1 Convolution Neural Network Architectures

The main tendencies in the modern state of CNN Architectures are:

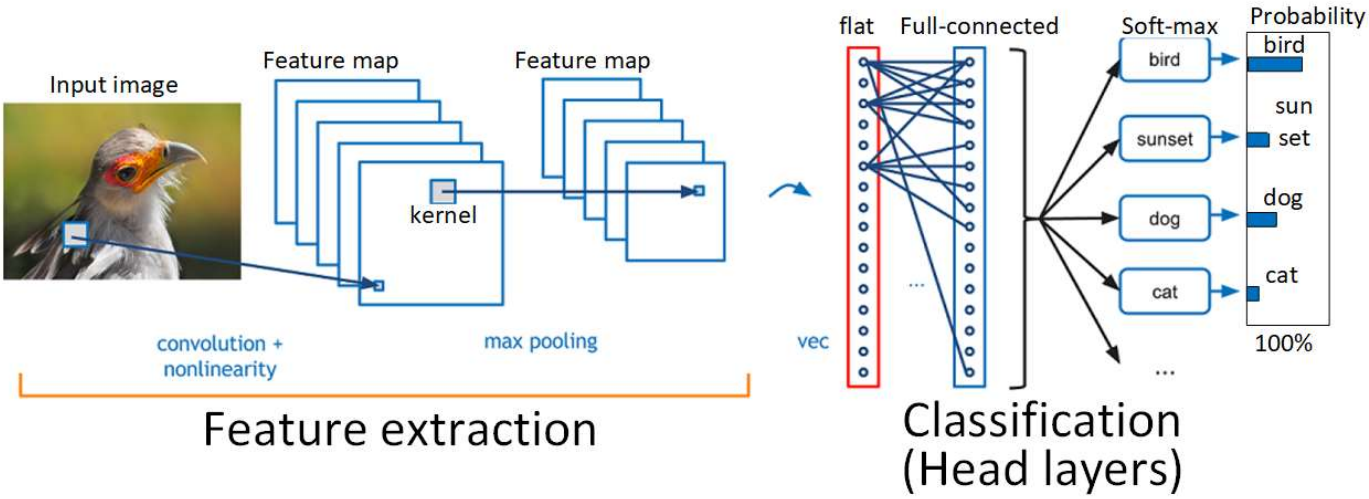
- **Optimization of the backbone structure** (search for the best for common datasets such as ImageNet and COCO).
- **Optimization of head part for different tasks** (i.e. classification, localization, object detection, semantic segmentation, instance segmentation, panoptic segmentation, style-transfer, GAN and so on).
- Optimization of the computational cost (or costless architectures) for small devices like CPU or Raspberry Pi (**HardWare Specific Optimization**).
- Using of new layer structures as attention mechanisms and transformer layers in the backbone instead of convolutions.
- Use of Atuo search of architectures (Like an AutoML, NAS, NetAdpat, and e.t.c. - as optimization of model as its self (wider,deeper and e.t.c., as hyperparameter optimization.).

1.1 Backbones of CNN.

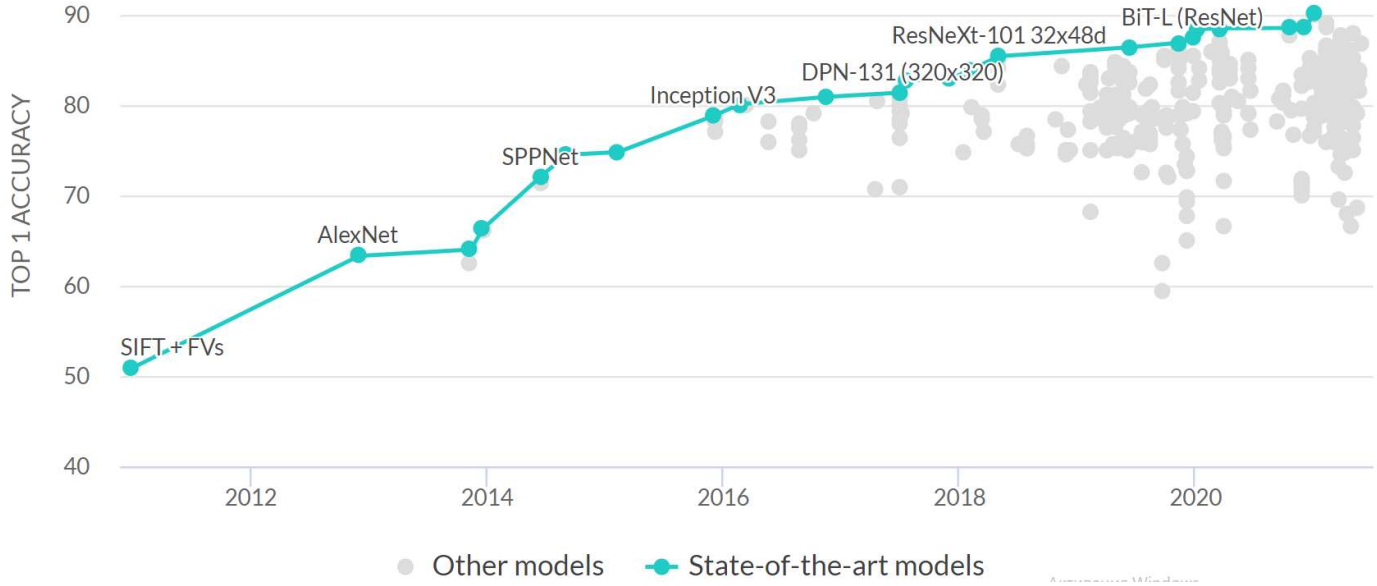
The aim of the backbone consists in feature extractor.

After backbone you may several head layers for specific tasks.

In most cases backbones are tested for the classification task on the large datasets like ImageNet.



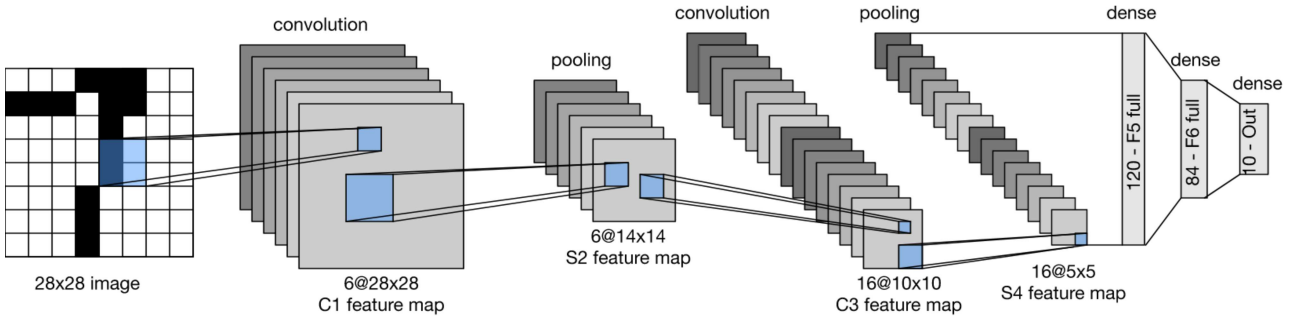
Top-1 accuracy on the imageNet (taken here https://paperswithcode.com/sota/image-classification-on-imagenet (https://paperswithcode.com/sota/image-classification-on-imagenet)).

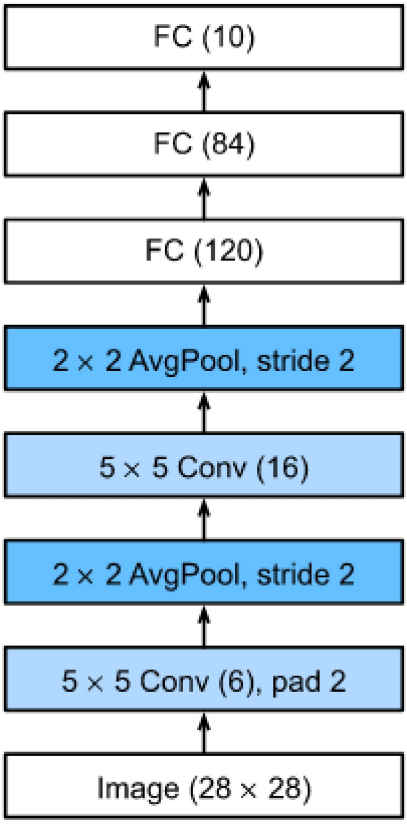


1.1.1 LeNet, AlexNet, ZFNet, VGG

LeNet (1998)

- The first modern idea of CNN network was LeCun network (Lenet, LeNet-5) (1998 year).
- The network consists in 5 layers, 2 convolution and 3 full-connected.
- The CNN LeNet interleave convolutions, nonlinearities, and (often) pooling operations.
- LeNet was arguably the first successful deployment of such a network.





AlexNet(2012)

AlexNet is a classic convolutional neural network architecture. It consists of convolutions, max pooling and dense layers as the basic building blocks. Grouped convolutions are used in order to fit the model across two GPUs.

Main specificity in comparison with LeNet:

- ReLu (speed up in 6x times).
- Using of Max-pooling instead of average pooling.
- Using of Data augmentation
- Using Dropout 0.5 (but computational time increased in 2 times).

    Dropout after first and second layers

- Batch size = 128
- Optimizer SGD with Momentum 0.9.
- 60M parameters.
- 1 week with 2 GPU (50x to CPU).
- 7 hidden layers.
- AlexNet's first layer, the convolution window shape is 11×11 .

    Since most images in ImageNet are more than ten times higher and wider than the MNIST images, objects in ImageNet data tend to occupy more pixels. Consequently, a larger convolution window is needed to capture the object.

- The convolution window shape in the second layer is reduced to 5×5 , followed by 3×3 .
- Maximum pooling layers with a window 3×3 and a stride of 2.

- After the last convolutional layer there are two fully-connected layers with 4096 outputs.

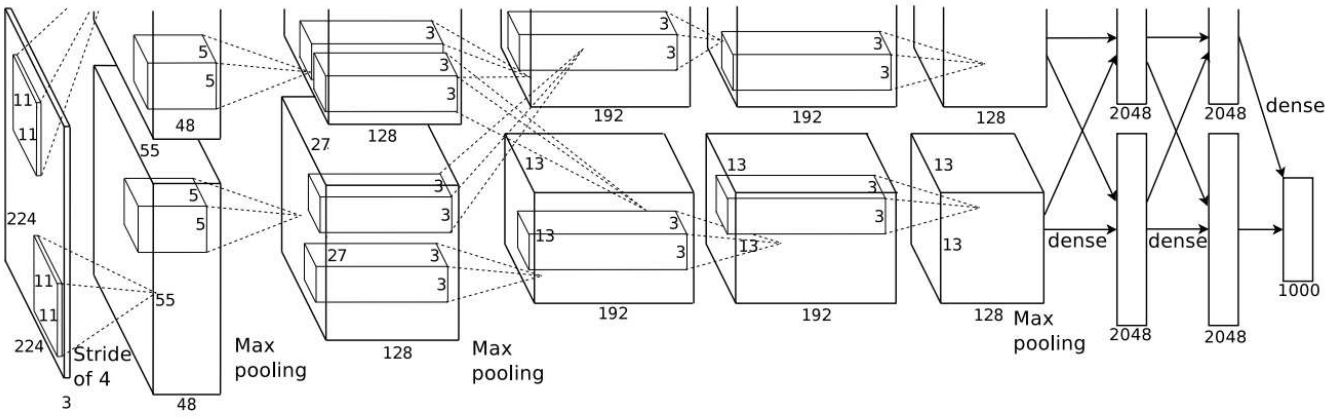
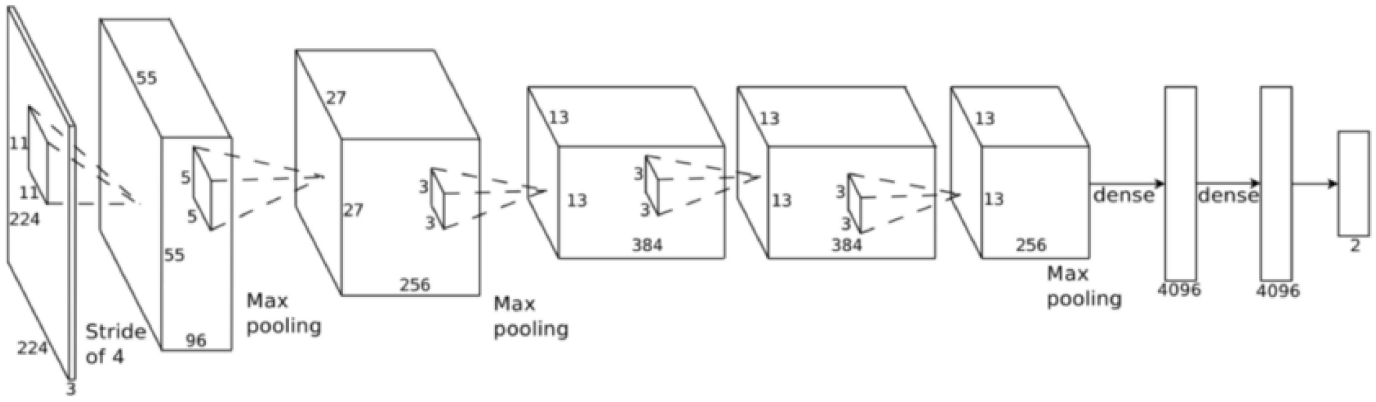
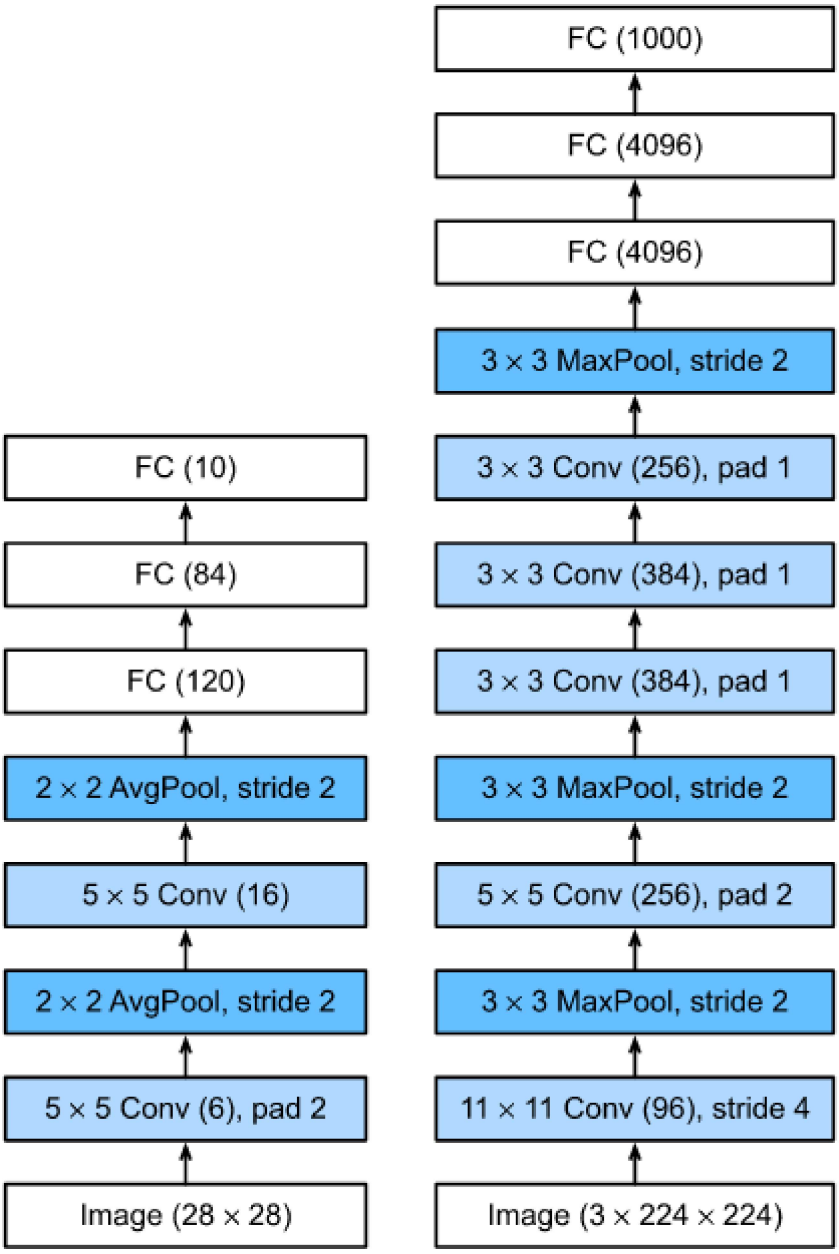


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

original image

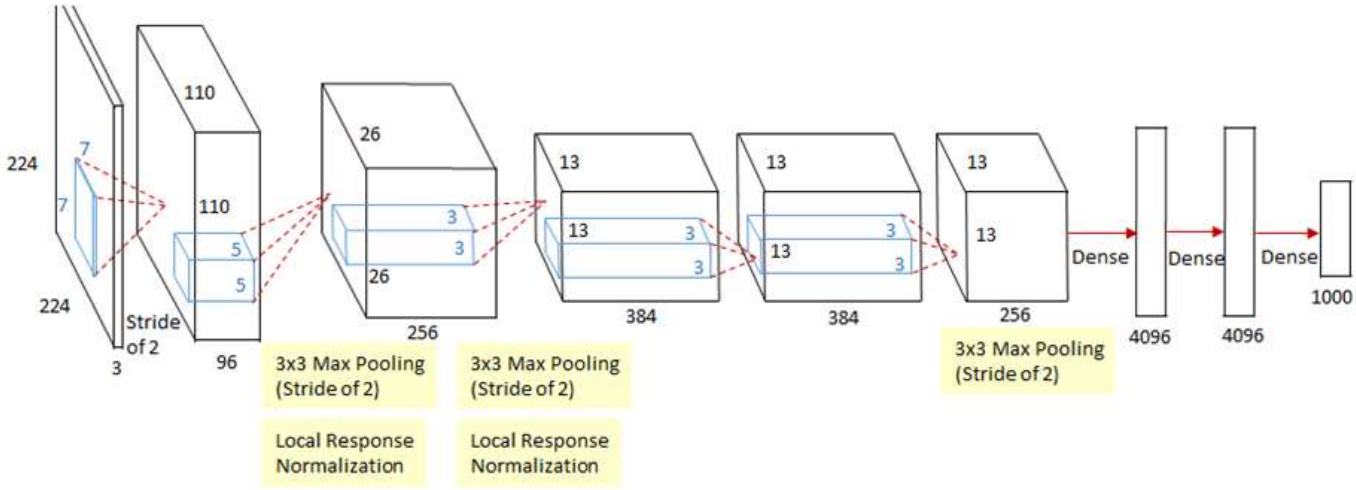




From LeNet (left) to AlexNet (right).

The other similar to ALexNet idea is the

ZFNet (2013). However it does not use group convolutions



VGG Net, 2014

The main idea under this architecture is using blocks of convolutions with non-linearities (activation functions) instead of only one convolution (so-called cascade convolutions). The idea was proposed by **Visual Geometry Group (VGG)** .

> It is important to note that one convolution 5×5 with stride 1 can be replaced on two convolution 3×3 without loose of receptive filed.
>
> However, in this case the full number of parameters reduced to $2 \times 3 \times 3 = 18$ versus $1 \times 5 \times 5 = 25$
>
> And the number of non-linearities can also be increased. Also 7×7 conv can be replaced with 3 3×3 convolutions.

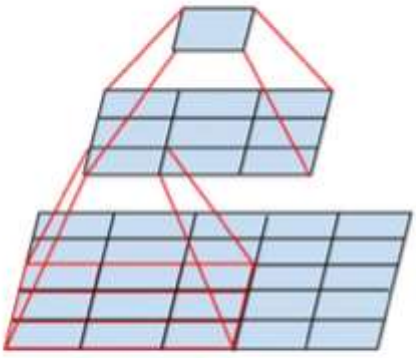
It was originally proposed several variants **VGG-11, VGG-13, VGG-16** and **VGG-19**, but only VGG-16 is popular up-to-date.

Note

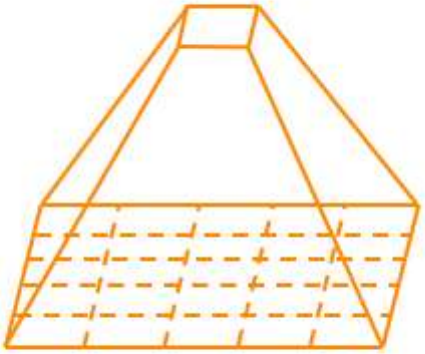
- The pooling is performed only after block of convolutions.
- Each convolution need to be done with same padding (without decreasing of the output of size).
- All kernels size is 3×3.

The main drawbacks of VGG

- It is painfully slow to train.
- The network architecture weights themselves are quite large (concerning disk/bandwidth).
- too few number of layers - due to high number of parameters increasing of layers leads to high probability of the gradient vanishing.
- Huge amount of the parameters can also lead to the high probability of the over fitting.

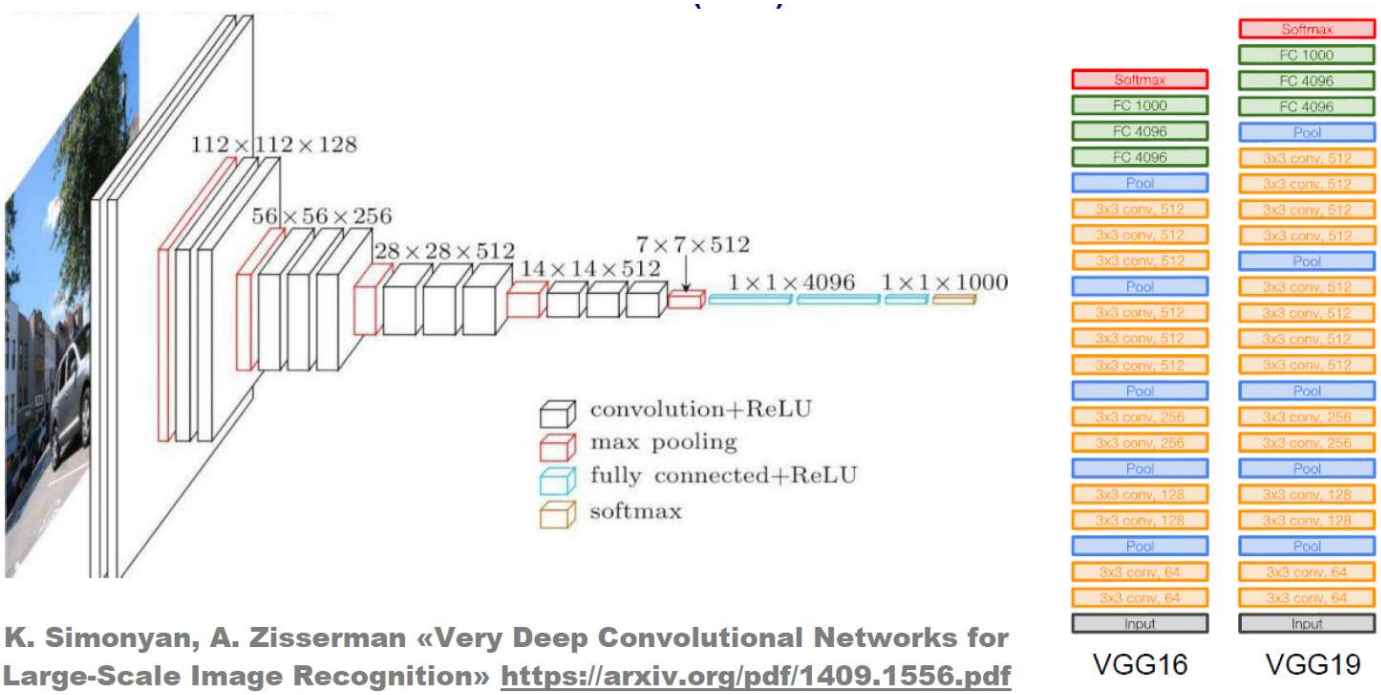


two successive
3x3 convolutions

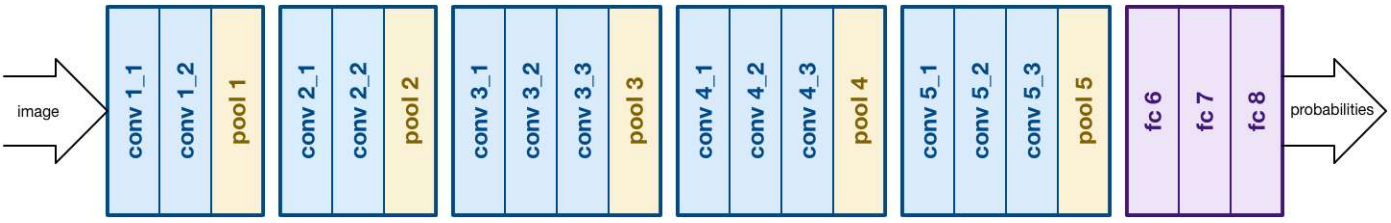


5x5 convolution

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



K. Simonyan, A. Zisserman «Very Deep Convolutional Networks for Large-Scale Image Recognition» https://arxiv.org/pdf/1409.1556.pdf



1.1.2 Network In Network and InceptionNet

Network In Net, NiN 2013

LeNet and AlexNet share a common design pattern: extract features exploiting spatial structure via a sequence of convolution and pooling layers and then post-process the representations via fully-connected layers.

Beside the sequential increasing of the layers we can increase the width of each layer.

The original idea behind Network In Net is to apply a fully-connected layer at

each pixel location (for each height and width).

However, it was rethinked with using of a 1×1 convolutional layer.

