EfficientNet Architecture Overview

give me the entire architecture of efficientnet

EfficientNet is a family of convolutional neural network architectures introduced by **Tan and Le (2019)** in the paper “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. It scales **depth, width, and resolution** using a **compound coefficient** to achieve better accuracy and efficiency.

🔍 EfficientNet: Key Concepts

1. Compound Scaling

Instead of scaling dimensions arbitrarily, EfficientNet uses a **compound coefficient ϕ** to scale:

- **Depth (d)** → more layers
- **Width (w)** → more channels
- **Resolution (r)** → larger input images

The scaling is defined as:

$$\begin{aligned} \text{depth: } d &= \alpha^{\phi} \\ \text{width: } w &= \beta^{\phi} \\ \text{resolution: } r &= \gamma^{\phi} \\ \text{subject to: } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \end{aligned}$$

🏗️ EfficientNet Architecture Overview

EfficientNet builds upon **MBConv blocks** (Mobile Inverted Bottleneck Convs) with:

- Depthwise separable convolutions
- Squeeze-and-Excitation blocks

- Swish activation (later SiLU)

The base model is **EfficientNet-B0**, and others like B1 to B7 are scaled versions.

📐 EfficientNet-B0 Architecture (Baseline)

Here is a detailed block-wise architecture for **EfficientNet-B0**:

Stage	Operator	Input Resolution	Channels (out)	Repeats	Kernel	Stride	SE Ratio
1	Conv3x3	224×224	32	1	3×3	2	-
2	MBConv1, k3x3	112×112	16	1	3×3	1	0.25
3	MBConv6, k3x3	112×112	24	2	3×3	2	0.25
4	MBConv6, k5x5	56×56	40	2	5×5	2	0.25
5	MBConv6, k3x3	28×28	80	3	3×3	2	0.25
6	MBConv6, k5x5	14×14	112	3	5×5	1	0.25
7	MBConv6, k5x5	14×14	192	4	5×5	2	0.25
8	MBConv6, k3x3	7×7	320	1	3×3	1	0.25
9	Conv1x1 + Pool + FC	7×7	1280	-	1×1	1	-

> ⚠️ **MBConv stands for Mobile Inverted Bottleneck Convolution.**

🔄 MBConv Block Breakdown

Each MBConv block consists of:

1. Expansion (Conv 1×1)

2. Depthwise Convolution (k×k)
3. SE block (Squeeze-and-Excitation)
4. Projection (Conv 1×1)
5. Residual connection (if input == output and stride = 1)

🧠 EfficientNet Variants (B1–B7)

They are scaled versions of B0 using compound scaling:

Model	φ	Resolution	Depth	Width	Parameters	Top-1 Accuracy
EfficientNet-B0	0	224×224	1.0×	1.0×	5.3M	77.1%
EfficientNet-B1	1	240×240	1.1×	1.0×	7.8M	79.1%
EfficientNet-B2	2	260×260	1.2×	1.1×	9.2M	80.1%
EfficientNet-B3	3	300×300	1.4×	1.2×	12M	81.6%
EfficientNet-B4	4	380×380	1.8×	1.4×	19M	83.0%
EfficientNet-B5	5	456×456	2.2×	1.6×	30M	83.6%
EfficientNet-B6	6	528×528	2.6×	1.8×	43M	84.0%
EfficientNet-B7	7	600×600	3.1×	2.0×	66M	84.4%

🔗 Official and Reference Implementations

- **Official TensorFlow implementation**:
  https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet
  (Highly recommended if you want paper-level fidelity)

- **PyTorch implementation** (popular):
  https://github.com/lukemelas/EfficientNet-PyTorch
- **Keras (TF 2.x):**
  https://github.com/qubvel/efficientnet

💡 Bonus: EfficientNet-Lite

- Optimized for **mobile/edge devices**
- Removed **swish activation** (replaced by ReLU6)
- Smaller SE blocks

If you'd like a **modular PyTorch or Keras code walkthrough of EfficientNet-B0**, I can provide that next. Would you like it layer-by-layer?

explain MBConv6, k3x3 w.r.t se ratio 0.25 completely

Absolutely. Let's **decompose and deeply understand** the **MBConv6, k3x3 block** with **Squeeze-and-Excitation (SE) ratio 0.25**, which is a **core building block of EfficientNet.**

🔍 What is MBConv?

