Lightweight Neural Network Architectures

Andrii Polukhin

pandrii000@gmail.com LinkedIn

Data Science UA

October 11, 2022

About me

- ML Engineer in Data Science UA and Samba.TV
- Mentor in PRJCTR and 10:11
- Getting a Master's degree in Mathematics and searching for Ph.D. supervisor in Computer Science
- Writing about AI in Telegram: @eiaioi

Table Of Contents

1 Problem statement

2 DL Optimization Pipeline

3 DL Families Overview

4 MobileNet Family
- MobileNet v1 (2017)
- MobileNet v2 (2018)
- MobileNet v3 (2019)
- MobileNet Summary

5 Model Scaling Formula Family
- Background
- EfficientNet v1 (2019)
- EfficientNet v2 (2021)
- TinyNet (2020)
- Model Scaling Formula Summary

6 Neural Architecture Search Family
- Background
- NASNet (2017)
- PNASNet (2017)
- ChamNet (2018)
- MNASNet (2019)
- FBNet (2019)
- AmoebaNet (2019)
- Neural Architecture Search Summary

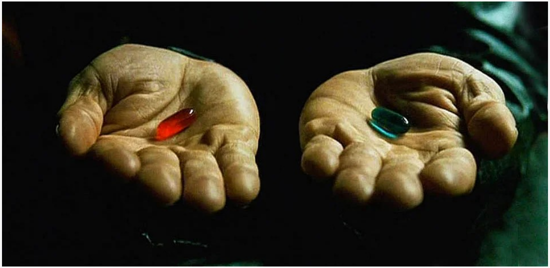
7 Group Convolution Family
- Background
- CondenseNet (2017)
- ShuffleNet v1 (2017)
- ShuffleNet v2 (2018)
- MixNet (2019)
- GhostNet (2020)
- DiCENet (2020)
- MicroNet (2021)
- Group Convolution Summary

8 Squeeze & Excitation Family
- SqueezeNet (2016)
- SqueezeNeXt (2018)
- SENet (2017)
- Squeeze & Excitation Summary

9 Mobile Transformer Family
- MobileViT (2022)
- EdgeViTs (2022)
- Mobile Transformer Summary

Problem statement

Larger model produces **better** results, but runs **slower**.
Smaller model produces **worse** results, but runs **faster**.



DL Optimization Pipeline

1. **Model Selection – lecture objective**
   - MobileNet, FBNet, MobileViT, etc
2. Model Optimization
   - With changing model architecture: pruning, low-rank factorization, knowledge distillation, singular value decomposition, weight clustering
   - Without changing model architecture: quantization
   - Combination of the methods above
3. Non-Model Optimization
   - Software accelerators using mobile device hardware: DeepX, CNNdroid, RSTensorFlow, DeepMon, CADNN
   - Mobile hardware designs for DL: TrueNorth, VPU, EIE, DianNao, FPGA15

DL Families Overview

- MobileNet v1 (2017)
- MobileNet v2 (2018)
- MobileNet v3 (2019)

MobileNet Family

- EfficientNet v1 (2019)
- EfficientNet v2 (2021)
- TinyNet (2020)

Model Scaling Formula Family

- NASNet (2017)
- PNASNet (2017)
- ChamNet (2018)
- MNASNet (2019)
- FBNet (2019)
- AmoebaNet (2019)

Neural Architecture Search Family

DL Families Overview

- ShuffleNet v1 (2017)
- CondenseNet (2017)
- ShuffleNet v2 (2018)
- MixNet (2019)
- GhostNet (2020)
- DiCENet (2020)
- MicroNet (2021)

Group Convolution Family

- SqueezeNet (2016)
- SENet (2017)
- SqueezeNeXt (2018)

Squeeze & Excitation Family

- MobileViT (2022)
- EdgeViTs (2022)

Mobile Transformer Family

MobileNet Family

MobileNet Family

- MobileNet v1 (2017)
- MobileNet v2 (2018)
- MobileNet v3 (2019)

MobileNet Family

MobileNet v1 (2017)

- MobileNet v1 (2017) – 10x faster and smaller than VGG16 (2014)
  - ❌ Convolution ✅ Depth-wise Separable Convolution
  - ❌ ReLU ✅ ReLU6
  - Width and resolution hyper-parameters

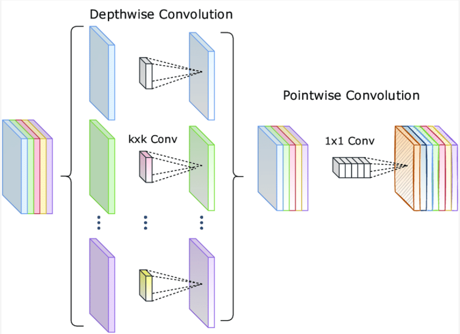


Figure: Depth-wise Separable Convolution (x9 faster than Conv3x3)

MobileNet v1 (2017) – ReLU6

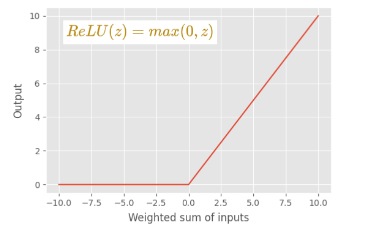


Figure: ReLU

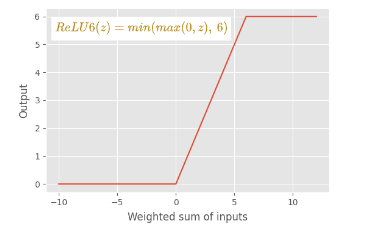


Figure: ReLU6

The authors of the MobileNet paper found that ReLU6 is more robust than regular ReLU when using low-precision computation.

MobileNet v1 (2017) – Width and resolution hyper-parameters

and width multiplier α, the number of input channels M becomes αM and the number of output channels N becomes αN.

The computational cost of a depthwise separable convolution with width multiplier α is:

$$\boxed{D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F \quad (6)}$$

where $\alpha \in (0, 1]$ with typical settings of 1, 0.75, 0.5 and 0.25. $\alpha = 1$ is the baseline MobileNet and $\alpha < 1$ are reduced MobileNets. Width multiplier has the effect of re-

Figure: **Width multiplier**

Thinner models (reduce channels)

We can now express the computational cost for the core layers of our network as depthwise separable convolutions with width multiplier α and resolution multiplier ρ:

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F \quad (7)$$

where $\rho \in (0, 1]$ which is typically set implicitly so that the input resolution of the network is 224, 192, 160 or 128. $\rho = 1$ is the baseline MobileNet and $\rho < 1$ are reduced computation MobileNets. Resolution multiplier has the effect of reducing computational cost by ρ^2.

