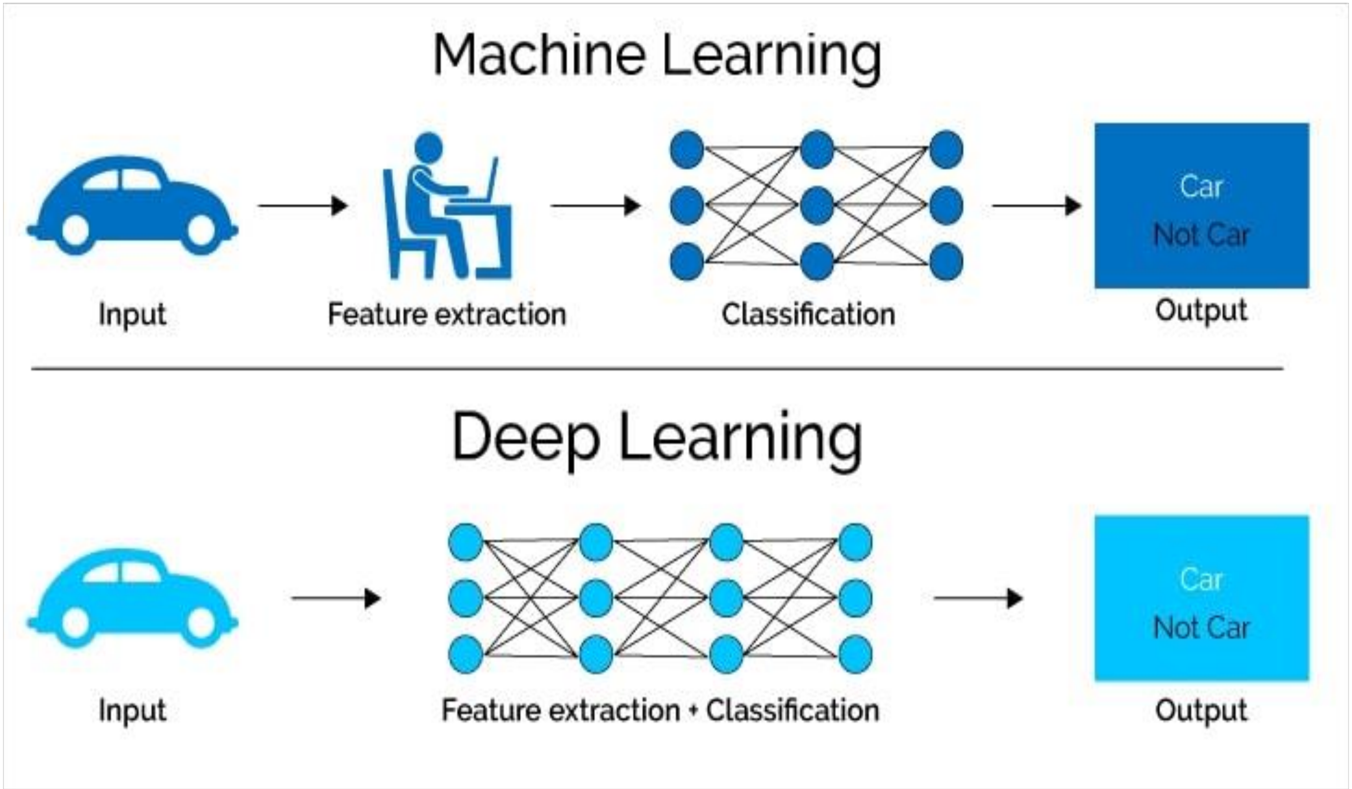
Deep Learning Project – MobileNetv2

UCF101 dataset (https://www.crcv.ucf.edu/data/UCF101.php) **Image Recognition**

3조: 김용삼, 노지윤, 박선홍, 송휘경, 안병윤, 이찬영, 정혜리, 한수연

Content

What's Difference?



HOG (Histogram of oriented Gradients) feature Extractor and SVM (Support Vector Machine) model

Bag of features model

Examples of this are SIFT, MSER, etc.

Viola-Jones algorithm

The advantage of Viola-Jones is that it has a detection time of 2 fps which can be used in a real-time face recognition system

Convolution Neural Network (CNN) is one of the most popular ways of doing object recognition. It is widely used and most state-of-the-art neural networks used this method for various object recognition related tasks such as image classification. This CNN network takes an image as input and outputs the probability of the different classes. If the object present in the image then it's output probability is high else the output probability of the rest of classes is either negligible or low. The advantage of Deep learning is that we don't need to do feature extraction from data as compared to machine learning.

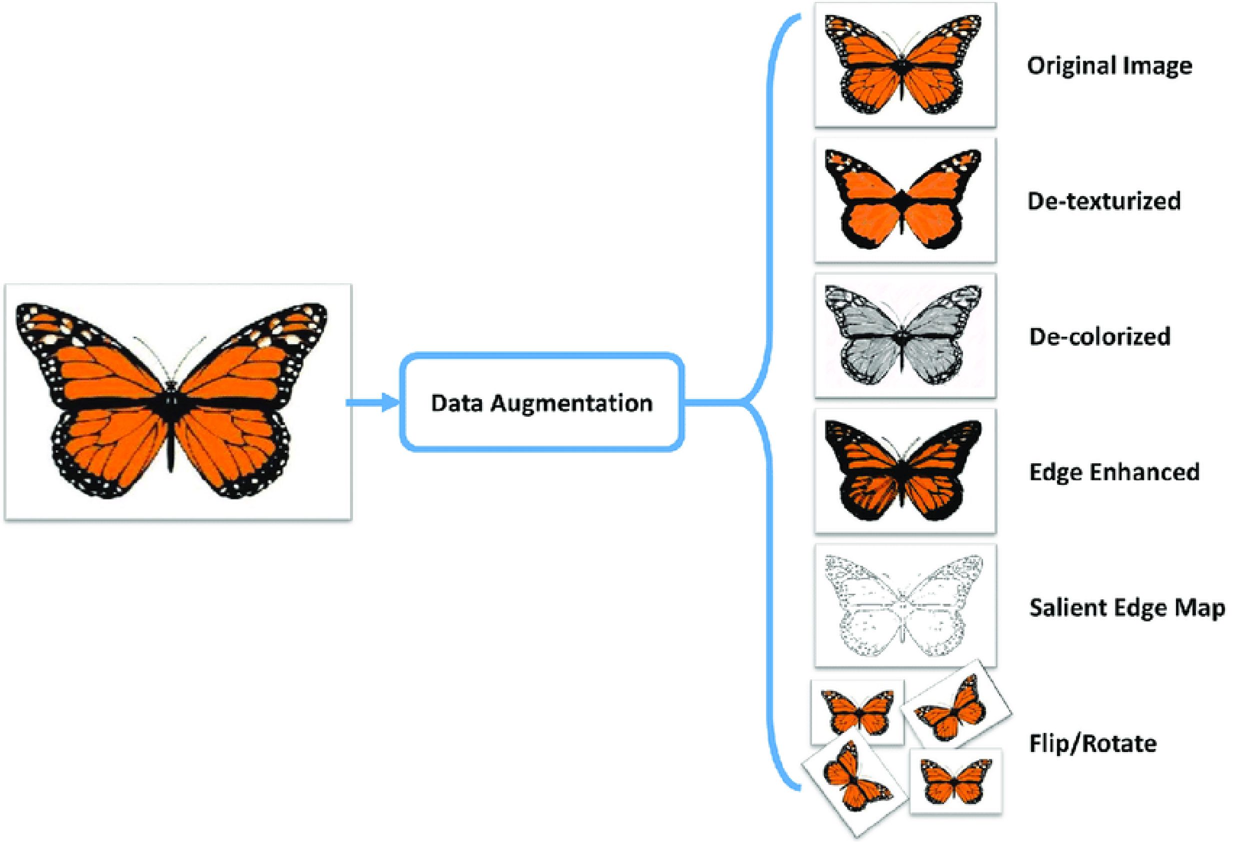
Object Recognition

Semantic vs Instance

001 Image Classification	002 Object Localization	003 Object Detection	004 Image Segmentation
			
Single Object	Bounding Box position, height, width	Multi-Objects Bounding box = rectangle	Pixel-wise masks Mask R-CNN

