LATEST ARTIFICIAL INTELIGENCE PAPER STUDY

# EfficientNet:
# Rethinking Model Scaling for Convolutional Neural Networks

IMAGE CLASSIFICATION

5th week

JESS RYU

# INDEX

## 1. COMPOUND SCALING

- WHAT IS MODEL SCALING IN DEEP LEARNING?
- INTRODUCE ABOUT COMPOUND SCALING
- THE RESULT USING COMPOUND SCALIING

## 2. EFFICIENTNET

- WHAT IS EFFICIENTNET?
- EFFICIENTNET'S ARCHITECTURE
- COMPARE WITH OTHER MODELS

# 1. COMPOUND SCALING

# WHAT IS MODEL SCALING IN DEEP LEARNING?

- Model scaling is that changes model's width, depth, resolution to improve model's performance.

- Width: the number of channels
- Depth: the number of layers
- Resolution: up pixel numbers of input image

# THE EFFECT OF SCALING

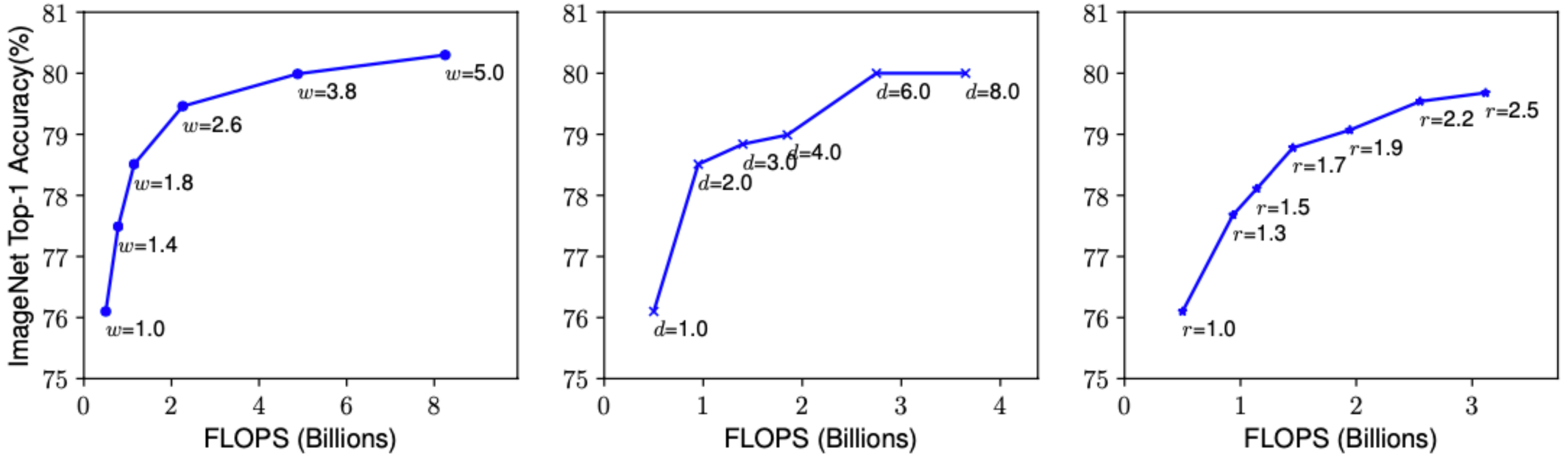
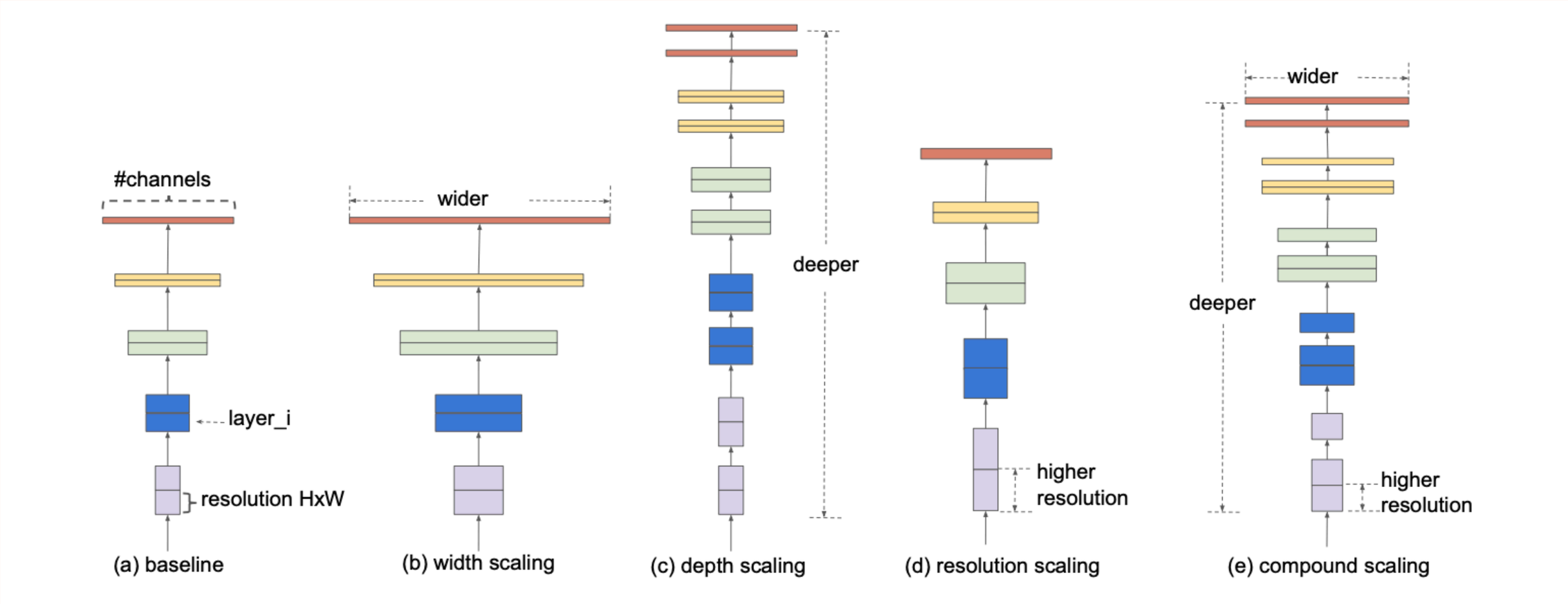