Figure: **Resolution multiplier**

reduce width and height

MobileNet v2 (2018)

- MobileNet v2 (2018) – 0.3x faster and smaller, +1% accurate than MobileNet v1 (2017)
  - ❌ Non-linear bottleneck ✅ Linear bottleneck
  - Inverted residual block
  - Expansion - Projection way

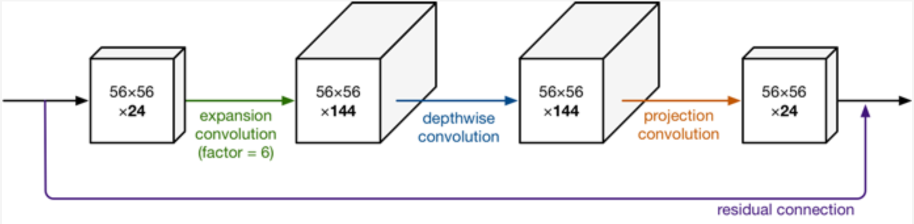


Figure: Residual connection as building block

MobileNet v2 (2018) – Linear Bottlenecks & Inverted residual block

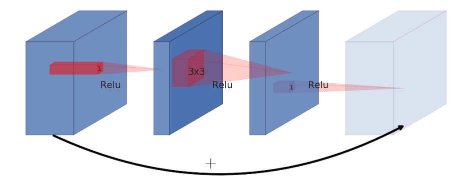


Figure: Residual connection

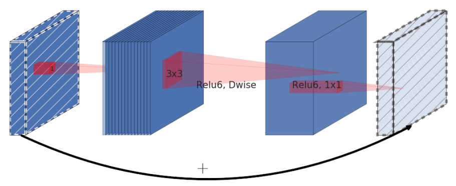


Figure: Inverted residual connection

- The diagonally hatched texture – linear layers.
- The last layer is the beginning of the next block.
- Thickness of each block indicates its relative number of channels.

MobileNet v2 (2018) – Expansion - Projection way

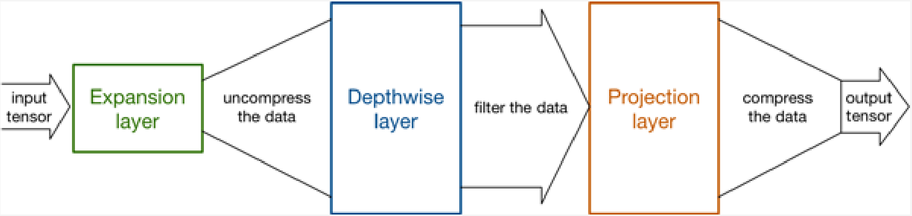


Figure: MobileNet v2 block

The expansion layer acts as an decompressor (like unzip) that first restores the data to its full form, then the depthwise layer performs whatever filtering is important at this stage of the network, and finally the projection layer compresses the data to make it small again.

MobileNet v3 (2019)

- MobileNet v3 (2019) – 2x faster, 30% smaller, -3% accurate than MobileNet v2 (2018)
  - ❌ ReLU6 ✅ HardSwish
  - Squeeze-and-Excitation module
  - MnasNet & NetAdapt
  - Redesigning Expensive Layers

MobileNet v3 (2019) – HardSwish (H-Swish)

- $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$
- $h\text{-sigmoid}(x) = \frac{\text{ReLU6}(x+3)}{6}$
- $\text{swish}(x) = x\ \text{sigmoid}(x)$
- $\text{h-swish}(x) = x\frac{\text{ReLU6}(x+3)}{6}$

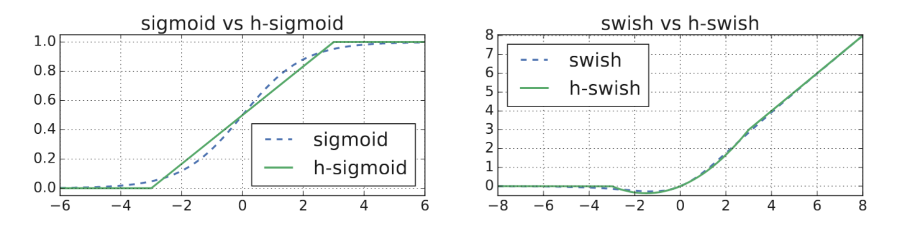


Figure: H-Sigmoid and H-Swish

MobileNet v3 (2019) – Squeeze-and-Excitation module

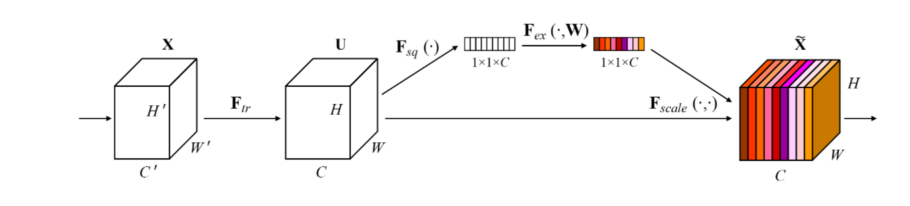


Figure: Squeeze-and-excitation Block

- For any given transformation F_{tr} mapping the input X to the feature maps U where $U \in R^{HWC}$, e.g. a convolution, we can construct a corresponding SE block to perform **feature recalibration**.
- The features U are first passed through a squeeze operation, which produces a channel descriptor by aggregating feature maps across their spatial dimensions HW. The function of this descriptor is to produce an embedding of the global distribution of channel-wise feature responses, allowing information from the global receptive field of the network to be used by all its layers.
- The aggregation is followed by an excitation operation, which takes the form of a simple self-gating mechanism that takes the embedding as input and produces a collection of per-channel modulation weights.
- These weights are applied to the feature maps U to generate the output of the SE block which can be fed directly into subsequent layers of the network.

