Analysis and Design of Deep Neural Networks

Part 1

Fall 2022

Ahmad Kalhor-University of Tehran

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)
- DNNs 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:
- $f \in \{sign, \mathrm{step}, \tanh, sigmoid, Relu, identity \cdots\}$

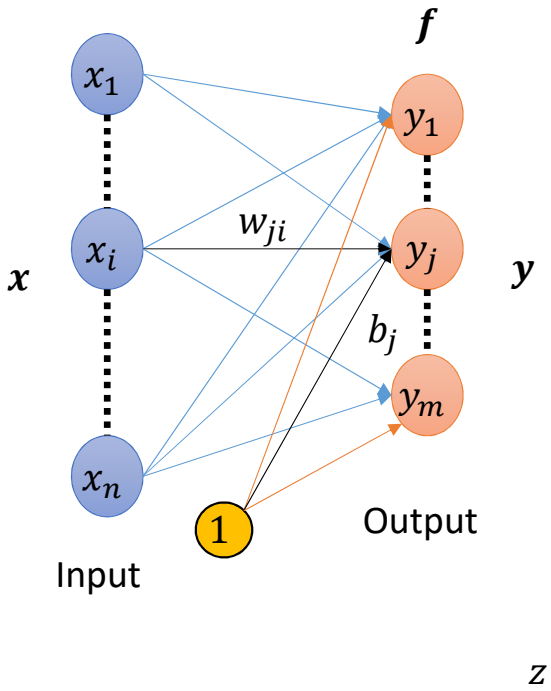
Functionality

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

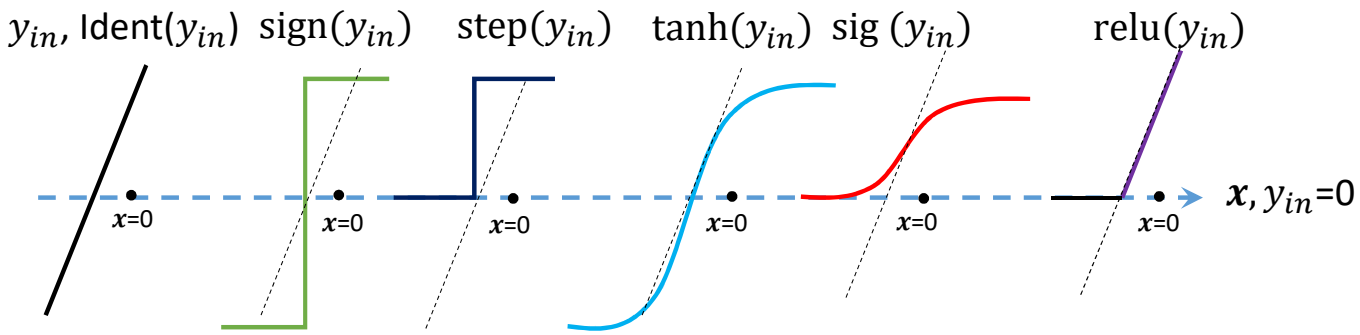
$$y_{in_j} = \mathrm{w_{j1}x_1} + \cdots + \mathrm{w_{jn}x_n} + \mathrm{b_j}$$

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

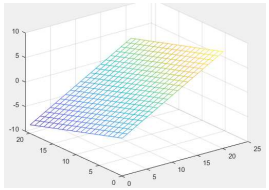
$$y_j = f(y_{in_j})$$



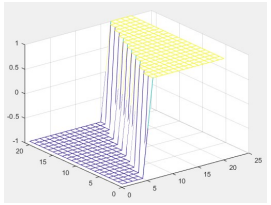
Example(1) for $n = 1$, $f(yin)$, $y_{in} = 2x + 1$ w:slope parameter , b: $(up \uparrow down \downarrow) shift\ parameter$



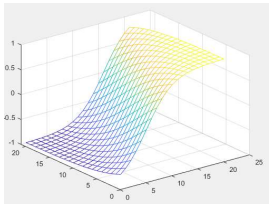
Example(2) for $n = 2$, $f(yin)$, $y_{in} = 2x_2 - x_1$+0



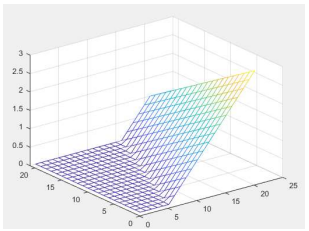
$y_{in}, ident(y_{in})$



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:
- $f^{1,2} \in \{sign, \text{step}, \tanh, sigmoid, Relu, identity \cdots\}$

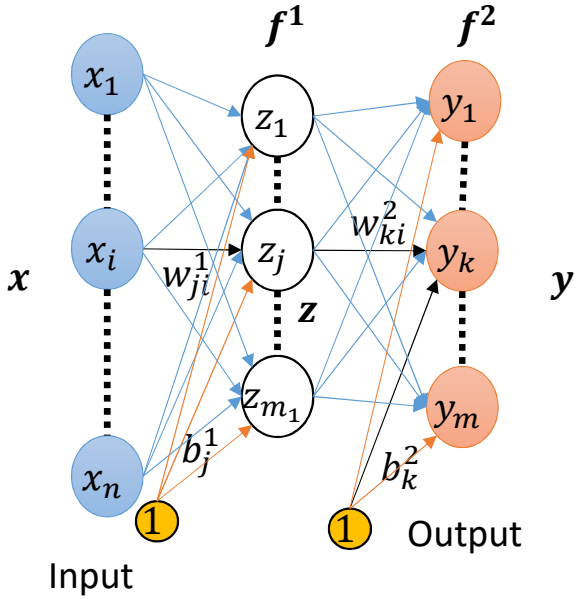
Overall Functionality

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

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

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

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



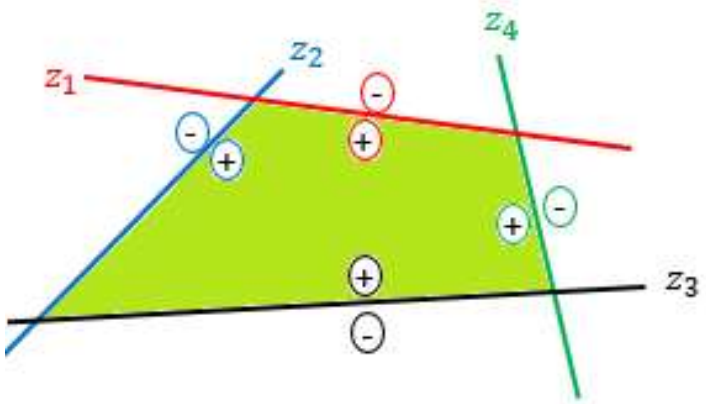
A Desired Functionality in classification /Regression problems

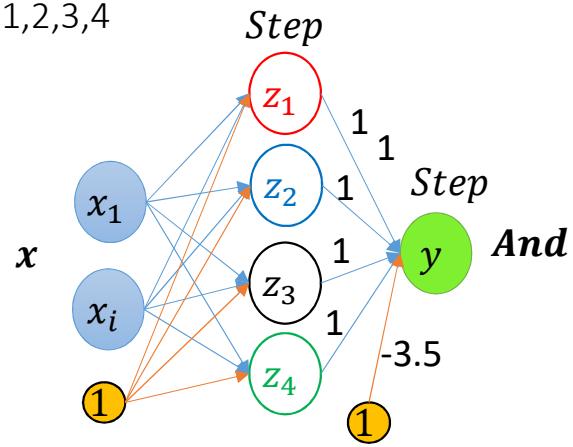
(1) A hyper plane: $z_{in_j} = w_{j1}^1 x_1 + \cdots + w_{jn}^2 x_n + b_j^1$

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

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

Overall Functionality

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

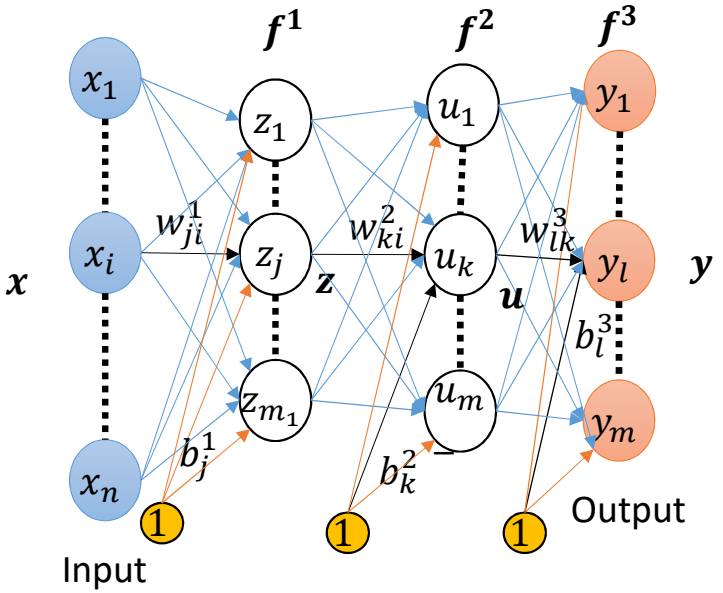
(2) Folding, Rectifying,.. on the hyper plane: $z_j = f^1(z_{in_j})$

(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

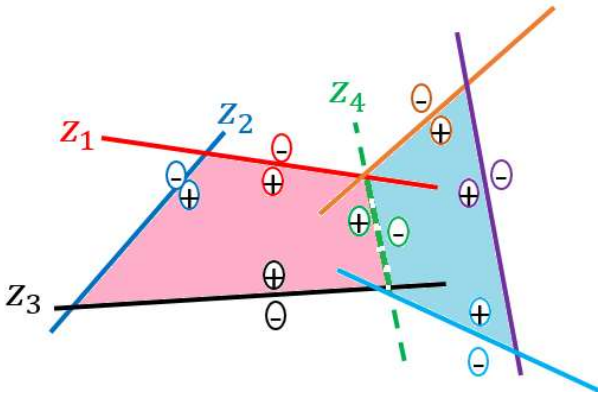
(1) A hyper plane: $z_{in_j} = w_{j1}^1 x_1 + \cdots + w_{jn}^1 x_n + b_j^1$

(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.

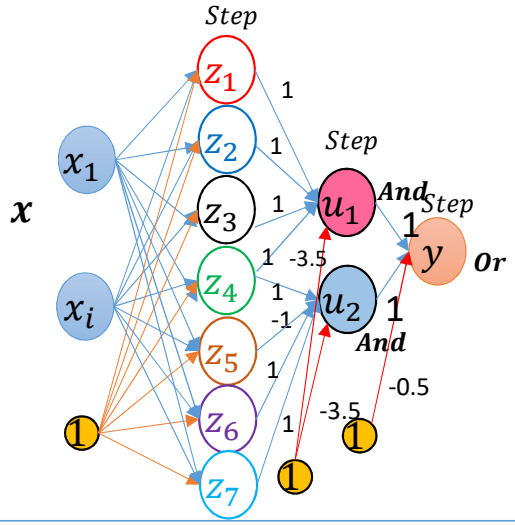
(4) An "and" or "or logic gate for former convex hyper-planes is defined by "lth" neuron of last layer, by which "lth" non-convex hyper-polygon will be resulted.

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



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

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

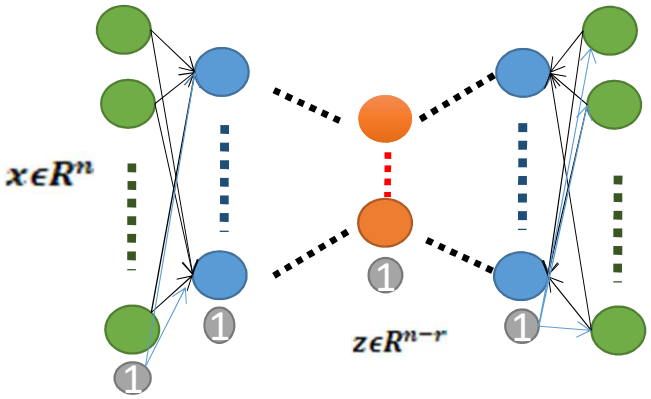


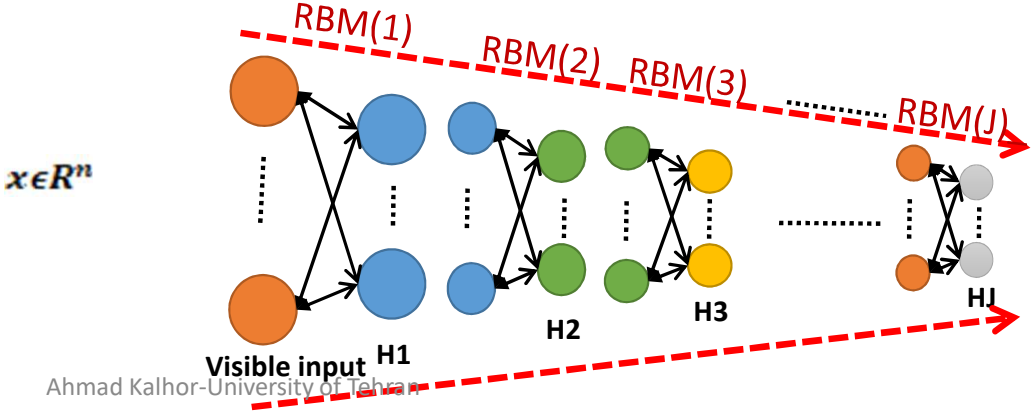
- A block of two FC layers can transform the input space to multi non-convex hyper- polygon partitions
- Using soft activation functions, a block of two FC layers can transform the input space to multi non-convex hyper- polygon volume maps

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

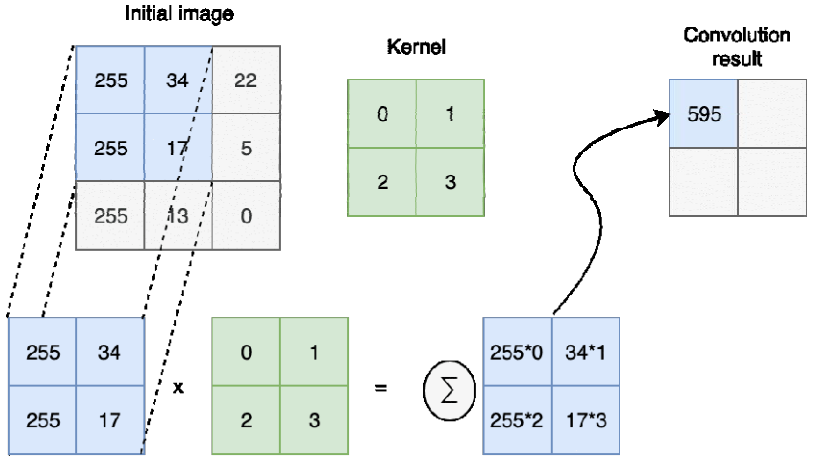
* Most of exampled are from https://ikhlestov.github.io/pages/machine-learning/convolutions-types/, Illarion Khlestov.

Simple Convolution

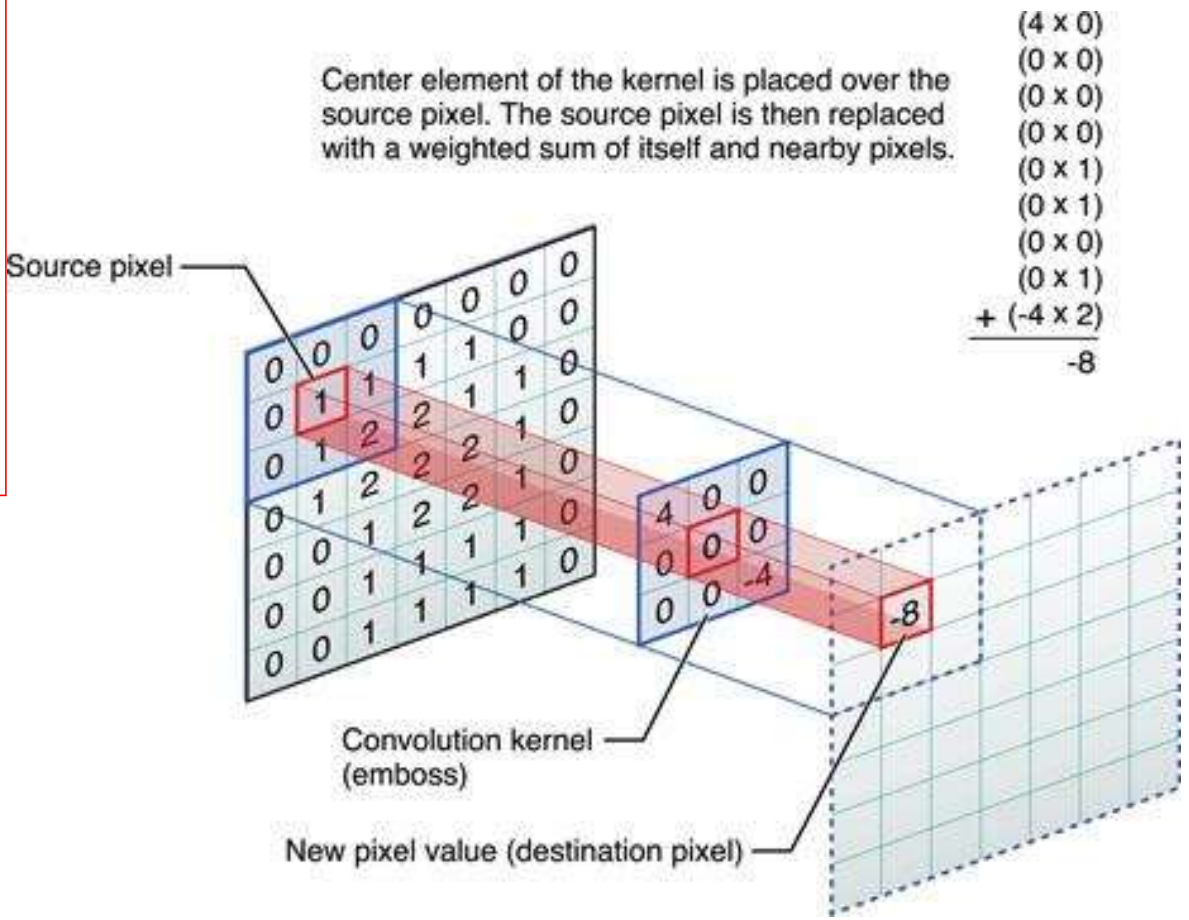
Take multiply dot products with same filter with some width/height shift.

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)

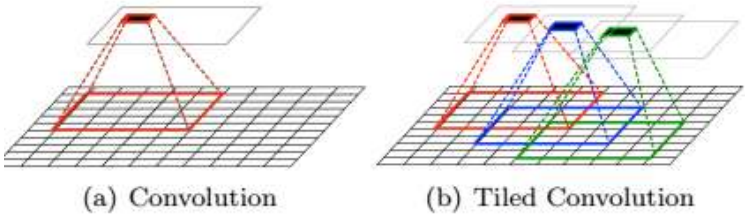


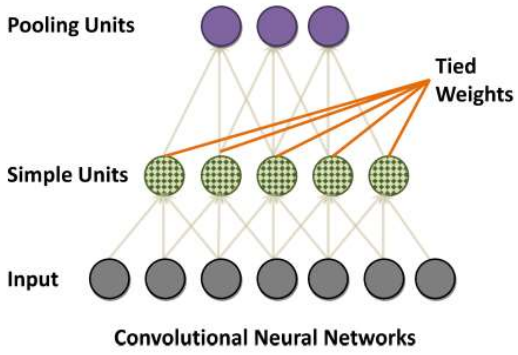
Tiled Convolution

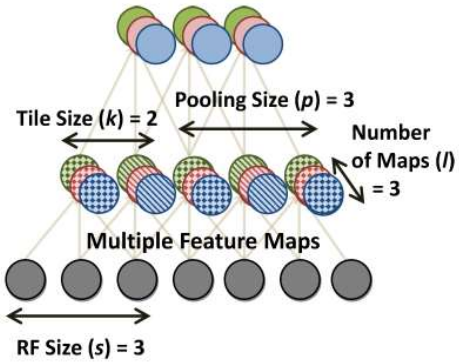
- It means that we can learn multiples feature maps rather than one feature map by considering several different filters with the same size.
- Separate kernels are learned within the same layer
- In the tiled convolution we can extract different kinds of frequent patterns from the input signals.

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

Example of Tiled convolution on two dimensional signal







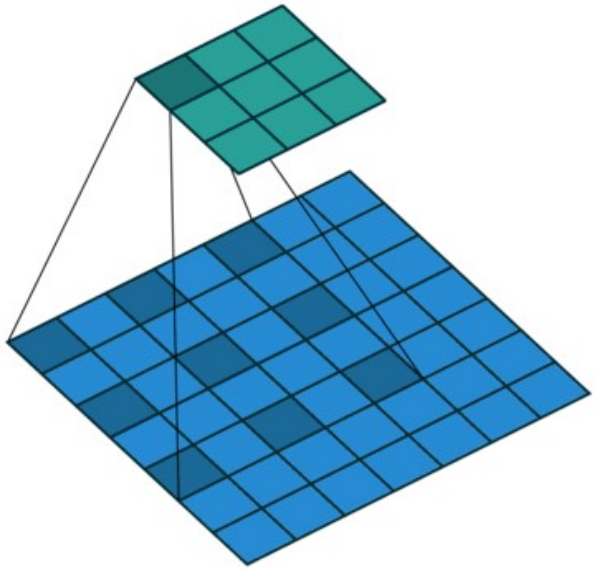
Some notes about the convolution layer

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.

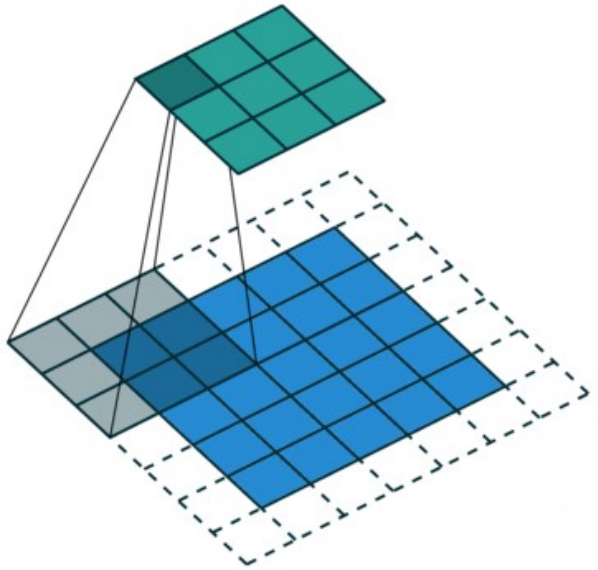
Dilated Convolution

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)

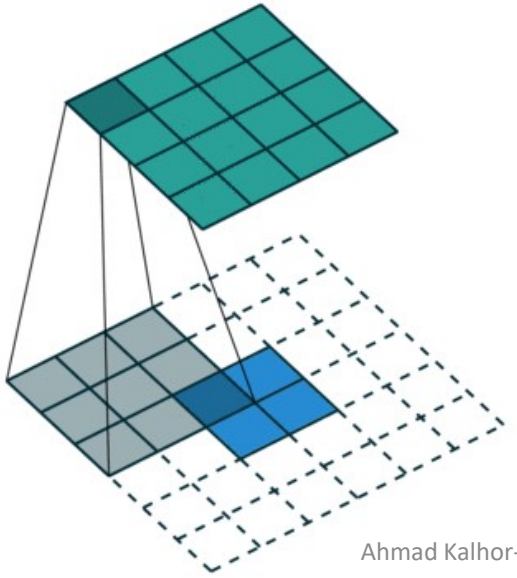


Standard Convolution (l=1)

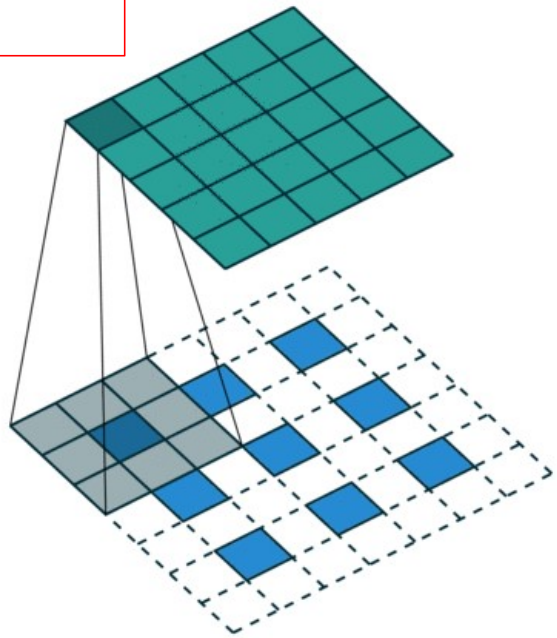
Deconvolution (Transposed Convolution)

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)).

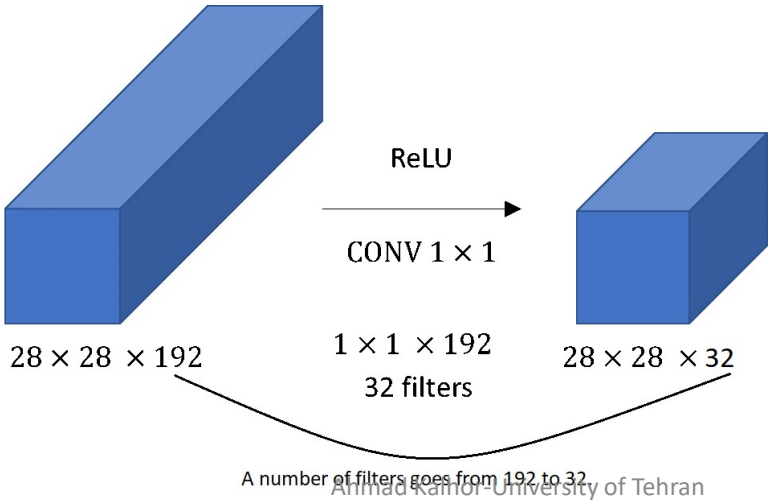


In case of stride two and padding, the transposed convolution would look like this



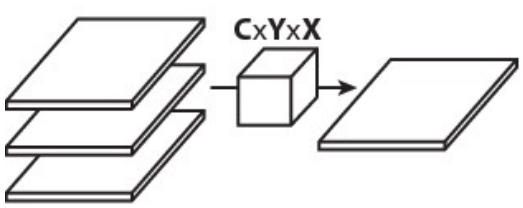
1x1 Convolutions

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.

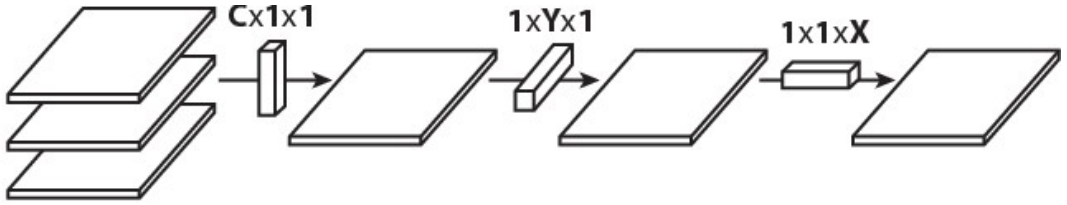


Flattened Convolutions

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



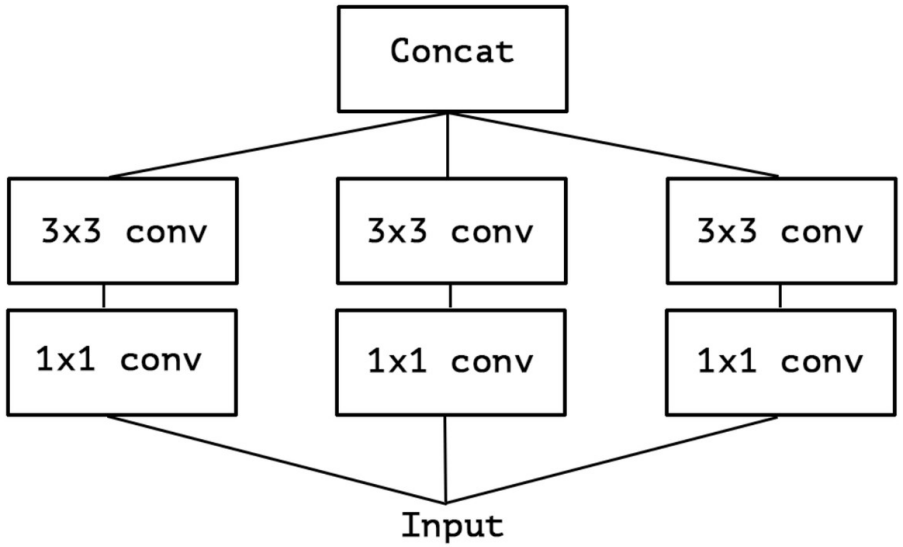
(b) 1D convolutions over different directions

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

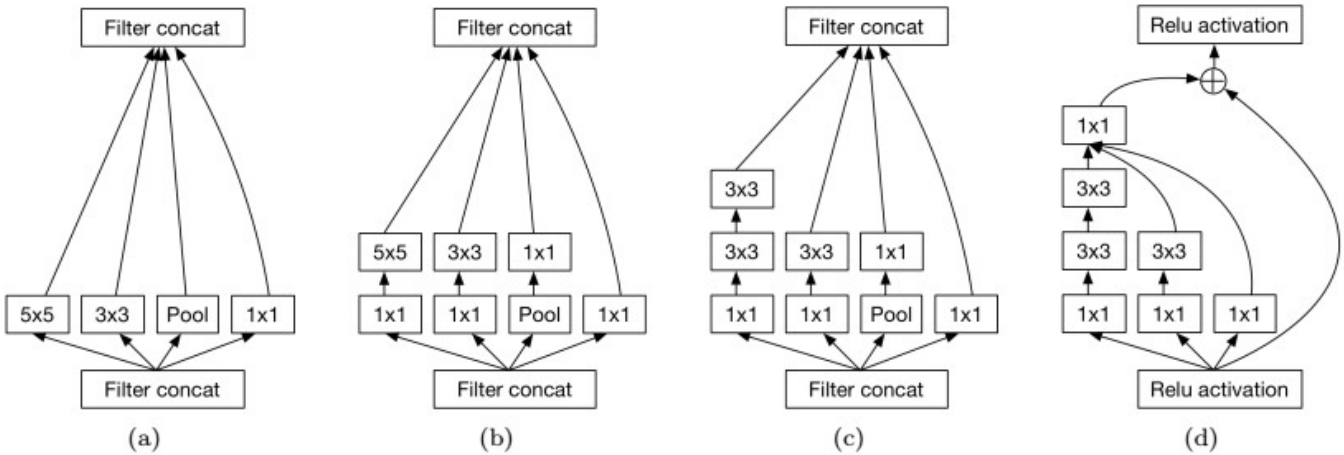


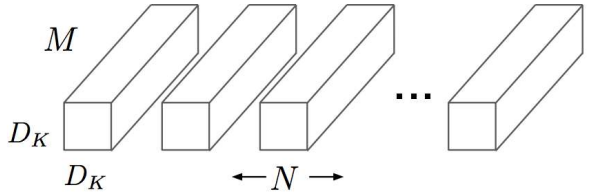
Figure 5: (a) Inception module, naive version. (b) The inception module used in [10]. (c) The improved inception module used in [41] where each 5×5 convolution is replaced by two 3×3 convolutions. (d) The Inception-ResNet-A module used in [42].

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

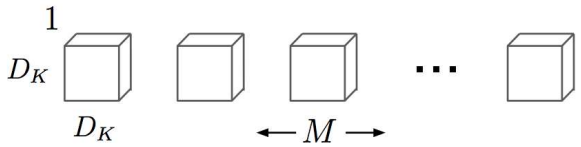
* C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

Depthwise Separable Convolutions

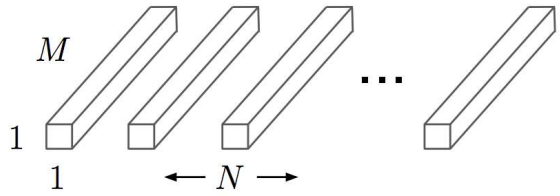
A lot about such convolutions published in the (Xception paper) or (MobileNet paper).
Consist of:
•**Depthwise convolution**, i.e. a spatial convolution performed independently over each channel of an input.
•**Pointwise convolution**, i.e. a 1x1 convolution, projecting the channels output by the depthwise 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



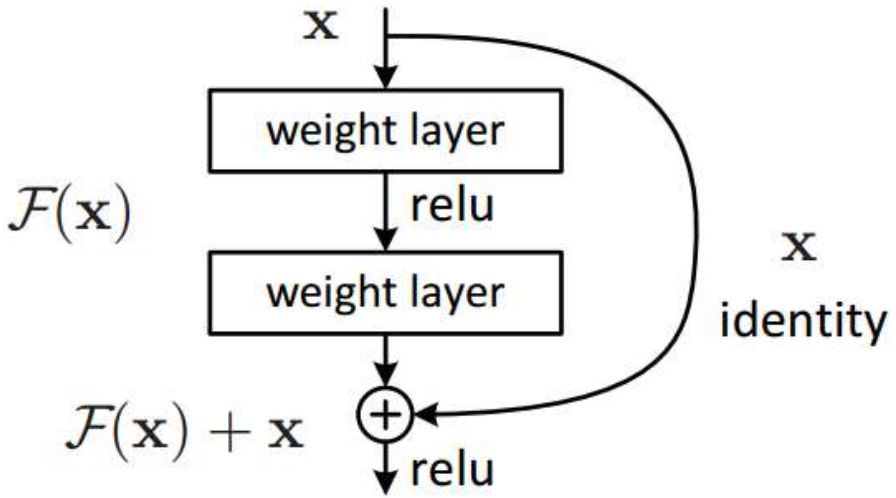
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

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

Residual Blocks

To avoid missing information and provide a solution for vanishing gradient

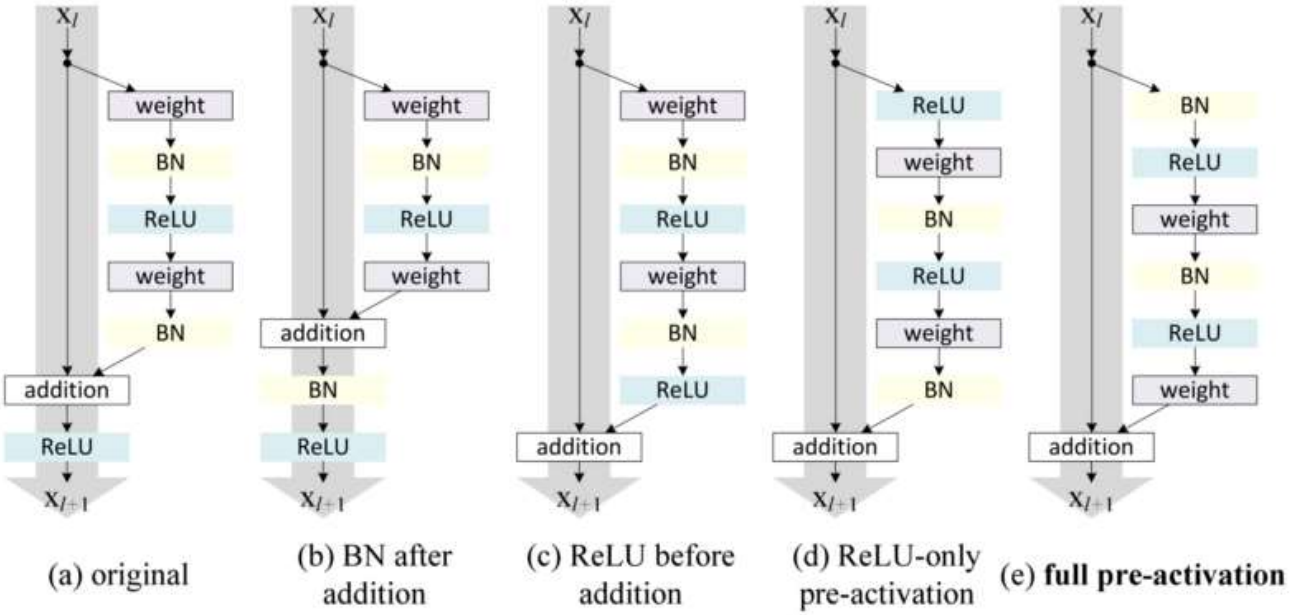
In traditional neural networks, each layer feeds into the next layer. In a network with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away. That's it. But understanding the intuition behind why it was required in the first place, why it is so important, 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.

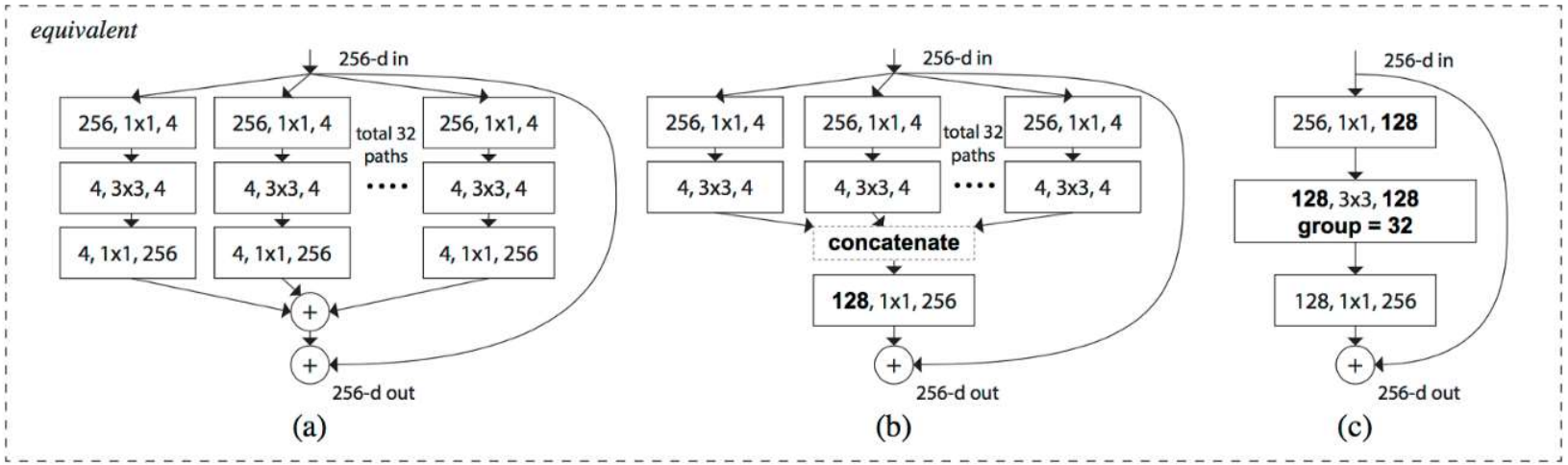


Figure 3. Equivalent building blocks of ResNeXt. **(a)**: Aggregated residual transformations, the same as Fig. 1 right. **(b)**: A block equivalent to (a), implemented as early concatenation. **(c)**: A block equivalent to (a,b), implemented as grouped convolutions [24]. Notations in **bold** text highlight 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

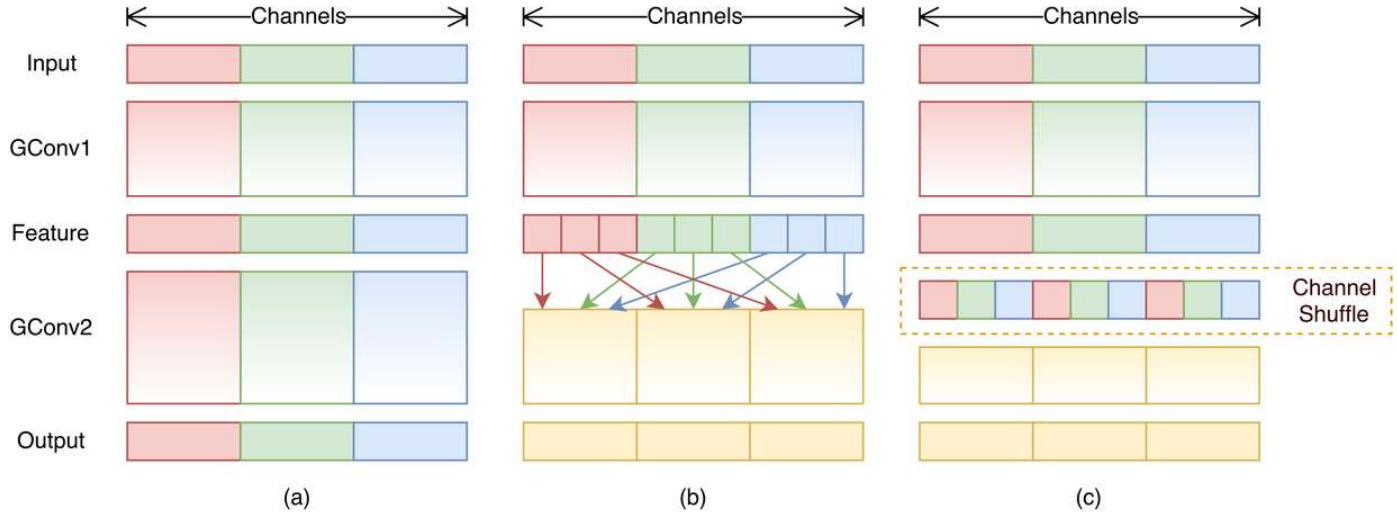


Figure 1: Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when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

Some notes about convolution layers

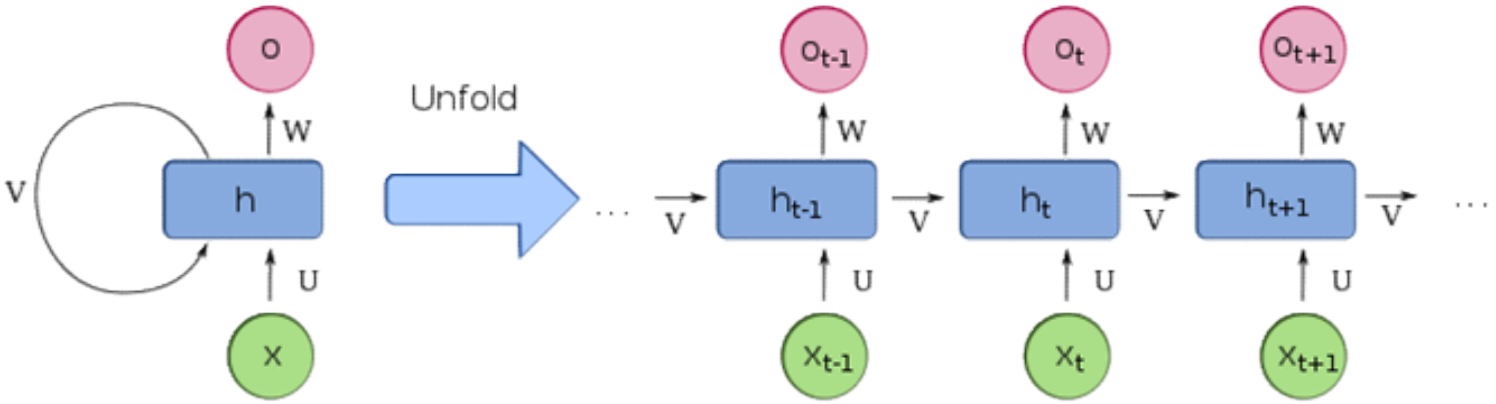
1. There are many extensions of simple convolutional layers.
2. Deconvolution layers are introduced as the backward pass of their corresponding convolution.
3. Residual blocks use a direct skip connection from inputs to outputs(as an identity function) to avoid missing information and to solve the problem 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



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

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

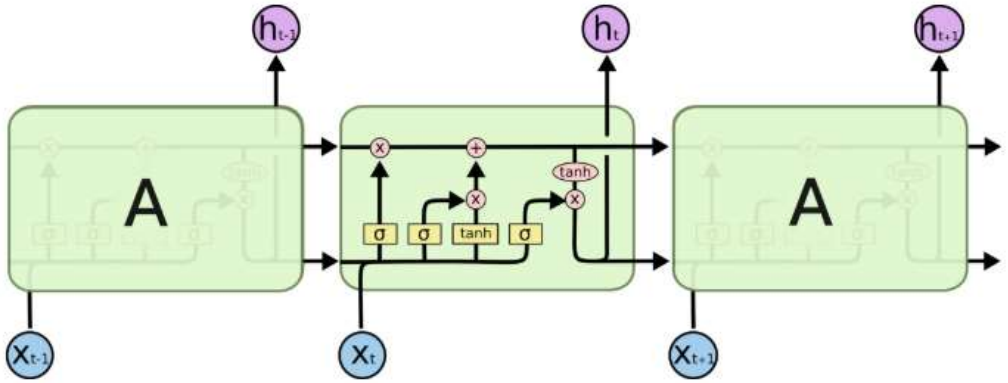
$$\boldsymbol{o}_t = \mathbf{g}\Big(\mathrm{W}\mathbf{f}\big(\mathrm{V}\mathbf{f}\big(\mathrm{V}\ldots\big(\mathrm{V}\mathbf{f}\big(\mathrm{V}\boldsymbol{h}_0 + \mathrm{U}\boldsymbol{x}_1\big)\ldots + \mathrm{U}\boldsymbol{x}_{t-1}\big)\big) + \mathrm{U}\boldsymbol{x}_t\big)\Big), t = 1,2,\cdots$$

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:

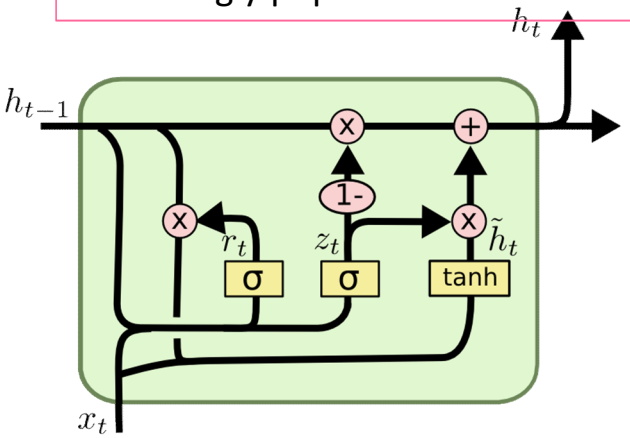
1. LSTM is a module of f*our interacted layers*.
2. A cell state is a key concept in a LSTM proving (packaging and carrying) all informative data required to save long/short dependencies among the sequenced patterns.
3. Using exogenous input and the hidden state, cell state interacts by three independent 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.



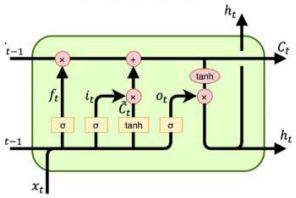
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

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

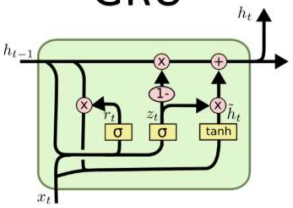
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

LSTM



GRU

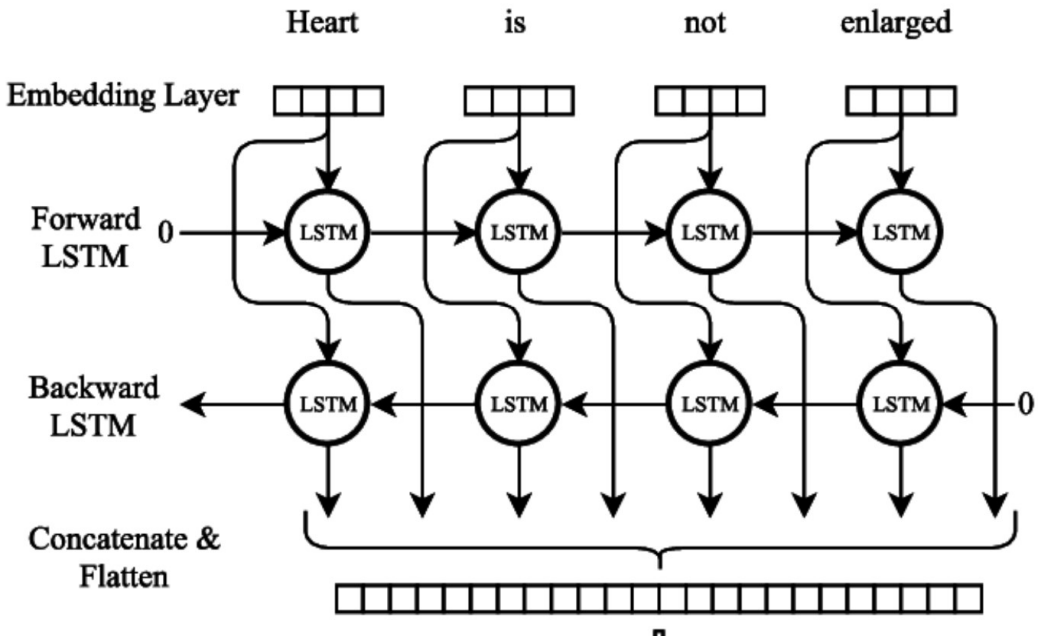


Bidirectional RNNs

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.

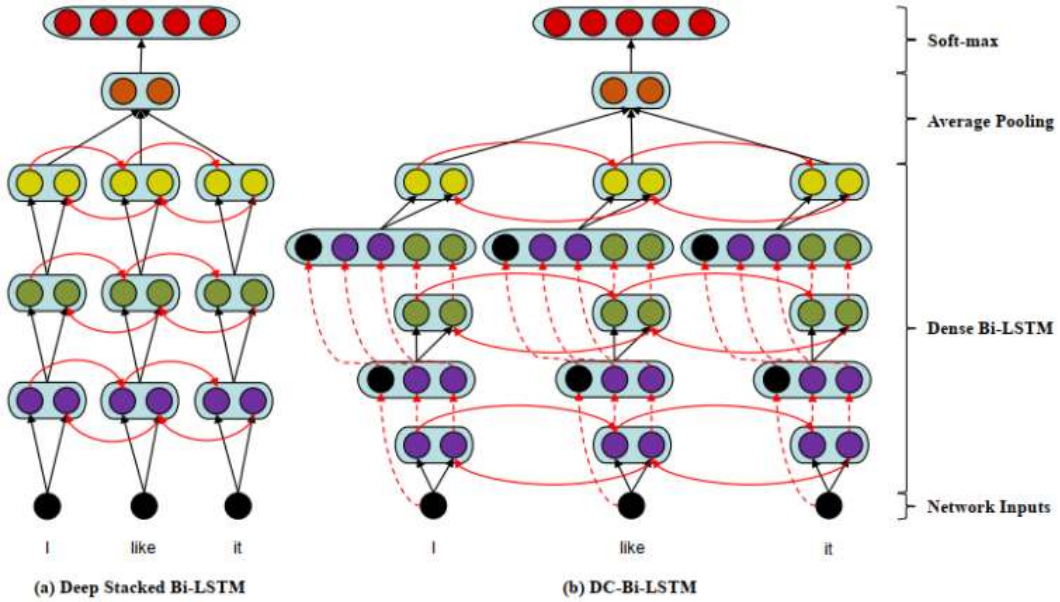


It can be used in categorization
Missed words prediction and so on.

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)

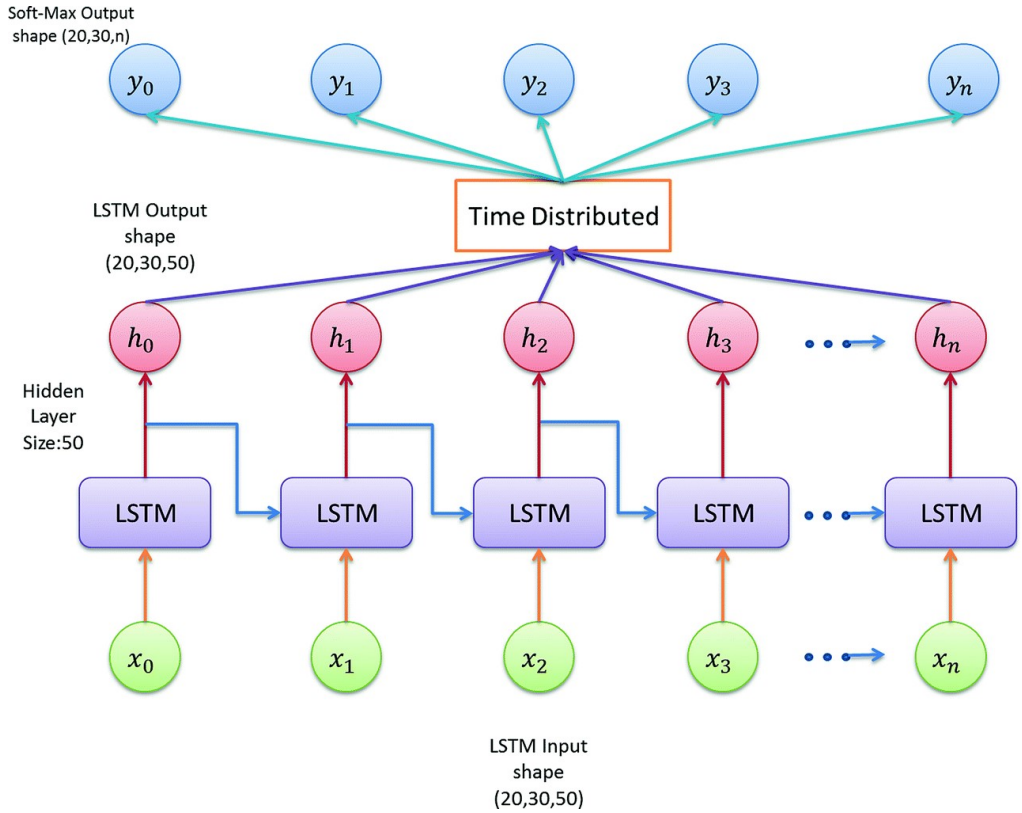


Deep Stacked Bi-LSTM Densely Connected Bidirectional

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.

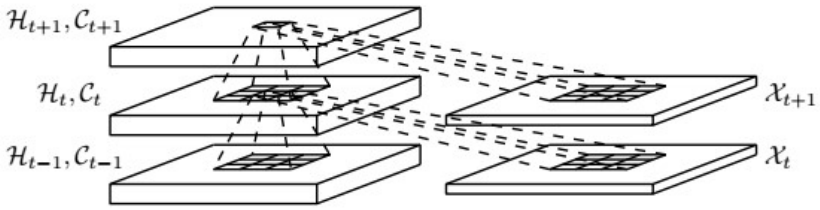
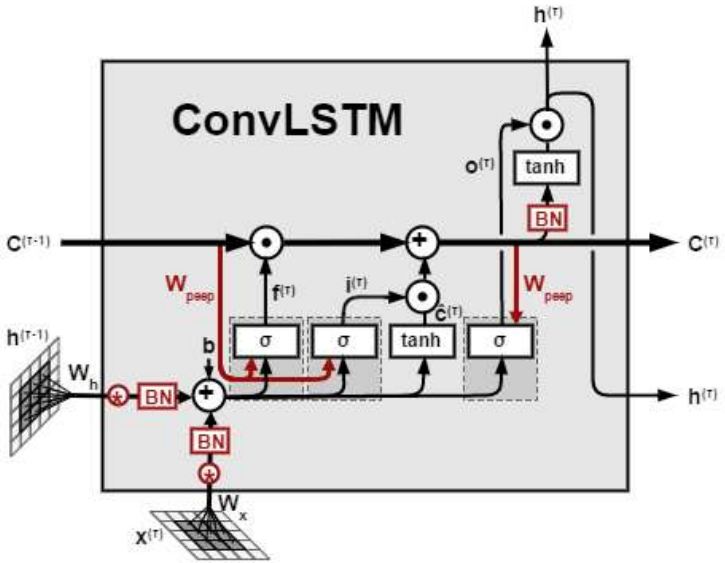


Figure 2: Inner structure of ConvLSTM

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



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

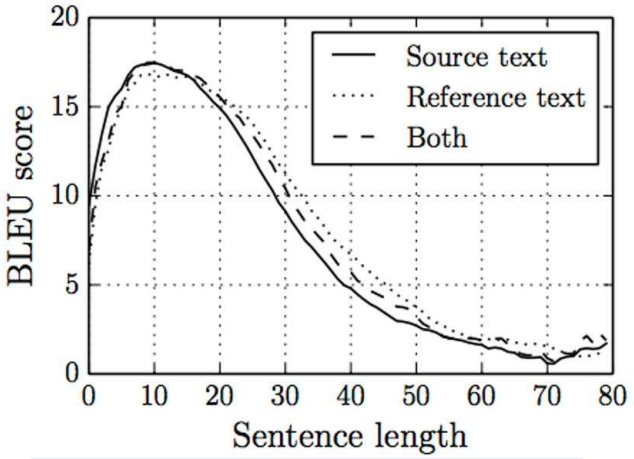
- In translation machines, LSTM/GRU as a seq2seq model can not predict successful translation for a sentence, when its length increases.

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

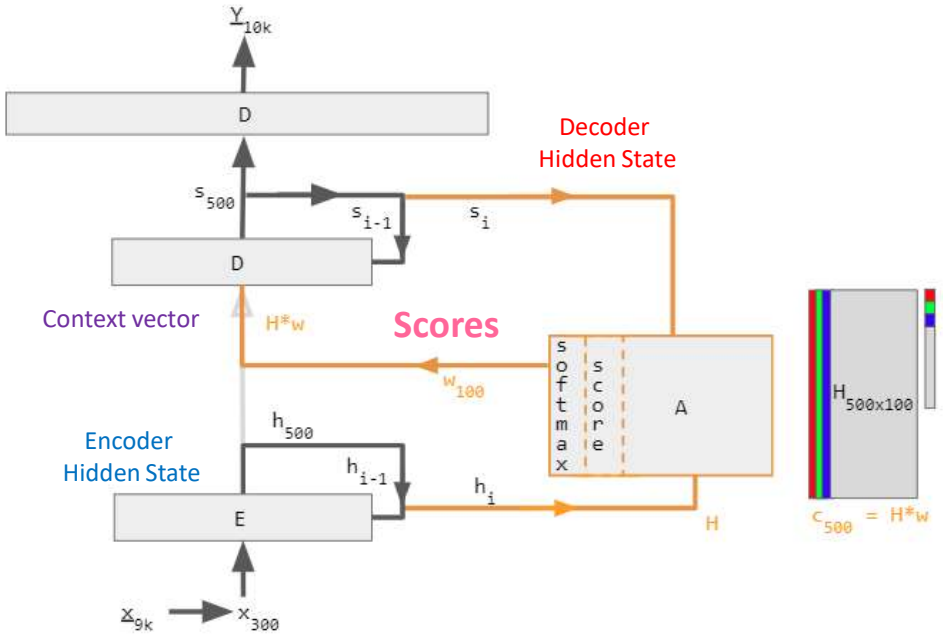
Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio

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.

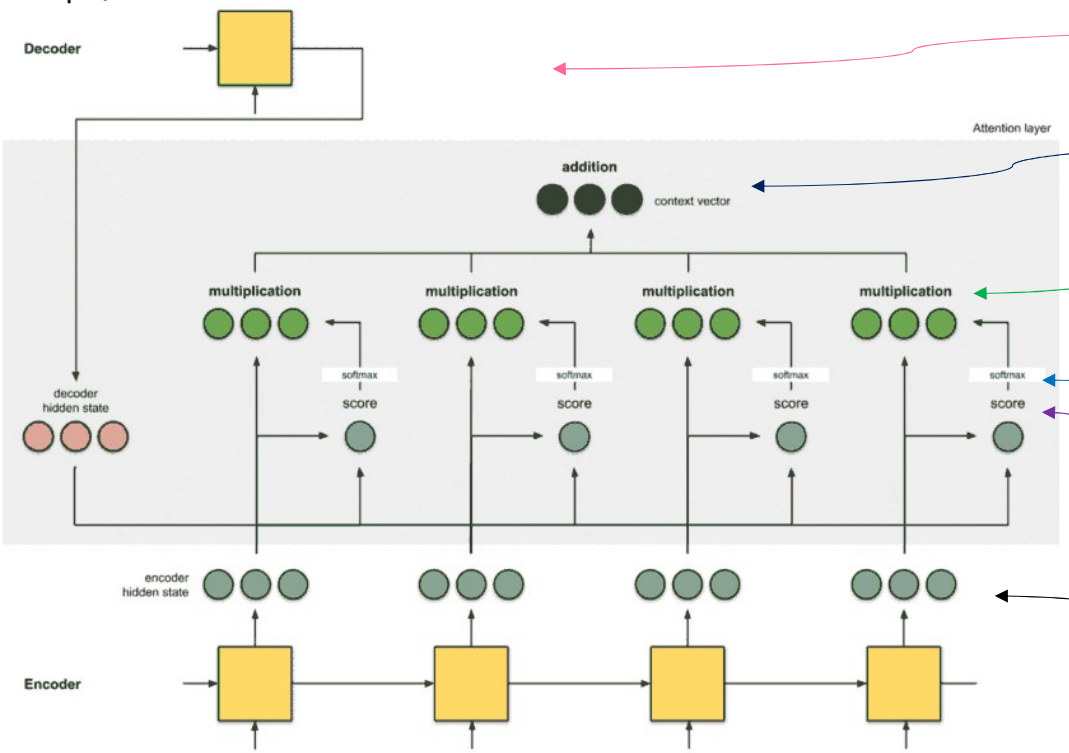
Numerical subscripts are examples of vector sizes. Lettered subscripts i and i-1 indicate time step.



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.

