# MobileNets

**: Efficient Convolutional Neural Networks for Mobile Vision Applications**

**2021. 08. 03.**

**이동건**

# 0. Contents

1. Introduction
2. Prior Work
3. MobileNet Architecture
    3.1. Depthwise Separable Convolution
    3.2. Network Structure and Training
    3.3. Width Multiplier: Thinner Models
    3.4. Resolution Multiplier: Reduced Representation

# 0. Contents

4. Experiments
    4.1. Model Choices
    4.2. Model Shrinking Hyperparameters
    4.3. Fine Grained Recognition
    4.4. Large Scale Geolocalizaton
    4.5. Face Attributes
    4.6. Object Detection
    4.7. Face Embeddings
5. Conclusion
6. References

# 1. Introduction

- **efficiency** > accuracy
- Desirable Properties
  - Sufficiently high accuracy
  - Low computational complexity
  - Low energy usage
  - Small model size

# 1. Introduction

- AlexNet 이후로 accuracy를 높이기 위해 layer를 더욱 깊게 쌓아 네트워크의 depth를 늘리는 것이 일반적인 트렌드가 되었다.
- Deep한 네트워크를 설계하여 accuracy를 개선하는 것은 모델의 size, speed 와 같은 efficiency 측면에서 볼 때 문제가 될 수 있다.
- Real world에서는 computer resource가 제한적이기 때문에 문제가 된다.

- 본 논문에서는 small, low latency model을 설계하기 위한 방법을 통해, efficient network architecture를 제안한다.

# 1. Introduction

- Small Deep Neural Network의 중요성
  - network를 작게 만들면 학습이 빠르게 될 것이고,
    임베디드 환경에서 딥러닝을 구성하기에 더 적합해짐
  - 무선 업데이트로 Deep Neural Network를 업데이트 해야한다면,
    적은 용량으로 빠르게 업데이트 해주어야
    업데이트의 신뢰도와 통신 비용 등에 도움이 될 것

# 2. Prior Work

**Small Deep Neural Network 기법**

- Remove Fully-Connected Layers
  - 파라미터의 90% 정도가 FC layer에 분포되어 있는만큼, FC layer를 제거하면 경량화가 됨
  - CNN기준으로 필터(커널)들은 파라미터 쉐어링을 해서 다소 파라미터의 갯수가 작지만, FC layer에서는 파라미터 쉐어링을 하지 않기 때문에 엄청나게 많은 수의 파라미터가 존재하게 됨
- Kernel Reduction (3 x 3 → 1 x 1)
  - (3 x 3) 필터를 (1 x 1) 필터로 줄여 연산량 또는 파라미터 수를 줄이는 기법
  - 이 기법은 대표적으로 SqueezeNet에서 사용됨

- Shuffle Operation
- Evenly Spaced Downsampling
  - Downsampling 하는 시점과 관련되어 있는 기법
  - Downsampling을 초반에 많이 할 것인지 아니면 후반에 많이 할 것인지 선택하게 되는데, 그것을 극단적으로 하지 않고 균등하게 하자는 컨셉
  - 초반에 Downsampling을 많이하게 되면 네트워크 크기는 줄게 되지만, feature를 많이 잃게 되어 accuracy가 줄어들게 되고
  - 후반에 Downsampling을 많이하게 되면 accuracy 면에서는 전자에 비하여 낫지만 네트워크의 크기가 많이 줄지는 않게됨
  - 따라서 이것의 절충안으로 적절히 튜닝하면서 Downsampling을 하여 Accuracy와 경량화 두 가지를 모두 획득하자는 것

- Channel Reduction : MobileNet 적용
  - Channel 숫자를 줄여서 경량화
- **Depthwise Seperable Convolution** : MobileNet 적용
  - 이 컨셉은 L.Sifre의 Ph. D. thesis에서 가져온 컨셉이고 이 방법으로 경량화를 할 수 있다.
- Distillation & Compression : MobileNet 적용

# 2. Prior Work

- 최근에도 small and efficient neural network를 설계하기 위한 연구가 있지만 기존의 방법들은 pretrained network를 압축하거나 small network를 directly하게 학습시켰던 두 분류로 나눌 수 있다.
- 본 논문에서는 제한된 latency, size 안에서 small network를 구체적으로 설계할 수 있는 아키텍처를 제안한다.

# 2. Prior Work

> *MobileNets primarily focus on optimizing for latency but also yield small networks.*
>
> *Many papers on small networks focus only on size but do not consider speed.*

