/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 01 02 : ResNet 논문 시작**

1.ResNet의 배경과 그 의의

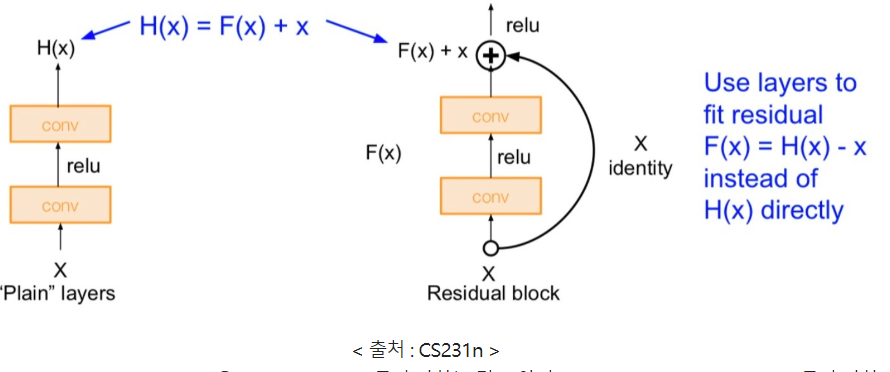
-일반적인 네트워크에서는 레이어를 더 많이 쌓을수록 vanishing gradient 문제나 performance degradation문제가 있었음->이를 resnet을 통해 극복했다.

-이전의 VGG,GoogLeNet은 레이어 수가 20 내외였음 -> resNet은 152개의 레이어를 쌓았음(더 깊은 레이어로 성능을 낼 수 있다.)

-기존의 질문:네트워크가 deep할수록 더 좋은 성능을 보이지 않을까? => degradation problem: 망이 깊어지면 vanishing/exploding gradient문제에 의해 성능이 떨어짐. (overfitting의 원인이 아닌 model의 depth가 깊어짐에 따라 training-error가 높아짐)

2. residual Learning

H(x)이 학습해야 할 타겟 함수, x가 input이라하면 F(x) = H(x)-x로 하여 F(x) + x를 H(x)에 근사하도록 하는 것(이 방법이 H(x)가 여러 비선형 layer로 이루어져 있을 때 더 쉽다). 그리고 이를 feed-forward neural network에 적용한 것이 shortcut connection이다.



-왼쪽: 일반 구조 convolution neural network. 입력이 들어오면 layer를 거쳐 연산들이 연속적으로 이루어짐

-오른쪽(residual 구조): 입력을 출력과 더해줌. 이 때, 입력에서 출력으로 더해지는 값(x)를 identity라 한다. 즉, feature를 추출하기 전 후를 더하는 특징이 있음

=> x값 역시 전달 받으므로 더 작은 변화에도 민감하게 반응하여 학습이 가능

=>기존의 vanishing gradient 문제 개선. 즉, 레이어가 깊어져도 학습이 잘 된다.

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 01 07:ResNet 논문 리딩**

1.Network Architecture(Plain network와 residual network 비교)

1)plain network: layer들을 단순히 쌓은 네트워크. 해당 네트워크는 VGGNet을 많이 모방함. 해당 network에는 2가지 룰이 존재한다.(각 layer들은 입출력의 feature map size가 같을 때, 동일한 크기의 필터를 사용한다. 만약 feature map size가 절반이 되면 filter의 개수를 2배로 한다.feature map 크기를 줄이는 down-sampling을 할 때엔 pooling말고 stride = 2로 한다.)

2) Residual network: plain network에 기반하여 shortcut connection을 추가한 ResNet.입출력의 feature map size가 동일하면 identity shortcut을 사용하고, feature map size가 증가할 때는 stride=2로 하여 두 방안 중 하나를 이용함(identity shortcut을 사용하되, 증가한 dimension 맞출 때는 1 X1 conv사용하거나 0성분을 붙임)

2. Experiments

1. ImageNet classification(ImageNet 2012 classification dataset 사용)

-Plain Networks:34-layer에서 validation error > 18-layer validation error(degradation문제 발견). 하지만 backward,forward모두 문제가 없었고, BN을 통해 학습했다는 점에서 vanishing gradient문제는 아님

-Residual Networks: layer가 깊어질수록 error는 2.8% 감소. 또한 traning error가 줄어들고 있어 일반화를 잘 하고 있음을 알 수 있음=>degradation 문제 해결.

