Analysis and Design of Deep Neural Networks

Chapter 3
Deep Architectures and
Layer Wise Analysis and Design Algorithms

Fall 2023

3. Deep Architectures and Layer Wise Analysis and Design Algorithms

3.1 Architecture of CNNs

3.2 Layer Wise Analysis Algorithms

- Layer-wise Model evaluation
- Pre-train Model ranking
- Model Confidence and Guarantee

3.3 Architecture of Region Based CNNs_part1

3.4 Layer Wise Design Algorithms_part1

- Model Compressing

3.5 Architecture of Region Based CNNs_part2

3.6 Layer Wise Design Algorithms_part2

- Layer-wise forward learning

3.7 Architecture of Transformers

3.8 Layer Wise Design Algorithms_part3

- Auto Encoders
- Layer-wise branching/Fusion
- Artificial General Intelligent

3.9 Other Layer-wise Algorithms in Literature

3.1 Architectures of CNNs

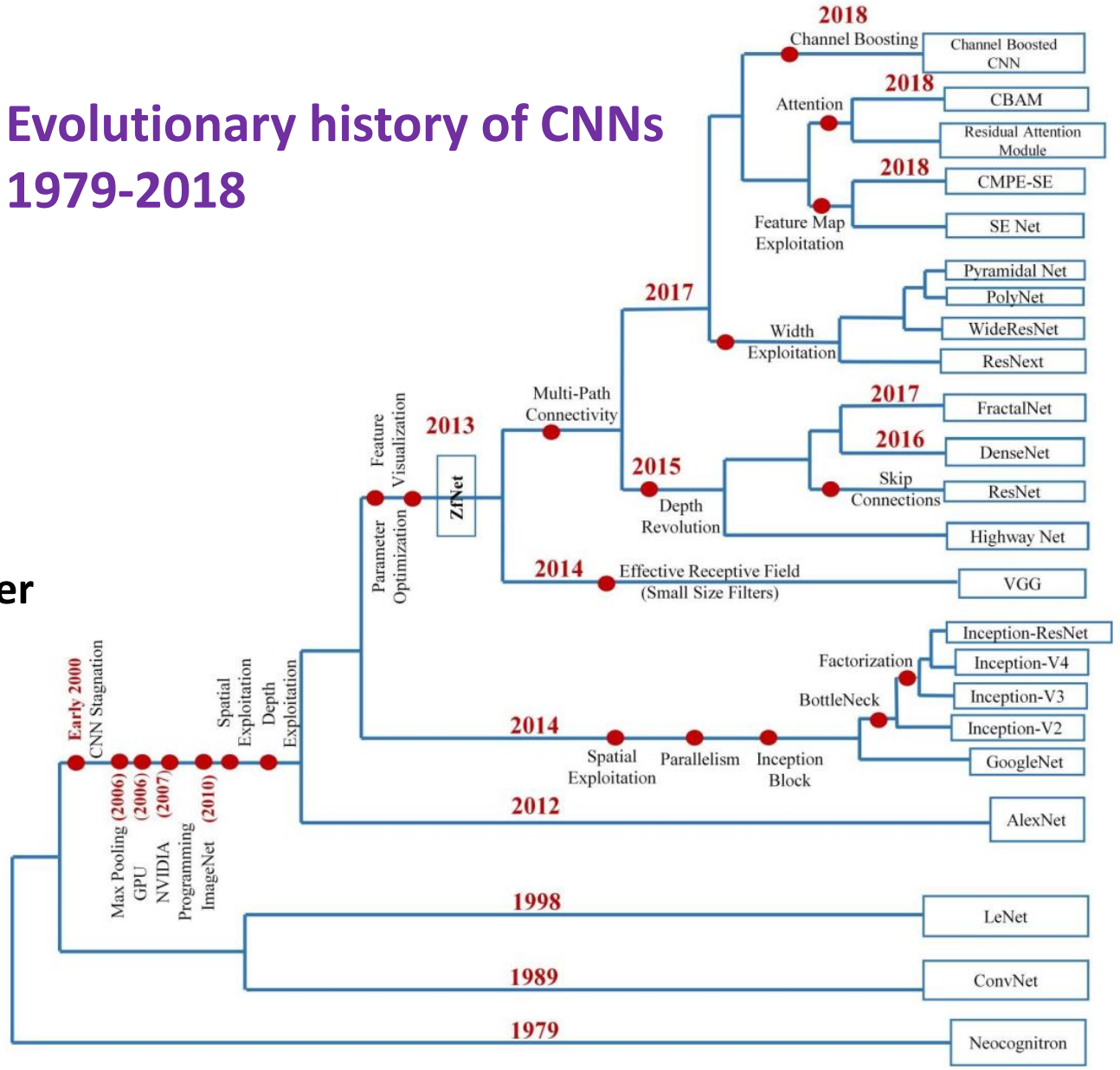
3.1.1 CNNs

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**

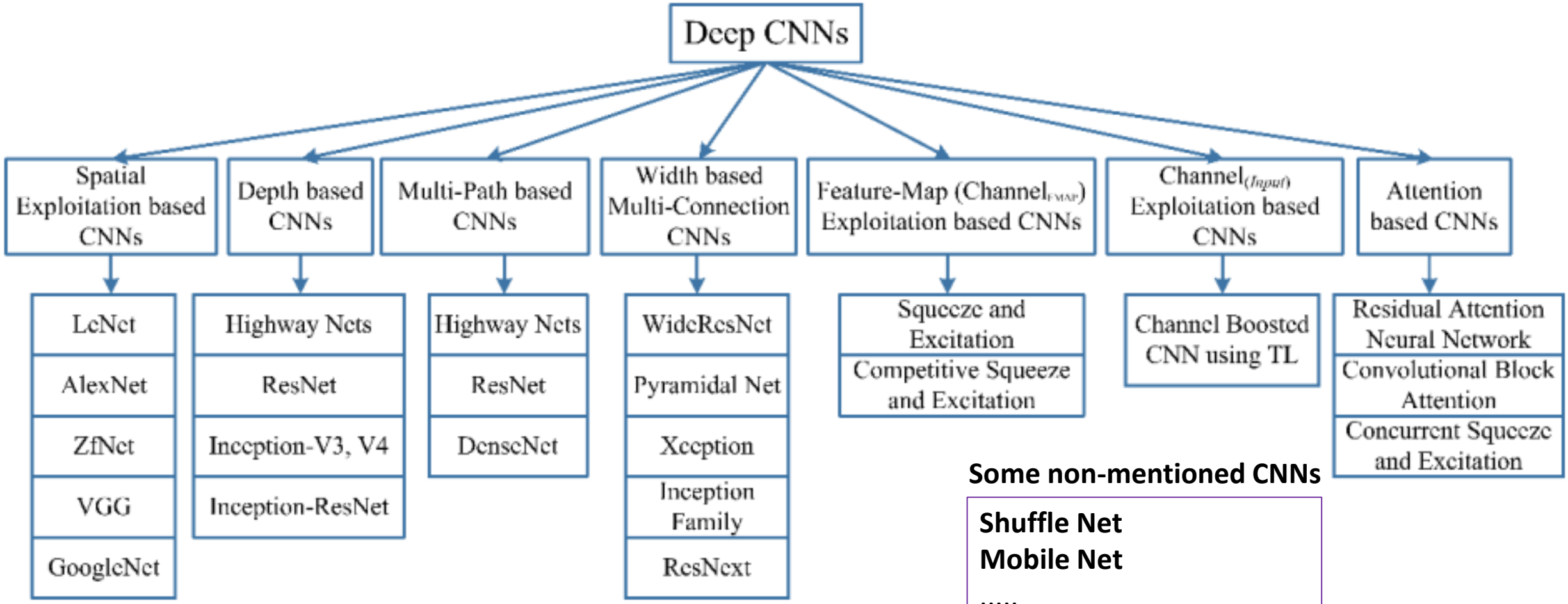
In 1998, LeCuN proposed an improved version of ConvNet, which was famously known as LeNet-5, and it started the use of CNN in classifying characters in a document recognition related applications (LeCun et al. 1995, 1998).

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

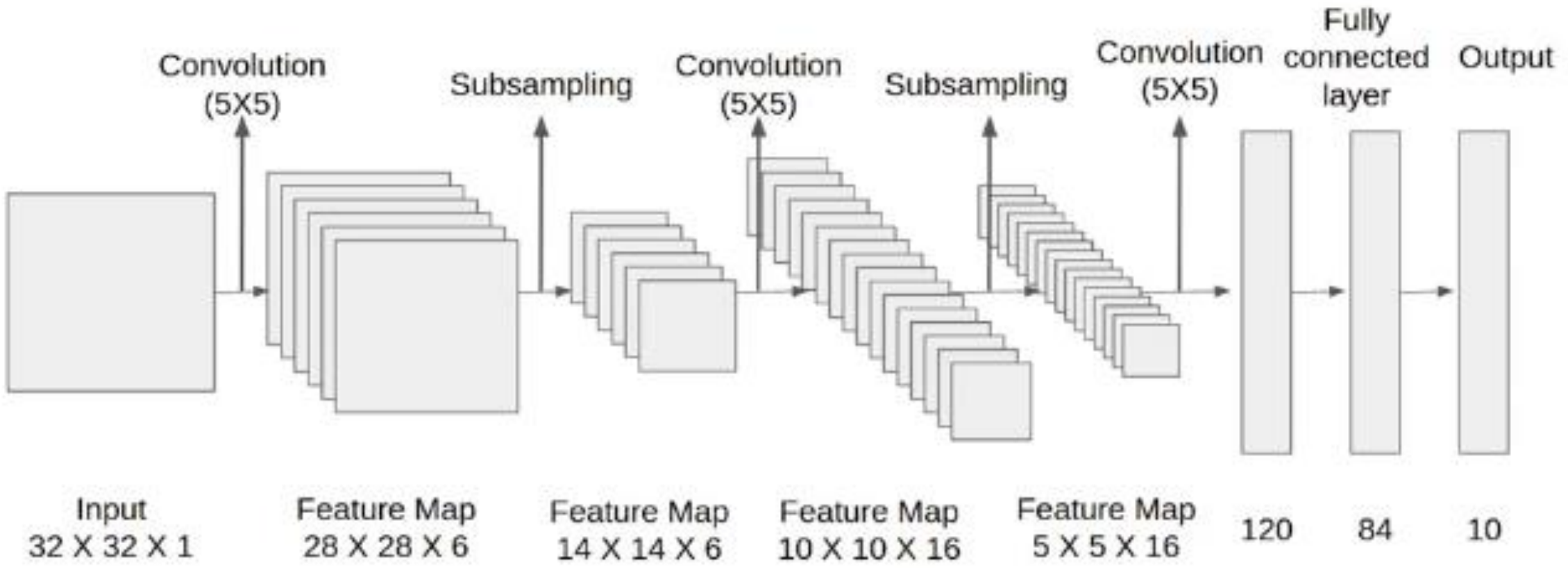
CNNs: **LeNet** **AlexNet** **ZFNet** **VGG** **GoogleNet**

.

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)

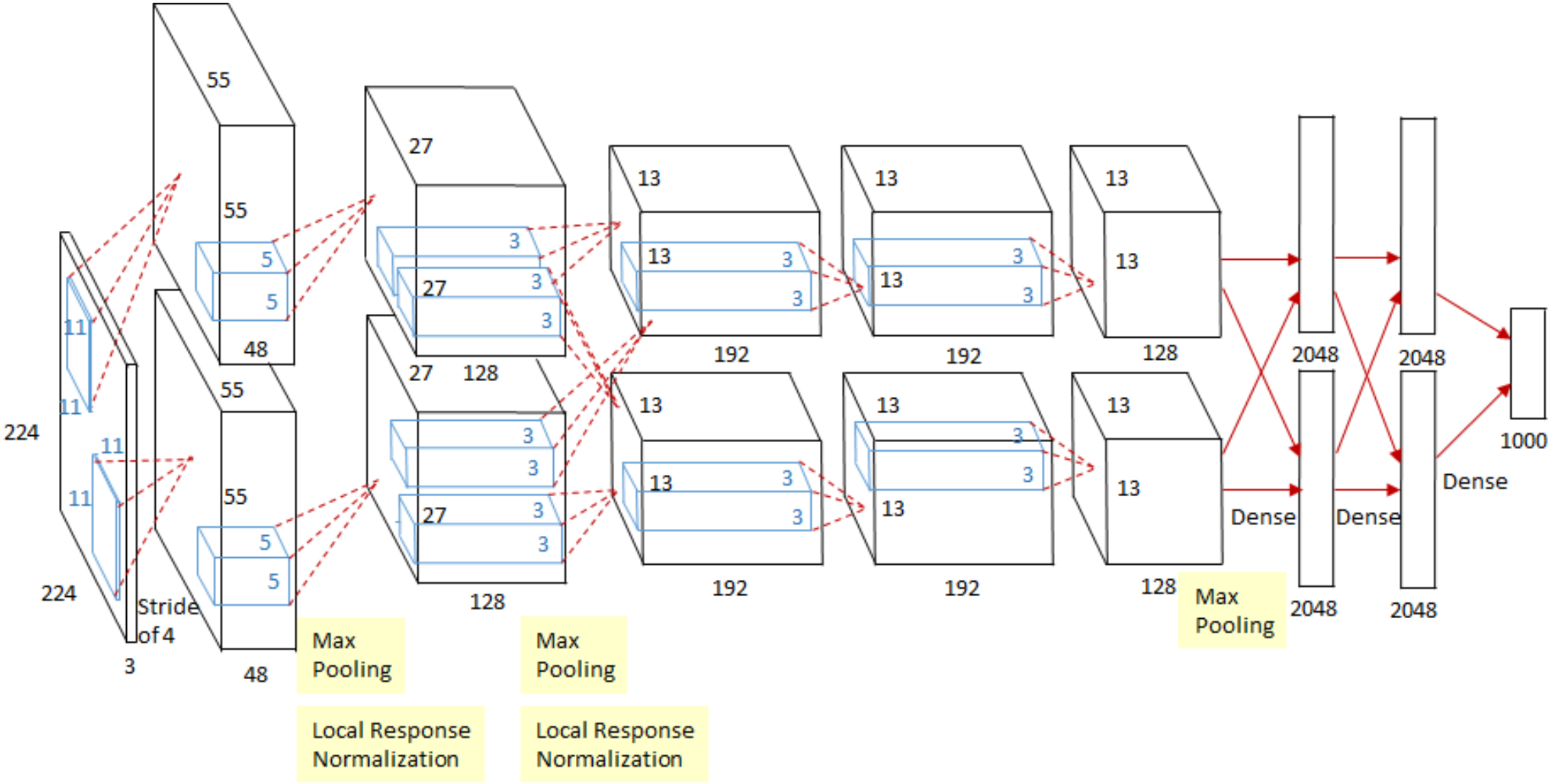
Lenet-5

First popular CNN architecture



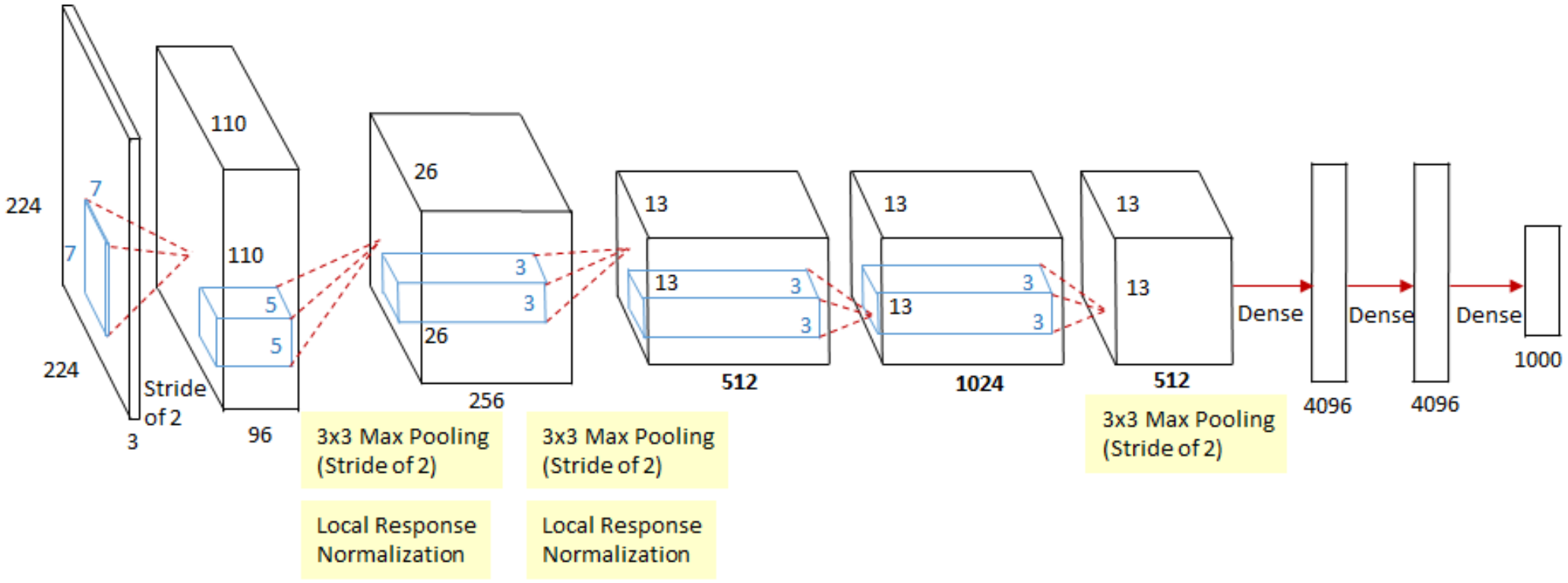
AlexNet

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



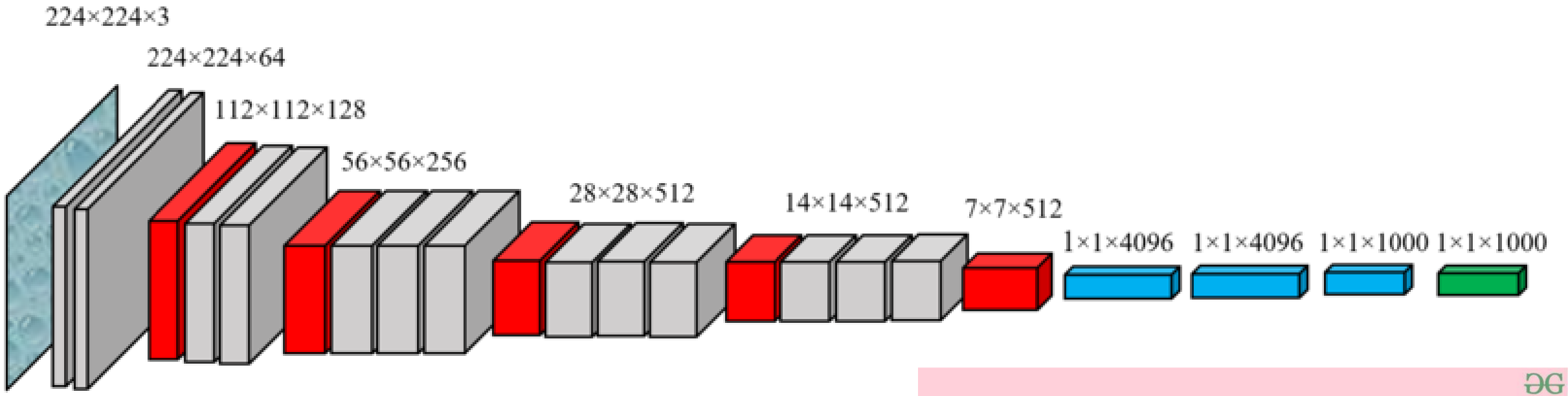
ZFNET

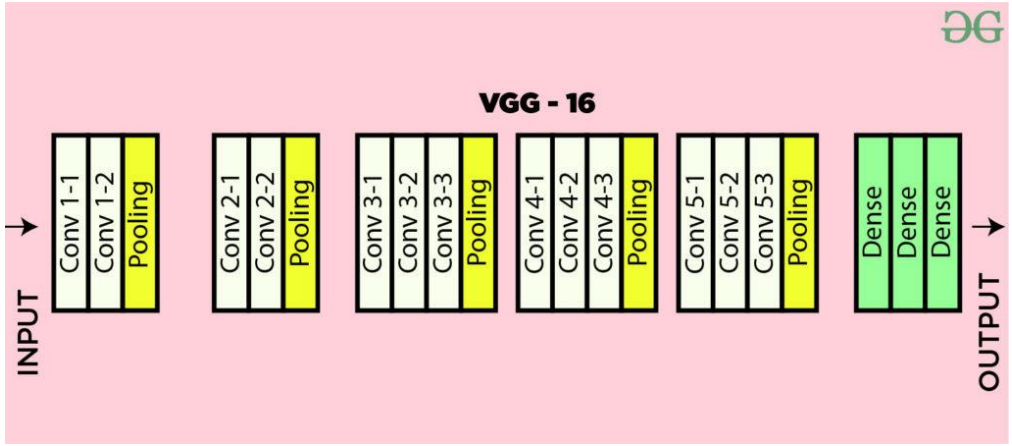
Visualization of intermediate layers



VGG

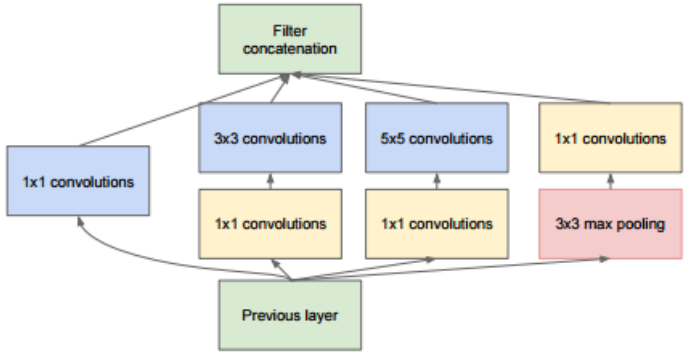
- Homogenous topology
- Uses small size kernels





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.

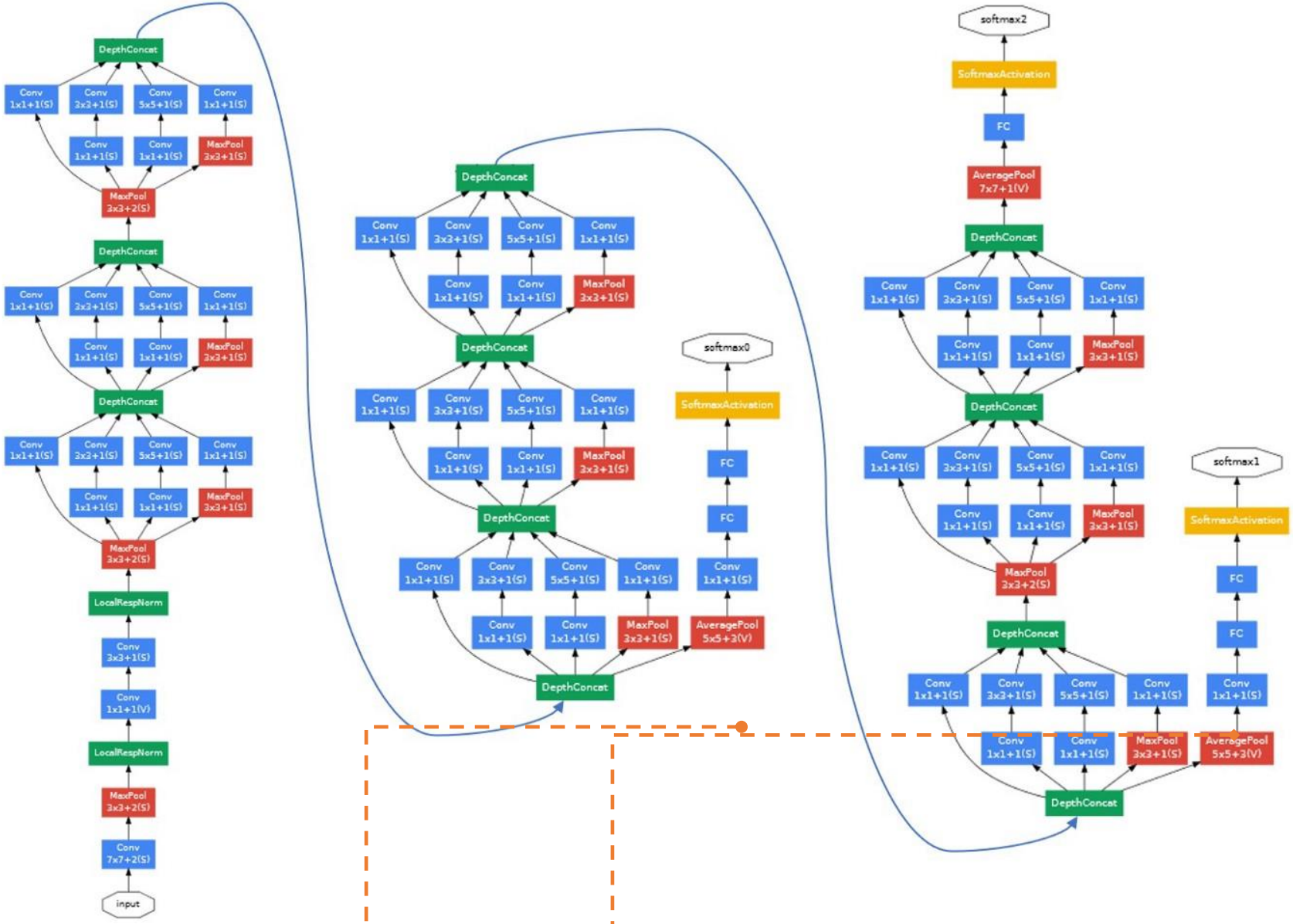


Table 5a Major challenges associated with implementation of Spatial exploitation based CNN architectures.

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

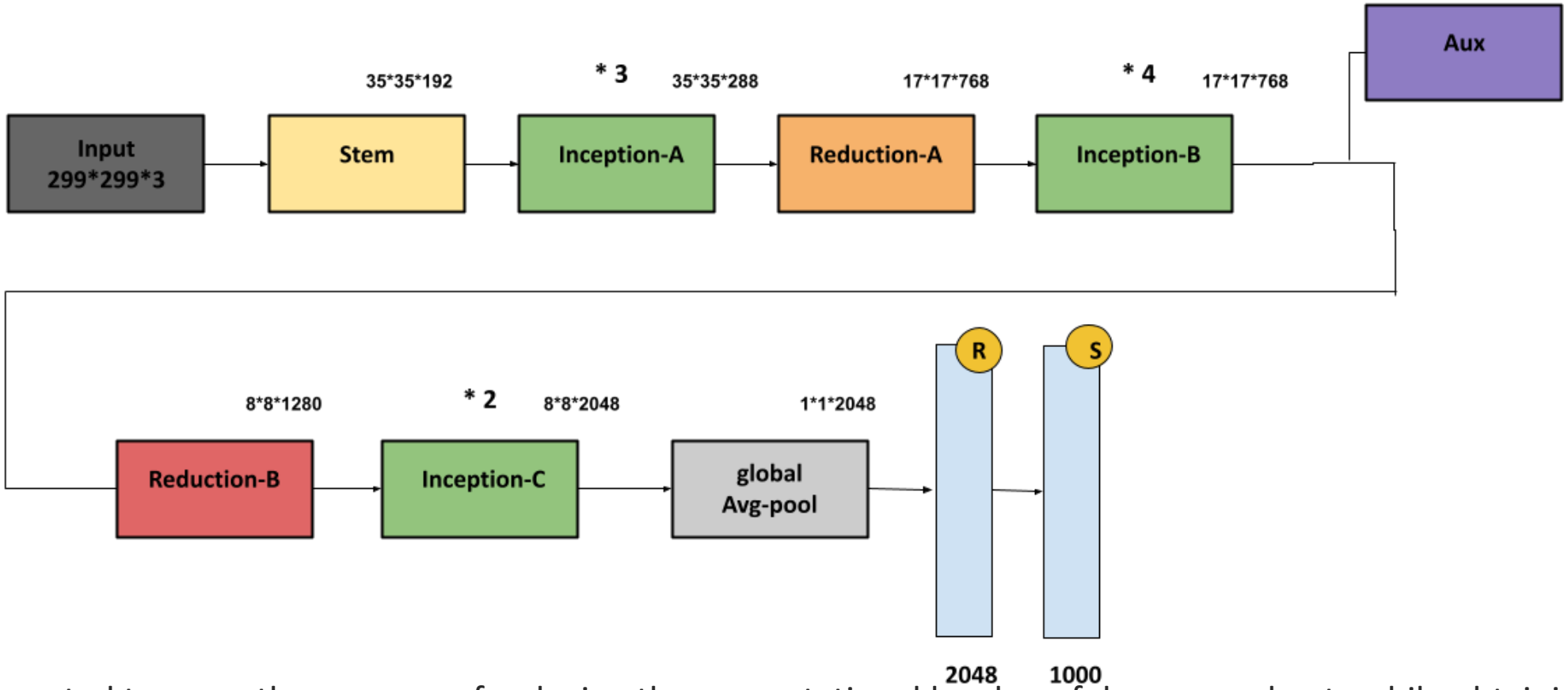
Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6	159	Depth + Width	(Szegedy et al. 2016b)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

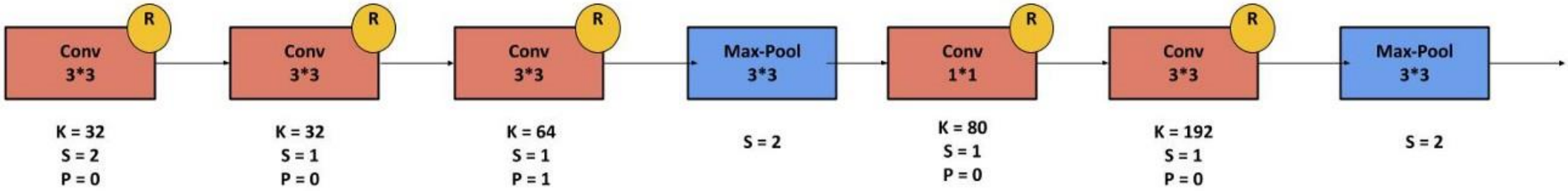
- Handles the problem of a representational bottleneck

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

- Replace large size filters with small filters

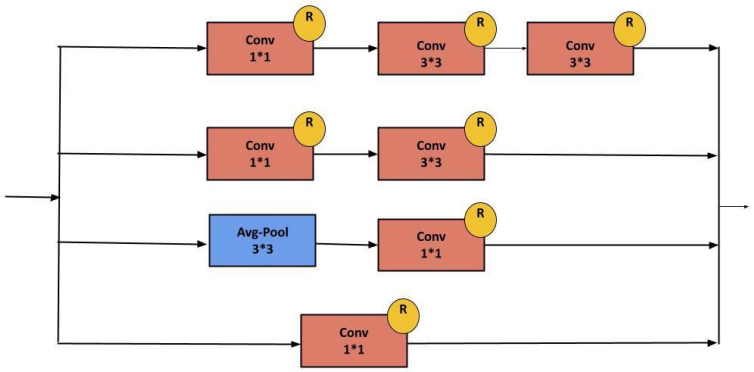


Inception is created to serve the purpose of reducing the computational burden of deep neural nets while obtaining state-of-art performance.

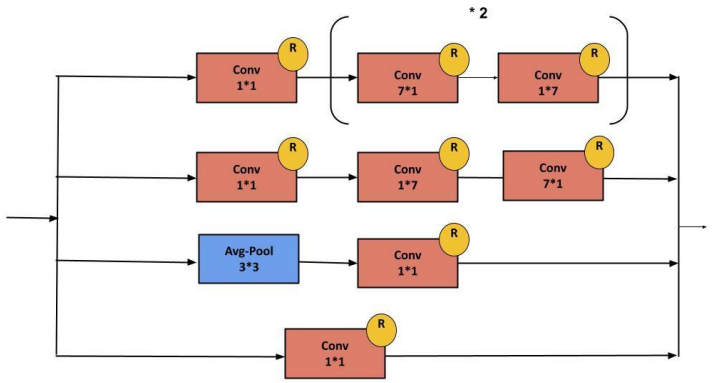


STEM

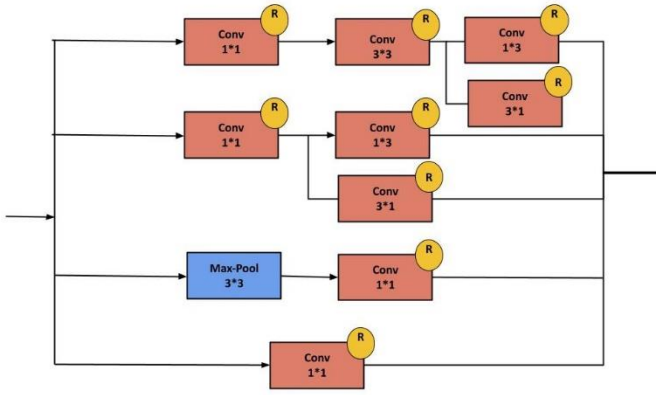
The Stem is a particular convolutional network module before the Inception-resnet blocks



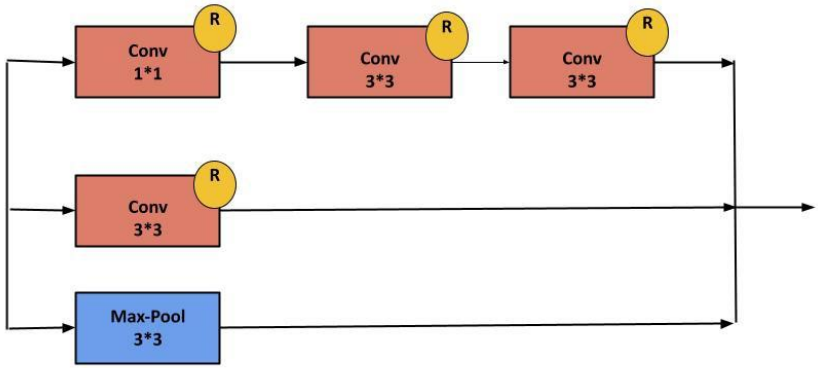
Inception-A Block :



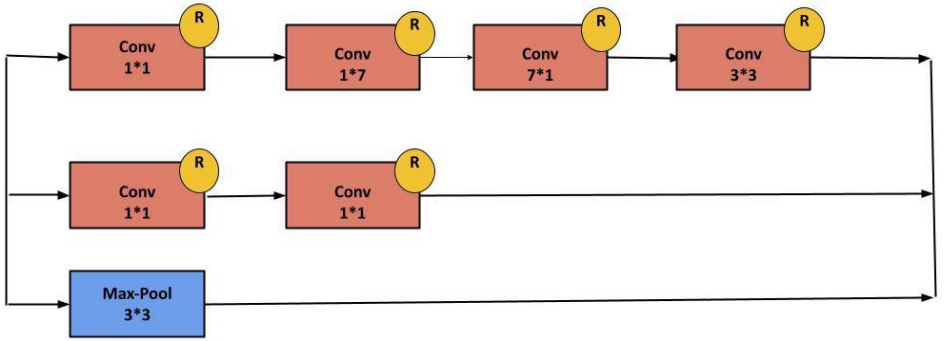
Inception-B Block :



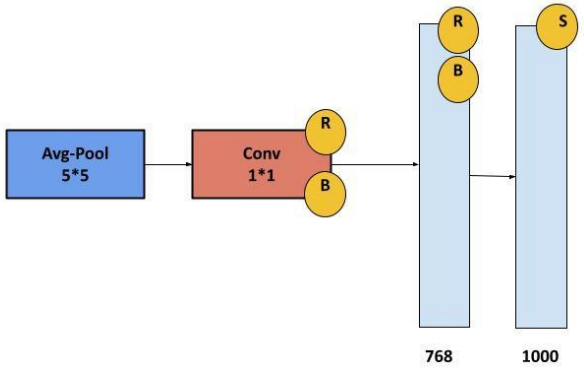
Inception-C Block :



Reduction-A Block :



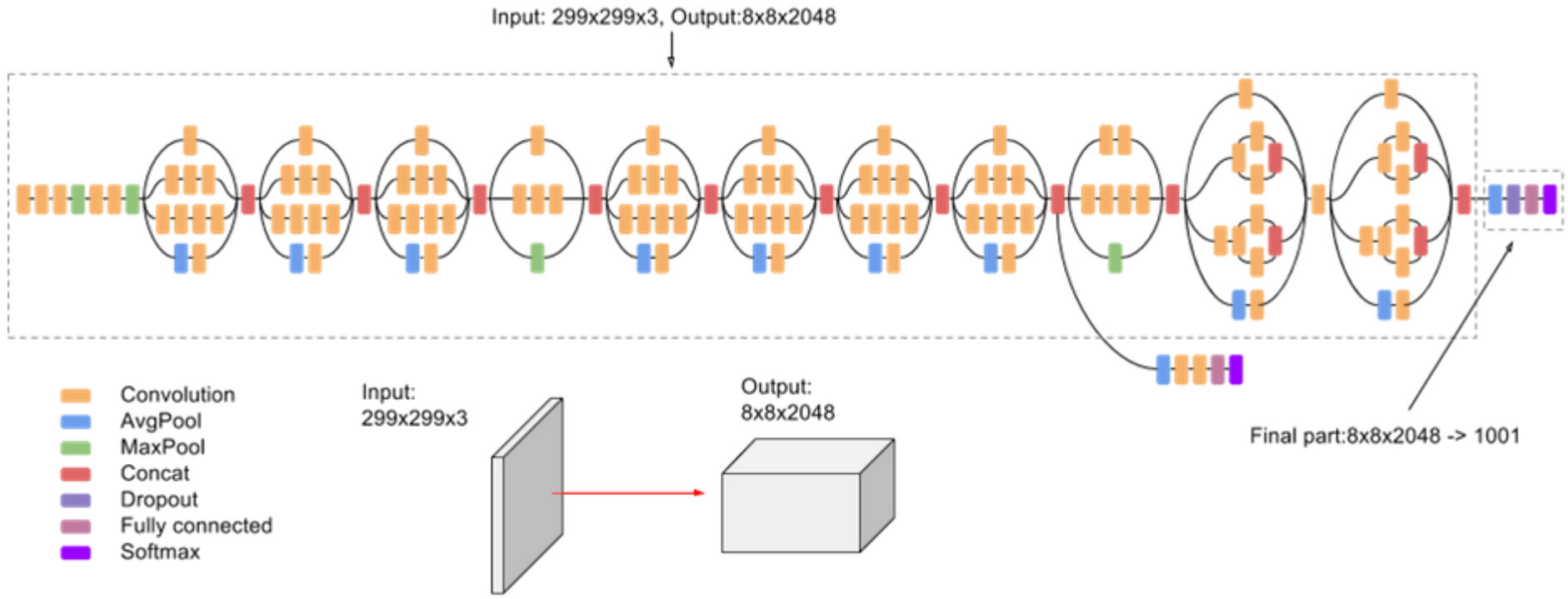
Reduction-B Block :



Auxiliary Classifier Block :

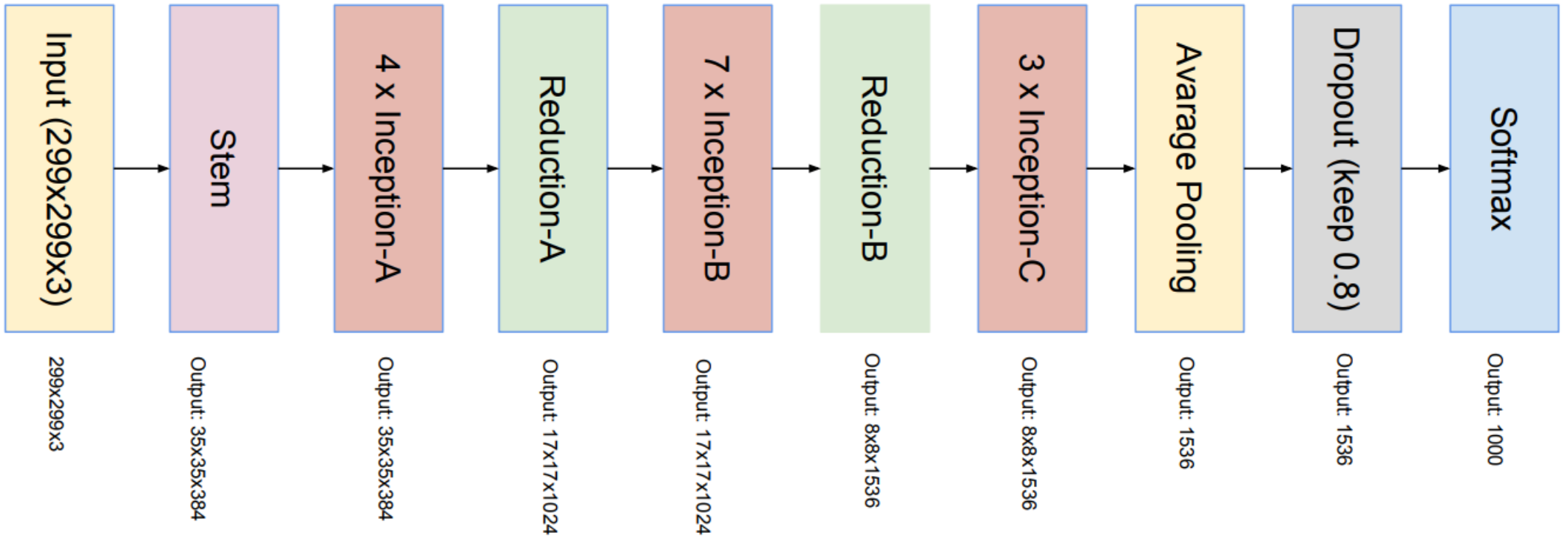
Auxiliary Classifiers are **type of architectural component that seek to improve the convergence of very deep networks**. They are classifier heads we attach to layers before the end of the network.

Inception 3

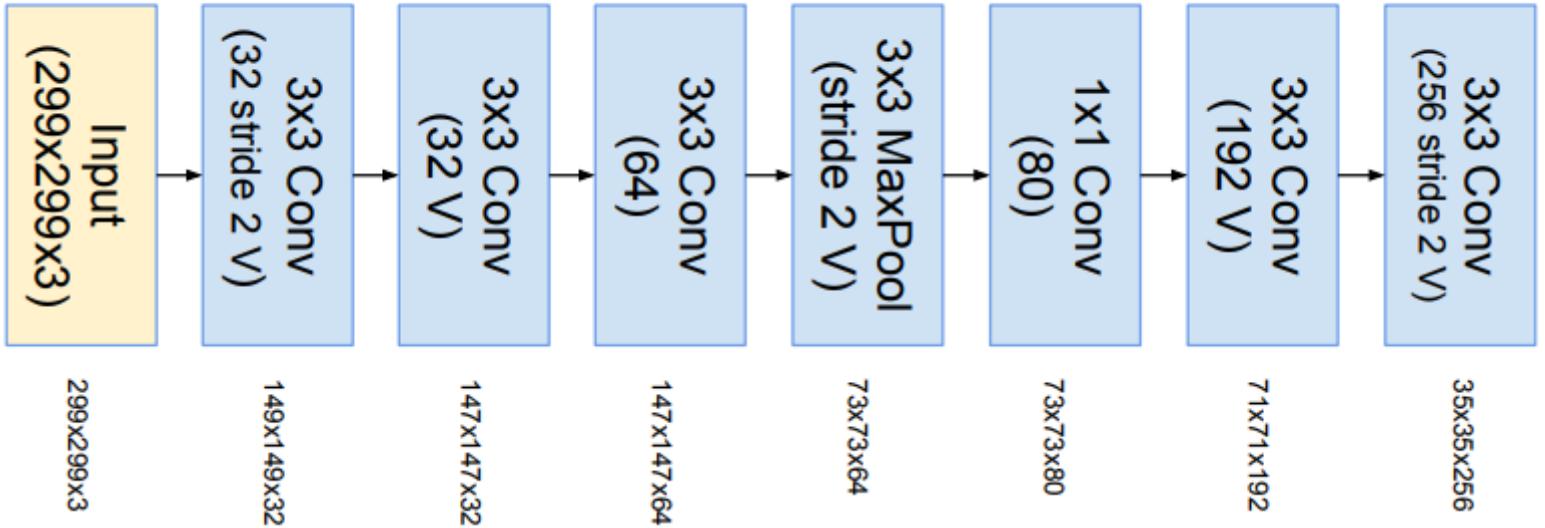

Input: 299x299x3, Output:8x8x2048
Final part:8x8x2048 -> 1001
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax
Input:
299x299x3
Output:
8x8x2048

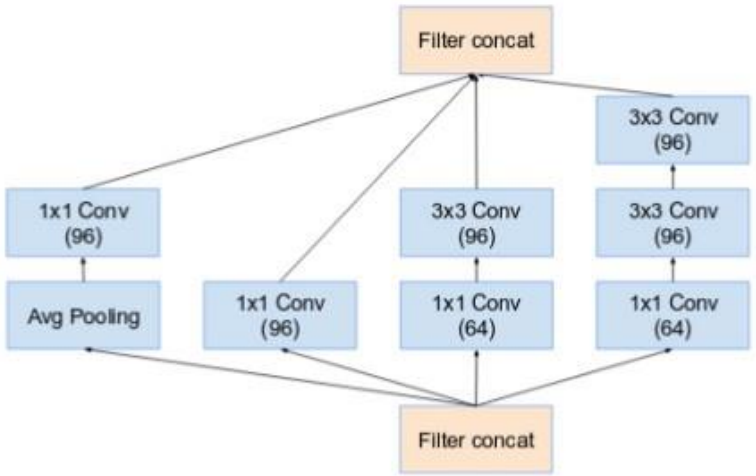
Inception 4

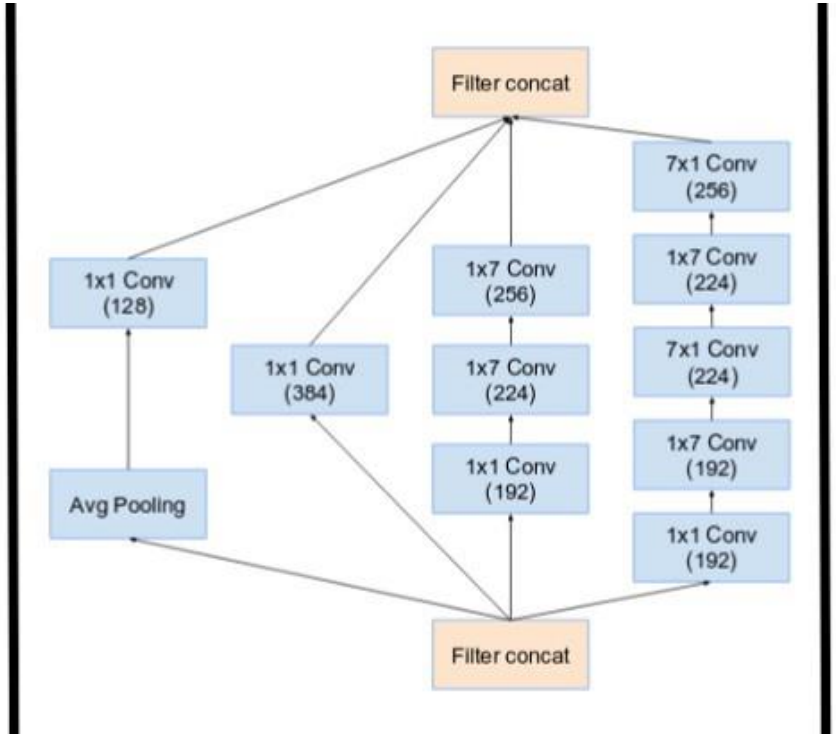
- Deep hierarchies features, Multi level feature representation
 -inception-v4 use more inception modules than Inception-v3.
- Split transform and merge idea Uses asymmetric filters

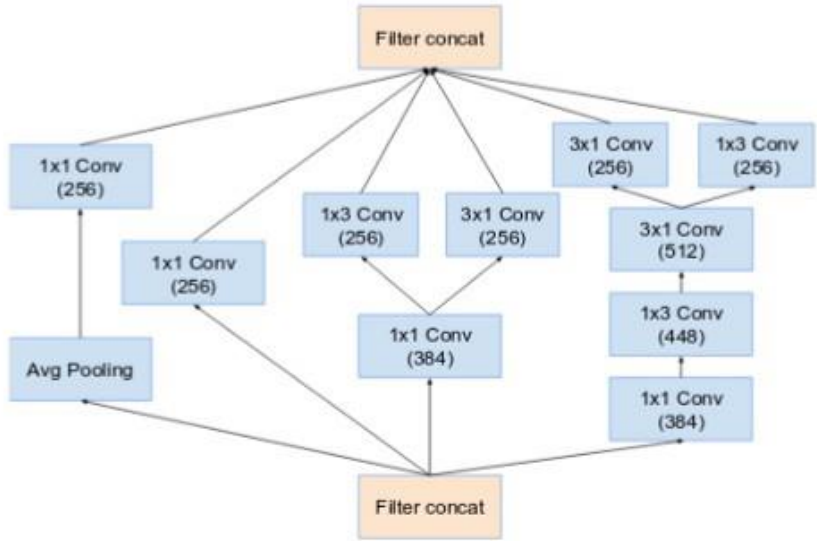


STEM

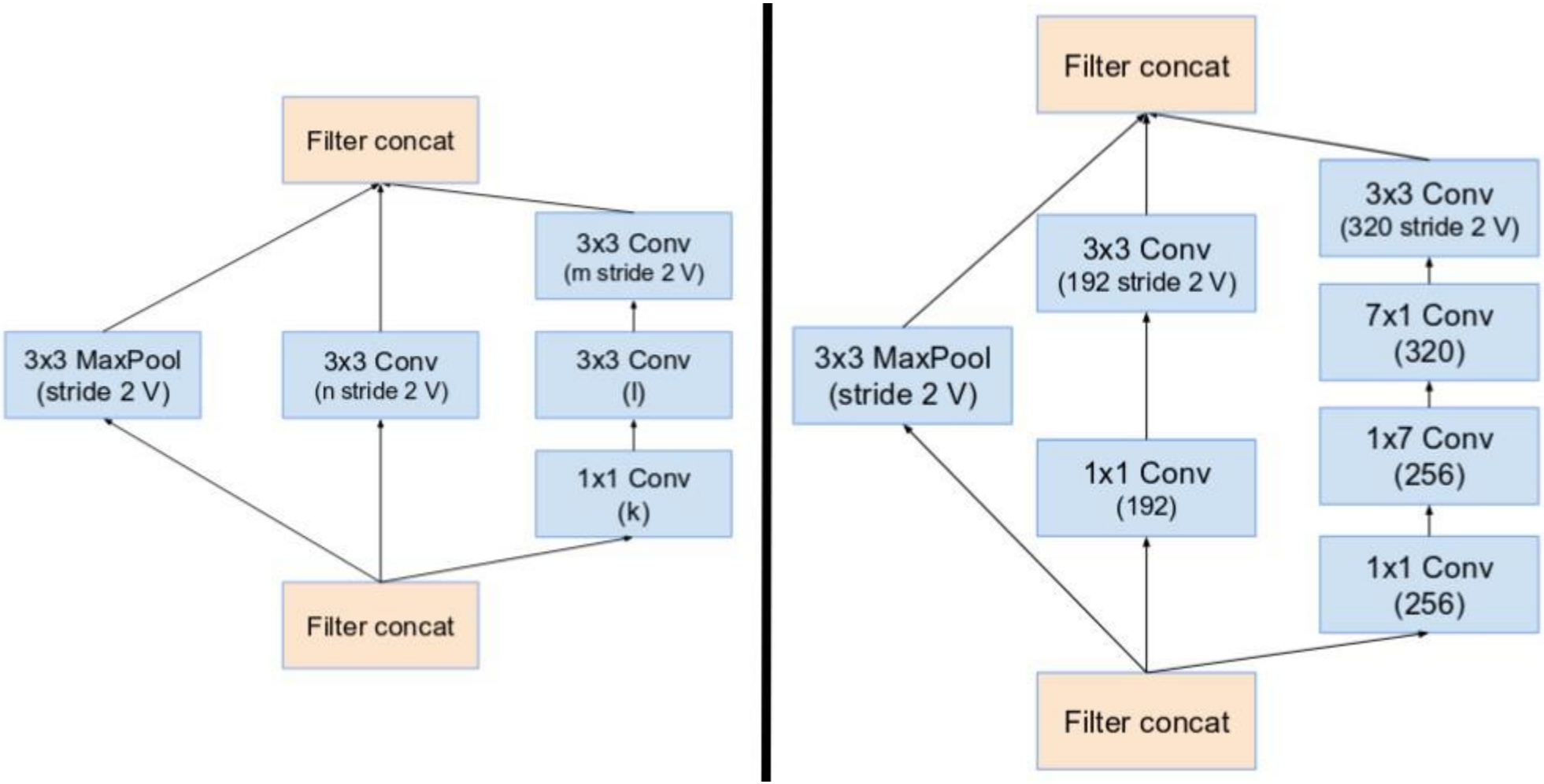

Input
(299x299x3)
299x299x3
3x3 Conv
(32 stride 2 V)
149x149x32
3x3 Conv
(32 V)
147x147x32
3x3 Conv
(64)
147x147x64
3x3 MaxPool
(stride 2 V)
73x73x64
1x1 Conv
(80)
73x73x80
3x3 Conv
(192 V)
71x71x192
3x3 Conv
(256 stride 2 V)
35x35x256







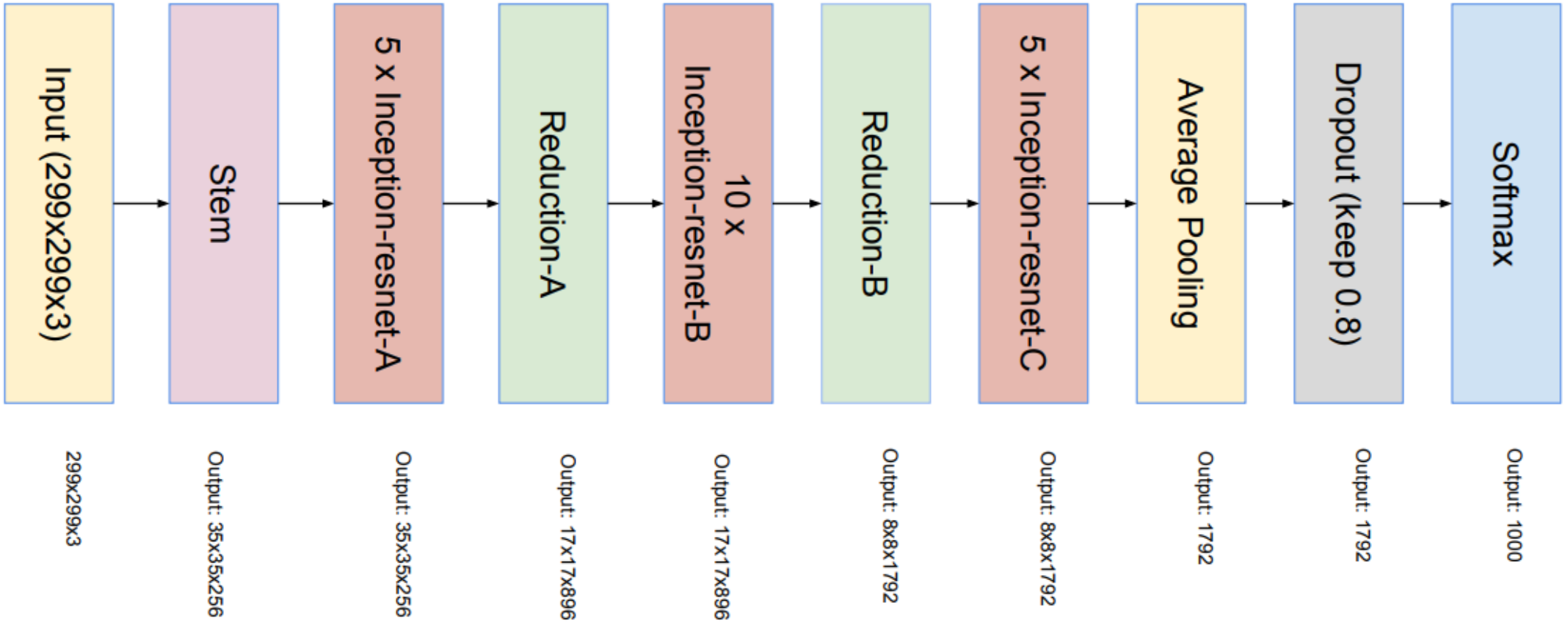
Inception Modules A, B, C of Inception-v4



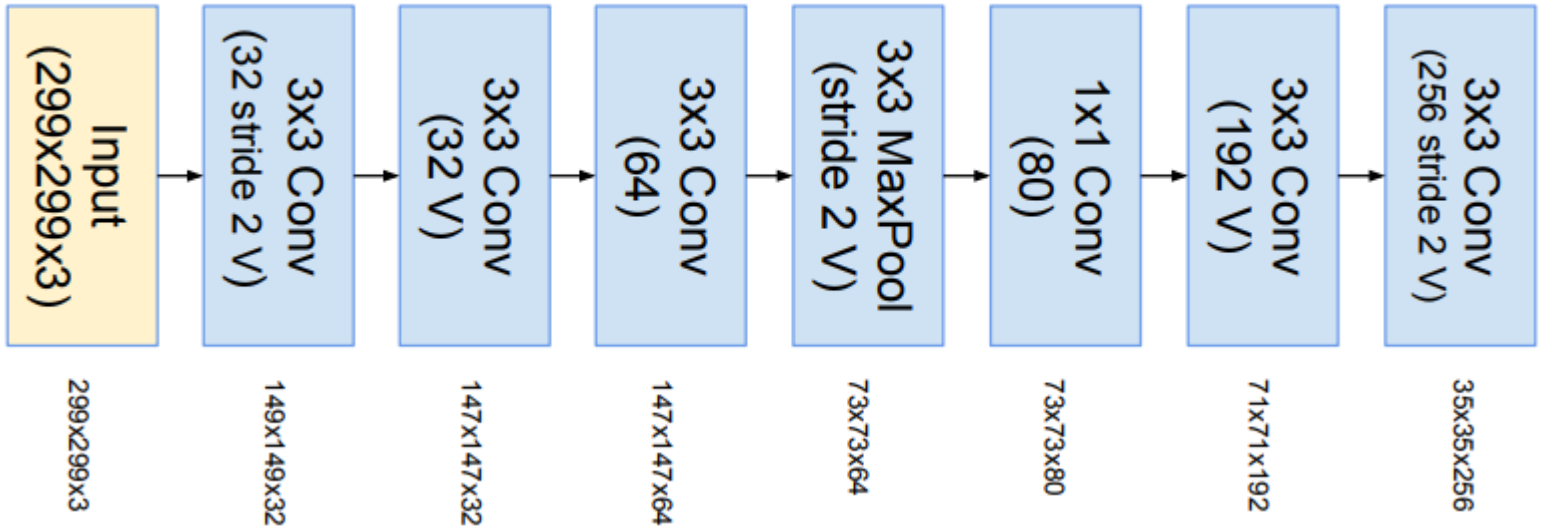
Reduction Blocks A, B of Inception-v4

Inception Resnet

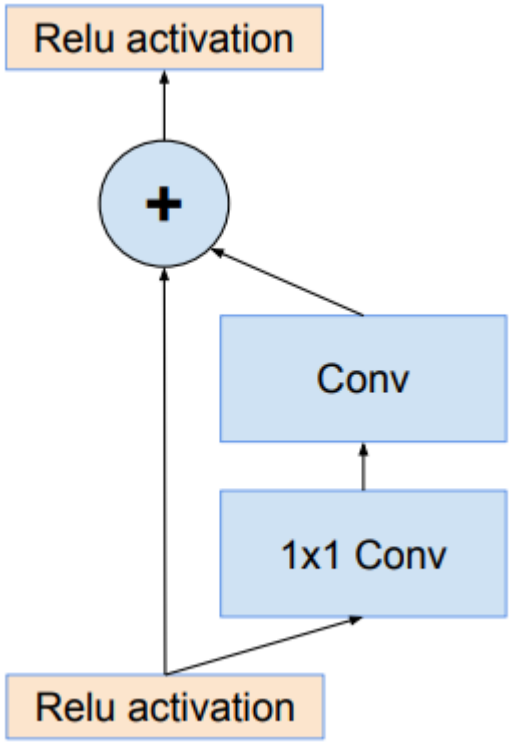
- Uses split transform merge idea and residual links

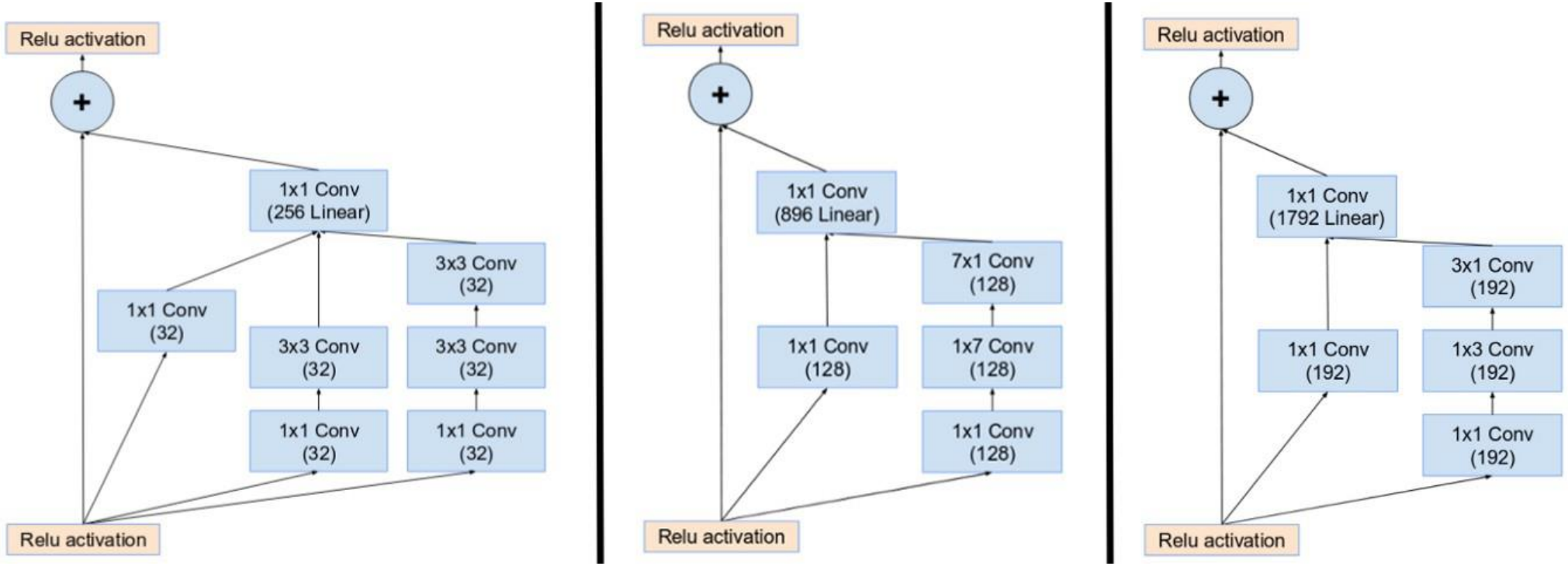


STEM

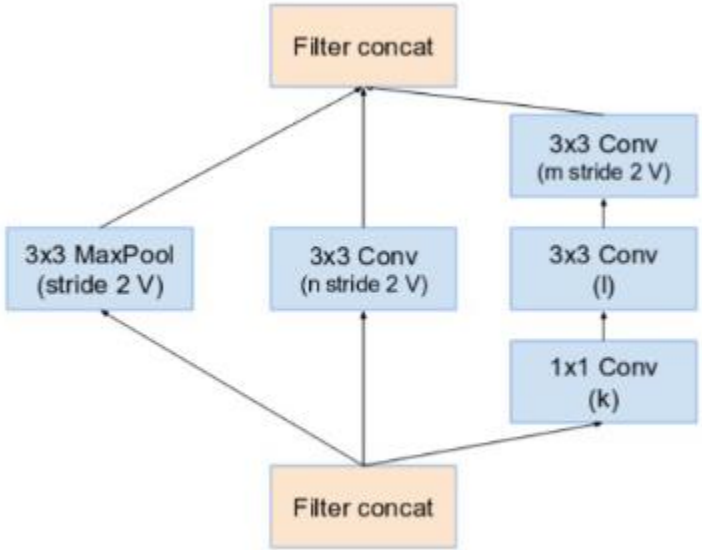


Inception Resnet Conncetion

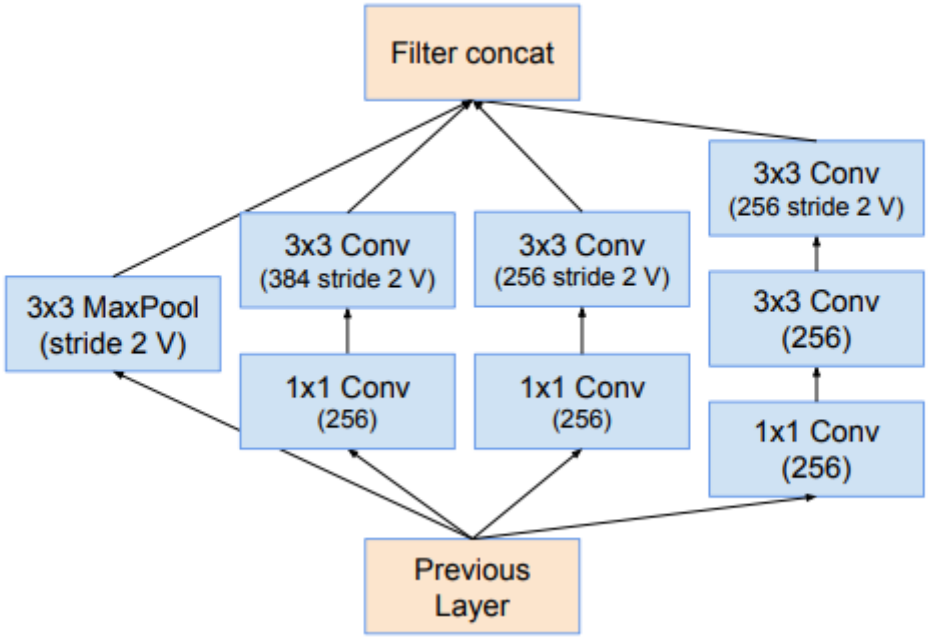




Inception modules A, B, C of Inception ResNet V1



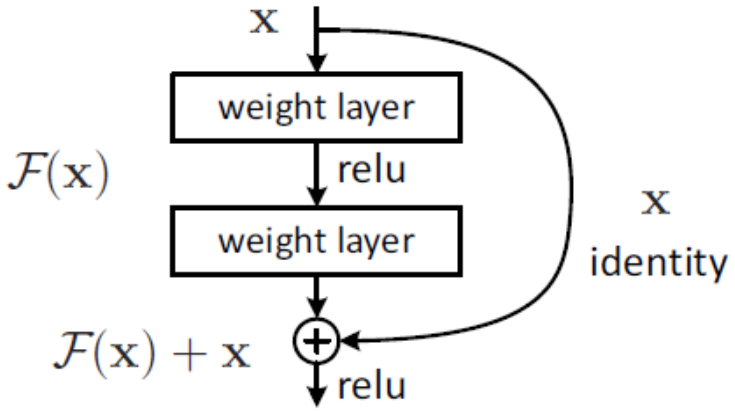
Reduction A schema



Inception ResNet-v1 Reduction Block B

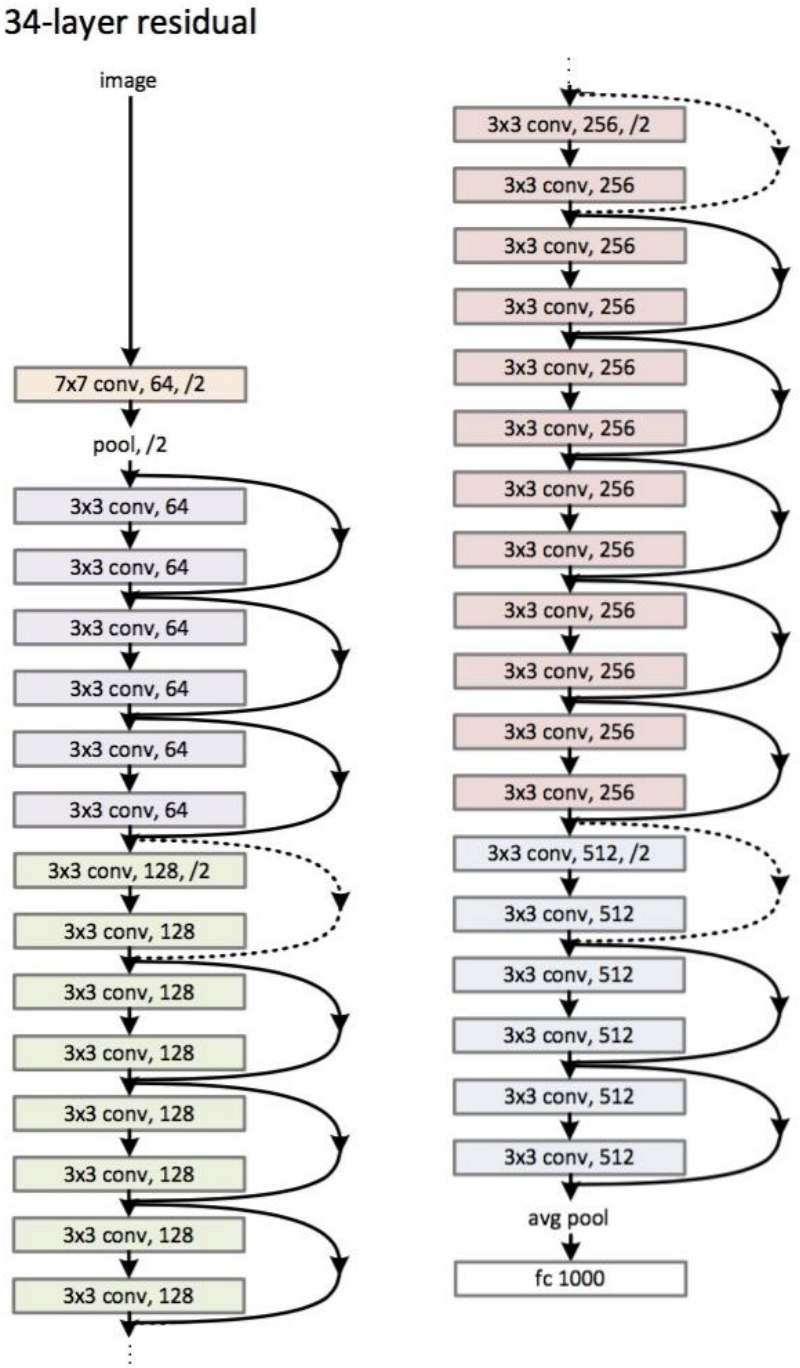
Resnet

- Residual learning
- Identity mapping based skip connections



Residual Blocks

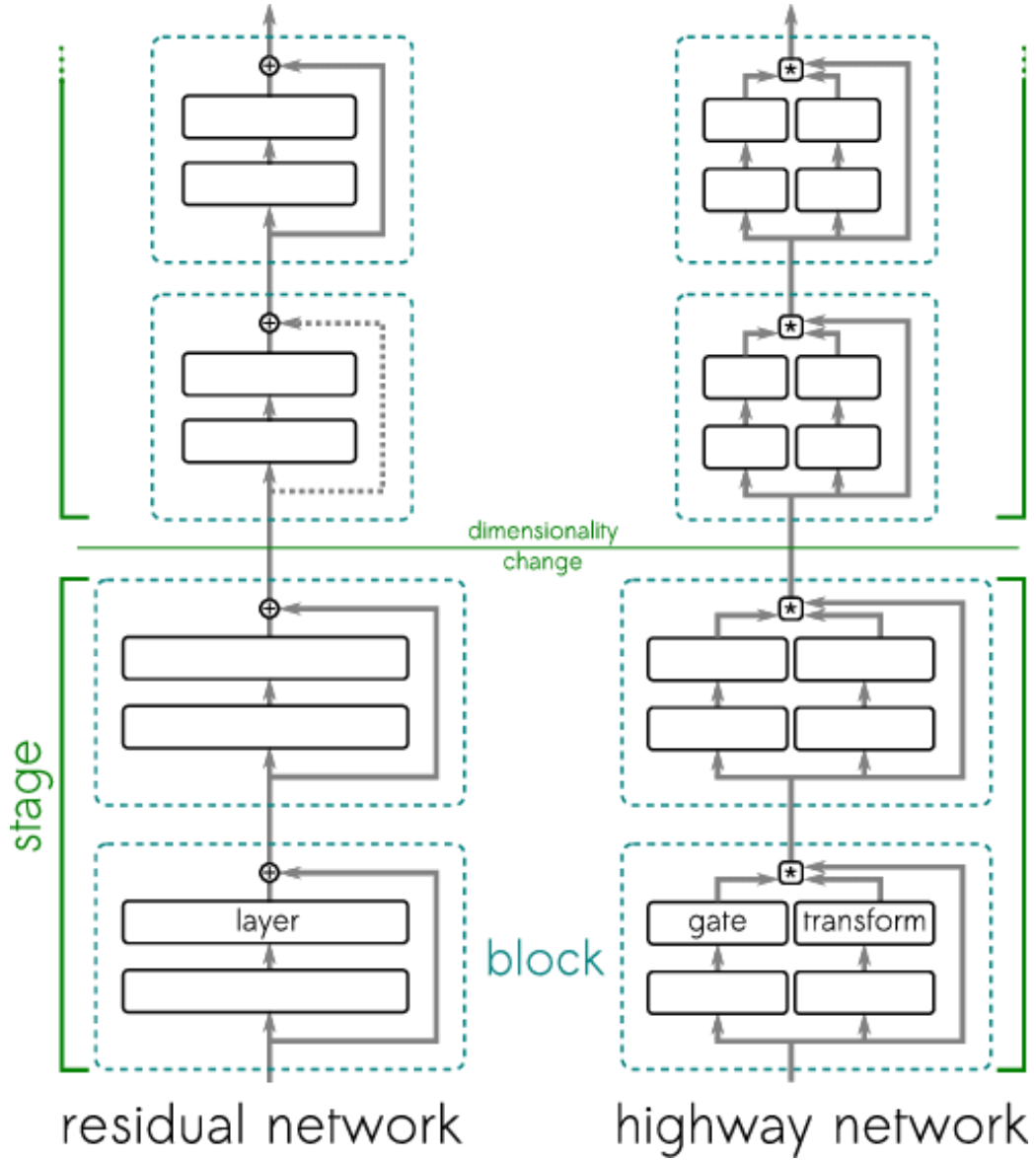
34-layers residual



Highway Network*

- Introduced an idea of Multi-path

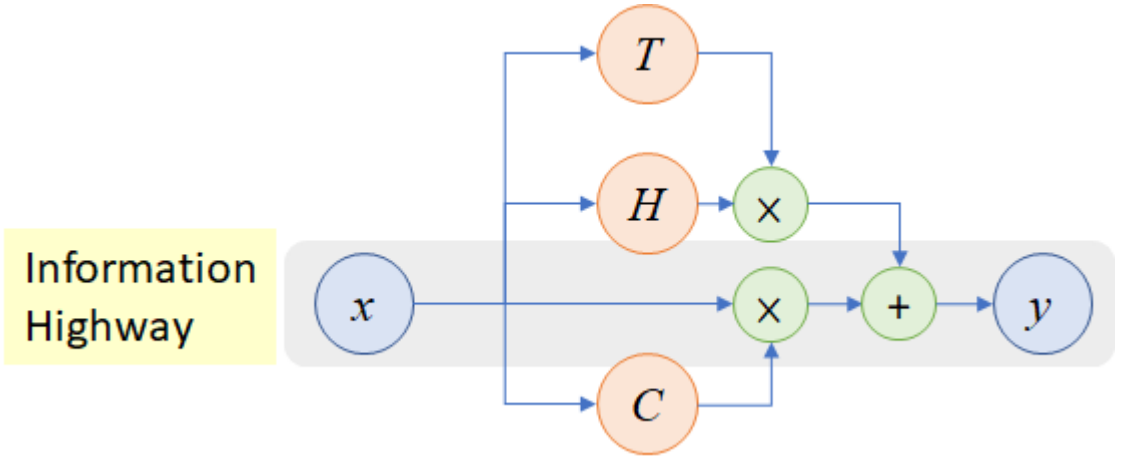
- In machine learning, a highway network is **an approach to optimizing networks and increasing their depth**.
- Highway networks use learned gating mechanisms to regulate information flow, inspired by Long Short-Term Memory (LSTM) recurrent neural networks.
- Highway networks have been used as part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