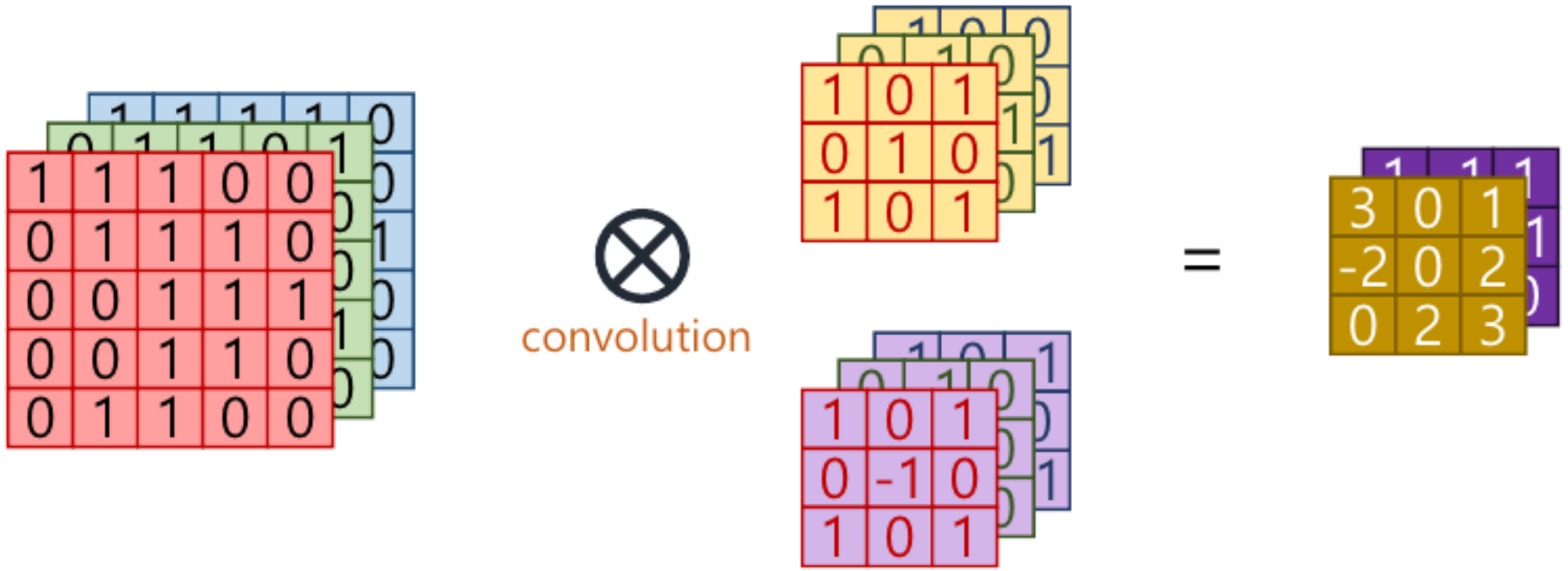
- 제안하는 MobileNets은 latency를 최적화하면서 small network를 만드는 것에 초점을 맞춘다.

# 3. MobileNet Architecture

- 3.1. Depthwise Separable Convolution
- 3.2. Network Structure and Training
- 3.3. Width Multiplier: Thinner Models
- 3.4. Resolution Multiplier: Reduced Representation

