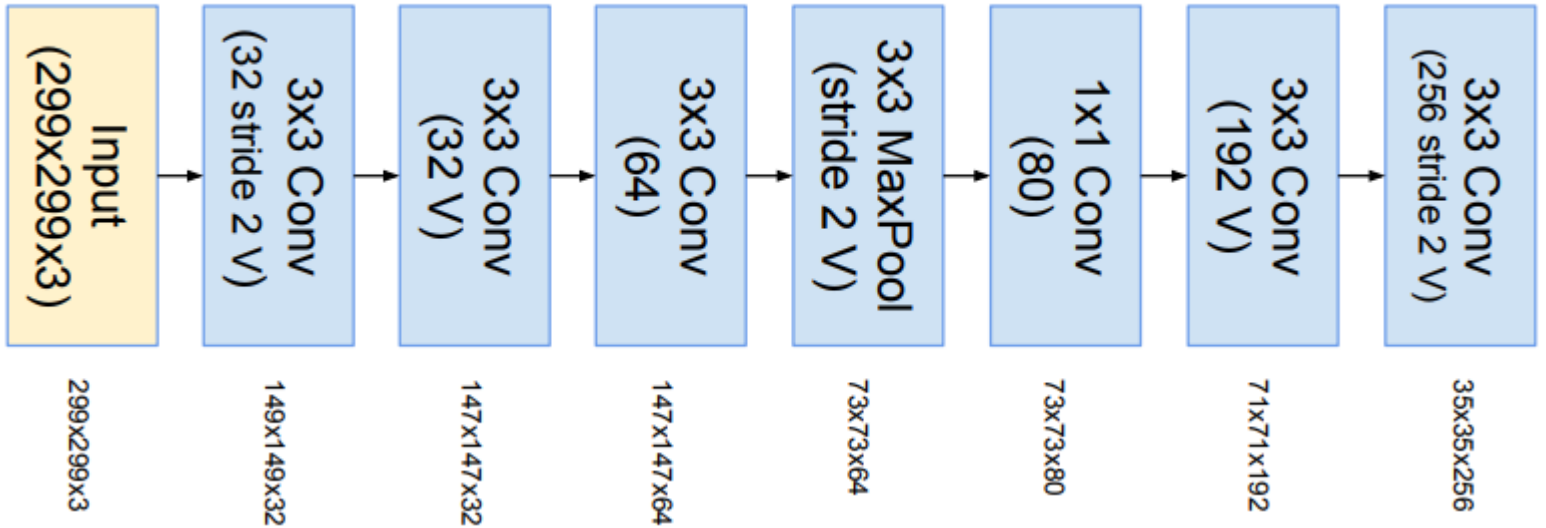
Reduction Blocks A, B of Inception-v4

Inception Resnet

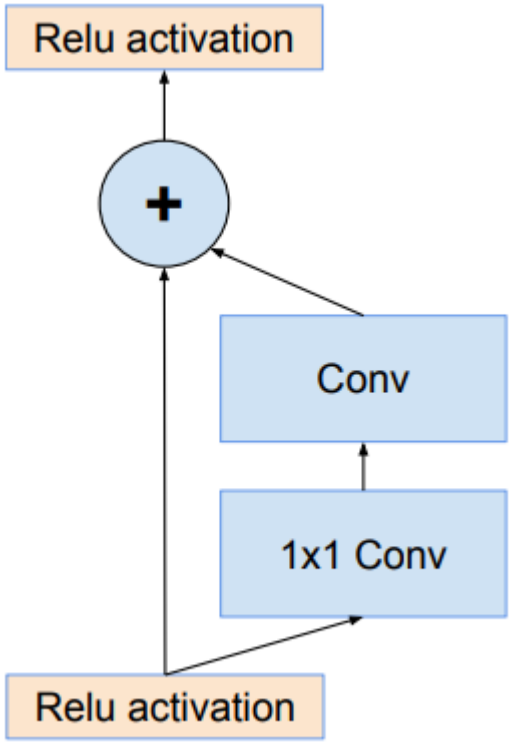
- Uses split transform merge idea and residual links

