201900

MUHAMMAD HASSAN MUKHTAR

BSAI Fall 2020

1. How Compound Scaling helps in improving accuracy in Efficiency-Net?

Compound scaling, as applied in EfficientNets, is a technique that involves scaling multiple dimensions of a neural network architecture simultaneously to improve its efficiency and accuracy. Here's how it helps in improving accuracy in EfficientNets:

Balancing Model Depth, Width, and Resolution: Compound scaling involves uniformly scaling the network width, depth, and resolution simultaneously. This balanced scaling ensures that the model becomes more efficient across different dimensions without sacrificing accuracy. By increasing the depth, width, and resolution of the network in a balanced manner, the model can capture more complex patterns and features in the data, leading to improved accuracy.

Efficient Resource Utilization: By scaling the network dimensions efficiently, compound scaling helps in better utilization of computational resources. Instead of simply increasing the size of the network in one dimension, which might lead to overfitting or inefficient resource usage, compound scaling ensures that resources are allocated optimally across different aspects of the network, thereby improving its overall efficiency.

Improved Generalization: EfficientNets, through compound scaling, are able to achieve better generalization performance. By balancing the model depth, width, and resolution, the network becomes more capable of learning diverse features from the data while avoiding overfitting. This leads to improved generalization performance, allowing the model to perform well on unseen data.

Adaptability to Different Scales: Compound scaling enables EfficientNets to adapt to different scales of datasets and tasks. By scaling the network architecture across multiple dimensions, EfficientNets can efficiently handle a wide range of input resolutions and complexities, making them versatile and suitable for various computer vision tasks, from image classification to object detection and segmentation.

2. How Uniform Scaling is done in Efficiency-Net explain with the help of formula?

Uniform scaling in EfficientNet involves scaling the network width, depth, and resolution simultaneously in a balanced manner using a compound coefficient. Here's how it's done:

Compound Coefficient φ: EfficientNet introduces a compound coefficient denoted as φ. This coefficient uniformly scales the network width (number of channels), depth (number of layers), and resolution (input image size) simultaneously. The compound coefficient serves as a single parameter to control the scaling of the entire network architecture.

Scaling Equations: The scaling equations used in EfficientNet are:

- Depth (d): $d = \alpha^{\varphi}$
- Width (w): $w = \beta^{\varphi}$
- Resolution (r): $r = \gamma^{\varphi}$

Here, α, β, and γ are constants determined through a grid search or experimentation.

Constraints: The constants α, β, and γ are subject to constraints such that:

- $\alpha * \beta^2 * \gamma^2 \approx 2$
- $\alpha \geq 1, \beta \geq 1, \gamma \geq 1$

These constraints ensure that each dimension (depth, width, and resolution) is scaled by at least a factor of 1, and the overall scaling remains balanced.

Implementation: To implement uniform scaling, the compound coefficient φ is specified, and then the depth, width, and resolution of the network are scaled according to the equations mentioned above. This ensures that all dimensions of the network architecture are scaled proportionally to maintain efficiency and accuracy. *For Efficient-Net B0 $\varphi = 1$ and $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.*

Adjusting Model Size: By varying the compound coefficient φ, users can adjust the overall size of the EfficientNet model. A larger φ value leads to a larger model with increased depth, width, and resolution, while a smaller φ value results in a smaller and more efficient model.

3. How is efficiency and accuracy balanced in Efficiency-Net?

Compound Scaling: EfficientNet uses compound scaling, where the network width, depth, and resolution are uniformly scaled together using a compound coefficient (φ). This ensures that as the model size increases or decreases, all dimensions of the network architecture are adjusted in a balanced manner. By maintaining this balance, efficiency (in terms of computational resources) and accuracy can be preserved across different model sizes.