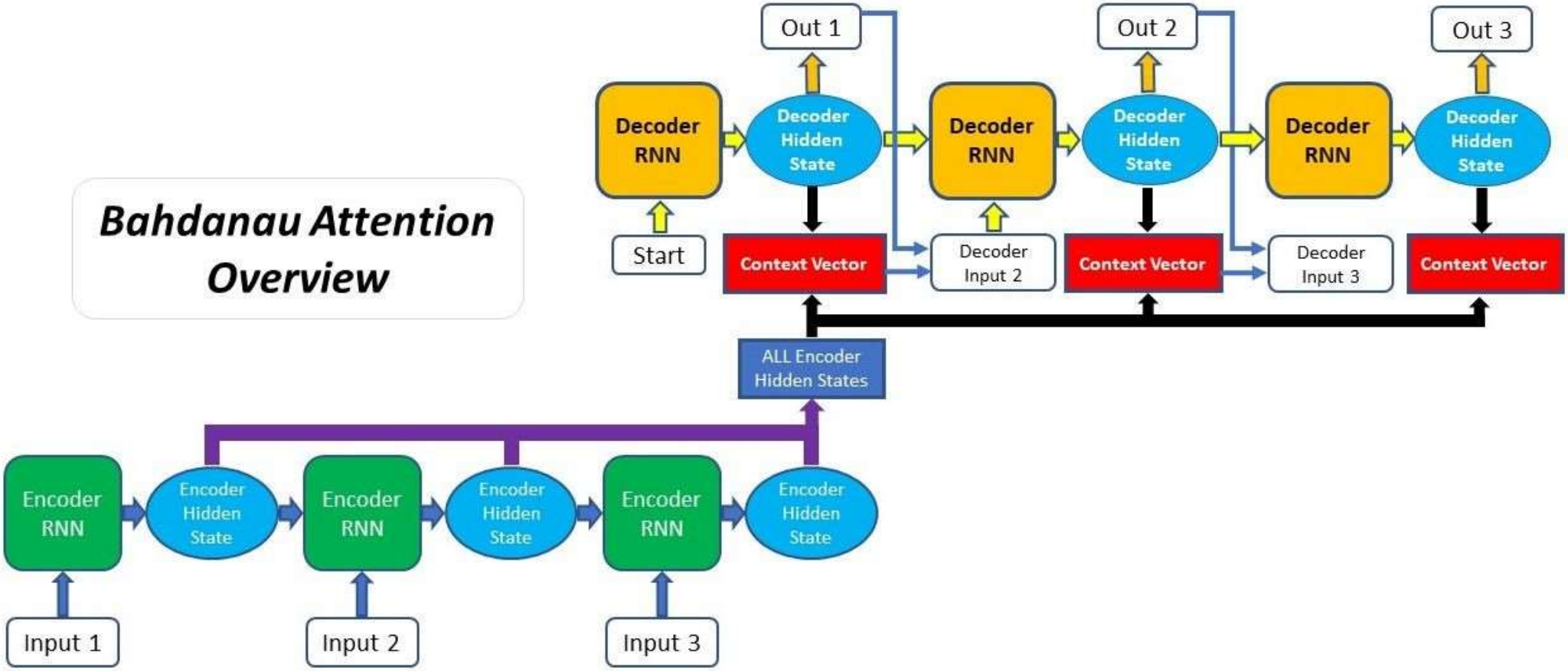
***Encoder-Decoder Recurrent Neural Network Models for Neural Machine Translation**
by Jason Brownlee on January 1, 2018 in Deep Learning for Natural Language Processing

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

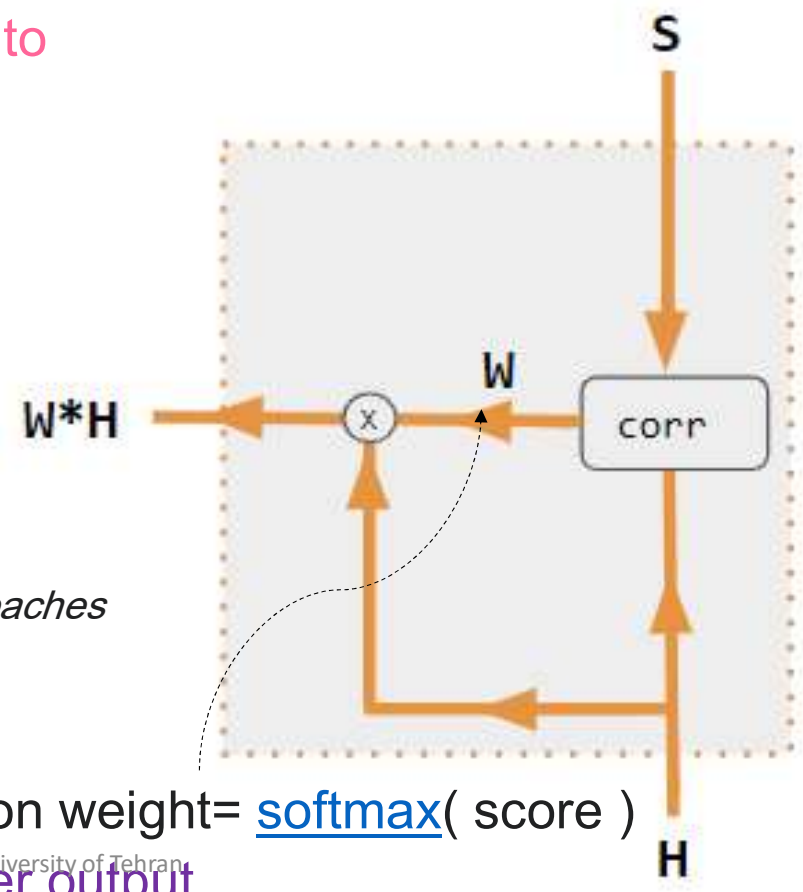
Attention variants

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

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

w = attention weight= softmax(score)

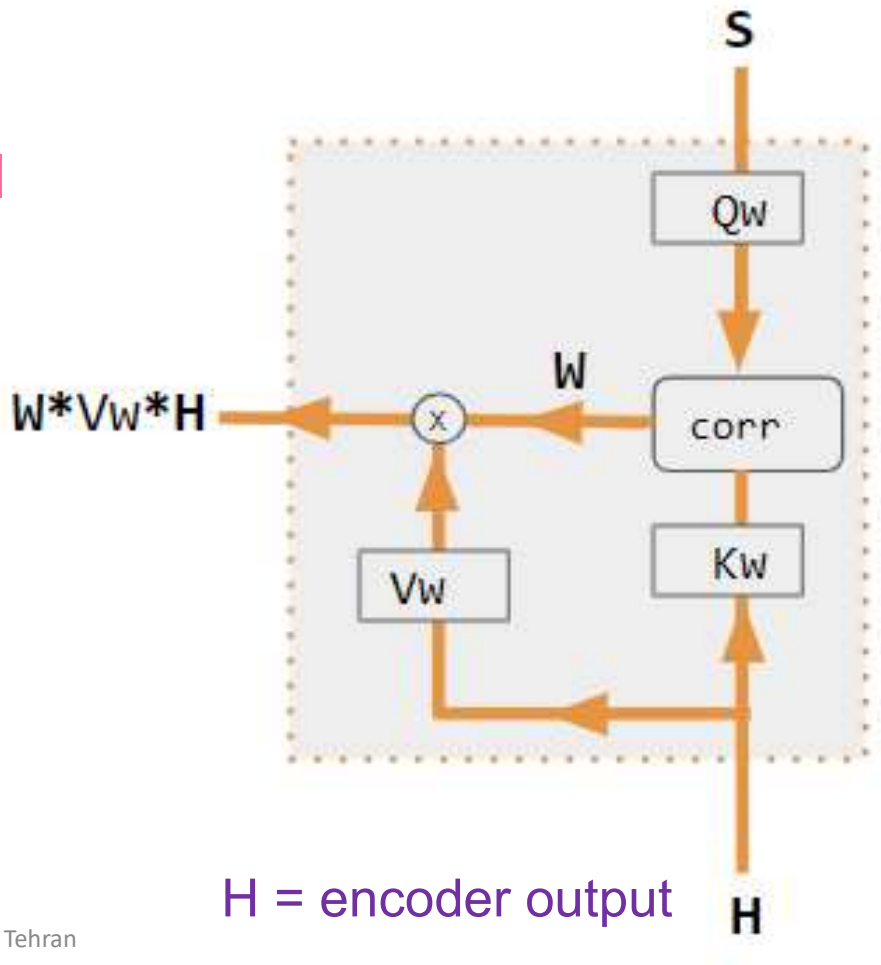
H = encoder output

2. encoder-decoder QKV

s = decoder input to Attention

Both Encoder & Decoder are needed to calculate Attention.

c = context vector

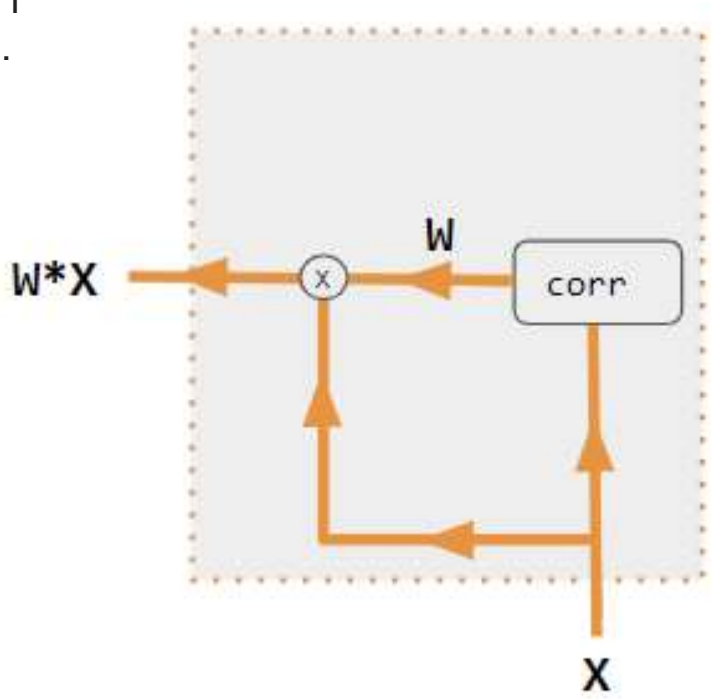


H = encoder output

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

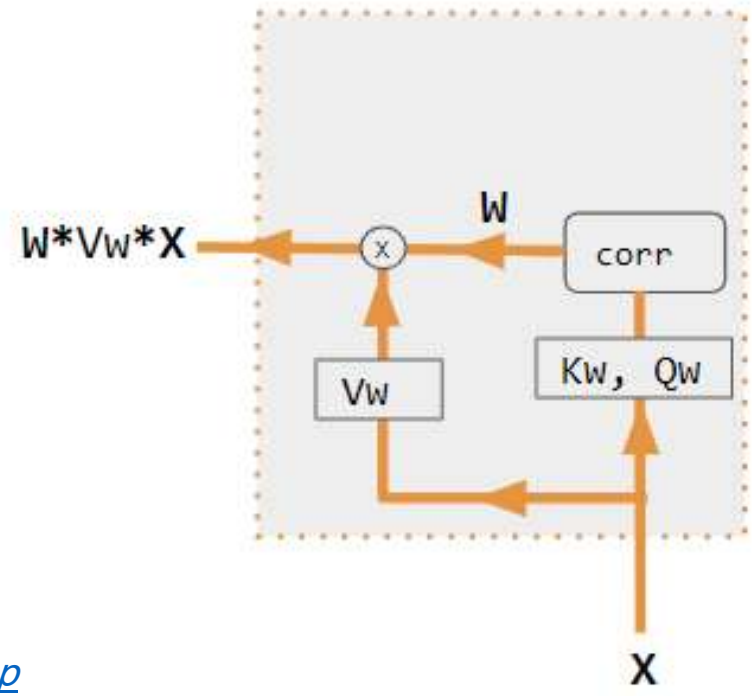
Decoder is NOT used to calculate Attention. With only 1 input into corr, W is an auto-correlation of dot products.
$w_{ij} = x_i * x_j$ [10]



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

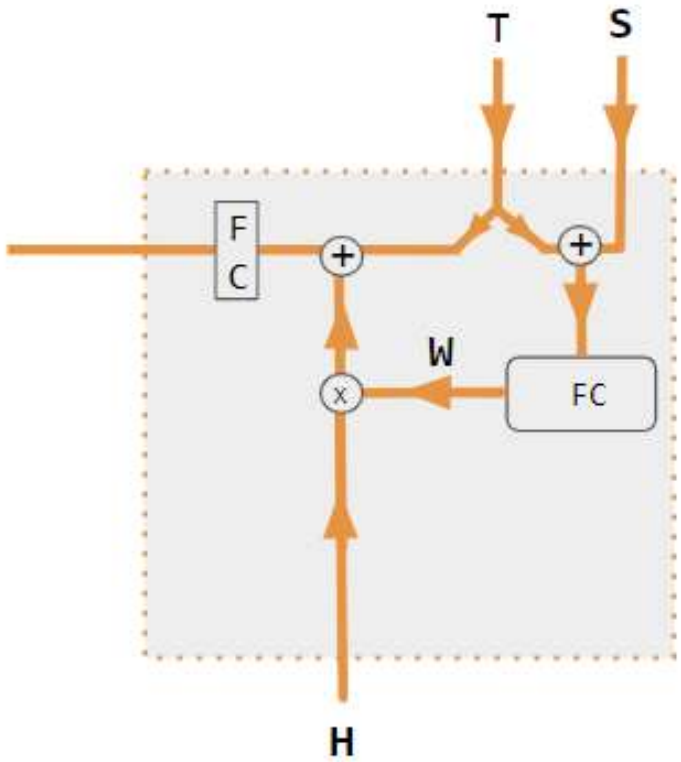
4. encoder-only QKV

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.

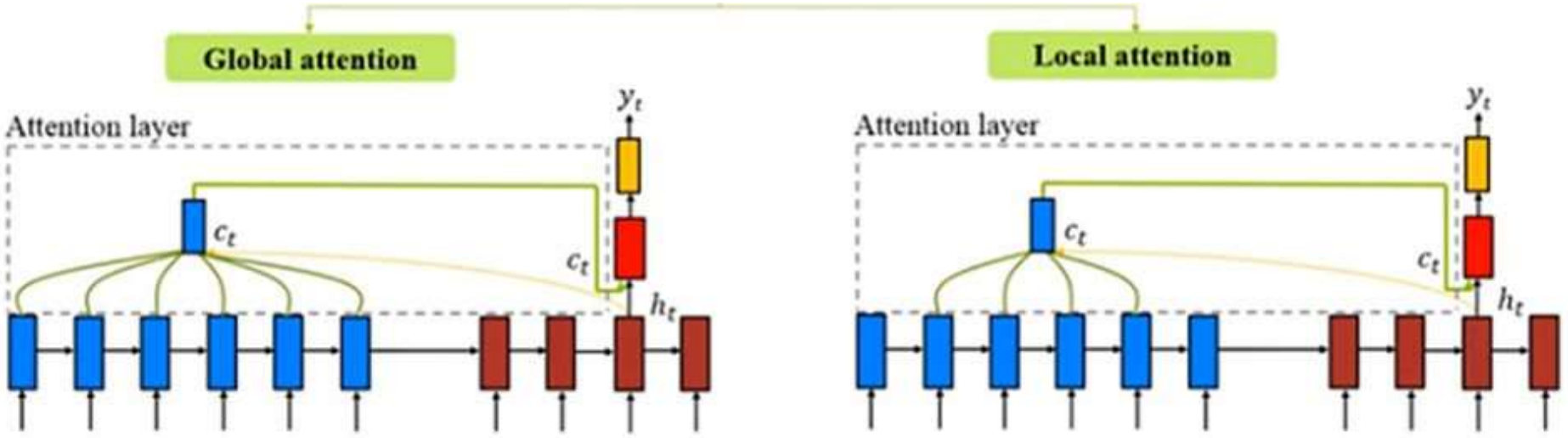
S = decoder hidden state, T = target word embedding.

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.*

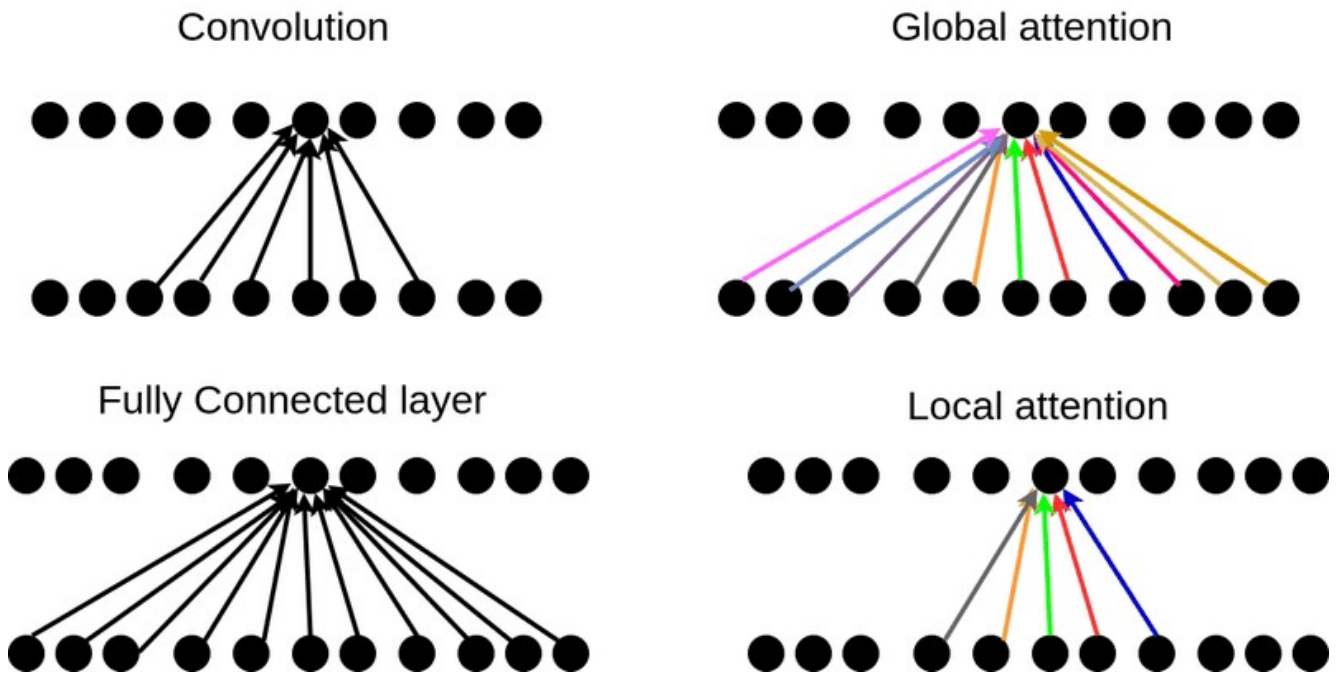
Types of attention

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

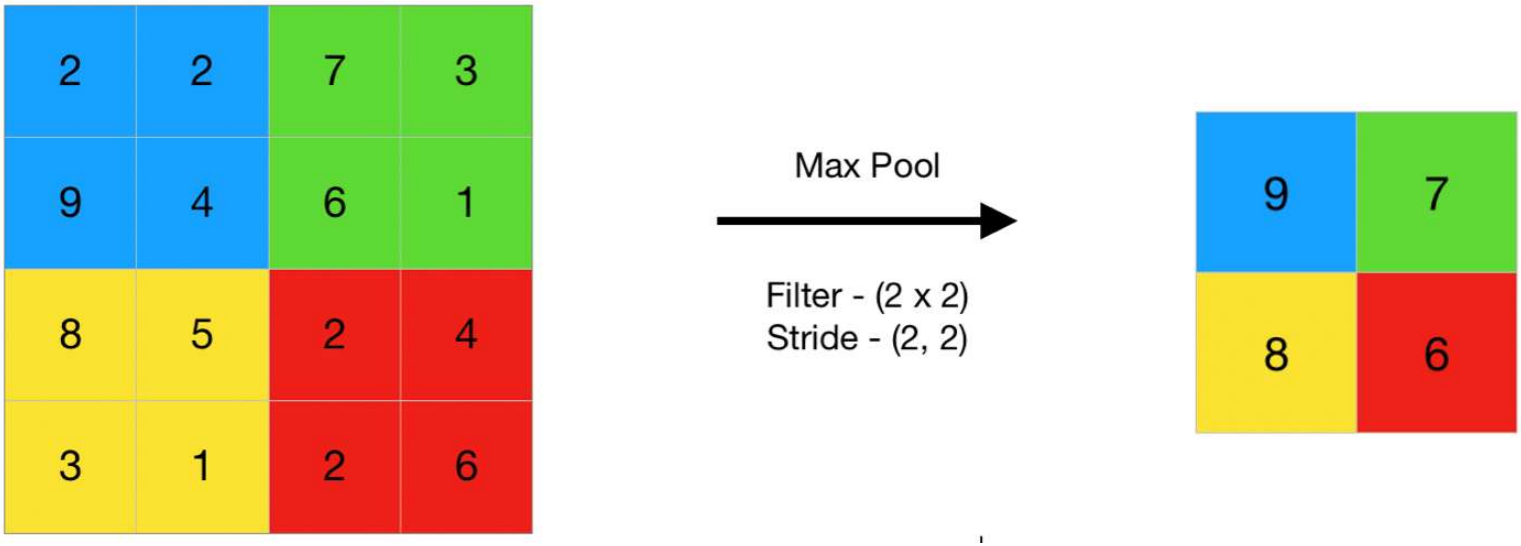
to reduce the spatial dimensions of the feature maps(subsampling) and to provide small spatial invariances

- Max pooling
- Min pooling
- Average pooling
- Global Pooling
- Spatial Pyramid Pooling
- **Region of Interest Pooling**

Other pooling extensions

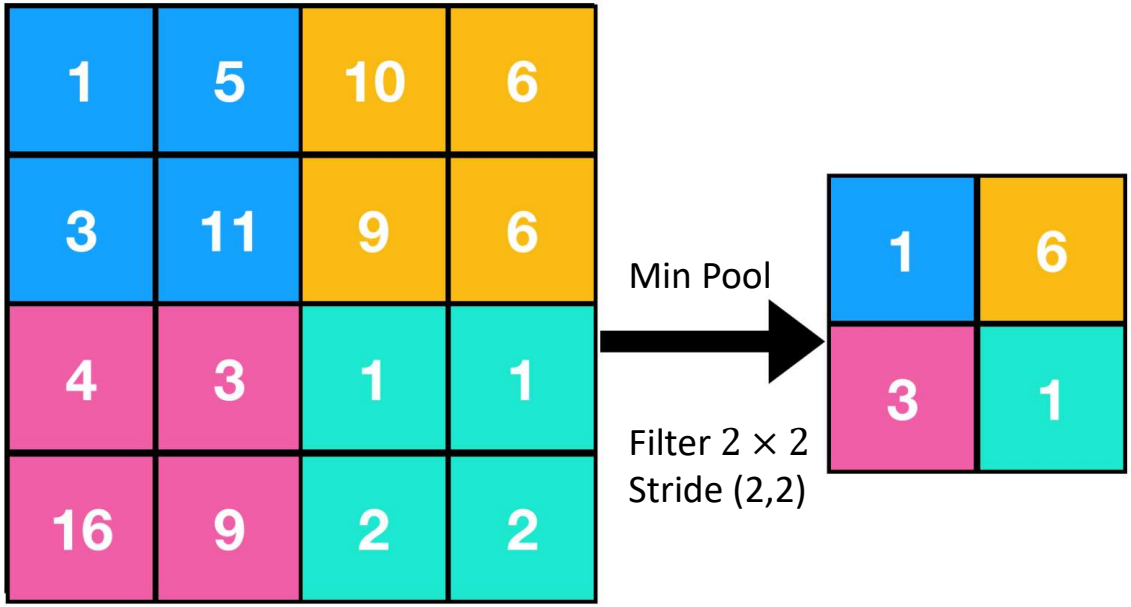
Mixed Pooling
Lp pooling
Stochastic pooling
Spatial Pyramid Pooling
Region of Interest Pooling
Multi-scale order-less pooling (MOP)
Super-pixel Pooling
PCA Networks
Compact Bilinear Pooling
Lead Asymmetric Pooling (LAP)
Edge-aware Pyramid Pooling
Spectral Pooling
Row-Wise Max-Pooling
Intermap Pooling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

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



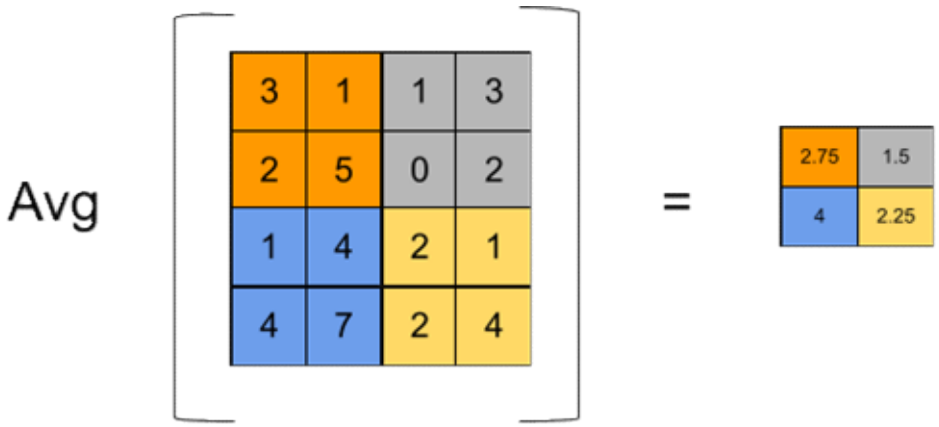
- Through Max Pooling, among units of a patch at each feature map, choose one which has the maximum similarity to the filter and ignore other ones.

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)

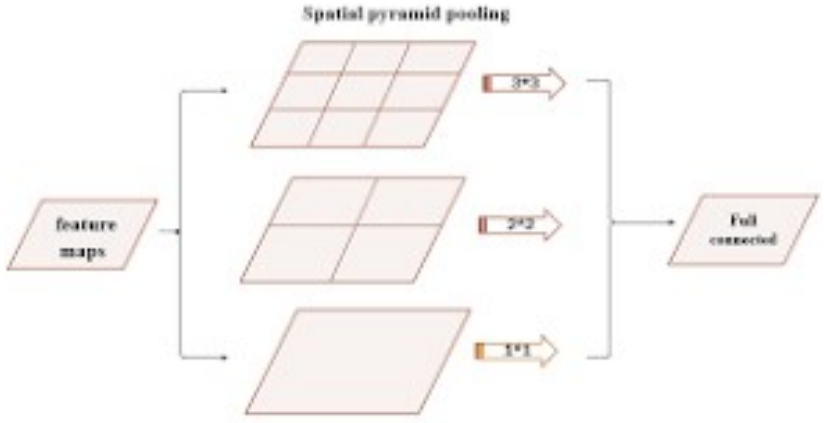
- Assuming all units of a patch are almost equal in information but with additive mean-zero noise, average pooling performs as a smooth function increasing the signal to noise ratio.

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.

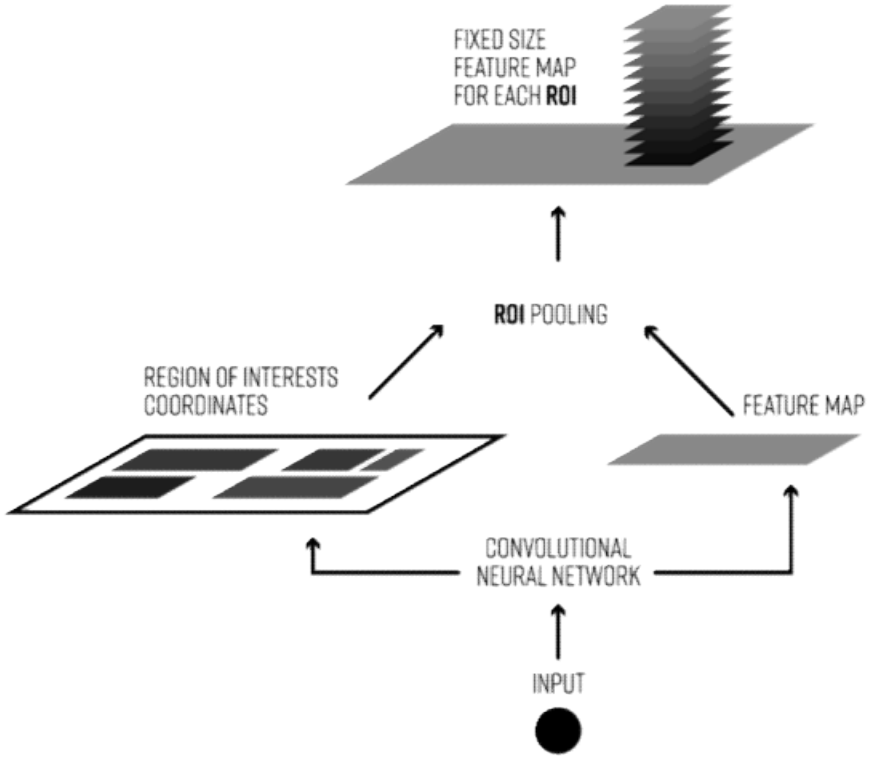
Pyramid Pooling

- Spatial Pyramid Pooling (SPP) is **a pooling layer that removes the fixed-size constraint of the network**, i.e. a CNN does not require a fixed-size input image. Specifically, we add an SPP layer on top of the last convolutional layer.



Region of Interest Pooling

- Region of Interest Pooling, or RoIPool, is **an operation for extracting a small feature map (e.g., 7 × 7) from each RoI in detection and segmentation based tasks**. Features are extracted from each candidate box, and thereafter in models like Fast R-CNN, are then classified and bounding box regression performed.



1.1.6 Normalizing Layers

to speed up and stabilize the learning process

- Batch Normalization
- Weight Normalization
- Layer Normalization
- Group/Instance Normalization
- Weight Standarization
- **Local Response Normalization**

*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: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

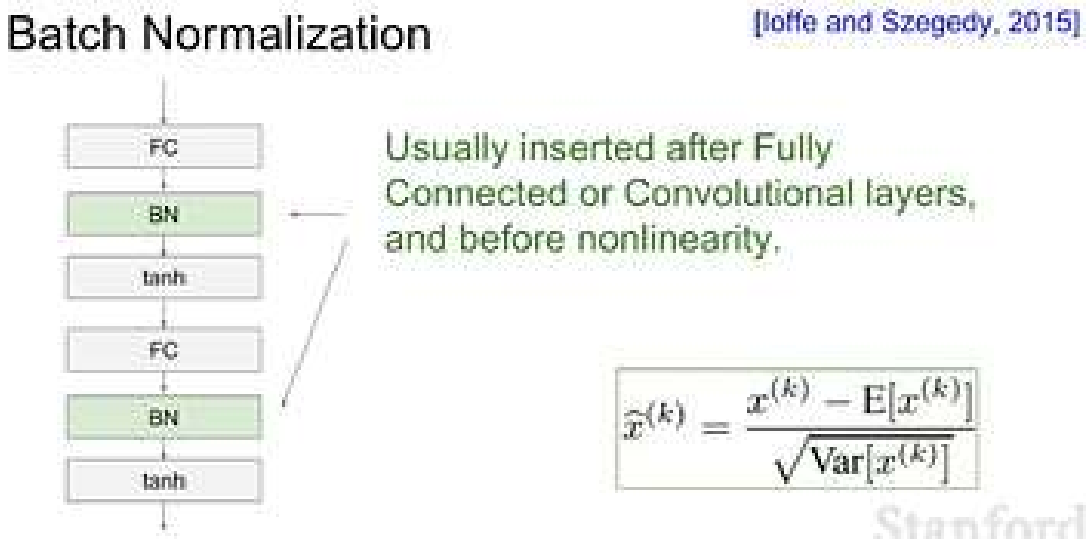
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

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.

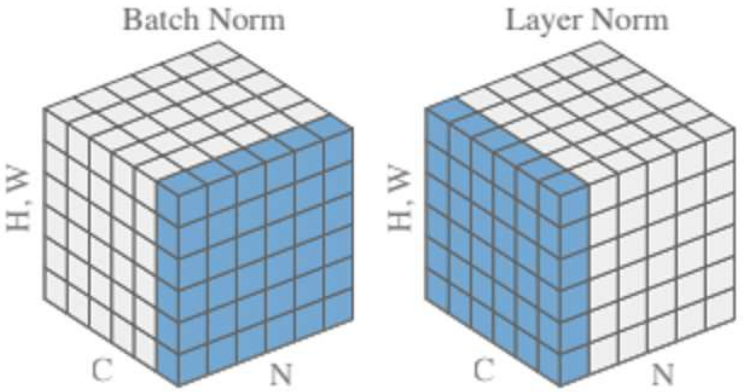
A Comparison between Batch Norm. and Wight Norm.

	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)

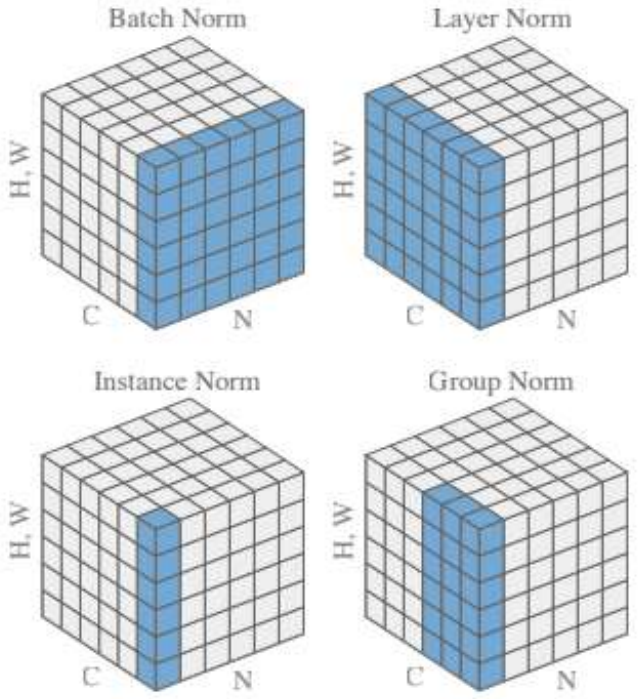
H,W: spatial size of feature maps

* Geoffrey Hinton et al. 2016

Group/Instance Normalization(GN)

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

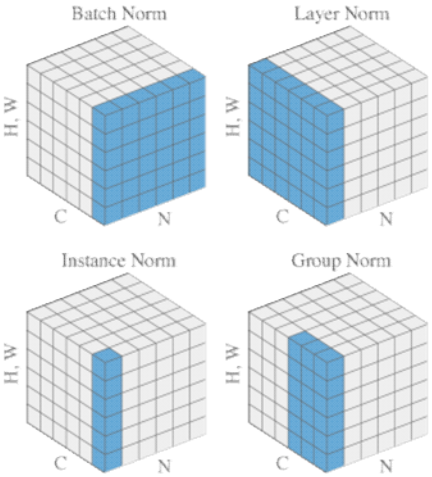
- Similar to layer Normalization, Group Normalization is also applied along the feature direction but unlike LN, it divides the features into certain groups and normalizes each group separately.
- In practice, Group normalization performs better than layer normalization.
- its parameter *num_groups*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

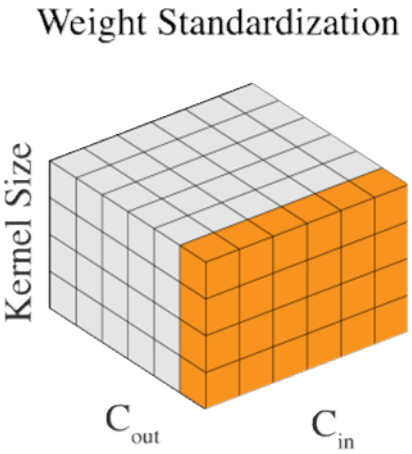


Weight standardization

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).





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

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

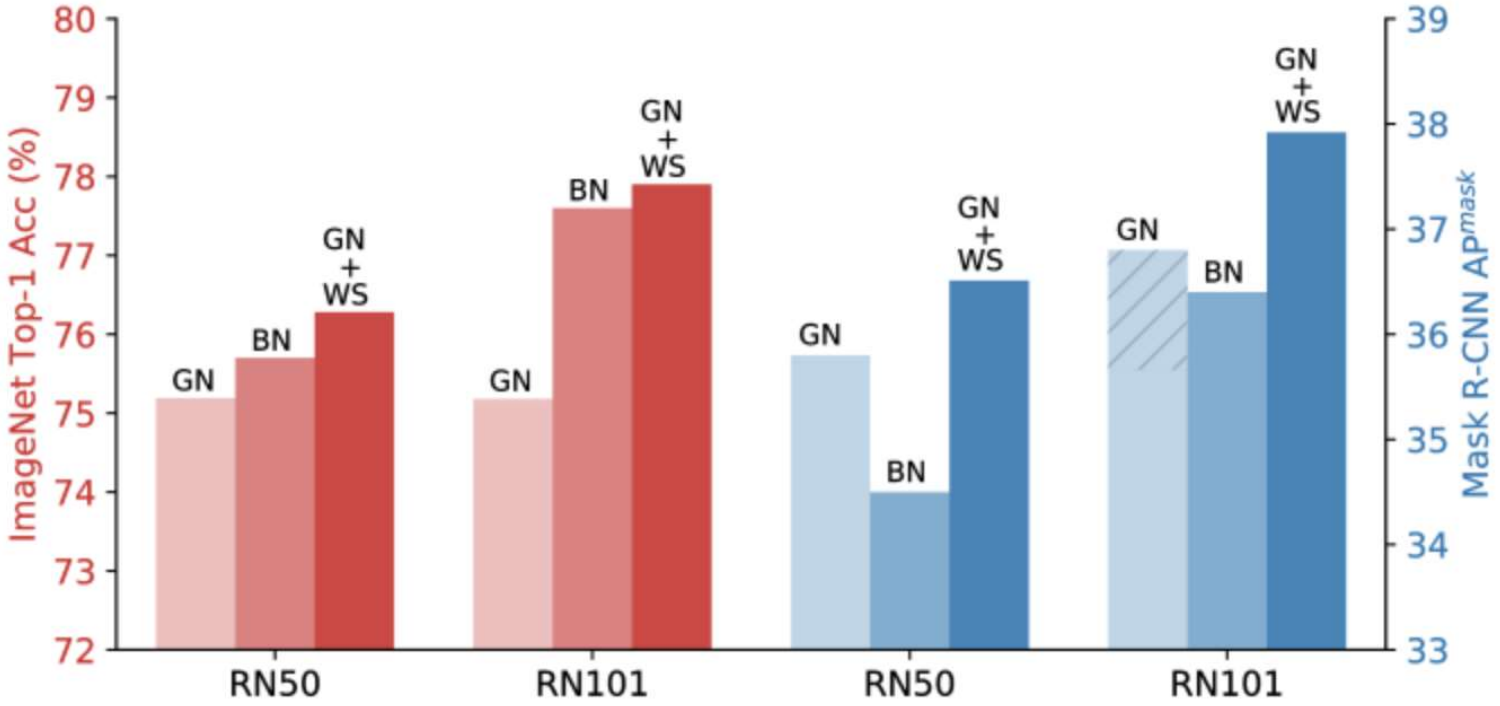
$$\mu_{\boldsymbol{W}_{i,\cdot}} = \frac{1}{I}\sum_{j=1}^{I} \boldsymbol{W}_{i,j}, \quad \sigma_{\boldsymbol{W}_{i,\cdot}} = \sqrt{\frac{1}{I}\sum_{i=1}^{I}(\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}})^2}$$

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

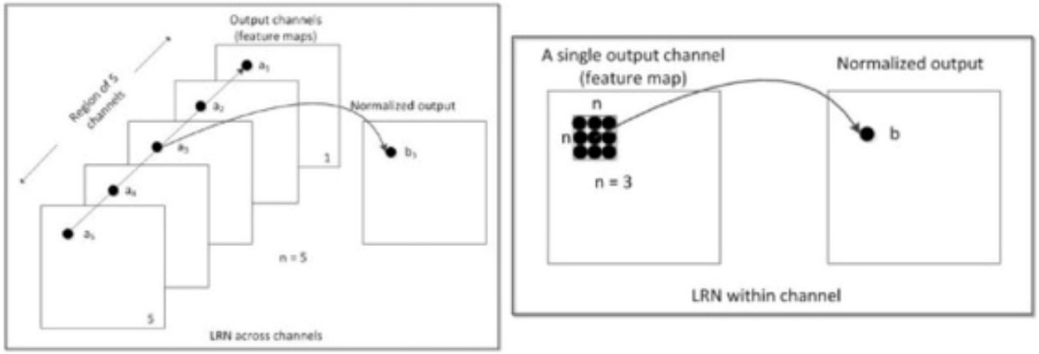


*Siyuan Qiao et al. Weight Standardization. GN+WS effect on classification and object detection task[4]

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



1.2 Architectures

1.2.1 CNNs

1.2.2 Region based CNNs (R-CNNs)

1.2.3 DNNs for Segmentation

1.2.4 Transformers

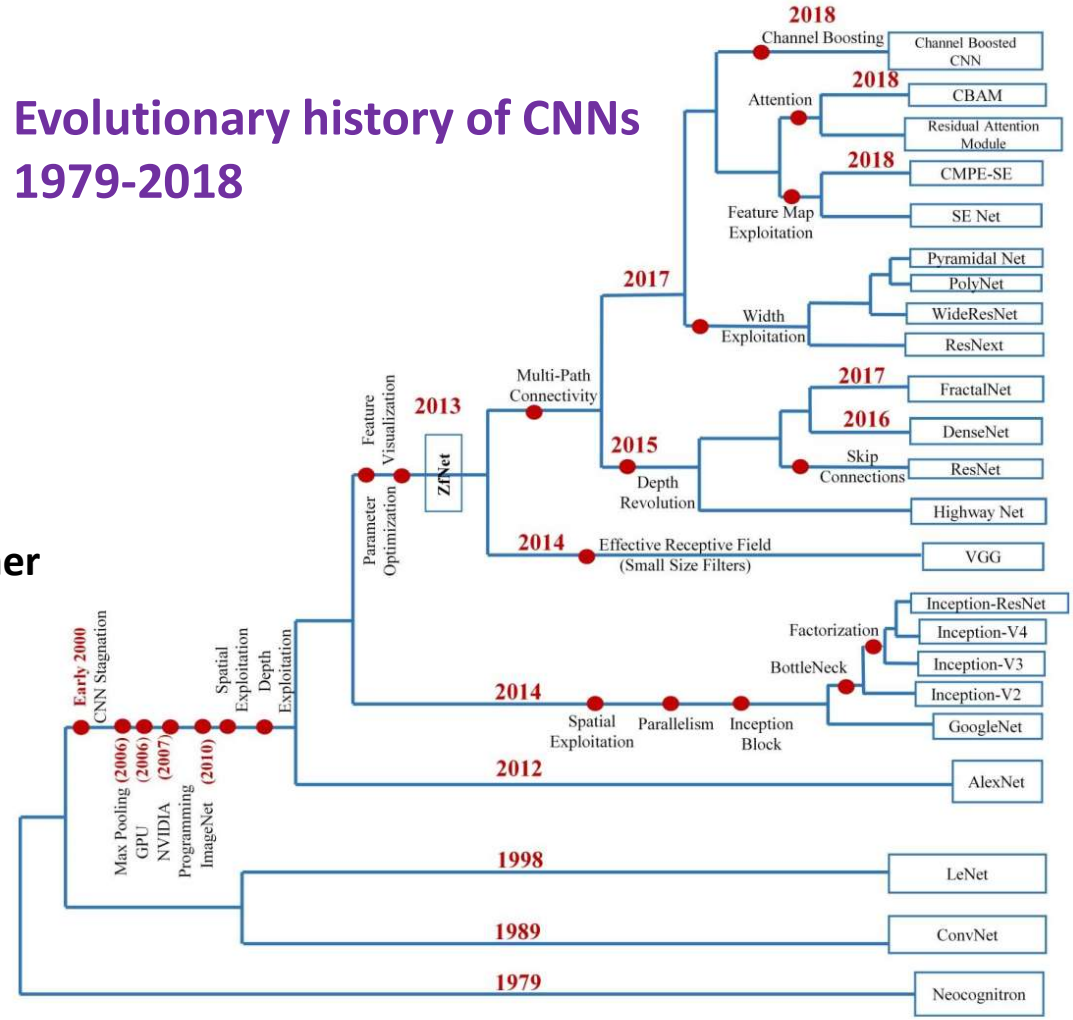
1.2.1 CNNs

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**

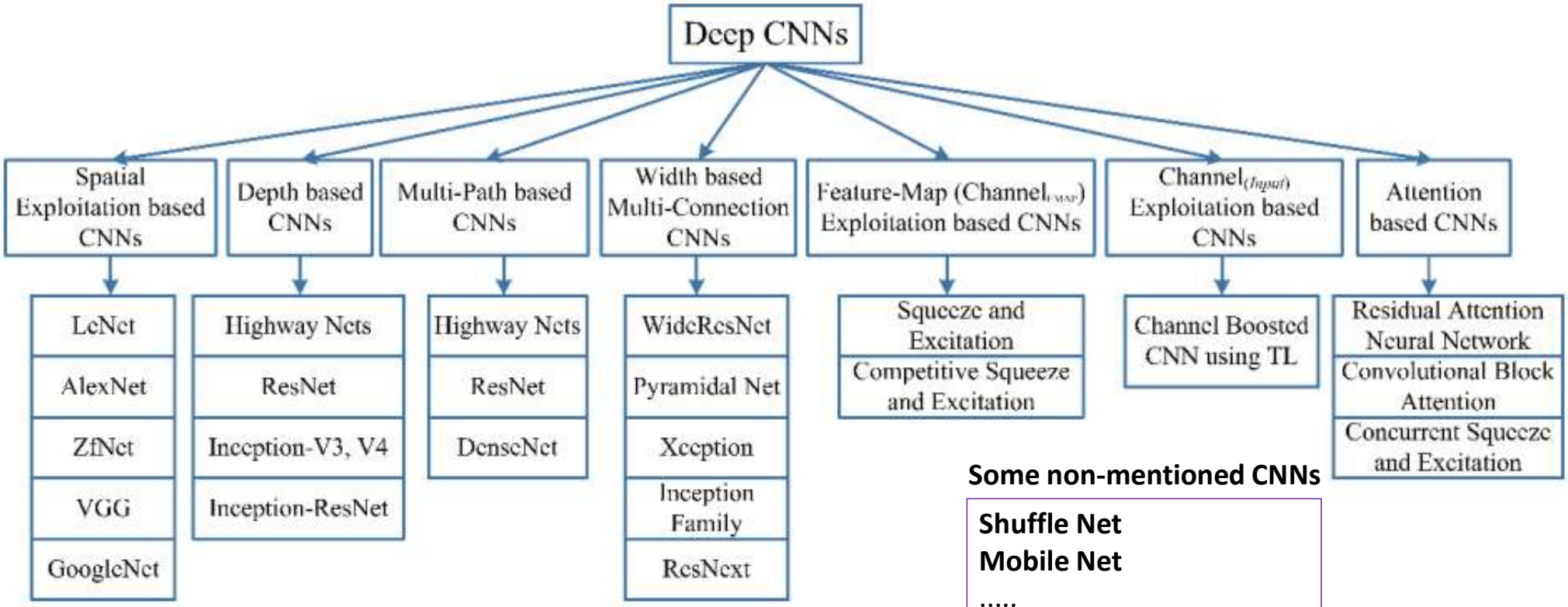
In 1998, LeCuN proposed an improved version of ConvNet, which was famously known as LeNet-5, and it started the use of CNN in classifying characters in a document recognition related applications (LeCun et al. 1995, 1998).

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

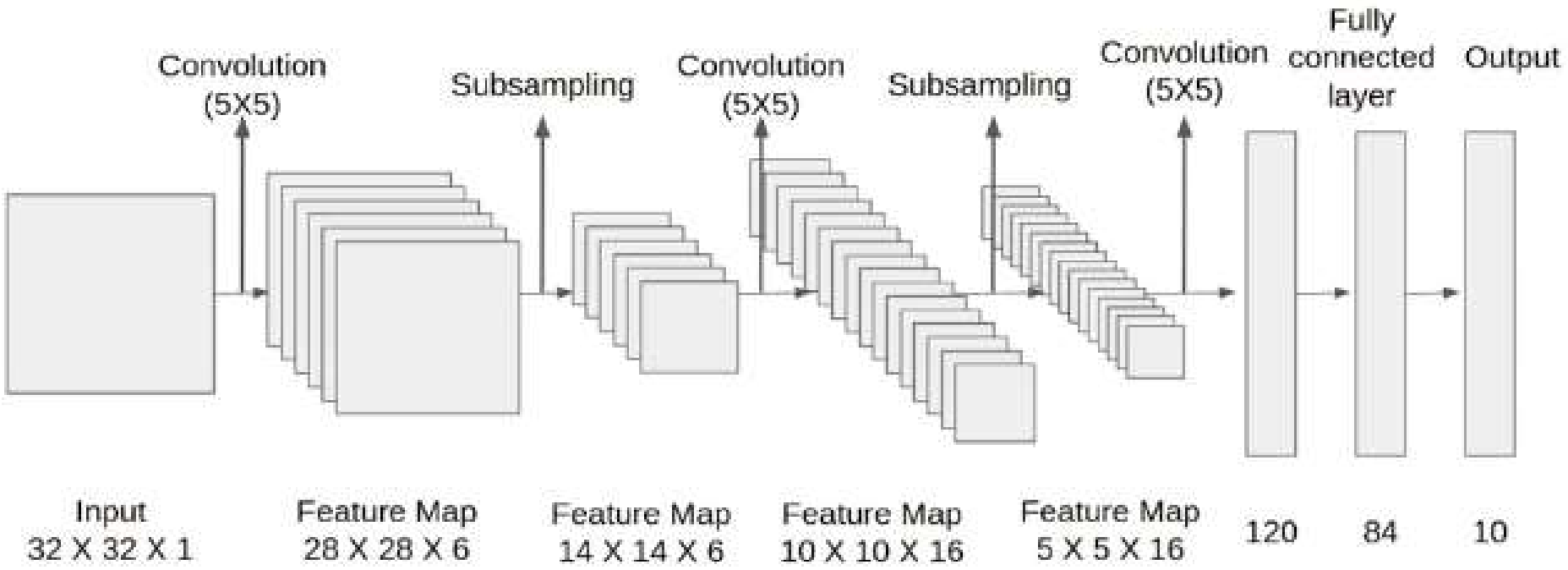
CNNs: **LeNet** **AlexNet** **ZFNet** **VGG** **GoogleNet**

.

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)

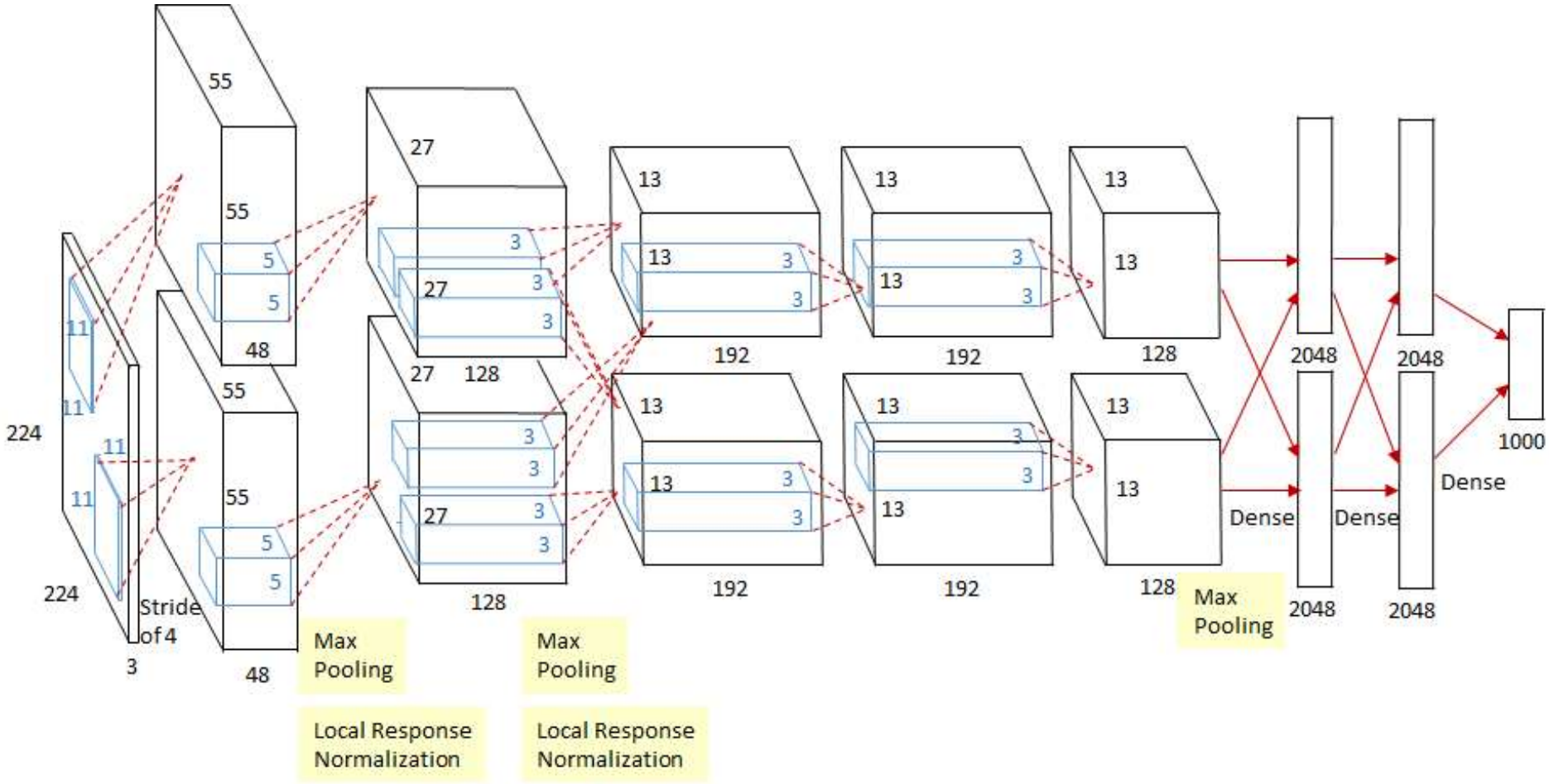
Lenet-5

First popular CNN architecture



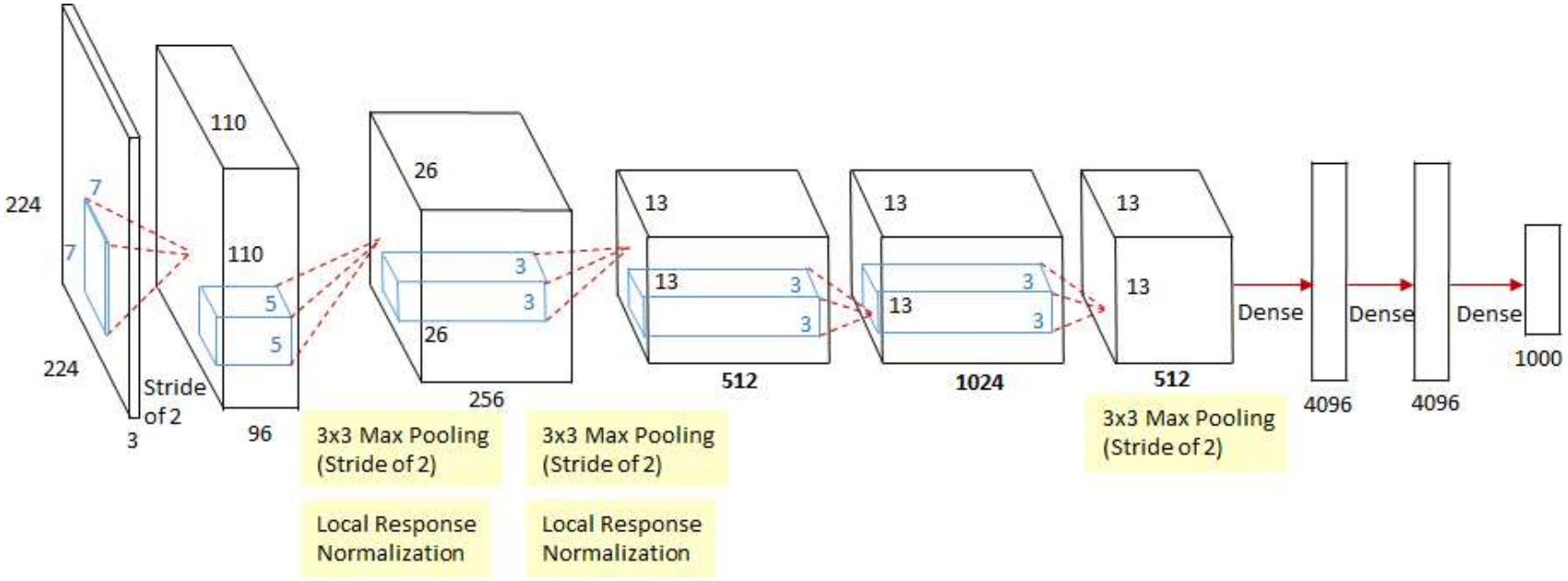
AlexNet

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



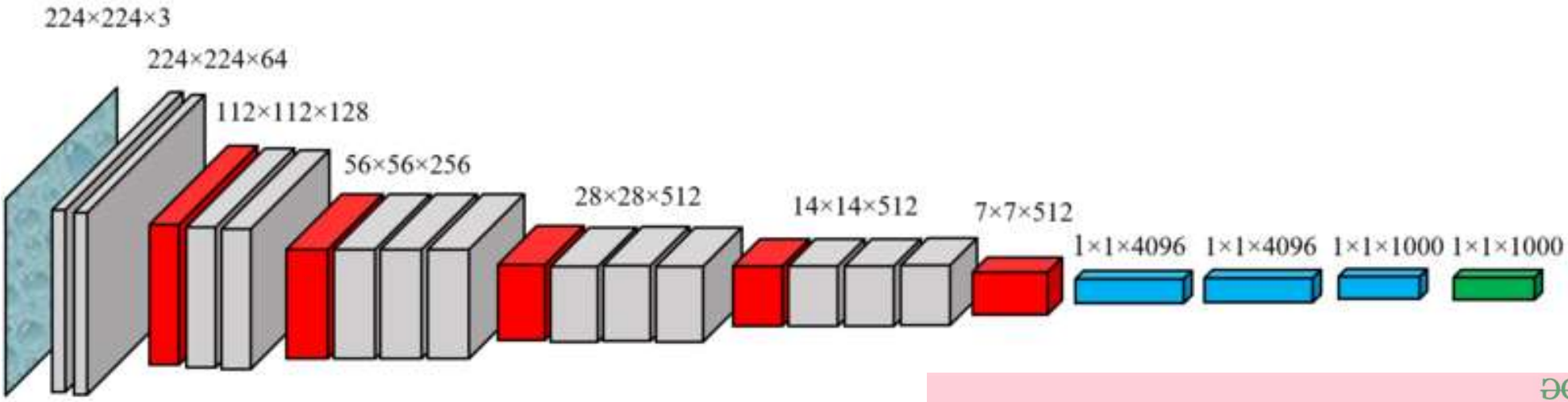
ZFNET

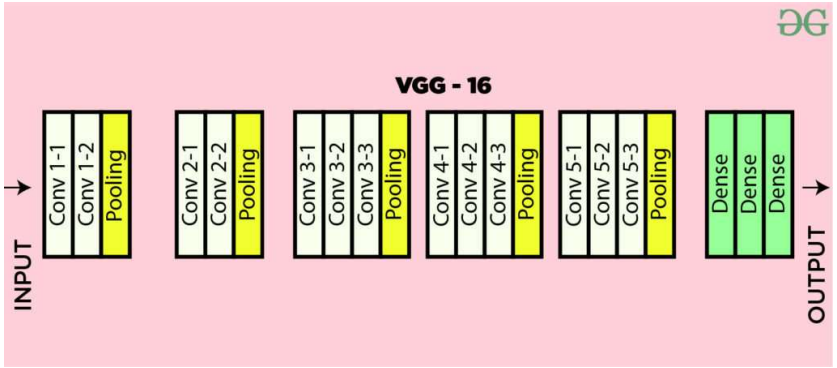
Visualization of intermediate layers



VGG

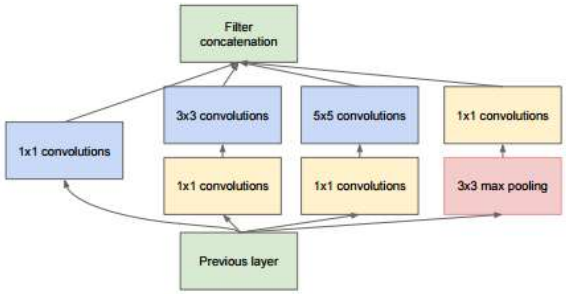
- Homogenous topology
- Uses small size kernels





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.

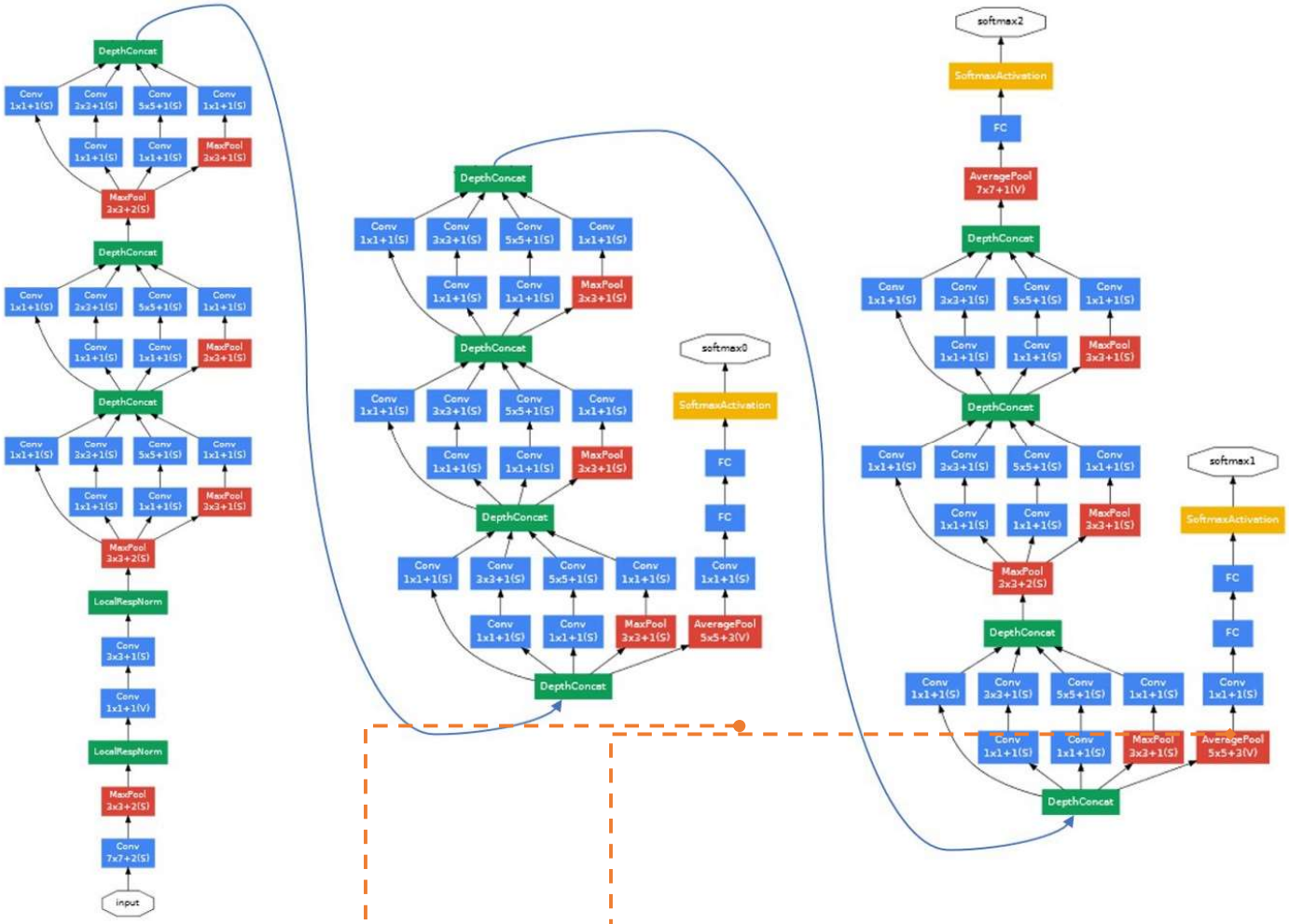


Table 5a Major challenges associated with implementation of Spatial exploitation based CNN architectures.