MobileNet v3 (2019) – Squeeze-and-Excitation module

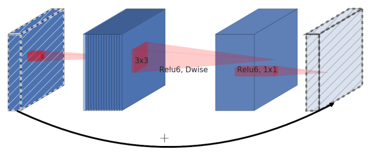


Figure: MobileNet v2

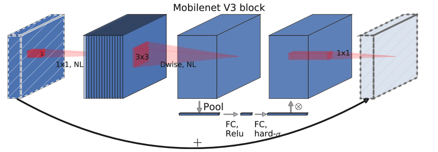


Figure: MobileNet v3

- Compared with MobileNetV2, MobileNetV3 has inserted the Squeeze and Excitation (SE) module, which is originated in SENet.
- H-Sigmoid is used to replace sigmoid in SE module for efficient computation.

MobileNet Summary

- MobileNet v1 (2017) – 10x faster and smaller than VGG16 (2014)
  - ❌ Convolution ✅ Depth-wise Separable Convolution
  - ❌ ReLU ✅ ReLU6
  - Width and resolution hyper-parameters
- MobileNet v2 (2018) – 0.3x faster and smaller, +1% accurate than MobileNet v1 (2017)
  - ❌ No residual connection ✅ Inverted residual block
  - ❌ Non-linear bottleneck ✅ Linear bottleneck
  - Expansion - Projection way
- MobileNet v3 (2019) – 2x faster, 30% smaller, -3% accurate than MobileNet v2 (2018)
  - ❌ HardSwish ✅ ReLU6
  - Squeeze-and-Excitation module
  - NetAdapt architecture optimization

Model Scaling Formula Family

Background – Width, Depth, Resolution

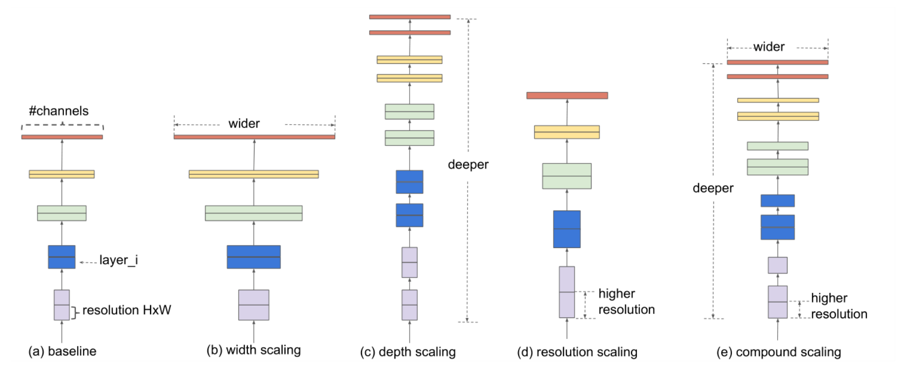


Figure: Width, Depth, Resolution

Model Scaling Formula Family

- EfficientNet v1 (2019)
- EfficientNet v2 (2021)
- TinyNet (2020)

Model Scaling Formula Family

EfficientNet v1 (2019)

- EfficientNet v1 (2019) – 6x faster than ResNet and GPipe
  - ❌ Guess hyper-parameters ✅ Model scaling formula

$$
\begin{aligned}
\text{depth: } & d = \alpha^{\phi} \\
\text{width: } & w = \beta^{\phi} \\
\text{resolution: } & r = \gamma^{\phi} \\
\text{s.t. } & \alpha \cdot \beta^{2} \cdot \gamma^{2} \approx 2 \\
& \alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned} \tag{3}
$$

Figure: EfficientNet Formula

EfficientNet v2 (2021)

- EfficientNet v2 (2021) – 2x faster than EfficientNet v1
  - ❌ Static training parameters ✅ Progressive training
  - ❌ Depthwise layers in early layers ✅ Depthwise layers in later stages
  - ❌ MBConv ✅ Fused-MBConv
  - Training-Aware NAS and Scaling

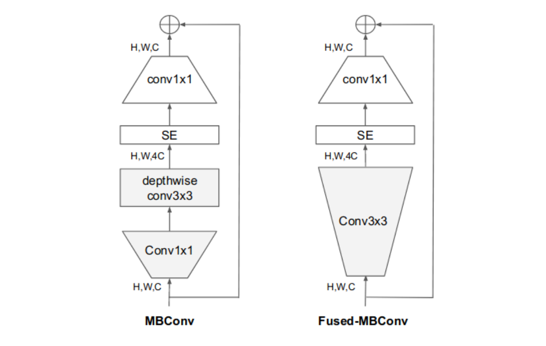


Figure: Structure of MBConv and Fused-MBConv

TinyNet (2020)

- TinyNet (2020) – +2% accurate than EfficientNet v1 (2019)
  - Improved Model scaling formula in EfficientNet v1 for constraint $0 < c < 1$, using nonparametric Guassign process regression
  - $w = \sqrt{\frac{c}{r^2 d}}, 0 < c < 1.$, w is width, r is resolution, d is depth

Model	FLOPs	Acc.	Model	FLOPs	Acc.
EfficientNet-B^{-1}	200M	75.8%	EfficientNet-B^{-2}	97M	72.1%
shrink B0 by $r = 0.70$	196M	74.9%	shrink B0 by $r = 0.46$	103M	70.3%
shrink B0 by $d = 0.45$	196M	76.5%	depth underflow†	-	-
shrink B0 by $w = 0.65$	205M	77.2%	shrink B0 by $w = 0.38$	99M	73.2%
TinyNet-B (ours)	201M	**77.6%**	TinyNet-C (ours)	97M	**74.1%**

Figure: Comparison to EfficientNet Rule

Model Scaling Formula Summary

- EfficientNet v1 (2019) – 6x faster than ResNet and GPipe
  - ❌ Guess hyper-parameters ✅ Model scaling formula
- EfficientNet v2 (2021) – 2x faster than EfficientNet v1
  - ❌ Static training parameters ✅ Progressive training
  - ❌ Depthwise layers in early layers ✅ Depthwise layers in later stages
  - ❌ MBConv ✅ Fused-MBConv
  - Training-Aware NAS and Scaling
- TinyNet (2020) – +2% accurate than EfficientNet v1 (2019)
  - Improved Model scaling formula in EfficientNet v1 for constraint $0 < c < 1$, using nonparametric Guassign process regression
  - $w = \sqrt{\frac{c}{r^2 d}}, 0 < c < 1.$, w is width, r is resolution, d is depth

Neural Architecture Search Family

Background – Neural Architecture Search

We can categorize methods for NAS according to three dimensions:

- **Search Space.** The search space defines which architectures can be represented in principle.
- **Search Strategy.** The search strategy details how to explore the search space.
- **Performance Estimation Strategy.** The objective of NAS is typically to find architectures that achieve high predictive performance on unseen data.