Efficient Building Blocks: EfficientNet employs efficient building blocks, such as *inverted residual blocks* with linear bottlenecks and *squeeze-and-excitation* (SE) blocks. These blocks are carefully designed to maximize representational power while minimizing computational cost. For example, the linear bottleneck reduces the number of parameters and computations in the middle of the block, while the SE block enhances feature interdependencies to improve accuracy.

Efficient Attention Mechanisms: The use of attention mechanisms, such as the SE block mentioned above, allows the model to focus on the most relevant features, thereby enhancing accuracy without significantly increasing computational cost. These mechanisms help the model efficiently allocate computational resources to important regions of the input data.

Regularization and Training Techniques: EfficientNet incorporates regularization techniques like dropout and data augmentation to prevent overfitting and improve generalization performance. Additionally, efficient training strategies, such as mixed-precision training and knowledge distillation, are used to speed up training and inference without sacrificing accuracy.

Model Scaling Strategy: The scaling strategy used in EfficientNet aims to find the optimal balance between model size and performance. By systematically increasing the model size using the compound scaling method and evaluating performance on a validation set, EfficientNet achieves state-of-the-art accuracy while maintaining efficiency across different model sizes.

4. Problem Formulation for Efficiency-Net?

Let's break down each formula:

Formula (1):

$$N = \prod_{i=1 \text{ to } s} (F_i^{L_i}) \circ X_{H_i W_i C_i}$$

N: The overall ConvNet (neural network)

∏ (product symbol): Combines multiple layers

i: Index for each stage (1 to s)

F_i: Operator (function) for each layer

L_i: Number of times layer F_i is repeated in stage i

X_H_iW_iC_i: Input tensor (data) for layer i, with shape (dimensions):

H_i: Spatial dimension (height)

W_i: Spatial dimension (width)

C_i: Channel dimension (features)

This formula represents the ConvNet as a sequence of stages, each with multiple identical layers (F_i repeated L_i times) processing input data with specific dimensions.

Formula (2):

$$\max_{(w,d,r)} \text{Accuracy\_N}(w,d,r)$$

max: Find the maximum value

w, d, r: Scaling coefficients (variables) for:

w: Network width (channel dimension)

d: Network depth (number of layers)

r: Network resolution (spatial dimensions)

Accuracy_N: Measure of the ConvNet performance (accuracy)

This formula aims to find the optimal scaling coefficients (w, d, r) that maximize the ConvNet accuracy as mentioned in Question No 2.

Constraints:

$$N(w,d,r) = \prod_{i=1 \text{ to } s} (F_i^{dL_i}) \circ X_{rH_i\ rW_i\ wC_i}$$

Same as Formula (1), but with scaled dimensions:

dL_i: Scaled number of layer repetitions

rH_i, rW_i: Scaled spatial dimensions

wC_i: Scaled channel dimension

Memory(N) ≤ target memory

Memory(N): Total memory required by the scaled ConvNet

target memory: Maximum allowed memory

FLOPS(N) ≤ target flops

FLOPS(N): Total processing power required by the scaled ConvNet

target flops: Maximum allowed processing power

These constraints ensure the scaled ConvNet stays within the limits of available resources (memory and processing power).

5. How is the distribution and scaling of resources is Done?

The distribution and scaling of resources (memory and processing power) for the ConvNet layers can be explained as follows:

Memory Distribution:

- Each layer F_i has a fixed memory requirement based on its input and output dimensions (H_i, W_i, C_i).
- The total memory required by the ConvNet N is the sum of the memory requirements of all layers: Memory(N) = Σ(i=1 to s) Memory(F_i^L_i).
- The memory requirement for each layer is scaled by the coefficient w (network width): Memory(F_i^L_i) = w * Memory(F_i).
- The total memory required by the scaled ConvNet N(w,d,r) is: Memory(N(w,d,r)) = Σ(i=1 to s) Memory(F_i^dL_i) = Σ(i=1 to s) w * Memory(F_i)

Processing Power Distribution:

- Each layer F_i has a fixed processing power requirement based on its input and output dimensions (H_i, W_i, C_i).
- The total processing power required by the ConvNet N is the sum of the processing power requirements of all layers: FLOPS(N) = Σ(i=1 to s) FLOPS(F_i^L_i).
- The processing power requirement for each layer is scaled by the coefficients d (network depth) and r (network resolution): FLOPS(F_i^dL_i) = d * r^2 * FLOPS(F_i).
- The total processing power required by the scaled ConvNet N(w,d,r) is: FLOPS(N(w,d,r)) = Σ(i=1 to s) FLOPS(F_i^dL_i) = Σ(i=1 to s) d * r^2 * FLOPS(F_i)

Scaling:

- The scaling coefficients w, d, and r are applied to the ConvNet layers to scale the memory and processing power requirements.
- The scaling is done uniformly across all layers, meaning that each layer is scaled by the same factor.
- The scaled ConvNet N(w,d,r) has a total memory requirement of Memory(N(w,d,r)) and a total processing power requirement of FLOPS(N(w,d,r)).

The goal of the optimization problem is to find the optimal scaling coefficients w, d, and r that maximize the accuracy of the ConvNet while satisfying the resource constraints (memory and processing power).

6. Inception vs EfficiencyNet?

EfficientNet and Inception are both popular convolutional neural network (CNN) architectures used for various computer vision tasks like image classification. While they share the goal of improving the efficiency of CNNs, they achieve it through different means.

Scaling Factor:

- EfficientNet introduces a novel scaling method that uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. This ensures that all aspects of the network are scaled simultaneously, leading to better performance.
- Inception also uses scaling, but it primarily focuses on widening the network (increasing the number of filters) rather than scaling depth or resolution as uniformly as EfficientNet.

Depth-wise Separable Convolution:

- Both architectures utilize depth-wise separable convolutions to reduce computational cost. This technique separates the spatial and channel-wise convolutions, significantly reducing the number of parameters and operations.

- EfficientNet emphasizes depth-wise separable convolutions more heavily, especially in the later stages of the network.

Model Complexity:

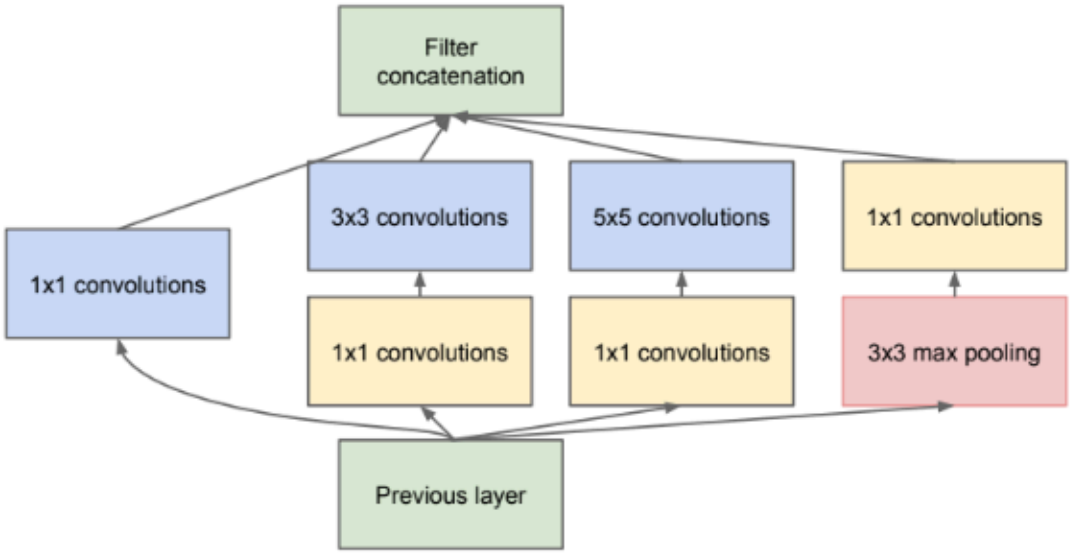
- EfficientNet generally achieves higher accuracy with fewer parameters compared to Inception models. This is because of its more systematic approach to scaling and the use of depth-wise separable convolutions.
- Inception models, while efficient compared to older architectures like AlexNet or VGG, may have more parameters and require more computational resources than similarly performing EfficientNet models.

