# Efficient Models

By Yuhang Song

[This Note is recorded from 7th Sep 2022 to 26th Sep]

## Efficient Methods

From last week`s content, we know that the energy is mainly consumed in DNNs by data transferring(in memory), and MAC(Multiplication & Accumulation) operations. In order to reduce the energy consumption, we need methods or models that reducing the cost of memory and compute in DNNs, that is, the Neural Network Compression.

According to recent paper surveries [0][1][3], we can classifies the current existing efficient compression methods by their different objectives in the following way:

| CLASS | OBJECTIVE | MAJOR METHODS |
|---|---|---|
| Compact Model | Design smaller base models that can still achive acceptable accuracy. | Efficient Lightweight DNN Models |
| Tensor Decomposition | Decompose bloated original tensors into more smaller tensors. | -- |
| Data Quantization | Reduce the number of data bits. | Network Quantization |
| Network Sparsification | Sparse the computational graph(number of connection/neurons). | Network Pruning |
| Knowledge Transferring | Learning output distributions of trained large model. | Knowledge Distillation |
| Architecture Optimazation | Optimaze the model parameters. | NAS, Genetic Algorithm |

# Compact Model

Mordern DNNs` performence impovement is mainly driven by deeper and wider networks with increaseed parameters and operations(MACs)[0], the compact models tends to reduce overhead while maintaining accuracy as much as possible.

Compact models try to design smalleer based models by allowing deeper/wider networks with expanded FMs, more complex branch topology, and more flexible Conv arithmetic. They can be classified based on the following two aspects:

1. Ensure large spatial correlations(Receptive Field) while remains compact.
2. Ensure deeper channel correlations while remains compact.

## Overview on Current DNN Models

I have taken an overview on most of the popular models that are proposed in the DNN field, all of them have different characteristics over network accuracy, speed, and size. Taken the example from the paper[0] which summerize the most frequently used DNN models:

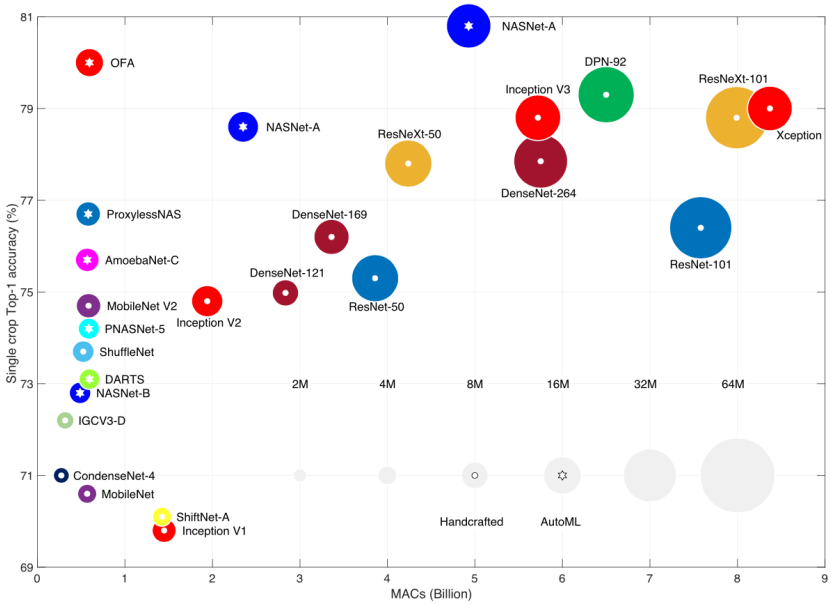



**Fig. 10.** ***Accuracy results of various CNN models on ImageNet with the number of parameters and MACs. Inspired by Canziani et al. [71].***

From above illustrated models, the following DNN models are typically being referred as **Lightweight Models**:

- SqueezeNets
- ShuffleNets
- MobileNets
- GoogleNet (Inception)
- EfficientNets
- NasNets
- DenseNets

...

I will then evaluate each of them by understanding the foundamentals of the idea, implementations, as well as the comparisons & commonalities between them.

# MobileNets

## Main Idea

Paper presented on 2017 by Google [3]. Lightweighted Deep Convolutional Neural Network with drastic number of parameters and calculation operations drop while maintaining reasonable accuracy. Targeting to deploy or train on edge devices with less power consumption, as well as the model size, together with faster inference time (latency).

The main idea used in MobileNet is construct relatively compact deep neural networks with use a form of factorized convolutions, namely **Depthwise Separable Convolution**, to enable both spitial correlations and channel correlations.

Different from the conventional convolution operations, Depthwise Separable Convolution is performed as follows:

1. [Depthwise Convolution] Apply one single filter to each channel of input featuree map. Ensures the spatial correlation.
2. [Pointwise Convolution] Apply filter of size 1x1 to the combined output of the Depthwise Convolution. Ensures the channel correlation.

Depthwise Separable Convolution performs convolution channel by channel, then combines the resulting tensors by using Nx1x1x(#Channel) fileters to produce output. Saving dramastic parameters

comparing with the standard method.

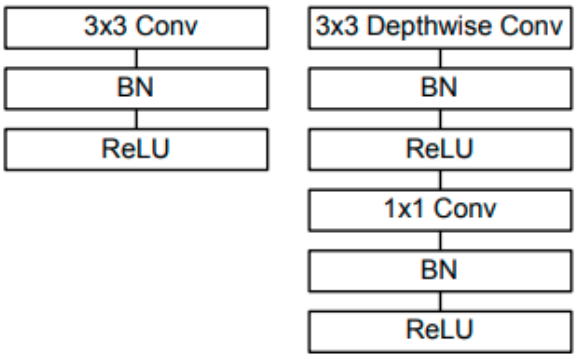



Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

As shown above, treat the Depthwise Seperable Convolution as two separate layers, both Depthwise Convolution and Pointwise Convolution is followed by a batch normalization layer and Relu nonlinearity.

The whole architecture proposed by the paper is as following :

Table 1. MobileNet Body Architecture

| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | $3 \times 3 \times 3 \times 32$ | $224 \times 224 \times 3$ |
| Conv dw / s1 | $3 \times 3 \times 32$ dw | $112 \times 112 \times 32$ |
| Conv / s1 | $1 \times 1 \times 32 \times 64$ | $112 \times 112 \times 32$ |
| Conv dw / s2 | $3 \times 3 \times 64$ dw | $112 \times 112 \times 64$ |
| Conv / s1 | $1 \times 1 \times 64 \times 128$ | $56 \times 56 \times 64$ |
| Conv dw / s1 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 128$ | $56 \times 56 \times 128$ |
| Conv dw / s2 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 256$ | $28 \times 28 \times 128$ |
| Conv dw / s1 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 256$ | $28 \times 28 \times 256$ |
| Conv dw / s2 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 512$ | $14 \times 14 \times 256$ |
| $5\times$ Conv dw / s1 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| $5\times$ Conv / s1 | $1 \times 1 \times 512 \times 512$ | $14 \times 14 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| Conv / s1 | $1 \times 1 \times 512 \times 1024$ | $7 \times 7 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 1024$ dw | $7 \times 7 \times 1024$ |
| Conv / s1 | $1 \times 1 \times 1024 \times 1024$ | $7 \times 7 \times 1024$ |
| Avg Pool / s1 | Pool $7 \times 7$ | $7 \times 7 \times 1024$ |
| FC / s1 | $1024 \times 1000$ | $1 \times 1 \times 1024$ |
| Softmax / s1 | Classifier | $1 \times 1 \times 1000$ |

Table 2. Resource Per Layer Type

| Type | Mult-Adds | Parameters |
|---|---|---|
| Conv $1 \times 1$ | 94.86% | 74.59% |
| Conv DW $3 \times 3$ | 3.06% | 1.06% |
| Conv $3 \times 3$ | 1.19% | 0.02% |
| Fully Connected | 0.18% | 24.33% |

By using this architecture, the paper also concludes the resources used by each layer type, the most calculations and parameters are from 1x1 pointwise convolutions, as shown in table 2.