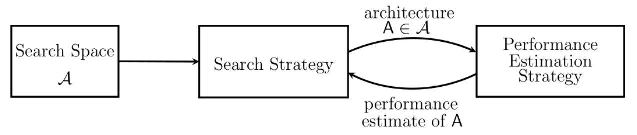


Figure: NAS Overview

Neural Architecture Search Family

- NASNet (2017)
- PNASNet (2017)
- ChamNet (2018)
- MNASNet (2019)
- FBNet (2019)
- AmoebaNet (2019)

} Neural Architecture Search Family

NASNet (2017)

- NASNet (2017) – 3x faster, +4% accurate than Inception (2015)
  - NASNet Cell-based Search Space
  - ScheduledDropPath regularization technique

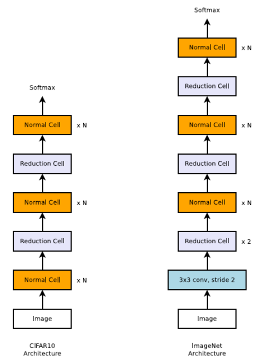


Figure: NASNet Search Space

NASNet (2017) – ScheduledDropPath

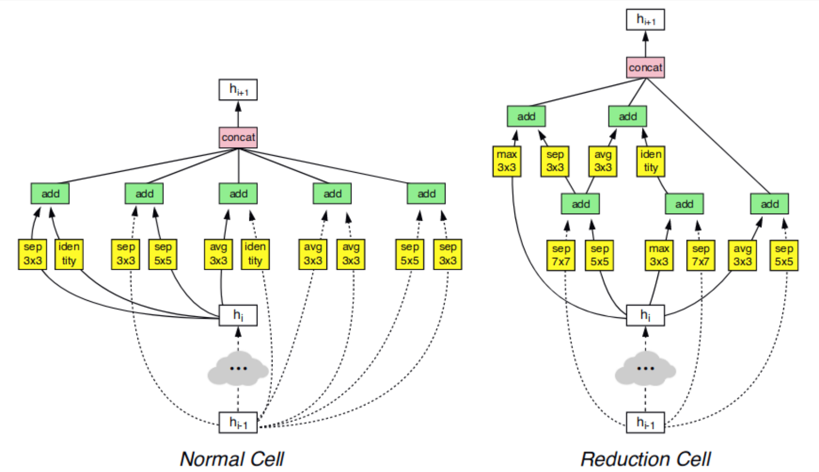


Figure: **ScheduledDropPath.** During training, stochastically drop out each path (i. e. edge with a yellow box) with a probability that is linearly increased over the course of training.

PNASNet (2017)

- PNASNet (2017) – 8x faster, 5x efficient than NASNet (2017)
  - ❌ RL and Evolution algorithm (EA) ✅ Sequential model-base-optimization (LSTM, MLP)

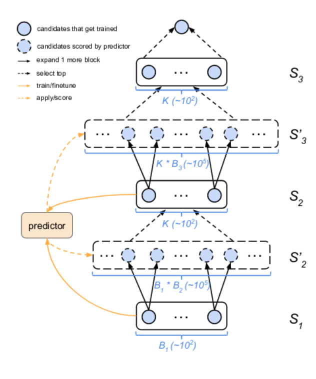


Figure: PNASNet search procedure

ChamNet (2018)

- ChamNet (2018) – +8% accurate than ResNet-50 with same latency
  - Chameleon adaptive genetic algorithm, where the gene of and NN is a vector of hyp-s (#Filters and #Bottlenects).

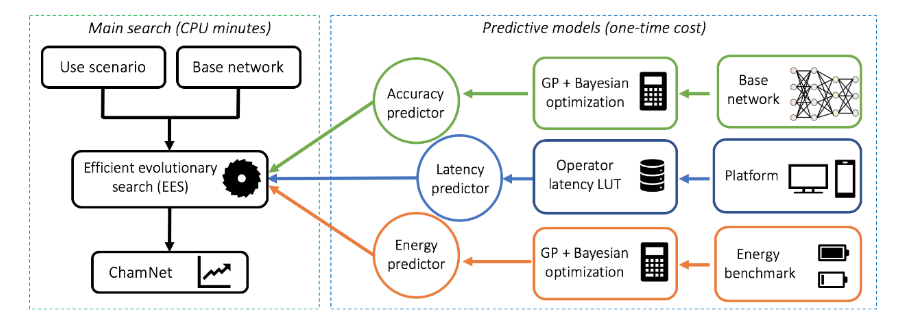


Figure: Chameleon adaptation framework

MNASNet (2019)

- MNASNet (2019) – 2x faster than MobileNet v2 (2018) and NASNet (2017)
  - ❌ Minimize FLOPS to reduce latency ✅ Minimize latency directly
  - RNN Optimization

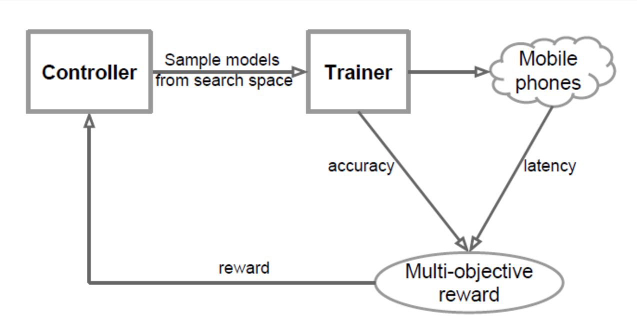


Figure: An Overview of Platform-Aware Neural Architecture Search for Mobile

FBNet (2019)

- FBNet (2019) – 420x faster search than MNASNet (2019)
  - Differentiable neural architecture search (DNAS) framework that uses gradient-based methods to optimize ConvNet architectures