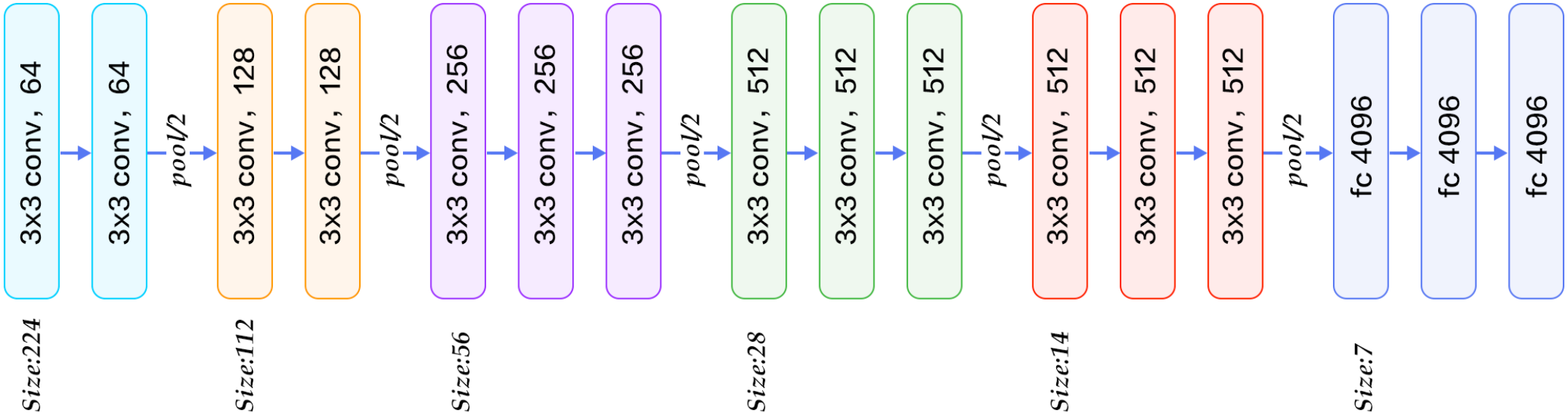
# 3.1. Depthwise Separable Convolution

- Standard Convolution



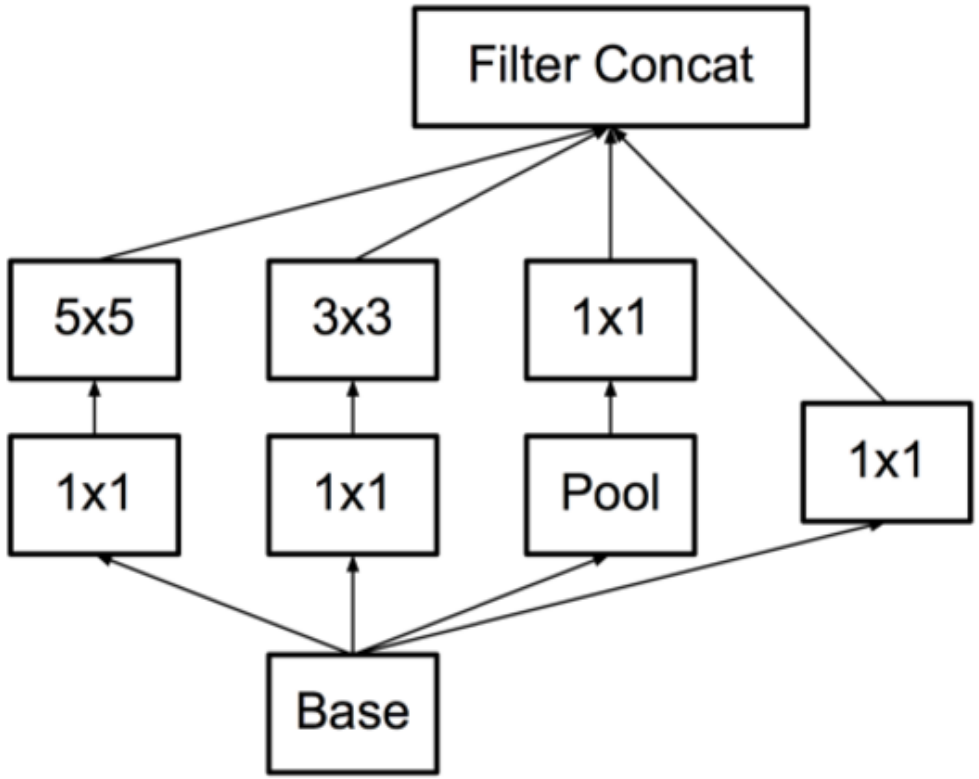
# 3.1. Depthwise Separable Convolution

- VGG

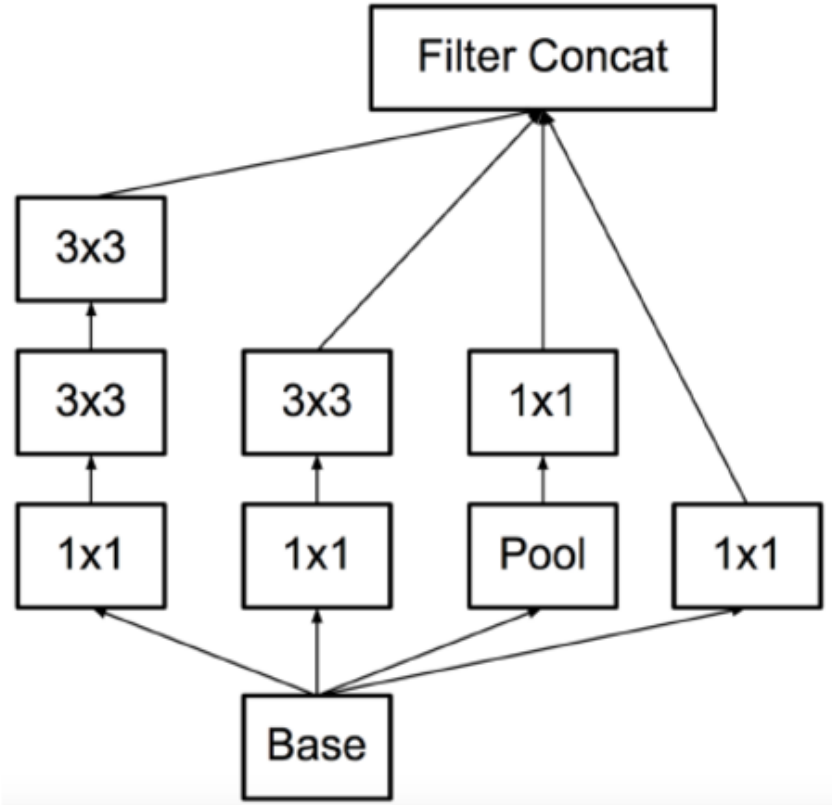


# 3.1. Depthwise Separable Convolution

- Inception-v3



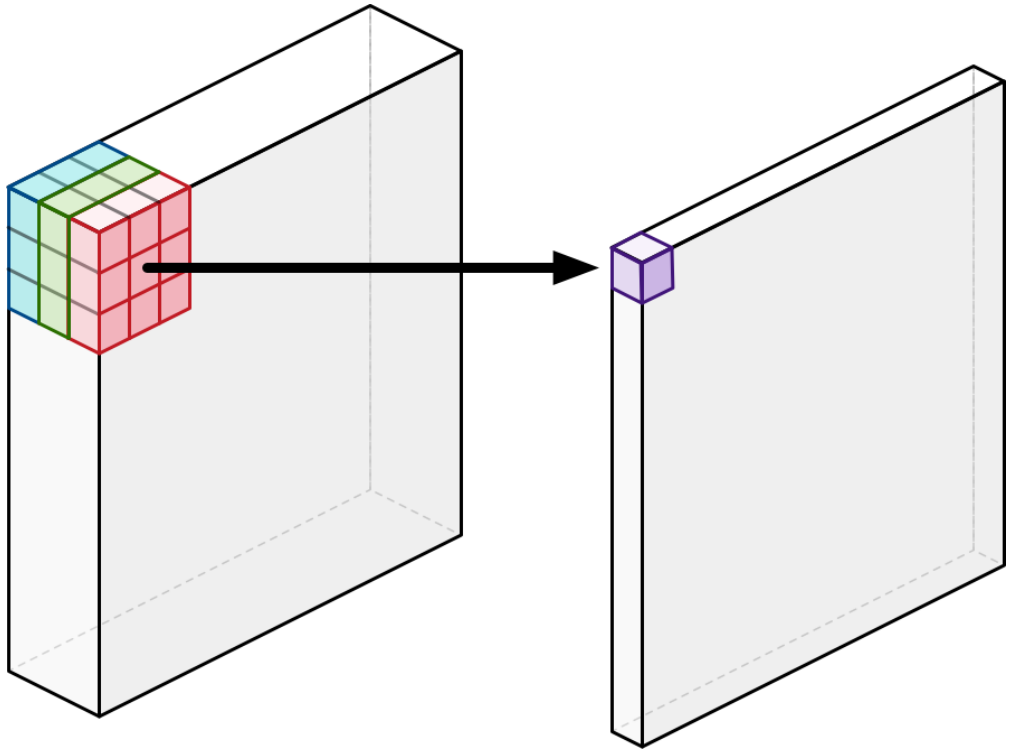

original inception module




factorizing inception module

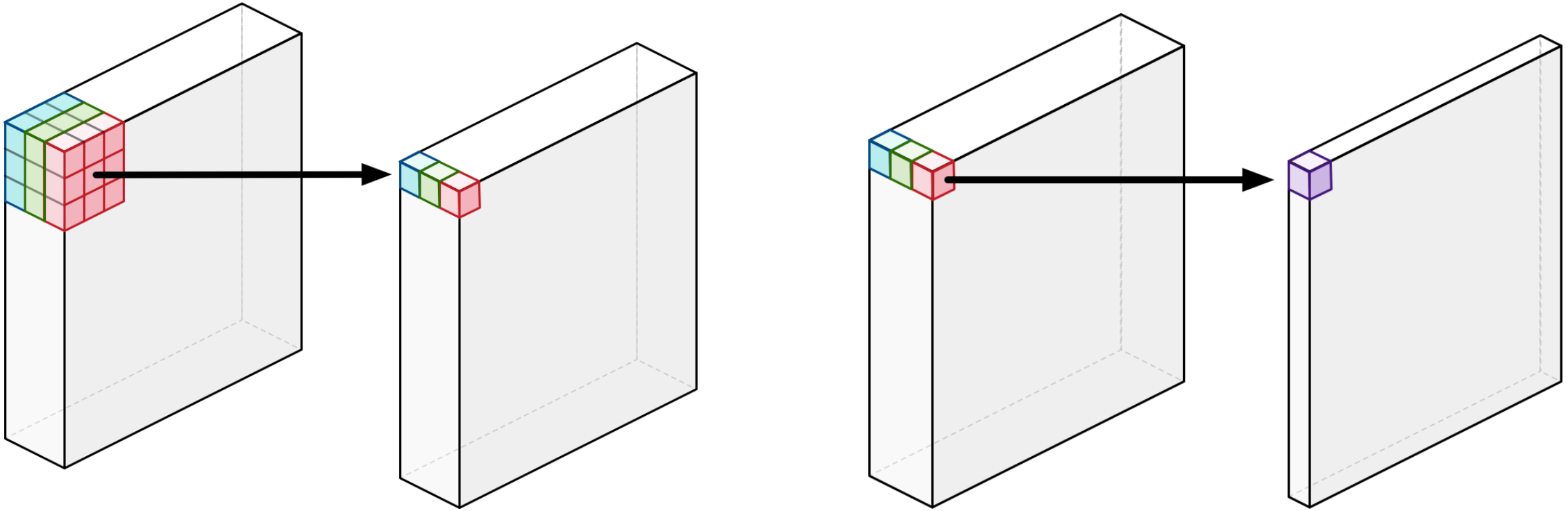