Preprocessing

Original Image + = Data Augmentation



001 Normalization

002 Remove Noise

003 Random Transformation

004 Color Jitter

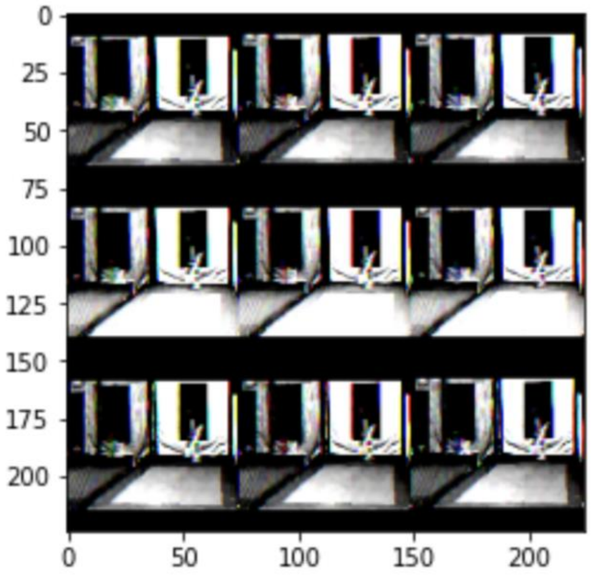
005 Dimensional Transpose
(Include, Segmentation)

Normalization

```
_mean = video.mean() / 255
_std = video.std() / 255
print('shape   :', video.shape)
print('RGB mean:', _mean)
print('RGB std :', _std)
```

```
shape   : (20, 224, 224, 3)
RGB mean: 0.30887114350948713
RGB std : 0.2780395522695637
```

```
aug_f = transforms.Compose([transforms.ToTensor(),
                            transforms.Normalize(_mean, _std)])
```



Random Augmentation

OpenCV / Pytorch

Horizontal / Vertical Flip

Affine

Resized Crop

Gray Scale

Perspective

Rotation (expand=True)

Filtering

Bluring

Morphology (smoothing edges)

Segmentation

Morphology

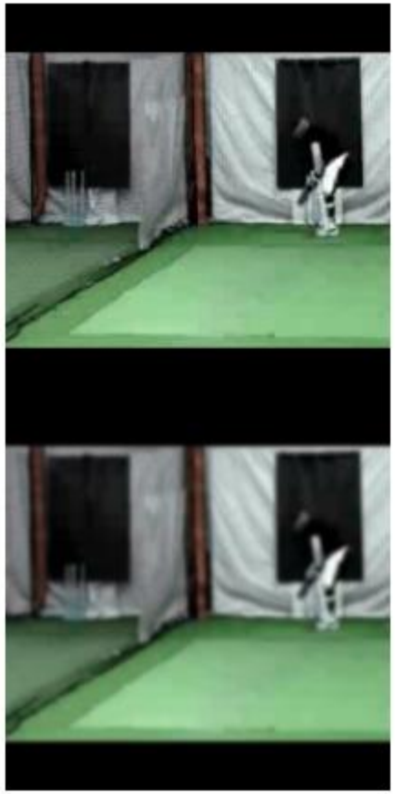
GrabCut / Noise

Random Choice

Filtering

Gaussian Blur

```
cv2_imshow( image)
cv2_imshow(denoised_img2) # GaussianBlur
```



Median Blur

```
cv2_imshow( image)
cv2_imshow(denoised_img3) # MedianBlur
```

NIMeans

```
cv2_imshow( image)
cv2_imshow(denoised_img1) # NIMeans
```

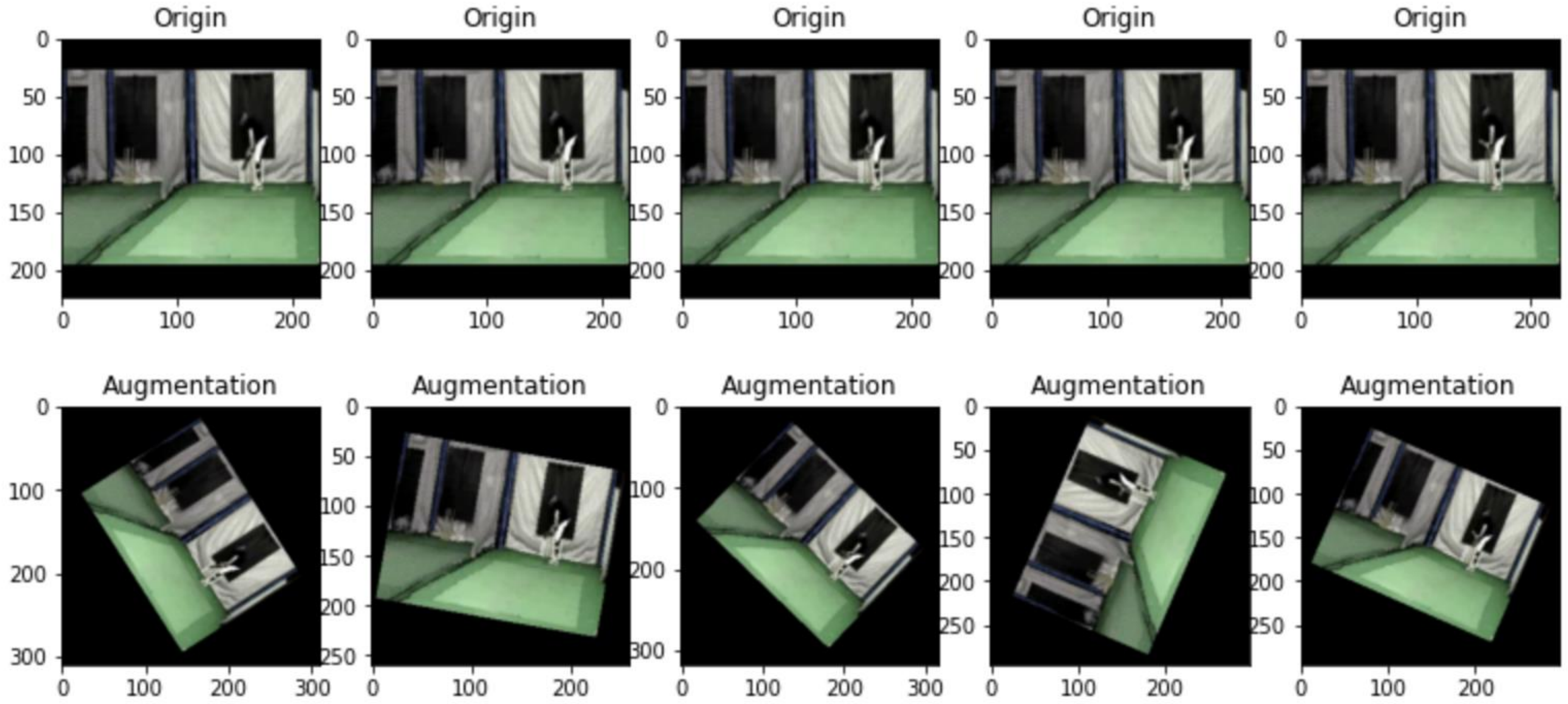
Bilateral

```
cv2_imshow( image)
cv2_imshow(denoised_img4) # Bilateral
```