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

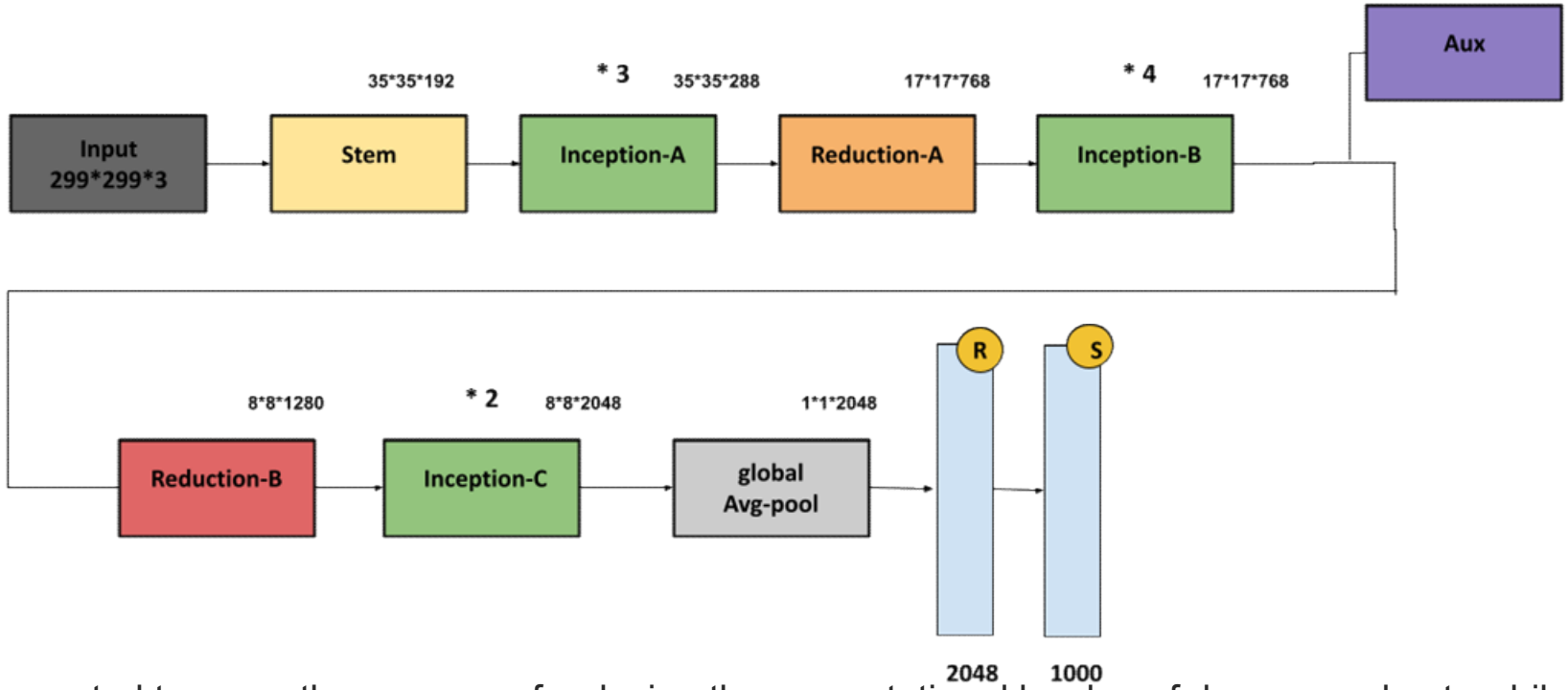
CNNs which improve themselves by increasing the depth to utilize more cascaded filters and to achieve better feature representation

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

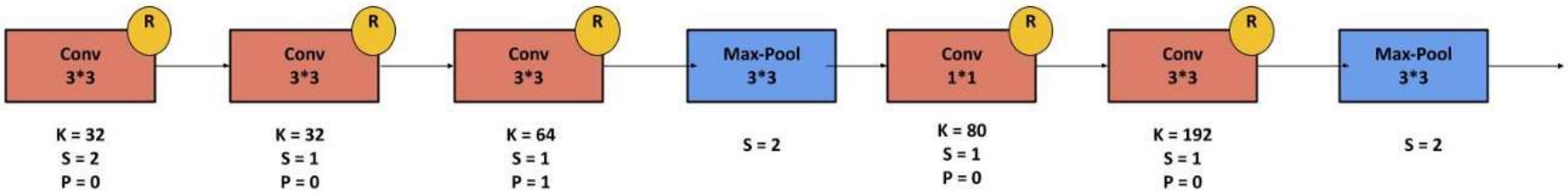
Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6	159	Depth + Width	(Szegedy et al. 2016b)
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

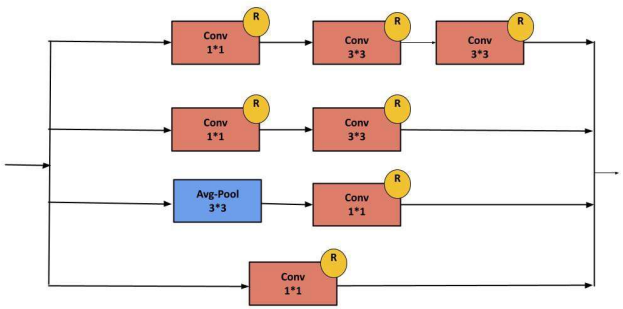


Inception is created to serve the purpose of reducing the computational burden of deep neural nets while obtaining state-of-art performance.

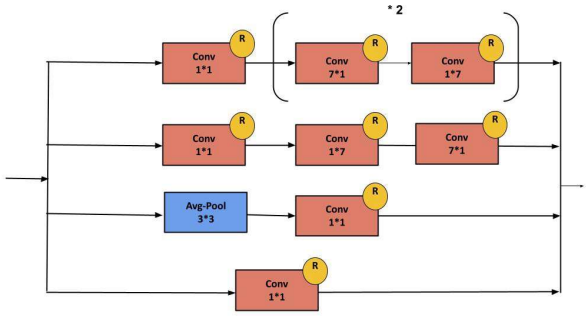


STEM

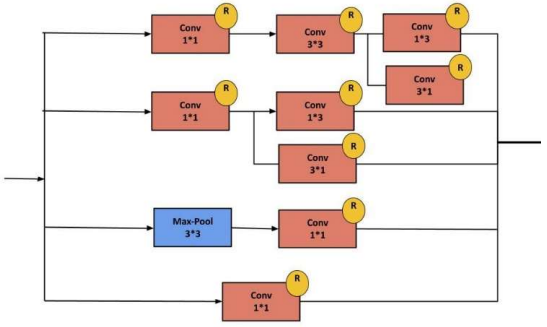
The Stem is a particular convolutional network module before the Inception-resnet blocks



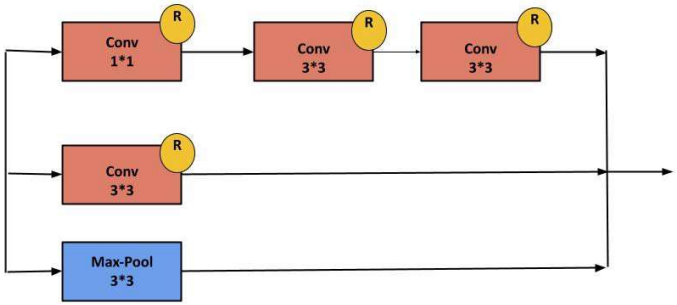
Inception-A Block :



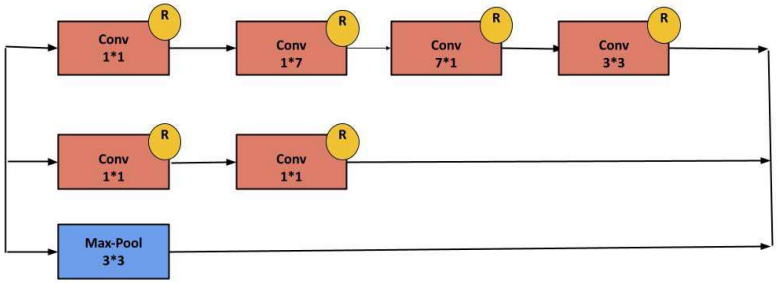
Inception-B Block :



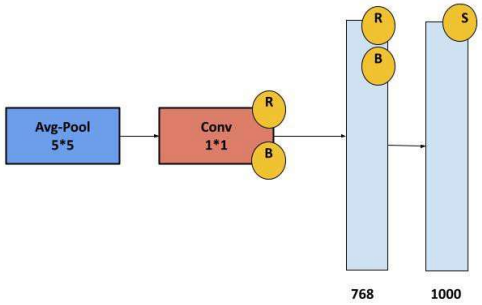
Inception-C Block :



Reduction-A Block :



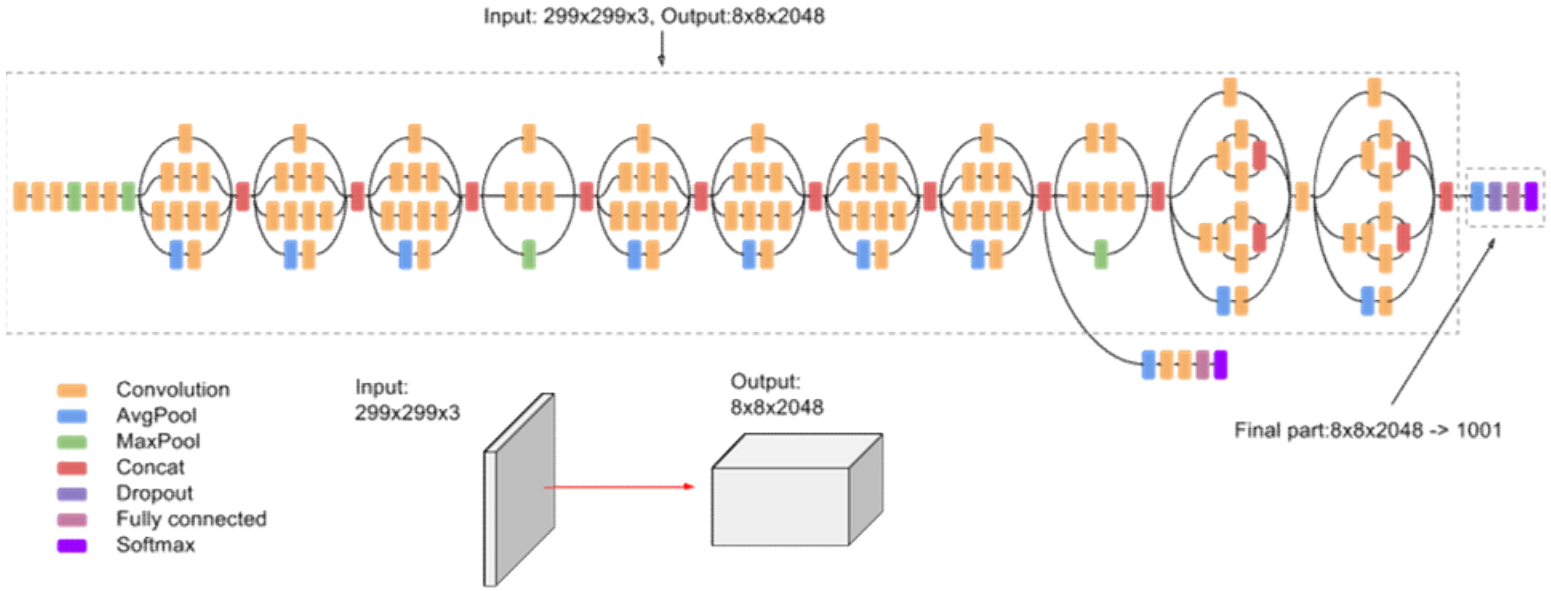
Reduction-B Block :



Auxiliary Classifier Block :

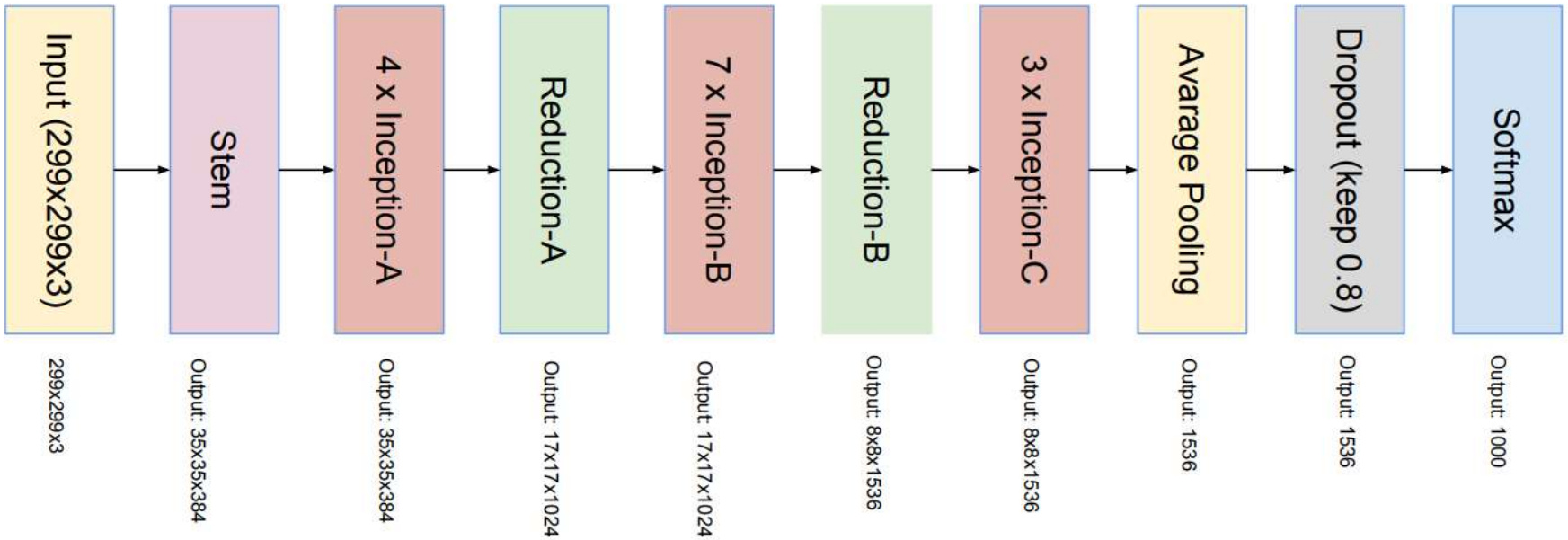
Auxiliary Classifiers are **type of architectural component that seek to improve the convergence of very deep networks**. They are classifier heads we attach to layers before the end of the network.

Inception 3

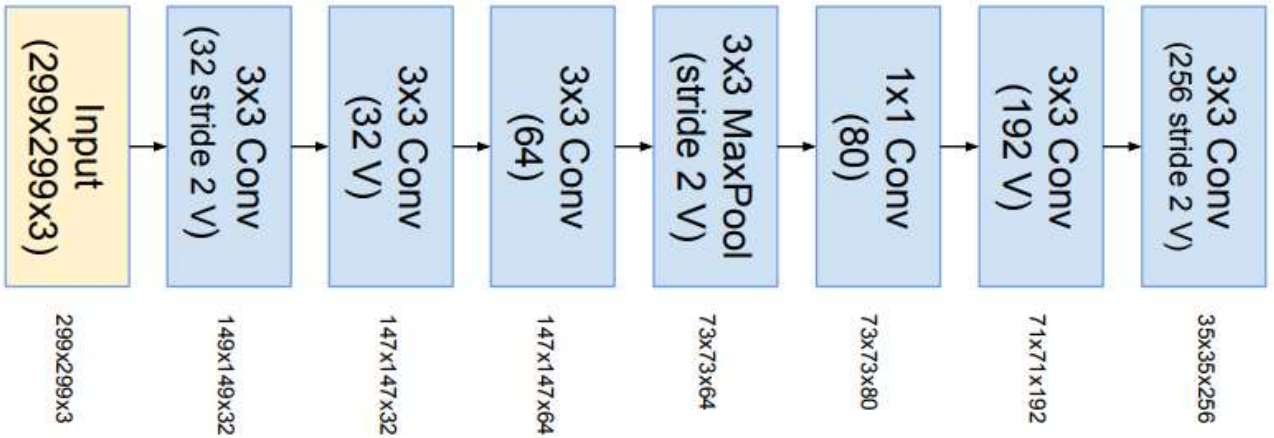

Input: 299x299x3, Output:8x8x2048
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax
Input:
299x299x3
Output:
8x8x2048
Final part:8x8x2048 -> 1001

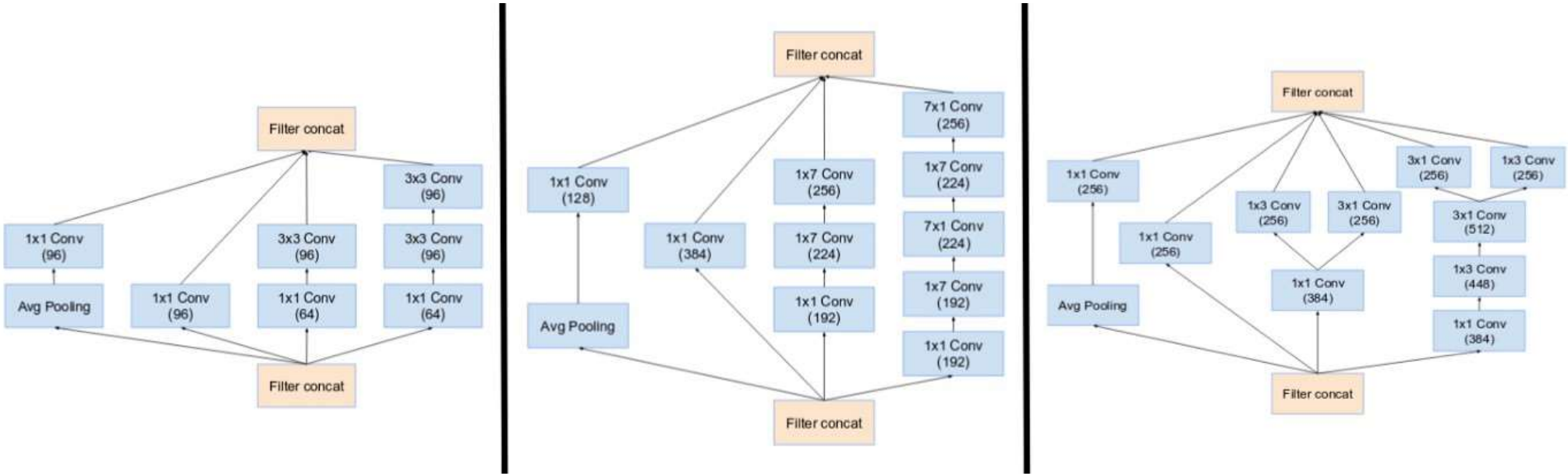
Inception 4

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

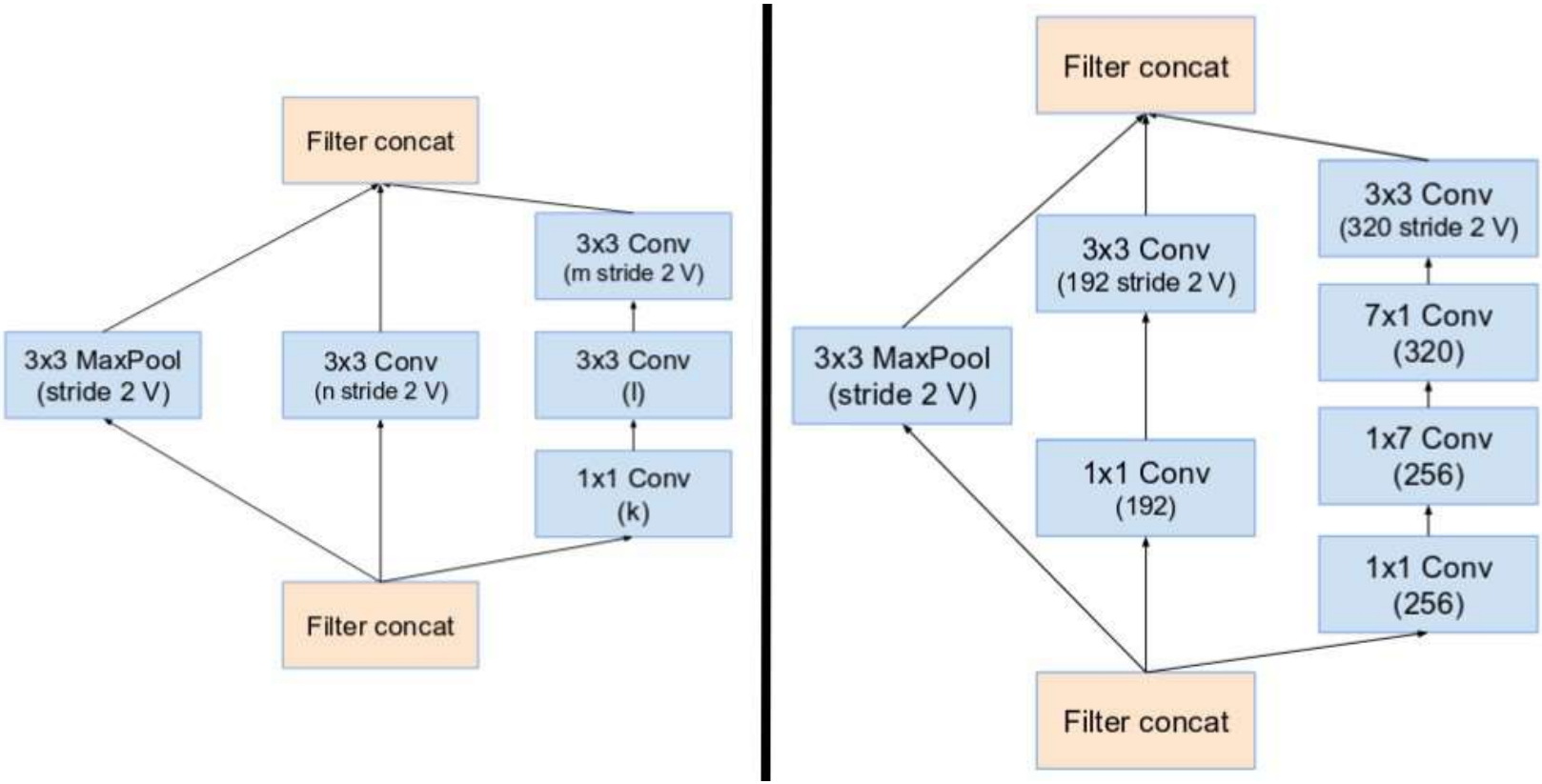


STEM


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



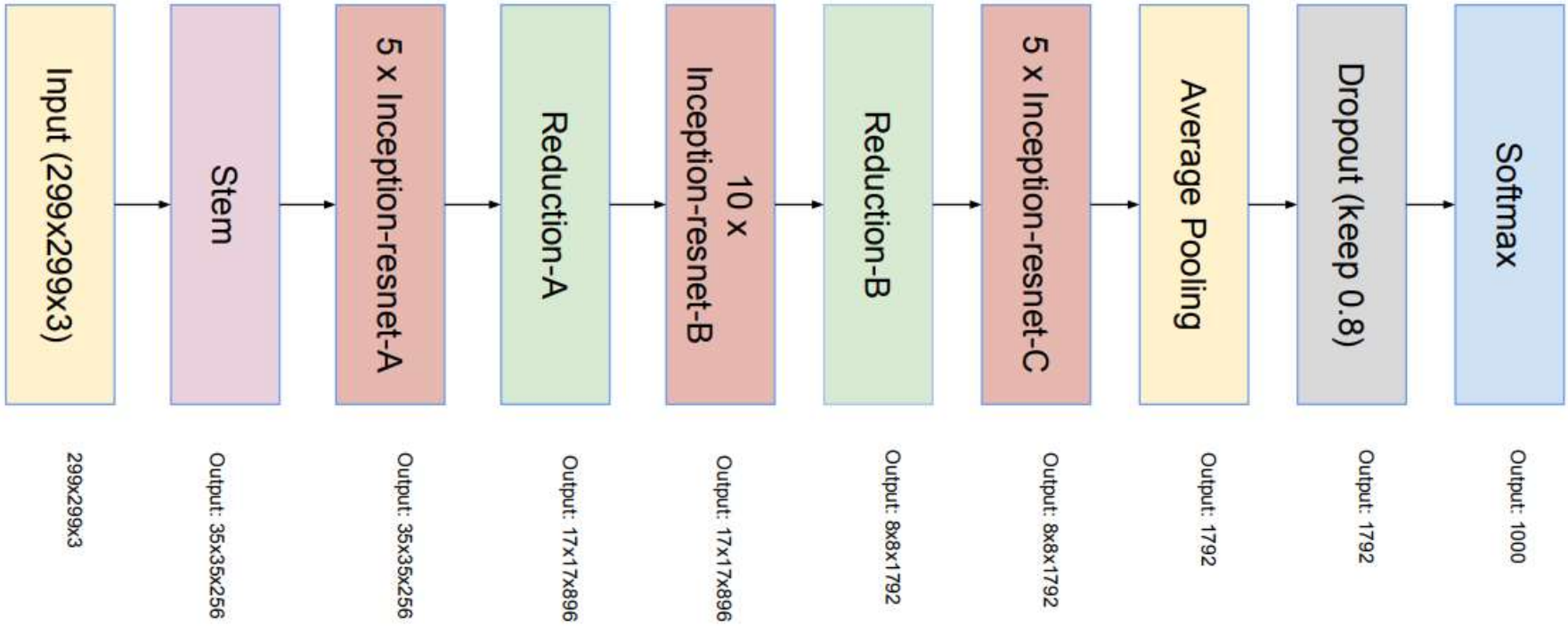
Inception Modules A, B, C of Inception-v4



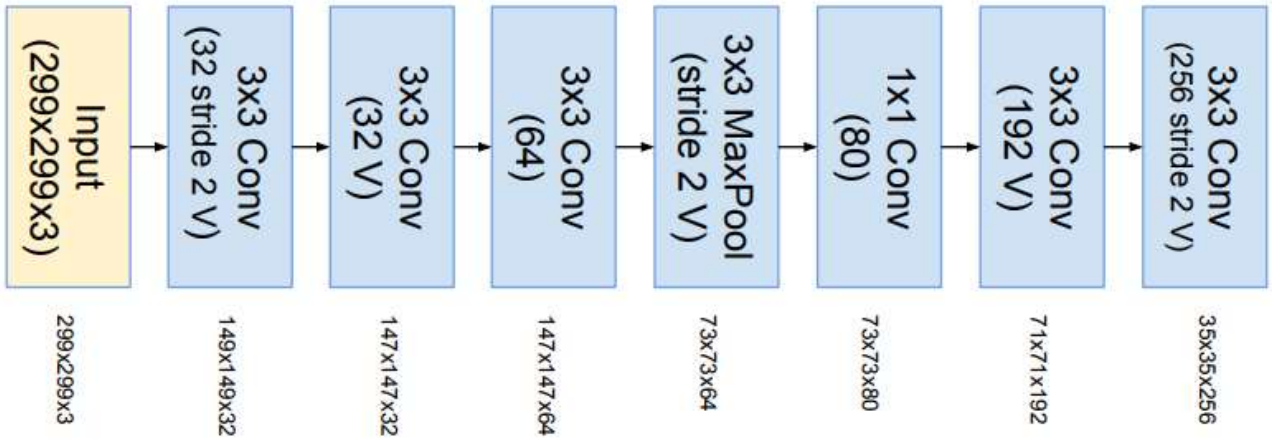
Reduction Blocks A, B of Inception-v4

Inception Resnet

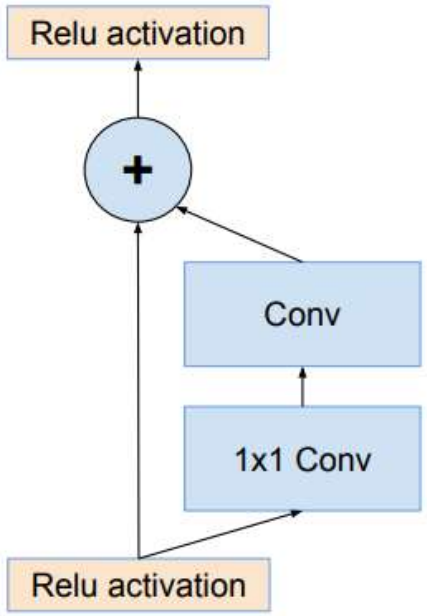
- Uses split transform merge idea and residual links

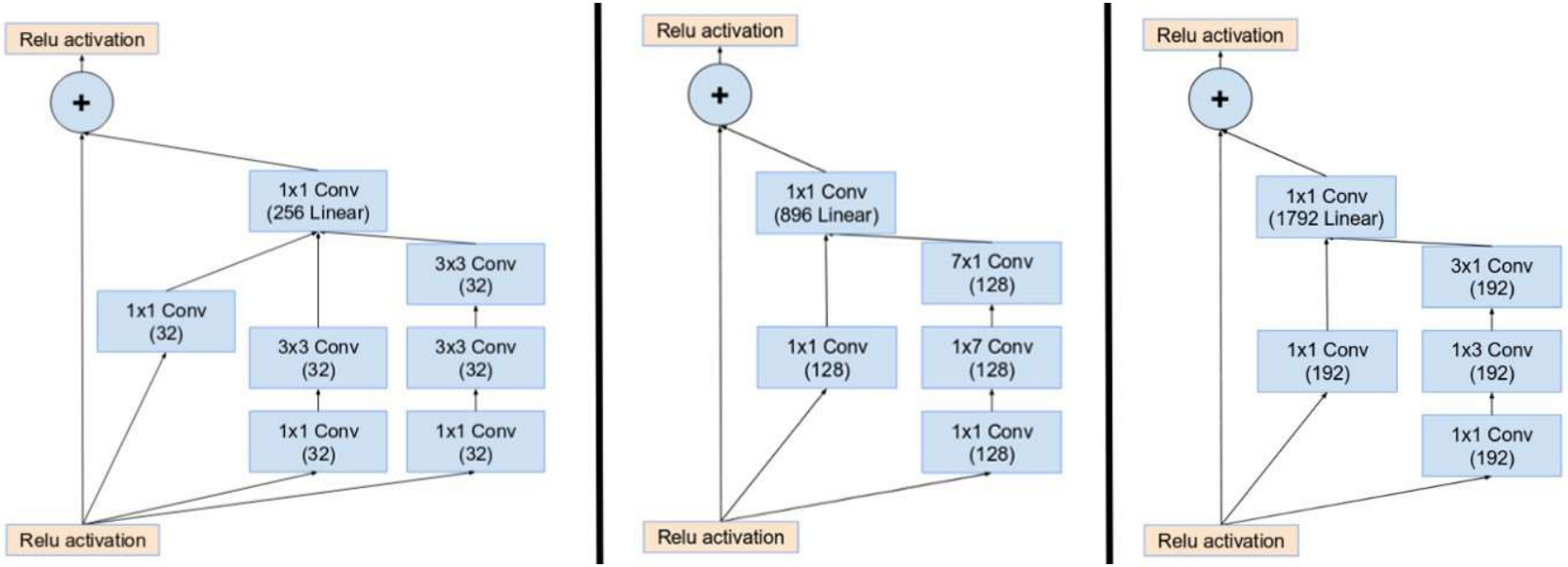


STEM

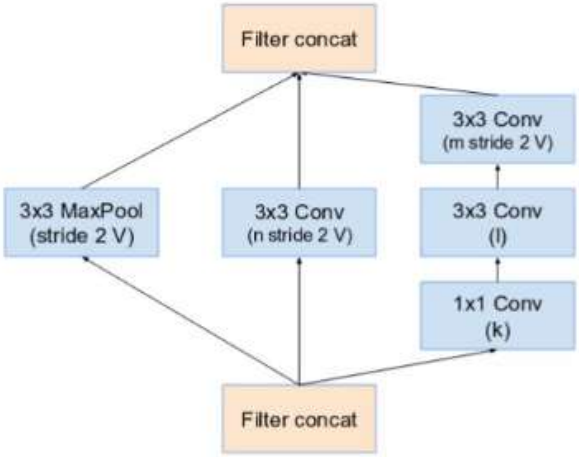


Inception Resnet Conncetion

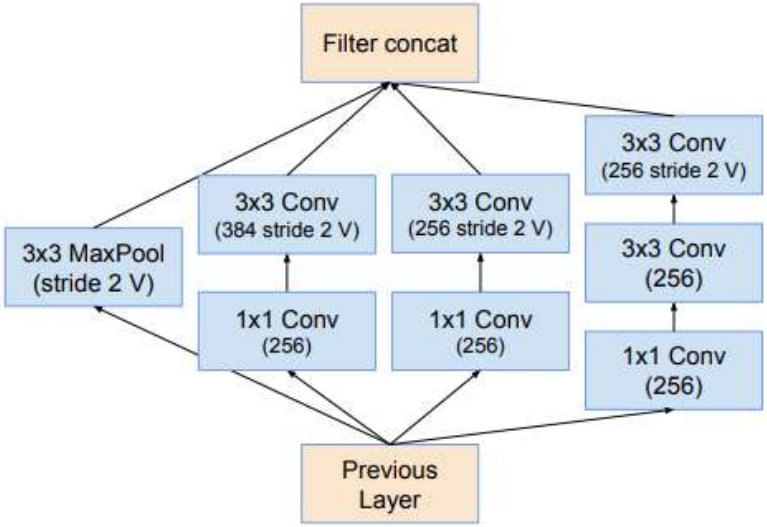




Inception modules A, B, C of Inception ResNet V1



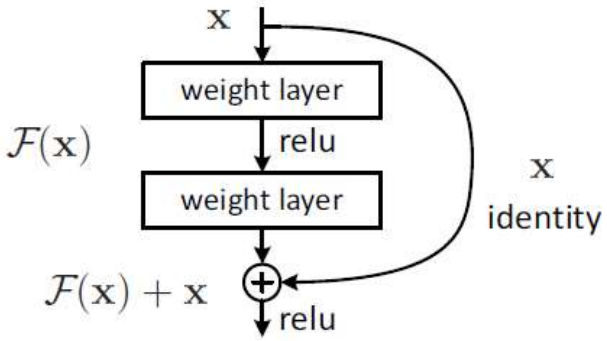
Reduction A schema



Inception ResNet-v1 Reduction Block B

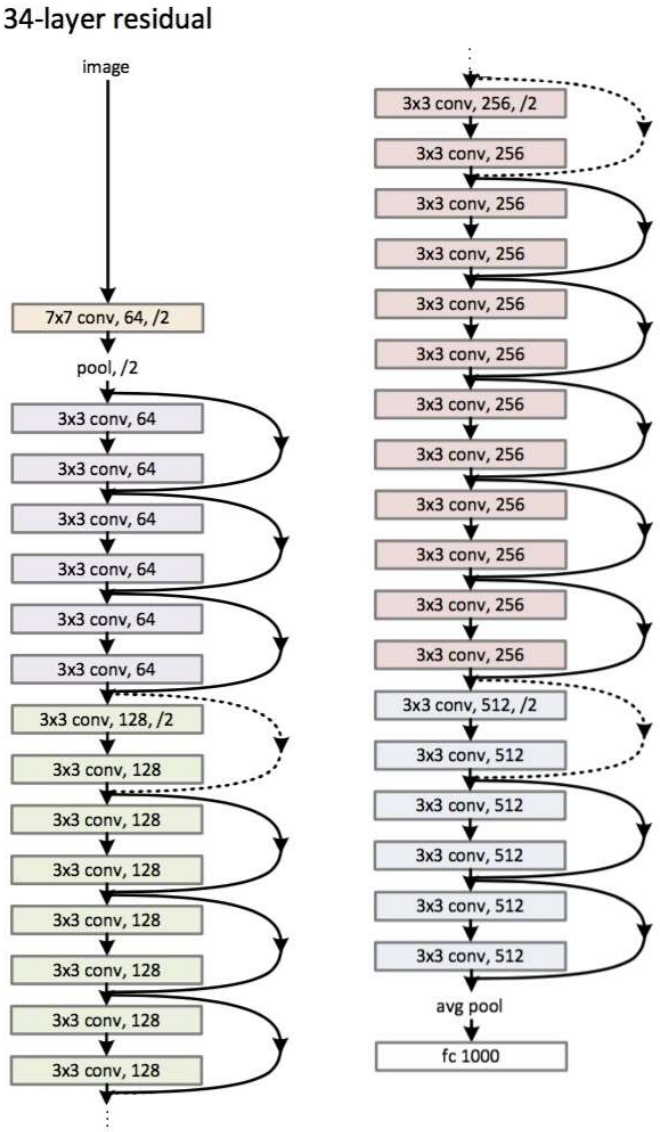
Resnet

- Residual learning
- Identity mapping based skip connections



Residual Blocks

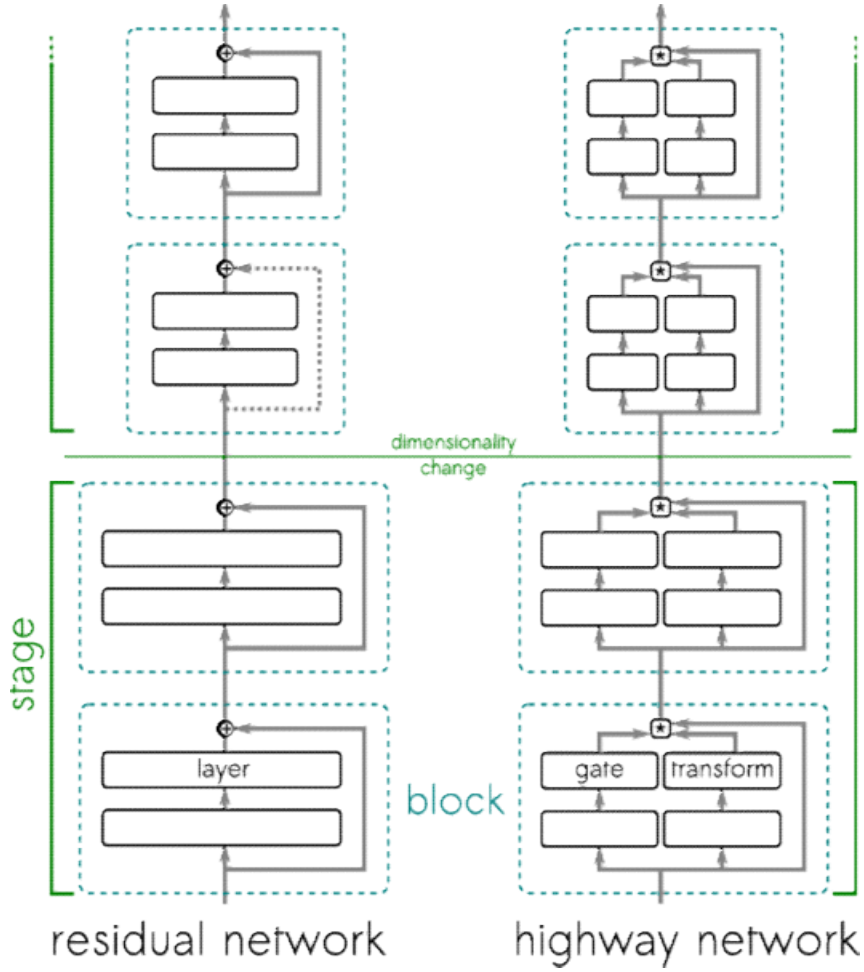
34-layers residual



Highway Network*

- Introduced an idea of Multi-path

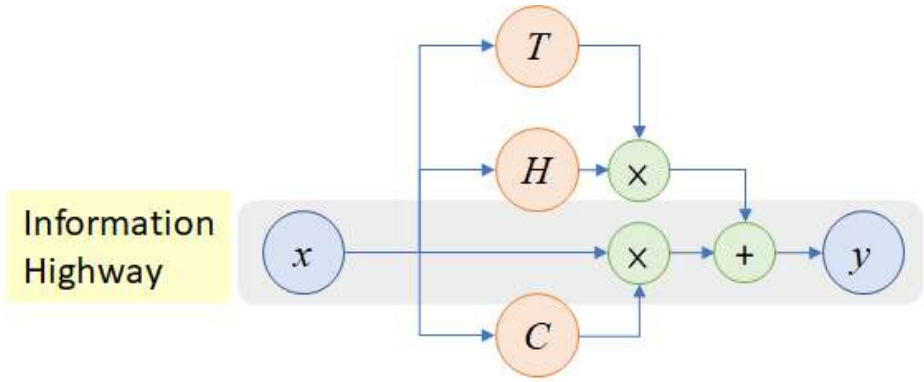
- In machine learning, a highway network is **an approach to optimizing networks and increasing their depth**.
- Highway networks use learned gating mechanisms to regulate information flow, inspired by Long Short-Term Memory (LSTM) recurrent neural networks.
- Highway networks have been used as part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

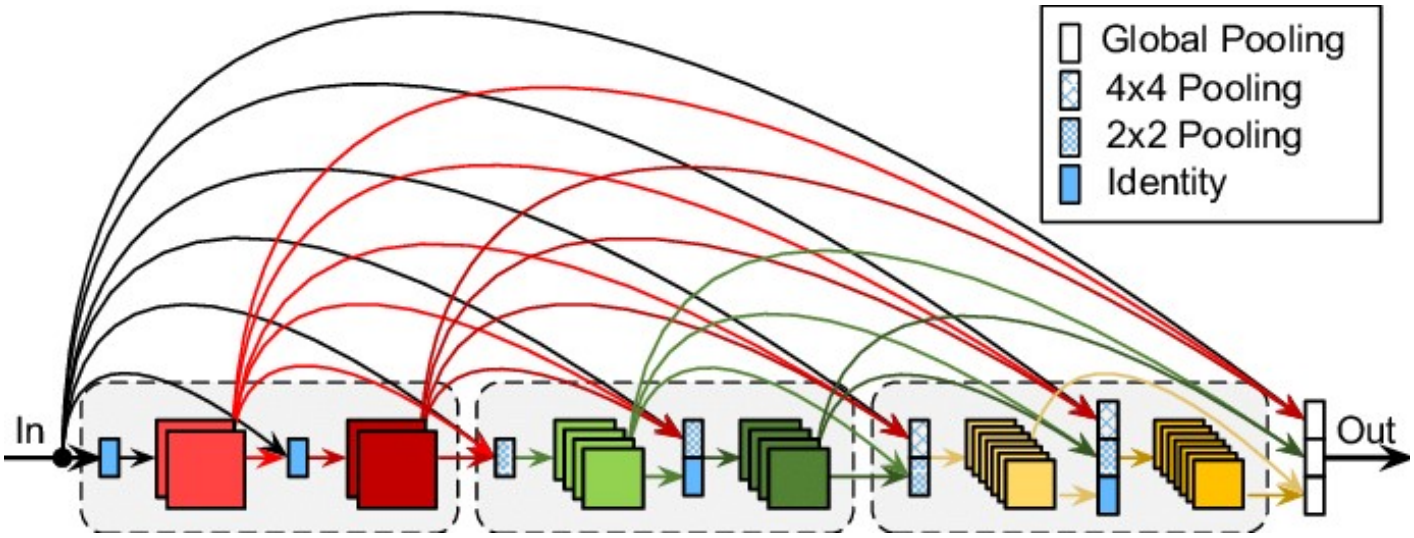
CNNs: Highway Networks ResNet DenseNet

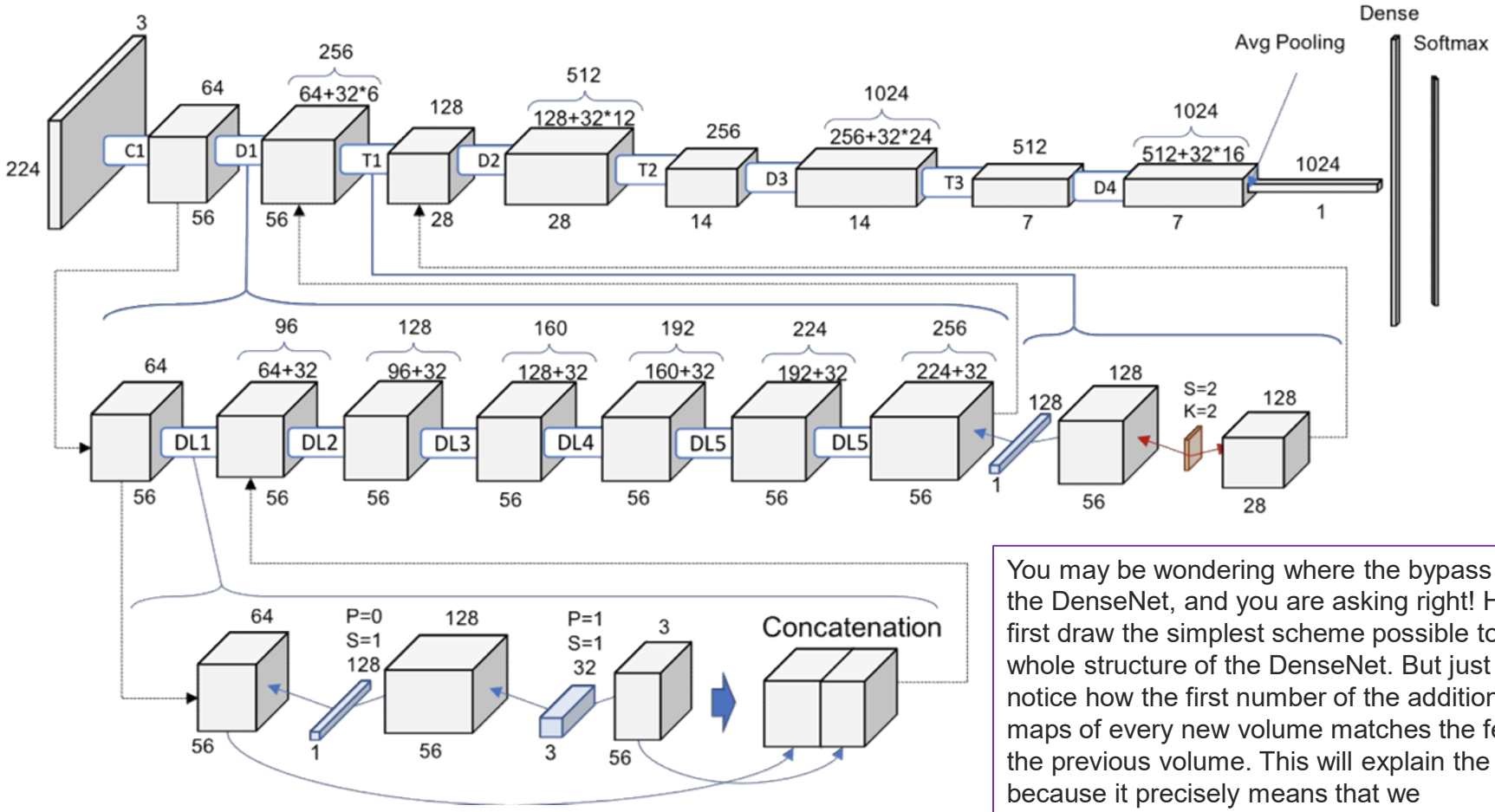
Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76	19	Depth + Multi-Path	(Srivastava et al. 2015a)
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M	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)

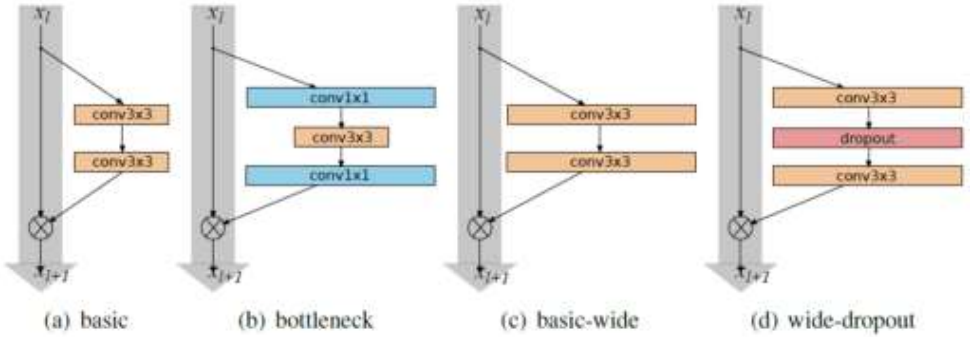
Wide ResNet

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*.

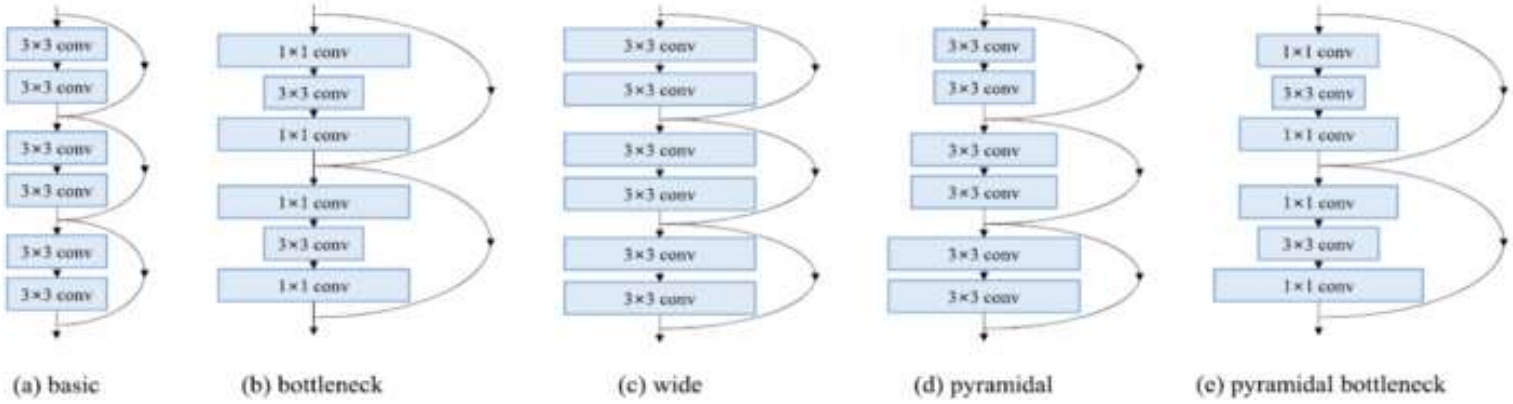


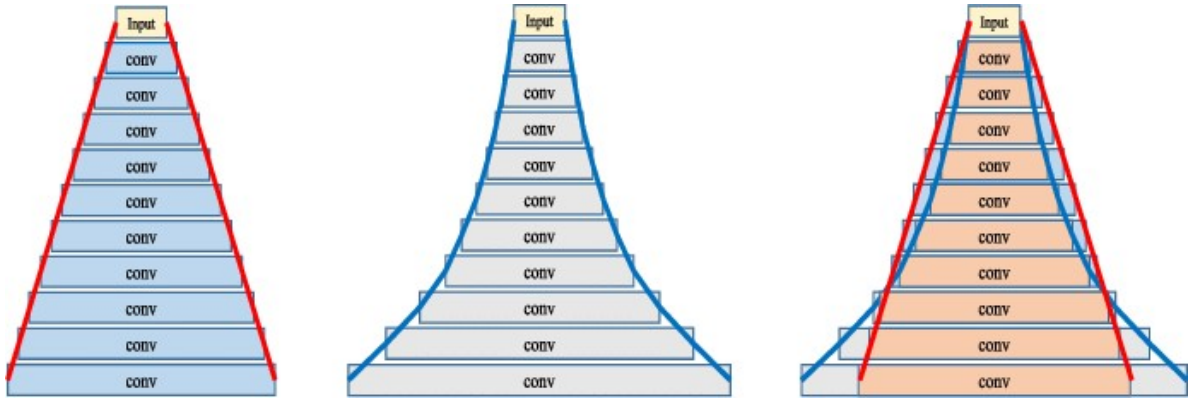
(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

	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,

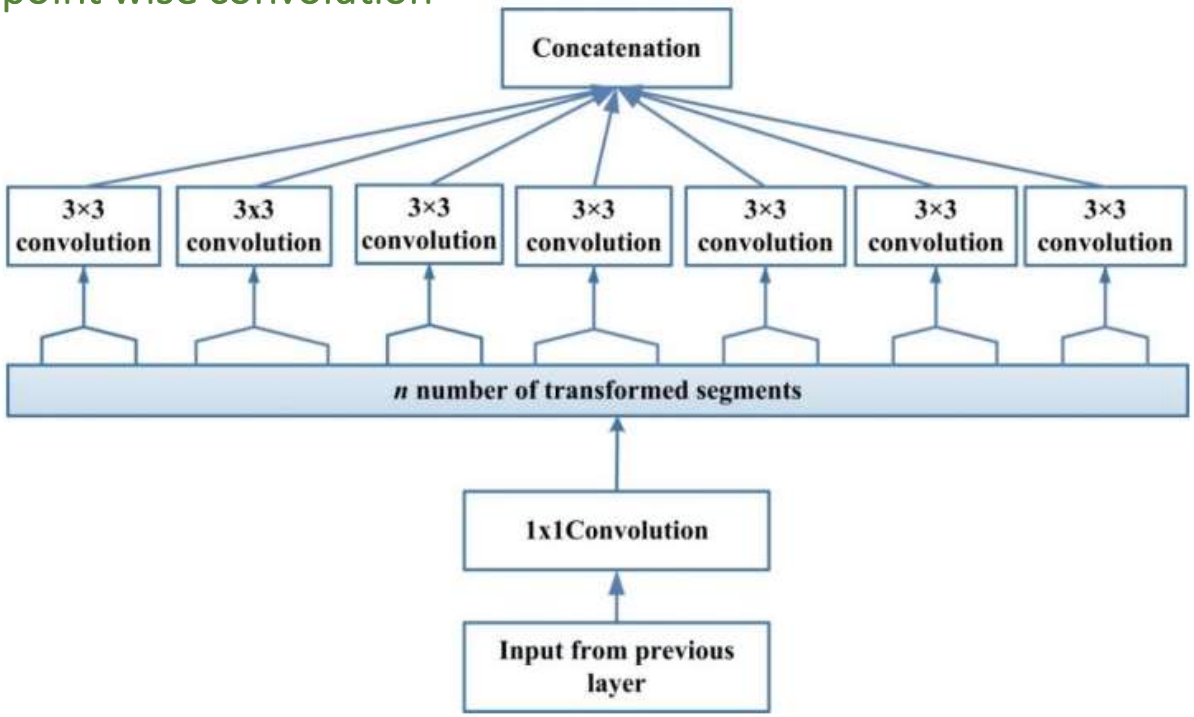


Fig. 8 Xception building block and its *n* sets of transformation.

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.

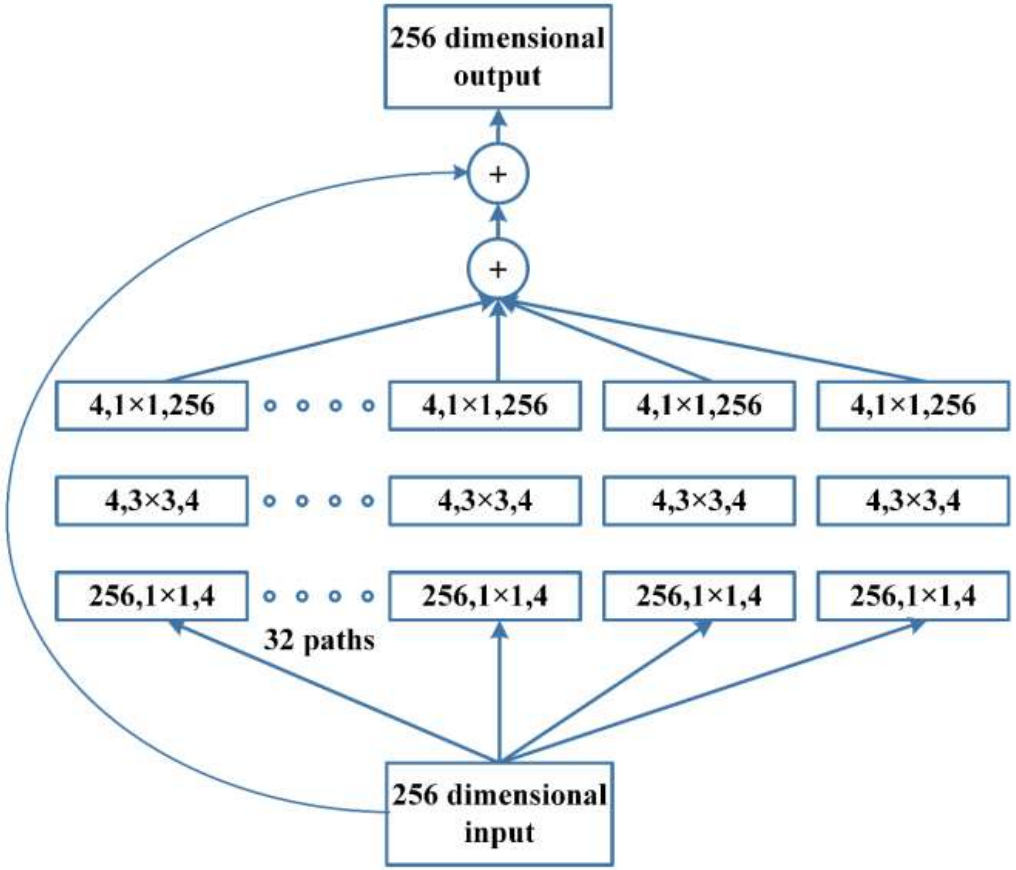


Fig. 9 ResNeXt building block showing the different paths of transformation.

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

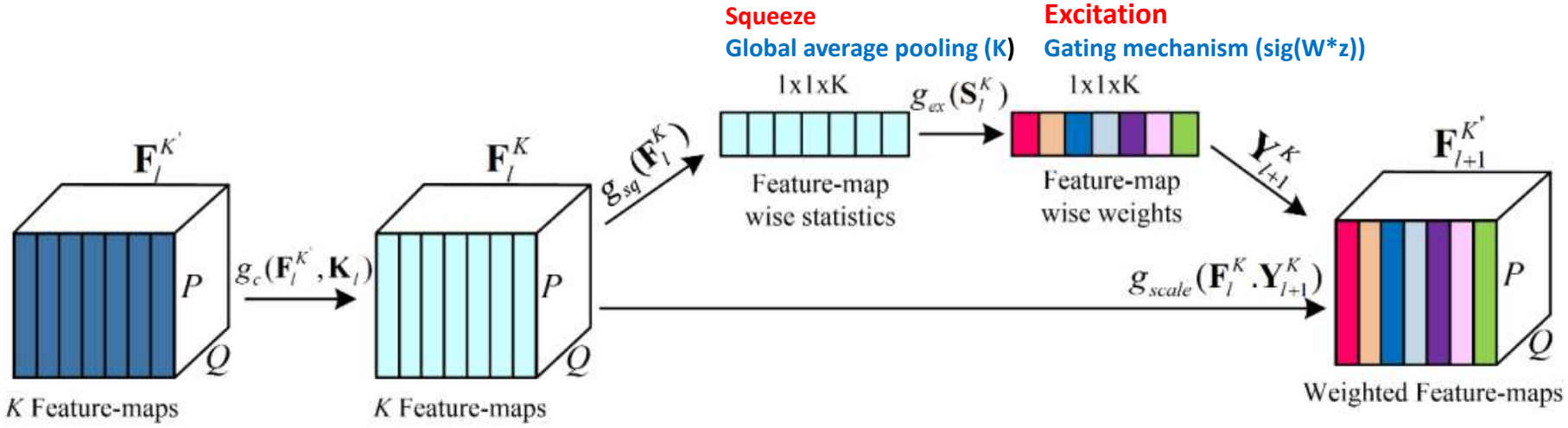
Model interdependencies between feature maps

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

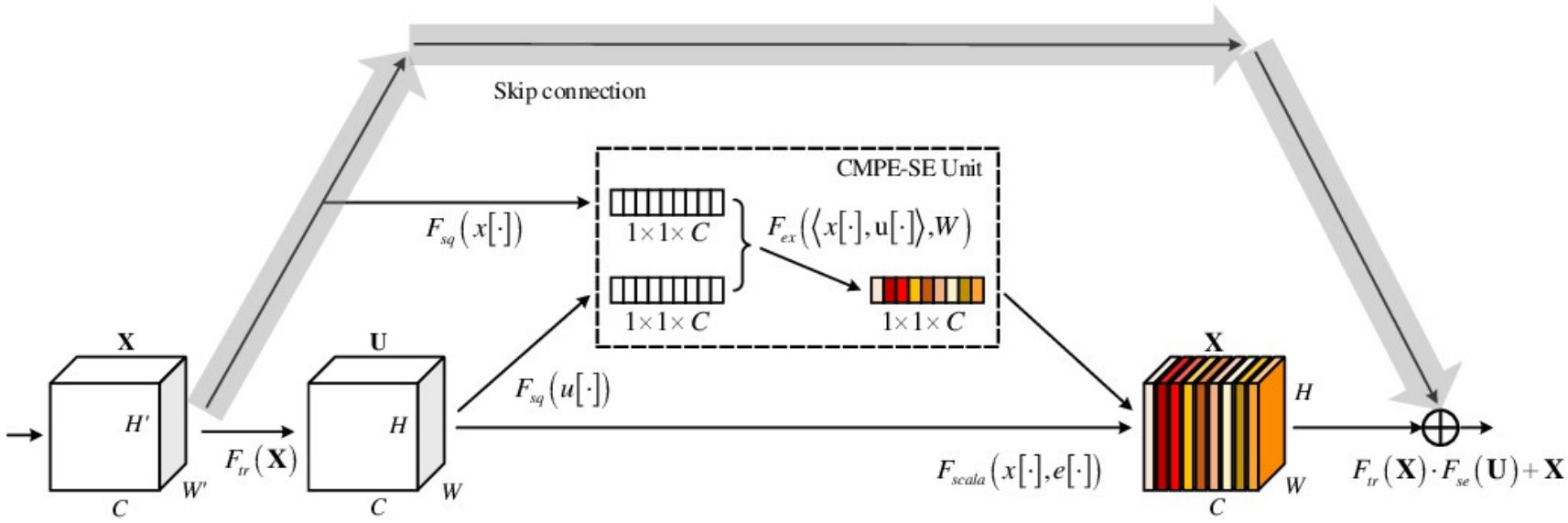


Table 5e Major challenges associated with implementation of Feature-Map exploitation based CNN architectures.