# 3.1. Depthwise Separable Convolution

- Standard Convolution



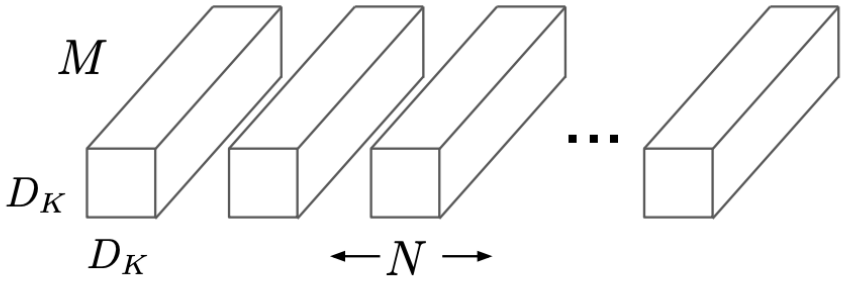
# 3.1. Depthwise Separable Convolution

- Depthwise Convolution + Pointwise Convolution (1x1 convolution)

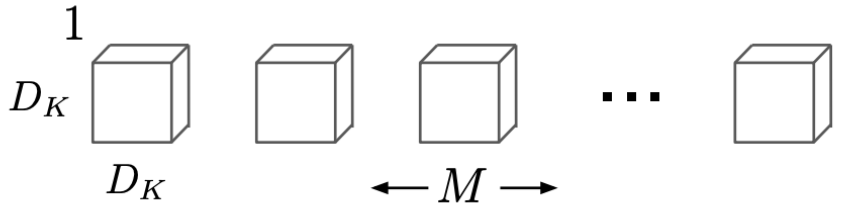


# 3.1. Depthwise Separable Convolution

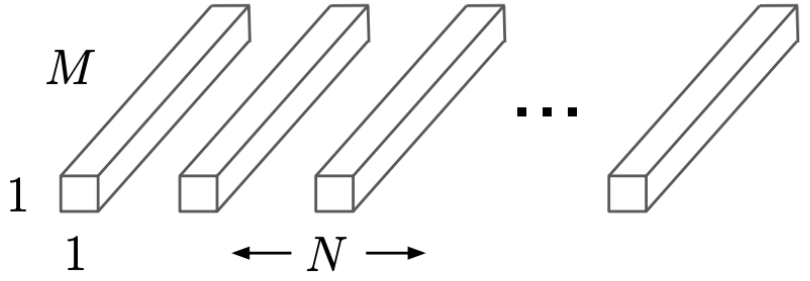
- Depthwise Convolution + Pointwise Convolution (1x1 convolution)