=>더 깊은 system의 residual을 학습하는 것이 더 효율적. 또한 plain network보다 빠르게 optimization을 수행함

1. Deeper BottleNeck architectures(34-layer보다 깊은 망일 때 사용)

-차원이 큰 input 처리시, 1x1 filter로 차원 축소 -> 3 x3 filter로 feature 추출 -> 1x1 filter로 차원 원상복구를 시킴. 이 때 parameter-free shortcut방식을 사용하여야 한다.

-실험 결과: ResNet-50, ResNet-101, ResNet-152로 갈수록 성능이 좋아짐. Degradation문제는 발생하지 않았음

3. 총 요약(가져갈 내용)

Skip connection을 이용한 Residual Block, Bottleneck Block을 제안함. Highway network보다 단순화된 형태로써 더 optimize하기 쉽고, 더 깊은 모델로부터 쉽게 accuracy를 얻을 수 있음을 증명해 내었음.

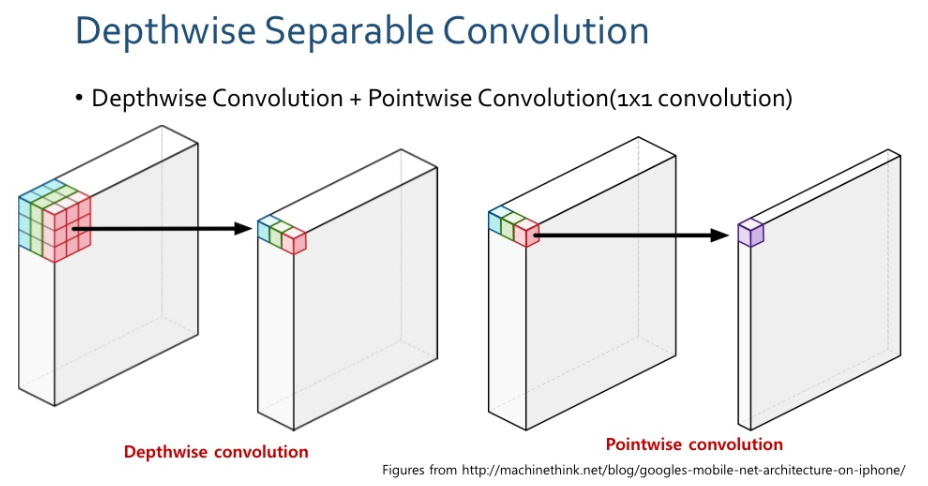
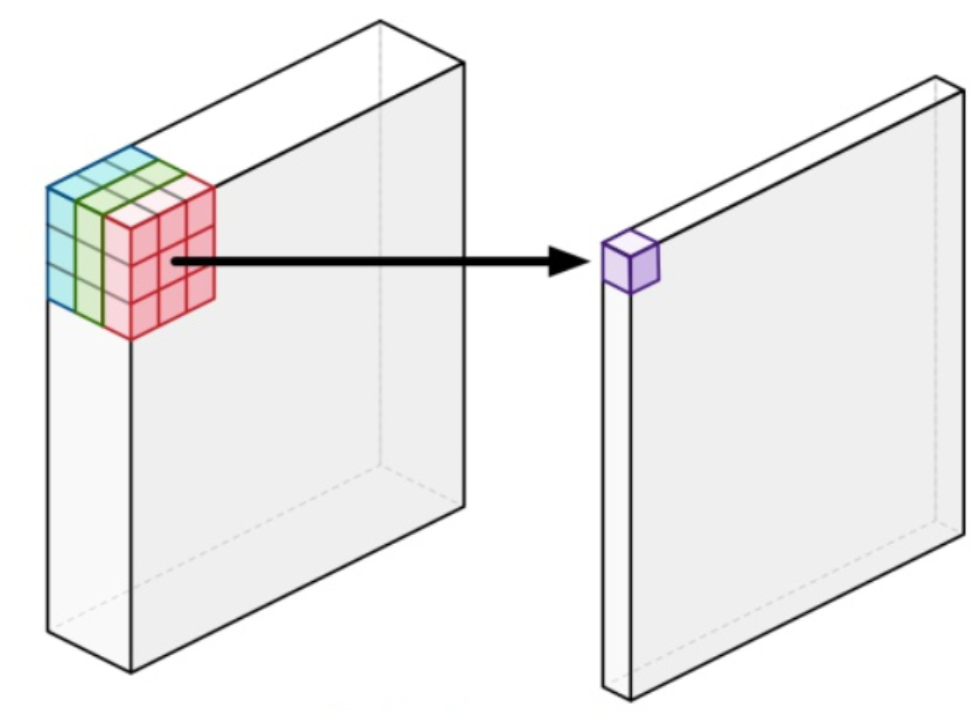
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 01 11: MobileNet V1 논문 시작**