Random Transformation

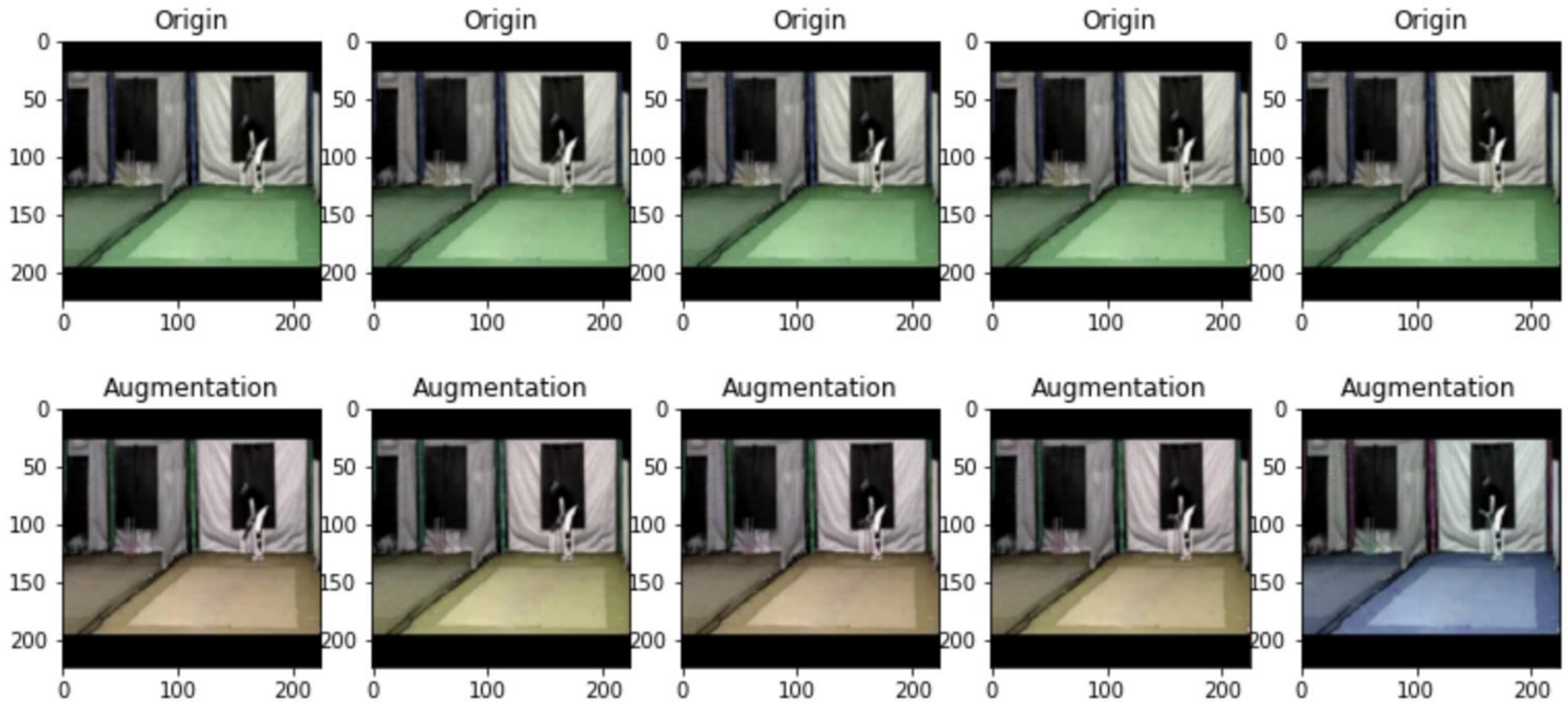
Pytorch

```
af = transforms.RandomRotation(90, expand=True)
display_augmented_images(af)
```



Color Jitter


```
af = transforms.ColorJitter(hue=(-0.5, 0.5))
display_augmented_images(af)
```



Modeling - MobileNet

What's MobileNet?

Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

기존의 학습된 모델의 정확도를
유지하면서 크기가 작고,
연산을 간소화 하는

'경량 딥러닝 알고리즘'

MobileNet은 Accuracy와 trade-off 관계를 가지는

'연산량을 줄여'

임베디드, 모바일 기기에서
동작시킬 수 있도록 만든
알고리즘이다.

〈표 1〉 경량 딥러닝(Lightweight Deep Learning) 연구 동향

	접근방법	연구 방향
경량 알고리즘 연구	모델 구조 변경	잔여 블록, 병목 구조, 밀집 블록 등 다양한 신규 계층 구조를 이용하여 파라미터 축소 및 모델 성능을 개선하는 연구(ResNet, DenseNet, SqueezeNet)
	합성곱 필터 변경	합성곱 신경망의 가장 큰 계산량을 요구하는 합성곱 필터의 연산을 효율적으로 줄이는 연구(MobileNet, ShuffleNet)
	자동 모델 탐색	특정 요소(지연시간, 에너지 소모 등)가 주어진 경우, 강화 학습을 통해 최적 모델을 자동 탐색하는 연구(NetAdapt, MNasNet)
알고리즘 경량화 연구	모델 압축	가중치 가지치기, 양자화/이진화, 가중치 공유 기법을 통해 파라미터의 불필요한 표현력을 줄이는 연구(Deep Compression, XNOR-Net)
	지식 증류	학습된 기본 모델을 통해 새로운 모델의 생성 시 파라미터값을 활용하여 학습시간을 줄이는 연구(Knowledge Distillation, Transfer Learning)
	하드웨어 가속화	모바일 기기를 중심으로 뉴럴 프로세싱 유닛(NPU)을 통해 추론 속도를 향상시키는 연구
	모델 압축 자동 탐색	알고리즘 경량화 연구 중 일반적인 모델 압축 기법을 적용한 강화 학습 기반의 최적 모델 자동 탐색 연구(PocketFlow, AMC)

MobileNet v1

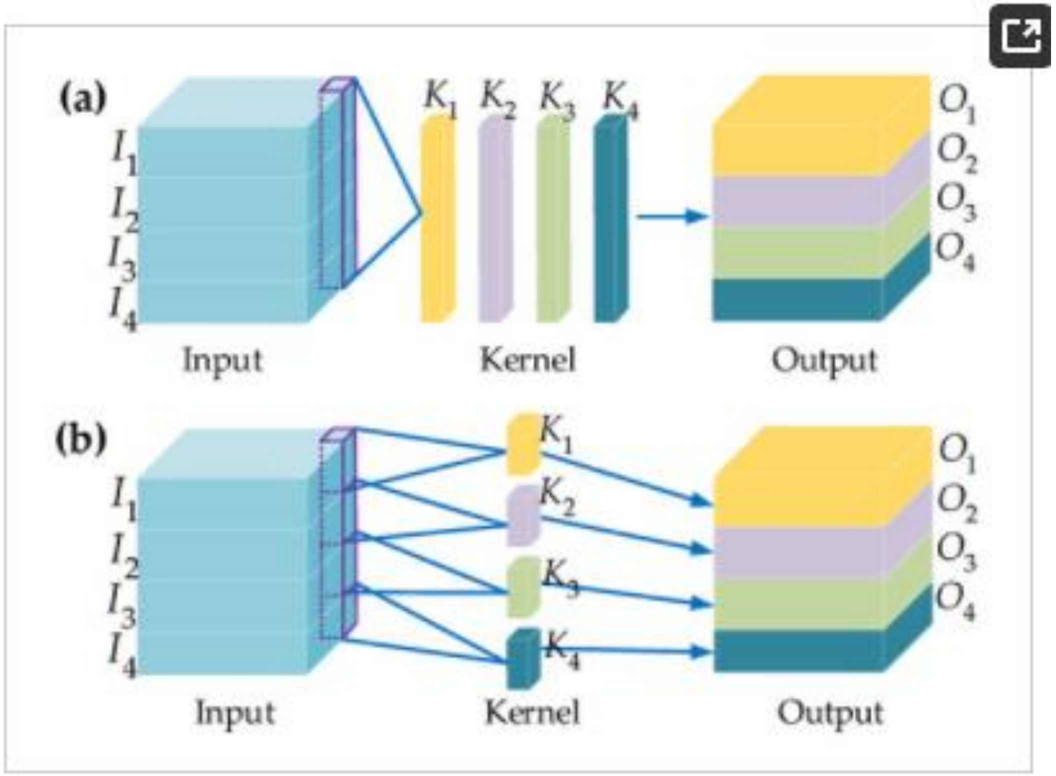
2017년 논문 "MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications"에서 제시된 경량 딥러닝 알고리즘.