(a) Standard Convolution Filters




(b) Depthwise Convolutional Filters




(c) $1 \times 1$ Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

- $D_K$ = 필터의 height/width 크기
- $D_F$ = Feature map의 height/width 크기
- $M$ = 인풋 채널의 크기
- $N$ = 아웃풋 채널의 크기 (필터의 개수)

- **Standard Convolution**의 대략적 계산 비용
  - $D_K \times D_K \times M \times N \times D_F \times D_F$
- **Depthwise Separable Convolution**의 대략적 계산 비용
  - $D_K \times D_K \times M \times D_F \times D_F + D_F \times D_F \times M \times N$

- 두 Convolution의 계산 비용 차이 (**Depthwise Separable Version / Standard Version**)
  - $(D_K \times D_K \times M \times D_F \times D_F + D_F \times D_F \times M \times N)/(D_K \times D_K \times M \times N \times D_F \times D_F) = 1/N + 1/{D_K}^2$

- 여기서 $N$은 아웃풋 채널의 크기이고 $D_K$는 필터의 크기인데,
  $N$이 $D_K$ 보다 일반적으로 훨씬 큰 값이므로 ,
  반대로 $1/D_K^2$ 값이 되어 $1/D_K^2$ 배 만큼
  계산이 줄었다고 할 수 있다.
- 이 때, $D_K$는 보통 $3$이므로 $1/9$ 배 정도 계산량이 감소한다.

# 3.2. Network Structure and Training

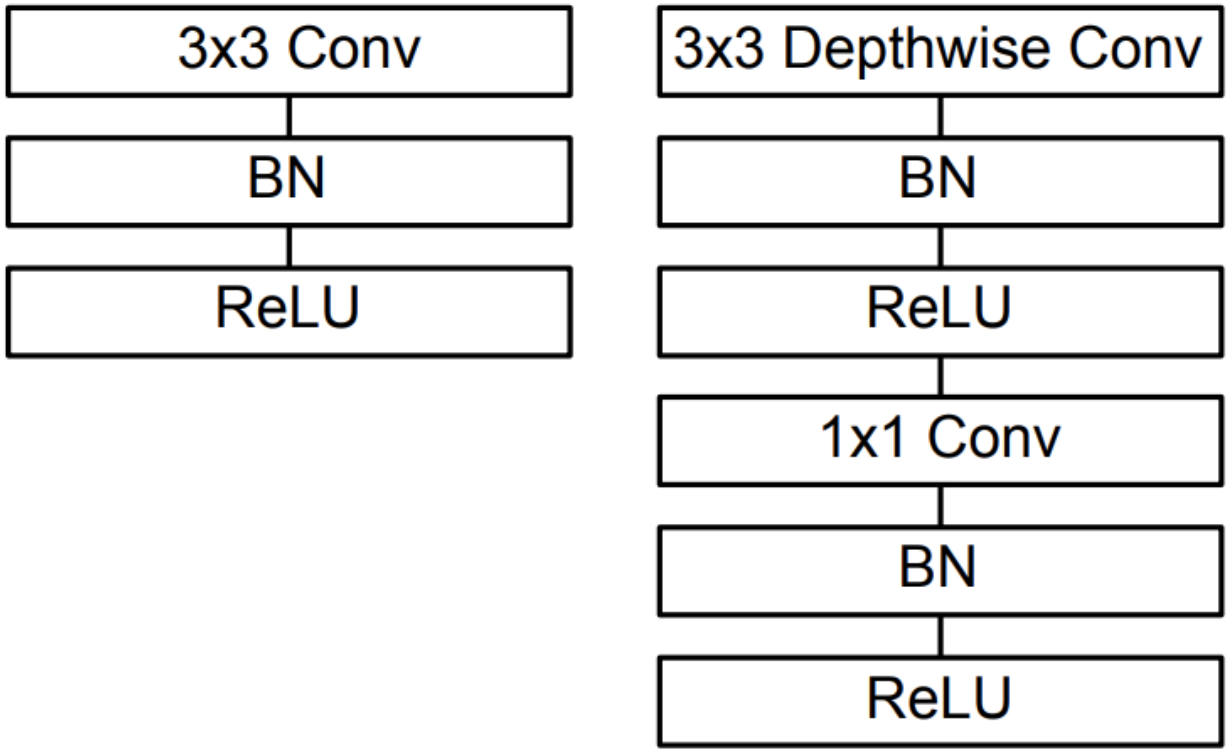



Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

# 3.2. Network Structure and Training

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% |

## 3.3. Width Multiplier: Thinner Models

## 3.4. Resolution Multiplier: Reduced Representation

- 두 값 모두 기존의 컨셉에서 조금 더 작은 네트워크를 만들기 위해 사용되는 scale 값이고 값의 범위는 $0$ ~ $1$이다.

# 3.3. Width Multiplier: Thinner Models

- width multiplier는 논문에서 $\alpha$로 표현하였고, input과 output의 채널에 곱해지는 값이다.
  - 논문에서 thinner model을 위한 상수로 사용되었으며, 채널의 크기를 일정 비율 줄여가면서 실험하였다.
- 즉, 채널의 크기를 조정하기 위해 사용되는 값으로 채널의 크기가 $M$ 이면 $\alpha M$으로 표현한다.
- 논문에서 사용된 $\alpha$ 값은 $1, 0.75, 0.5, 0.25$ 이다.