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

Continuing about High way networks

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



The structure of a hidden layer follows the equation:

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

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

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

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

(3) Multi-Path based CNNs

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

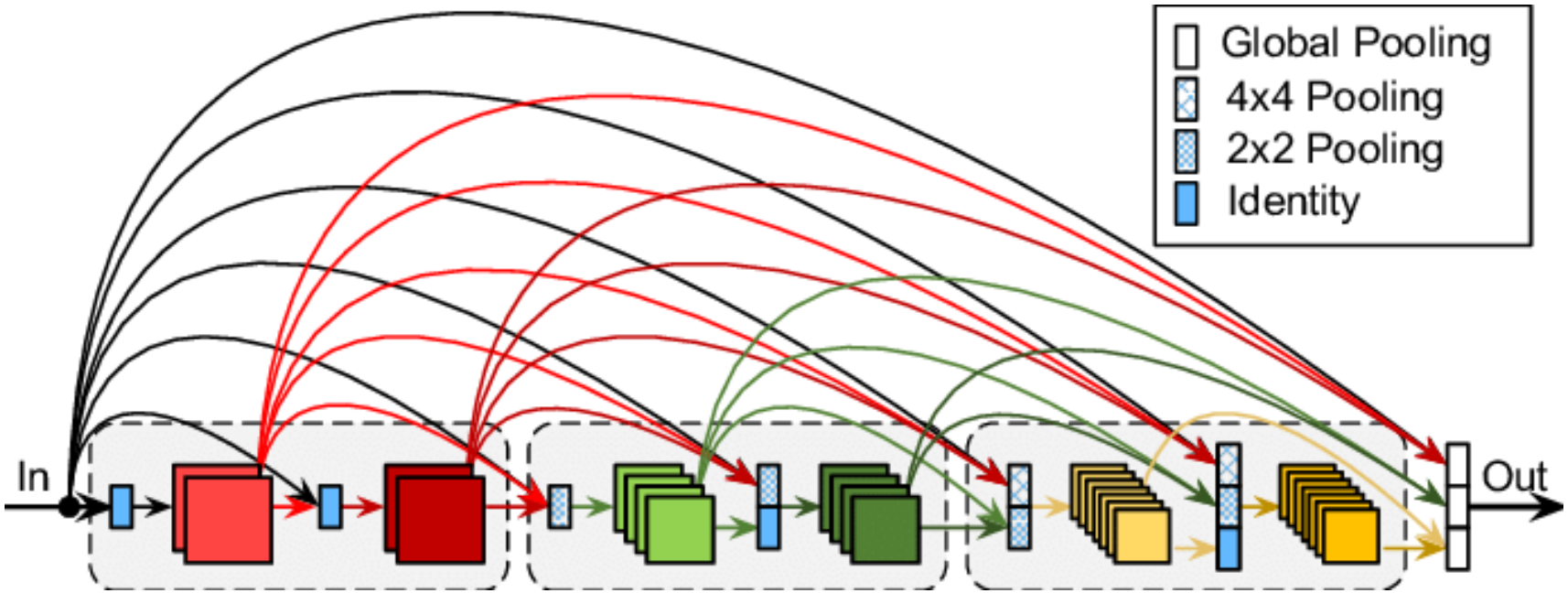
CNNs: Highway Networks ResNet DenseNet

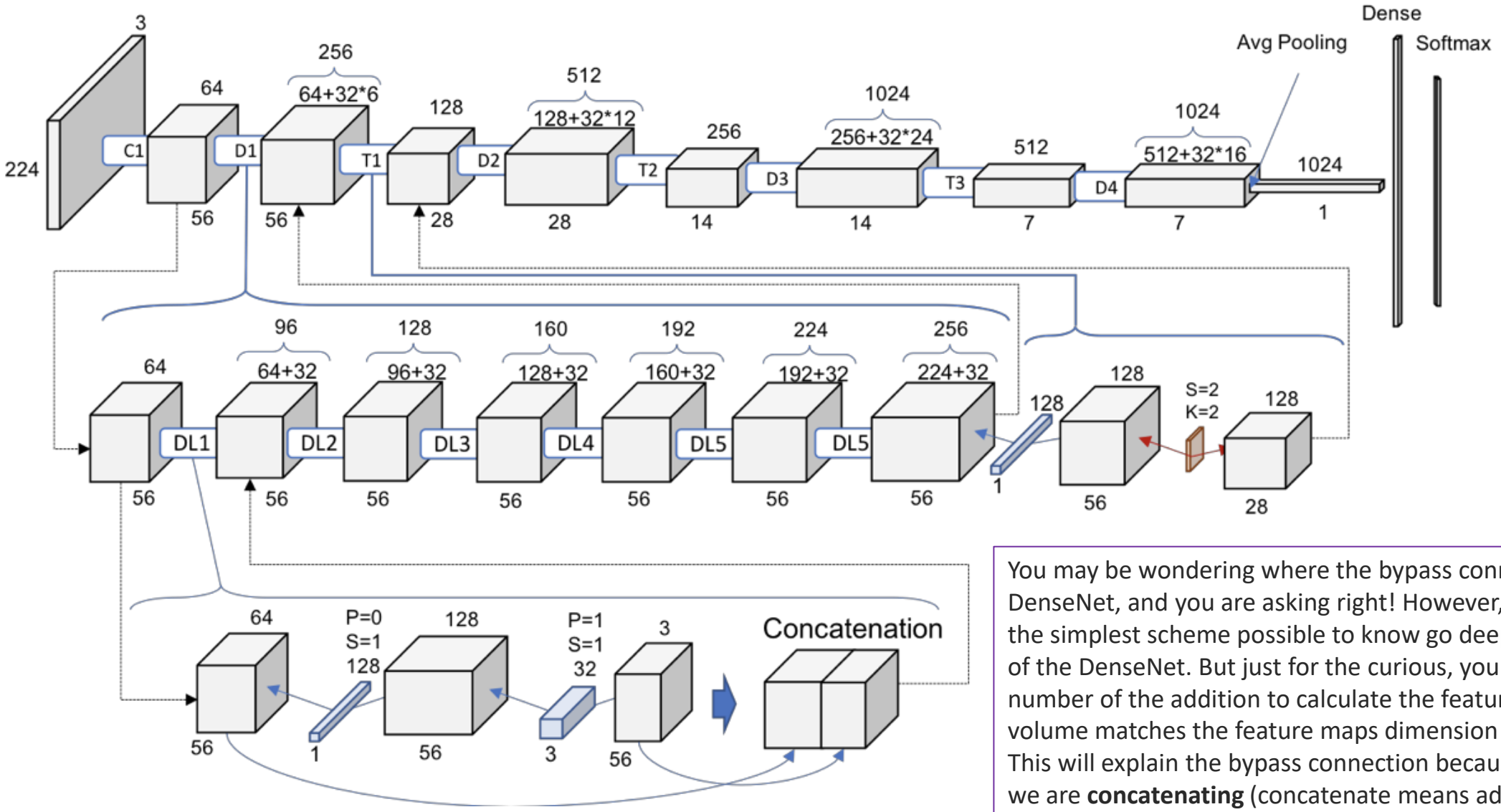
Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76	19	Depth + Multi-Path	(Srivastava et al. 2015a)
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M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

DenseNet

- Cross-layer information flow

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





Full schematic representation of ResNet-121

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

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

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

Width based Multi-Connection CNNs

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

CNNs: Wide ResNet Pyramidal Net Xception ResNeXt Inception Family

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

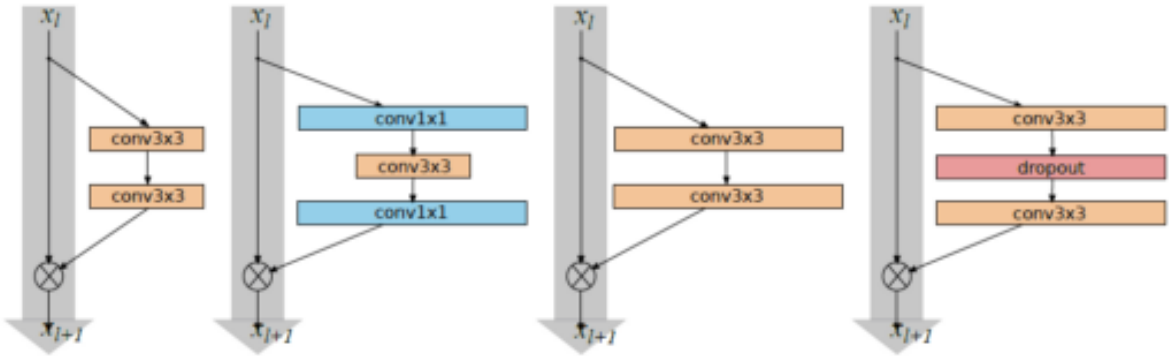
Wide ResNet

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

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

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

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

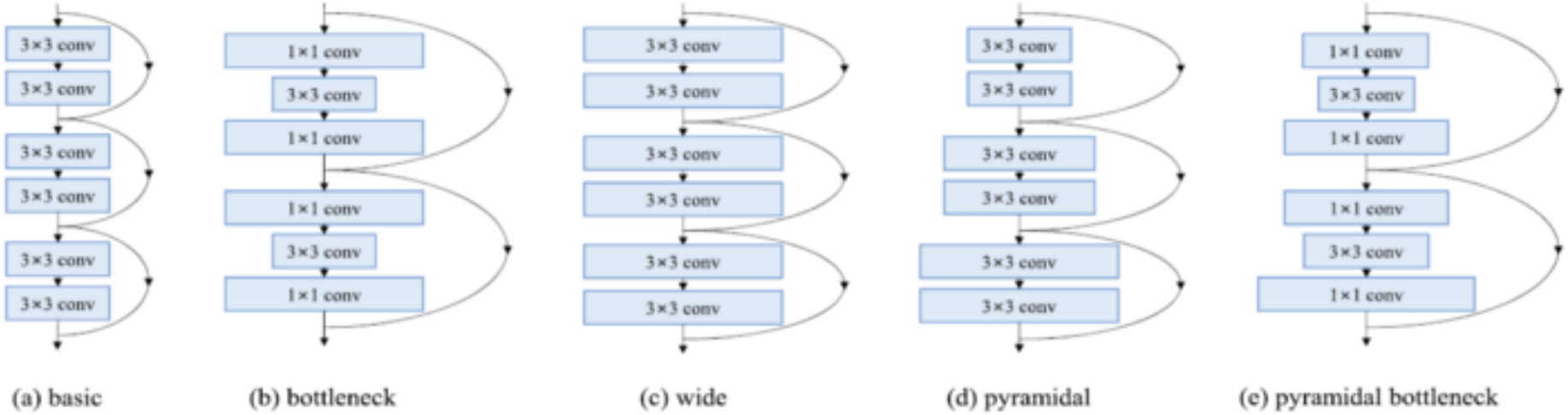


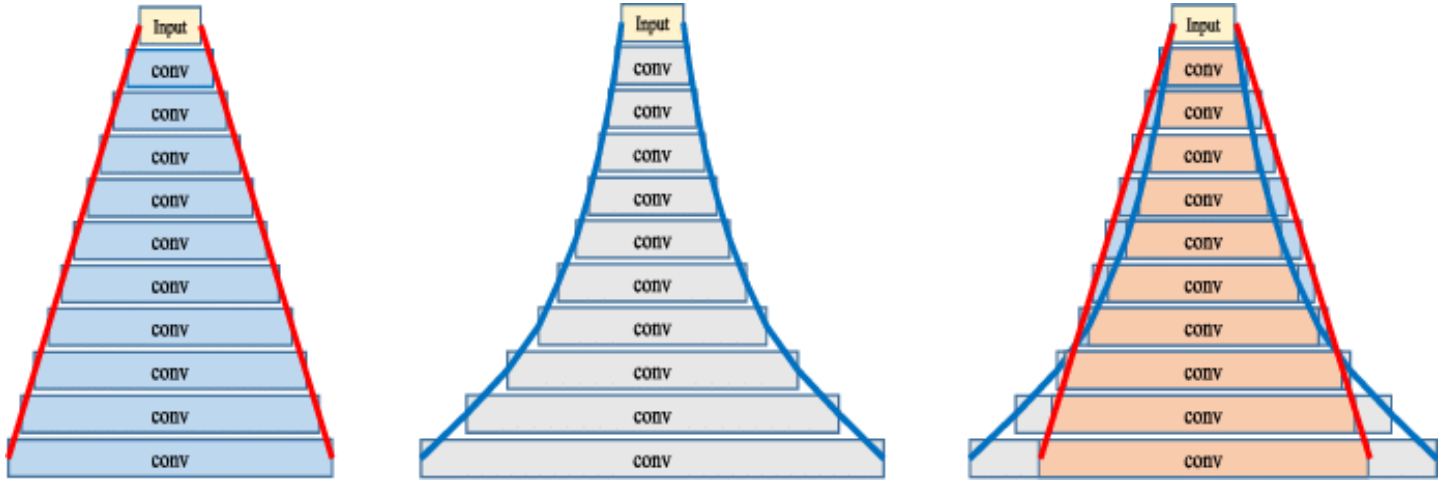
(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

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

Pyramidal Net

Pyramidal Net increases the width gradually per residual unit.





Xception

- Depth wise convolution followed by point wise convolution

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

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

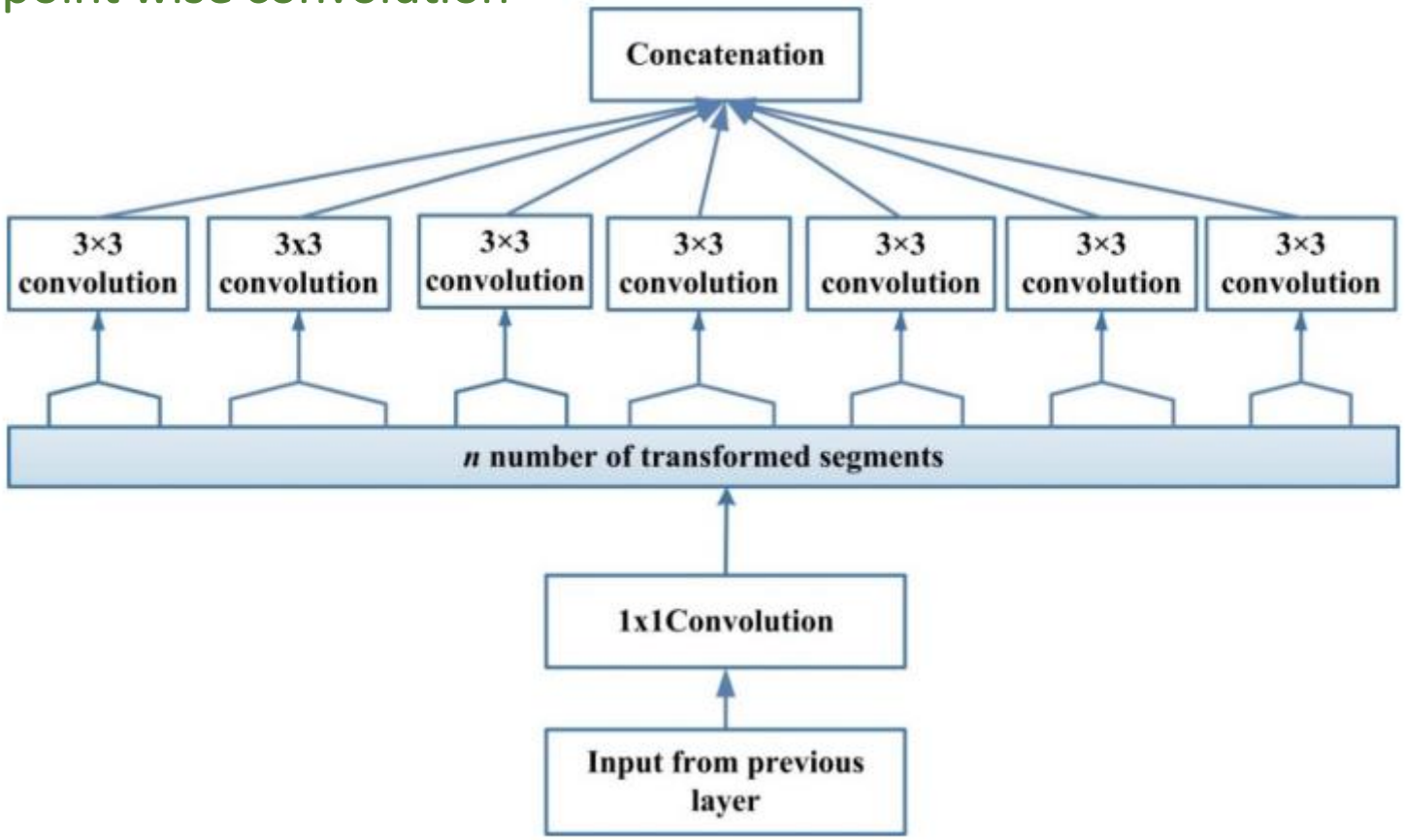


Fig. 8 Xception building block and its *n* sets of transformation.

ResNeXt

-Cardinality
- Homogeneous topology
- Grouped convolution

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

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

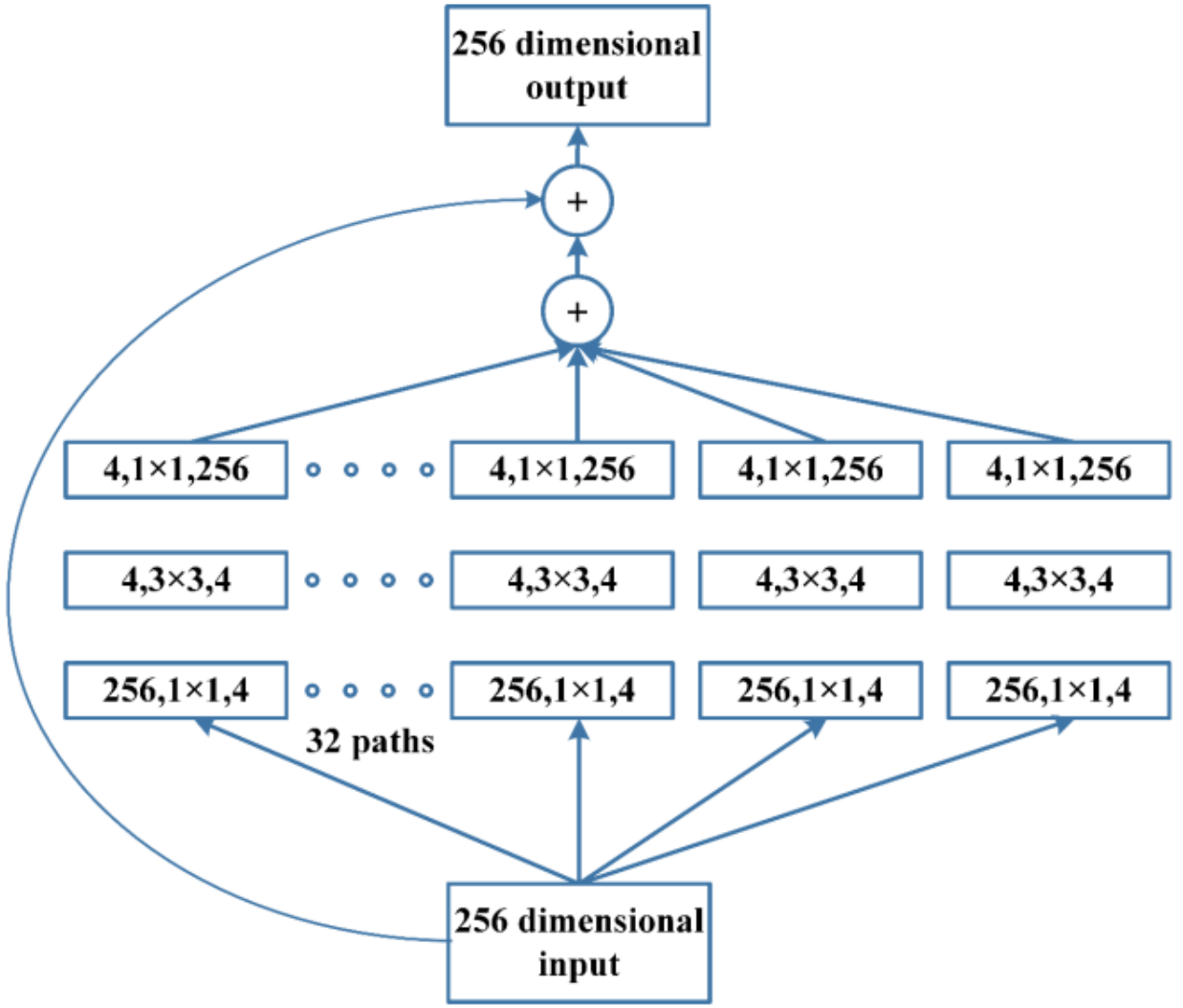


Fig. 9 ResNeXt building block showing the different paths of transformation.

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

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

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

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

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

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

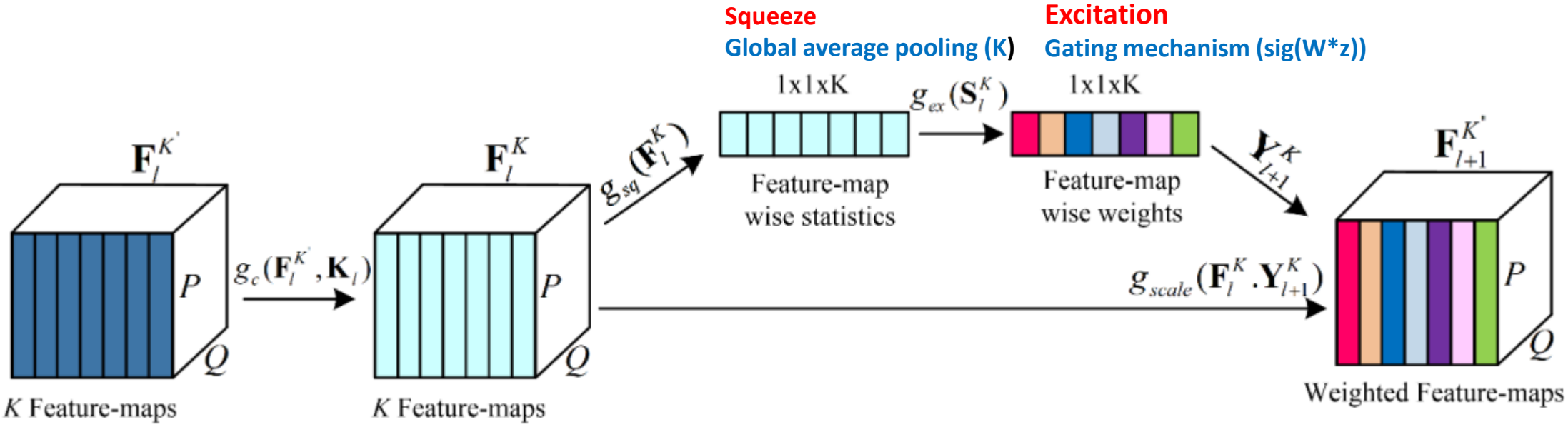
(5) Squeeze and Excitation Network

Model interdependencies between feature maps

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

maps that are commonly known as channels in literature

Squeeze-and-Excitation Networks (SENets) **introduce a building block for CNNs that improves channel interdependencies at almost no computational cost**. ...
Let's add parameters to each channel of a convolutional block so that the network can adaptively adjust the weighting of each feature map



Competitive Squeeze and Excitation Networks

*Hu et al. 2018b

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

Competitive Squeeze-Excitation Architecture for Residual block

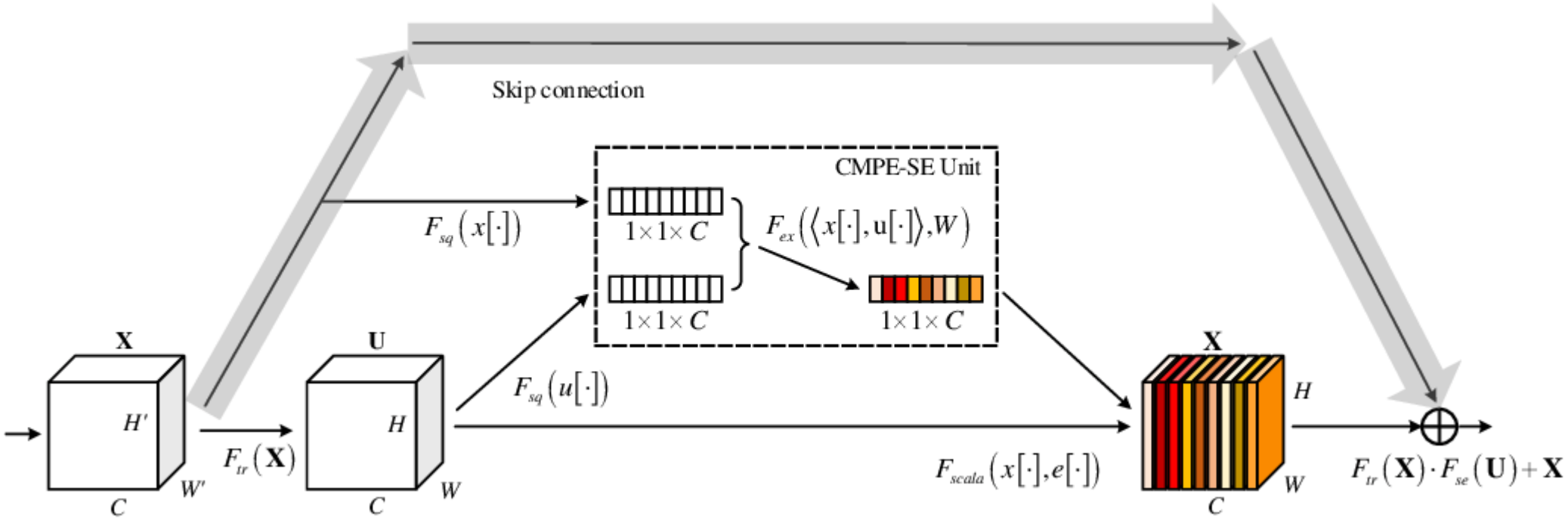


Table 5e Major challenges associated with implementation of Feature-Map exploitation based CNN architectures.

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

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

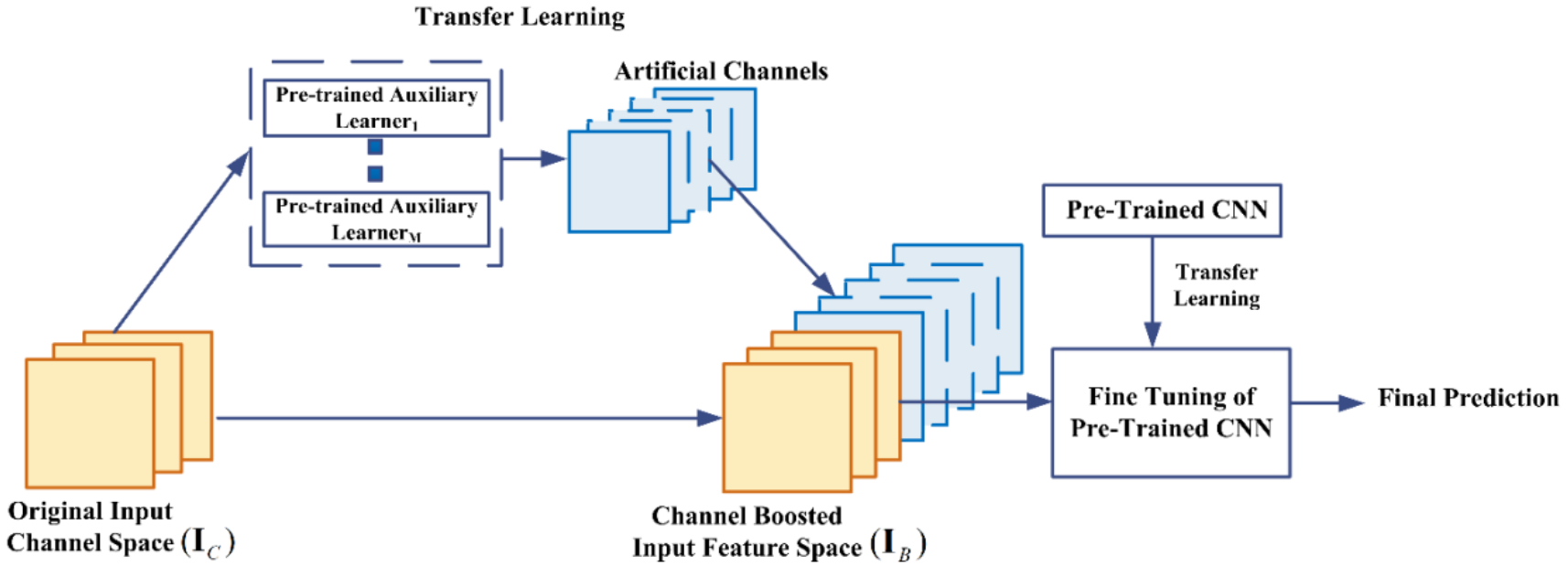
CNNs which improve themselves by using auxiliary learners for channel boosting (input channel dimension)

CNNs: Channel Boosted CNN using TL

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

Channel Boosted CNN using TL (transfer Learning)

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



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

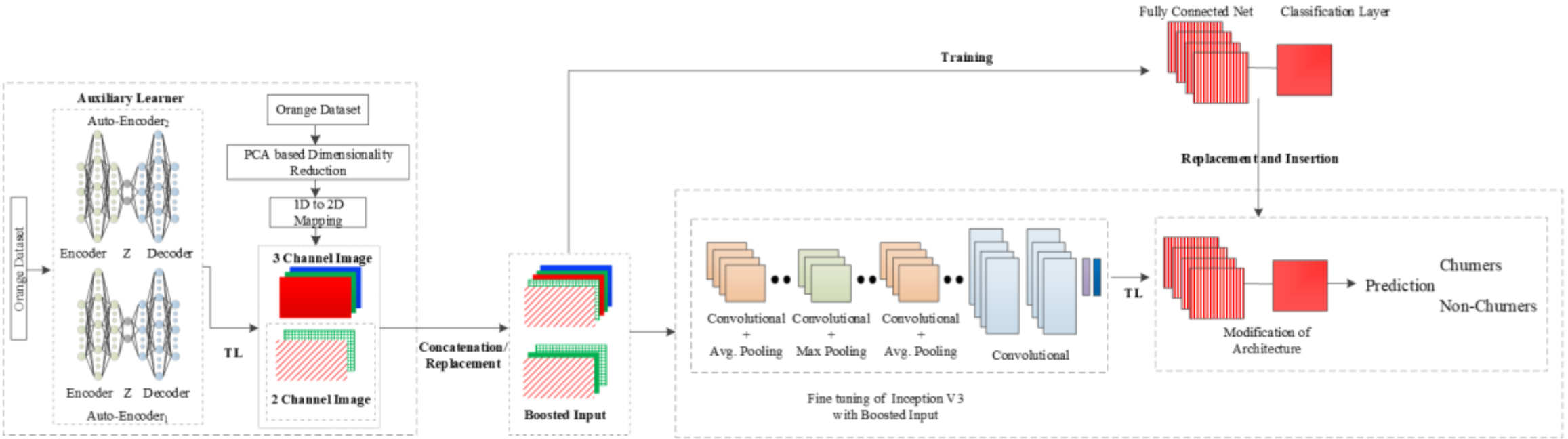


Fig. 4. Details of the working of the proposed CB-CNN. *CB-CNN1* is trained by using Boosted Input$_1$. Boosted Input$_1$ is comprised of three channels that is generated by replacing the input channels of original feature space with two auxiliary channels. Whereas, Boosted Input$_2$ is used to train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

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

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

(7) Attention based CNNs

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

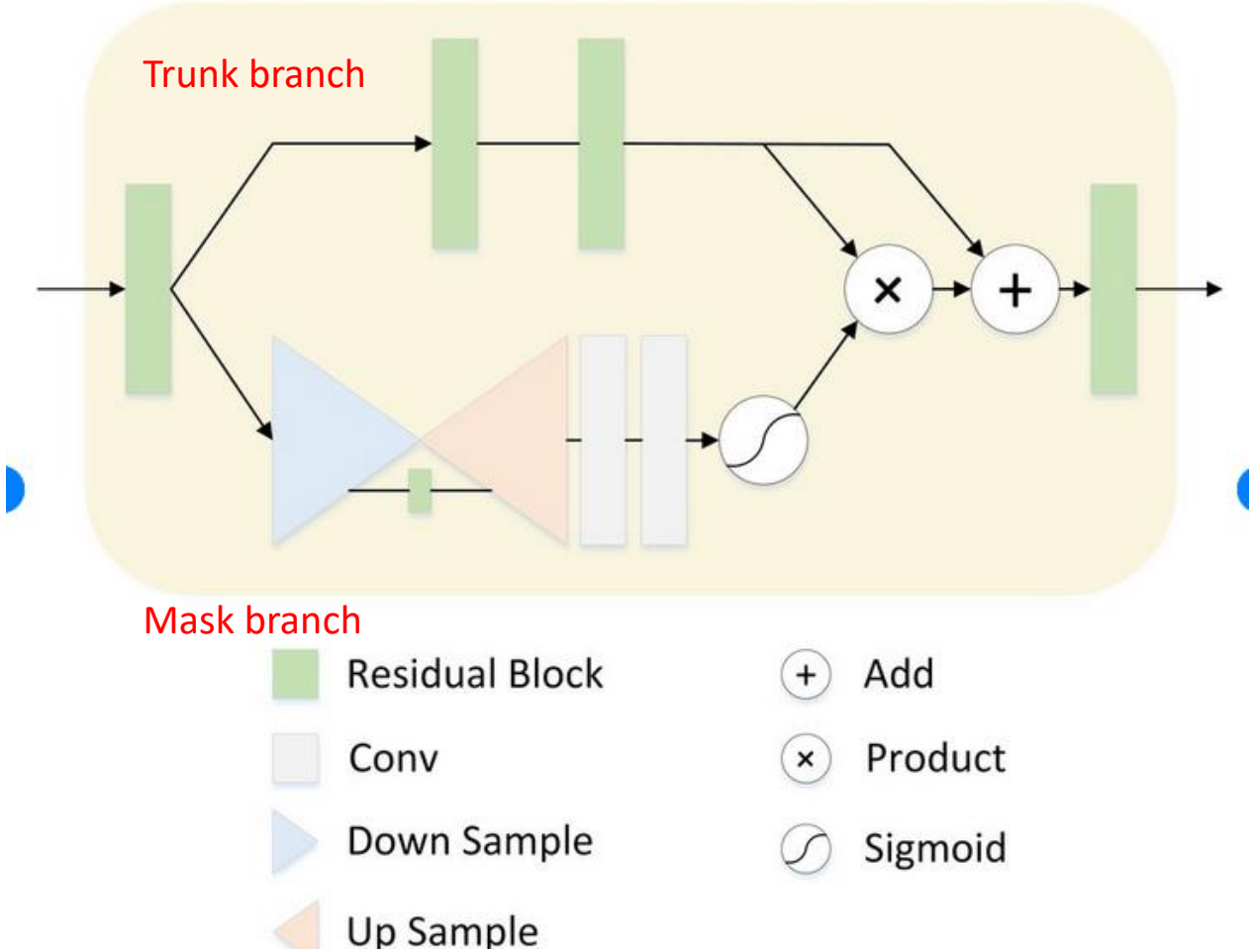
CNNs: Residual Attention Neural Network Convolutional Block Attention Module Concurrent Spatial and Channel Excitation Mechanism

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

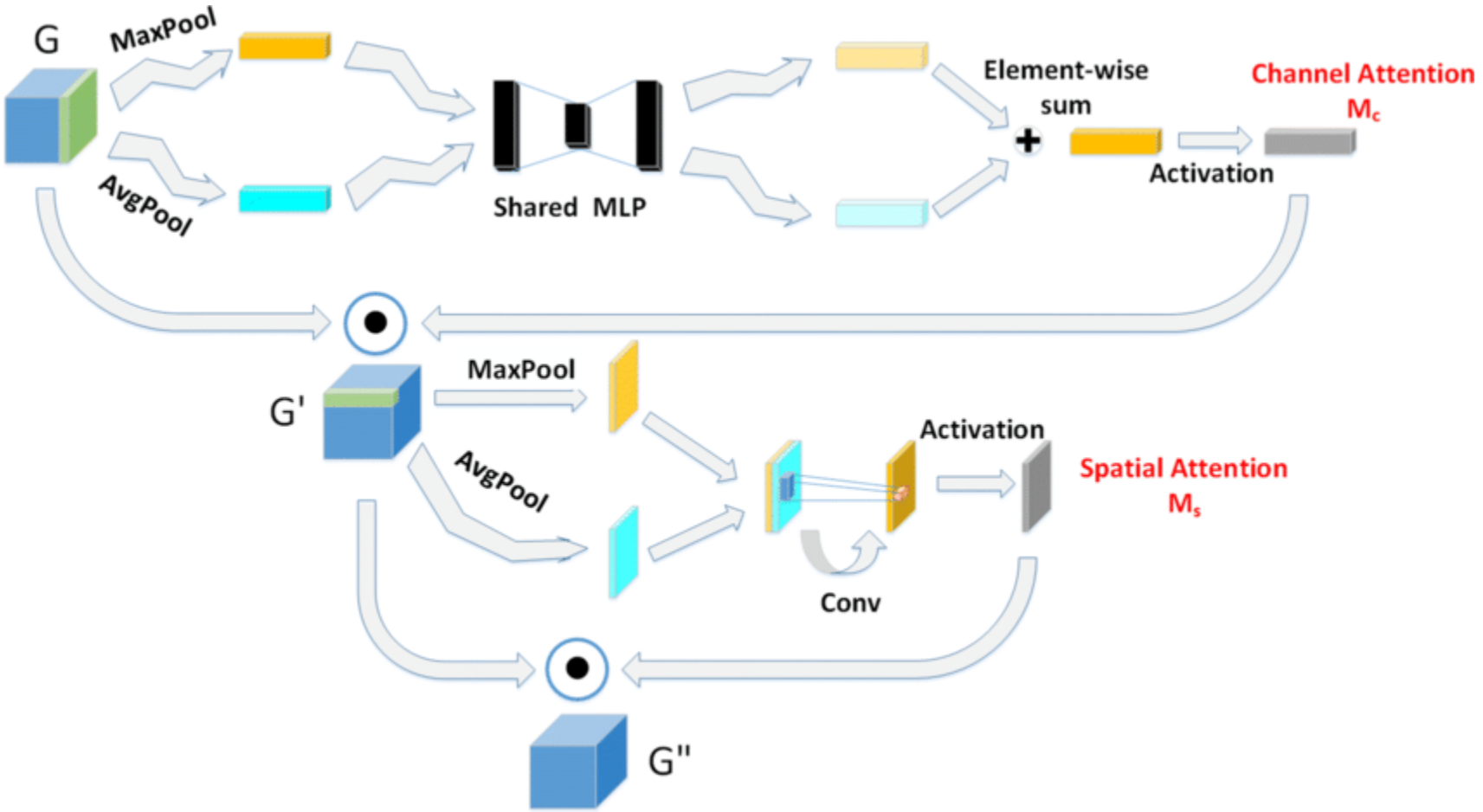
Residual Attention Neural Network*[Sik-Ho Tsang](#) Apr 11, 2019

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

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

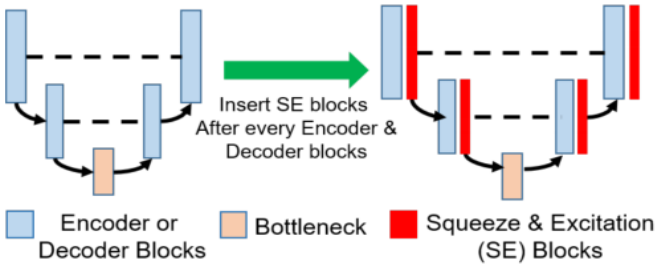


CBAM: Convolutional Block Attention Module

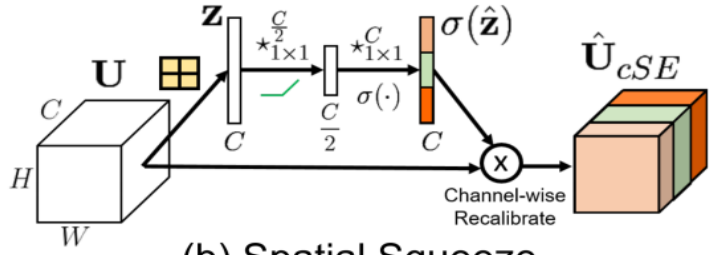


Concurrent Spatial and Channel Excitation Mechanism

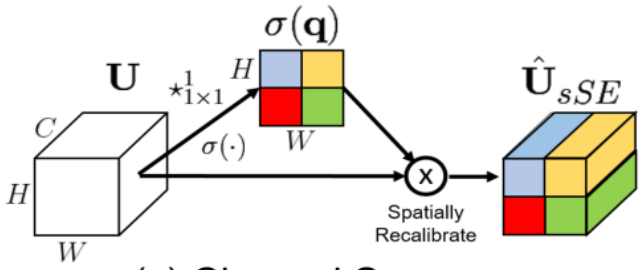
F-CNN: Fully Convolutional NN.



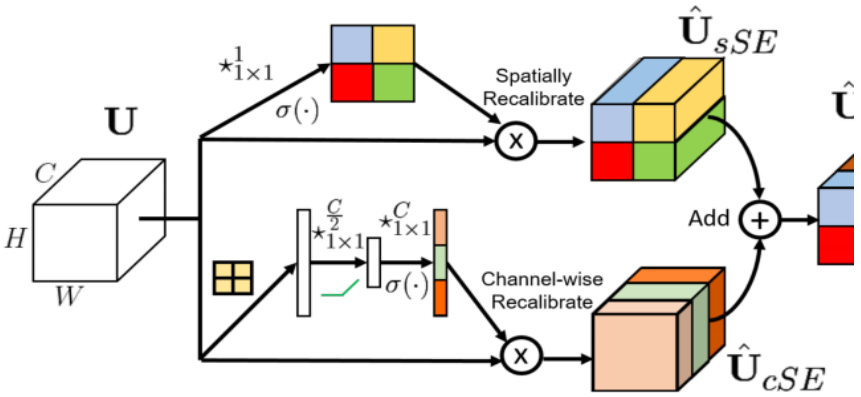
(a) SE block in F-CNN



(b) Spatial Squeeze and Channel Excitation (cSE)



(c) Channel Squeeze and Spatial Excitation (sSE)



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

$\star^{p}_{m\times n}$ Convolution with m x n kernel p channels

ReLU Global Pooling $\sigma(\cdot)$ Sigmoid

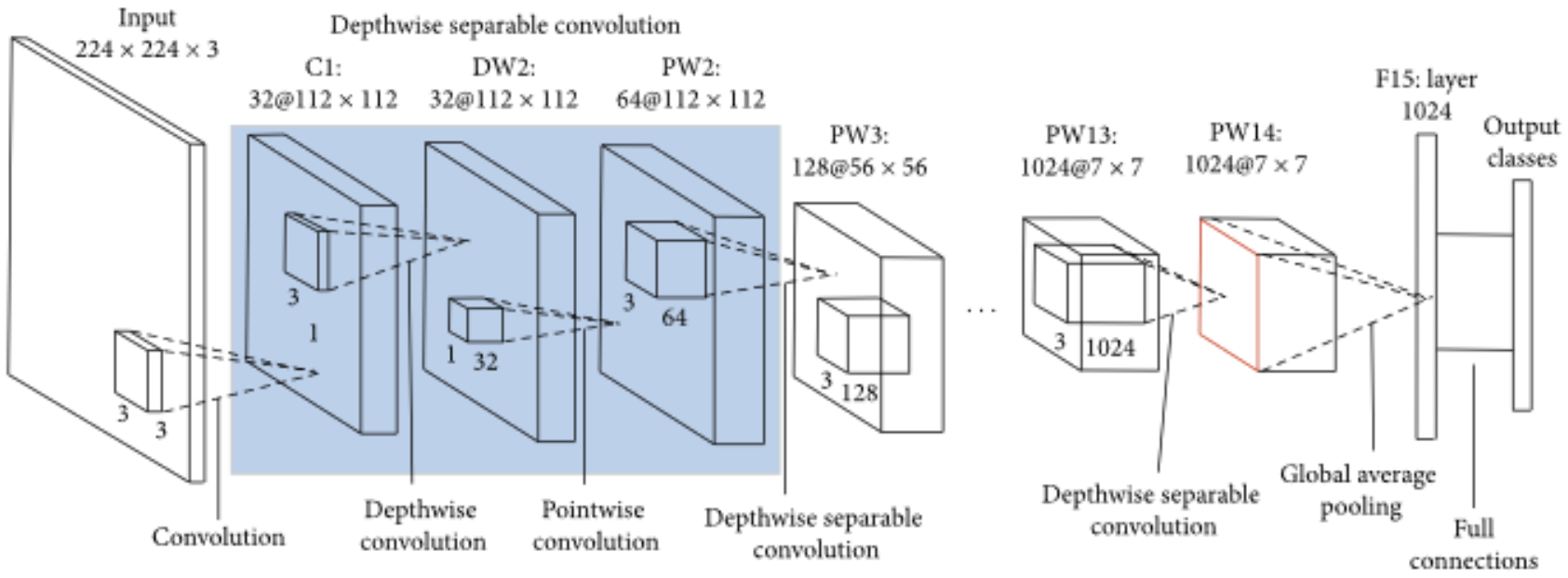
Table 5g Major challenges associated with implementation of Attention based CNN architectures.

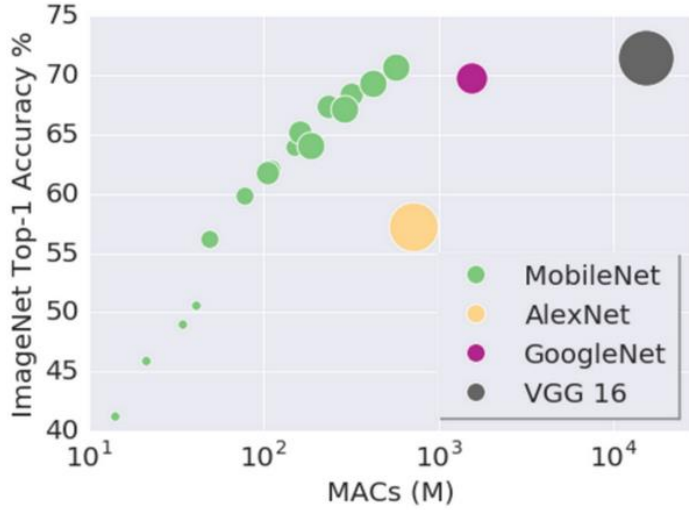
Attention	Attention Networks advantages to choose which patch is the area of the focus or most important in an image	
Architecture	**Strength**	**Gaps**
Residual Attention Neural Network	• Generates attention aware feature-maps • Easy to scale up due to residual learning • Provides different representations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



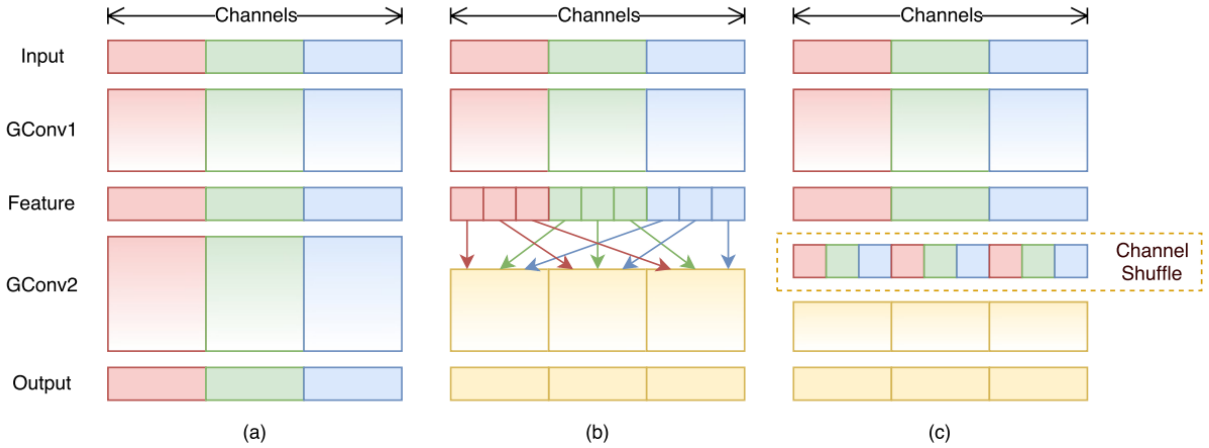


The speed and power consumption of the network is proportional to the number of MACs (Multiply-Accumulates) which is a measure of the number of fused Multiplication and Addition operation

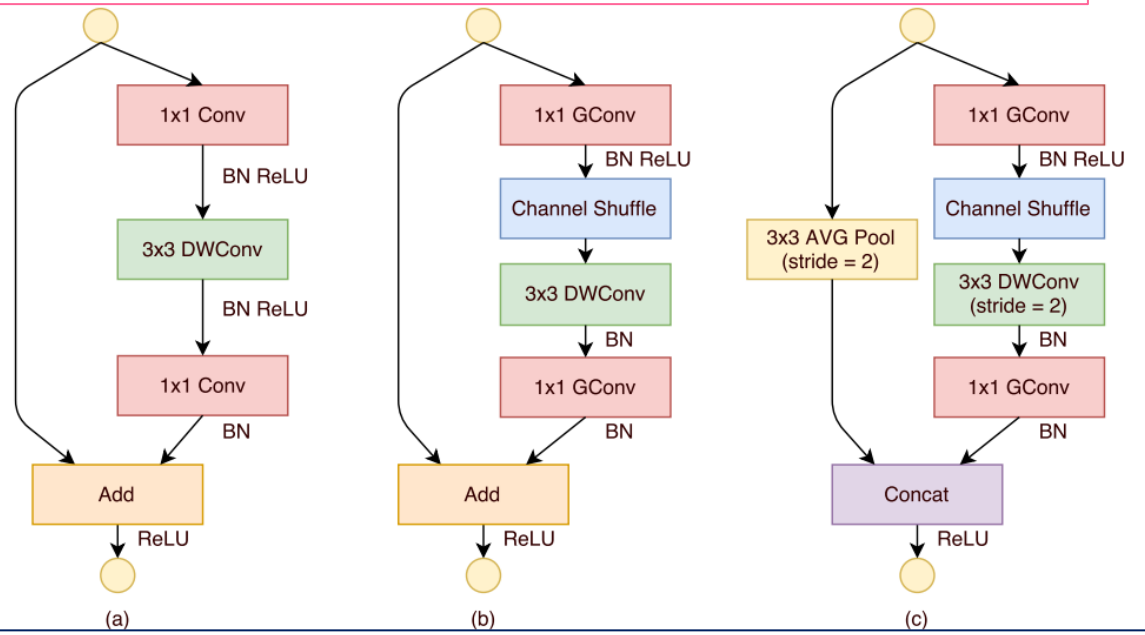
ShuffleNet

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**.

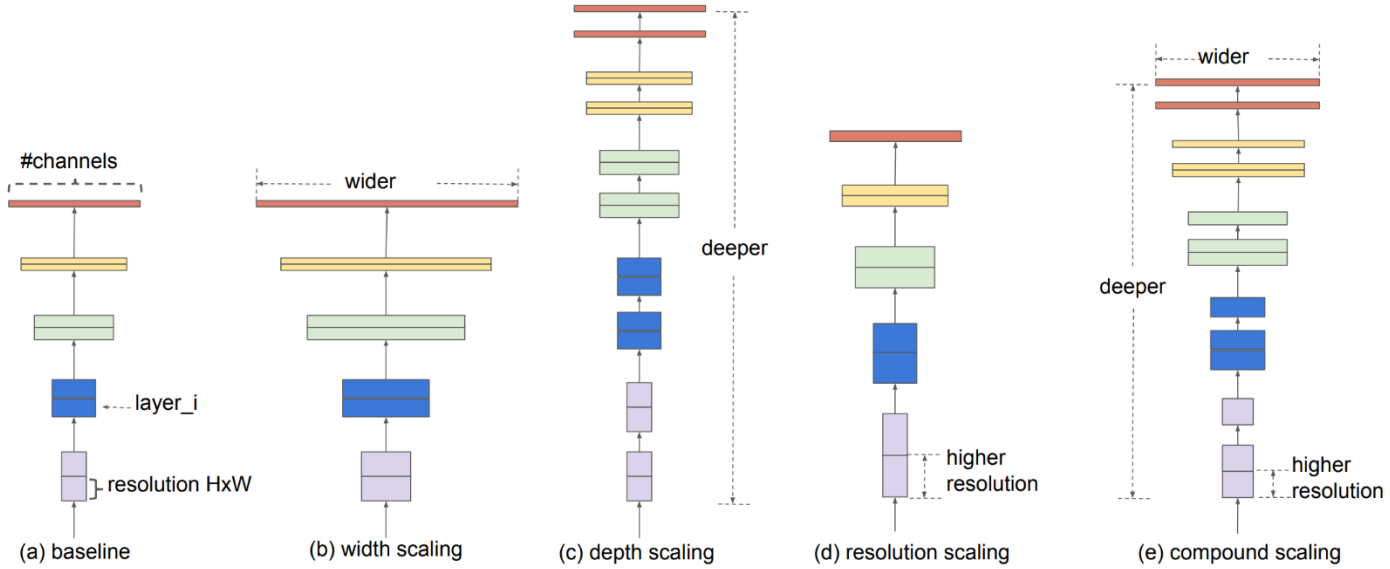
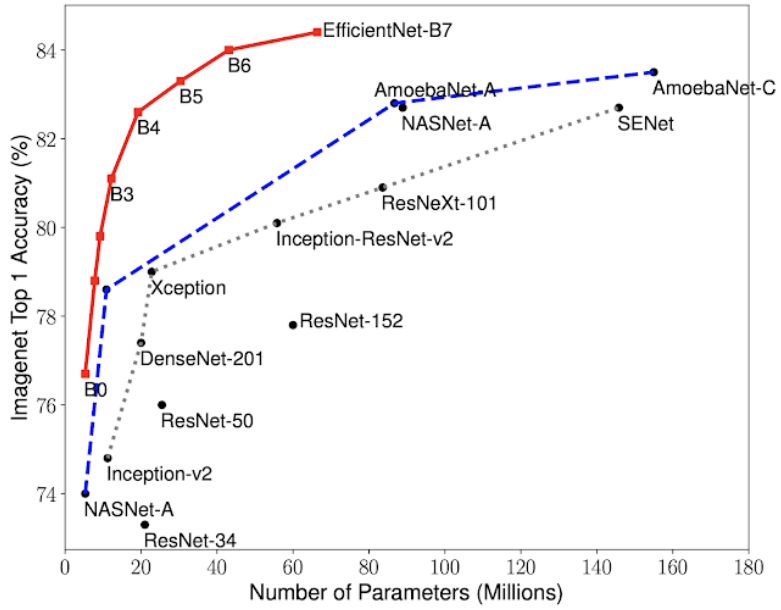


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.



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	-	-

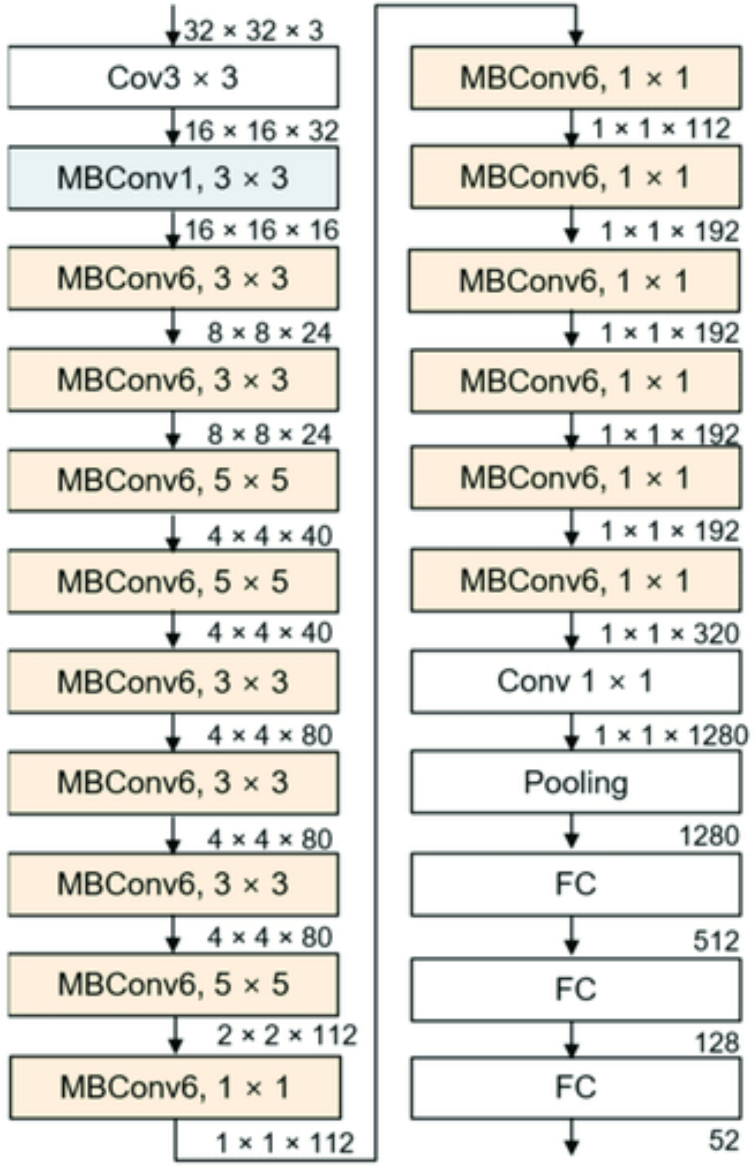
We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

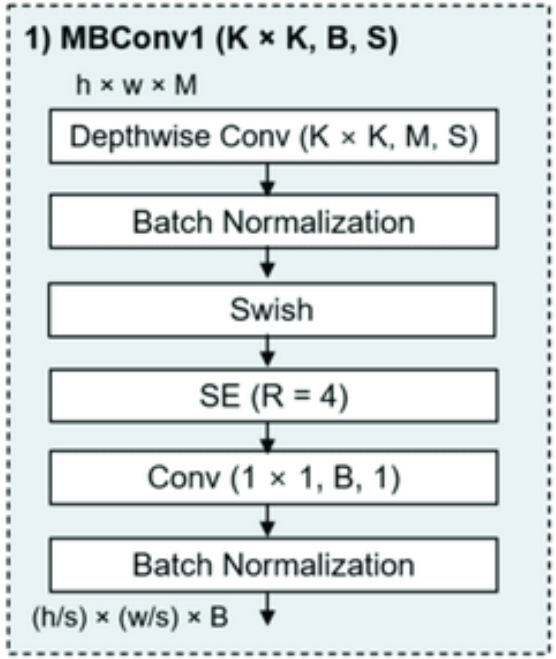
FLOPS = **floating point operations per second**

EfficientNetB0

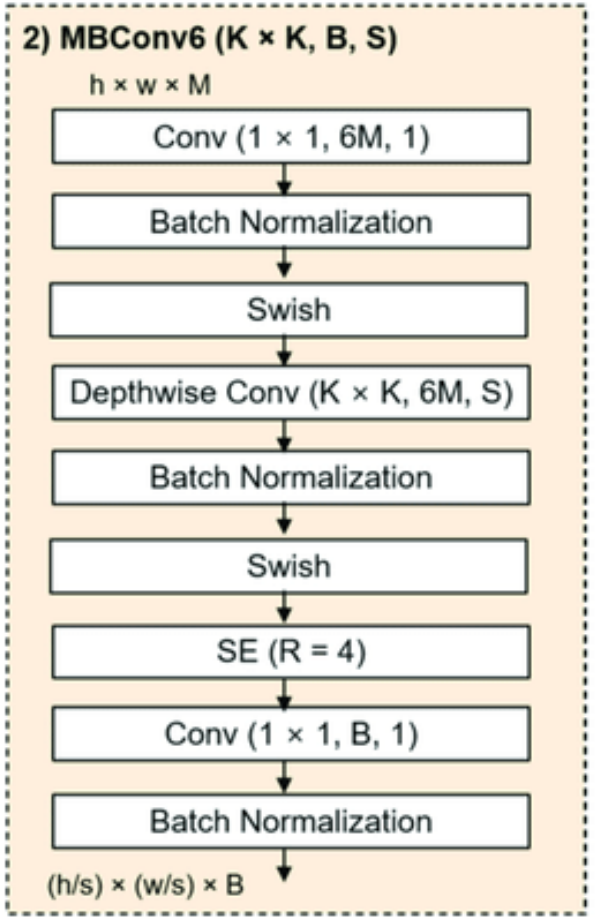
The structure of an EfficientNetB0 model with the internal structure of MBConv1 and MBConv6. Compared to MBConv1, MBConv6 has three layers at the top. The number of feature maps as the output is 6.

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



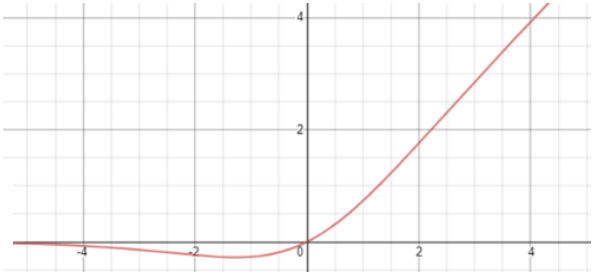


K : kernel size
M : Input Feature maps
B : Output Feature maps
S : Stride
R : Reduction ration of SE



Formally stated, the Swish activation function is…

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



3.2. Layer-wise Analysis Algorithm

The concept of dataflow or information-flow

- By applying the input data, $\boldsymbol{x}$ ($\boldsymbol{x} \equiv \boldsymbol{x}^1$) to a deep neural network with N_L layers, the data flows layer by layer.
- Dataflow (information-flow) denotes the data which transforms layer by layer within a deep neural network :

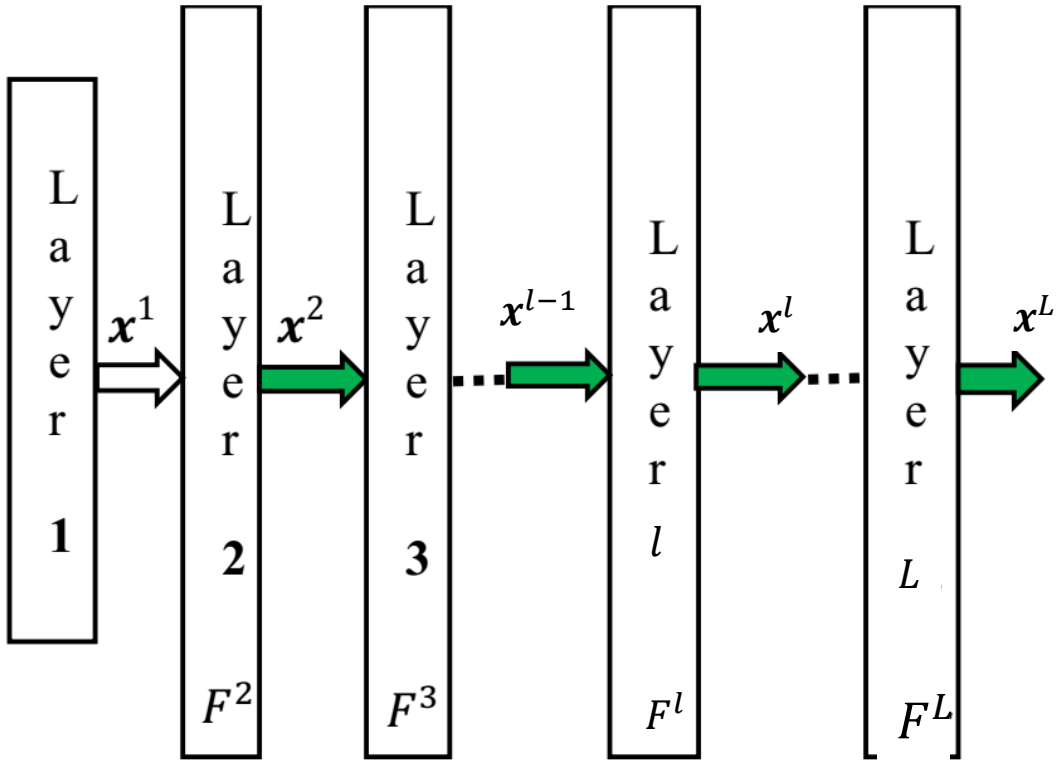
$$\boldsymbol{x}^1 \rightarrow \cdots \boldsymbol{x}^{l-1} \rightarrow \boldsymbol{x}^l \rightarrow \cdots \rightarrow x^L$$

- L denotes the number of layers in the model
- $\boldsymbol{x}^l$ denotes the dataflow at layer l, which is reshaped as a vector and its length is n_L.

$$\boldsymbol{x}^l \boldsymbol{\epsilon} \boldsymbol{R}^{\boldsymbol{n_l} \times \boldsymbol{1}}$$

- One can compute SI(SmI) for dataflow at layer $\boldsymbol{l}$:

$Data^l = \{(\boldsymbol{x}_i^l, l_i)\}_{i=1}^m$ $\quad\quad$ $\boldsymbol{l}$=1,2, … , L



It seems the above DNN is a feedforward network.
However, each layer can be a RNN or a LSTM module, too.
In the case of RNN or LSTM, it is assumed that the hidden state is within the layer.