MBConv stands for **Mobile Inverted Bottleneck Convolution**, first introduced in MobileNetV2 and later enhanced with **SE blocks** and **Swish activation** in EfficientNet.

⚙️ MBConv6, k3x3 (with SE ratio 0.25)

🔧 Specifications:

- **6** → Expansion factor (input channels × 6)
- **k3x3** → Kernel size of the depthwise convolution
- **SE ratio = 0.25** → SE bottleneck channels = 0.25 × expanded channels

---

📐 Full Architecture of MBConv6, k3x3

Assume:

- Input tensor: shape **(B, H, W, C)** — i.e., batch × height × width × input channels
- Let's say `C = 32` (input channels)

Now step-by-step:

---

1. 🔼 Expansion Phase (Pointwise Conv1x1)

- Expand channels:

$$\text{Expanded Channels} = C_{exp} = C \times 6 = 32 \times 6 = 192$$

- Apply **1x1 convolution** to map from 32 → 192 channels
- Shape: **(B, H, W, 192)**
- Followed by:
  - **BatchNorm**
  - **Swish (SiLU)** activation

---

2. 🔁 Depthwise Convolution (Kernel = 3×3)

- Apply a **depthwise separable convolution** (1 filter per input channel, not across channels)
- Kernel size = 3×3, **stride = 1 or 2**

- Output shape: still **(B, H, W, 192)** (if stride = 1)
- Followed by:
  - **BatchNorm**
  - **Swish activation**

---

3. 💡 Squeeze-and-Excitation (SE) Block

This improves channel-wise attention. Here's the breakdown:

a. 🔽 Squeeze

- Perform **Global Average Pooling** → output shape: **(B, 192, 1, 1)** → flattened to **(B, 192)** (One scalar per channel)

b. 🔄 Excitation

- First Dense Layer:

$$\text{Reduced Channels} = 192 \times 0.25 = 48$$

  - Linear (FC) → shape becomes (B, 48)
  - Activation: **Swish**
- Second Dense Layer:
  - Linear → back to (B, 192)
  - Activation: **Sigmoid**
- Output: channel-wise scale factors in range (0, 1)

c. 🔁 Recalibrate

- Multiply these 192 scale factors back with the depthwise output (broadcasted across H and W):

$$\text{Output shape still: } (B, H, W, 192)$$

---

4. 🔽 Projection Phase (Pointwise Conv1x1)

- **1x1 convolution** to project back to original channel size: **192 → 32**
- Shape: **(B, H, W, 32)**
- Followed by:
  - **BatchNorm**
  - **No activation**

---

5. 🔁 Residual Connection (Skip Connection)

- If **stride = 1** and **input_channels == output_channels**:

$$\text{Output} = \text{Input} + \text{Projected Output}$$

- Else → no skip connection

🔄 Summary Table for MBConv6, k3x3, SE=0.25

Phase	Operation	Channels (in→out)	Notes
Expansion	1×1 Conv + BN + Swish	32 → 192	Expand channels
Depthwise	3×3 DW Conv + BN + Swish	192 → 192	Kernel = 3×3
SE Block	GAP → FC(48) → FC(192)	192 → 48 → 192	SE ratio = 0.25
Projection	1×1 Conv + BN	192 → 32	Reduce back to original dim
Residual	Add input (if possible)	32 + 32 → 32	Only if stride = 1

📊 Visual Diagram

```scss
```

```
Input (B, H, W, 32)
    │
    ├── Conv1x1 (expand to 192) ── BN ── Swish
    │       ↓
    ├── Depthwise Conv 3x3 ────── BN ── Swish
    │       ↓
    ├── Global Avg Pool (→192)
    │       ↓
    ├── FC → Swish → FC → Sigmoid (→192)
    │       ↓
    ├── Multiply with feature map (channel-wise)
    │       ↓
    ├── Conv1x1 (192 → 32) ─────── BN
    │       ↓
    ├── Skip Connection (if stride=1 & in=out)
    ▼
Output (B, H, W, 32)
```