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

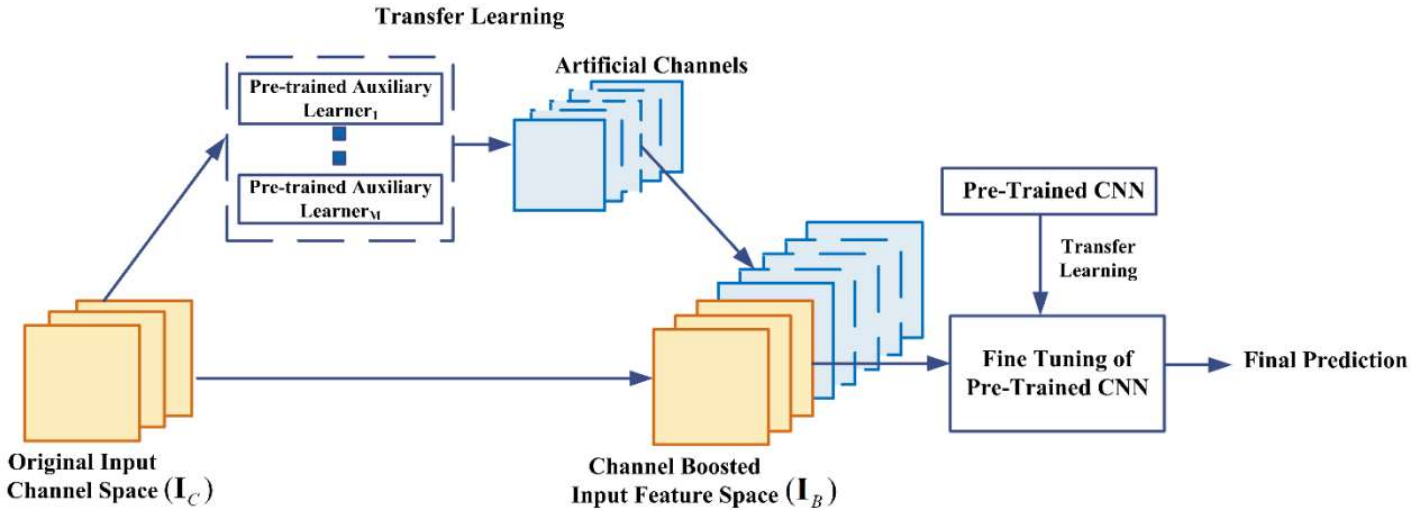
CNNs which improve themselves by using auxiliary learners for channel boosting (input channel dimension)

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.

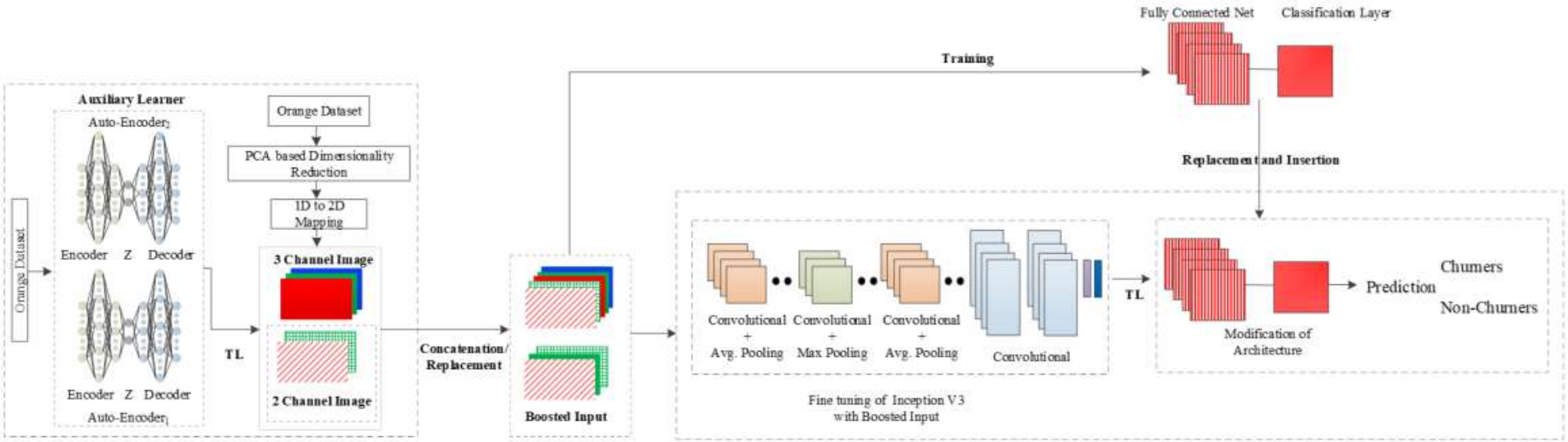


Fig. 4. Details of the working of the proposed CB-CNN. *CB-CNN1* is trained by using Boosted Input$_1$. Boosted Input$_1$ is comprised of three channels that is generated by replacing the input channels of original feature space with two auxiliary channels. Whereas, Boosted Input$_2$ is used to train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

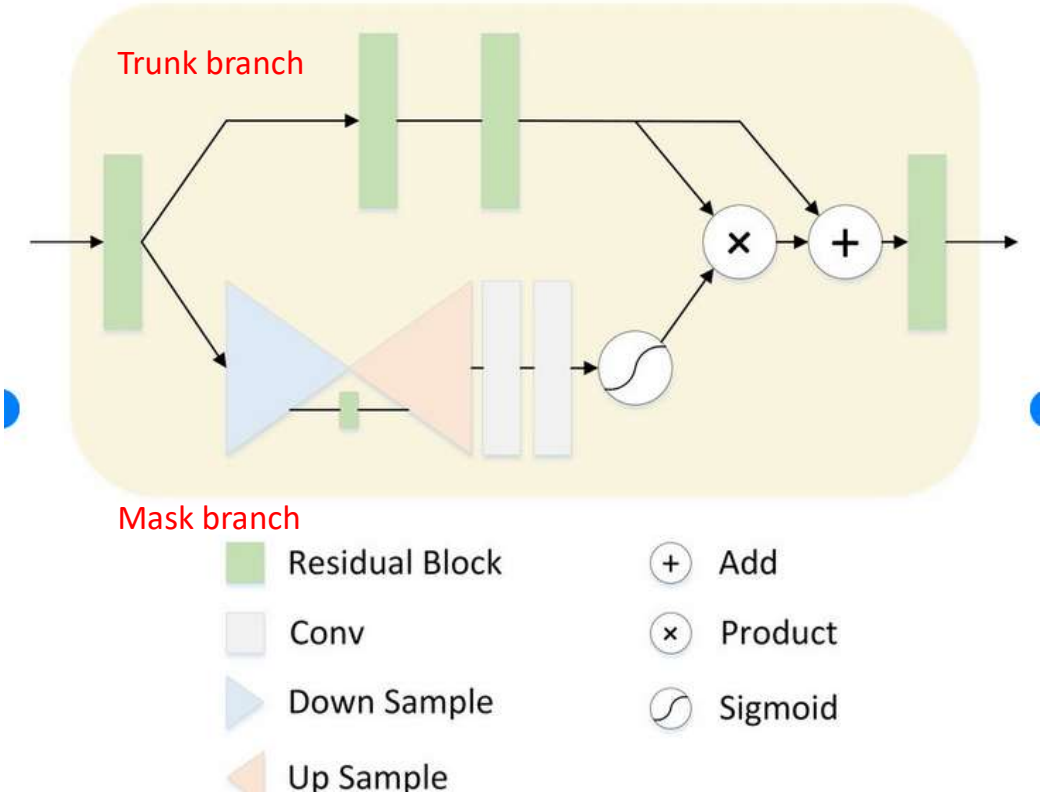
Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Residual Attention Neural Network	2017	- Introduced an attention mechanism	8.6 M	CIFAR-10: 3.90 CIFAR-100: 20.4 ImageNet: 4.8	452	Attention	(Wang et al. 2017a)
Convolutional Block Attention Module (ResNeXt101 (32x4d) + 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*

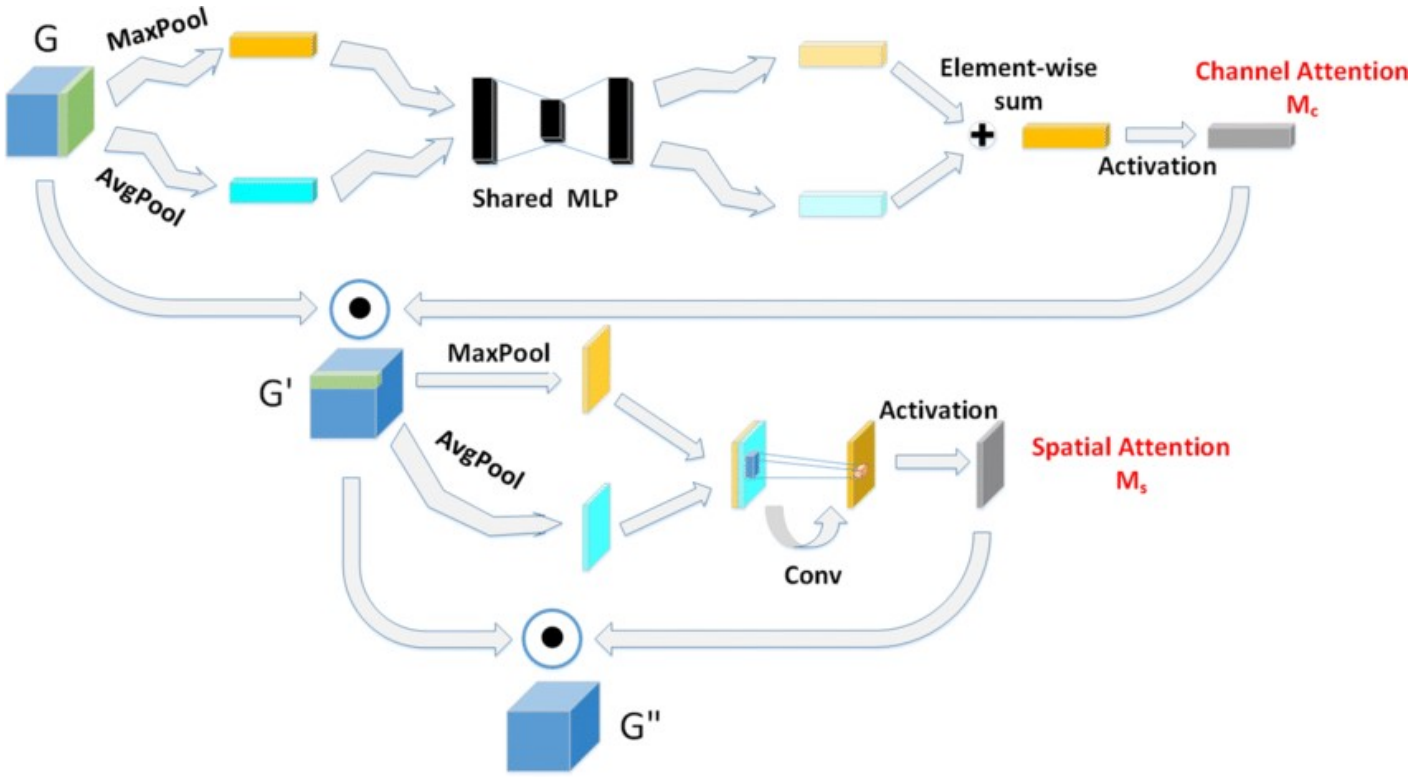
Sik-Ho Tsang Apr 11, 2019

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.

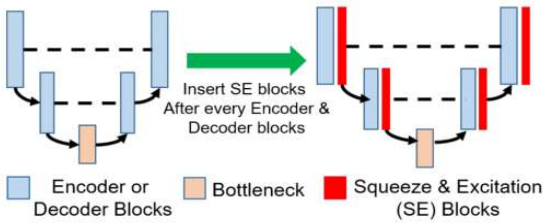


CBAM: Convolutional Block Attention Module

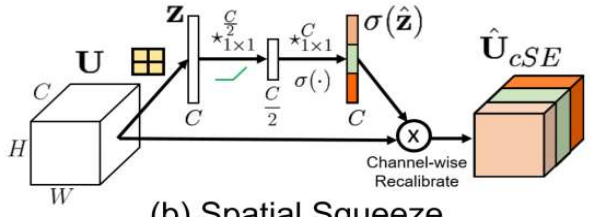


Concurrent Spatial and Channel Excitation Mechanism

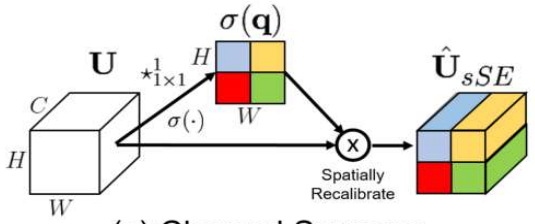
F-CNN: Fully Convolutional NN.



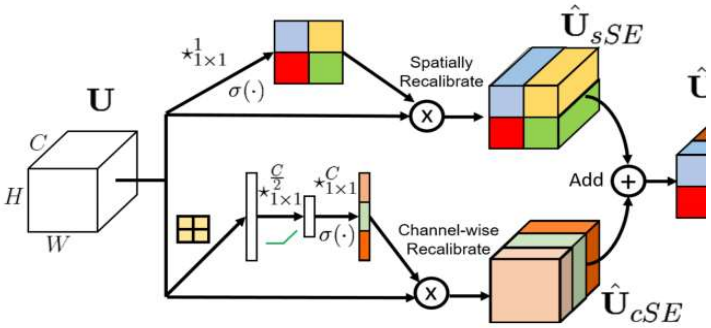
(a) SE block in F-CNN



(b) Spatial Squeeze
and Channel Excitation (cSE)



(c) Channel Squeeze
and Spatial Excitation (sSE)



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

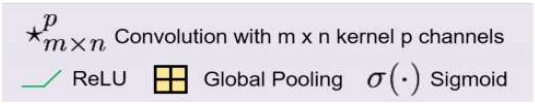


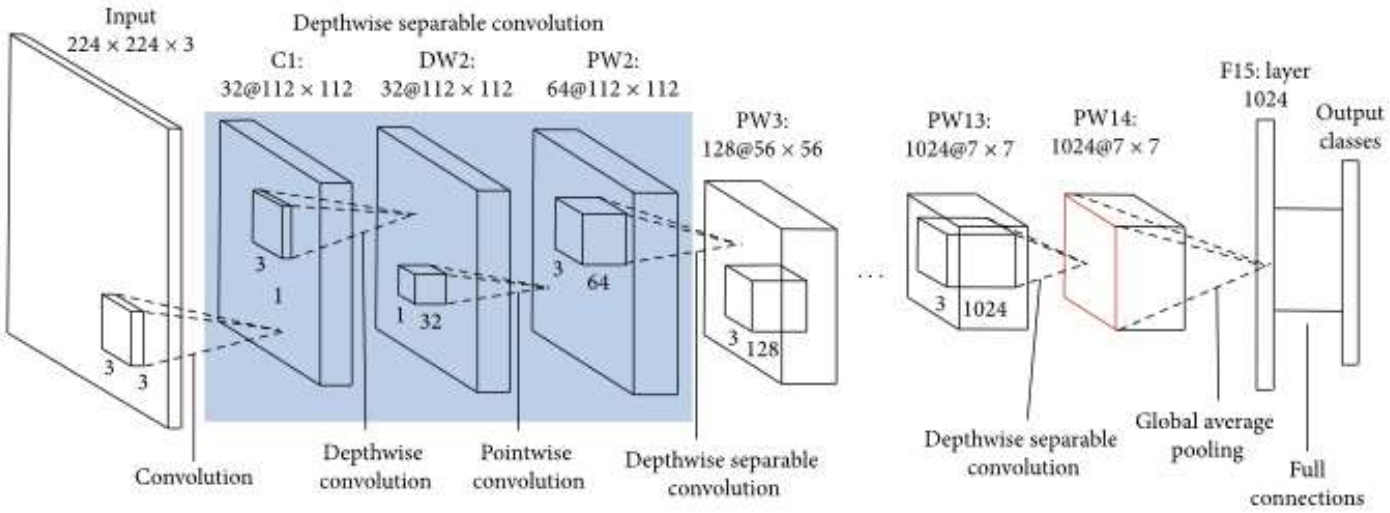
Table 5g Major challenges associated with implementation of Attention based CNN architectures.

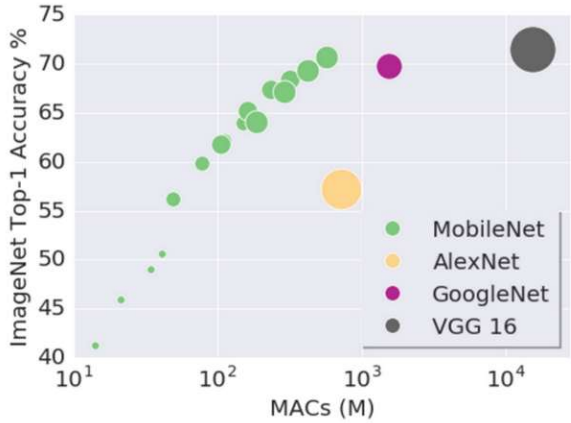
Attention	Attention Networks advantages to choose which patch is the area of the focus or most important in an image	
Architecture	**Strength**	**Gaps**
Residual Attention Neural Network	• Generates attention aware feature-maps • Easy to scale up due to residual learning • Provides different representations 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.



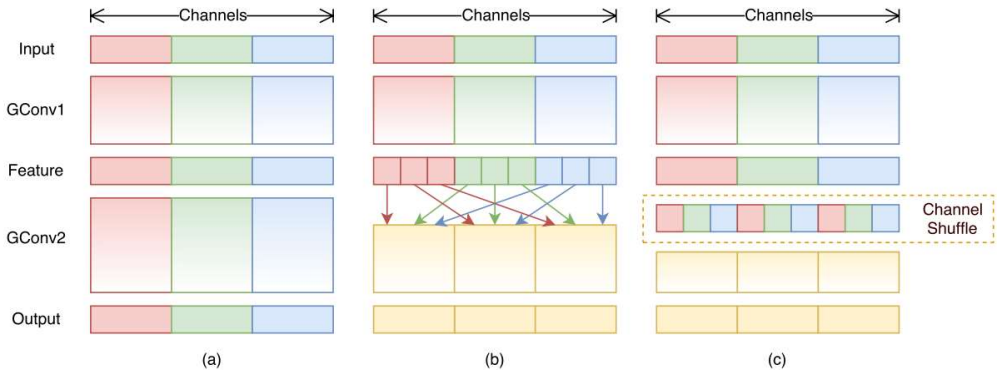


The speed and power consumption of the network is proportional to the number of MACs (Multiply-Accumulates) which is a measure of the number of fused Multiplication and Addition operation

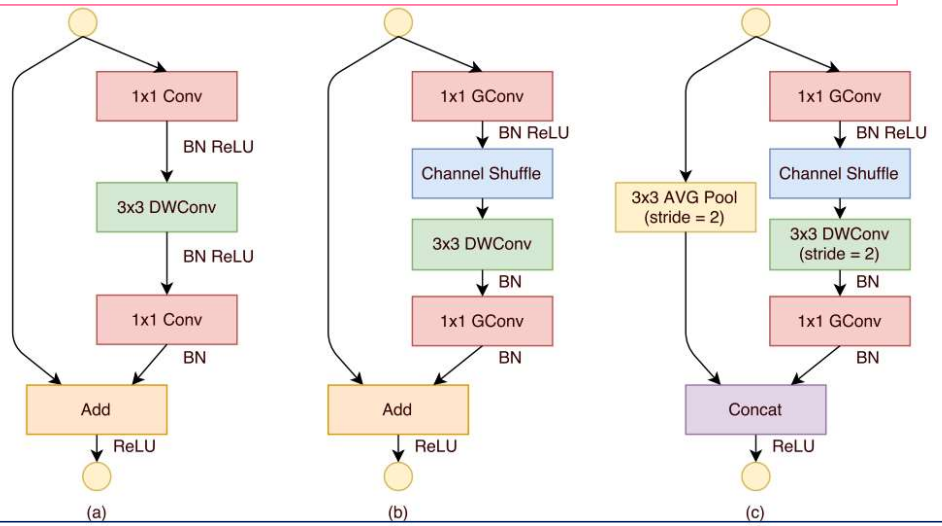
ShuffleNet

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 unit with 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**.

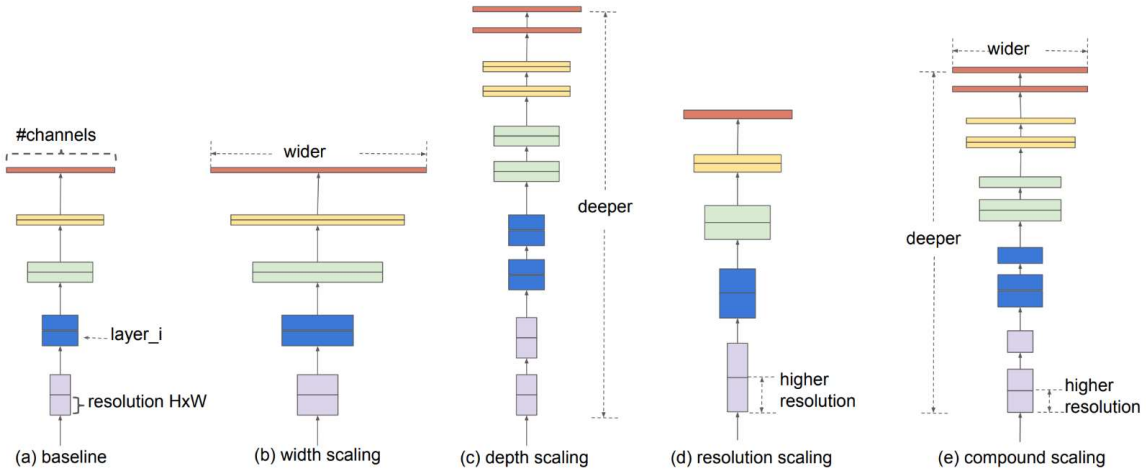
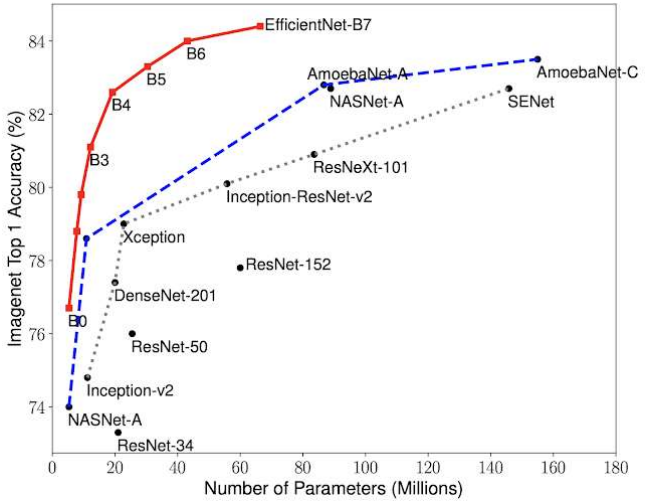


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.



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	-	-

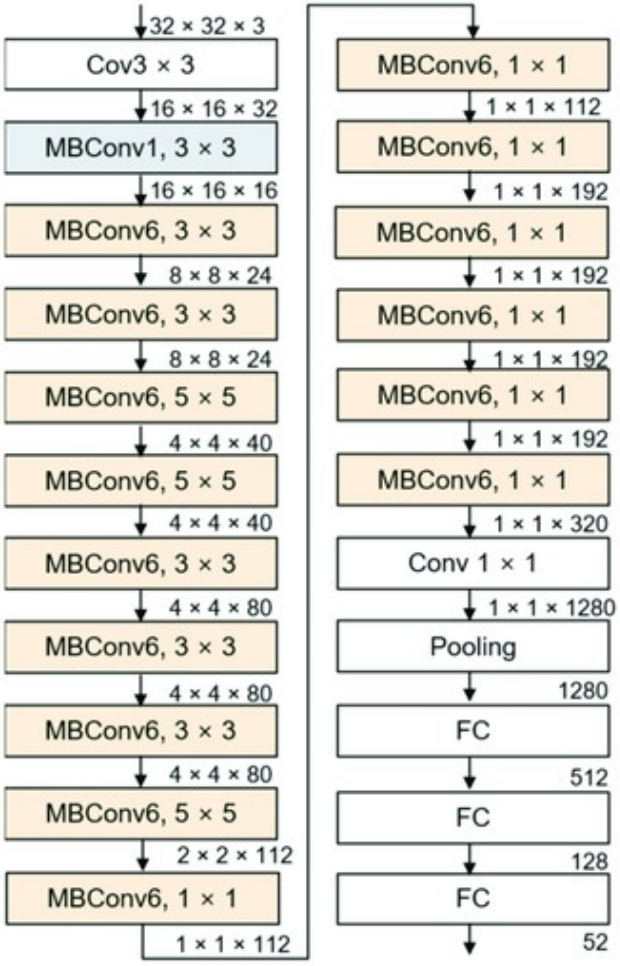
We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

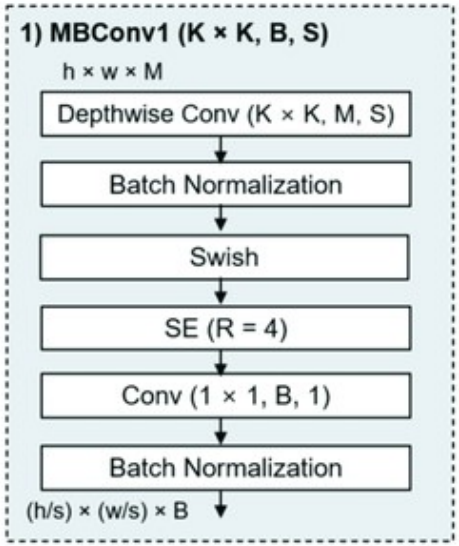
FLOPS = **floating point operations per second**

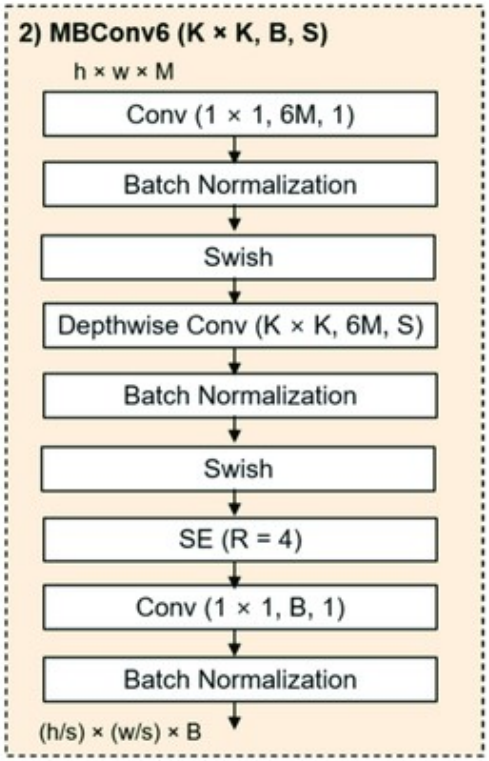
EfficientNetB0

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.

A MBConv is **a Inverted Linear BottleNeck layer with Depth-Wise Separable Convolution** and with squeeze and excitation connection added to it.



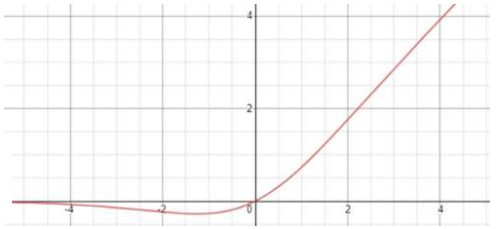




K : kernel size
M : Input Feature maps
B : Output Feature maps
S : Stride
R : Reduction ration of SE

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.

1.2.2 Region Based CNNs(RCNNs)

CNNs that detect different types of objects in an image (Hybrid of classification and Regression problems)

Pioneer works

- *Viola-Jones*
- *HOG Detector*
- *DPM:* Deformable Parts Model

Two stage detectors

- RCNN
- *SPP-Net* (Spatial Pyramid Pooling)
- Fast RCNN
- Faster RCNN (FRCN)
- FPN
- RFCN (Region Fully Connected Network)
- Mask RCNN

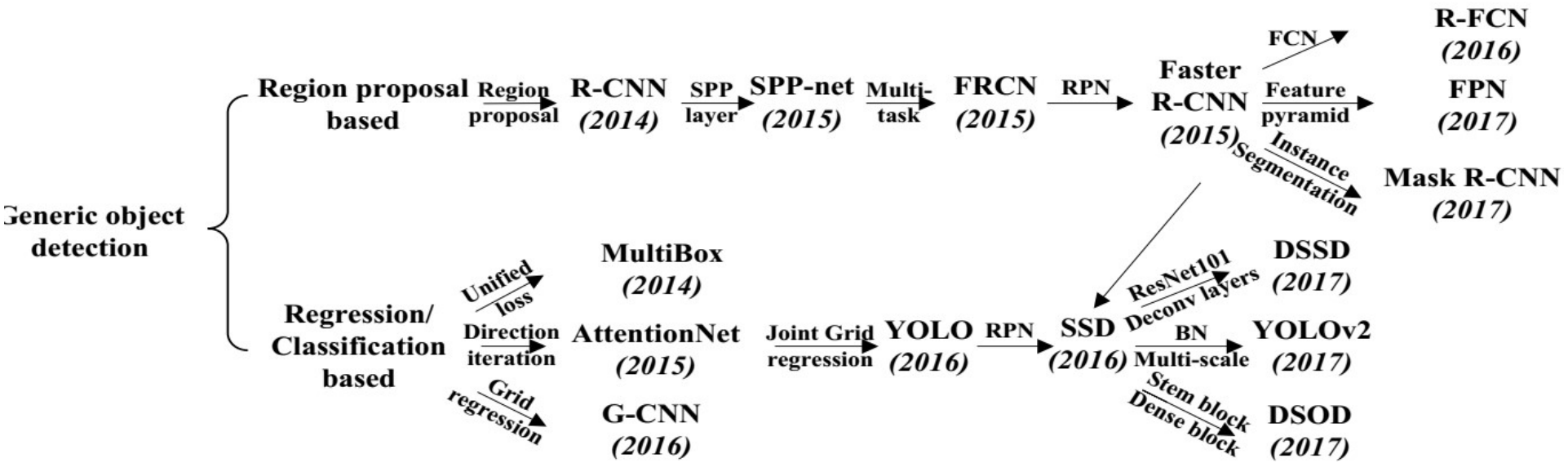
Single stage detectors

1. YOLO
2. SSD
3. YLOLO2, YOLO9000
4. Retina Net
5. YOLOv3
6. Center Net
7. EfficientDet
8. YOLOv4
9. Swin Transformer
10. YOLOX

* S. A. Zaidi (2021)

Evolutionary history of RCNNs 2014-2021

* Zhong-Qiu Zhao, 2019



Retina Net(2018) → YOLOv3(2018) →

Center Net (2019) →

EfficientDet(2020) → YOLOv4 (2020) →

Swin Transformer (2021)

Data-sets for object detections

TABLE I: Comparison of various object detection datasets.

Dataset	Classes	Train			Validation			Test
		Images	Objects	Objects/Image	Images	Objects	Objects/Image	
PASCAL VOC 12	20	5,717	13,609	2.38	5,823	13,841	2.37	10,991
MS-COCO	80	118,287	860,001	7.27	5,000	36,781	7.35	40,670
ILSVRC	200	456,567	478,807	1.05	20,121	55,501	2.76	40,152
OpenImage	600	1,743,042	14,610,229	8.38	41,620	204,621	4.92	125,436

(a) PASCAL VOC 12 (b) MS-COCO (c) ILSVRC (d) OpenImage

Fig. 2: Sample images from different datasets.

Performance Comparison

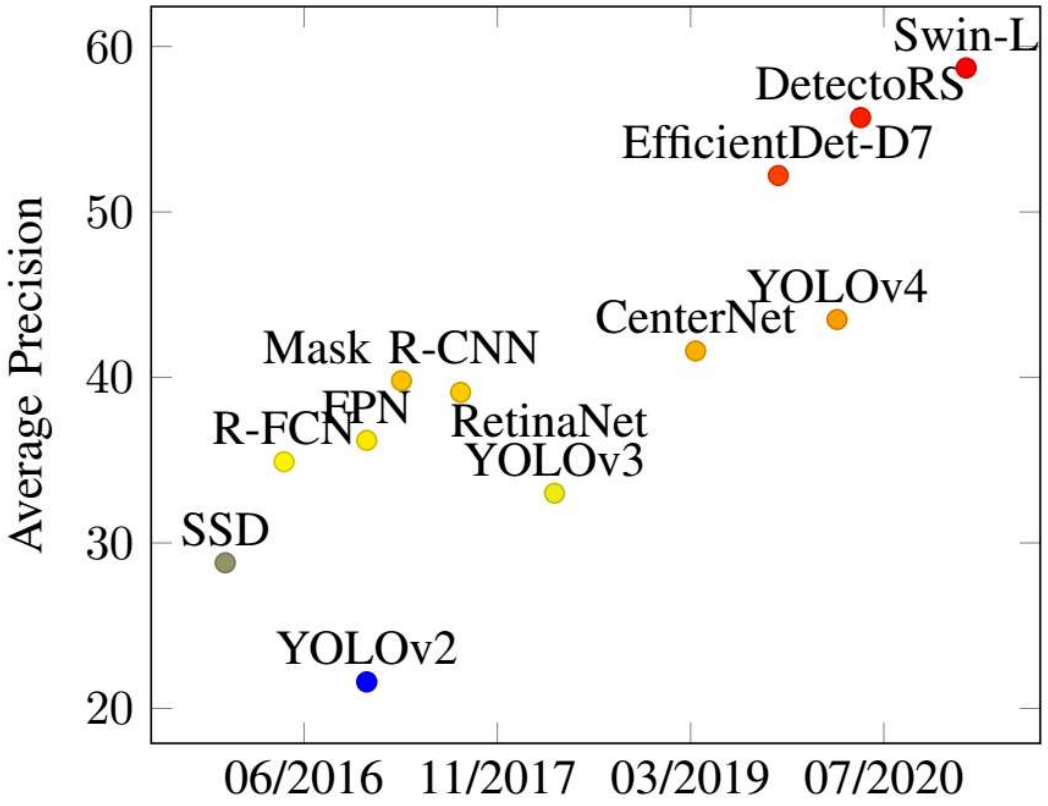
TABLE III: Performance comparison of various object detectors on MS COCO and PASCAL VOC 2012 datasets at simila input image size.

Model	Year	Backbone	Size	$AP_{[0.5:0.95]}$	$AP_{0.5}$	FPS
R-CNN*	2014	AlexNet	224	-	58.50%	~0.02
SPP-Net*	2015	ZF-5	Variable	-	59.20%	~0.23
Fast R-CNN*	2015	VGG-16	Variable	-	65.70%	~0.43
Faster R-CNN*	2016	VGG-16	600	-	67.00%	5
R-FCN	2016	ResNet-101	600	31.50%	53.20%	~3
FPN	2017	ResNet-101	800	36.20%	59.10%	5
Mask R-CNN	2018	ResNeXt-101-FPN	800	39.80%	62.30%	5
DetectoRS	2020	ResNeXt-101	1333	53.30%	71.60%	~4
YOLO*	2015	(Modified) GoogLeNet	448	-	57.90%	45
SSD	2016	VGG-16	300	23.20%	41.20%	46
YOLOv2	2016	DarkNet-19	352	21.60%	44.00%	81
RetinaNet	2018	ResNet-101-FPN	400	31.90%	49.50%	12
YOLOv3	2018	DarkNet-53	320	28.20%	51.50%	45
CenterNet	2019	Hourglass-104	512	42.10%	61.10%	7.8
EfficientDet-D2	2020	Efficient-B2	768	43.00%	62.30%	41.7
YOLOv4	2020	CSPDarkNet-53	512	43.00%	64.90%	31
Swin-L	2021	HTC++	-	57.70%	-	-

[a]Models marked with * are compared on PASCAL VOC 2012, while others on MS COCO.Rows colored gray are real-time detectors (>30 FPS).

AP: Average Precision FPS: Frame per second

Performance of Object Detectors on MS COCO dataset.



(1) Pioneer Works

1. *Viola-Jones*
2. *HOG Detector*
3. *DPM:* Deformable Parts Model

Viola-Jones:

- Primarily designed for face detection, in 2001 Viola-Jones as an object detector was an accurate and powerful detector.
- It combined multiple techniques like Haar-like features, integral image, Adaboost and cascading classifier.
- Viola Jones algorithm is still used in small devices as it is very efficient and fast.

HOG: Histogram of Oriented Gradients

- In 2005, Dalal and Triggs proposed HOG.
- HOG extracts gradient and its orientation of the edges to create a feature table.
- The image is divided into grids and the feature table is then used to create histogram for each cell in the grid.
- HOG features are generated for the region of interest and fed into a linear SVM classifier for detection.
- The detector was proposed for pedestrian detection; however, it could be trained to detect various classes.

DPM: Deformable Parts Model

- DPM was introduced by Felzenszwalb et al. and was the winner Pascal VOC challenge in 2009.
- It used individual "part" of the object for detection and achieved higher accuracy than HOG.
- It follows the philosophy of *divide and rule*; parts of the object are individually detected during inference time and a probable arrangement of them is marked as detection.

(1) Two stage Object detectors

- RCNN
- *SPP-Net* (Spatial Pyramid Pooling)
- Fast RCNN
- Faster RCNN (FRCN)
- FPN
- RFCN (Region Fully Connected Network)
- Mask RCNN

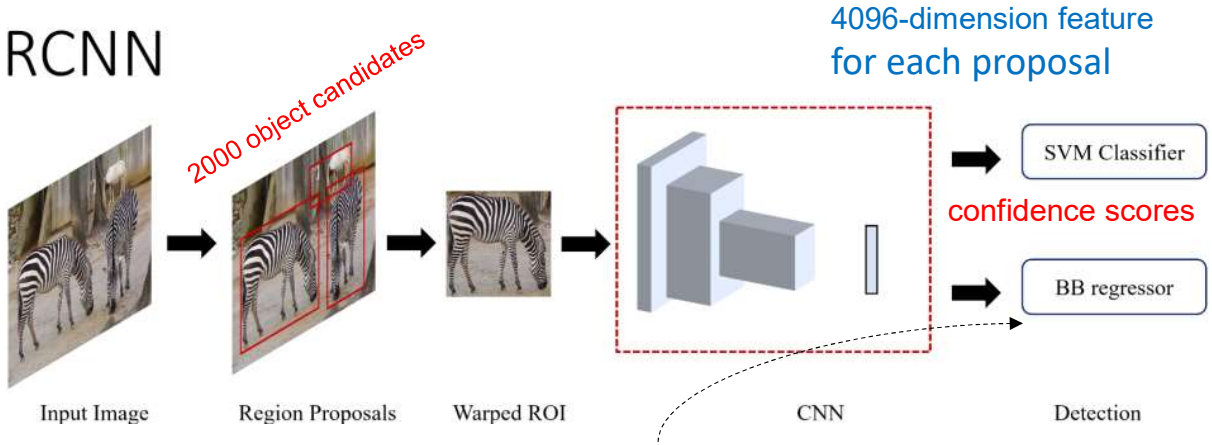
- A network which has a separate module to generate region proposals is termed as a two-stage detector.
- These models try to find an arbitrary number of objects proposals in an image during the first stage and then classify and localize them in the second.
- As these systems have two separate steps, they generally take longer to generate proposals, have complicated architecture and lacks global context.

Region-based Convolutional Neural Network (RCNN)

*R. Girshick 2014

A mean-subtracted input image is first passed through the region proposal module, which produces 2000 object candidates.
This module find parts of the image which has a higher probability of finding an object using Selective Search.
These candidates are then warped and propagated through a CNN network, which extracts a 4096-dimension feature vector for each proposal.
Girshick et al. used AlexNet as the backbone architecture of the detector.
The feature vectors are then passed to the trained, class-specific Support Vector Machines (SVMs) to obtain confidence scores.
Non-maximum suppression (NMS) is later applied to the scored regions, based on its IoU and class.
Once the class has been identified, the algorithm predicts its bounding box using a trained bounding-box regressor which predicts four parameters i.e., center coordinates of box along with its width and height .

A multistage training process



Bounding-Box regressor
four parameters i.e., center coordinates of box along with its width and height

Region Proposal Networks abbreviated as RPN.
To generate these so called “proposals” for the region where the object lies, a small network is slide over a convolutional feature map that is the output by the last convolutional layer.
This module find parts of the image which has a higher probability of finding an object using Selective Search.

SPP-Net (SPP-Net is considerably faster than the R-CNN model with comparable accuracy)

*K. He , 2015

SPP(Spatial Pyramid Pooling)-net only shifted the convolution layers of CNN before the region proposal module and added a pooling layer, thereby making the network independent of size/aspect ratio and reducing the computations.

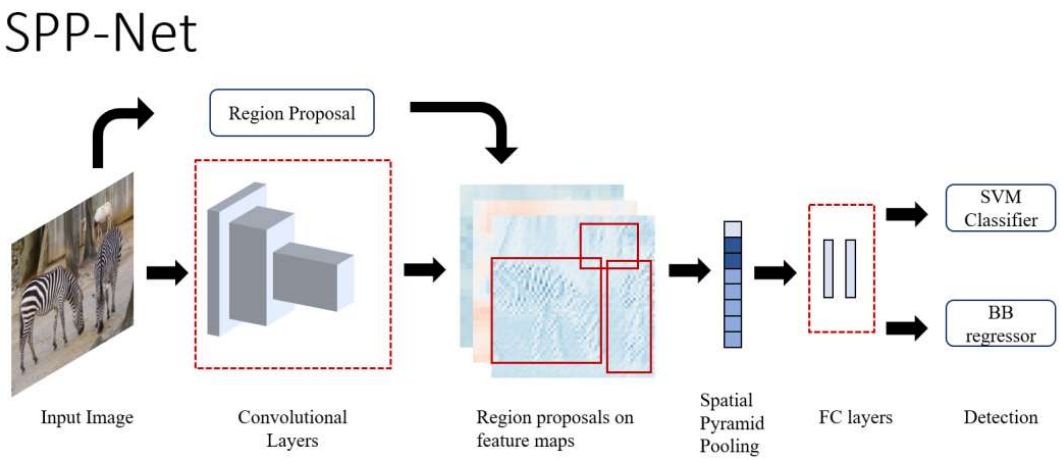
The selective search algorithm is used to generate candidate windows.

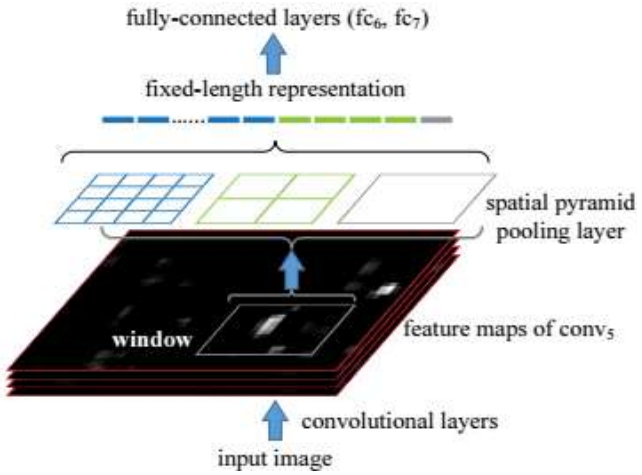
Feature maps are obtained by passing the input image through the convolution layers of a ZF-5 network.

The candidate windows are then mapped on to the feature maps, which are subsequently converted into fixed length representations by spatial bins of a pyramidal pooling layer.

This vector is passed to the fully connected layer and ultimately, to SVM classifiers to predict class and score.

Similar to R-CNN, SPP-net has as post processing layer to improve localization by bounding box regression.

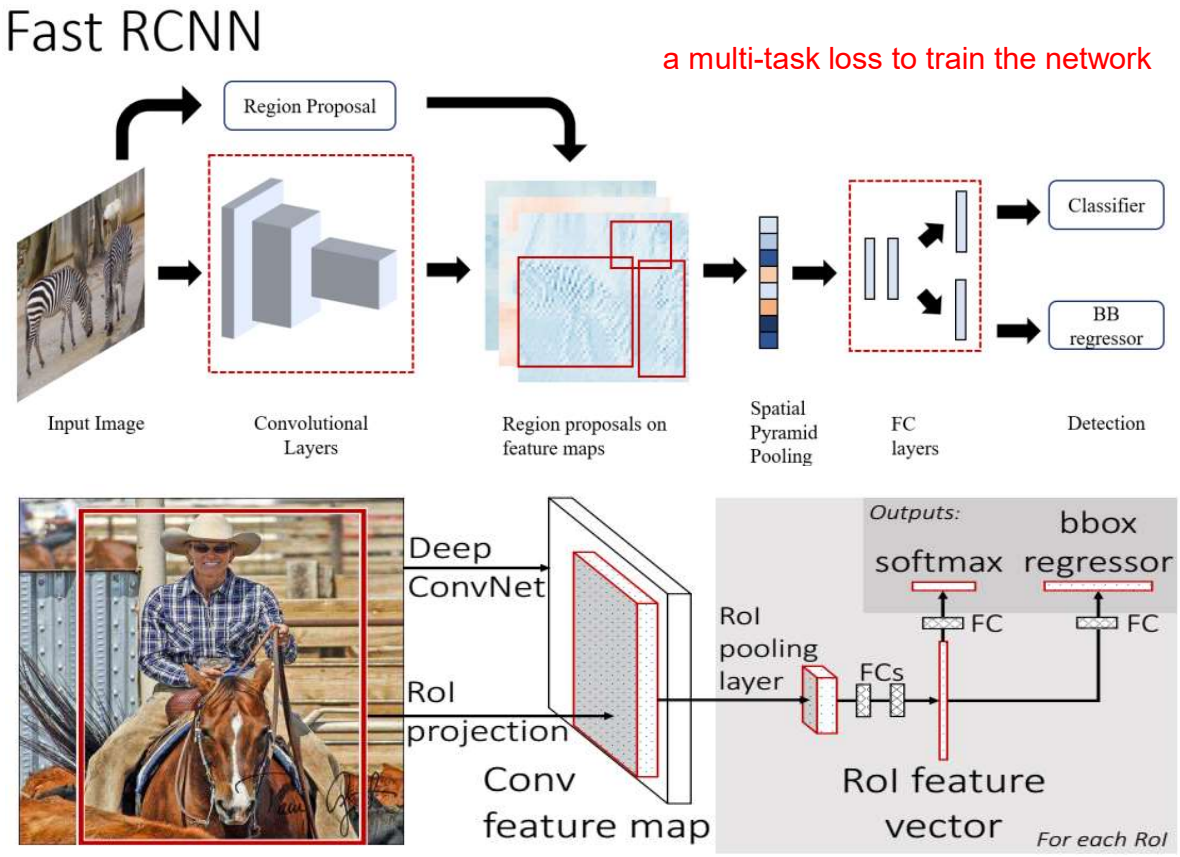




Fast RCNN

*R. Girshick 2015

- R-CNN/SPPNet need to train multiple systems separately.
- Fast R-CNN solved this by creating a single end-to-end trainable system.
- The network takes as input an image and its object proposals.
- The image is passed through a set of convolution layers and the object proposals are mapped to the obtained feature maps.
- Girshick replaced pyramidal structure of pooling layers from SPP-net with a single spatial bin, called RoI (**Region of interest**) pooling layer.
- The RoI pooling layer is a special case of the SPP layer, which has only one pyramid level.
- This layer is connected to 2 fully connected layer and then branches out into a *N+1*-class SoftMax layer and a bounding box regressor layer, which has a fully connected layer as well.
- The model also changed the loss function of bounding box regressor from L2 to smooth L1 to better performance, while introducing a multi-task loss to train the network.

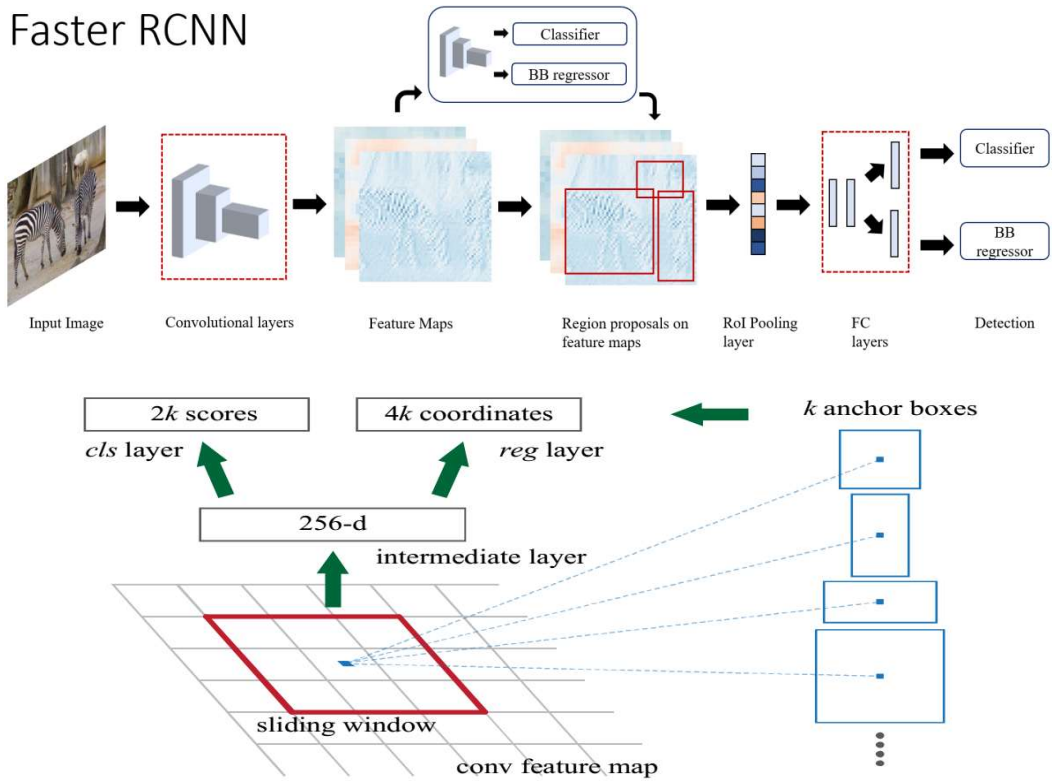


It simplified training procedure, removed pyramidal pooling and introduces a new loss function. The object detector, without the region proposal network, reported near real time speed with considerable accuracy

Faster RCNN

* S. Ren..2016

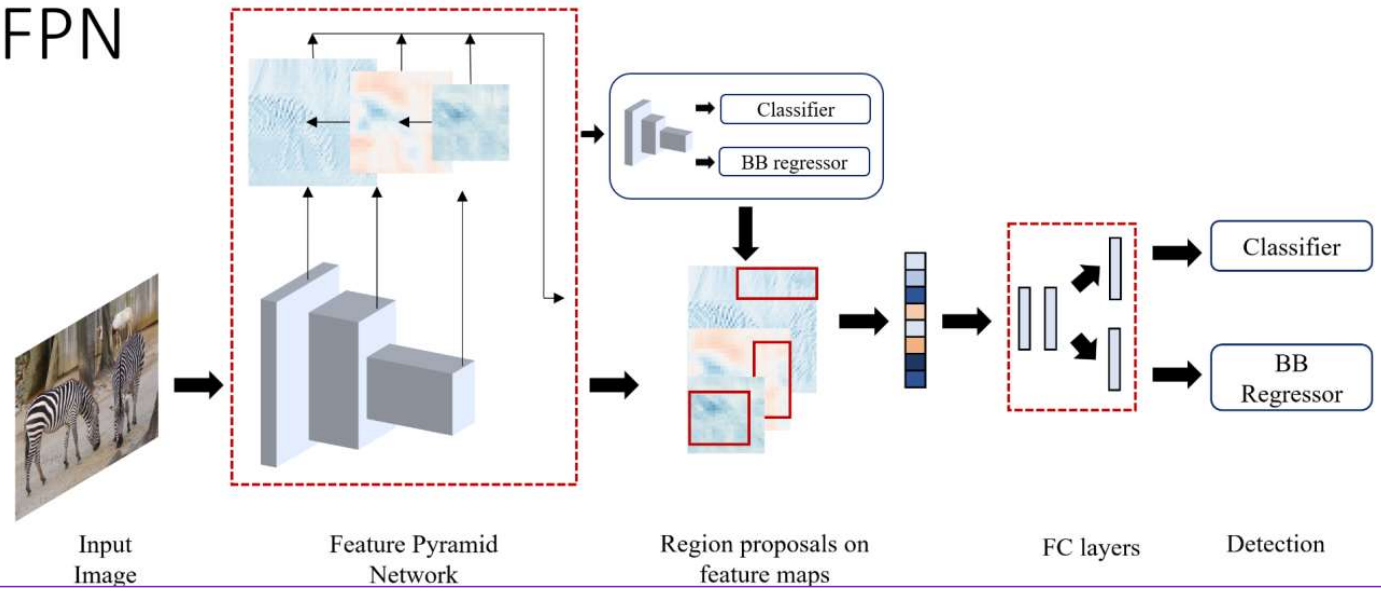
Faster RCNN takes an arbitrary input image and outputs a set of candidate windows.
Each such window has an associated *objectness score* which determines likelihood of an object.
Unlike its predecessors which used image pyramids to solve size variance of objects, RPN introduces Anchor boxes.
It used multiple bounding boxes of different aspect ratios and regressed over them to localize object.
The input image is first passed through the CNN to obtain a set of feature maps.
These are forwarded to the RPN, which produces bounding boxes and their classification.
Selected proposals are then mapped back to the feature maps obtained from previous CNN layer in RoI pooling layer, and ultimately fed to fully connected layer, which is sent to classifier and bounding box regressor.
Faster R-CNN is essentially Fast R-CNN with RPN as region proposal module.
Faster R-CNN improved the detection accuracy over the previous state-of-art by more than 3% . It fixed the bottleneck of slow region proposal and ran in near real time at 5 frames per second.



The RPN in Faster R-CNN K predefined anchor boxes are convoluted with each sliding window to produce fixed-length vectors which are taken by cls and reg layer to obtain corresponding outputs

FPN(Feature Pyramid Network)

*T.-Y. Lin... 2017



FPN has a top-down architecture with lateral connections to build high-level semantic features at different scales.

The FPN has two pathways, a bottom-up pathway which is a ConvNet computing feature hierarchy at several scales and a top-down pathway which up samples coarse feature maps from higher level into high resolution features.

These pathways are connected by lateral connection by a *1x1* convolution operation to enhance the semantic information in the features.

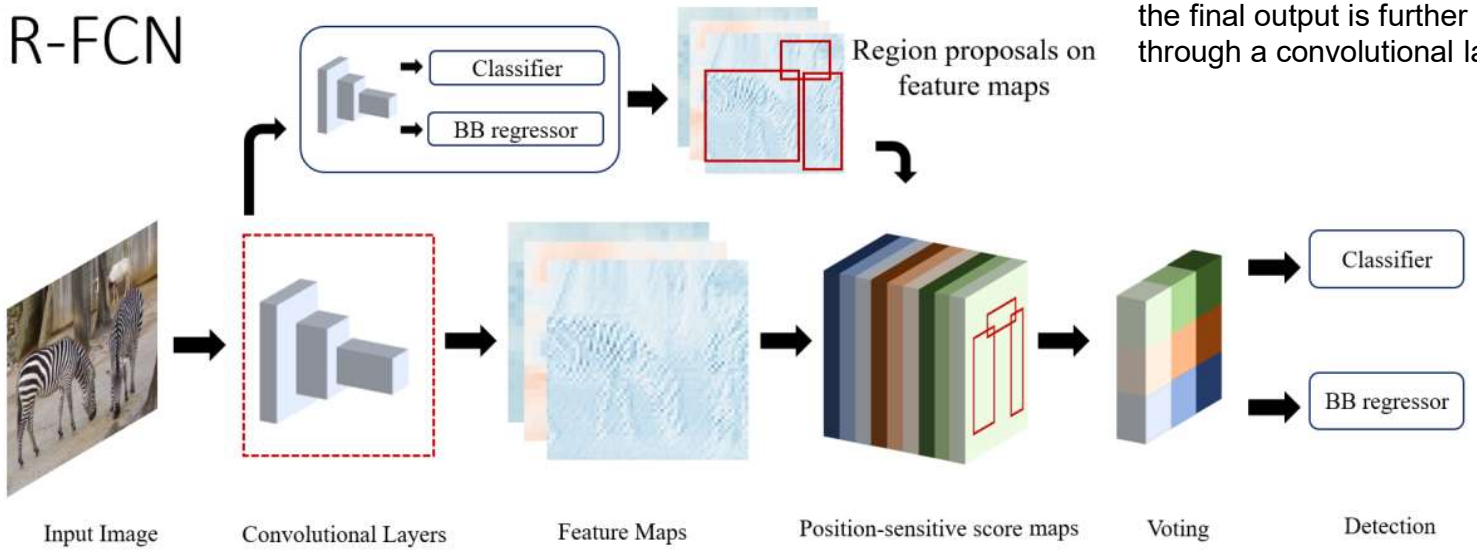
FPN is used as a region proposal network (RPN) of a ResNet-101 based Faster R-CNN here.

FPN could provide high-level semantics at all scales, which reduced the error rate in detection.

It became a standard building block in future detections models and improved accuracy their accuracy across the table. It also lead to development of other improved networks

RFCN(Region-based Fully Convolutional Network)

* J. Dai..2016

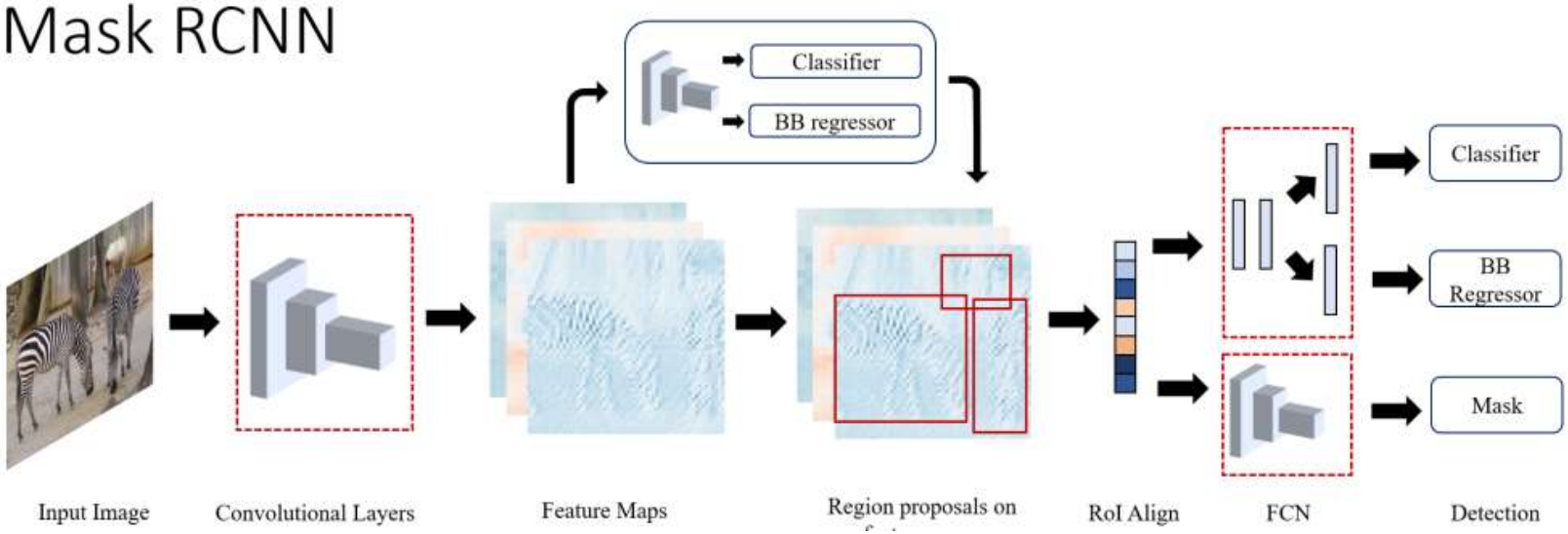


the final output is further processed through a convolutional layer

R-FCN detector is a combination of four convolutional networks.
The input image is first passed through the ResNet-101 to get feature maps.
An intermediate output (Conv4 layer) is passed to a Region Proposal Network (RPN) to identify RoI proposals.
while the final output is further processed through a convolutional layer and is input to classifier and regressor.
The classification layer combines the generated the position-sensitive map with the RoI proposals to generate predictions while the regression network outputs the bounding box details.
R-FCN is trained in a similar 4 step fashion as Faster-RCNN [44] whilst using a combined cross-entropy and box regression loss.