Note

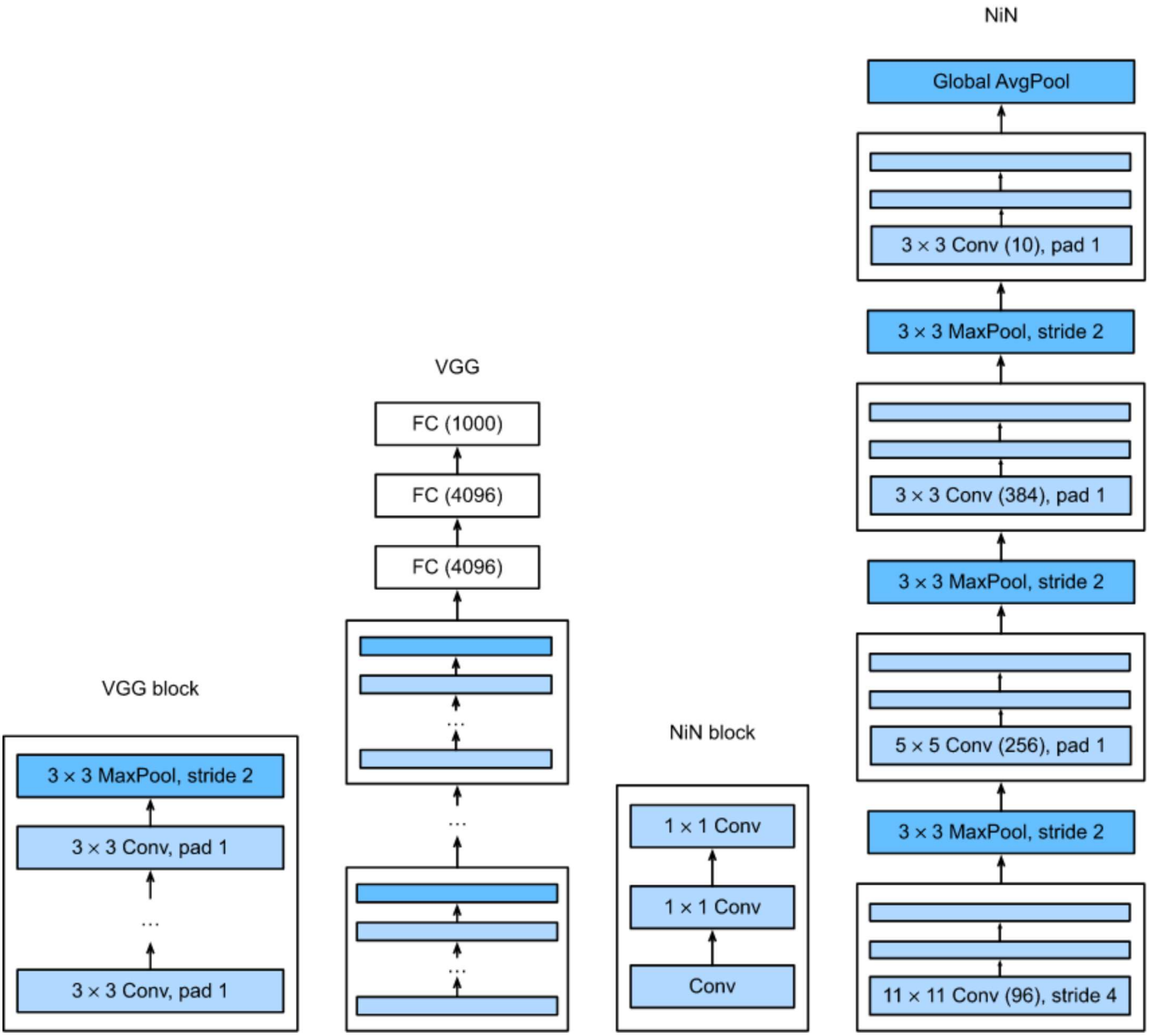
- The original NiN network was proposed shortly after AlexNet and clearly draws some inspiration.
- NiN uses convolutional layers with window shapes of 11×11 , 5×5 , and 3×3 , and the corresponding numbers of output channels are the same as in AlexNet. li> Each NiN block is followed by a maximum pooling layer with a stride of 2 and a window shape of 3×3 .
- NiN avoids fully-connected layers altogether, instead of using flatten (vectorization) here global average pooling is applied with output as vector of logits.

    Removing the fully-connected layers reduces overfitting. NiN has dramatically fewer parameters.
- Each NiN block allow for more per-pixel nonlinearity in comparison with AlexNet due to additional convolution layers.

Advantages of NiN is reducing the number of required model parameters.

However, in practice, this design sometimes requires increased model training

time.



Inception Net (GoogLeNet), 2014

The main drawback of the VGG Net was high number of parameters with small number of layers - i.e. high probability of gradient vanishing and low receptive field

The one of the possible solution is to optimization of the each layer structure.

It also can be combined VGG idea with NiN idea wich are similar in essence.

The main idea here is how to select the best full receptive filed for each layer.

- It means the best result accuracy, and decreasing of the full number of parameters can be achieved by using a series of convolutions at different scales in one layer with 1×1 convolution for decrease the number of channels for keep full number of parameters small - **"inception cell"**.
- Beside the convolution in the same layer can also be used max-pooling with same stride to keep full dimension the same.

Note

- For additionally increase the receptive fileld convolutions with large spatial filters (such as 5x5 or 7x7) are beneficial, but the computation is disproportionately expensive. The researchers decide to not use filters above 5x5.
- The Inception block is equivalent to a subnetwork with four paths.

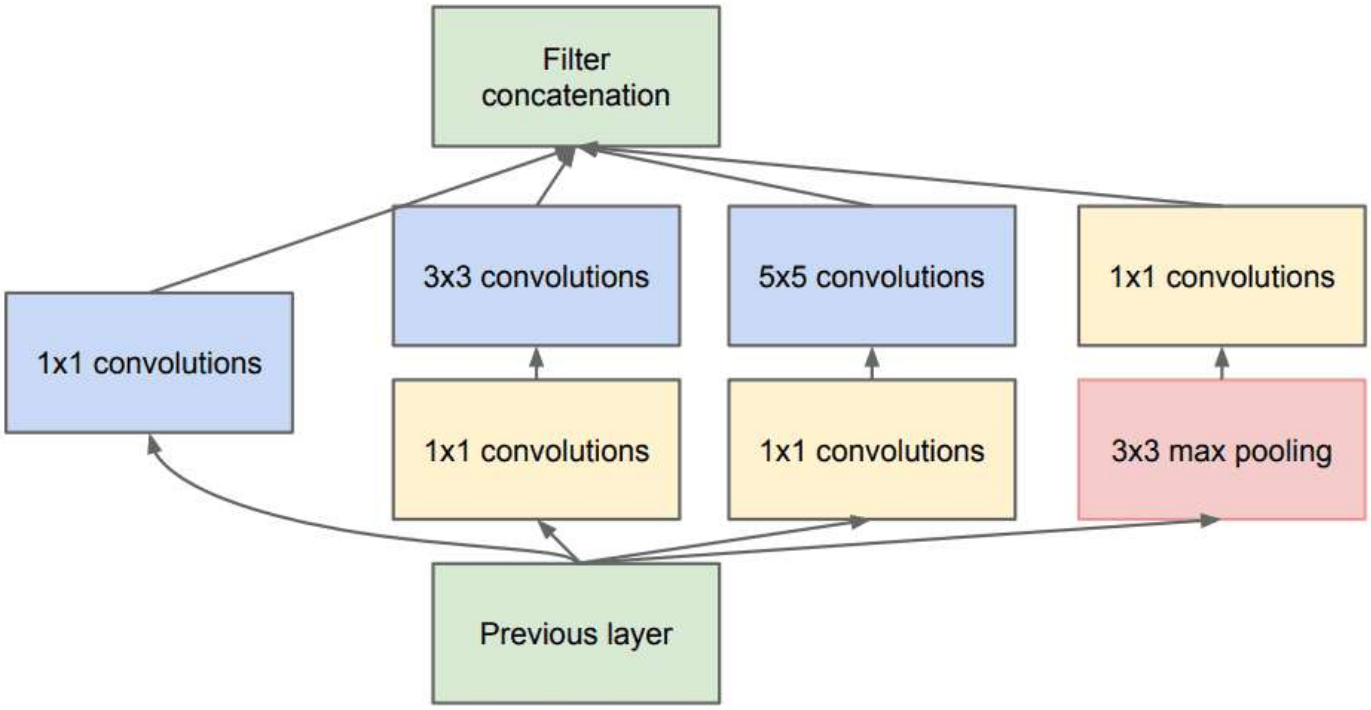
  In essence, the authors claim that they try to approximate a sparse convnet with normal dense layers.

  The authors believe that only a small number of neurons are

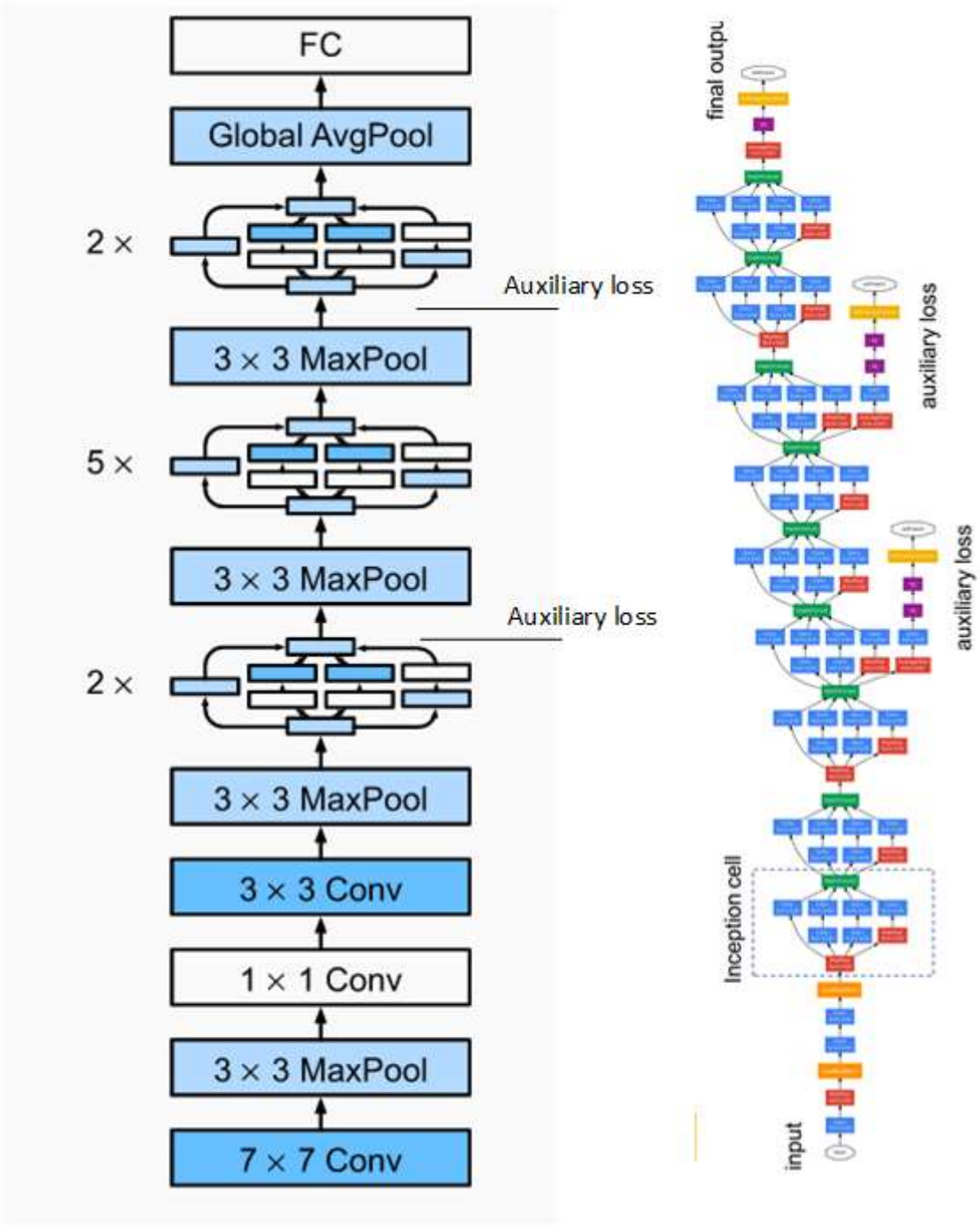
effective. This comes in line with the Hebbian principle: “Neurons that fire together, wire together”.

- The other idea under GoogLeNet is to using **auxiliary outputs** as the intermediate results which can be summed with final one as regularization of weights.
    - It was later discovered that the earliest auxiliary output had no discernible effect on the final quality of the network.
    - The addition of auxiliary outputs primarily benefited the end performance of the model, converging at a slightly better value than the same network architecture without an auxiliary branch.
    - In many modern implementations the auxiliary outputs are not applied instead the batch normalization is using.

Example of the inception block



The InceptionNet(GoogLeNet) V1 architecture consists of 9 inception modules stacked together, with max-pooling layers between (to halve the spatial dimensions). It consists of 22 layers (27 with the pooling layers). It uses global average pooling after the last inception module.



Inception V2, V3 (2015)

- Factorize 5x5 and 7x7 (in InceptionV3) convolutions to two and three 3x3 sequential convolutions respectively.

  This improves computational speed, the same as VGG.

- use spatially separable convolutions 1xn-nx1 (for instance, a 3x3 kernel is decomposed into two: a 1x3 and a 3x1 kernels, which are applied sequentially, also 5x1-1x5 and 7x1-1x7).
- Due to the reducing of each convolution parameters each inception modules became wider (more feature maps).

  Authors tried to distribute the computational budget in a balanced way between the depth and width of the network.
- Added batch normalization, use auxiliary outputs only in last layers.
- Use label smoothing:

  instead of one-hot-encoding

  $$p(y|x_i) = \begin{cases} 1 & \text{if } y = y_i \\ 0 & \text{otherwise} \end{cases}$$

  it could be introduce **label smoothing** as

  $$p'(y|x_i) = (1-\varepsilon)p(y|x_i) + \varepsilon u(y|x_i) = \begin{cases} 1-\varepsilon+\varepsilon u(y|x_i) & \text{if } y = y_i \\ \varepsilon u(y|x_i) & \text{otherwise} \end{cases},$$

  where $\varepsilon \in [0, 1]$, $u(y|x) = \frac{1}{K}$ or uniform distribution and $\sum_{y=1}^{K} p'(y|x_i) = 1$.

  Modified loss function will be

  $$L' = -\sum_{y=1}^{K} p'(y|x_i) \log q_\theta(y|x_i) = -\sum_{y=1}^{K} \left[(1-\varepsilon)p(y|x_i) + \varepsilon u(y|x_i)\right] l$$
  $$+ \varepsilon H_i(u, q_\theta)$$

  and $\log q_\theta(y|x_i) = \hat{y}_i$;

Note

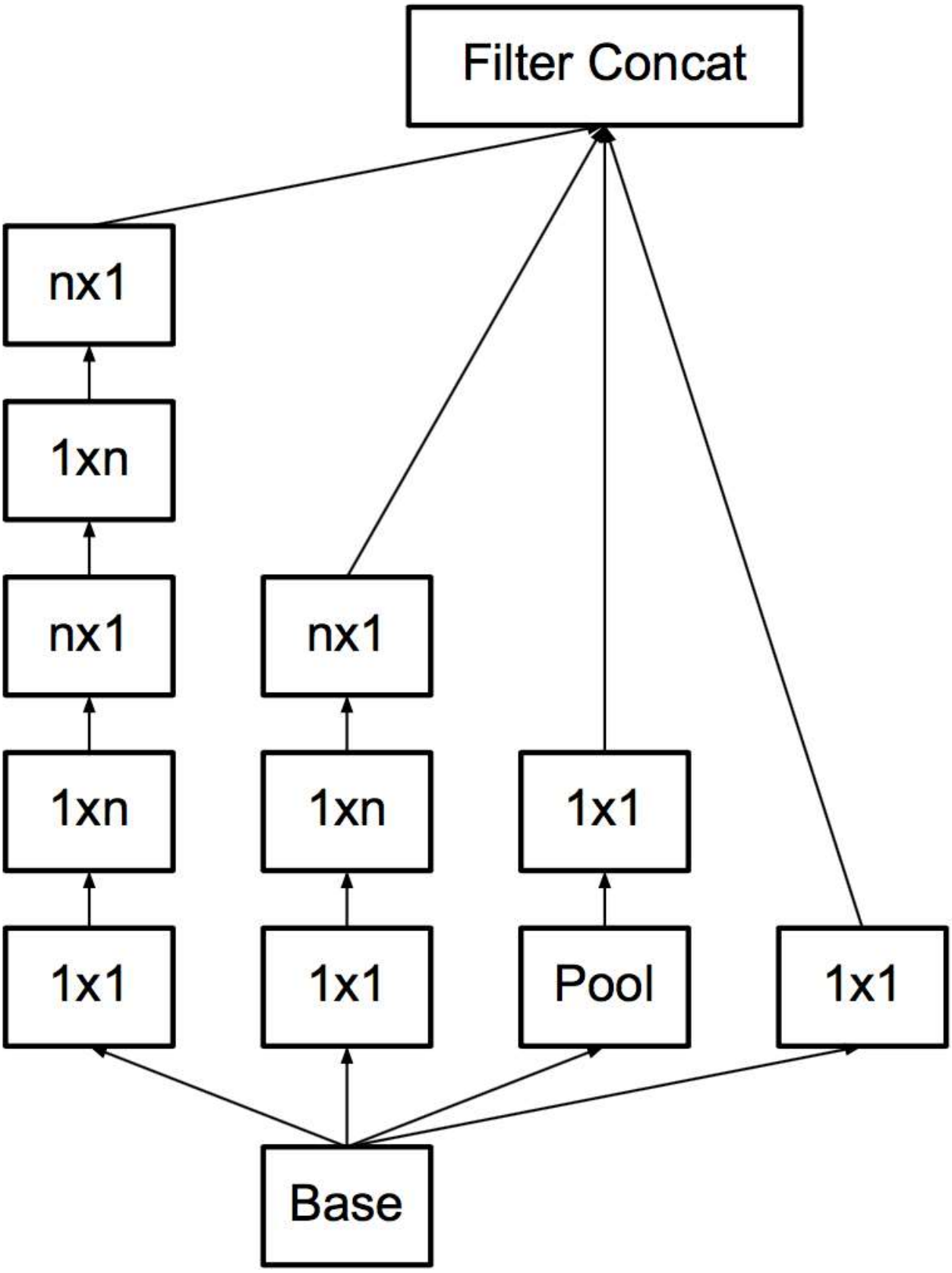
We could see that for each example in the training dataset, the loss contribution is a mixture of the cross entropy between the one-hot encoded distribution and the predicted distribution, and the cross entropy between the noise distribution and the predicted distribution. During training, if the model learns to predict the distribution confidently, will go close to zero, but will increase dramatically. Therefore, with label smoothing, we actually introduced a regularizer to prevent the model from predicting too confidently.

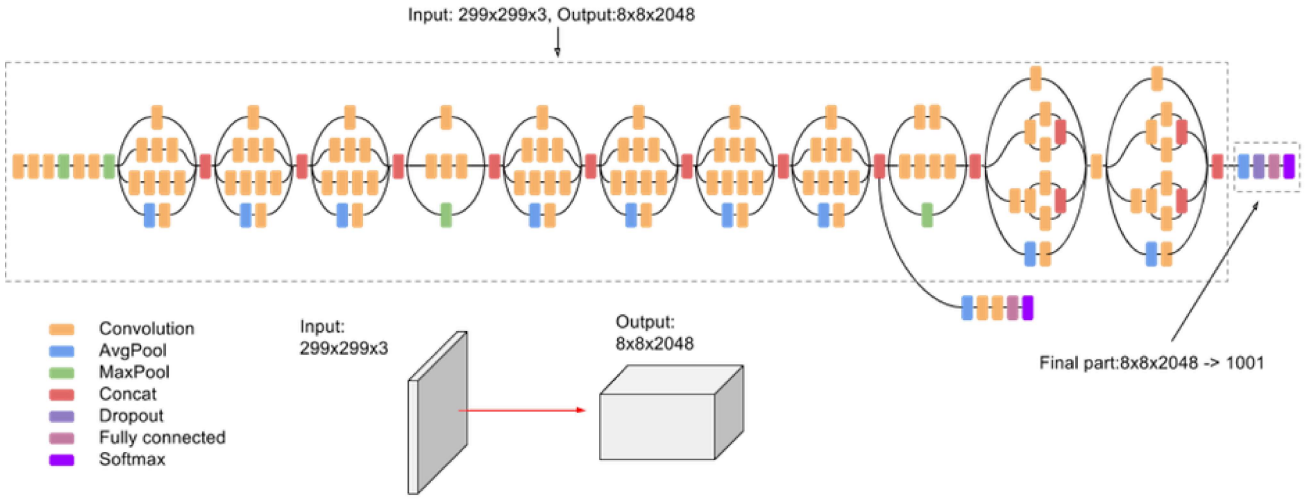
- **The main ideas behind inception 2,3**:
  - instead of only make layers deeper it is more effective to make it as wider as deeper (with keeping full number of parameters the same).
    - it is more simply to train wide layer then two deep layers, but by deep layers we increase receptive field.

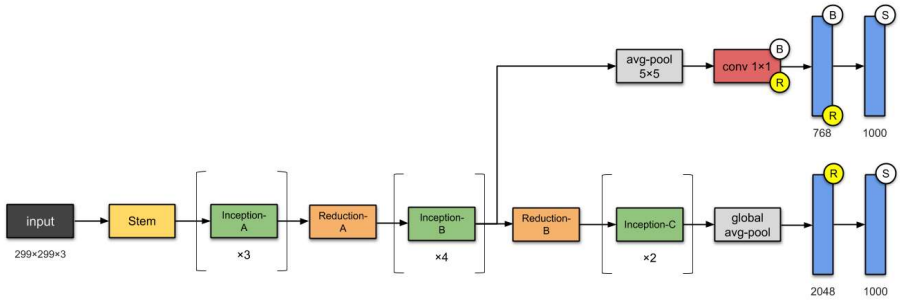
- It is better to make last layer more wide (then add more last layers) to train with more accuracy.

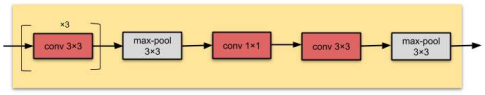
- it is bad to sharply change the number of parameters (especially in the beginning, **representational bottlenecks**), thus by make wide layers we can smoothly variate layer number of parameters.
- it could be not optimal to have pooling after layer, thus authors use pooling in parallel with convolution in the layer.

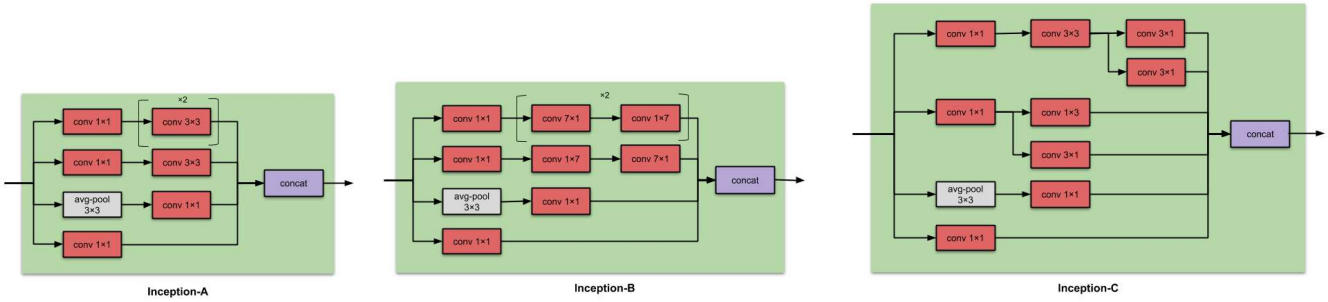
  pooling allow to rest features without noises, but reduce number of non-linearities to extract new features (work as bottleneck), thus it is better have as sequence $pool - conv - activation$, as $conv - activation - pool$ here they both implemented in parallel.

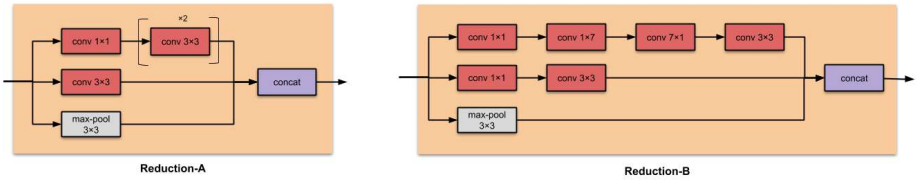

Filter Concat
nx1
1xn
nx1
1xn
1x1
nx1
1xn
1x1
Pool
1x1
Base











1.1.3 ResNet, DenseNet and other skip-connection variants

ResNet(2015).

The degradation problem - While expecting that the model will converge

with higher accuracy with increasing of number of layers (using batchnorm, L2 and e.t.c.) , on the practice, it cold lead to reducing precision (increasing of the variance) in comparison with their shallower counterpart.

In other words, In the limit, simply stacking more layers degrades the model's ultimate performance.

> It could be explained as by increasing the the layers we need to provide that during the training our new model need to include the previous result but not shift it. The last can be due to vanishing and other problems with weights updating.
>
> In other words for non-nested function classes, a larger function class does not always move closer to the "truth" function.

We can regularize the all model behavior if:

New layer will be copy of previous one (hard to provide it on practice).

- New layer will be identity ($f(x) = x$). However, if simply use identity layer, then accuracy will not increase.