1.핵심 아이디어& 특징

* 모바일, 디바이스 등에서 사용될 목적으로 설계된 CNN 구조. Xception에서 사용한 Depthwise separable convolution 방법을 사용하여 기존의 CNN 구조에 비해 파라미터 수를 줄였다.
* MobileNet의 Depthwise separable convolution연산: Xception과 거의 같음(depthwise convolution 연산과 pointwise convolution 연산 사이에도 batch normalization과 ReLU 활성화 함수 단계가 추가된 된 점이 차별성)

2. mobileNet V1

1. depthwise separable convolution

(기존 convolution) (depthwise separable convolution)

* 아이디어:input data에 필터 적용시, 모든 채널을 한번에 계산해서 아웃풋을 만들어야 한다는 점(왜 여러 채널 모두에 연산을 다 해야 하는가) ->기존 convolution이 전체의 channel방향과 spatial한 방향 모두를 한꺼번에 고려했다면, 이제 그 둘을 완전히 분리하겠다
* 목적: 파라미터 개수와 연산량을 줄임으로써 연산 속도를 증가시킴
* 연산 과정: Depthwise Convolution에서는 채널 방향은 고려하지 않음(연산 결과가 채널 개수만큼 나옴) => Pointwise Convolution에서 채널 방향 연산 수행( 1 x 1 convolution을 적용)
* 파라미터 개수와 연산량: Depthwise conv 후 Pointwise conv를 수행하는 방식이므로 두 방식의 param을 더한다. 총 param수는 K2C + CM = ***C(K2 + M).*** 연산량 은 두 작업을 더하므로 K2CHW + CMHW = ***CHW(K2 + M)*** .Standard Conv의 연산량 : ***K2CMHW이므로 연산량에서*** 연산량에서 약 ***K2배***의 차이
* ( ***W : input의 width,H : input의 height,C : input의 channel,K : kernel의 크기,M : output의 channel)***

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 01 16: 공백**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 01 22:MobileNet V1 논문 리딩**

1.MobileNet V1 (이어서): shrinking hyperparameter

- depthwise separable convolution 연산 방법 외에 네트워크의 사이즈를 좀 더 줄이기 위해 추가 하이퍼파라미터를 사용(값의 범위는 0~1)

1-1)α (width multiplier)

- input channel과 output channel을 조절해주는 parameter

-인풋과 아웃풋의 채널에 곱해짐. 전체 channel 수를 줄이기 위해 사용.

- α의 값은 1, 0.75, 0.5, 0.25를 가질 수 있으며, 기본은 α = 1

1-2)**ρ (**Resolution Multiplier)

- input 이미지의 크기를 조절하는 parameter

-input의 height와 width에 곱해짐. 이미지 크기를 줄이기 위해 사용.

-기본적으로 (224, 224, 3) 이미지를 인풋으로 넣고 실험해본 결과 파라미터 값은 1, 0.857, 0.714, 0.571이 됨. (사이즈 기준 224, 192, 160, 128)

2.측정 결과

* narrow network(네트워크의 height, width가 작은 것)와 shallow network(네트워크의 깊이가 얕은 것) 비교: narrow가 더 성능이 좋다.네트워크 경량화를 해야 한다면 height,width를 줄이고 깊이있게 쌓자.
* Width Multiplier와의 관계: width multipler가 줄어들수록 accuracy감소. channel의수가 줄어들면 param수와 연산량이 줄어들지만 그에따라 Accuracy도 감소한다.
* Resolution과의 관계 :resolution이 감소할수록 연산량이 감소하게 되고, accuracy도 감소한다.

3. 해당 논문의 의의

* Small Deep Neural Network가 중요한 이유?