# 3.4. Resolution Multiplier: Reduced Representation

- 반면 resolution multiplier는 input의 height와 width에 곱해지는 상수값이다.
- height와 width가 $D_F$ 이면 $\rho D_F$가 된다.
- 기본적으로 (224, 224, 3) 이미지를 input으로 넣고 실험할 때,
  상수 $\rho$ $(1, 0.857, 0.714, 0.571)$에 따라서
  사이즈가 변한다. (224, 192, 160 or 128)

# 3.3. & 3.4. Width & Resolution Multiplier

- 이렇게 width, resolution multiplier가 적용되면 계산 비용은 다음과 같이 정의된다.
  - 채널에 $\alpha$를 곱하고, feature map에는 $\rho$를 곱한다.

“ $D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$ ”

# 3.3. & 3.4. Width & Resolution Multiplier

Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

| Layer/Modification | Million Mult-Adds | Million Parameters |
|---|---|---|
| Convolution | 462 | 2.36 |
| Depthwise Separable Conv | 52.3 | 0.27 |
| $\alpha = 0.75$ | 29.6 | 0.15 |
| $\rho = 0.714$ | 15.1 | 0.15 |

# 4. Experiments

4.1. Model Choices
4.2. Model Shrinking Hyperparameters
4.3. Fine Grained Recognition
4.4. Large Scale Geolocalizaton
4.5. Face Attributes
4.6. Object Detection
4.7. Face Embeddings

# 4.1. Model Choices

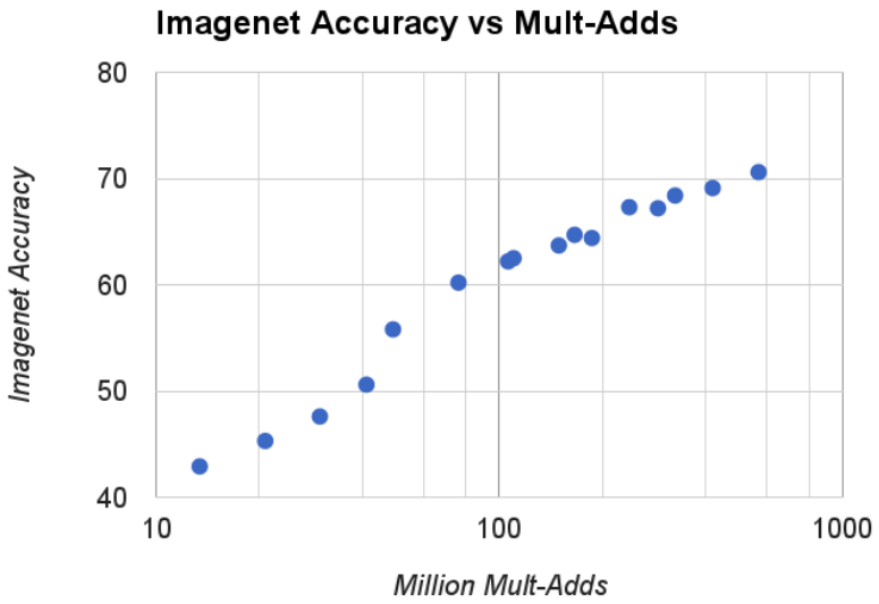



Figure 4. This figure shows the trade off between computation (Mult-Adds) and accuracy on the ImageNet benchmark. Note the log linear dependence between accuracy and computation.

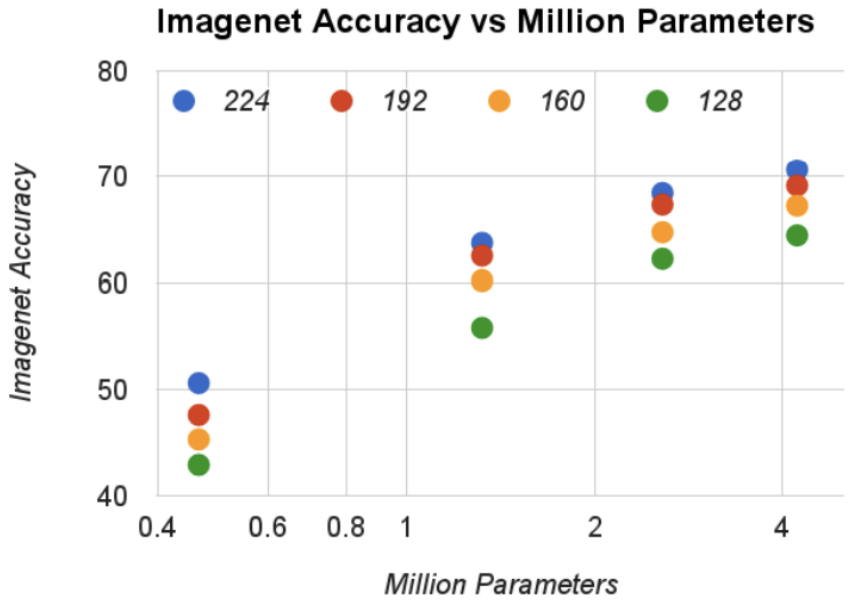



Figure 5. This figure shows the trade off between the number of parameters and accuracy on the ImageNet benchmark. The colors encode input resolutions. The number of parameters do not vary based on the input resolution.