The complexity measure SI (SmI) in DNNs

Some definitions

- Disturbance: non relevant information which disturb the feature space
- Distortion: Uncertainties due to (1) the different forms of appearances of the features, (2)the environment constraints on the measuring process, and (3) different measuring parameters.
- Common features: In classification problems, features which are common between examples of different classes they avoid discrimination between different classes.
- Exclusive features: Features which discriminate different classes in a classification problem or maximize smoothness in a regression problem.

Two important notes:

1. In a DNN, it is expected that after a certain number of filter layers, a feature space with negligible disturbance, distortion, and common features is appeared.
2. Using a metric formulation, it is proved that by intensifying exclusive features and weakening disturbances, distortions, and common features (complexity), the SI(SmI) of dataflow will increase gradually through layers of a DNN.

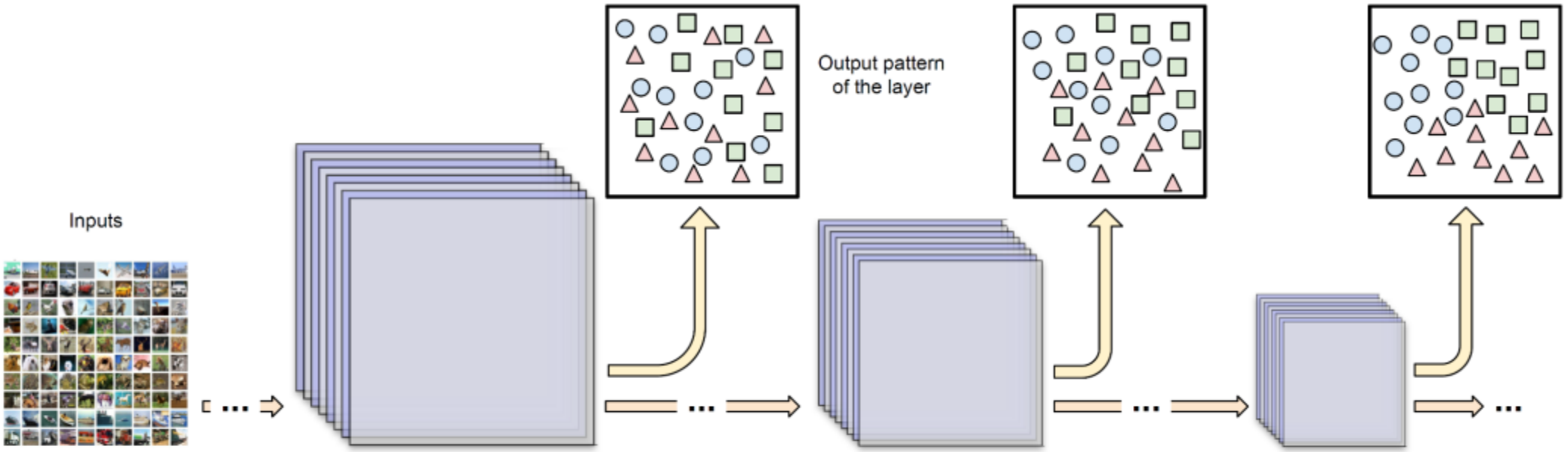
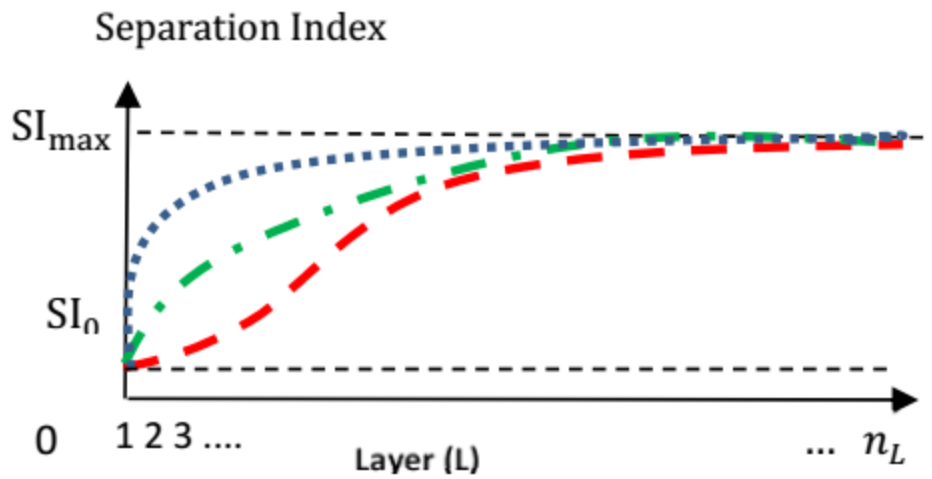


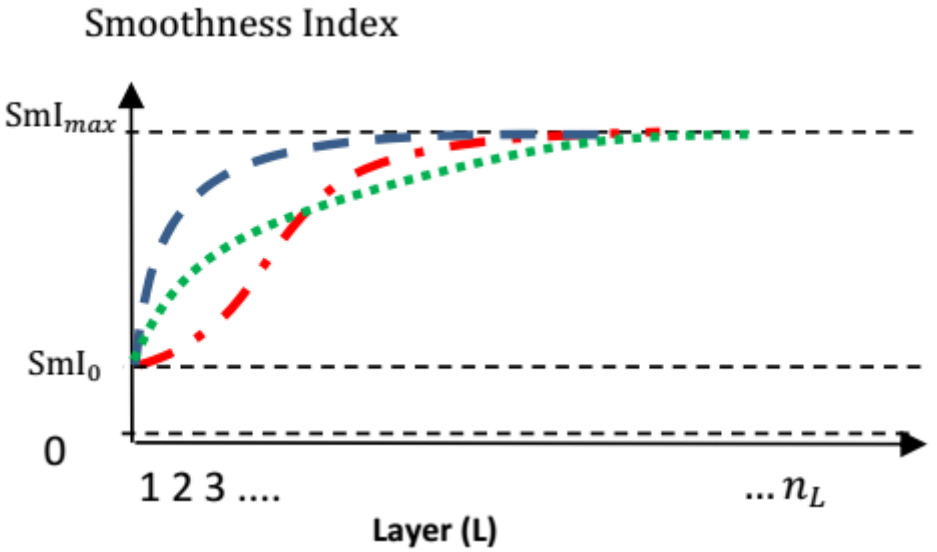
Figure 1. Output pattern of each layer through the network.

It is expected that the complexity of data decreases layer by layer in a deep neural network.

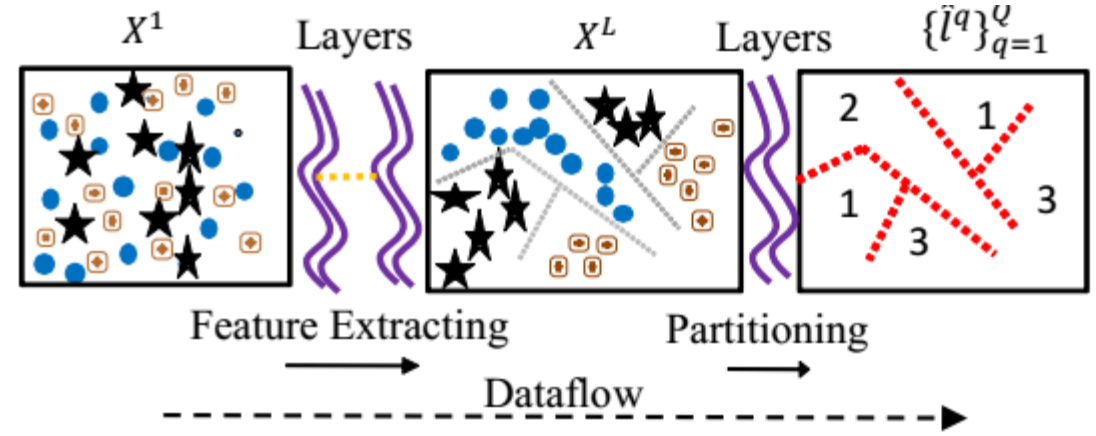
Some important notes

1. In fact, by decreasing the complexity of dataflow through “filter” layers, the SI(SmI) will increase.
2. The SI(SmI) may increase by different trends: different rises, different settling, with smooth or oscillatory changes.
3. Increasing SI(SmI) by Fully Connected (FC) layers is not desired. FC layers due to its redundancies make overfitting and avoid generalization.

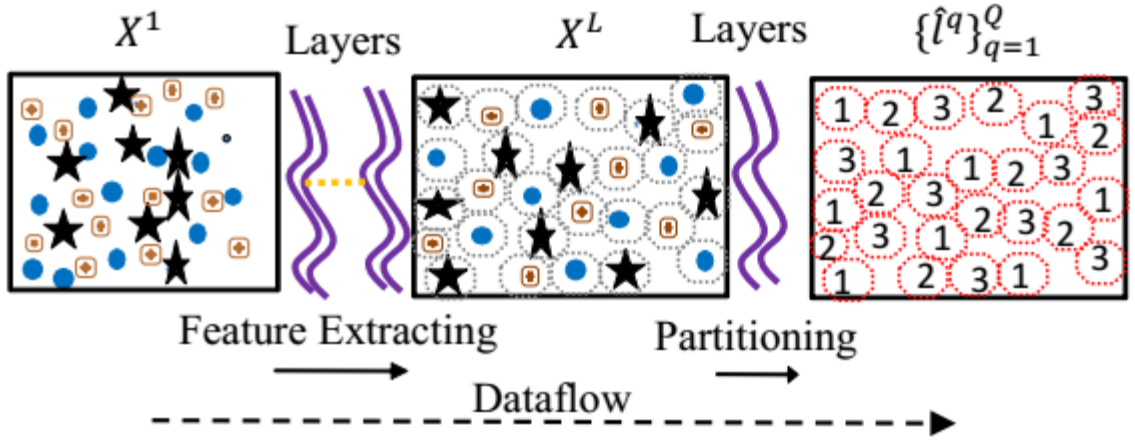




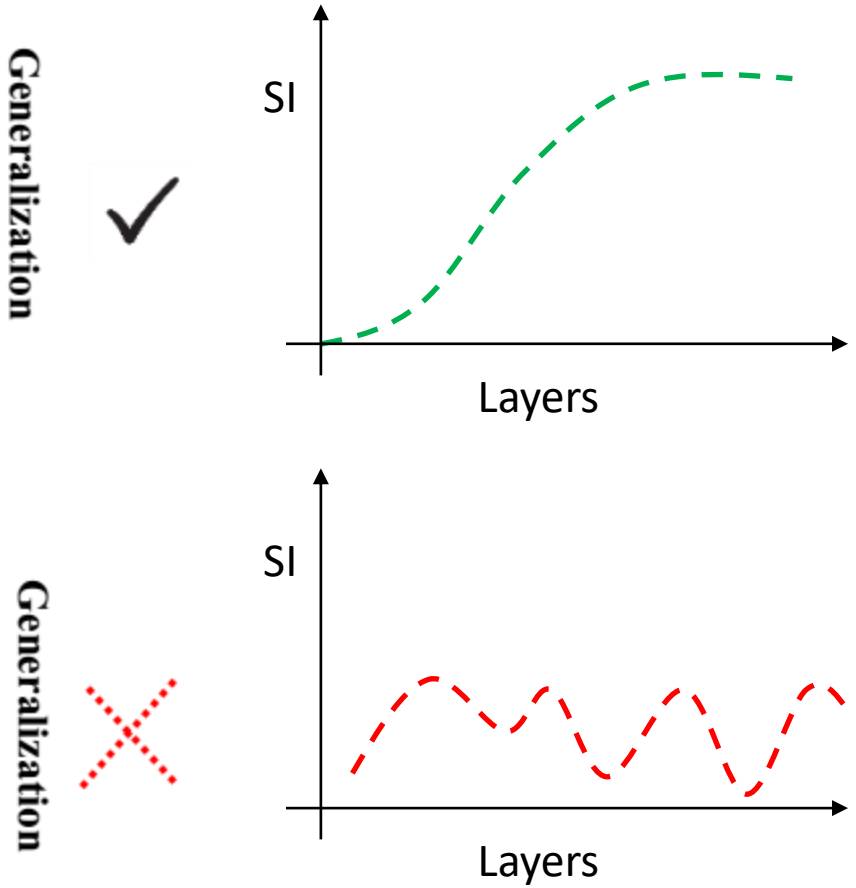
Generalization and Separation index



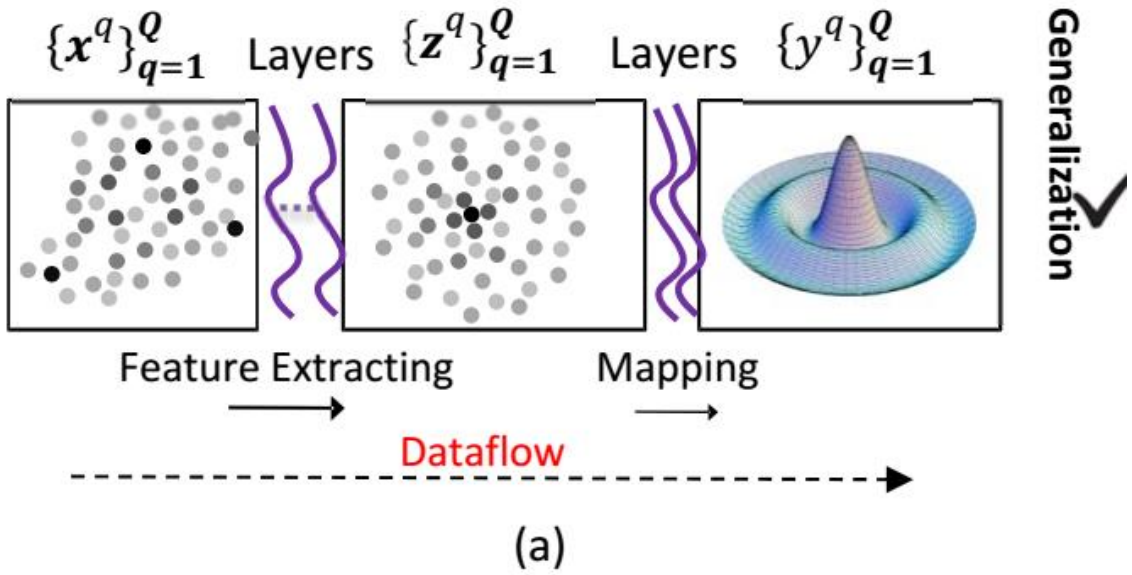
(a)



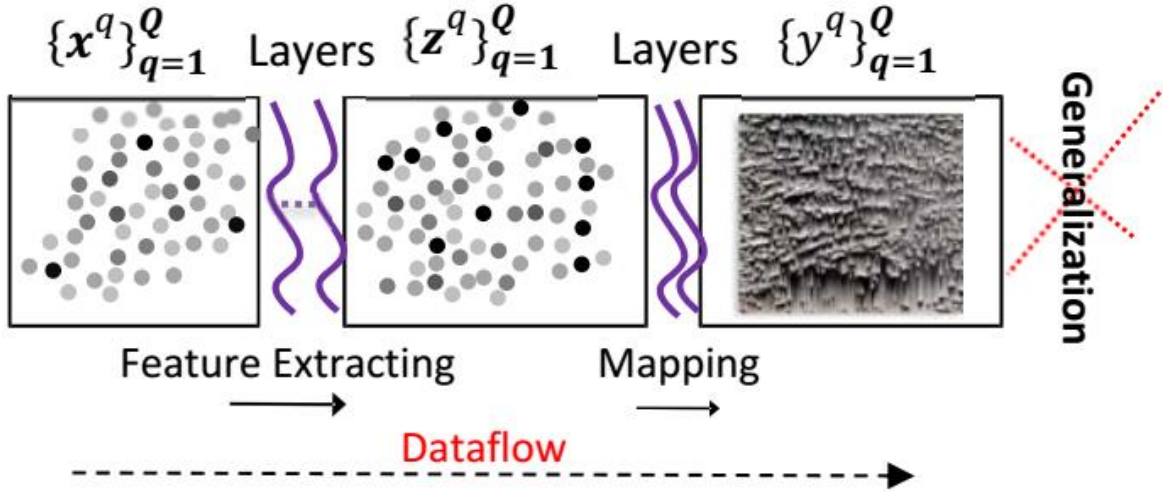
(b)



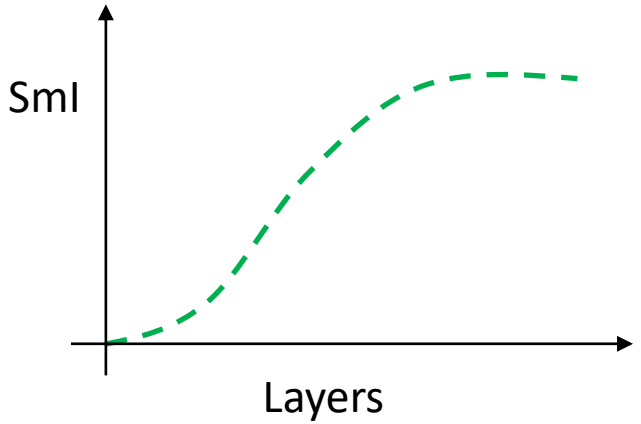
Generalization and Smoothness index

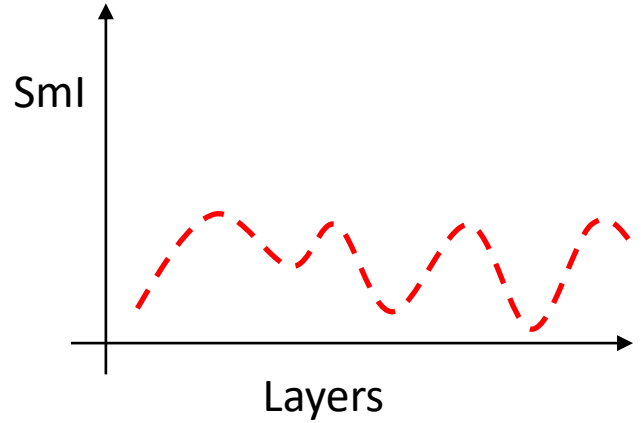


(a)



(b)





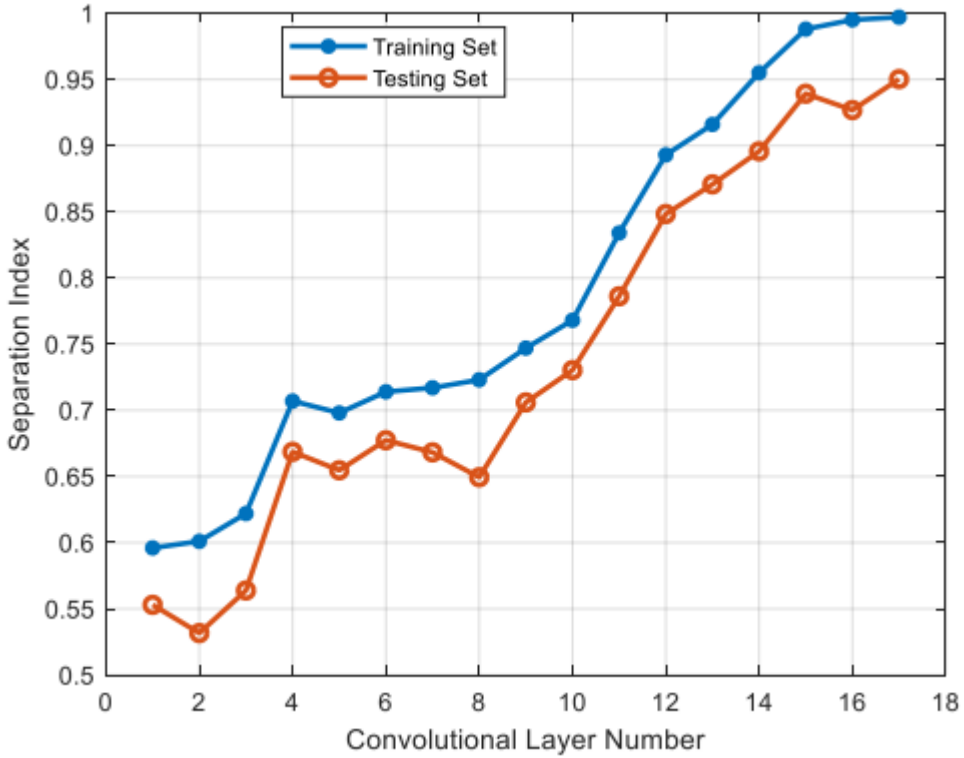
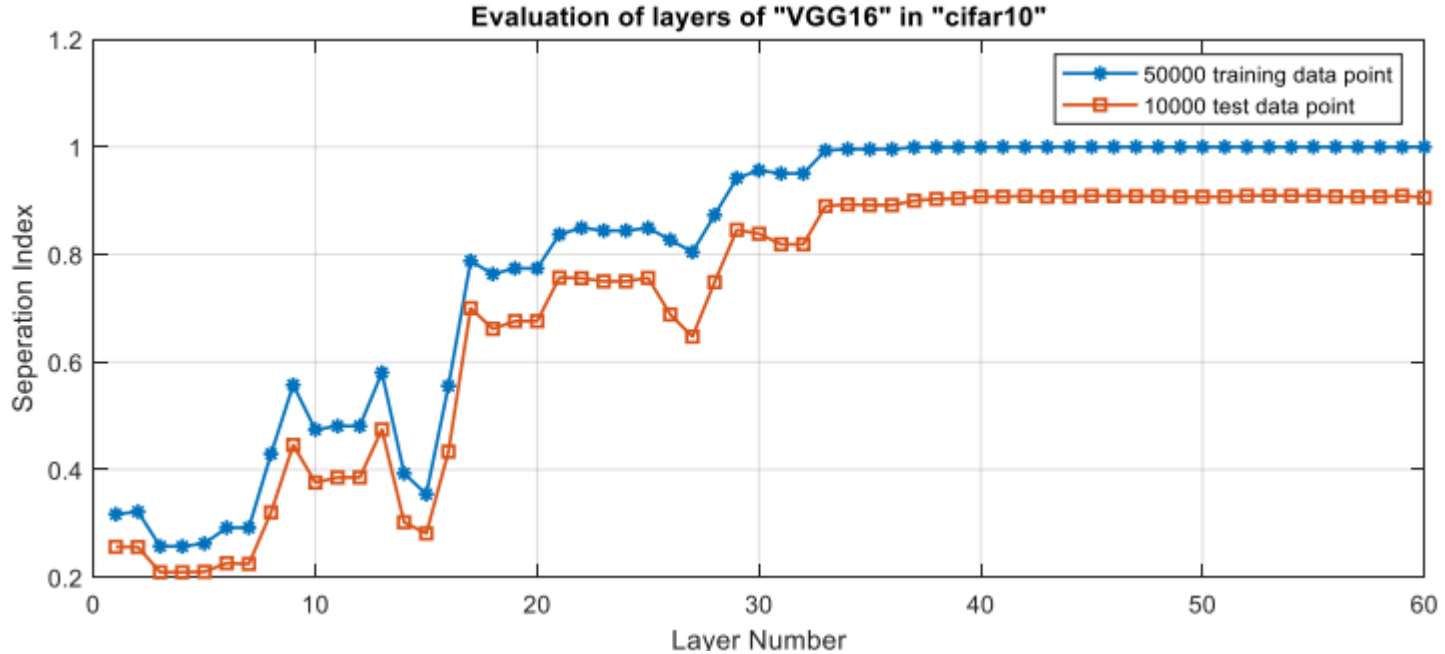
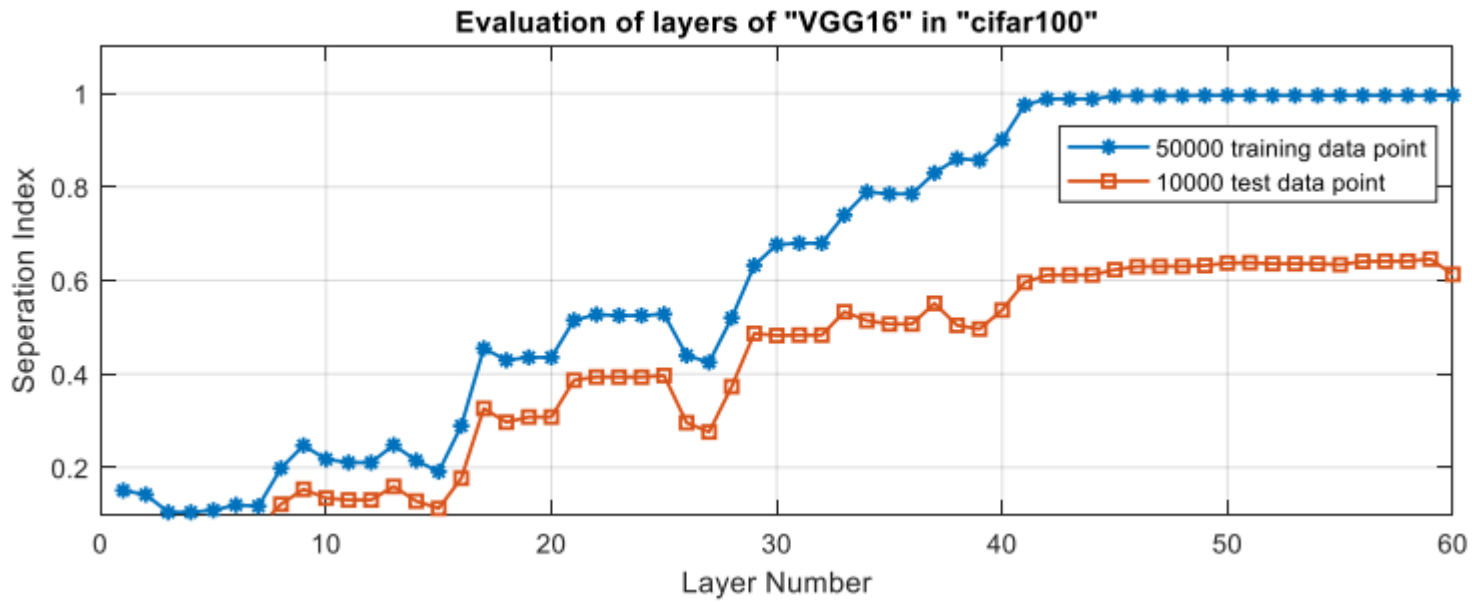


Fig. 10. Evaluation of dataflow through layers of the "Resnet18" network in the classification of Fashion-MNIST.

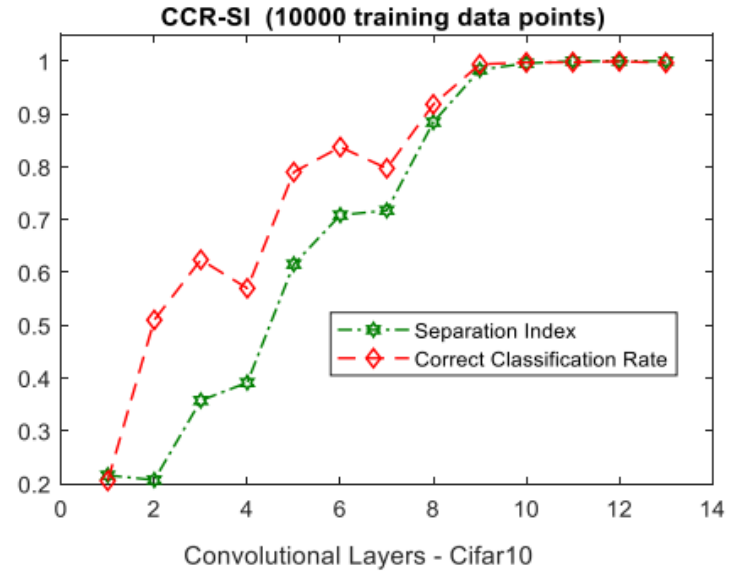




Correct Classification rate and SI

To compare correct classification rate and SI:

1. After each convolution layer of a pertained network, two dense layers are added in order to predict the true labels. After two dense layers, a batch normalization layer is utilized and after them, a softmax layer is applied.
2. Considering the sum of squared error as the loss function, the "Adam" optimizer with more than 100 epochs has been utilized.
3. This process is performed on the "cifar10" dataset on pre-trained "VGG-16" and "Fashion-MNIST" dataset on trained "Resnet18" separately.



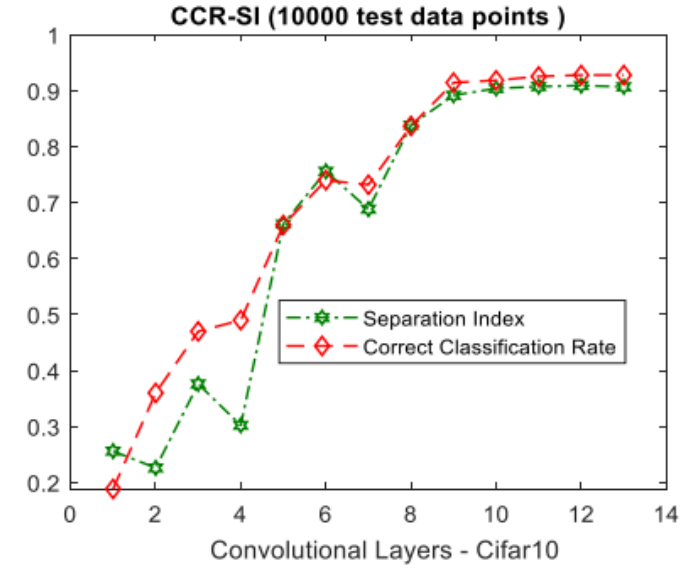
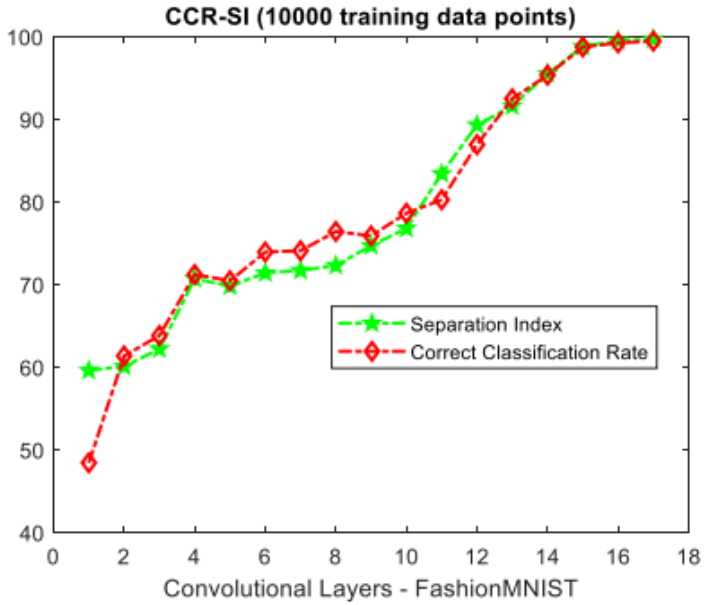


Fig. 11. The plots of separation index and correct classification rates at convolution layers in a pertained "vgg-16" network utilized for classification of "cifar10" for both training and test sets.



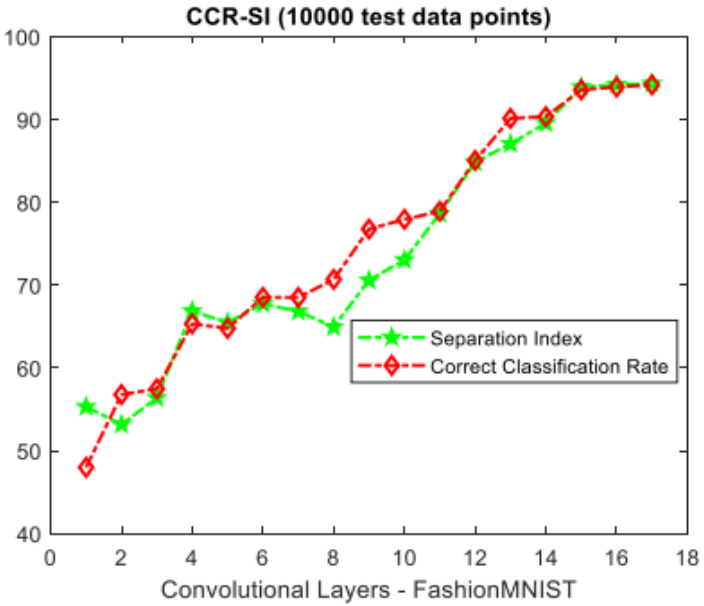


Fig. 12. The plots of separation index and correct classification rates at convolution layers in a trained "Resnet18" network utilized for classification of "Fashion-MNIST" for both training and test sets.

Pre-train Model ranking

- Assume there are M different pre-trained models which are trained by M different source data.
- Assume for j=1,2,...,M, $there\ are\ \ L_j$ layers before fully connected layers .
- Now, we want rank n_{model} pre-trained models to be used in transfer learning for a target data:

$Data=\{(x_i, y_i)\}_{i=1}^{m}$, y_i for classification problems denotes the same label l_i

Algortihm2: (To suggest a pre-trained model in transfer learning)

1. Apply the target data to jth pre-trained model and provide following dataflow at the last layer before fully connected layers:

$$Data^{j,L_j}=\left\{\left(x_i^{j,L_j}, y_i\right)\right\}_{i=1}^{m}$$

2. For jth pre-trained model select the best subset of x_i^{j,L_j} as $x_i^{j^*,L_j}$ which maximizes $SI(SmI)$

$$Data^{j^*,L_j}=\left\{\left(x_i^{j^*,L_j}, y_i\right)\right\}_{i=1}^{m} \qquad x_i^{j^*,L_j} \subseteq x_i^{j,L_j}$$

3. Now rank the pre-trained models as best candidates for the target data in transfer learning as follows:

$$Best\_Models = \{j_1 \dots, j_M\} \quad \text{where} \quad SI\left(Data^{j_1^*,L_{j_1}}\right) > SI\left(Data^{j_2^*,L_{j_2}}\right) > \cdots > SI(Data^{j_M^*,L_{j_M}})$$

Model Confidence and Guarantee (SI, SmI)

Assumptions

1. There is a training dataset $Data = \{(x_i, l_i)\}_{i=1}^{m}$ ($Data = \{(x_i, y_i)\}_{i=1}^{m}$) for a classification(regression) problem.
2. The $SI(Data)$ ($SmI\ (Data)$) is computed for the dataset.
3. There is a test dataset which is homogenous with the training dataset.
4. Based on the Nearest Neighbor (NN) model, we want to predict the target $l\ (\boldsymbol{y})$ of a new test example $\boldsymbol{x}$, as $\hat{l}(\hat{\boldsymbol{y}})$.
5. It is assumed that $x_{i_j^*}$ denotes the jth nearest neighbor of input data to x.
6. There is an uncertainty in measuring x . It is assumed that x has maximum measuring error (distance) γ to its true value x_{true}: $\|x_{true} - x\| < \gamma$.
7. It is aimed to know the confidence level of the prediction by the NN model.
8. It is aimed to know if there is a guarantee to predict true label in classification problem or to have a bounded output error in a regression problem.

Model Confidence and Guarantee by SI (without training data)

Assume dw_{max} denotes the maximum distance of nearest neighbor exampled with the the same label and db_{min} denotes the minimum inter distance between examples with different labels .

$Conf(\hat{l}, l_{i_1^*})$ denotes the confidence for prediction of $\hat{l} = l_{i_1^*}$ (by the NN model).

Guar $(\hat{l}, l_{i_1^*})$ denotes the guarantee that prediction of $\hat{l} = l_{i_1^*}$ (by the NN model) is true.

Guar $(\hat{l}, l_{i_1^*})$ =1 means that there is a guarantee for the prediction.

- $Conf(\hat{l}, l_{i_1^*})$=$SI(Data)$
- $Guar(\hat{l}, l_{i_1^*}) = \delta(SI, 1) * sign(1 - \frac{dw_{max}+\gamma}{db_{min}-\gamma})$

γ: the maximum uncertainty in measuring x :$\|x_{true} - x\| < \gamma$

❖ If $SI(Data) = 1$ and $\gamma < 0.5(db_{min} - dw_{max})$, then there is a guarantee for true prediction of the test example by the nearest neighbor model.

Model Confidence and Guarantee by SI (with training data)

For input test example x define:

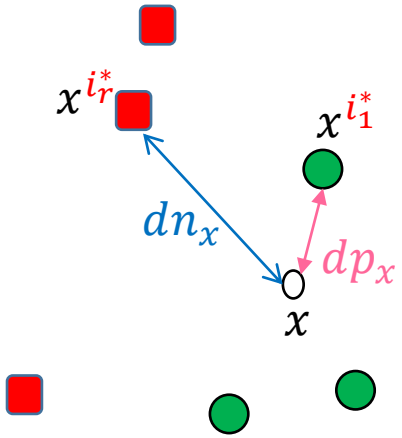
$dp_x = \|x - x_{i_1^*}\|$

$dn_x = \|x - x_{i_r^*}\|$ $\quad r = \min_{j=1,..,m} j$ subjec to $\delta(l_{i_j^*}, l_{i_1^*}) = 0$

- $Conf(\hat{l}, l_{*_1}) = SI$
- $Guar(\hat{l}, l_{*_1}) = \delta(SI, 1) * sign(1 - \frac{dp_x + \gamma}{dn_x - \gamma})$

❖ If $SI(Data) = 1$ and $\gamma < 0.5(dn_x - dp_x)$, then there is a guarantee for true prediction of the test example by the $model$.

About those cases that we have $SI^r(Data) = 1$ for $r \gg 1$ the guarantee is satisfied for much higher γ than the cases which we have $SI^1(Data) = 1$.



Model Confidence and Guarantee by SmI(with training data)

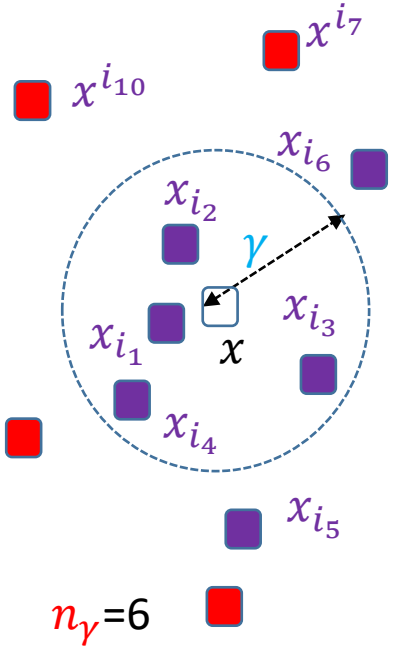
- Assume that for each data point x_i, $\alpha(i^*)=\|y_i - y_{i^*}\|$, where $x_{i^*}$is the nearest neighbor ofx_i.
- Assuming $SmI(Data) = 1$, $\alpha(i^*)$ denotes the distance between the output y_i and its nearest neighbor, y_{imin}.
- In an ideal case a$ssume\ that\ the\ divesity\ of\ training\ dataset\ is\ so\ high$ and $for\ each\ test\ data$ point x (when there is no measuring uncertainty) $\left\|y - y_{i_1^*}\right\| \leq \alpha(i_1^*)$.
- Now assume, γ is the maximum measuring uncertainty of input x.
- Find all training examples: $\left\{x_{i_j^*}\right\}_{j=1}^{n_\gamma}$ that each one is a nearest neighbor of a point of the hyper-circle set $C = \{\check{x}, \|\check{x} - x\| \leq \gamma\}$.

❖ it is guaranteed that if $for\ each\ test\ data\ pint, x, when\ \gamma \geq 0$

$$\left\|y - y_{i_1^*}\right\| \leq \bar{e}_y$$

$$\bar{e}_y = \max_{j=1,\dots,n_\gamma} \left(\left\|y_{i_1^*} - y_{i_j^*}\right\| + \alpha(i_j^*) \right)$$

- About those cases that we have $SmI^r(Data) = 1$ for $r \gg 1$, $\bar{e}_y$ is much lower than the cases which we have $SmI^1(Data) = 1$.



3.3 Region Based CNNs_part1
(Detection Problems)

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

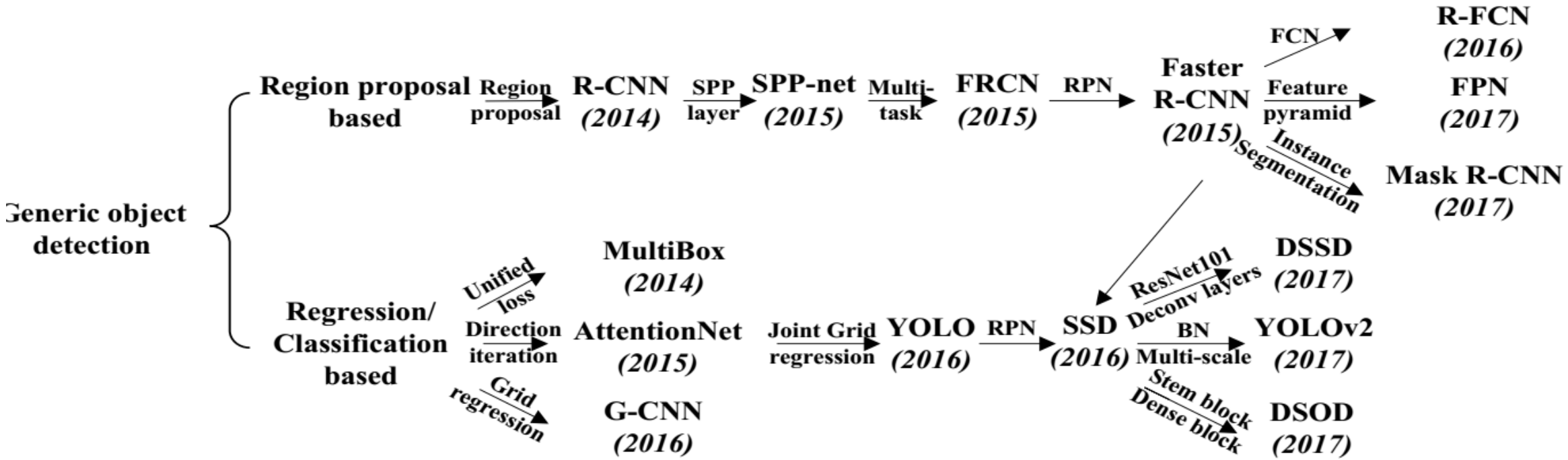
Single stage detectors

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

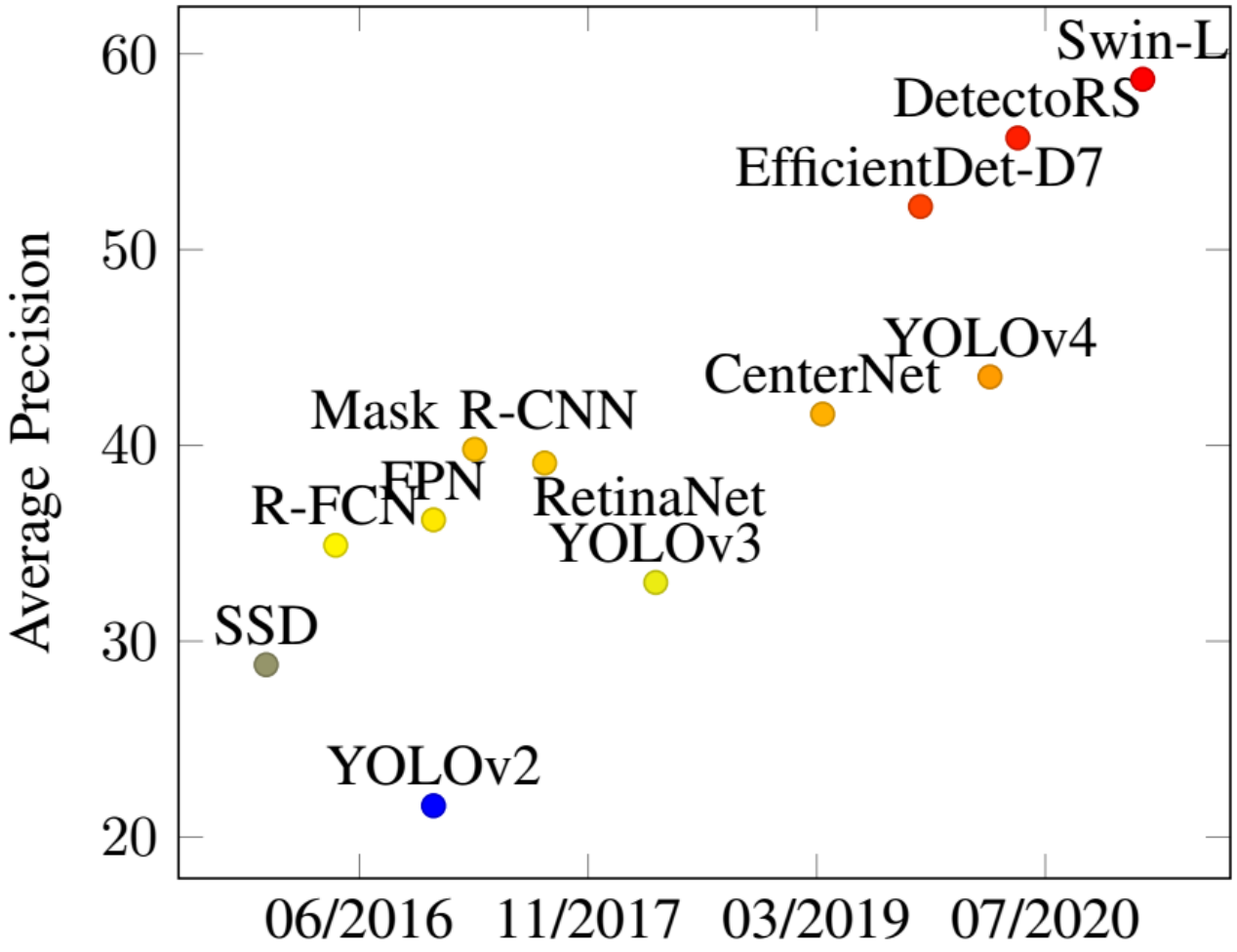
TABLE III: Performance comparison of various object detectors on MS COCO and PASCAL VOC 2012 datasets at simila input image size.

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

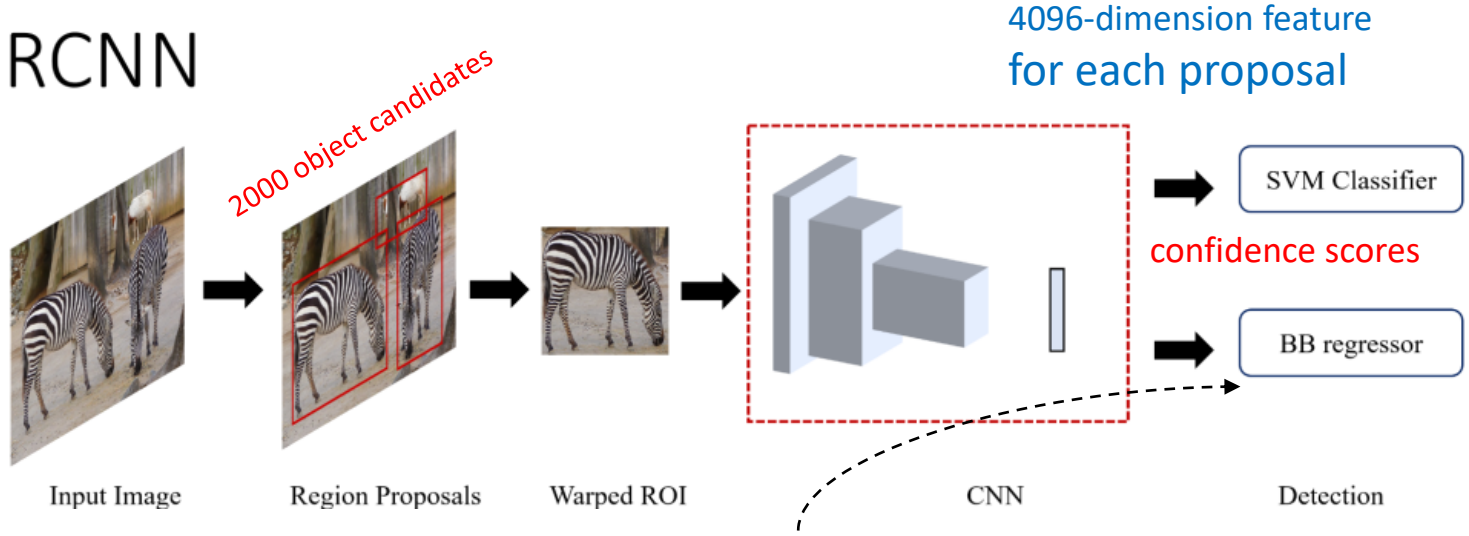
- A network which has a separate module to generate region proposals is termed as a two-stage detector.
- These models try to find an arbitrary number of objects proposals in an image during the first stage and then classify and localize them in the second.
- As these systems have two separate steps,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

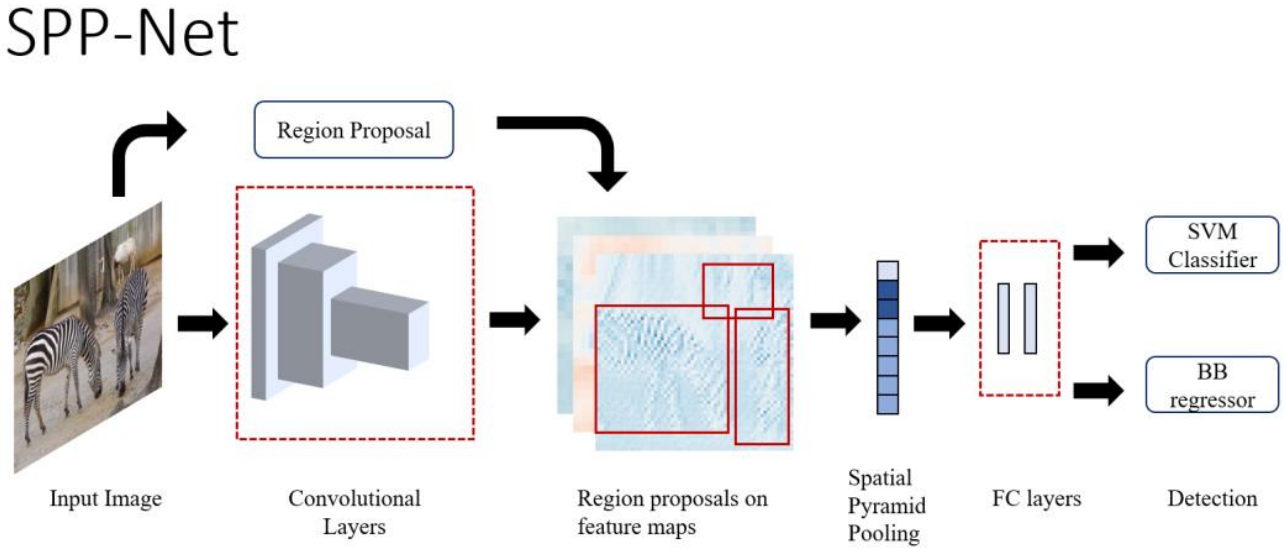
The selective search algorithm is used to generate candidate windows.

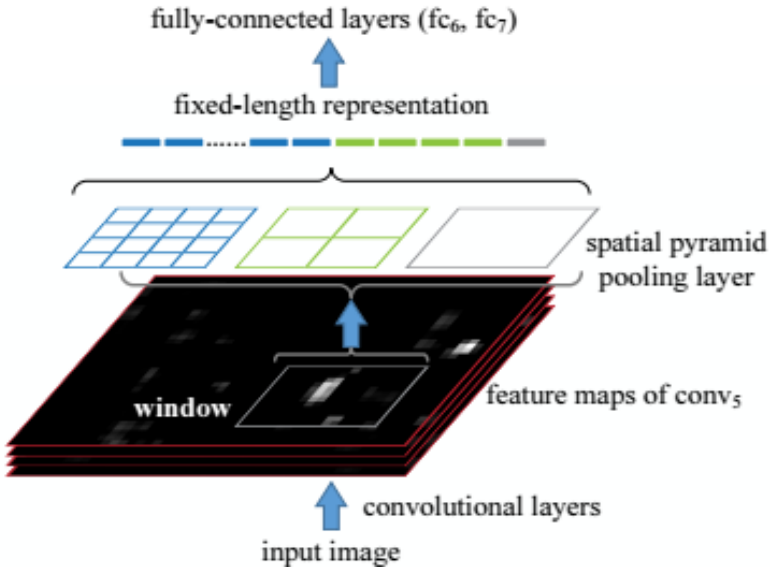
Feature maps are obtained by passing the input image through the convolution layers of a ZF-5 network.

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.

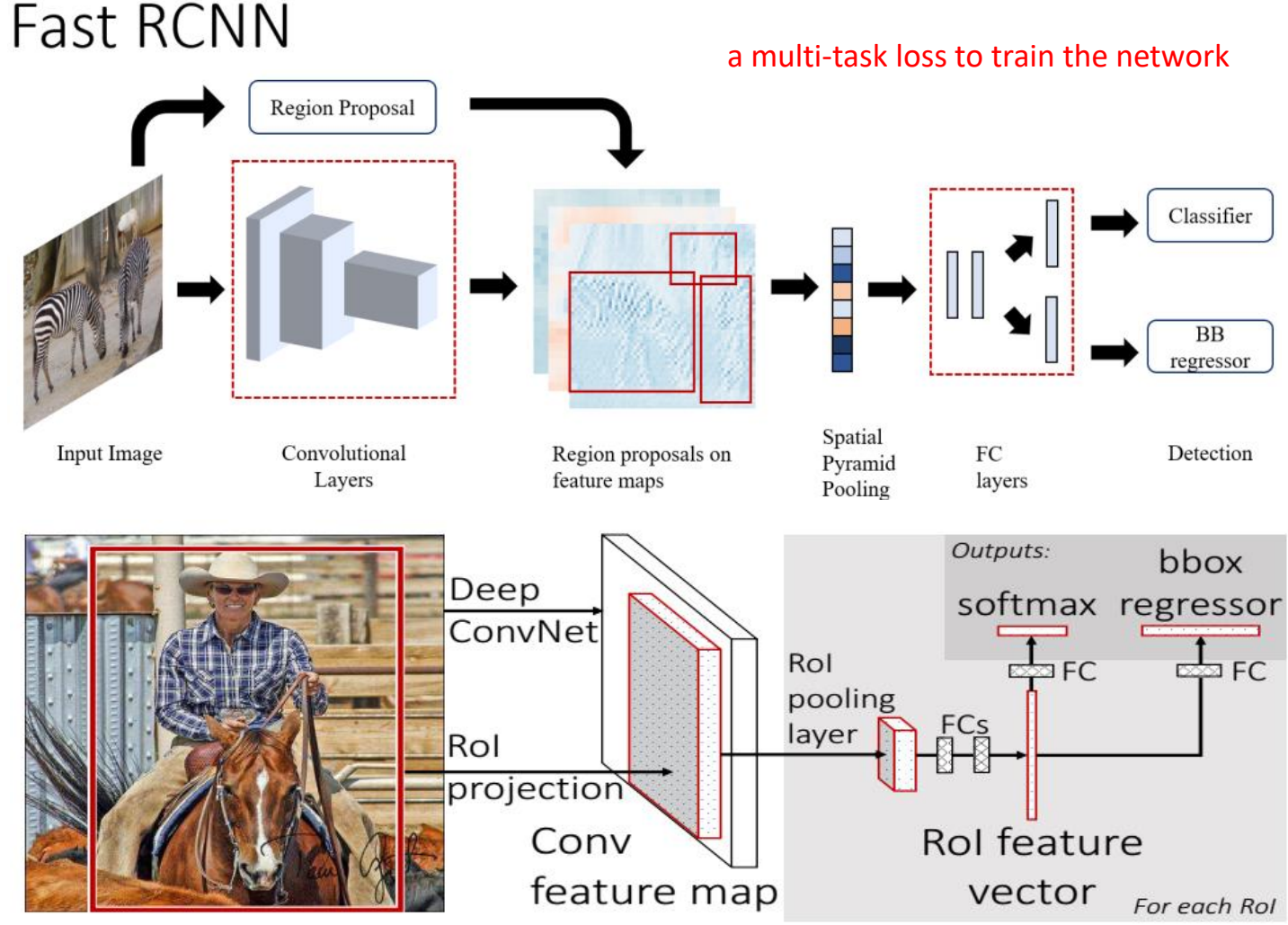




Fast RCNN

*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.

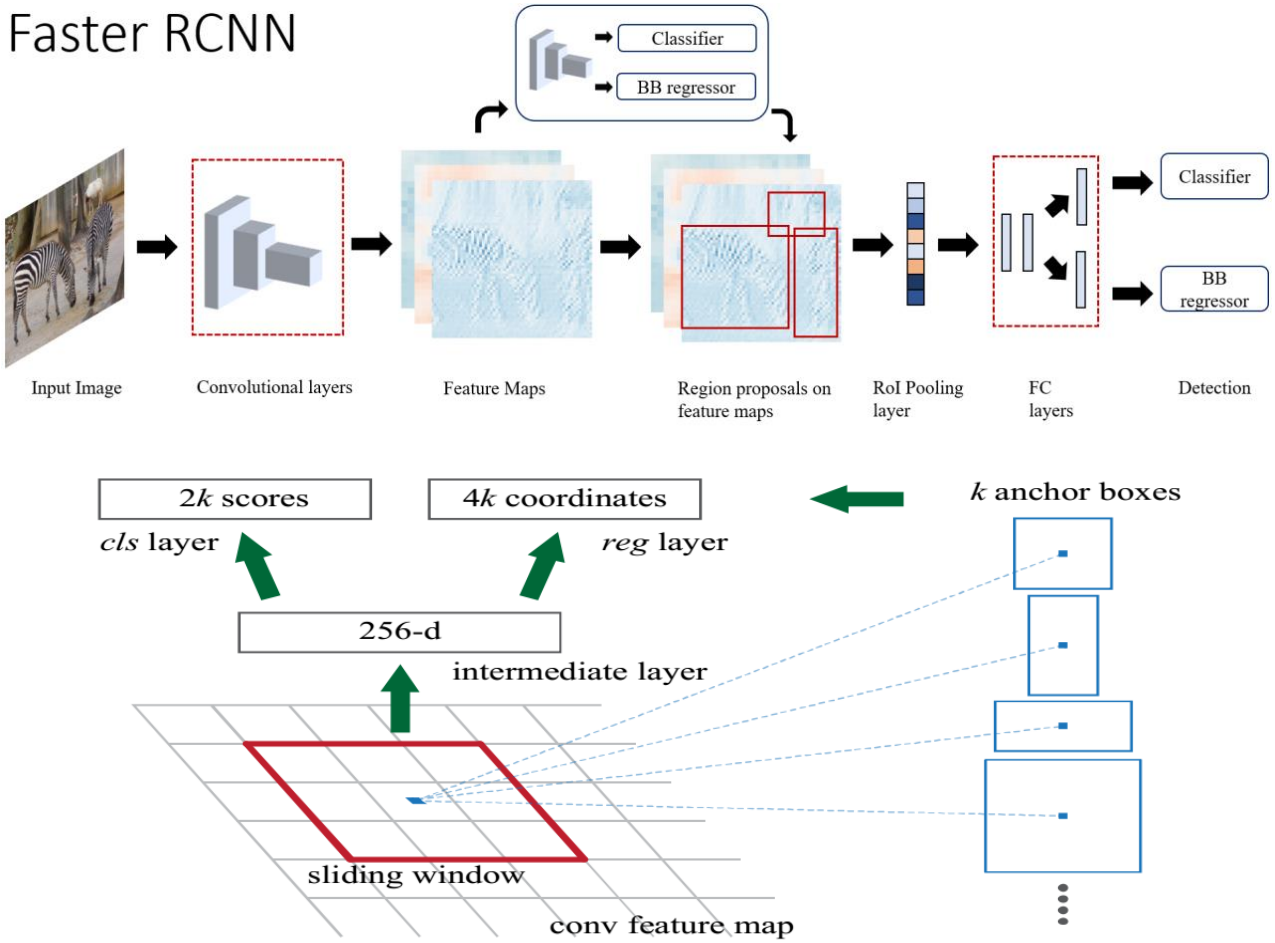


It simplified training procedure, removed pyramidal pooling and introduces a new loss function. The object detector, without the region proposal network, reported near real time speed with considerable accuracy

Faster RCNN

* S. Ren..2016

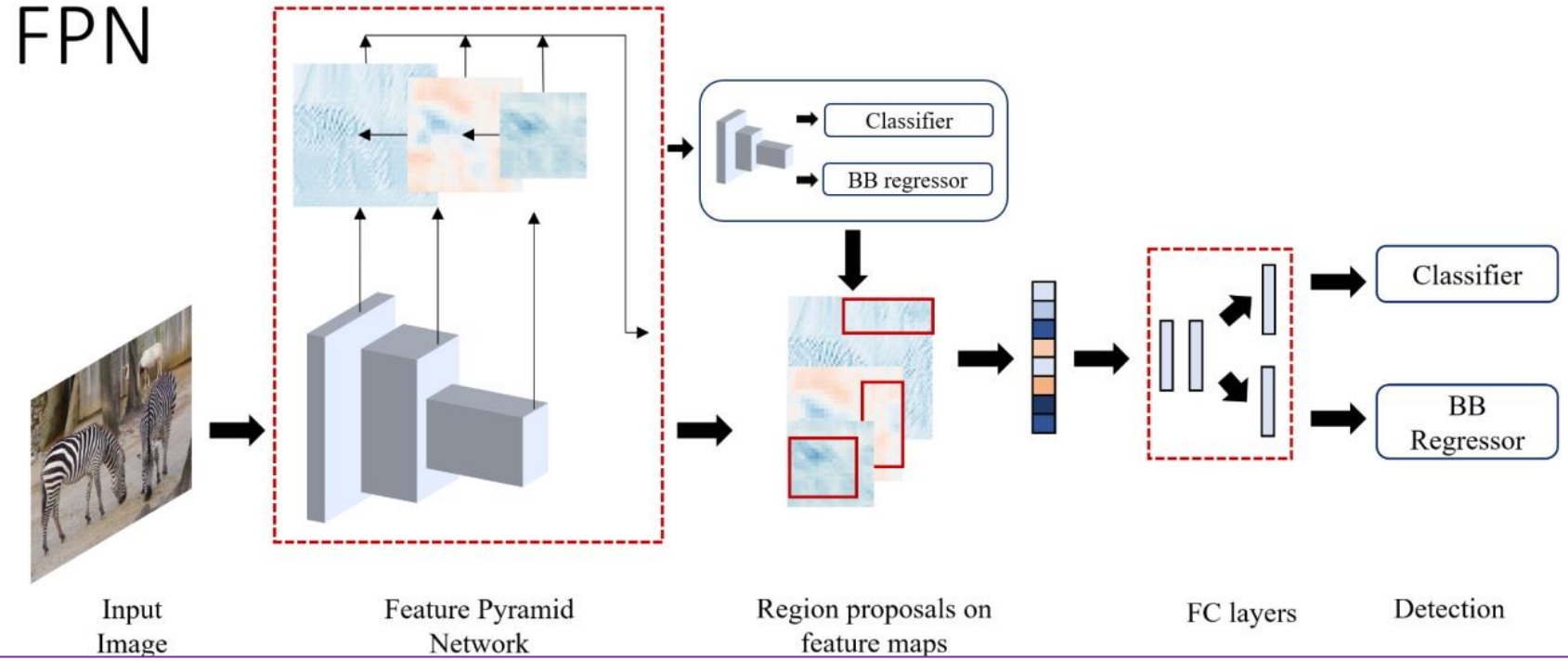
Faster RCNN takes an arbitrary input image and outputs a set of candidate windows.
Each such window has an associated *objectness score* which determines likelihood of an object.
Unlike its predecessors which used image pyramids to solve size variance of objects, RPN introduces Anchor boxes.
It used 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.

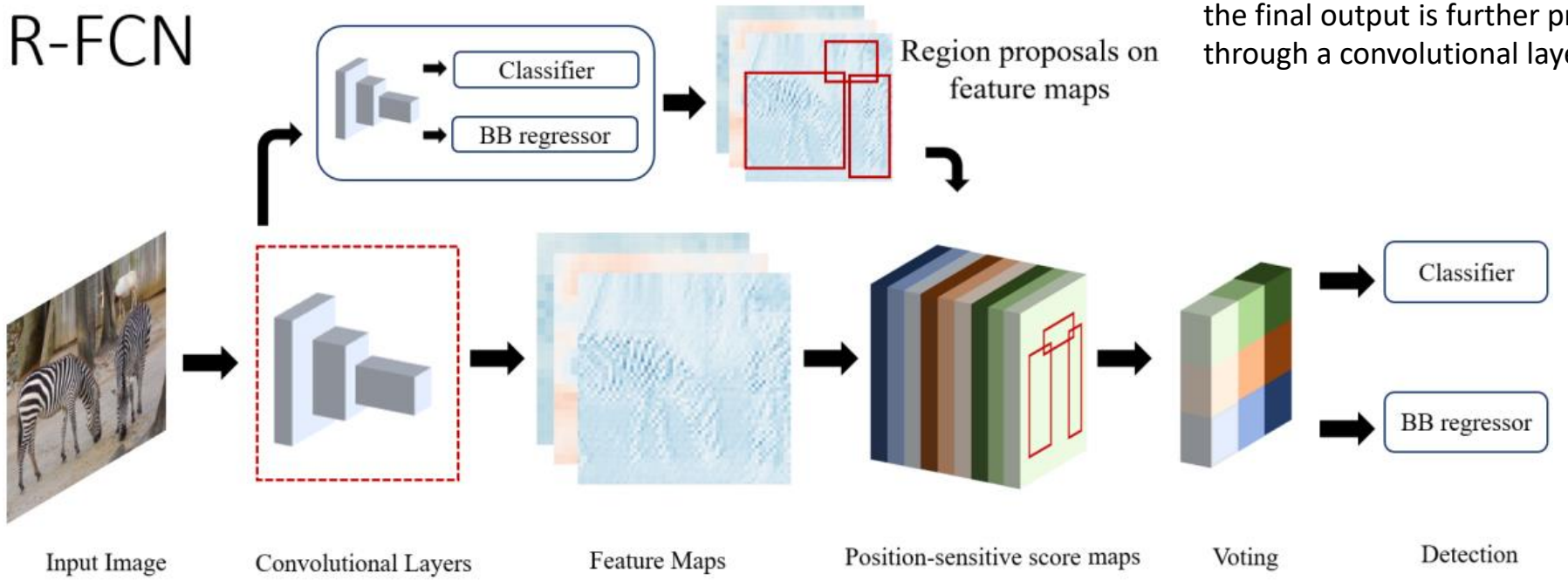
FPN could provide high-level semantics at all scales, which reduced the error rate in detection.