The solution is to use combination of identity in parallel with conventional block i.e.

Residual Block.

- Residual function can be considered as a refinement step in which we learn how to adjust the input feature map for the output one.

  Here the also idea that every additional layer should more easily reduce to the identity function then the rest one.

  In other words if the layer is no layer work is need, then their weights need to be gradually reduced to zero and res only the identity part.

- The original resnet was vggnet adaptation.
- There was proposed a several variants ResNet-18,-34, -50, -101, -151.

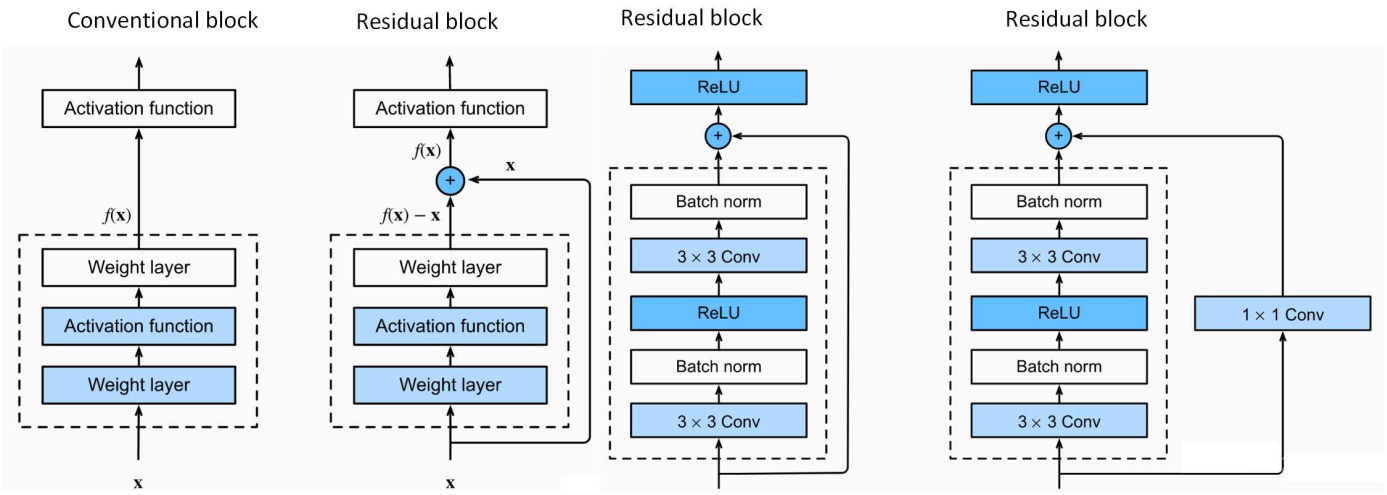
  Later it was also proposed the ResNet-1001, however it accuracy does not dramatically higher then for ResNet-151, thus it it not popular solution up-to-day.

- Each resnet block need to have the same channels and feature map size on the input and the output.

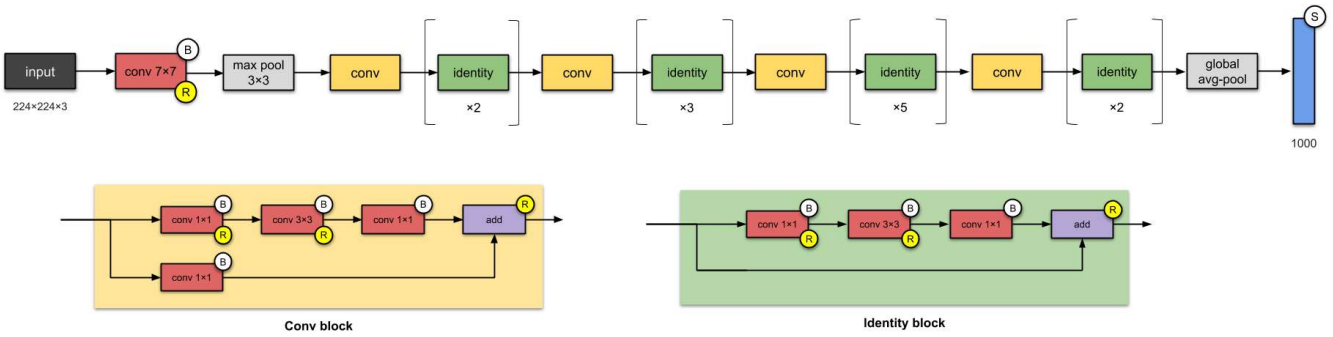
  In the identity part: either use corresponding structure of convolutions or padd rest channels with zeros, or use 1x1 conv .

- The other ideas under resnet are:

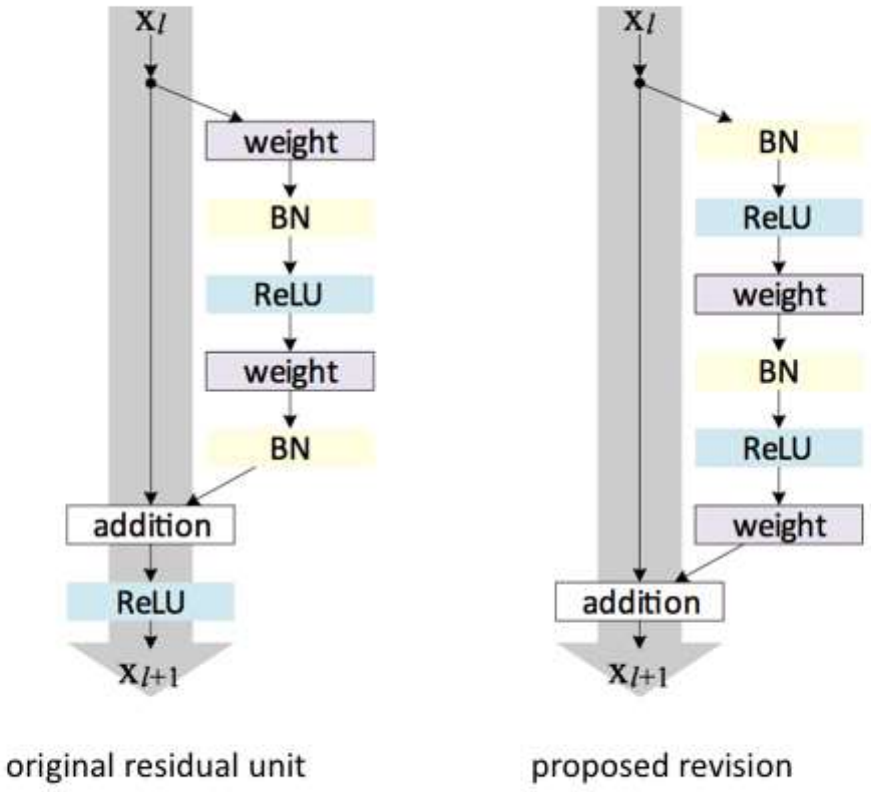
- when introducing n skip-residual blocks, the networks utilize 2^n different paths and therefore features can be learned with a large range of different receptive fields.
- During the back propagation identity work as negative feedback allowing to reduce or get around layers with zero-gradient (vanishing gradient).
- The idea is similar to the HighWayNet (2015) where it was proposed to use in parallel two connection with weights $(1 - a)$ and a.



ResNet 18



The latter modification of residual layer is the following

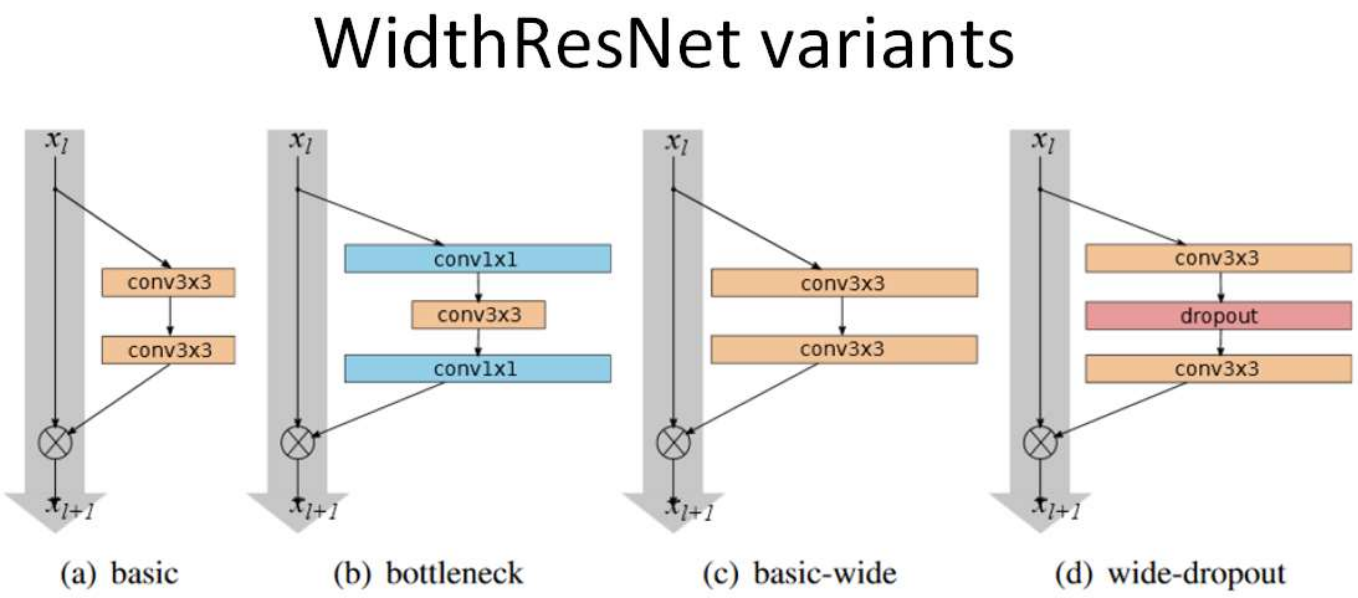


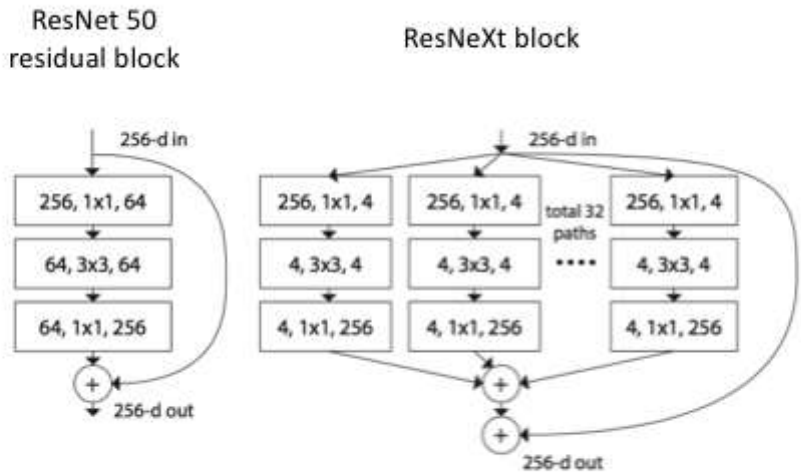
original residual unit proposed revision

The other resnet-like ideas (or architectures) are:

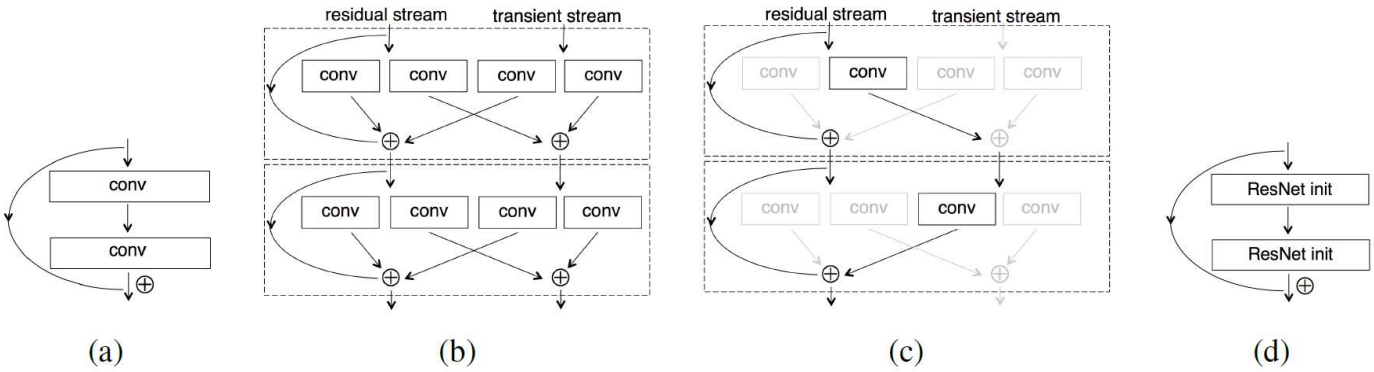
- **Wide ResNet** (with more wide layers).
- **Stochastic Net** randomly dropping entire ResBlocks during training (with rest only identity part) rest all parts (without dropping) in evaluation.
- **ResNet in ResNet** the idea is similar to network in network, but with using of a identity paths.

- **ResNeXt** (2017) replaces the standard residual block with one that leverages a "split-transform-merge" strategy (ie. branched paths within a cell) - like inception module.
  - The idea behind resNetXt is similar to group convolution, it was discovered that each group can learn different features. It is also was shown that increasing cardinality is more effective at benefiting model performance than increasing the width or depth of the network.
  - Each layer her proposed to have cardinality the same as expression power, by experiments authors proposed the best cardinality layer architectures by their opinion.

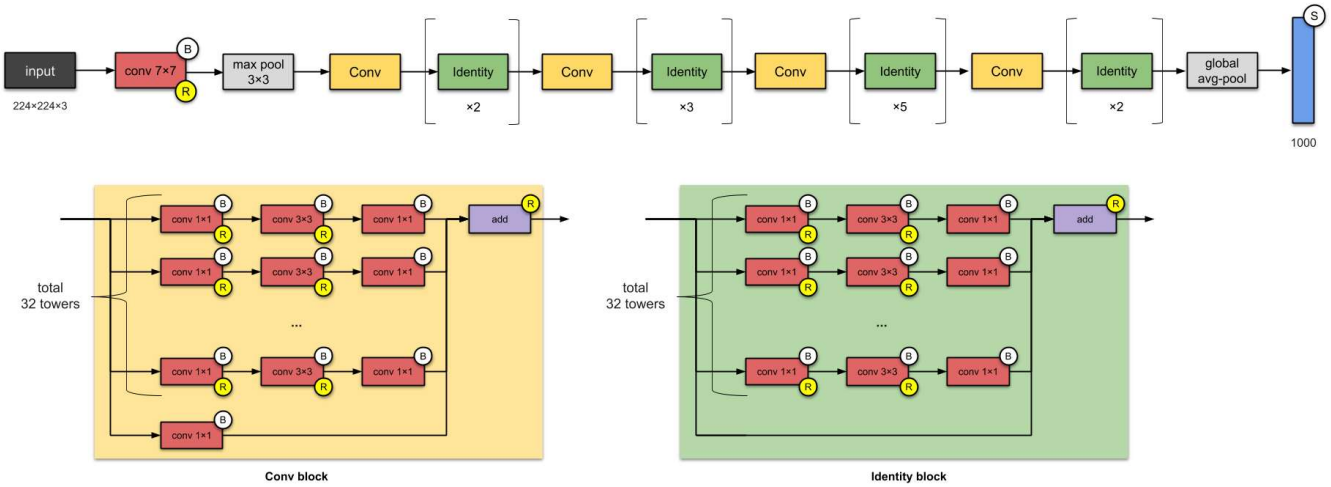




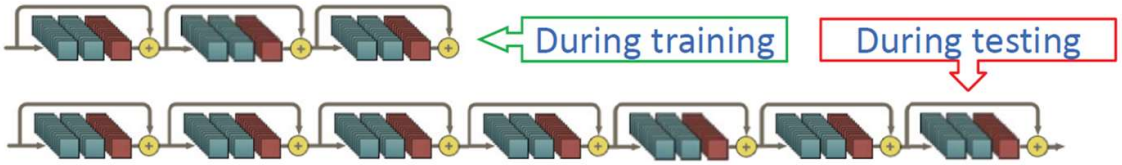
RiR

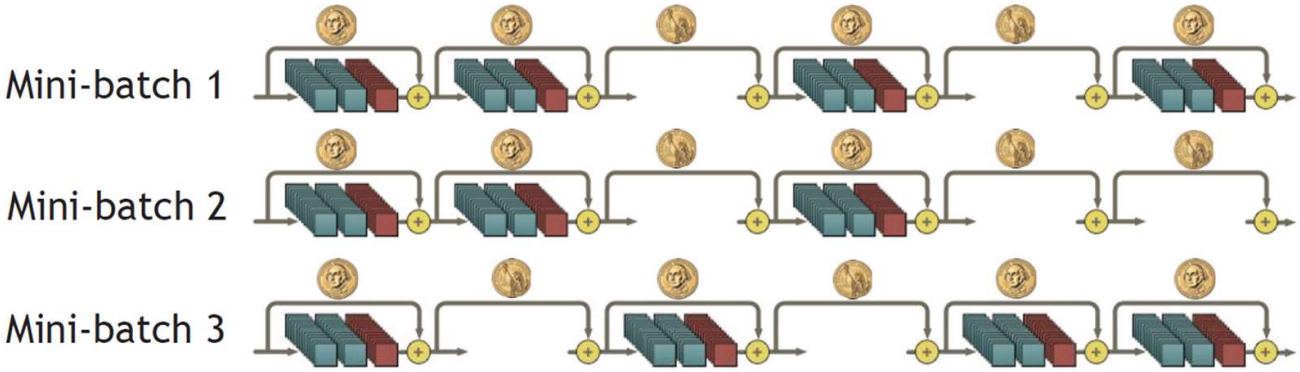


ReNetXt



Stochastic net

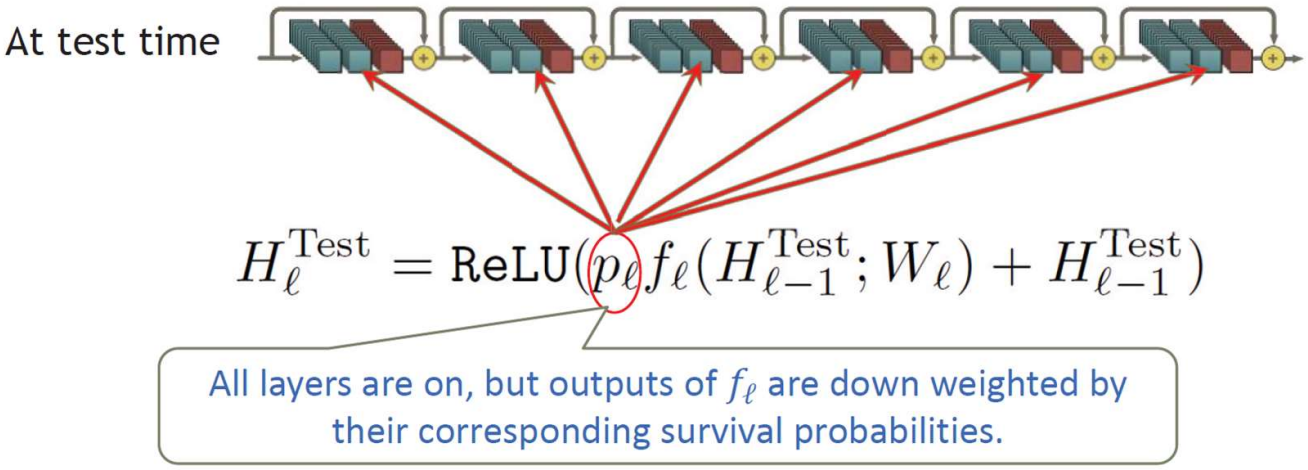




$$H_\ell = \texttt{ReLU}(b_\ell f_\ell(H_{\ell-1}) + \text{id}(H_{\ell-1}))$$

Bernoulli random variable

$b_\ell \sim \texttt{Bernoulli}(p_\ell)$ with $p_\ell = (1 - \frac{\ell}{L}) \times 1 + \frac{\ell}{L} \times p_L$

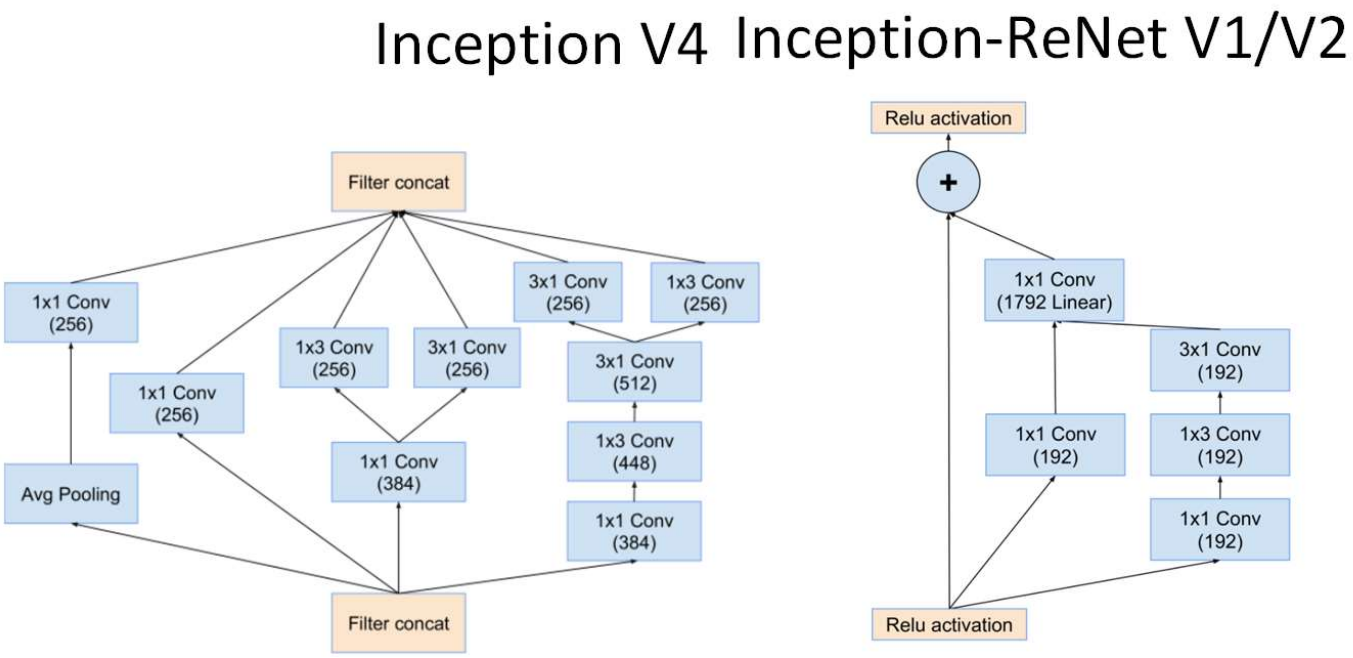


$$H_\ell^{\text{Test}} = \texttt{ReLU}(p_\ell f_\ell(H_{\ell-1}^{\text{Test}}; W_\ell) + H_{\ell-1}^{\text{Test}})$$

All layers are on, but outputs of f_ℓ are down weighted by their corresponding survival probabilities.

- **Inception V4, Inception-ReNet V1, Inception-ReNet V2 (2016)**

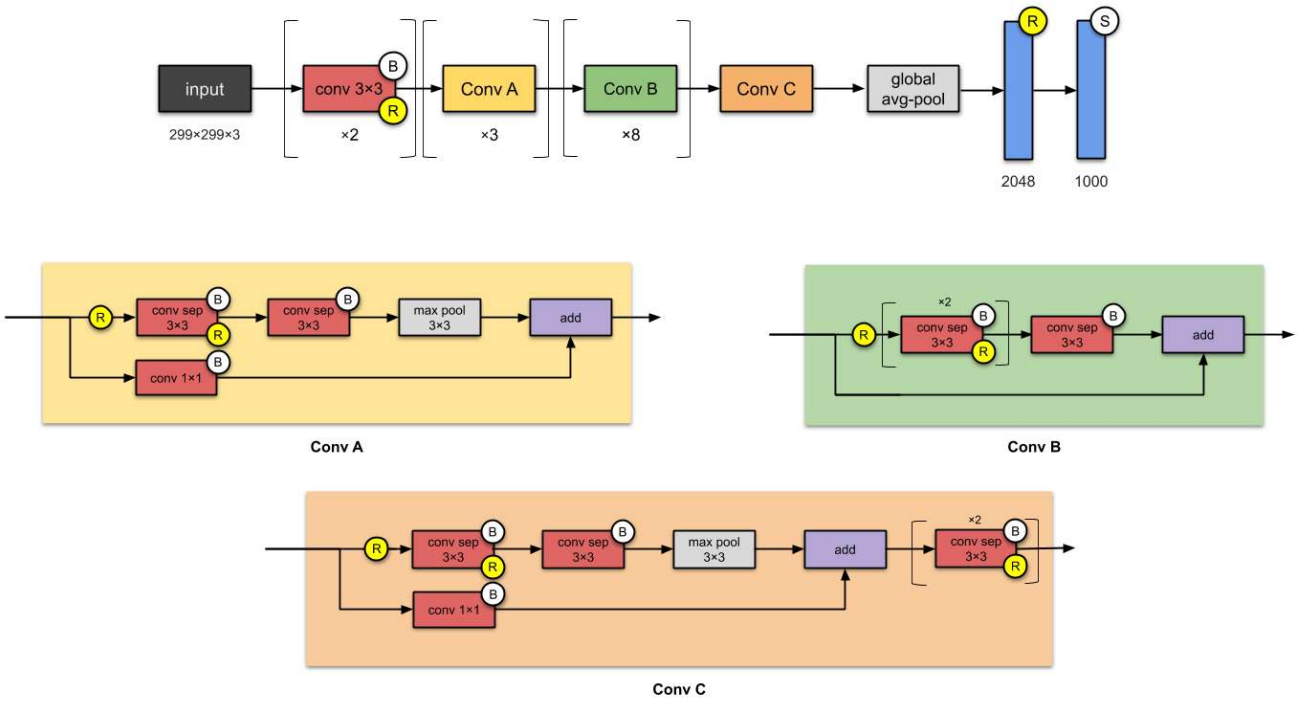
  The join together the resnet and inception V3 allow to obtain the new models.