Mask RCNN

* K. He...2018



Mask R-CNN extends on the Faster R-CNN by adding another branch in parallel for pixel-level object instance segmentation.
The branch is a fully connected network applied on RoIs to classify each pixel into segments with little overall computation cost.
It uses similar basic Faster R-CNN architecture for object proposal, but adds a mask head parallel to classification and bounding box regressor head.
The authors chose the ResNeXt-101 as its backbone along with the feature Pyramid Network (FPN) for better accuracy and speed.
The loss function of Faster R-CNN is updated with the mask loss and as in FPN, it uses 5 anchor boxes with 3 aspect ratio. Overall training of Mask R-CNN is similar to faster R-CNN

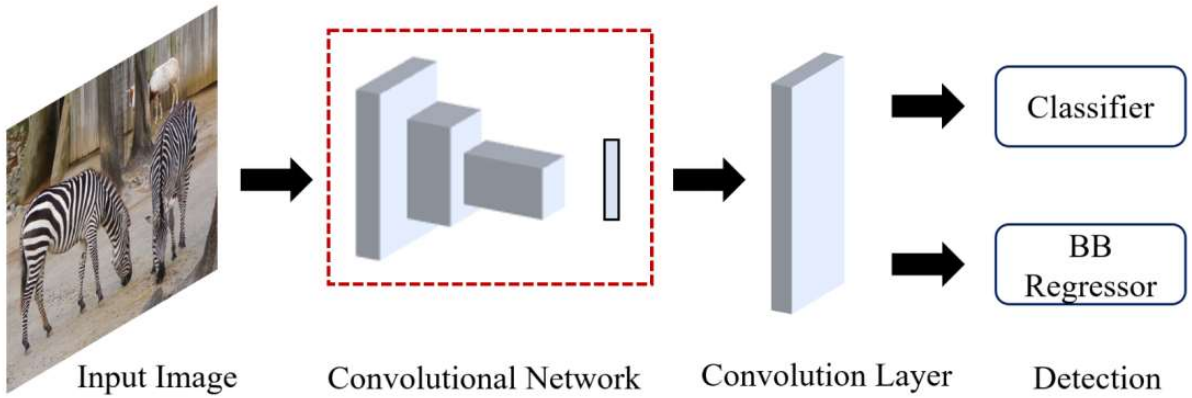
3. One stage object detectors

1. YOLO
2. SSD
3. YLOLO2, YOLO9000
4. Retina Net
5. YOLOv3
6. Center Net
7. EfficientDet
8. YOLOv4
9. Swin Transformer
10. YOLOx

- Single-stage detectors classify and localize semantic objects in a single shot using dense sampling.
- They use predefined boxes/keypoints of various scale and aspect ratio to localize objects.
- It edges two-stage detectors in real-time performance and simpler design.

YOLO (YOLO

* J. Redmon ...201



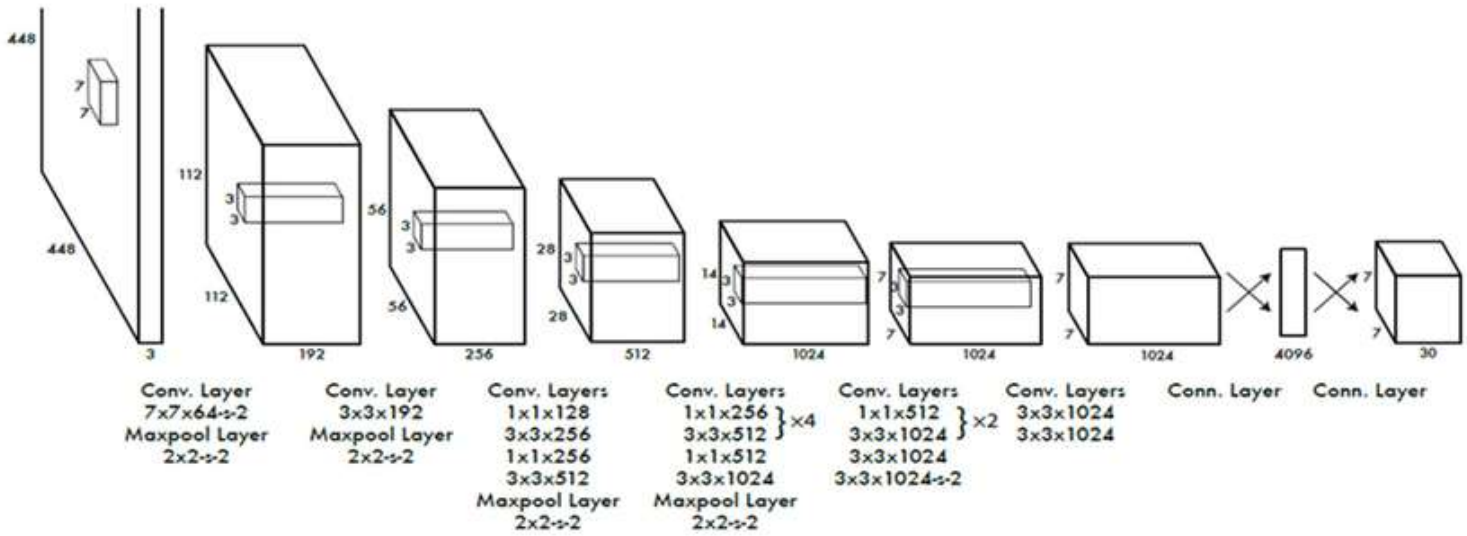
YOLO reframed it as a regression problem, directly predicting the image pixels as objects and its bounding box attributes. In YOLO, the input image is divided into a $S \times S$ grid and the cell where the object's center falls is responsible for detecting it.

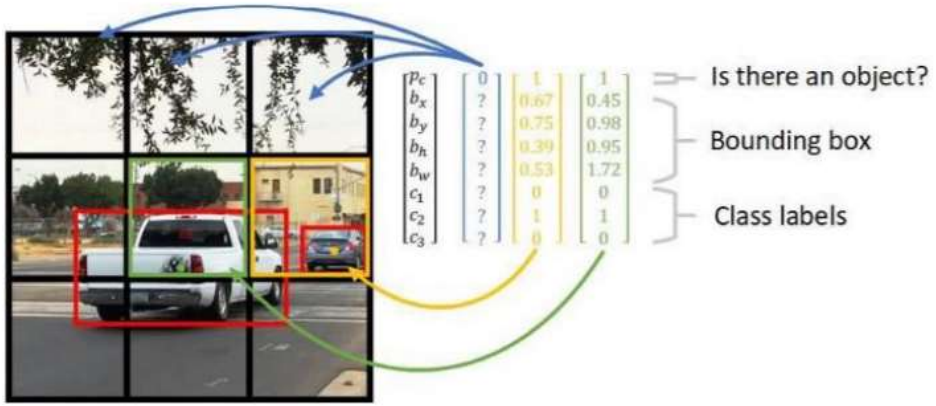
A grid cell predicts multiple bounding boxes, and each prediction array consists of 5 elements: center of bounding box – x and y, dimensions of the box – w and h, and the confidence score.

YOLO was inspired from the GoogLeNet model for image classification, which uses cascaded modules of smaller convolution networks.

It is pre-trained on ImageNet data till the model achieves high accuracy and then modified by adding randomly initialized convolution and fully connected layers.

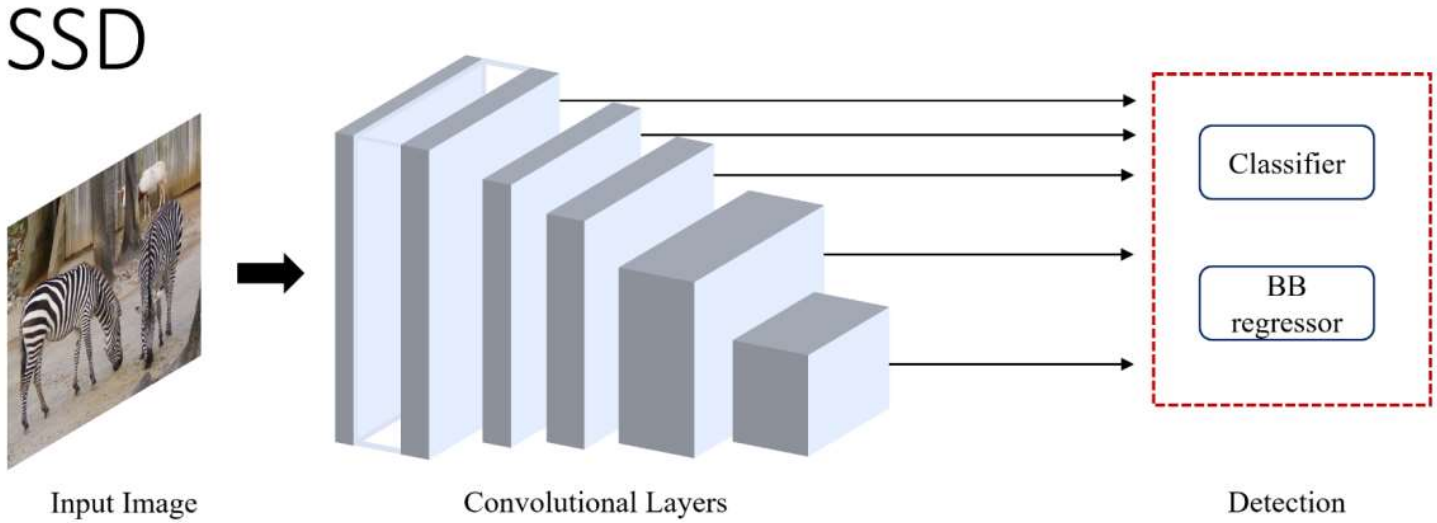
At training time, grid cells predict only one class as it converges better, but it is be increased during the inference time. Multitask loss, combined loss of all predicted components, is used to optimize the model.


448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2
Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x256
3x3x512
×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x512
3x3x1024
×2
3x3x1024-s-2
Conv. Layers
3x3x1024
3x3x1024
Conn. Layer
Conn. Layer

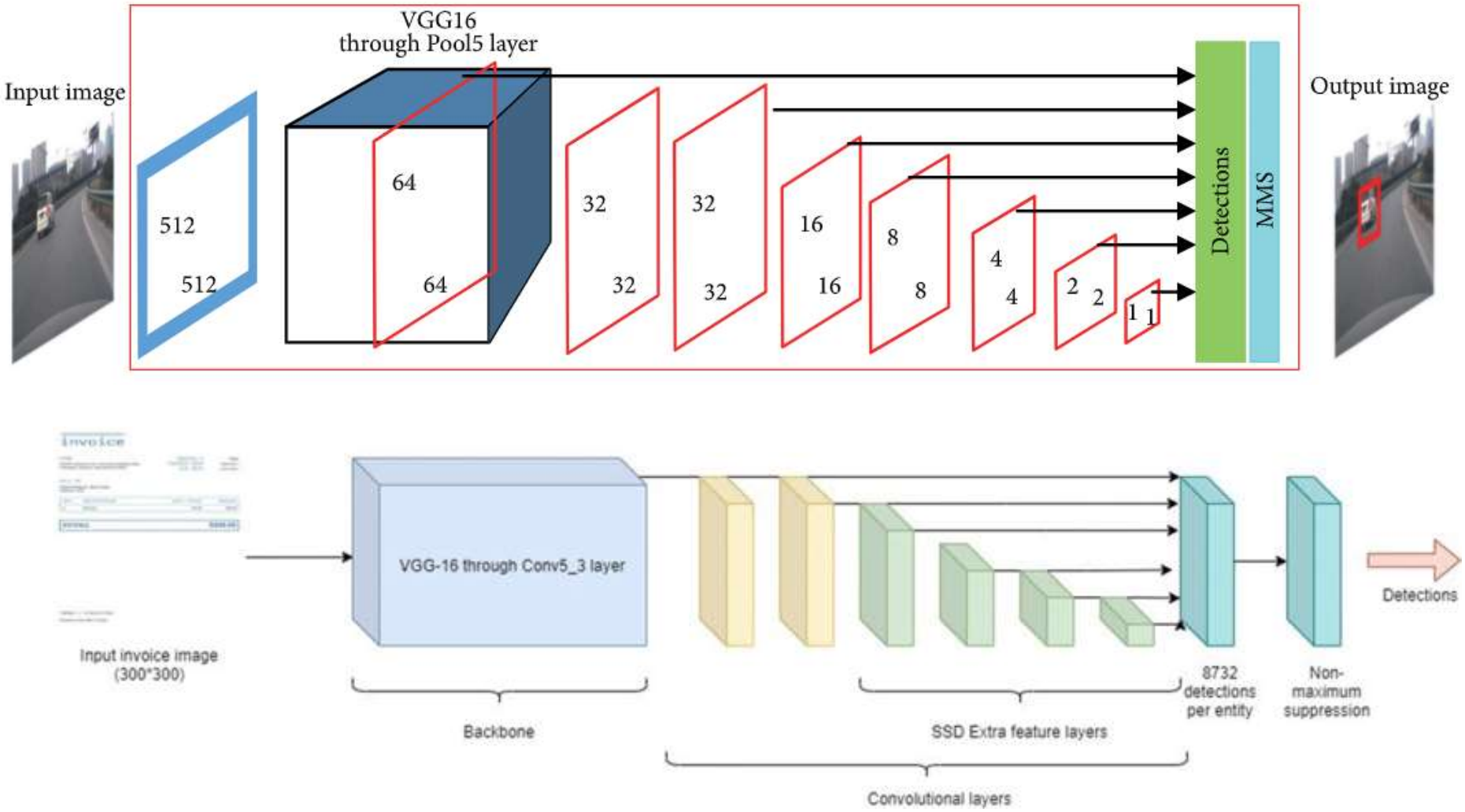

p_c
b_x
b_y
b_h
b_w
c_1
c_2
c_3
0
?
?
?
?
?
?
?
1
0.67
0.75
0.39
0.53
0
1
0
1
0.45
0.98
0.95
1.72
0
1
0
Is there an object?
Bounding box
Class labels

SSD(SingleShot Detector)

*W. Liu, 2016



SSD was the first single stage detector that matched accuracy of contemporary two stage detectors like Faster R-CNN [44], while maintaining real time speed.
SSD was built on VGG-16, with additional auxiliary structures to improve performance.
These auxiliary convolution layers, added to the end of the model, decrease progressively in size. SSD detects smaller objects earlier in the network when the image features are not too crude, while the deeper layers were responsible for offset of the default boxes and aspect ratios.
Even though SSD was significantly faster and more accurate than both state-of-art networks like YOLO and Faster R-CNN, it had difficulty in detecting small objects.
This issue was later solved by using better backbone architectures like ResNet and other small fixes


Input image
VGG16
through Pool5 layer
512
512
64
64
32
32
32
32
16
16
8
8
4
4
2
2
1
1
Detections
MMS
Output image
Input invoice image
(300*300)
VGG-16 through Conv5_3 layer
Backbone
SSD Extra feature layers
Convolutional layers
8732
detections
per entity
Non-
maximum
suppression
Detections

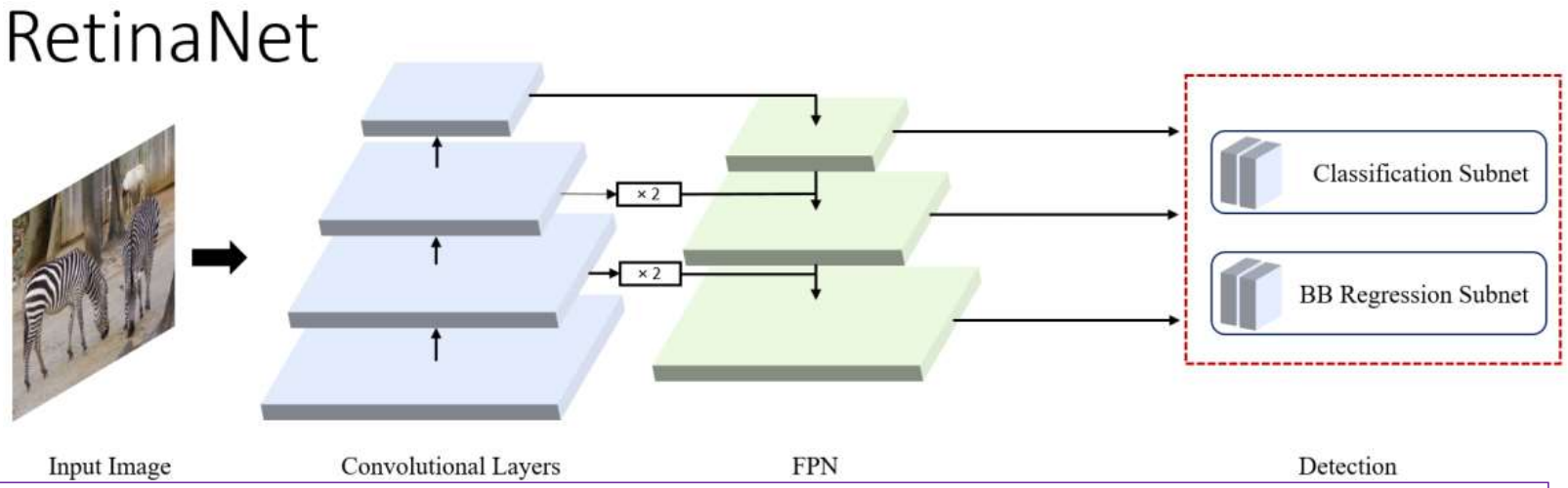
NMS: Non-maximum Suppression

Non max suppression is **a technique used mainly in object detection that aims at selecting the best bounding box out of a set of overlapping boxes**.

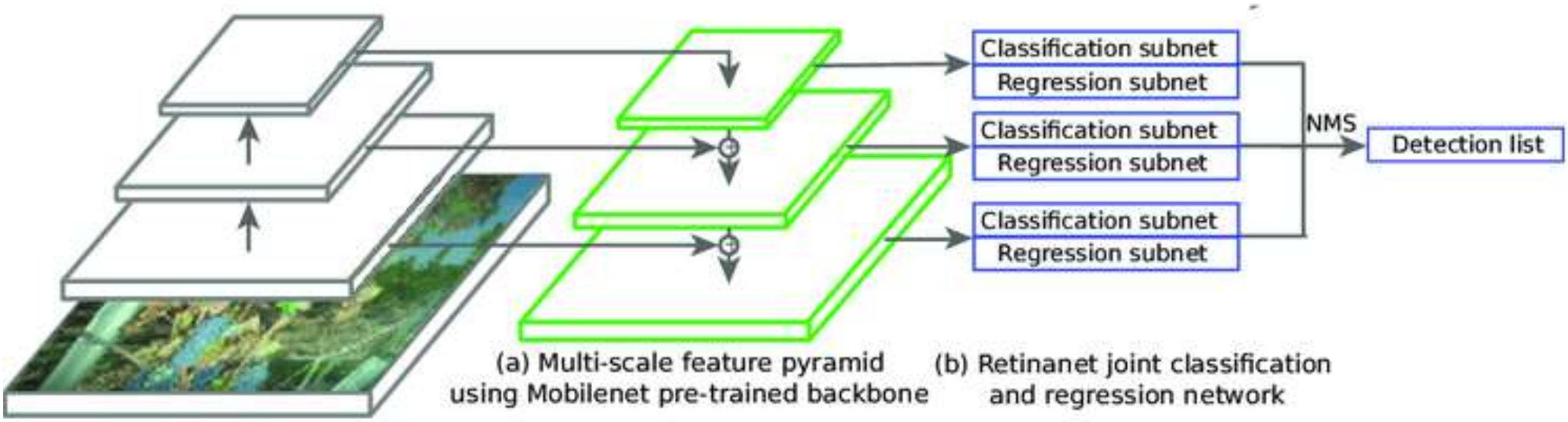
The first step in NMS is to remove all the predicted bounding boxes that have a detection probability that is less than a given NMS threshold.

RetinaNet

***Lin 2020**

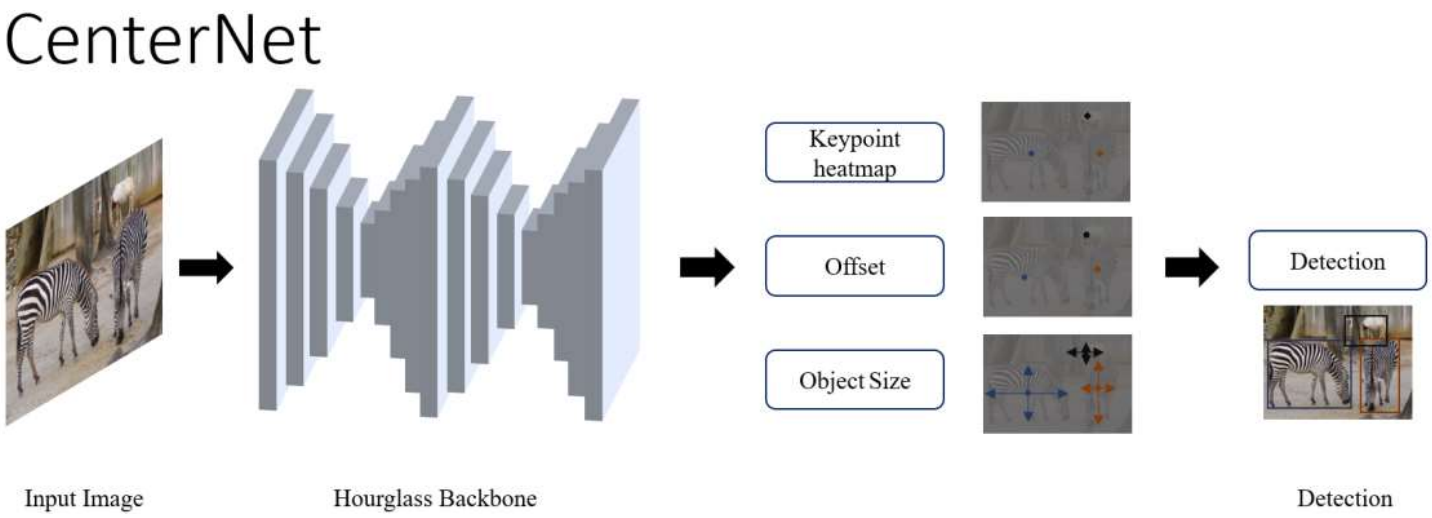


Given the difference between the accuracies of single and two stage detectors, Lin et al. suggested that the reason single stage detectors lag is the “extreme foreground-background class imbalance”
They proposed a reshaped cross entropy loss, called Focal loss as the means to remedy the imbalance.
Focal loss parameter reduces the loss contribution from easy examples.
The authors demonstrate its efficacy with the help of a simple, single stage detector, called RetinaNet, which predicts objects by dense sampling of the input image in location, scale and aspect ratio.
It uses ResNet augmented by Feature Pyramid Network (FPN) as the backbone and two similar subnets - classification and bounding box regressor.


Classification subnet
Regression subnet
Classification subnet
Regression subnet
Classification subnet
Regression subnet
NMS
Detection list
(a) Multi-scale feature pyramid using Mobilenet pre-trained backbone
(b) Retinanet joint classification and regression network

CenterNet

X. Zhou…"Objects as points." 2019



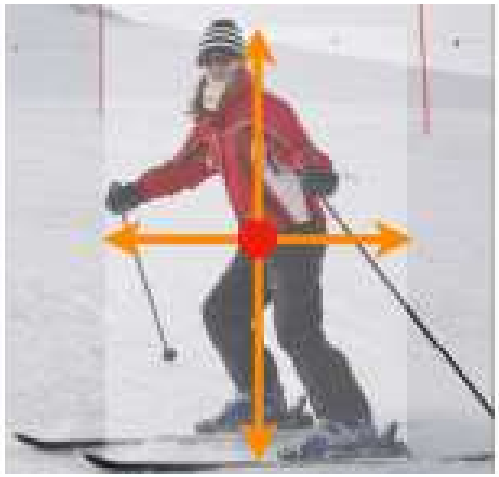
Zhou et al. in [68] takes a very different approach of modelling objects as points, instead of the conventional bounding box representation, CenterNet predicts the object as a single point at the center of the bounding box.

The input image is passed through the FCN that generates a heatmap, whose peaks correspond to center of detected object.

It uses a ImageNet pretrained stacked Hourglass-101 as the feature extractor network and has 3 heads – heatmap head to determine the object center, dimension head to estimate size of object and offset head to correct offset of object point (overlap)

Center Point

Offset Point



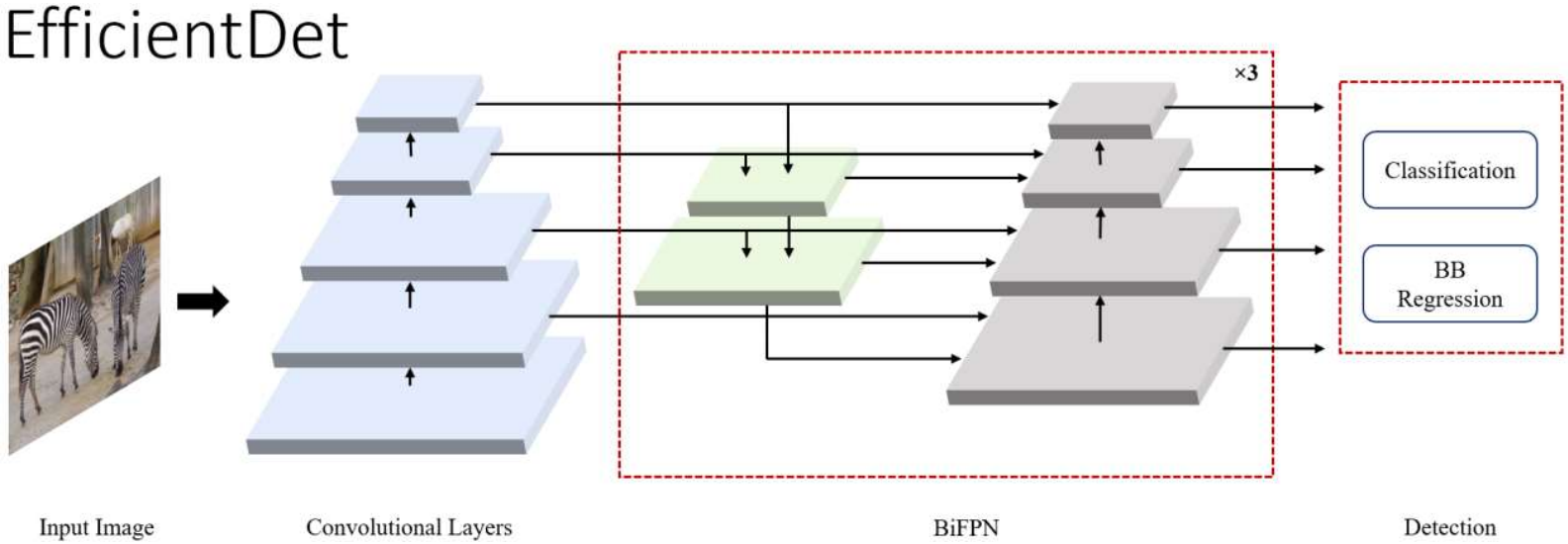
Rectangle

Offset Head

- This head is used to recover from the discretization error caused due to the downsampling of the input.
- After the prediction of the center points, we have to map these coordinates to a higher dimensional input image.
- This will cause a value disturbance as the original image pixel indices are integers and we will be predicting the float values.
- So to solve this issue they predict the local offsets O_hat. These local offset values are shared between objects present in an image.

EfficientDet

*M. Tan 2020



EfficientDet utilizes EfficientNet as the backbone network with multiple sets of BiFPN layers stacked in series as feature extraction network.

it builds towards the idea of scalable detector with higher accuracy and efficiency. It introduces efficient multi-scale features, BiFPN and model scaling. BiFPN is bi-directional feature pyramid network with learnable weights for cross connection of input features at different scales.

.

Yolox

*Zheng Ge ...2021

We switch the YOLO detector to an anchor-free manner and conduct other advanced detection techniques, i.e., *a decoupled head and the leading label assignment strategy SimOTA to achieve state-of-the-art results across a large scale range of models*

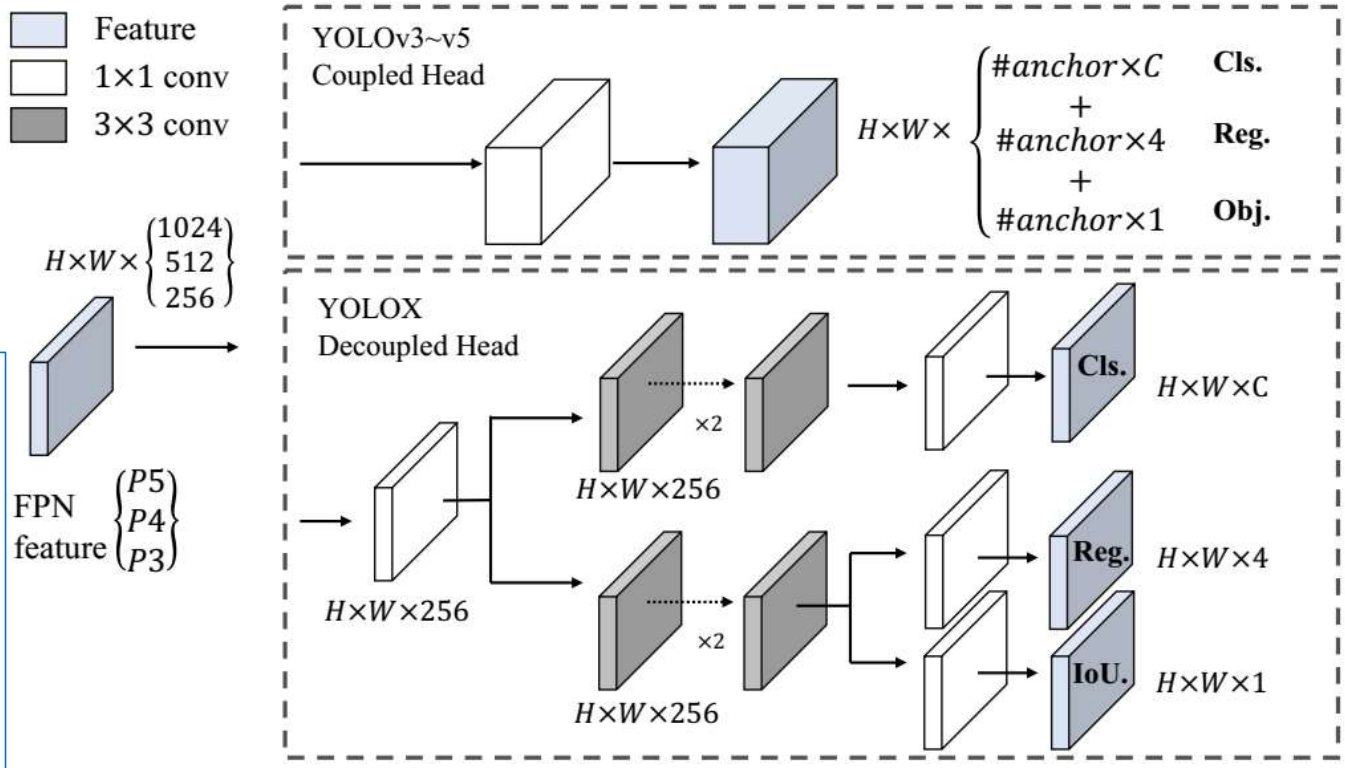
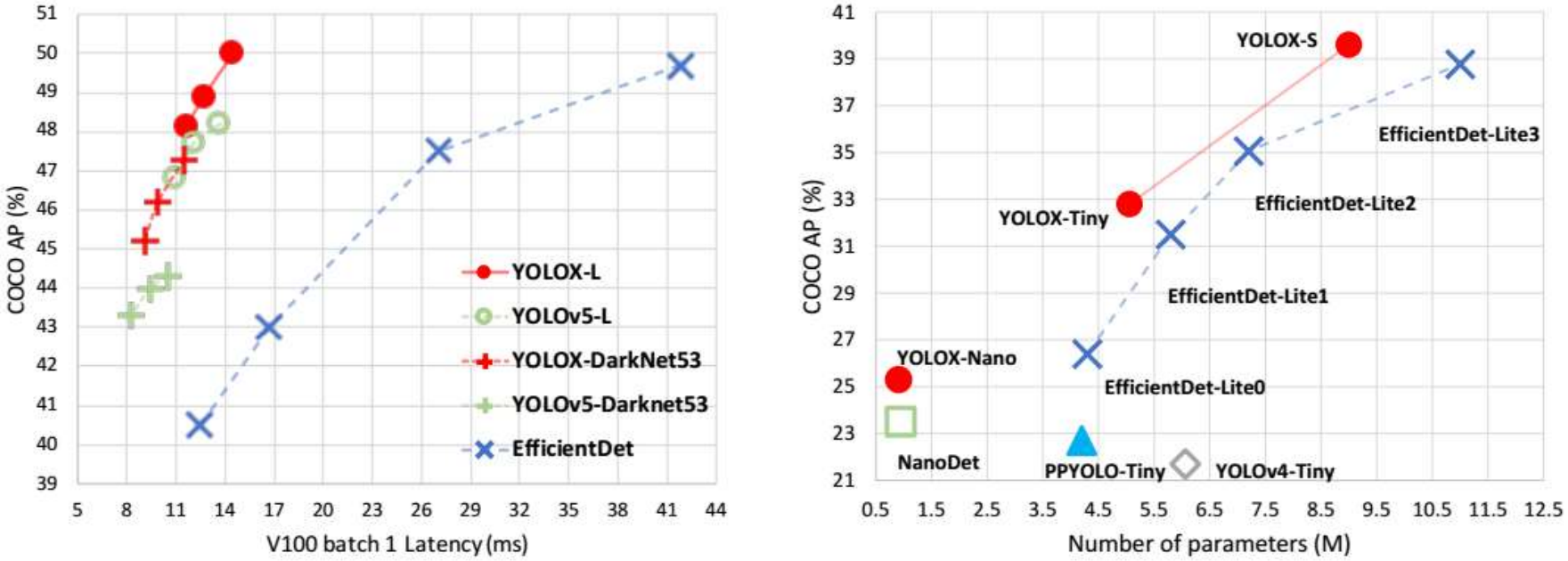


Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a 1 × 1 conv layer to reduce the feature channel to 256 and then add two parallel branches with two 3 × 3 conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch.

YOLOx



Speed-accuracy trade-off of accurate models (top) and Size-accuracy curve of lite models on mobile devices (bottom) for YOLOX and other state-of-the-art object detectors

Methods	AP (%)	Parameters	GFLOPs	Latency	FPS
YOLOv3-ultralytics[2]	44.3	63.00 M	157.3	10.5 ms	95.2
YOLOv3 baseline	38.5	63.00 M	157.3	10.5 ms	95.2
+decoupled head	39.6 **(+1.1)**	63.86 M	186.0	11.6 ms	86.2
+strong augmentation	42.0 **(+2.4)**	63.86 M	186.0	11.6 ms	86.2
+anchor-free	42.9 **(+0.9)**	63.72 M	185.3	11.1 ms	90.1
+multi positives	45.0 **(+2.1)**	63.72 M	185.3	11.1 ms	90.1
+SimOTA	**47.3** **(+2.3)**	63.72 M	185.3	11.1 ms	90.1
+NMS free (optional)	46.5 (-0.8)	67.27 M	205.1	13.5 ms	74.1

Table 2: Roadmap of YOLOX-Darknet53 in terms of AP (%) on COCO *val*. All the models are tested at 640×640 resolution, with FP16-precision and batch=1 on a Tesla V100. The latency and FPS in this table are measured without post-processing.

Open Problems in RCNNs

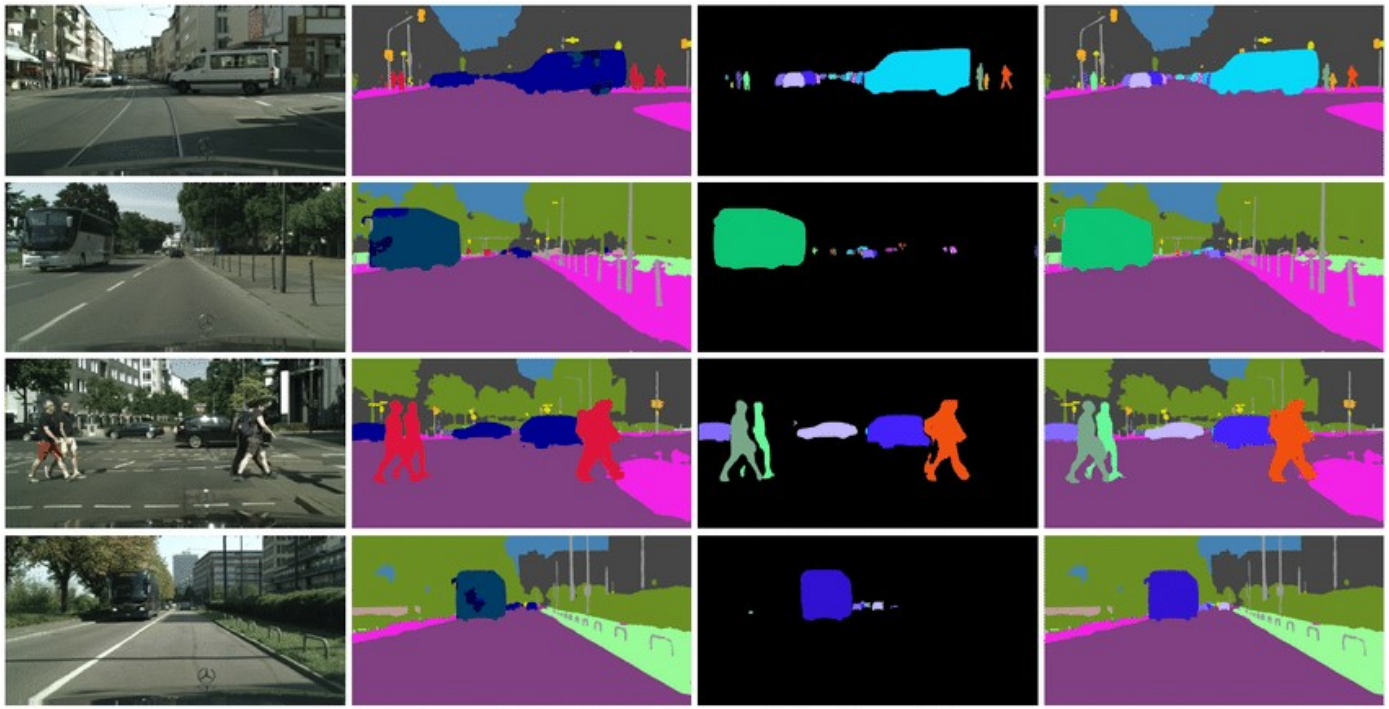
- **AutoML** :The use of automatic neural architecture search (NAS) for determining the characteristics of object detector
- **Lightweight detectors:** While lightweight networks have shown great promise by matching classification errors with the full-fledged models, they still lack in detection accuracy by more than 50%.
- **Weakly supervised/few shot detection:** Most of the state of-the-art object detection models are trained on millions of bounding box annotated data, which is unscalable as annotating data requires time and resources.
- **Domain transfer:** Domain transfer refers to use of a model trained on labeled image of a particular source task on a separate, but related target task. It encourages reuse of trained model and reduces reliance on the availability of a large dataset to achieve high accuracy.
- **3D object detection:** 3D object detection is a particularly critical problem for autonomous driving. Even though models have achieved high accuracy, deployment of anything below human level performance will bring up safety concerns.
- **Object detection in video:** Object detectors are designed to perform on individual image which lack correlation between themselves. Using spatial and temporal relationship between the frames for object recognition is an open problem.

1.2.3 DNNs for Segmentation

DNNs which broke down an image into various subgroups called Image segments

There are three manners for segmentation: Semantic segmentation, Instance segmentation, and Panoptic segmentation

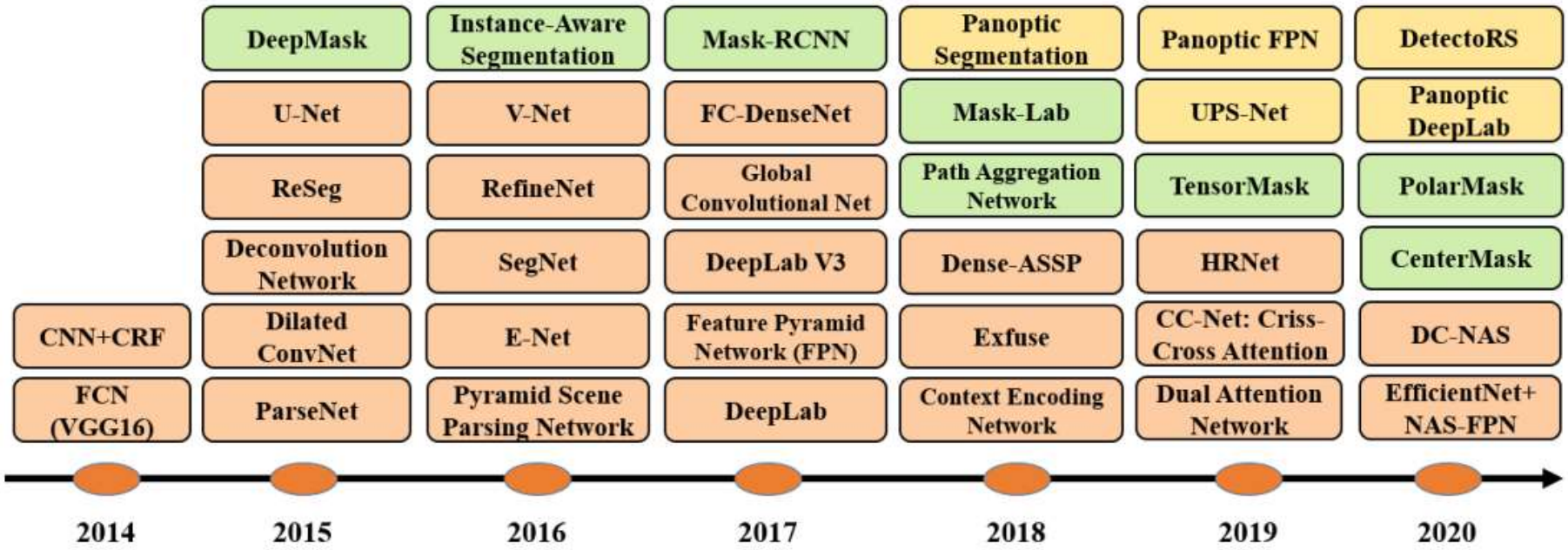
Panoptic segmentation is **proposed to unify the typically distinct tasks of semantic segmentation and instance segmentation**. The proposed task requires the generation of a rich and complete coherent scene segmentation, which is an important step towards a real-world visual system.



Semantic Seg. Instance Seg. Panoptic Seg.

A timeline of DL-based Segmentation algorithms

*S. Minaee 2020



The timeline of DL-based segmentation algorithms for 2D images, from 2014 to 2020.
Orange, green, and yellow blocks refer to semantic, instance, and panoptic segmentation algorithms respectively

Datasets for Segmentation

- 2D images RGB

1. PASCAL Visual Object Classes (VOC)
2. PASCAL Context
3. Microsoft Common Objects in Context (MS COCO)
4. Cityscapes
5. ADE20K /MIT Scene Parsing (SceneParse150)
6. SiftFlow
7. Stanford background
8. Berkeley Segmentation Dataset (BSD)
9. Youtube-Objects
10. KITTI

- 2.5 D images RGB –D

1. NYU-D V2
2. SUN-3D
3. SUN RGB-D
4. UW RGB-D Object Dataset
5. ScanNet

Using RGB and Depth

- 3 D images

1. Stanford 2D-3D
2. ShapeNet Core
3. Sydney Urban
4. Objects Dataset:

3D image datasets are popular in robotic, medical image analysis, 3D scene analysis, and construction applications.
Three dimensional images are usually provided via meshes or other volumetric representations, such as point clouds.

RGB-D

Metrics For Segmentation Models

- **Pixel accuracy**

where P_{ij} is the number of pixels of class *i* predicted as belonging to class *j*.

$$\text{PA} = \frac{\sum_{i=0}^{K} p_{ii}}{\sum_{i=0}^{K} \sum_{j=0}^{K} p_{ij}}$$

- **Mean Pixel Accuracy (MPA)**

MPA is the extended version of PA, in which the ratio of correct pixels is computed in a per-class manner and then averaged over the total number of classes

$$\text{MPA} = \frac{1}{K+1} \sum_{i=0}^{K} \frac{p_{ii}}{\sum_{j=0}^{K} p_{ij}}$$

- **Intersection over Union (IoU)**

$$\text{IoU} = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

(IoU) or the **Jaccard Index** is one of the most commonly used metrics in semantic segmentation. It is defined as the area of intersection between the predicted segmentation map and the ground truth, divided by the area of union between the predicted segmentation map and the ground truth

- **Mean-IoU**

  the average IoU over all classes

-
- **Precision / Recall / F1 score**

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

$$\text{F1-score} = \frac{2\ \text{Prec}\ \text{Rec}}{\text{Prec} + \text{Rec}}$$

- **Dice coefficient**

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \qquad \text{Dice} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} = \text{F1}$$

Comparison

TABLE 1
Accuracies of segmentation models on the PASCAL VOC test set. (* Refers to the model pre-trained on another dataset (such as MS-COCO, ImageNet, or JFT-300M).)

Method	Backbone	mIoU
FCN [31]	VGG-16	62.2
CRF-RNN [39]	-	72.0
CRF-RNN* [39]	-	74.7
BoxSup* [117]	-	75.1
Piecewise* [40]	-	78.0
DPN* [41]	-	77.5
DeepLab-CRF [78]	ResNet-101	79.7
GCN* [118]	ResNet-152	82.2
RefineNet [115]	ResNet-152	84.2
Wide ResNet [119]	WideResNet-38	84.9
PSPNet [56]	ResNet-101	85.4
DeeplabV3 [12]	ResNet-101	85.7
PSANet [98]	ResNet-101	85.7
EncNet [114]	ResNet-101	85.9
DFN* [99]	ResNet-101	86.2
Exfuse [120]	ResNet-101	86.2
SDN* [45]	DenseNet-161	86.6
DIS [123]	ResNet-101	86.8
DM-Net* [58]	ResNet-101	87.06
APC-Net* [60]	ResNet-101	87.1
EMANet [95]	ResNet-101	87.7
DeeplabV3+ [83]	Xception-71	87.8
Exfuse [120]	ResNeXt-131	87.9
MSCI [61]	ResNet-152	88.0
EMANet [95]	ResNet-152	88.2
DeeplabV3+* [83]	Xception-71	89.0
EfficientNet+NAS-FPN [135]	-	90.5

TABLE 2
Accuracies of segmentation models on the Cityescapes dataset.

Method	Backbone	mIoU
FCN-8s [31]	-	65.3
DPN [41]	-	66.8
Dilation10 [79]	-	67.1
DeeplabV2 [78]	ResNet-101	70.4
RefineNet [115]	ResNet-101	73.6
FoveaNet [124]	ResNet-101	74.1
Ladder DenseNet [125]	Ladder DenseNet-169	73.7
GCN [118]	ResNet-101	76.9
DUC-HDC [80]	ResNet-101	77.6
Wide ResNet [119]	WideResNet-38	78.4
PSPNet [56]	ResNet-101	85.4
BiSeNet [126]	ResNet-101	78.9
DFN [99]	ResNet-101	79.3
PSANet [98]	ResNet-101	80.1
DenseASPP [81]	DenseNet-161	80.6
SPGNet [127]	2xResNet-50	81.1
DANet [93]	ResNet-101	81.5
CCNet [96]	ResNet-101	81.4
DeeplabV3 [12]	ResNet-101	81.3
AC-Net [129]	ResNet-101	82.3
OCR [44]	ResNet-101	82.4
GS-CNN [128]	WideResNet	82.8
HRNetV2+OCR (w/ASPP) [44]	HRNetV2-W48	83.7
Hierarchical MSA [137]	HRNet-OCR	85.1

TABLE 4
Accuracies of segmentation models on the ADE20k validation dataset.

Method	Backbone	mIoU
FCN [31]	-	29.39
DilatedNet [79]	-	32.31
CascadeNet [132]	-	34.9
RefineNet [115]	ResNet-152	40.7
PSPNet [56]	ResNet-101	43.29
PSPNet [56]	ResNet-269	44.94
EncNet [114]	ResNet-101	44.64
SAC [133]	ResNet-101	44.3
PSANet [98]	ResNet-101	43.7
UperNet [134]	ResNet-101	42.66
DSSPN [130]	ResNet-101	43.68
DM-Net [58]	ResNet-101	45.5
AC-Net [129]	ResNet-101	45.9

TABLE 3
Accuracies of segmentation models on the MS COCO stuff dataset.

Method	Backbone	mIoU
RefineNet [115]	ResNet-101	33.6
CCN [59]	Ladder DenseNet-101	35.7
DANet [93]	ResNet-50	37.9
DSSPN [130]	ResNet-101	37.3
EMA-Net [95]	ResNet-50	37.5
SGR [131]	ResNet-101	39.1
OCR [44]	ResNet-101	39.5
DANet [93]	ResNet-101	39.7
EMA-Net [95]	ResNet-50	39.9
AC-Net [129]	ResNet-101	40.1
OCR [44]	HRNetV2-W48	40.5

TABLE 5
Instance Segmentation Models Performance on COCO test-dev 2017

Method	Backbone	FPS	AP
YOLACT-550 [76]	R-101-FPN	33.5	29.8
YOLACT-700 [76]	R-101-FPN	23.8	31.2
RetinaMask [170]	R-101-FPN	10.2	34.7
TensorMask [69]	R-101-FPN	2.6	37.1
SharpMask [171]	R-101-FPN	8.0	37.4
Mask-RCNN [64]	R-101-FPN	10.6	37.9
CenterMask [74]	R-101-FPN	13.2	38.3

TABLE 6
Panoptic Segmentation Models Performance on the MS-COCO val dataset. $*$ denotes use of deformable convolution.

Method	Backbone	PQ
Panoptic FPN [139]	ResNet-50	39.0
Panoptic FPN [139]	ResNet-101	40.3
AU-Net [140]	ResNet-50	39.6
Panoptic-DeepLab [142]	Xception-71	39.7
OANet [172]	ResNet-50	39.0
OANet [172]	ResNet-101	40.7
AdaptIS [173]	ResNet-50	35.9
AdaptIS [173]	ResNet-101	37.0
UPSNet* [143]	ResNet-50	42.5
OCFusion* [174]	ResNet-50	41.3
OCFusion* [174]	ResNet-101	43.0
OCFusion* [174]	ResNeXt-101	45.7

Taxonomy of DNNs for Segmentation

1. Fully Conventional Neural Networks
2. Convolutional Models With Graphical Models
3. Encoder-Decoder Based Models
4. Multi-Scale and Pyramid Network Based Models
5. R-CNN Based Models (for Instance Segmentation)
6. Dilated Convolutional Models and DeepLab Family
7. Recurrent Neural Network Based Models
8. Attention-Based Models
9. Generative Models and Adversarial Training
10. **CNN Models With Active Contour Models**

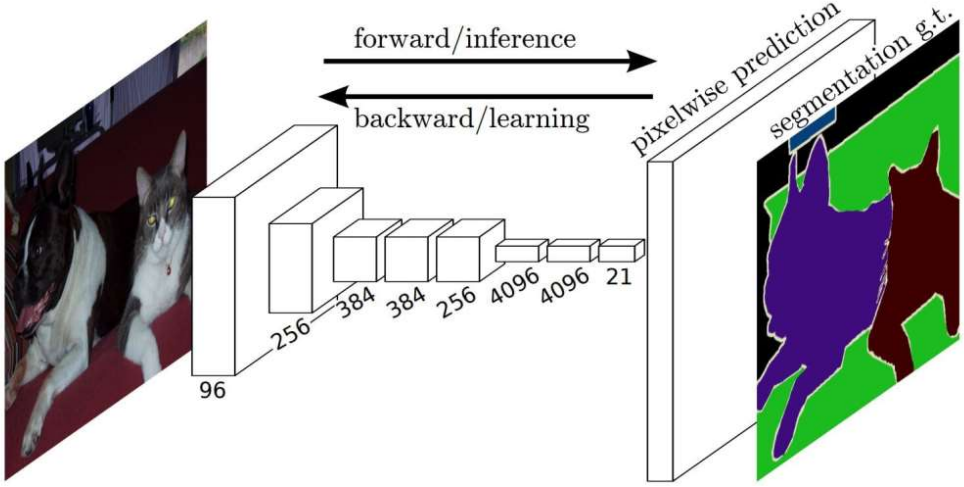
1. Fully Conventional Neural Networks

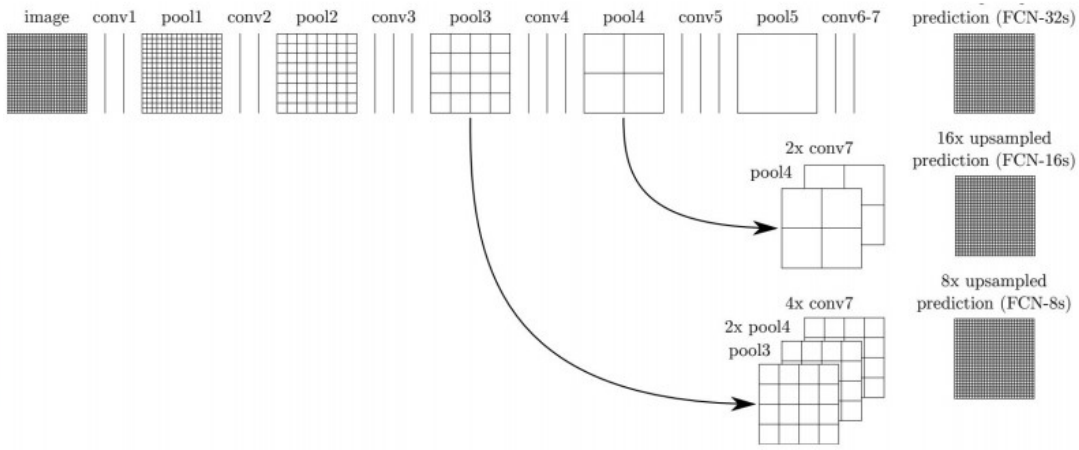
It includes only convolutional layers

The applied backbones is CNN architectures such as VGG16/GoogLeNet

FCN

J. Long... 2015

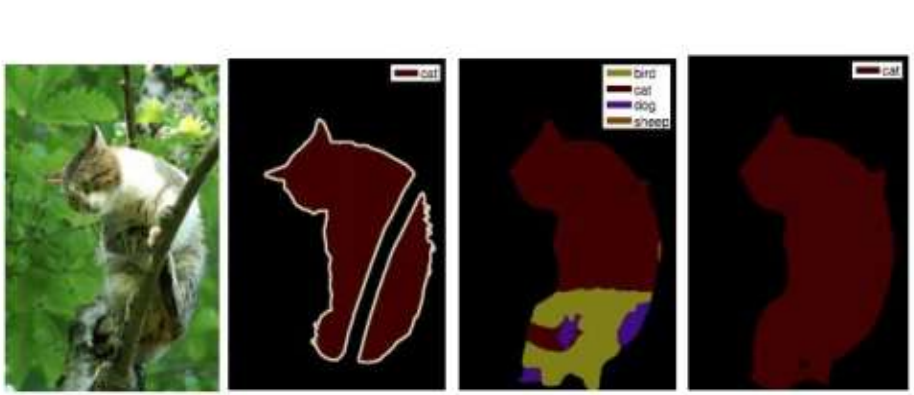


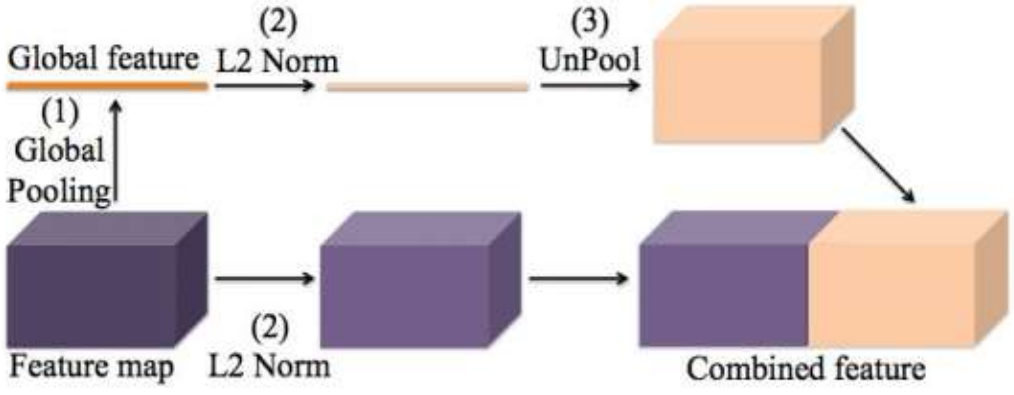


Through the use of skip connections in which feature maps from the final layers of the model are up-sampled and fused with feature maps of earlier layers, the model combines semantic information (from deep, coarse layers) and appearance information (from shallow, fine layers) in order to produce accurate and detailed segmentations.

ParseNet

W. Liu..2015





ParseNet adds global context to FCNs by using the average feature for a layer to augment the features at each location.

The feature map for a layer is pooled over the whole image resulting in a context vector.

This context vector is normalized and un-pooled to produce new feature maps of the same size as the initial ones.

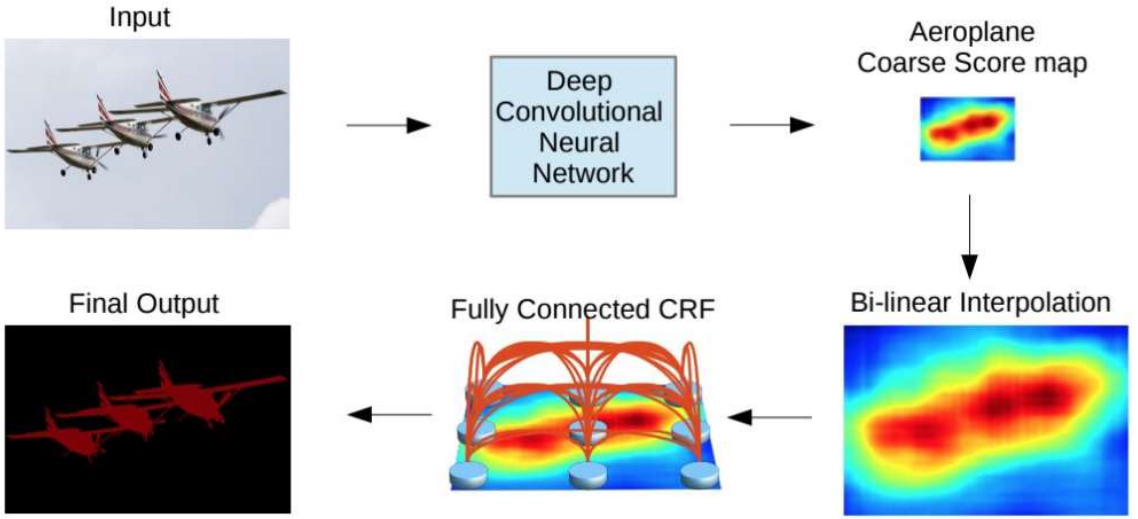
2. Convolutional Models With Graphical Models

As discussed, FCN ignores potentially useful scene-level semantic context. To integrate more context, several approaches incorporate probabilistic graphical models, such as Conditional Random Fields (CRFs) and Markov Random Field (MRFs), into DL architectures.

CNN+CRF (Condition Random Field)

Chen..2014

Chen proposed a semantic segmentation algorithm based on the combination of CNNs and fully connected CRFs. (CNN+CRF)
They showed that responses from the final layer of deep CNNs are not sufficiently localized for accurate object segmentation.
To overcome the poor localization property of deep CNNs, they combined the responses at the final CNN layer with a fully-connected CRF.
They showed that their model is able to localize segment boundaries at a higher accuracy rate than it was possible with previous methods.



The coarse score map of a CNN is upsampled via interpolated interpolation, and fed to a fully-connected CRF to refine the segmentation result.

Conditional random fields (CRFs) are **a class of statistical modeling methods used for structured prediction**. Whereas a classifier predicts a label for a single sample without onsidering "neighbouring" samples, a CRF can take context into account.

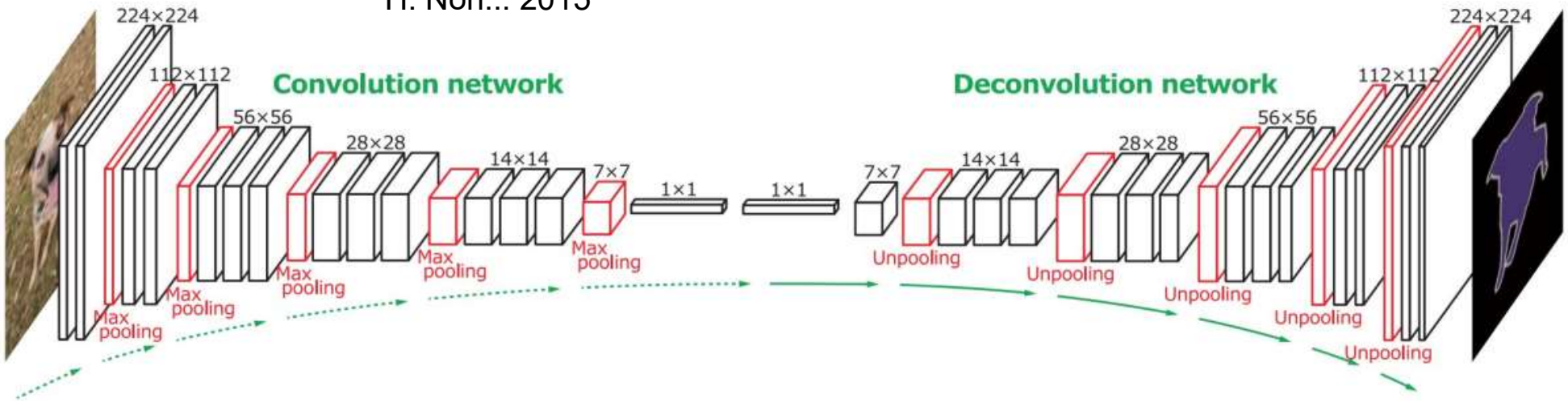
3. Encoder-Decoder Based Models

image segmentation based on the convolutional encoder-decoder architecture.