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

R-FCN

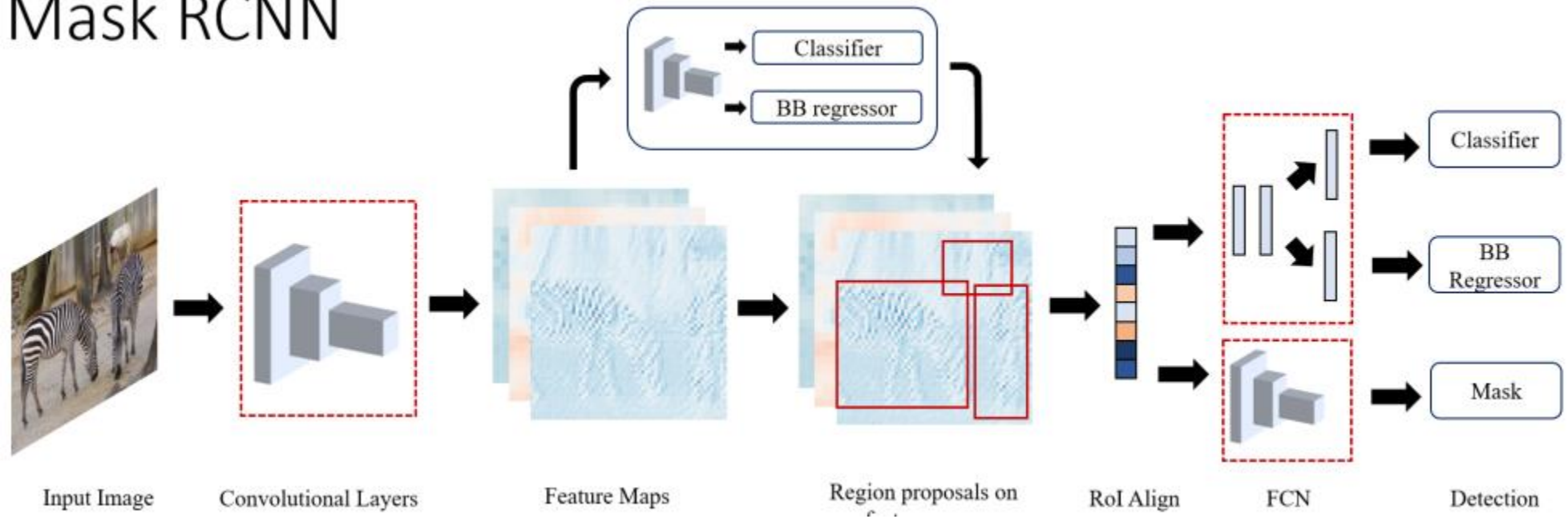


the final output is further processed through a convolutional layer

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

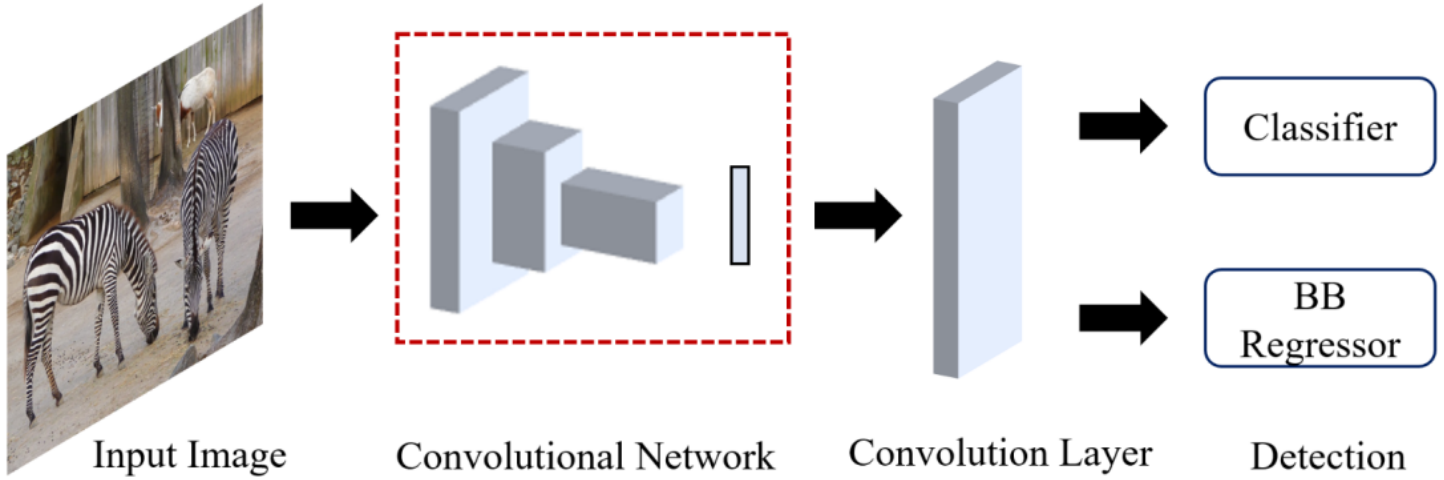
3. One stage object detectors

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

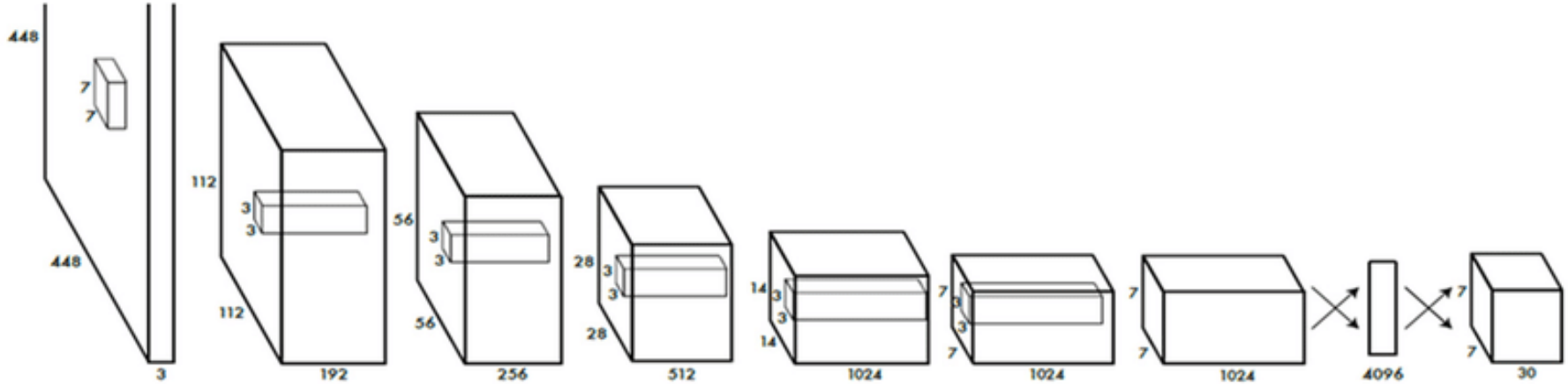
- Single-stage detectors classify and localize semantic objects in a single shot using dense sampling.
- They use predefined boxes/keypoints of various scale and aspect ratio to localize objects.
- It edges two-stage detectors in real-time performance and simpler design.

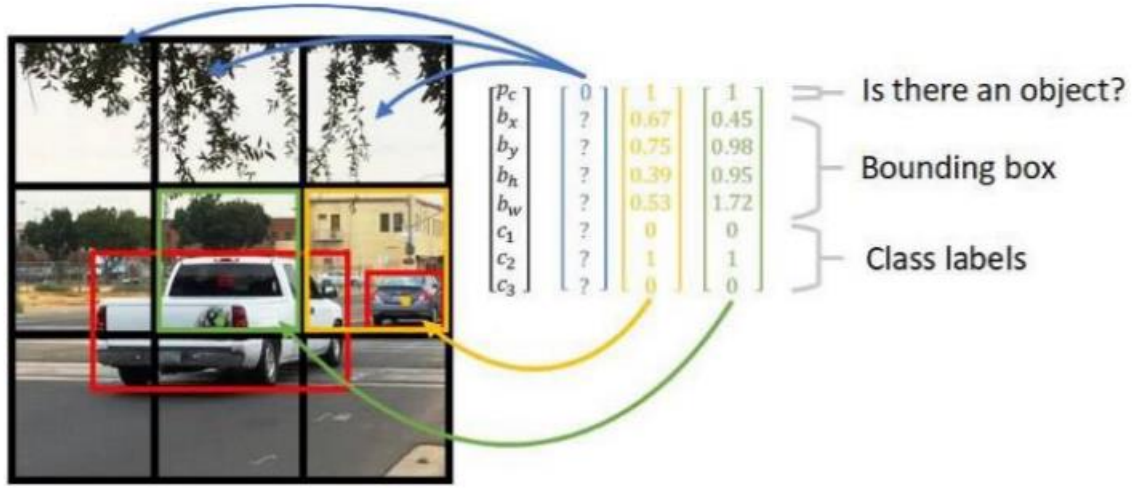
YOLO (You Only Look Once)

* J. Redmon ...2016



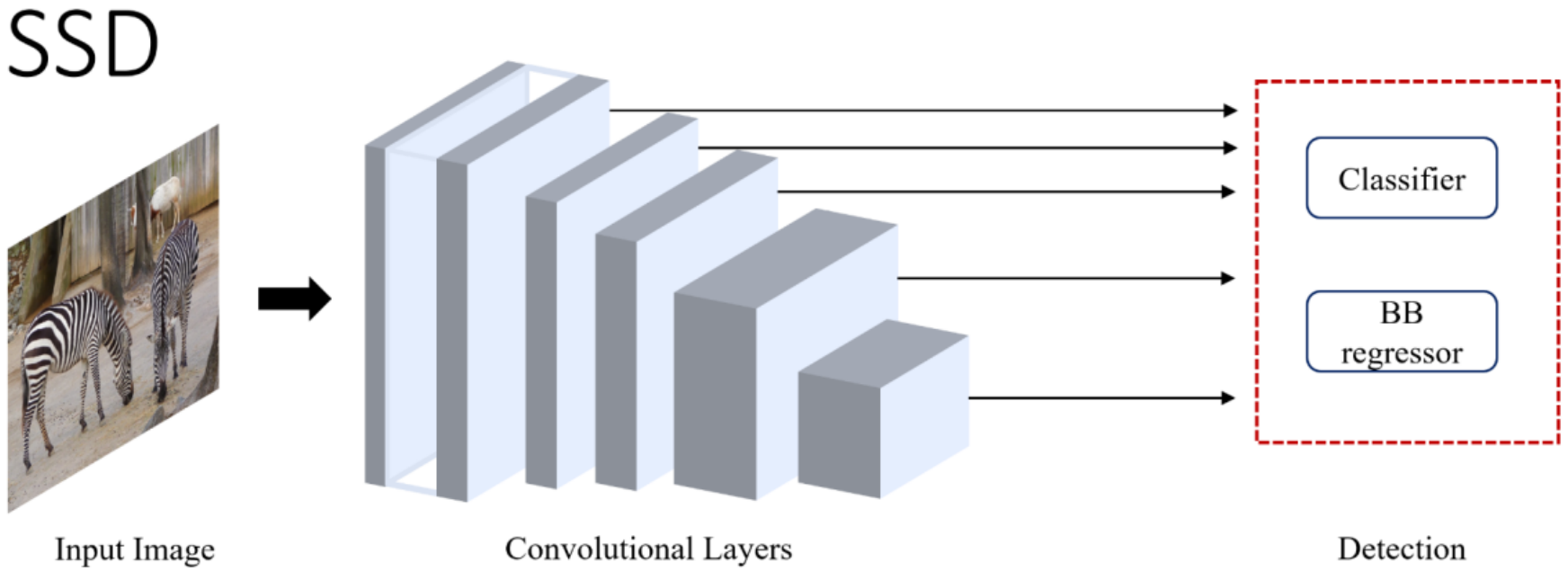
YOLO reframed it as a regression problem, directly predicting the image pixels as objects and its bounding box attributes.
In YOLO, the input image is divided into a *S x S* grid and the cell where the object's center falls is responsible for detecting it.
A grid cell predicts multiple bounding boxes, 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
448
7
7
3
112
112
3
3
192
56
56
3
3
256
28
28
3
3
512
14
14
3
3
1024
7
7
3
3
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
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
×4
Conv. Layers
1x1x512
3x3x1024
3x3x1024
3x3x1024-s-2
×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 1
? 0.67 0.45
? 0.75 0.98
? 0.39 0.95
? 0.53 1.72
? 0 0
? 1 1
? 0 0
Is there an object?
Bounding box
Class labels

SSD(SingleShot Detector)

*W. Liu, 2016



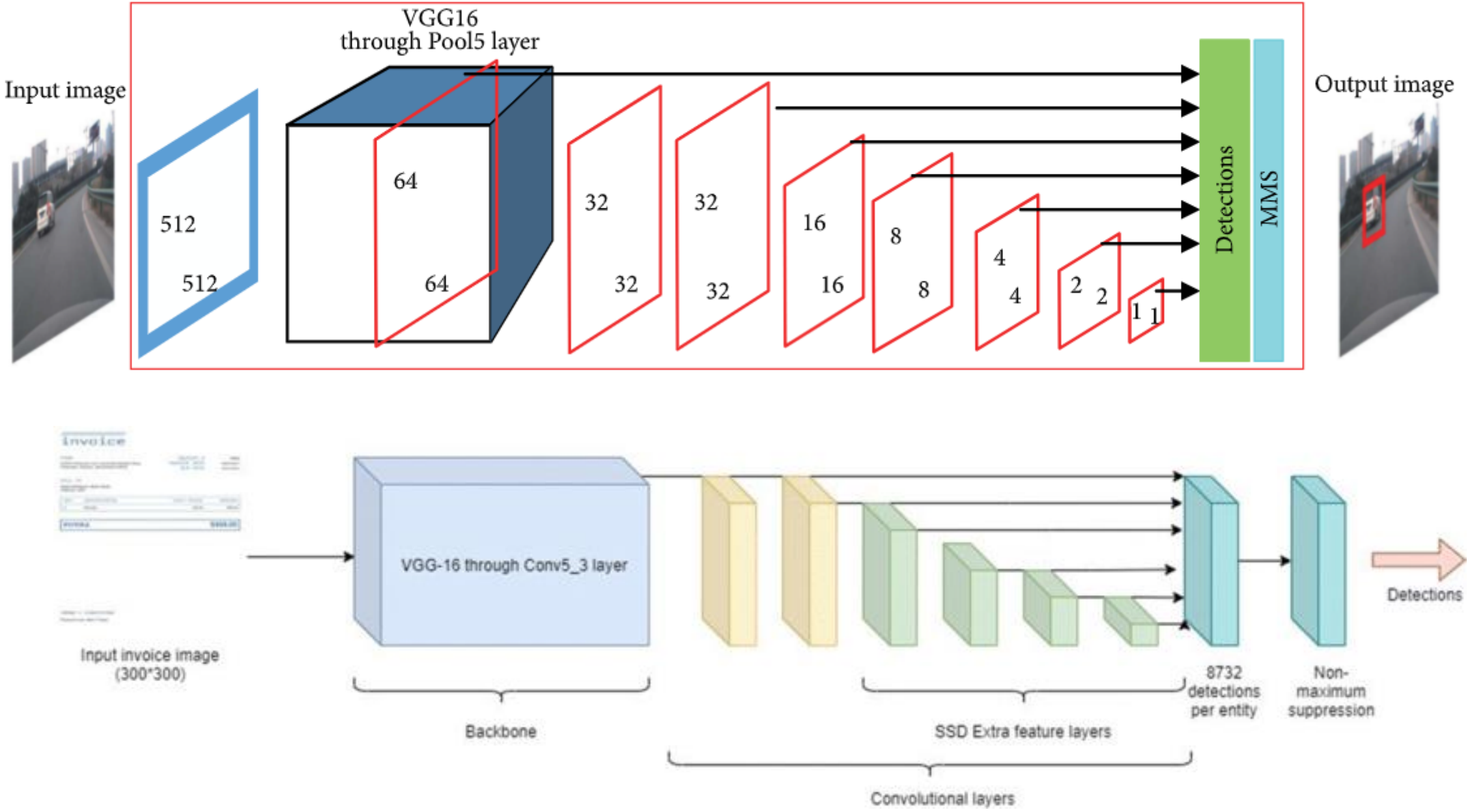
SSD was the first single stage detector that matched accuracy of contemporary two stage detectors like Faster R-CNN [44], while maintaining real time speed.

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

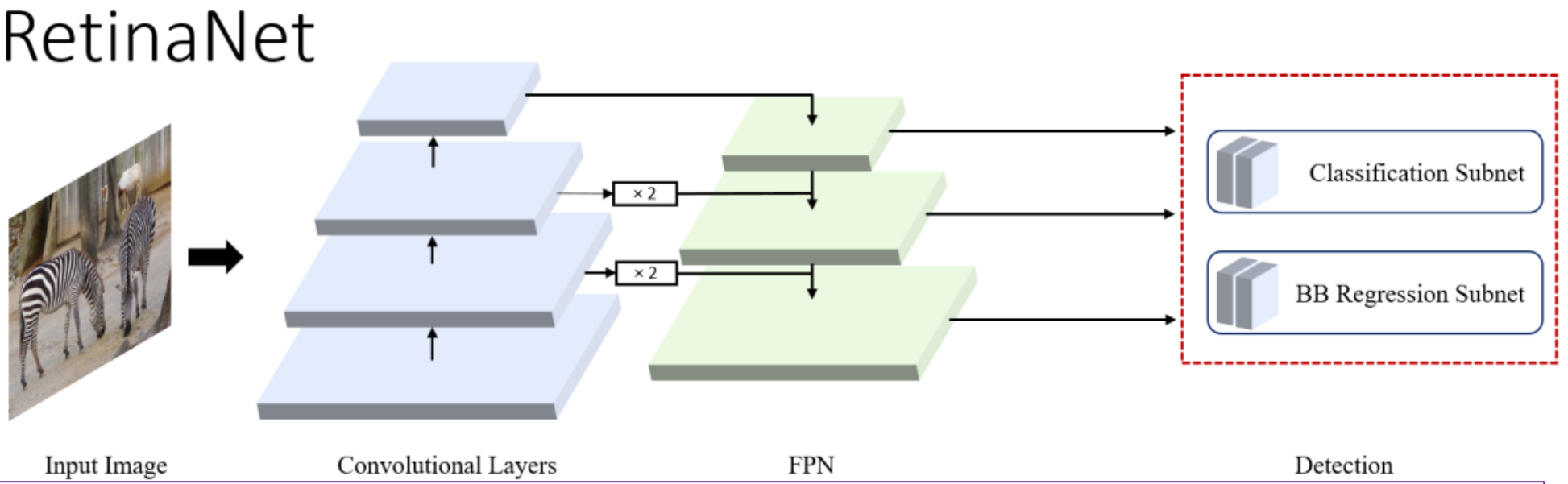
NMS: Non-maximum Suppression

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



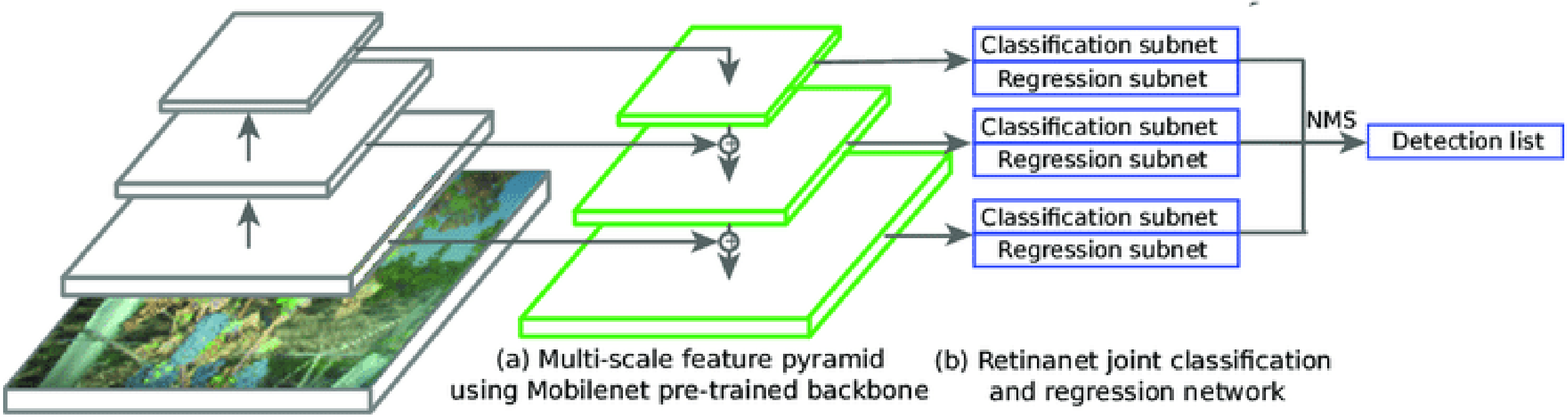
Given the difference between the accuracies of single and two stage detectors, Lin et al. suggested that the reason single stage detectors lag is the “extreme foreground-background class imbalance”

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.

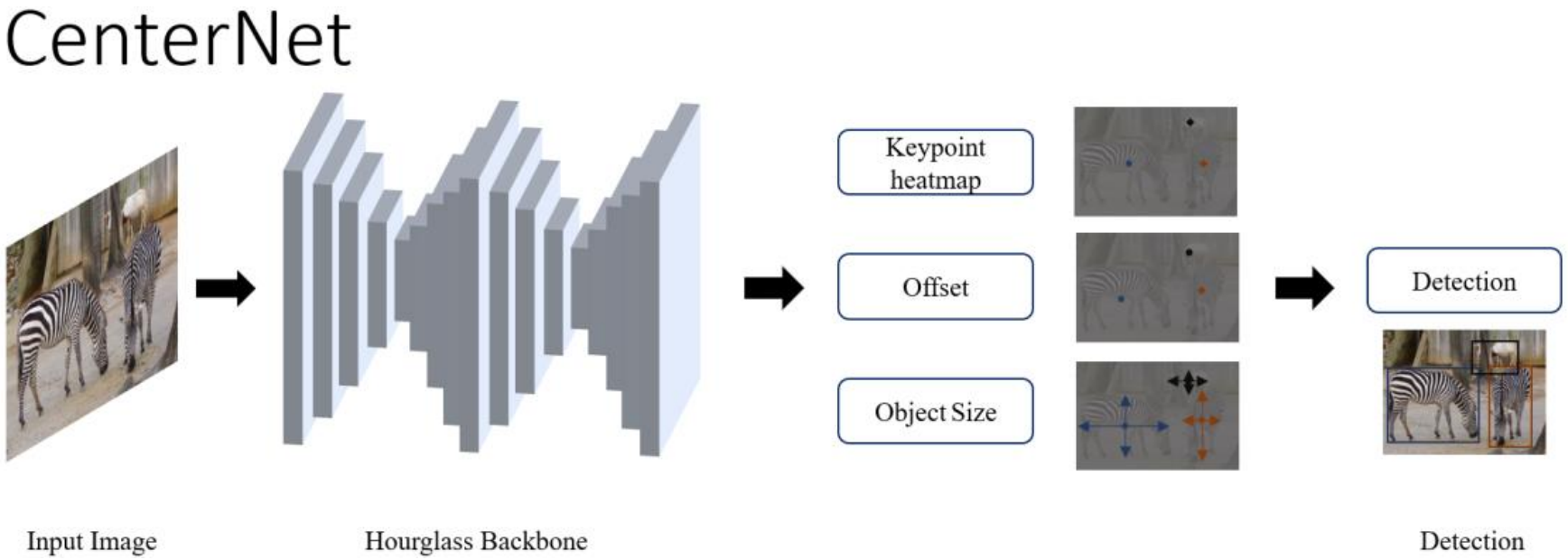


(a) Multi-scale feature pyramid using Mobilenet pre-trained backbone

(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

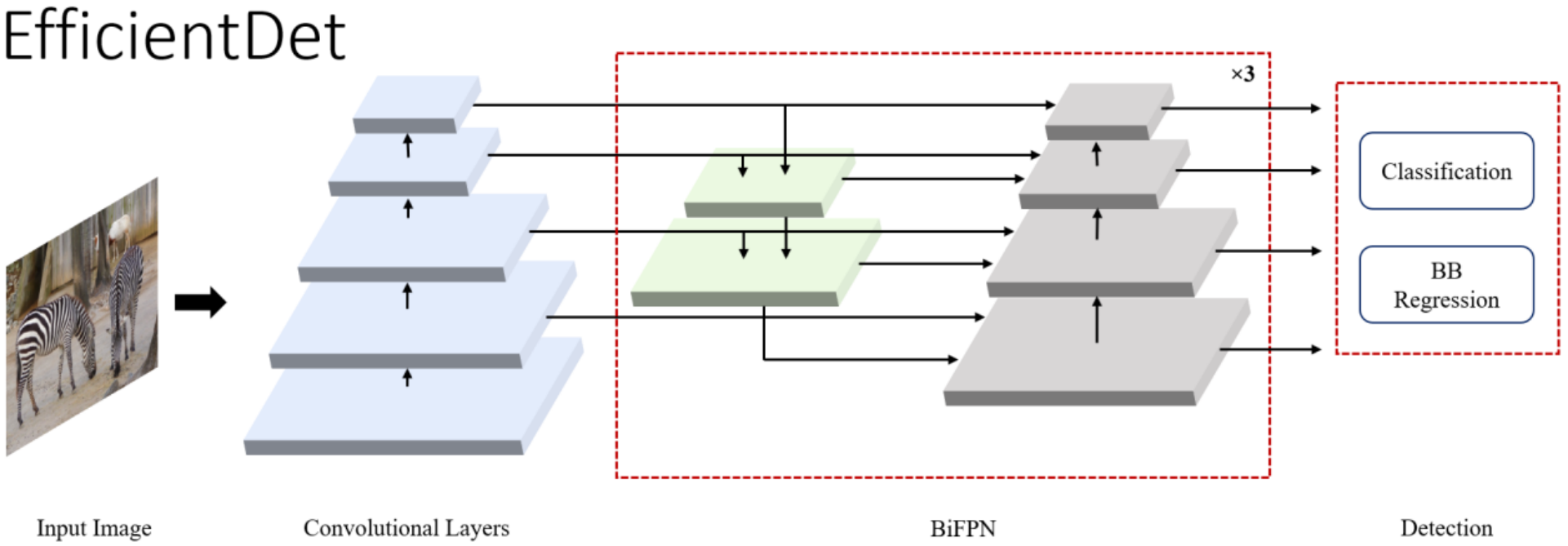
Rectangle

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*

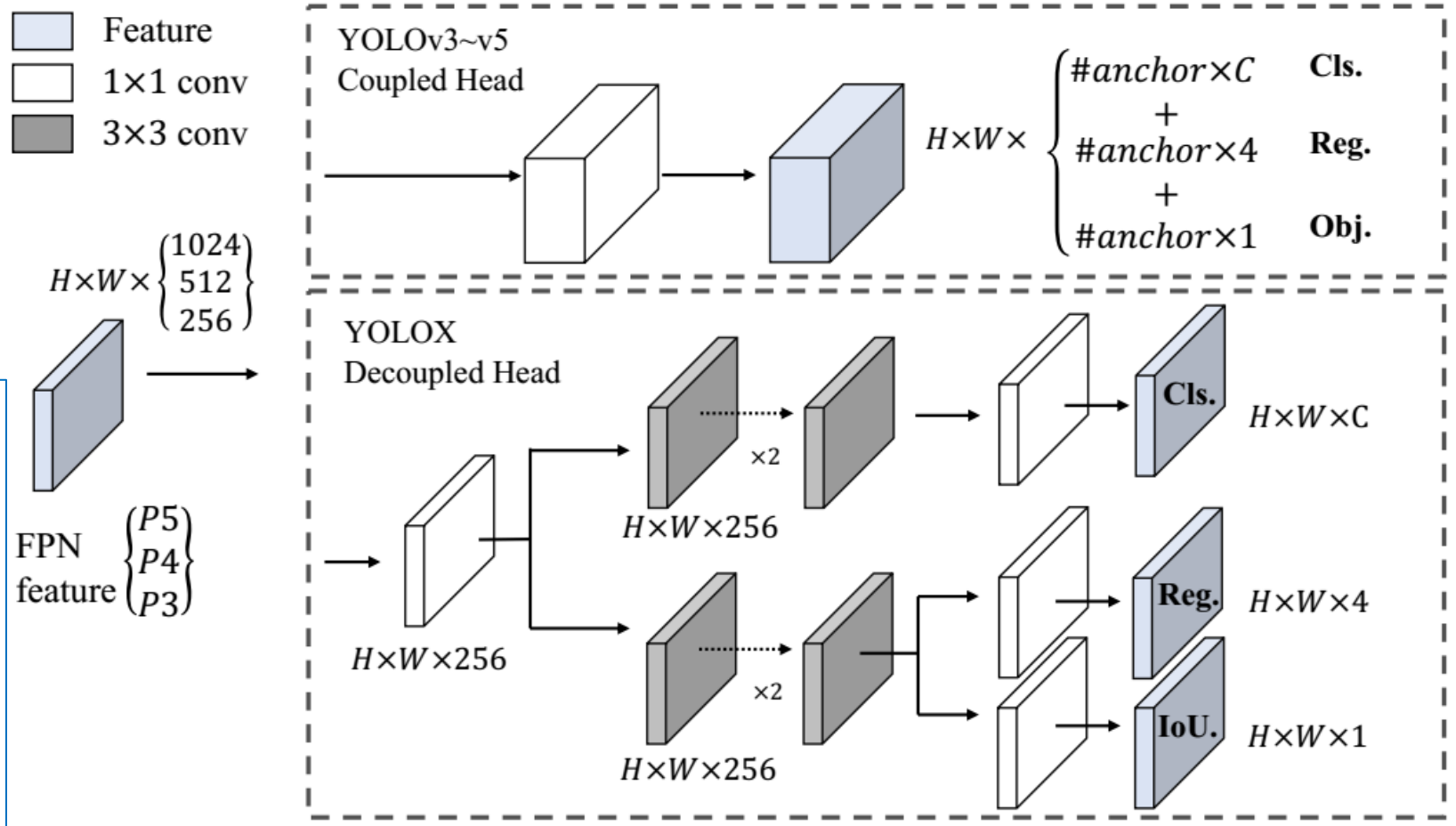
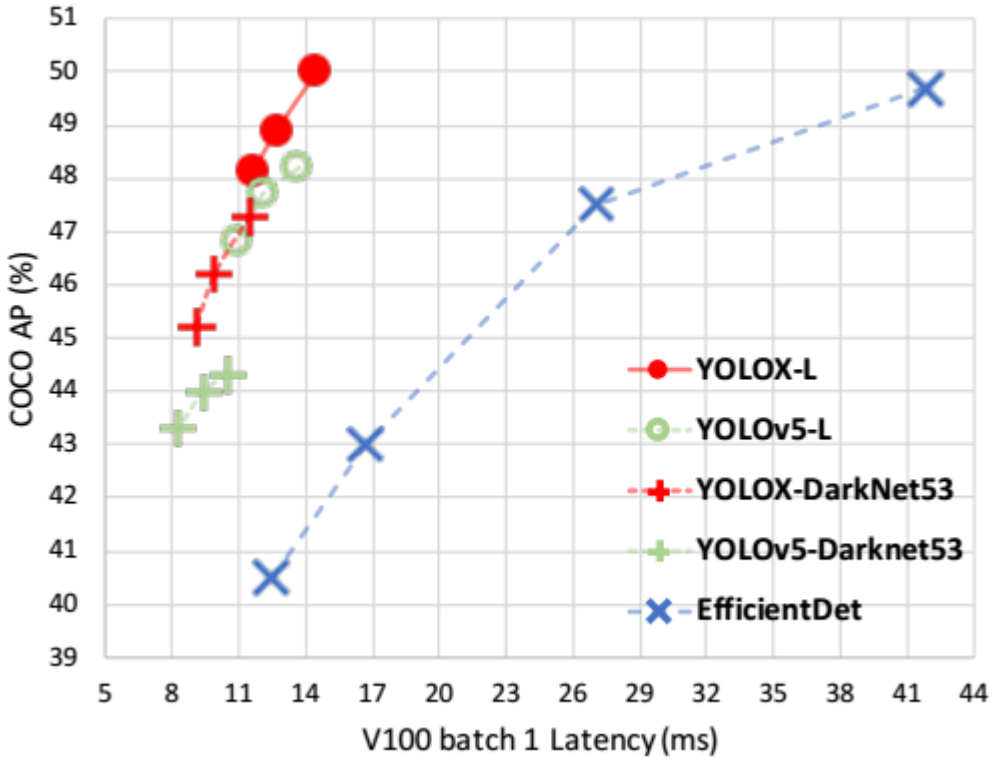
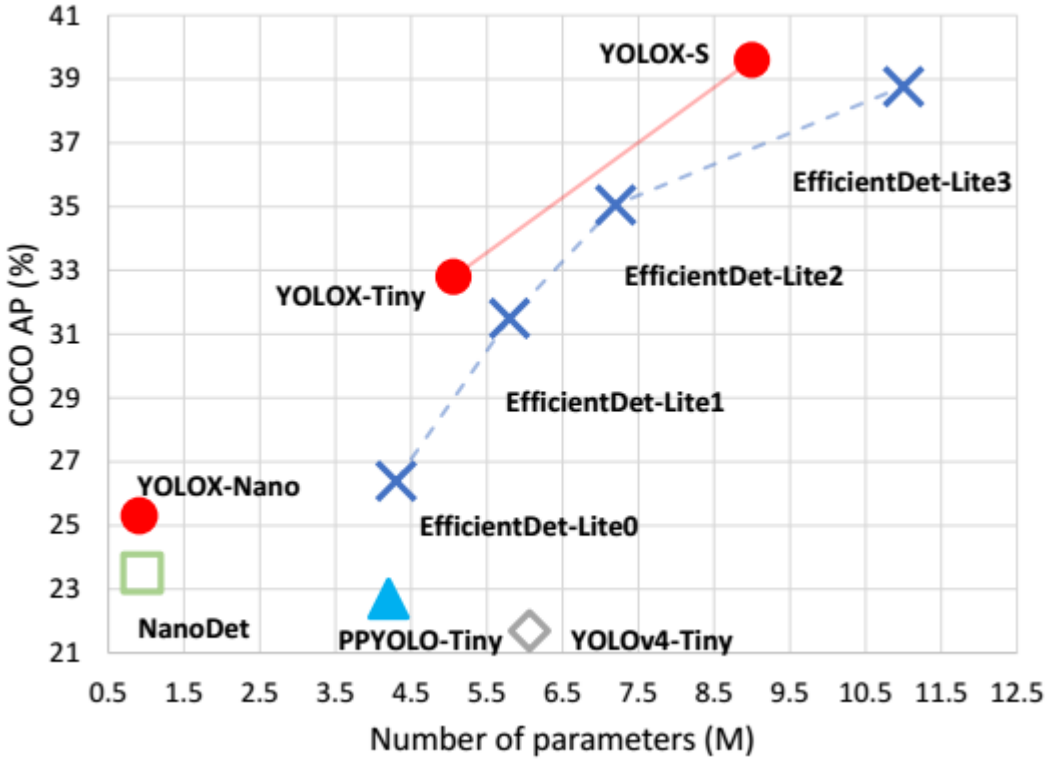


Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a 1 × 1 conv layer to reduce the feature channel to 256 and then add two parallel branches with two 3 × 3 conv layers each for classification and regression tasks respectively.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

3.4 Layer Wise Design Algorithms_part1
Model Compressing

Model Compressing

In model compressing some layers and Feature Maps(units)

Some Definitions:

Filter Part: The convolution and pooling layers (before the flat layer) which extract features from the spatial or temporal correlated inputs on an image or any other input signal. In addition for the sequenced inputs, If there are one or more than one RNN modules in the filter part, they are considered in the filter part, too.

FC Part: one, two, or more than two fully connected layers which are designed after the “Filter Part” which build an classifier or regression model for the extracted feature space.

Algorithm3:

1- Find and remove the last layers of the “Filter Part” which do not increase the SI(SmI), significantly.

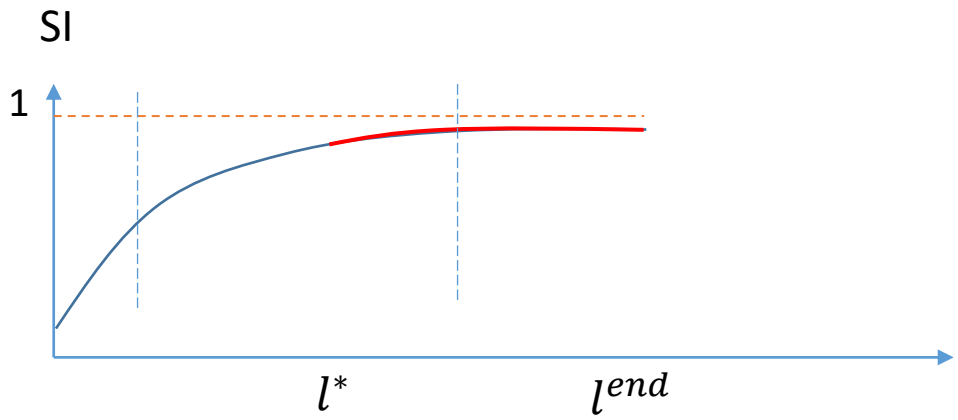
2- Find and remove all feature maps(units) of the last layer of the filter part which do not increase the SI(SmI), significantly.

3- Find and remove some other layers or feature maps form the filter part that by removing them, the separation index at the last layer of the filter part does not drop significantly

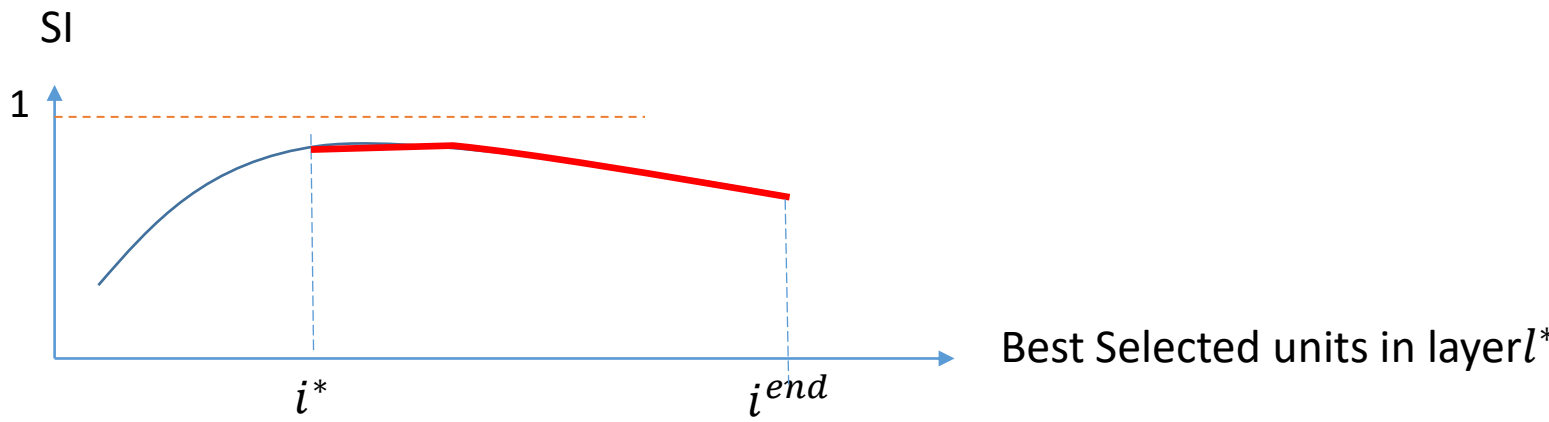
3- Design a new “FC part” with respect the number of units at the last later of the “filter part”.

Compressing Procedure

1.Compute SI(SmI) for layers of the DNN and then remove all last layers (after layer l^*) which do no increase SI(SmI), significantly.



2.Remove all extra feature maps(units) at layer l^* which do no increase SI(SmI), significantly.



3. Flat the units at layer l^* and add Fully Connoted layers and fine tune

TABLE VIII
Comparison with the prior art of VGG-16 on CIFAR-10.

Compressing method	Retraining needed	FLOPs	Pruned	Accuracy
VGG16-base	-	313.7M (0.0%)	0.0%	94.04%
PFEC **[17]**	Yes	206M (34.3%)	63.3%	93.40%
VP **[45]**	Yes	190M (39.1%)	73.4%	93.18%
SS **[46]**	Yes	183.13M (41.6%)	٧٣,٢%	٩٣,٠٢%
GAL-0.05 **[47]**	No	189.49M (39.6%)	77.2%	92.03%
CP **[48]**	No	107.58M (65.1%)	77.6%	92.03%
HRFM **[49]**	Yes	108.61M (65.3%)	82.1%	92.34%
GAL-0.1 **[47]**	No	171.89M (45.2%)	82.2%	90.73%
Our method	**No**	**75.6M (76.0%)**	**87.5%**	**93.49%**
HRFM **[49]**	Yes	73.70M (76.5%)	88.2%	91.23%

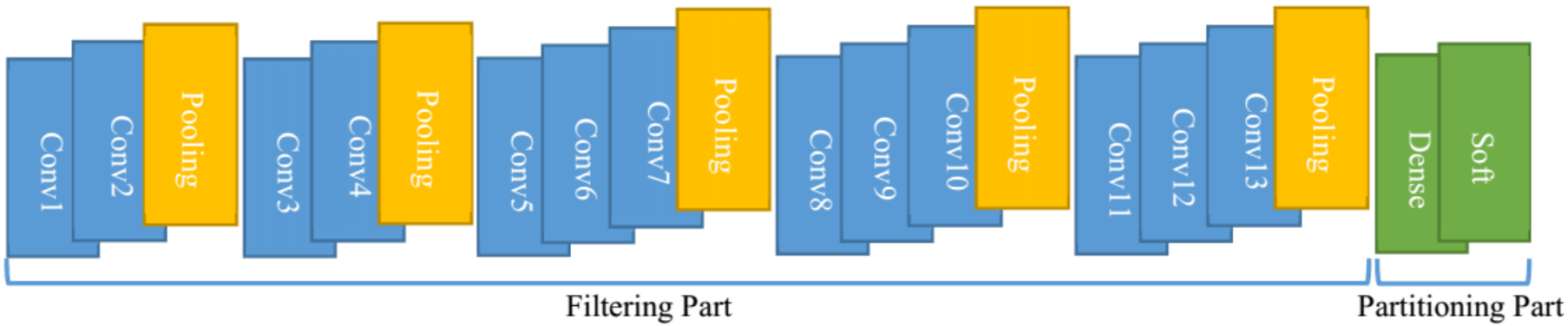


Fig. 5. The architecture of the "VGG-16" network.

TABLE IX
Comparison with the prior art of GoogLeNet on CIFAR-10.

Compressing method	Retraining needed	FLOPS	Pruned	Accuracy
GoogleNet-base	-	1.52B(0.0%)	0.0%	95.05%
PFEC **[17]**	Yes	1.0B(32.9%)	42.9%	94.54%
HRFM **[49]**	Yes	0.69B(54.9%)	55.4%	94.53%
GAL-Apo **[50]**	No	0.76B(50.0%)	53.7%	92.11%
GAL-0.05 **[48]**	No	0.94B(38.2%)	49.3%	93.93%
HRFM **[49]**	Yes	0.45B(70.4%)	69.8%	94.07%
Our method	**No**	**0.39B(74.4%)**	**77.6%**	**95.00%**

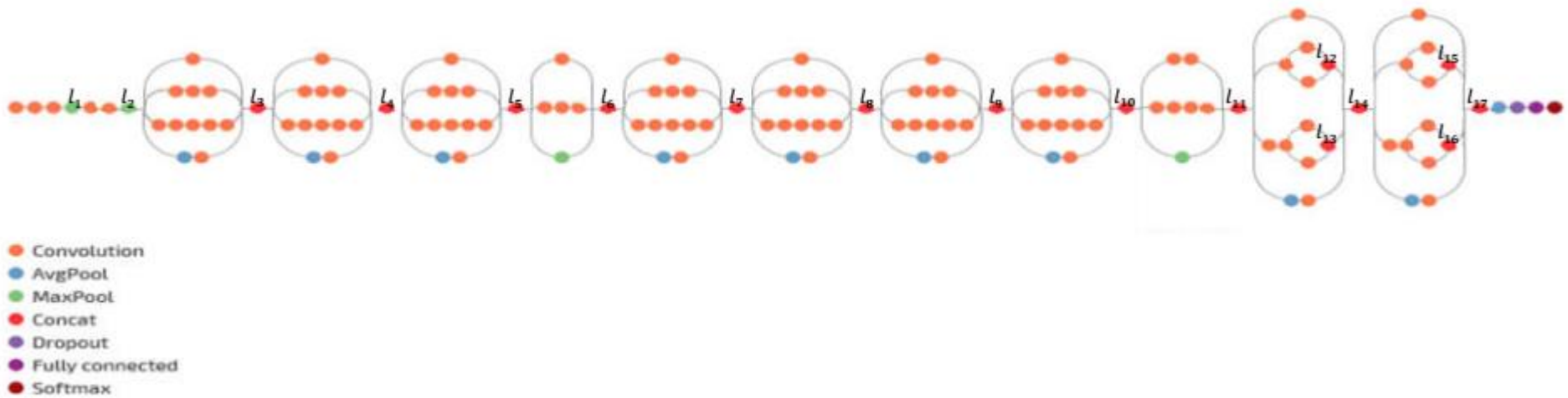
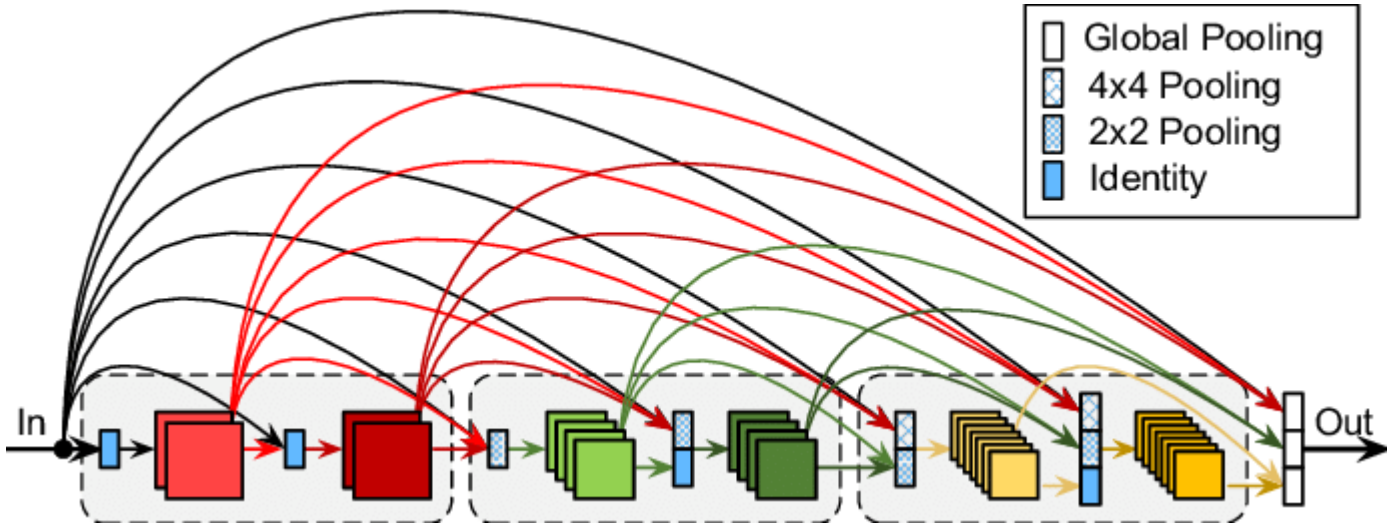


Fig. 7. The architecture of the "Inception V3" network.

TABLE X
Comparison with the prior art of DenseNet-40 on CIFAR-10.

Compressing method	Retraining needed	FLOPS	Pruned	Accuracy
DenseNet-base	-	0.29B(0.0%)	0.0%	94.22%
ECNS [51]	Yes	0.12B(58.3%)	67.2%	94.35%
HRFM [49]	Yes	0.11B(61.8%)	55.1%	94.53%
GAL-0.05 [48]	No	0.13B(55.6%)	57.9%	92.11%
VP [45]	No	0.16B(45.8%)	60.7%	93.16%
Our method	**No**	**0.07B(76.1%)**	**78.8%**	**94.17%**



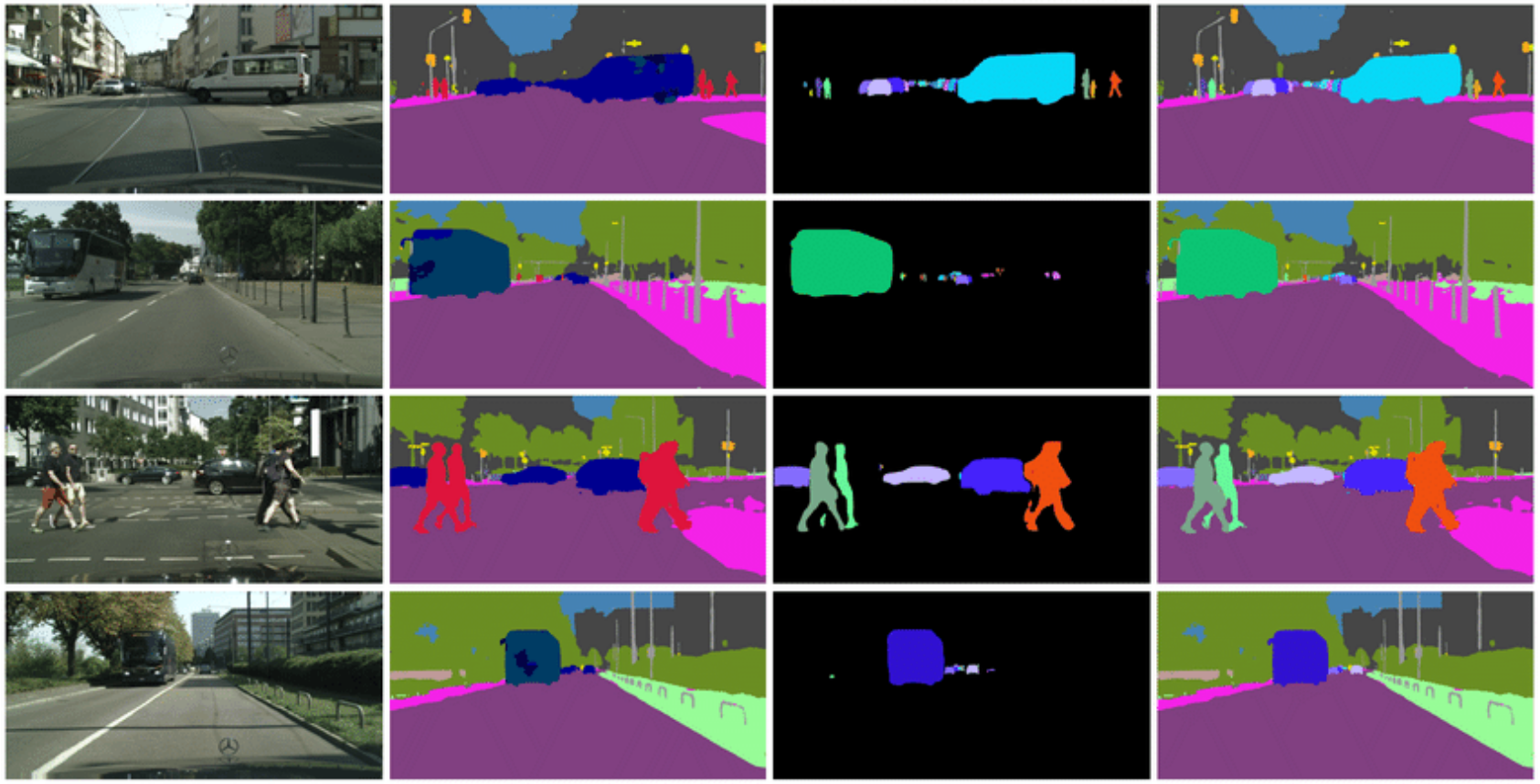
3.5 Region Based CNNs_part2

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

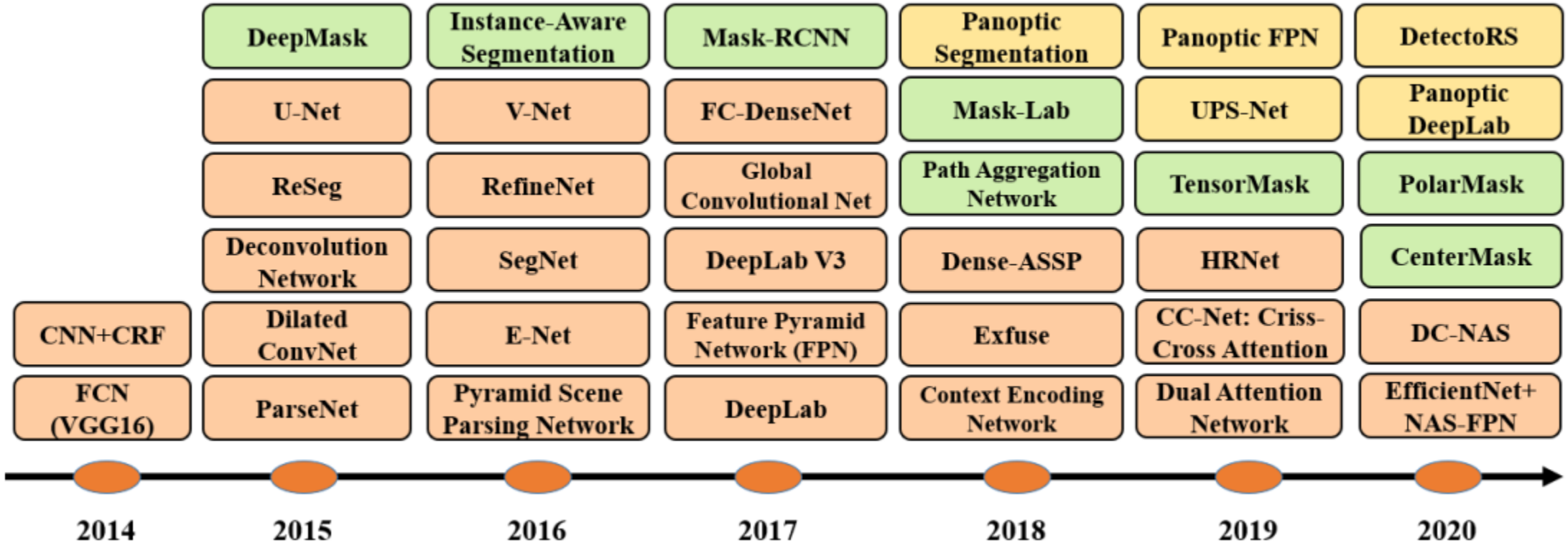
Panoptic segmentation is **proposed to unify the typically distinct tasks of semantic segmentation and instance segmentation**. The proposed task requires the generation of a rich and complete coherent scene segmentation, which is an important step towards a real-world visual system.



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.

Metrics For Segmentation Models

- **Pixel accuracy**

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

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

- **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

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

- **Intersection over Union (IoU)**

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

(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**

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

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

- **Dice coefficient**

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

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**

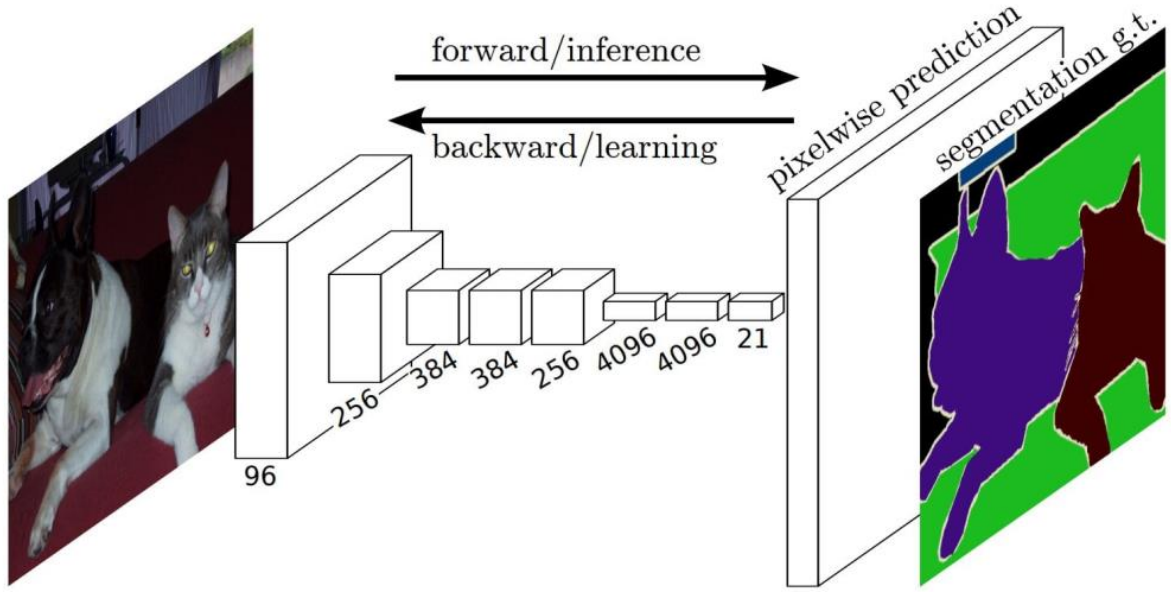
1. Fully Conventional Neural Networks

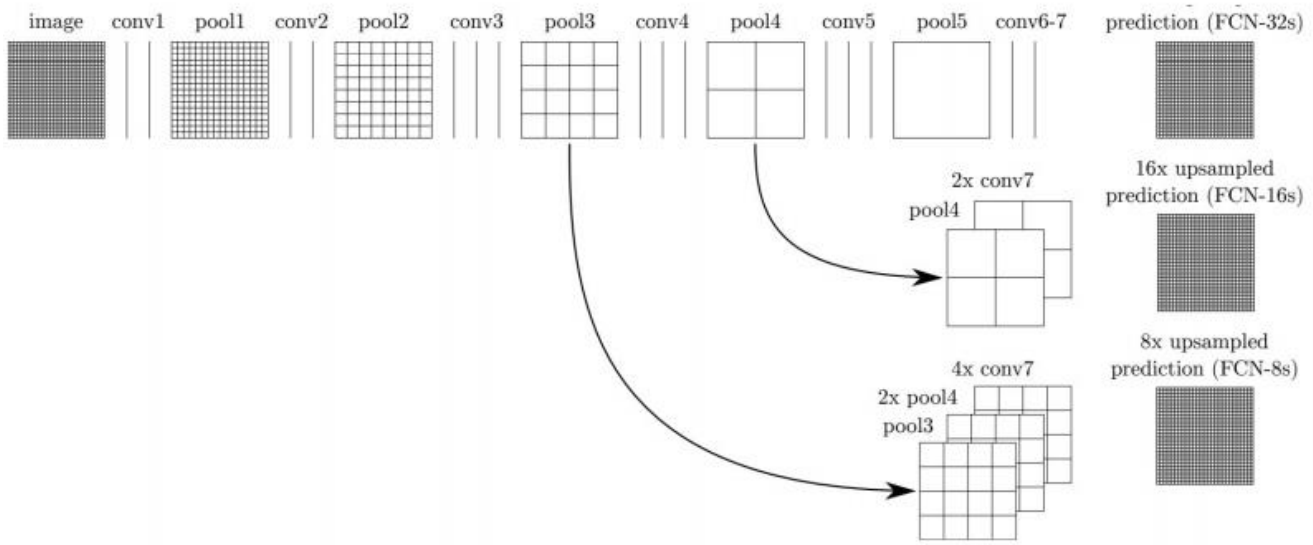
It includes only convolutional layers

The applied backbones is CNN architectures such as VGG16/GoogLeNet

FCN

J. Long... 2015

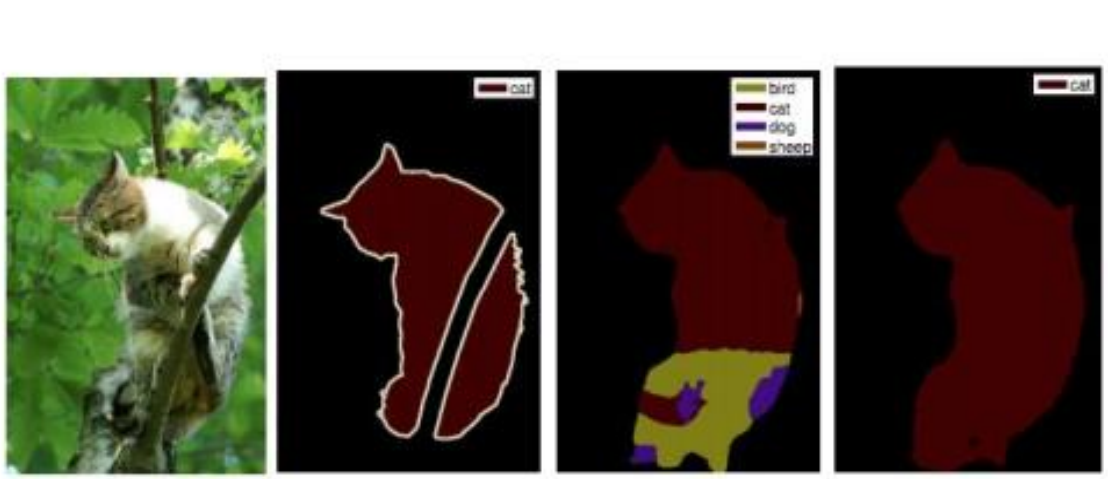




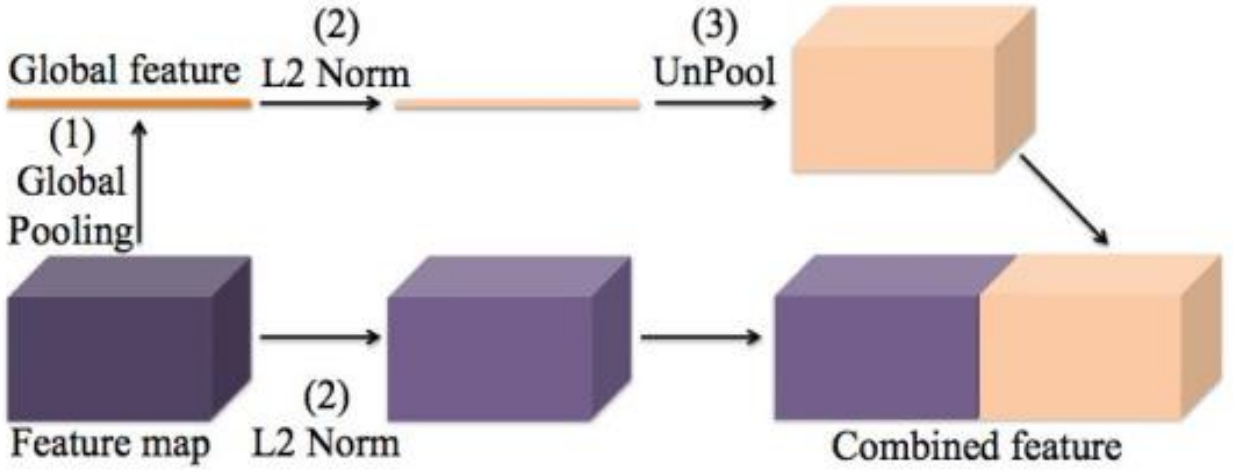
Through the use of skip connections in which feature maps from the final layers of the model are up-sampled and fused with feature maps of earlier layers, the model combines semantic information (from deep, coarse layers) and appearance information (from shallow, fine layers) in order to produce accurate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

ParseNet

W. Liu..2015



(a) Image (b) Truth (c) FCN (d) ParseNet



(e) ParseNet contexture module overview.

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)

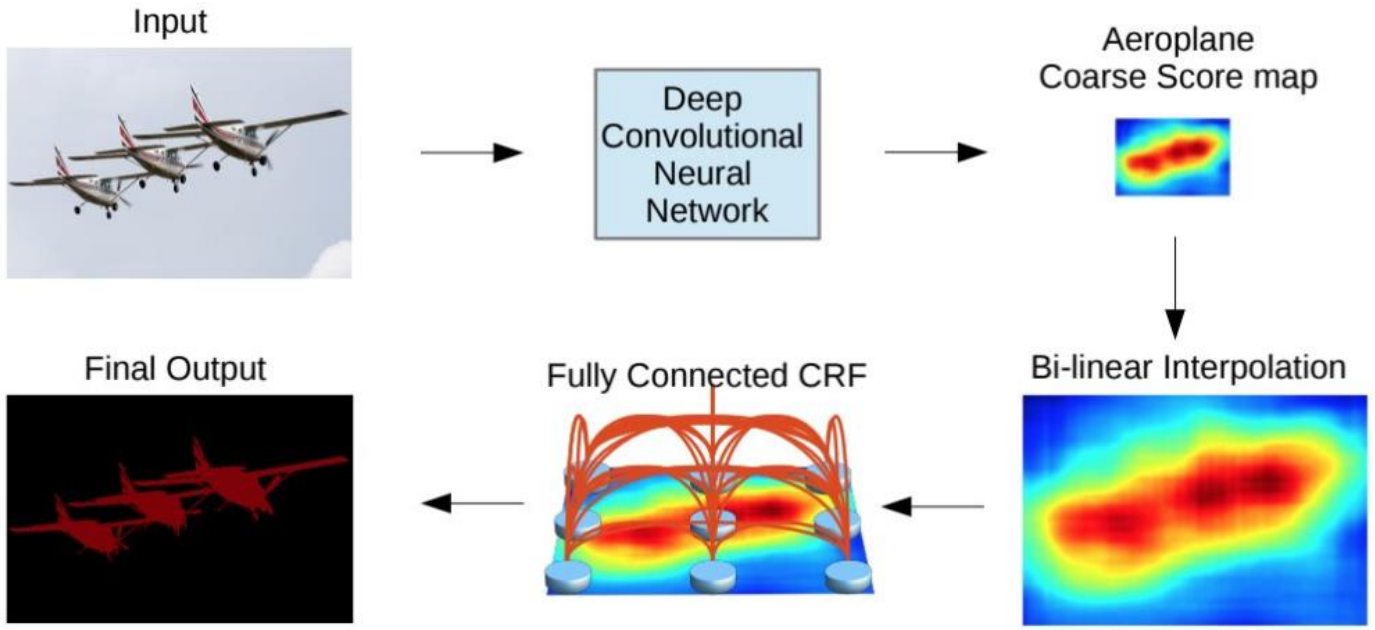
Chen..2014

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.

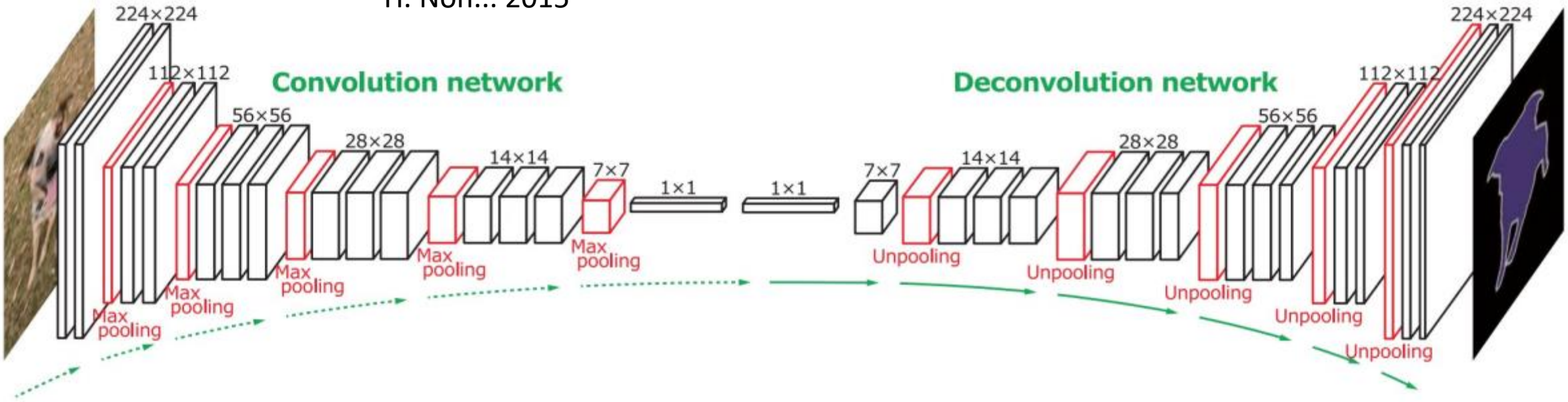
3. Encoder-Decoder Based Models

image segmentation based on the convolutional encoder-decoder architecture.

A. Encoder-Decoder Models for General Segmentation

DSS (Deconvolutional semantic segmentation)

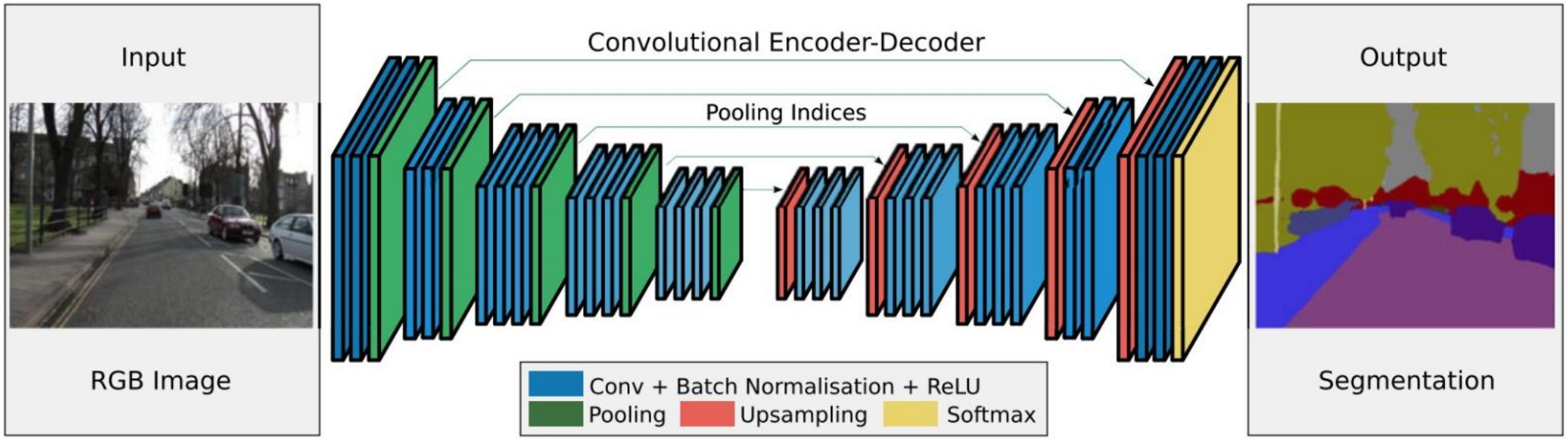
H. Noh... 2015



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)

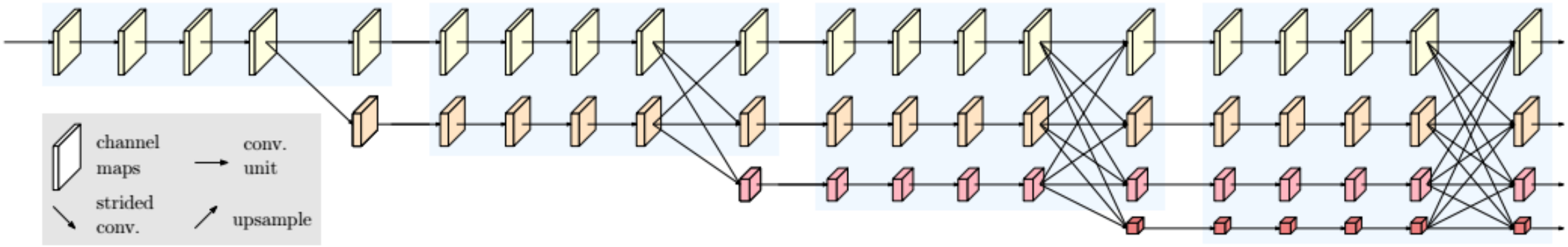
Y. Yuan...2019

Satge1
(high resolution Conv.)