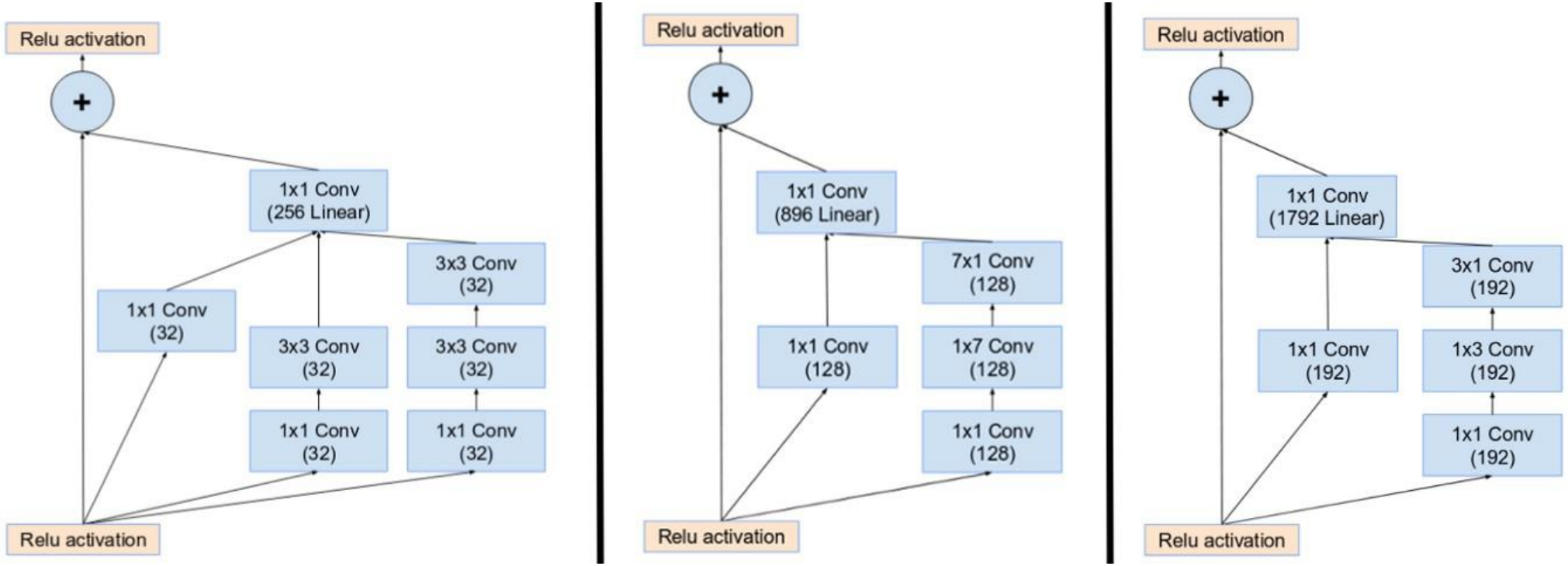


STEM

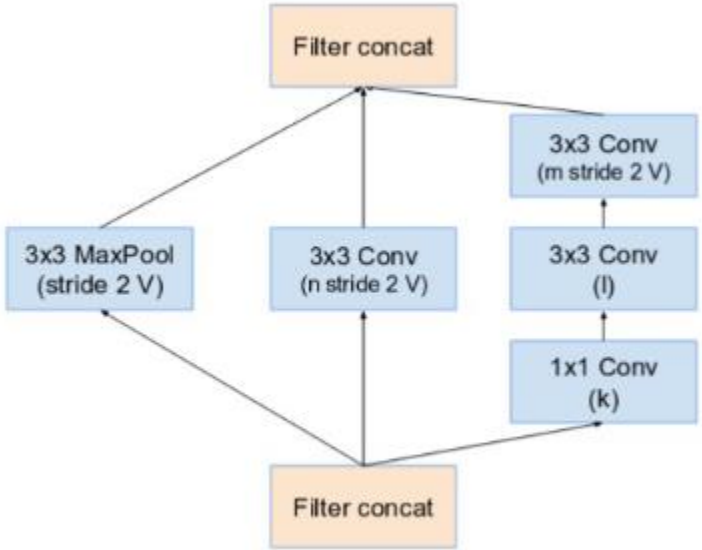


Inception Resnet Conncetion

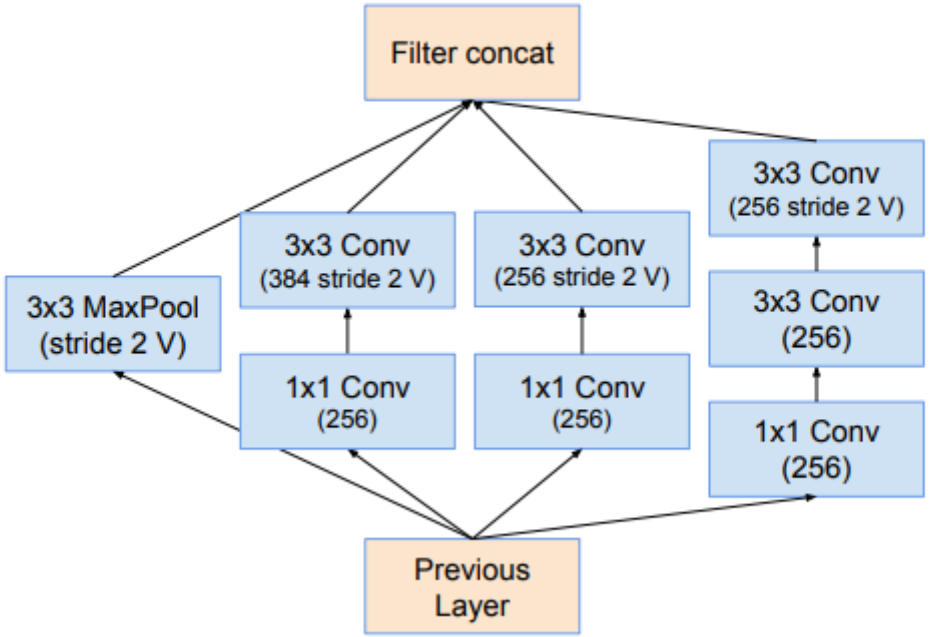




Inception modules A, B, C of Inception ResNet V1



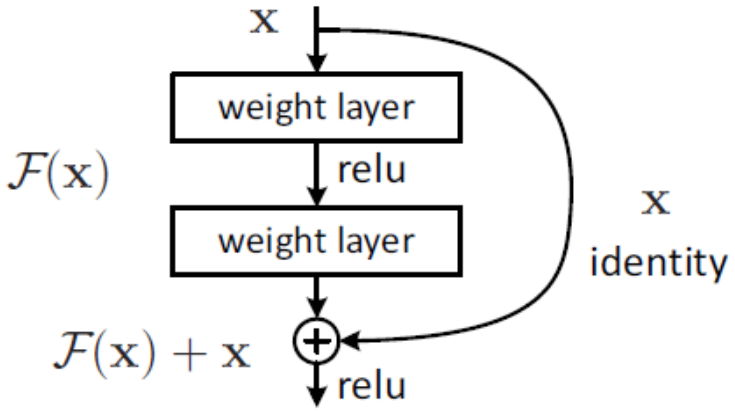
Reduction A schema



Inception ResNet-v1 Reduction Block B

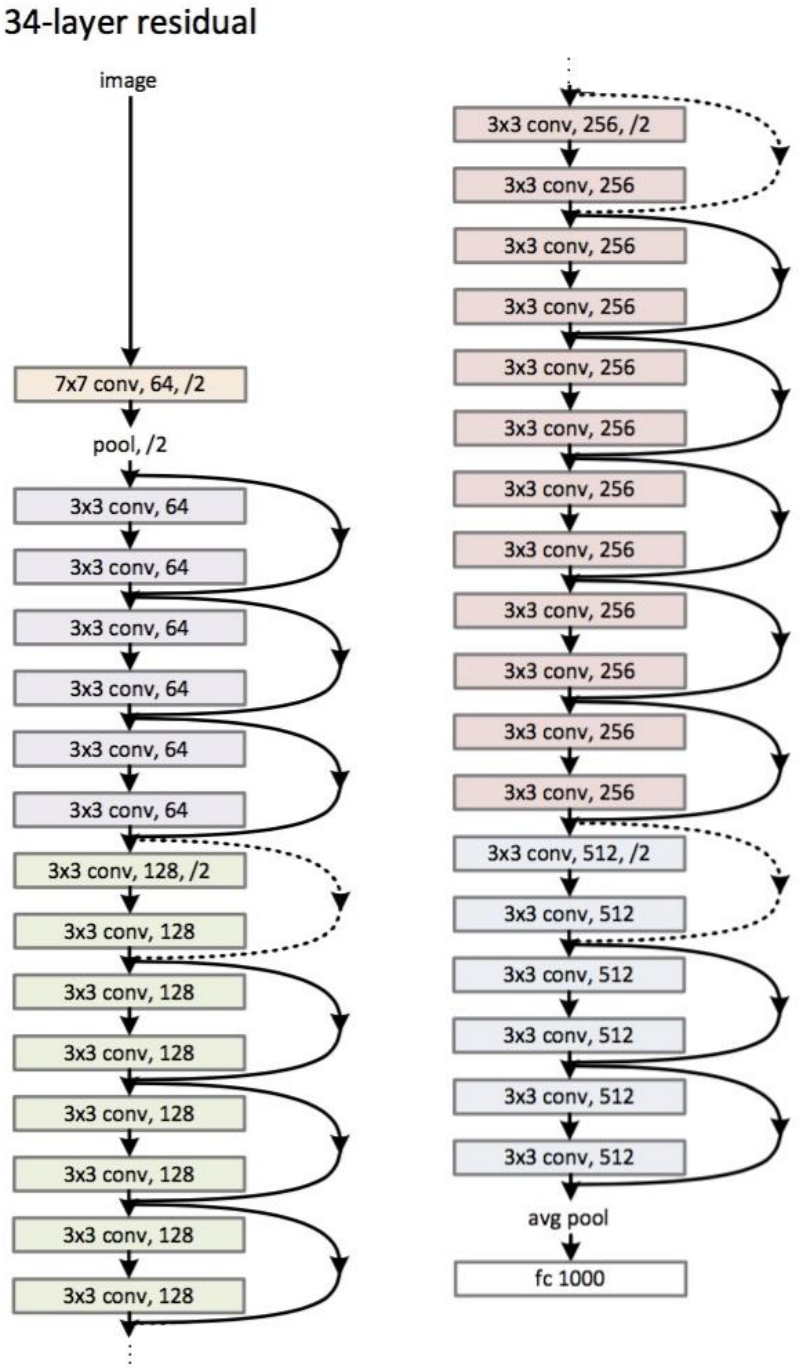
Resnet

- Residual learning
- Identity mapping based skip connections



Residual Blocks

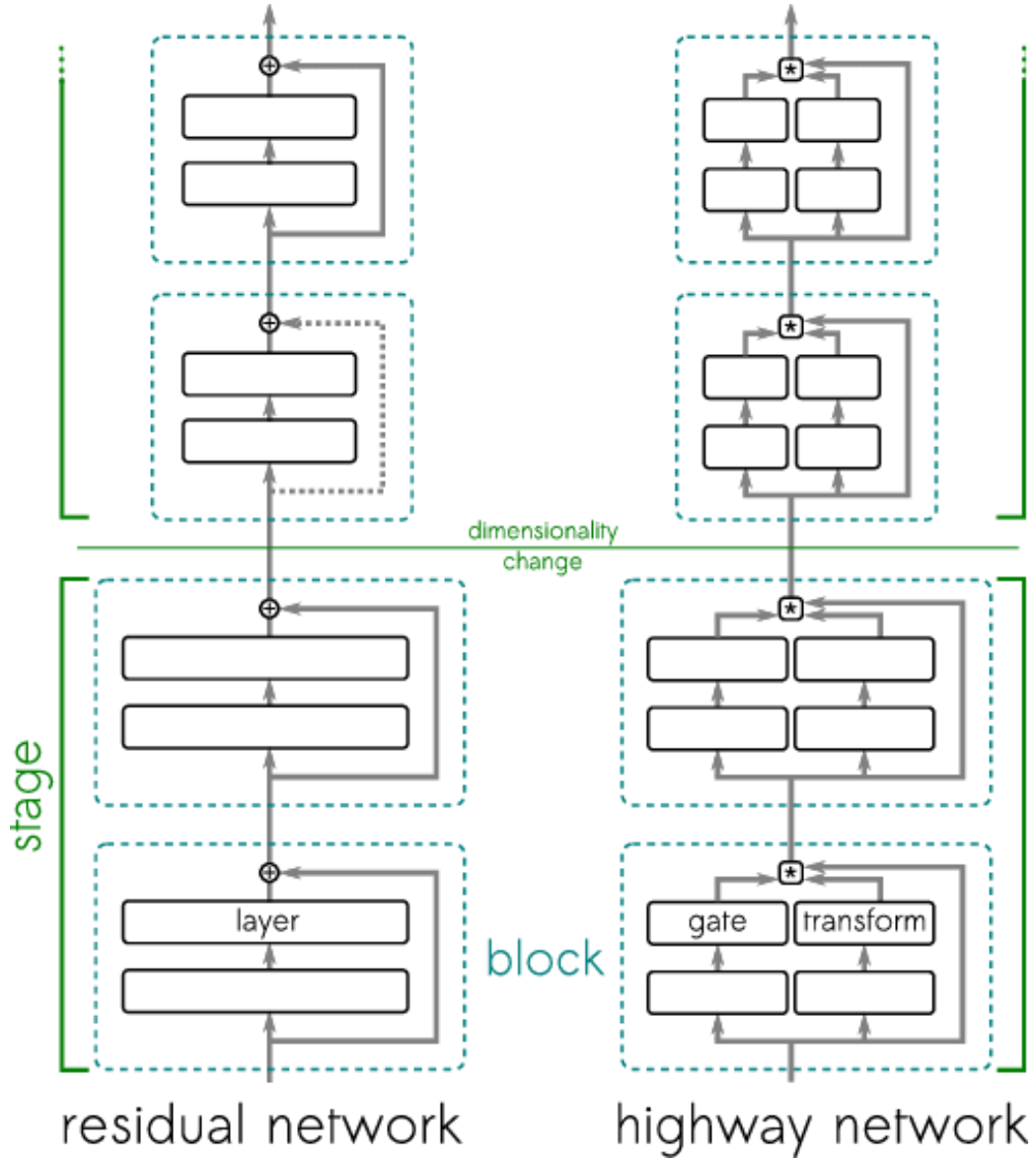
34-layers residual



Highway Network*

- Introduced an idea of Multi-path

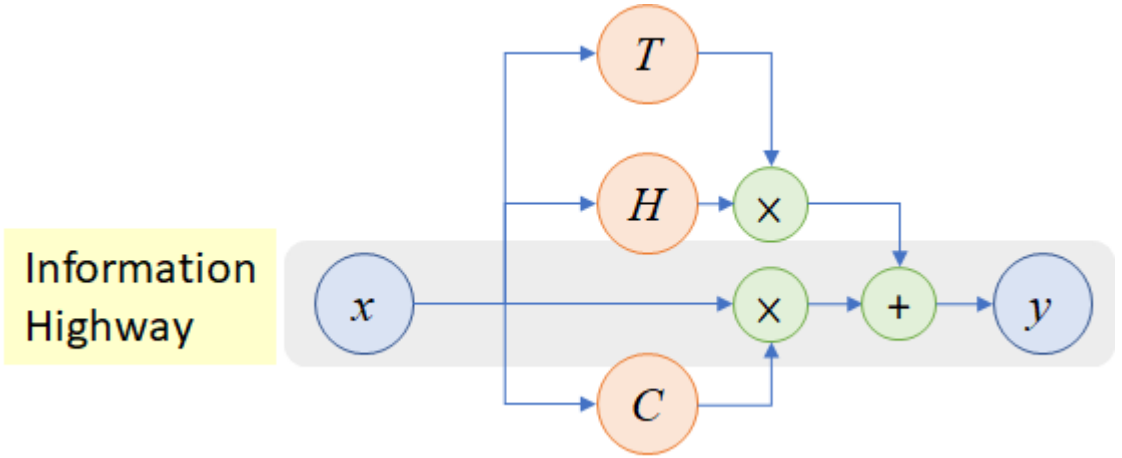
- In machine learning, a highway network is **an approach to optimizing networks and increasing their depth**.
- Highway networks use learned gating mechanisms to regulate information flow, inspired by Long Short-Term Memory (LSTM) recurrent neural networks.
- Highway networks have been used as part of text sequence labeling and speech recognition tasks



*Srivastava, Rupesh Kumar; Greff, Klaus; Schmidhuber, Jürgen (2 May 2015). "Highway Networks". arXiv:1505.00387 [cs.LG].

Continuing about High way networks

The model has two gates in addition to the **$H(W_H, x)$** gate: the transform gate **$T(W_T, x)$** and the carry gate **$C(W_C, x)$**. Those two last gates are non-linear transfer functions (by convention Sigmoid function). The **$H(W_H, x)$** function can be any desired transfer function.
The carry gate is defined as **$C(W_C, x) = 1 - T(W_T, x)$**. While the transform gate is just a gate with a sigmoid transfer function.



The structure of a hidden layer follows the equation:

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot C(x, W_C) = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))$$

The advantage of a Highway Network over the common deep neural networks is that solves or partially prevents the Vanishing gradient problem, thus leading to easier to optimize neural networks.

Table 5b Major challenges associated with implementation of Depth based CNN architectures.

Depth	With the increase in depth, the network can better approximate the target function with a number of nonlinear mappings and improved feature representations. Main challenge faced by deep architectures is the problem of vanishing gradient and negative learning.	
Architecture	**Strength**	**Gaps**
Inception-V3	• Exploited asymmetric filters and bottleneck layer to lessen the computational cost of deep architectures	• Complex architecture design • Lack of homogeneity
Highway Networks	• Introduced training mechanism for deep networks • Used auxiliary connections in addition to direct connections	• Parametric gating mechanism, difficult to implement
Inception-ResNet	• Combined the power of residual learning and inception block	-
Inception-V4	• Deep hierarchies of features, multilevel feature representation	• Slow in learning
ResNet	• Decreased the error rate for deeper networks • Introduced the idea of residual learning • Alleviates the effect of vanishing gradient problem	• A little complex architecture • Degrades information of feature-map in feed forwarding • Over adaption of hyper-parameters for specific task, due to the stacking of same modules

(3) Multi-Path based CNNs

CNNs which improve themselves by using skip connection cross layers to avoid information missing and gradient vanishing