네트워크를 작게 만듦-> 학습 속도 향상 및 임베디드 환경에서 딥러닝 구성이 용이해짐. 만약 무선 업데이트를 사용하는 시스템의 경우, 적은 용량을 사용할 수록 빠르게 업데이트 해주면서 통신 비용 및 속도 면에서 효율적이다.

* MobileNet의 출시 의의: 임베디드 시스템에서 실행할 수있는 모델의 등장. 이로 인해 ShuffleNet (V1 및 V2), MNasNet, CondenseNet, EffNet 등을 비롯한 여러 연구들이 후속으로 나올 수 있게 됨

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 01 29: SqeezeNet 논문 시작**

1. introduction & motivation

-파라미터 수가 적을 때의 장점들(간단하게 내용만 이해하고 넘김)

1. More efficient distributed training병렬학습 때 효율이 좋음
2. Less overhead when exporting new models to clients: 실시간으로 서버와 소통해야 하는 시스템에서 좋음
3. Feasible FPGA and embedded deployment: 10MB 이하의 휘발성 메모리를 가진 FPGA에 모델을 올릴 수 있음. 또한 inference할 경우 생기는 병목현상(bottleneck)없어짐

2. related work

1)model compression

-해당 논문의 핵심은 정확도를 유지하면서도 매개변수가 없는 모델을 찾는 것: 기존의 CNN model을 압축해보자.

-기존 연구들 : SVD( singular value decomposition-특이값 분해)를 프리트레이닝된 CNN에 적용하거나, Network Pruning을 적용하는 방법이 있었음.( Network Pruning에 양자화와 허프만 코딩을 결합한 방식도 제시되었음)

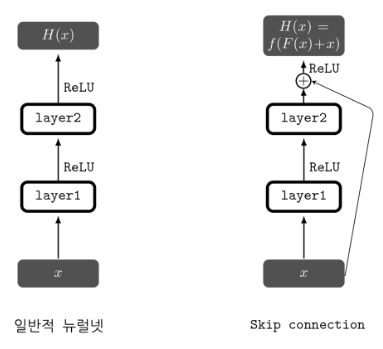
2) CNN CNN Microarchitecture & Macroarchitectur**e**

-CNN Micro: 모델의 일부,특정 부분(convolution filter의 크기나 갯수),

-CNN macro: 모델 전체, input부터 끝까지 전체 구조.여러 개의 모듈이 end-to-end CNN 구조로 구성됨(VGGNet, ResNet, DenseNet)

-skip connection의 유무로 둘을 구분한다.

추가 :skip connection



-네트워크의 입력과 출력이 더해진 것을 다음 레이어의 입력으로 사용하는 것

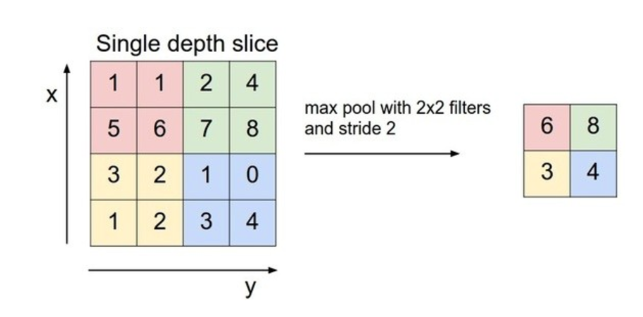
-스킵 연결을 구현 하는 것은 덧셈 연산의 추가 만으로 가능하다. 이는 추가적인 연산량이나 파라미터가 많이 필요하지 않다. 또한 역전파 시에 그레디언트가 잘 흘러갈 수 있게 해준다는 장점도 있음

3. squeezenet

-매개변수 크기를 줄이면서(1,2번 전략) 정확도를 최대한 보존시키는(3번 전략) 3가지 방법(위 전략을 이용해서 후에 fire module을 적용한다.)