Depthwise Separable Convolution

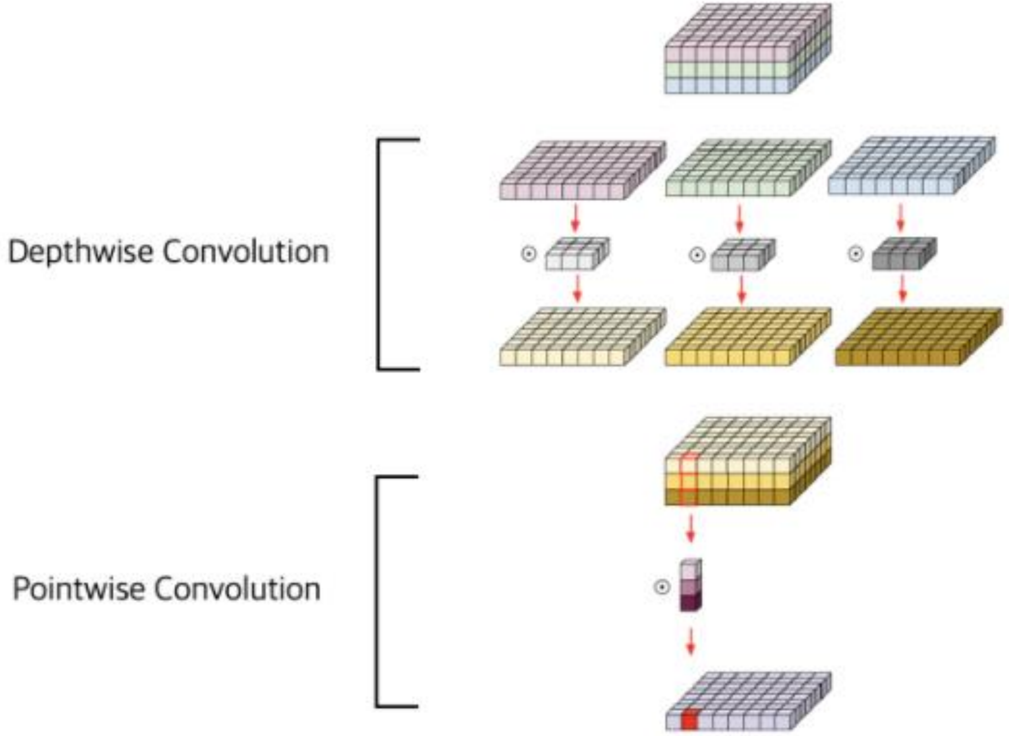
Pointwise Separable Convolution

Width, Resolution Multiplier

Key Point



(a)
I_1
I_2
I_3
I_4
K_1 K_2 K_3 K_4
O_1
O_2
O_3
O_4
Input
Kernel
Output
(b)
I_1
I_2
I_3
I_4
K_1
K_2
K_3
K_4
O_1
O_2
O_3
O_4
Input
Kernel
Output



Depthwise Convolution
Pointwise Convolution

Basic Convolution

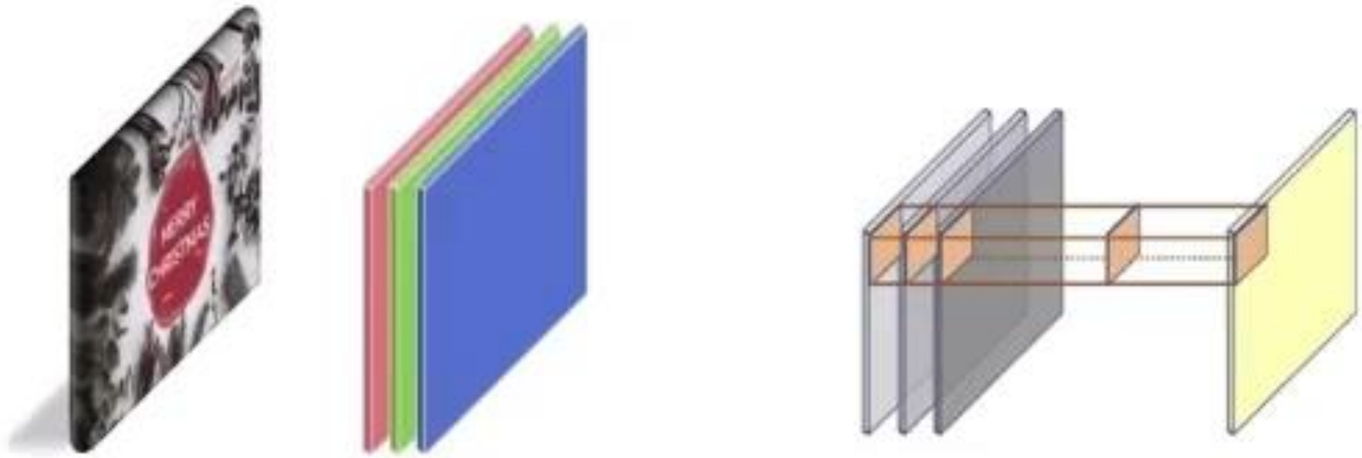
$$\boxed{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$

Depthwise Separable Convolution

$$\boxed{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F}$$

Pointwise Separable Convolution

$$\boxed{M \cdot N \cdot D_F \cdot D_F}$$



연산량 차이

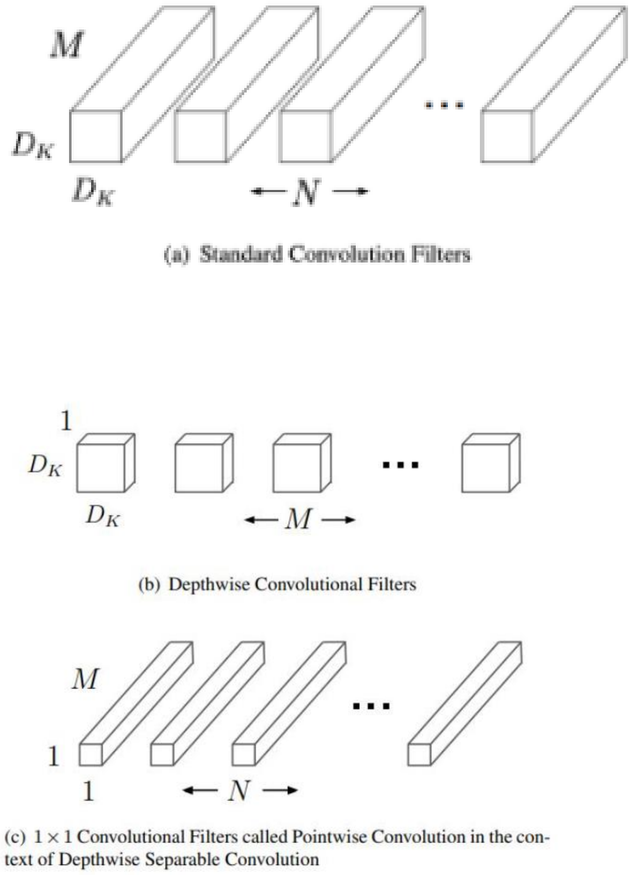


그림 4. Convolution 방법 비교

Depthwise : (# of 입력 필터) * 입력데이터 * 1 (1 channel)

+ Pointwise : 입력데이터 * 채널 수 * 필터 수(1x1 Conv)