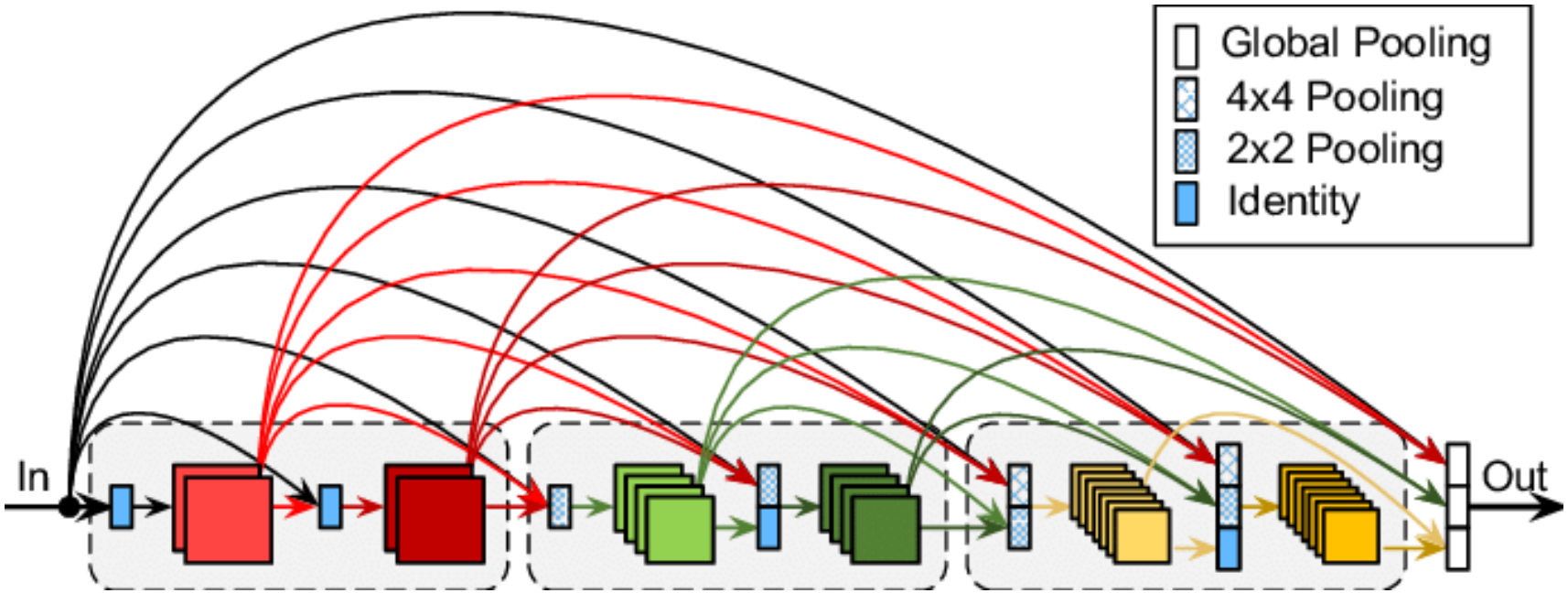
CNNs: Highway Networks ResNet DenseNet

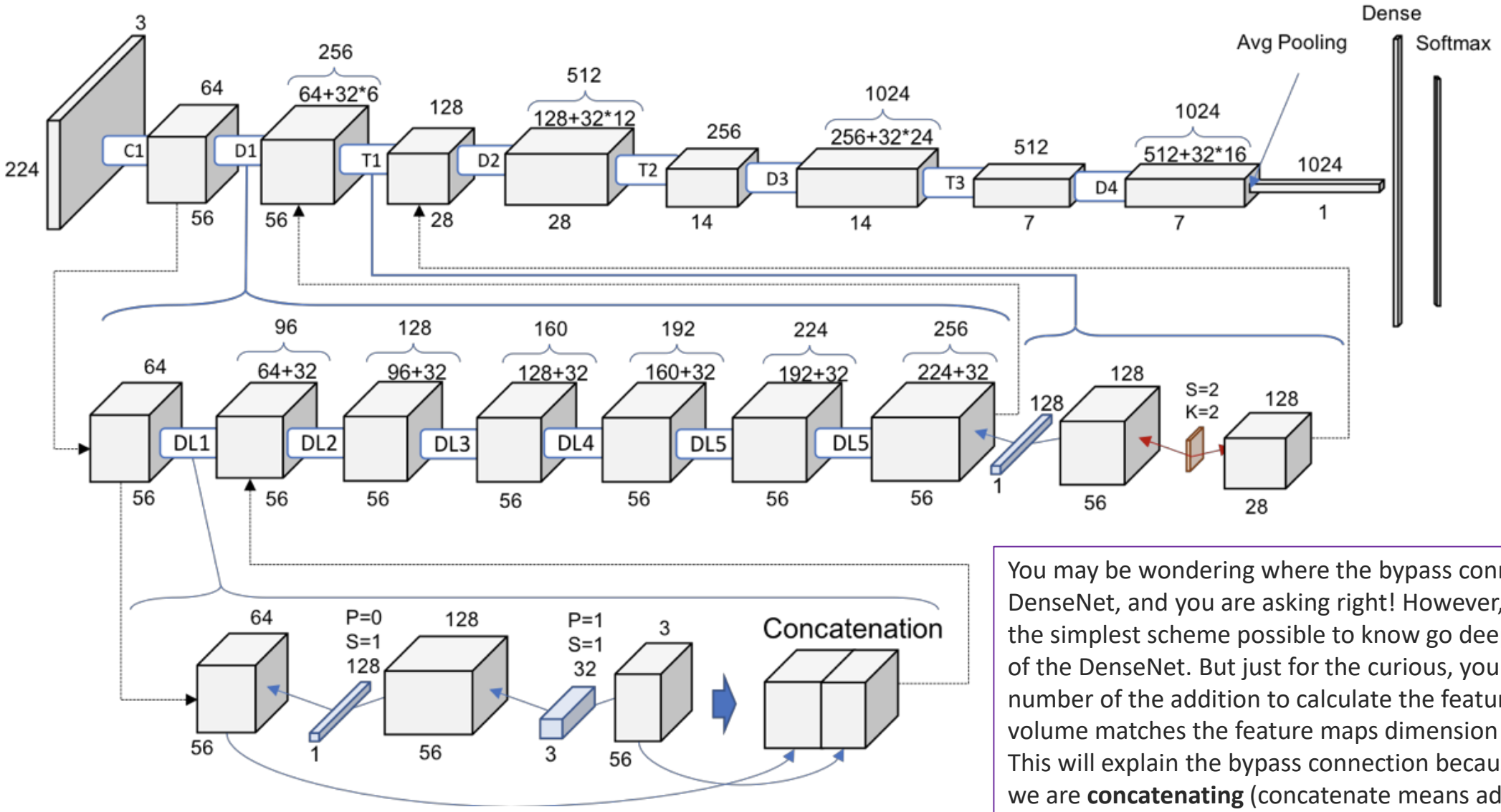
Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76	19	Depth + Multi-Path	(Srivastava et al. 2015a)
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M	ImageNet: 3.6 CIFAR-10: 6.43	152 110	Depth + Multi-Path	(He et al. 2015a)
DenseNet	2017	- Cross-layer information flow	25.6 M 25.6 M 15.3 M 15.3 M	CIFAR-10+: 3.46 CIFAR100+:17.18 CIFAR-10: 5.19 CIFAR-100: 19.64	190 190 250 250	Multi-Path	(Huang et al. 2017)

DenseNet

- Cross-layer information flow

A problem with very deep networks was the problems to train, because of the mentioned flow of information and gradients. DenseNets solve this issue since ***each layer has direct access to the gradients from the loss function*** and the original input image.





Full schematic representation of ResNet-121

You may be wondering where the bypass connections are all over the DenseNet, and you are asking right! However, I just wanted to first draw the simplest scheme possible to know go deeper on the whole structure of the DenseNet. But just for the curious, you can notice how the first number of the addition to calculate the feature maps of every new volume matches the feature maps dimension of the previous volume. This will explain the bypass connection because it precisely means that we are **concatenating** (concatenate means add dimension, but not add values!) **new information to the previous volume**, which is being **reused**.

Table 5c Major challenges associated with implementation of Multi-Path based CNN architectures.

Multi-Path	Shortcut paths provides the option to skip some layers. Different types of the shortcut connections used in literature are zero padded, projection, dropout, 1x1 connections, etc.	
Architecture	**Strength**	**Gaps**
Highway Networks	• Mitigates the limitations of deep networks by introducing cross layer connectivity.	• Gates are data dependent and thus may become parameter expensive
ResNet	• Use of identity based skip connections to enable cross layer connectivity • Information flow gates are data independent and parameter free • Can easily pass the signal in both directions, forward and backward	• Many layers may contribute very little or no information • Relearning of redundant feature-maps may happen
DenseNet	• Introduced depth or cross-layer dimension • Ensures maximum data flow between the layers in the network • Avoid relearning of redundant feature-maps • Low and high level both features are accessible to decision layers	• Large increase in parameters due to increase in number of feature-maps at each layer

Width based Multi-Connection CNNs

CNNs which improve themselves by making parallel use of multiple processing units within a layer
Parallel and homogenous topology in each block

CNNs: Wide ResNet Pyramidal Net Xception ResNeXt Inception Family

Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
WideResNet	2016	- Width is increased and depth is decreased	36.5 M	CIFAR-10: 3.89 CIFAR-100: 18.85	28 -	Width	(Zagoruyko and Komodakis 2016)
PyramidalNet	2017	- Increases width gradually per unit	116.4 M 27.0 M 27.0 M	ImageNet: 4.7 CIFAR-10: 3.48 CIFAR-100: 17.01	200 164 164	Width	(Han et al. 2017)
Xception	2017	- Depth wise convolution followed by point wise convolution	22.8 M	ImageNet: 0.055	126	Width	(Chollet 2017)
ResNeXt	2017	- Cardinality - Homogeneous topology - Grouped convolution	68.1 M	CIFAR-10: 3.58 CIFAR-100: 17.31 ImageNet: 4.4	29 - 101	Width	(Xie et al. 2017)
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6	159	Depth + Width	(Szegedy et al. 2016b)
Inception-V4	2016	- Split transform and merge idea Uses asymmetric filters	35 M	ImageNet: 4.01	70	Depth +Width	(Szegedy et al. 2016a)
Inception-ResNet	2016	- Uses split transform merge idea and residual links	55.8M	ImageNet: 3.52	572	Depth + Width + Multi-Path	(Szegedy et al. 2016a)

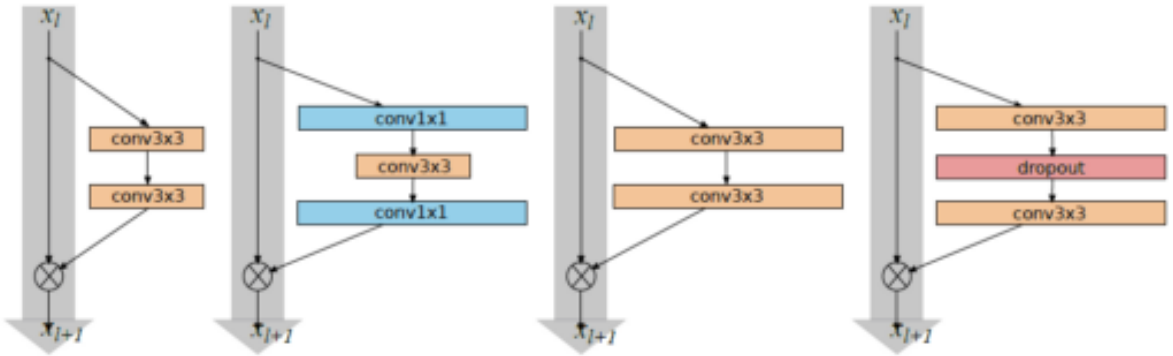
Wide ResNet

a variant of Resent to reduce the depth and increase the width

In WRNs, plenty of parameters are tested such as the design of the ResNet block, how deep (deepening factor *l*) and how wide (widening factor *k*) within the ResNet block.

When *k*=1, it has the same width of *ResNet*. While *k*>1, it is *k* time wider than *ResNet*.

WRN-*d*-*k*: means the WRN has the depth of *d* and with widening factor *k*.