*Figure 3.* **Scaling Up a Baseline Model with Different Network Width ($w$), Depth ($d$), and Resolution ($r$) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

FLOPS : floating point operations per second

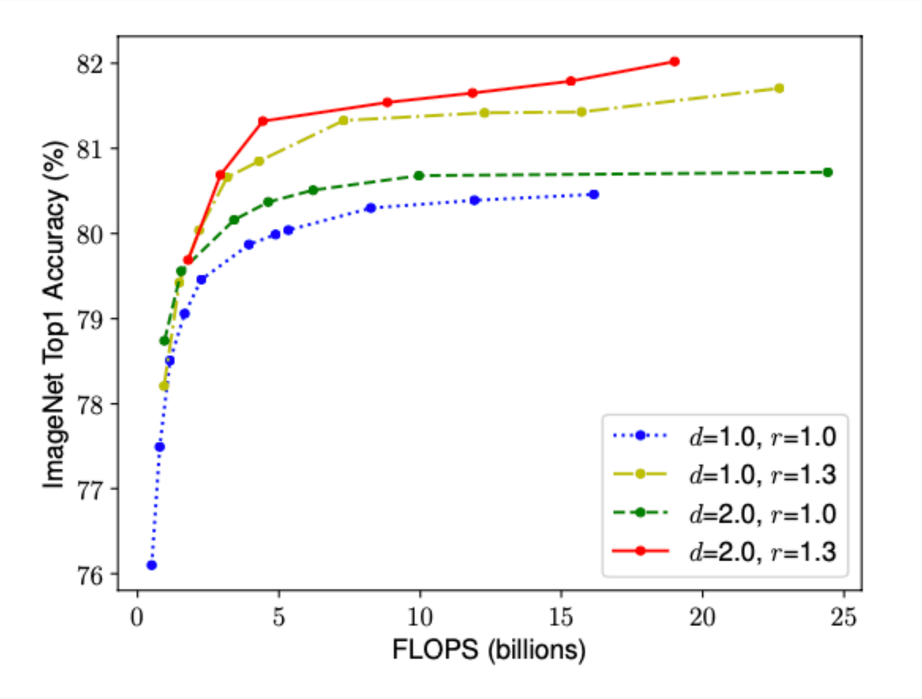
# HOW TO FIND PERFECT BALANCE?