🧪 PyTorch Code Reference (Simplified)

```python
class MBConv(nn.Module):
    def __init__(self, in_channels, out_channels, expand_ratio=6, se_ratio=0.25, kernel_size=3, stride=1):
        super().__init__()
        mid_channels = in_channels * expand_ratio

        self.use_residual = (in_channels == out_channels and stride == 1)

        self.expand = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, 1, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.SiLU()
        ) if expand_ratio != 1 else nn.Identity()

        self.dwconv = nn.Sequential(
            nn.Conv2d(mid_channels, mid_channels, kernel_size, stride,
                      padding=kernel_size//2, groups=mid_channels, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.SiLU()
        )

        se_channels = int(mid_channels * se_ratio)
        self.se = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(mid_channels, se_channels, 1),
            nn.SiLU(),
            nn.Conv2d(se_channels, mid_channels, 1),
            nn.Sigmoid()
        )

        self.project = nn.Sequential(
            nn.Conv2d(mid_channels, out_channels, 1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

    def forward(self, x):
        identity = x
        x = self.expand(x)
        x = self.dwconv(x)
        x = x * self.se(x)
        x = self.project(x)
        return x + identity if self.use_residual else x
```

Would you like this written in **TensorFlow/Keras** too, or want me to help you visualize **all MBConv blocks in EfficientNet-B0 layer-by-layer with shapes**?

yes

Great! Let's now go **layer-by-layer through EfficientNet-B0**, explaining each **MBConv block with shapes, SE application**, and **what happens at each step**, so you can visualize the architecture clearly and code it from scratch.

🧠 EfficientNet-B0 Detailed Layer-by-Layer Architecture

> Input: **224×224×3**
> SE Ratio: **0.25**
> Activation: **Swish**
> Note: `MBConv6` → expansion ratio = 6, `MBConv1` → no expansion

✅ Stage 0: Stem

Layer	Type	Output Shape	Notes
Conv2D	3×3, stride=2	112×112×32	Initial conv layer
BN + Swish	-	112×112×32	

✅ Stage 1: MBConv1, k3x3 (No expansion)

Layer	Type	Output Shape	Notes
MBConv1	k=3, stride=1	112×112×16	No expansion, SE applied, 1 block

🔁 SE Block:

- Squeeze → 16 → FC → 4 → FC → 16 → Sigmoid
- Scale: (B, 112, 112, 16) × (B, 1, 1, 16)

✅ Stage 2: MBConv6, k3x3, 2 blocks

Layer	Type	Output Shape	Notes
MBConv6_1	stride=2	56×56×24	Expand 16→96→SE→96→24, no residual
MBConv6_2	stride=1	56×56×24	Residual connection enabled

🔁 **SE Block per block:**

- Squeeze → 96 → FC → 24 → FC → 96 → Sigmoid

✅ Stage 3: MBConv6, k5x5, 2 blocks

Layer	Type	Output Shape	Notes
MBConv6_1	stride=2	28×28×40	Expand 24→144→SE→144→40
MBConv6_2	stride=1	28×28×40	Residual enabled

🔁 **SE Block:**

- SE channels = 144 × 0.25 = 36

✅ Stage 4: MBConv6, k3x3, 3 blocks

Layer	Type	Output Shape	Notes
MBConv6_1	stride=2	14×14×80	Expand 40→240→SE→240→80
MBConv6_2	stride=1	14×14×80	Residual
MBConv6_3	stride=1	14×14×80	Residual

🔁 **SE Block:**

- SE channels = 240 × 0.25 = 60

✅ Stage 5: MBConv6, k5x5, 3 blocks

Layer	Type	Output Shape	Notes
MBConv6_1	stride=1	14×14×112	Expand 80→480→SE→480→112
MBConv6_2	stride=1	14×14×112	Residual
MBConv6_3	stride=1	14×14×112	Residual

🔁 SE Block:

- SE channels = 480 × 0.25 = 120

✅ Stage 6: MBConv6, k5x5, 4 blocks

Layer	Type	Output Shape	Notes
MBConv6_1	stride=2	7×7×192	Expand 112→672→SE→672→192
MBConv6_2/3/4	stride=1	7×7×192	Residual