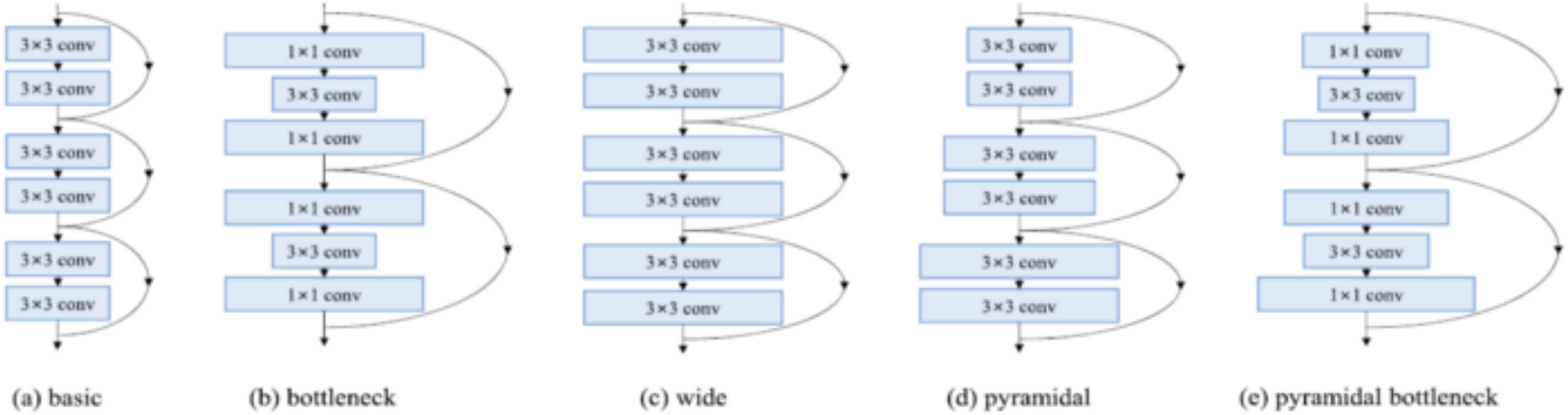


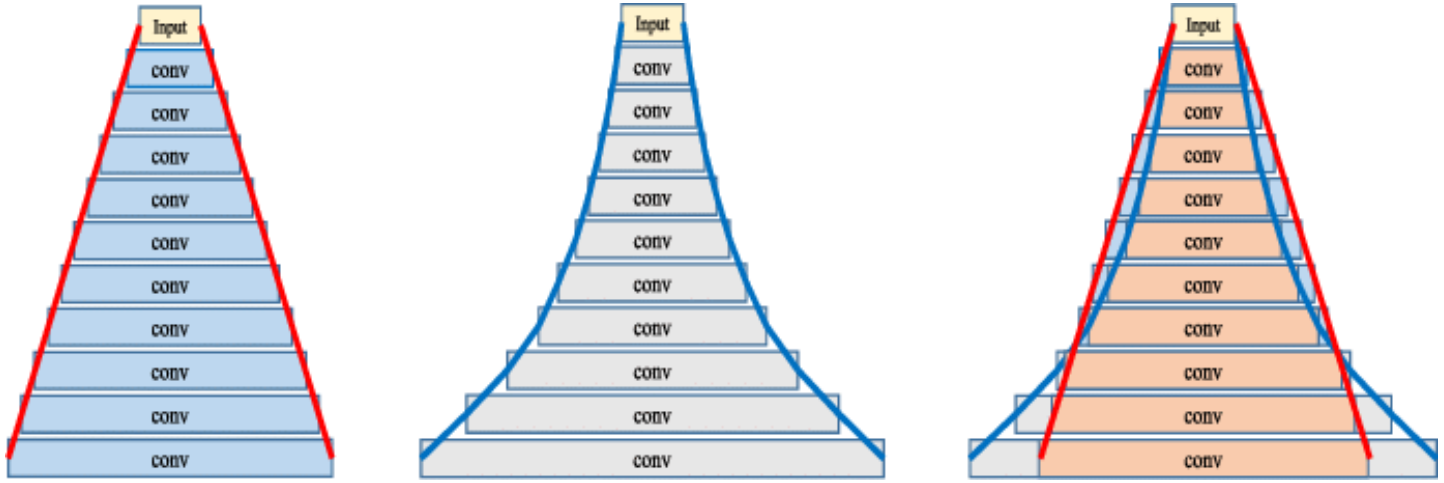
(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

	depth-*k*	# params	CIFAR-10	CIFAR-100
NIN [20]			8.81	35.67
DSN [19]			8.22	34.57
FitNet [24]			8.39	35.04
Highway [28]			7.72	32.39
ELU [5]			6.55	24.28
original-ResNet[11]	110	1.7M	6.43	25.16
	1202	10.2M	7.93	27.82
stoc-depth[14]	110	1.7M	5.23	24.58
	1202	10.2M	4.91	-
pre-act-ResNet[13]	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.9M	4.53	21.18
	16-8	11.0M	4.27	20.43
	28-10	36.5M	**4.00**	**19.25**

Pyramidal Net

Pyramidal Net increases the width gradually per residual unit.





Xception

- Depth wise convolution followed by point wise convolution

Xception modified the original inception block by making it wider and replacing the different spatial dimensions (1x1, 5x5, 3x3) with a single dimension (3x3) followed by a 1x1 convolution to regulate computational complexity.

Xception makes the network computationally efficient by decoupling spatial and feature-map (channel) correlation,

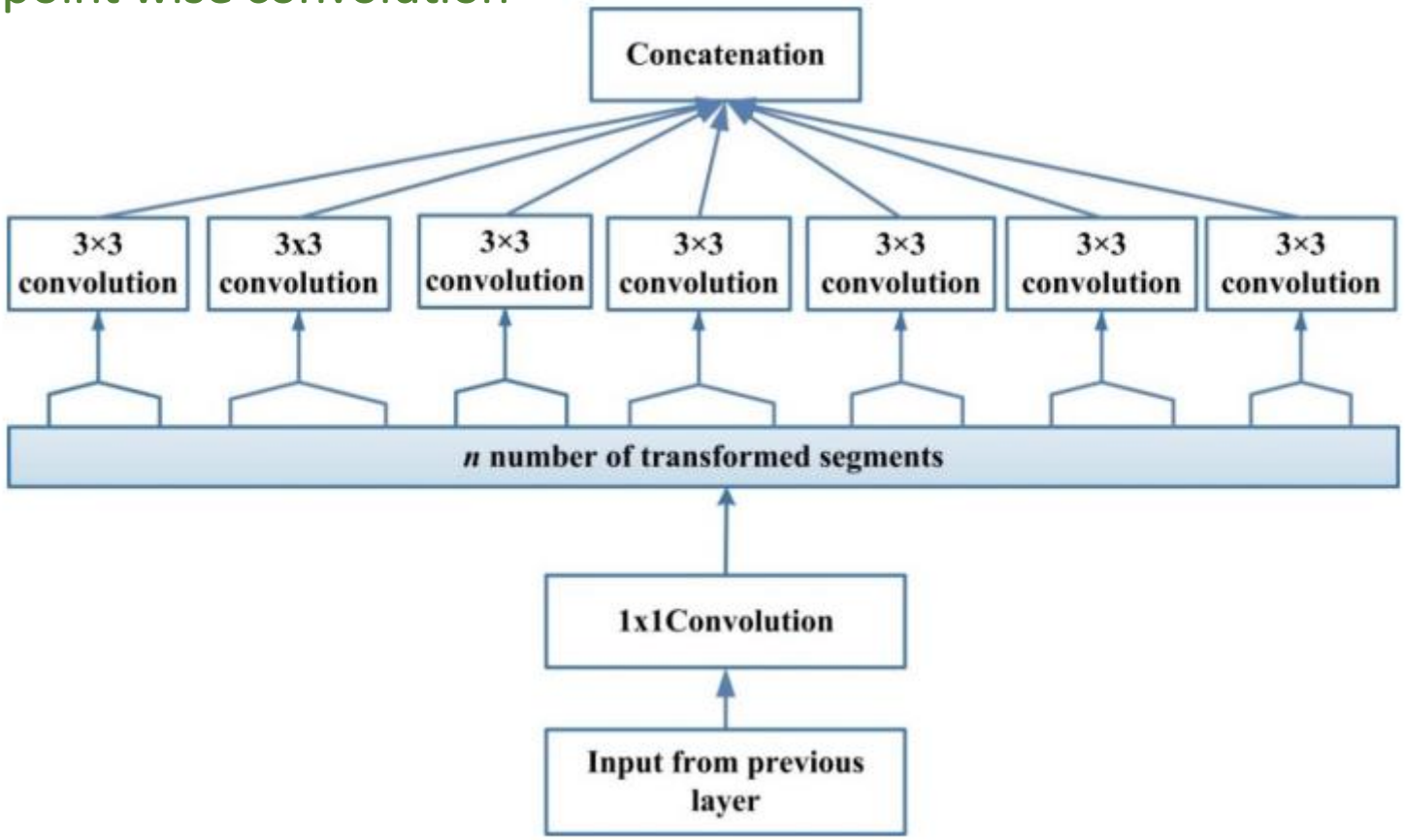


Fig. 8 Xception building block and its *n* sets of transformation.

ResNeXt

-Cardinality
- Homogeneous topology
- Grouped convolution

ResNeXt, also known as Aggregated Residual Transform Network, is an improvement over the Inception Network (Xie et al. 2017).
Xie et al. exploited the concept of the split, transform, and merge in a powerful but simple way by introducing a new term; cardinality (Szegedy et al. 2015).
Cardinality is an additional dimension, which refers to the size of the set of transformations (Han et al. 2018; Sharma and Muttoo 2018).

Refer to the following figure, the architecture includes 32 same topology blocks so the value of cardinality is 32.

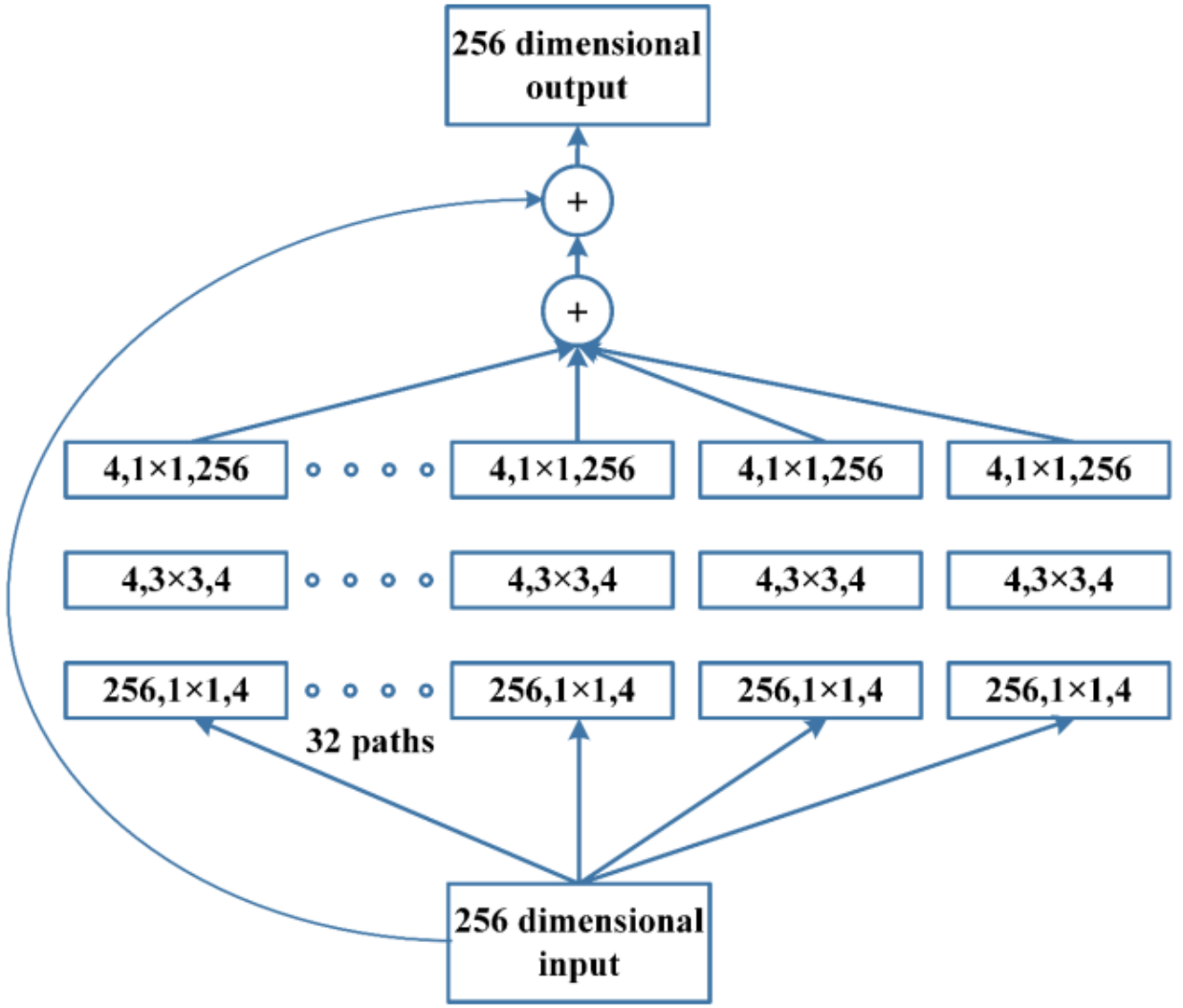


Fig. 9 ResNeXt building block showing the different paths of transformation.

Table 5d Major challenges associated with implementation of Width based CNN architectures.