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$

$$= \frac{1}{N} + \frac{1}{D_K^2}$$

기존연산 : (# of 입력 필터) * 입력데이터 * 채널 수

MobileNet Structure

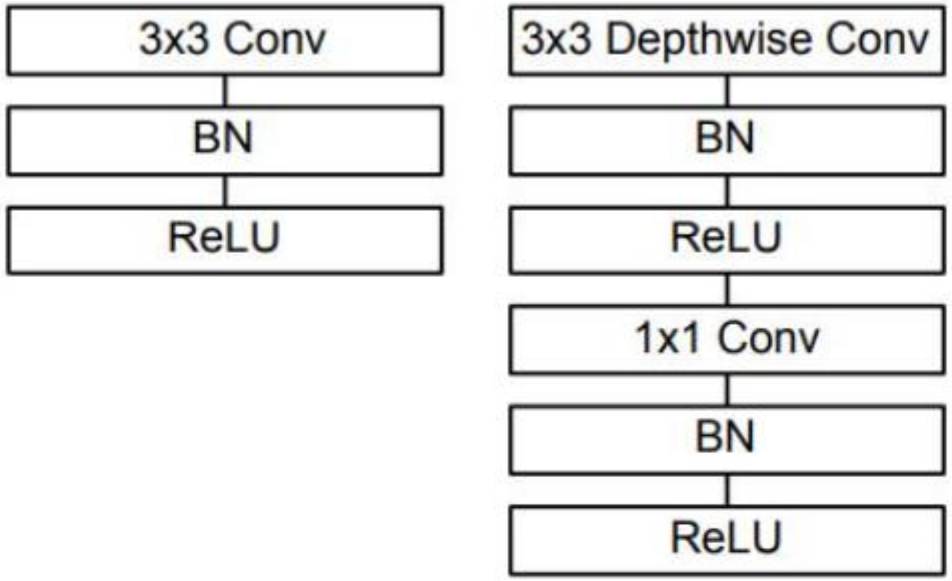


그림 5. MobileNet 기본 구조

BN(Batch Normalization) ?

각 레이어마다 배치 정규화 과정을 통해
가중치의 차이를 완화하여 보다 안정적 학습을 추구

MobileNet Comparison to Popular Model

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

표. 4 MobileNet과 유명 CNN 구조와 성능 비교

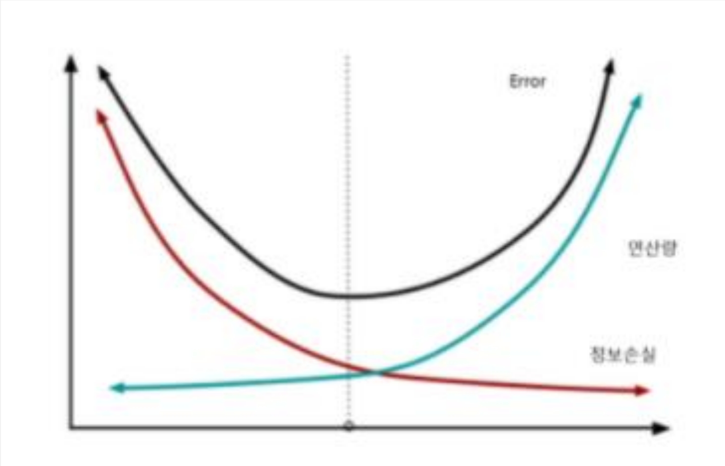
다른 유명한 CNN 알고리즘과 비교했을 경우에도 Parameter와 Multi-adds(연산량)가 더 적음에도 불구하고 성능에 큰 차이가 없음.