1. Replace 3x3 filters with 1x1 filters: 3x3 convolution filter -> 1x1 fileter로 교체(파라미터 개수 1/9로 줄임)
2. Decrease the number of input channels to 3x3 filters: 인풋 채널의 수를 줄여야 한다.( 파라미터의 총 수는 (인풋 채널 개수) x (필터 개수) x (레이어의 필터 사이즈)이므로 )
3. Downsample late in the network so that convolution layers have large

activation:Downsampling part를 네트워크 후반부에 집중시킴( 큰 activation map을 가지고 있을수록 성능이 더 높으므로)

-추가: downsample이란?(spatial Pooling)

(max pooling)

이미지의 특성상 매우 많은 픽셀과 feature이 존재함-> 원활한 학습을 위하여 feature를 줄이는 subsampling을 해야 함. subsampling이란max pooling 또는 필터 자체의 stride를 높이는 방식. 이미지의 spatial resolution을 줄임 -> 필터가 볼 수 있는 영역을 좁힘-> 이미지의 정보를 압축시킴

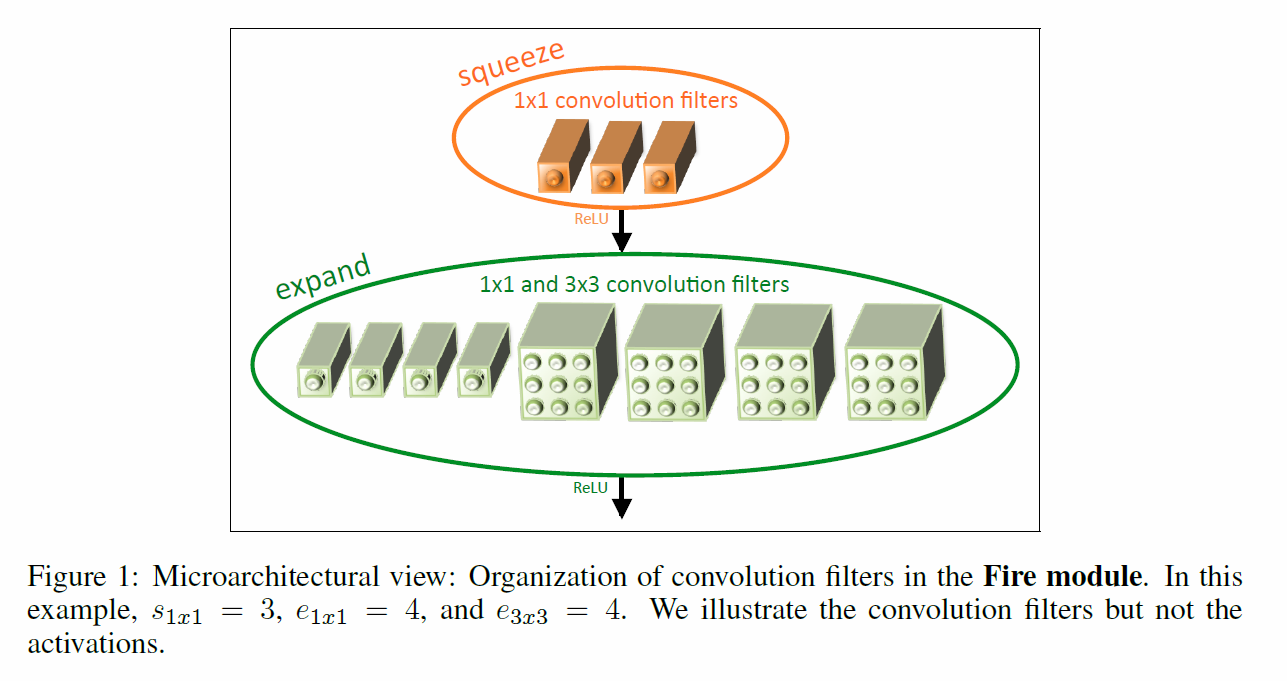
sampling 을 통해서 얻을 수 있는 장점

* 전체 data의 사이즈가 줄어들기 때문에 연산에 들어가는 컴퓨팅 리소스가 적어지고
* 데이타의 크기를 줄이면서 소실이 발생하기 때문에, 오버피팅을 방지할 수 있음