Width	Earlier, it was assumed that to improve accuracy, the number of layers have to be increased. However, by increasing the number of layers, the vanishing gradient problem arises and training might get slow. So, the concept of widening a layer was also investigated.	
Architecture	**Strength**	**Gaps**
Wide ResNet	• Showes the effectiveness of parallel use of transformations by increasing the width of ResNet and decreasing its depth • Enables feature reuse • Have shown that dropouts between the convolutional layer are more effective	• Over fitting may occur • More parameters than thin deep networks
Pyramidal Net	• Introduces the idea of increasing the width gradually per unit • Avoids rapid information loss • Covers all possible locations instead of maintaining the same dimension till last unit	• High spatial and time complexity • May become quite complex, if layers are substantially increased
Xception	• Introduce the concept that learning across 2D followed by 1 D is easier than to learn filters in 3 D space • Depth-wise separable convolution is introduced • Use of cardinality to learn good abstractions	• High computational cost
Inception	• Varying size filters inside inception module increases the output of the intermediate layers • Varying size filters are helpful to capture the diversity in high-detail images	• Increase in space and time complexity
ResNeXt	• Introduced cardinality to avail diverse transformations at each layer • Easy parameter customization due to homogenous topology • Uses grouped convolution	• High computational cost

(4) Feature-Map (Channel*FMap*) Exploitation based CNNs

CNNs which improve themselves by adding a block for the selection of feature-maps (channels)

CNNs: Squeeze and Excitation Network, Competitive Squeeze and Excitation Networks

Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Squeeze & Excitation Networks	2017	- Models interdependencies between feature-maps	27.5 M	ImageNet: 2.3	152	Feature-Map Exploitation	(Hu et al. 2018a)
Competitive Squeeze & Excitation Network CMPE-SE-WRN-28	2018	- Residual and identity mappings both are used for rescaling the feature-map	36.92 M 36.90 M	CIFAR-10: 3.58 CIFAR-100: 18.47	152 152	Feature-Map Exploitation	(Hu et al. 2018b)

(5) Squeeze and Excitation Network

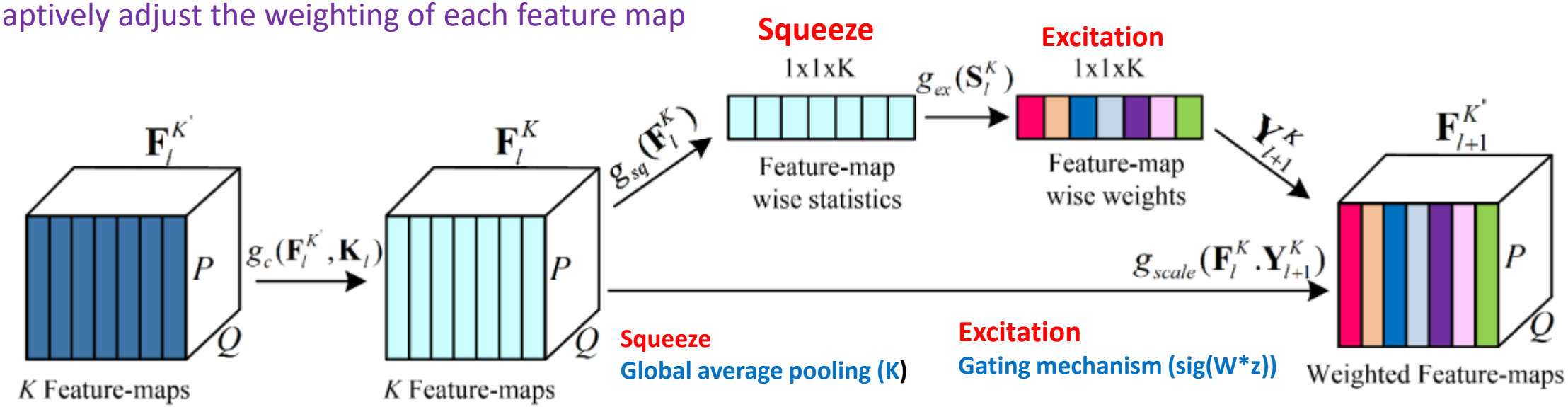
Model interdependencies between feature maps

Squeeze and Excitation block showing the computation of masks for the recalibration of feature-

maps that are commonly known as channels in literature

Squeeze-and-Excitation Networks (SENets) **introduce a building block for CNNs that improves channel interdependencies at almost no computational cost**. ...

Let's add parameters to each channel of a convolutional block so that the network can adaptively adjust the weighting of each feature map



Competitive Squeeze and Excitation Networks

*Hu et al. 2018b

- The authors used the idea of SEblock to improve the learning of deep residual networks .
- SE-Network recalibrates the feature-maps based upon their contribution in class discrimination. However, the main concern with SE-Net is that in ResNet, it only considers the residual information for determining the weight of each feature-map (Hu et al. 2018a).
- This minimizes the impact of SEblock and makes ResNet information redundant.
- Hu et al. addressed this problem by generating feature-map wise motifs (statistics) from both residual and identity mapping based feature-maps.
- In this regard, global representation of feature-maps is generated using global average pooling
  operation, whereas relevance of feature-maps is estimated by establishing competition between feature descriptors of residual and identity mappings.
- This phenomena is termed as inner imaging. CMPE-SE block not only models the relationship between residual feature-maps but also maps their relation with identity feature-map.

Competitive Squeeze-Excitation Architecture for Residual block

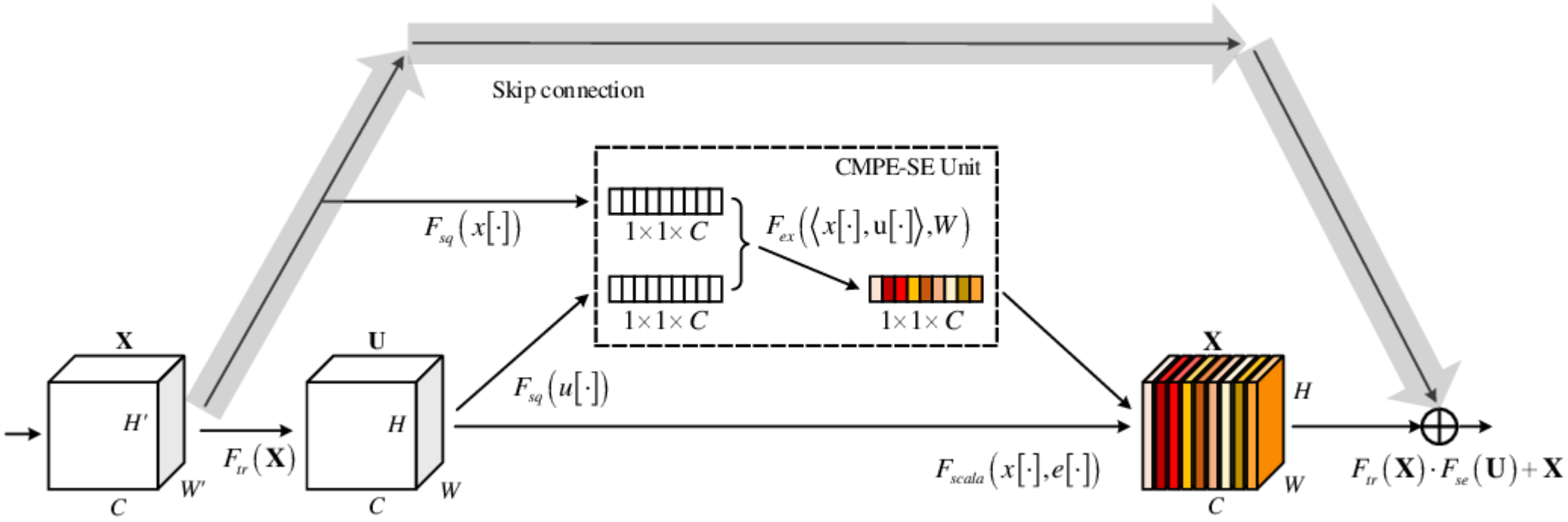


Table 5e Major challenges associated with implementation of Feature-Map exploitation based CNN architectures.

Feature-Map Selection	As the deep learning topology is extended, more and more features maps are generated at each step. Many of the Feature-maps might be important for classification task, others might redundant or less important. Hence, feature-map selection is another important dimension in deep learning architectures.	
Architecture	**Strength**	**Gaps**
Squeeze and Excitation Network	• It is a block-based concept • Introduced a generic block that can be added easily in any CNN model due to its simplicity • Squeezes less important features and vice versa	• In ResNet, it only considers the residual information for determining the weight of each channel
Competitive Squeeze and Excitation Networks	• Uses feature-map wise statistics from both residual and identity mapping based features • Makes a competition between residual and identity feature-maps	• Doesn't support the concept of attention

(6) Channel(*Input*) Exploitation based CNNs

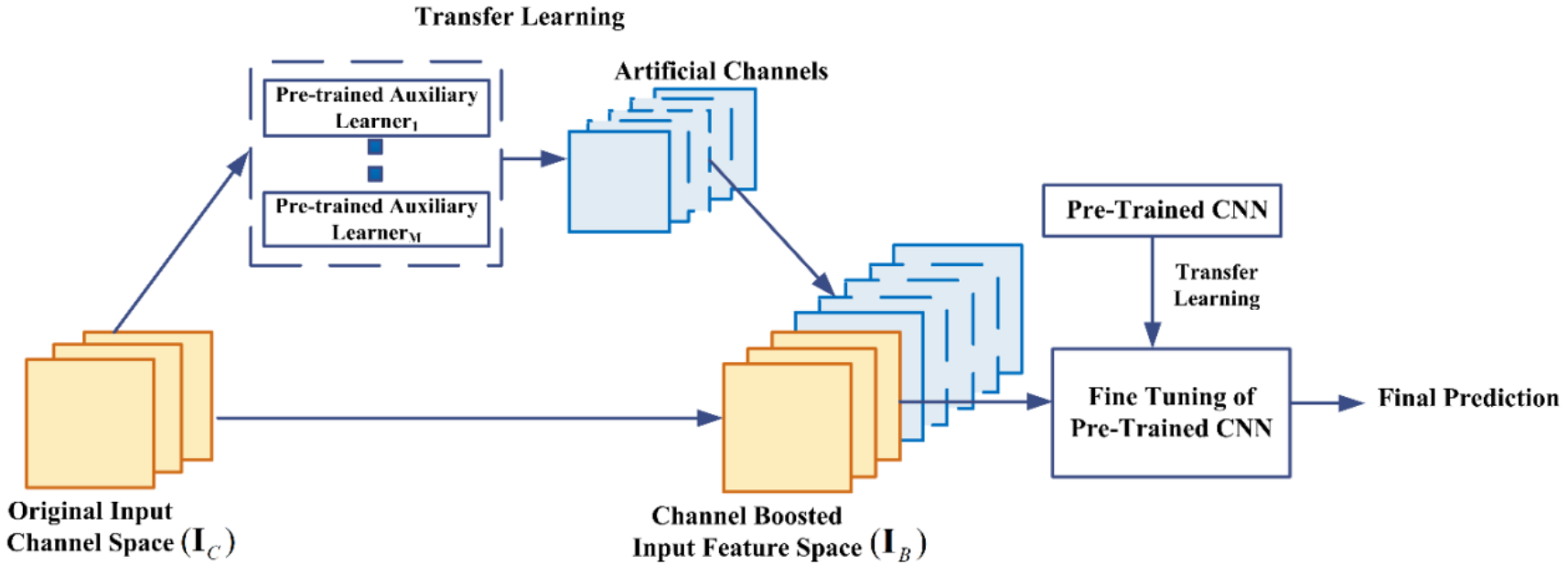
CNNs which improve themselves by using auxiliary learners for channel boosting (input channel dimension)

CNNs: Channel Boosted CNN using TL

Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Channel Boosted CNN	2018	- Boosting of original channels with additional information rich generated artificial channels	-	-	-	Channel Boosting	(Khan et al. 2018a)

Channel Boosted CNN using TL (transfer Learning)

Basic architecture of CB-CNN showing the deep auxiliary learners for creating artificial channels



In the proposed methodology, **a deep CNN is boosted by various channels available through TL from already trained Deep Neural Networks, in addition to its original channel**. The deep architecture of CNN then exploits the original and boosted channels down the stream for learning discriminative patterns.