# 4.2. Model Shrinking Hyperparameters

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 |

Table 5. Narrow vs Shallow MobileNet

| Model | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| 0.75 MobileNet | 68.4% | 325 | 2.6 |
| Shallow MobileNet | 65.3% | 307 | 2.9 |

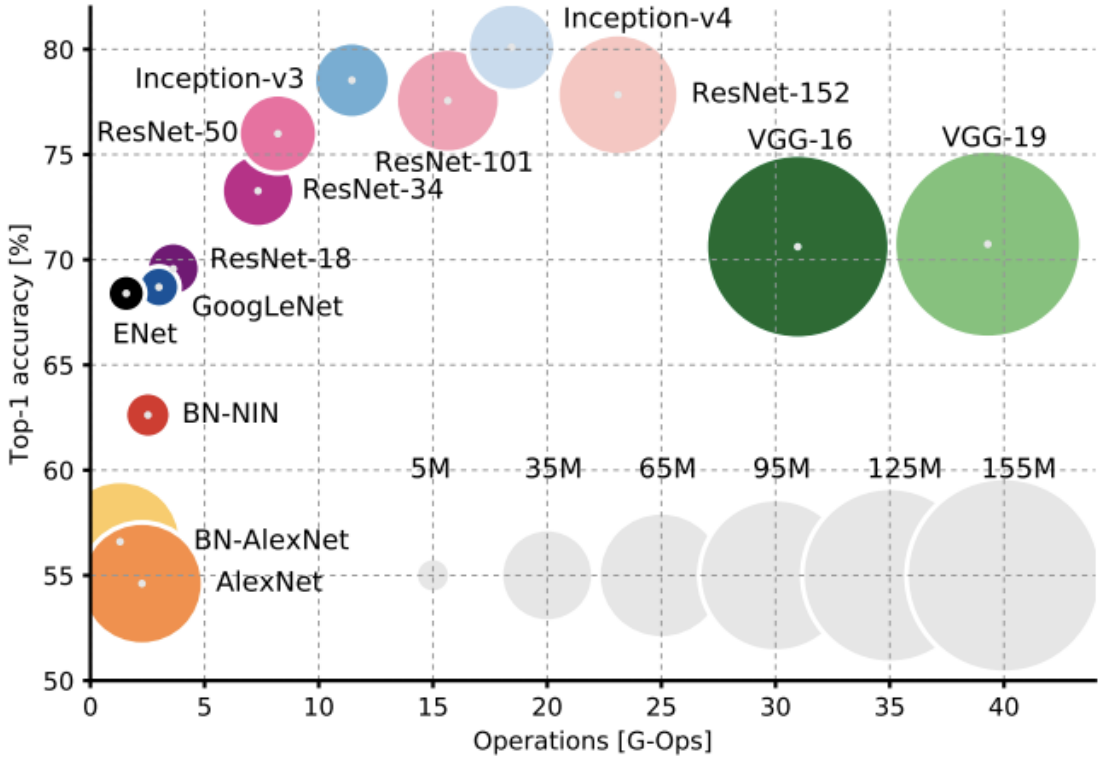
Table 6. MobileNet Width Multiplier

| Width Multiplier | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| 1.0 MobileNet-224 | 70.6% | 569 | 4.2 |
| 0.75 MobileNet-224 | 68.4% | 325 | 2.6 |
| 0.5 MobileNet-224 | 63.7% | 149 | 1.3 |
| 0.25 MobileNet-224 | 50.6% | 41 | 0.5 |

Table 7. MobileNet Resolution

| Resolution | ImageNet Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| 1.0 MobileNet-224 | 70.6% | 569 | 4.2 |
| 1.0 MobileNet-192 | 69.1% | 418 | 4.2 |
| 1.0 MobileNet-160 | 67.2% | 290 | 4.2 |
| 1.0 MobileNet-128 | 64.4% | 186 | 4.2 |