Satge2
(Two resolution Blocks)

Satge3
(Three resolution Blocks)

Satge4
(Four resolution Blocks)



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

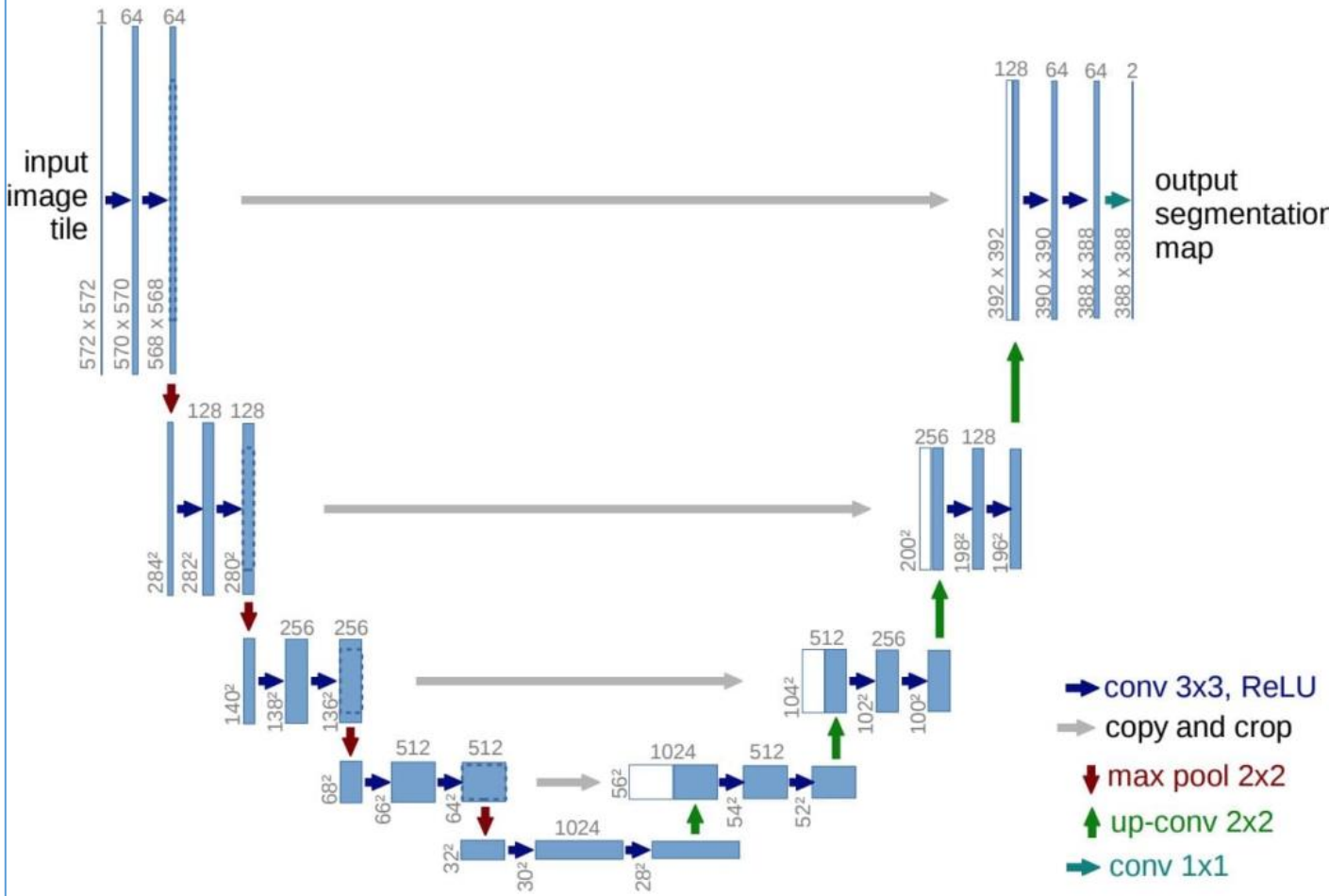
U-Net is proposed for segmenting biological microscopy images.

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.

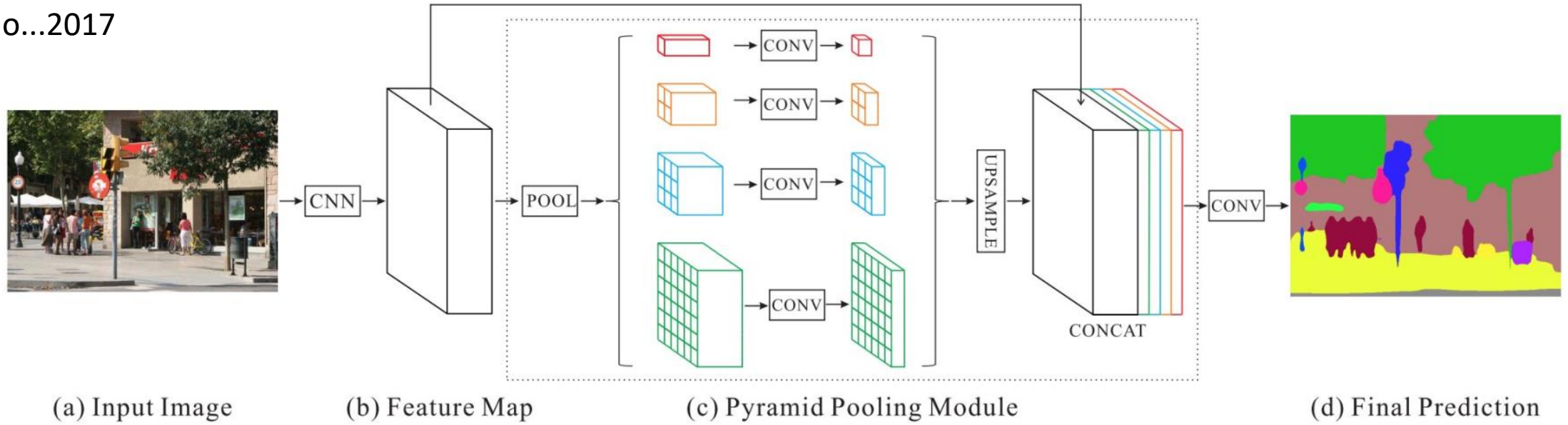


4. Multi-Scale and Pyramid Network Based Models

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

PSPN Pyramid scene parsing network

Zhao...2017

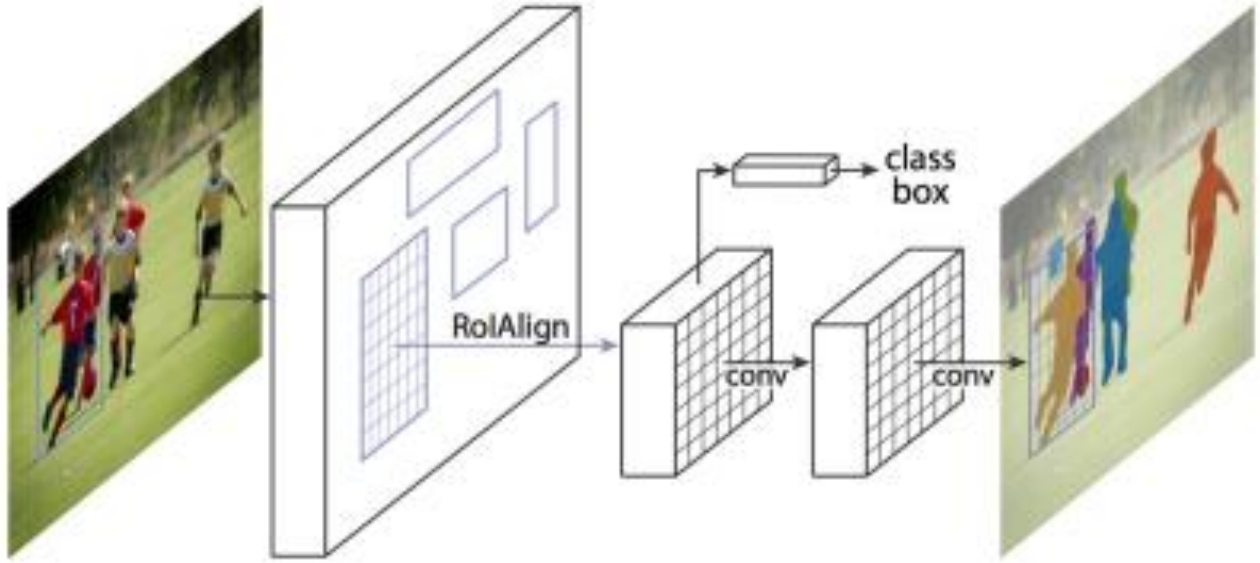


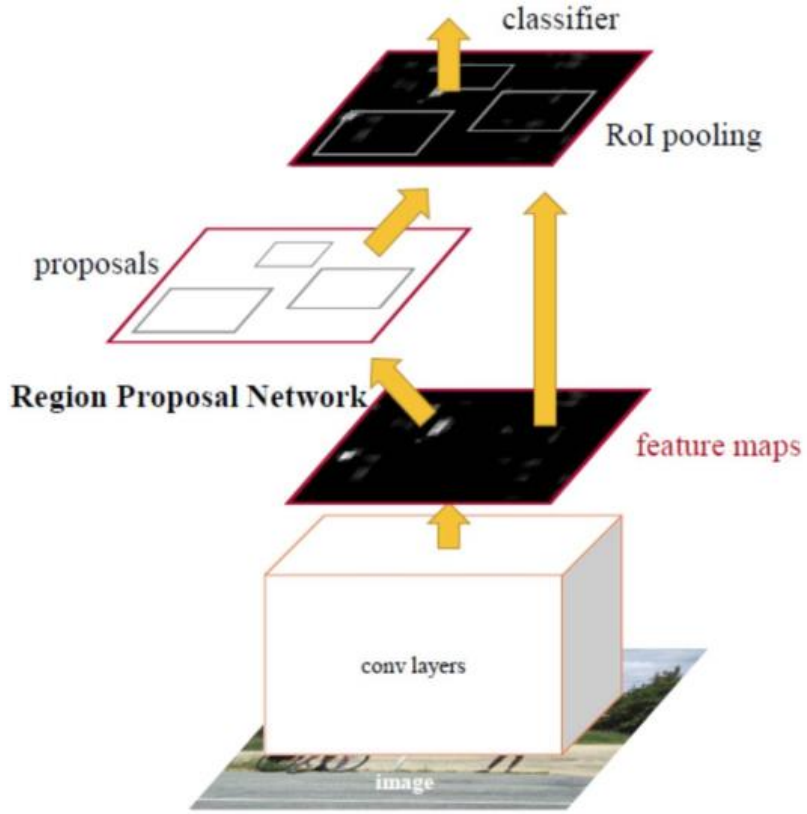
A CNN produces the feature map and a pyramid pooling module aggregates the different sub-region representations. Up-sampling and concatenation are used to form the final feature representation from which, the final pixel-wise prediction is obtained through convolution.

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

Mask R-CNN

K. He...2017





Mask R-CNN architecture for instance segmentation

MaskLab(Instance Segmentation)

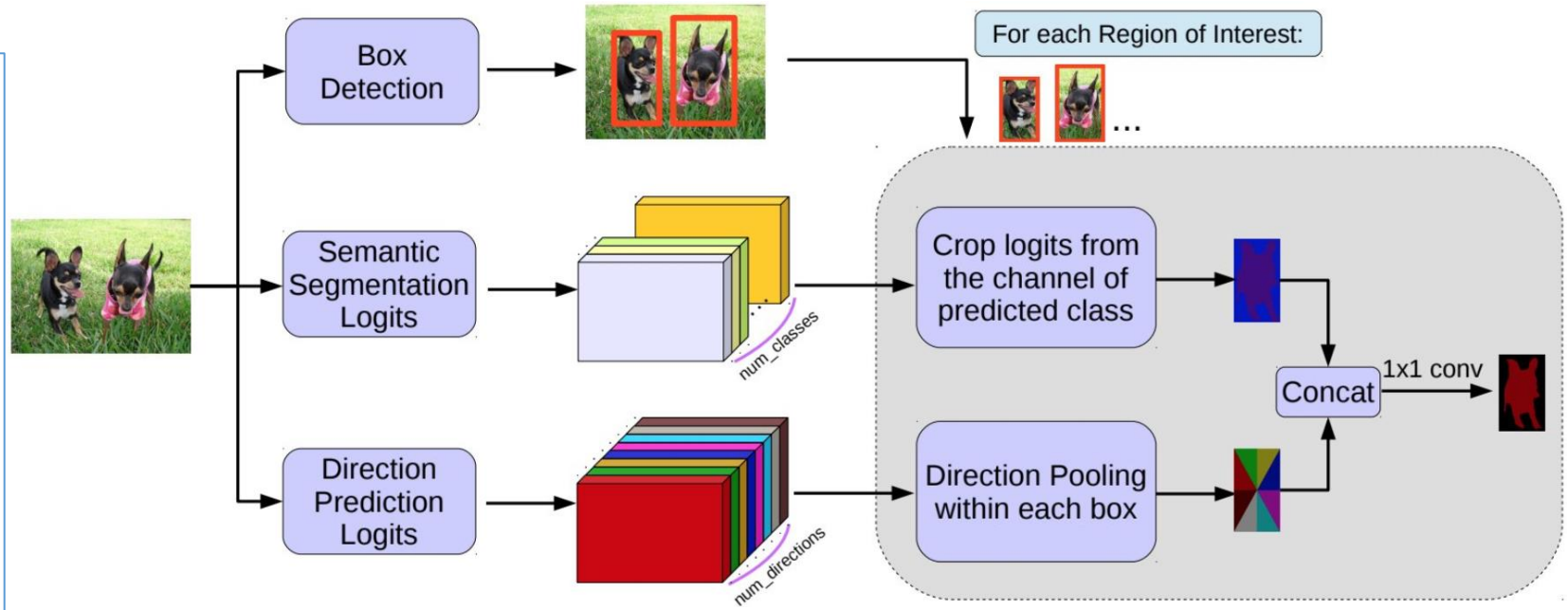
L. Chen...2018

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.



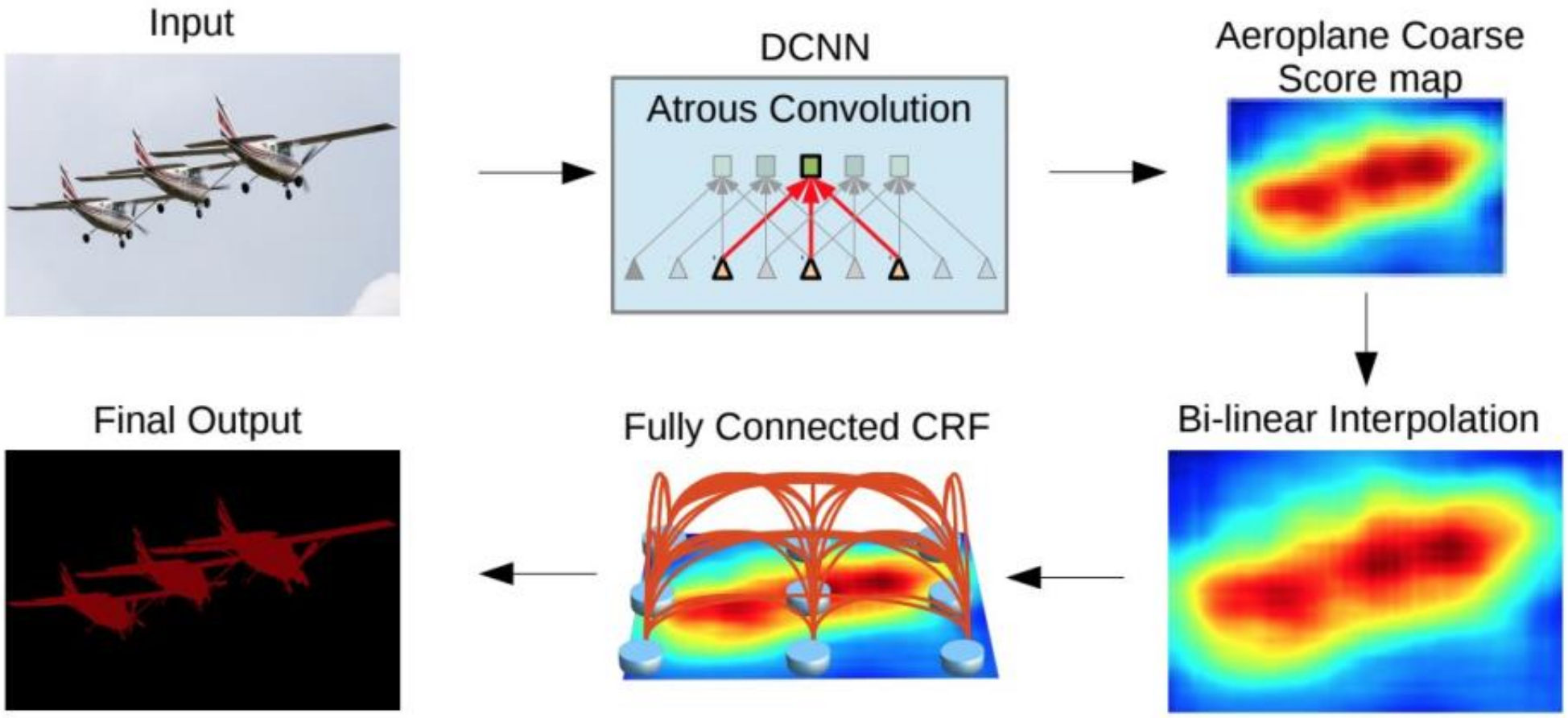
MaskLab generates three outputs—refined box predictions (from Faster R-CNN), semantic segmentation logits for pixel-wise classification, and direction prediction logits for predicting each pixel's direction toward its instance center.

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.

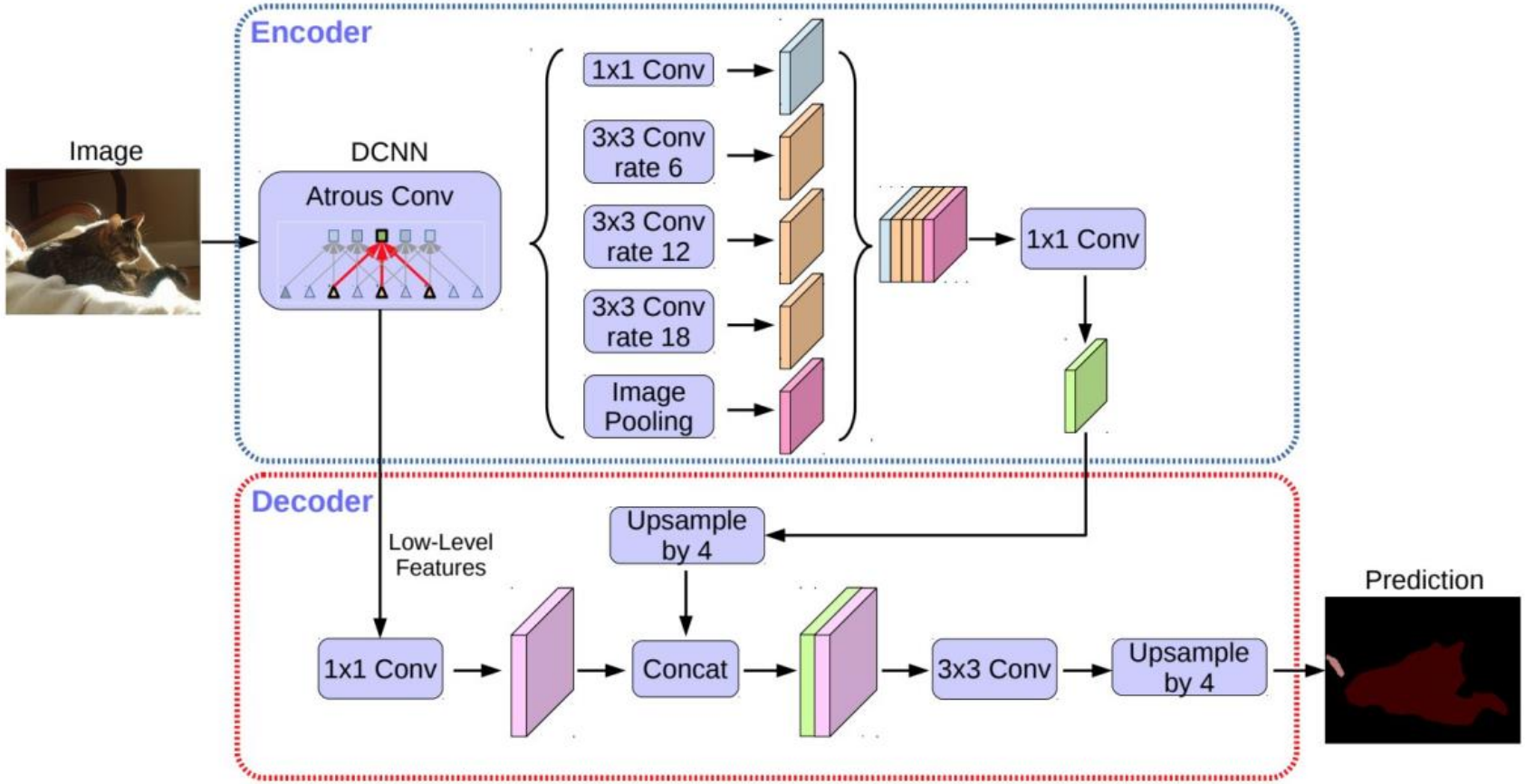
DeepLabv3+

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.

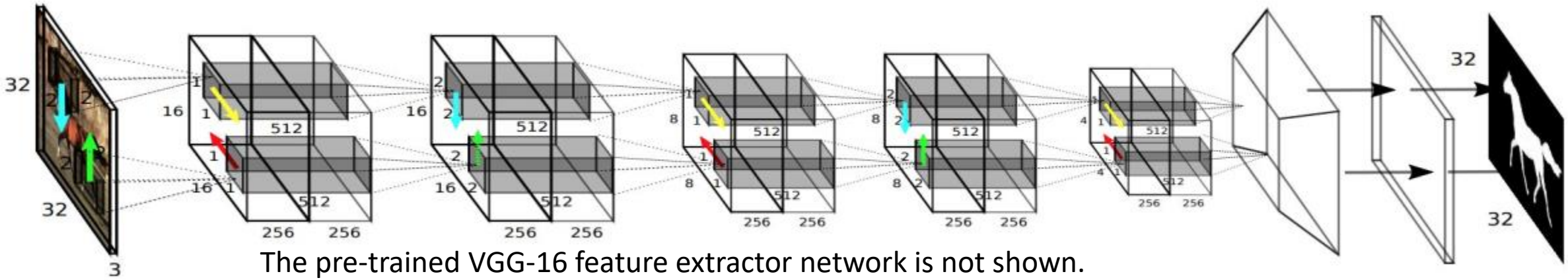
The most relevant model has a modified Xception backbone with more layers, dilated depthwise separable convolutions instead of max pooling and batch normalization.



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.

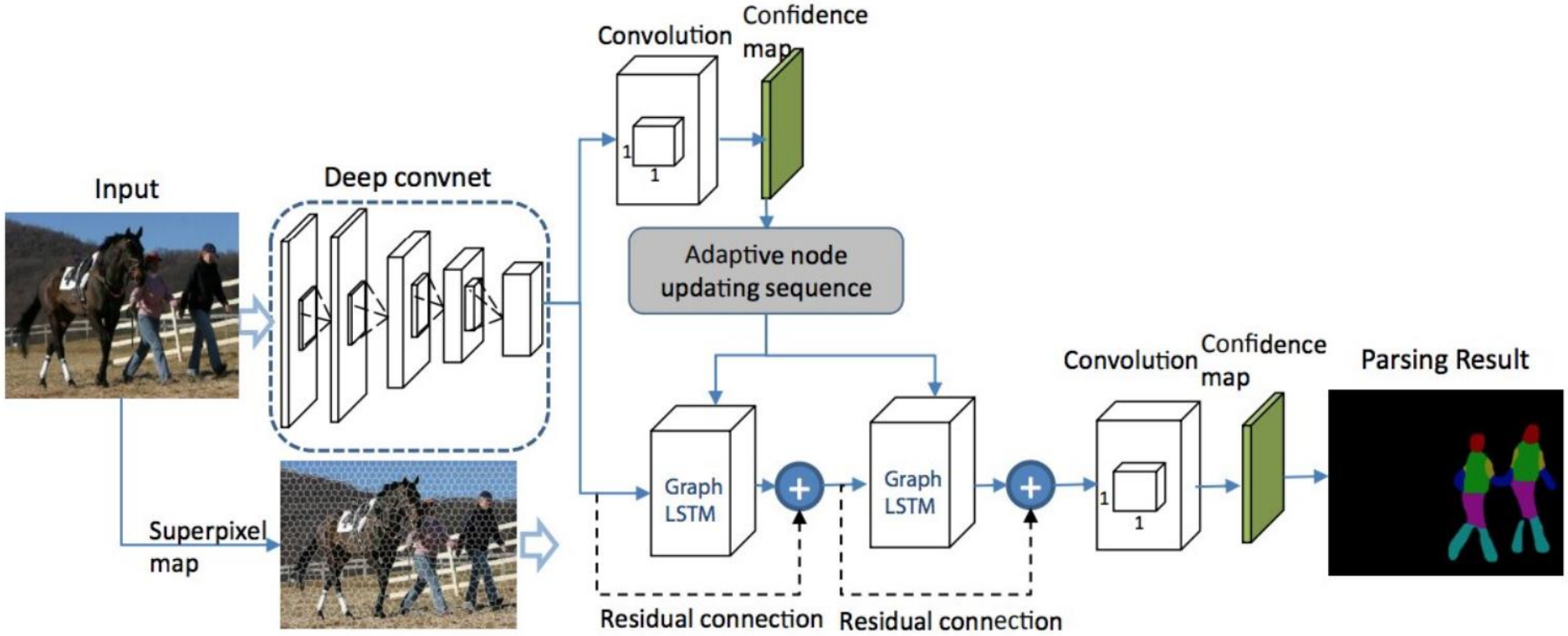
To perform image segmentation with the ReSeg model , ReNet layers are stacked on top of pre-trained VGG-16 convolutional layers that extract generic local features.

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).

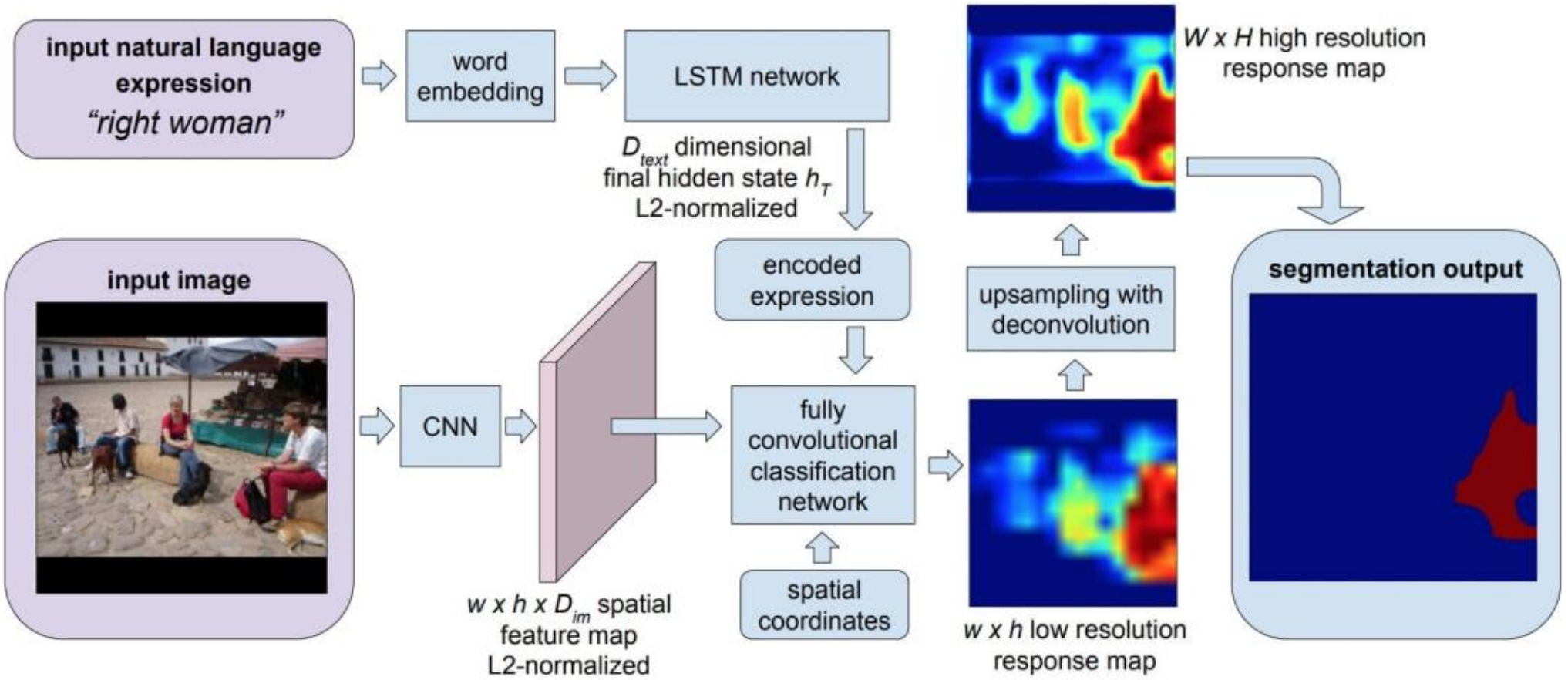
Instead of evenly dividing an image to pixels or patches in existing multi-dimensional LSTM structures (e.g., row, grid and diagonal LSTMs), they take each arbitrary-shaped super pixel as a semantically consistent node, and adaptively construct an undirected graph for the image, where the spatial relations 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

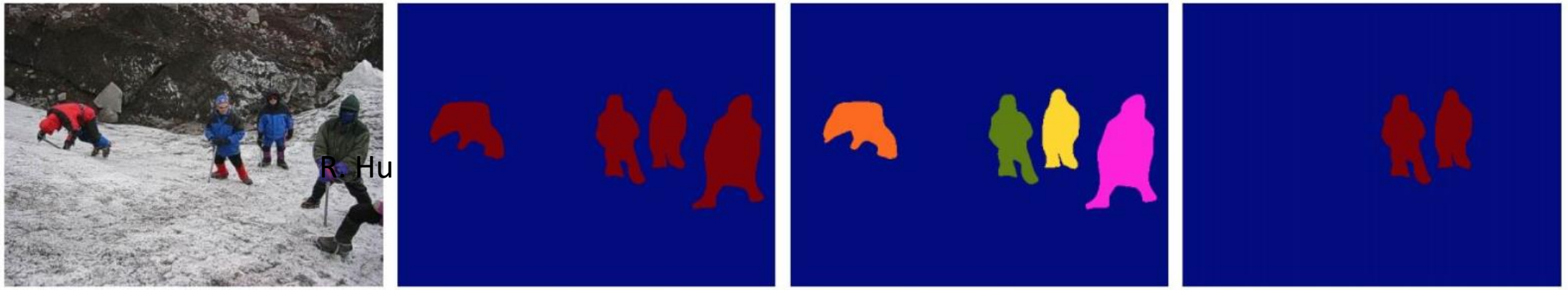
R. Hu...2016

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.

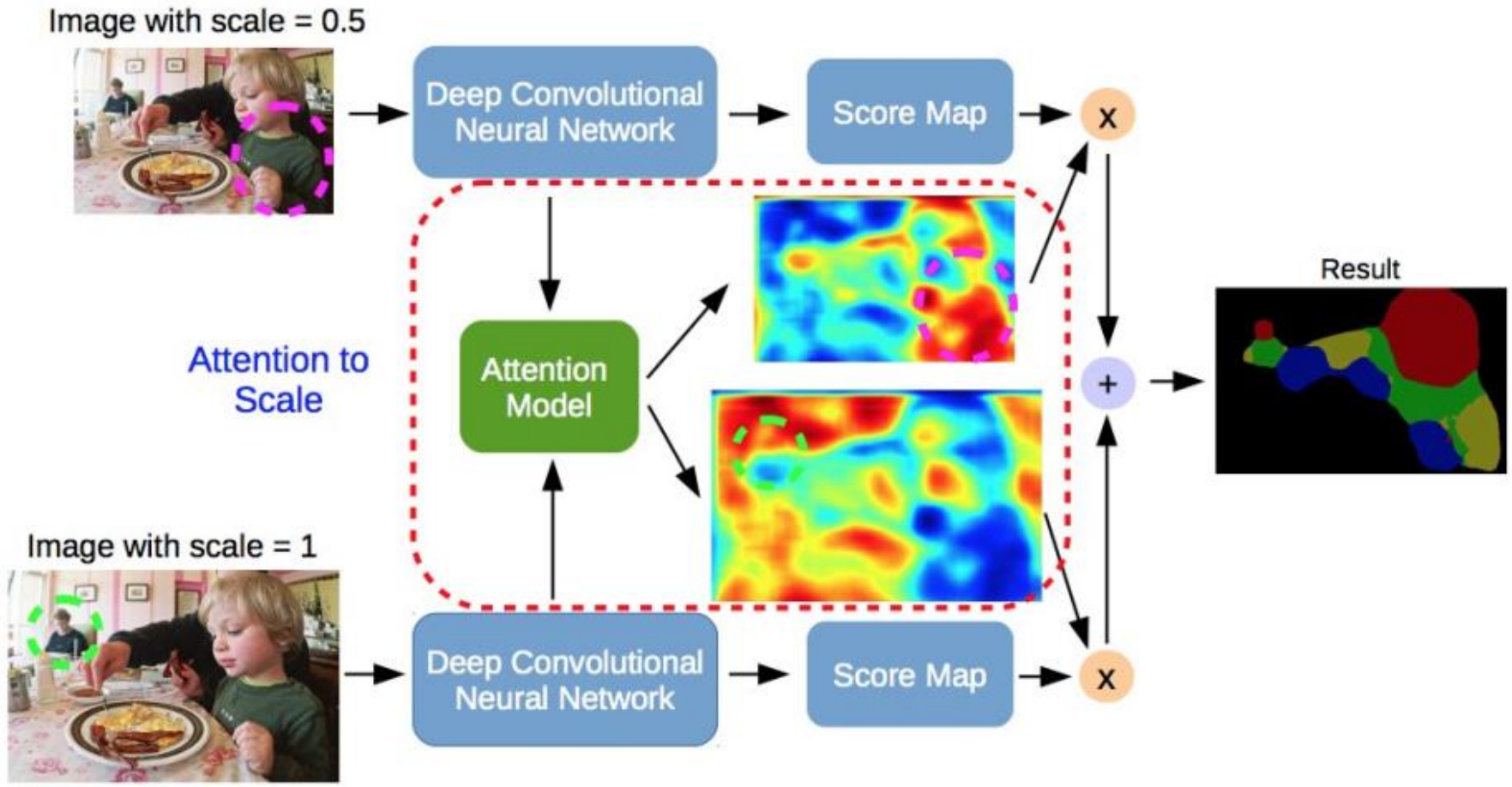


Segmentation masks generated for the query “people in blue coat”

8. Attention-Based Models

Attention-based semantic segmentation model

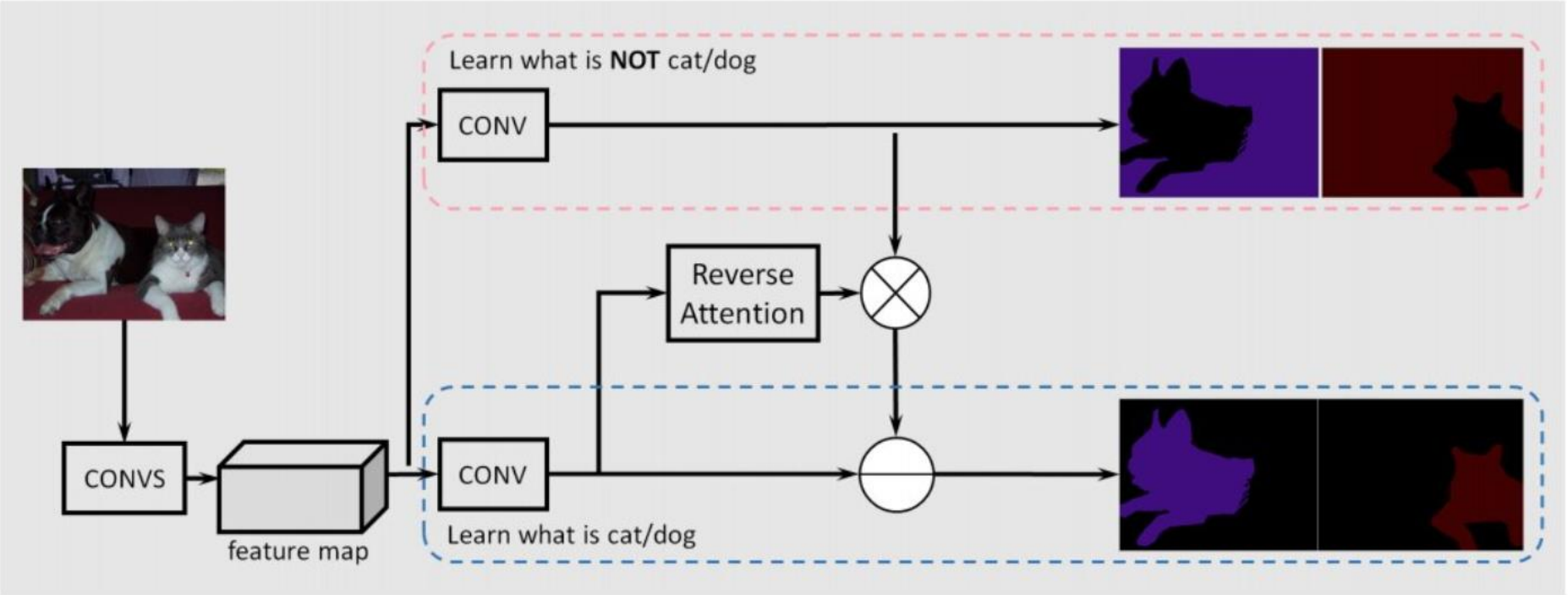
Chen ...2016



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**,”

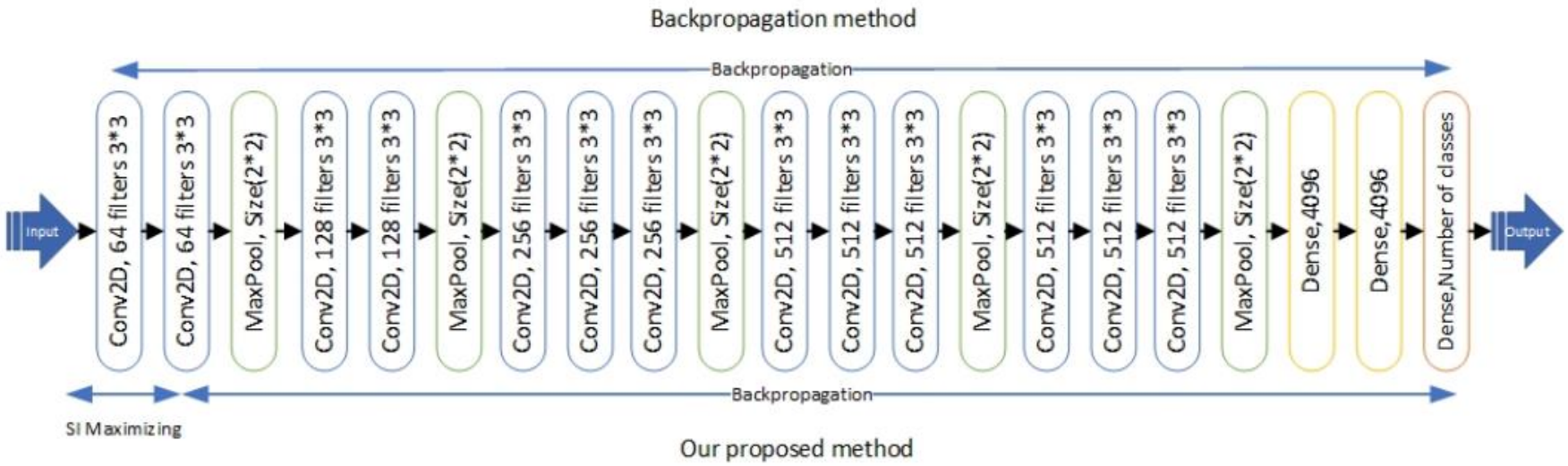
3.6 Layer Wise Design Algorithms_part2

Layer-wise forward learning

3.6.1 Forward learning in the first layer

For classification problems but the method can be generalized for regression problems

Forward learning in the first layer



$\{\boldsymbol{x}^q\}_{q=1}^{Q}$ $\xrightarrow{\text{Filters: } \{(\boldsymbol{\theta}_l, b_l)\}_{l=1}^{M} + \text{Relu}}$ $\{\mathbf{z}^q\}_{q=1}^{Q}$

Input layer Quasi - LS First conv.layer

Figure 3: Applying M convolutional filters and biases to the input patterns and then activating the convolved units by Relu function the feature maps are resulted.

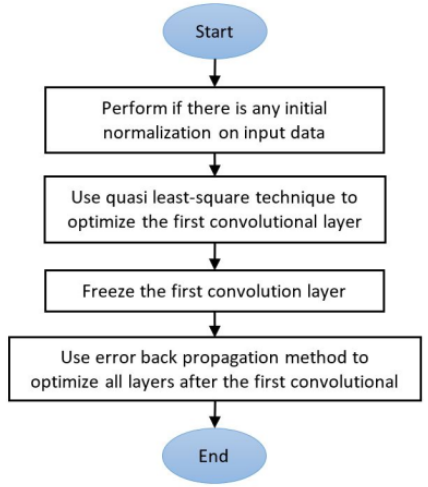


Figure 2: This flowchart indicates the learning method in which the quasi-LS is utilized to learn the first convolution layer, and other layers are learned by an error backpropagation method.

Algorithm to train the first layer by Quasi Least Square (QLS) technique

1. Define the matrix of patches from the input data: X
2. Subtract the matrix from their mean: $\widetilde{\mathrm{X}} = X - mean(X)$
3. Define Auto-correlation matrix of patches: $\mathrm{W_{np}} = \widetilde{\mathrm{X}}^{\mathrm{T}}\widetilde{\mathrm{X}}$
4. Apply SVD technique to get Eigen values and Eigen vectors: $W = \sum_{i=1}^{n_p} \lambda_i v_i v_i^T$
5. Rank Eigen vectors by their Eigen values and then initiate all filters parameters and the biases by Eigen vectors, their negatives and the mean of patches.

   $\left\{\left\{\boldsymbol{\theta}^j, b^j\right\}\right\}_{j=1}^{n_F}$ n_F:number of filters
6. Based on the filter parameters at the first (convolutional)layer compute the second layer and for each example(anchor) find nearest neighbor for both positive and negative examples and define triplet loss for all data batch.

   $$J_{triplet} = \sum_{i=1}^{m}(\left\|x_i^2 - x_{ipos}^2\right\|^2) - \sum_{i=1}^{m}(\left\|x_i^2 - x_{ineg}^2\right\|^2)$$
7. Use a QLS technique to minimize $J_{triplet}$ and update all filters.

   $\boldsymbol{\theta}^j := \boldsymbol{\theta}^j + \mu\, \widetilde{\boldsymbol{\theta}}^j$ $\quad b^j := b^j + \mu \tilde{b}^j$ $\quad \mu$=learning rate
8. Repeat steps 6 and 7 for several epochs to get maximum possible SI on the second layer.

Now, one can train further layers by Backpropagation Algorithms.

Comparison between our method and backpropagation algorithm

Dataset	Learning Method	**AlexNet**	**VGG16**	**ResNet50**	**InceptionV3**
CIFAR10	Backpropagation	84.61	92.95	93.17	94.20
	Our proposed method	**85.84**	**94.81**	**95.02**	**95.43**
	Percentage of improvement	1.23	1.86	1.85	1.23
CIFAR10 - (plane,truck)	Backpropagation	96.45	97.18	97.64	97.09
	Our proposed method	**97.09**	**98.36**	**98.49**	**97.77**
	Percentage of improvement	0.64	1.18	0.85	0.68
CIFAR10 - (plane,cat,bird)	Backpropagation	96.21	96.80	96.88	96.81
	Our proposed method	**96.62**	**97.63**	**97.16**	**97.14**
	Percentage of improvement	0.41	0.83	0.28	0.33
CIFAR100	Backpropagation	62.22	70.98	75.30	76.31
	Our proposed method	**62.45**	**71.74**	**75.58**	**76.72**
	Percentage of improvement	0.23	0.76	0.28	0.41
FASHION-MNIST	Backpropagation	92.53	94.17	95.24	95.78
	Our proposed method	**92.65**	**94.73**	**95.38**	**95.91**
	Percentage of improvement	0.12	0.56	0.14	0.13

Table 5: Comparing the accuracy of our proposed method in different architectures and datasets with backpropagation method

3.6.2 Layer-wise forward learning

Layer Wise Forward Learning

Algorithm

For L=1:L_{final}

While (SI ($Data^l$) may still increase)

For $l = 1: L$

$$\boldsymbol{g}_{\boldsymbol{p}^l} = \frac{\partial \text{Loss}_{\text{triplet}}(Data^l)}{\boldsymbol{p}^l}$$

$$\boldsymbol{p}^l := \boldsymbol{p}^l - \gamma \boldsymbol{g}_{\boldsymbol{p}^l}$$

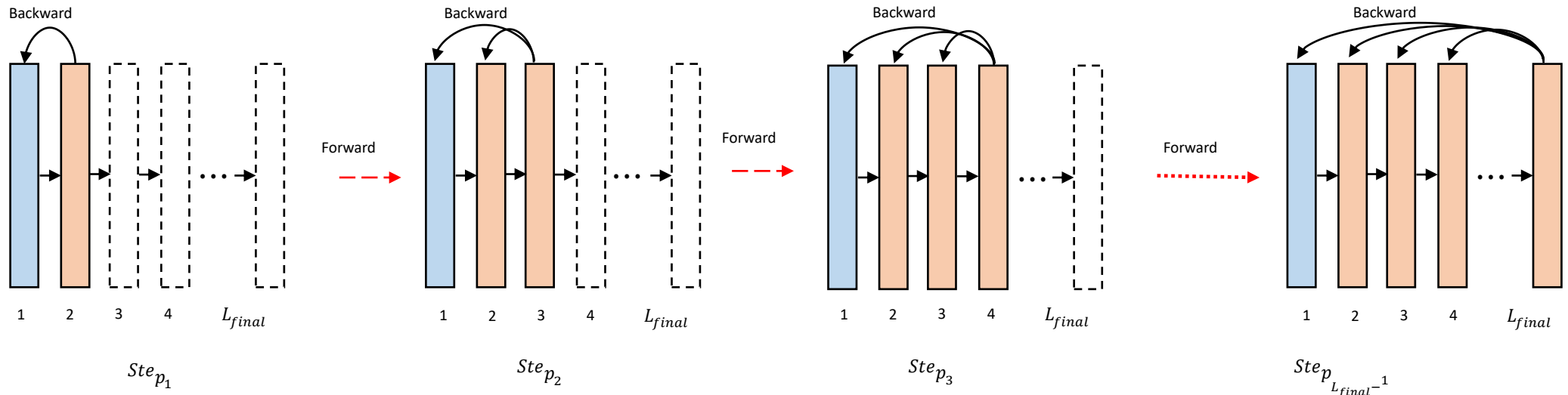
$Data^l = \{(\boldsymbol{x}_i^L, \boldsymbol{y}_i^L)\}$ *Data at layer l*

Triplet Loss

$$\text{Loss}_{\text{triplet}}(Data^L) = \frac{1}{m}\sum_{i=1}^{m}\left(\left\|\boldsymbol{x}_i^L - \boldsymbol{x}_{i_p^*}^L\right\|^2 - \left\|\boldsymbol{x}_i^L - \boldsymbol{x}_{i_n^*}^L\right\|^2\right)$$

$$i_p^* = \underset{\forall q \neq i}{\arg\ min} \frac{1}{\delta(\boldsymbol{y}_i, \boldsymbol{y}_q) + \varepsilon}\left\|\boldsymbol{x}_i^L - \boldsymbol{x}_q^L\right\|^2 \quad \varepsilon \to 0^+$$

$$i_n^* = \underset{\forall q \neq i}{\arg\ min} \frac{1}{1 - \delta(\boldsymbol{y}_i, \boldsymbol{y}_q) + \varepsilon}\left\|\boldsymbol{x}_i^L - \boldsymbol{x}_q^L\right\|^2$$



Comparison between our method and backpropagation algorithm

VGG١٦	ResNet٥٩	InceptionV٣	EfficientNetB٠		
٩٢.٩٥	٩٣.١٧	٩٤.٢٠	٩٨.١١	Backpropagation	CIFAR١٠
٩٣.٦٧	**٩٤.٦٥**	**٩٤.٦٣**	**٩٨.٢٤**	Our method	
٧٠.٩٨	٧٥.٣٠	٧٦.٣١	٨٨.١٤	Backpropagation	CIFAR١٠٠
٧١.١٩	**٧٥.٤٧**	**٧٦.٤٩**	**٨٨.١٧**	Our method	
٩٤.١٧	٩٥.٣٨	٩٥.٩١	٩٧.٢٣	Backpropagation	Fashion-MNIST
٩٤.٥٢	**٩٦.٦١**	**٩٦.٢١**	**٩٧.٤١**	Our method	

index is feasible by training the first layers with the Forward learning approach.

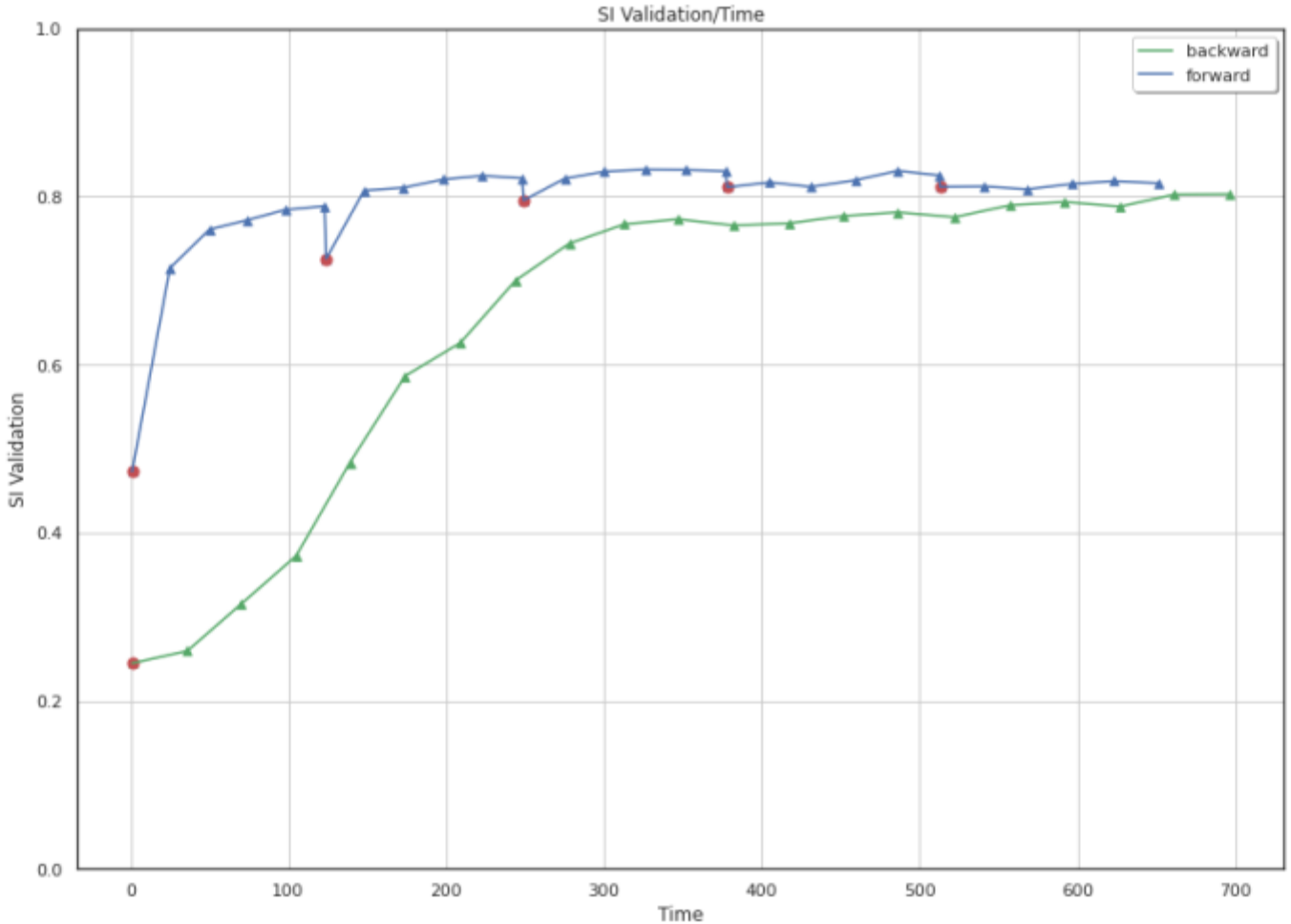


Figure 2. Comparing the separation index of the forward learning and backward learning approaches over time. Each triangle shows a single epoch in the learning process and the red dots in the Forward learning method are convolutional layers.

The Text Classification Error rate

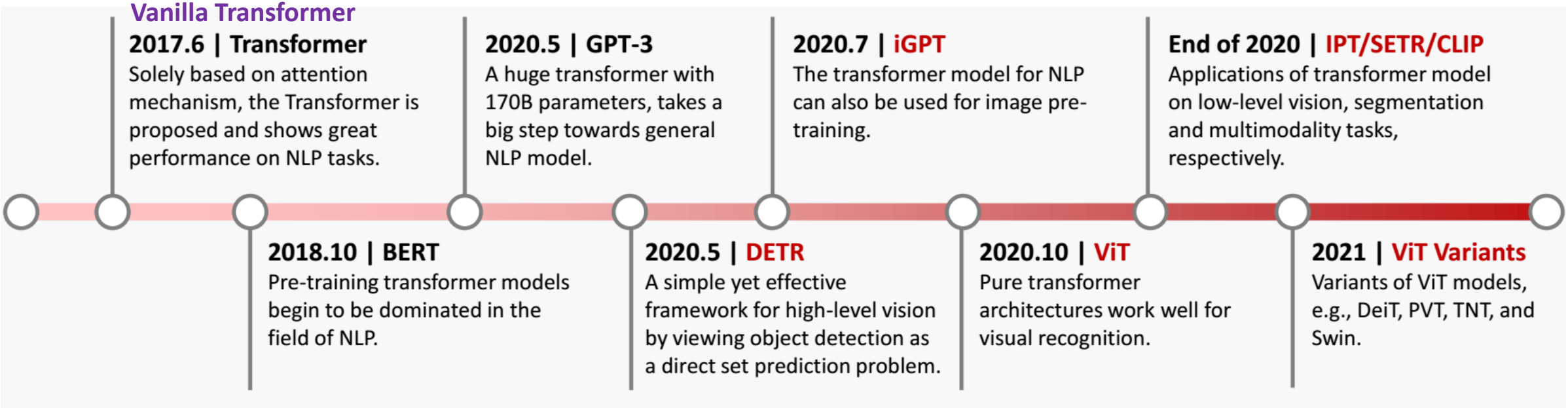
Dataset	Learning Strategy	VD-CNN(Depth=9)	VD-CNN(Depth=17)	VD-CNN(Depth=29)
AG's News	Our Method	9.35	8.71	8.72
	Backpropagation (Cross Entropy)	10.17	9.29	9.36
DBPedia	Our Method	1.53	1.30	1.23
	Backpropagation (Cross Entropy)	1.64	1.42	1.36

3.7 Architecture of Transformers

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

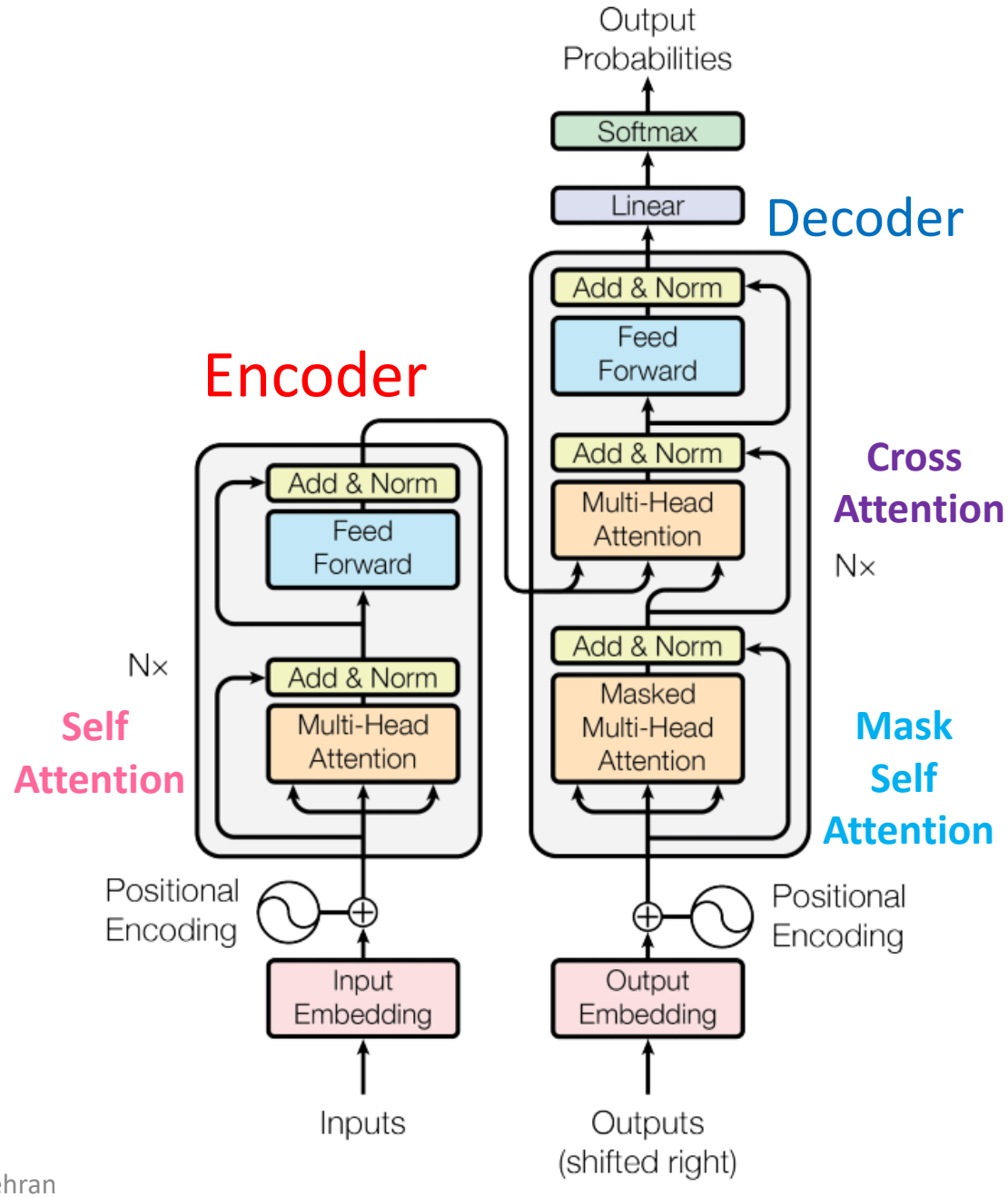


The vision transformer models are marked in red.

Vanilla Transformer

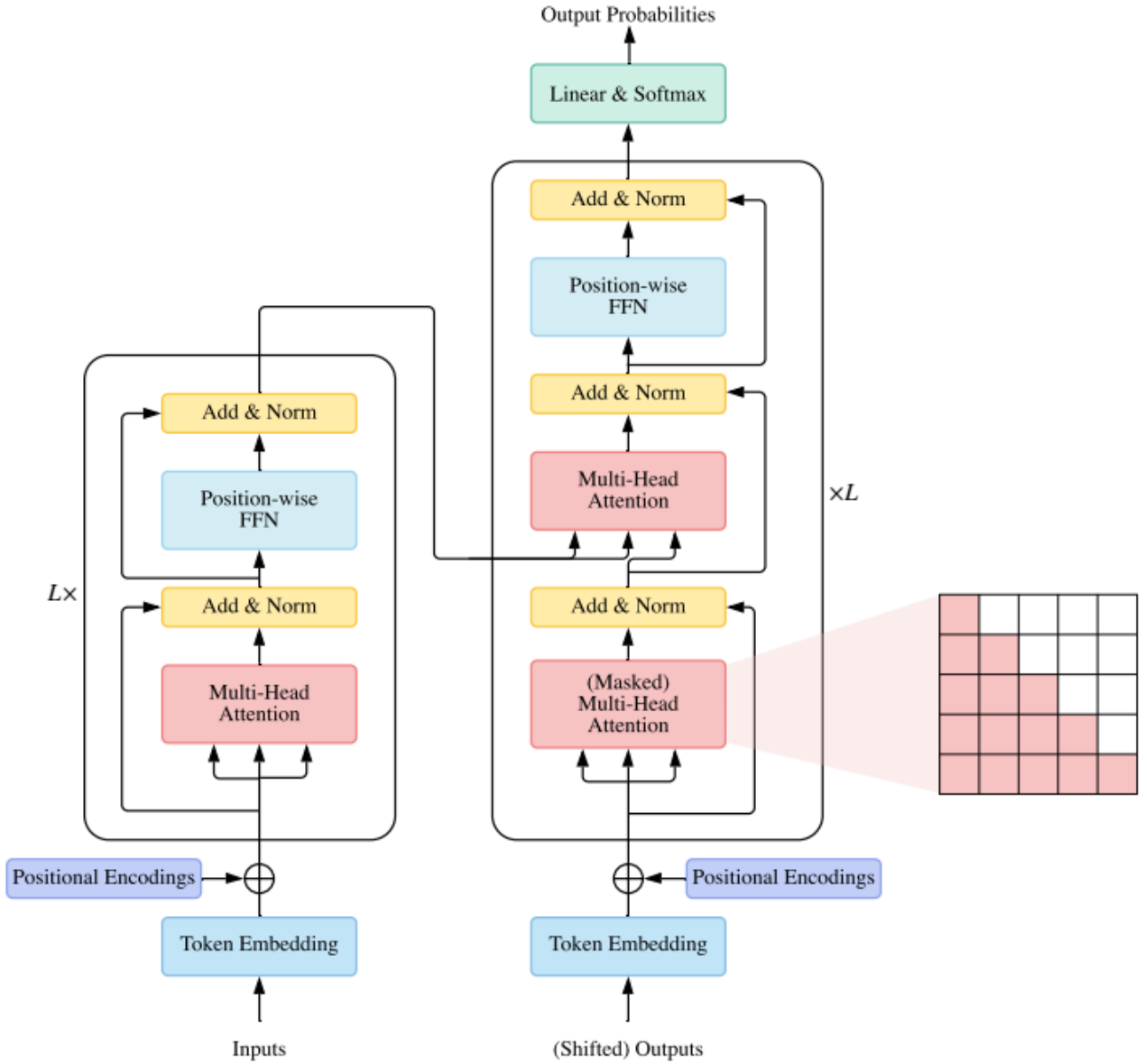
*Vaswani...2017 (Attention is all you need)

- 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.



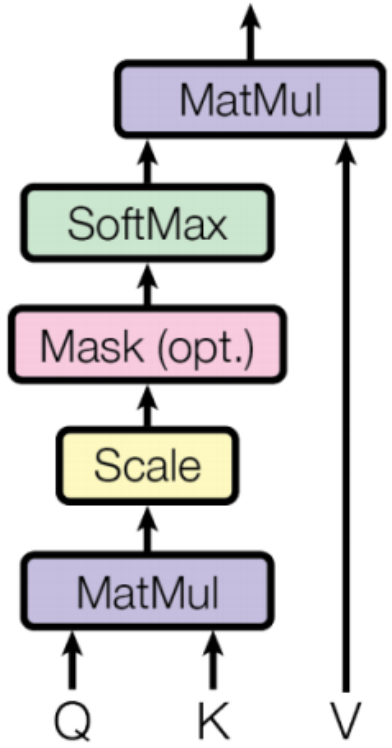
The key modules of the vanilla Transformer

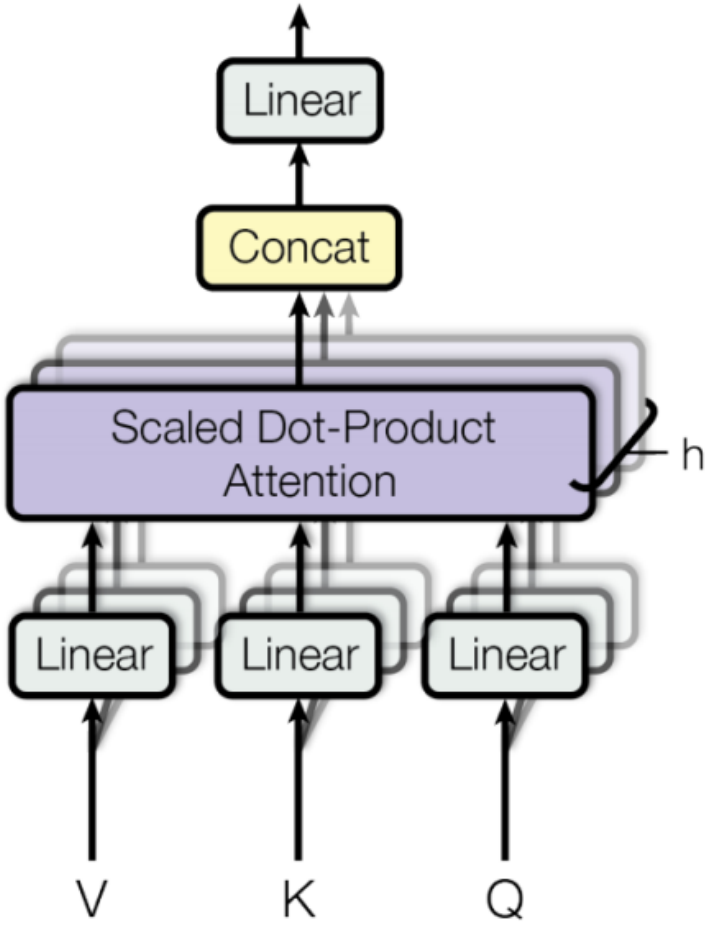
1. **Attention Modules**
   - Single Attention
   - Multi Head-Attention
     - Self-attention
     - Masked Self-attention
     - Cross-attention
2. **Position-wise FFN** (Feed Forward Network)
3. **Residual Connection and Normalization.**
4. **Position Encodings**.



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

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

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

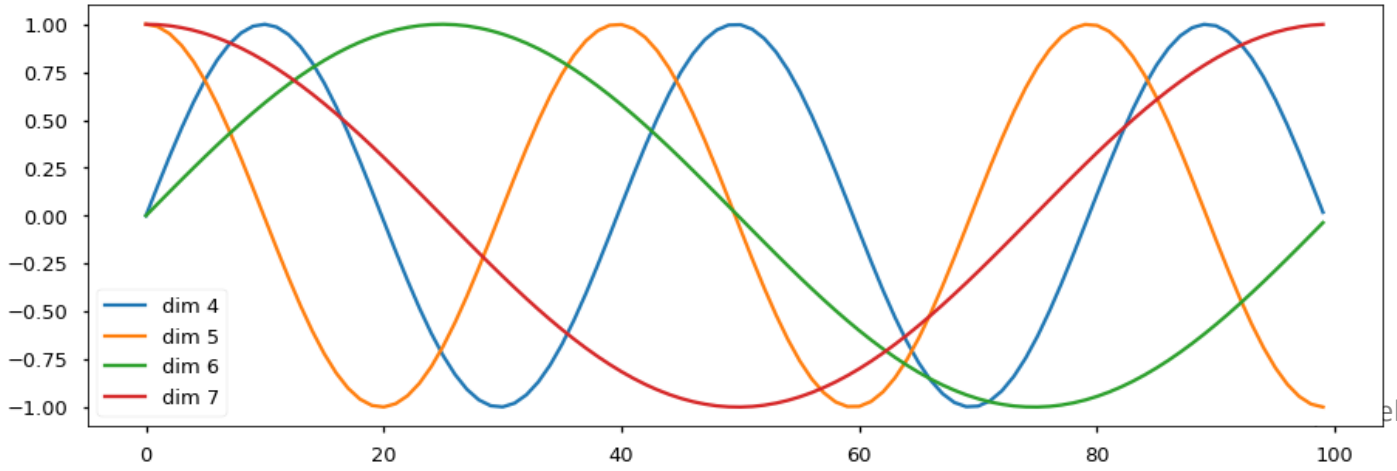
where SelfAtention(·) denotes self attention module and LayerNorm(·) denotes the layer normalization operation

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 $10000 \cdot 2\pi$. 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

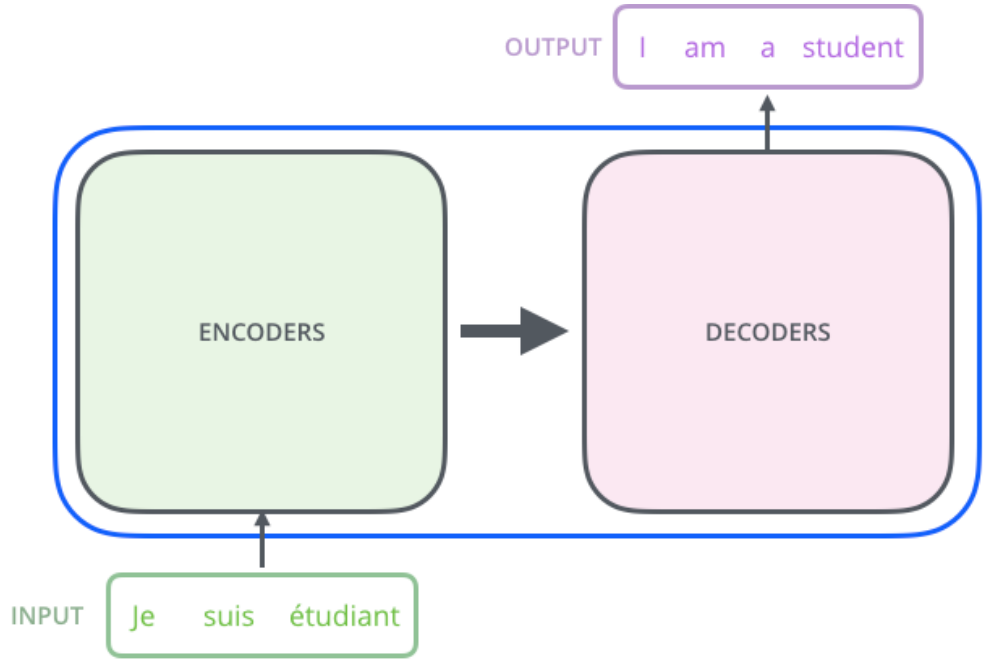
In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically.