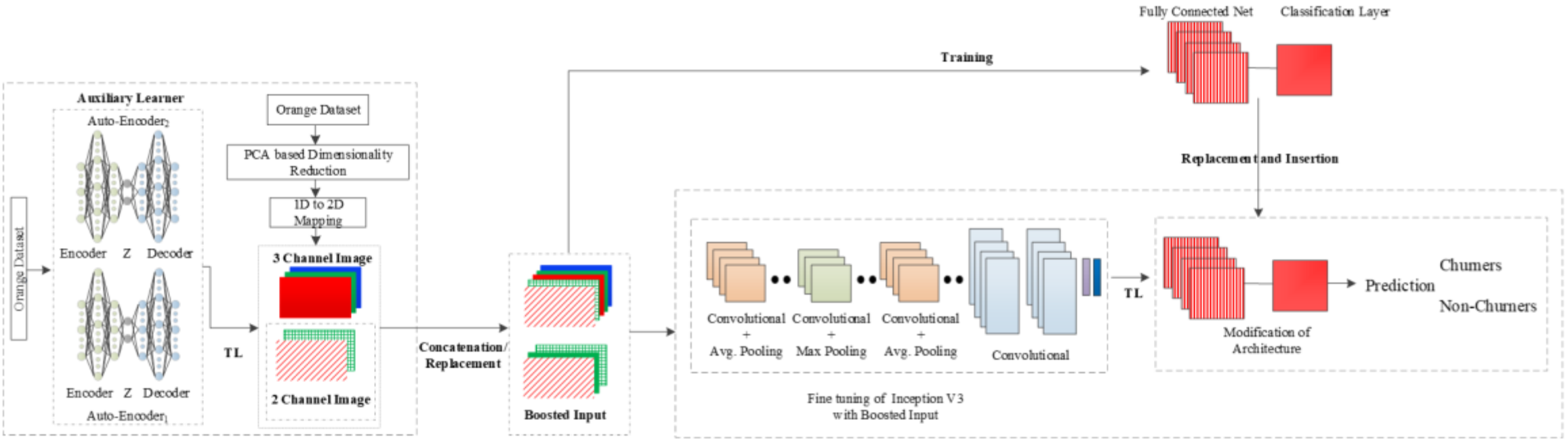


Fig. 4. Details of the working of the proposed CB-CNN. *CB-CNN1* is trained by using Boosted Input$_1$. Boosted Input$_1$ is comprised of three channels that is generated by replacing the input channels of original feature space with two auxiliary channels. Whereas, Boosted Input$_2$ is used to train *CB-CNN2* that is generated by concatenating original channel space with three auxiliary channels.

Table 5f Major challenges associated with implementation of Channel Boosting based CNN architectures.

Channel Boosting	The learning of CNN also relies on the input representation. The lack of diversity and absence of class discernable information in the input may affect CNN performance. For this purpose, the concept of channel boosting (input channel dimension) using auxiliary learners is introduced in CNN to boost the representation of the network (Khan et al. 2018a).	
Architecture	**Strength**	**Gaps**
Channel Boosted CNN using Transfer Learning	• It boosts the number of input channels for improving the representational capacity of the network • Inductive Transfer Learning is used in a novel way to build a boosted input representation for CNN	• Increases in computational load may happen due to the generation of auxiliary channels

(7) Attention based CNNs

CNNs which use attention mechanism to pay attention to context-relevant parts

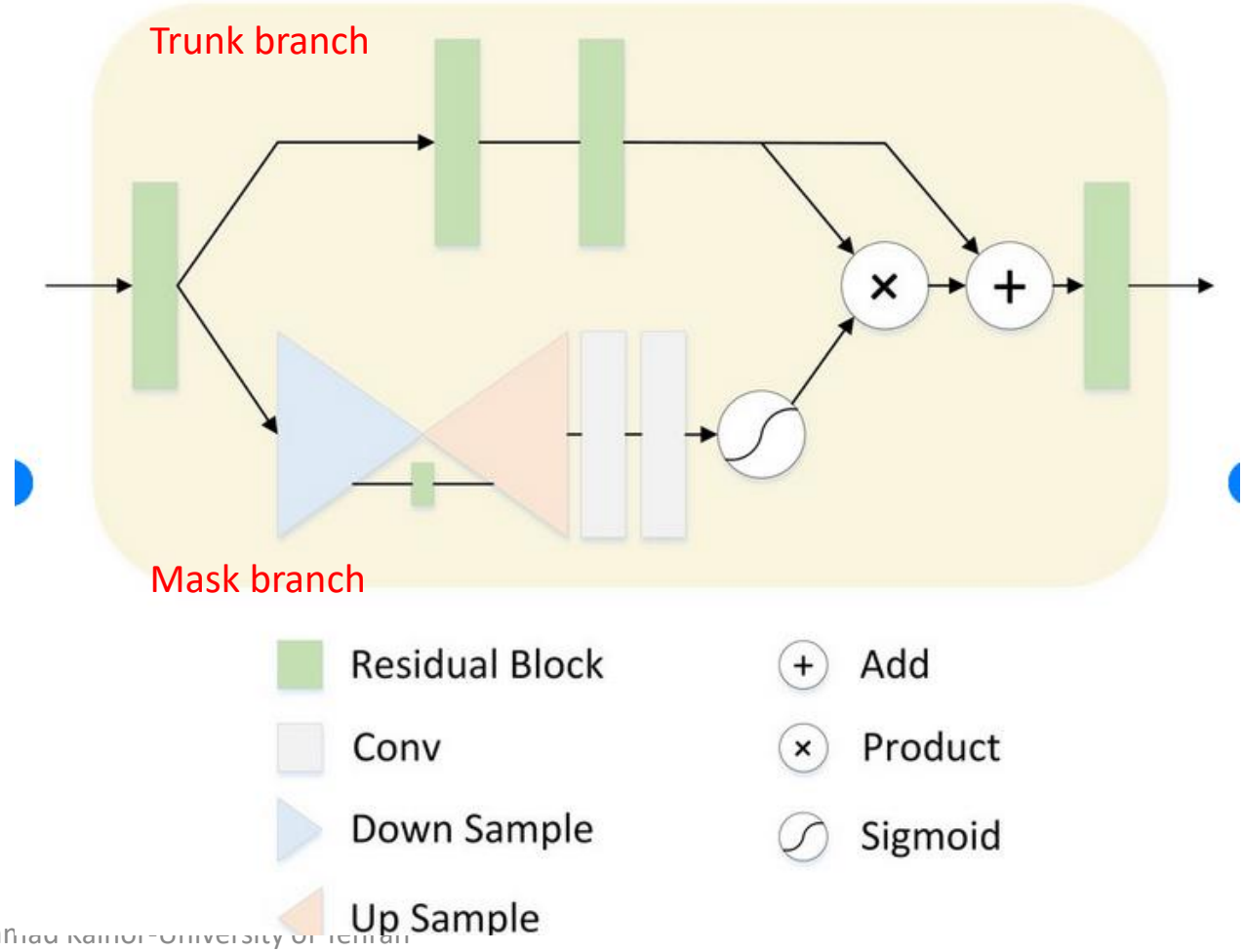
CNNs: Residual Attention Neural Network Convolutional Block Attention Module Concurrent Spatial and Channel Excitation Mechanism

Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Residual Attention Neural Network	2017	- Introduced an attention mechanism	8.6 M	CIFAR-10: 3.90 CIFAR-100: 20.4 ImageNet: 4.8	452	Attention	(Wang et al. 2017a)
Convolutional Block Attention Module (ResNeXt101 (32x4d) + CBAM)	2018	- Exploits both spatial and feature-map information	48.96 M	ImageNet: 5.59	101	Attention	(Woo et al. 2018)
Concurrent Spatial & Channel Excitation Mechanism	2018	- Spatial attention - Feature-map attention - Concurrent placement of spatial and channel attention	-	MALC: 0.12 Visceral: 0.09	-	Attention	(Roy et al. 2018)

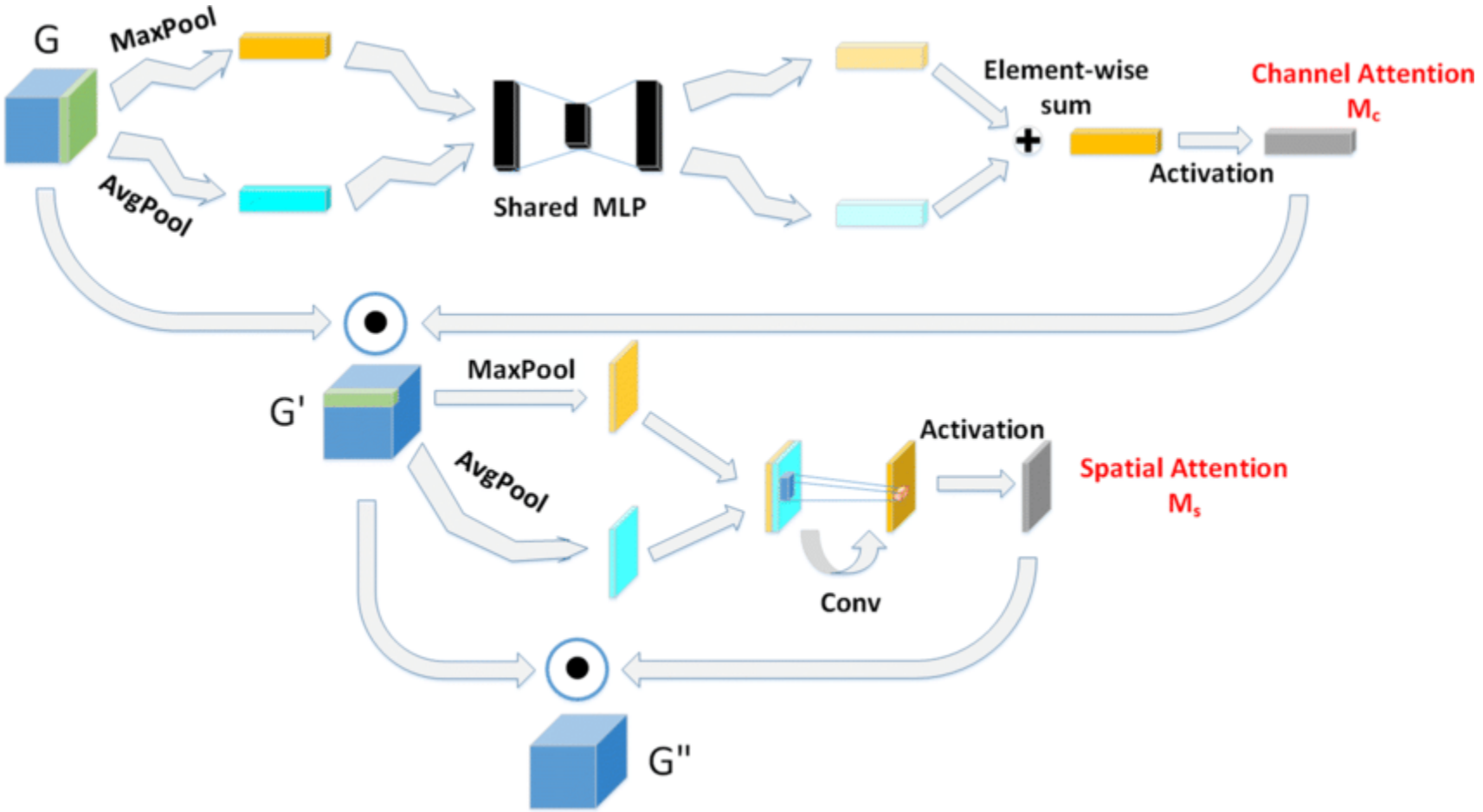
Residual Attention Neural Network*[Sik-Ho Tsang](#) Apr 11, 2019

Multiple attention module is stacked to generate attention-aware features. Attention residual learning is used for very deep network.

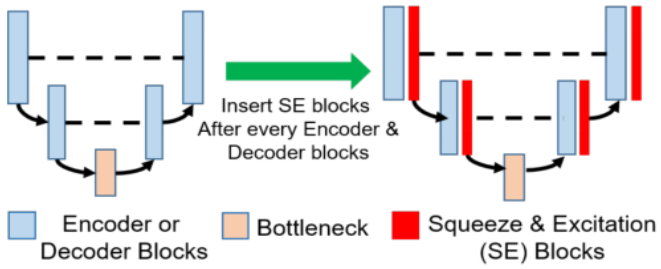
Attention module. The top branch is the trunk branch that consists of two residual blocks. The bottom branch is the mask branch.