# 4.2. Model Shrinking Hyperparameters



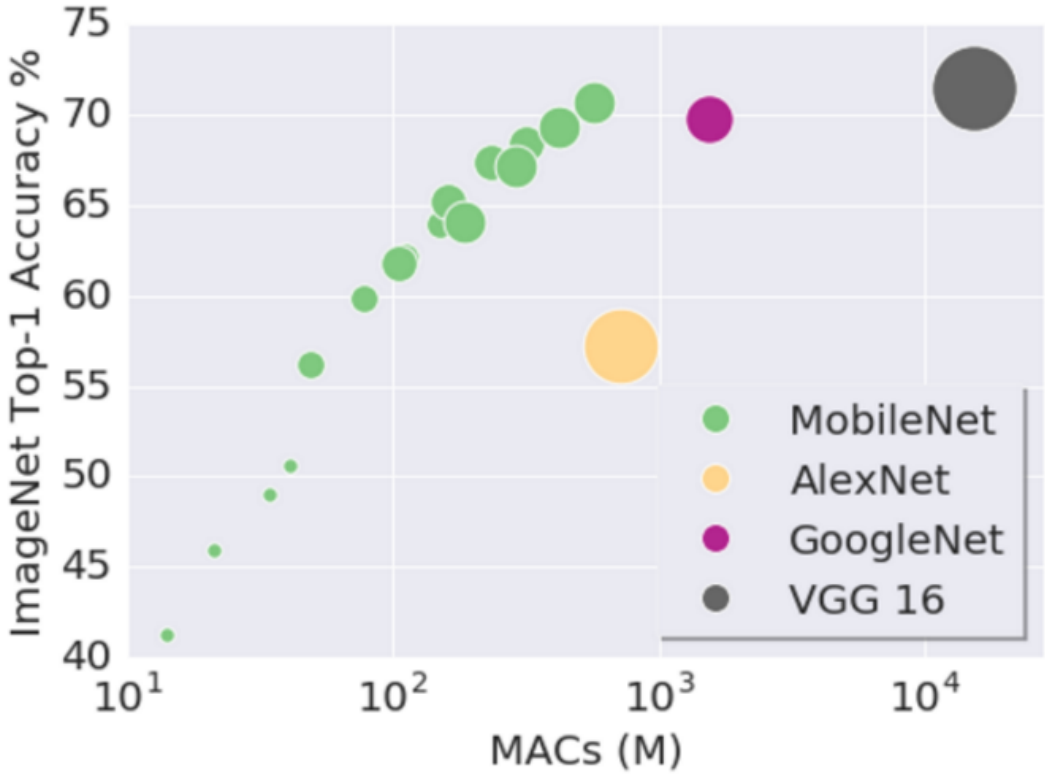




Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.**

# 4.2. Model Shrinking Hyperparameters

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 |

# 4.3. Fine Grained Recognition

Table 10. MobileNet for Stanford Dogs

| Model | Top-1 Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| Inception V3 [18] | 84% | 5000 | 23.2 |
| 1.0 MobileNet-224 | 83.3% | 569 | 3.3 |
| 0.75 MobileNet-224 | 81.9% | 325 | 1.9 |
| 1.0 MobileNet-192 | 81.9% | 418 | 3.3 |
| 0.75 MobileNet-192 | 80.5% | 239 | 1.9 |

# 4.4. Large Scale Geolocalizaton

- PlaNet : 52M parameters, 5.74B multi-adds
- mobilenet : 13M parameters, 0.58M multi-adds

Table 11. Performance of PlaNet using the MobileNet architecture. Percentages are the fraction of the Im2GPS test dataset that were localized within a certain distance from the ground truth. The numbers for the original PlaNet model are based on an updated version that has an improved architecture and training dataset.

| Scale | Im2GPS [7] | PlaNet [35] | PlaNet MobileNet |
|---|---|---|---|
| Continent (2500 km) | 51.9% | 77.6% | 79.3% |
| Country (750 km) | 35.4% | 64.0% | 60.3% |
| Region (200 km) | 32.1% | 51.1% | 45.2% |
| City (25 km) | 21.9% | 31.7% | 31.7% |
| Street (1 km) | 2.5% | 11.0% | 11.4% |

# 4.6. Object Detection

Table 13. COCO object detection results comparison using different frameworks and network architectures. mAP is reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95)

| Framework Resolution | Model | mAP | Billion Mult-Adds | Million Parameters |
|---|---|---|---|---|
| SSD 300 | deeplab-VGG | 21.1% | 34.9 | 33.1 |
| | Inception V2 | 22.0% | 3.8 | 13.7 |
| | MobileNet | 19.3% | 1.2 | 6.8 |
| Faster-RCNN 300 | VGG | 22.9% | 64.3 | 138.5 |
| | Inception V2 | 15.4% | 118.2 | 13.3 |
| | MobileNet | 16.4% | 25.2 | 6.1 |
| Faster-RCNN 600 | VGG | 25.7% | 149.6 | 138.5 |
| | Inception V2 | 21.9% | 129.6 | 13.3 |
| | Mobilenet | 19.8% | 30.5 | 6.1 |


Figure 6. Example objection detection results using MobileNet SSD.

# 4.5. Face Attributes & 4.7. Face Embeddings

Table 12. Face attribute classification using the MobileNet architecture. Each row corresponds to a different hyper-parameter setting (width multiplier $\alpha$ and image resolution).

| Width Multiplier / Resolution | Mean AP | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| 1.0 MobileNet-224 | 88.7% | 568 | 3.2 |
| 0.5 MobileNet-224 | 88.1% | 149 | 0.8 |
| 0.25 MobileNet-224 | 87.2% | 45 | 0.2 |
| 1.0 MobileNet-128 | 88.1% | 185 | 3.2 |
| 0.5 MobileNet-128 | 87.7% | 48 | 0.8 |
| 0.25 MobileNet-128 | 86.4% | 15 | 0.2 |
| Baseline | 86.9% | 1600 | 7.5 |

Table 14. MobileNet Distilled from FaceNet

| Model | 1e-4 Accuracy | Million Mult-Adds | Million Parameters |
|---|---|---|---|
| FaceNet [25] | 83% | 1600 | 7.5 |
| 1.0 MobileNet-160 | 79.4% | 286 | 4.9 |
| 1.0 MobileNet-128 | 78.3% | 185 | 5.5 |
| 0.75 MobileNet-128 | 75.2% | 166 | 3.4 |
| 0.75 MobileNet-128 | 72.5% | 108 | 3.8 |

# 5. Conclusions

# 6. References


- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications
- JinWon Lee YouTube - PR-044: MobileNet
- JINSOL KIM - MobileNets~
- eremo2002 - Paper Review : MobileNets~
- Jay S. YouTube - [AI 논문 해설] 모바일넷 MobileNet 알아보기

- Google's MobileNets on the iPhone
- An Analysis of Deep Neural Network Models for Practical Applications

# 7. 코드 실습

## References

- tensorflow GitHub - models/research/slim/nets/mobilenet_v1.md
- Zehaos GitHub - MobileNet/nets/mobilenet.py
- [논문 구현] PyTorch로 MobileNetV1(2017) 구현
- MG2033 GitHub - MobileNet/model.py
- eremo2002 - MobileNetV1

# Applications

- chrispolo GitHub - Mobilenet-  v1-Mask-RCNN-for-detection
- hollance GitHub - MobileNet-CoreML