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 02 05:SqeezeNet 논문 리딩**

**1.fire module**

****

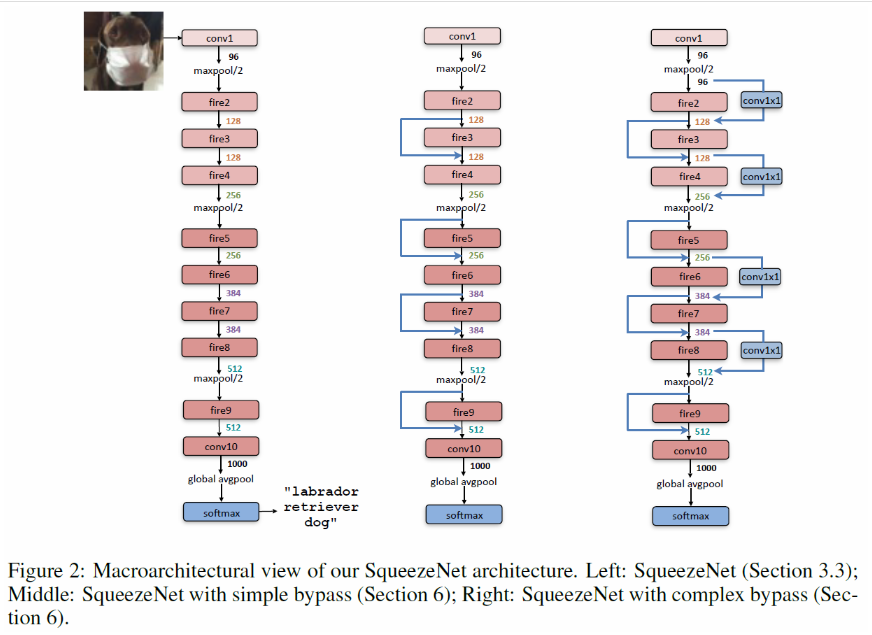
**-**Squeeze layer와 Expansion layer인 2개의 layer로 구성됨

* Squeeze layer
  + 1x1 convolution
  + 역할: input 데이터의 모든 channel을 하나로 합쳐서 channel의 수를 줄여줌.(channel 수를 조절하는 역할)
* expansion layer
  + 1 x 1 convolution이 3 x 3 convolution과 섞임
  + 1 x 1 convolution의 목적: spatial structure를 감지하기 어렵지만 이전 layer의 channel들을 다양한 방법으로 결합시켜주기 위함
  + 3 x 3 convolution의 목적: 이미지 내의 structure를 잡아내기 위함
  + \*주의할 점: 1 x 1 convolution과 3 x 3 convolution 결과의 크기가 같아야 하므로 padding에 유의할 것

-하이퍼 파라미터(3가지)

:squeeze layer에서 1x1 필터의 총 갯수,: expand layer에서의 1x1 필터 총 갯수,expand layer에서의 3x3 필터의 총 갯수.  < + (input channel 수를 제한해줌)

2. SqueezeNet Architecture



* 특징: 총 8개의 fire module + input/output 각각에 1개의 convolution layer 사용.
* 연산 순서:convolution layer -> max pooling -> fire module -> convolution filter -> GAP(Global Average Pooling). Pooling layer를 conv1, fire4, fire8, conv10 이후에 배치(activation map을 크게 함)
* 핵심: Fully Connected Layer를 사용하지 않았다.(fully connected layer 대신 Global Average Pooling을 사용) Fully connected layer을 사용하면 너무 많은 파라미터를 가져야 하며, 이에 따른 overfitting의 문제도 발생 가능함. 이를 해결하기 위해 Global Average Pooling 사용(이전의 convolution layer로 부터 각각의 channel을 받은 후, 모든 값에 대하여 average 구하는 방식)

=>이점: weight 값이 없기 때문에 model size를 늘리지 않고, overfitting발생 확률이 줄어든다.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 02 11:비움**

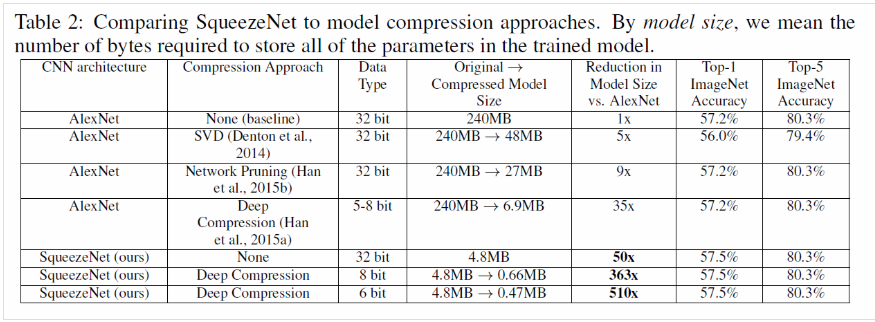
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 02 15:SqeezeNet 논문 리딩**