Additionally, the paper subsequently introduced 2 multipliers to allow tradeoffs between model size, latency, and its accurary.

- Width Multiplier: Used to thinner the models uniformly at each layer, denoted by $\alpha \in (0, 1]$. Denote the number of input channel as $M$ and number of output channel as $N$, after applying this multiplier they become $\alpha M$ and $\alpha N$.
- Resolution Multiplier: Used to reduce the size of the input image and internal feature maps uniformly in each layer, denoted by $\rho \in (0, 1]$. Denote the size of feature maps as $D_F$, after applying this multiplier they become $\rho D_F$.

### Evaluations

Denotions Table: $D_K$ (Kearnel Size); $D_F$ (Input Feature Map Size); $M$ (Number of Input Channels); $N$ (Number of Output Channels).

A standard convolution method has the cost of:

$(D_K)^2 * M * N * (D_F)^2$

By using depthwise separable convolution, the cost is:

$(D_K)^2 * (D_F)^2 * M + M * N * (D_F)^2$

By using depthwise seeparable convolution, we get a reduction in computation of:

$$\frac{(D_K)^2 * (D_F)^2 * M + M * N * (D_F)^2}{(D_K)^2 * M * N * (D_F)^2} = \frac{1}{N} + \frac{1}{D_K^2}$$

As for the accuracies comparison, the same MobileNet architecture uses standard convolution and the one uses deepthwise separable comvolution is given by the following:

Table 4. Depthwise Separable vs Full Convolution MobileNet

| Model | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| Conv MobileNet | 71.7% | 4866 | 29.3 |
| MobileNet | 70.6% | 569 | 4.2 |

MobileNet using depthwise separable convolution loses only 1% of the accuracy while keeping 8-9 times less MAC operations and parameters. The paper also compared MobileNet to other widely used models:

Table 8. MobileNet Comparison to Popular Models

| Model | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| 1.0 MobileNet-224 | 70.6% | 569 | 4.2 |
| GoogleNet | 69.8% | 1550 | 6.8 |
| VGG 16 | 71.5% | 15300 | 138 |

Table 9. Smaller MobileNet Comparison to Popular Models

| Model | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| 0.50 MobileNet-160 | 60.2% | 76 | 1.32 |
| Squeezenet | 57.5% | 1700 | 1.25 |
| AlexNet | 57.2% | 720 | 60 |

The paper also indicates that, by expirements, the width multiplier $\alpha$ should be $> 0.25$ to maintain good performance and the resolution of the input image should be greater than $128$, otherwise the accuracy could be droping below $60\%$.

### Question

- As also been discussed and strongly recommanded in [0], large amount of current existing compact models are using pointwise convolution, including MobileNet. Pointwise convolution combined with depthwise convolution is relatively efficient and saving comparing to general convolution, yet it is responsable to most of the complexity comparing with other layers, as we can see in MobileNet. Can we reduce its complexity?

## MobileNet V2

### Theory

Intuitively, the computation & memory cost of DNN models depend largely on the input resolutions and the number of tensors channels of the inner computing graph. Conventional convolution methods extracts feature information by internal Conv kernels that map the input into higher dimensions(channels) to allow the original image to form a "manifold of interest"[3] we wish to learn, by set of those inner-layer activations. Since it has been long assumed that manifolds of interest in DNNs could be embedded in low-dimensional subspaces, in order to design compact models, we can reduce the operation space by simply reducing the dimensionality of the layers, while remaining the shape of the "manifold of interest".

MobileNet V1 uses width multiplier/resolution multipliers[2] to allow tradeoffs between computation and accuracy. Following above intuition, those multipliers allows one to reduce the dimensionality until the manifold of interests spans the entire space[3].

However, due to the non-linear property of Conv layers, the "manifold of interests" may collapse. The paper[3] gives an intuitive illustration of this idea by embedding low-dimensional manifolds into n-dimensional spaces:

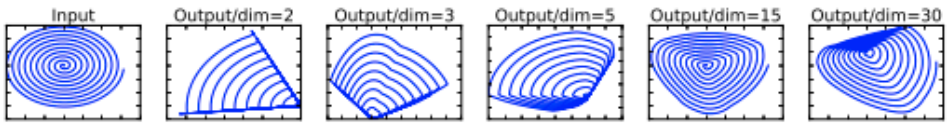



Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an $n$-dimensional space using random matrix $T$ followed by ReLU, and then projected back to the 2D space using $T^{-1}$. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

It is can be observed that the information losses a lot when low-dimentional(e.g. when n=2,3) manifold is transformed with $ReLU$; whereas the information mostly reserved when high-dimentional manifold is transformed with $ReLU$.

This result suggesting use more dimensionalities as possible, and it is conflicting with the idea of lightweighted models including MobileNet V1.

MobileNet V2 is trying to address this problem, preventing the information loss, as well as remaining even lower parameters/memory usage.

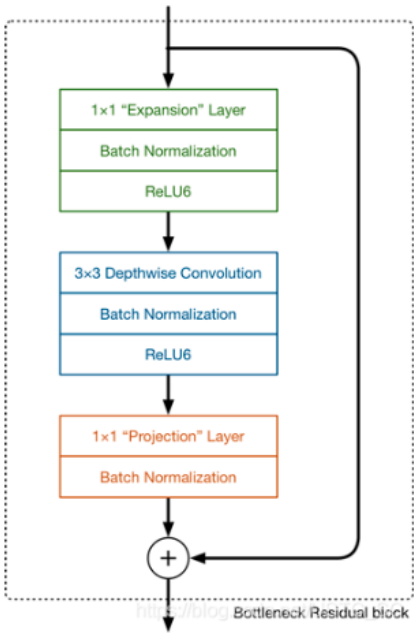
### Main Idea

The insights used for MobileNet V2 architecture can be views as the follows:

- **Depthwise Separable Convolution** - Saving parameter and operation complexities.
- **Linear Bottlenecks** - Using the partial non-linearity property of $ReLU$ function, insert a linear transformation after high-dimensional $ReLU$ transformation to address the conflict of complexity and manifold collapsion.
- **Inverted Residuals** - More memory efficient to shortcut connections between linear layers. (When stride=2 this can be optional according to the paper.)