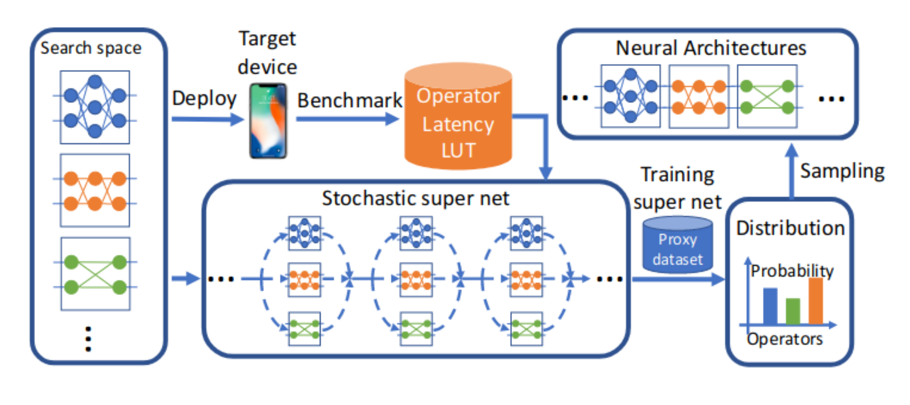


Figure: FBNet for ConvNet design

AmoebaNet (2019)

- AmoebaNet (2019) – a few times faster search than RL
  - ❌ RL ✅ Aging evolution (at the earlier stages of the search)

Algorithm 1 Aging Evolution

```
population ← empty queue                              ▷ The population.
history ← ∅                                           ▷ Will contain all models.
while |population| < P do                             ▷ Initialize population.
    model.arch ← RANDOMARCHITECTURE()
    model.accuracy ← TRAINANDEVAL(model.arch)
    add model to right of population
    add model to history
end while
while |history| < C do                                ▷ Evolve for C cycles.
    sample ← ∅                                        ▷ Parent candidates.
    while |sample| < S do
        candidate ← random element from population
                              ▷ The element stays in the population.
        add candidate to sample
    end while
    parent ← highest-accuracy model in sample
    child.arch ← MUTATE(parent.arch)
    child.accuracy ← TRAINANDEVAL(child.arch)
    add child to right of population
    add child to history
    remove dead from left of population               ▷ Oldest.
    discard dead
end while
return highest-accuracy model in history
```

Figure: Aging Evolution

Neural Architecture Search Summary

- NASNet (2017) – 3x faster, +4% accurate than Inception (2015)
  - NASNet Cell-based Search Space
  - ScheduledDropPath regularization technique
- PNASNet (2017) – 8x faster, 5x efficient than NASNet (2017)
  - ❌ RL and Evolution algorithm (EA) ✅ Sequential model-base-optimization
- ChamNet (2018) – +8% accurate than ResNet-50 with same latency
  - Chaneleon EES
- MNASNet (2019) – 2x faster than MobileNet v2 (2018) and NASNet (2017)
  - ❌ Minimize FLOPS to reduce latency ✅ Minimize latency directly
  - RNN Optimization
- FBNet (2019) – 420x faster search than MNASNet (2019)
  - Differentiable neural architecture search (DNAS) framework that uses gradient-based methods to optimize ConvNet architectures
- AmoebaNet (2019) – a few times faster search than RL
  - ❌ RL ✅ Aging evolution (at the earlier stages of the search)

Group Convolution Family

Background – Group Convolution

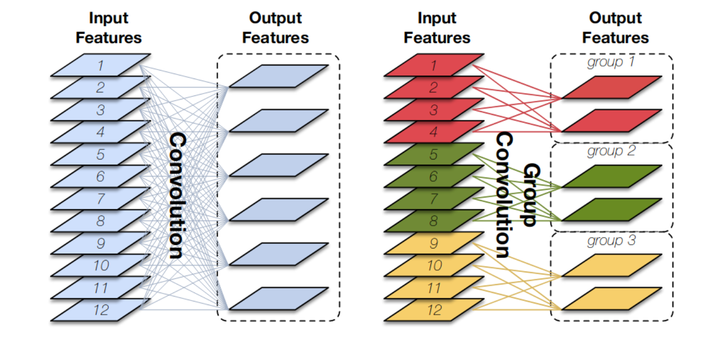


Figure: Convolution and Group Convolution

Group Convolution Family

- ShuffleNet v1 (2017)
- CondenseNet (2017)
- ShuffleNet v2 (2018)
- MixNet (2019)
- GhostNet (2020)
- DiCENet (2020)
- MicroNet (2021)

} Group Convolution Family

CondenseNet (2017)

- CondenseNet (2017) – 2x smaller than ShuffleNet (2017)
  - Weights pruning in the early stages of training
  - ❌ Convolution ✅ Group Convolution
  - ❌ Pre-define groups of convolutions ✅ Learn input feature grouping automatically during training

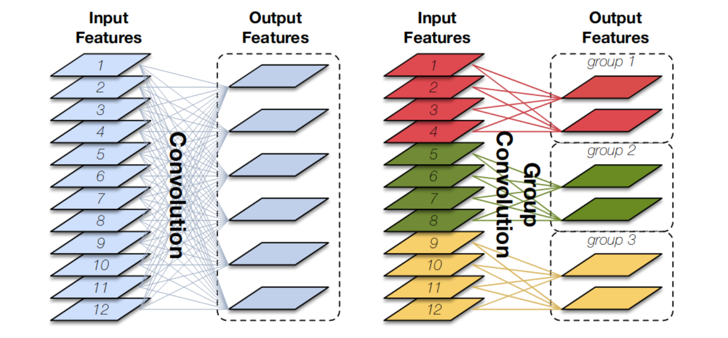


Figure: Convolution and Group Convolution

CondenseNet (2017) – Learned Group Convolution

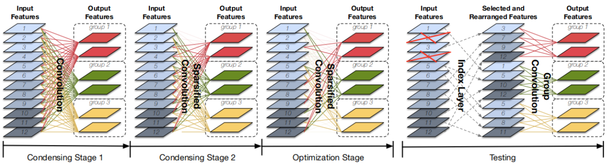


Figure 3. Illustration of learned group convolutions with $G=3$ groups and a condensation factor of $C=3$. During training a fraction of $(C-1)/C$ connections are removed after each of the $C-1$ condensing stages. Filters from the same group use the same set of features, and during test-time the *index layer* rearranges the features to allow the resulting model to be implemented as standard group convolutions.

Figure: Group Convolution Learning Process

At the end of each $C1$ condensing stages we prune $\frac{1}{C}$ of the filter weights. By the end of traing, only $\frac{1}{C}$ of the weights remain in each filter group.