A. Encoder-Decoder Models for General Segmentation

DSS (Deconvolutional semantic segmentation)

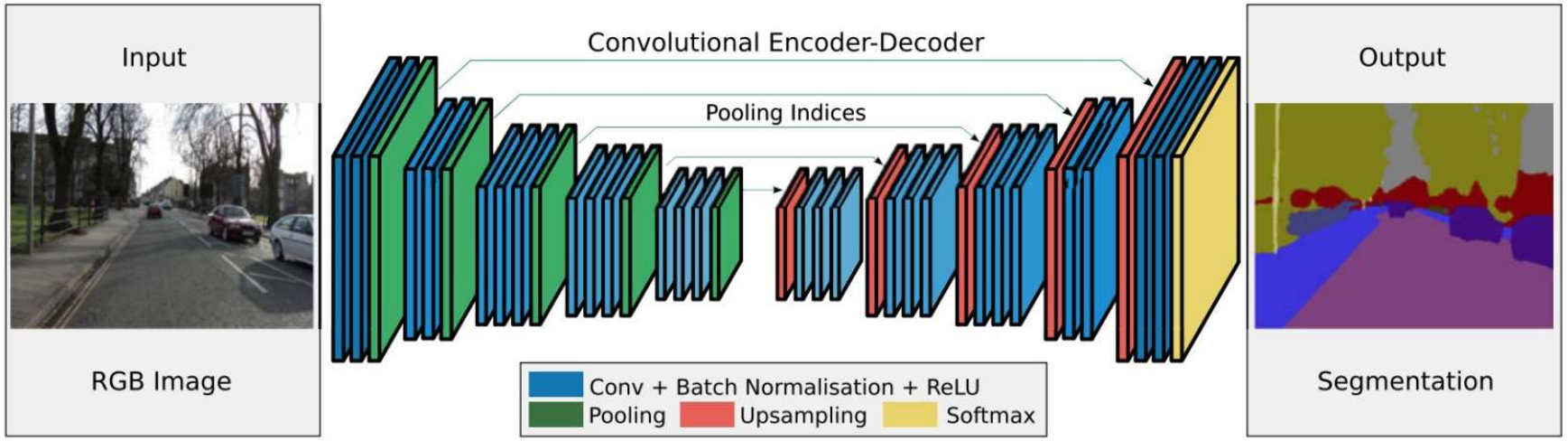
H. Noh... 2015



A convolution network based on the VGG 16-layer net, is a multi-layer deconvolution network to generate the accurate segmentation map.

SegNet

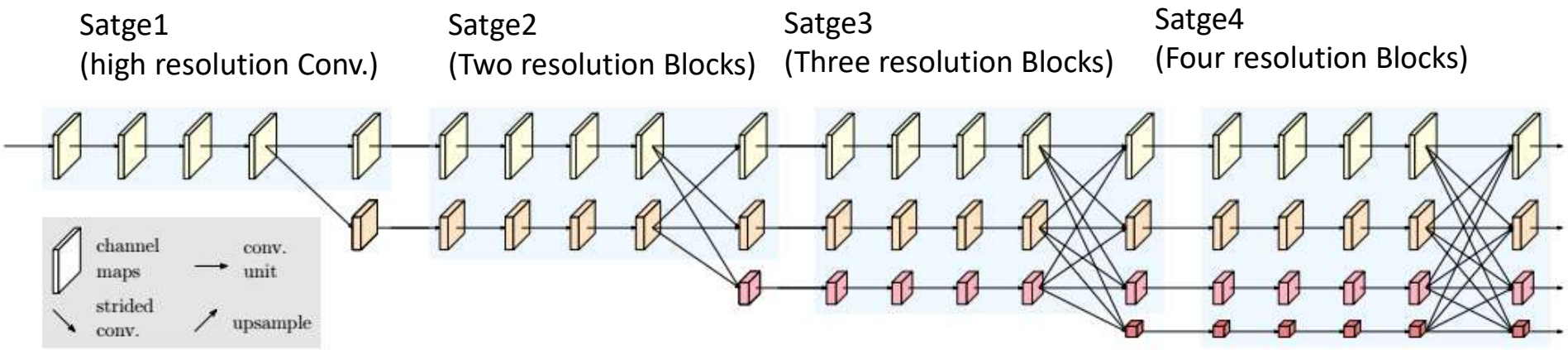
A. Kendall ...2015



SegNet has no fully-connected layers; hence, the model is fully convolutional. A decoder up-samples its input using the transferred pool indices from its encoder to produce a sparse feature map(s)

HRNET(high-resolution network)

Y. Yuan...2019



HRNet consists of parallel high-to-low resolution convolution streams with repeated information exchange across multi-resolution steams.

There are four stages:

The 1st stage consists of high-resolution convolutions.

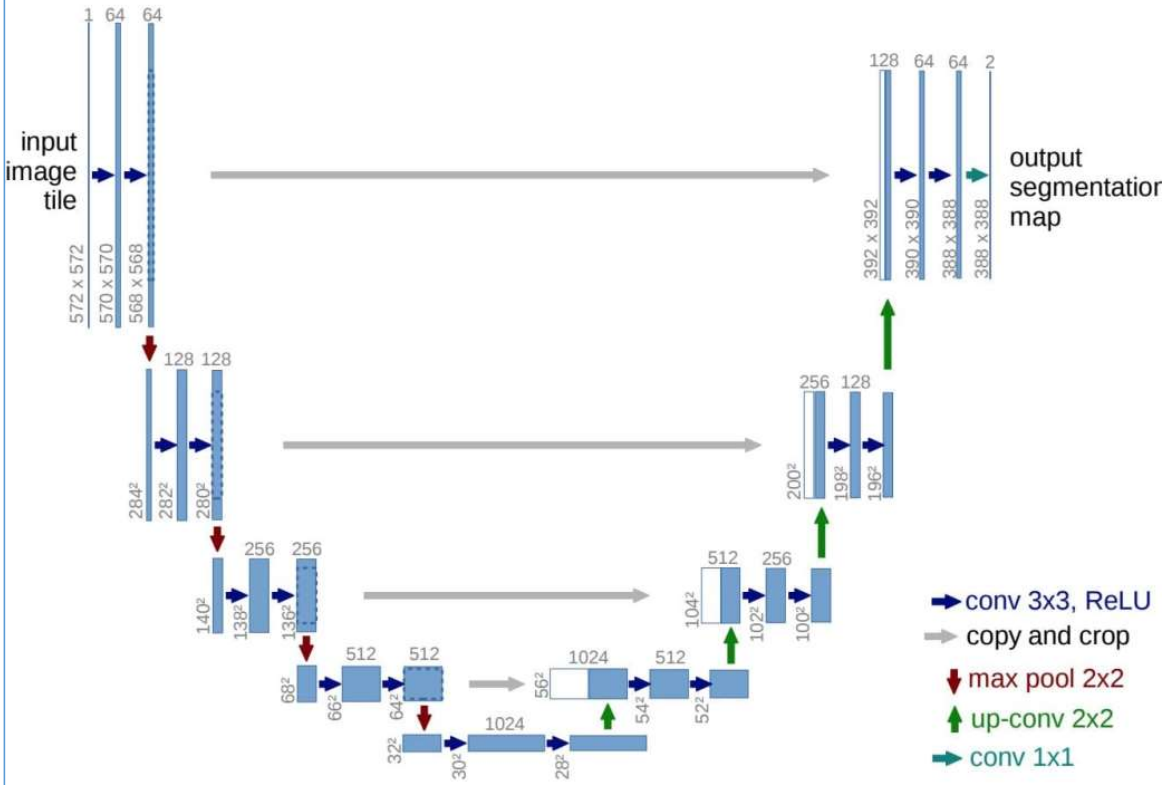
The 2nd (3rd, 4th) stage repeats two-resolution (three-resolution, four-resolution) blocks.

B. Encoder-Decoder Models for Medical and Biomedical Image Segmentation

UNet

Ronneberger....2015

U-Net is proposed for segmenting biological microscopy images.
Their network and training strategy relies on the use of data augmentation to learn from the very few annotated images effectively.
The U-Net architecture comprises two parts, a contracting path to capture context, and a symmetric expanding path that enables precise localization.
The down-sampling or contracting part has a FCN-like architecture that extracts features with 3 × 3 convolutions. The up-sampling or expanding part uses up-convolution (or deconvolution), reducing the number of feature maps while increasing their dimensions.
Feature maps from the down-sampling part of the network are copied to the up-sampling part to avoid losing pattern information.

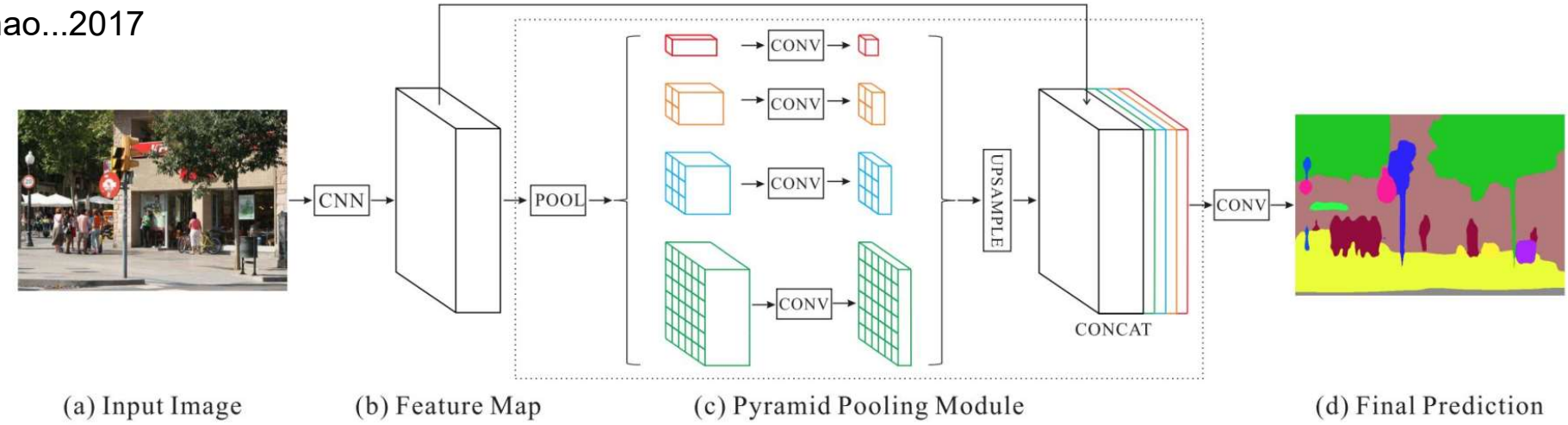


4. Multi-Scale and Pyramid Network Based Models

DNNs whose Backbone is from CNNs with Multi-Scale and Pyramid Network

PSPN Pyramid scene parsing network

Zhao...2017

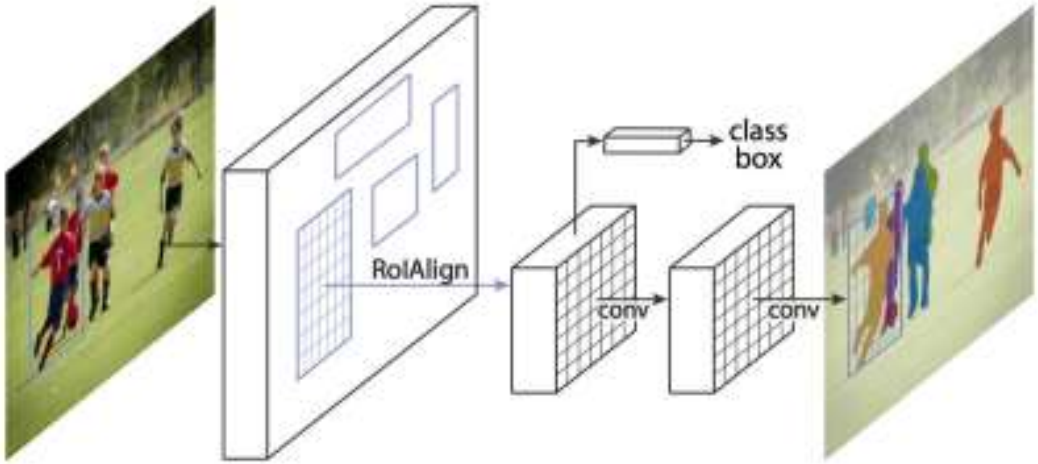


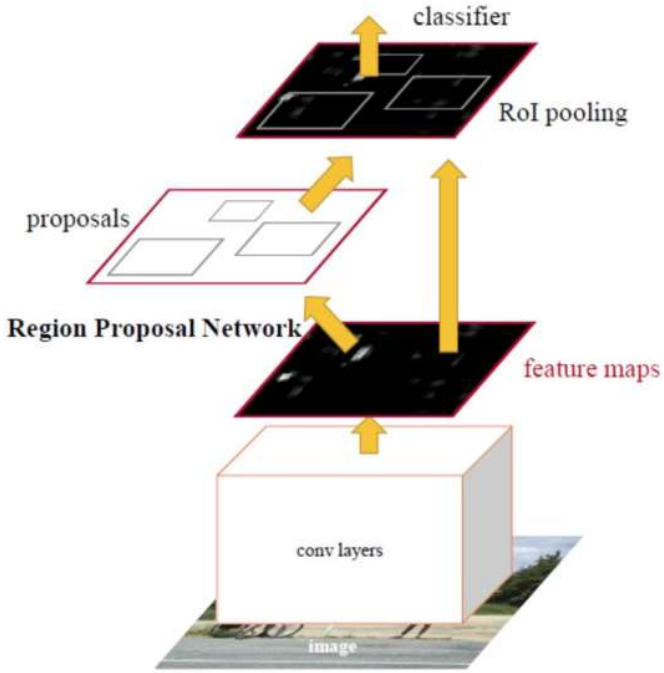
A CNN produces the feature map and a pyramid pooling module aggregates the different sub-region representations.
Up-sampling and concatenation are used to form the final feature representation from which, the final pixel-wise prediction is obtained through convolution.

5. R-CNN Based Models (for Instance Segmentation)

Mask R-CNN

K. He...2017



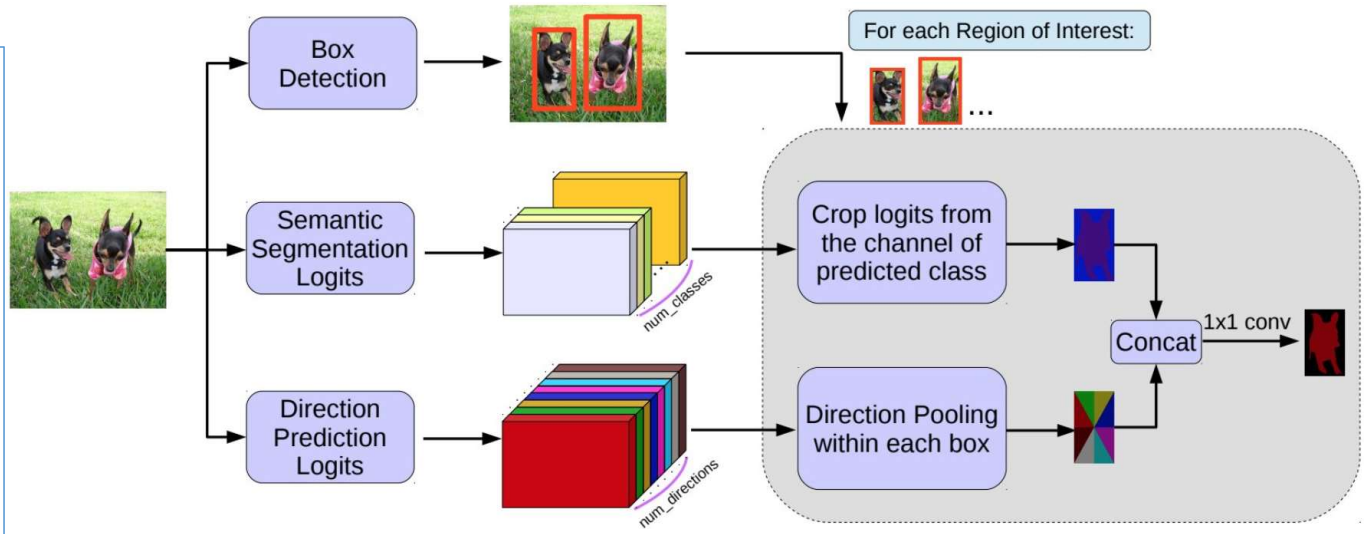


Mask R-CNN architecture for instance segmentation

MaskLab(Instance Segmentation)

L. Chen...2018

A model by refining object detection with semantic and direction features based on Faster R-CNN.
This model produces three outputs, box detection, semantic segmentation, and direction prediction.
Building on the FasterRCNN object detector, the predicted boxes provide accurate localization of object instances.
Within each region of interest, MaskLab performs foreground/background segmentation by combining semantic and direction prediction.



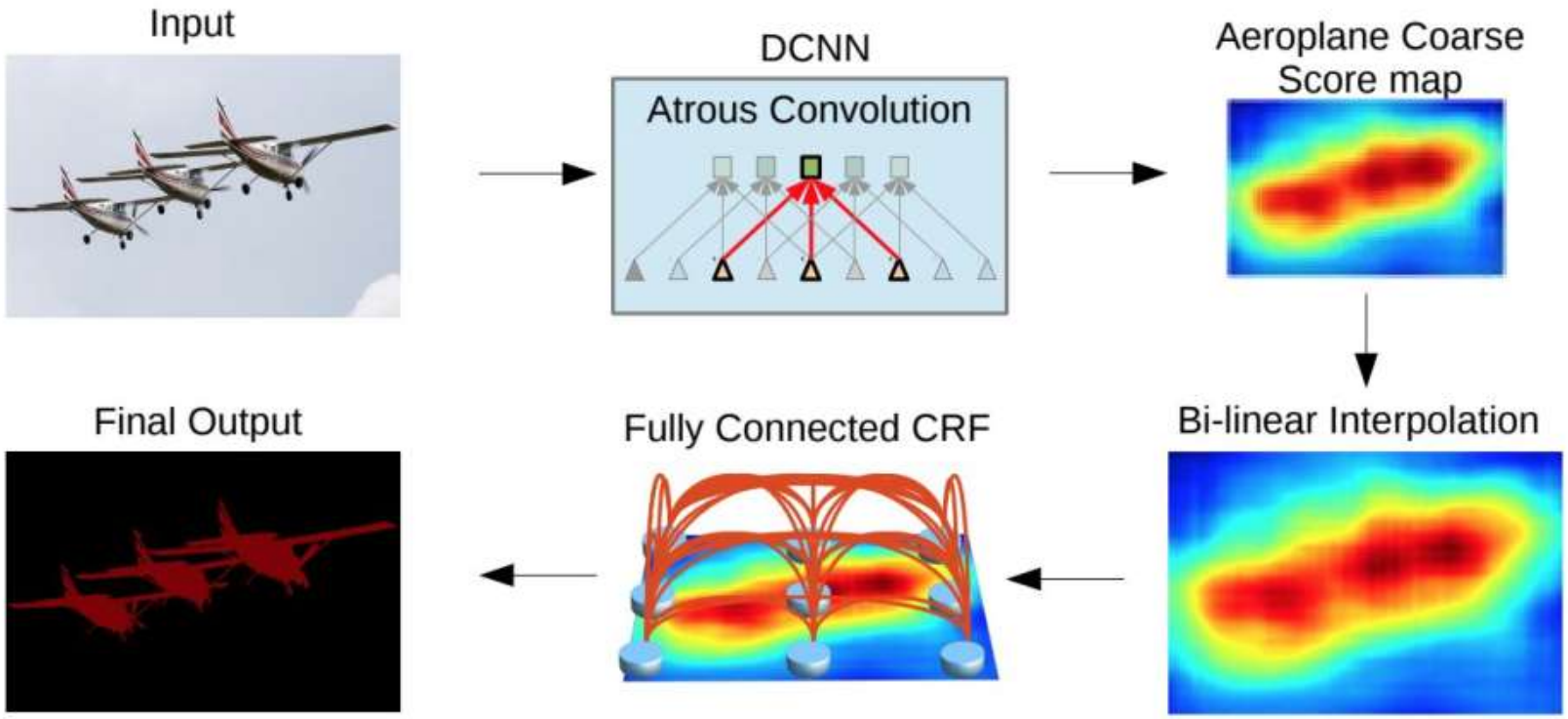
MaskLab generates three outputs—refined box predictions (from Faster R-CNN), semantic segmentation logits for pixel-wise classification, and direction prediction logits for predicting each pixel's direction toward its instance center.

6. Dilated Convolutional Models and DeepLab Family

DeepLab

Chen. 2017

Similar to CNN+CRF 2014

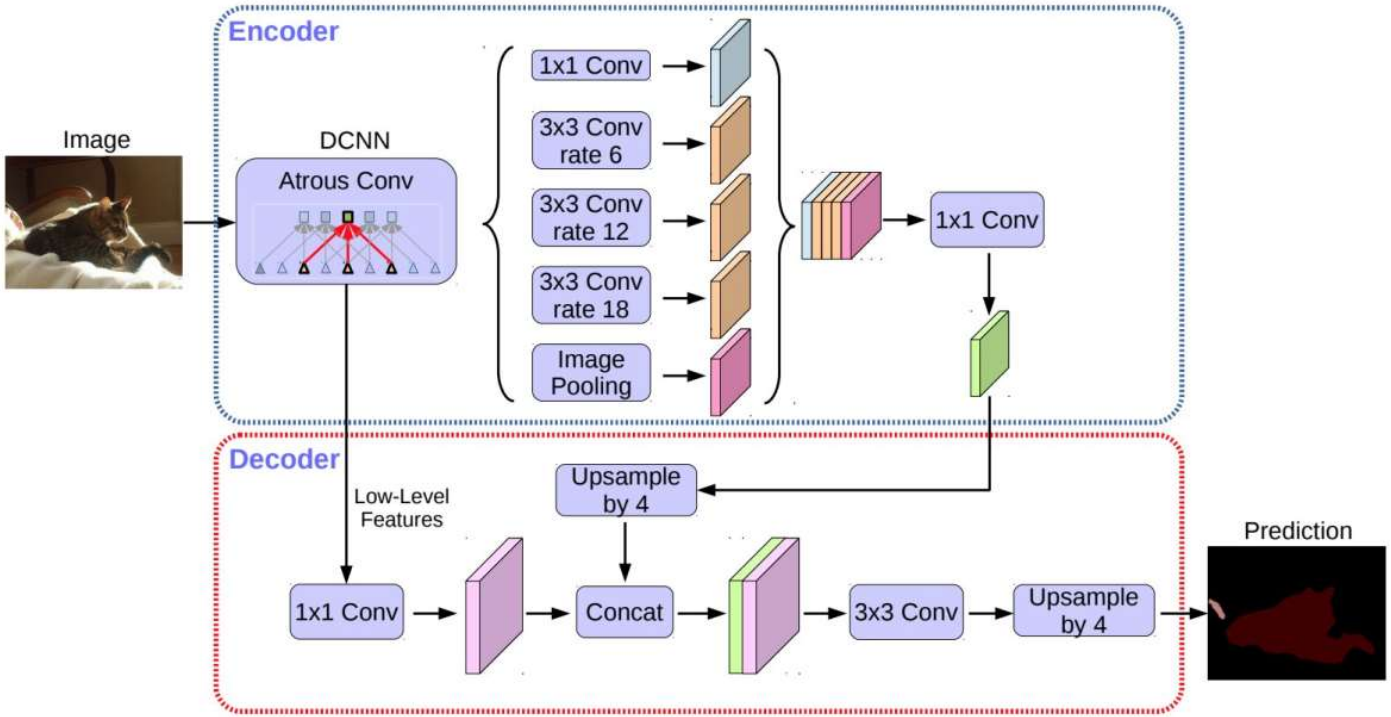


A CNN model such as VGG-16 or ResNet-101 is employed in fully convolutional fashion, using dilated convolution. A bilinear interpolation stage enlarges the feature maps to the original image resolution. Finally, a fully connected CRF(Conditional Random Field) refines the segmentation result to better capture the object boundaries.

DeepLabv3+

Chen....2018

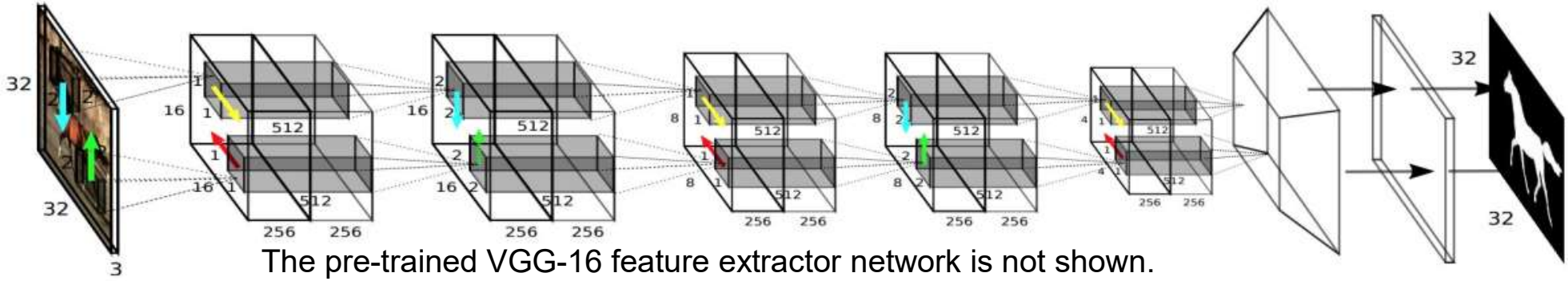
Deeplabv3+ uses an encoder-decoder architecture, including atrous separable convolution, composed of a depthwise convolution (spatial convolution for each channel of the input) and pointwise convolution (1×1 convolution with the depthwise convolution as input).
They used the DeepLabv3 framework as encoder.
The most relevant model has a modified Xception backbone with more layers, dilated depthwise separable convolutions instead of max pooling and batch normalization.



7. Recurrent Neural Network Based Models

ReSeg

Visin ..2016



The pre-trained VGG-16 feature extractor network is not shown.

This model is mainly based on ReNet (which was developed for image classification).

Each ReNet layer is composed of four RNNs that sweep the image horizontally and vertically in both directions, encoding patches/activations, and providing relevant global information.

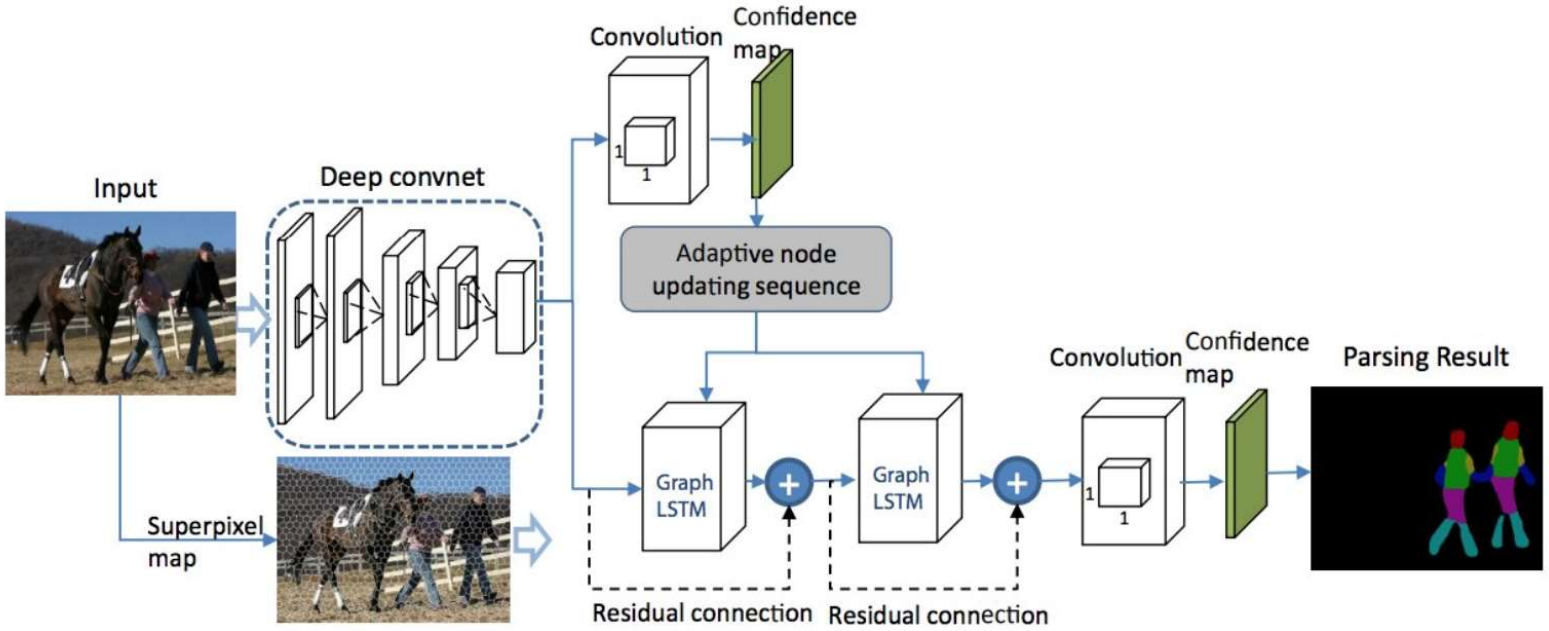
To perform image segmentation with the ReSeg model , ReNet layers are stacked on top of pre-trained VGG-16 convolutional layers that extract generic local features.

ReNet layers are then followed by up-sampling layers to recover the original image resolution in the final predictions.

Gated Recurrent Units (GRUs) are used because they provide a good balance between memory usage and computational power.

Graph-LSTM

Liang 2016

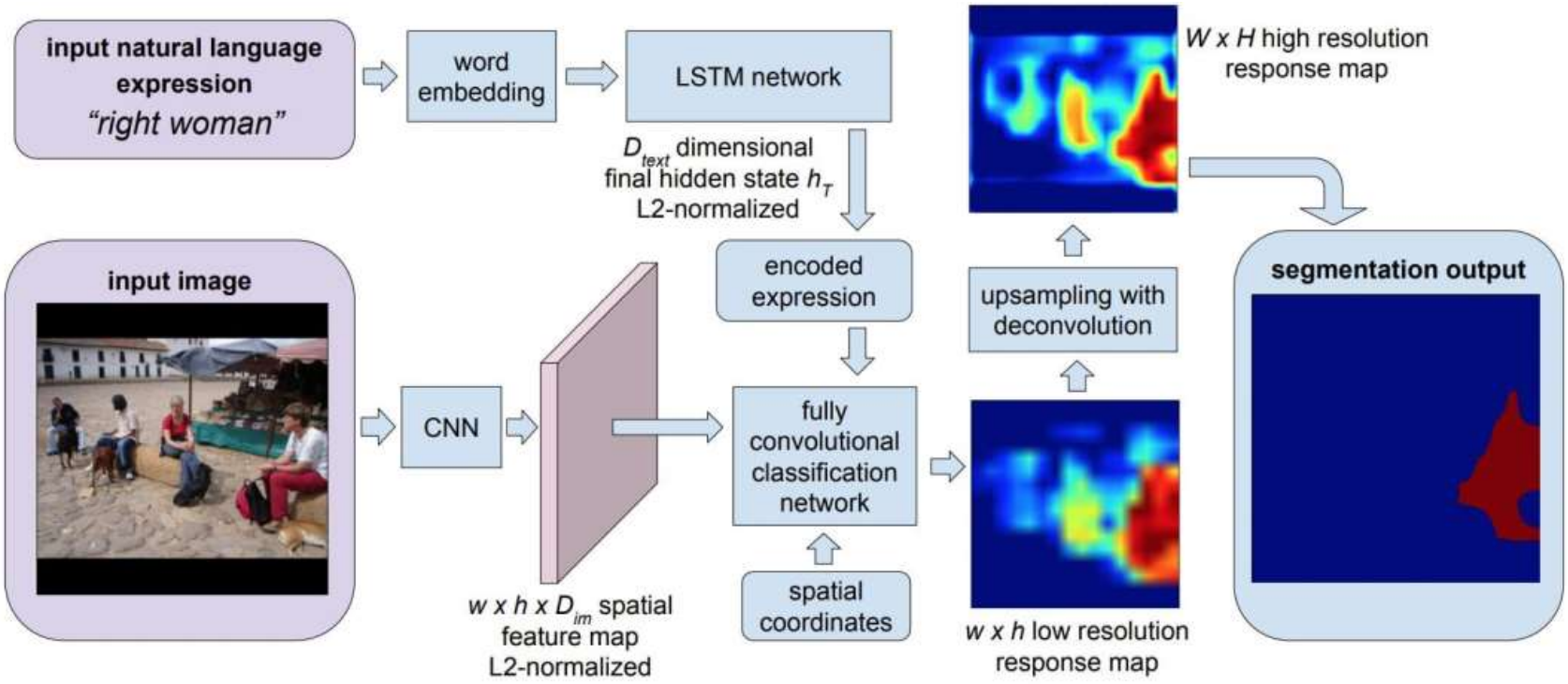


A semantic segmentation model based on the Graph LSTM network is proposed. (Graph LSTM : A generalization of LSTM from sequential data or multidimensional data to general graph-structured data).
Instead of evenly dividing an image to pixels or patches in existing multi-dimensional LSTM structures (e.g., row, grid and diagonal LSTMs), they take each arbitrary-shaped super pixel as a semantically consistent node, and adaptively construct an undirected graph for the image, where the spatial relations of the super pixels are naturally used as edges.
To adapt the Graph LSTM model to semantic segmentation, LSTM layers built on a super-pixel map are appended on the convolutional layers to enhance visual features with global structure context.

CNN+LSTM

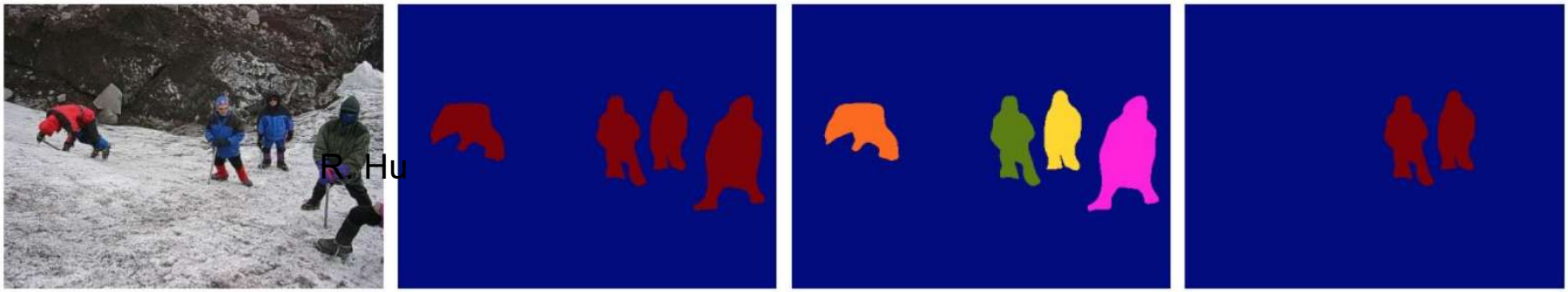
R. Hu...2016

The CNN+LSTM architecture for segmentation from natural language expressions.



To produce pixel-wise segmentation for language expression, they propose an end-to-end trainable recurrent and convolutional model that jointly learns to process visual and linguistic information.
In the considered model, a recurrent LSTM network is used to encode the referential expression into a vector representation, and an FCN is used to extract a spatial feature map from the image and output a spatial response map for the target object.

An example segmentation result of this model (for the query "people in blue coat") is shown.



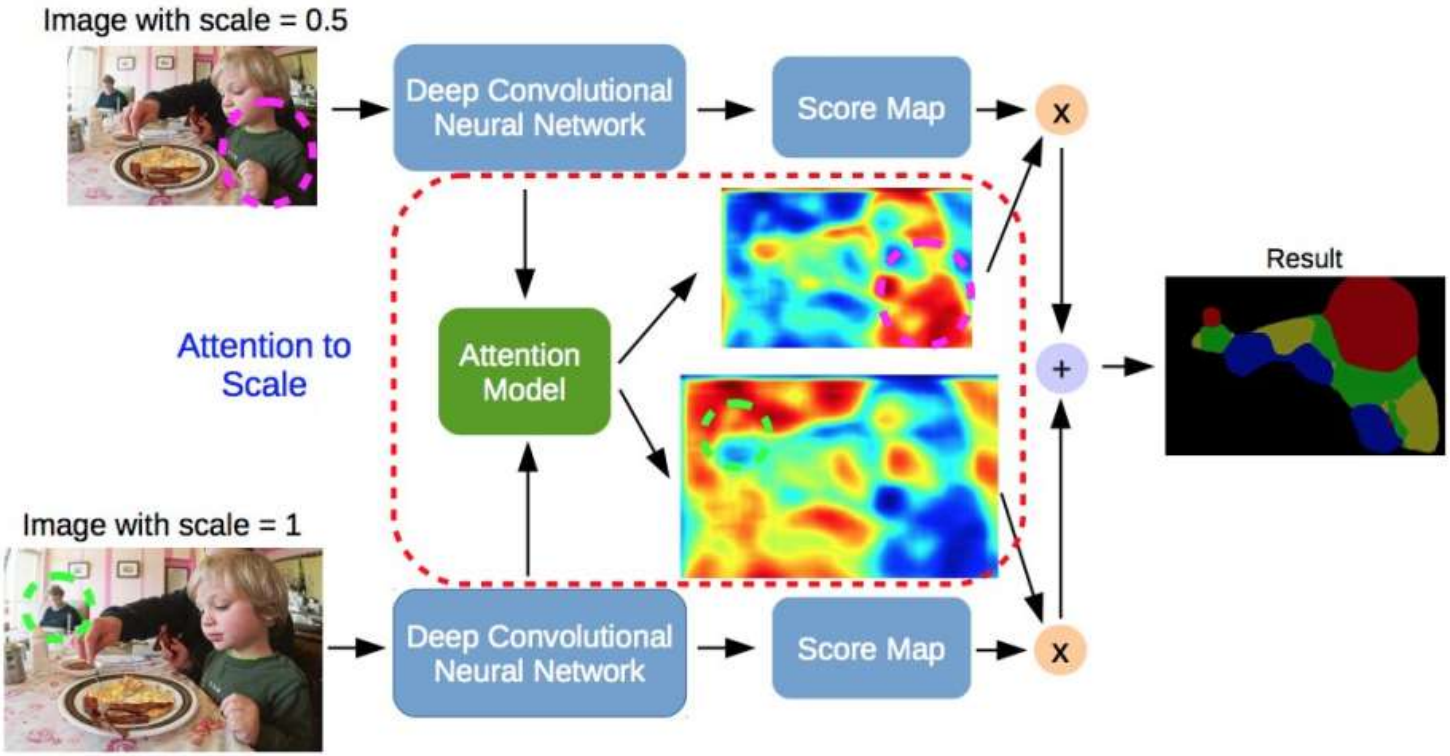
(a) input image (b) object class segmentation of class ***people*** (c) object instance segmentation of class ***people*** (d) segmentation from expression *"people in blue coat"*

Segmentation masks generated for the query "people in blue coat"

8. Attention-Based Models

Attention-based semantic segmentation model

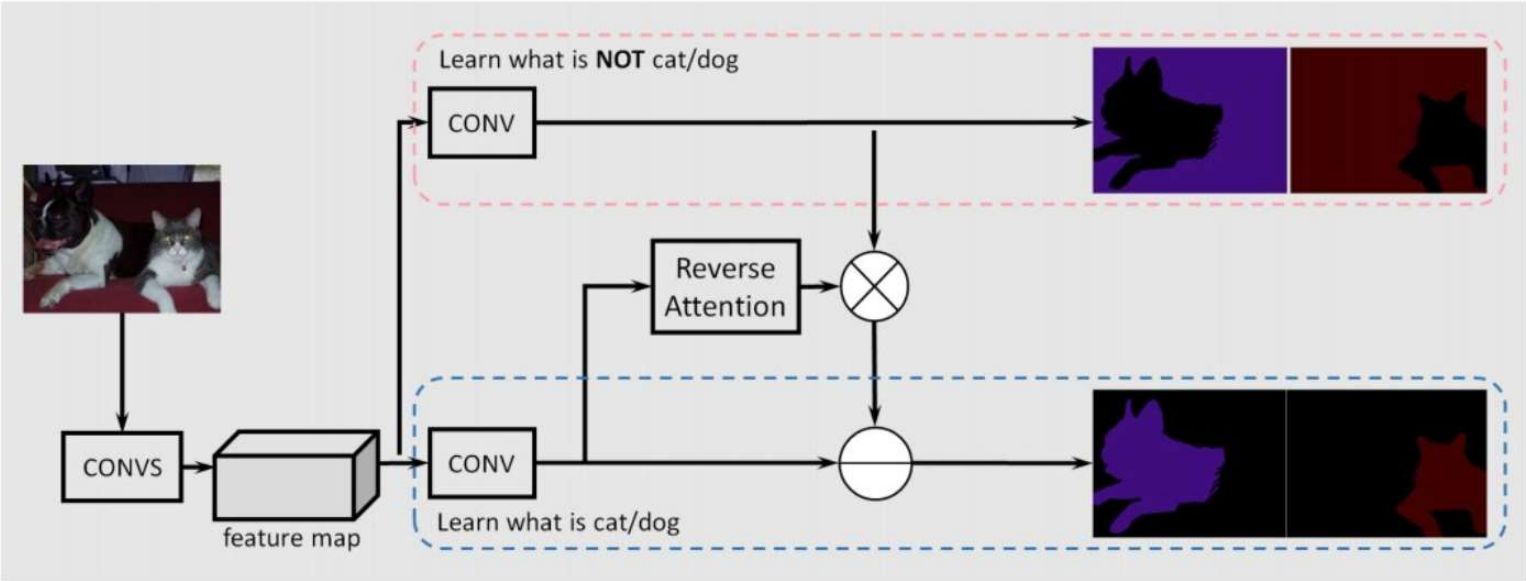
Chen ...2016



The attention model learns to assign different weights to objects of different scales.
e.g., the model assigns large weights on the small person (green dashed circle) for features from scale 1.0, and large weights on the large child (magenta dashed circle) for features from scale 0.5.

Semantic segmentation with reverse attention

Huang....2017



In contrast to other works in which convolutional classifiers are trained to learn the representative semantic features of labeled objects, Huang *et al* proposed a semantic segmentation approach using reverse attention mechanisms. Their Reverse Attention Network (RAN) architecture trains the model to capture the opposite concept (i.e., features that are not associated with a target class) as well. The RAN is a three-branch network that performs the direct, and reverse-attention learning processes simultaneously.

Other categories

9 Generative Models and Adversarial Training

C. Yu.. 2018:"Learning a discriminative feature network for semantic segmentation"

N. Souly....2016, "Semi supervised semantic segmentation using generative adversarial network'"

C. Hung....2018"Adversarial learning for semi-supervised semantic segmentation,"
Y. Xue.....2018 "Segan: Adversarial network with multi-scale loss for medical image segmentation,"

Majurski...2019"Cell image segmentation using generative adversarial networks, transfer learning, and augmentations,"

10 CNN Models With Active Contour Models

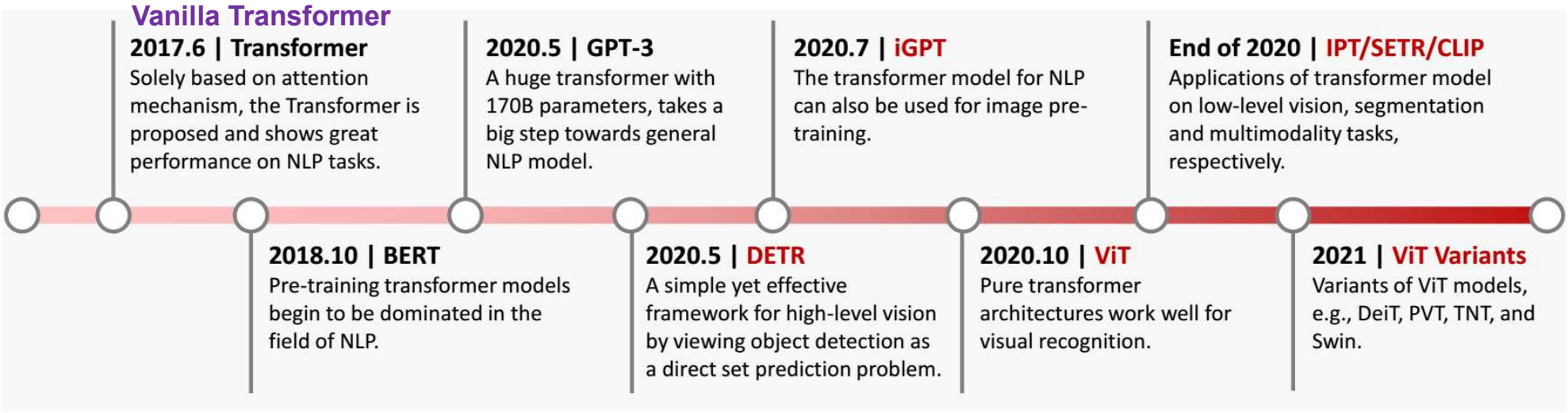
Chen...2019"Learning active contour models for medical image segmentation,"
Le...1028 "Reformulating level sets as deep recurrent neural network approach to semantic segmentation,"

Rupprecht...2016 "**Deep active contours**,"

1.2.4 Transformers

A transformer is a deep learning model that generates significant weights to each part of the input data through using mechanisms of self-attention and cross-attention.
It is used primarily in the fields of natural language processing (NLP), computer vision (CV), and speech processing.

- TimeLine of Key milestones in the development of transformer * Kai Han Feb2022

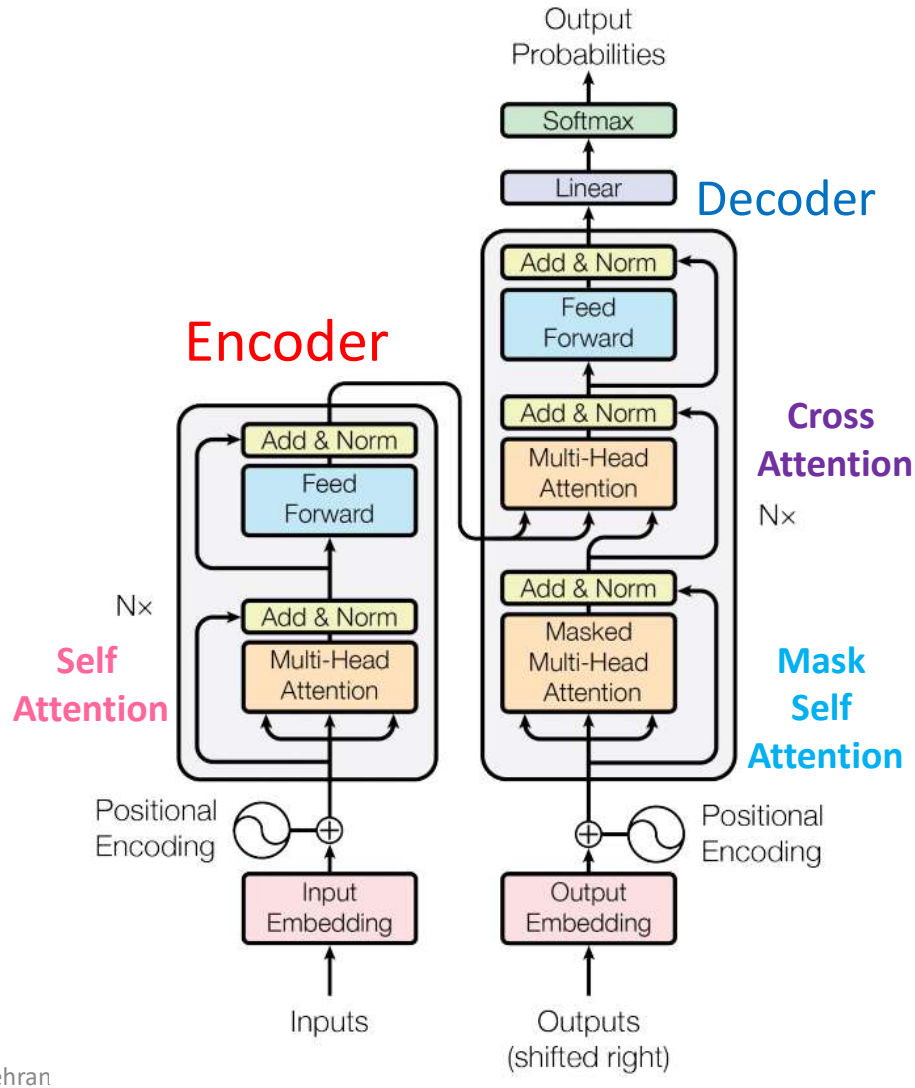


The vision transformer models are marked in red.

Vanilla Transformer

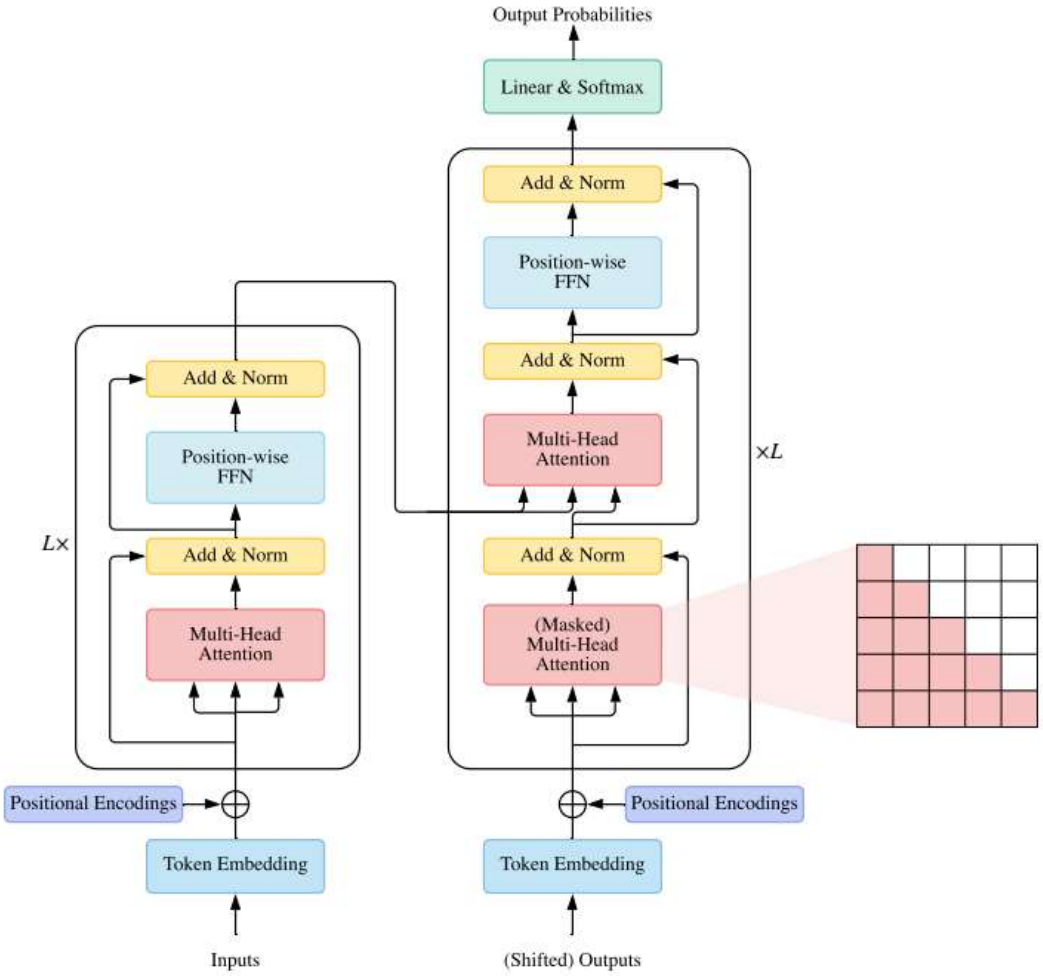
*Vaswani...2017 (Attention is all you need)

- a sequence-to-sequence model and consists of an encoder and a decoder, each of which is a stack of "N" identical blocks each *encoder block* is mainly composed of a multi-head self-attention module and a position-wise feed-forward network (FFN). In this work, the encoder is composed of a stack of N=6 identical layers.
- For building a deeper model, a residual connection is employed around each module, followed by Layer Normalization module.
- Compared to the encoder blocks, decoder blocks additionally insert cross-attention modules between the multi-head self-attention modules and the position-wise FFNs. In this work, the decoder is also composed of a stack of N=6 identical layers.
- Furthermore, the self-attention modules in the decoder are adapted to prevent each position from attending to subsequent positions.



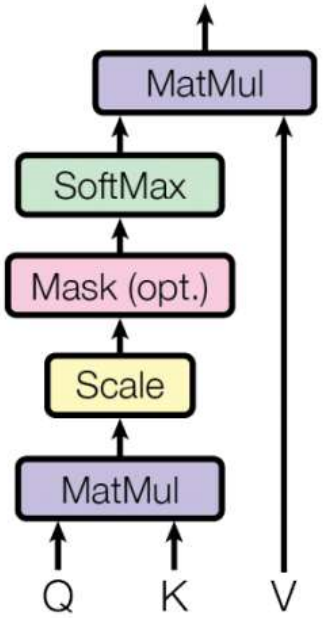
The key modules of the vanilla Transformer

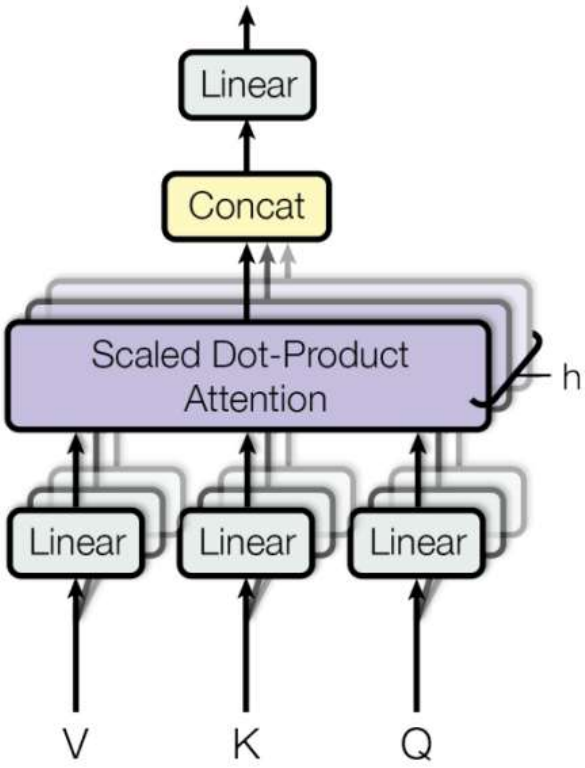
1. **Attention Modules**
   - Single Attention
   - Multi Head-Attention
     - Self-attention
     - Masked Self-attention
     - Cross-attention
2. **Position-wise FFN** (Feed Forward Network)
3. **Residual Connection and Normalization.**
4. **Position Encodings**.



Overview of vanilla Transformer architecture

In this work we employ h=8 parallel attention layers, or heads





Single-attention function Multi-head attention function.

1. Attention Modules

- **Single attention function**

Transformer adopts attention mechanism with Query-Key-Value (QKV) model. Given the packed matrix representations of queries , keys , and values , the scaled dot-product attention used by Transformer is given by

$$A = softmax(\frac{QK^T}{\sqrt{D_k}}) \text{ , } \mathrm{Q} \in R^{N \times D_k} \text{ , } \mathrm{K} \in R^{M \times D_k}, \mathrm{V} \in R^{M \times D_v}$$

N: length of Query, M: length of Keys(or values)- D_k: Dimension of query and key D_v: Dimension of values

- Softmax is applied in a row-wise manner.
- The dot-products of queries and keys are divided by $\sqrt{D_k}$ to alleviate gradient vanishing problem of the softmax function.

- **Multi head attention function**

Instead of simply applying a single attention function, Transformer uses multi-head attention, where the original queries, keys and values are with H different sets of learned projections.

$$MultiHeadAtn(Q, K, V) = Concat(head_1, \cdots, head_H)W^o$$

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \; i = 1,2, \ldots, H$$

The model then concatenates all the outputs.

Three types of attention (in terms of the source of queries and key-value pairs).

- *Self-attention*

  In Transformer encoder, we set Q = K = V = X, where X is the outputs of the previous layer.

- *Masked Self-attention*

  In the Transformer decoder, the self-attention is restricted such that queries at each position can only attend to all key-value pairs up to and including that position.

  To enable parallel training, this is typically done by applying a mask function to the un-normalized attention matrix $\hat{A} = \exp(\frac{QK^T}{\sqrt{D_k}})$ where the illegal positions are masked out by setting $\hat{A}_{ij} = -\infty \; if \; i < j$ This kind of self-attention is often referred to as autoregressive or causal attention

- *Cross-attention*

  The queries are projected from the outputs of the previous (decoder) layer, whereas the keys and values are projected using the outputs of the encoder.

2. Position-wise FFN.

The position-wise FFN is a fully connected feed-forward module that operates separately and identically on each position

$$\mathrm{FFN}(\mathrm{H}') = \mathrm{ReLU}(\mathrm{H}'\mathrm{W}^1 + \mathrm{b}^1)\mathrm{W}^2 + \mathrm{b}^2$$

where H′ is the outputs of previous layer, and $W^1 \in R^{D_m \times D_f}$, $W^2 \in R^{D_f \times D_m}$, $b^1 \in R^{D_f}$, $b^2 \in R^{D_m}$ are trainable parameters. Typically the intermediate dimension D_f f the FFN is set to be larger than D_m

3. Residual Connection and Normalization

In order to build a deep model, Transformer employs a residual connection [49] around each module, followed by Layer Normalization [4]. For instance, each Transformer encoder block may be written as

$$\mathrm{H}' = \mathrm{LayerNorm}(\mathrm{SelfAtention}(\mathrm{X}) + \mathrm{X})$$

$$\mathrm{H} = \mathrm{LayerNorm}(\mathrm{FFN}(\mathrm{H}') + \mathrm{H}')$$

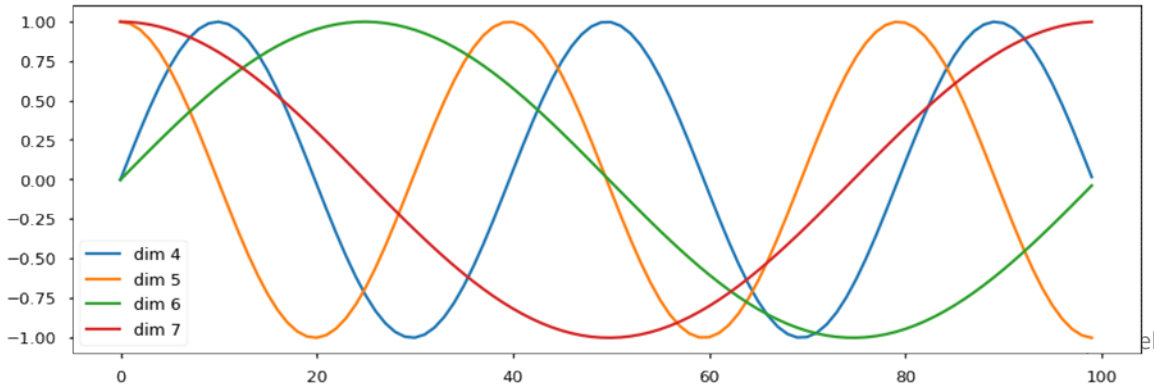
where SelfAtention(·) denotes self attention module and LayerNorm(·) denotes the layer normalization operation

4. Position Encodings.

Since Transformer doesn't introduce recurrence or convolution, it is ignorant of positional information (especially for the encoder). Thus additional positional representation is needed to model the ordering of tokens

Positional Encoding

- Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add “positional encodings” to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension dmodel
- as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed (cite).
- In this work, we use sine and cosine functions of different frequencies: PE(pos,2i)=sin(pos/100002i/dmodel)
- PE(pos,2i+1)=cos(pos/100002i/dmodel)
- where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to 10000·2π. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PEpos+k can be represented as a linear function of PEpos.
- In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of Pdrop=0.1
- .



ehran

Position-wise Feed-Forward Networks

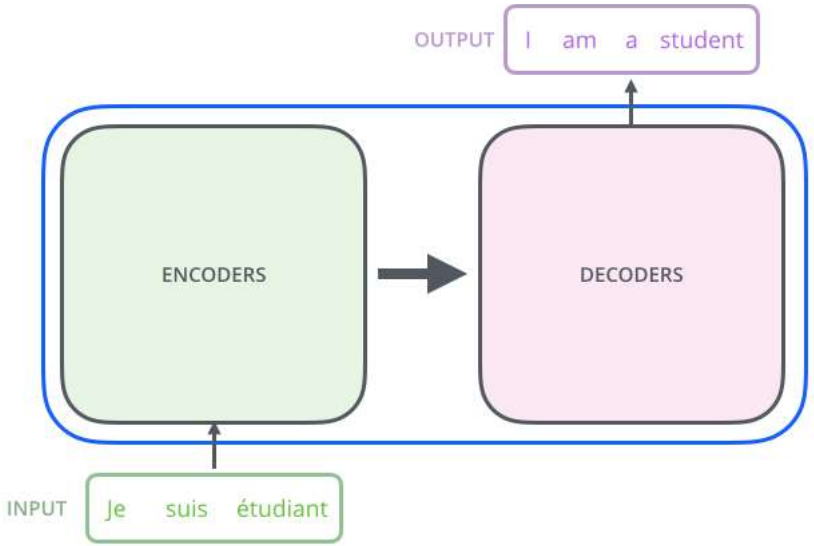
In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically.