This design is named the **Bottleneck Residual Block**, with a 1x1 filter performing transformation to increase the dimensionality of input activated by $ReLU6$, then perform 3x3 depthwise convolution, subsequently using another 1x1 filter to perform linear transformation to produce the output, reduce the dimensions as well as preventing lose of information. Lastly, add shortcut connections between consecutive linear layers to improve the ability of a gradient to propagate across layers. Each layer is followed by a batnormalization.

As illustrated in below:




| Input | Operator | Output |
|---|---|---|
| $h \times w \times k$ | 1x1 conv2d , ReLU6 | $h \times w \times (tk)$ |
| $h \times w \times tk$ | 3x3 dwise s=$s$, ReLU6 | $\frac{h}{s} \times \frac{w}{s} \times (tk)$ |
| $\frac{h}{s} \times \frac{w}{s} \times tk$ | linear 1x1 conv2d | $\frac{h}{s} \times \frac{w}{s} \times k'$ |

Table 1: *Bottleneck residual block* transforming from $k$ to $k'$ channels, with stride $s$, and expansion factor $t$.

The whole architecture of the proposed MobileNet V2 is given by the following:

| Input | Operator | $t$ | $c$ | $n$ | $s$ |
|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 32 | 1 | 2 |
| $112^2 \times 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 \times 16$ | bottleneck | 6 | 24 | 2 | 2 |
| $56^2 \times 24$ | bottleneck | 6 | 32 | 3 | 2 |
| $28^2 \times 32$ | bottleneck | 6 | 64 | 4 | 2 |
| $14^2 \times 64$ | bottleneck | 6 | 96 | 3 | 1 |
| $14^2 \times 96$ | bottleneck | 6 | 160 | 3 | 2 |
| $7^2 \times 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 \times 320$ | conv2d 1x1 | - | 1280 | 1 | 1 |
| $7^2 \times 1280$ | avgpool 7x7 | - | - | 1 | - |
| $1 \times 1 \times 1280$ | conv2d 1x1 | - | k | - | |

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated $n$ times. All layers in the same sequence have the same number $c$ of output channels. The first layer of each sequence has a stride $s$ and all others use stride 1. All spatial convolutions use $3 \times 3$ kernels. The expansion factor $t$ is always applied to the input size as described in Table 1.

## Evaluation

Note that despite V2 has more deeper network than V1, its the computation cost and accuracy is better that V1, since the network`s nature(BottleNeck Depthwise Conv) allows smaller input and output dimensions.

The paper experiments the performance of MobileNet V2 over MobileNet V1, ShuffleNet, and NAS method on image classification task, the result is given as the following:

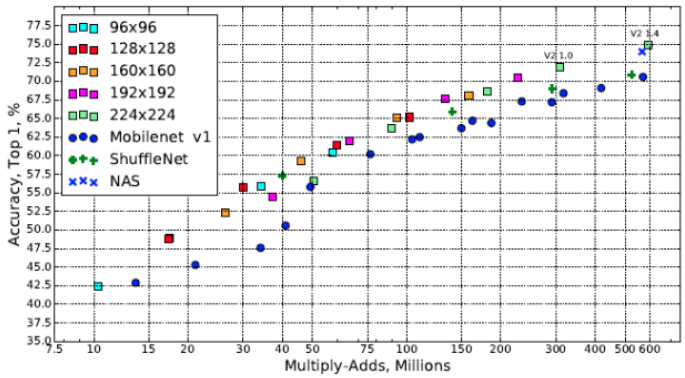



Figure 5: Performance curve of MobileNetV2 vs MobileNetV1, ShuffleNet, NAS. For our networks we use multipliers 0.35, 0.5, 0.75, 1.0 for all resolutions, and additional 1.4 for for 224. Best viewed in color.

The result suggests that with even resolution of 96x96, MobileNet V2 produces acceptable result than MobileNet ($\geq 60\%$) with about $10\%$ less parameters.

## MobileNet V3

### Main Idea & Background Knowledge

Proposed by Google in 2019. The MobileNet V3 achives better performance, with $3.2\%$ more accurate on ImageNet classification while reducing latency by $20\%$, in comparison to previous MobileNet V2. It is mainly constructed from the following ideas:

- Bottleneck Block with SENet block introduced.
- 2 Neural Architecture Search algorithms are used, for block optimization and layer optimization.
- Redesigned some of the redundent expensive structures.
- New Nonlinearities.

In order to understand the design details of MobileNet V3, we need to firstly take a look at Mnasnet[5] & Netadapt[6] & SENet[8].

---

**Mnasnet: Platform-Aware NAS for Mobile [5]**

---

Proposed by Google in 2019, Mnasnet is an Neural Architecture Search guilded auto designed model, its main conribution is proposed that the conventional FLOPS proxy used widely in NAS methods are not the accurate approximation of the model lantency, it novelly used the real lantency measuremeants from running model on physical devices and thus guild the design.

The following figure is an overview of Platform-Aware Neural Architecture Search method for mobile:

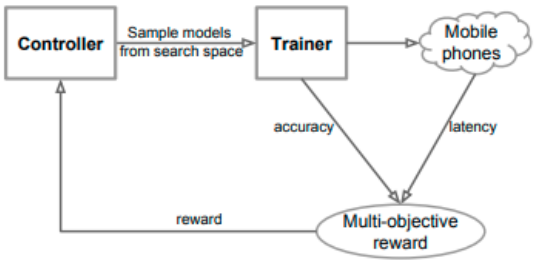



Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile.**

The RNN controller firstly sample model parameters from the search space, then the trainer excutes with selected parameters, and accordingly evaluate the accuracy of the current model, the model is

subsequently passed into real mobile phones to obtain the real lantency, together with model accuracy and latency, compute the manually definded reward and feedback to the controller.

The objective function for Plateform-Aware NAS is definded as:

$$\begin{aligned} \max_{m} \quad & ACC(m) \\ w.r.t. \quad & LAT(m) \leq T \end{aligned}$$

However, given the computational cost of performing NAS, we are more interested in finding multiple Pareto-optimal solutions in a single architecture search, instead of maximize a single metric.

Plateform-Aware NAS uses a customized weighted product method to approximate Pareto optimal solutions, the goal is defined to:

$$\max_{m} \quad ACC(m) \times [\frac{LAT(m)}{T}]^{w}$$

In which the $w$ is the weight factor defined as:

$$w = \begin{cases} \alpha & LAT(m) \leq T \\ \beta & otherwise \end{cases}$$

Empirically, the paper suggests that doubling the latency usually brings about 5% higher accuracy gain, by idea of obtaining Pareto-optimal, given two models, (1) M1 has latency $l$ and accuracy $a$, (2) M2 has latency $2l$ and accuracy $1.05a$, they should have similar reward, definded as follows:

$$Reward(M1) = a \cdot (\frac{l}{T})^{\beta} \approx Reward(M2) = a \cdot (1 + 5\%) \cdot (\frac{2l}{T})^{\beta}$$

Solving for above equation gives $\beta \approx -0.07$, Plateform-Aware NAS uses $\alpha = \beta = -0.07$ in their experiments unless explicitly stated.