# THE ANSWER IS COMPOUND SCALING

# INTRODUCE ABOUT COMPOUND SCALING

- Compound scaling is a method to find good balanced scaling factors




- width is fixed
- depth same + resolution different
  : acc increases about 1%
- resolution same + depth different
  : acc increases about 0.3%
- The best fator's ratio is red line

# INTRODUCE ABOUT COMPOUND SCALING

$$\text{depth: } d = \alpha^{\phi}$$

$$\text{width: } w = \beta^{\phi}$$

$$\text{resolution: } r = \gamma^{\phi}$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

- **Compound scaling notation**

  **Compound coefficient: φ (fai)**
  **α, β, γ are constants (small grid search)**

- **1. Select model**
  **2. Fix φ = 1**
  **3. Find α, β, γ by small grid search**
  **4. Fix α, β, γ**
  **5. Control φ**

# THE RESULT USING COMPOUND SCALING

*Table 3.* **Scaling Up MobileNets and ResNet.**

| Model | FLOPS | Top-1 Acc. |
|---|---|---|
| Baseline MobileNetV1 (Howard et al., 2017) | 0.6B | 70.6% |
| Scale MobileNetV1 by width ($w$=2) | 2.2B | 74.2% |
| Scale MobileNetV1 by resolution ($r$=2) | 2.2B | 72.7% |
| **compound scale ($d$=1.4, $w$=1.2, $r$=1.3)** | **2.3B** | **75.6%** |
| Baseline MobileNetV2 (Sandler et al., 2018) | 0.3B | 72.0% |
| Scale MobileNetV2 by depth ($d$=4) | 1.2B | 76.8% |
| Scale MobileNetV2 by width ($w$=2) | 1.1B | 76.4% |
| Scale MobileNetV2 by resolution ($r$=2) | 1.2B | 74.8% |
| **MobileNetV2 compound scale** | **1.3B** | **77.4%** |
| Baseline ResNet-50 (He et al., 2016) | 4.1B | 76.0% |
| Scale ResNet-50 by depth ($d$=4) | 16.2B | 78.1% |
| Scale ResNet-50 by width ($w$=2) | 14.7B | 77.7% |
| Scale ResNet-50 by resolution ($r$=2) | 16.4B | 77.5% |
| **ResNet-50 compound scale** | **16.7B** | **78.8%** |