Model Variants:

- EfficientNet offers different variants (e.g., EfficientNet-B0 to B7) that are scaled versions of the base architecture, allowing for a balance between efficiency and accuracy across different computational budgets.
- Inception also has different versions (e.g., Inception-V1, V2, V3), each introducing improvements over the previous one, but these improvements might not be as directly related to efficiency as in Efficient-Net.

Key Figures:

- EfficientNet has shown state-of-the-art performance on various benchmarks like ImageNet classification while being computationally efficient due to **MBConv** (Mobile Inverted Bottleneck Convolution).
- Inception has been influential in pushing the boundaries of CNN design, especially with the introduction of modules like **Inception Blocks**, but its performance might not match the efficiency of newer architectures like EfficientNet.



Inception module with dimension reductions

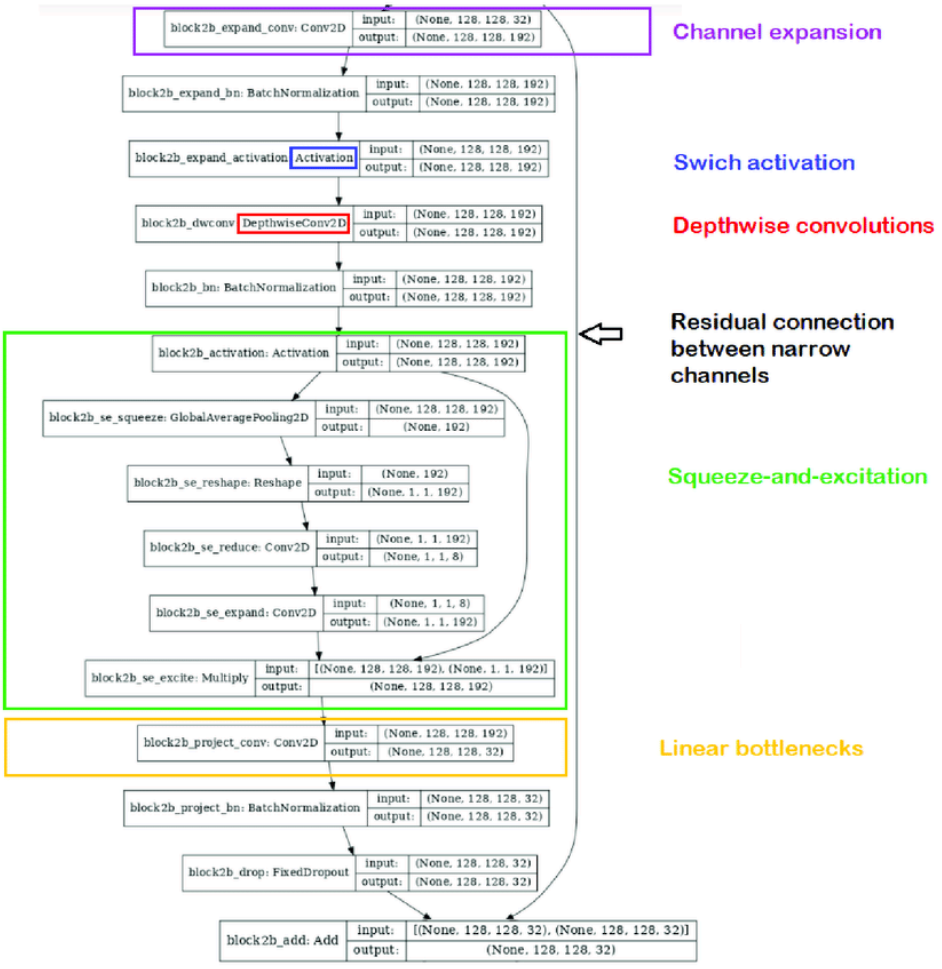


Illustration of MBConv6 with Squeeze-and-Excitation block and inverted residual connection.