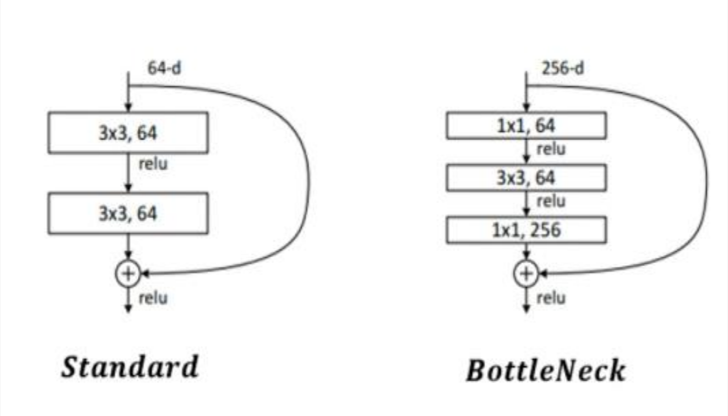
MobileNet v2

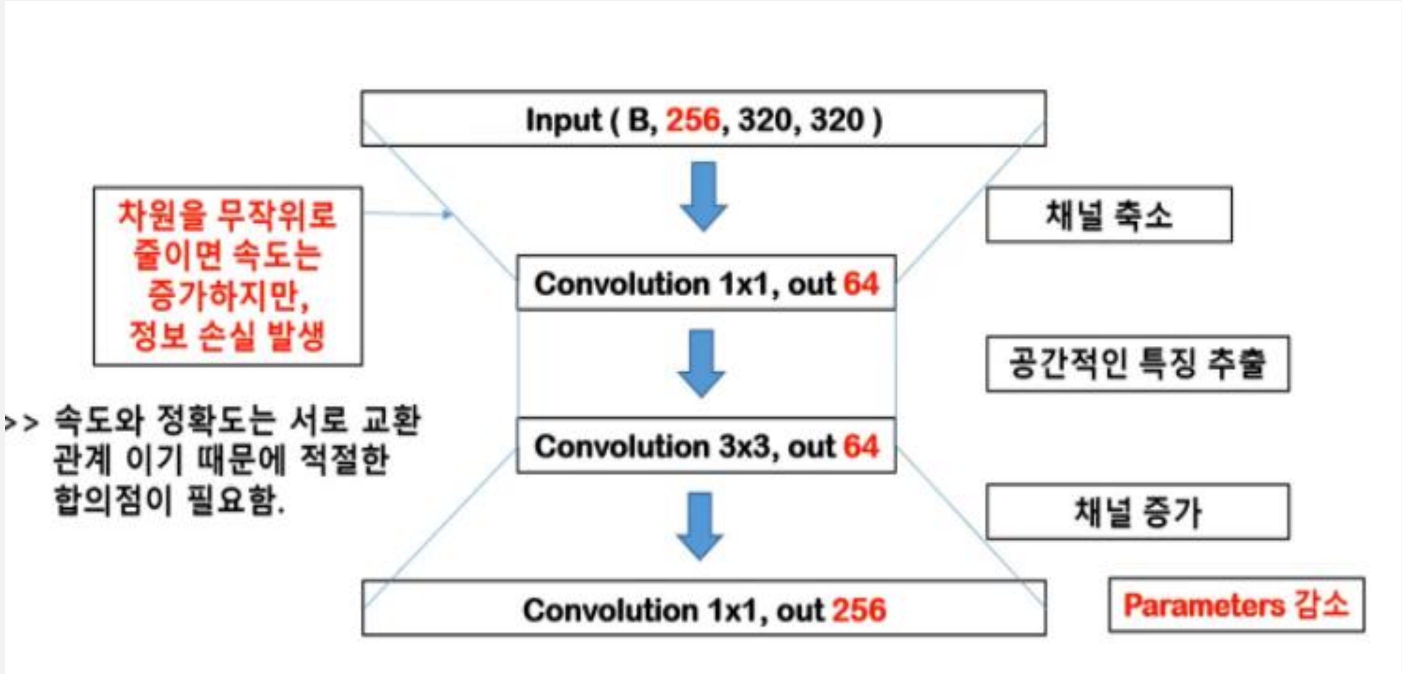
Linear Bottlenecks

Inverted Residual

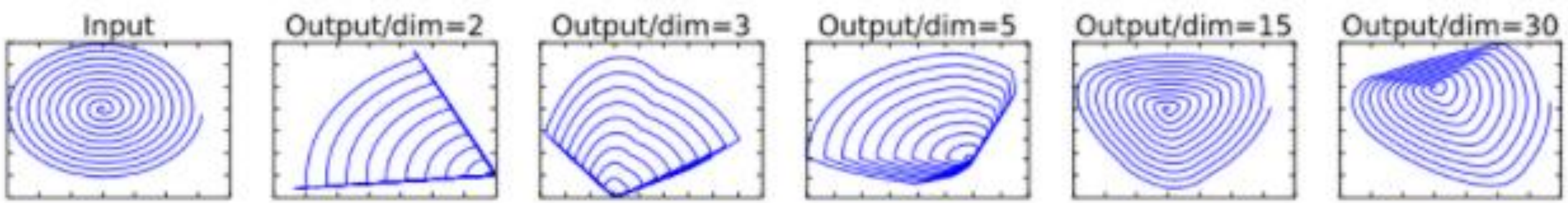
Bottlenecks layer



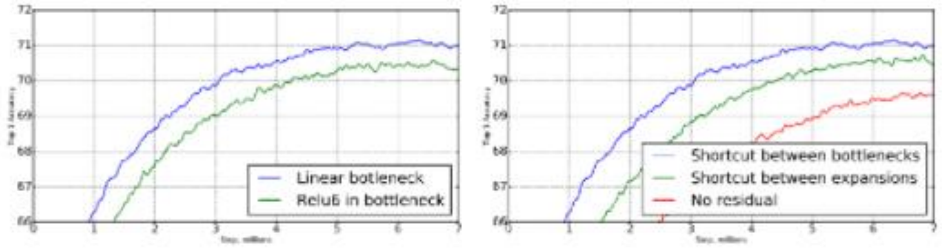




Linear Bottlenecks

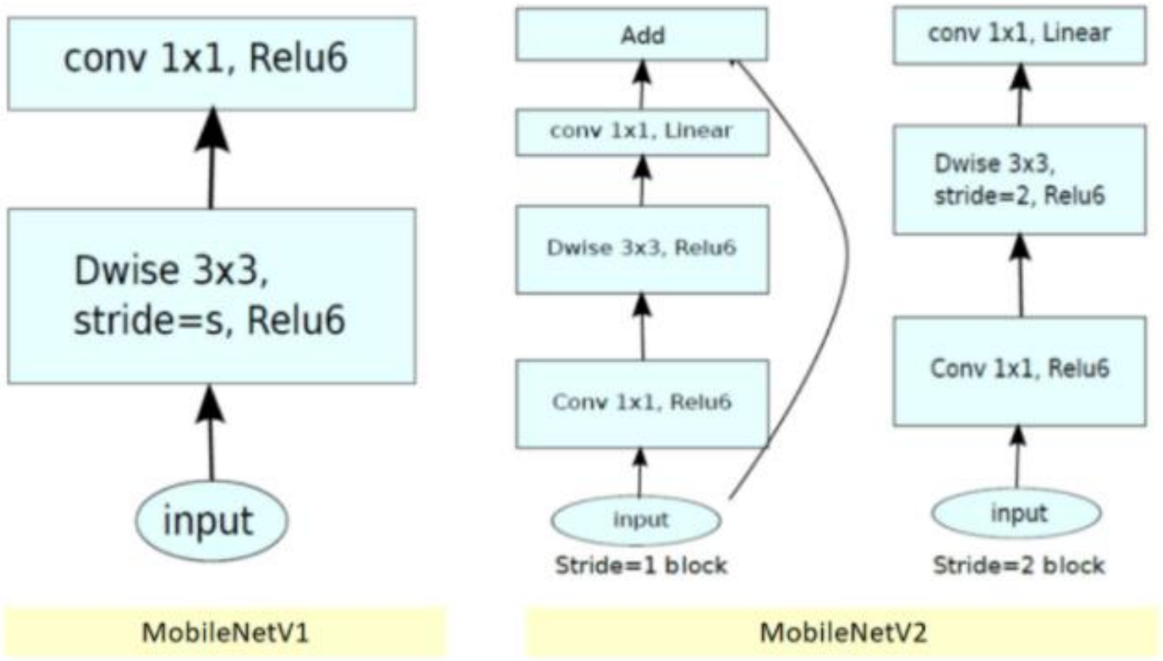


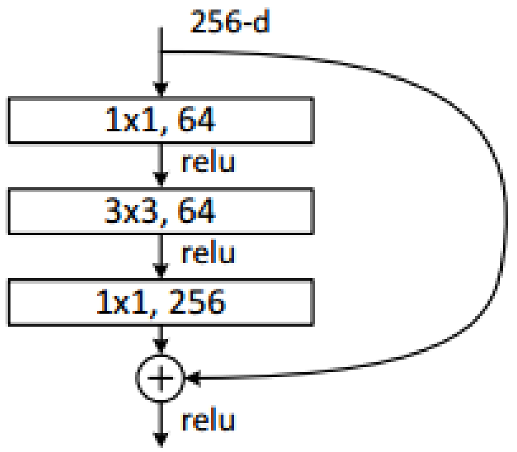
입력 값이 채널 수가 적은 ReLU 계층을 통과하게 되면 정보의 손실이 발생,
반면에 입력 값이 채널 수가 많은 레이어를 통과하는 경우 정보가 보존됩니다.



(a) Impact of non-linearity in the bottleneck layer. (b) Impact of variations in residual blocks.