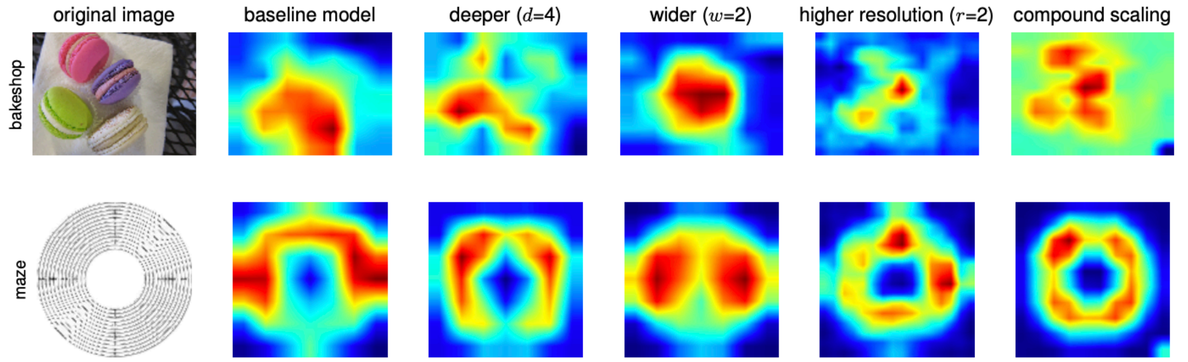



*Figure 7.* **Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods-** Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details. Model details are in Table 7.

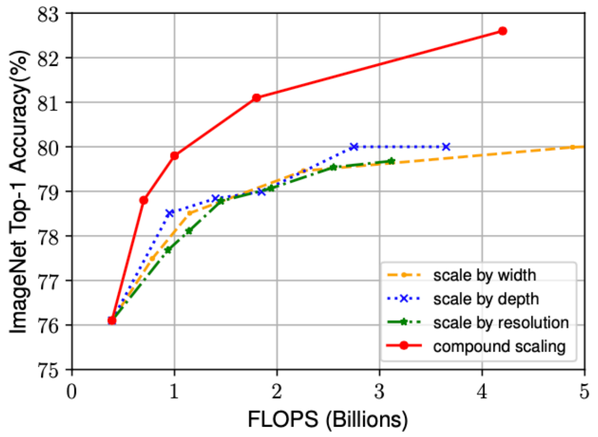



*Figure 8.* **Scaling Up EfficientNet-B0 with Different Methods.**

*Table 7.* **Scaled Models Used in Figure 7.**

| Model | FLOPS | Top-1 Acc. |
|---|---|---|
| Baseline model (EfficientNet-B0) | 0.4B | 77.3% |
| Scale model by depth ($d$=4) | 1.8B | 79.0% |
| Scale model by width ($w$=2) | 1.8B | 78.9% |
| Scale model by resolution ($r$=2) | 1.9B | 79.1% |
| **Compound Scale ($d$=1.4, $w$=1.2, $r$=1.3)** | **1.8B** | **81.1%** |

# 2. EFFICIENTNET

# WHAT IS EFFICIENTNET?

- proposed mobile-size baseline network model
- The model is designed for mobile from AutoML
- Performance good & Param's number & FLOPS down
- It is similar to MnasNET (neural architecture search)

**Flow: mobileNetV1 -> mobileNetV2 -> MnasNet -> efficientNet**

# EFFICIENTNET'S ARCHITECTURE

*Table 1.* **EfficientNet-B0 baseline network** – Each row describes a stage $i$ with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

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

- **Swish activation**
- **Squeeze & Excitation Block (SE Block, SE Net)**
- **Mobile Inverted Residual Block (MBConv)**

| EN B0 | kernel size | Repeat | input filter | output filter | expand ratio | se_ratio | strides |
|---|---|---|---|---|---|---|---|
| MBConv1 (Block1) | 3 | 1 | 32 | 16 | 1 | 0.25 | 1 |
| MBConv6 (Block2) | 3 | 2 | 16 | 24 | 6 | 0.25 | 2 |
| MBConv6 (Block3) | 5 | 2 | 24 | 40 | 6 | 0.25 | 2 |
| MBConv6 (Block4) | 3 | 3 | 40 | 80 | 6 | 0.25 | 2 |
| MBConv6 (Block5) | 5 | 3 | 80 | 112 | 6 | 0.25 | 1 |
| MBConv6 (Block6) | 5 | 4 | 112 | 192 | 6 | 0.25 | 2 |
| MBConv6 (Block7) | 3 | 1 | 192 | 320 | 6 | 0.25 | 1 |