1.SqeezeNet 설계 시 세부 사항(간단 정리)

* 1x1과 3x3 필터의 output activation이 같기 때문에 expand module의 3x3 필터로 들어가는 데이터에 1-pixel짜리 zero padding 추가
* Squeeze와 expand layer 모두 ReLU 적용
* Fire9 module 이후로 dropout 50% 적용
* 초기 학습률 0.04로 설정 후 점차 감소시킴

2. 정확도 요약 및 정리



squeezeNet사용시:50배 정도 모델 사이즈 감소, 성능은 AlexNet의 top-1 & top-5 accuray에 비슷하거나 이보다 좋음. 압축되지 않은 32bit data를 사용시 & deep compression을 적용한 8bit, 6bit짜리 데이터 타입을 사용시:모델 사이즈 최대 510배 줄어들었음

-정리: 논문에서는 AlexNet과 비슷한 성능을 내면서 용량을 1/50으로 줄인 네트워크 생성. 정확도를 높이는 방안만을 고려하기 보단 파라미터 수를 줄이는 등 사이즈를 줄이는 방식으로도 획기적인 네트워크를 만들 수 있다.

3.SqueezeNet 특징 & 총 요약

* 스퀴즈넷의 질문의 시작점: 만약 같은 성능(정확도)를 보여준다면, 모델이 작으면 작을수록 좋을 것이다.(이전까지 CNN은 정확도를 높이는 연구만 집중되었음)
* SqueezeNet: AlexNEt과 비슷한 성능, 파라미터 사이즈는 알렉스 넷의 1/50, 모델 사이즈 0.5MB
* 핵심 이슈:

1. 기존의 3 x 3 필터 일부 -> 1 x 1 필터로 대체(weight의 수를 1/9로 줄임)
2. input 채널 수를 3 x 3 필터 사용(convolution layer에서 파라미터 수는 채널 수, 필터 수에 영향을 받음)
3. downsample을 늦게 적용하여 convolution layer가 큰 activation map을 가짐
   * 데이터 downsample을 늦게 할수록 (예를 들어 stride를 1보다 크게 하여서 downsample 하는 작업을 늦게 적용함) 더 많은 정보들이 layer에 포함될 수 있습니다.
   * 이 방법을 통하여 모델은 가능한 작게 만들지만, 정확도는 최대한 크게 만들 수 있음

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 02 20:MobileNet V3 리딩 시작**

1.introduction(on-device deep learning을 위한 시스템)

on device deep learning은 유저의 정보를 서버와 공유하지 않고도(서버를 사용하지 않고도) 사용자에게 최적화된 구동을 할 수 있게 해야 함. 이 때 고려 요소로 높은 정확도와 효율성(latency가 적음,전력 소모를 줄여 배터리 사용시간을 늘리게끔)이 존재. 즉, 신경망 효율성의 발전들은 높은 정확도와 낮은 지연시간을 통해 사용자 경험을 향상시킬 뿐만 아니라, 감소된 전력 소비를 통해 배터리 수명을 유지하도록 도움을 준다.

=> 논문에서는 accuracy-latency 최적화를 통해 on-device computer vision을 강화하고 더 정확하고 효율적인 모델을 제안( MobiletNetV3 Large, MobiletNetV3 Small). MobileNet V2과 MnasNet 등에서 사용된 layer 들을 기반으로 한 구조이며 swish nonlinearity를 fixed point 연산에 최적화시킨 hard-swish activation function을 제안하였음.