🔁 SE Block:

- SE channels = 672 × 0.25 = 168

✅ Stage 7: MBConv6, k3x3, 1 block

Layer	Type	Output Shape	Notes
MBConv6	stride=1	7×7×320	Expand 192→1152→SE→1152→320

🔁 SE Block:

- SE channels = 1152 × 0.25 = 288

✅ Stage 8: Head

Layer	Type	Output Shape	Notes
Conv2D	1×1, 320→1280	7×7×1280	Final conv layer before pooling
BN + Swish	-	7×7×1280	
Global AvgPool	-	1×1×1280	Reduces spatial dims
Dropout	-	1280	Dropout(p=0.2)
FC	-	1000	ImageNet classifier

🧱 Full Block Table Summary (EfficientNet-B0)

Stage	Block Type	Repeats	Input Res	Output Channels	Expansion	Kernel	Stride	SE Channels
0	Conv3×3	1	224×224	32	-	3×3	2	-
1	MBConv1	1	112×112	16	1	3×3	1	4
2	MBConv6	2	112×112	24	6	3×3	2,1	24
3	MBConv6	2	56×56	40	6	5×5	2,1	36
4	MBConv6	3	28×28	80	6	3×3	2,1	60
5	MBConv6	3	14×14	112	6	5×5	1	120
6	MBConv6	4	14×14	192	6	5×5	2,1	168

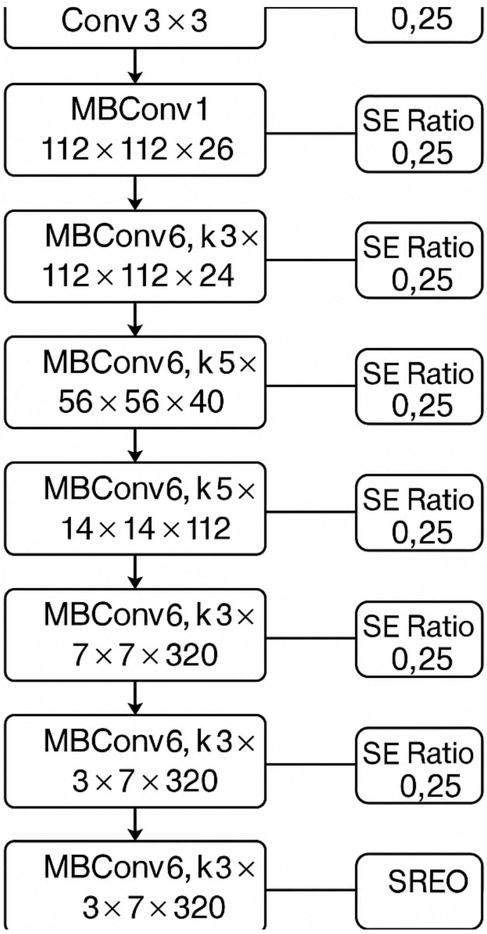
Stage	Block Type	Repeats	Input Res	Output Channels	Expansion	Kernel	Stride	SE Channels
7	MBConv6	1	7×7	320	6	3×3	1	288
8	Conv1x1 + Pool	1	7×7	1280	-	1×1	1	-

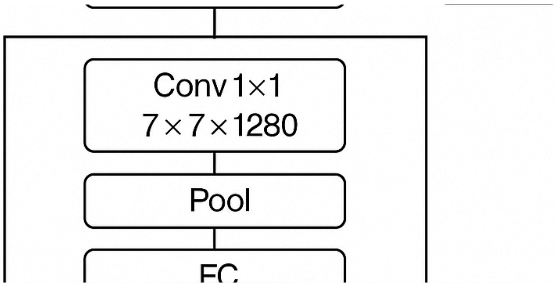
Would you like me to now:

- 🧱 Generate a **full Keras or PyTorch class for EfficientNet-B0** (modular)?
- 📊 Visualize the block diagram?
- 🧪 Write minimal reproducible MBConv code with SE block?

Let me know which direction you want next!