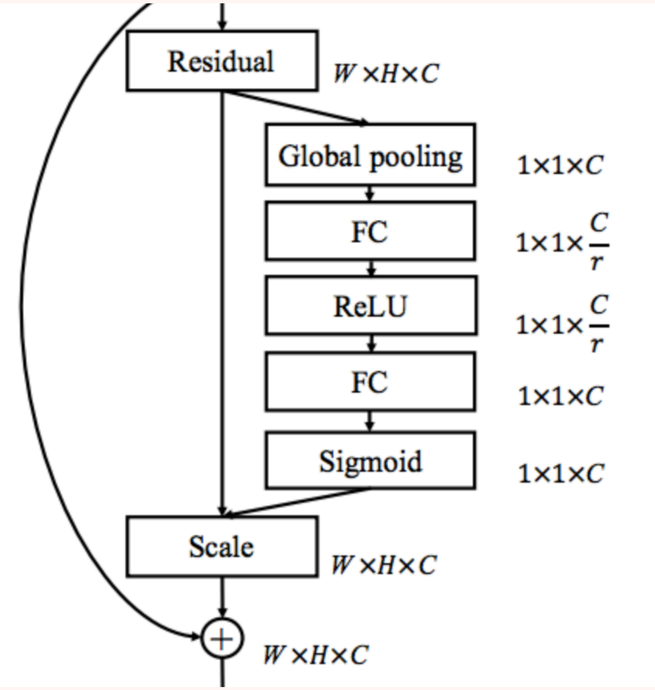
# SWISH ACTIVATION

- Google brain team suggested a new activation
- It is better than Relu  ( solved nullifies negetive values)
- Formula : swish(x)  = x * sigmoid(x)

```
# x * sigmoid(x)

def swish_activation(x):
    return x * (1 / (1+np.exp(-x)))
```

# SQUEEZE AND EXCITATION BLOCK




- Find Each channel feature -> 1 x 1 feature map
- Improve Performance > increase FLOPS = useful

```
def se_block(x, filter_num, squeeze_ratio):
    se = tf.keras.layers.GlobalAveragePooling2D()(x)
    se = tf.keras.layers.Reshape((1,1,filter_num))(se)
    squeezed_filters = max(1, int(filter_num * squeeze_ratio))
    se = tf.keras.layers.Conv2D(squeezed_filters,(1,1),activation='relu')(se)
    se = tf.keras.layers.Conv2D(filter_num,(1,1),activation='sigmoid')(se)
    return multiply()([x, se])
```

# INVERTED RESIDUAL BLOCK

```
def residual_block(x, squeeze, expand):
    re = tf.keras.layers.Conv2D(squeeze, (1,1),padding='same', activation='relu')(x)
    re = tf.keras.layers.Conv2D(squeeze, (3,3),padding='same', activation='relu')(re)
    re = tf.keras.layers.Conv2D(expand, (1,1),padding='same', activation='relu')(re)
    return Add()([re, x])
```

```
def inverted_residual_block(x, expand, squeeze):
    block = tf.keras.layers.Conv2D(expand, (1,1),padding='same',activation='relu')(x)
    block = tf.keras.layers.DepthwiseConv2D((3,3),padding='same', activation='relu')(block)
    block = tf.keras.layers.Conv2D(squeeze, (1,1),padding='same', activation='relu')(block)
    return Add()([block, x])
```

# MOBILE INVERTED RESIDUAL BLOCK

```python
def mbConv_block(
    input_data,repeat_num, kernel_size,input_filter,output_filter,expand_ratio,se_ratio,strides, drop_ratio):


    # expansion phase (1,1)
    expanded_filter =  input_filter * expand_ratio
    x = tf.keras.layers.Conv2D(expanded_filter, 1,  padding='same',  use_bias=False)(input_data)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('swish')(x)

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

    # Squeeze and excitation phase (1,1)
    squeezed_filter = max(1, int(input_filter * se_ratio))
    se = tf.keras.layers.GlobalAveragePooling2D()(x)
    se = tf.keras.layers.Reshape((1, 1, expanded_filter))(se)
    se = tf.keras.layers.Conv2D(squeezed_filter,1)(se)
    se = tf.keras.layers.Activation('swish')(se)
    se = tf.keras.layers.Conv2D(expanded_filter,1, activation='sigmoid')(se)
    x = tf.keras.layers.multiply([x, se])

    # Output phase (1,1)
    x = tf.keras.layers.Conv2D(output_filter, 1, padding='same', use_bias=False)(x)
    x = tf.keras.layers.BatchNormalization()(x)

    if repeat_num == 1:
        pass
    else:
        x = tf.keras.layers.Dropout(drop_ratio)(x)

    return x
```