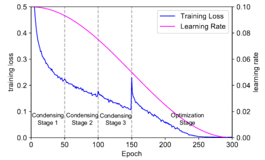


Figure 4. The cosine shape learning rate and a typical training loss curve with a condensation factor of $C=4$.

Figure: Training Process

ShuffleNet v1 (2017)

- ShuffleNet v1 (2017) – 13x faster than AlexNet (2012), +7% accurate than MobileNet v1 (2017)
  - ❌ Convolution ✅ Group Convolution with feature shuffling

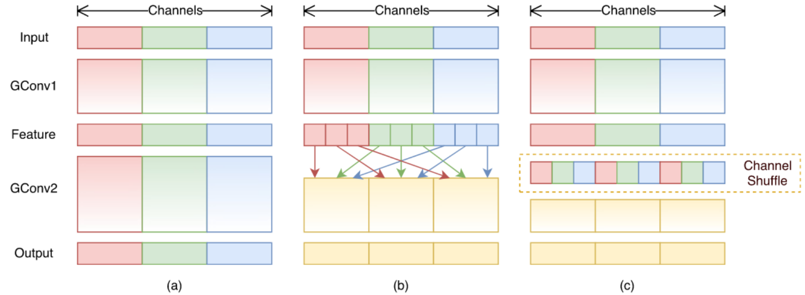


Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when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

Figure: Channel shuffle

ShuffleNet v1 (2017) – ShuffleNet Unit

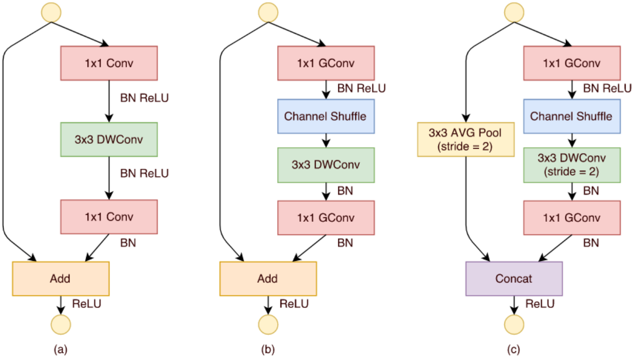


Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

Figure: ShuffleNet Unit

ShuffleNet v2 (2018)

- ShuffleNet v2 (2018) – 58% faster than MobileNet v2 (2018), 63% faster than ShuffleNet v1 (2017)
  - ❌ Channel Shuffle between 1x1 conv and 3x3 DWConv ✅ Channel Shuffle at the end of the block
  - ❌ ReLU after Concat ✅ ReLU before Concat
  - Split features in two groups before the convolution: identity and x

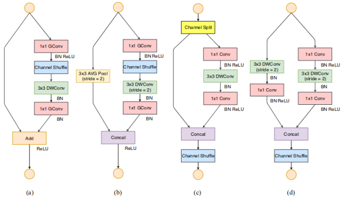


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling (2×); (c) our basic unit; (d) our unit for spatial down sampling (2×). **DWConv**: depthwise convolution. **GConv**: group convolution.

Figure: Channel shuffle v2

MixNet (2019)

- MixNet (2019) – +4% accurate than MobileNet v2 (2018)
  - ❌ Convolution ✅ MixConv, mixes up multiple kernel sizes in one convolution

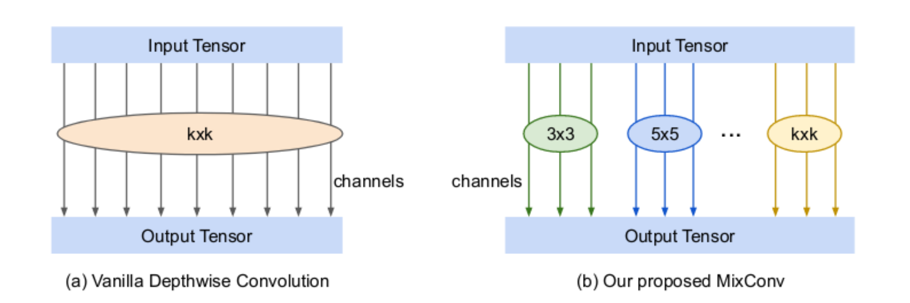


Figure: Mixed depthwise convolution (MixConv)
Unlike vanilla depthwise convolution that applies a single kernel to all channels, MixConv partitions channels into groups and apply different kernel size to each group.

GhostNet (2020)

- GhostNet (2020) – +1% accurate than MobileNet v3 (2019)
  - ❌ # Feature maps equals # Kernels ✅ Few times more feature maps by applying linear operations on them

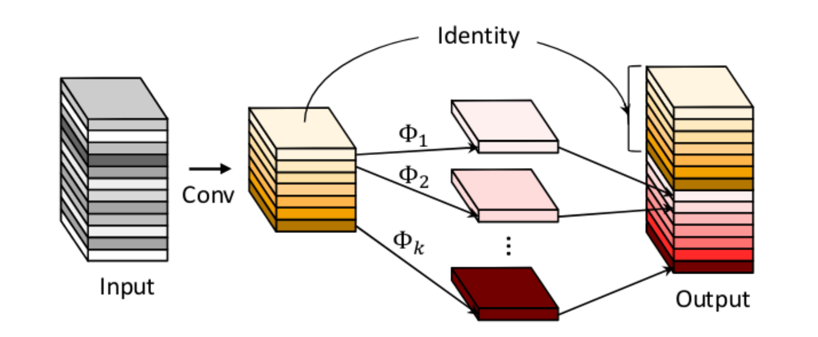


Figure: An illustration of the convolutional layer and the proposed Ghost module for outputting the same number of feature maps. Φ represents the cheap operation (Depthwise Convolution).