Visualize the block diagram?





why even need squeeze excitation block

Great question. The **Squeeze-and-Excitation (SE) block** plays a critical role in **channel-wise attention**, which enhances the representational power of a CNN without significant computational cost. Let's break it down:

🎯 Objective:

To allow the network to **learn which channels are more important** for a given task and **reweight them adaptively**.

🧠 Intuition Behind SE Block

CNNs process features spatially and across channels, but **not all channels contribute equally** to the final prediction. Some may represent edges, others colors, textures, etc. SE blocks **learn to emphasize** the most informative channels while **suppressing less useful** ones.

Think of it as a **soft attention mechanism** over the channels.

🔧 How It Works: (Step-by-Step in MBConv)

Assume feature map `X` of shape `(B, H, W, C)`

1. Squeeze

- Global Average Pooling → shape becomes `(B, C)`
- Captures **global context** per channel (like summary stats)

$$z_c = \frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} X_{i,j,c}$$

2. Excitation

- Pass through 2 FC layers:
    - FC1 reduces dimensions → `C → C × r` (usually r=0.25)
    - Activation: Swish or ReLU
    - FC2 expands → `C × r → C`
    - Activation: **Sigmoid** → values between 0 and 1 (soft gates)

This gives a vector **s** ∈ $\mathbb{R}^c$ of weights.

3. Scale

- Multiply original feature map channel-wise with `s`
- Broadcast `s` across spatial dimensions

$$\tilde{X}_c = s_c \cdot X_c$$

📈 Why It's Needed (Benefits)

Benefit	Description
Adaptive recalibration	Learns which channels are important and adjusts dynamically per input
Boosts performance	Consistently improves accuracy on image classification, detection, etc.
Minimal cost	Adds very few parameters and FLOPs (esp. with bottleneck reduction ratio `r=0.25`)
Encodes global context	Squeeze step captures context beyond local receptive field
Plug-and-play	Can be added to almost any CNN block (ResNet, MobileNet, etc.)

🧪 Real-world Analogy

Imagine you’re analyzing a news article with multiple sections: Politics, Tech, Sports, Health.

> SE block is like a reader who first **skims the whole article** (squeeze), then decides, **“Tech and Health seem more relevant today,”** (excitation), and focuses more on those (scale).

📚 Evidence in Research

- Introduced in: **"Squeeze-and-Excitation Networks" (Hu et al., 2018)**
  📄 Paper
- Achieved **ImageNet top-1 accuracy of 85.1%** on SENet-154
- Now used in:
  - EfficientNet (SE + MBConv)
  - ResNeSt (Split-Attention)
  - MobileNetV3 (Lite SE)
  - YOLOv5 (optional)

🚫 Without SE block

If you **remove SE**, the network treats all channels equally. It still learns discriminative filters, but lacks an **explicit mechanism** to modulate their importance **based on global context** of the input.

✅ When is SE most effective?

- When channels represent **semantically diverse information**
- In **deeper** models (EfficientNet, ResNeXt, etc.)
- Tasks requiring **fine-grained discrimination** (e.g., medical imaging, Re-ID)

Would you like me to show:

- Minimal PyTorch/Keras code for SE block?
- Visualization of how it adjusts channel activations before/after?

Let me know how deep you want to go into SE mechanics.

simple code in keras with comments

Here’s a **simple and clean Keras implementation of a Squeeze-and-Excitation (SE) block**, with detailed comments explaining each step. You can plug this into any CNN block like MBConv, ResNet, etc.

✅ Keras Implementation of SE Block

python

```python
from tensorflow.keras import layers

def se_block(input_tensor, se_ratio=0.25):
    """
    Squeeze-and-Excitation Block

    Args:
        input_tensor: Input feature map (B, H, W, C)
        se_ratio: Reduction ratio for bottleneck (default = 0.25)

    Returns:
        Tensor with SE channel-wise recalibration applied
    """
    channels = input_tensor.shape[-1]  # Get the number of channels (C)
    reduced_channels = max(1, int(channels * se_ratio))  # C * 0.25

    # Squeeze: Global Average Pooling → (B, C)
    se = layers.GlobalAveragePooling2D()(input_tensor)

    # Reshape to (B, 1, 1, C) to match input spatial dims
    se = layers.Reshape((1, 1, channels))(se)

    # Excitation: FC1 (reduce) → Swish → FC2 (expand) → Sigmoid
    se = layers.Conv2D(reduced_channels, kernel_size=1, activation='swish')(se)
    se = layers.Conv2D(channels, kernel_size=1, activation='sigmoid')(se)

    # Scale: Multiply original input by the excitation weights
    output_tensor = layers.Multiply()([input_tensor, se])

    return output_tensor
```