# EFFICIENTNET B0'S ARCHITECTURE

*Table 1.* **EfficientNet-B0 baseline network** – Each row describes a stage $i$ with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

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

```python
def Model():

    model_input =  tf.keras.layers.Input(shape = (224,224,3))

    # stem
    x = tf.keras.layers.Conv2D(32, (3,3), padding='same',strides = (2,2))(model_input)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Activation('swish')(x)

    # mbConv_blocks
    MBConv1_1 = mbConv_block(x,3,1,32,16,1,0.25,1,0.2)

    MBConv6_2_1 = mbConv_block(MBConv1_1,3,1,16,24,6,0.25,2,0.2)
    MBConv6_2_2 = mbConv_block(MBConv1_1,3,2,16,24,6,0.25,2,0.2)
    x = tf.keras.layers.add([MBConv6_2_1, MBConv6_2_2])

    MBConv6_3_1 = mbConv_block(x,5,1,24,40,6,0.25,2,0.2)
    MBConv6_3_2 = mbConv_block(x,5,2,24,40,6,0.25,2,0.2)
    x = tf.keras.layers.add([MBConv6_3_1, MBConv6_3_2])

    MBConv6_4_1 = mbConv_block(x,3,1,40,80,6,0.25,2,0.2)
    MBConv6_4_2 =  mbConv_block(x,3,2,40,80,6,0.25,2,0.2)
    MBConv6_4_3 =  mbConv_block(x,3,3,40,80,6,0.25,2,0.2)
    x = tf.keras.layers.add([MBConv6_4_1, MBConv6_4_2, MBConv6_4_3])

    MBConv6_5_1 = mbConv_block(x,5,1,80,112,6,0.25,1,0.2)
    MBConv6_5_2 =  mbConv_block(x,5,2,80,112,6,0.25,1,0.2)
    MBConv6_5_3 =  mbConv_block(x,5,3,80,112,6,0.25,1,0.2)
    x = tf.keras.layers.add([MBConv6_5_1, MBConv6_5_2, MBConv6_5_3])


    MBConv6_6_1 = mbConv_block(x,5,1,112,192,6,0.25,2,0.2)
    MBConv6_6_2 =  mbConv_block(x,5,2,112,192,6,0.25,2,0.2)
    MBConv6_6_3 =  mbConv_block(x,5,3,112,192,6,0.25,2,0.2)
    MBConv6_6_4 =  mbConv_block(x,5,4,112,192,6,0.25,2,0.2)
    x = tf.keras.layers.add([MBConv6_6_1, MBConv6_6_2, MBConv6_6_3, MBConv6_6_4])

    MBConv6_7 = mbConv_block(x,3,1,192,320,6,0.25,1,0.2)

    # output
    output = tf.keras.layers.GlobalAveragePooling2D()(x)

    model = tf.keras.Model(model_input, output)

    return model
```

# COMPARE WITH OTHER MODELS

## Compound scaling ratio

| Model | width | depth | Resolution | dropout |
|---|---|---|---|---|
| EN B0 | 1.0 | 1.0 | 224 | 0.2 |
| EN B1 | 1.0 | 1.1 | 240 | 0.2 |
| EN B2 | 1.1 | 1.2 | 260 | 0.3 |
| EN B3 | 1.2 | 1.4 | 300 | 0.3 |
| EN B4 | 1.4 | 1.8 | 380 | 0.4 |
| EN B5 | 1.6 | 2.2 | 456 | 0.4 |
| EN B6 | 1.8 | 2.6 | 528 | 0.5 |
| EN B7 | 2.0 | 3.1 | 600 | 0.5 |