Xception (2016)

Xception (eXtream Inception) is an adaptation from Inception, where the Inception modules have been replaced with depthwise separable convolutions.

- this means performing 1×1 to every channel, then performing a 3×3 to each output. This is identical to replacing the Inception module with depthwise separable convolutions.



DenseNet, (2017)

The expansion of the ReNet idea can be similar to moving gradient (and feature extraction) through different paths. Thus it could be build a several paths for one layer.

In this case the low-level information can be shared between layers of the each block.

It also need here to replace summation to concatenation.

$$\mathbf{x} \to [\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})]), f_3([\mathbf{x}, f_1(\mathbf{x}), f_2([\mathbf{x}, f_1(\mathbf{x})])]), \ldots] \,.$$

In other words "concatenating feature-maps learned by different layers increases variation in the input of subsequent layers and improves

efficiency." - channel-wise concatenation

- Due to concatenation it no need to have the same channel size in the each skip connection.
- concatenation instead of summation allow to more optimal reuse information.

- DenseNet join the ideas of ResNet (2015), HighWayNet(2015) and fractalNet (2017).
  - The idea is to ensuring maximum information (and gradient) flow.

    Within this instead of drawing representational power from extremely deep or wide architectures, DenseNets exploit the potential of the network through feature reuse.

    Every layer has access to its preceding feature maps, and therefore, to the collective knowledge.
  - By the DenseNet trick the whole number of parameters is smaller then for other method due to the lack of learn redundant feature maps.

    Some variations of ResNets have proven that many layers are barely contributing and can be dropped. Instead, DenseNets layers are very narrow and just add a small set of new feature-maps.

- Furthermore, each layer has direct access to the gradients from the block input and thus learn easier.

- For variating the whole model size we need to use additional transition layers.

  In trans layers down-sample the image dimensions with average pooling.

  Transition Layers need to keep the dimensionality under control when composing the network by adding transition layers that shrink the number of channels again.

  Transition Layers applying down sampling using batch normalization, a 1x1 convolution and a 2x2 pooling layers.

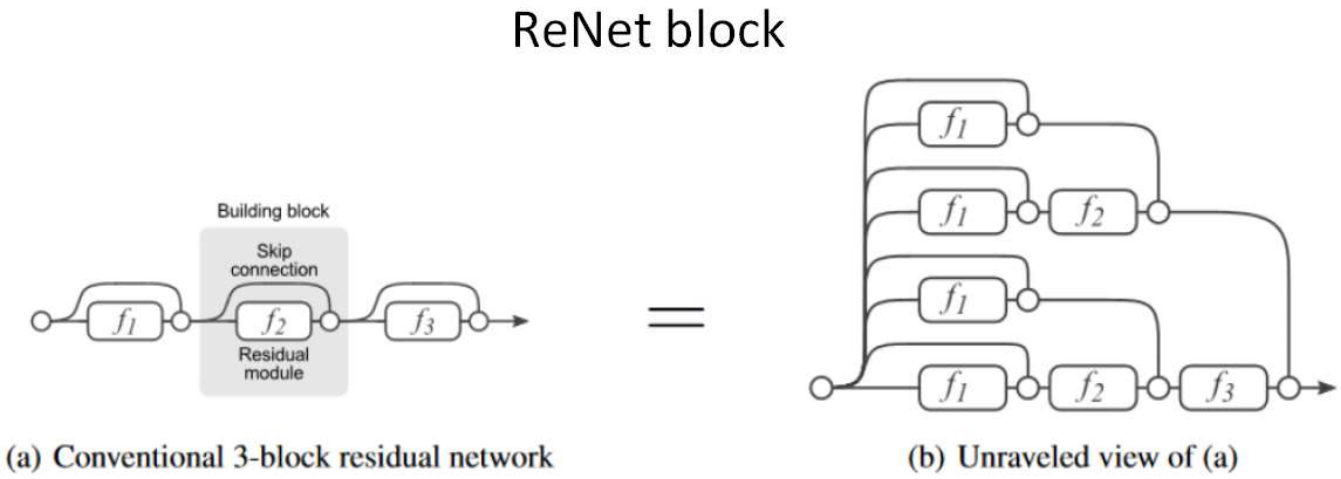
- The number of feature maps of the total architecture is the growth rate - additional densnet parameter.

- **The growth rate (k)** controls the amount of information to be added by each layer.

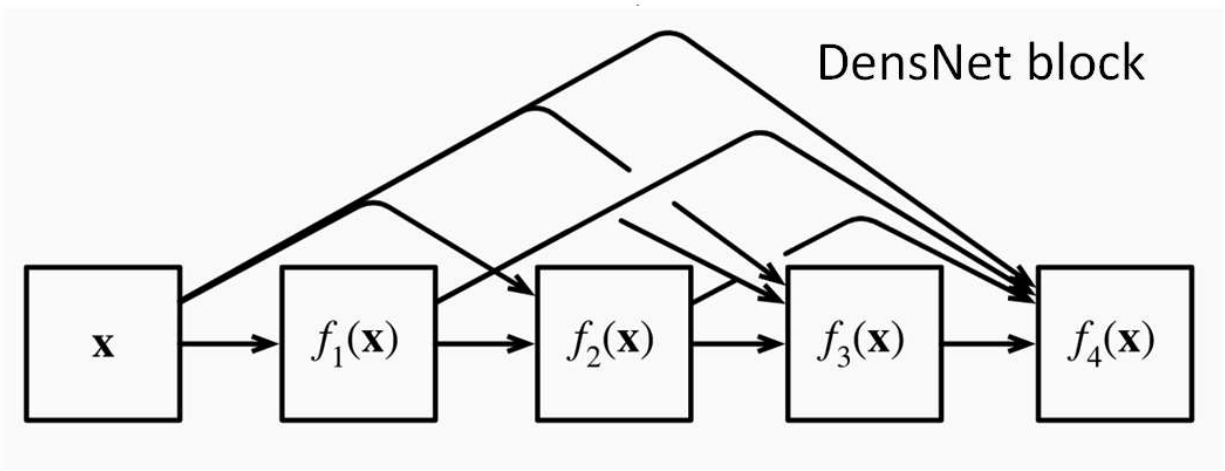
- Traditional architecture is DenseNet-121, however it was also proposed DenseNet-169, -201, -264, and some small modification of transition layers DenseNet-B, DenseNet-BC (-C).

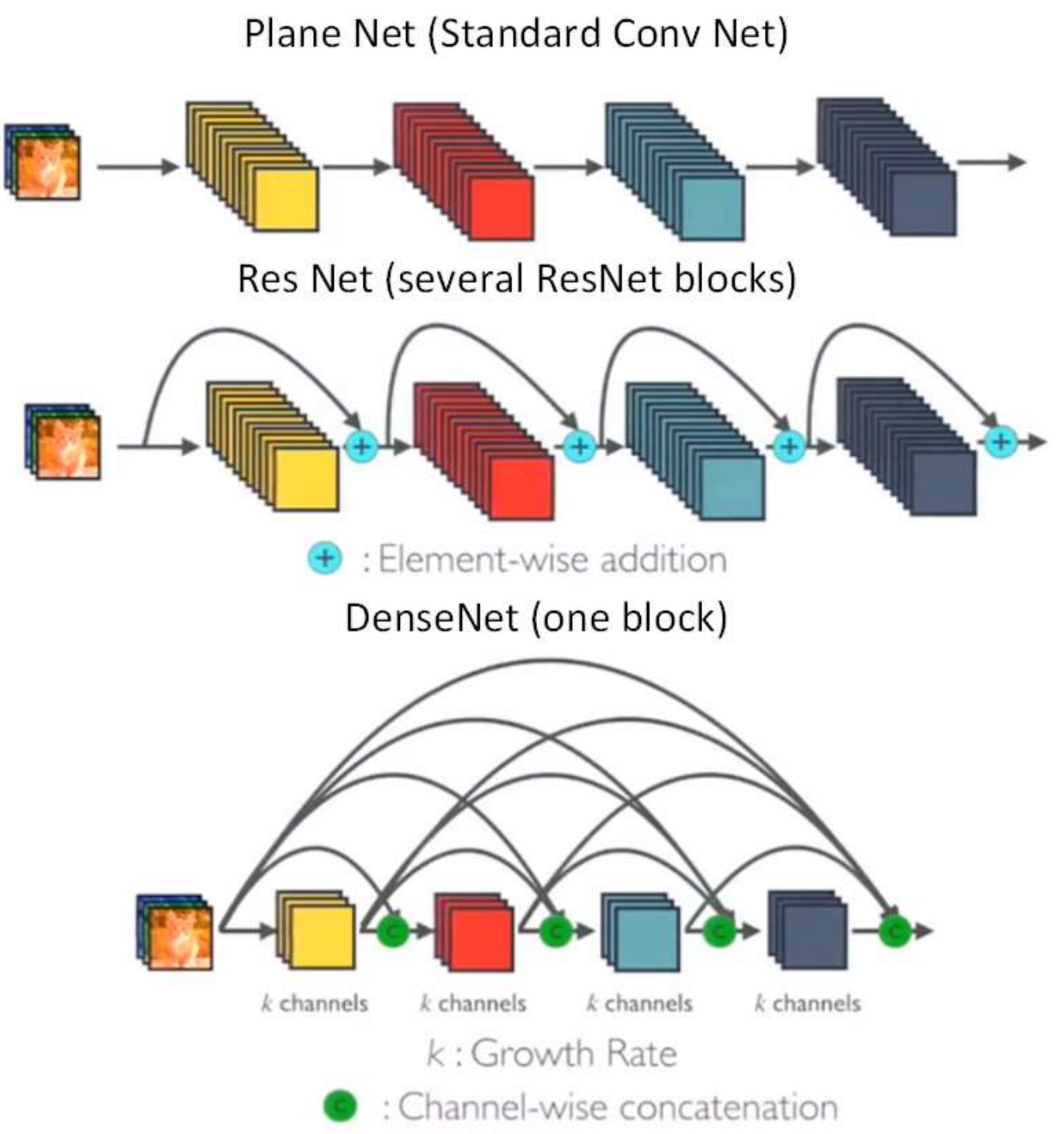
  The B, BC variants have some compression factor in transition layer $0 < \theta \leq 1$ which determine number of feature map after transition (0.5

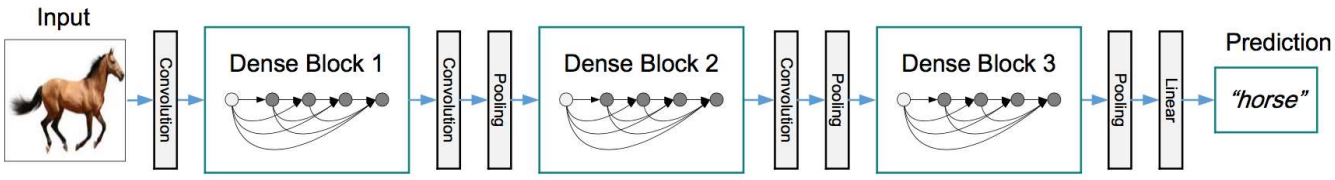
for -C).

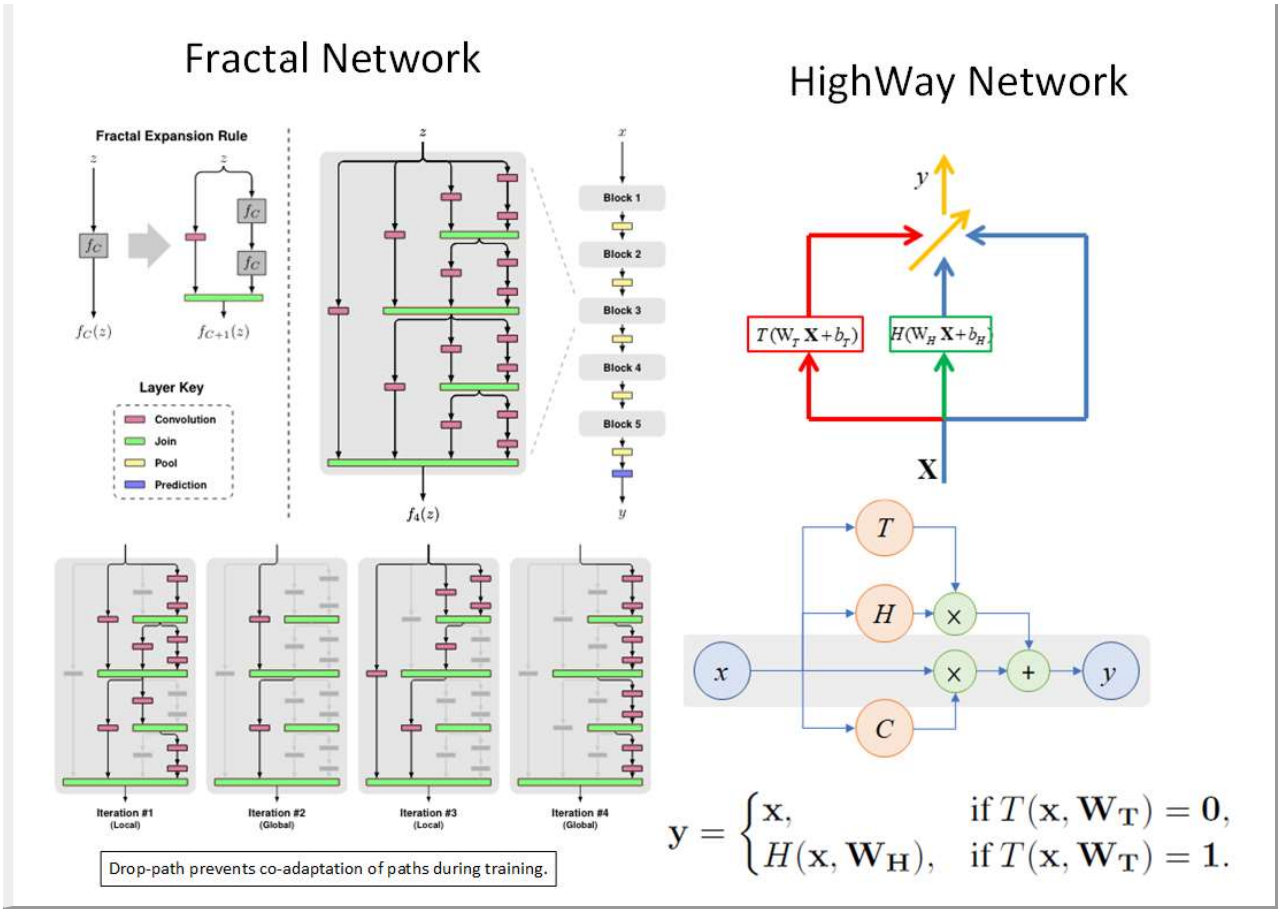


(a) Conventional 3-block residual network (b) Unraveled view of (a)




Plane Net (Standard Conv Net)
Res Net (several ResNet blocks)
: Element-wise addition
DenseNet (one block)
k channels
k channels
k channels
k channels
k : Growth Rate
: Channel-wise concatenation


Input
Convolution
Dense Block 1
Convolution
Pooling
Dense Block 2
Convolution
Pooling
Dense Block 3
Pooling
Linear
Prediction
"horse"



BiT (Big Transform Architecture), (2019).

The one of the modern modification of ResNet architecture is the Big Transform Architecture.

The model is based on the standard ResNet (50, 101 and 152) with increase depth and width (in 1,3 and 4 times) it is also replaced BatchNorm (BN) with GroupNorm and Weight Standardization

The model was pre-trained on a very large generic dataset.

It was proposed three versions of BiT:

- BiT-L (Large size) : pretrained upon the dataset JFT-300M (contains 10 to 1 billion image datasets).

- BiT-M (Medium size): pretrained upon the dataset Imagenet-21K (contains 14 million image datasets)
- BiT-S (Small size): trained upon the dataset ILSVRC-2012 (1.3 million image datasets)

It was also proposed BiT-HyperRule for fine-tuning, where all hyperparameters are fixed except:

- training schedule length,
- resolution,

    if image size are less than 96 X 96 then we will resize and crop images to 160 X 160 and 128 X 128 consecutively otherwise , if more than 96 X 96 then we will resize and crop images to 448 X 448 and 384 X 384
- use MixUp regularization f the size of the dataset is more than 20K.
- Use batch size of 512

Note

The reasons to replace BN with GroupNorm and Weights Standardization:

- when training large models with small per-device batches, BN performs poorly or incurs inter-device synchronization cost.
- Second, due to the requirement to update running statistics, BN is detrimental for transfer.
- It was shown that GN, when combined with WS, allow to improve performance on small-batch training for ImageNet and COCO. ALso they useful for training with large batch sizes, and has a significant impact on transfer learning.

1.1.4 MobileNet and its variants

SqueezeNet, (2016).

SqueezeNet the first attempt to build CNN architectures that can easily be deployed in mobile applications.

The idea: -efficient building blocks, in order to reduce the number of parameters.

- The main block is the fireblock:

1. squeeze channels with three 1×1 convolutions.
2. expand the output with a several 3×3 and 1×1 convolutions in parallel.

- At the very end is a convolution layer that performs the classification, followed by global average pooling. (the classification layer both has ReLU and softmax applied to it.)

  Note, There are no fully-connected layers.

- SqueezeNet is the most fast and simple architecture, but has a relatively small performance.

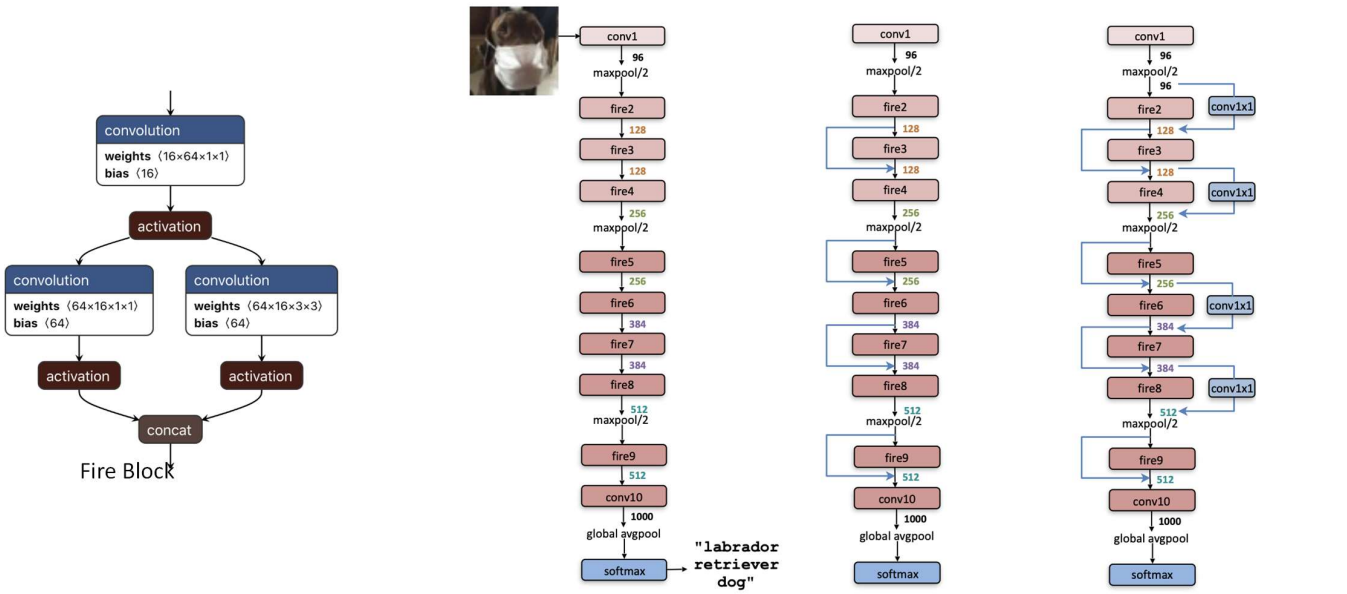


Figure 2: Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet (Section 3.3); Middle: SqueezeNet with simple bypass (Section 6); Right: SqueezeNet with complex bypass (Section 6).

The similar to SqueezeNet idea lay in

SqueezNeXt network (2018)

The SqueezeNext block consists in

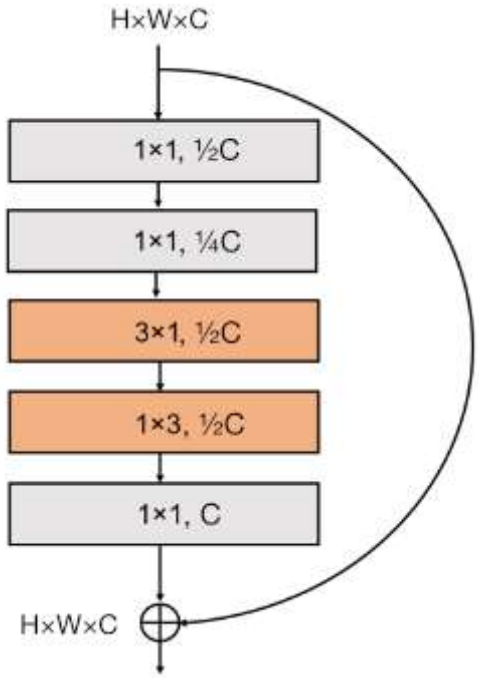
- sequentially squeezing number of channels with two bottleneck layers (to C/2, and C/4).

    It is proposed that two stages allow to make more non-linearities - i.e. more accuracy squeezing

- Spatial-separable convolution 3×1 and 1×3

    The order of these two convolutions alternates. If this block has 3×1 followed by 1×3, the next block does 1×3 first and 3×1 second, and so on.

- The back expansion of channels.
- residual connection.
- SqueezeNext incorporates a final bottleneck layer to reduce the input channel size to the last fully connected layer.
- The depth and nuber of layers here are hyperparameters.



SqueezeNext

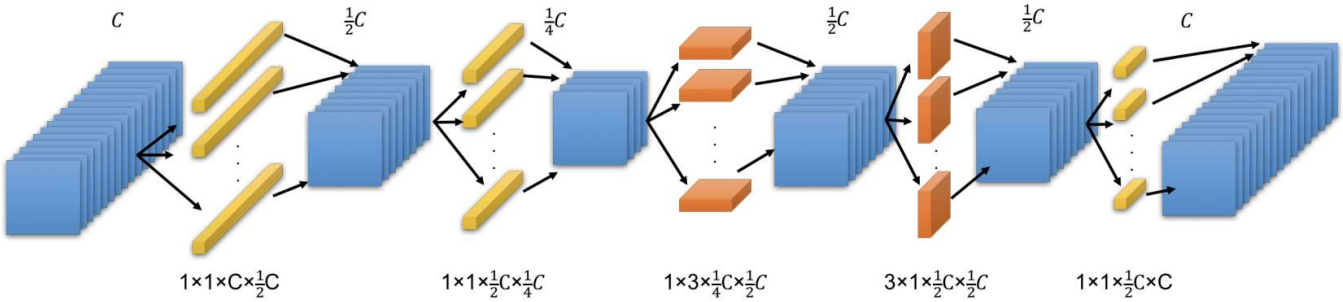


Figure 2: *Illustration of a SqueezeNext block. An input with C channels is passed through a two stage bottleneck module. Each bottleneck module consists of* 1×1 *convolutions reducing the input channel size by a factor of 2. The output is then passed through a separable* 3×3 *convolution. The order of* 1×3 *and* 3×1 *convolutions is changed throughout the network. The output from the separable convolution is finally passed through an expansion module to match the skip connection's channels (the skip connection is not shown here).*

MobileNet V1(2017), **MobileNet V2**(2018)

MobileNets the first success attempt to build CNN architectures that can easily be deployed in mobile applications.

The main idea under Mobile Net is to replace conventional convolution with deepwise separable convolution.

- MobileNet has two parameters

    - Width multiplier (depth multiplier, α) - how many channels are used by the model's convolution layers.
    - Resolution multiplier (ρ) - how to reduce the dimension of the channels in the internal and input layers.

        In practice we implicitly set ρ by setting the input resolution.

- MobileNet v1 consists of 13 building blocks.
- Each MobileNet V1 block is the DeepWise Separable Convolution with batchnorm.
    - MobileNet blocks does not use max pooling
    - Some of the depthwise layers have stride 2 instead of max pooling.
- At the end MobileNet V1 use a global average pooling layer followed by a fully-connected layer or a 1×1 convolution for doing the classification.
- All MobileNet V1/V2 blocks use ReLU6 is used as the activation function instead of plain old ReLU.

    ReLU6 is more robust than regular ReLU when using low-precision computation.
- MobileNet v2 architecture consists of 17 building blocks.
- At the end MobileNet V2 use a regular 1×1 convolution, a global average pooling layer, and a classification layer.