🧪 Example Usage in a Conv Block:

```python
def example_conv_block_with_se(input_tensor):
    x = layers.Conv2D(64, 3, padding='same', activation='swish')(input_tensor)
    x = layers.BatchNormalization()(x)

    # Apply SE block
    x = se_block(x, se_ratio=0.25)

    return x
```

Let me know if you want:

- 🔁 Integration inside **MobileNet/ResNet/EfficientNet**
- 🔍 Visualization of SE outputs

- 🔄 Variant: Lite SE (used in MobileNetV3)

Just say the word!

code MBConv6, k3x3, SE=0.25

Here's a complete and clean **Keras implementation of the** `MBConv6, k3x3, SE=0.25` **block** — as used in EfficientNet. It includes:

- **Expansion (Conv1x1)**
- **Depthwise convolution (3x3)**
- **Squeeze-and-Excitation block (ratio=0.25)**
- **Projection (Conv1x1)**
- **Residual connection (if applicable)**

✅ `mbconv_block()` in Keras (MBConv6, k3x3, SE=0.25)

python

```python
from tensorflow.keras import layers

def se_block(input_tensor, se_ratio=0.25):
    """Squeeze-and-Excitation block"""
    channels = input_tensor.shape[-1]
    reduced_channels = max(1, int(channels * se_ratio))

    se = layers.GlobalAveragePooling2D()(input_tensor)
    se = layers.Reshape((1, 1, channels))(se)
    se = layers.Conv2D(reduced_channels, kernel_size=1, activation='swish')(se)
    se = layers.Conv2D(channels, kernel_size=1, activation='sigmoid')(se)
    return layers.Multiply()([input_tensor, se])
```

---

python

```python
def mbconv_block(inputs, in_channels, out_channels, kernel_size=3, stride=1, expand_ratio=6, se_ratio=0.25):
    """
    MBConv6 Block with:
    - Expansion
    - Depthwise Convolution (3x3)
    - Squeeze-and-Excitation
    - Projection
    - Optional Residual
    """
    # === Expansion phase ===
    x = inputs
    expanded_channels = in_channels * expand_ratio

    if expand_ratio != 1:
        x = layers.Conv2D(expanded_channels, kernel_size=1, padding='same', use_bias=False)(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation('swish')(x)

    # === Depthwise convolution ===
    x = layers.DepthwiseConv2D(kernel_size=kernel_size, strides=stride, padding='same', use_bias=False)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('swish')(x)

    # === Squeeze-and-Excitation ===
    x = se_block(x, se_ratio=se_ratio)

    # === Projection phase ===
    x = layers.Conv2D(out_channels, kernel_size=1, padding='same', use_bias=False)(x)
    x = layers.BatchNormalization()(x)

    # === Residual connection ===
    if stride == 1 and in_channels == out_channels:
        x = layers.Add()([x, inputs])

    return x
```

🧪 Example Usage:

```python
from tensorflow.keras import Input, Model

# Input shape: (Batch, H, W, Channels)
inp = Input(shape=(56, 56, 24))
x = mbconv_block(inp, in_channels=24, out_channels=24, kernel_size=3, stride=1,
expand_ratio=6, se_ratio=0.25)
model = Model(inputs=inp, outputs=x)
model.summary()
```

🔄 Notes:

- You can change `kernel_size=5` for `MBConv6, k5x5`.
- `stride=2` for downsampling.
- Use this inside a loop to build EfficientNet stages.
- For expansion ratio = 1, expansion phase is skipped.

Want the full **EfficientNet-B0 model built using these blocks** next? Or a version with skip/residual visualization?