In additional, Factorized Hierarchical Search Space is used to ensure the diersity of models, the baseline search structure is shown in below:

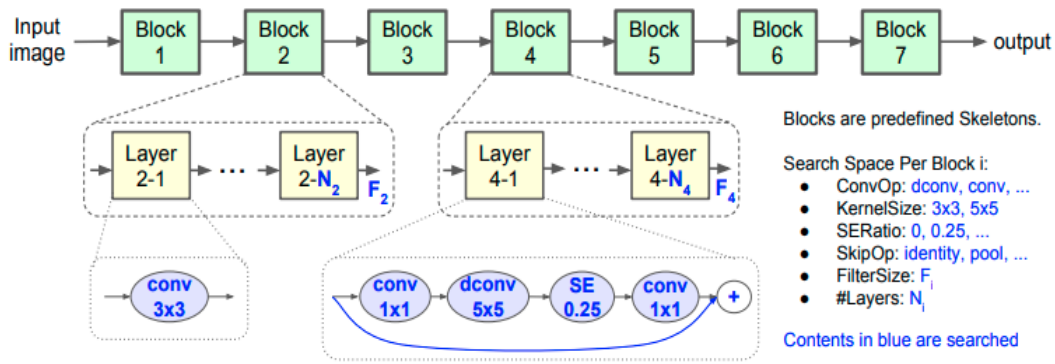



Figure 4: **Factorized Hierarchical Search Space.** Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output resolutions are different but all other layers have stride 1. For each block, we search for the operations and connections for a single layer and the number of layers $N$, then the same layer is repeated $N$ times (e.g., Layer 4-1 to 4-$N_4$ are the same). Layers from different blocks (e.g., Layer 2-1 and 4-1) can be different.

Use Reinforcement Learning approach to find Pareto optimal solution for multi-objective search problem.

---

**Netadapt: Platform-aware neural network adaptation for mobile applications [6]**

---

Proposed by Google in 2018. Netadapt is an layer by layer network compression algorithm, which alters the number of filters in each layer to obtain given resource budget, for a certain pre-trained

model.

It is propsed in the pape, that similar to [5], the traditional measurements of #parameters and #MACs might not be sufficient to conclude the latency and energy consumption. Netadapt also uses the direct metric to guild the filters pruning.

One main difference between Netadapt filters pruning and energy-aware pruning[7] is that Netadapt uses empirical matric per layer to estimate the real resouce consumption, so that no further detailed lower-level knowledge is required for estimating the real matrics.

The problem can be formulated as the following:

$$\max_{Net} \quad Acc(Net)$$
$$w.r.t. \quad Res_j(Net_i) \leq Bud_j,\ j\ =\ 1, ..., m$$

Whereas in Netadapt, considering maintaining the accuracy needs re-train for each alternation step, it breaks above problem into following series of easier problems and solves it iteratively:

$$\max_{Net_i} \quad Acc(Net_i)$$
$$w.r.t. \quad Res_j(Net_i) \leq Res_j(Net_{i-1}) - \Delta R_{i,j},\ j\ =\ 1, ..., m$$

Where $\Delta R_{i,j}$ is called "Resource Reduction Schedule", similar to the concept of learning rate schedule, It is an hyper-parameter stands for the reduction step size for each iteration.

The full algorithm of the proposed method is shown in below:

**Algorithm 1:** NetAdapt

**Input:** Pretrained Network: $Net_0$ (with $K$ CONV and FC layers), Resource Budget: $Bud$, Resource Reduction Schedule: $\Delta R_i$
**Output:** Adapted Network Meeting the Resource Budget: $\hat{Net}$

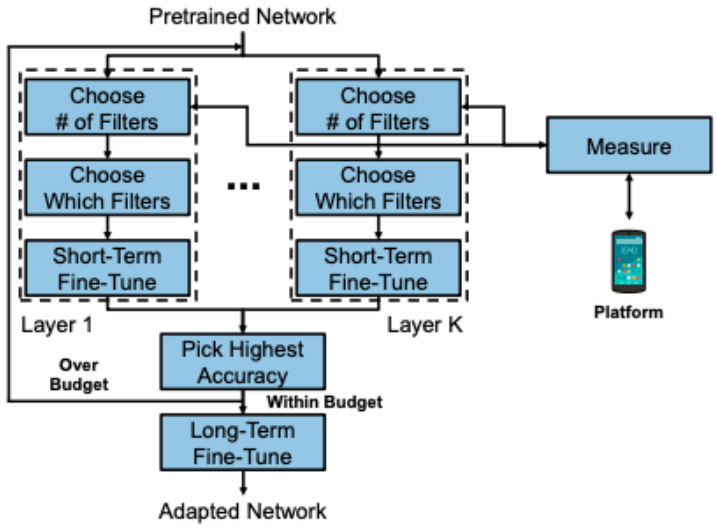
```
i = 0;
Res_i = TakeEmpiricalMeasurement(Net_i);
while Res_i > Bud do
    Con = Res_i - ΔR_i;
    for k from 1 to K do
        /* TakeEmpiricalMeasurement is also called inside
           ChooseNumFilters for choosing the correct number of filters
           that satisfies the constraint (i.e., current budget). */
        N_Filt_k, Res_Simp_k = ChooseNumFilters(Net_i, k, Con);
        Net_Simp_k = ChooseWhichFilters(Net_i, k, N_Filt_k);
        Net_Simp_k = ShortTermFineTune(Net_Simp_k);
    Net_{i+1}, Res_{i+1} = PickHighestAccuracy(Net_Simp_:, Res_Simp_:);
    i = i + 1;
N̂et = LongTermFineTune(Net_i);
return N̂et;
```

Where the algorithm iteratively find solutions that within current resource budget in each iteration, layer-by-layer, perform ShortTermFineTune after each filter-pruned layer. Each time a layer is pruned, a new network with only that specific layer is pruned is generated and stored.

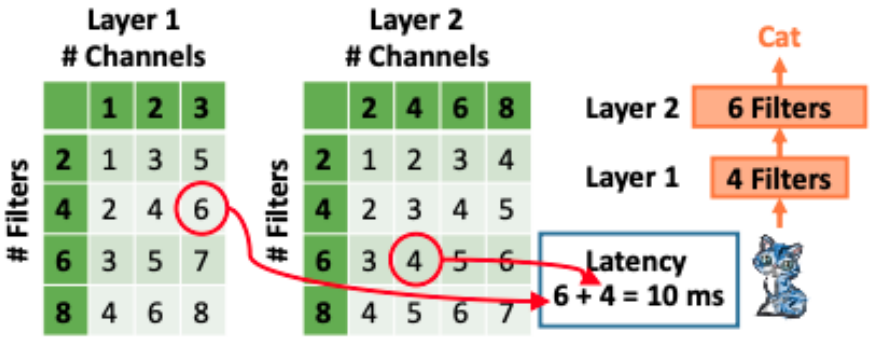
After all the layers have been evaluated, select one network with the highest accuracy from the stored $K$ (Equal to number of layers) networks.