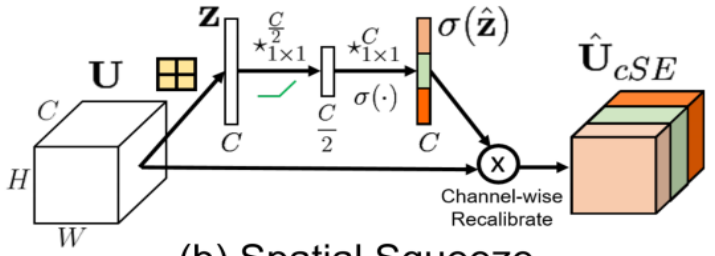
CBAM: Convolutional Block Attention Module



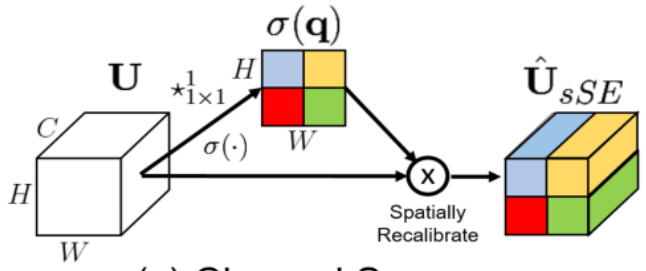
Concurrent Spatial and Channel Excitation Mechanism



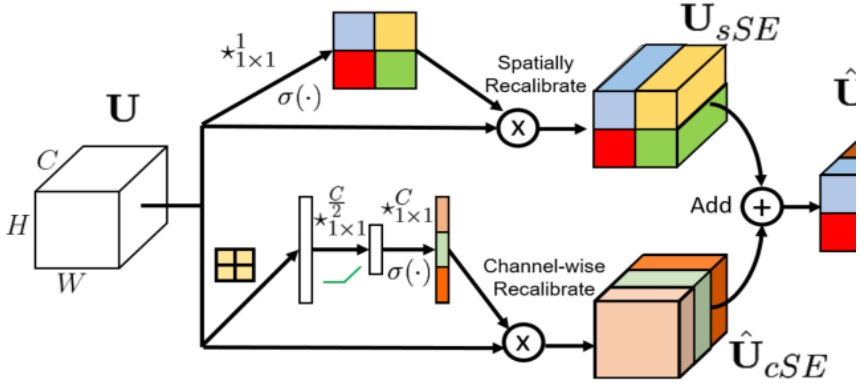
(a) SE block in F-CNN



(b) Spatial Squeeze
and Channel Excitation (cSE)



(c) Channel Squeeze
and Spatial Excitation (sSE)



(d) Concurrent Spatial and Channel
Squeeze and Channel Excitation (scSE)

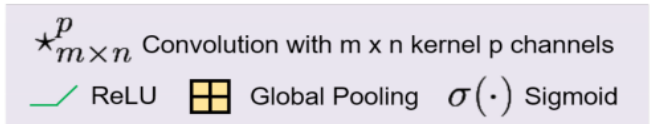


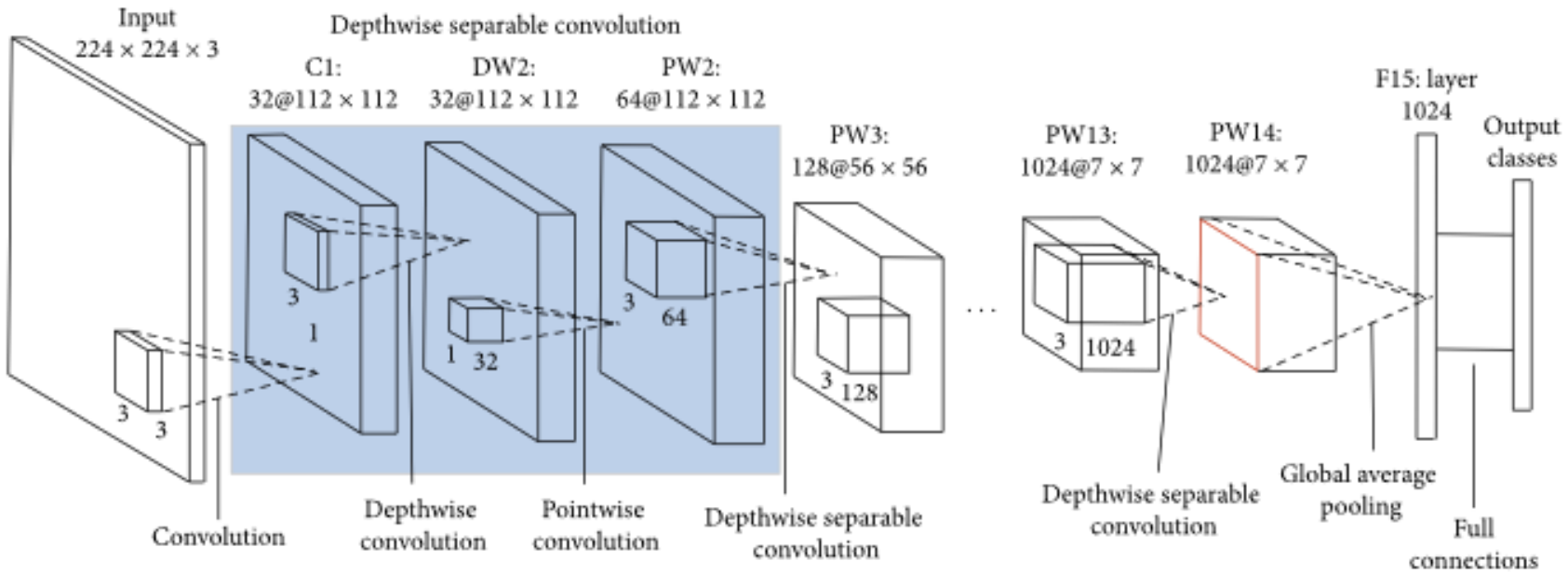
Table 5g Major challenges associated with implementation of Attention based CNN architectures.

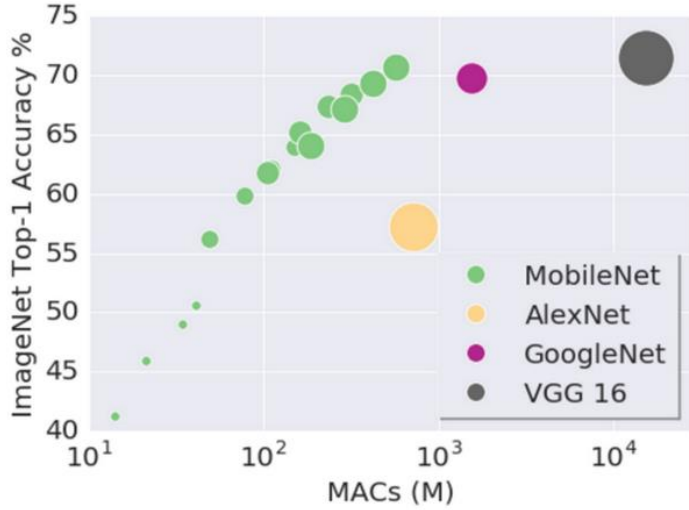
Attention	Attention Networks advantages to choose which patch is the area of the focus or most important in an image	
Architecture	**Strength**	**Gaps**
Residual Attention Neural Network	• Generates attention aware feature-maps • Easy to scale up due to residual learning • Provides different representations of the focused patches • Adds soft weights on features using bottom up top-down feedforward attention	• Complex model
Convolutional Block Attention Module	• CBAM is a generic block designed for feed forward convolutional neural networks. • Generate both feature-map and spatial attention in a sequential manner • Channel attention maps help what to focus. • Spatial attention helps where to focus. • Increases efficient flow of information. • Uses global average pooling and max pool simultaneously.	• Increase in computational load may happen

Mobile Net

it uses depthwise separable convolutions to build lightweight deep neural networks

What is MobileNet?
MobileNet is **a type of convolutional neural network designed for mobile and embedded vision applications**. They are based on a streamlined architecture that uses depthwise separable convolutions to build lightweight deep neural networks that can have low latency for mobile and embedded devices.



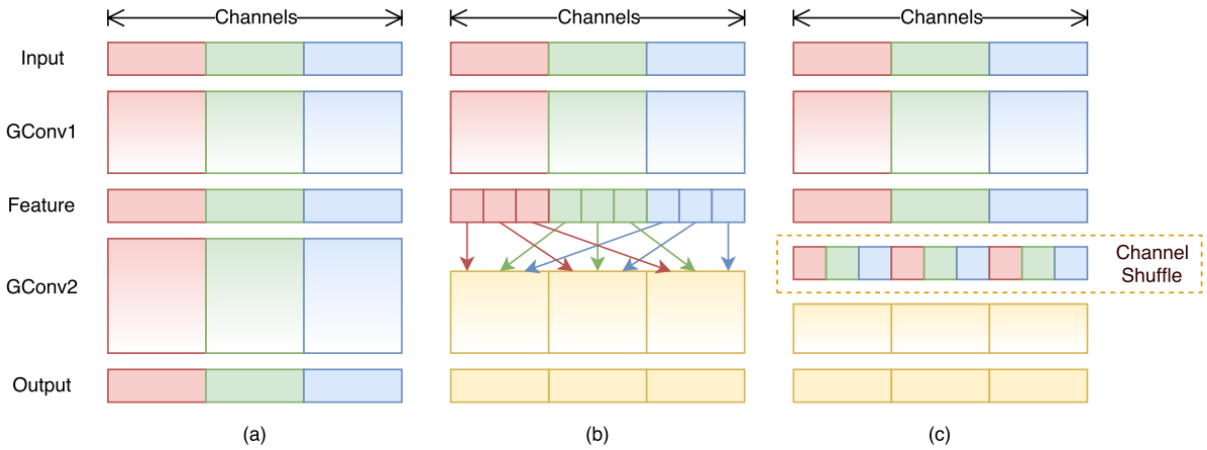


The speed and power consumption of the network is proportional to the number of MACs (Multiply-Accumulates) which is a measure of the number of fused Multiplication and Addition operation

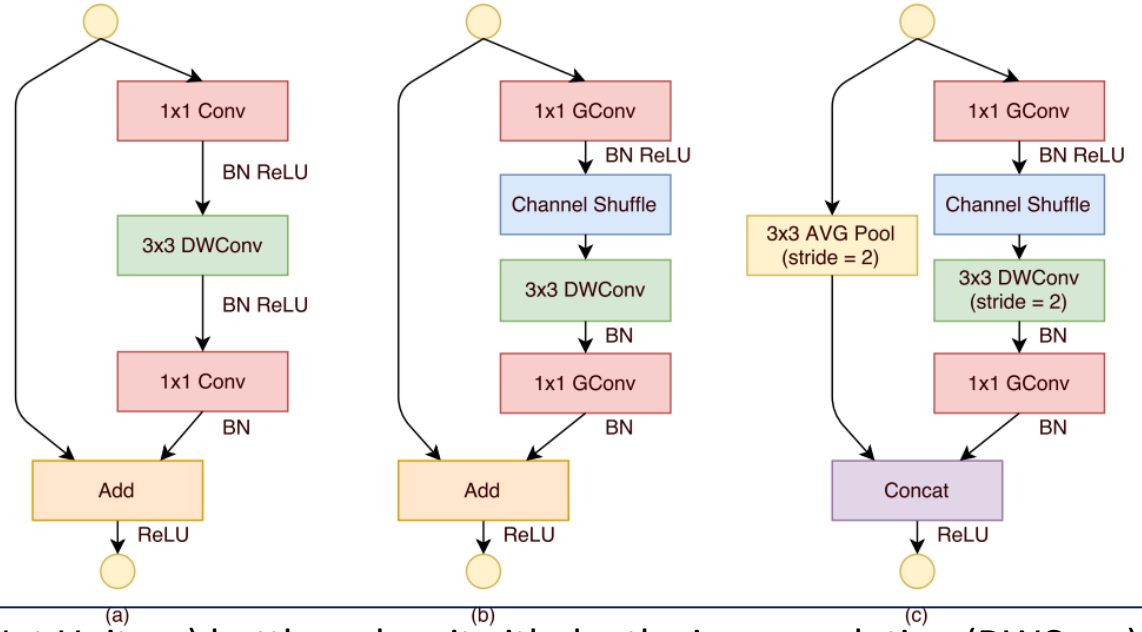
ShuffleNet

An Extremely Efficient Convolutional Neural Network for Mobile

The ShuffleNet utilizes pointwise group convolution and channel shuffle **to reduce computation cost while maintaining accuracy**. It manages to obtain lower top-1 error than the MobileNet system on ImageNet classification, and achieves ~13x actual speedup over AlexNet while maintaining comparable accuracy



Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) No cross talk; b) GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.