- MobileNet V2 use blocks - bottleneck convolution block with expansion layer:
  - 1 stage - expansion (t times - expansion factor $\in [5, 10]$, 6 by default) - creating the manifold of interest ($D_f \times D_f \times C_{in} \rightarrow D_f \times D_f \times (t \times C_{in})$)
  - 2 stage - deep wise layer obtaining $(D_f/s) \times (D_f/s) \times (t \times C_{in})$, where s is the stride.
  - 3 stage - project layer (or bottleneck layer) - decrease the expansed dimension to the $(D_f/s) * (D_f/s) * C_{out}$, where C_{out} is the number of channels on the block output. The idea (or hypothesis) of this layer is to reduce dimension without reducing of information (use nested manifold).

    The trick that makes this all work, of course, is that the expansions and projections are done using convolutional layers with learnable parameters, and so the model is able to learn how to best compress/decompress the data at each stage in the network.
  - Residual connection to help with the flow of gradients through the network. (used when the number of in-channels and out-channels

are the same, and size of channels the same).

The authors call this an **inverted residual** because it goes between the bottleneck layers, which have only a small number of channels.

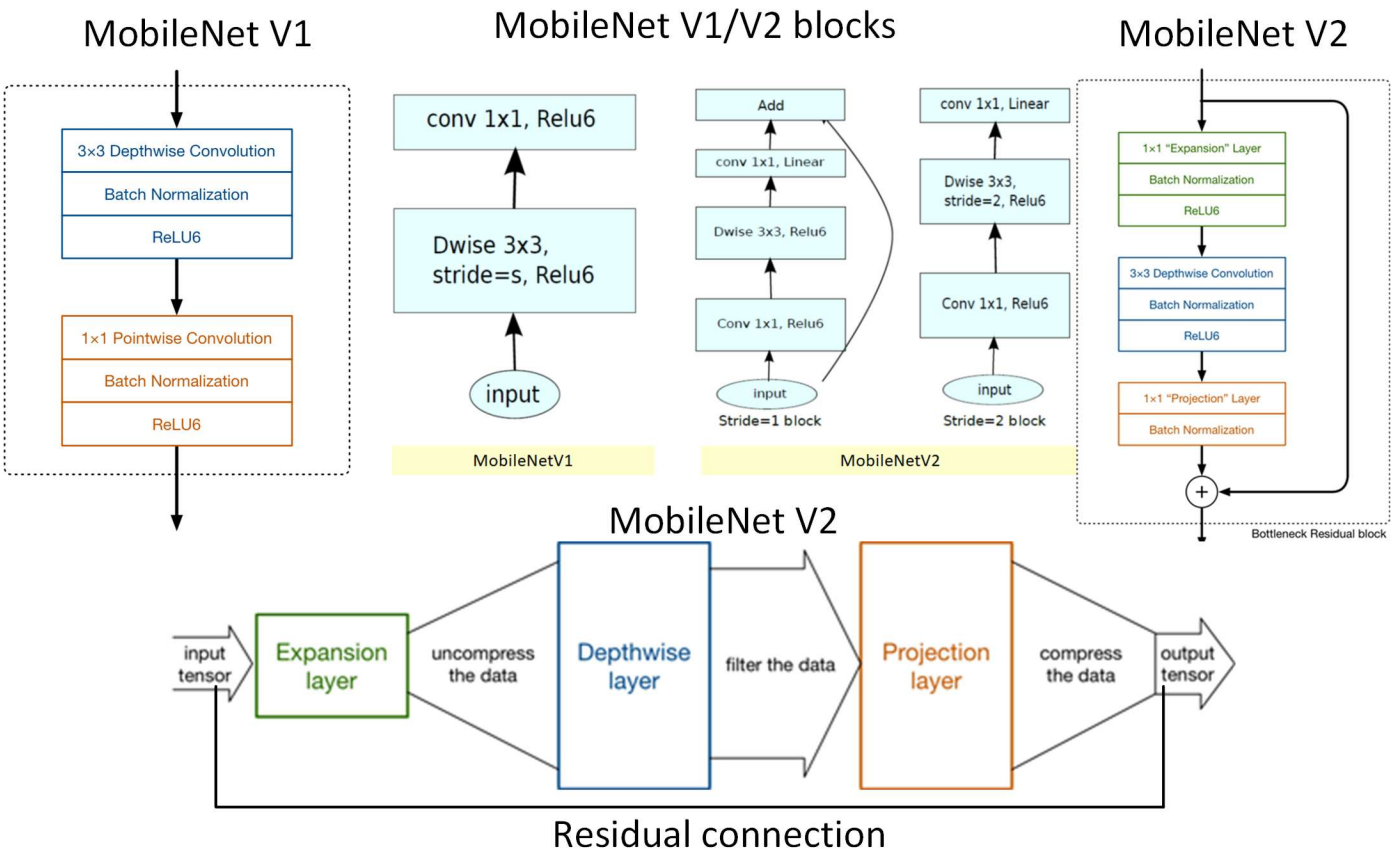
- As usual, each layer has batch normalization and the activation function is ReLU6.
- The output of the projection layer does not have an activation function.

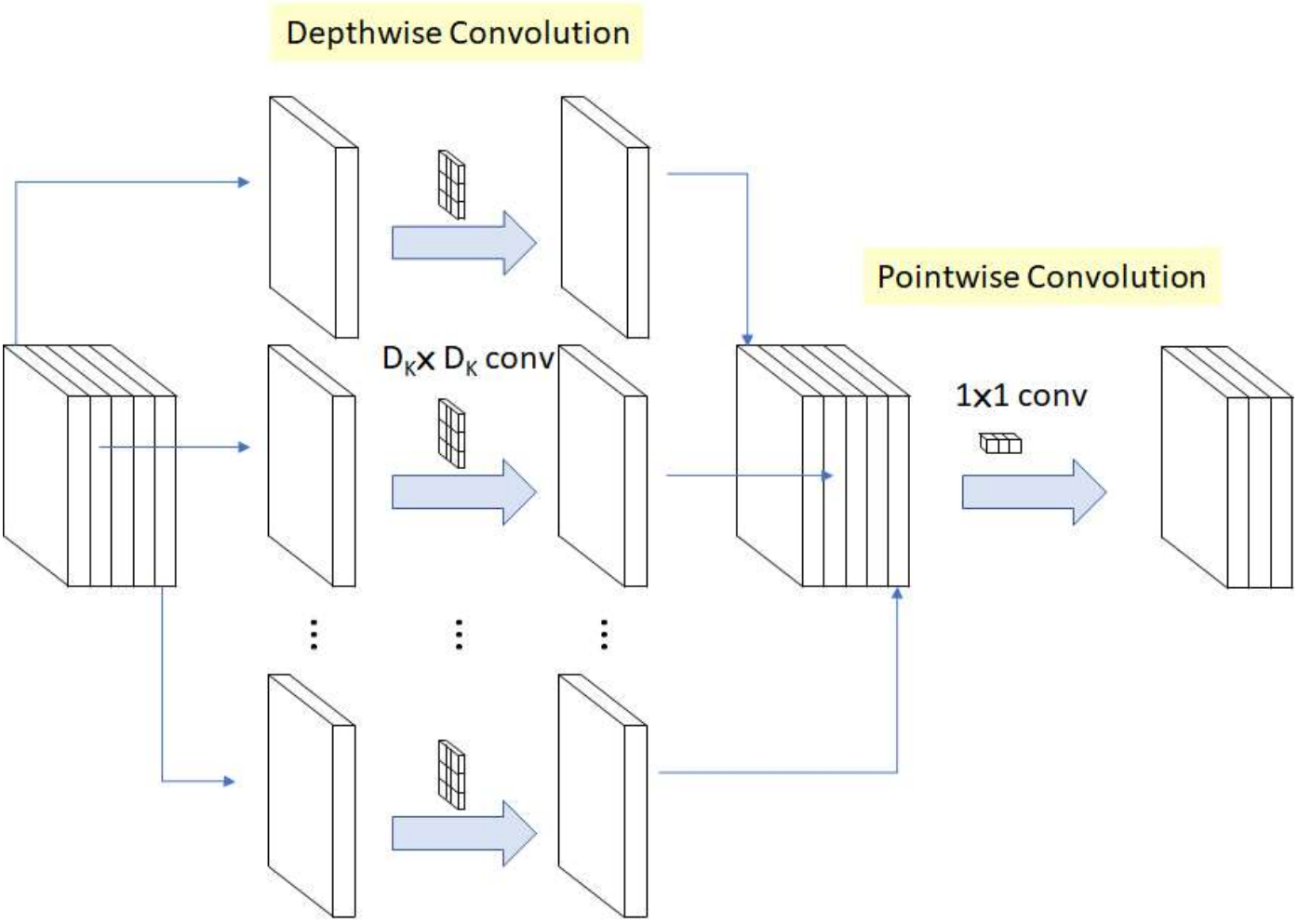
  The authors call this an **linear bottlenecks**.

  Since this layer produces low-dimensional data, the authors of the paper found that using a non-linearity after this layer can destroyed useful information.

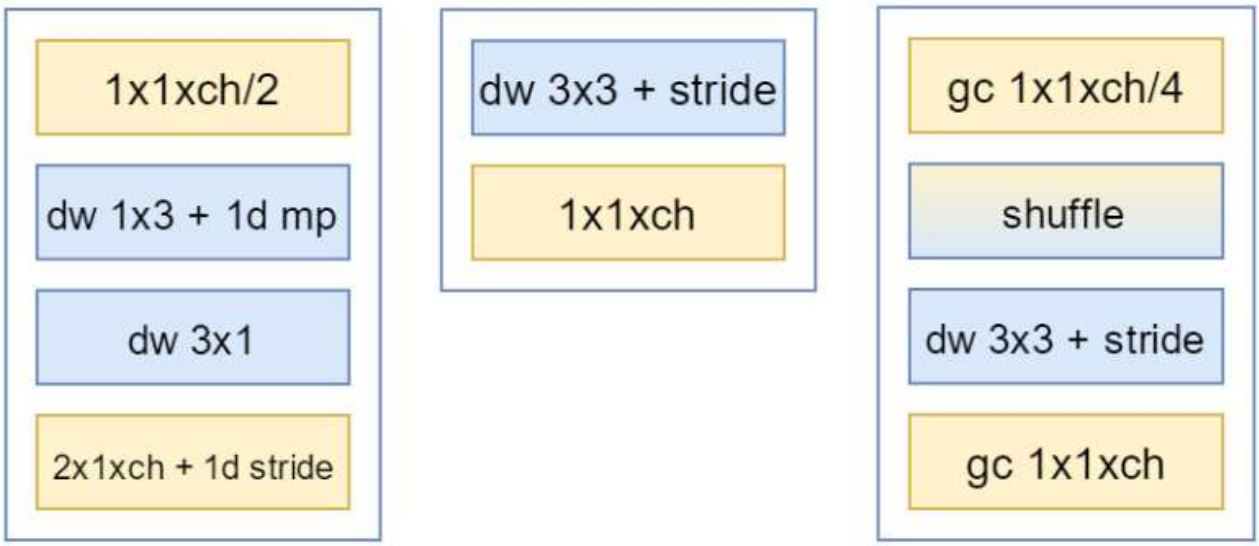
  MobileNet V1 use only stage 1 and 2 of MobileNet V2 block.

  Thus MobileNet V2 have much fewer parameters than V1 and a slightly improved score on the classification benchmark.





The other variant of the MobileNet V1 block is the EffNet block



(a) An EffNet block (ours) (b) A MobileNet block (c) A ShuffleNet block

ShuffleNet V1(2017), **ShuffleNet V2**(2018)

The a lot of convolution can have a high computation cost which can be bring down using group convolutions. **A group-wise convolution** divides the input feature maps into two or more groups in the channel dimension, and performs convolution separately on each group. Note,

> Group convolution is similar to depthwise convolution. In fact, depthwise convolution is sometimes implemented using the more general grouped convolution.

When the group convolution is applied the weights are not shared between the groups — each group can learns its own parameters - its own features.

Actually, it could be the drawback if it is no information flow between different groups - because it can limits the network's ability to learn.

However it is possible to increase the variations of network information by shuffling groups.

After the grouped convolution, the channel shuffle operation rearranges the output feature map along the channels dimension to increase the information of all system.

The ShuffleNetV1 block consists in

- 1 stage - 1×1 grouped convolution - bottleneck layer (reduces the number of channels by a factor of 4.
- 2 stage -shuffle channel .
- 3 stage 3×3 depthwise convolution with batchnorm but without ReLU.

  It was discovered that not using ReLU gave better results here.
- 4 stage 1×1 grouped convolution for expansion the number of channels again (to match those of the residual connection). Without channel shuffle.

  the authors found it didn't make any difference in that spot either use shuffle here or not.

- residual connection is added to output.
- Sometimes the depthwise layer can have stride 2. In that case, the residual branch has a 3×3 average pooling layer inside it and the results are concatenated with the main branch, not added.
- The authors stay that 8 groups seems to give the best results.

  For shuffle net v1 the full architecture starts with a regular 3×3 convolution with stride 2, which is followed by max pooling. Then there are three stages, each with 4 or 8 ShuffleNet blocks.

  The first block in a stage has stride 2. Finally, there is global average pooling and a fully-connected layer that does the classification.

The ShuffleNet V2 use modified block structure:

- 1 stage - split operation - sends half the channels through the left branch and the other half through the right branch

  Actually, split operation is like using two groups.
- 2 stage The 1×1 convolution (are no groupwise) with batchnorm and relu.
- 3 stage The 3×3 deepwise convolution with batchnorm and without relu.
- 4 stage The 1×1 convolution (are no groupwise) with batchnorm and relu.

- 5 stage connection of convolution result with the shortcut connection (residual connection)

    connection instead of add due to it is now reverse to split operation.

    Also elementwise operations such as addition are relatively expensive — they perform a lot of memory accesses but do little computation.

- 6 stage The channel shuffle.
- On the practice it's possible to merge concat, channel shuffle, and channel split into a single element-wise operation.
- All convolutions always have the same number of input and output channels.

    Authors found that, when the number of channels stays the same, the amount of memory accesses in the convolution is optimal.

- ShuffleNet V2 use modified first block structure:
    - In first blocks the depthwise convolution has stride 2,
    - For this blocks it is no split operation,

        Both branches work on the same data and the concat doubles the number of channels.

    - The residual branch in this case gets its own 3×3 depthwise convolution with stride 2, followed by a 1×1 convolution,

it is done to make sure the outputs of both branches have the same spatial dimensions.

Beside ShuffleNet V2 is also a newer variant, ShuffleNet v2+, which adds hard-swish, hard-sigmoid, and squeeze-and-excite, just like in MobileNet v3.

- The accuracy scores are among the best though network for CPU.
- The ShuffleNet model does not run well on the GPU due to the Shuffle operation have been designed for CPU. *Note*

The shuffle perform such to instance of each group in other one.

The shuffle algorithm: If G is the number of groups and K is the number of channels per group, and the tensor is (N, C, H, W), then:

- reshape the input tensor from (N, C, H, W) to (N, K, G, H, W)
- transpose the G and K dimensions in the array, so now the tensor has shape (N, G, K, H, W)
- reshape the tensor back to (N, C, H, W).

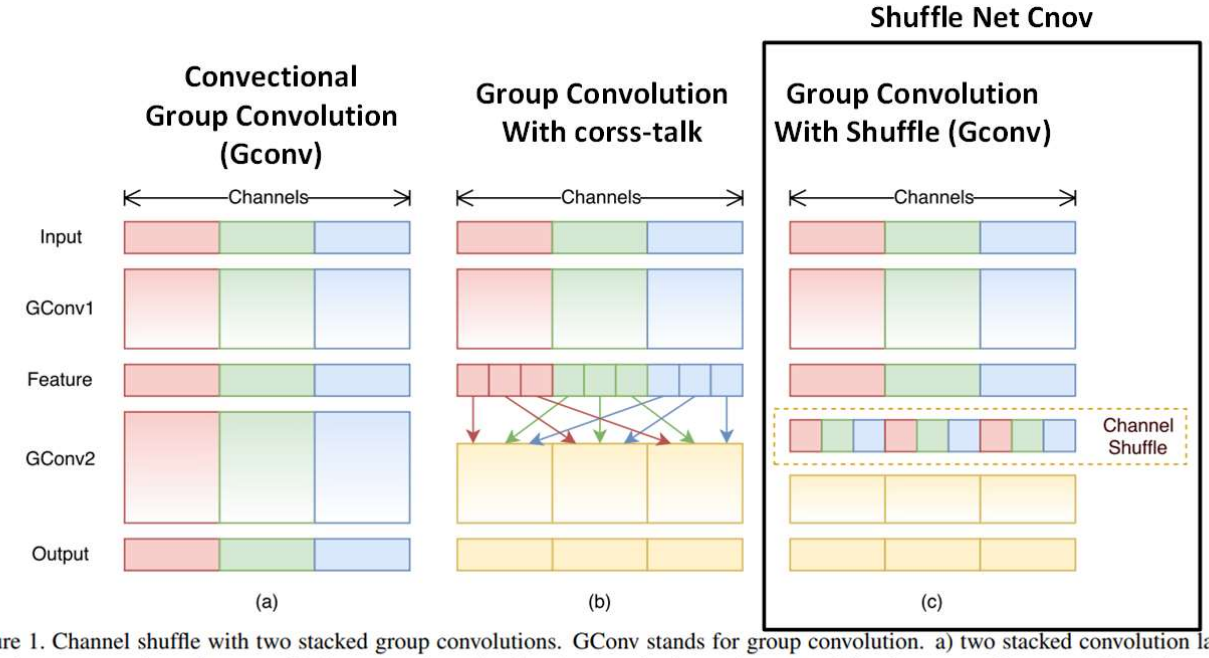
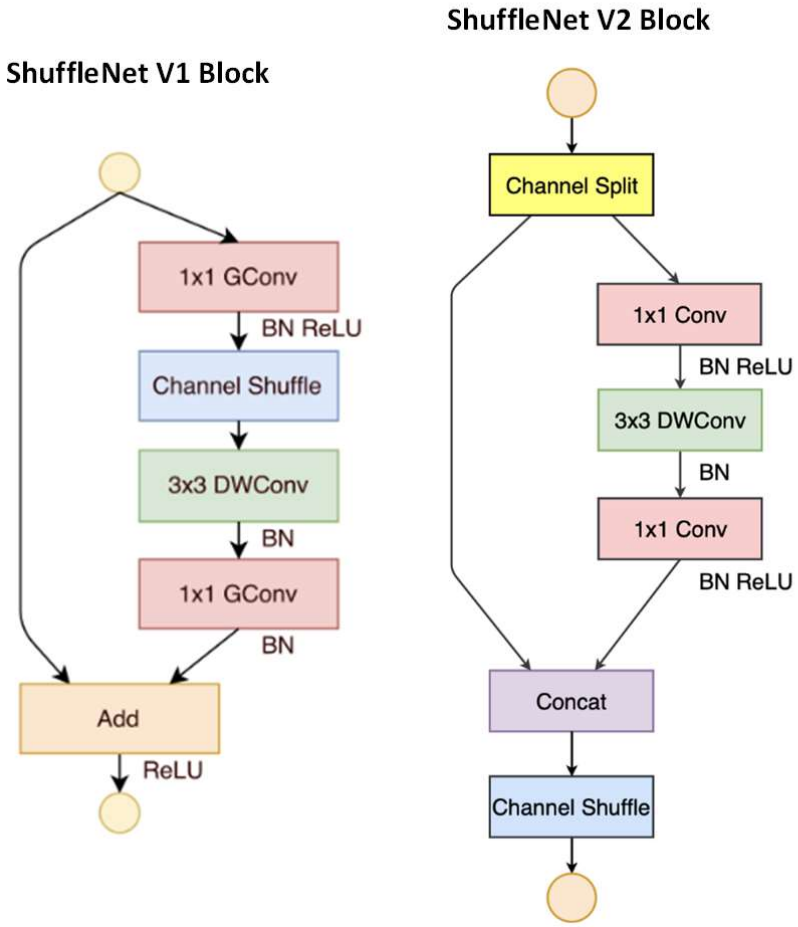
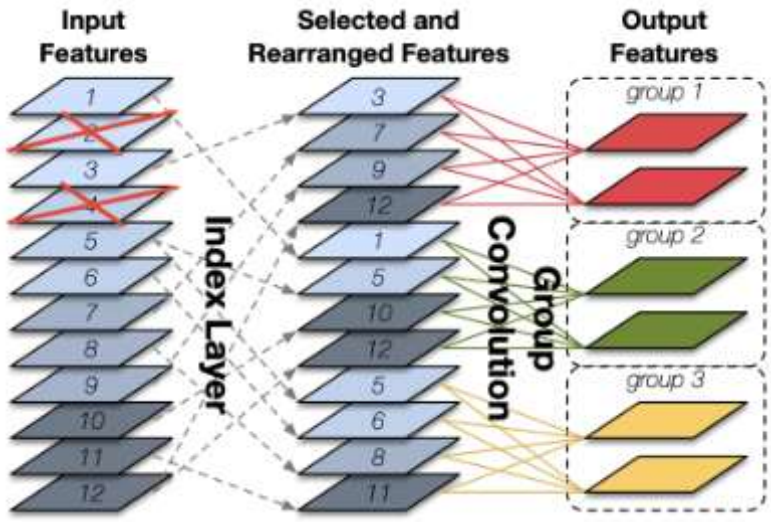


Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when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



The similar to ShuffleNet idea proposed in the **CondenseNet** (2017). But here the input features that are considered less important are pruned from

the model - during introduced pregroup stage, then all features are shuffled and divided into groups.



CondenseNet

Squeeze-and-Excitation Networks (2019)

The main idea under this block is recalibrate the channels to gain the information that they contain -to increase sensitivity to informative features.

map the channel dependency along with access to global information to performance gains.

Each SE block consists in

- a feature transformation (convolution) input X to get features U.
- squeeze operation to get a single value for each channel of output U. (Global Average Pooling, F_{sq})

F_{sq} give output X with dimensions $1 \times 1 \times C$, where C is the number of channels.

- The excitation operation on the output of the squeeze operation to get per-channel weights (F_{ex}).

  F_{ex} using two full connected layers $F_{ex} = \sigma(W_2 ReLU(W_1 X)) = s$, where σ is the sigmoid; s is the vector of coefficients each in the range from 0 to 1.

- Re-scaling weights with identity connected feature map (F_{scale}).

  $F_{scale} = s \odot U$ - scaling with excitation operation coefficients.

In the original idea re-scaling in earlier layers allows to:

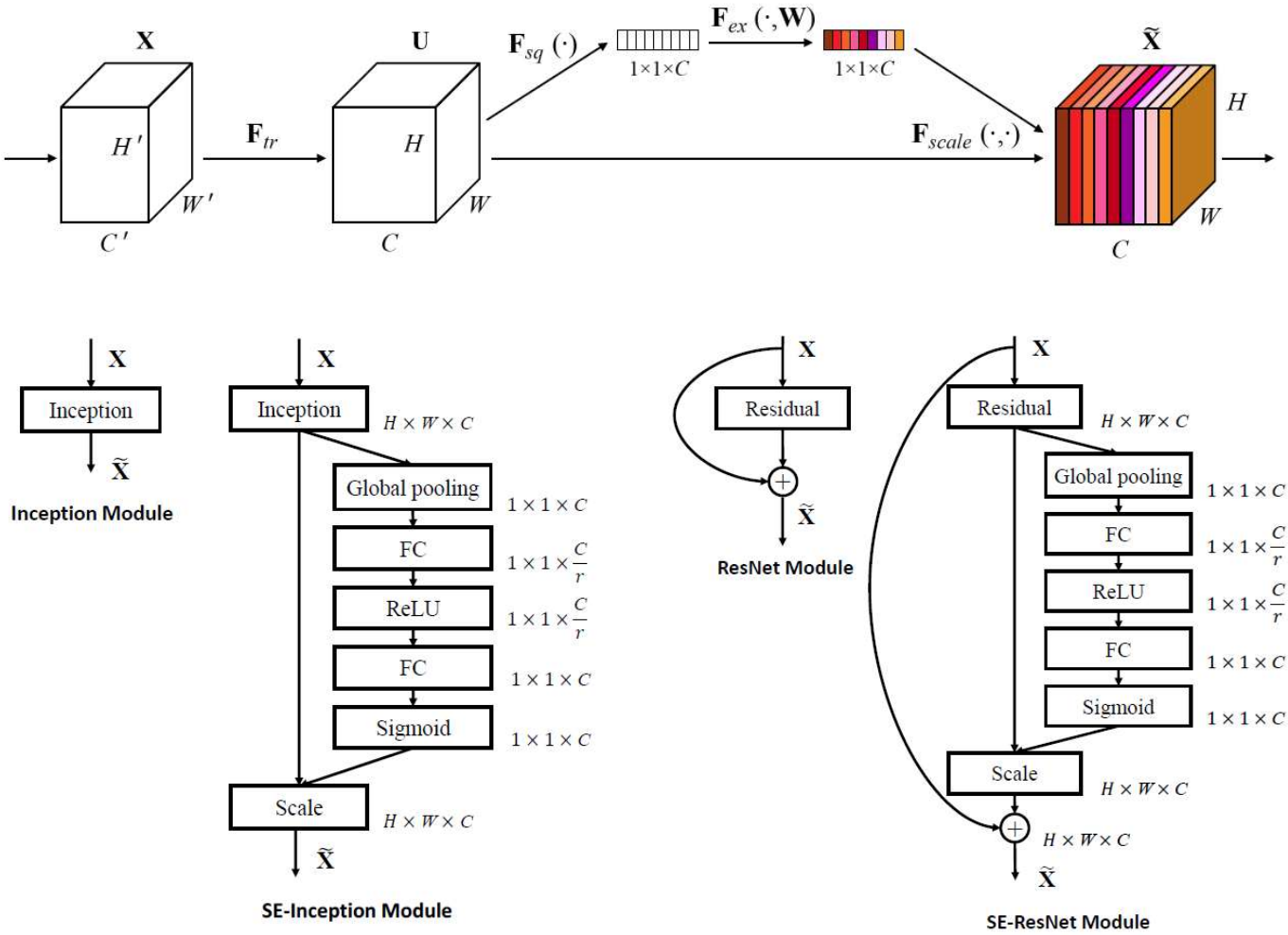
- excites informative features (in a class-agnostic manner as authors stay);
- strengthening the shared low-level representations.