Repeat the process until the budget limitation is satisified.

More intuitive explanations of the process is shown in below:




Since it is epensive to estimate the real matrics for resources consumption for the whole network, the paper propsed a method using layer-wise look-up tables for fast resource consumption estimation, obtained from the empirical data.




---

**Squeeze-and-Excitation Networks [8]**

---

The paper introduced a method to emphasis the channel-wise features and their correlations by "Attention" mechanism. In which for a convolution output $U = X * F_{tr}$, where $X$ is the input and $F_{tr}$ is the standard convolution operation, we can plug in behind an SE block to boost feature discriminability.

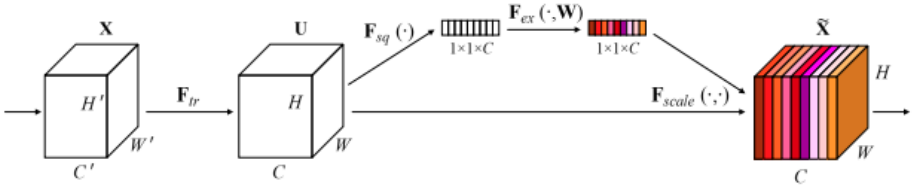



Figure 1: A Squeeze-and-Excitation block.

As illustrated above, a Squeeze-and-Excitation block consists of three steps:

- $F_{sq}(\cdot)$ : **Squeeze - channel wise average pooling operation**. Formally, the $c$-th element of output $z$ is defined as: $z_c = F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j)$.
- $F_{ex}(\cdot, W)$ : **Excitation - fully connected nonlinear activation**. Formally, the ouput $s$ is defined as: $s = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_2 \delta(W_1, z))$, where $\sigma$ is $Sigmiod$ activation, and $\delta$ is $ReLU$ activation.
- $F_{scale}(\cdot,\cdot)$ : **Scalling - channel wise factor multiplication**. Multiply the output from previous layer of the "importance scores" for each layer to thmselves.

Such design won the first place in the ILSVRC 2017 classification competition with top performing model ensemble achieves a $2.251\%$ top-5 error on the test set, that is $\approx 25\%$ relative improvement compared with previous winner.

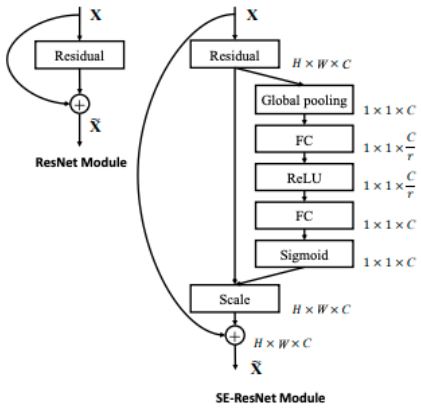



Figure 3: The schema of the original Residual module (left) and the SE-ResNet module (right).

The paper proposed that SE block can be used in any state-of-art DNN architectures, the figure shown above is an example of ResNet integrated with SE block, this is called SE-ResNet.

**Question:** Would it be sufficient to pooling channel wise spatial information by just 1 value? What if a input of large resolution, would it be better to use more values (a 2x2 descriptor for example) ?

---

Now back to MobileNet V3. There are two types of MobileNet V3 models according to the paper, the MobileNetV3-Large & MobileNetV3-Small, targeted at high and low resource use cases respectively.

## Block Structure

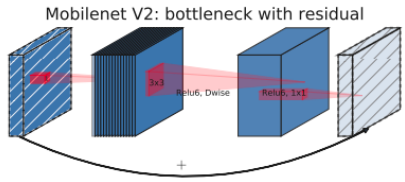



Figure 3. MobileNetV2 [39] layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have nonlinearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck (rather than expansion).

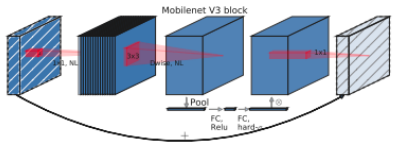



Figure 4. MobileNetV2 + Squeeze-and-Excite [20]. In contrast with [20] we apply the squeeze and excite in the residual layer. We use different nonlinearity depending on the layer, see section 5.2 for details.

The MobileNet V3 integrates lightweight attention module **Squeeze and Excitation** into its bottleneck block structure, where the SE block is placed after depthwise convolution inthe expansion for attention to be applied on the largest representation. As can be observed from above. Furthermore, MobileNet V3 uses compression rate as $\frac{1}{4}$ in fully connected layers, by expiriments, doing so increases the accuracy at the modest incrase of number of parameters, and with no discernible latency cost.

## Network Search

1 - Block-wise Search

MobileNet V3 used **Platform-aware Neural Architecture Search** approach for large mobile models, and found the similar results as in [5], therefore MobileNet V3 simply reuse the same MnasNet-A1 as initial large mobile model.

However, by observations that small mobile is more lantency-sensitive, in other words the model accuracy changes more dramatically with lantency for small models, the original assumption made in [5] for the empirical accuracy-lantency rate might not be suitable. Therefore, MobileNetV3-Small uses weight factor $w = -0.15$ insteead of $w = -0.07$.

2 - Layer-wise Search

After the rough blocks architecture is defined, MobileNet V3 then uses **NetAdapt** as complimentary to search for optimal individual layer configurations in a sequential manner, rather than trying to infer coarse but global architecture.

MobileNet V3 has modifed the algorithm by selecting the final proposals by one that maximize $\frac{\Delta Acc}{|\Delta Latency|}$, in which $\Delta Latency$ satisifies reduction schedule $\Delta R$. The intuition is that because our proposals are discrete, we prefer proposals that maximize the slope of the trade-off curve.

By setting $\Delta R = 0.01|L|$ and $T = 10000$, where $L$ is the lantency of the original model, and $T$ is the number of iterations for excuting NetAdapt algorithm. Like did in [6], the proposals for MobileNet V3 are allowed from the following two types of altering:

- Reduce the size of any expansion layer;
- Reduce bottleneck in all blocks that share the same bottleneck size to maintain residual connections.

### Redesigning Expensive Layers

The current model based on MobileNet V2 uses 1x1 pointwise convolution as the final layer for dimension expansion, and then after Avg Pooling + anotehr 1x1 convolution, reduce the both channel size and feature map size to prodece output. The final dimension expansion layer is important as it ensures rich features for prediction, however, this is lantency-expensive.

MobileNet V3 moves this layer past the final average pooling. This makes the computation of the features becomes nearly free in terms of computation and latency.