This consists of two linear transformations with a ReLU activation in between.

$$FFN(x)=max(0,xW1+b1)W2+b2$$

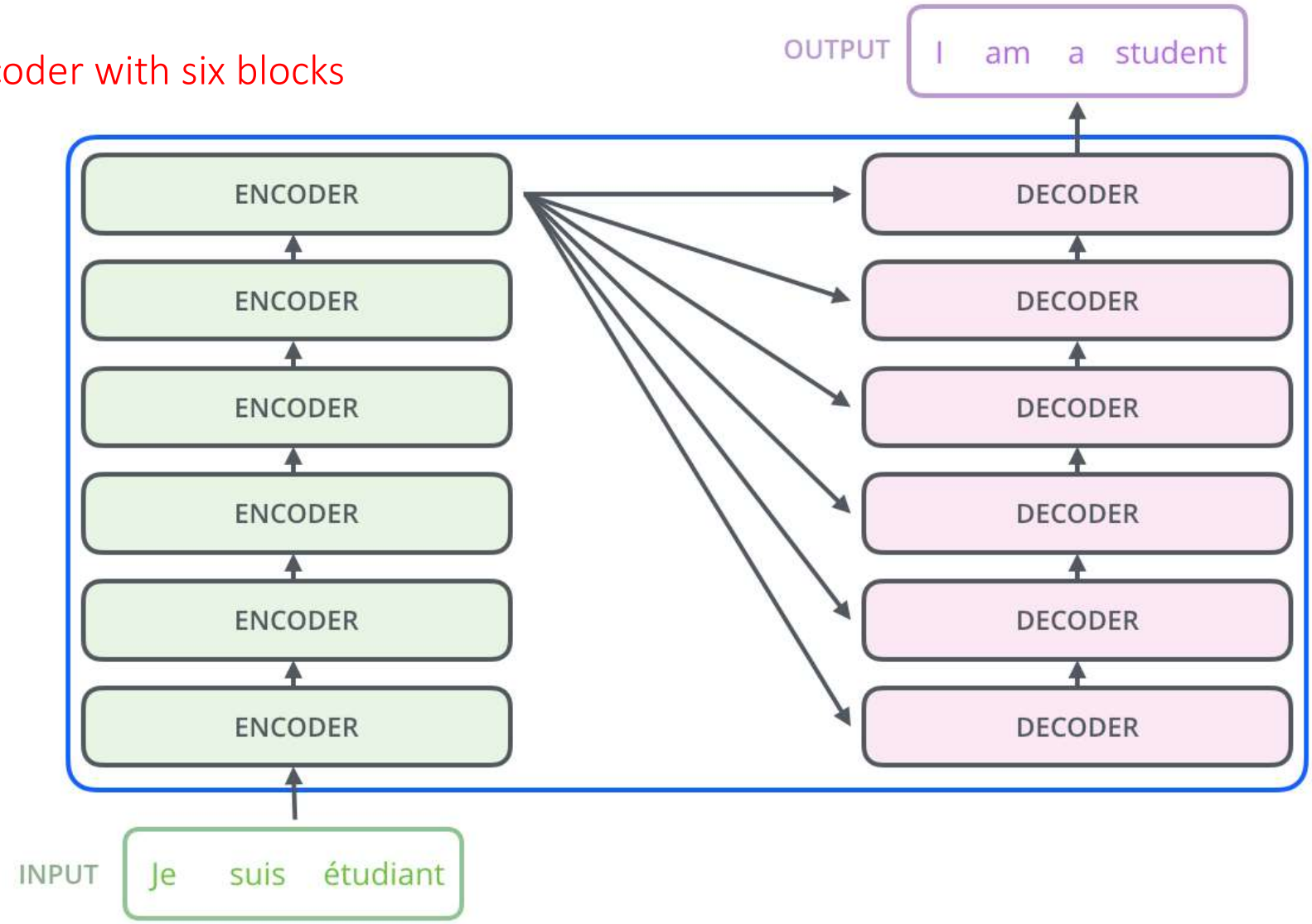
While the linear transformations are the same across different positions, they use different parameters from layer to layer.

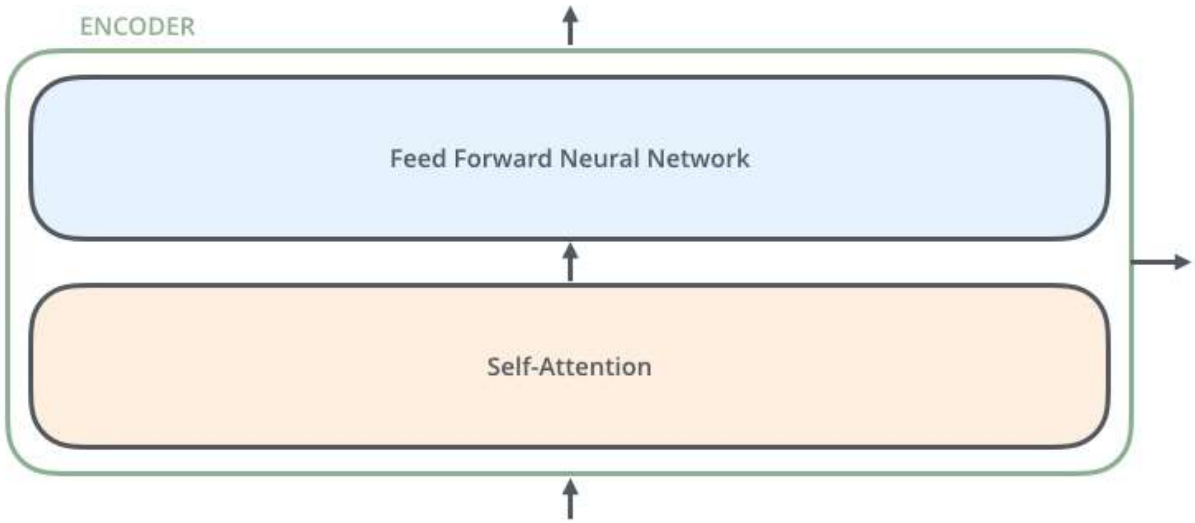
Another way of describing this is as two convolutions with kernel size 1.
The dimensionality of input and output is dmodel=512 , and the inner-layer has dimensionality dff=2048.



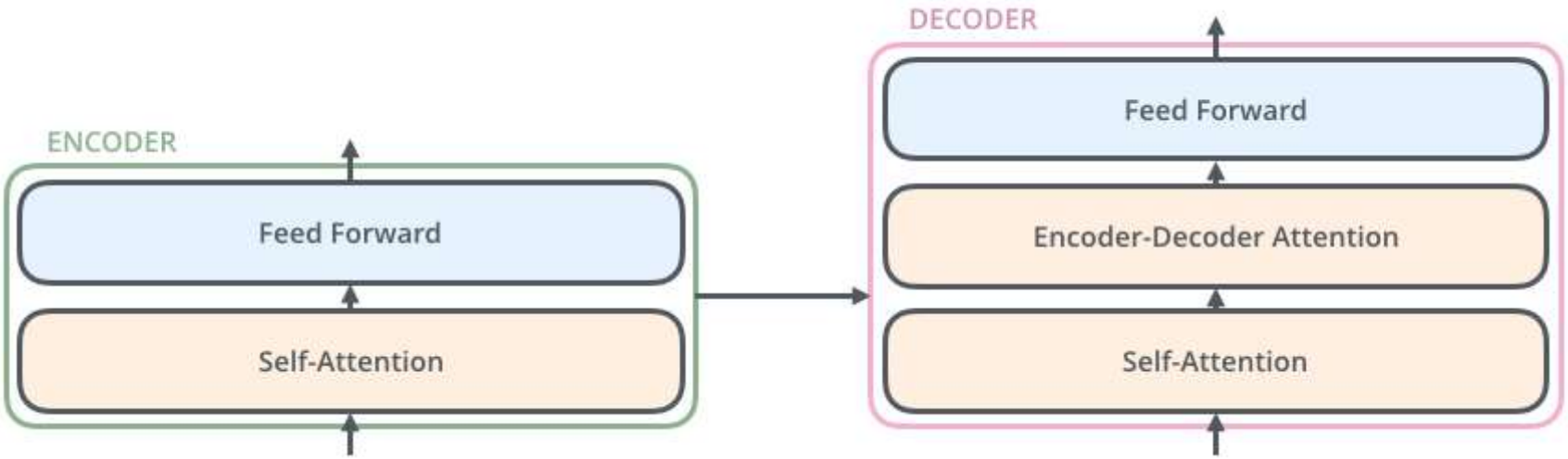
Jay Alammar
Visualizing machine learning one concept at a time.
@JayAlammar on Twitter. YouTube Channel

Encoder and Decoder with six blocks

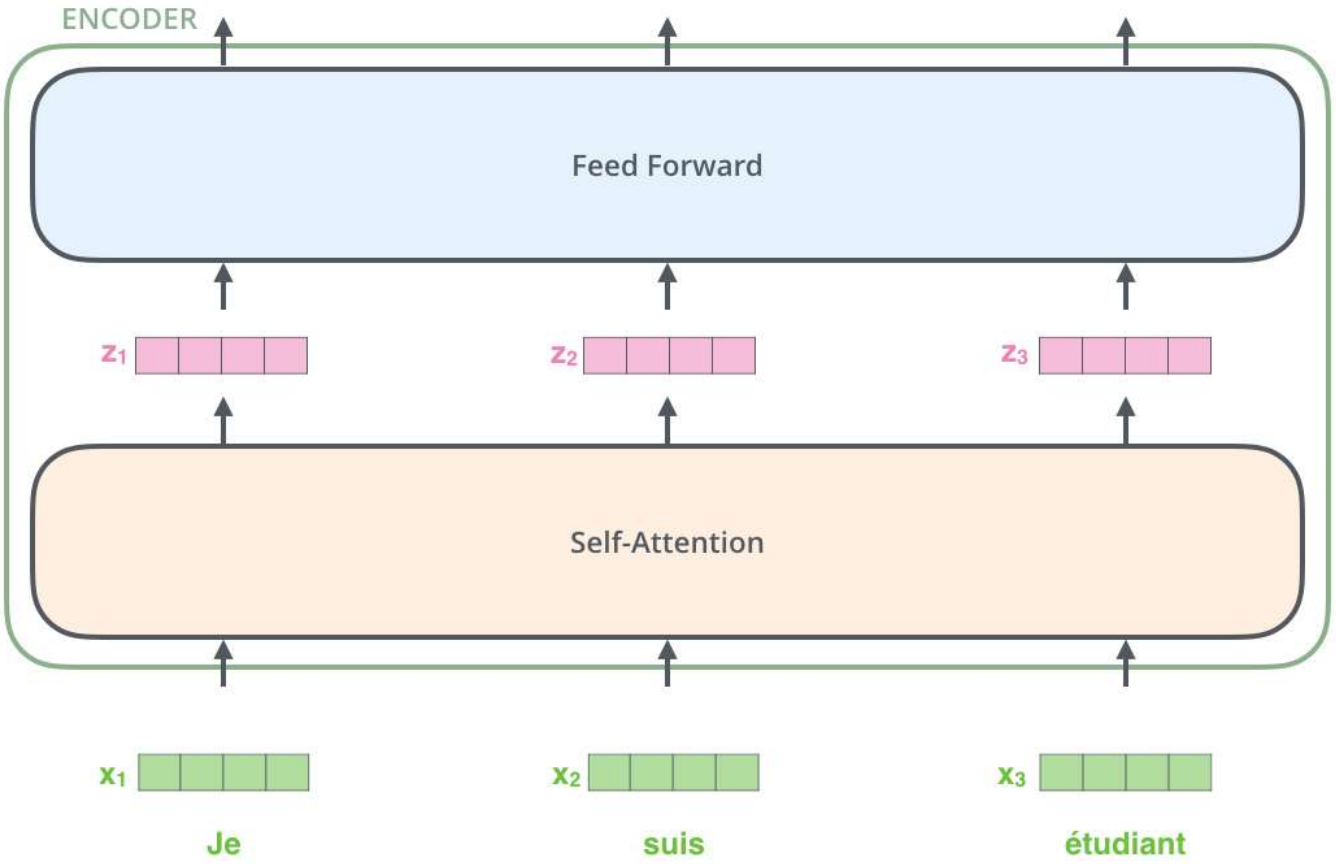



ENCODER
Feed Forward Neural Network
Self-Attention

Encoder and Decoder with one block

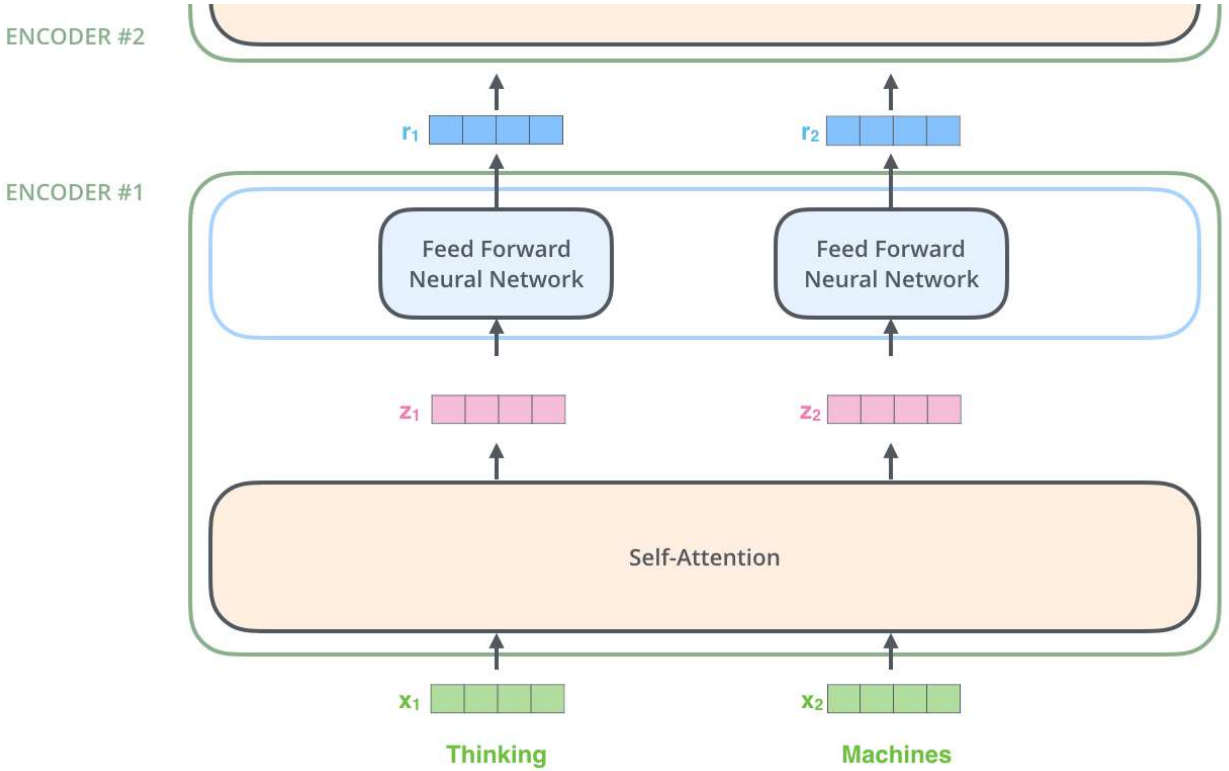


Bringing The Tensors Into The Picture



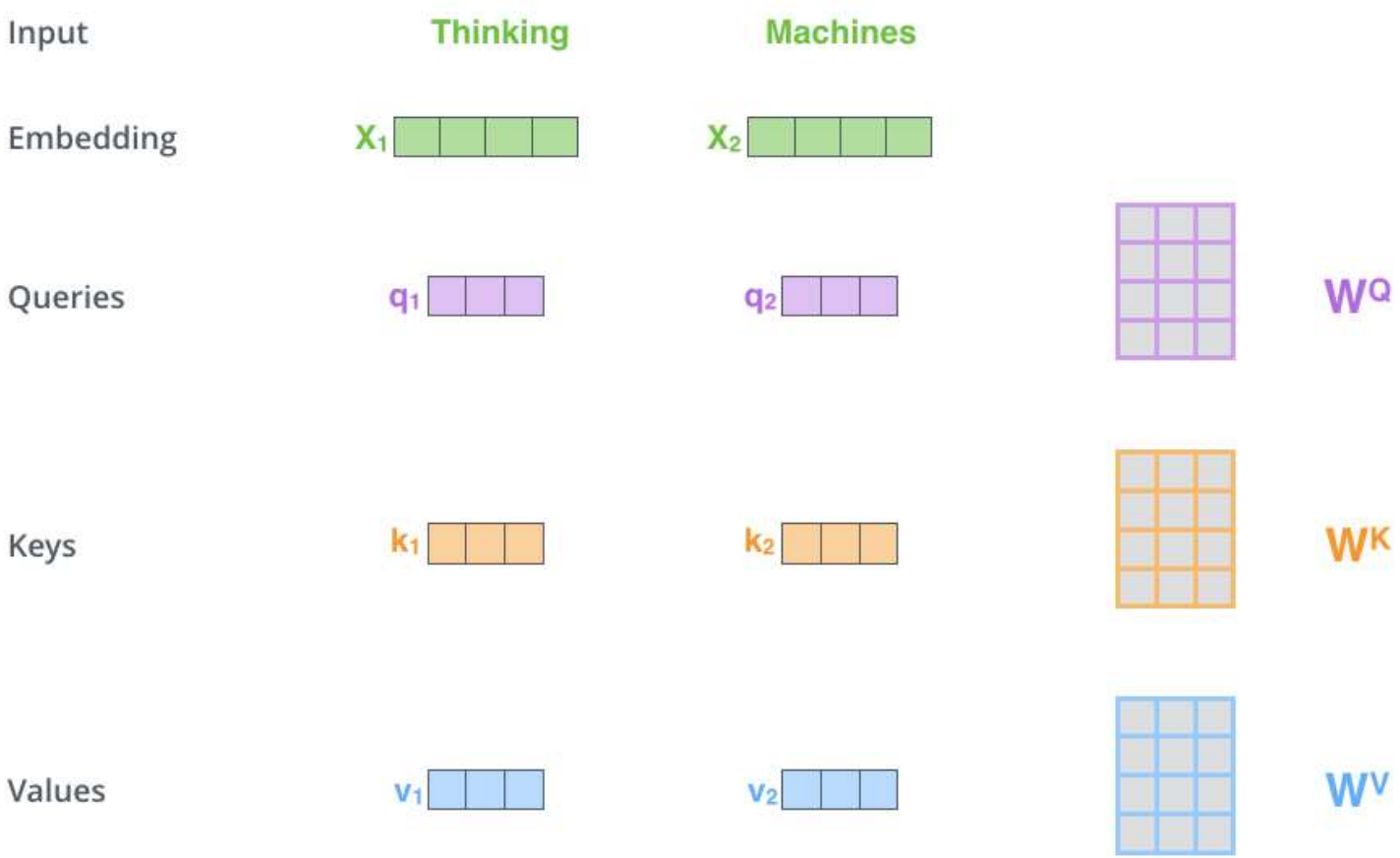
Now We're Encoding!

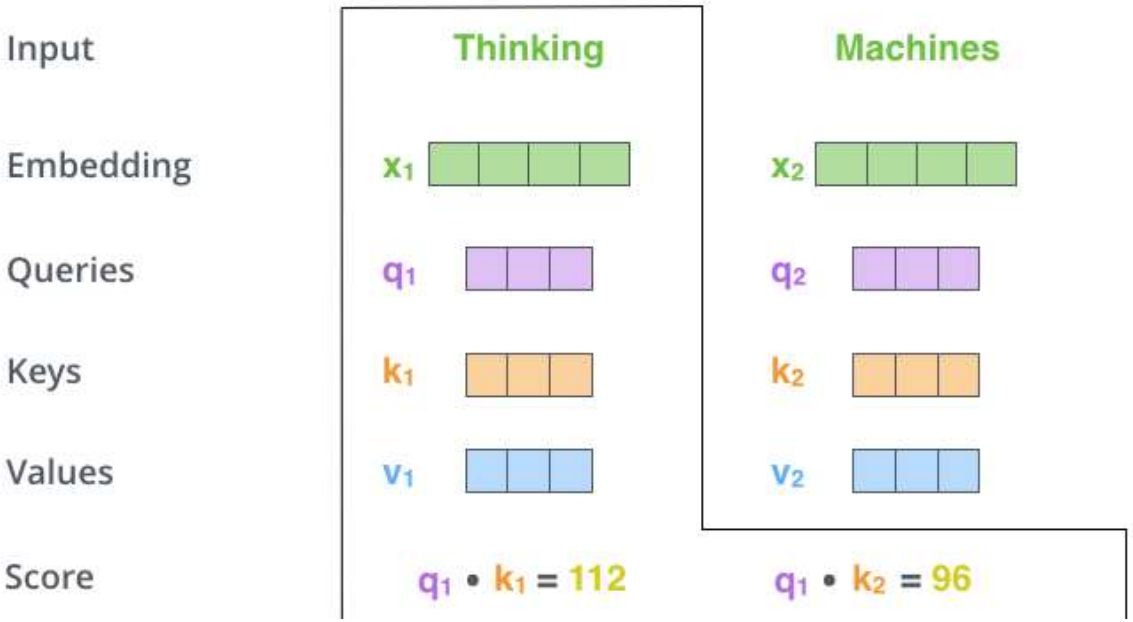
The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

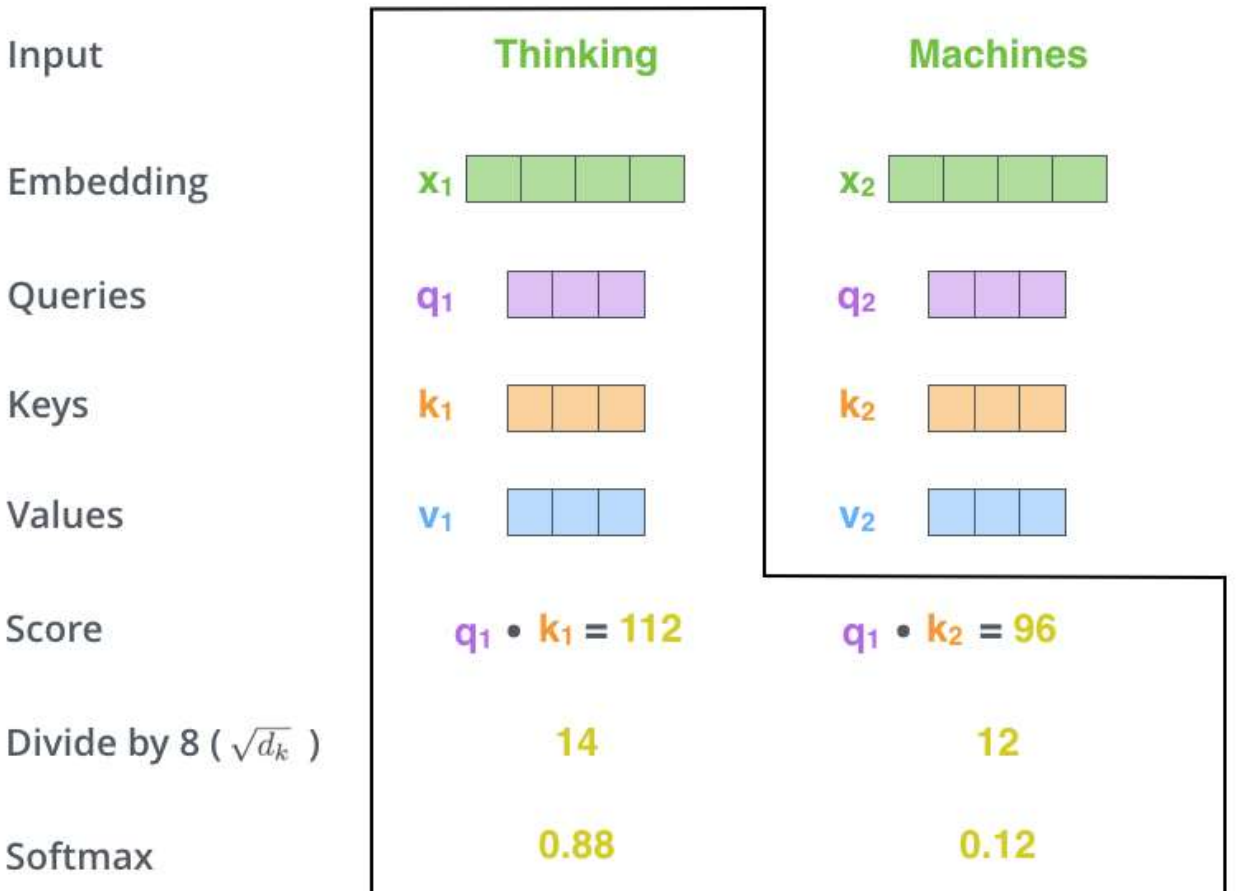


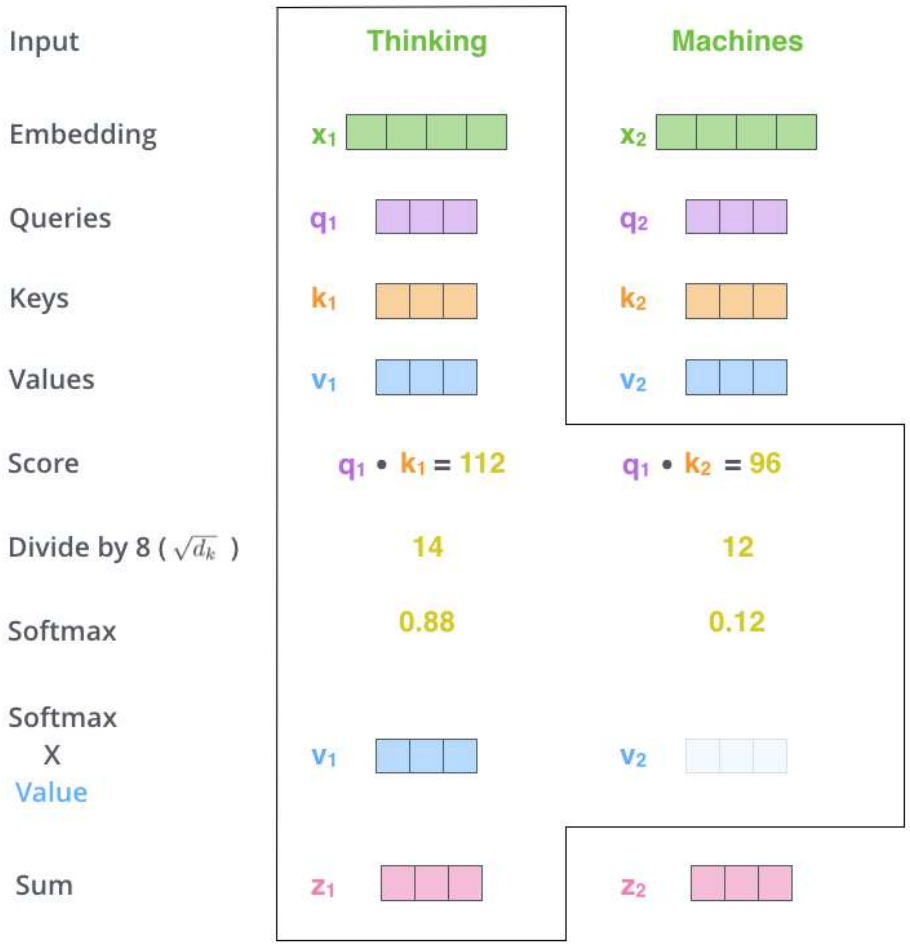
Self-Attention in Detail

Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.



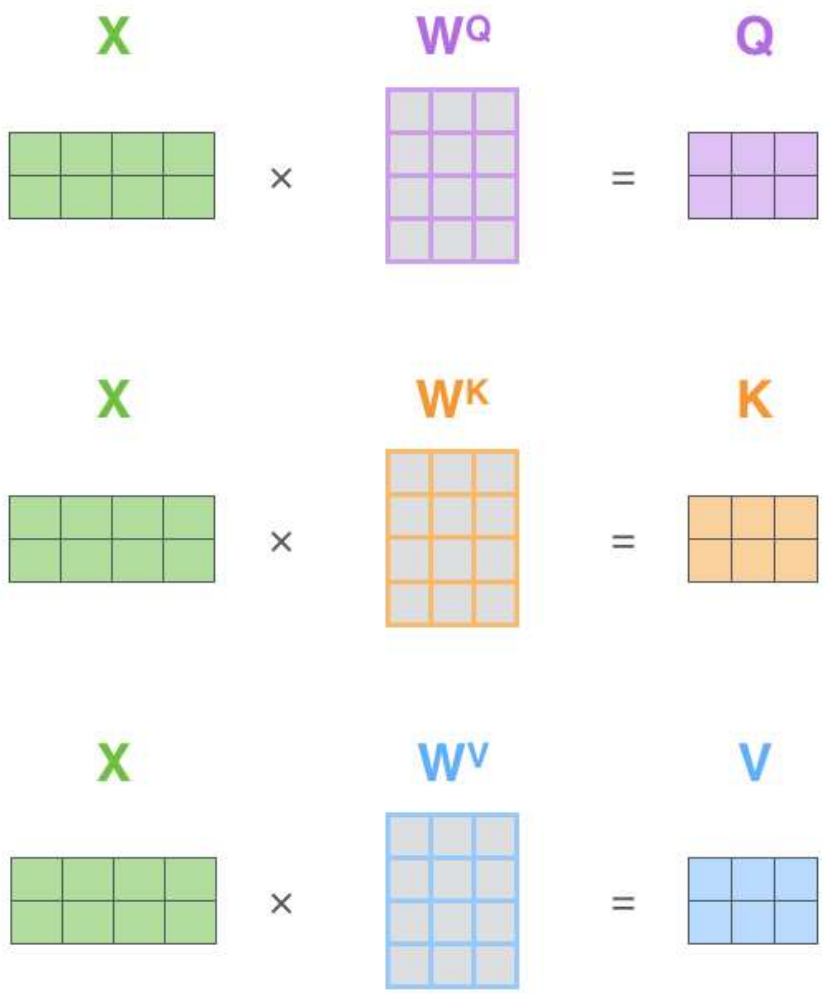

Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96


Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 (√dk)
14
12
Softmax
0.88
0.12

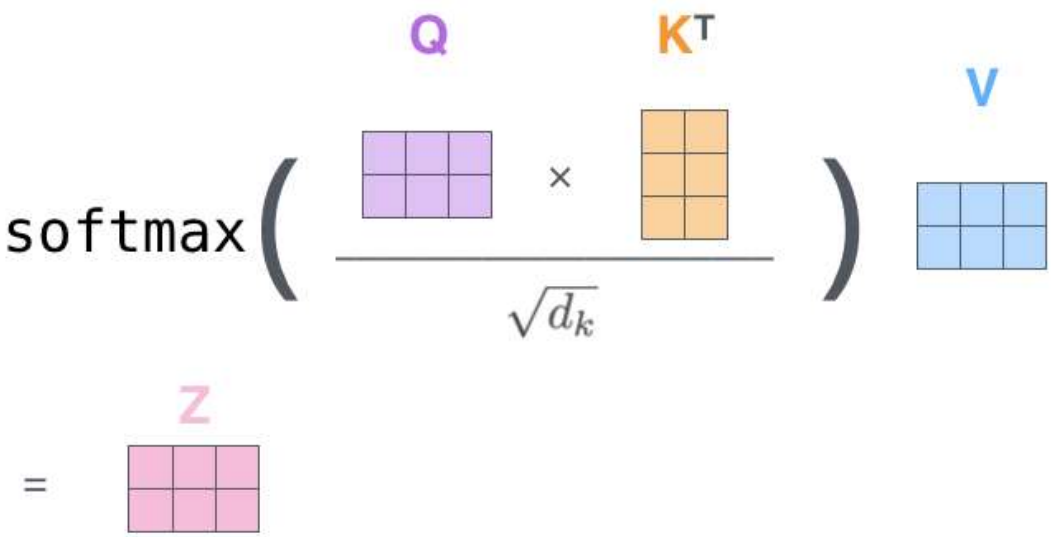

Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 (√dk)
14
12
Softmax
0.88
0.12
Softmax X Value
v1
v2
Sum
z1
z2

Matrix Calculation of Self-Attention

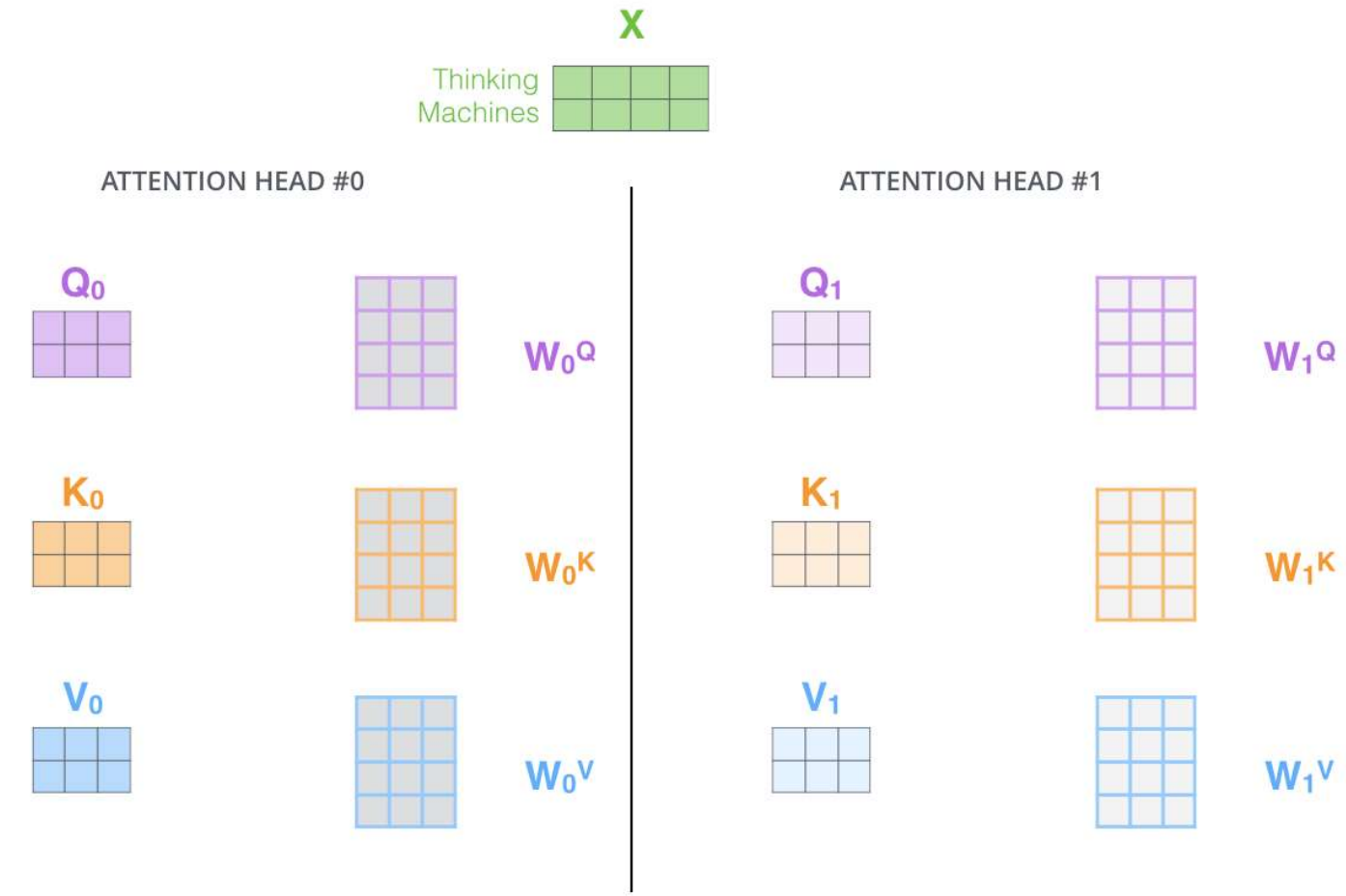
Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

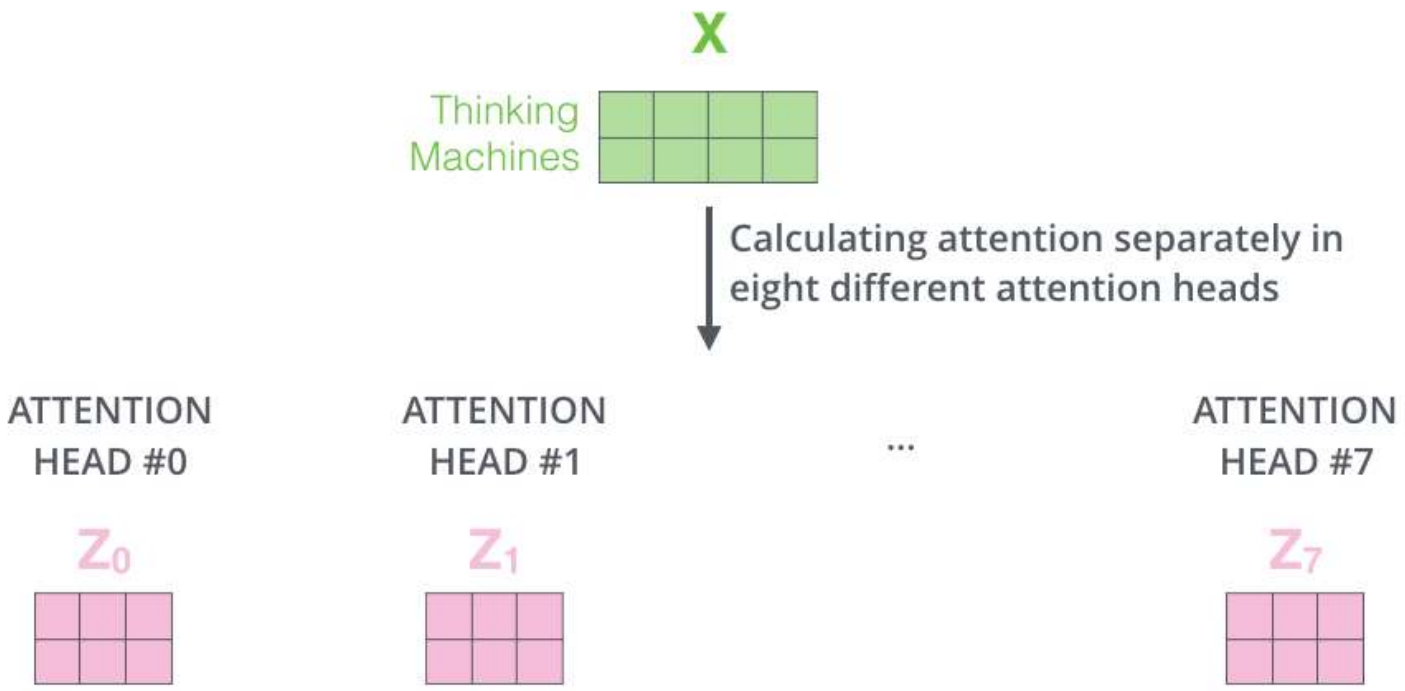


The self-attention calculation in matrix form



The Beast With Many Heads

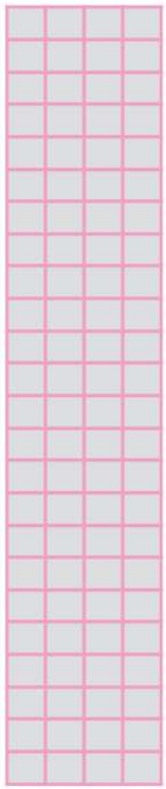



X
Thinking
Machines
Calculating attention separately in
eight different attention heads
ATTENTION
HEAD #0
ATTENTION
HEAD #1
...
ATTENTION
HEAD #7
Z0
Z1
Z7

1) Concatenate all the attention heads

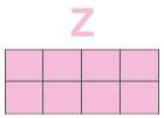
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



1) This is our input sentence*

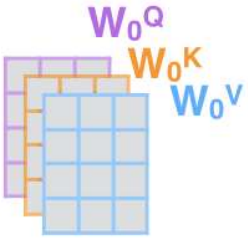
2) We embed each word*

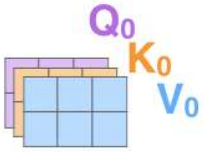
3) Split into 8 heads. We multiply X or R with weight matrices

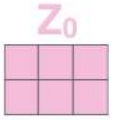
4) Calculate attention using the resulting Q/K/V matrices

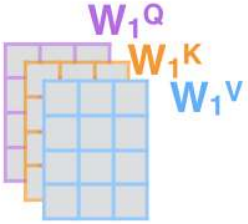
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

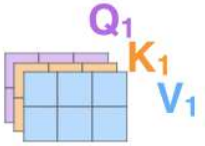
Thinking Machines

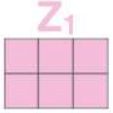


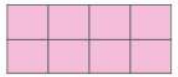




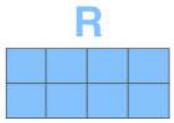


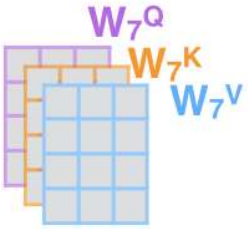


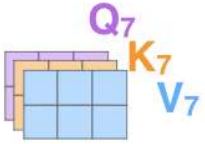


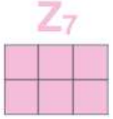


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



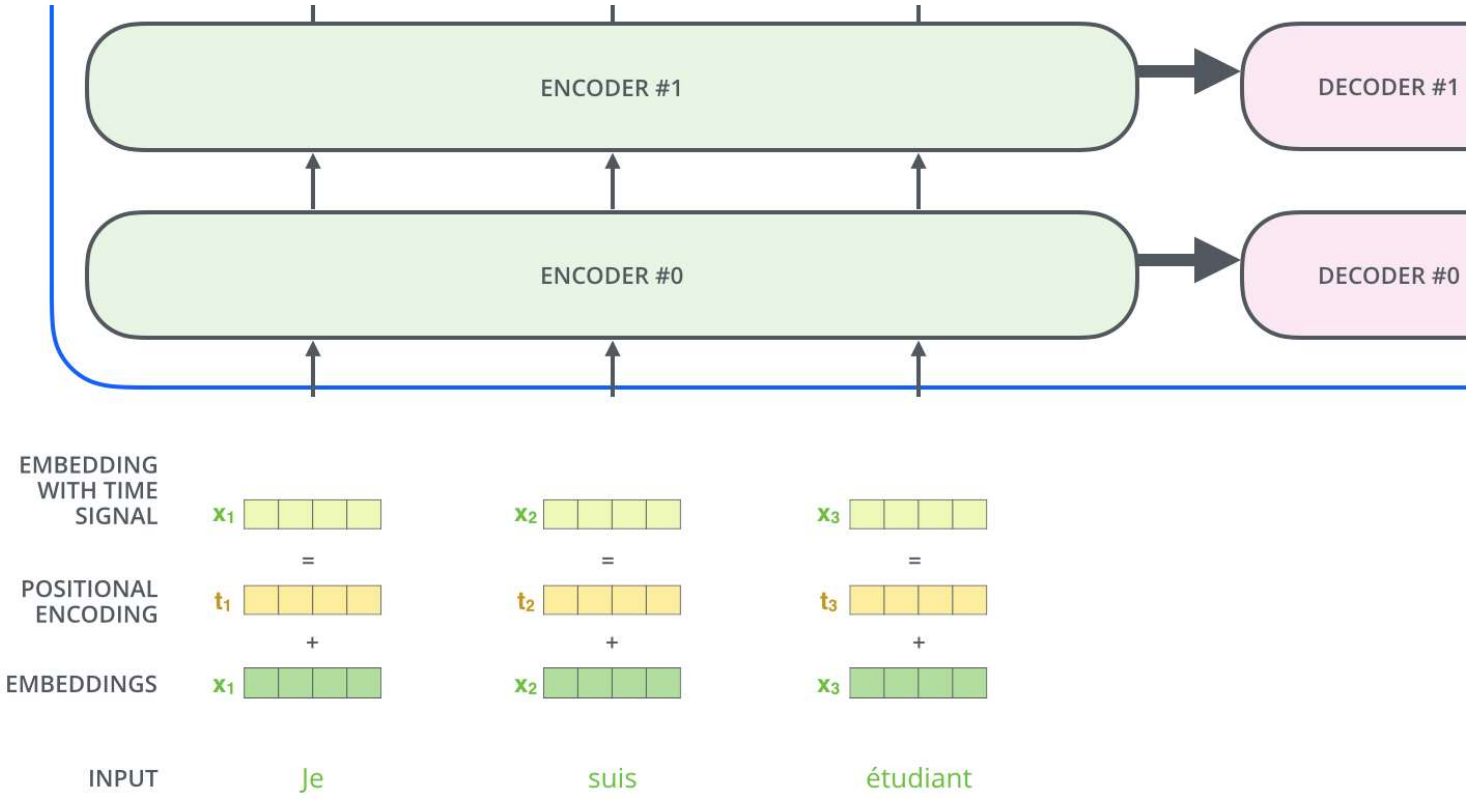


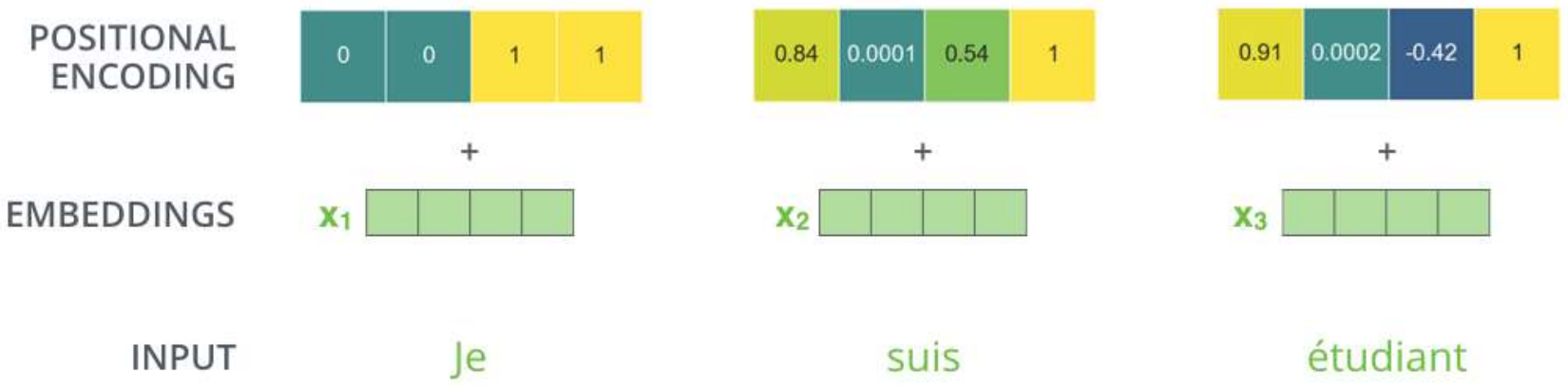




Representing The Order of The Sequence Using Positional Encoding

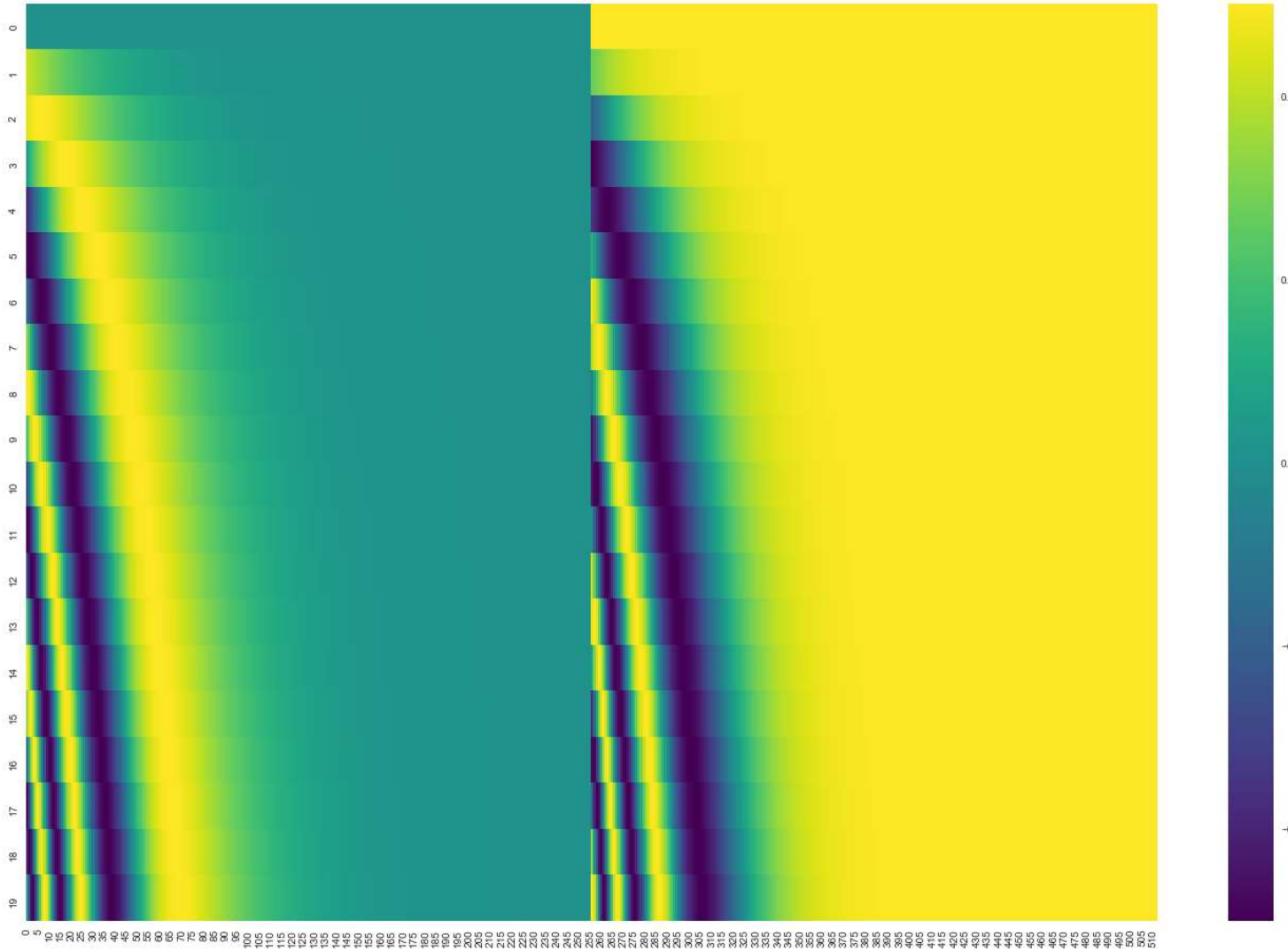
To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.



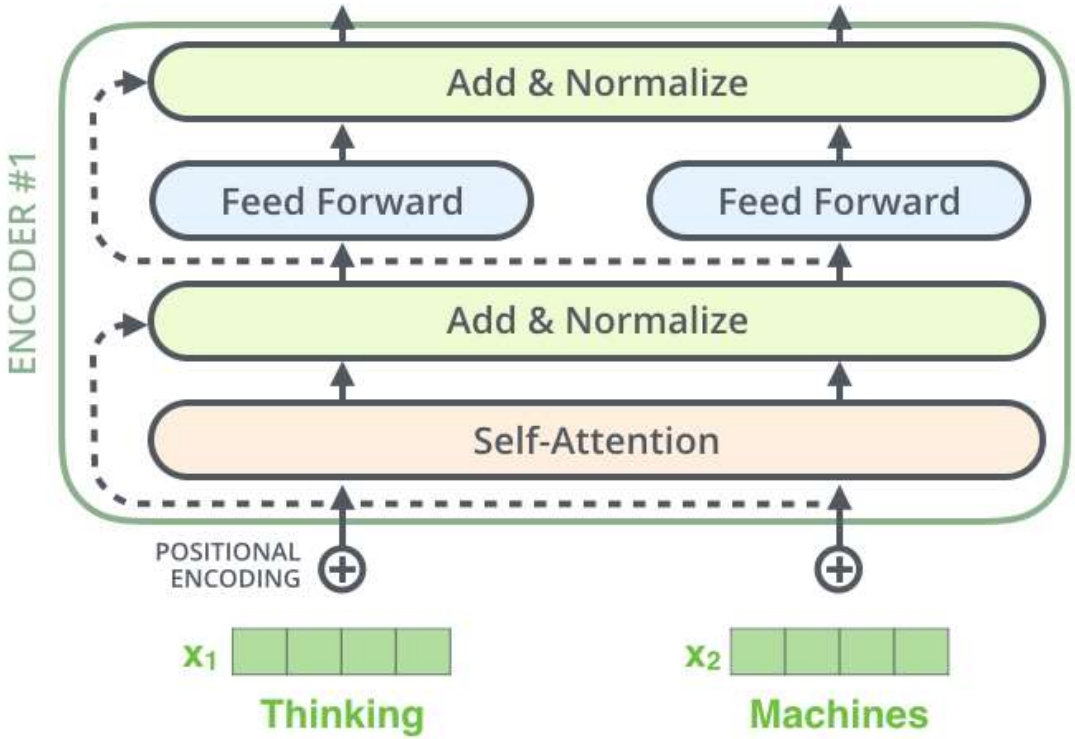


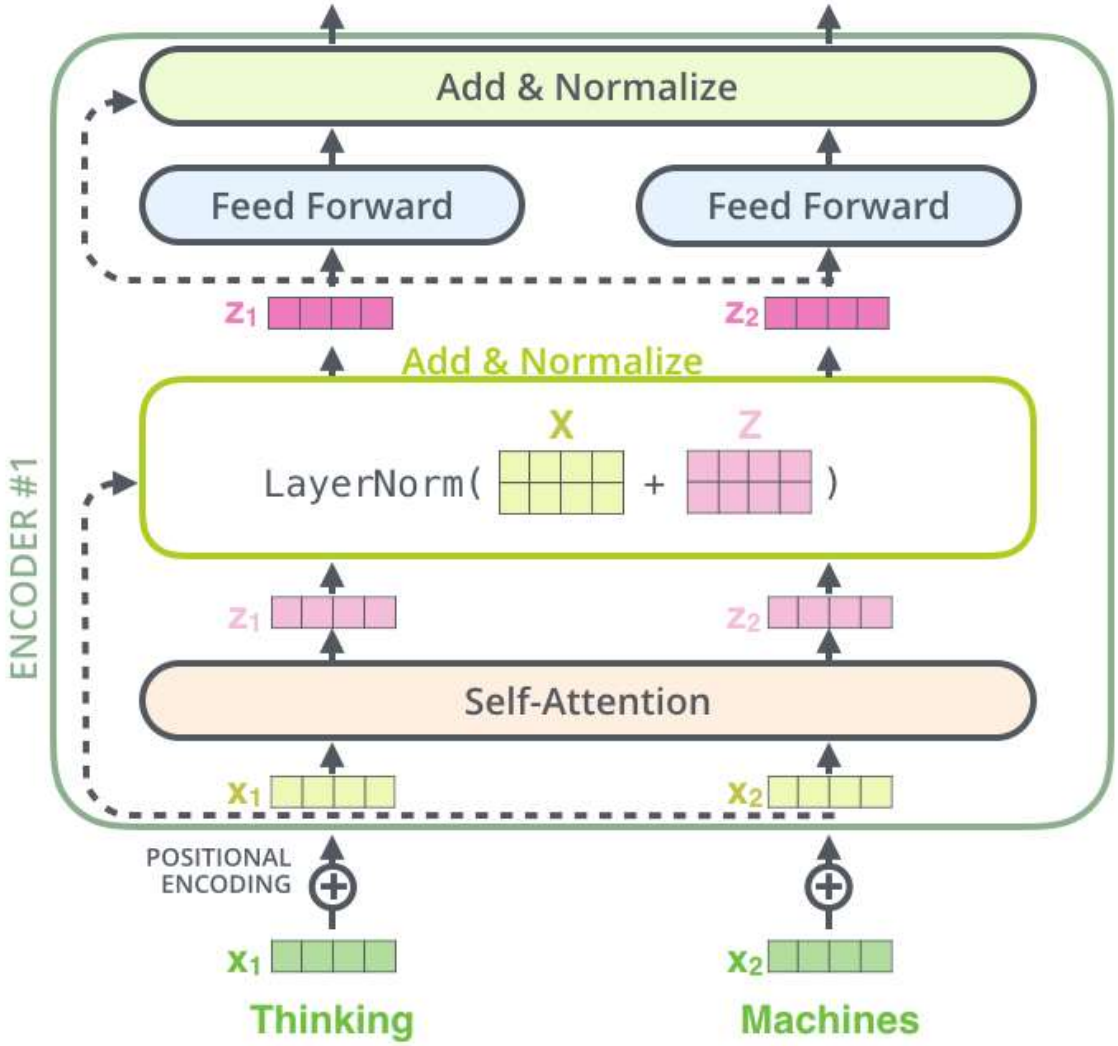
A real example of positional encoding with a toy embedding size of 4

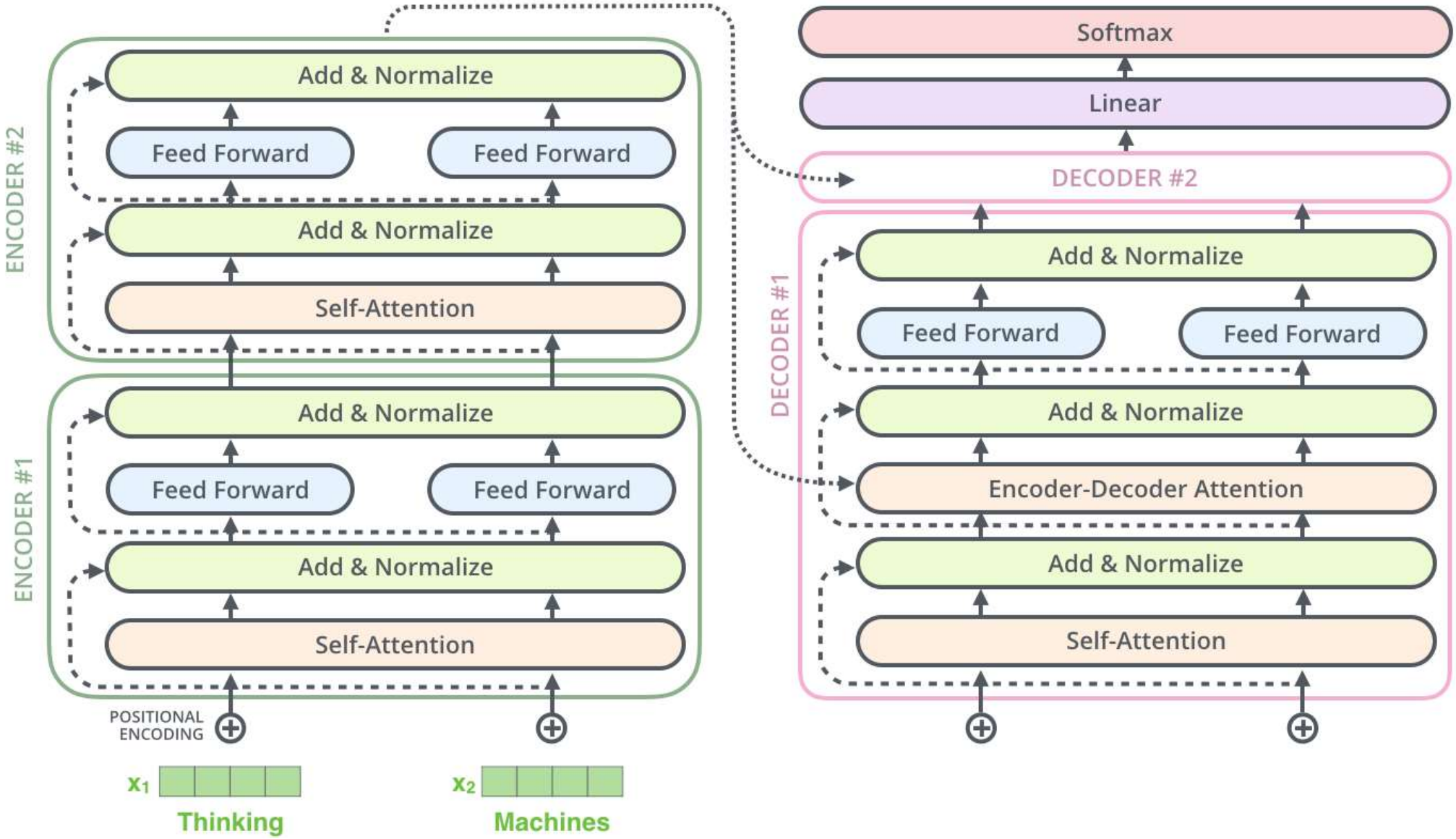
A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.