In later layers, the SE blocks become increasingly specialised, and respond to different inputs in a highly class-specific manner.

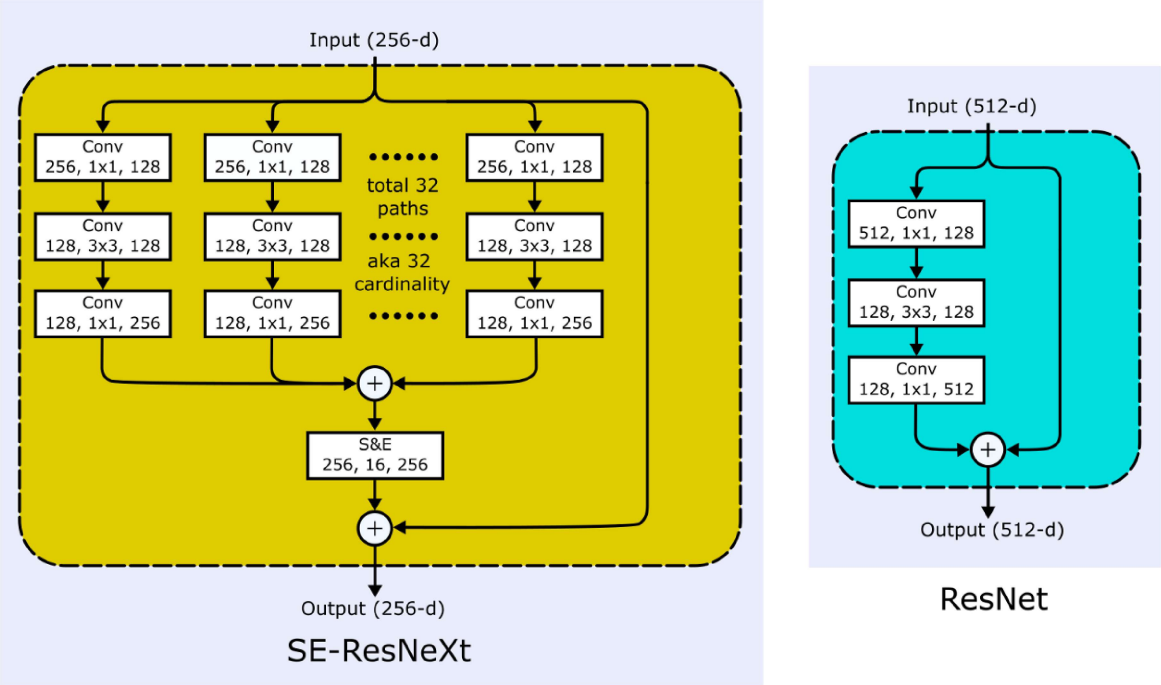
- As a consequence, the benefits of the feature recalibration performed by SE blocks can be accumulated through the network.
- SE Block can be considered as analogue of self-attention.

However, here sigmoid applied instead of soft-max to have analogue of multi-class weighting (instead of only one channel attention in soft-max).

- It cold be distinguished SE Inception block (described above) and Se-ResBlock with additional residual connection.
- SE blocks are also computationally lightweight and impose only a slight increase in model complexity and computational burden.



It could also be SE-ResNeXt



1.1.5 Neural Architecture Search - MNas, MobileNetV3, EfficientNet

MnasNet (2018).

The authors of MnasNet inspired by SENet and MobileNet v2 proposed to search the best architecture configuration. For this case authors apply the neural architecture search (NAS) to find an architecture that is specifically suited for mobile devices.

The obtained architecture is similar to MobileNet v2, however it include its own implementation of SE block:

- global average pooling operation (the Mean layer) that reduces the spatial dimensions of the c feature maps to 1×1×c.
- excitation part: performed by two 1×1 convolution layers with a non-linearity in between (ReLU in this case).

The first conv layer massively reduces the number of channels by a factor of 12 or 24,

- the second conv layer increases the number of channels again to C.

- A sigmoid: is applied to scale the results to be between 0 and 1.

It was proposed a several MNas variants with SE and (MNas-A1/A2) and without SE (MNas-B1).

Note

FusedBatchNorm is the representation of the BatchNorm operation through the convolution operation (and as part of it) to increase its speed. The fusing of batch normalization operation into convolution can be represented as:

$$\hat{\mathbf{y}} = \mathbf{W}_{BN} \cdot (\mathbf{W}_{conv} \cdot \mathbf{x} + \mathbf{b}_{conv}) + \mathbf{b}_{BN}, = \mathbf{W}_{fused} \cdot \mathbf{x} + \mathbf{b}_{fused}$$

where

- $\mathbf{W}_{BN}$ and $\mathbf{b}_{BN}$ are the representation of batch normalization as the convolution operation.
- $\mathbf{W}_{BN} = \text{diag}\left[\ldots, \frac{\gamma}{\sqrt{\hat{\sigma}^2+\epsilon}}, \ldots\right]$
- $\mathbf{b}_{BN} = \left[\ldots, \beta - \gamma_i \frac{\hat{\mu}}{\sqrt{\hat{\sigma}^2+\epsilon}}, \ldots\right]^T$
- $\mathbf{W}_{fused} = \mathbf{W}_{BN} \cdot \mathbf{W}_{conv}$

- $\mathbf{b}_{fused} = \mathbf{W}_{BN} \cdot \mathbf{b}_{conv} + \mathbf{b}_{BN}$
- γ, β, μ, σ are calculated as usual.

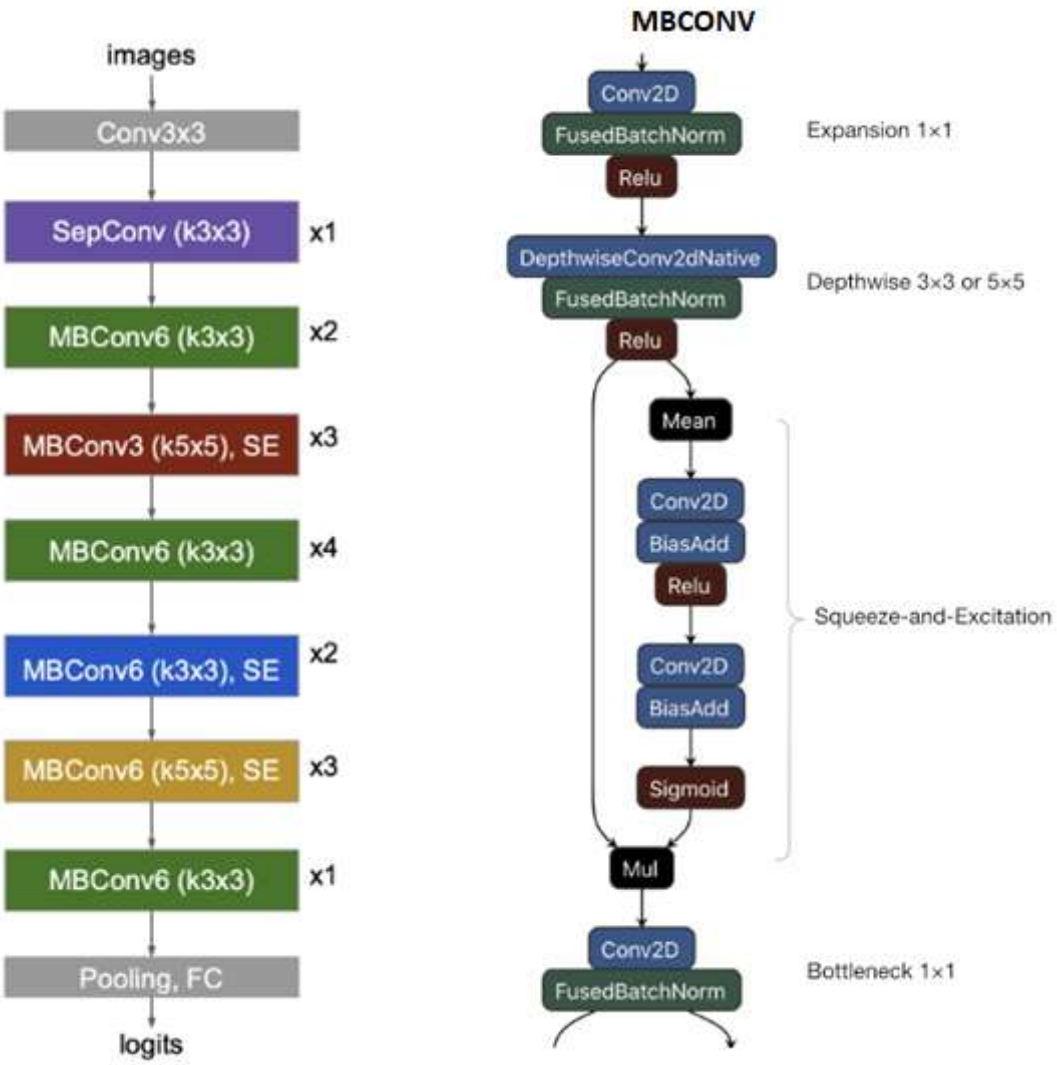
Note

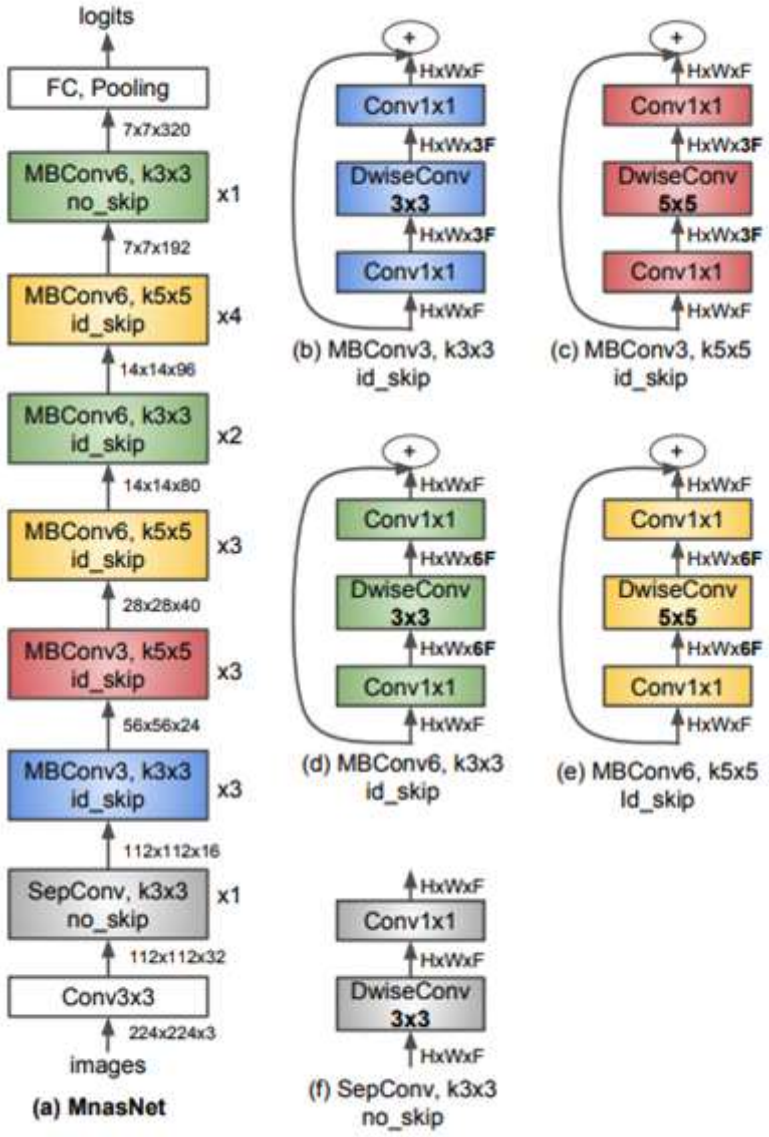
- MNas SE block use conv 1x1 operation instead of Full-connected layer due to equivalence of these layers.
- **The objective of neural architecture search** is to find a model that achieves a good trade-off between accuracy and latency. During the search, the latency of the potential new architectures is directly measured by executing them on mobile phones. This is called **platform-aware neural architecture search.**
- The Nas includes:
  - The search space: the type(s) of architectures that can be designed and optimized (or part of architecture, like conv, global pooling and e.t.c.).
  - The search strategy: the approach used to explore the search space (random search, reinforcement learning, hierarchical structure and e.t.c).
  - The performance estimation strategy: evaluates the performance of a possible architecture from its design

(without constructing and training it).

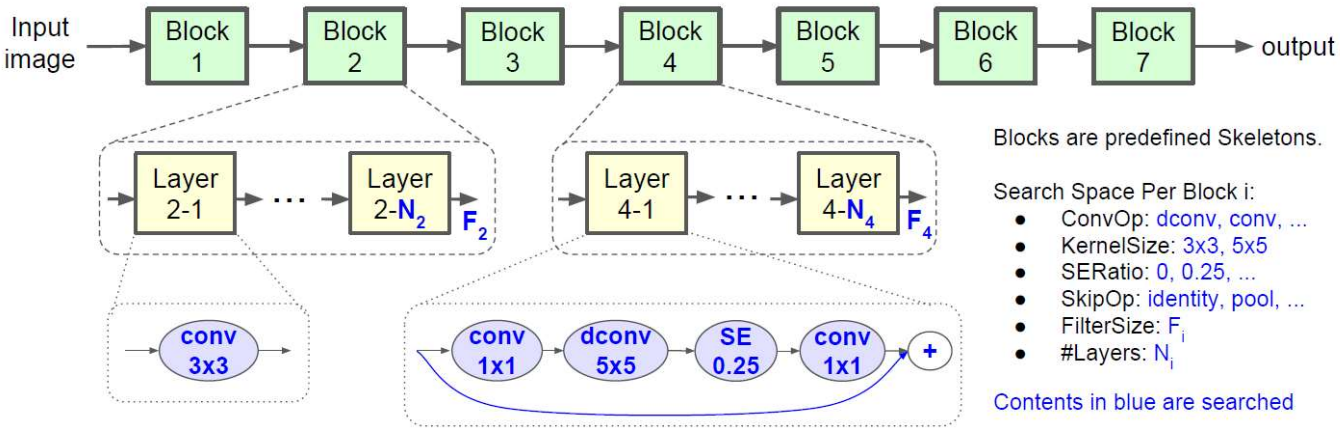
NAS is closely related to hyperparameter optimization and

is a subfield of automated machine learning (AutoML).





MNas without SE (MNas-B1)



General MNas scheme

MFBNet (2018)

The other idea of NAS search with fixing the overall structure of network

and search for the best block architecture. As base block the MobileNet V2 block was taken.

The block consists in

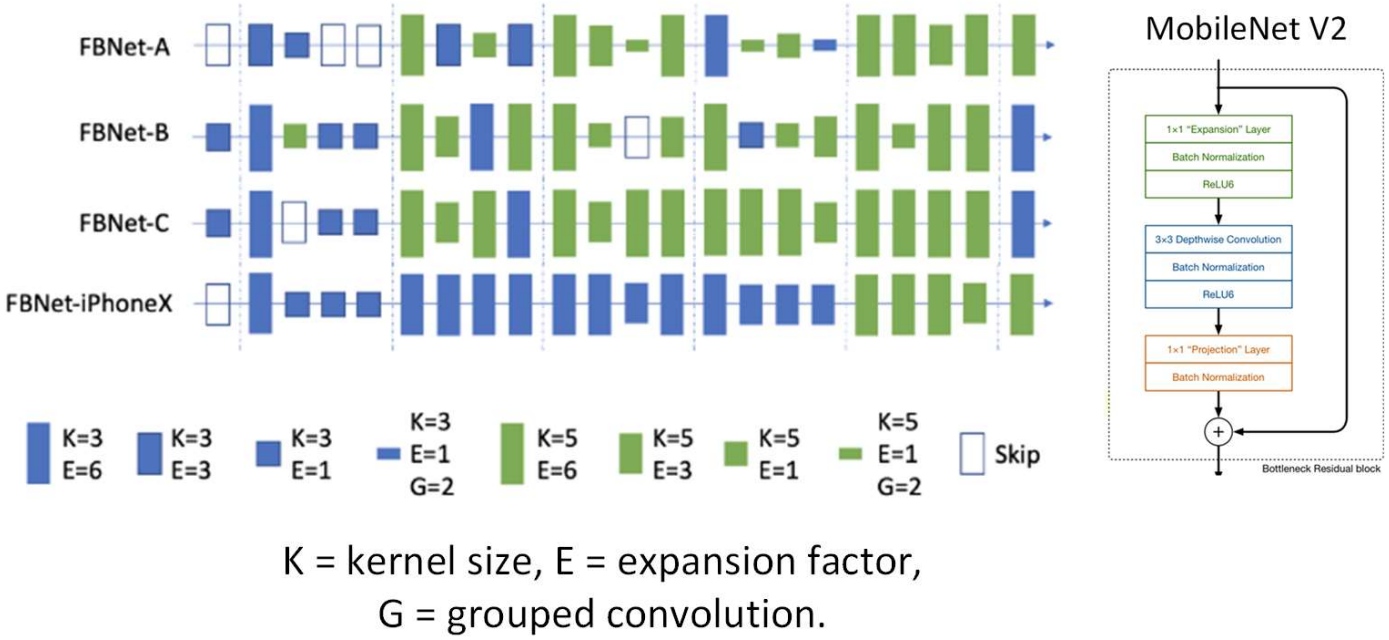
- 1×1 convolution for expansion + BN+ReLU
- depthwise convolution + NB + ReLU
- 1×1 bottleneck layer + BN (with no activation)
- residual connection (if output has same size as input)

In the search was the following assumptions:

- The first 1×1 conv layer can expand the number of channels by a factor 1, 3, or 6.
- The depthwise convolution can be 3×3 or 5×5.
- Both 1×1 layers can use group convolution.
- In total, there are 9 possible configurations of this building block that the search can choose between.
- The overall structure of the network is fixed: the number of layers, the number of filters per layer, which layers do stride 2, etc.
- Some of the blocks can be only identity one.
- The network starts with a search of configuration of single convolution layer, then there are several stages of four building blocks

each, followed by a final 1×1 convolution.

- Finally, always the global average pooling and a classifier layer
- All layers, except for the global pooling and the classifier, are found by searching.



K = kernel size, E = expansion factor,
G = grouped convolution.

MobileNet V3 (2019)

The MobileNet V3 architecture was partially found through automated network architecture search (NAS, block-wise search) and from NetAdapt, Which is used to fine-tune the number of convolution kernels of each layer after the network layer is determined for each module, so it is called Layer-wise Search.

The algorithm was additionally simplifies a pre-trained model until it

reaches a given latency, while keeping accuracy high.

Analyzed blocks was similar to one in ShuffleNet(V1 and V2), MNasNet, CondenseNet, EffNet and other.

The main change in the architecture in comparison with MobileNetV2:

- expensive layers were redesigned
- use of Hard-Swish instead of ReLU6 in deep layers

h_swish$(x) = x * \text{ReLU6}(x + 3)/6$;

Not all of the model uses h-swish: the first half of the layers use regular ReLU (except after the first conv layer). The MobileNet creators found that h-swish was only beneficial on the deeper layers.

- Include Squeeze-and-excitation modules (SE) in blocks
  - In this block sigmoid replaced with so-called hard-$\sigma = \text{ReLU6}(x + 3)/6$
  - The squeeze-and-excitation (SE) modules only reduce the number of channels by a factor 3 or 4.
- It was also optimize the last layers.

  The main idea is to exceed the number of channels to make classification.

  In MBN V3 all channels was 1x1 (instead of 7x7 in MBN V2).

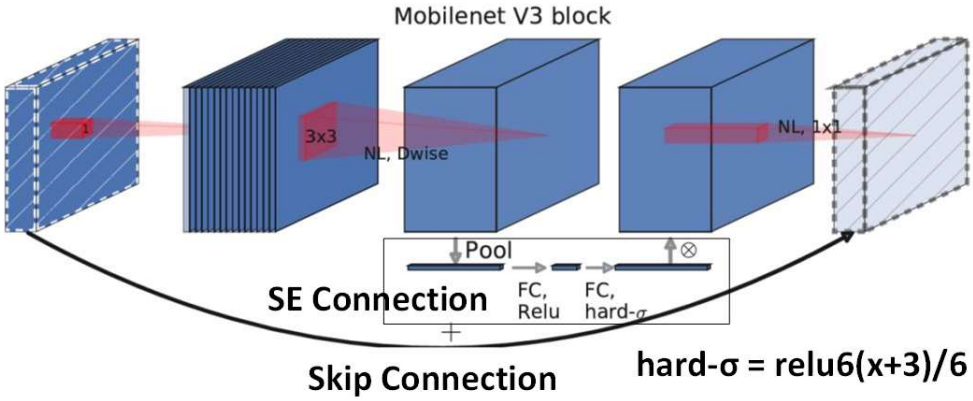
  Also there are not applied batch normalization.

It was proposed a several architecture variants:

- Large: full, described above;
- Small: this has fewer building blocks and smaller numbers of filters
- Large minimalistic: like Large, but doesn't have SE, h-swish, 5×5 convolutions
- Small minimalistic: likewise for the Small variant

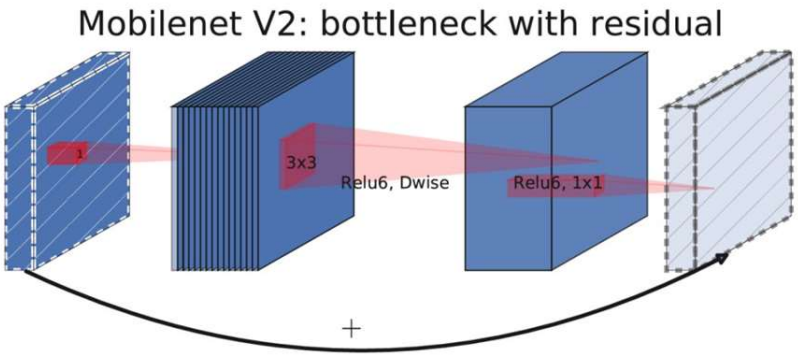
All of the MBNet variant allow to use depth-multiplier. *Note*

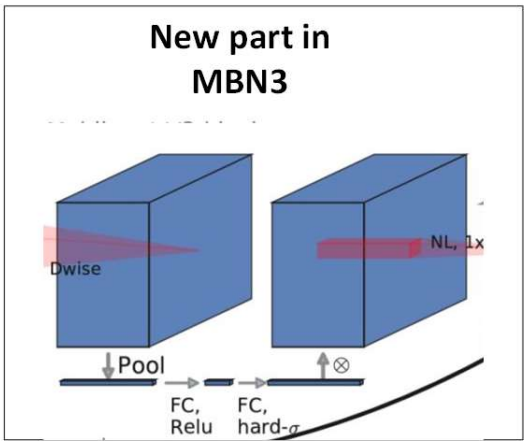
> Actually, MobileNet was designed for Android OS and TFLight framework, in some other cases it is can be more optimal to use other architectures, or at least it could be more fast to use original sigmoid and swish ($swish(x) = x\sigma(x)$).

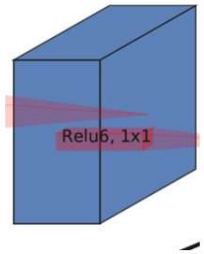


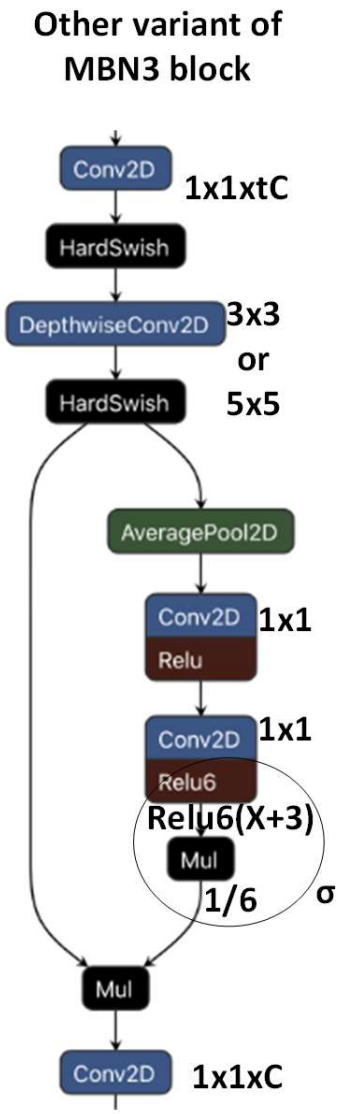
hard-σ = relu6(x+3)/6

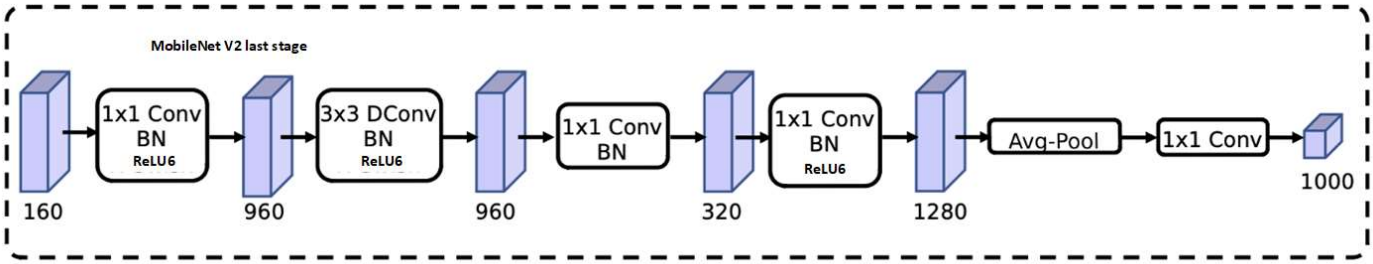
NL -

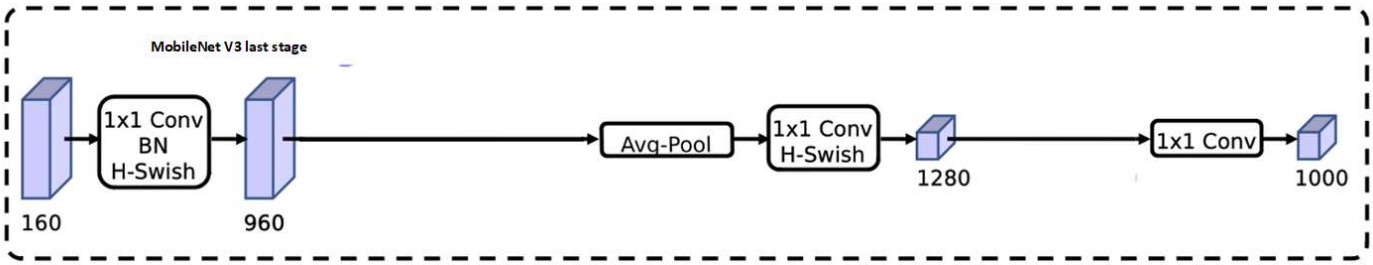












EfficientNet (2019)

The architecture was obtained using NAS architecture search. Within that it was proposed to use MobileNet V3 (MBN3)-like blocks. Due to architecture search with MBN3 block three parameters can be scaled up or down:

- **Network width**,$C \sim \alpha$: number of **filters (channels)** per convolution layer.
- **Network depth**, $L, \sim \beta$: **number of layers** in each stage of the model.