By moving this layer after average pooling, the previous projection layer is no longer needed to reduce the computation, thus the projection layer and its 3x3 depth filtering layer(Replaced by Average Pooling) of the last bottleneck block are all removed. Full structure is shown in below:

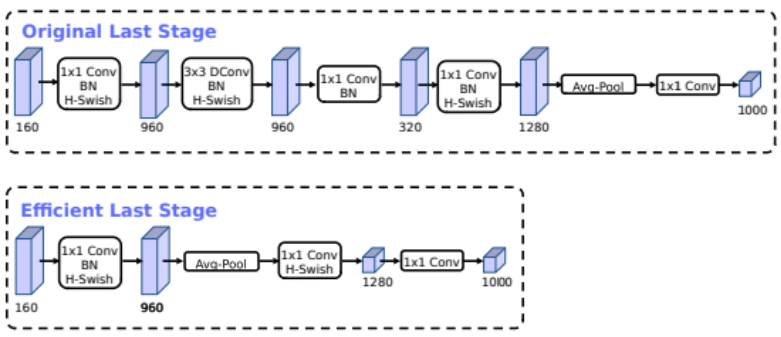



Figure 5. Comparison of original last stage and efficient last stage. This more efficient last stage is able to drop three expensive layers at the end of the network at no loss of accuracy.

This design choise reduces latency by $11\%$ of running time, that is $7ms$, and reduces $30$ $millions$ MAdds with almost no loss of accuracy.

Another improvement is that for the vary first convolutions, current mobile models tend to use 32 filters in a full 3x3 convolution to build initial filter banks for edge detection, often these filters are mirror images of each other, MobileNet V3 novelly used 16 filters instead while maintaining the same accuracy as 32 filters, but with additional $2\ millisecounds$ and $10\ million MAdds$ saving,

### Nonlinearities

The piece-wise linear hard analog of $Sigmoid$ and $Swish$ activations is introduces in MobileNet V3, formally:

$$hard-Sigmoid[x] = \frac{ReLU6(x+3)}{6}$$
$$and$$
$$hard-Swish[x] = x\frac{ReLU6(x+3)}{6}$$

It is found by the expiriment that those hard functions have no discernible differences than the soft ones in accuracy, but they are:

1. More friendly in quantized mode since it eliminates potential numerical precision loss caused by different implementations of the approximate sigmoid.
2. h-swish can be implemented as a pice-wise function to reduce the number of memory acesses driving the latency cost down substantially.

In addition, the paper also mentioned that because the cost of applying nonlinearity decreases as we go deeper into the network, due to the reductions of resolution size which halves each layer activation memories everytime, MobileNet V3 only uses h-swish functions in deeper layers(Secound half of the layers). Even with this, the h-swish function still introduce some latency cost, this can be addressed by using optimized implementation based on a piece-wise function.

### Architectures

The complete architecture designs for MobileNetV3-Large and MobileNetV3-Small is given in below:

| Input | Operator | exp size | #out | SE | NL | s |
|---|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 16 | - | HS | 2 |
| $112^2 \times 16$ | bneck, 3x3 | 16 | 16 | - | RE | 1 |
| $112^2 \times 16$ | bneck, 3x3 | 64 | 24 | - | RE | 2 |
| $56^2 \times 24$ | bneck, 3x3 | 72 | 24 | - | RE | 1 |
| $56^2 \times 24$ | bneck, 5x5 | 72 | 40 | ✓ | RE | 2 |
| $28^2 \times 40$ | bneck, 5x5 | 120 | 40 | ✓ | RE | 1 |
| $28^2 \times 40$ | bneck, 5x5 | 120 | 40 | ✓ | RE | 1 |
| $28^2 \times 40$ | bneck, 3x3 | 240 | 80 | - | HS | 2 |
| $14^2 \times 80$ | bneck, 3x3 | 200 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 184 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 184 | 80 | - | HS | 1 |
| $14^2 \times 80$ | bneck, 3x3 | 480 | 112 | ✓ | HS | 1 |
| $14^2 \times 112$ | bneck, 3x3 | 672 | 112 | ✓ | HS | 1 |
| $14^2 \times 112$ | bneck, 5x5 | 672 | 160 | ✓ | HS | 2 |
| $7^2 \times 160$ | bneck, 5x5 | 960 | 160 | ✓ | HS | 1 |
| $7^2 \times 160$ | bneck, 5x5 | 960 | 160 | ✓ | HS | 1 |
| $7^2 \times 160$ | conv2d, 1x1 | - | 960 | - | HS | 1 |
| $7^2 \times 960$ | pool, 7x7 | - | - | - | - | 1 |
| $1^2 \times 960$ | conv2d 1x1, NBN | - | 1280 | - | HS | 1 |
| $1^2 \times 1280$ | conv2d 1x1, NBN | - | k | - | - | 1 |

Table 1. Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of nonlinearity used. Here, HS denotes h-swish and RE denotes ReLU. NBN denotes no batch normalization. $s$ denotes stride.

| Input | Operator | exp size | #out | SE | NL | s |
|---|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d, 3x3 | - | 16 | - | HS | 2 |
| $112^2 \times 16$ | bneck, 3x3 | 16 | 16 | ✓ | RE | 2 |
| $56^2 \times 16$ | bneck, 3x3 | 72 | 24 | - | RE | 2 |
| $28^2 \times 24$ | bneck, 3x3 | 88 | 24 | - | RE | 1 |
| $28^2 \times 24$ | bneck, 5x5 | 96 | 40 | ✓ | HS | 2 |
| $14^2 \times 40$ | bneck, 5x5 | 240 | 40 | ✓ | HS | 1 |
| $14^2 \times 40$ | bneck, 5x5 | 240 | 40 | ✓ | HS | 1 |
| $14^2 \times 40$ | bneck, 5x5 | 120 | 48 | ✓ | HS | 1 |
| $14^2 \times 48$ | bneck, 5x5 | 144 | 48 | ✓ | HS | 1 |
| $14^2 \times 48$ | bneck, 5x5 | 288 | 96 | ✓ | HS | 2 |
| $7^2 \times 96$ | bneck, 5x5 | 576 | 96 | ✓ | HS | 1 |
| $7^2 \times 96$ | bneck, 5x5 | 576 | 96 | ✓ | HS | 1 |
| $7^2 \times 96$ | conv2d, 1x1 | - | 576 | ✓ | HS | 1 |
| $7^2 \times 576$ | pool, 7x7 | - | - | - | - | 1 |
| $1^2 \times 576$ | conv2d 1x1, NBN | - | 1024 | - | HS | 1 |
| $1^2 \times 1024$ | conv2d 1x1, NBN | - | k | - | - | 1 |

Table 2. Specification for MobileNetV3-Small. See table 1 for notation.

### Evaluations

| Network | Top-1 | MAdds | Params | P-1 | P-2 | P-3 |
|---|---|---|---|---|---|---|
| V3-Large 1.0 | **75.2** | **219** | 5.4M | **51** | **61** | **44** |
| V3-Large 0.75 | 73.3 | 155 | 4.0M | 39 | 46 | 40 |
| MnasNet-A1 | 75.2 | 315 | 3.9M | 71 | 86 | 61 |
| Proxyless[5] | 74.6 | 320 | 4.0M | 72 | 84 | 60 |
| V2 1.0 | 72.0 | 300 | 3.4M | 64 | 76 | 56 |
| V3-Small 1.0 | **67.4** | **56** | 2.5M | **15.8** | **19.4** | **14.4** |
| V3-Small 0.75 | 65.4 | 44 | 2.0M | 12.8 | 15.6 | 11.7 |
| Mnas-small [43] | 64.9 | 65.1 | 1.9M | 20.3 | 24.2 | 17.2 |
| V2 0.35 | 60.8 | 59.2 | 1.6M | 16.6 | 19.6 | 13.9 |

Table 3. Floating point performance on the Pixel family of phones (P-$n$ denotes a Pixel-$n$ phone). All latencies are in ms and are measured using a single large core with a batch size of one. Top-1 accuracy is on ImageNet.