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

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

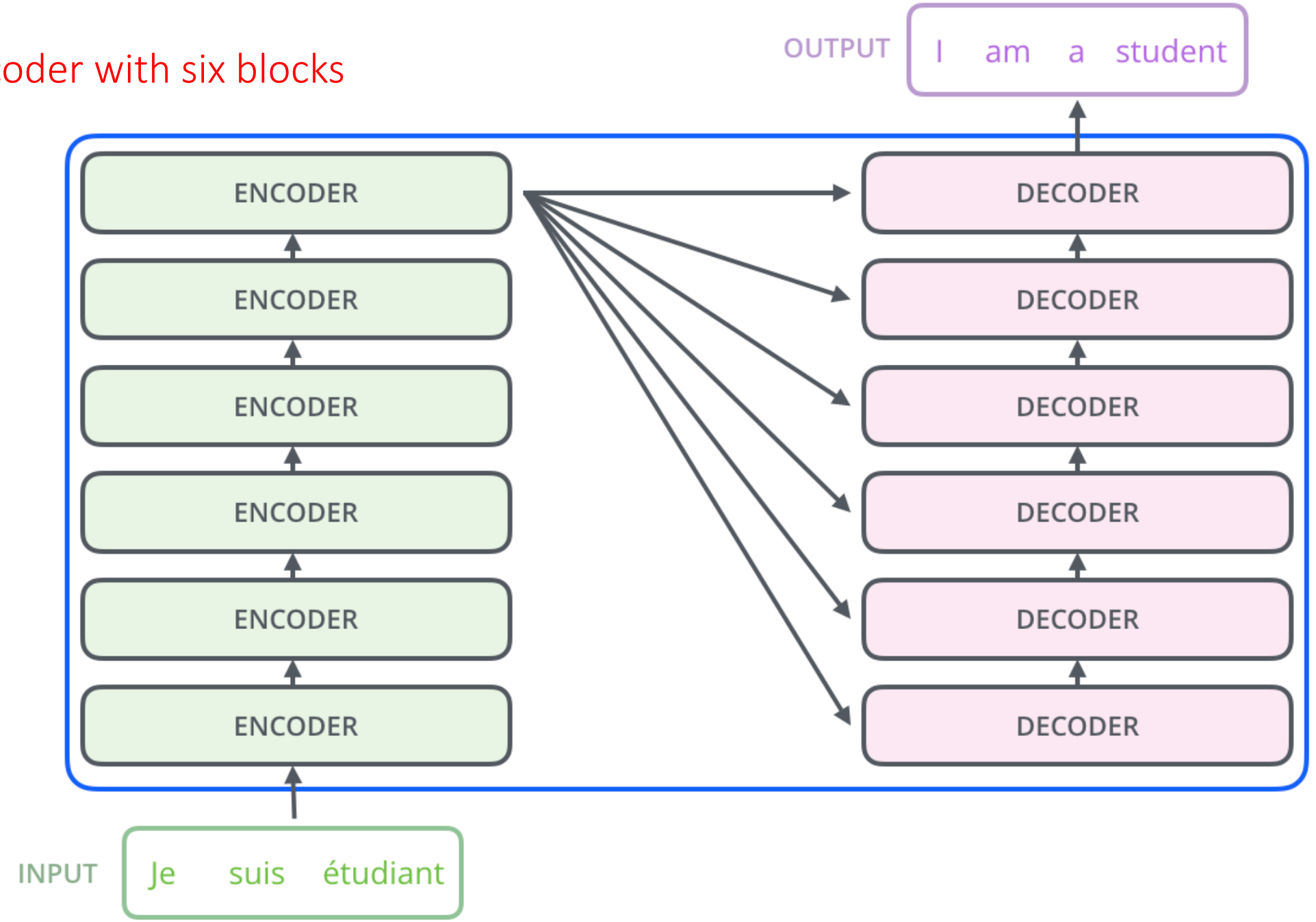
While the linear transformations are the same across different positions, they use different parameters from layer to layer.

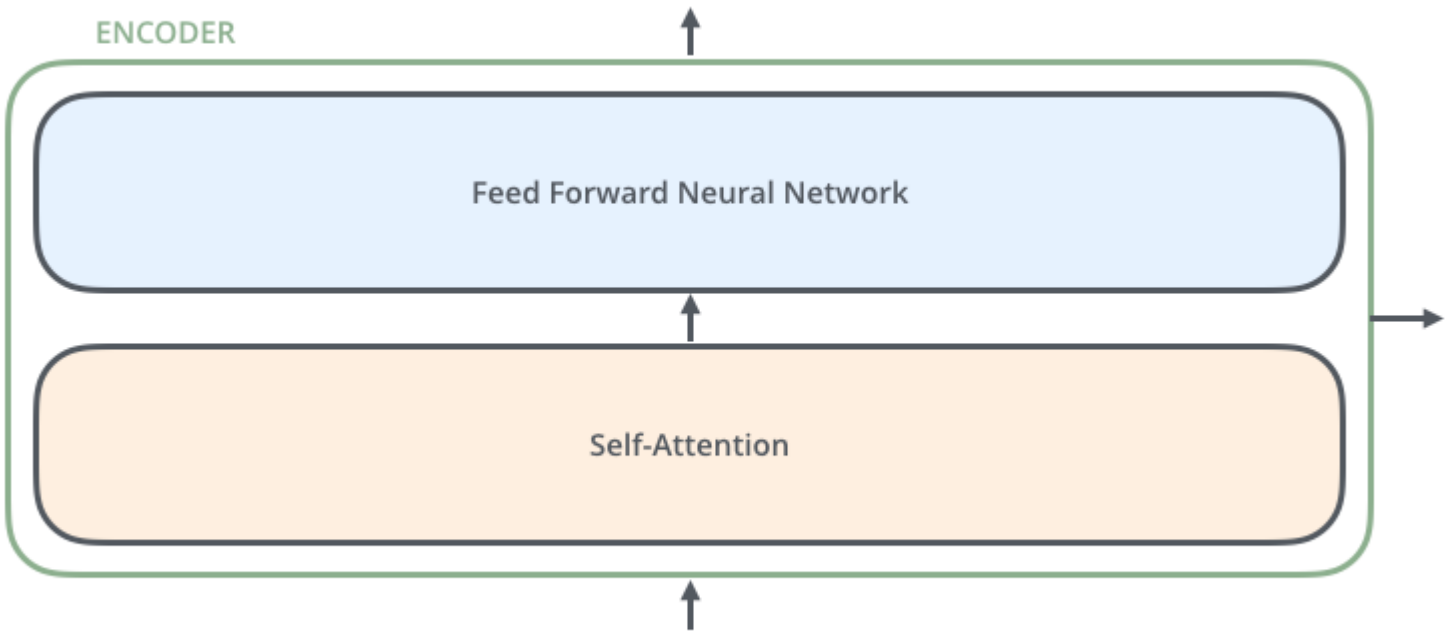
Another way of describing this is as two convolutions with kernel size 1.
The dimensionality of input and output is dmodel=512 , and the inner-layer has dimensionality dff=2048.



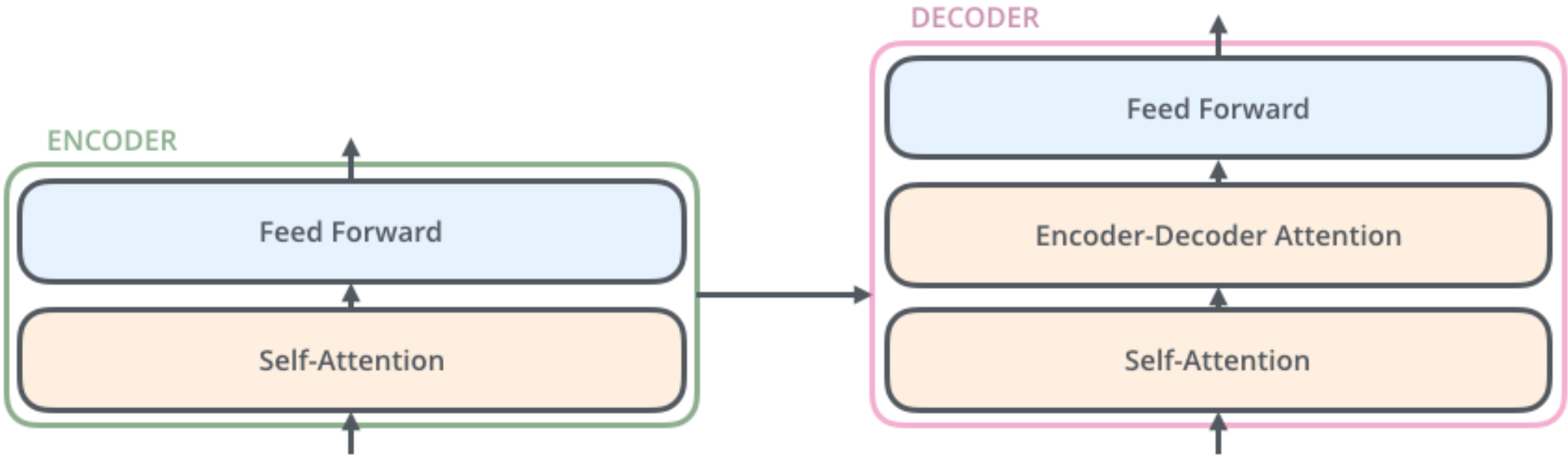
Jay Alammar
Visualizing machine learning one concept at a time.
@JayAlammar on Twitter. YouTube Channel

Encoder and Decoder with six blocks

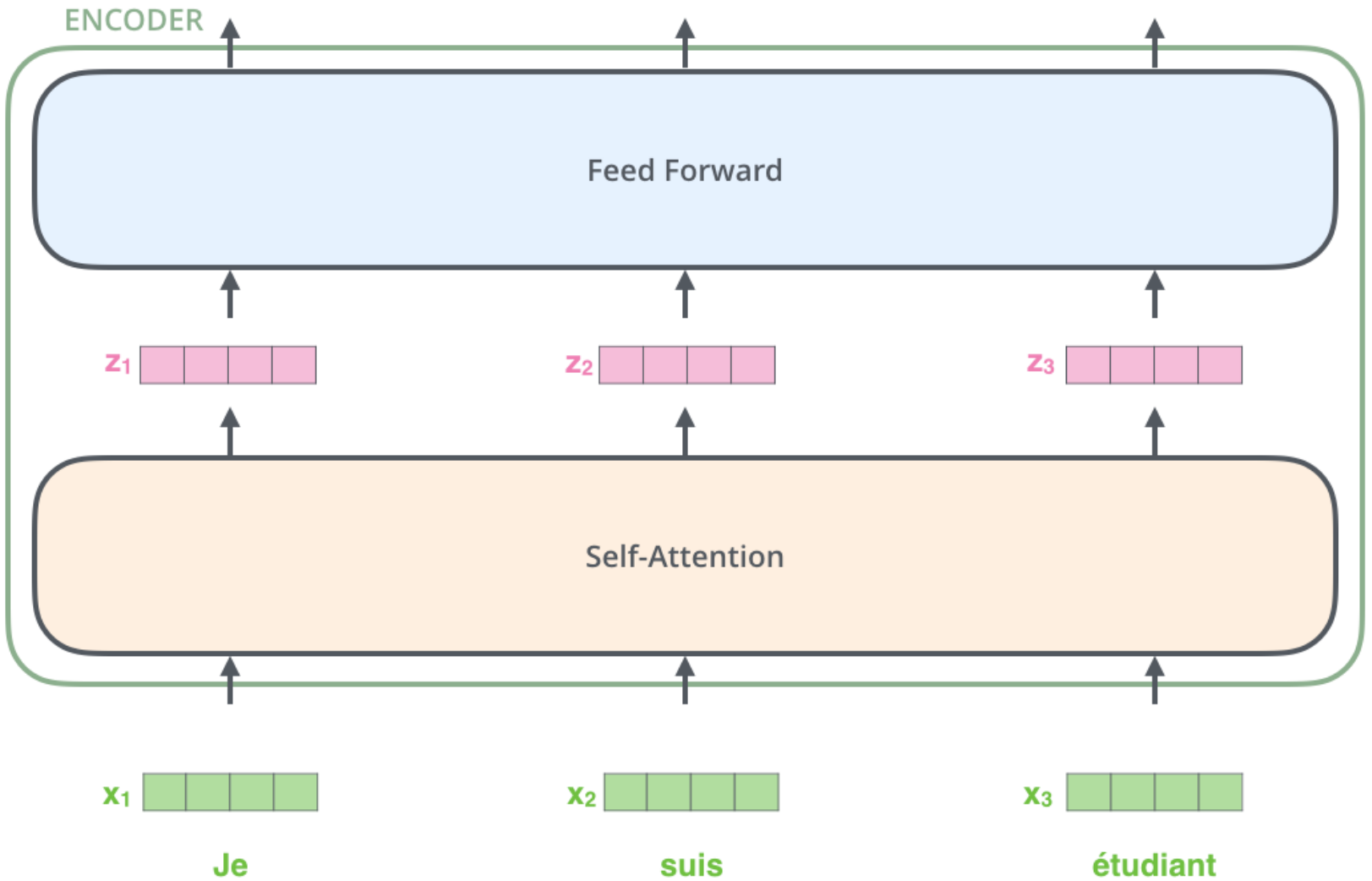



ENCODER
Feed Forward Neural Network
Self-Attention

Encoder and Decoder with one block

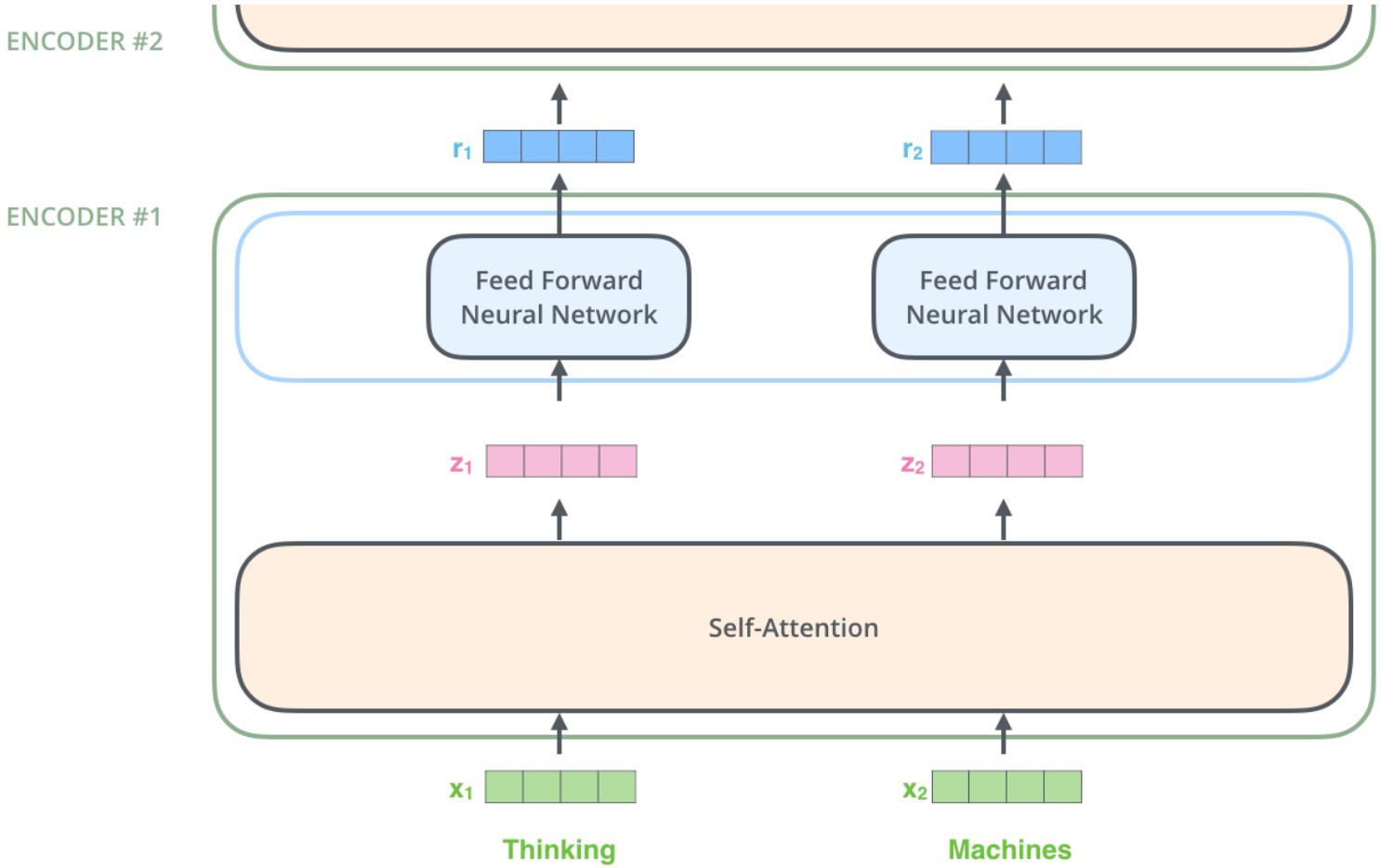


Bringing The Tensors Into The Picture



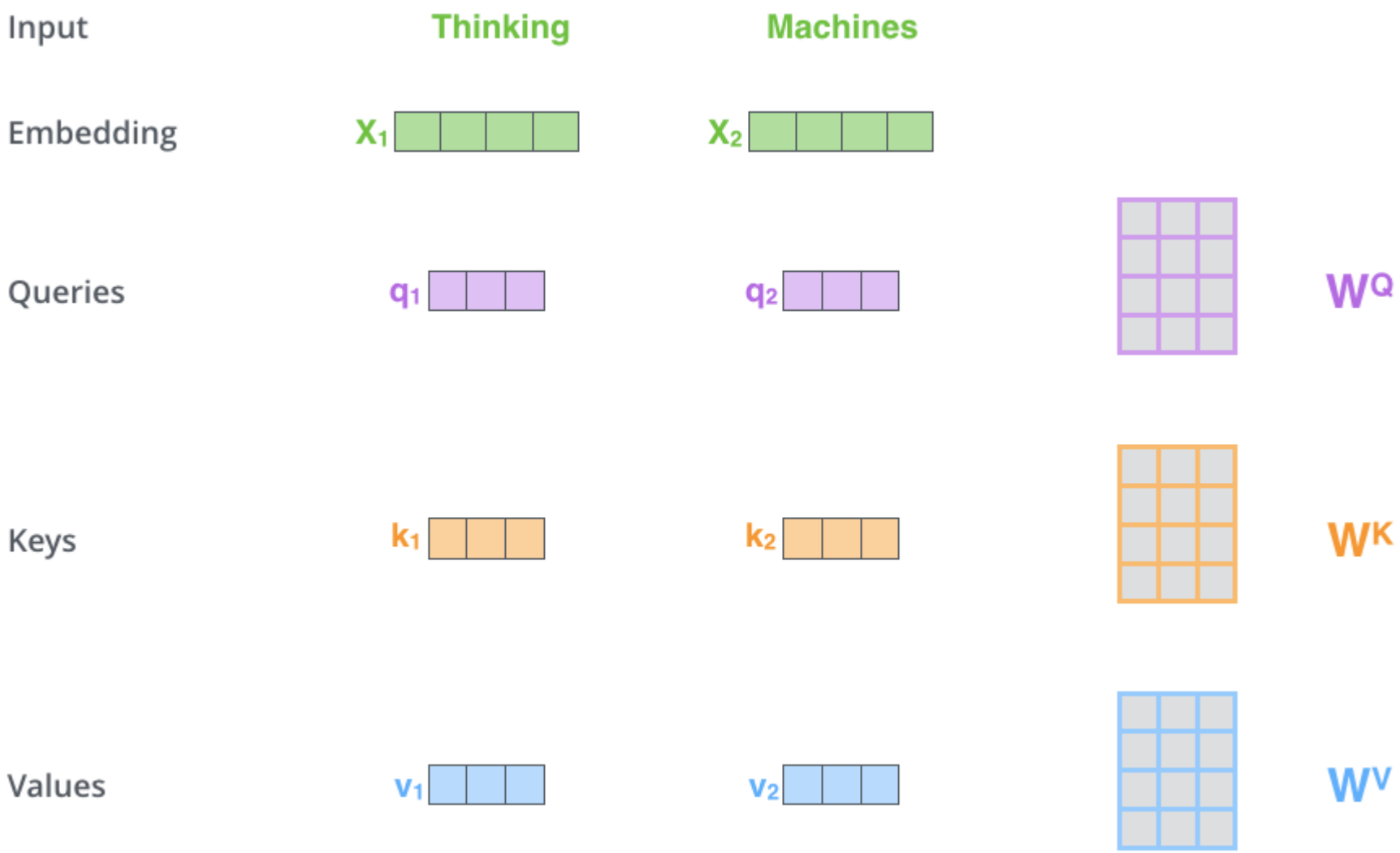
Now We're Encoding!

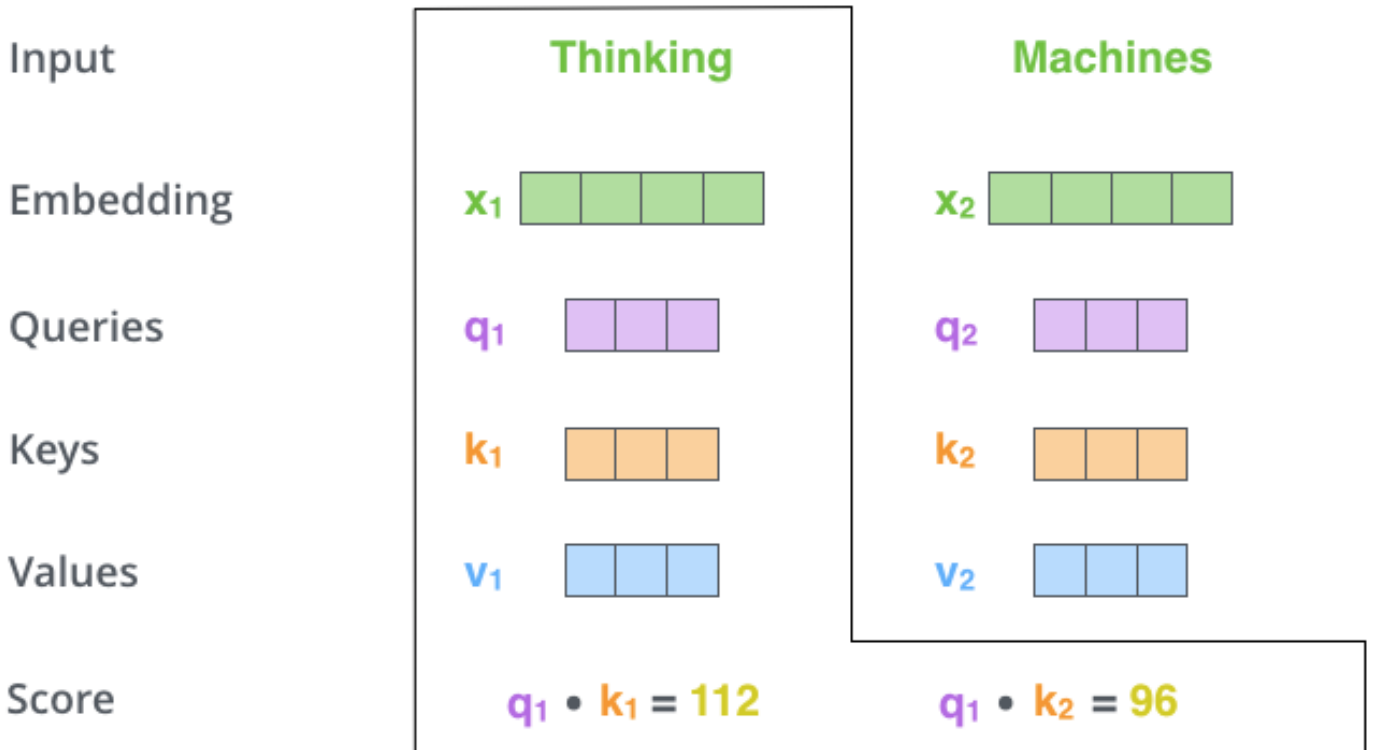
The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

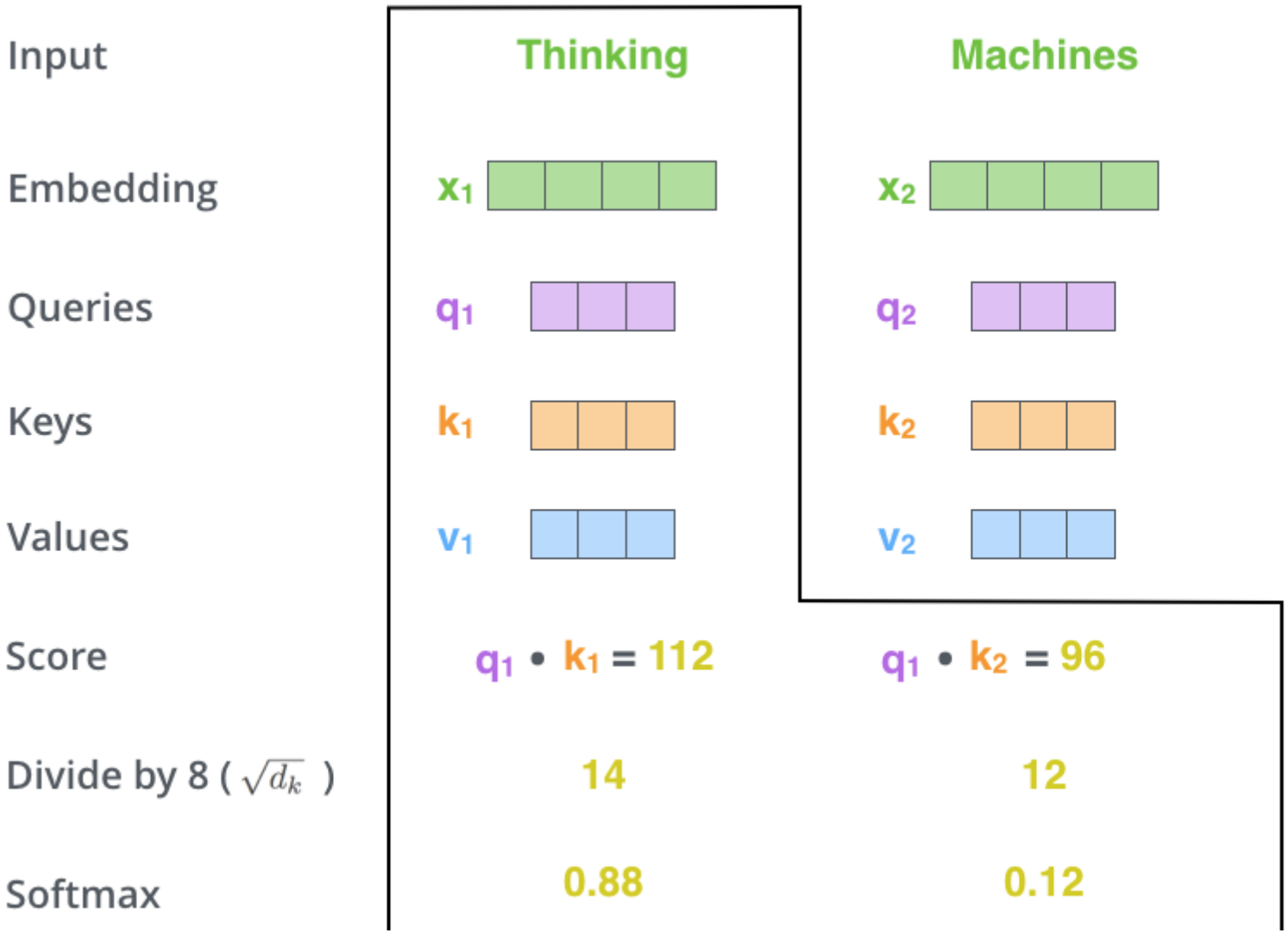


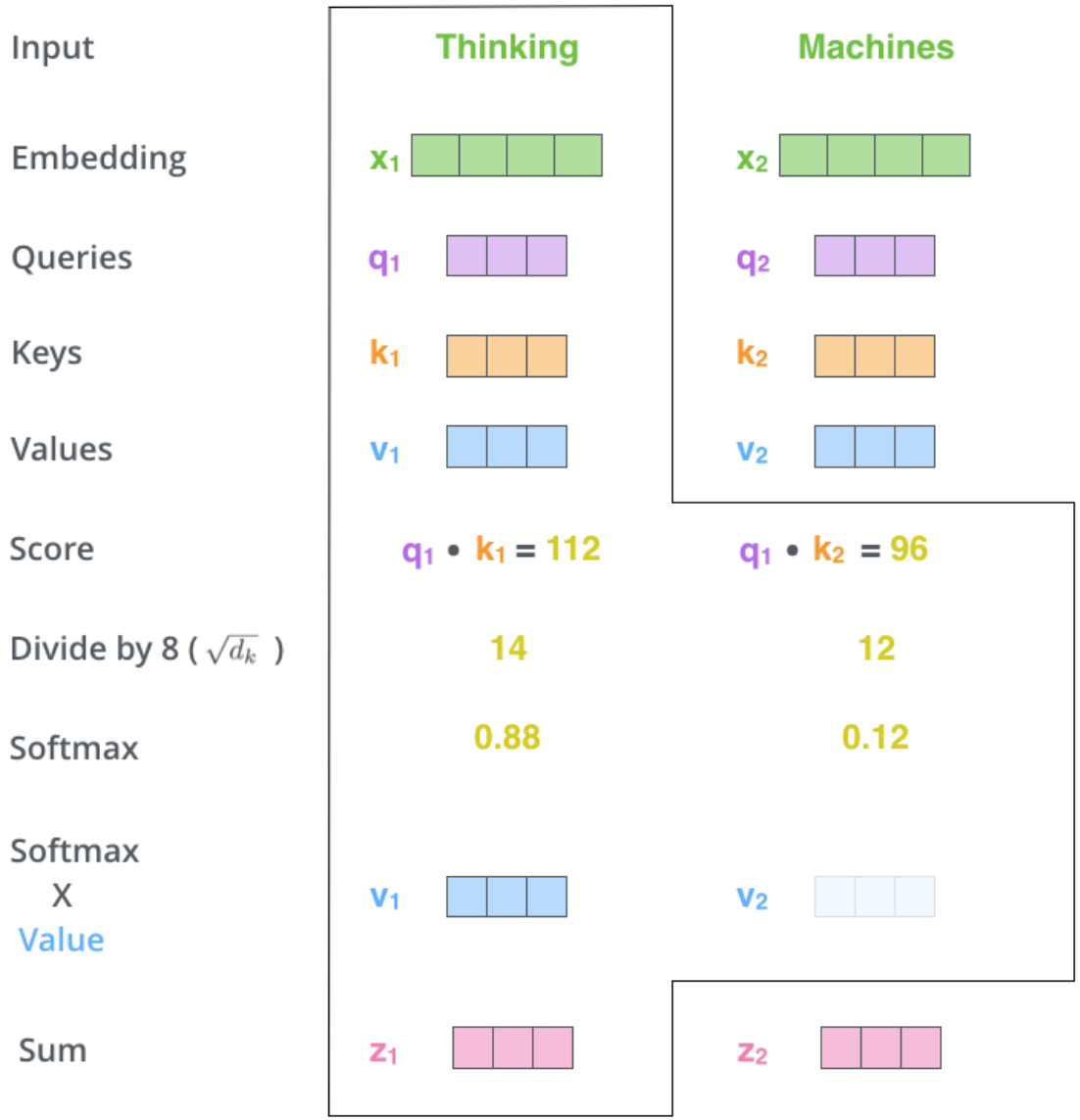
Self-Attention in Detail

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
x_1
x_2
Queries
q_1
q_2
Keys
k_1
k_2
Values
v_1
v_2
Score
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$

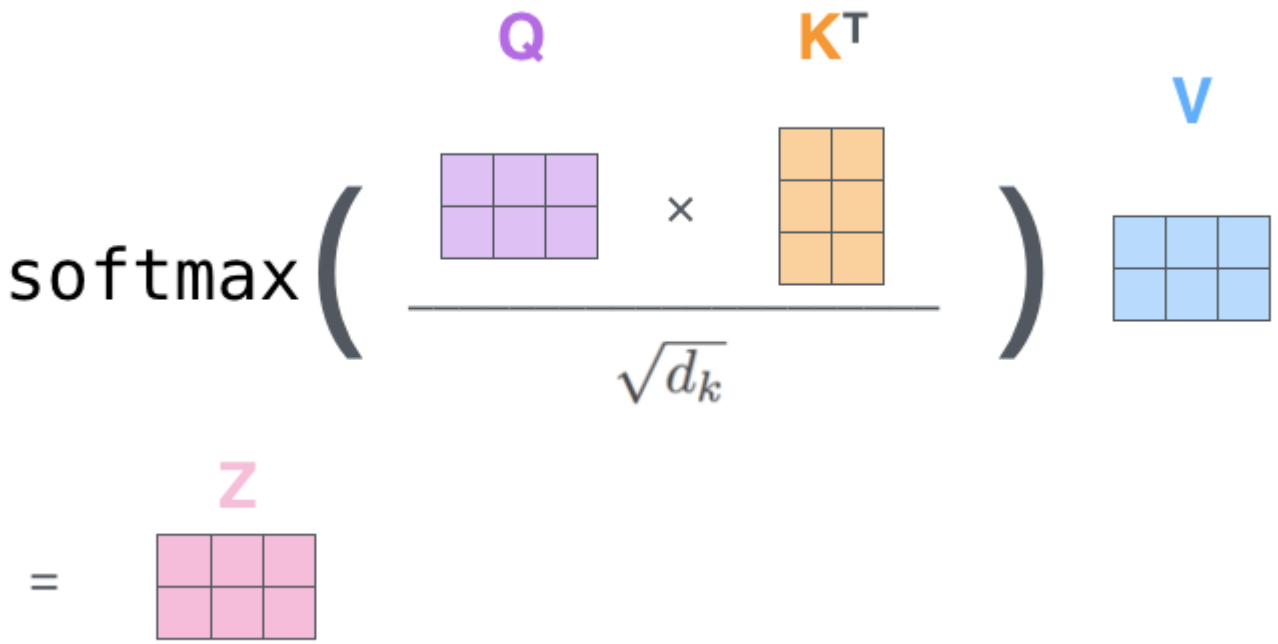

Input
Thinking
Machines
Embedding
x_1
x_2
Queries
q_1
q_2
Keys
k_1
k_2
Values
v_1
v_2
Score
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12


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

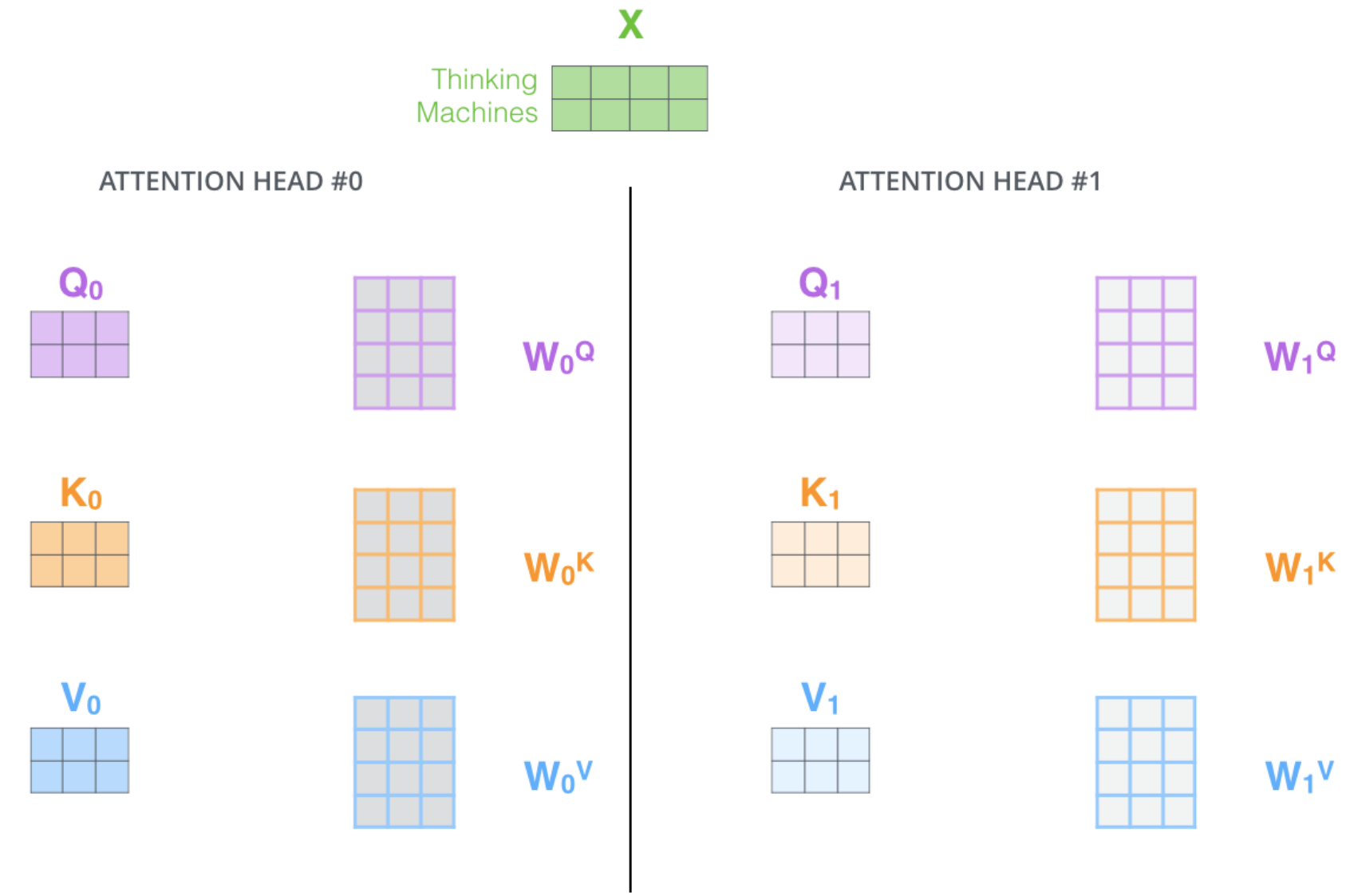
Matrix Calculation of Self-Attention

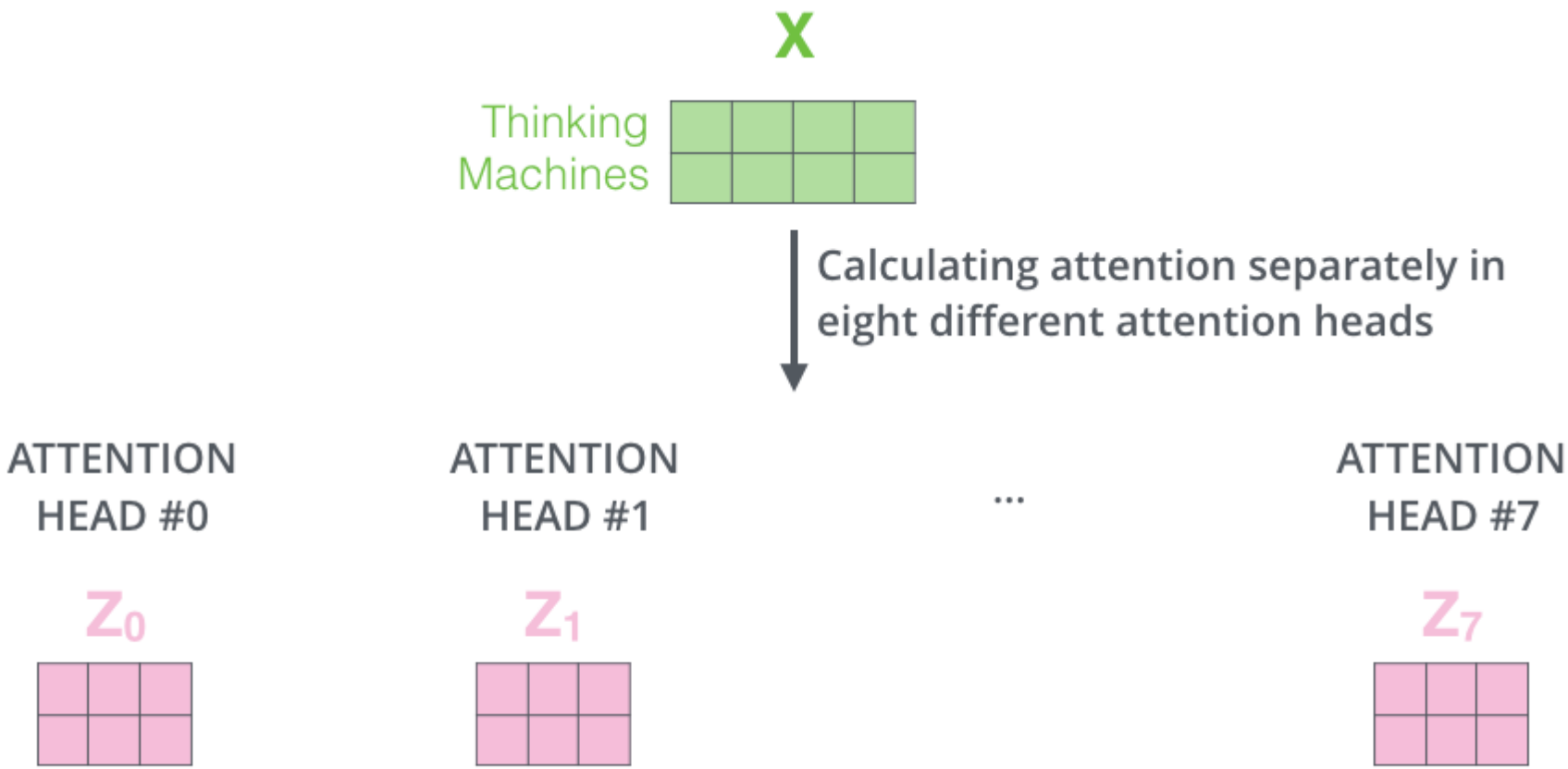
Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

The self-attention calculation in matrix form

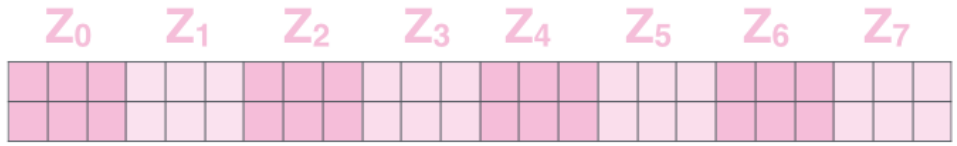


The Beast With Many Heads



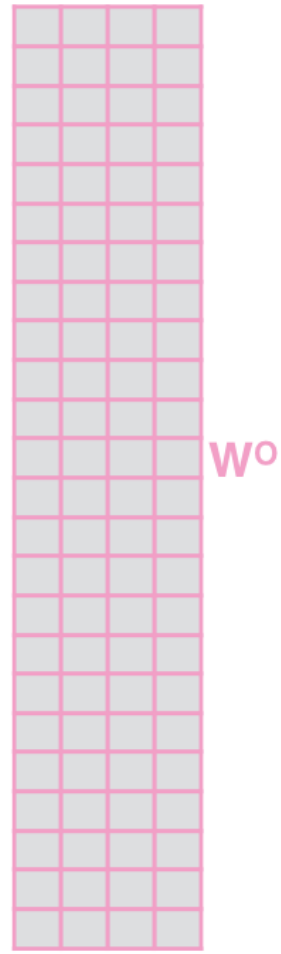

X
Thinking
Machines
Calculating attention separately in
eight different attention heads
ATTENTION
HEAD #0
ATTENTION
HEAD #1
...
ATTENTION
HEAD #7
Z0
Z1
Z7

1) Concatenate all the attention heads



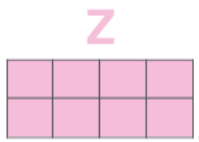
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=

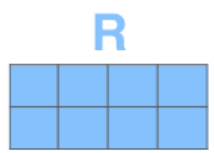


1) This is our input sentence*

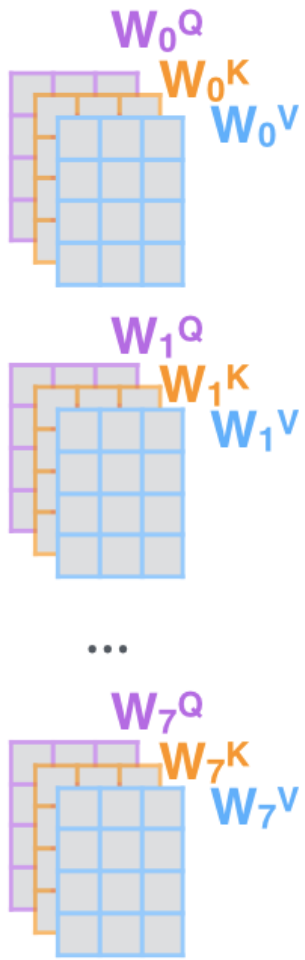
Thinking
Machines

2) We embed each word*

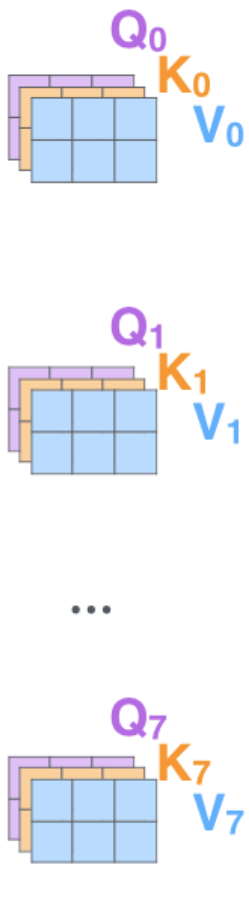
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



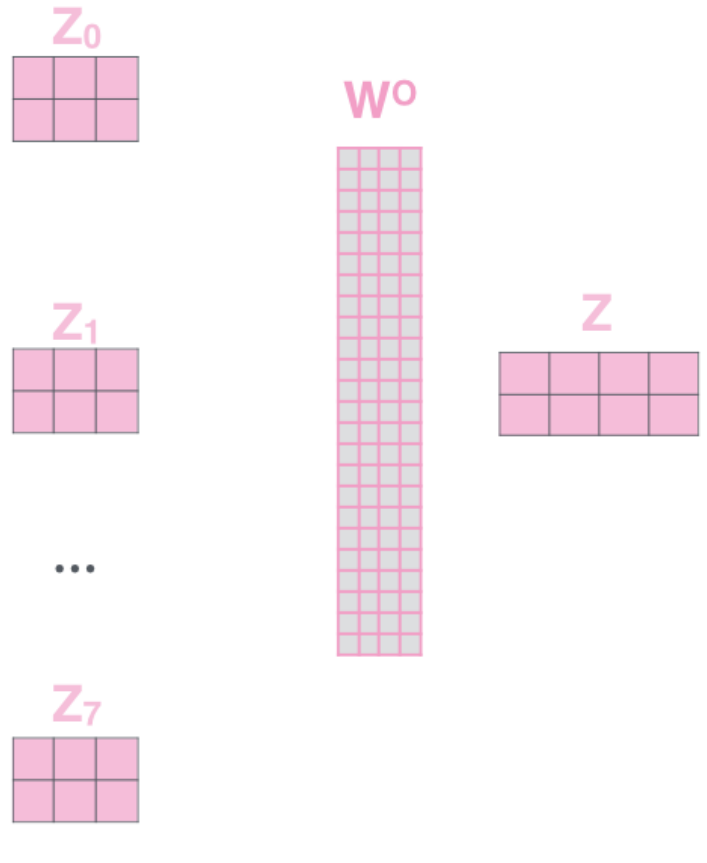
3) Split into 8 heads. We multiply X or R with weight matrices



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

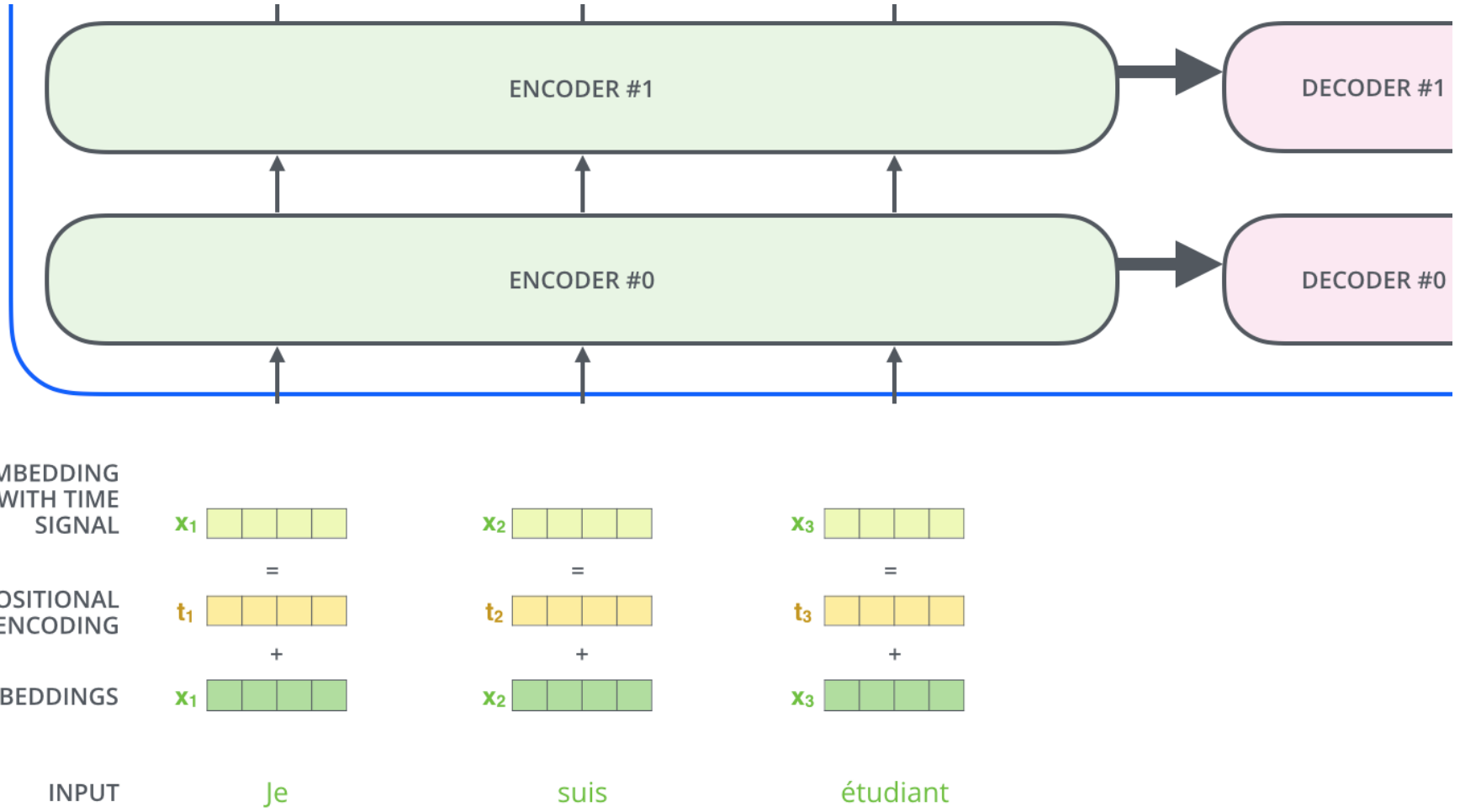


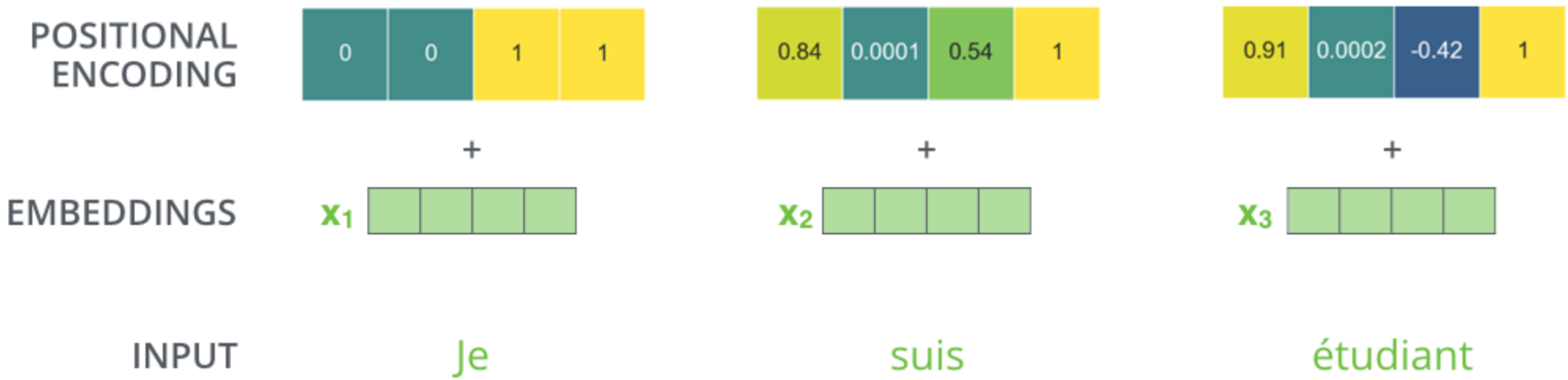
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



Representing The Order of The Sequence Using Positional Encoding

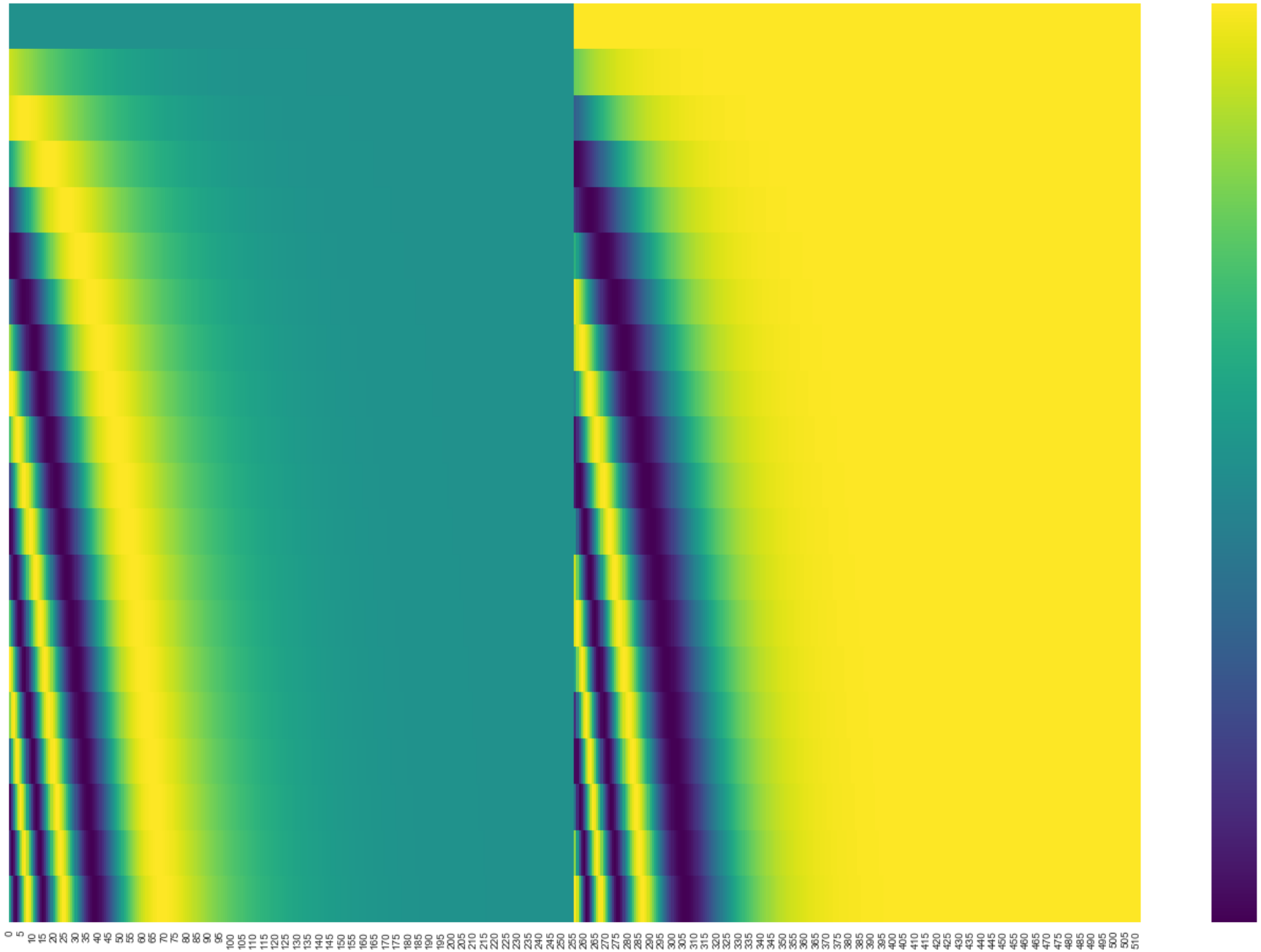
To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.



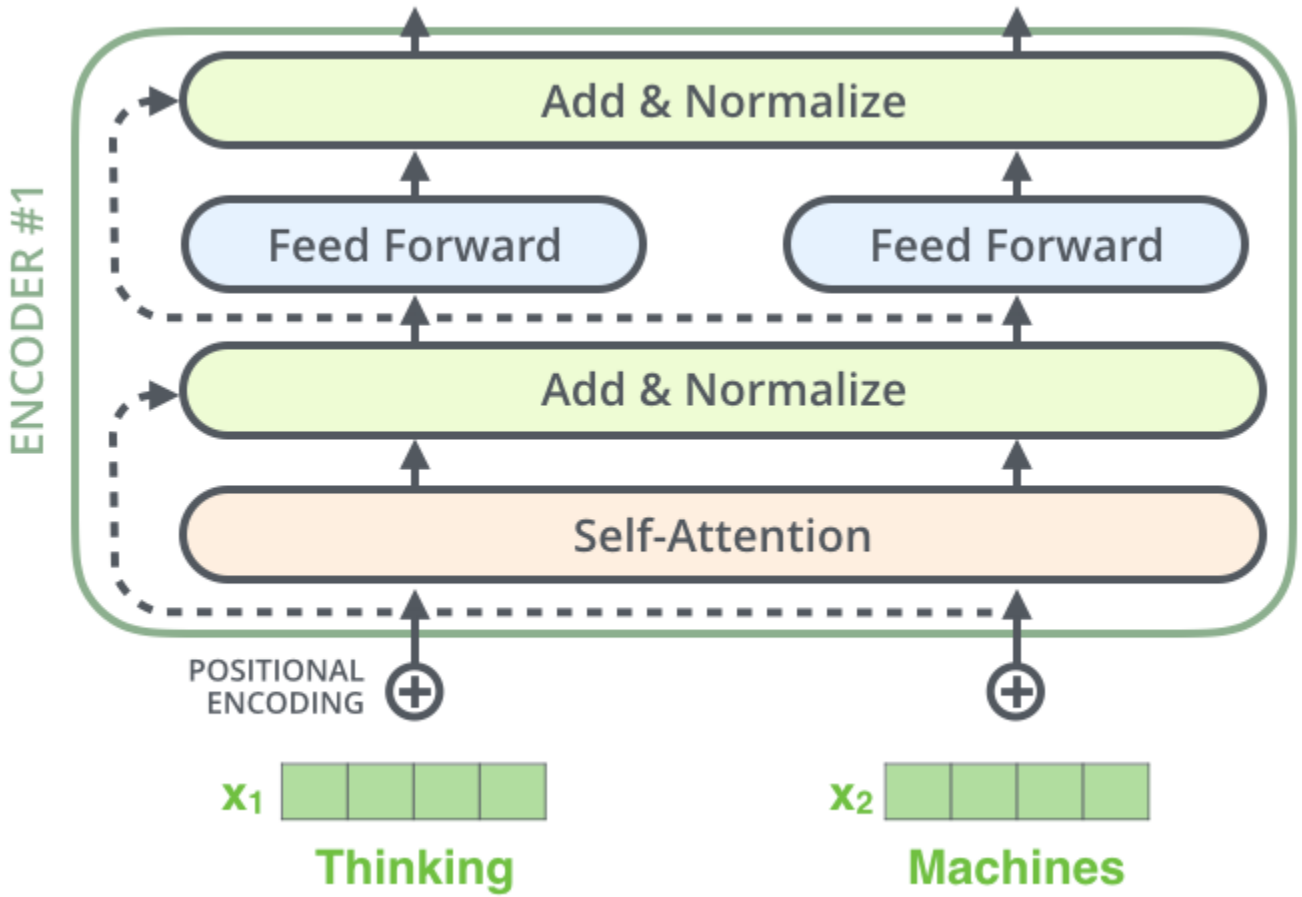


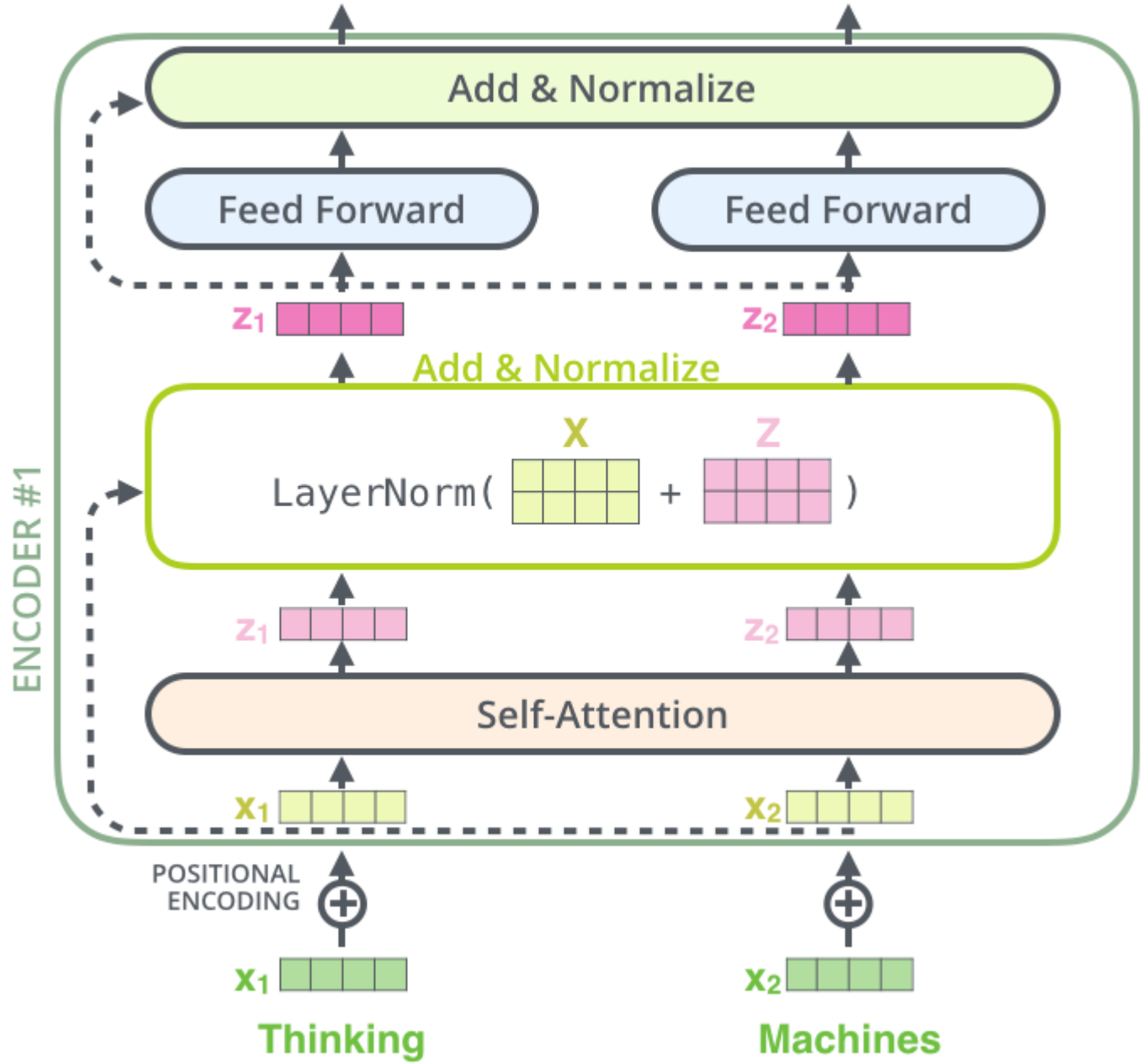
A real example of positional encoding with a toy embedding size of 4

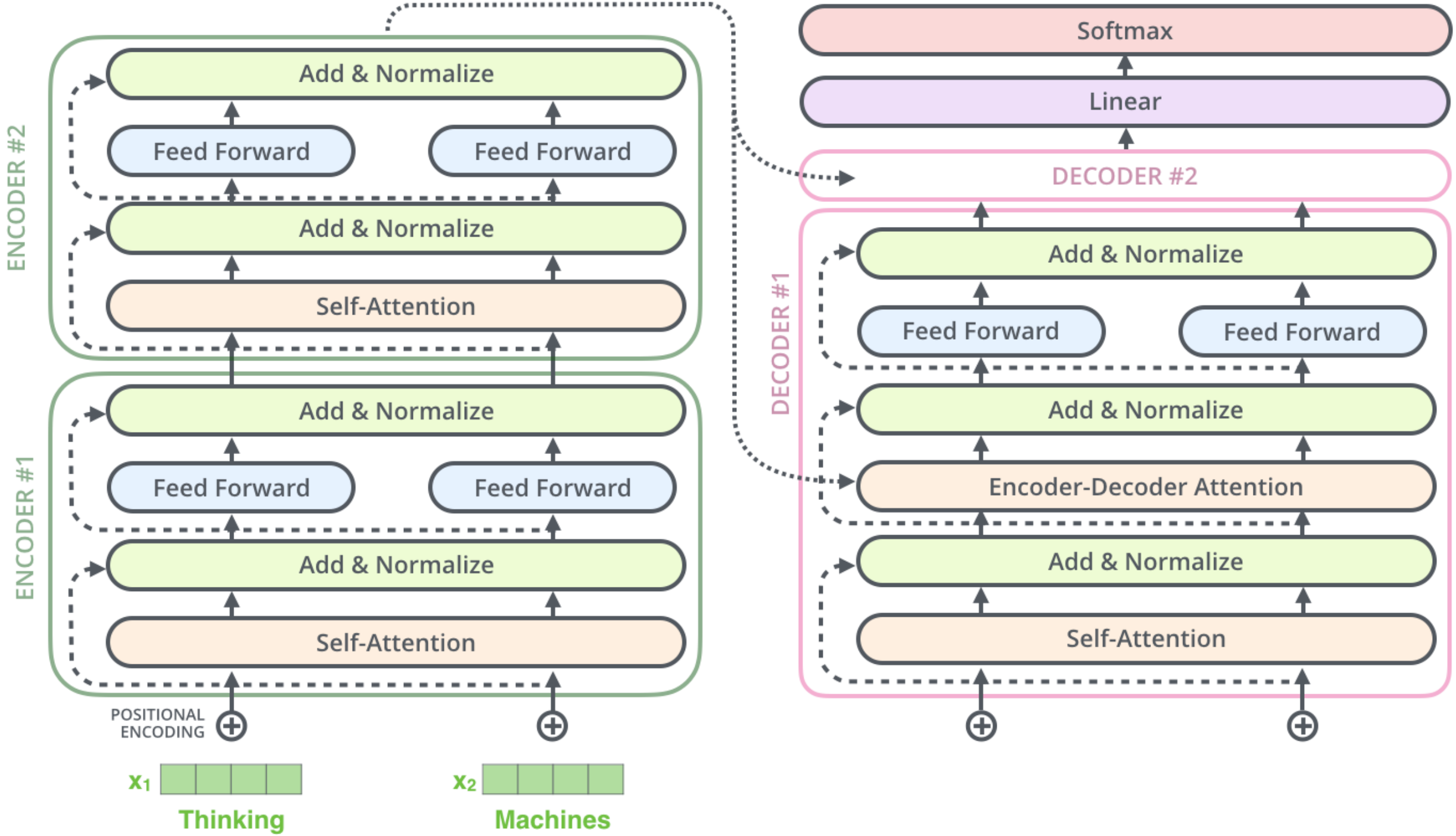
A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function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



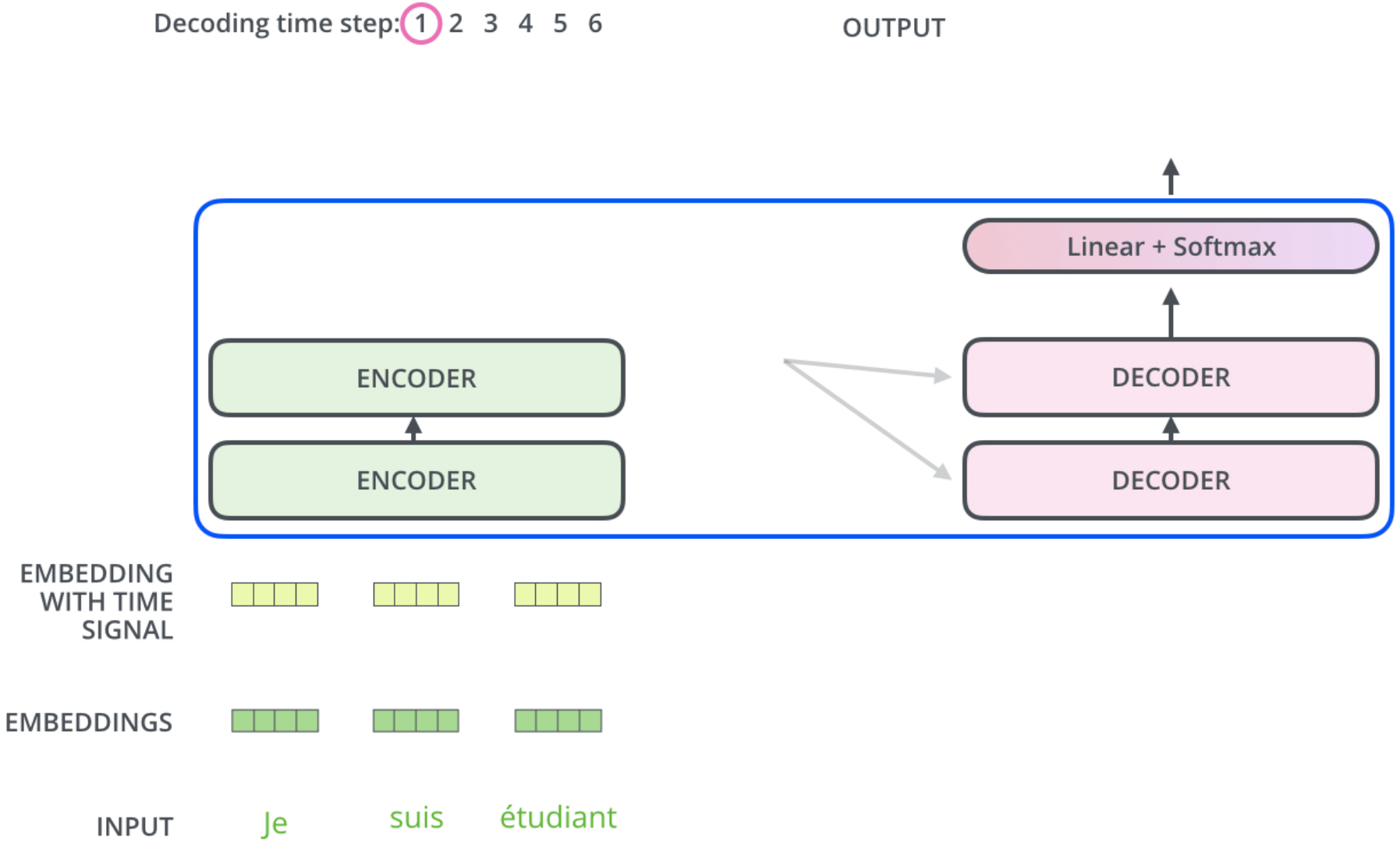
The Residuals

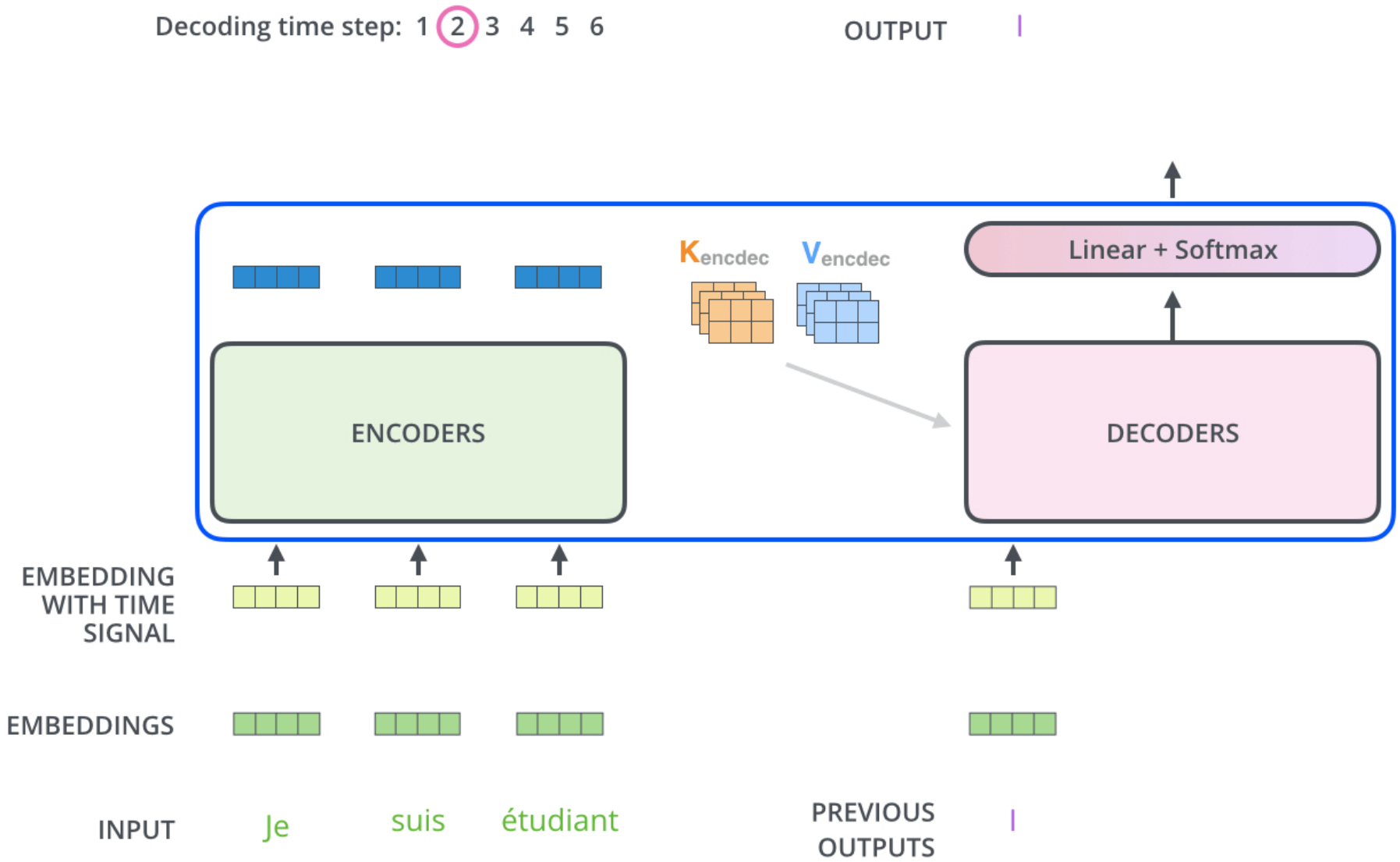



Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL ENCODING
x1
x2
Thinking
Machines


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

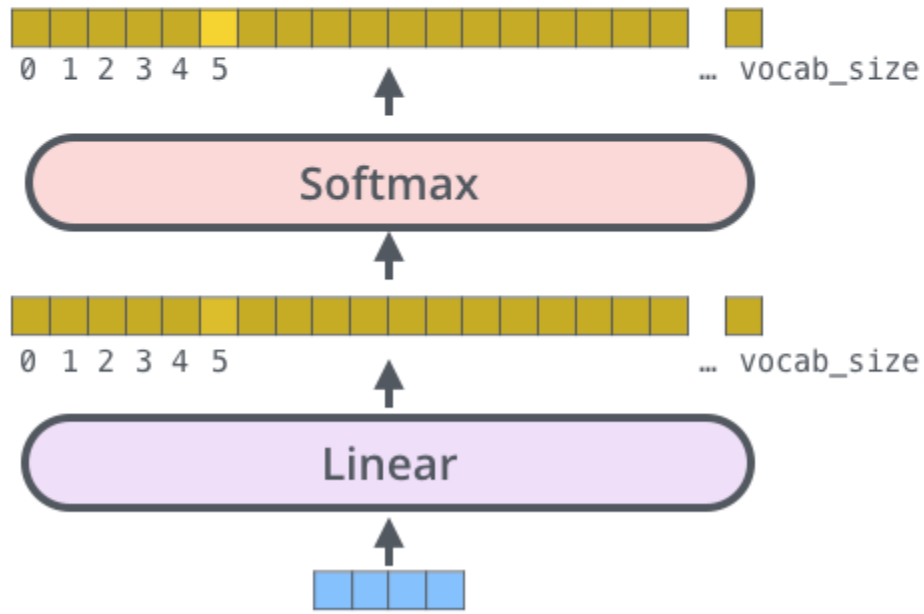
The Final Linear and Softmax Layer

Which word in our vocabulary is associated with this index?