201900@students.au.edu.pk

7. Discuss Table 1 Architecture of Efficiency-Net in Paper?

let's break down how each stage in the EfficientNet-B0 baseline network is computed:

Stage 1 (Conv 3x3):

- Operation: A 3x3 convolution is applied to the input image with a resolution of 224x224 pixels.
- Computation: This convolutional layer applies a set of 3x3 filters to the input image, producing feature maps with 32 channels. Each filter computes a weighted sum of the pixel values in a local region of the input image.

Stage 2 (MBConv1,k3x3):

- Operation: A mobile inverted bottleneck convolution (MBConv) with a kernel size of 3x3 is applied.
- Computation: MBConv is a lightweight convolutional block that consists of depthwise separable convolution followed by pointwise convolution. It reduces the input resolution to 112x112 while increasing the number of channels to 16. Depthwise convolution applies a separate filter for each input channel, followed by pointwise convolution that combines the output channels.

Stages 3 to 8 (MBConv6, k3x3 or k5x5):

- Operations: MBConv blocks with kernel sizes of 3x3 or 5x5 are applied.
- Computation: Each MBConv block repeats depthwise separable convolution multiple times, enhancing the depth of features extracted. The kernel sizes vary, with some blocks focusing on capturing spatial patterns with 3x3 convolutions, while others capture larger receptive fields with 5x5 convolutions. The number of channels increases progressively while the resolution decreases.

Stage 9 (Conv1x1 & Pooling & FC):

- Operations: This stage combines a 1x1 convolution, pooling operation, and fully connected layer.
- Computation: The 1x1 convolution reduces the number of channels to compress the feature maps, followed by global average pooling to aggregate spatial information into a single vector. Finally, a fully connected layer (or a linear classifier) maps the feature vector to the output classes. This stage produces a feature representation with a resolution of 7x7 and 1280 channels, ready for classification.

Now let's talk about MBConv

MBConv1:

- MBConv stands for Mobile Inverted Bottleneck Convolution.
- The "1" in MBConv1 refers to the expansion factor, which is the factor by which the number of input channels is expanded before applying the depthwise convolution.

Operation:

- It begins with expanding the number of channels of the input feature maps using a 1x1 convolution with a specific expansion factor (in this case, 1).
- Then, it applies a depthwise separable convolution, which consists of a depthwise convolution (which applies a separate 3x3 filter for each input channel) followed by a 1x1 pointwise convolution (which combines the output channels).
- Finally, a nonlinear activation function like ReLU is applied.

Purpose:

- MBConv1 aims to capture more complex patterns or features without significantly increasing the computational cost compared to traditional convolutional layers.
- It introduces non-linearity and additional capacity to the network, which can help improve its representational power.

MBConv6:

- Similar to MBConv1, MBConv6 also uses the Mobile Inverted Bottleneck Convolution.
- The "6" in MBConv6 refers to the expansion factor, which means the number of input channels is expanded six times before applying the depthwise convolution.

Operation:

- It starts by expanding the number of channels of the input feature maps using a 1x1 convolution with a larger expansion factor (in this case, 6).
- Then, it applies a depthwise separable convolution similar to MBConv1.
- Finally, a nonlinear activation function is applied.

Purpose:

- MBConv6 expands the feature maps more aggressively compared to MBConv1, allowing the network to capture richer and more complex patterns.
- Despite the higher computational cost due to the larger expansion factor, MBConv6 can potentially improve the network's expressive power and performance, especially for deeper layers where more complex features are learned.