*Table 2.* **EfficientNet Performance Results on ImageNet** (Russakovsky et al., 2015). All EfficientNet models are scaled from our baseline EfficientNet-B0 using different compound coefficient $\phi$ in Equation 3. ConvNets with similar top-1/top-5 accuracy are grouped together for efficiency comparison. Our scaled EfficientNet models consistently reduce parameters and FLOPS by an order of magnitude (up to 8.4x parameter reduction and up to 16x FLOPS reduction) than existing ConvNets.

| Model | Top-1 Acc. | Top-5 Acc. | #Params | Ratio-to-EfficientNet | #FLOPS | Ratio-to-EfficientNet |
|---|---|---|---|---|---|---|
| **EfficientNet-B0** | **77.3%** | **93.5%** | **5.3M** | **1x** | **0.39B** | **1x** |
| ResNet-50 (He et al., 2016) | 76.0% | 93.0% | 26M | 4.9x | 4.1B | 11x |
| DenseNet-169 (Huang et al., 2017) | 76.2% | 93.2% | 14M | 2.6x | 3.5B | 8.9x |
| **EfficientNet-B1** | **79.2%** | **94.5%** | **7.8M** | **1x** | **0.70B** | **1x** |
| ResNet-152 (He et al., 2016) | 77.8% | 93.8% | 60M | 7.6x | 11B | 16x |
| DenseNet-264 (Huang et al., 2017) | 77.9% | 93.9% | 34M | 4.3x | 6.0B | 8.6x |
| Inception-v3 (Szegedy et al., 2016) | 78.8% | 94.4% | 24M | 3.0x | 5.7B | 8.1x |
| Xception (Chollet, 2017) | 79.0% | 94.5% | 23M | 3.0x | 8.4B | 12x |
| **EfficientNet-B2** | **80.3%** | **95.0%** | **9.2M** | **1x** | **1.0B** | **1x** |
| Inception-v4 (Szegedy et al., 2017) | 80.0% | 95.0% | 48M | 5.2x | 13B | 13x |
| Inception-resnet-v2 (Szegedy et al., 2017) | 80.1% | 95.1% | 56M | 6.1x | 13B | 13x |
| **EfficientNet-B3** | **81.7%** | **95.6%** | **12M** | **1x** | **1.8B** | **1x** |
| ResNeXt-101 (Xie et al., 2017) | 80.9% | 95.6% | 84M | 7.0x | 32B | 18x |
| PolyNet (Zhang et al., 2017) | 81.3% | 95.8% | 92M | 7.7x | 35B | 19x |
| **EfficientNet-B4** | **83.0%** | **96.3%** | **19M** | **1x** | **4.2B** | **1x** |
| SENet (Hu et al., 2018) | 82.7% | 96.2% | 146M | 7.7x | 42B | 10x |
| NASNet-A (Zoph et al., 2018) | 82.7% | 96.2% | 89M | 4.7x | 24B | 5.7x |
| AmoebaNet-A (Real et al., 2019) | 82.8% | 96.1% | 87M | 4.6x | 23B | 5.5x |
| PNASNet (Liu et al., 2018) | 82.9% | 96.2% | 86M | 4.5x | 23B | 6.0x |
| **EfficientNet-B5** | **83.7%** | **96.7%** | **30M** | **1x** | **9.9B** | **1x** |
| AmoebaNet-C (Cubuk et al., 2019) | 83.5% | 96.5% | 155M | 5.2x | 41B | 4.1x |
| **EfficientNet-B6** | **84.2%** | **96.8%** | **43M** | **1x** | **19B** | **1x** |
| **EfficientNet-B7** | **84.4%** | **97.1%** | **66M** | **1x** | **37B** | **1x** |
| GPipe (Huang et al., 2018) | 84.3% | 97.0% | 557M | 8.4x | - | - |

We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

# THANK YOU :)