ShuffleNet Units. a) bottleneck unitwith depthwise convolution (DWConv) b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

Efficient Net

EfficientNet is **a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient.**

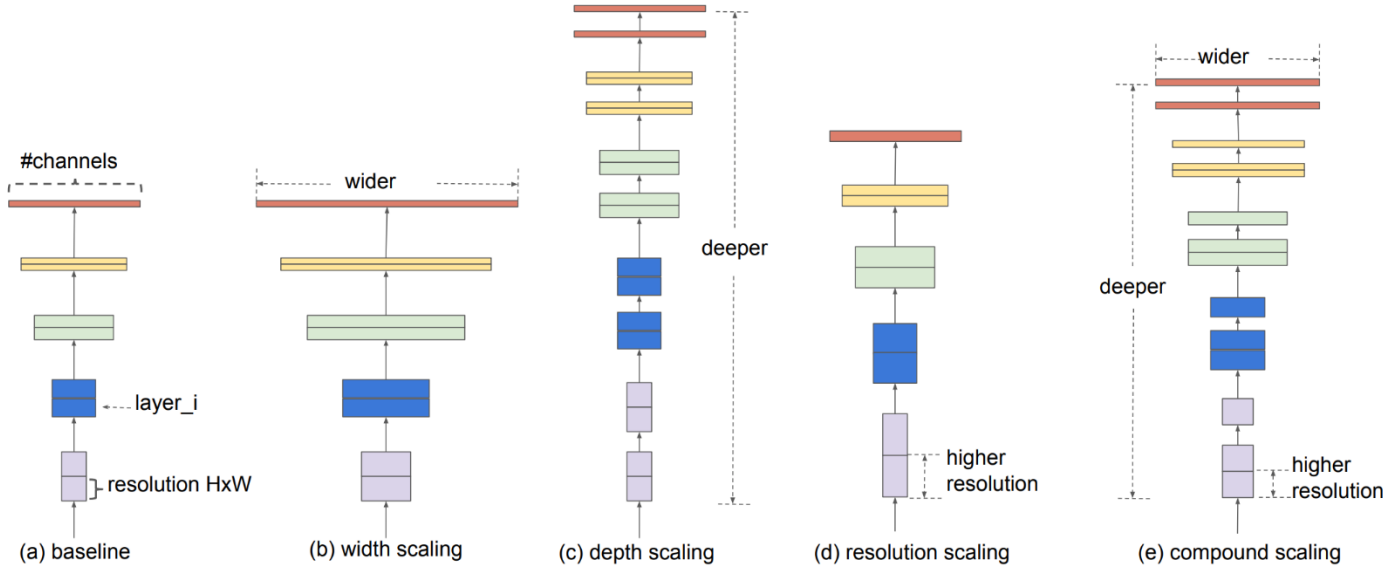
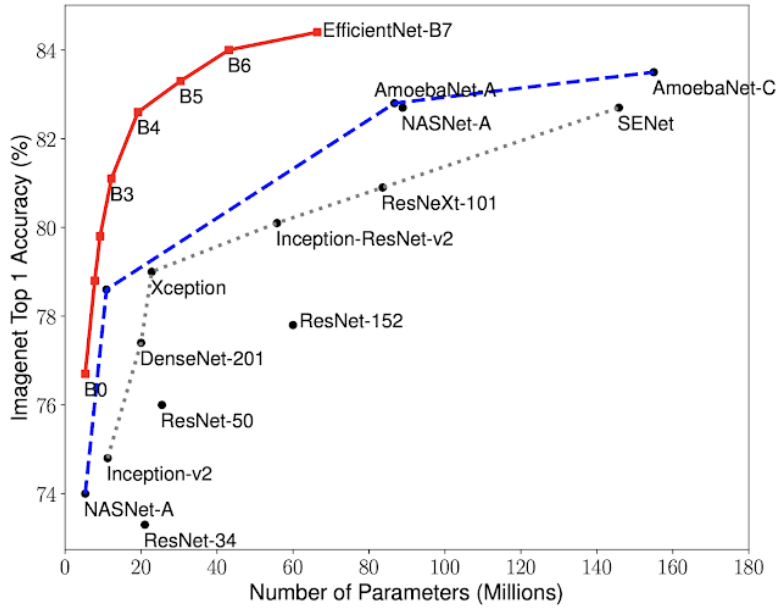


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.



Model Size vs. Accuracy Comparison. EfficientNet-B0 is the baseline network developed, while Efficient-B1 to B7 are obtained by scaling up the baseline network.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPS	Ratio-to-EfficientNet
EfficientNet-B0	**76.3%**	**93.2%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**78.8%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**79.8%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.1%**	**95.5%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.6%**	**96.3%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.3%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.9%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.4%**	**97.1%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

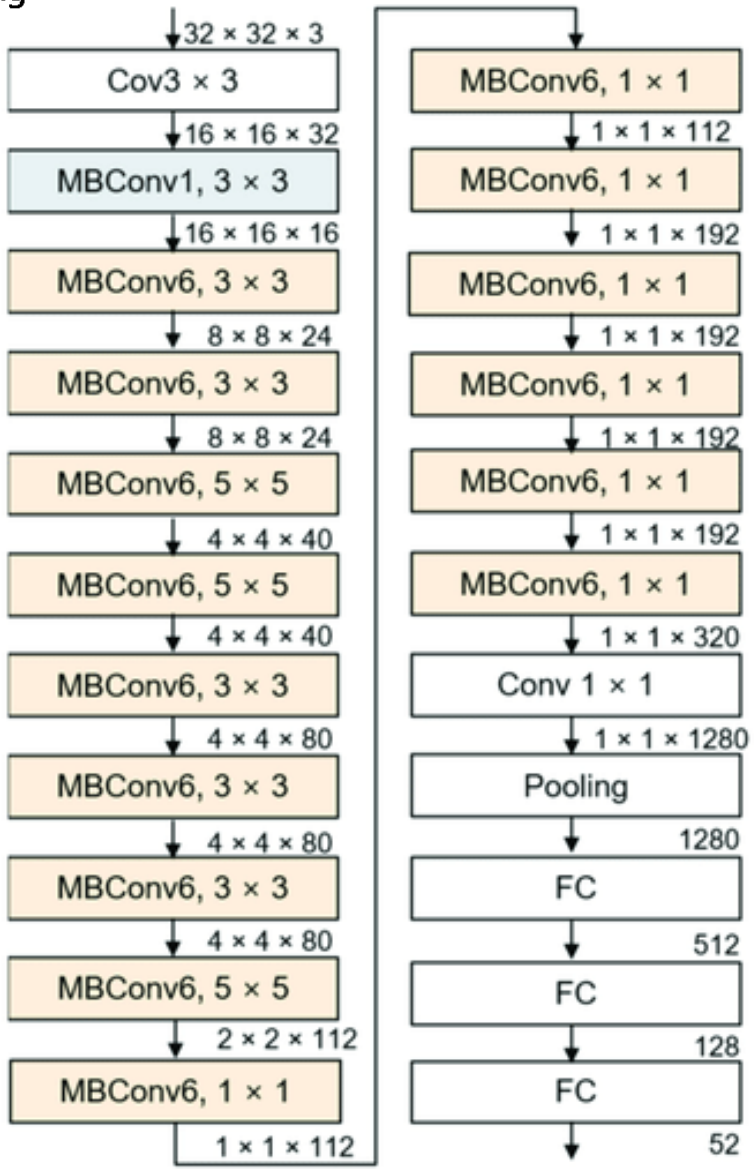
We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

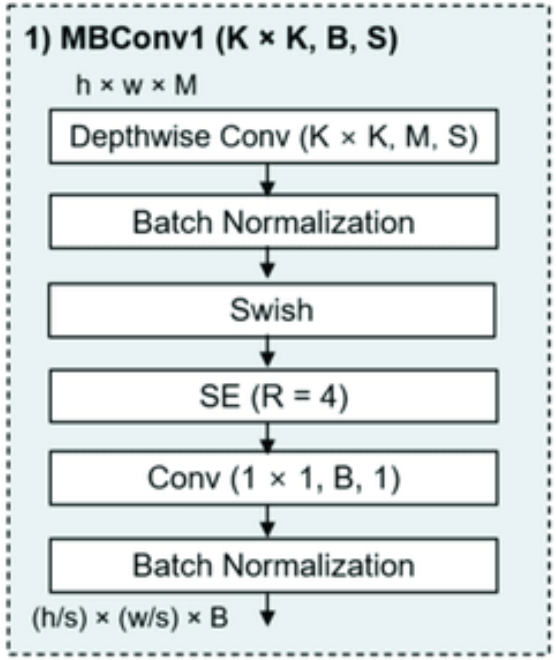
FLOPS = **floating point operations per second**

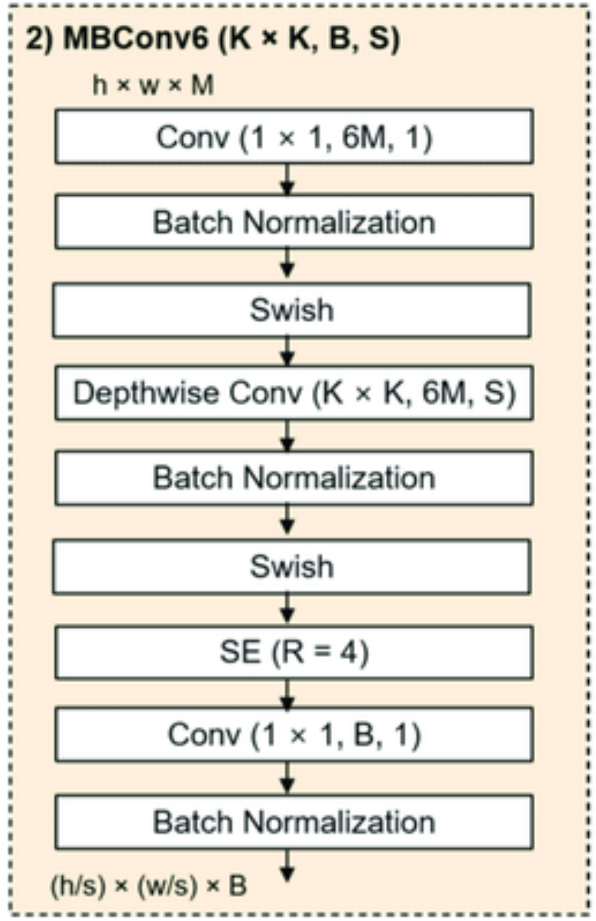
EfficientNetB0

1-XXX8PfRvndmBqCsbTUl8hw.png

The structure of an EfficientNetB0 model with the internal structure of MBConv1 and MBConv6. Compared to MBConv1, MBConv6 has three layers at the top. The number of feature maps as the output is 6.



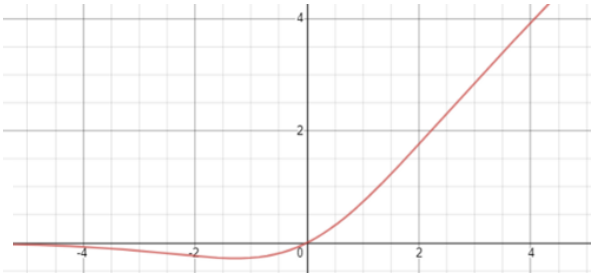




K : kernel size
M : Input Feature maps
B : Output Feature maps
S : Stride
R : Reduction ration of SE

Formally stated, the Swish activation function is…

$$f(x) = x * (1 + \exp(-x))^{-1}$$



Open Problems in CNNs

- **Interpretation:** CNNs are like a deep black box and thus may lack in interpretation and explanation
- **Layer Evaluation**: Each layer of CNN automatically tries to extract better and problem-specific features related to the task. However, for some tasks, it is imperative to know the nature of features extracted by the deep CNNs before classification. The idea of feature visualization in CNNs can help in this direction.
- **One shot Learning:** Deep CNNs are based on supervised learning mechanisms, and therefore, the availability of large and annotated data is required for its proper learning. In contrast, humans can learn and generalize from a few examples.
- **Robustness-Guarantee**: Hyper-parameter selection highly influences the performance of CNN. A little change in
the hyper-parameter values can affect the overall performance of a CNN.
- **Acceleration and Compressing:** The efficient training of CNN demands powerful hardware resources such as GPUs.
- Predictability--**Guarantee** one shortcoming of CNN is that it is generally unable to show good performance when used to estimate the pose, orientation, and location of an object.