    I.E. the number of stages stays the same, but stages can be made deeper by adding more layers to them.
- **Input resolution** $H \times W, \sim \gamma$: The **dimensions** of the input image.

    common 224×224 , the higher resolution can improve the accuracy, the resolution can be decreased.

    Note

        Intuitively, if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image.

For the parameter search it was proposed to use **compound scaling method**, that lets scale α, β, γ together.

The goal is to achieve the maximum possible accuracy while using as few FLOPS as possible.

For this it was proposed to use a base-line model F and its parameters of scaling:

$$\alpha_0^\phi, \beta_0^\phi, \gamma_0^\phi,$$

where

- $\alpha_0 = 1.2, \beta_0 = 1.1, \gamma_0 = 1.15,$
- ϕ is the change parameter, chosen such as the total FLOPS will increase by 2^ϕ (approximately) and $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.

Note

the square here is due to doubling network depth will double FLOPS, but doubling width or input resolution will increase FLOPS by four times.

EfficientNet-Bo - the first result of the proposed search (baseline network, $\phi = 1$)

The EfficientNet-B0 architecture is very similar to MnasNet,uses the same MBConv building blocks with 3×3 and 5×5 depthwise convolutions, squeeze-and-excitation (SE), and swish activation. There is nothing

particularly new in this architecture. **EfficientNet-B1-B7** are the modified variants of The EfficientNet-B0 with increased ϕ.

The number of stages and the types of building blocks in B1 – B7 stay the same as in B0, but resolution (γ), width (β), and depth (α) all become larger.

It was farther proposed also EfficientNet-EdgeTPU S, L and M (3 versions) and Light version - all with smaller number of parameters than for **B0**.

architecture optimized for TPU uints, designed without swish and SE blocks, sometimes Deepwise conv replaced with conventional conv.

Actually, there are exits a lot of EfficientNet modifications.

Note

- During the search for B0 the optimized parameter was

$$\text{ACC}(m) \cdot \left(\frac{\textbf{FLOPS}(m)}{T} \right)^{w}$$

where

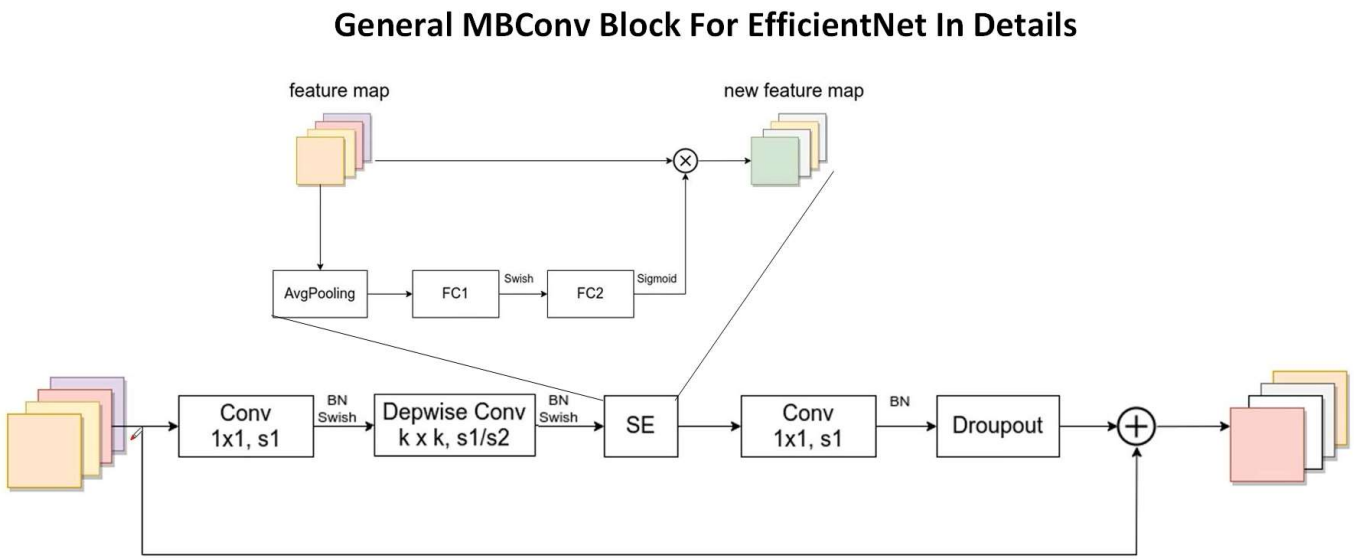
- ACC(m) and **FLOPS**(m) denote the accuracy and FLOPS of model m,
- T is the target FLOPS

  - $w = -0.07$ is a hyperparameter for controlling the trade-off between accuracy and FLOPS.
- The compound search aim is based on the trade off between computational cost (FLOP) and the following assumptions:
  - With more layers (depth) model can capture richer and more complex features, but the models are hard to train (due to the vanishing gradients)
  - Wider networks are much easier to train. They tend to be able to capture more fine-grained features but saturate quickly.
  - By training with higher resolution images, it can be captures more fine-grained details, but it is require more layers (more receptive field).
- Actually, the compound search performed into two steps:
  1. Fix $\phi = 1$, assume that twice more resources are available, and do a grid search of α, β, γ. The best acquired values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.2, \gamma = 1.15$ such as $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$;
  2. Fix α, β, γ and scale up ϕ with respect to the hardware (FLOPs + memory).

- Actually, EfficientNets use NAS to construct a baseline network (B0), then they use "compound scaling" to increase the capacity of the network without increasing the number of parameters greatly.
- In fact, this idea of Compound Scaling also works on existing MobileNet and ResNet architectures.

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

The achitecture for EfficientNet-B0



EfficientNet V2 (2020)

THe authors of EfficientNet V2 discovered that the extensive use of depthwise convolutions could be the training bottleneck.

> DepthWise convolutions in theory uses fewer parameters and FLOPSs than regular convolutions, but they are not fully utilized the hardware (as CPU as GPU) in some cases.

The Authors proposed to use so-called **fused convolution.**

> Here In the Fused-MBConv block, 1×1 and 3×3 convolution are fused together, forming a single convolution layer with 3×3 filter.
>
> To find where to place MBConv and Fused-MBConv layers, the authors use Neural Architecture Search.

The results of NAS show that the replacement of some of the MB Conv layers with the fused ones in early stages offers better performance with smaller models.

The other change to EfficientNet V1:

- uses smaller expansion ratio for MBConv

  tend to have less memory access overhead.
- Prefers to use 3×3 kernel sizes.

  but more layers are stacked to handle larger receptive fields.
- EfficientNet V2 completely removes the last stride-1 stage in the original EfficientNet, perhaps due to its large parameter size and memory access overhead.
- Authors used non-uniform scaling strategy to gradually add more layers at later stages.
- Authors also add a scaling rule to restrict maximum image sizes.
- Using of stochastic depth during the train in architecture search (similar as DropOut but replace the block on skip connection with some probability()see resnets).

In addition to NAS and compound search the EfficientNet V2 uses the **concept of progressive learning** that means that image sizes is increasing progressively during the learning and with increasing the regularization effect.

The main idea is to use different regularization on the different image size ((with increasing effect to high resolution). The following regularization is applied with intensity inversely proportional to the image size:

- dropout with change of probability;
- data augmentation with change of magnitude;
- mixup - cross-data augmentation

  $\text{new\_image}_i = \lambda \text{image}_i + (1 - \lambda)\text{image}_l$, $\text{new\_label}_i = \lambda \text{label}_i + (1 - \lambda)\text{l}$

  where l is random, and λ increased with the image size).

A huge regularization effect on small images would cause underfitting and a small regularization effect on large images would cause overfitting.

In addition, using of smaller images enables the network to use larger batch size, increasing the overall training procedure.

The authors proposed EfficientNet V2 in different variants S, M, L, XL.

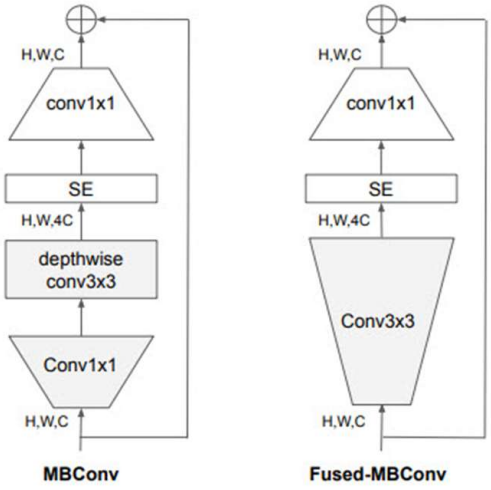
Note

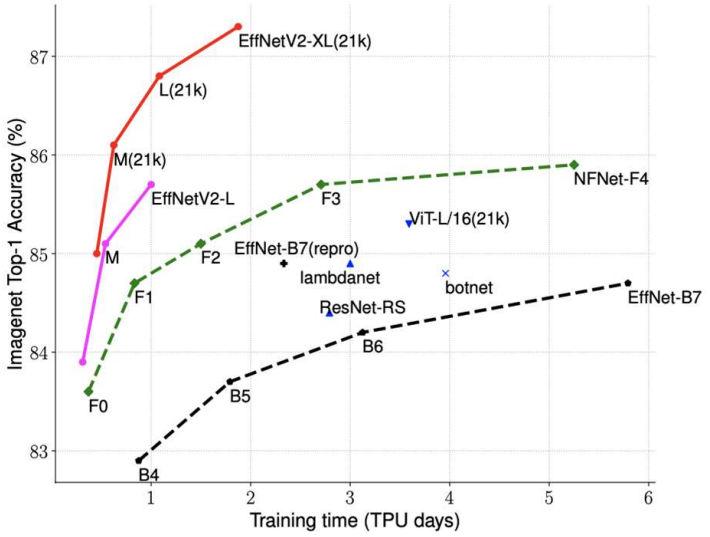
- Fused convolution was proposed for MobileDet architecture.

- Do not confuse the EfficientNet V2 fused convolution with the fused batchnorm (some times also called fused convolution).

Table 4. EfficientNetV2-S architecture – MBConv and Fused-MBConv blocks are described in Figure 2.

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	272	15
7	Conv1x1 & Pooling & FC	-	1792	1





EfficientNet With Noise Student, (2020)

The authors proposed to use semi-supervised self-training scheme for improvement performance of **EfficientNet V2**. The scheme consists of the follows:

- The quite-large dataset (Imagenet in the paper) divided into labeled and pseudo-unlabeled data (less part).

- The target-model called - student, also there taken teacher model (similar to student, but smaller).
- Train a teacher model on labeled images.
- Use the teacher to generate labels (float probability) on 300M unlabeled images (pseudo-labels),

  This stage work like in smooth-learning in inception V3, all possible labels are estimated.

Example of pseudo-label for class c and target y_i

$$: \begin{bmatrix} class & | & class_0 & \dots & class_c & \dots & class_{C-1} \\ label & | & y_{i0} & \dots & y_{ic} & \dots & y_{i,C-1} \\ instance & | & 0.1 & \dots & 0.91 & \dots & 0.05 \end{bmatrix}.$$

- Train a student model on the dataset combined of labeled and pseudo labeled images.
- Make new pseudo-labeling of pseudo-labeled data by student model.
- Change the student and teacher model and repeat previous three and current steps (train new student on the new dataset, make new pseudo-labeling and change student and teacher).
- During the student train use regularization technique randomly (called here noises).
- Noises are applied only for current student model. Teacher model (which make pseudo-labels) need to be without noises.

- As noises it could be applied different techniques such as dropout and stochastic depth (**model noises**) and random augmentation (**input noises**) to train the student model.
- Unlabeled Images need to be balanced for all classes.

Note

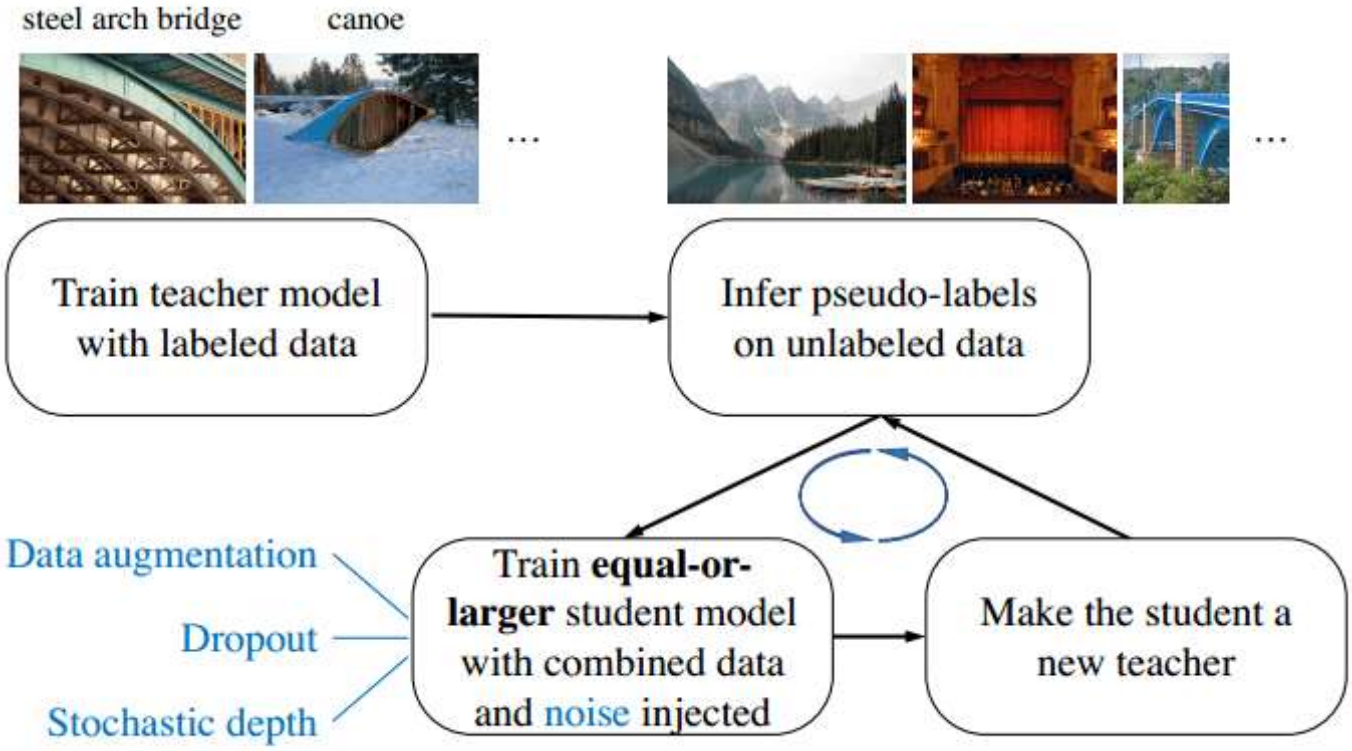
- Even if the architecture of the teacher and student can be the same, the capacity of the student model should be higher to make prediction on a large dataset (dataset with unlabeled data).
- It is stayed that noise applied to unlabeled data, enforces smoothness in the decision function.
- The technique is based on the Knowledge Distillation where on the contrary noise is often not used and a smaller student model is often used to be faster than the teacher.
- Techniques under the noise student:
  - **Supervised training**: The targets is defined as the one-hot vectors representing the ground-truth class,
  - **Knowledge distillation**: First train large model as supervised and make pseudo-labeling of all data, then train target model (which is

smaller).

- **Semi-supervised learning**: train the model on limited labeled data and make labeling for rest unlabeled data. Construct the target distribution as one-hot encoding (Hard label) for labeled data and as Soft labels for unlabeled data.
- **Smooth-learning**: add some predefined noises for one-hot encoded data.
- **Temperature learning**: make deterministic smoothing of softmax as $\text{softmax}_\tau(x_i) = exp(x_i/\tau)/\sum_j^K exp(x_j/\tau)$, where $\tau \in (0, 1]$
- **mixup** (cross-data augmentation)

  $\text{new\_image}_i = \lambda \text{image}_i + (1 - \lambda)\text{image}_l, \text{new\_label}_i = \lambda \text{label}_i + (1 - \lambda)\text{l}$

  where l is random, and λ predefined.



EfficientNet With Meta Pseudo-Labels (2020-2021)

The main drawback of just noise student model is that in the case when pseudo labels are inaccurate, the student will NOT surpass the teacher in accuracy (it would be bias of estimations - so called **confirmation bias**). The **Meto-Pseudo Labeling** consists in follows:

- Split data into labeled train, labeled validation and unlabeled train datasets.
- Pre-Train teacher model on labeled train data.
- Select one unlabeled sample x_u, one validation labeled sample x_v and one training labeled sample x_t.

- For the Teacher Model make prediction (pseudo-labeling $\hat{y}_u$) on selected unlabeled sample x_u (make only forward propagation).
- For the Teacher Model make one forward propagation for selected labeled train sample x_t and calculate a loss value L_t.
- For the Student Model pass one training stage (with backpropagation) for pseudo-labeled sample (for pair $\hat{y}_u$,x_u).
- For the Student Model make prediction on the selected validation sample x_v and calculate a loss value L_s.
- Make backpropagation of Teacher Model with join loss ($L_t + L_s$).
- Repeat from the item 3.

The discussed scheme have a drawback for large models - it is need to keep in training both network together. It could require a lot of time and actually obligate to keep both model in the GPU. However, if it is possible the scheme can be modified for using 3 models: Large Train Model, Reduced Train Model and Student (target) Model. This scheme is called **ReducedMLP (Reduced Meta Pseudo-Labels)** and consists in follows:

- Split data into labeled and unlabeled parts.
- Train Large Teacher model on the labeled part.

- Make pseudo-labeling of the unlabeled part.
- Make the **Meto-Pseudo Labeling** but only with small -Reduced Teacher and all data pseudo-labeled.

Note

Reduced Teacher need to be a small and efficient network, such as a multi-layered perceptron, is parameterized to be trained along with the student.

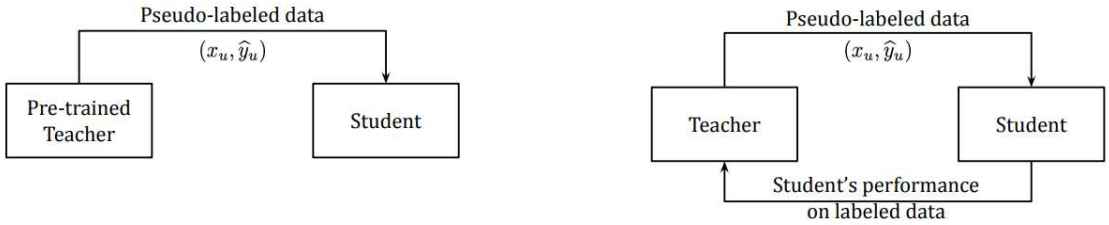


Figure 1: The difference between Pseudo Labels and Meta Pseudo Labels. **Left:** Pseudo Labels, where a fixed pre-trained teacher generates pseudo labels for the student to learn from. **Right:** Meta Pseudo Labels, where the teacher is trained along with the student. The student is trained based on the pseudo labels generated by the teacher (top arrow). The teacher is trained based on the performance of the student on labeled data (bottom arrow).

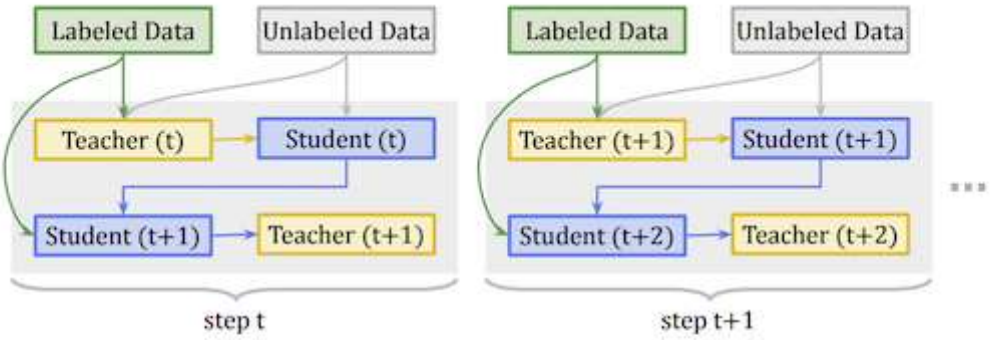


Figure 4: The MPL training procedure. At each step, the teacher receives both the MPL signal (Section 3.1) and the supervised signal from labeled data.

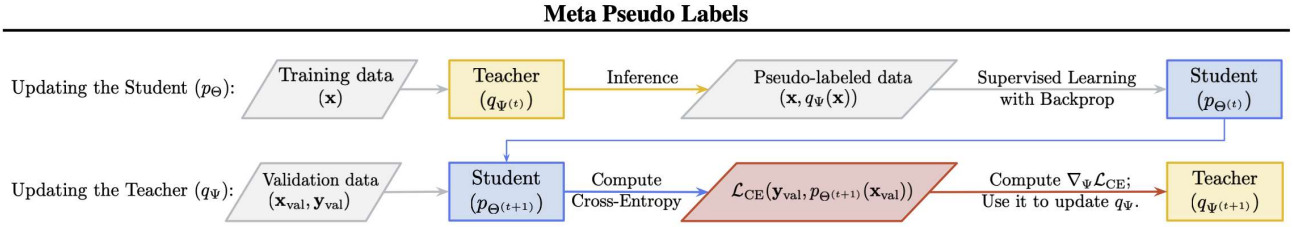


Figure 2: At each training step t, our Meta Pseudo Labels (MPL) update consists of two phases. **Updating the Student (top):** The teacher network q_Ψ assigns the conditional class distribution for a training example $\mathbf{x}$. The student p_Θ learns from $(\mathbf{x}, q_\Psi(\mathbf{x}))$ by standard supervised learning, updating from $\Theta^{(t)}$ to $\Theta^{(t+1)}$. **Updating the Teacher (bottom):** The teacher updates its parameters Ψ based on the resulting student's cross-entropy loss on validation data $(\mathbf{x}_{\text{val}}, \mathbf{y}_{\text{val}})$.

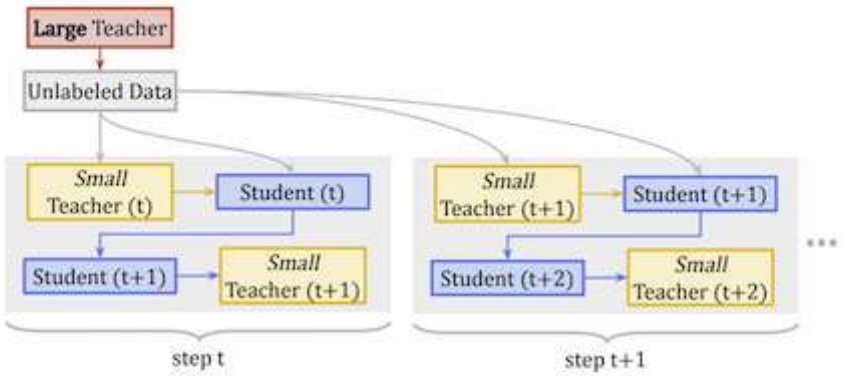


Figure 5: The ReducedMPL training procedure has 3 steps: (1) a large teacher q_{large} (red box) is pre-trained; (2) q_{large} assigns class distributions to the student's training data; (3) A small multi-layered perceptron q_Ψ calibrates the distributions computed by q_{large} to train the student. q_Ψ is trained along with the student, like the teacher in normal MPL.

1.1.6 Visual Transformer

Visual Transformer (2019)

The opposite to conventional way idea of Deep learning work with image is to use self-attention based architecture.

The main advantage of this approach is the great ability to pretraining unsupervised the model before traget-task-learning.