## Inceptions

(Which I call "Thee Width Learning" lol.)

### GoogLeNet & Inception V1

GoogLeNet won the first place of 2014 ILSVRC chanllenge, first proposed by Google in 2014, variase impovements are being made afterwards, including InceptionV2(BN), InceptionV3, InceptionV4 and Xception.

#### Theory

The paper[9] proposed that deeper(blocks)/wider(channels) DNN models are with large number of parameeters and thus more easily to be trained as overfitting model, where for high quality models, the training dataset could be very limited since preparing such dataset is not just trick, but expensive.

Another issue of those models is the increased computational cost and the memory cost, which is very energy-inefficient. The fundamental way of solving both issues that are widely used during that time is sparses the computational graph, however, since the lower-level hardware designs of our computing devices are mostly structured designed, sparse redundent weights randomly does not bring much benefits to the models. Structural Purning proposed later, however, is lossing the accuracy of DNN models.

The paper referenced the main result of [10], which states that:

---

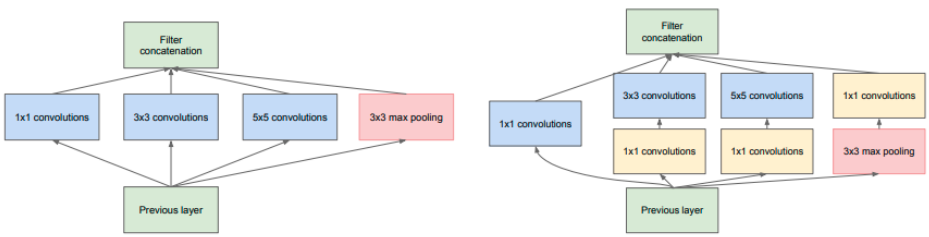
**If the probability distribution of the dataset is representable by a large, very sparse deep neural network, then the optimal network topology can be constructed layer by layer by analyzing the correlation statistics of the activations of the last layer and clustering neurons with highly correlated outputs.**

---

**Neurons That Fire Together, Wire Together. --Hebbian Priciple**

---

Based on above, the Inception structure is proposed, with dense wider design to approximate local sparsity and aggregate the resulting chennels as the output. The idea of Inception block is descripted as the following:




(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

In first attempt from 2(a), the Inception is consistes of combination of all state-of-art sparse layers with their output filter bancks concatenated into a single output. With 1x1 dimension reduction is applied, rather than the naive approach, the Inception block could be extracting more informative information (from difference sparse units + different scale spatial information with multi-kernel sizes), by using above designs, Inception is allowed to increaseing units at each block without computational complexity blowing-up, whilist keeping the lower-level sparse computational away.

An compact model using Inception blcoks is proposed in the paper, namely the GoogLeNet (With 2 additional auxiliary classifiers weighted 0.3 at lower/middle depth):

| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Table 1: GoogLeNet incarnation of the Inception architecture

This design wins VGG by ~1.5% less error rate.

### Inception V2 (BatchNorm)

Proposed by Google in 2015 [11]. According the widely used SGD meethod for DNN weights updating, the paper points that the layer output is largely effected by the inputs from the previous layer. Therefore, during training/testing stage, the distributions of the inputs matters to the model. It would be advantageous for the distribution of the inputs remain fixed over time so the network weight does not have to readjust to adapt new distribution. Such distribution transformation of inter-network nodes is called **Internal Covariate Shift(ICS)**.

The paper proposed method called **Batch Normalization**, which is nowadays commonly used in DL community.

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

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

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

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

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

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

During the backpropagation, the scale parameter $\gamma$ and shift parameter $\beta$ also needs to be learned by passing through the loss $l$ and compute the gradient with respect to these parameters. Batch Normalization also allows learning rate to be set higher without weights blowing-up. Occasionally speaking, with BN, the dropout layer is sometimes discared.

Inception V2 is based on Batch Normalization Method by adding BN after layer transformation, using $ReLU$ as activation functions, and replace the 5x5 Conv by two consecutive 3x3 Conv:

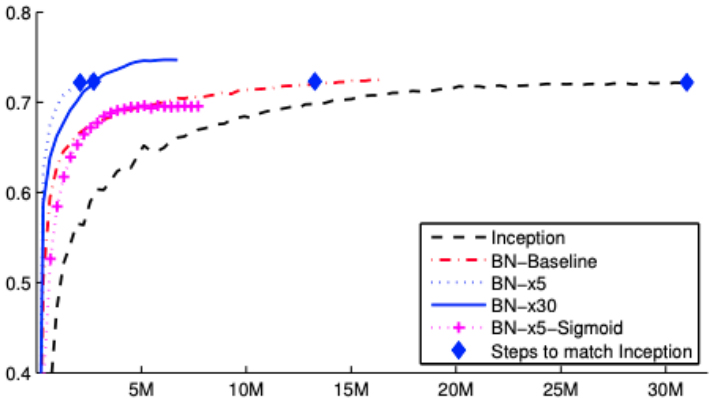



*Figure 2. Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

The Inception trained with different larger learning rate achives some result much faster interms of training episodes.

### Inception V3

Propsed by Google in 2015. [12] The paper relooked into the inception designs and propsed the following 4 General Design Principles regarding DNNs, that empirically:

1. Avoid extreme representational bottlenecks, especially in the early network. Otherwise information could be lost.
2. Higher dimensional representations are easier to process locally within a network, hence converges faster.
3. Spatial aggregation can be done over lower dimensional embeddings without much or any loss in rep- resentational power.
4. Balance network width and depth.

–

More efficient Inception could be achived by **Factorizing the Convolution**. First, factorize the original convolutions to the combinations of smaller combinations stacking on top of each other, by doing so, we made the in-block network deeper, hence imporved representation power of the non-linearity of the model, while significantly saving parameters and MACs. As shown in below:

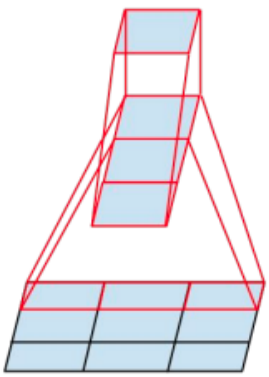


Figure 3. Mini-network replacing the $3 \times 3$ convolutions. The lower layer of this network consists of a $3 \times 1$ convolution with 3 output units.