Inverted Residual

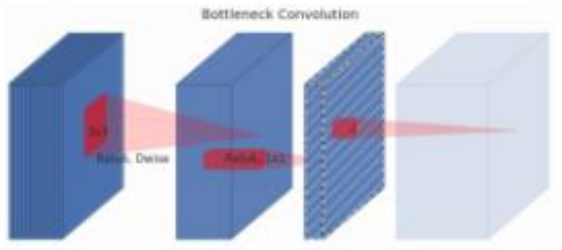




기존 BottleNeck

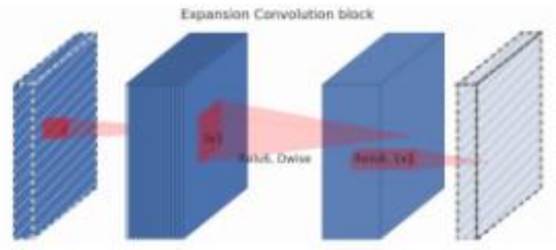
Inverted Residual

(c) Separable with linear bottleneck



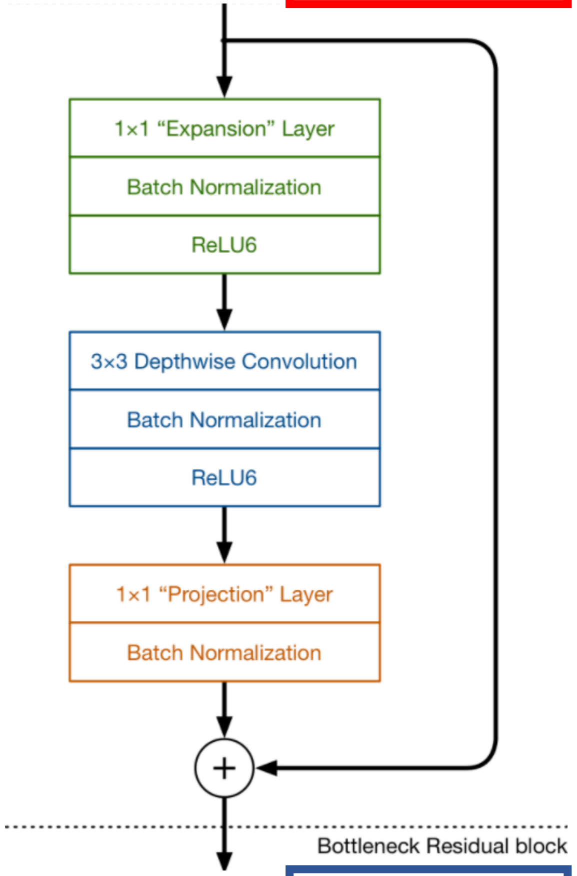
Wide -> Narrow -> Wide

(d) Bottleneck with expansion layer



Narrow -> Wide -> Narrow

No ReLU



No ReLU

MobileNet Comparison to Other Model

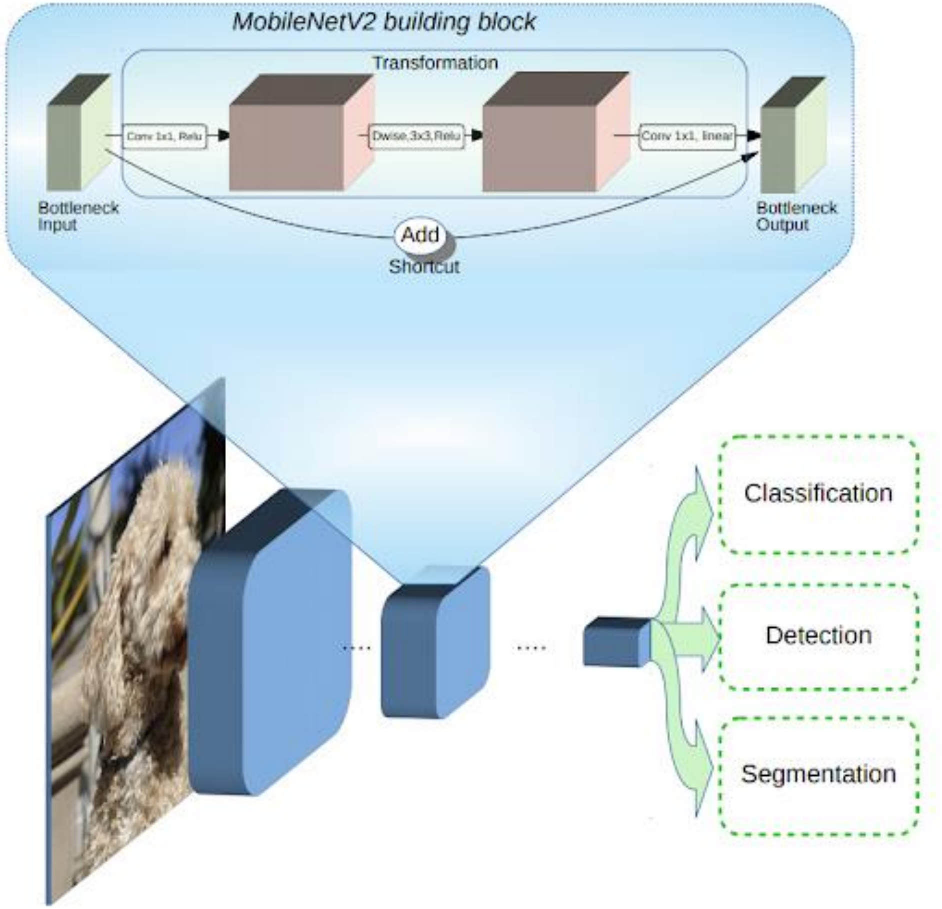
Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	64/1600	16/400	32/800
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	1600K	**400K**	600K

Table 5: Comparison of the size and the computational cost between SSD and SSDLite configured with MobileNetV2 and making predictions for 80 classes.

Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

** SSD?
사진의 변형 없이 그 한장으로 훈련, 검출

Modeling



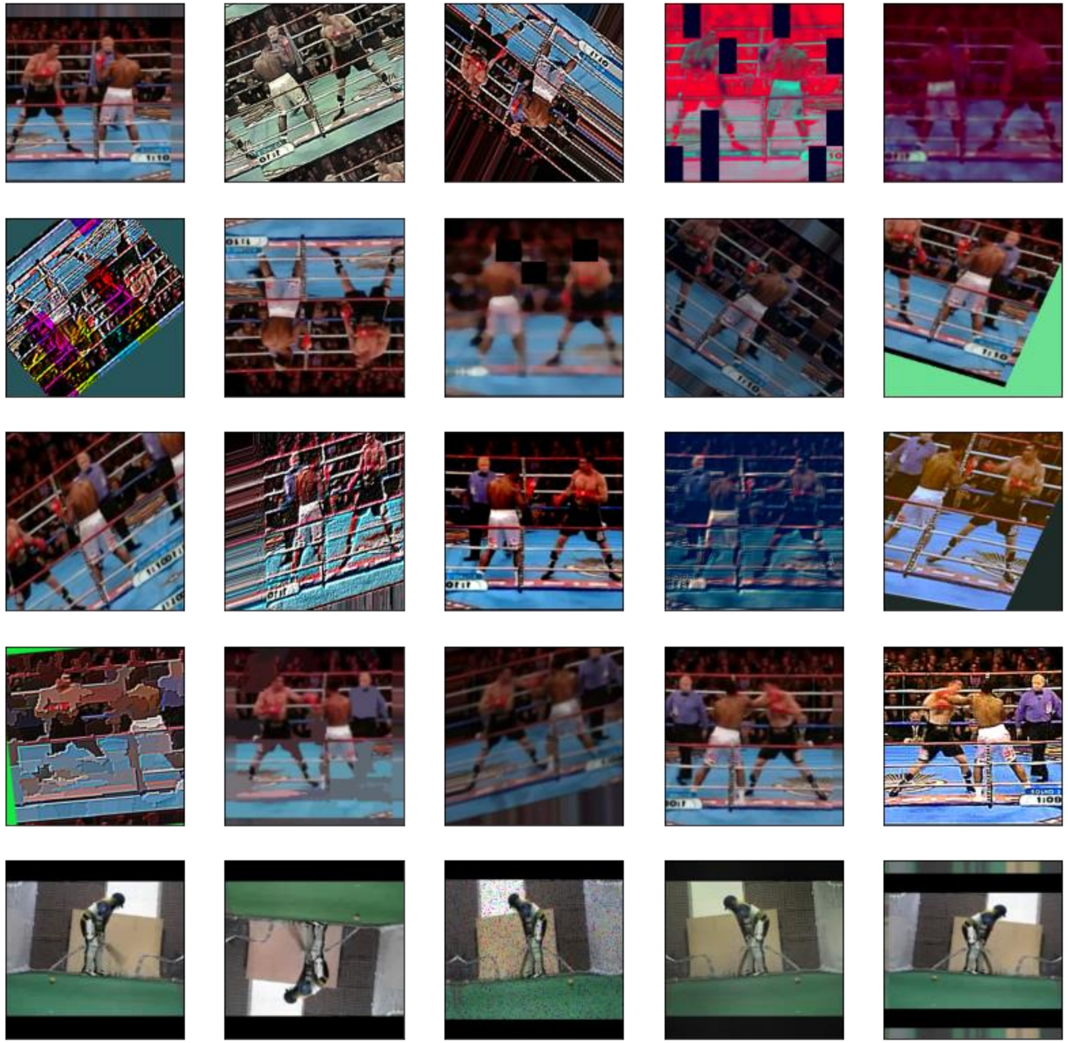
Overview of MobileNetV2 Architecture. Blue blocks represent composite convolutional building blocks as shown above.

001 Image Augmentation - ImgAug

002 VGG16 vs MobileNet v1 vs v2

Image Augmentation – ImgAug

ImgAug



```
# don't execute all of them, as that would often be way too strong
iaa.SomeOf((0, 5),
    [
        sometimes(iaa.Superpixels(p_replace=(0, 1.0), n_segments=(20, 200))), # convert images into their superpixel representation
        iaa.OneOf([
            iaa.GaussianBlur((0, 3.0)), # blur images with a sigma between 0 and 3.0
            iaa.AverageBlur(k=(2, 7)), # blur image using local means with kernel sizes between 2 and 7
            iaa.MedianBlur(k=(3, 11)), # blur image using local medians with kernel sizes between 2 and 7
        ]),
        iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)), # sharpen images
        iaa.Emboss(alpha=(0, 1.0), strength=(0, 2.0)), # emboss images
        # search either for all edges or for directed edges,
        # blend the result with the original image using a blobby mask
        iaa.SimplexNoiseAlpha(iaa.OneOf([
            iaa.EdgeDetect(alpha=(0.5, 1.0)),
            iaa.DirectedEdgeDetect(alpha=(0.5, 1.0), direction=(0.0, 1.0)),
        ])),
        iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5), # add gaussian noise to images
        iaa.OneOf([
            iaa.Dropout((0.01, 0.1), per_channel=0.5), # randomly remove up to 10% of the pixels
            iaa.CoarseDropout((0.03, 0.15), size_percent=(0.02, 0.05), per_channel=0.2),
        ]),
        iaa.Invert(0.05, per_channel=True), # invert color channels
        iaa.Add((-10, 10), per_channel=0.5), # change brightness of images (by -10 to 10 of original value)
        iaa.AddToHueAndSaturation((-20, 20)), # change hue and saturation
        # either change the brightness of the whole image (sometimes
        # per channel) or change the brightness of subareas
        iaa.OneOf([
            iaa.Multiply((0.5, 1.5), per_channel=0.5),
            iaa.FrequencyNoiseAlpha(
                exponent=(-4, 0),
                first=iaa.Multiply((0.5, 1.5), per_channel=True),
                second=iaa.LinearContrast((0.5, 2.0))
            )
        ]),
        iaa.LinearContrast((0.5, 2.0), per_channel=0.5), # improve or worsen the contrast
        iaa.Grayscale(alpha=(0.0, 1.0)),
        # sometimes(iaa.ElasticTransformation(alpha=(0.5, 3.5), sigma=0.25)), # move pixels locally around (with random strengths)
        # sometimes(iaa.PiecewiseAffine(scale=(0.01, 0.05))), # sometimes move parts of the image around
        # sometimes(iaa.PerspectiveTransform(scale=(0.01, 0.1)))
    ],
    random_order=True
)
],
random_order=True
```