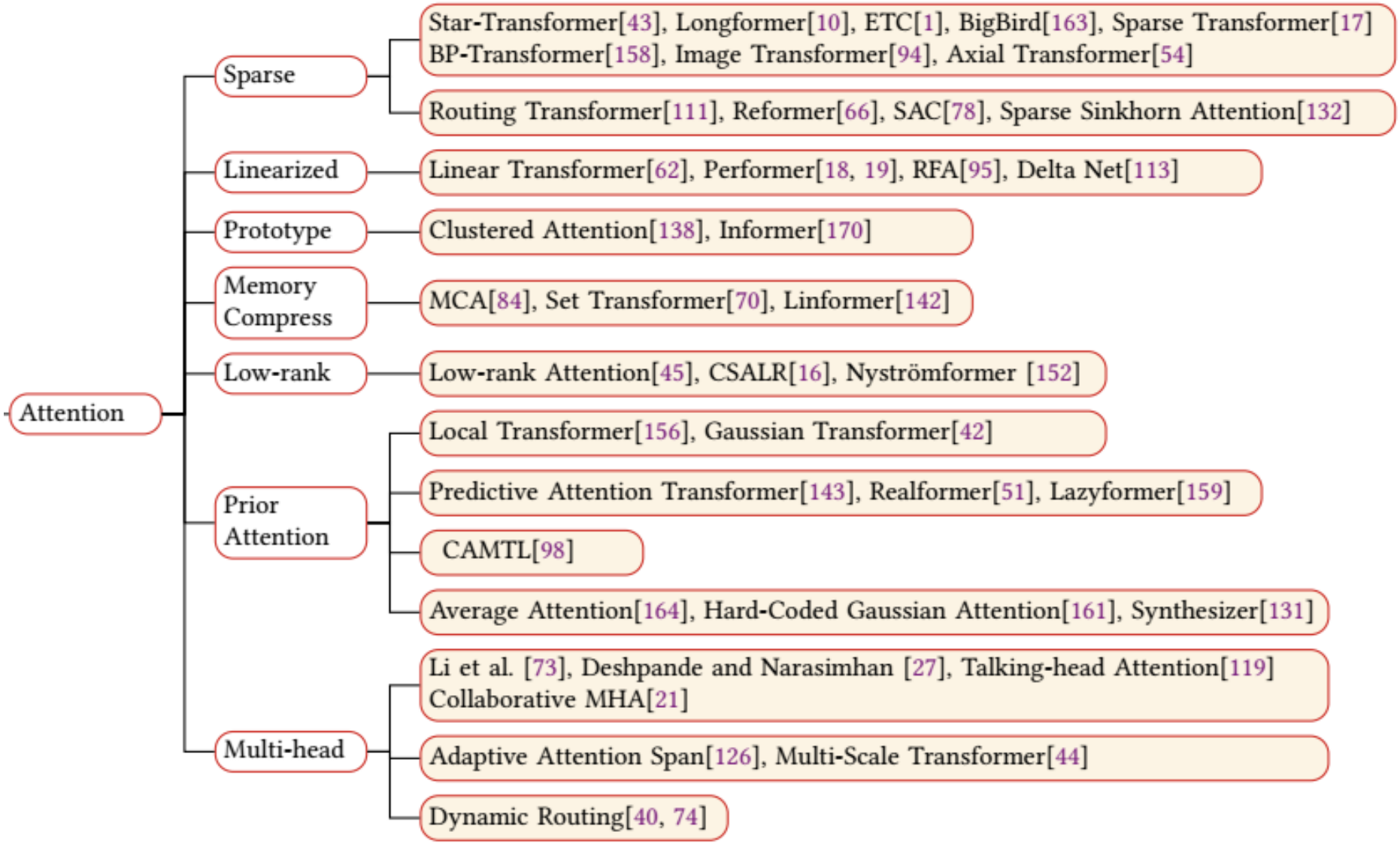
am

Get the index of the cell with the highest value **(argmax)**

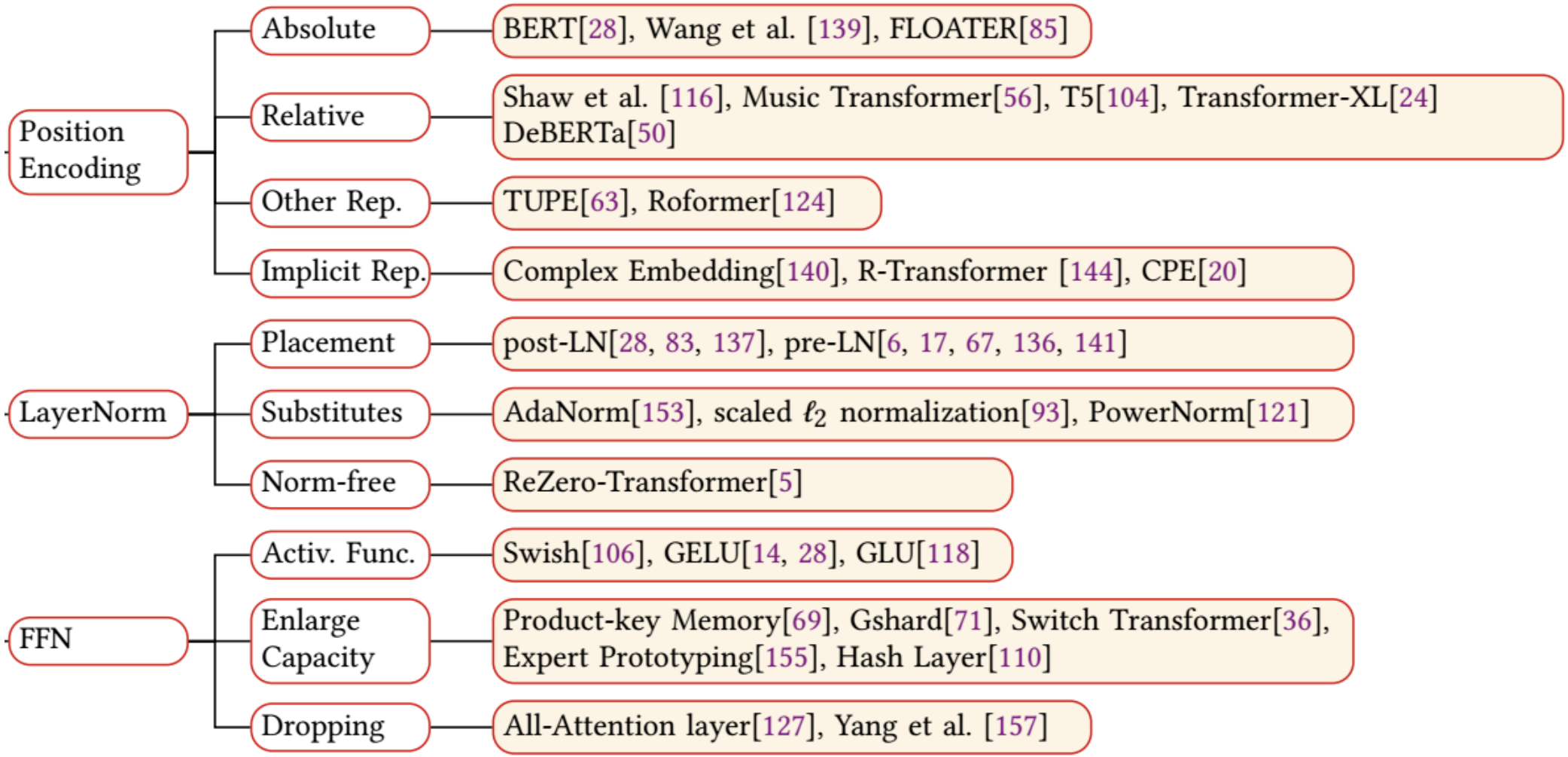
5



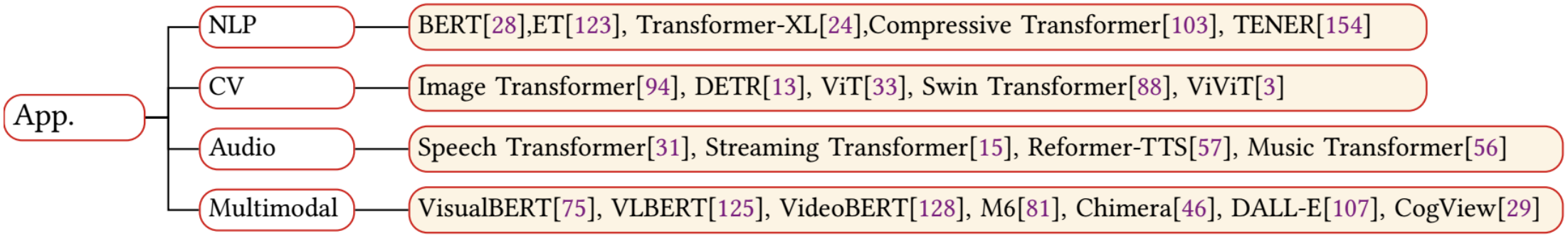
Taxonomy of Transformers in Module Level (Attention)



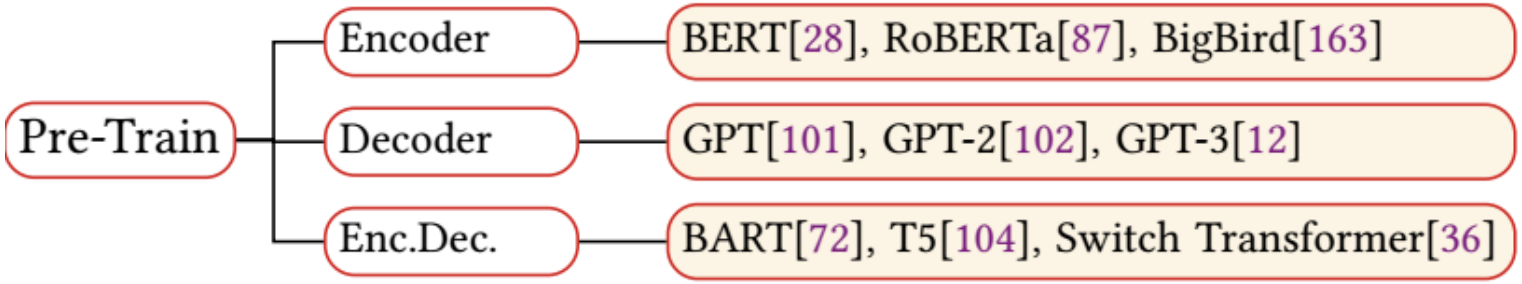
Taxonomy of Transformers in Module Level (Position-Layer Norm-FFN)



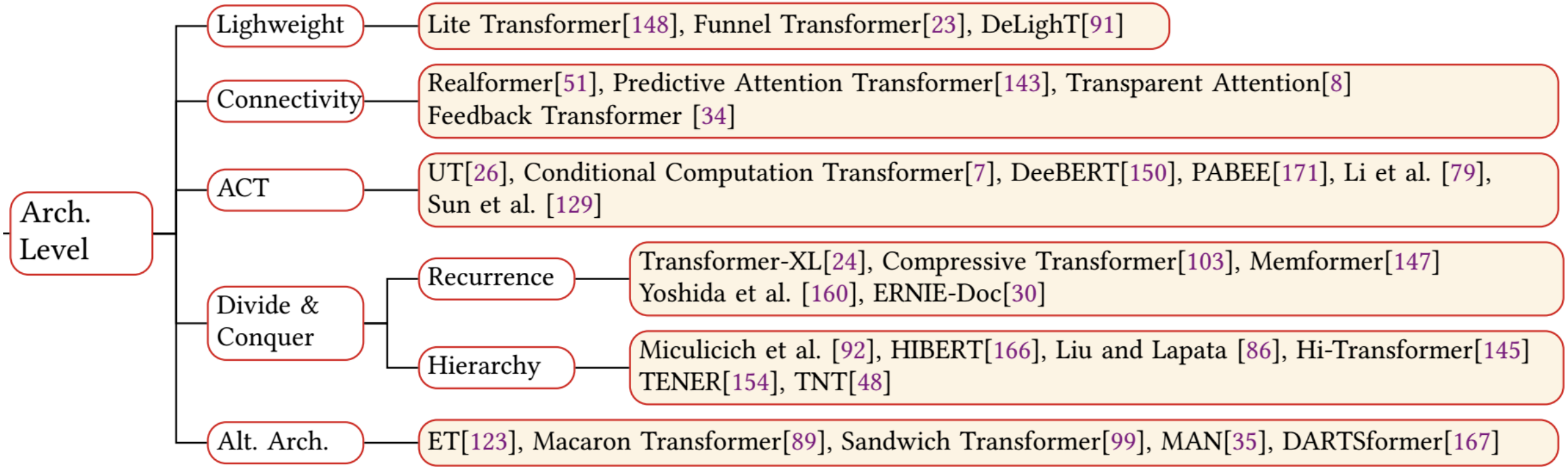
Taxonomy of Transformers in App



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).

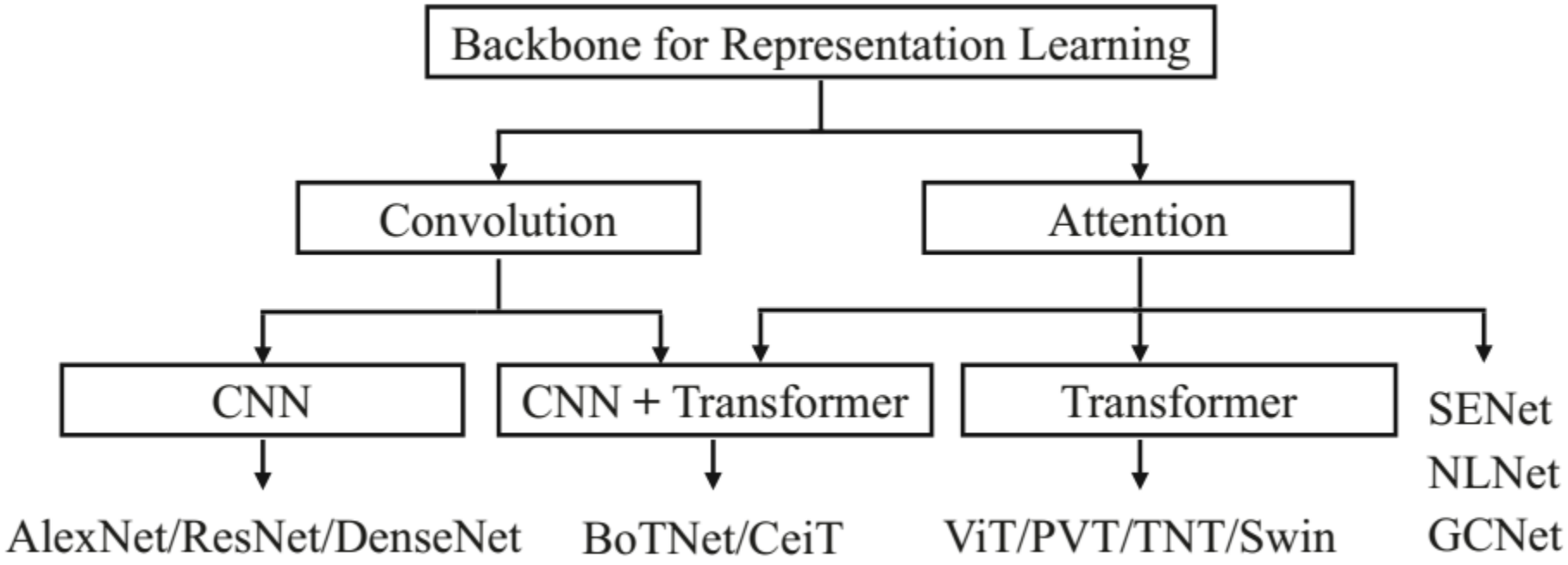
- *Encoder only*. Only the encoder is used and the outputs of the encoder are utilized as are presentation for the input sequence. This is usually used for classification or sequence labeling problems.

- *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.

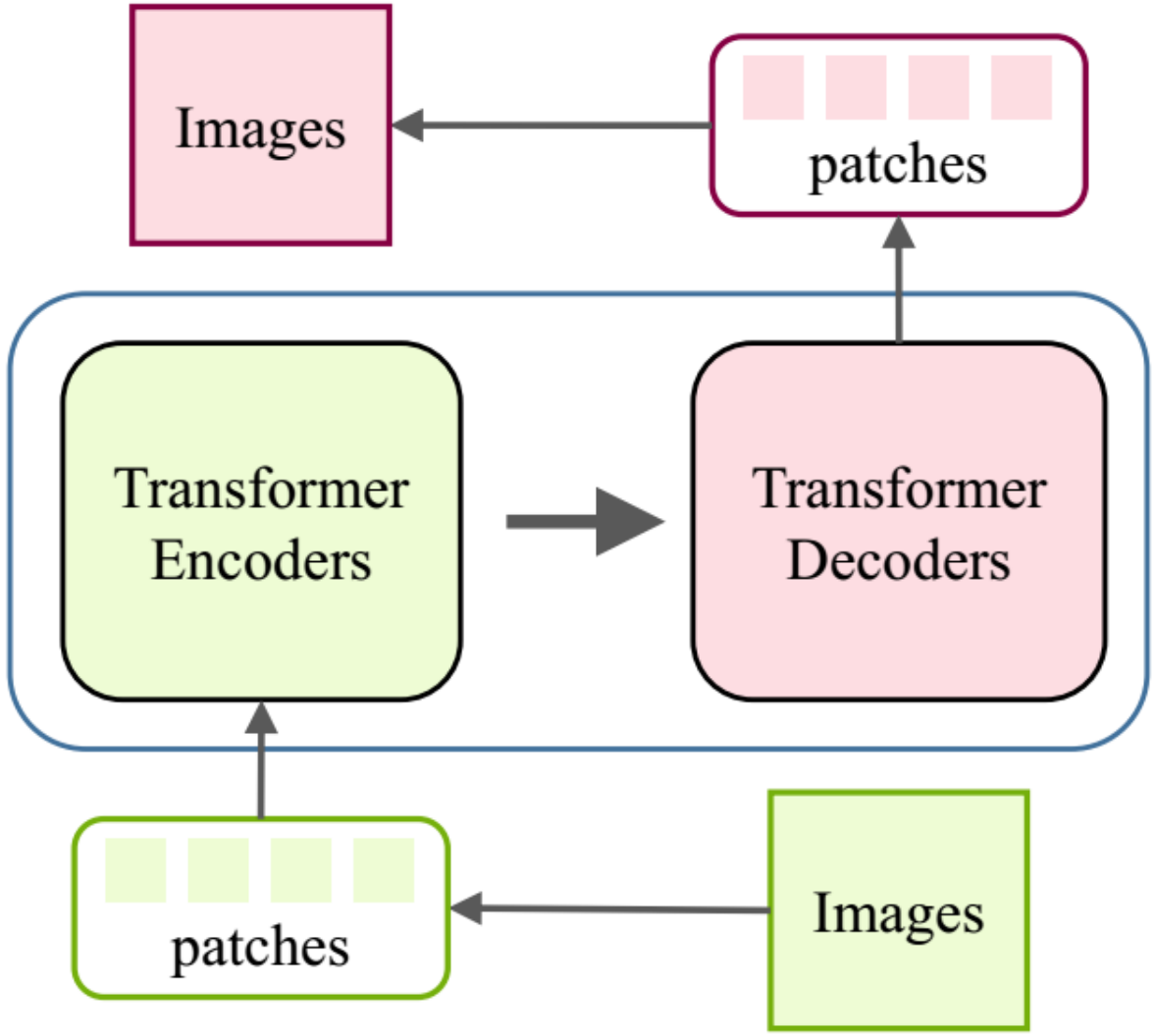
Vision Transformer (ViT)

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:



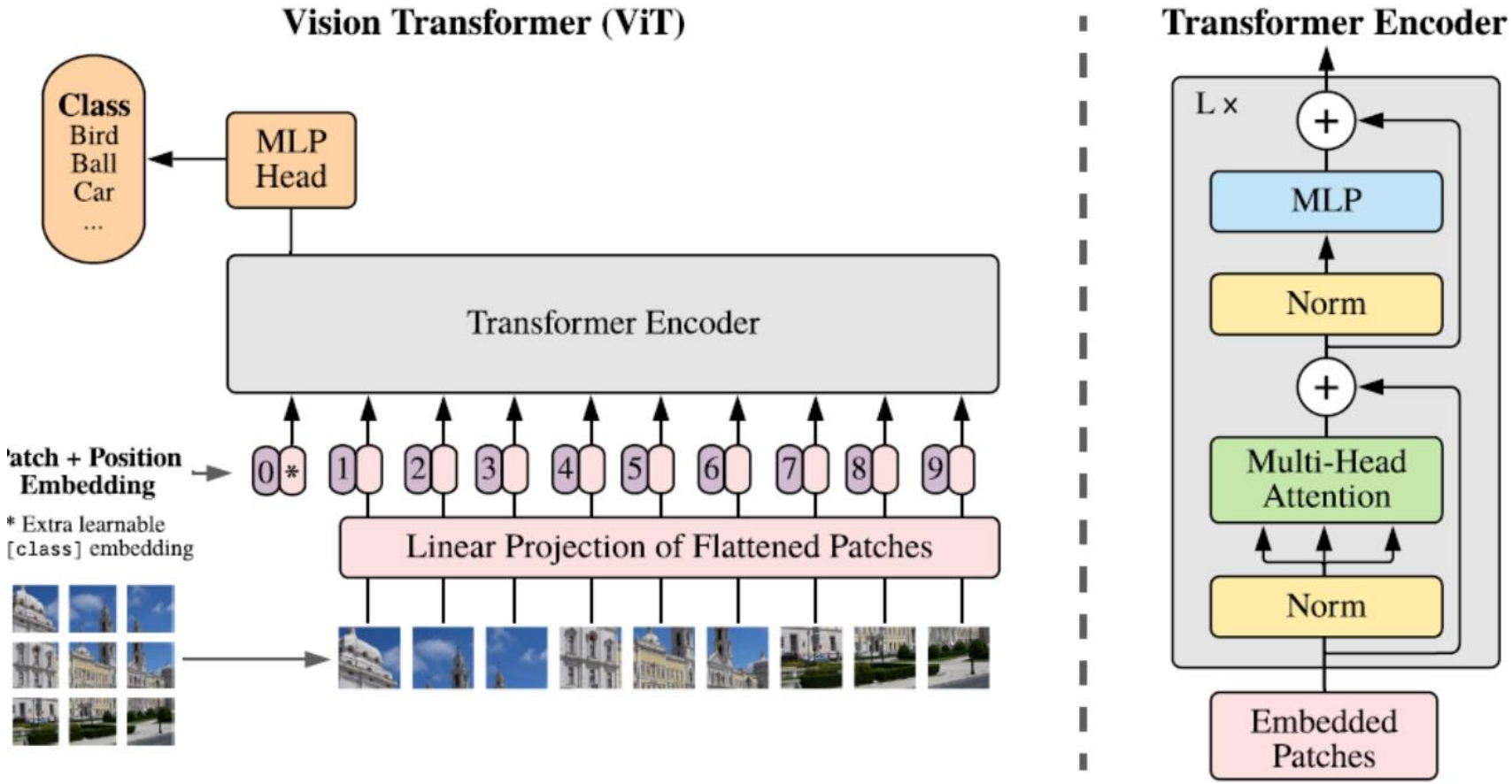
A generic framework for using transformer in image processing



Vision Transformer (ViT)

Dosovitskiy...2021

Vision Transformer (ViT) 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

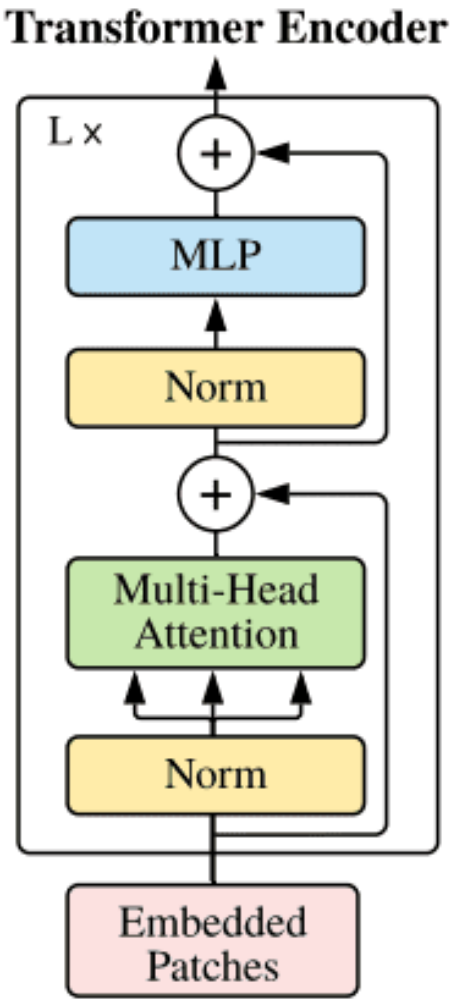
The total architecture is called Vision Transformer (ViT in short). Let's examine it step by step.

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

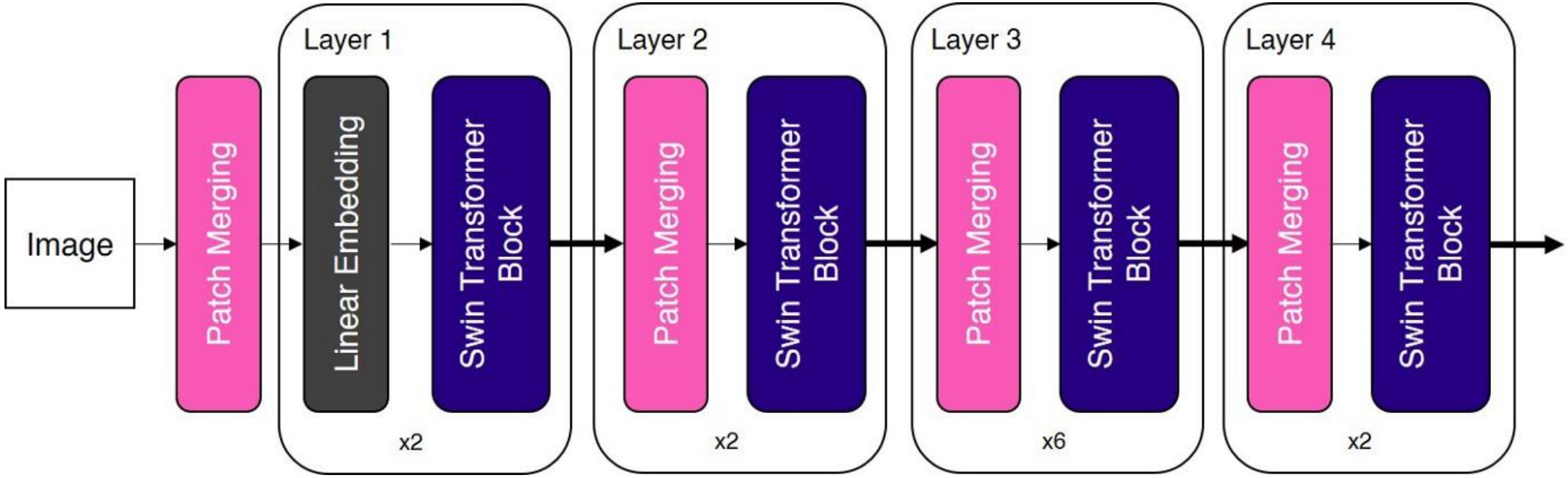
Heads refer to multi-head attention, while the MLP size refers to the blue module in the figure.
MLP stands for multi-layer perceptron but it's actually a bunch of linear transformation layers.
Hidden size D is the embedding size, which is kept fixed throughout the layers. Why keep it fixed? So that 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.

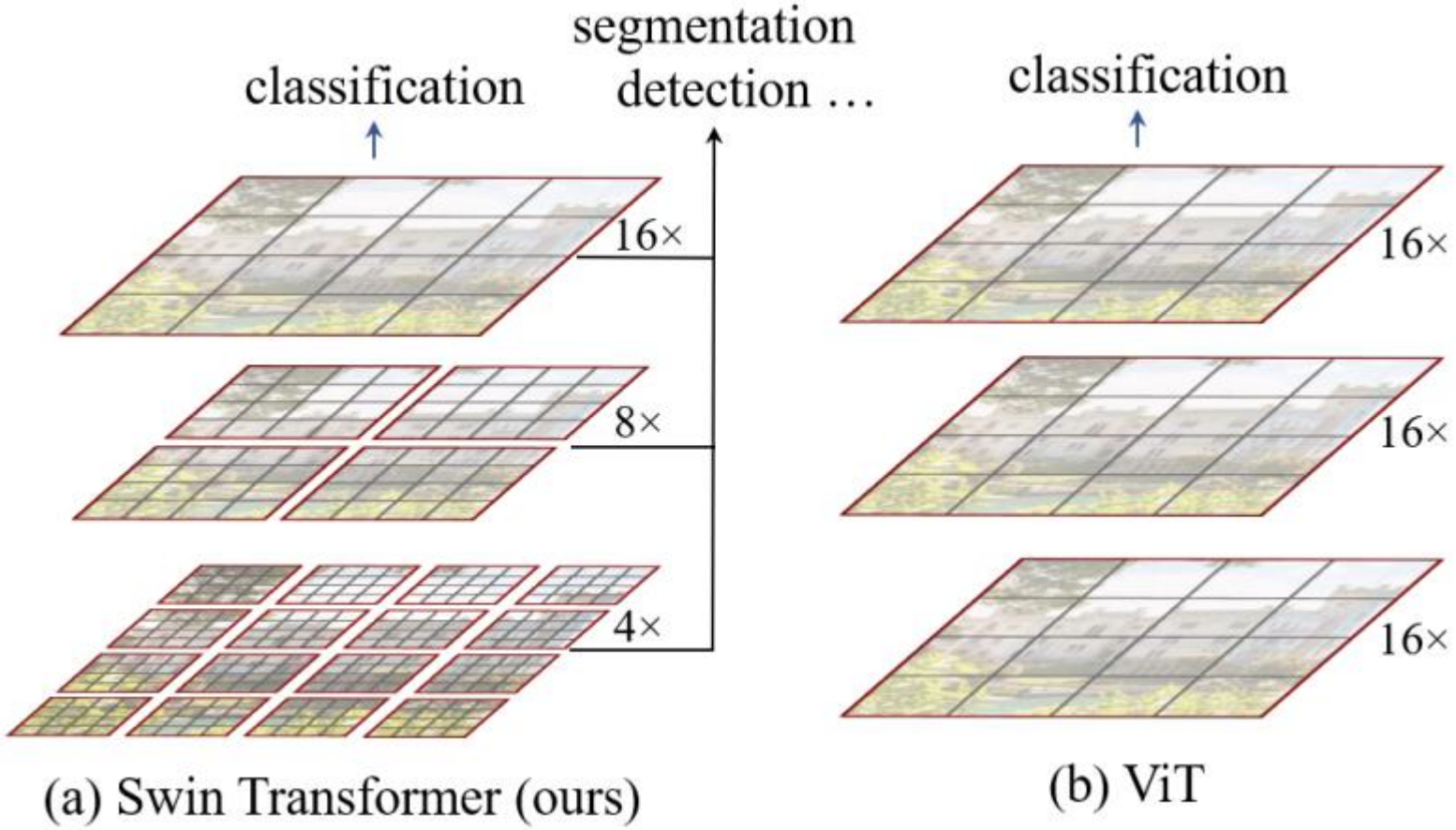


TABLE 1: Representative works of vision transformers.

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)**

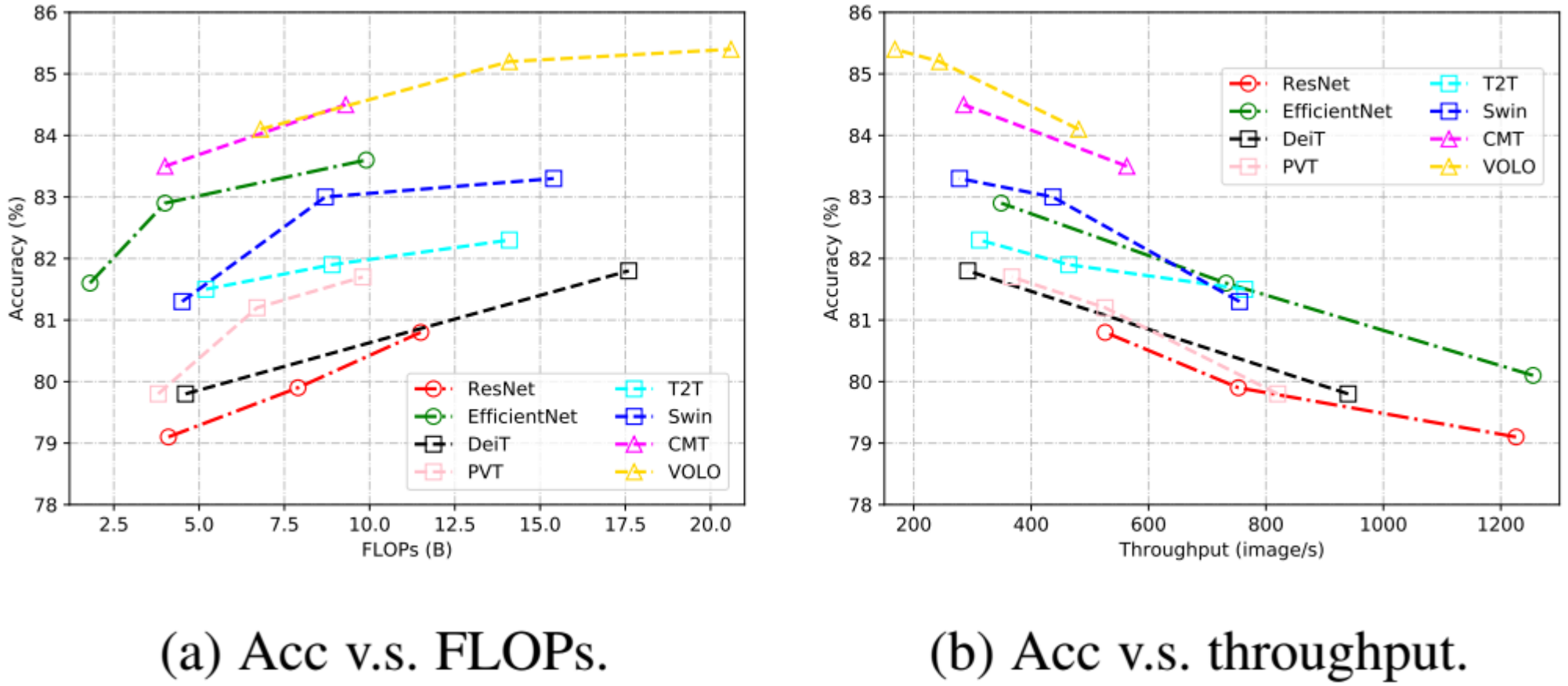
ImageNet result comparison of representative CNN and vision transformer models

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.

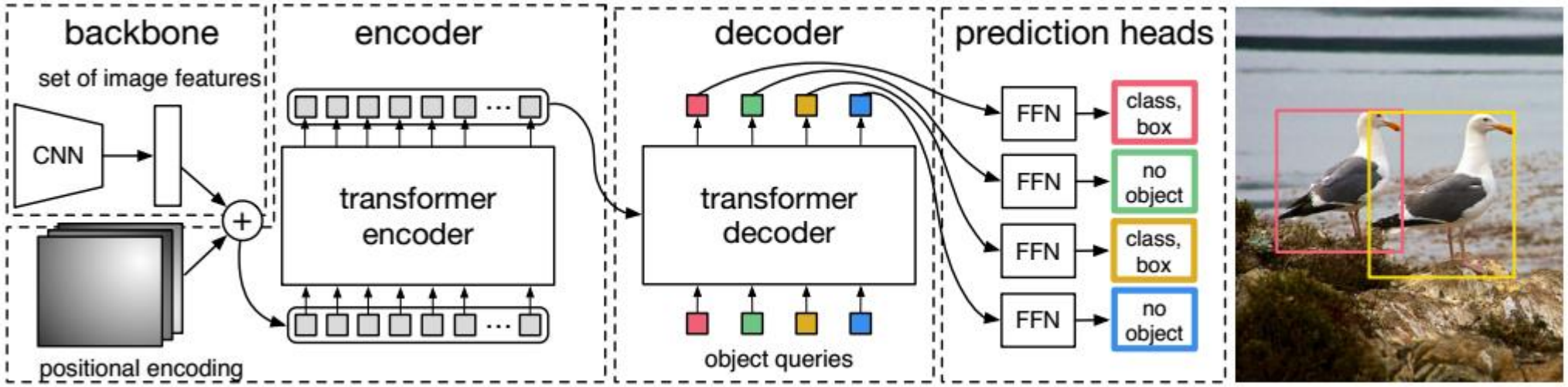
Throughput is how much information actually gets delivered in a certain amount of time.

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

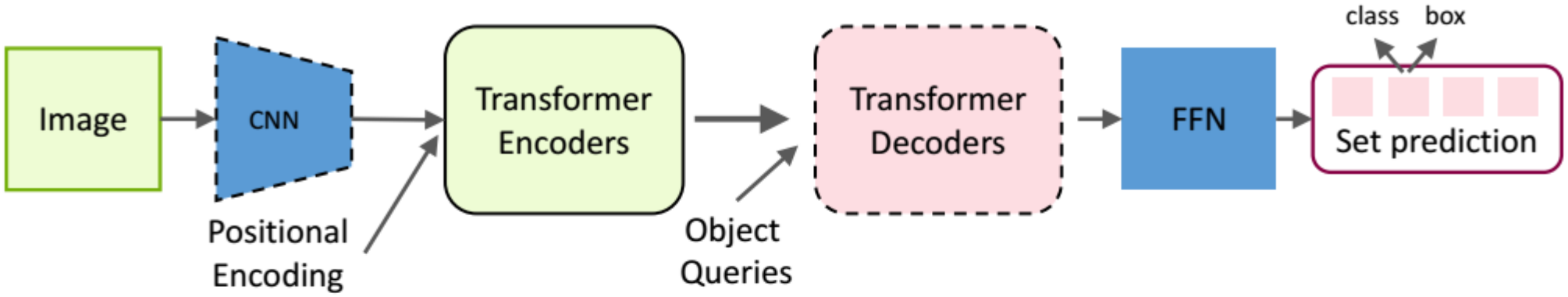
Transformers in Detection



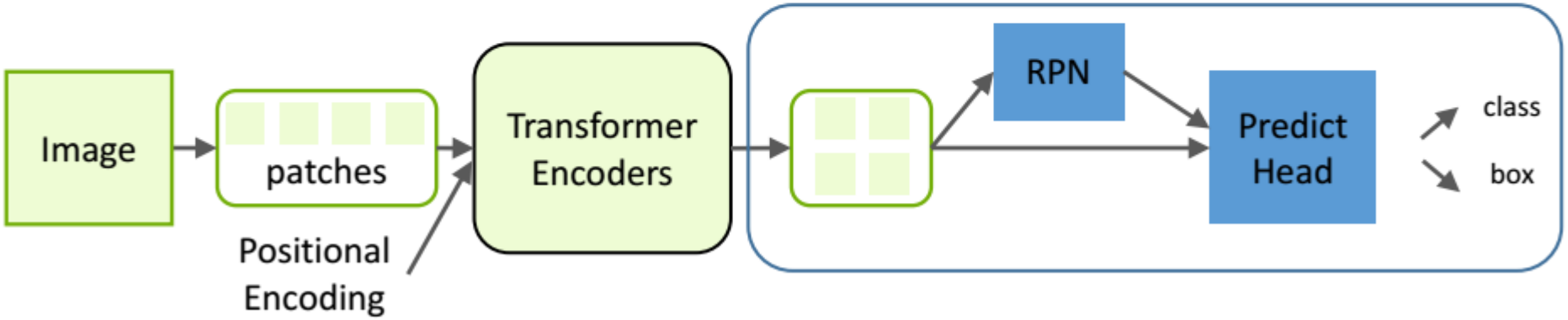
The overall architecture of DETR

*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

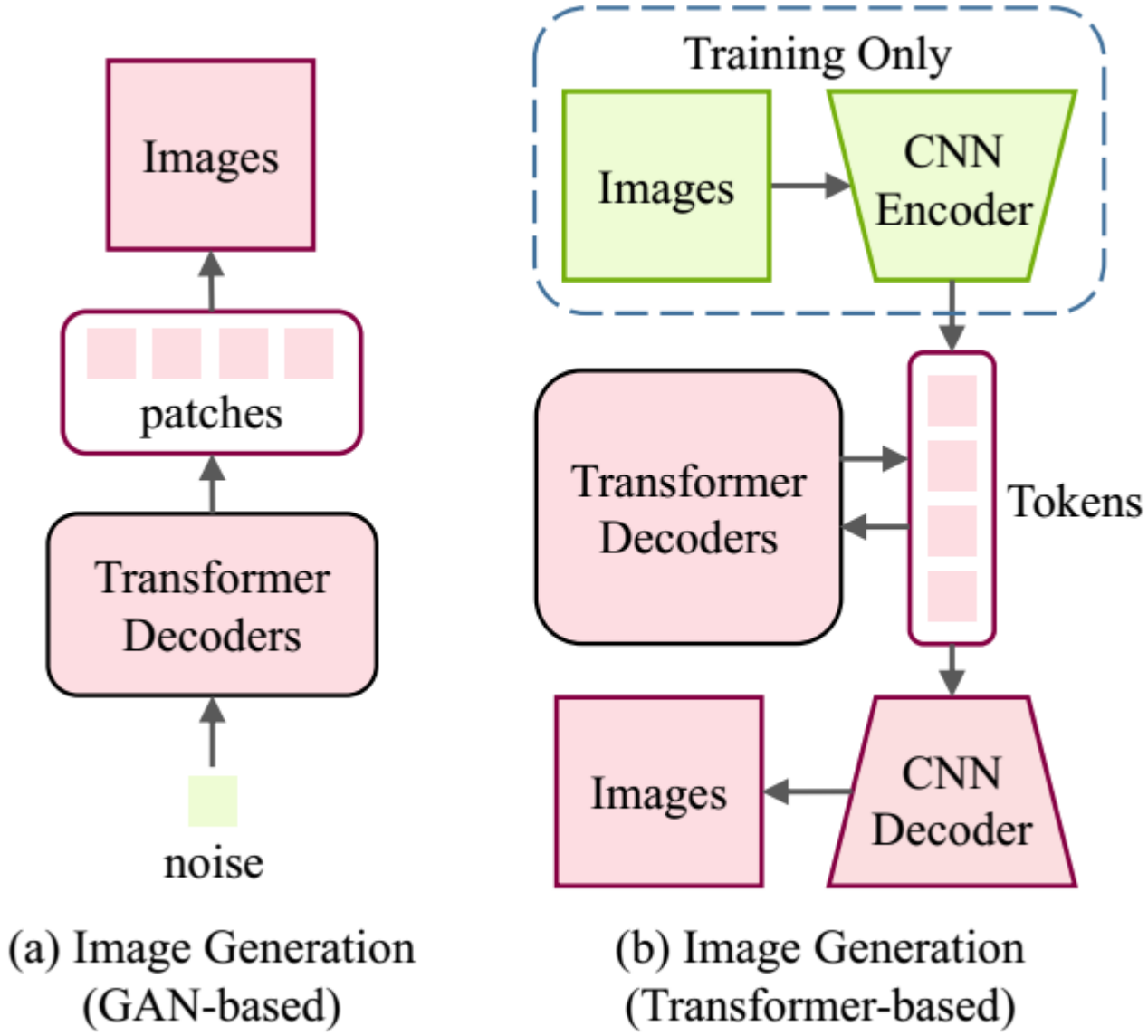


(b) Transformer-based backbone for detection

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



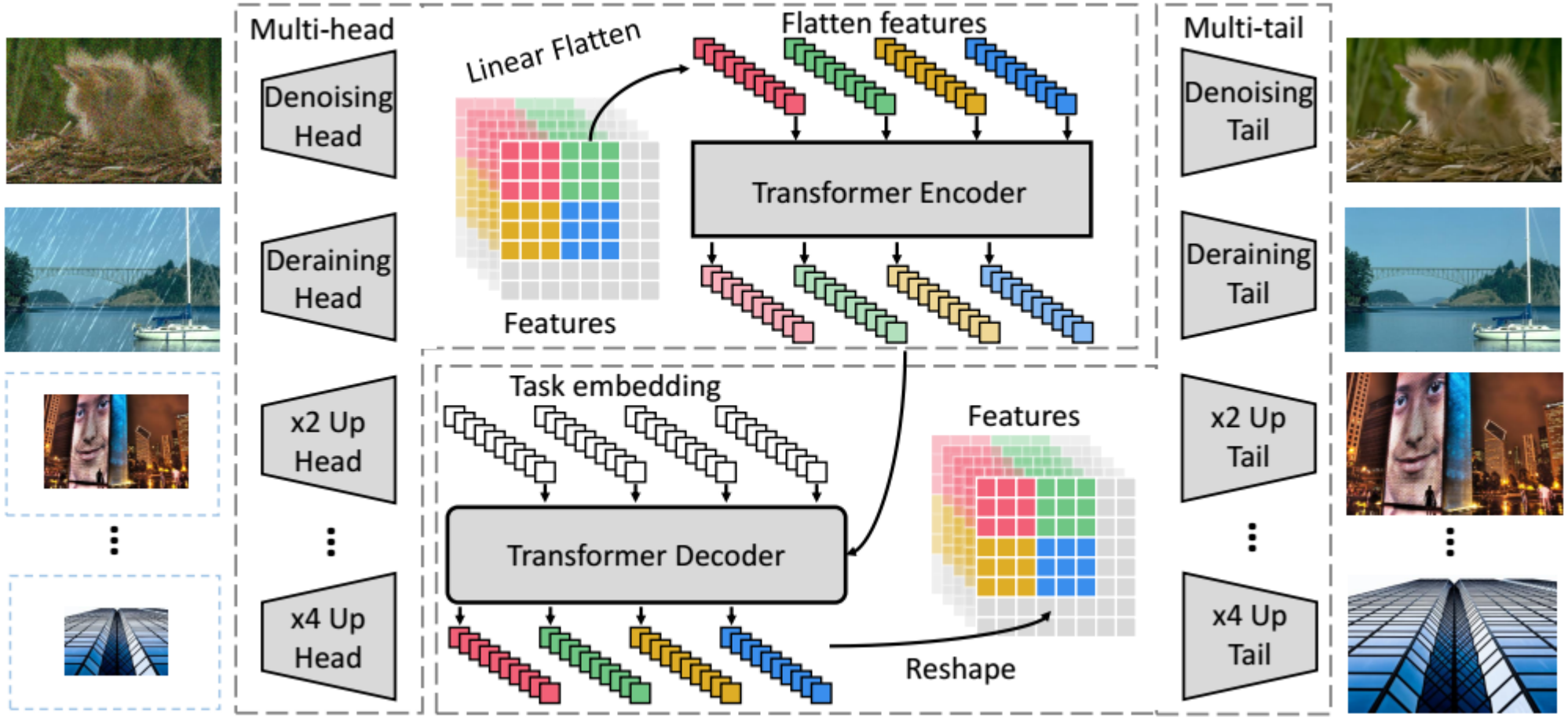
A generic framework for transformer in image generation

Transformers in Image Enhancement

IPT

Image Processing Transformer

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

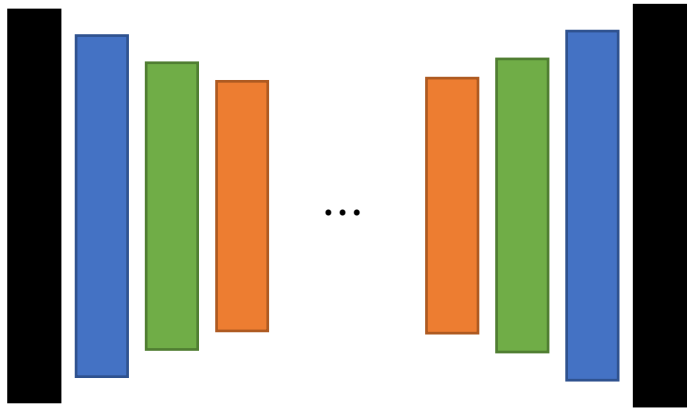


IPT fully utilizes the advantages of transformers by using large pre-training datasets. It achieves state-of-the-art performance in several image processing tasks, including super-resolution, denoising, and deraining.

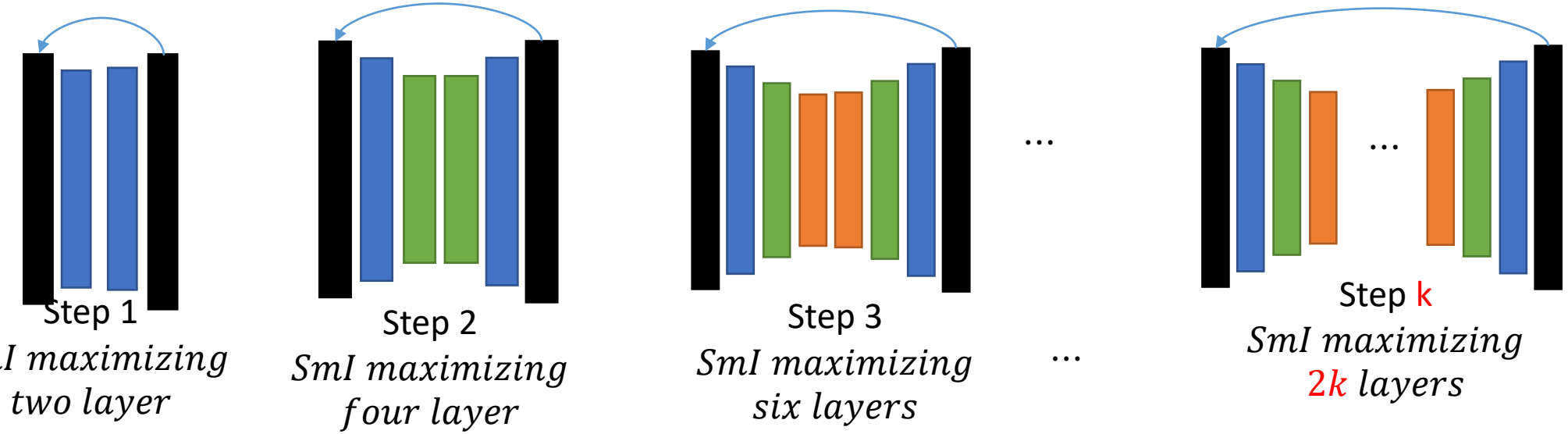
3.3.8 Layer-wise Forward Learning Part 3

Layer Wise Forward Learning of an Auto-Encoder

- Smoothness index can be used to train an auto-encoder step by step in a forward manner.
- At the first step two hidden layers are trained and then by each further step two new layers are added and trained to the AE.
- At each step, the training procedure is based on “SmI” maximizing.
- In fact, the smoothness index is approximated by a triplet loss and then the loss is minimized.
- Each example at the output layer decreases(increases) its distance to one of its K nearest neighbors that has nearest (furthest) distance at the target(input) space (reconstruction)



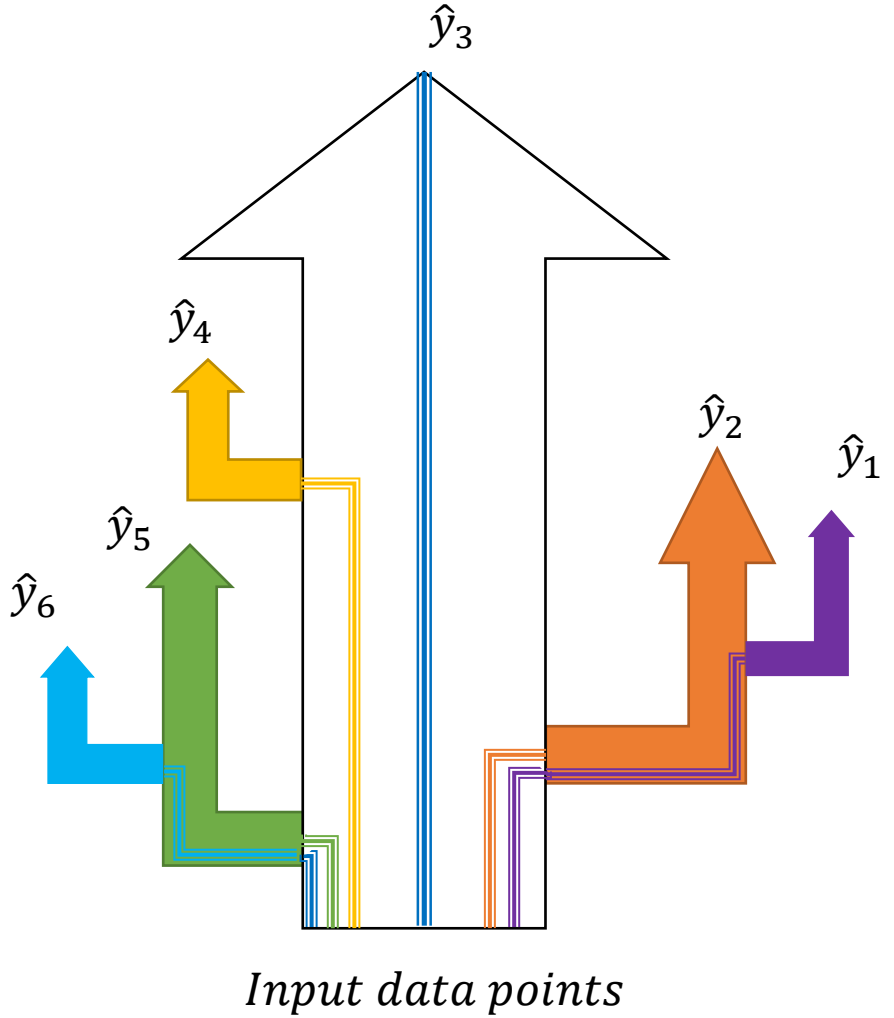
Layer-wise Learning through Smoothness Maximizing



3.3.4 Layer-wise branching

Layer-wise branching

- In a classification (regression) problem, the required features may be extracted earlier or later through layers of a DNN.
- If we design a new architecture of DNN which can conduct earlier features in some independent paths, more compact and more generalized DNNs can be learned.
- In such an architecture, there are a main body and some branches. The main body forms the main flow of data but each branch process a part of data-flow.
- We can use complexity measures like SI(SmI) to distinguish early features from other features and form a new branch for it.
- Each branch like the man body can define some sub-braches, too.
- In such a DNN, the amount of data-flow decreases through the main body just after each branching.