The resulting architecture will looks like the following:

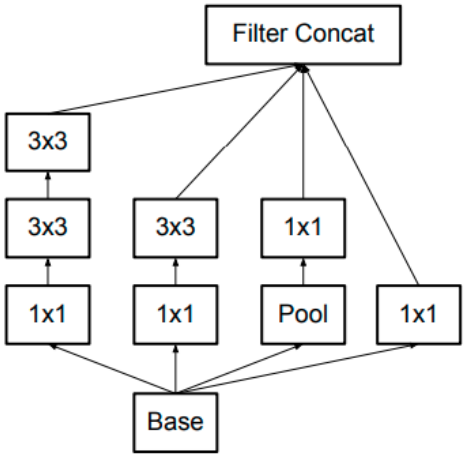



Figure 5. Inception modules where each $5 \times 5$ convolution is replaced by two $3 \times 3$ convolution, as suggested by principle 3 of Section 2.

Another approach to factorize the convolution, rather than replacing larger convolution by smaller convolutions, is to factorize the original convolution into asymmetric convolution.

Wherea kernel of 3x3 could be disolved into stacking of 1x3 and 3x1, with 33% less parameters. As shown in below:

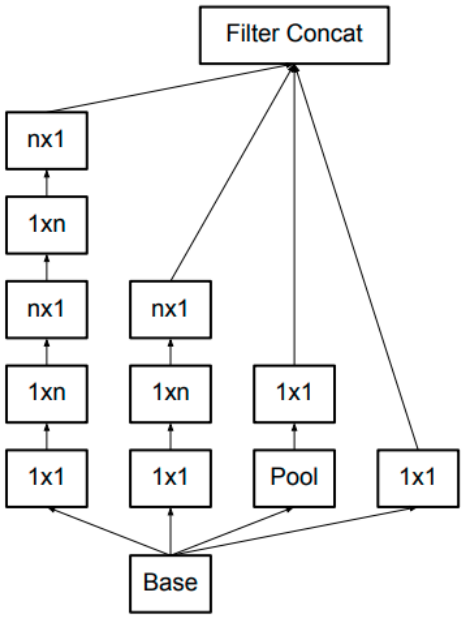



Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the $17 \times 17$ grid. (The filter sizes are picked using principle 3)

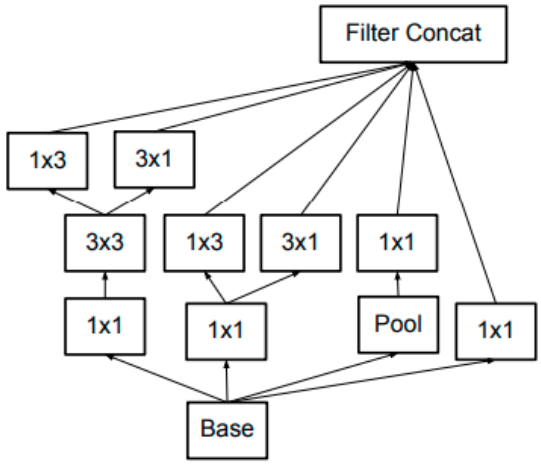



Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest $(8 \times 8)$ grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by $1 \times 1$ convolutions) is increased compared to the spatial aggregation.

In addition, the paper proposed an method of down-sampling. The current down-samplings are performed by either:

1. Pooling -> Enlarge Dimensions: This will suffer lose of information.
2. Enlarge Dimensions -> Pooling: Computationally costy.

The paper proposed a method called **Efficient Grid Size Reduction**, which leverage the speciality that Inception blocks have units, to parallely perform above actions to produce output. As shown in below:

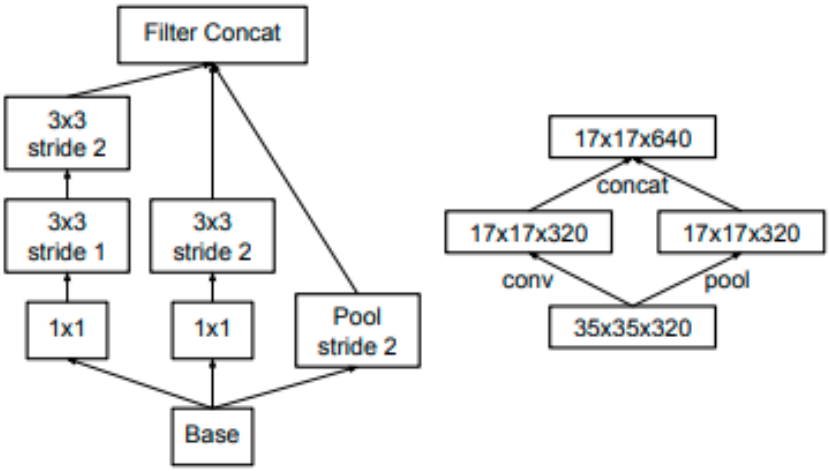



Figure 10. Inception module that reduces the grid-size while expands the filter banks. It is both cheap and avoids the representational bottleneck as is suggested by principle 1. The diagram on the right represents the same solution but from the perspective of grid sizes rather than the operations.

The complete structure of the InceptionV3 is given by:

| type | patch size/stride or remarks | input size |
|---|---|---|
| conv | 3×3/2 | 299×299×3 |
| conv | 3×3/1 | 149×149×32 |
| conv padded | 3×3/1 | 147×147×32 |
| pool | 3×3/2 | 147×147×64 |
| conv | 3×3/1 | 73×73×64 |
| conv | 3×3/2 | 71×71×80 |
| conv | 3×3/1 | 35×35×192 |
| 3×Inception | As in figure 5 | 35×35×288 |
| 5×Inception | As in figure 6 | 17×17×768 |
| 2×Inception | As in figure 7 | 8×8×1280 |
| pool | 8 × 8 | 8 × 8 × 2048 |
| linear | logits | 1 × 1 × 2048 |
| softmax | classifier | 1 × 1 × 1000 |

Table 1. The outline of the proposed network architecture. The output size of each module is the input size of the next one. We are using variations of reduction technique depicted Figure 10 to reduce the grid sizes between the Inception blocks whenever applicable. We have marked the convolution with 0-padding, which is used to maintain the grid size. 0-padding is also used inside those Inception modules that do not reduce the grid size. All other layers do not use padding. The various filter bank sizes are chosen to observe principle 4 from Section 2.

Such approach achieves $7.8\%$ and $3.8\%$ less error rate compared with InceptionV1 and V2 respectively.

## References

[0] Deng, Lei, et al. "Model compression and hardware acceleration for neural networks: A comprehensive survey." Proceedings of the IEEE 108.4 (2020): 485-532.

[1] Cheng, Yu, et al. "A survey of model compression and acceleration for deep neural networks." arXiv preprint arXiv:1710.09282 (2017).

[2] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[3] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

[4] Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., ... & Adam, H. (2019). Searching for mobilenetv3. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 1314-1324).

[5] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2820-2828).

[6] Yang, T. J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., ... & Adam, H. (2018). Netadapt: Platform-aware neural network adaptation for mobile applications. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 285-300).

[7] Yang, Tien-Ju and Chen, Yu-Hsin and Sze, Vivienne: Designing energyefficient convolutional neural networks using energy-aware pruning. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)

[8] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).

[9] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[10] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. CoRR, abs/1310.6343, 2013.

[11] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.

[12] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).