VGG16 Model

```
1 #생성된 모델 정보 출력
2 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 21,138,243
Trainable params: 6,423,555
Non-trainable params: 14,714,688
_________________________________________________________________
```

VGG16 Evaluate Result

```
Epoch 1/10
223/223 [==============================] - 84s 329ms/step - loss: 0.8131 - acc: 0.6772
Epoch 2/10
223/223 [==============================] - 74s 333ms/step - loss: 0.4488 - acc: 0.8190
Epoch 3/10
223/223 [==============================] - 77s 346ms/step - loss: 0.3920 - acc: 0.8458
Epoch 4/10
223/223 [==============================] - 72s 323ms/step - loss: 0.3223 - acc: 0.8824
Epoch 5/10
223/223 [==============================] - 75s 334ms/step - loss: 0.2906 - acc: 0.8883
Epoch 6/10
223/223 [==============================] - 75s 335ms/step - loss: 0.2735 - acc: 0.8963
Epoch 7/10
223/223 [==============================] - 73s 328ms/step - loss: 0.2540 - acc: 0.9037
Epoch 8/10
223/223 [==============================] - 75s 338ms/step - loss: 0.2473 - acc: 0.9053
Epoch 9/10
223/223 [==============================] - 74s 331ms/step - loss: 0.2348 - acc: 0.9135
Epoch 10/10
223/223 [==============================] - 75s 337ms/step - loss: 0.2440 - acc: 0.9051
0:12:35.473479
실행시간: 0:12:35
```

```python
# 이미지 테스트 cost와 정확도 리턴
# batch_size = 64 : 한번에 64개씩 테스트 해서 cost와 정확도의 평균을 계산
model.evaluate(
    X_test/255, y_test, batch_size = 64
)
```

```
43/43 [==============================] - 9s 169ms/step - loss: 0.1999 - acc: 0.9288
[0.19994378089904785, 0.9288321137428284]
```

MobileNet_v1 Model


```
#생성된 모델 정보 출력
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functio  (None, 7, 7, 1024)       3228864
 nal)

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 256)               12845312

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 16,074,947
Trainable params: 12,846,083
Non-trainable params: 3,228,864
_________________________________________________________________
```

MobileNet_v1 Evaluate Result

```
Epoch 1/10
223/223 [==============================] - 87s 346ms/step - loss: 1.0255 - acc: 0.7357
Epoch 2/10
223/223 [==============================] - 76s 341ms/step - loss: 0.3708 - acc: 0.8569
Epoch 3/10
223/223 [==============================] - 80s 359ms/step - loss: 0.3010 - acc: 0.8813
Epoch 4/10
223/223 [==============================] - 76s 341ms/step - loss: 0.2523 - acc: 0.9056
Epoch 5/10
223/223 [==============================] - 77s 344ms/step - loss: 0.2296 - acc: 0.9100
Epoch 6/10
223/223 [==============================] - 78s 348ms/step - loss: 0.2026 - acc: 0.9212
Epoch 7/10
223/223 [==============================] - 76s 339ms/step - loss: 0.1916 - acc: 0.9250
Epoch 8/10
223/223 [==============================] - 77s 344ms/step - loss: 0.2006 - acc: 0.9216
Epoch 9/10
223/223 [==============================] - 78s 349ms/step - loss: 0.1944 - acc: 0.9247
Epoch 10/10
223/223 [==============================] - 78s 350ms/step - loss: 0.1951 - acc: 0.9256
0:13:02.523512
실행시간: 0:13:02
```

```
# 이미지 테스트 cost와 정확도 리턴
# batch_size = 64 : 한번에 64개씩 테스트 해서 cost와 정확도의 평균을 계산
model.evaluate(
    X_test/255, y_test, batch_size = 64
)
```

```
43/43 [==============================] - 3s 54ms/step - loss: 0.1334 - acc: 0.9672
[0.13340088725090027, 0.9671533107757568]
```

MobileNet_v2 Model

```
1 #생성된 모델 정보 출력
2 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 62720)             0

 dense (Dense)               (None, 256)               16056576

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 3)                 771

=================================================================
Total params: 18,315,331
Trainable params: 16,057,347
Non-trainable params: 2,257,984
_________________________________________________________________
```

MobileNet_v2 Evaluate Result

```
Epoch 1/10
223/223 [==============================] - 89s 353ms/step - loss: 1.2197 - acc: 0.6958
Epoch 2/10
223/223 [==============================] - 79s 355ms/step - loss: 0.4648 - acc: 0.8088
Epoch 3/10
223/223 [==============================] - 83s 371ms/step - loss: 0.4085 - acc: 0.8272
Epoch 4/10
223/223 [==============================] - 77s 343ms/step - loss: 0.3518 - acc: 0.8594
Epoch 5/10
223/223 [==============================] - 80s 359ms/step - loss: 0.3239 - acc: 0.8652
Epoch 6/10
223/223 [==============================] - 79s 354ms/step - loss: 0.3074 - acc: 0.8774
Epoch 7/10
223/223 [==============================] - 77s 345ms/step - loss: 0.3123 - acc: 0.8816
Epoch 8/10
223/223 [==============================] - 78s 352ms/step - loss: 0.2896 - acc: 0.8879
Epoch 9/10
223/223 [==============================] - 79s 353ms/step - loss: 0.2739 - acc: 0.8890
Epoch 10/10
223/223 [==============================] - 80s 359ms/step - loss: 0.2904 - acc: 0.8872
0:14:18.809453
실행시간: 0:14:18
```

Using ImageGenerator

```python
# 이미지 테스트 cost와 정확도 리턴
# batch_size = 64 : 한번에 64개씩 테스트 해서 cost와 정확도의 평균을 계산
model.evaluate(
    X_test/255, y_test, batch_size = 64
)
```

```
43/43 [==============================] - 4s 65ms/step - loss: 0.0834 - acc: 0.9723
[0.08335311710834503, 0.9722627997398376]
```

Using ImgAug / Shuffle Image

```python
# 이미지 테스트 cost와 정확도 리턴
# batch_size = 64 : 한번에 64개씩 테스트 해서 cost와 정확도의 평균을 계산
model.evaluate(
    X_test/255, y_test, batch_size = 64
)
```

```
43/43 [==============================] - 4s 65ms/step - loss: 0.0812 - acc: 0.9723
[0.08119206875562668, 0.9722627997398376]
```

Using ImgAug

```python
# 이미지 테스트 cost와 정확도 리턴
# batch_size = 64 : 한번에 64개씩 테스트 해서 cost와 정확도의 평균을 계산
model.evaluate(
    X_test/255, y_test, batch_size = 64
)
```

```
43/43 [==============================] - 4s 65ms/step - loss: 0.0832 - acc: 0.9737
[0.08323327451944351, 0.9737226366996765]
```

Performance Comparison

Model	# of Parameters	Accuracy
VGG16	14714688	0.9288
MobileNet_v1	3228864	0.9672
MobileNet_v2	2257894	0.9737

Comments

Reference

https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html - mobilenet_v2

https://arxiv.org/abs/1704.04861 - mobilenet_v1 paper

https://arxiv.org/abs/1801.04381 - mobilenet_v2 paper