Some notes

- Branch in point where an increasing jump on SI(SmI) is occurred.
- Find the appropriate part of data (class/output/examples) to be branched.
- Branching is done as a forward design operation

Utilizing SI for Branching

DukeMTMC-reID: 34183 images

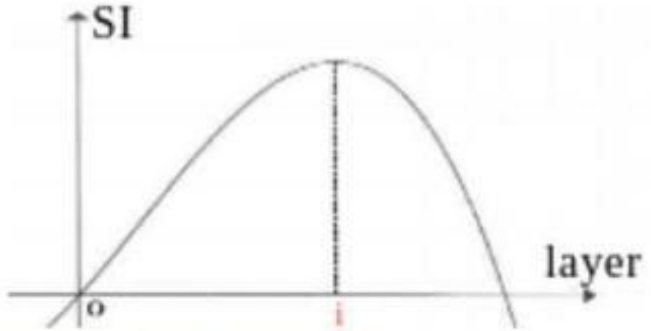
pretrained network:

person re identification

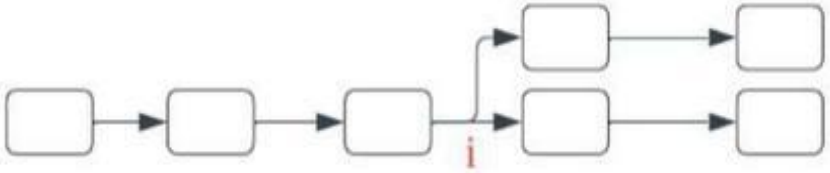
{ ID,  }

Task1: person reID

1.Check the information flow through network for attribute recognition



2.generating branches

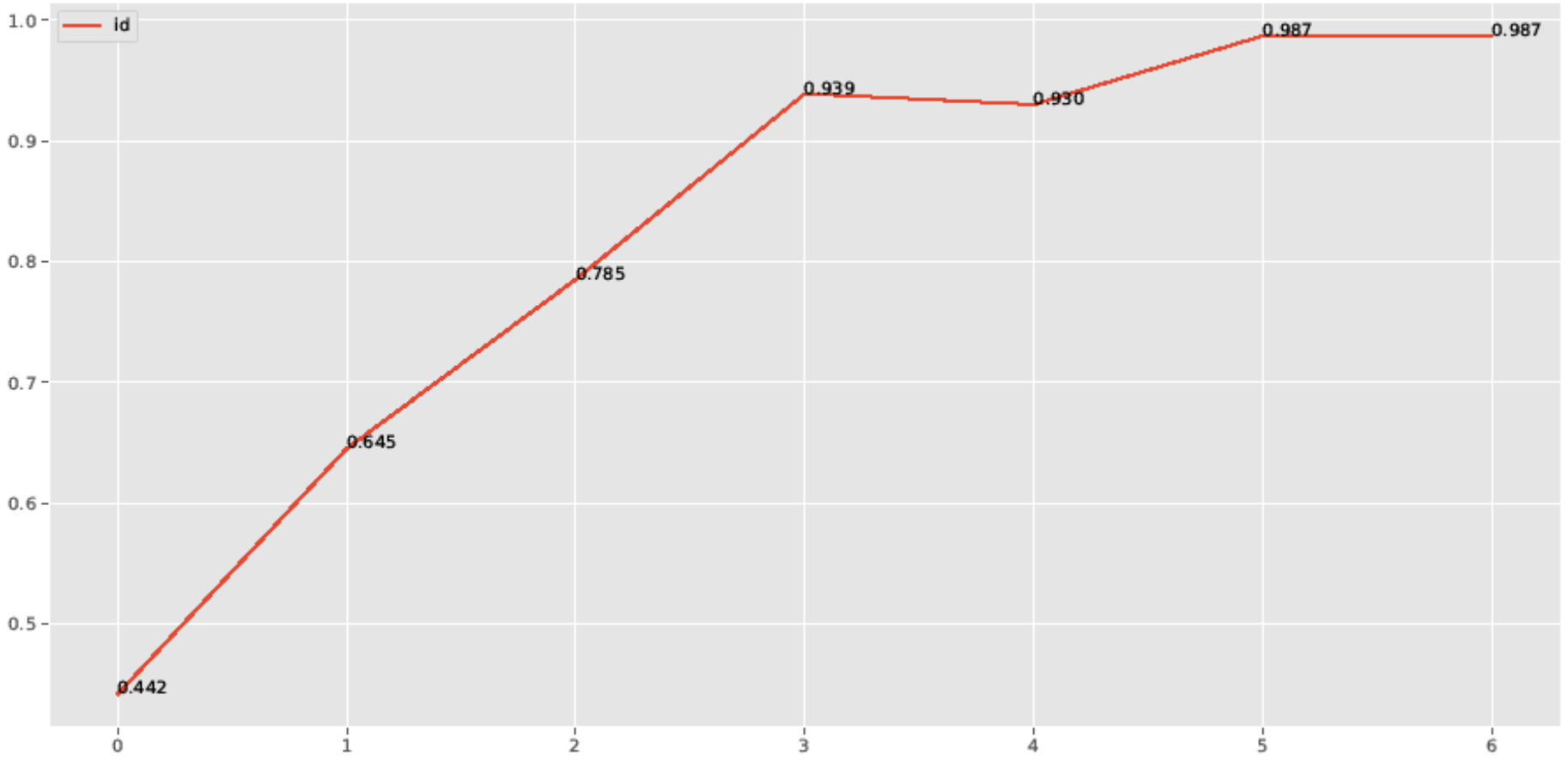


3.Train the branche for attribute recognition

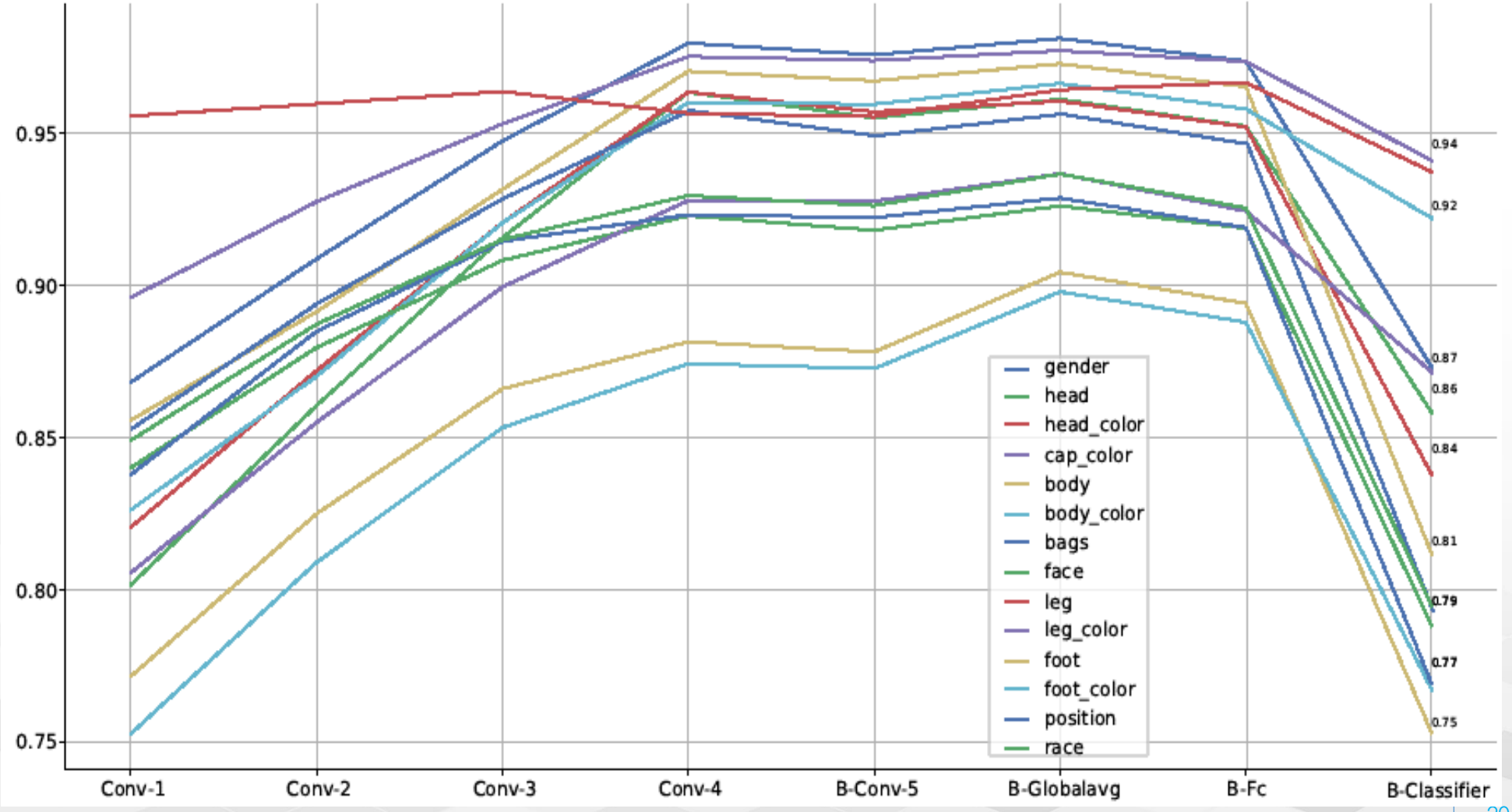
{ attributes,  }

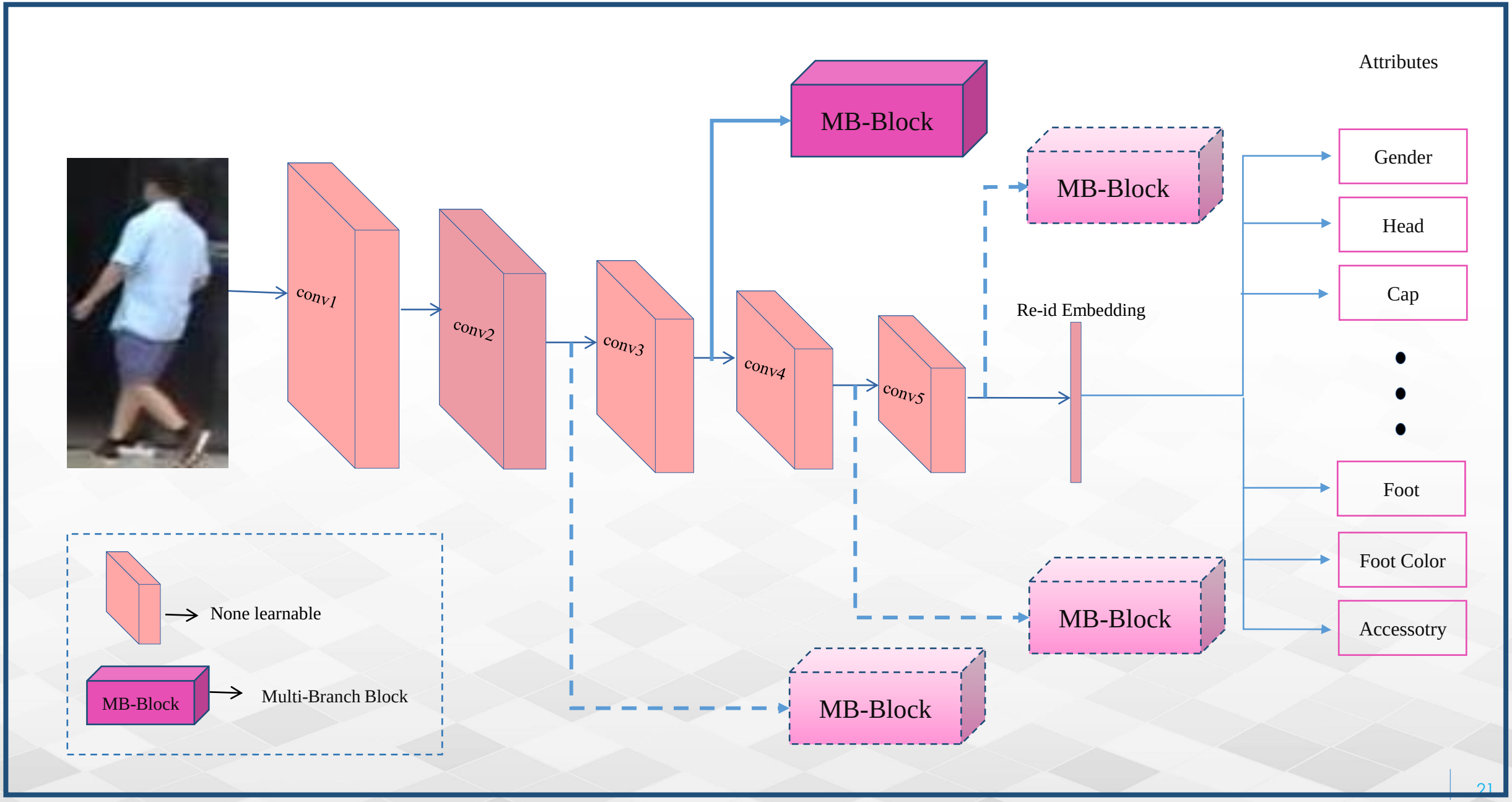
Task2:attribute recognition

Separation Index for a classification

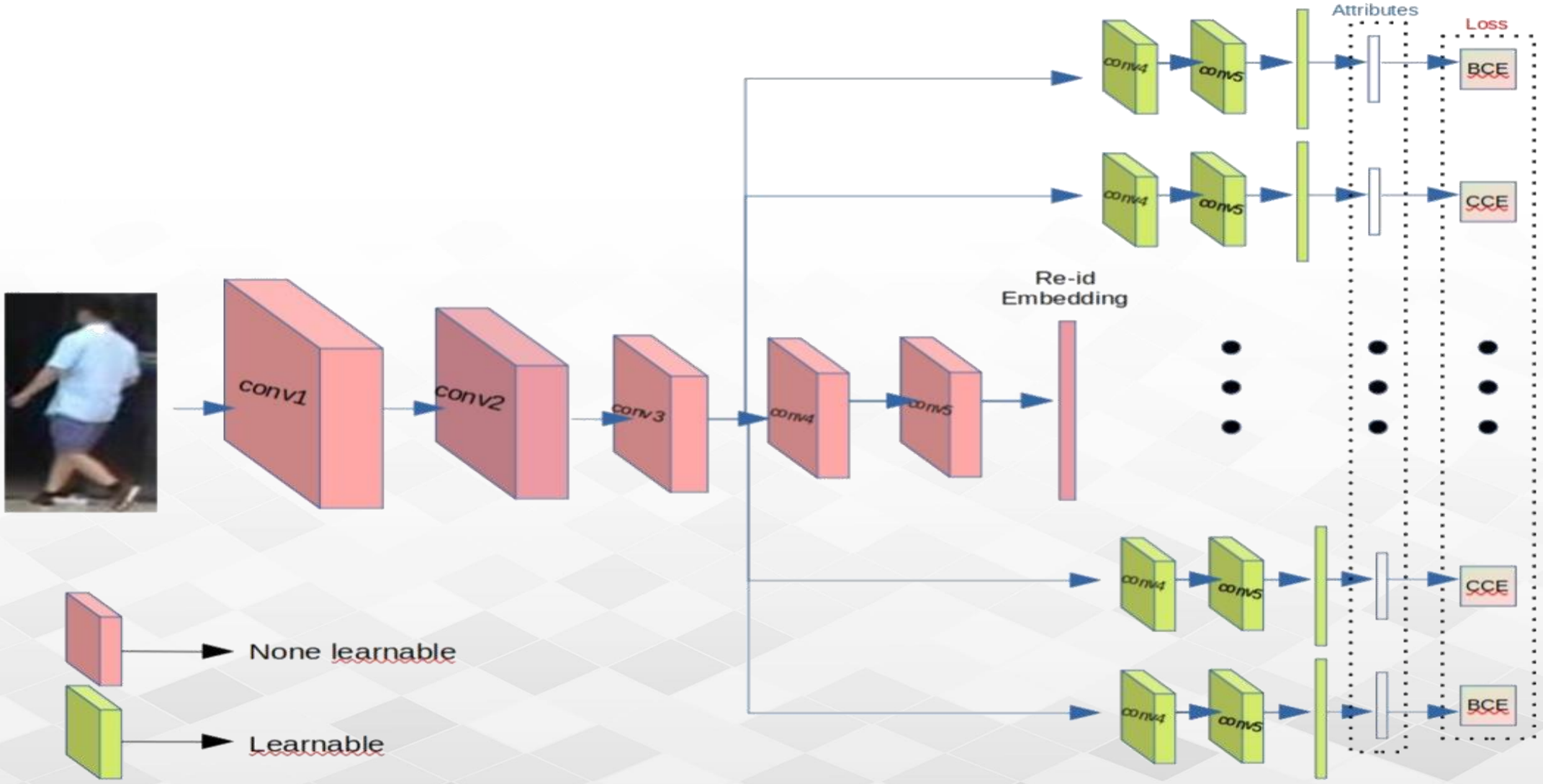


Separation Index for Multi-label Multi classification




Attributes
MB-Block
MB-Block
conv1
conv2
conv3
conv4
conv5
Re-id Embedding
Gender
Head
Cap
Foot
Foot Color
Accessotry
MB-Block
MB-Block
None learnable
MB-Block
Multi-Branch Block

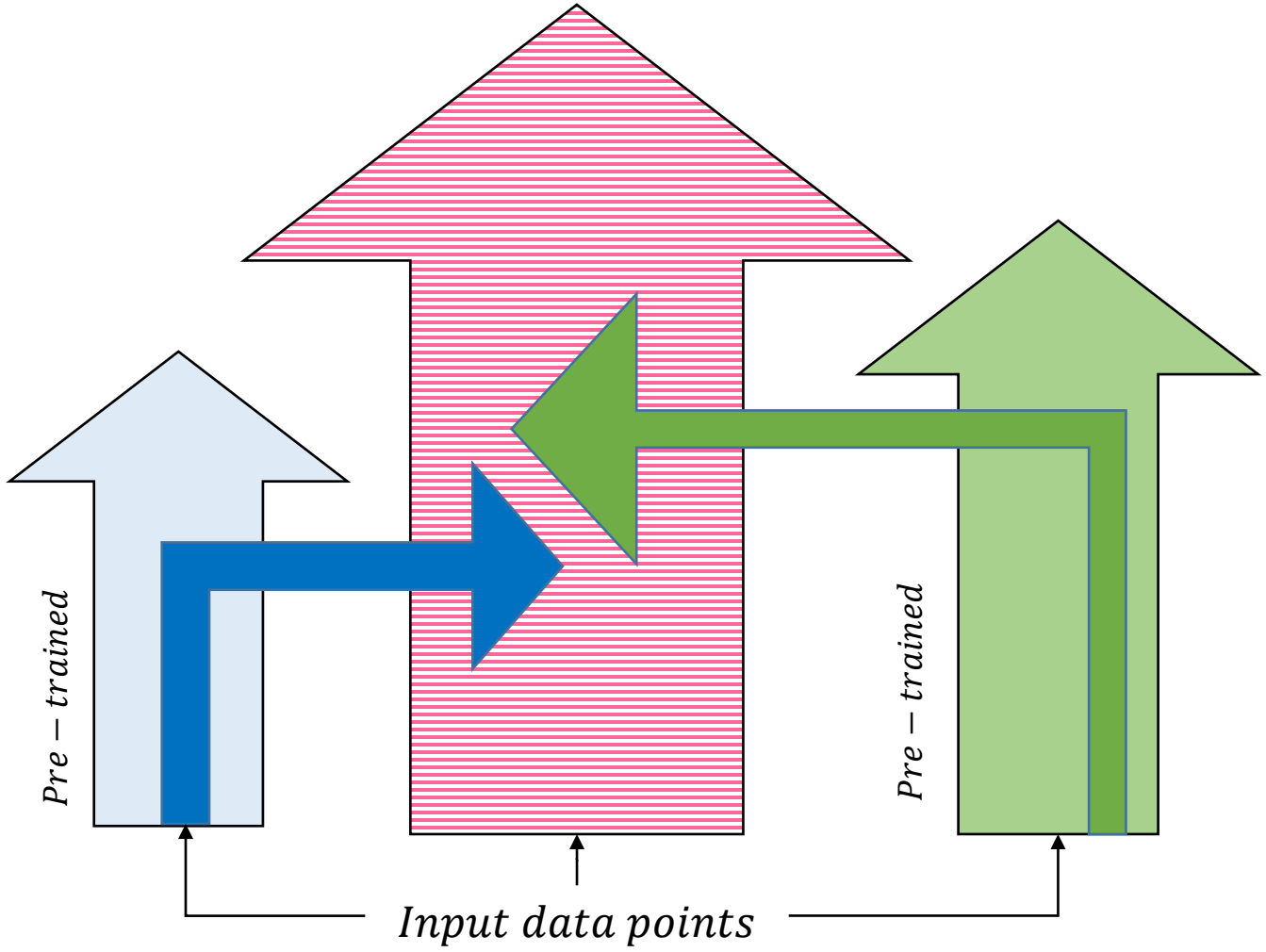
Multi-Branch Network



3.3.6 Layer-wise Fusion

Layer-wise Fusion

- In a classification (regression) problem, the required features may be extracted by fusing some prepared features from some pre-trained models.
- If we design a new architecture of DNN which can combine desired features from some pre-trained DNNs, more generalized DNNs can be learned.
- In such an architecture, there are a main body which is integrated by some branches from other DNNs at different points. The main body forms the main flow of data but each branch insert information-flow.
- We can use complexity measures like SI(SmI) to find some desired features from other pre-trained DDNs.



Some notes

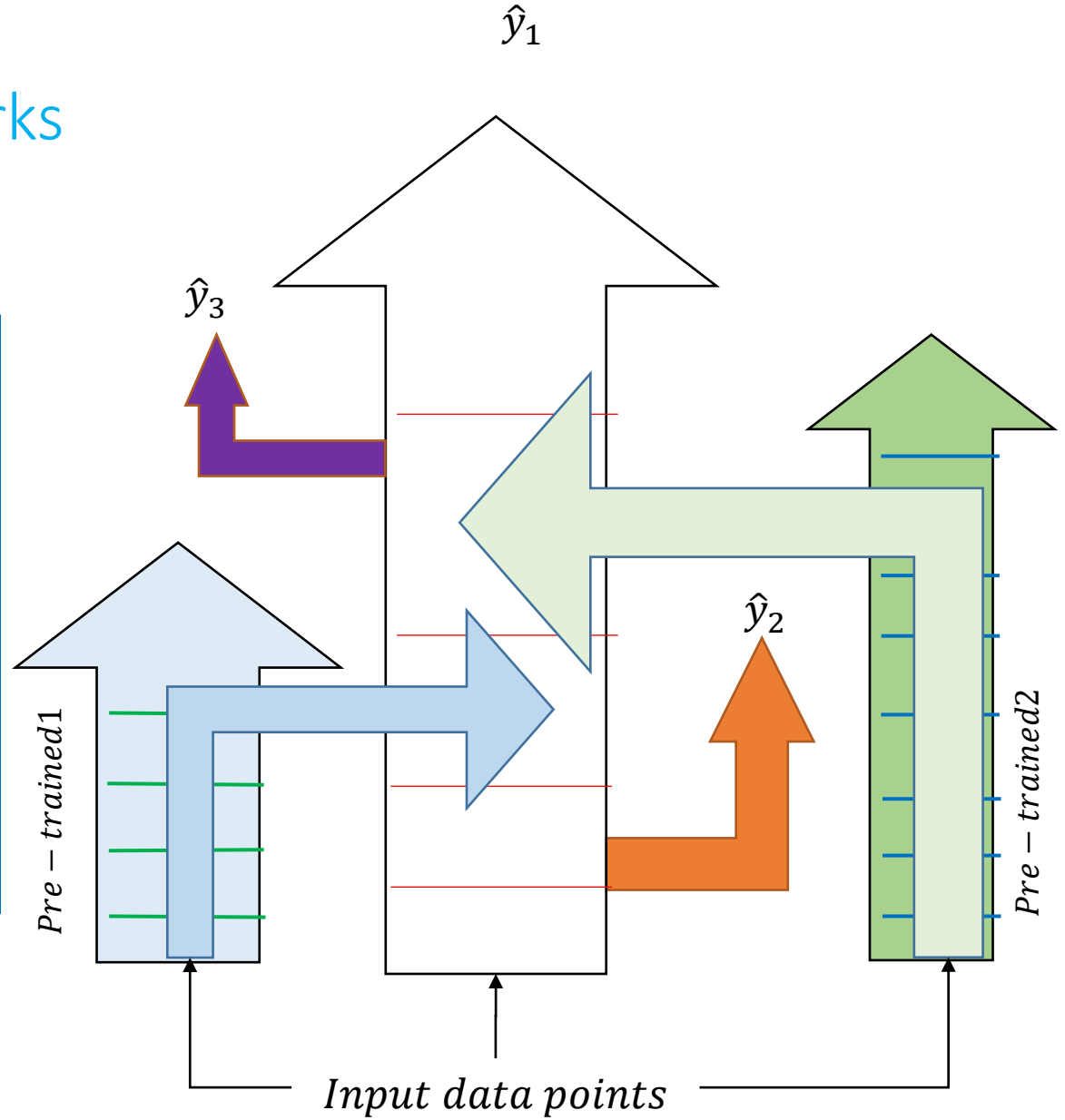
- Fuse at any point where SI(SmI) may increase significantly.
- Fusion is done in as a forward design operation.

3.3.7 Forward Design

Forward Design of Deep Neural Networks

In forward design of deep neural network, flowing tasks are done in a way that SI(SmI) maximizes in a classification(regression) problem.

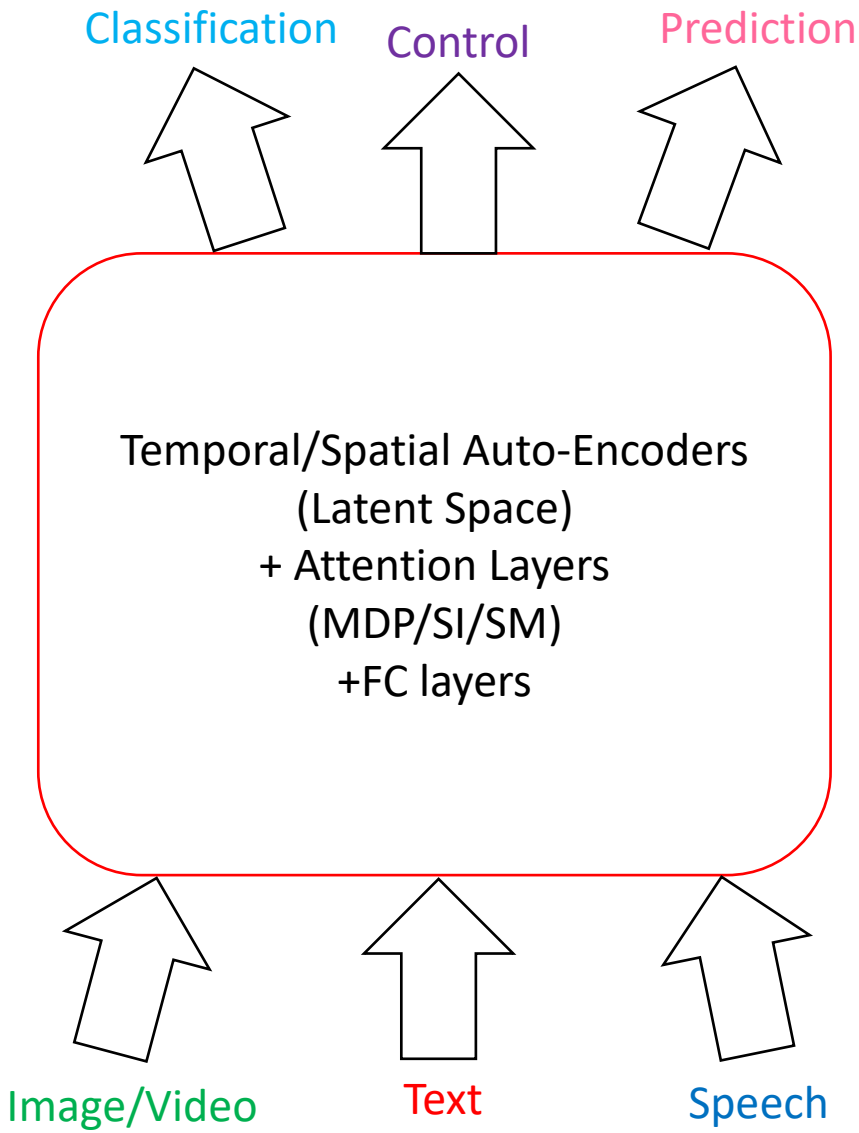
1. Organization of different layers
2. Adjusting the resolution, width, and depth of the network
3. Adding new branches on some layers where SI(SmI) increases significantly.
4. Fusing at some nodes from some other pre-trained networks through which SI(SmI) increases significantly.



3.3.8 Forward Multi-Task Design

A neural network which receives multi-modal data and it is trained to do multi-tasks in a supervised method. Such network works as a generalist AI agent. The key idea is that by appropriate temporal/spatial encoders, different data steams are encoded as the latent space; then the required features for each task are chosen from the latent space by using SmI(SI).

- Language Tasks
  - Speech recognition
  - Speech generating
  - ...
- Vision Tasks
  - Image recognition
  - Action Recognition
  - Text recognition
  - ...
- Physical world tasks
  - Motion tasks
  - Grasping tasks
  - ...

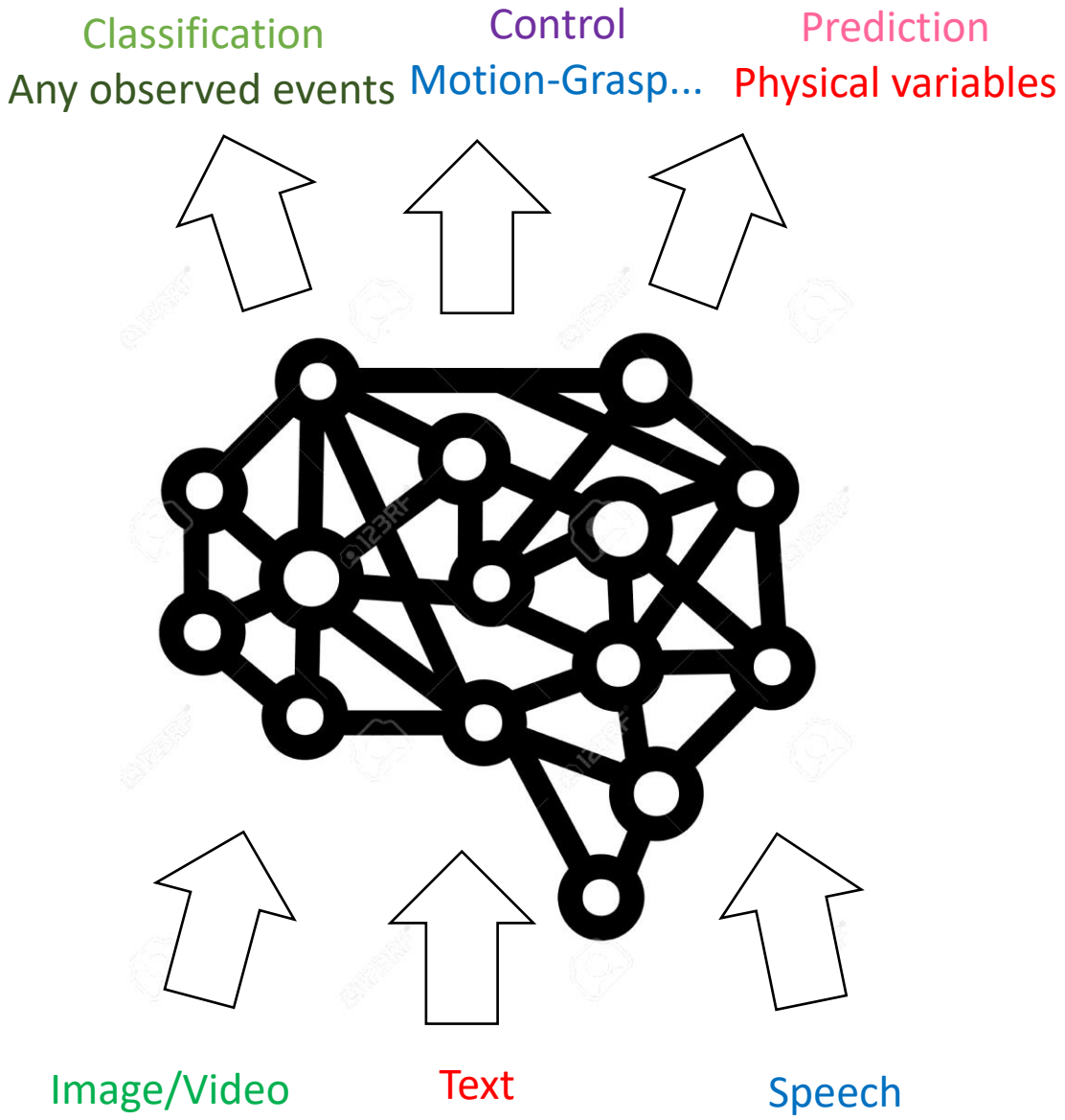


Narrow AI→ General and wide AI

Artificial General Inteligent

A neural network which receives multi-modal data and it is trained to do multi-tasks in an un-supervised method.
Such network works as a generalist AI agent which must learn without using labels and targets.
Some Steps

1. Receive all measuring signals: Visual, inertial, force,.....
2. Encode them as latent space by using appropriate spatial, temporal, and recurrent encoders
3. Using some attention layers and mechanisms to reveals some useful features which maximizes the gathered information from the environment
4. Learning to motion in the environment and to label objects .
5. Learning to do the required physical tasks.

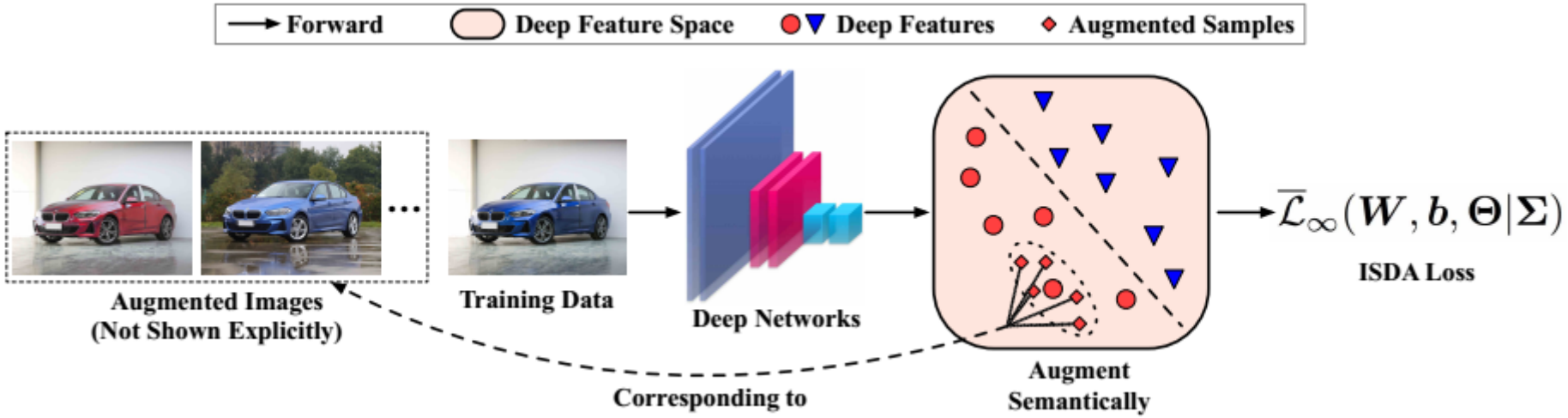


3.9 Related works in local Layer-wise learning

- To introduce a learning approach that is more plausible with the learning in the brain
- To have better learning and design in accuracy, compactness, speed,...
- To increase the generalization and robustness of DNNs

(1) Implicit Semantic Data Augmentation for Deep Networks

Yulin Wang et al. 2020, Department of Automation, Tsinghua University, Beijing, China



The proposed implicit semantic data augmentation (ISDA) has two important components, i.e., (1) online estimation of class-conditional covariance matrices and (2) optimization with a robust loss function.

The first component aims to find a distribution from which we can sample meaningful semantic transformation directions for data augmentation, while the second saves us from explicitly generating a large amount of extra training data, leading to remarkable efficiency compared to existing data augmentation techniques.

Comparison Tables

$Image\ net \rightarrow$

$Cifar10 + Cifar100 \downarrow$

Table 1: Single crop error rates (%) of different deep networks on the validation set of ImageNet. We report the results of our implementation with and without ISDA. The better results are **bold-faced**, while the numbers in brackets denote the performance improvement achieved by ISDA. We also report the theoretical computational overhead and the additional training time introduced by ISDA in the last two columns, which is obtained with 8 Tesla V100 GPUs.

Networks	Params	Top-1 / Top-5 Error Rates (%)		Additional Cost (Theoretical)	Additional Cost (Wall Time)
		Baselines	ISDA		
ResNet-50 [4]	25.6M	23.0 / 6.8	**21.9**$_{(1.1)}$ **/ 6.3**	0.25%	7.6%
ResNet-101 [4]	44.6M	21.7 / 6.1	**20.8**$_{(0.9)}$ **/ 5.7**	0.13%	7.4%
ResNet-152 [4]	60.3M	21.3 / 5.8	**20.3**$_{(1.0)}$ **/ 5.5**	0.09%	5.4%
DenseNet-BC-121 [5]	8.0M	23.7 / 6.8	**23.2**$_{(0.5)}$ **/ 6.6**	0.20%	5.6%
DenseNet-BC-265 [5]	33.3M	21.9 / 6.1	**21.2**$_{(0.7)}$ **/ 6.0**	0.24%	5.4%
ResNeXt-50, 32x4d [33]	25.0M	22.5 / 6.4	**21.3**$_{(1.2)}$ **/ 5.9**	0.24%	6.6%
ResNeXt-101, 32x8d [33]	88.8M	21.1 / 5.9	**20.1**$_{(1.0)}$ **/ 5.4**	0.06%	7.9%

Table 2: Evaluation of ISDA on CIFAR with different models. The average test error over the last 10 epochs is calculated in each experiment, and we report mean values and standard deviations in three independent experiments. The best results are **bold-faced**.

Method	Params	CIFAR-10	CIFAR-100
ResNet-32 [4]	0.5M	$7.39 \pm 0.10\%$	$31.20 \pm 0.41\%$
ResNet-32 + ISDA	0.5M	$\mathbf{7.09 \pm 0.12\%}$	$\mathbf{30.27 \pm 0.34\%}$
ResNet-110 [4]	1.7M	$6.76 \pm 0.34\%$	$28.67 \pm 0.44\%$
ResNet-110 + ISDA	1.7M	$\mathbf{6.33 \pm 0.19\%}$	$\mathbf{27.57 \pm 0.46\%}$
SE-ResNet-110 [34]	1.7M	$6.14 \pm 0.17\%$	$27.30 \pm 0.03\%$
SE-ResNet-110 + ISDA	1.7M	$\mathbf{5.96 \pm 0.21\%}$	$\mathbf{26.63 \pm 0.21\%}$
Wide-ResNet-16-8 [35]	11.0M	$4.25 \pm 0.18\%$	$20.24 \pm 0.27\%$
Wide-ResNet-16-8 + ISDA	11.0M	$\mathbf{4.04 \pm 0.29\%}$	$\mathbf{19.91 \pm 0.21\%}$
Wide-ResNet-28-10 [35]	36.5M	$3.82 \pm 0.15\%$	$18.53 \pm 0.07\%$
Wide-ResNet-28-10 + ISDA	36.5M	$\mathbf{3.58 \pm 0.15\%}$	$\mathbf{17.98 \pm 0.15\%}$
ResNeXt-29, 8x64d [33]	34.4M	$3.86 \pm 0.14\%$	$18.16 \pm 0.13\%$
ResNeXt-29, 8x64d + ISDA	34.4M	$\mathbf{3.67 \pm 0.12\%}$	$\mathbf{17.43 \pm 0.25\%}$
DenseNet-BC-100-12 [5]	0.8M	$4.90 \pm 0.08\%$	$22.61 \pm 0.10\%$
DenseNet-BC-100-12 + ISDA	0.8M	$\mathbf{4.54 \pm 0.07\%}$	$\mathbf{22.10 \pm 0.34\%}$
DenseNet-BC-190-40 [5]	25.6M	3.52%	17.74%
DenseNet-BC-190-40 + ISDA	25.6M	**3.24%**	**17.42%**

Algorithm 1 The ISDA Algorithm.

Input: $\mathcal{D}$, λ_0
Randomly initialize $\boldsymbol{W}$, $\boldsymbol{b}$ and $\boldsymbol{\Theta}$
for $t = 0$ **to** T **do**
  Sample a mini-batch $\{\boldsymbol{x}_i, y_i\}_{i=1}^{B}$ from $\mathcal{D}$
  Compute $\boldsymbol{a}_i = G(\boldsymbol{x}_i, \boldsymbol{\Theta})$
  Estimate the covariance matrices Σ_1, Σ_2, ..., Σ_C
  Compute $\overline{\mathcal{L}}_\infty$ according to Eq. (4)
  Update $\boldsymbol{W}$, $\boldsymbol{b}$, $\boldsymbol{\Theta}$ with SGD
end for
Output: $\boldsymbol{W}$, $\boldsymbol{b}$ and $\boldsymbol{\Theta}$

$$\mathcal{L}_\infty(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{\Theta}|\Sigma) \leq \frac{1}{N}\sum_{i=1}^{N} -log\left(\frac{e^{\boldsymbol{w}_{y_i}^T \boldsymbol{a}_i + b_{y_i}}}{\sum_{j=1}^{C} e^{\boldsymbol{w}_j^T \boldsymbol{a}_i + b_j + \frac{\lambda}{2}(\boldsymbol{w}_j^T - \boldsymbol{w}_{y_i}^T)\Sigma_{y_i}(\boldsymbol{w}_j - \boldsymbol{w}_{y_i})}}\right) \triangleq \overline{\mathcal{L}}_\infty.$$

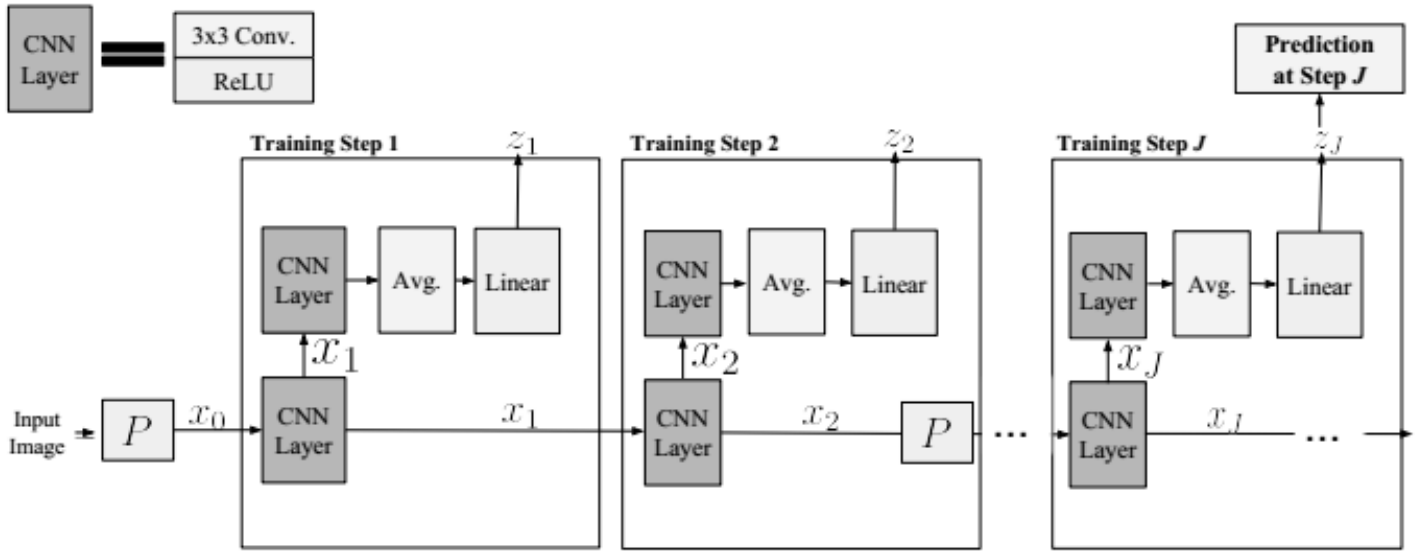
λ is a positive coefficient to control the strength of semantic data augmentation. The strength at initial epochs is chosen low but linearly it increases: $\lambda = (t/T) \times \lambda 0$

(2) Greedy Layerwise Learning Can Scale to ImageNet

Belilovsky et al. (University of Montral)-*Proceedings of the 36 th International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019.

We find that solving sequential 1-hidden-layer auxiliary problems lead to a CNN that exceeds AlexNet performance on ImageNet.

Extending this training methodology to construct individual layers by solving 2-and-3-hidden layer auxiliary problems, we obtain an 11-layer network that exceeds several members of the VGG model family on ImageNet, and can train a VGG-11 model to the same accuracy as end-to-end learning.



k = 2-hidden layer

Algorithm 1 Layer Wise CNN

Input: Training samples $\{x_0^n, y^n\}_{n \leq N}$
for $j \in 0..J-1$ **do**
  Compute $\{x_j^n\}_{n \leq N}$ (via Eq.(1))
  $(\theta_j^*, \gamma_j^*) = \arg\min_{\theta_j, \gamma_j} \hat{\mathcal{R}}(z_{j+1}; \theta_j, \gamma_j)$
end for

Comparison tables

Layer-wise Trained	Acc. (Ens.)
SimCNN ($k = 1$ train)	88.3 (88.4)
SimCNN ($k = 2$ train)	90.4 (90.7)
SimCNN($k = 3$ train)	91.7 (**92.8**)
BoostResnet (Huang et al., 2017)	82.1
ProvableNN (Malach et al., 2018)	73.4
(Mosca et al., 2017)	81.6
Reference e2e	
AlexNet	89
VGG [1]	92.5
WRN 28-10 (Zagoruyko et al. 2016)	**96.0**
Alternatives	[Ref.]
Scattering + Linear	82.3
FeedbackAlign (Bartunov et al., 2018)	62.6 [67.6]

Table 2. Results on CIFAR-10. Compared to the few existing methods using *only* layerwise training schemes we report much more competitive results to well known benchmark models that like ours do not use skip connnections.In brackets e2e trained version of the model is shown when available.

	Top-1 (Ens.)	Top-5 (Ens.)
SimCNN ($k = 1$ train)	58.1 (59.3)	79.7 (80.8)
SimCNN ($k = 2$ train)	65.7 (67.1)	86.3 (87.0)
SimCNN ($k = 3$ train)	69.7 (71.6)	88.7 (89.8)
VGG-11 ($k = 3$ train)	67.6 (70.1)	88.0 (89.2)
VGG-11 (e2e train)	67.9	88.0
Alternative	[Ref.]	[Ref.]
DTargetProp (Bartunov et al., 2018)	1.6 [28.6]	5.4 [51.0]
FeedbackAlign (Xiao et al., 2019)	6.6 [50.9]	16.7 [75.0]
Scat. + Linear (Oyallon et al., 2018)	17.4	N/A
Random CNN	12.9	N/A
FV + Linear (Sánchez et al., 2013)	54.3	74.3
Reference e2e CNN		
AlexNet	56.5	79.1
VGG-13	69.9	89.3
VGG-19	72.9	90.9
Resnet-152	78.3	94.1

Table 3. Single crop validation acc. on ImageNet. Our SimCNN models use $J = 8$. In parentheses see the ensemble prediction.

k = 1 :the auxiliary classifier consists of only the linear *A* and *L* operators (1-hidden layer NN)
K>1 :K hidden layers for auxiliary classifiers

(3) Hebbian Semi-Supervised Learning in a Sample Efficiency Setting?

Gabriele Lagani, 2021, *Computer Science Department, University of Pisa, Pisa, Italy*

Highlights

1. A semi supervised training strategy that combines Hebbian learning with gradient descent

   Hebbian for internal layers + SGD for the final fully connected layer

2. All internal layers (both convolutional and fully connected) are pre-trained using an unsupervised approach based on Hebbian learning
3. The last fully connected layer (the classification layer) is trained using Stochastic Gradient Descent (SGD)
4. The learning approach has been applied to CIFAR10, CIFAR100, tiny ImageNet
5. In regimes when the number of available labeled samples is low the proposed approach outperforms the other approaches in almost all the cases

Hebian

The classification layer is placed on top of the various internal layers (*L*1; : : : ; *L*5)

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = \eta\, y\, \mathbf{x}$$

"neurons that fire together wire together."

$$\Delta\mathbf{w} = \eta\, y\, \mathbf{x} - \lambda\, \mathbf{w}$$

Adding Stable Term

In particular, the term λ can be chosen in the form $\lambda = \eta y$

$$\Delta\mathbf{w} = \eta\, y\, \mathbf{x} - \eta\, y\, \mathbf{w} = \eta\, y\, (\mathbf{x} - \mathbf{w})$$

In a complex neural network, we need a strategy to prevent neurons from learning redundant information.

The Hebbian PCA learning rule is obtained by minimizing the representation error loss function, defined as:

$$L(\mathbf{w_i}) = E[(\mathbf{x} - \sum_{j=1}^{i} y_j\, \mathbf{w_j})^2]$$

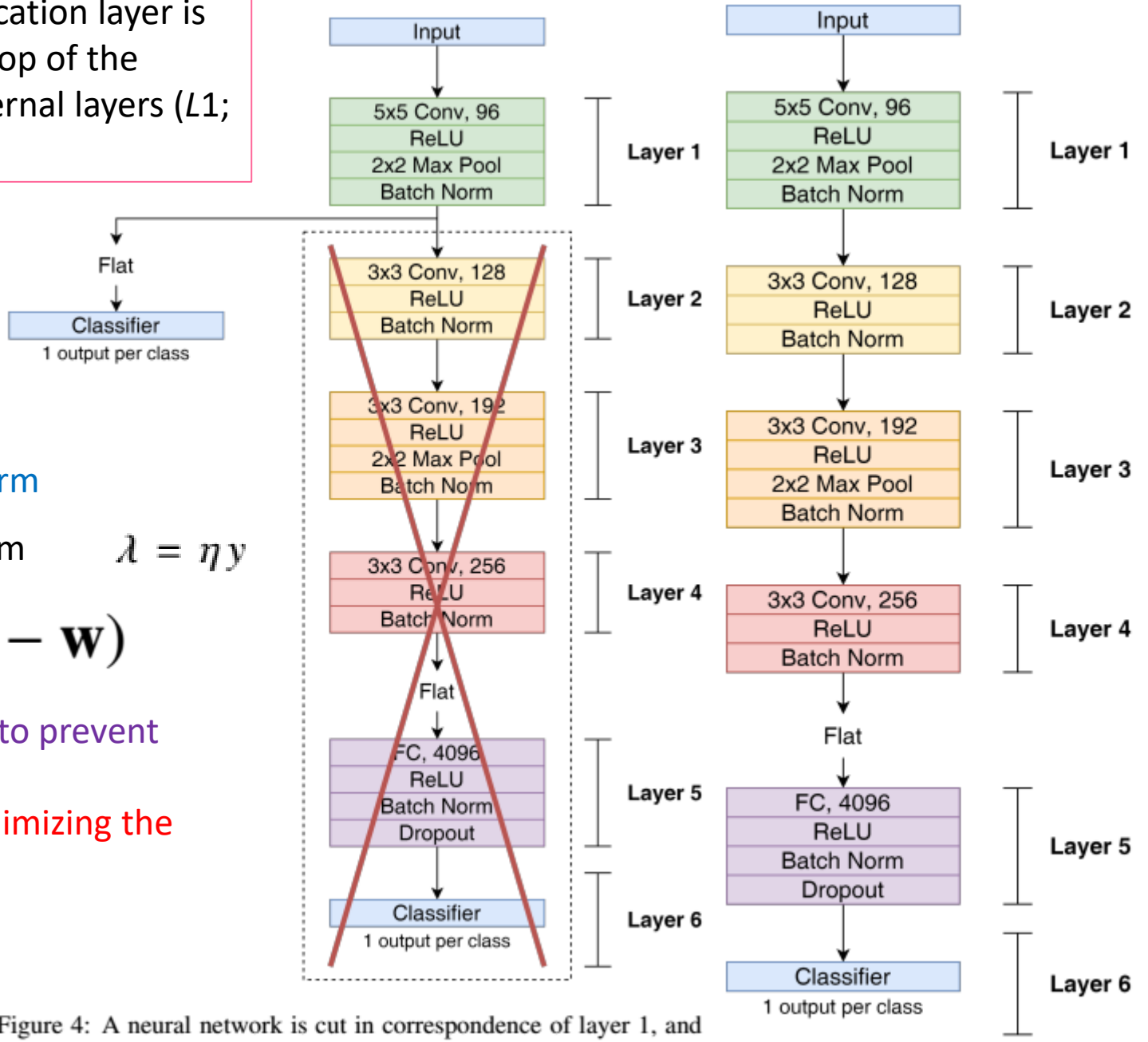


Figure 4: A neural network is cut in correspondence of layer 1, and a linear classifier is placed on top of the features extracted from that layer, in order to evaluate their quality in classification tasks.

: 1: The neural network used for the experiments.

Comparison Table(CIFAR10)

Table 1: CIFAR10 accuracy (top-1) and 95% confidence intervals, obtained with a linear classifier on top of various layers, for the various sample efficiency regimes. Results obtained with supervised backprop (BP), VAE-based semi-supervised approach(VAE), Hebbian PCA (HPCA), and HPCA plus Fine Tuning (HPCA+FT) are compared. It is possible to observe that, in regimes where the number of available samples is low
(roughly between 1% and 5% of the total available samples), HPCA performs better than BP and VAE approaches in almost all the cases, leading to an improvement up to almost 5% (on layer 3, in the 1% regime) w.r.t. non-Hebbian approaches. HPCA+FT helps to further boost accuracy.

Regime: r%- where the percentage of labeled samples is r%

Regimes	Method	L1	L2	L3	L4	L5
1%	BP	33.27 ±0.44	34.56 ±0.34	36.80 ±0.52	35.47 ±0.58	35.18 ±0.57
	VAE	33.54 ±0.27	34.41 ±0.84	29.92 ±1.25	24.91 ±0.66	22.54 ±0.60
	HPCA	36.78 ±0.46	37.26 ±0.14	41.31 ±0.57	39.33 ±0.72	38.46 ±0.44
	HPCA+FT	**37.01** ±0.42	**37.65** ±0.19	**41.88** ±0.53	**40.06** ±0.65	**39.75** ±0.50
2%	BP	37.33 ±0.25	39.01 ±0.19	42.34 ±0.51	41.50 ±0.32	41.10 ±0.39
	VAE	37.65 ±0.35	39.13 ±0.40	36.52 ±0.47	29.39 ±0.32	26.78 ±0.72
	HPCA	41.13 ±0.30	**41.63** ±0.18	45.76 ±0.41	44.70 ±0.45	43.15 ±0.45
	HPCA+FT	**41.60** ±0.28	42.12 ±0.24	**46.56** ±0.38	**45.61** ±0.19	**45.51** ±0.43
3%	BP	40.49 ±0.26	41.90 ±0.40	45.13 ±0.53	45.26 ±0.22	44.52 ±0.24
	VAE	41.22 ±0.27	43.16 ±0.44	42.60 ±0.87	31.91 ±0.44	29.00 ±0.33
	HPCA	44.16 ±0.42	44.84 ±0.08	48.92 ±0.17	47.70 ±0.57	45.60 ±0.27
	HPCA+FT	**44.74** ±0.08	**45.61** ±0.28	**49.75** ±0.41	**48.94** ±0.45	**48.80** ±0.27
4%	BP	43.38 ±0.22	45.43 ±0.18	49.51 ±0.49	48.96 ±0.48	48.80 ±0.24
	VAE	44.39 ±0.30	45.88 ±0.39	46.01 ±0.40	34.26 ±0.21	31.15 ±0.35
	HPCA	46.37 ±0.16	47.16 ±0.28	50.70 ±0.26	49.45 ±0.15	47.75 ±0.54
	HPCA+FT	**47.10** ±0.25	**48.26** ±0.09	**52.00** ±0.16	**51.05** ±0.29	**51.28** ±0.28
5%	BP	45.11 ±0.21	47.57 ±0.29	50.61 ±0.32	50.54 ±0.23	50.42 ±0.14
	VAE	46.31 ±0.39	48.21 ±0.21	48.98 ±0.34	36.32 ±0.35	32.75 ±0.32
	HPCA	47.51 ±0.65	48.69 ±0.37	51.69 ±0.56	50.44 ±0.43	48.51 ±0.32
	HPCA+FT	**48.49** ±0.44	**50.14** ±0.46	**53.33** ±0.52	**52.49** ±0.16	**52.20** ±0.37
10%	BP	51.60 ±0.40	54.60 ±0.31	57.97 ±0.28	**57.63** ±0.23	57.30 ±0.22
	VAE	53.83 ±0.26	**56.33** ±0.22	57.85 ±0.22	52.26 ±1.08	45.67 ±1.15
	HPCA	52.57 ±0.29	53.29 ±0.25	56.09 ±0.38	54.24 ±0.28	52.68 ±0.36
	HPCA+FT	**54.36** ±0.32	56.08 ±0.28	**58.46** ±0.15	56.54 ±0.23	**57.35** ±0.18
25%	BP	60.43 ±0.26	64.96 ±0.18	66.63 ±0.17	68.04 ±0.05	68.04 ±0.20
	VAE	**62.51** ±0.24	**67.26** ±0.32	**68.48** ±0.21	**68.79** ±0.29	**68.70** ±0.15
	HPCA	58.30 ±0.28	59.20 ±0.24	59.98 ±0.20	57.54 ±0.20	56.46 ±0.18
	HPCA+FT	61.45 ±0.26	65.25 ±0.16	64.71 ±0.17	62.43 ±0.13	64.77 ±0.22
100%	BP	61.59 ±0.08	67.67 ±0.11	73.87 ±0.15	83.88 ±0.04	84.71 ±0.02
	VAE	**67.53** ±0.22	**75.83** ±0.31	**80.78** ±0.28	**84.27** ±0.35	**85.23** ±0.26
	HPCA	64.69 ±0.29	65.92 ±0.14	64.43 ±0.21	61.24 ±0.22	61.16 ±0.33
	HPCA+FT	66.76 ±0.13	75.16 ±0.20	79.90 ±0.18	83.55 ±0.33	84.38 ±0.22

(4) Greedy InfoMax for Self-Supervised Representation Learning(GIM)

Sindy Lowe et al. University of Amsterdam

Highlights

1. we propose a novel deep learning method for local self-supervised representation learning that does not require labels nor end-to-end backpropagation but exploits the natural order in data instead.
2. Inspired by the observation that biological neural networks appear to learn without backpropagating a global error signal, we split a deep neural network into a stack of gradient-isolated modules.
3. Each module is trained to maximize the mutual information between its consecutive outputs using the InfoNCE bound from Oord et al.(2018).

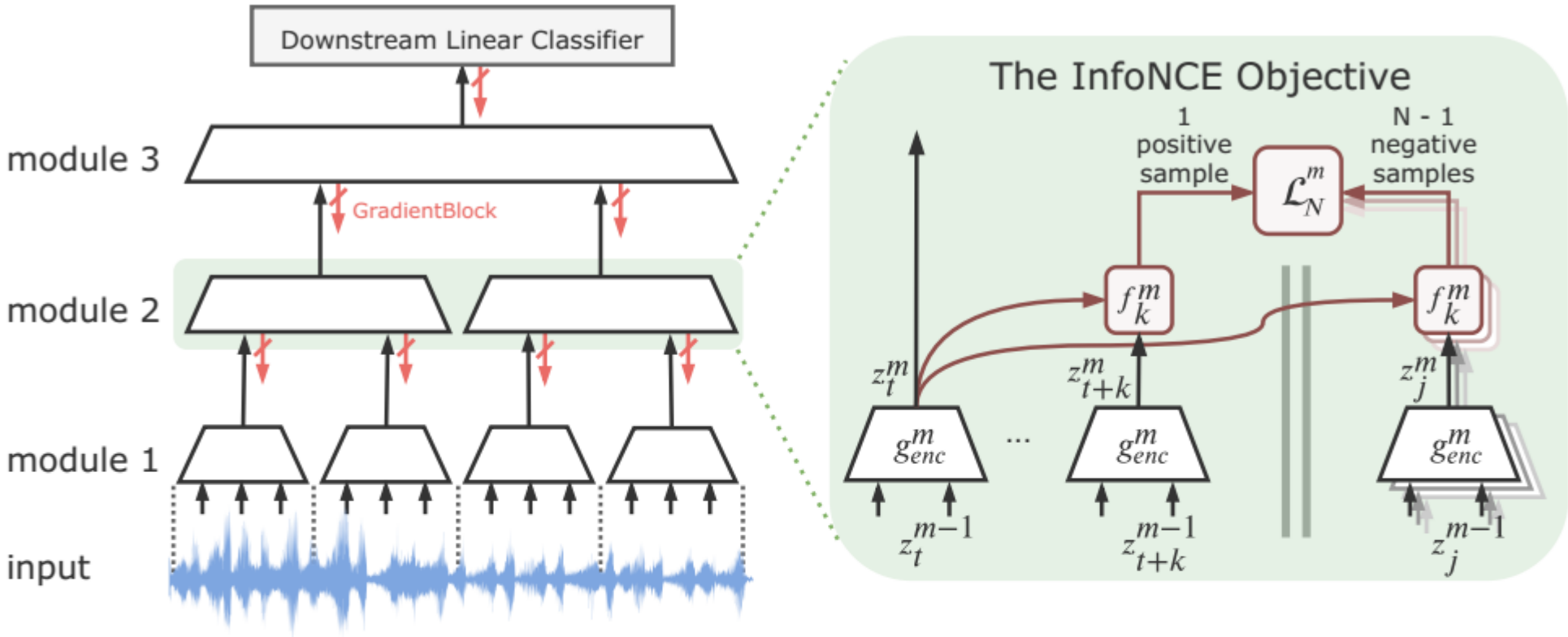


Figure 1. The Greedy InfoMax Learning Approach. **(Left)** For the self-supervised learning of representations, we stack a number of modules through which the input is forward-propagated in the usual way, but gradients do not propagate backwards. Instead, every module is trained greedily using a local loss. **(Right)** Every encoding module maps its inputs z_t^{m-1} at time-step t to $g_{enc}^m(\text{GradientBlock}(z_t^{m-1})) = z_t^m$, which is used as the input for the following module. The InfoNCE objective is used for its greedy optimization. This loss is calculated by contrasting the predictions of a module for its future representations z_{t+k}^m against negative samples z_j^m, which enforces each module to maximally preserve the information of its inputs. We employ an additional autoregressive module g_{ar}, which is not depicted here.

Architecture & Comparison

Table 3. General outline of our architecture

Layer	Output Size (Sequence Length × Channels)	Parameters Kernel	Stride	Padding
Input	20480×1			
Conv1	$4095^2 \times 512$	10	5	2
Conv2	$1023^2 \times 512$	8	4	2
Conv3	$512^2 \times 512$	4	2	2
Conv4	$257^2 \times 512$	4	2	2
Conv5	128×512	1	2	1
GRU	128×256	-	-	-

[2]For applying the InfoNCE objective on these layers, we randomly sample a time-window of size 128 to decrease the dimensionality.

Table 1. Results for classifying speaker identity and phone labels in the LibriSpeech dataset. All models use the same audio input sizes and the same architecture. GIM creates representations that are useful for audio classification tasks despite its greedy training and lack of a global objective.

Method	Phone Classification Accuracy	Speaker Classification Accuracy
MFCC features	39.7%	17.6%
Randomly initialized	27.6%	1.9%
Supervised	74.6%	98.5%
Greedy Supervised	71.1%	84.5%
CPC (Oord et al., 2018)[a]	64.6%	97.4%
Greedy InfoMax (GIM)	60.0%	97.5%

[a]In our reimplementation, we achieved 62.2% for the phone and 98.8% for the speaker classification task.

(5)Layer-Wise Contrastive Unsupervised Representation Learning

Stephen Zhao, 2019

- In this work, we focus on the set of unsupervised learning approaches that Arora et al. (2019) call “contrastive unsupervised representation learning”.
- We use a layer-wise adaptation, starting within the context of images.
- We train feature representations of semantically similar images (i.e. nearby patches within the same image) to be closer than that of unrelated images (e.g. random patches), to learn convolutional neural network filters from unlabeled data.

Figure 3: Performance on CIFAR-10. The learning rate is 0.001 for the first two-thirds of the training, and is 0.0001 afterwards. Results are averaged over 3 independent runs for each line (keeping transferred filters constant across runs, but allowing for fine-tuning).

(6) LoCo: Local Contrastive Representation Learning

Yuwen Xiong et al.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

- In this work, we discover that by overlapping local blocks stacking on top of each other, we effectively increase the decoder depth and allow upper blocks to implicitly send feedbacks to lower blocks.
- This simple design closes the performance gap between local learning and end-to-end contrastive learning algorithms for the first time.
- Aside from standard ImageNet experiments, we also show results on complex downstream tasks such as object detection and instance segmentation directly using readout features

LoCo Architecture

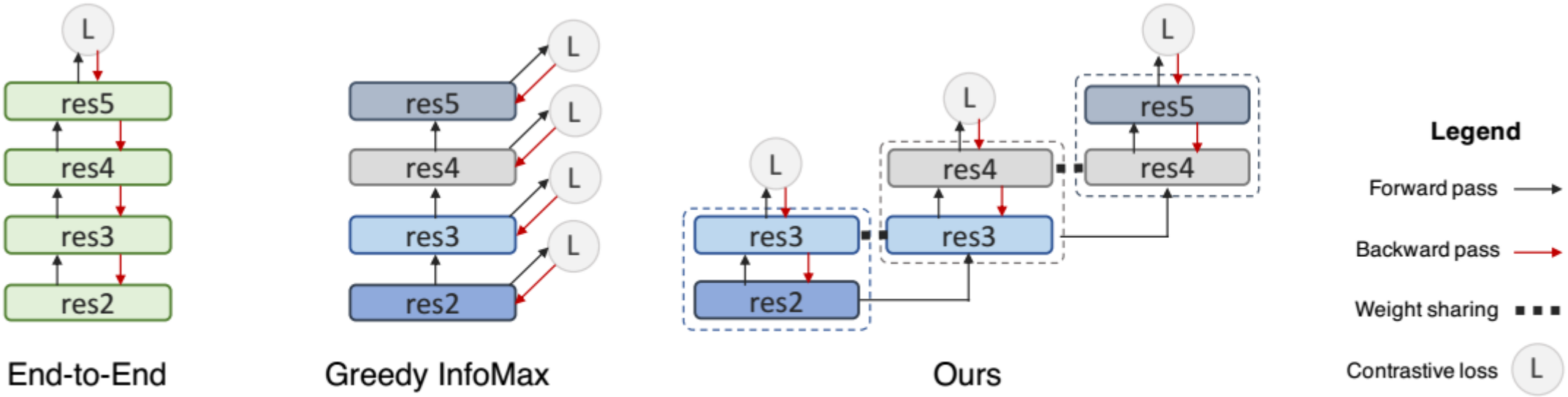


Figure 1: Comparison between End-to-End, Greedy InfoMax (GIM) and LoCo

In this paper, we used SimCLR as our main baseline, since it has superior performance on ImageNet, and we apply the changes proposed in GIM(Greedy Infomax) on top of SimCLR as our local learning baseline.
In our experiments, we find that simply applying GIM on SimCLR results in a significant loss in performance and in the next section we will explain our techniques to bridge the performance gap.

Comparison tables

Greedy InfoMax (GIM)
SimCLR: T. Chen... A simple framework for contrastive learning of visual representations, 2020.

Method	Architecture	Acc.	Local
Local Agg. [62]	ResNet-50	60.2	
MoCo [22]	ResNet-50	60.6	
PIRL [45]	ResNet-50	63.6	
CPC v2 [56]	ResNet-50	63.8	
SimCLR* [11]	ResNet-50	69.3	
SimCLR [11]	ResNet-50	**69.8**	
GIM [39]	ResNet-50	64.7	✓
LoCo (Ours)	ResNet-50	69.5	✓
SimCLR [11]	ShuffleNet v2-50	69.1	
GIM [39]	ShuffleNet v2-50	63.5	✓
LoCo (Ours)	ShuffleNet v2-50	**69.3**	✓

Table 1: ImageNet accuracies of linear classifiers trained on representations learned with different unsupervised methods, SimCLR* is the result from the SimCLR paper with 1000 training epochs.

Method	Arch	COCO		Cityscapes	
		AP^{bb}	AP	AP^{bb}	AP
Supervised	R-50	33.9	31.3	33.2	27.1
Backbone weights with 100 Epochs					
SimCLR	R-50	32.2	29.9	33.2	28.6
GIM	R-50	27.7 (-4.5)	25.7 (-4.2)	30.0 (-3.2)	24.6 (-4.0)
Ours	R-50	32.6 (+0.4)	30.1 (+0.2)	33.2 (+0.0)	28.4 (-0.2)
SimCLR	Sh-50	32.5	30.1	33.3	28.0
GIM	Sh-50	27.3 (-5.2)	25.4 (-4.7)	29.1 (-4.2)	23.9 (-4.1)
Ours	Sh-50	31.8 (-0.7)	29.4 (-0.7)	33.1 (-0.2)	27.7 (-0.3)
Backbone weights with 800 Epochs					
SimCLR	R-50	34.8	32.2	34.8	30.1
GIM	R-50	29.3 (-5.5)	27.0 (-5.2)	30.7 (-4.1)	26.0 (-4.1)
Ours	R-50	34.5 (-0.3)	32.0 (-0.2)	34.2 (-0.6)	29.5 (-0.6)
SimCLR	Sh-50	33.4	30.9	33.9	28.7
GIM	Sh-50	28.9 (-4.5)	26.9 (-4.0)	29.6 (-4.3)	23.9 (-4.8)
Ours	Sh-50	33.6 (+0.2)	31.2 (+0.3)	33.0 (-0.9)	28.1 (-0.6)

Table 2: Mask R-CNN results on COCO and Cityscapes. Backbone networks are frozen. "R-50" denotes ResNet-50 and "Sh-50" denotes ShuffleNet v2-50.

Average Precision (AP)

Average Precision bounding box (APbb)

(7) Progressive Stage-wise Learning for Unsupervised Feature Representation Enhancement

Zefan Li et al., Shanghai Jiao Tong University

In this work, we explore new dimensions of unsupervised learning by proposing the **P**rogressive **S**tage-wise **L**earning (**PSL**) framework.

For a given unsupervised task, we design multilevel tasks and define different learning stages for the deep *network.* *(Each new task has overlap with the former one)*

Early learning stages are forced to focus on low-level tasks while late stages are guided to extract deeper information through harder tasks.

We discover that by progressive stage-wise learning, unsupervised feature representation can be effectively enhanced.

Our extensive experiments show that PSL consistently improves results for the leading unsupervised learning methods.

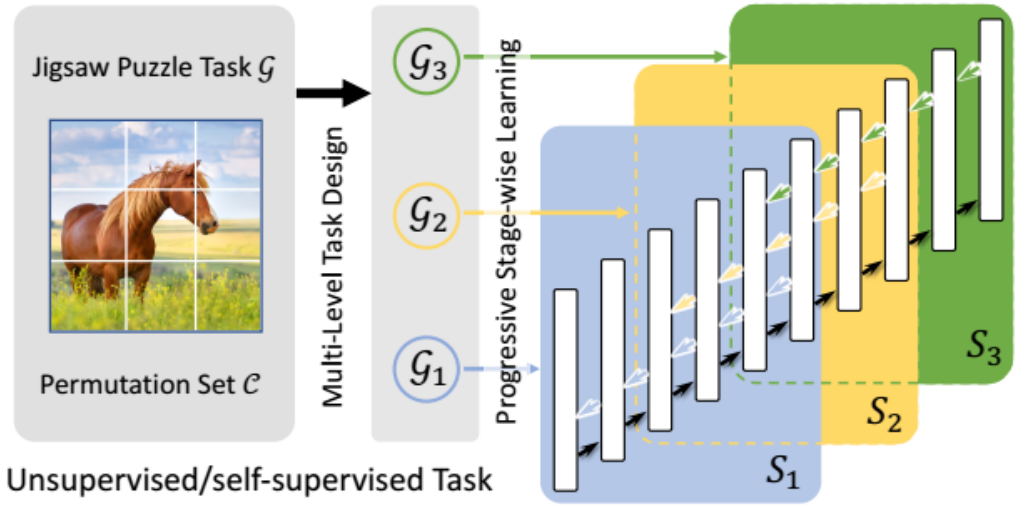


Figure 1. We present the framework of the proposed Progressive Stage-wise Learning (**PSL**) algorithm, aiming for improving unsupervised/self-supervised task. We take the jigsaw puzzle task $\mathcal{G}$ for example. We first do multi-level task partition $\mathcal{G} \rightarrow \{\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3\}$ with an increased task complexity and perform progressive stage-wise training for different learning stages of the network. The black arrow denotes forward pass while colored arrow represents the backward pass of each learning stage (i.e., S_1, S_2, and S_3).

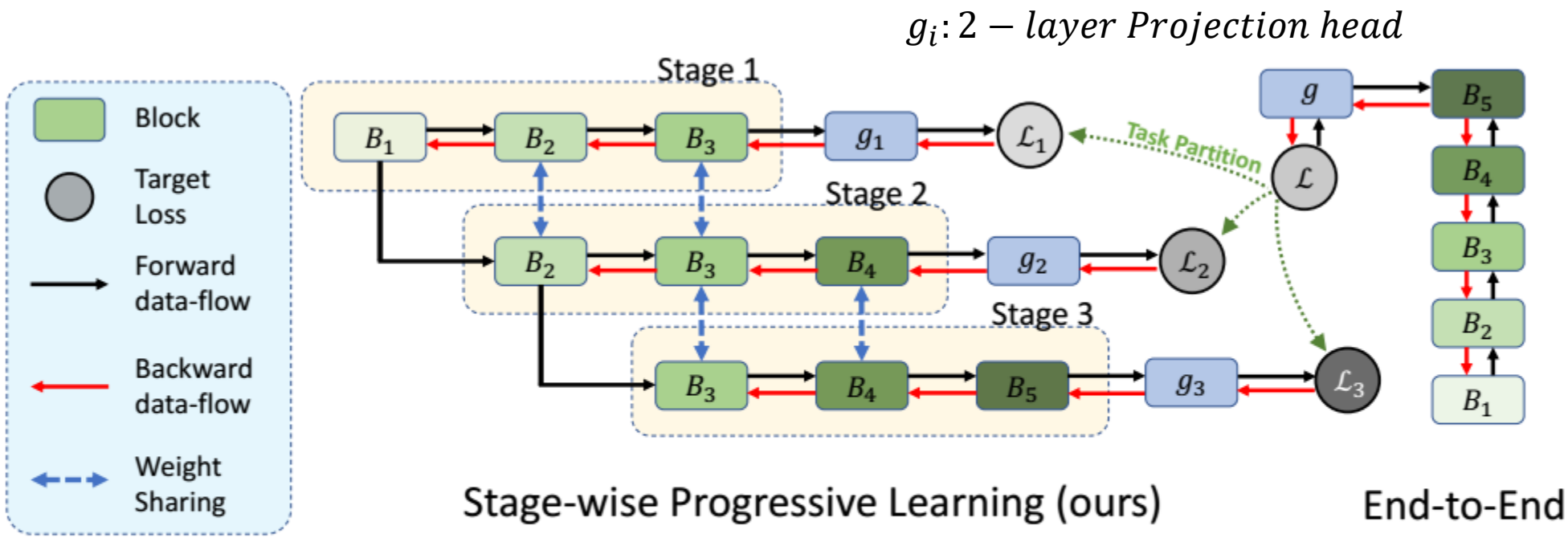


Figure 2. We present the detail of the proposed Stage-wise Progressive Learning framework. In the right is the end-to-end learning scheme while we present PSL in the middle. g and $\{g_i\}_{i=1}^{3}$ are projection heads, mapping the intermediate representation to the target feature space. After the training is completed, we throw away the projection heads and use the backbone network for downstream tasks.

Comparison Tables

Task	1% labels	10% labels
Supervised	48.4	80.4
Jigsaw [38]	45.4	79.6
Jigsaw+PSL	**48.7**	**83.5**
Rotation [21]	52.1	82.5
Rotation+PSL	**54.8**	**83.7**

Table 7. **Semi-supervised learning on ImageNet.** We use ResNet-50 as our backbone networks and report single-crop top-5 accuracy on the ImageNet validation set. All models are self-supervised trained on ImageNet and finetuned on 1% and 10% of the ImageNet training data, following [46, 37]. The supervised results are presented for reference.

Method	Arch	# Param(M)	Top 1
Colorization [49]	R50	24	39.6
BigBiGAN [15]	R50	24	56.6
LA [51]	R50	24	58.8
NPID++ [37]	R50	24	59.0
MoCo [26]	R50	24	60.6
PIRL [37]	R50	24	63.6
CPC v2 [28]	R50	24	63.8
PCL [34]	R50	24	65.9
SwAV [9]	R50	24	75.3
Jigsaw [38]	R50	24	45.7
Jigsaw +PSL	R50	24	**50.9**
Rotation [21]	Rv50w4×	86	47.3
Rotation*	Rv50	24	48.6
Rotation+PSL	Rv50	24	**53.3**
SimCLR [10]	R50	24	61.9
SimCLR+PSL	R50	24	**64.3**
MoCov2 [11]	R50	24	67.5
MoCov2+PSL	R50	24	**68.1**

Table 4. ImageNet accuracy of linear classifiers trained on self-supervised learned representations. All are reported as unsupervised pre-training on ImageNet, followed by supervised linear classification and evaluated on the ImageNet validation set. Note that Rotation [21] uses $\mathcal{R}_2$ as the transformation set while Rotation* uses $\mathcal{R}_3$ as the transformation set. SimCLR results are obtained by 200 training epochs with batchsize 256.

End of Chapter 3