DiCENet (2020)

- DiCENet (2020) – +3% accurate than MobileNet v2 (2018) and ShuffleNet v2 (2018)
  - ❌ Depth-wise Separable Convolution ✅ DiCE Unit

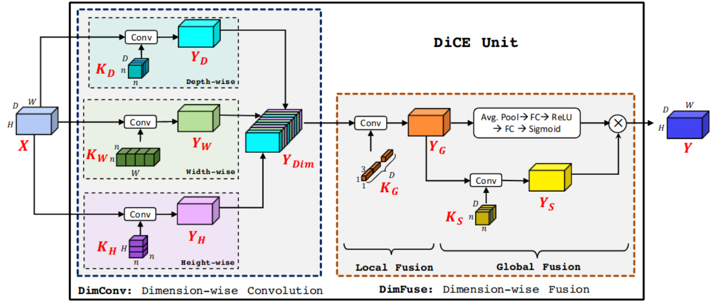


Fig. 3: **DiCE unit** efficiently encodes the spatial and channel-wise information in the input tensor **X** using dimension-wise convolutions (DimConv) and dimension-wise fusion (DimFuse) to produce an output tensor **Y**. For simplicity, we show kernel corresponding to each dimension independently. However, in practice, these three kernels are executed simultaneously, leading to faster run-time. See Section 3.4 and 6 for more details.

Figure: DiCE Unit

MicroNet (2021)

- MicroNet (2021) – 2x smaller, 3x faster than MobileNet v3 (2019)
  - ❌ Convolution ✅ Micro-Factorized Grouped convolution
  - ❌ ReLU ✅ Dynamic Shift-Max

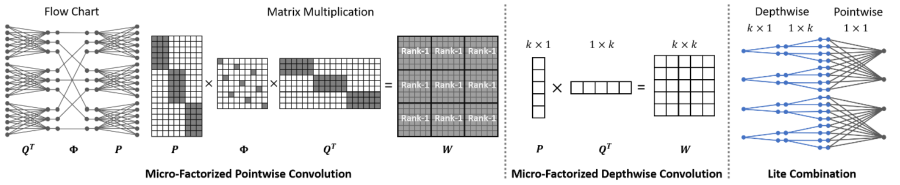


Figure 2. **Micro-Factorized pointwise and depthwise convolutions. Left:** factorizing a pointwise convolution into two group-adaptive convolutions, where the group number $G = \sqrt{C/R} = \sqrt{18/2} = 3$. The resulting matrix $\boldsymbol{W}$ can be divided into $G \times G$ blocks, of which each block has rank 1. **Middle:** factorizing a $k \times k$ depthwise convolution into a $k \times 1$ and a $1 \times k$ depthwise convolutions. **Right:** lite combination of Micro-Factorized pointwise and depthwise convolutions.

Figure: Micro-Factorized pointwise and depthwise convolutions.
This low rank approximation reduces the computational complexity from $O(k^2C)$ to $O(kC)$

MicroNet (2021) – Dynamic Shift-Max

Dynamic Shift-Max, an activation function, that fuses an input feature map and takes maximum with its circular group shift.

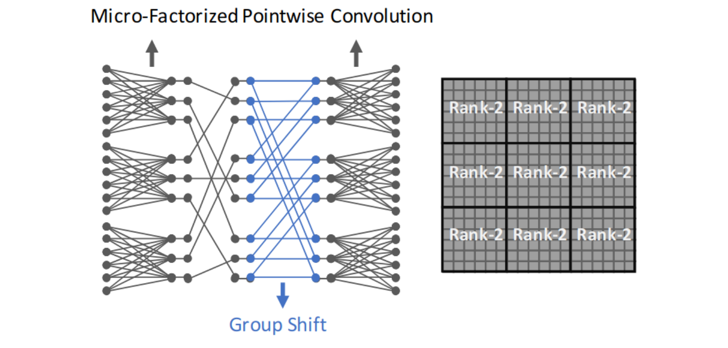


Figure: Dynamic Shift-Max

Group Convolution Summary

- ShuffleNet v1 (2017) – 13x faster than AlexNet (2012), +7% accurate than MobileNet v1 (2017)
  - ❌ Convolution ✅ Group Convolution with feature shuffling
- CondenseNet (2017) – 2x smaller than ShuffleNet (2017)
  - Weights pruning in the early stages of training
  - ❌ Convolution ✅ Group Convolution
  - ❌ Pre-define groups of convolutions ✅ Learn input feature grouping automatically during training
- ShuffleNet v2 (2018) – 58% faster than MobileNet v2 (2018), 63% faster than ShuffleNet v1 (2017)
  - ❌ Channel Shuffle between 1x1 conv and 3x3 DWConv ✅ Channel Shuffle at the end of the block
  - ❌ ReLU after Concat ✅ ReLU before Concat
  - ❌ Two-grouped 1x1 convolution ✅ Split features in two groups before the convolution

Group Convolution Summary

- MixNet (2019) – +4% accurate than MobileNet v2 (2018)
  - ❌ Convolution ✅ MixConv, mixes up multiple kernel sizes in one convolution
- GhostNet (2020) – +1% accurate than MobileNet v3 (2019)
  - ❌ # Feature maps equals # Kernels ✅ Few times more feature maps by applying linear operations on them
- DiCENet (2020) – +3% accurate than MobileNet v2 (2018) and ShuffleNet v2 (2018)
  - ❌ Depth-wise Separable Convolution ✅ DiCE Unit
- MicroNet (2021) – 2x smaller, 3x faster than MobileNet v3 (2019)
  - ❌ Convolution ✅ Micro-Factorized Grouped convolution
  - ❌ ReLU ✅ Dynamic Shift-Max

Squeeze & Excitation Family

Squeeze & Excitation Family

- SqueezeNet (2016)
- SqueezeNeXt (2018)
- SENet (2017)

Squeeze & Excitation Family

SqueezeNet (2016)

- SqueezeNet (2016) – 50x smaller than AlexNet (2012)
  - ❌ Convolution ✅ Fire module

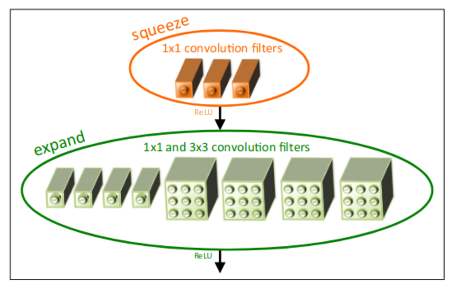


Figure: Organization of convolution filters in the Fire module

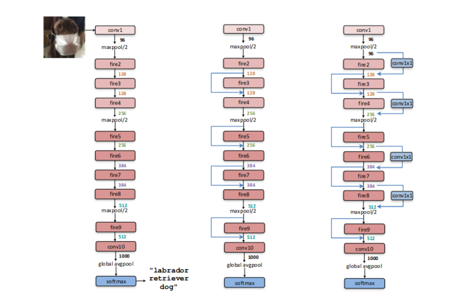


Figure: Macroarchitectural view of our SqueezeNet

SqueezeNeXt (2018)