More over the transformer model quite simple due to use only multilayer perceptron components.

The transformer block consists in:

- Multi-head self-attention block part (with layer-norm first and skip-connection);

  The meaning of each elf-attention part is to learn

  - MLP-merge part (with layer-norm first and skip-connection);

The ViT work as the follows:

- Split an image into patches

  for image $H \times W \times C$ and patch size p create array $N \times (P \times P \cdot C)$ where $N = \frac{HW}{P^2}$

- Flatten paths to x_p with size $N \times (P^2 \cdot C)$, where $x_p = [x_p^1, \ldots, x_p^N]^T$ and each x_p^i is a vector with length $P^2 \cdot C$.

  for instance for image path [16,16,3] ($p = 16, c = 3$) the results is flattened array of vectors with length $16 \cdot 16 \cdot 3$.

- Make linear projection of each path with trainable weights matrix W_{emb} as $z_i = x_p^i \cdot W_{emb}$, where W_{emb} have dimension $P^2 \cdot C \times D$.

The meaning of this operation is to have the farther fixed dimension.

each z_i is the vector with dimension $1 \times D$. *Note* it is applied one matrix W_{emb} for all paths.

- create an embedding vector $z = [z_0, z_1, \ldots, z_N]$, where z_0 is extra vector of so-called learnable embedding.

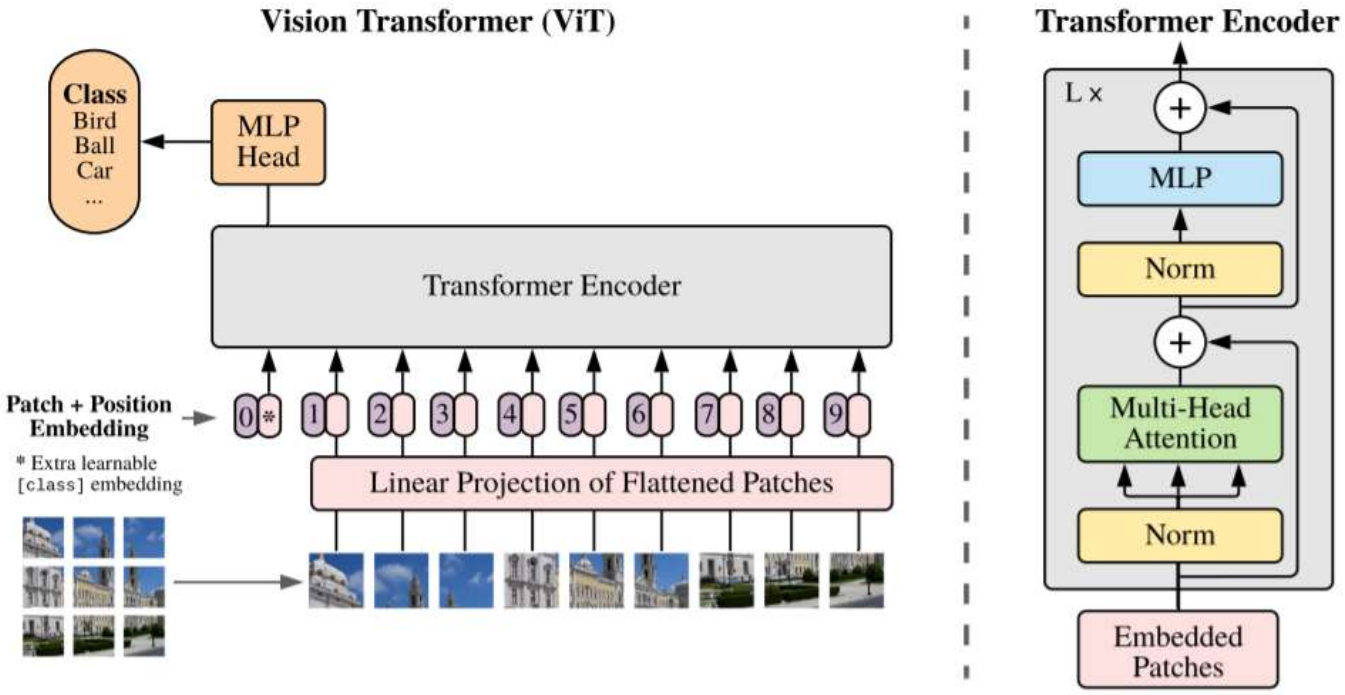
  The meaning of this embedding it is train during pre-training stage as self-supervised, in fine-tune training this vector will be used to predict the class of the input image.

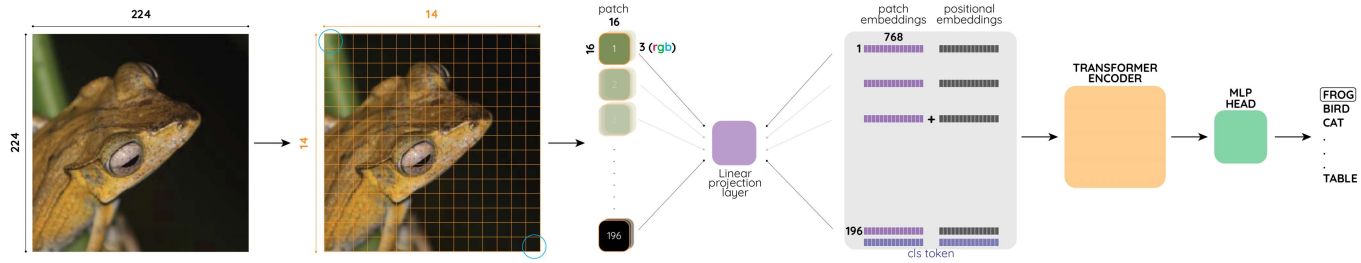
- Add positional encoding $z' = z + E_{pos}$, where E_{pos} has dimension $N + 1 \times D$ and values - the number of raw in each raw.

  The meaning of positional encoding is to mark each path with it number.

- The following feature-extractor part of the model (Transformer Encoder) consist in a several Transformer-blocks.
- After the all block Multilayer perceptron head is used for classification.

Transformer network work well only for large pre-training dataset with then fine-turning on relatively small one.





Several transformer variants in computer vision

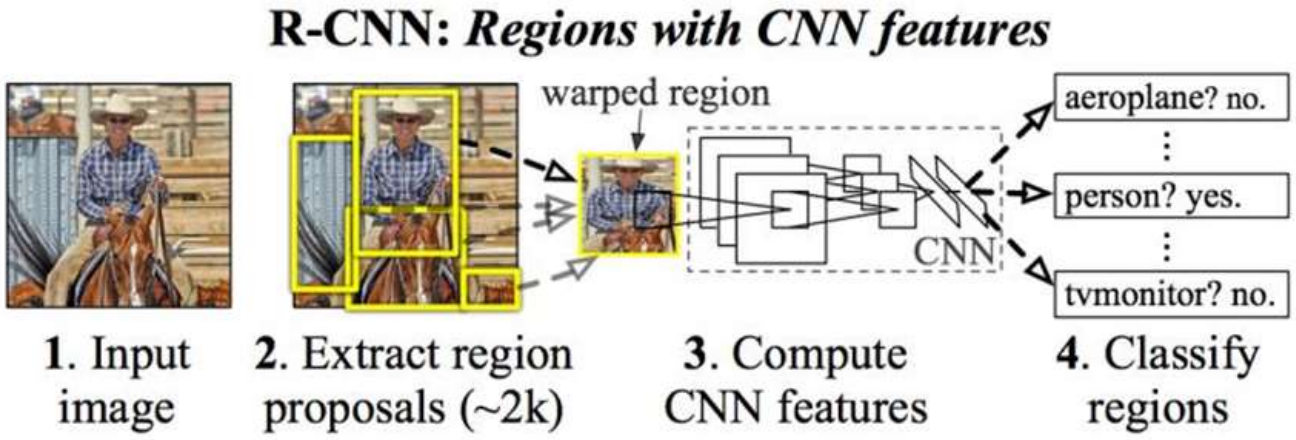
Category	Sub-category	Method	Highlights	Publication
Bakcbone	Image classification	iGPT [21]	Pixel prediction self-supervised learning, GPT model	ICML 2020
		ViT [36]	Image patches, standard transformer	ICLR 2021
High/Mid-level vision	Object detection	DETR [15]	Set-based prediction, bipartite matching, transformer	ECCV 2020
		Deformable DETR [193]	DETR, deformable attention module	ICLR 2021
		ACT [189]	Adaptive clustering transformer	arXiv 2020
		UP-DETR [33]	Unsupervised pre-training, random query patch detection	arXiv 2020
		TSP [143]	New bipartite matching, encoder-only transformer	arXiv 2020
	Segmentation	Max-DeepLab [155]	PQ-style bipartite matching, dual-path transformer	arXiv 2020
		VisTR [159]	Instance sequence matching and segmentation	arXiv 2020
		SETR [190]	sequence-to-sequence prediction, standard transformer	arXiv 2020
	Pose Estimation	Hand-Transformer [67]	Non-autoregressive transformer, 3D point set	ECCV 2020
		HOT-Net [68]	Structured-reference extractor	MM 2020
		METRO [92]	Progressive dimensionality reduction	arXiv 2020
Low-level vision	Image enhancement	IPT [20]	Multi-task, ImageNet pre-training, transformer model	arXiv 2020
		TTSR [167]	Texture transformer, RefSR	CVPR 2020
	Image generation	Image Transformer [113]	Pixel generation using transformer	ICML 2018
Video processing	Video inpainting	STTN [178]	Spatial-temporal adversarial loss	ECCV 2020
	Video captioning	Masked Transformer [191]	Masking network, event proposal	CVPR 2018
Efficient transformer	Decomposition	ASH [103]	Number of heads, importance estimation	NeurIPS 2019
	Distillation	TinyBert [73]	Various losses for different modules	EMNLP Findings 2020
	Quantization	FullyQT [118]	Fully quantized transformer	EMNLP Findings 2020
	Architecture design	ConvBert [72]	Local dependence, dynamic convolution	NeurIPS 2020

The transformers in vision models

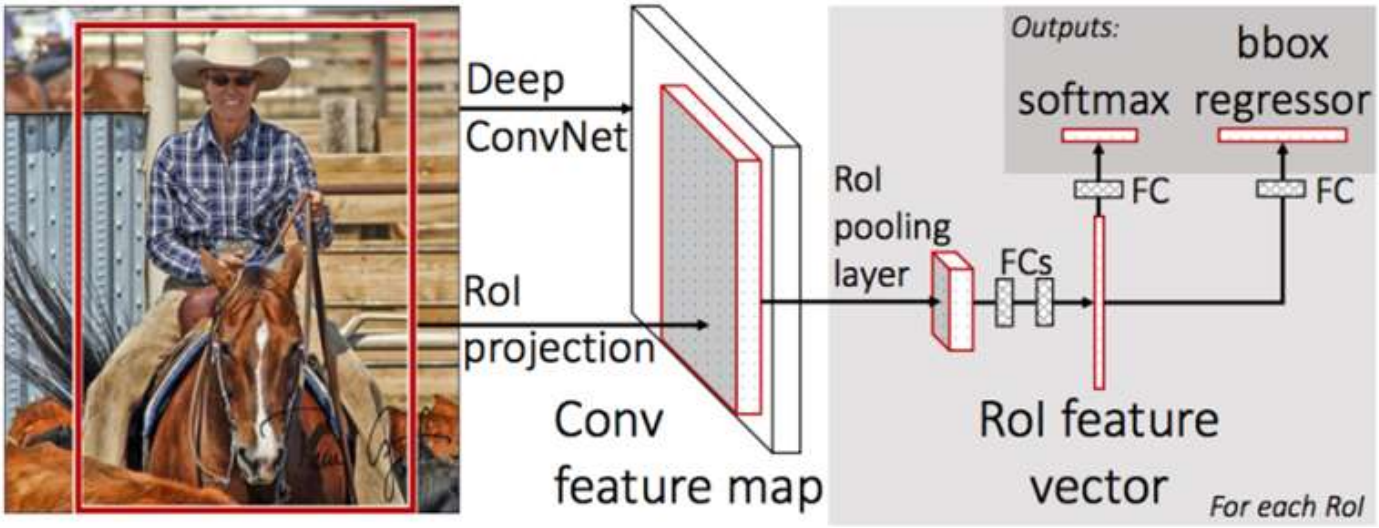
Some State-Of-The art models of image classification

Model name	Number of parameters [Millions]	ImageNet Top 1 Accuracy	Year
AlexNet	60 M	63.3 %	2012
Inception V1	5 M	69.8 %	2014
VGG 16	138 M	74.4 %	2014
VGG 19	144 M	74.5 %	2014
Inception V2	11.2 M	74.8 %	2015
ResNet-50	26 M	77.15 %	2015
ResNet-152	60 M	78.57 %	2015
Inception V3	27 M	78.8 %	2015
DenseNet-121	8 M	74.98 %	2016
DenseNet-264	22M	77.85 %	2016
BiT-L (ResNet)	**928 M**	87.54 %	2019
NoisyStudent EfficientNet-L2	**480 M**	**88.4** %	2020
Meta Pseudo Labels	**480 M**	**90.2** %	2021

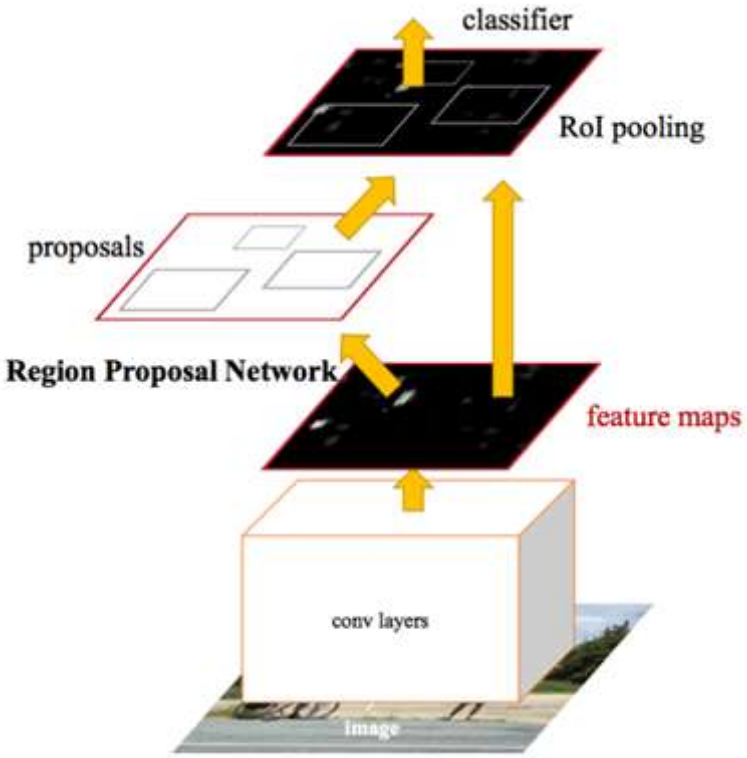
1.2 Other tasks



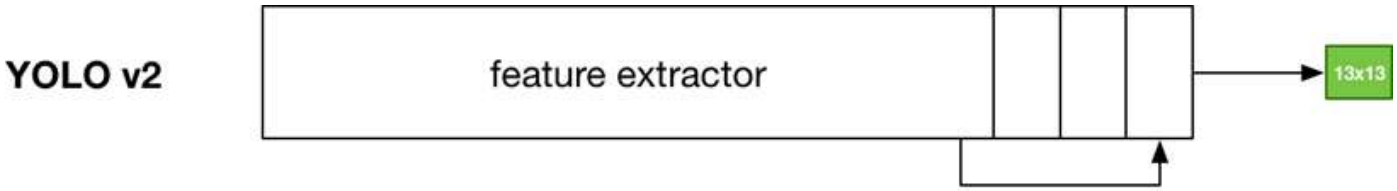
Fast-RCNN

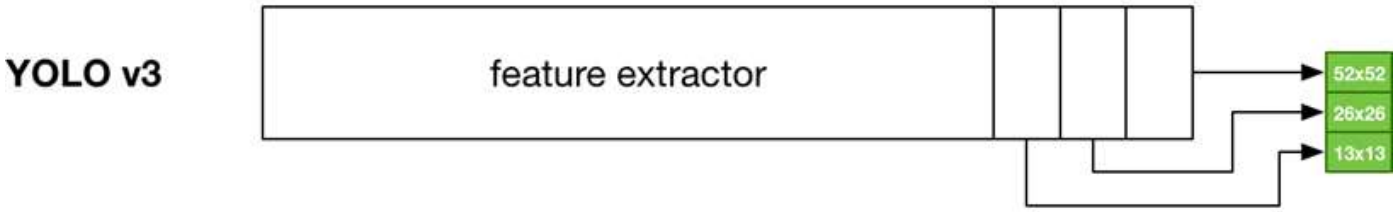


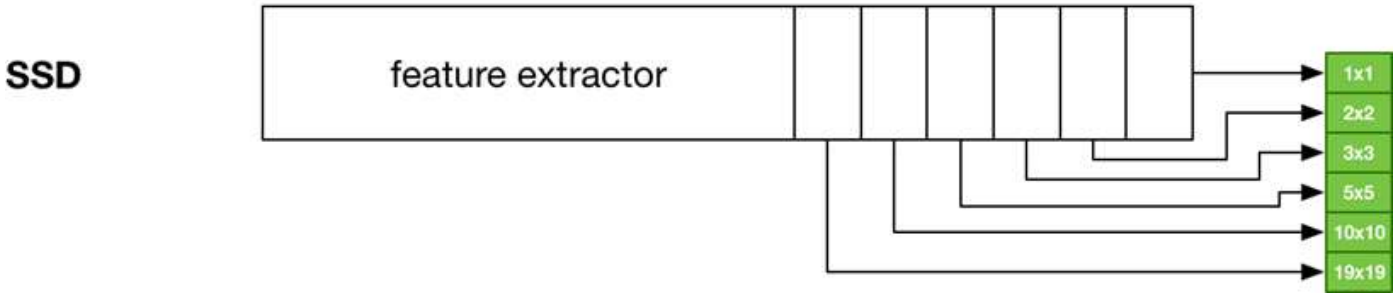
Faster-RCNN



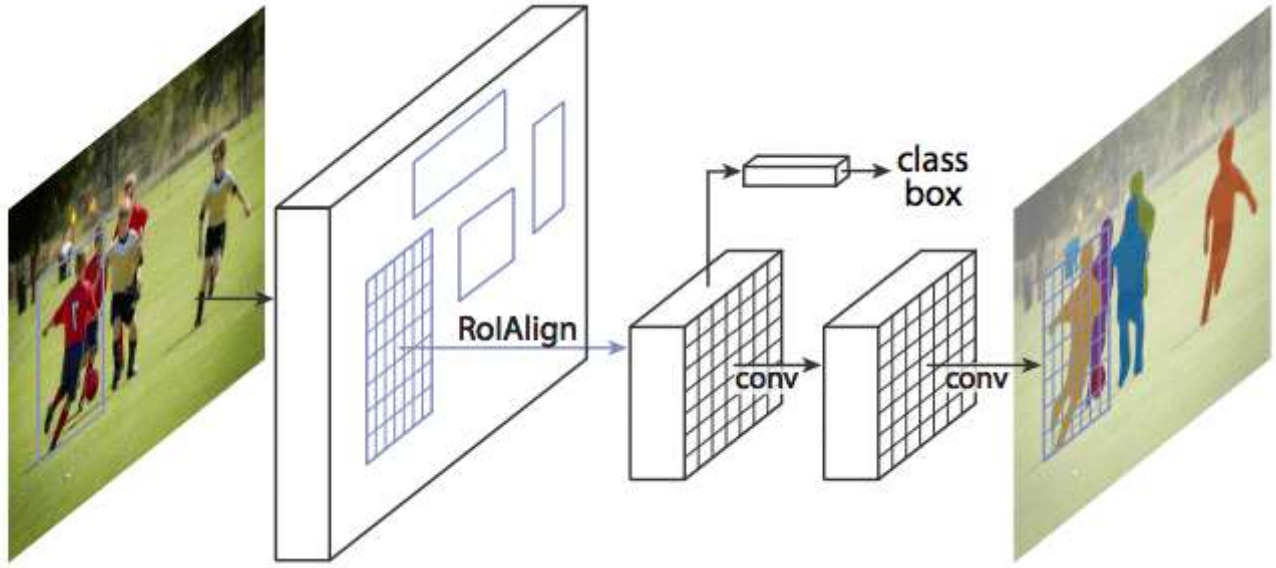
YOLO-one shot detector



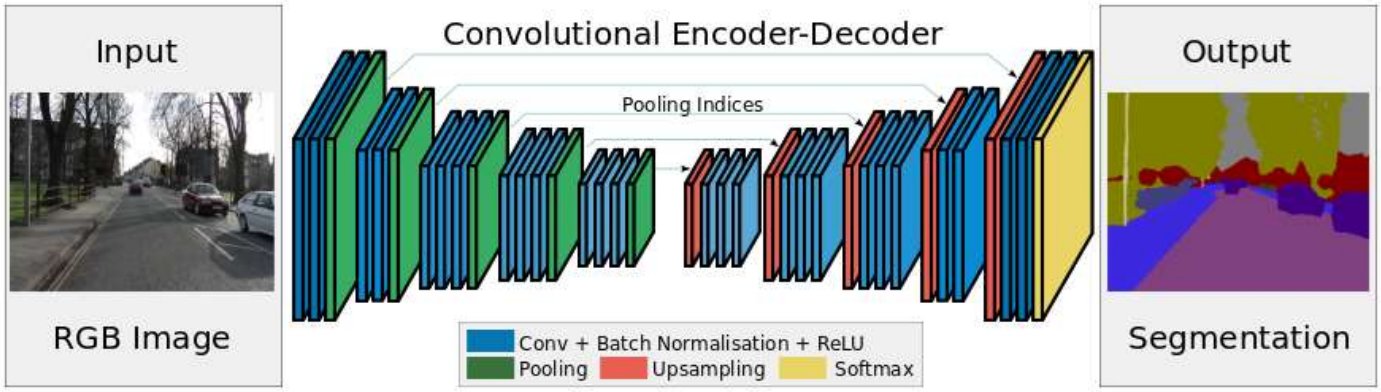




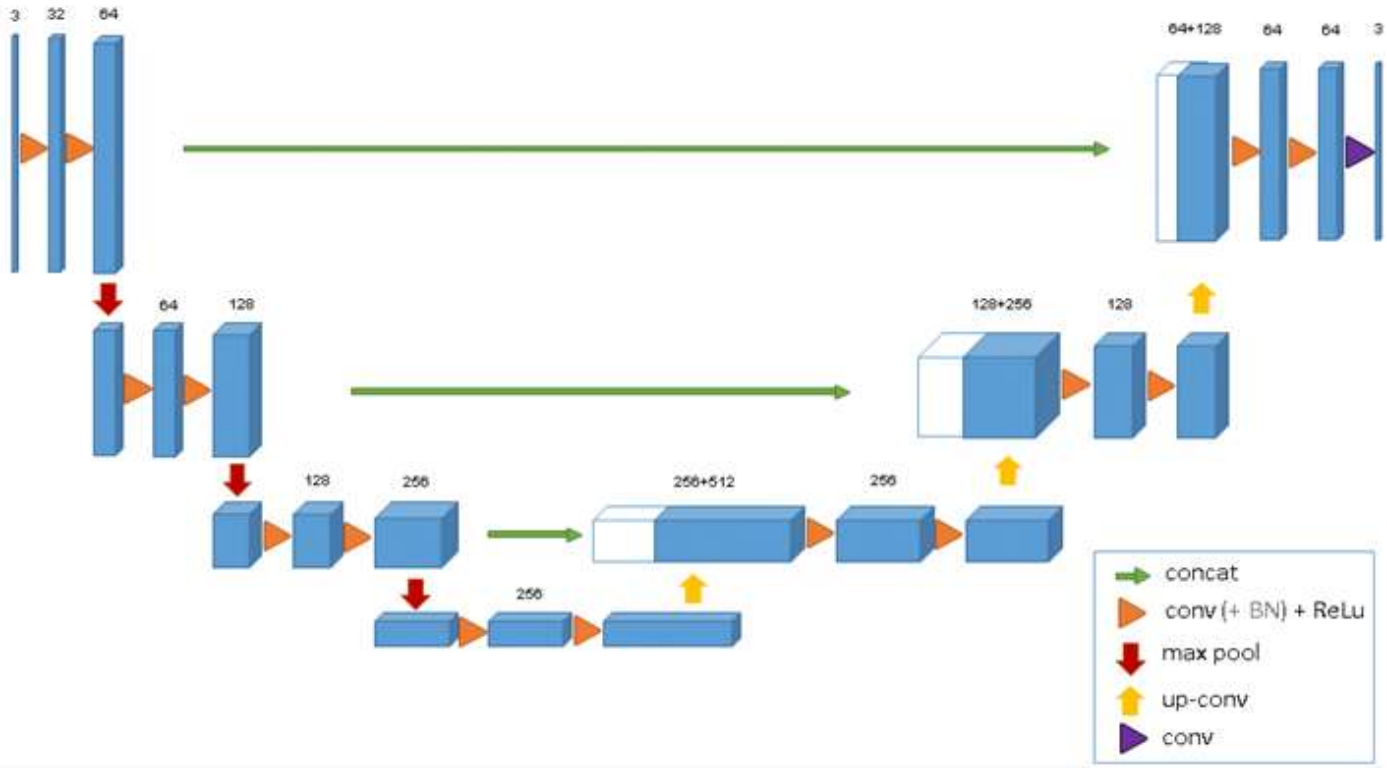
Instance segmentation



Semantic segmentation



U Net Semantic segmentation and Edge Detection



Panoptic Segmentation