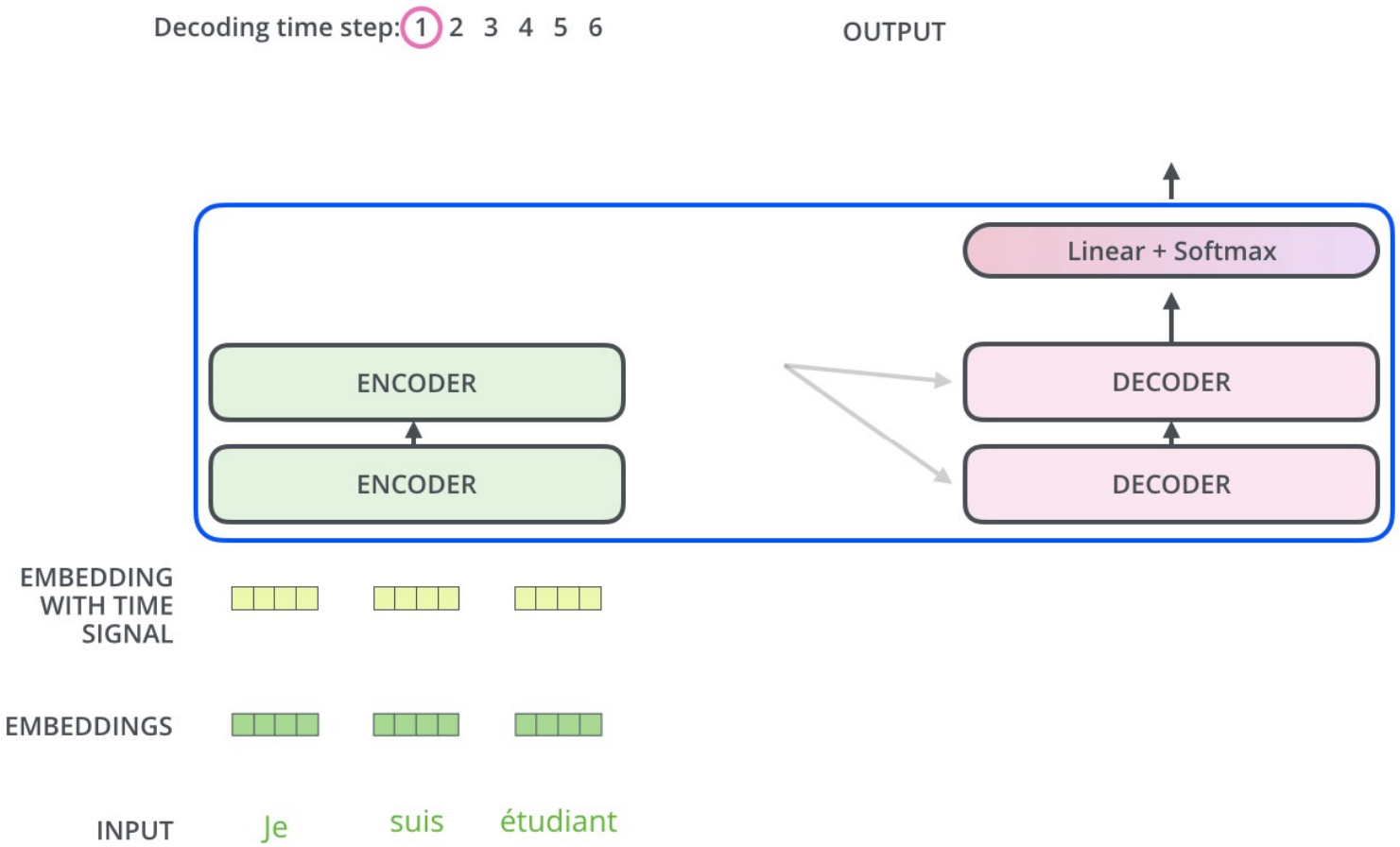
The Residuals

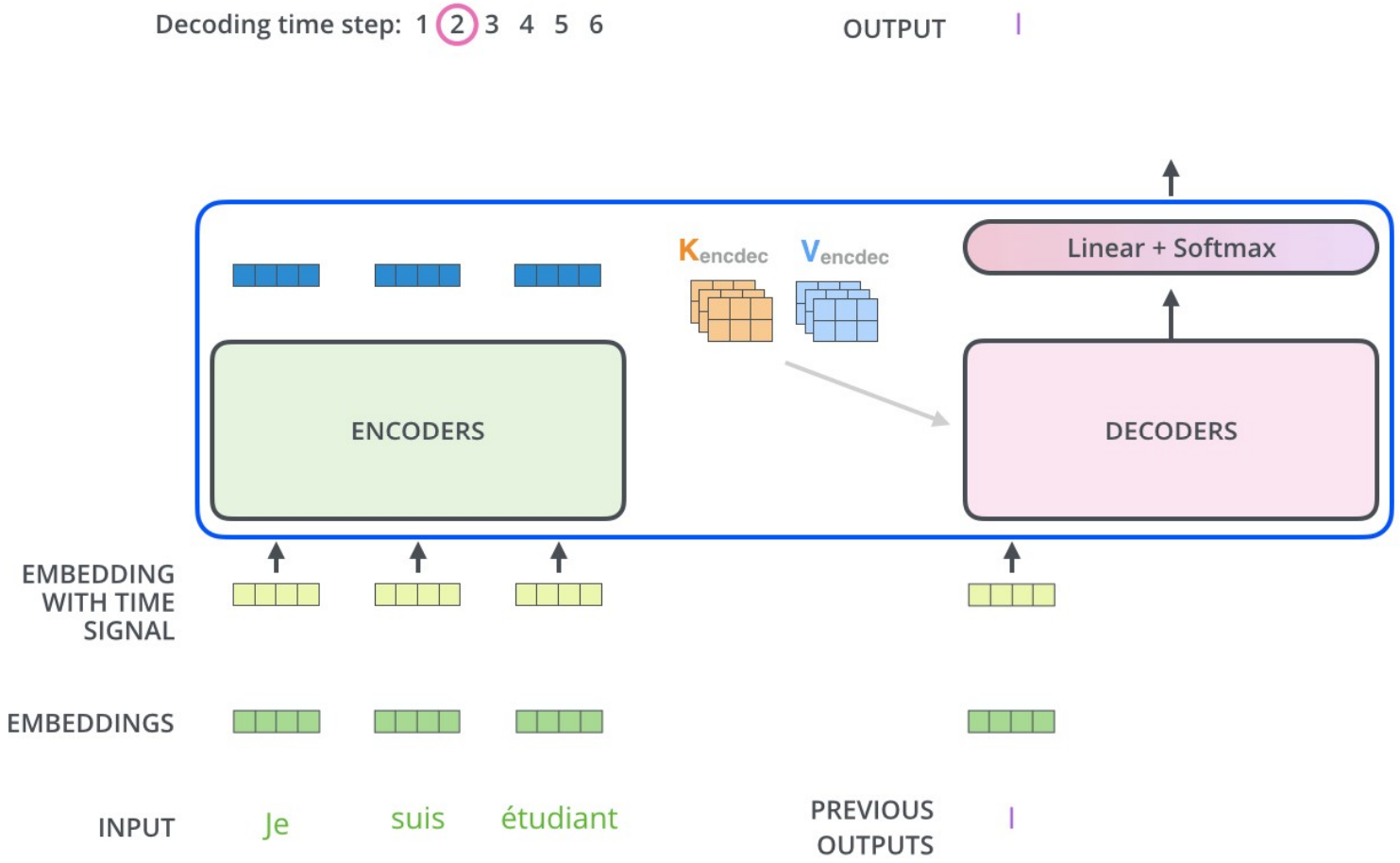



Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
ENCODER #1
z1
z2
Self-Attention
x1
x2
POSITIONAL ENCODING
x1
x2
Thinking
Machines

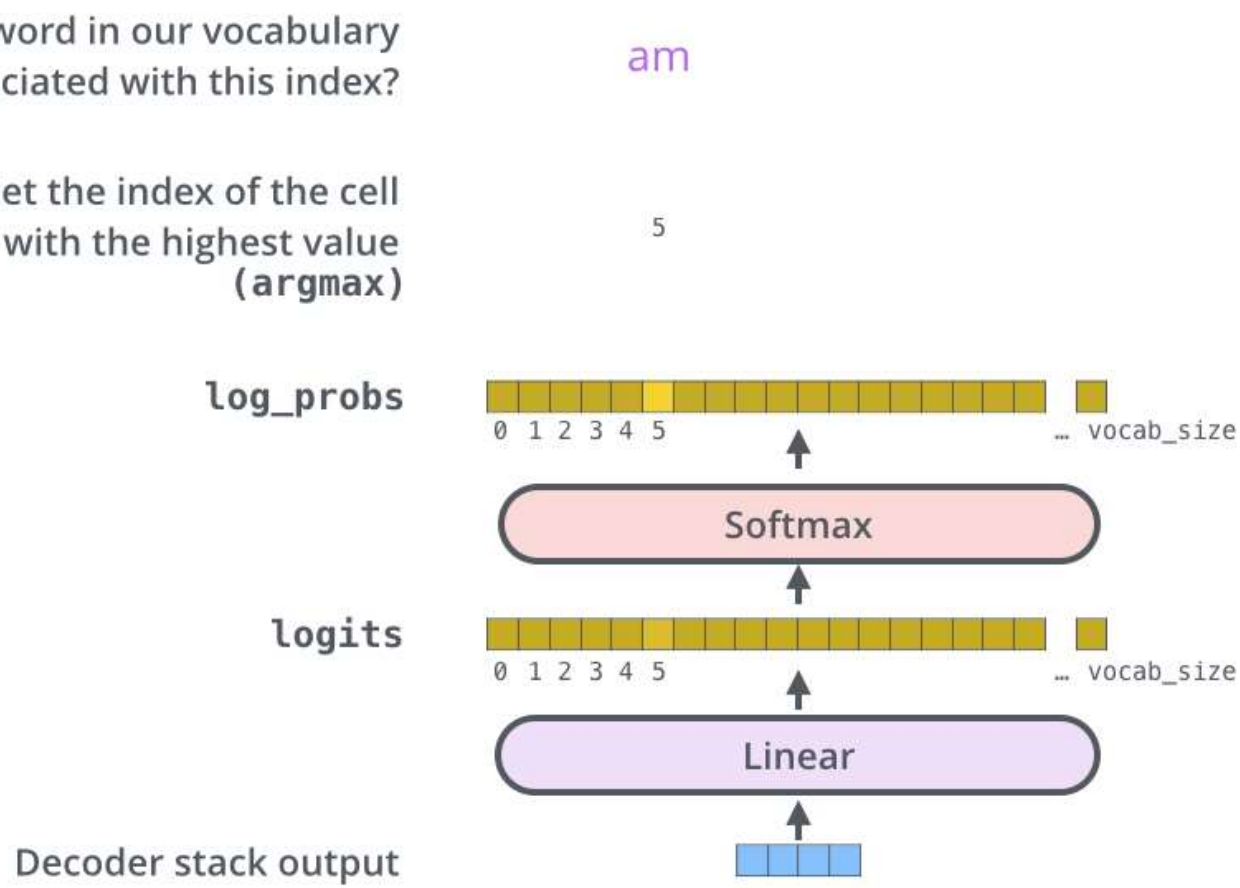

Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
x2
Thinking
Machines

After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).

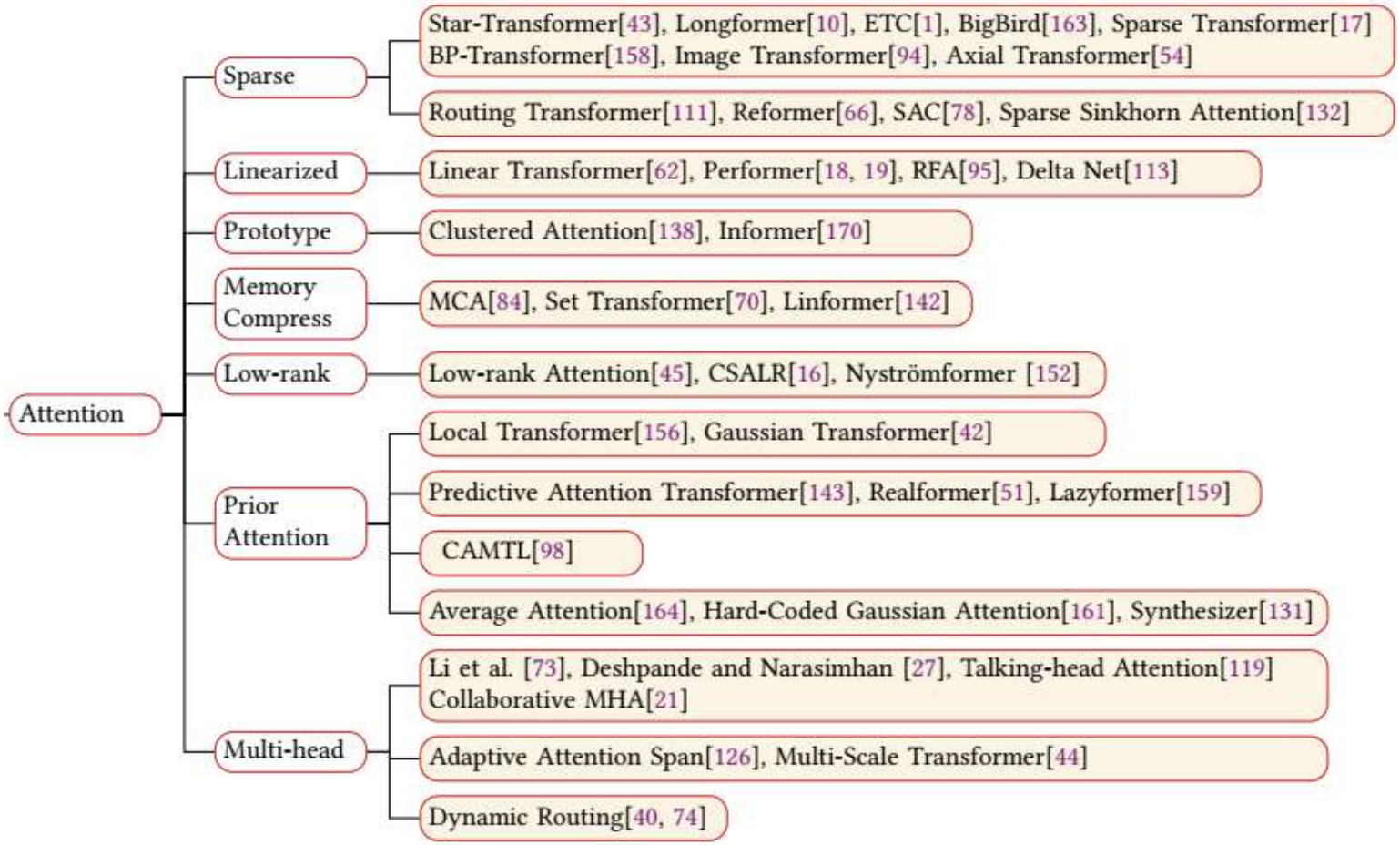



Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
K encdec
V encdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

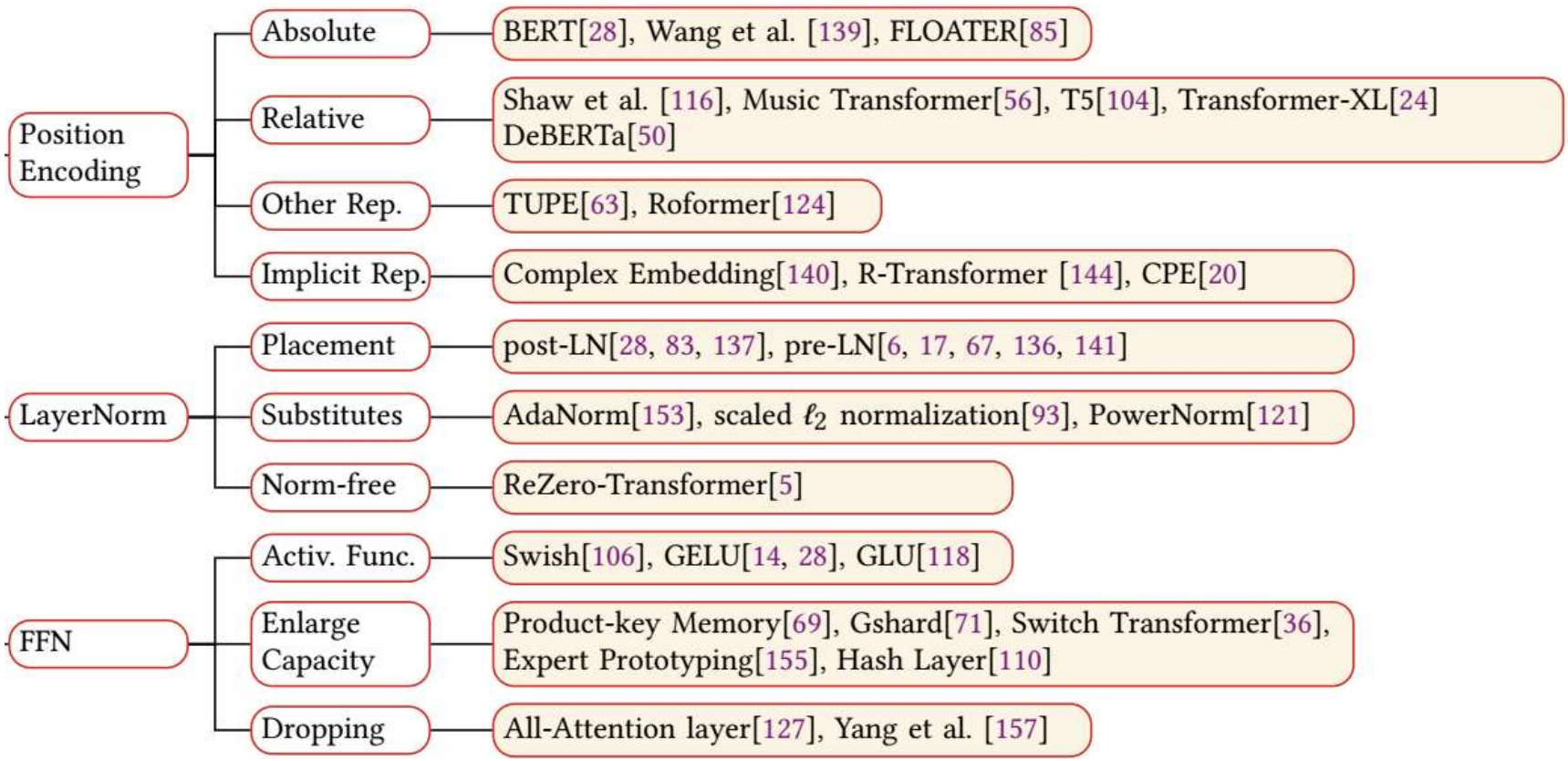
The Final Linear and Softmax Layer



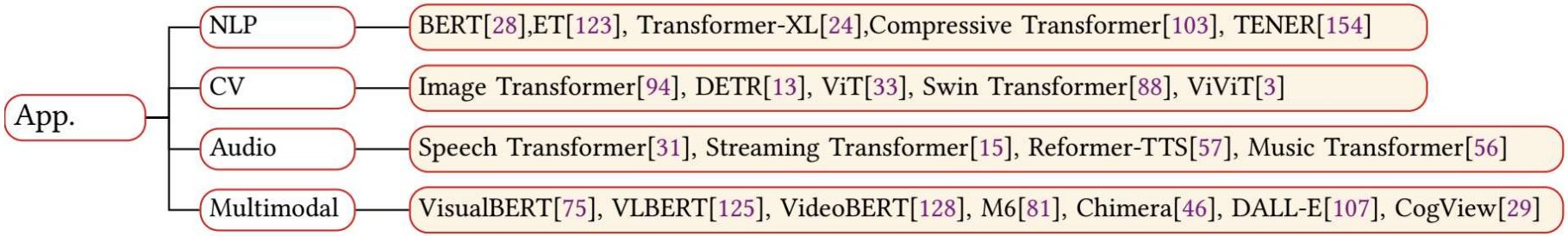
Taxonomy of Transformers in Module Level (Attention)



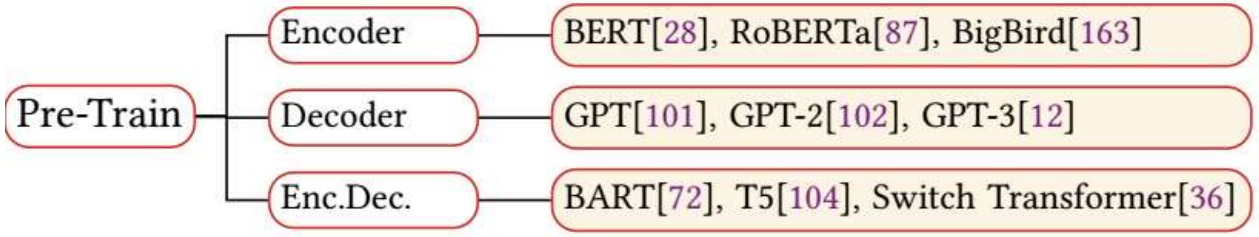
Taxonomy of Transformers in Module Level (Position-Layer Norm-FFN)



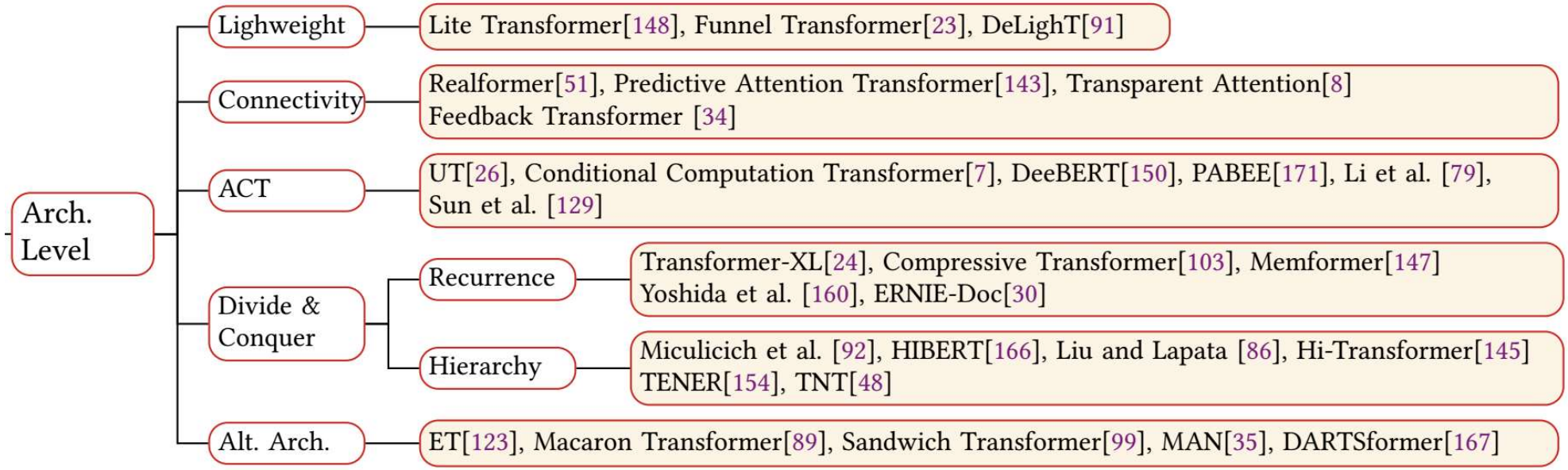
Taxonomy of Transformers in App



Taxonomy of Transformers in Pre-Train



Taxonomy of Transformers in Arch. Level



ACT: Adaptive Computation Time

Model Usage

Generally, the Transformer architecture can be used in three different ways:

- *Encoder-Decoder*. The full Transformer architecture as introduced is used. This is typically used in sequence-to-sequence modeling (e.g., neural machine translation).

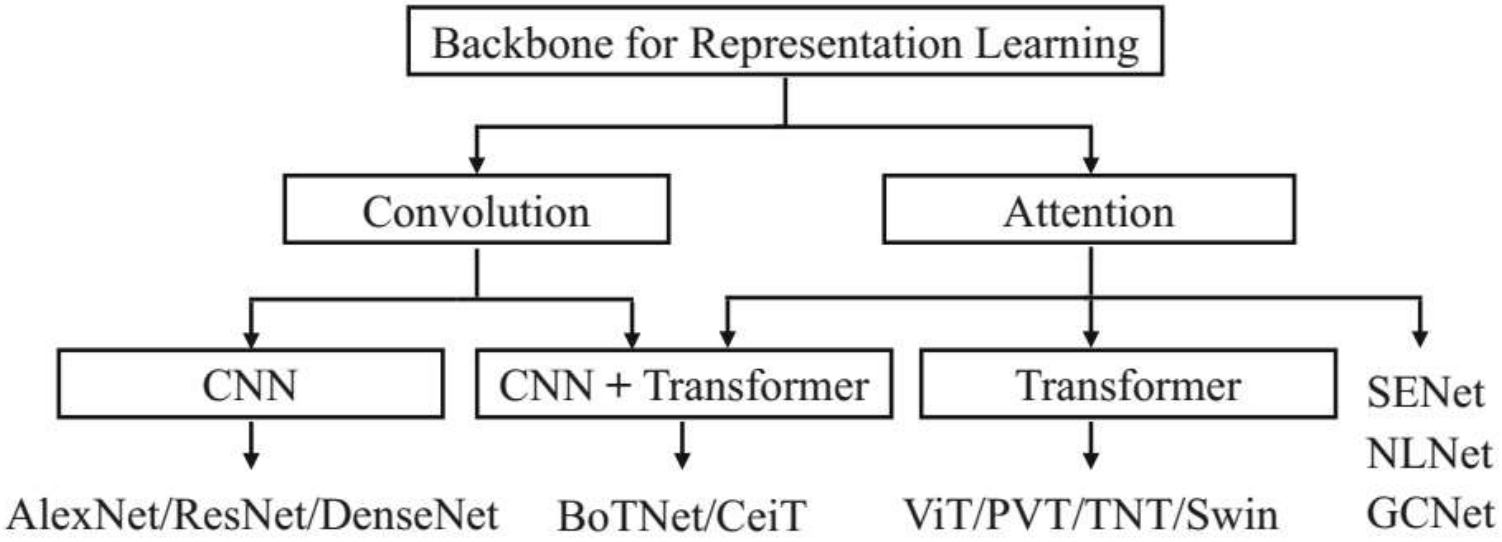
- *Encoder only*. Only the encoder is used and the outputs of the encoder are utilized as are presentation for the input sequence. This is usually used for classification or sequence labeling problems.

- *Decoder only*. Only the decoder is used, where the encoder-decoder cross-attention module is also removed. This is typically used for sequence generation, such as language modeling.

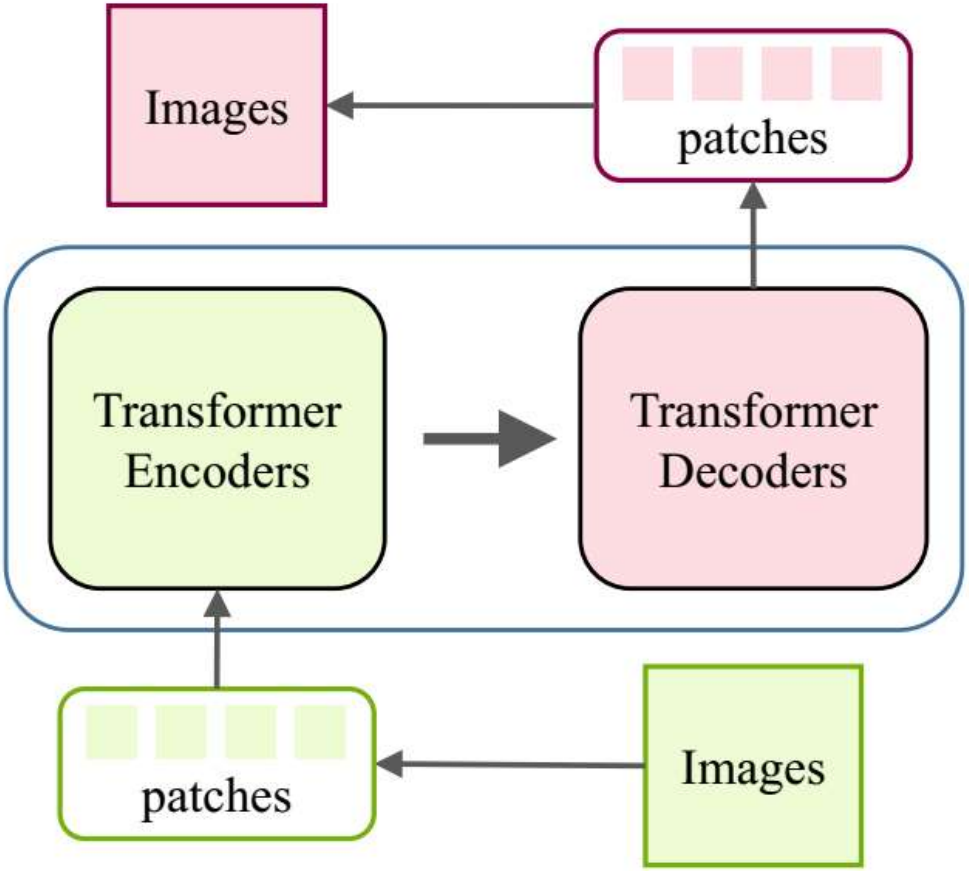
Vision Transformer (ViT)

The applications of transformer based models in computer vision, including image classification, high/mid-level vision, low-level vision and video processing.

Convolution and Attention in Computer Vision Problems:



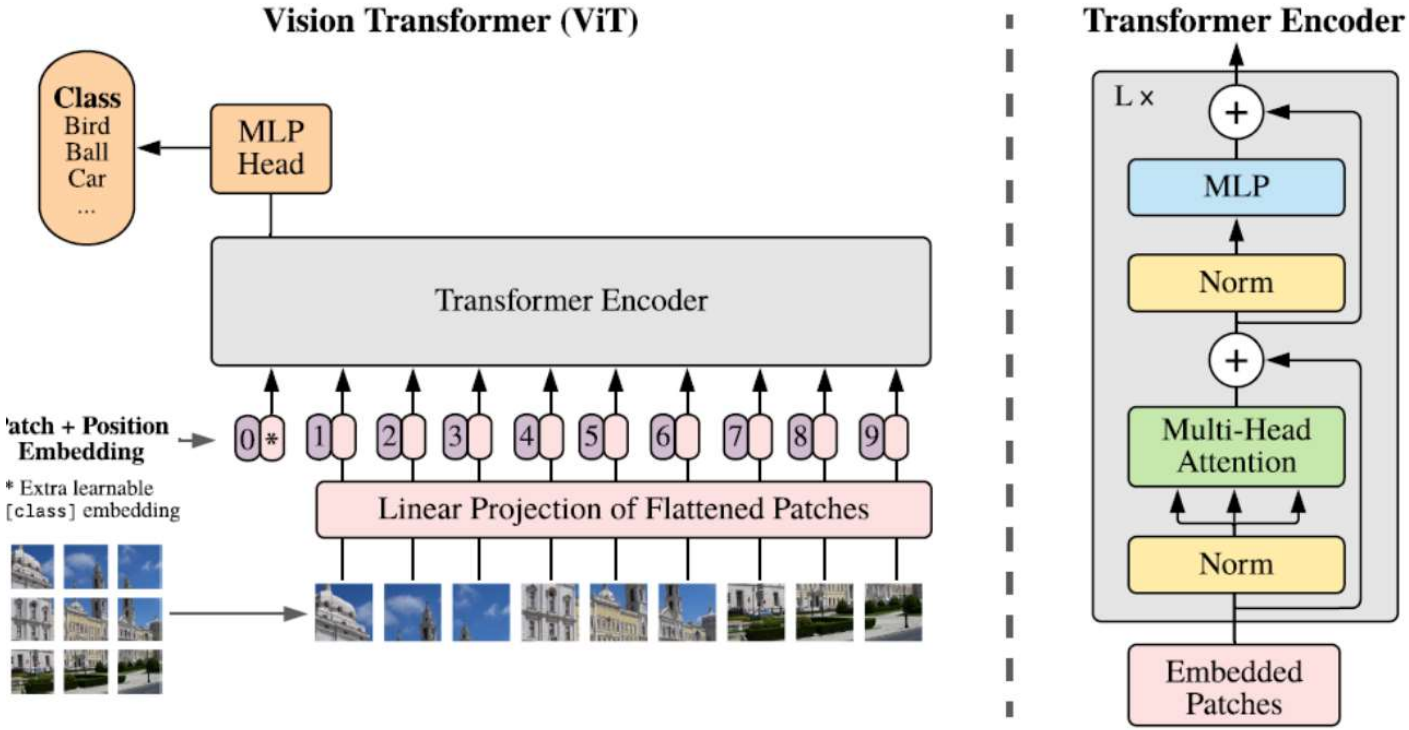
A generic framework for using transformer in image processing



Vision Transformer (ViT)

Dosovitskiy...2021

Vision Transformer is a pure transformer directly applies to the sequences of image patches for image classification task.
It follows transformer's original design as much as possible.



The framework of ViT

How the Vision Transformer works

* Nikolas Adaloglou

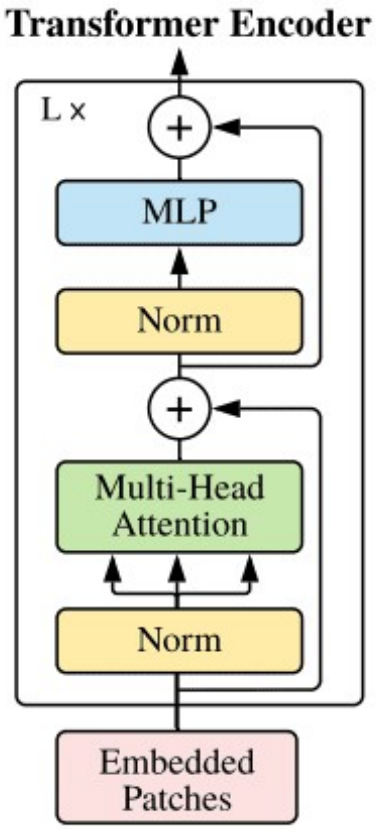
The total architecture is called Vision Transformer (ViT in short). Let's examine it step by step.

1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the flattened patches
4. Add positional embedding **(they model positional embeddings with trainable linear layers)**
5. Feed the sequence as an input to a standard transformer encoder
6. Pretrain the model with image labels (fully supervised on a huge dataset)
7. Fine tune on the downstream dataset for image classification

Image patches are basically the sequence tokens (like words).

In fact, the encoder block is identical to the original transformer proposed by Vaswani et al. (2017)

The only thing that changes is the number of those blocks.



To this end, and to further prove that with more data they can train larger ViT variants, 3 models were proposed:

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

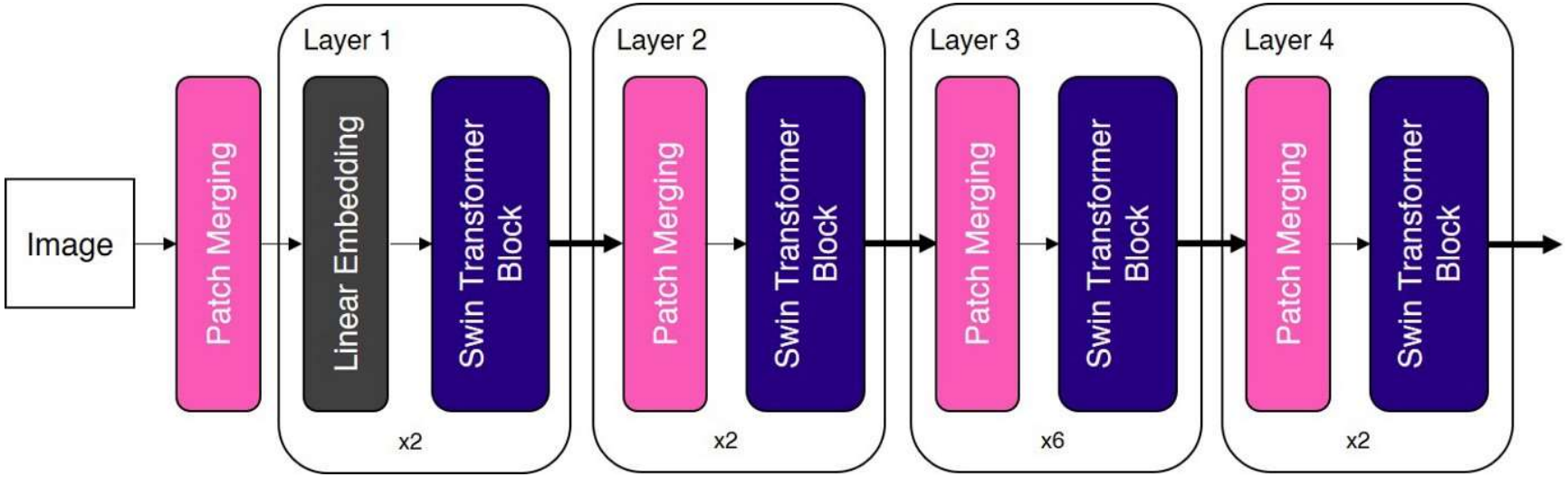
Heads refer to multi-head attention, while the MLP size refers to the blue module in the figure.
MLP stands for multi-layer perceptron but it's actually a bunch of linear transformation layers.
Hidden size D is the embedding size, which is kept fixed throughout the layers. Why keep it fixed? So that we can use short residual skip connections.
But is this enough?
Yes and no. Actually, we need a massive amount of data and as a result computational resources.

Swin Transformer

- Swin Transformer (Liu et al., 2021) is **a transformer-based deep learning model with state-of-the-art performance in vision tasks**. Unlike the Vision Transformer (ViT) (Dosovitskiy et al., 2020) which precedes it, Swin Transformer is highly efficient and has greater accuracy.
- In 2020, the Vision Transformer (ViT) garnered much attention from the AI community, notable for its pure transformer architecture with promising results in vision tasks. Despite its promise,
- ViTs suffer from several shortcomings:
  1. Most notably, ViTs struggle with high resolution images as its computational complexity is *quadratic* to the image size.
  2. Furthermore, the fixed scale tokens in ViTs are unsuitable in vision tasks where the visual elements are of variable scale.
- A flurry of research work followed ViT, and most of them made enhancements to the standard transformer architecture in order to address the above-mentioned shortcomings.
- In 2021, Microsoft researchers published the Swin Transformer (Liu et al., 2021), arguably one of the most exciting piece of research following up from the original ViT

Swin Transformers

The Swin Transformer introduced two key concepts to address the issues faced by the original ViT — **hierarchical feature maps** and **shifted window attention.** In fact, the name of Swin Transformer comes from "**S**hifted **win**dow **Transformer**". The overall architecture of the Swin Transformer is shown below.



As we can see, the 'Patch Merging' block and the 'Swin Transformer Block' are the two key building blocks in Swin Transformer.

The shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection.

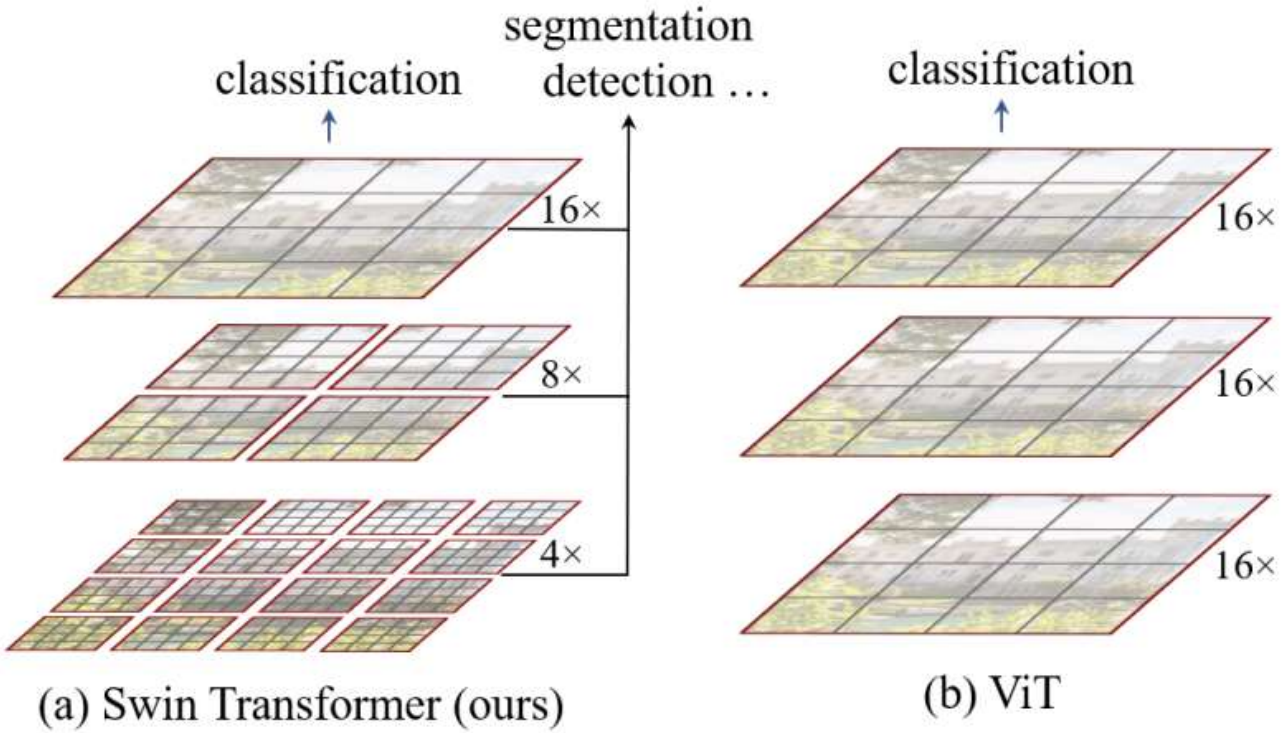


TABLE 1: Representative works of vision transformers.

Category	Sub-category	Method	Highlights	Publication
Backbone	Supervised pretraining	ViT [15]	Image patches, standard transformer	ICLR 2021
		TNT [29]	Transformer in transformer, local attention	NeurIPS 2021
		Swin [30]	Shifted window, window-based self-attention	ICCV 2021
	Self-supervised pretraining	iGPT [14]	Pixel prediction self-supervised learning, GPT model	ICML 2020
		MoCo v3 [31]	Contrastive self-supervised learning, ViT	ICCV 2021
High/Mid-level vision	Object detection	DETR [16]	Set-based prediction, bipartite matching, transformer	ECCV 2020
		Deformable DETR [17]	DETR, deformable attention module	ICLR 2021
		UP-DETR [32]	Unsupervised pre-training, random query patch detection	CVPR 2021
	Segmentation	Max-DeepLab [25]	PQ-style bipartite matching, dual-path transformer	CVPR 2021
		VisTR [33]	Instance sequence matching and segmentation	CVPR 2021
		SETR [18]	Sequence-to-sequence prediction, standard transformer	CVPR 2021
	Pose Estimation	Hand-Transformer [34]	Non-autoregressive transformer, 3D point set	ECCV 2020
		HOT-Net [35]	Structured-reference extractor	MM 2020
		METRO [36]	Progressive dimensionality reduction	CVPR 2021
Low-level vision	Image generation	Image Transformer [27]	Pixel generation using transformer	ICML 2018
		Taming transformer [37]	VQ-GAN, auto-regressive transformer	CVPR 2021
		TransGAN [38]	GAN using pure transformer architecture	NeurIPS 2021
	Image enhancement	IPT [19]	Multi-task, ImageNet pre-training, transformer model	CVPR 2021
		TTSR [39]	Texture transformer, RefSR	CVPR 2020
Video processing	Video inpainting	STTN [28]	Spatial-temporal adversarial loss	ECCV 2020
	Video captioning	Masked Transformer [20]	Masking network, event proposal	CVPR 2018
Multimodality	Classification	CLIP [40]	NLP supervision for images, zero-shot transfer	arXiv 2021
	Image generation	DALL-E [41]	Zero-shot text-to image generation	ICML 2021
		Cogview [42]	VQ-VAE, Chinese input	NeurIPS 2021
	Multi-task	UniT [43]	Different NLP & CV tasks, shared model parameters	ICCV 2021
Efficient transformer	Decomposition	ASH [44]	Number of heads, importance estimation	NeurIPS 2019
	Distillation	TinyBert [45]	Various losses for different modules	EMNLP Findings 2020
	Quantization	FullyQT [46]	Fully quantized transformer	EMNLP Findings 2020
	Architecture design	ConvBert [47]	Local dependence, dynamic convolution	NeurIPS 2020

*High-level vision deals with the interpretation and use of what is seen in the image **(events)**

*Mid-level vision deals with how this information is organized as objects and their attributes **(classification, detection, segmentation)**

***Low-level vison deals with how this information is organized as edges, blobs, and… (In image generation and enhancement)**

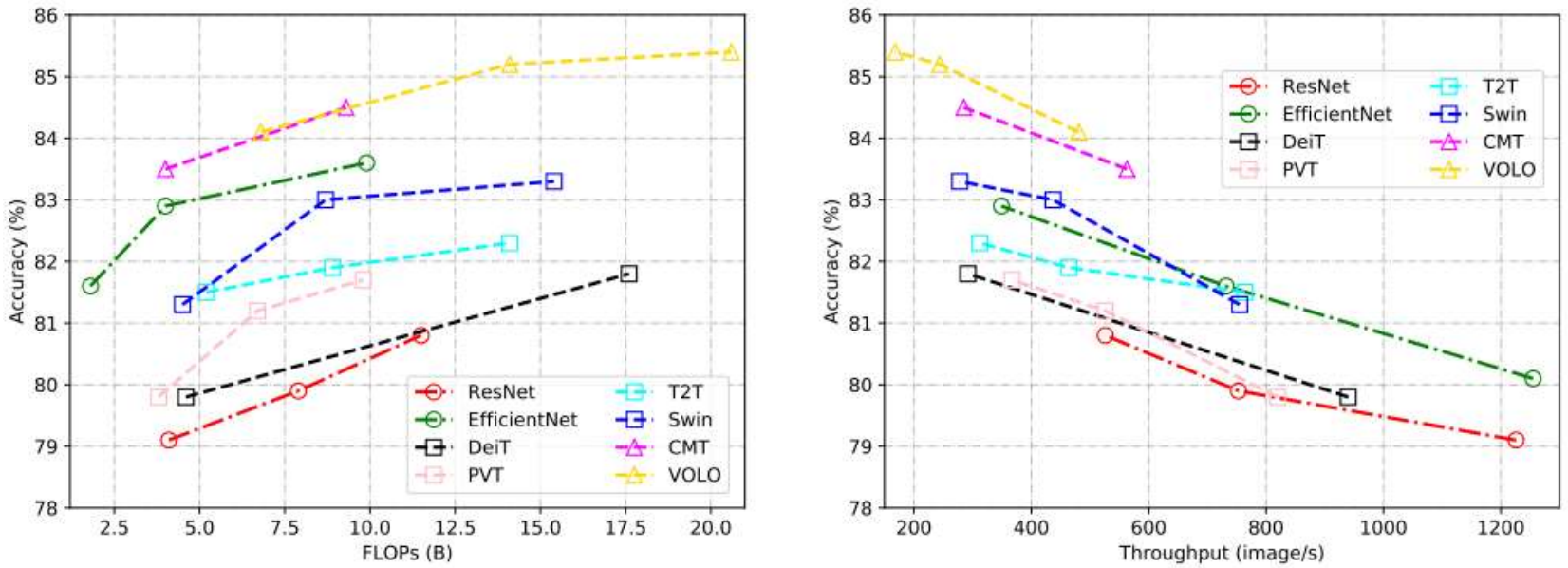
ImageNet result comparison of representative CNN and vision transformer models

Pure transformer means only using a few convolutions in the stem stage.
CNN + Transformer means using convolutions in the intermediate layers.

Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
CNN				
ResNet-50 [12], [67]	25.6	4.1	1226	79.1
ResNet-101 [12], [67]	44.7	7.9	753	79.9
ResNet-152 [12], [67]	60.2	11.5	526	80.8
EfficientNet-B0 [93]	5.3	0.39	2694	77.1
EfficientNet-B1 [93]	7.8	0.70	1662	79.1
EfficientNet-B2 [93]	9.2	1.0	1255	80.1
EfficientNet-B3 [93]	12	1.8	732	81.6
EfficientNet-B4 [93]	19	4.2	349	82.9

Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
Pure Transformer				
DeiT-Ti [15], [59]	5	1.3	2536	72.2
DeiT-S [15], [59]	22	4.6	940	79.8
DeiT-B [15], [59]	86	17.6	292	81.8
T2T-ViT-14 [67]	21.5	5.2	764	81.5
T2T-ViT-19 [67]	39.2	8.9	464	81.9
T2T-ViT-24 [67]	64.1	14.1	312	82.3
PVT-Small [72]	24.5	3.8	820	79.8
PVT-Medium [72]	44.2	6.7	526	81.2
PVT-Large [72]	61.4	9.8	367	81.7
TNT-S [29]	23.8	5.2	428	81.5
TNT-B [29]	65.6	14.1	246	82.9
CPVT-S [85]	23	4.6	930	80.5
CPVT-B [85]	88	17.6	285	82.3
Swin-T [60]	29	4.5	755	81.3
Swin-S [60]	50	8.7	437	83.0
Swin-B [60]	88	15.4	278	83.3
CNN + Transformer				
Twins-SVT-S [62]	24	2.9	1059	81.7
Twins-SVT-B [62]	56	8.6	469	83.2
Twins-SVT-L [62]	99.2	15.1	288	83.7
Shuffle-T [65]	29	4.6	791	82.5
Shuffle-S [65]	50	8.9	450	83.5
Shuffle-B [65]	88	15.6	279	84.0
CMT-S [94]	25.1	4.0	563	83.5
CMT-B [94]	45.7	9.3	285	84.5
VOLO-D1 [95]	27	6.8	481	84.2
VOLO-D2 [95]	59	14.1	244	85.2
VOLO-D3 [95]	86	20.6	168	85.4
VOLO-D4 [95]	193	43.8	100	85.7
VOLO-D5 [95]	296	69.0	64	86.1

FLOPs and throughput comparison of representative CNN and vision transformer models (ImageNet Top-1 (%)).



(a) Acc v.s. FLOPs.

(b) Acc v.s. throughput.

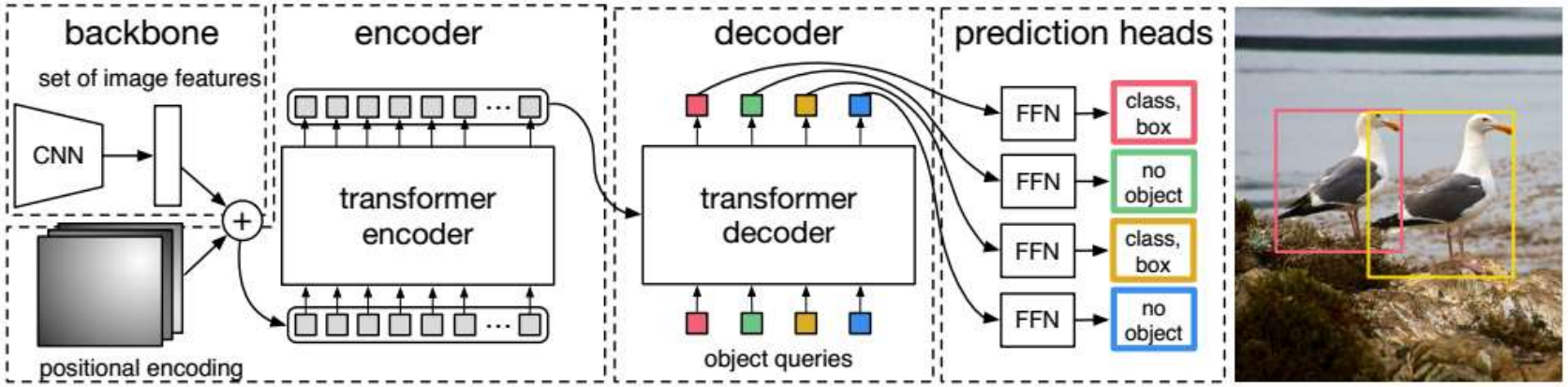
Throughput is how much information actually gets delivered in a certain amount of time.

FLOP: Floating point operations per second

ImageNet Benchmark (Image Classification) | Papers With Code

Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**Model soups** (ViT-G/14)	90.94%		1843M	✓	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	Transformer JFT-3B
2	**CoAtNet-7**	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
3	**ViT-G/14**	90.45%		1843M	✓	Scaling Vision Transformers			2021	Transformer JFT-3B
4	**CoAtNet-6**	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
5	**V-MoE-15B** (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts			2021	Transformer
6	**Meta Pseudo Labels** (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels			2021	EfficientNet JFT-300M

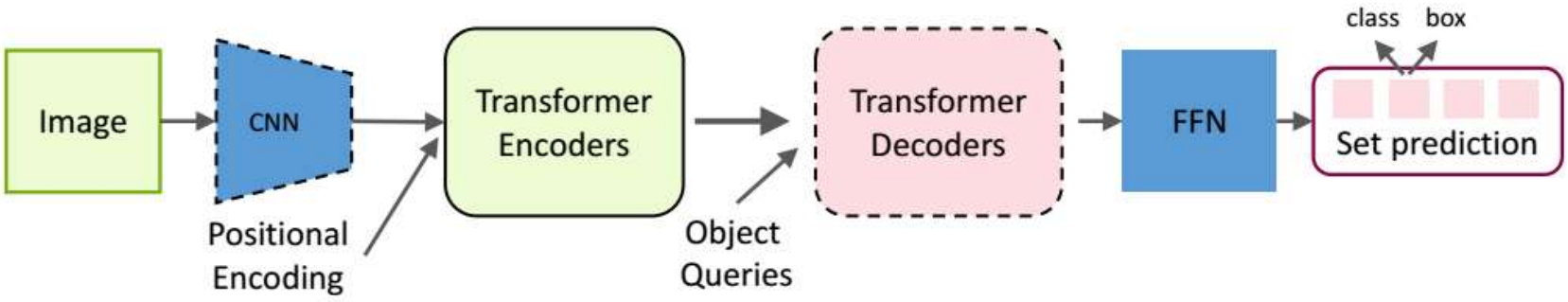
Transformers in Detection



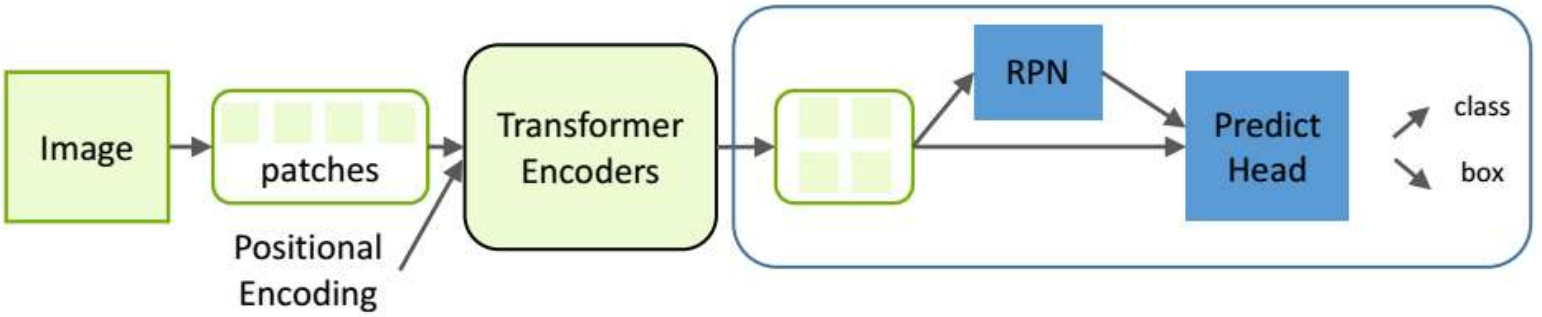
The overall architecture of DETR

*N. Carion et al. End-to-end object detection with transformers. In *ECCV*, 2020

General framework of transformer-based object detection



(a) Transformer-based set prediction for detection

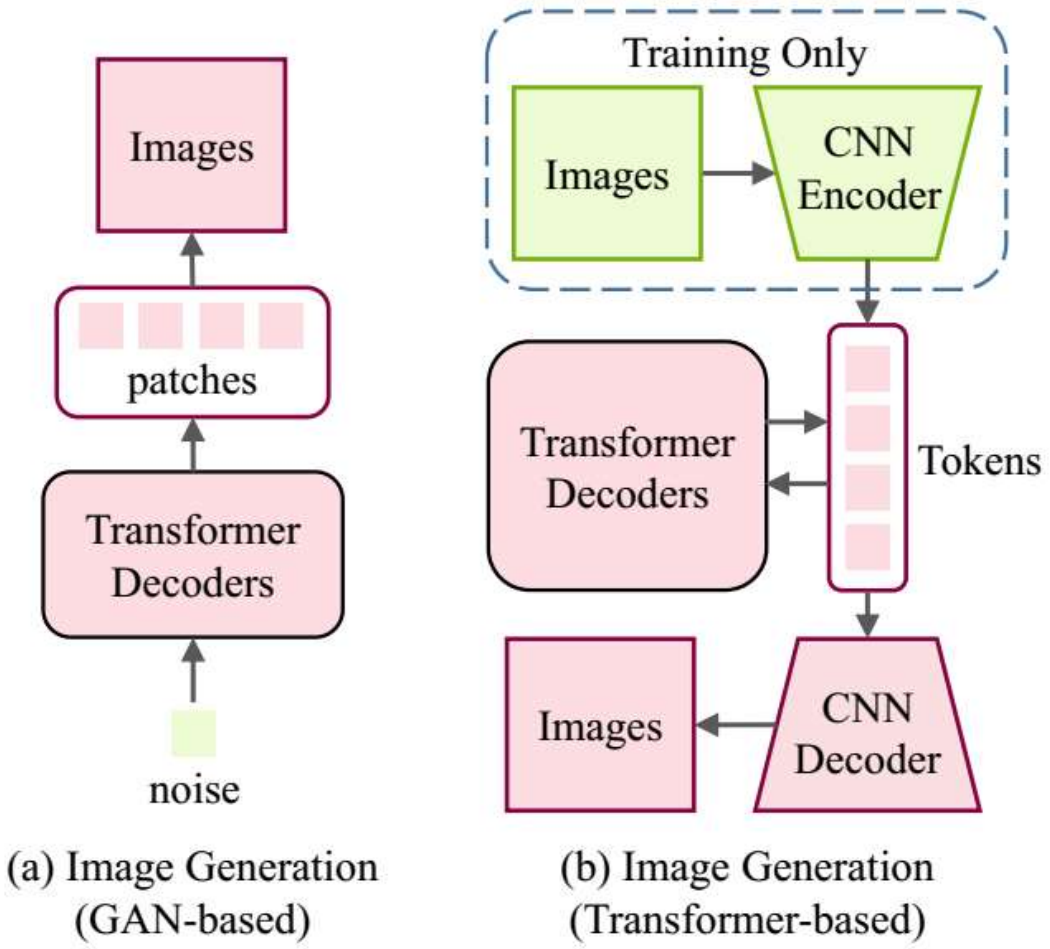


(b) Transformer-based backbone for detection

TABLE 3: Comparison of different transformer-based object detectors on COCO 2017 val set. Running speed (FPS) is evaluated on an NVIDIA Tesla V100 GPU as reported in [17]. †Estimated speed according to the reported number in the paper. ‡ViT backbone is pre-trained on ImageNet-21k. *ViT backbone is pre-trained on an private dataset with 1.3 billion images.

Method	Epochs	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	#Params (M)	GFLOPs	FPS
CNN based										
FCOS [125]	36	41.0	59.8	44.1	26.2	44.6	52.2	-	177	23†
Faster R-CNN + FPN [13]	109	42.0	62.1	45.5	26.6	45.4	53.4	42	180	26
CNN Backbone + Transformer Head										
DETR [16]	500	42.0	62.4	44.2	20.5	45.8	61.1	41	86	28
DETR-DC5 [16]	500	43.3	63.1	45.9	22.5	47.3	61.1	41	187	12
Deformable DETR [17]	50	46.2	65.2	50.0	28.8	49.2	61.7	40	173	19
TSP-FCOS [120]	36	43.1	62.3	47.0	26.6	46.8	55.9	-	189	20†
TSP-RCNN [120]	96	45.0	64.5	49.6	29.7	47.7	58.0	-	188	15†
ACT+MKKD (L=32) [121]	-	43.1	-	-	61.4	47.1	22.2	-	169	14†
SMCA [123]	108	45.6	65.5	49.1	25.9	49.3	62.6	-	-	-
Efficient DETR [124]	36	45.1	63.1	49.1	28.3	48.4	59.0	35	210	-
UP-DETR [32]	150	40.5	60.8	42.6	19.0	44.4	60.0	41	-	-
UP-DETR [32]	300	42.8	63.0	45.3	20.8	47.1	61.7	41	-	-
Transformer Backbone + CNN Head										
ViT-B/16-FRCNN‡ [113]	21	36.6	56.3	39.3	17.4	40.0	55.5	-	-	-
ViT-B/16-FRCNN* [113]	21	37.8	57.4	40.1	17.8	41.4	57.3	-	-	-
PVT-Small+RetinaNet [72]	12	40.4	61.3	43.0	25.0	42.9	55.7	34.2	118	-
Twins-SVT-S+RetinaNet [62]	12	43.0	64.2	46.3	28.0	46.4	57.5	34.3	104	-
Swin-T+RetinaNet [60]	12	41.5	62.1	44.2	25.1	44.9	55.5	38.5	118	-
Swin-T+ATSS [60]	36	47.2	66.5	51.3	-	-	-	36	215	-
Pure Transformer based										
PVT-Small+DETR [72]	50	34.7	55.7	35.4	12.0	36.4	56.7	40	-	-
TNT-S+DETR [29]	50	38.2	58.9	39.4	15.5	41.1	58.8	39	-	-
YOLOS-Ti [126]	300	30.0	-	-	-	-	-	6.5	21	-
YOLOS-S [126]	150	37.6	57.6	39.2	15.9	40.2	57.3	28	179	-
YOLOS-B [126]	150	42.0	62.2	44.5	19.5	45.3	62.1	127	537	-

Transformers in Image Generation



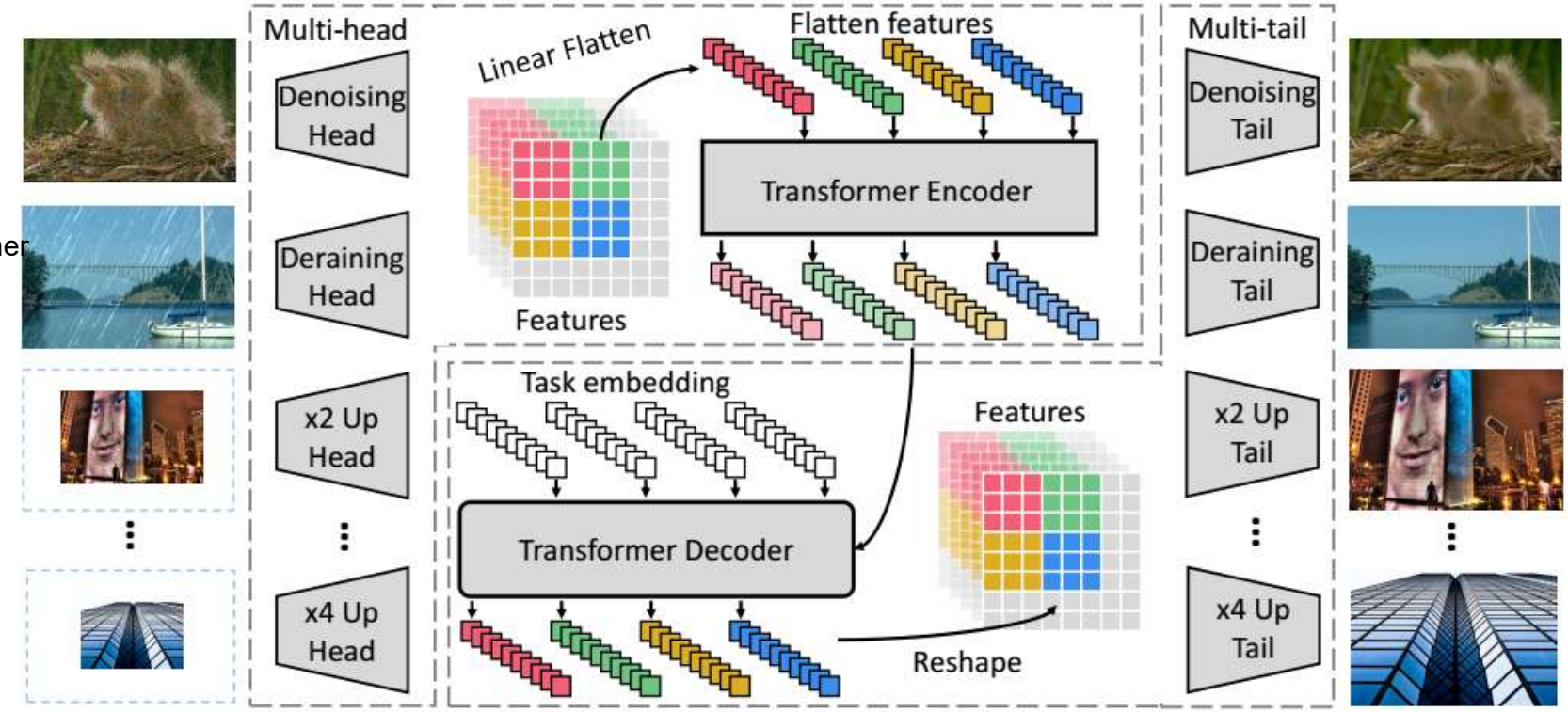
A generic framework for transformer in image generation

Transformers in Image Enhancement

IPT

Image Processing Transformer

[19] H. Chen et al. Pre-trained image processing transformer. In *CVPR*, 2021



IPT fully utilizes the advantages of transformers by using large pre-training datasets. It achieves state-of-the-art performance in several image processing tasks, including super-resolution, denoising, and deraining.

End of Part 1