- SqueezeNeXt (2018) – 1.3x smaller than MobileNet v1 (2017), 2.5x faster SqueezeNet
  - ❌ No residual connection ✅ Residual connection
  - ❌ Convolution ✅ Depth-wise Separable Convolution

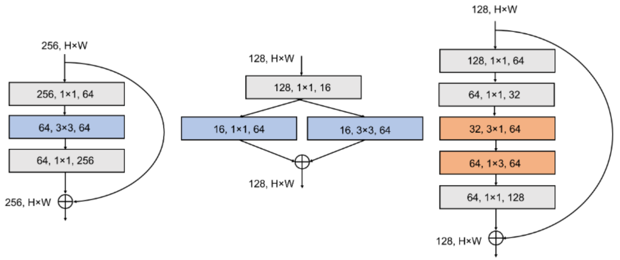


Figure 1: *Illustration of a ResNet block on the left, a SqueezeNet block in the middle, and a SqueezeNext (SqNxt) block on the right. SqueezeNext uses a two-stage bottleneck module to reduce the number of input channels to the* 3×3 *convolution. The latter is further decomposed into separable convolutions to further reduce the number of parameters (orange parts), followed by a* 1×1 *expansion module.*

Figure: SqueezeNeXt

SENet (2017)

- SENet (2017) – +4% accurate than MobileNet (2017) and ResNet (2016)
  - Squeeze-and-Excitation Block

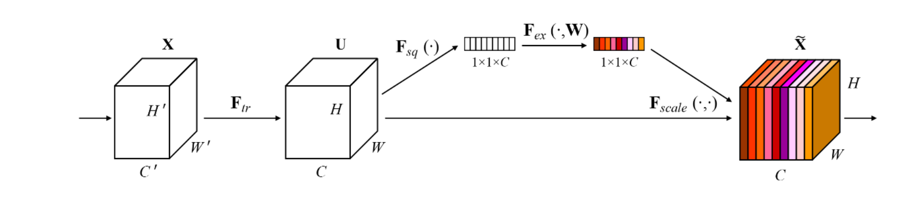


Figure: Squeeze-and-excitation Block

Squeeze & Excitation Summary

- SqueezeNet (2016) – 50x smaller than AlexNet (2012)
  - ❌ Convolution ✅ Fire module
- SqueezeNeXt (2018) – 1.3x smaller than MobileNet v1 (2017), 2.5x faster SqueezeNet
  - ❌ No residual connection ✅ Residual connection
  - ❌ Convolution ✅ Depth-wise Separable Convolution
- SENet (2017) – +4% accurate than MobileNet (2017) and ResNet (2016)
  - Squeeze-and-Excitation Block

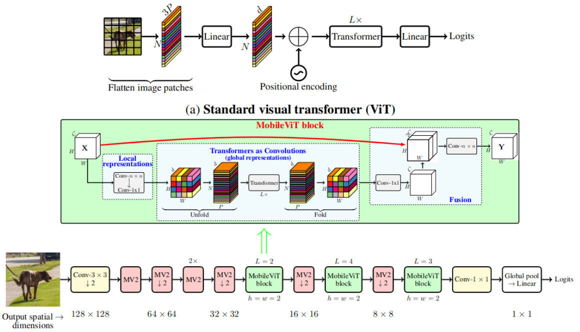
Mobile Transformer Family

Mobile Transformer Family

- MobileViT (2022)
- EdgeViTs (2022)

MobileViT (2022)

- MobileViT (2022) – +3% accurate than MobileNet v3 (2019), +6% accurate than DeIT (2021)
  - ❌ Convolution, Attention ✅ MobileViT Block



(b) **MobileViT**. Here, Conv-$n \times n$ in the MobileViT block represents a standard $n \times n$ convolution and MV2 refers to MobileNetv2 block. Blocks that perform down-sampling are marked with $\downarrow 2$.

Figure: MobileViT

EdgeViTs (2022)

- EdgeViTs (2022) – +12% accurate than MobileViT (2022)
  - Local-Global-Local (LGL) information exchange bottleneck
    - Aggregate information from neighbor tokens with DWConv
    - Sparse delegate tokens for long-range information exchange
    - Transposed convolutions to update information in tokens

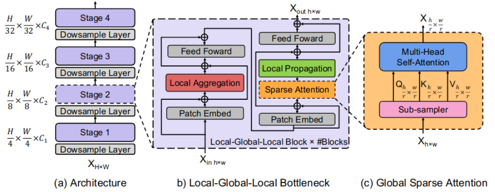


Fig. 2. (a) Schematic overview of our four stages EdgeViT architecture, with each stage consisting of a stack of (b) *Local-Global-Local* (LGL) blocks constructed with local aggregation module, sparse-self-attention and local propagation module, patch embedding (PE) and Feed Forward Network (FFN). In this example, h and w refer to input height and width of stage-2: $h = \frac{H}{8}$ and $w = \frac{W}{8}$. C_i refers to the number of channels for stage-i and r denotes the sub-sampling rate.

Figure: EdgeViT

EdgeViTs (2022) – LGL

Fig. 3. Illustration of three key operations involved in the proposed *Local-Global-Local* (LGL) transformer block. In this example, we showcase how *the target token* (the orange square) at the center conducts information exchange with all the others in three sequential steps: (a) Local information from neighbor tokens within the yellow area is first aggregated to the target token. (b) Global sparse attention is then computed among the target token and other selected delegates in orange color. (c) Global context information encoded in the target token is last propagated to its neighbor *non-delegate* tokens within the pink area.

Figure: EdgeViT

Mobile Transformer Summary

- MobileViT (2022) – +3% accurate than MobileNet v3 (2019), +6% accurate than DeIT (2021)
  - ❌ Convolution, Attention ✅ MobileViT Block
- EdgeViTs (2022) – +12% accurate than MobileViT (2022)
  - Local-Global-Local (LGL) information exchange bottleneck
    - Aggregate information from neighbor tokens with DWConv
    - Sparse delegate tokens for long-range information exchange
    - Transposed convolutions to update information in tokens

Thank you!