2.relation work(**정확도와 효율성 사이에서 최적의 trade-off에 대한 Deep 신경망 아키텍처 선행 연구들 서술함)**

3. Efficient Mobile Building Blocks//V1, V2의 특징

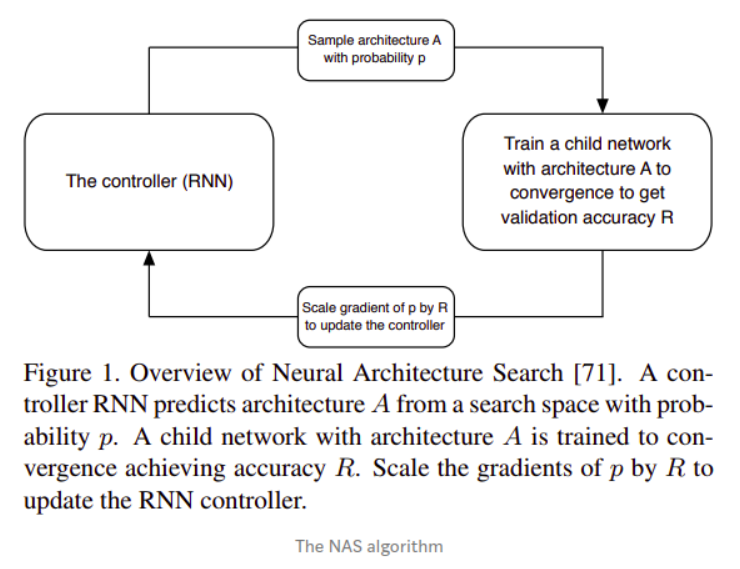
1. V1: 매개 변수 수를 줄이기 위해 convolution layer 대신 **Depth-wise Separable Convolutions** 을 도입.Depthwise Separable Convolutions은 공간 필터링에 대한 가벼운 무게 심층 컨볼루션 (Light Weight Depthwise Convolutions)와 feature 생성에 대한 더 무거운 1 x 1 Pointwise Convolutions으로 이루어진다.
2. V2:훨씬 더 효율적인 계층 구조를 만들기 위해 (중요도가 떨어지는 정보의 영향력을 고려해 모델의 효율화를 시킬 수 있는) linear bottleneck과 inverted residual structure를 제안.
3. MnasNet: Bottleneck에 압착 (Squeeze) 와 자극 (Excitation) 에 기초한 경량 주의 모듈들을 도입. 이 때 압착과 자극 모듈이 RestNet과 다른 위치에서 통합된다.
4. V3: 위 방법들의 조합을 building block으로 사용.
   1. layer: swish nonlinearity
   2. sqeeze & excitation: swish nonlinearity
   3. sigmoid: hard sigmoid 사용(fixed point arithmetic의 정확도를 보존하기 위해)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 02 26:Mobile V3 논문 리딩**

1.  **Platform-Aware** network search: mobileNet V3에서는 전체 네트워크 구조를 탐색하기 위해 NAS를 사용. 그 후 계층 별 필터 개수 탐색 위해 NetAdapt 알고리즘 사용

1. NAS for Block-wise search(전체 네트워크 구조를 찾기 위한 접근법)



* MnasNet과 유사하게 global network structure를 찾기 위해 platform-aware neural architecture approach를 사용한 결과, MNasNet과 유사한 결과를 얻었으며, Large mobile model의 target latency는 80ms을 목표로 함. MnasNet-A1[43]을 initial Large mobile model로 재사용했으며, NetAdapt[48]과 다른 optimization 방법들을 적용시켜 이를 최적화함
* 문제: 이 디자인은 small mobile model에 최적화 되지 못함(초기 보상 설계가 작아서) + 작은 모델의 지연시간에 따라 정확도가 훨씬 더 극적으로 변한다는 것을 관찰
* 해결 방안1:서로 다른 지연시간들에 따른 더 큰 정확도 변화를 보상하기 위해 더 작은 가중치 w = -0.15 (원래는 w = -0.07) 가 필요
* 해결 방안2: 논문의 강화된 새로운 weight factor인 w에 따라 from scratch로 새로운 initial seed model을 NAS를 이용해 찾았음. 그 후 NetAdapt와 other opimization 들을 적용시켜 최종적으로 MobileNetV3-Small 을 찾았다.

\*추가: NAS란?

강화학습을 이용한 뉴럴네트워크 구조 탐색 방법([“Neural Architecture Search with Reinforcement Learning”](https://arxiv.org/pdf/1611.01578.pdf)를 줄여서 NAS라 함)

NAS의 기본 아이디어는 RNN으로 구성된 controller를 이용해 child network를 만들고, 특정 데이터셋 (ex. CIFAR-10)에 대해 validation accuracy가 수렴할 때까지 child network를 학습시킨다. 그러면 수렴한 validation accuracy를 reward로 설정하여 reward의 기댓값을 높이는 방향으로 policy gradient를 이용해 controller를 학습한다.

2)NetAdapt for Layer-wise Search

전체 구조를 추론하기 보다는 순차적인 방식으로 각 계층들을 미세하게 조정할 수 있도록 해줌.

* 진행 방법

1.Platform-aware NAS로 찾은 seed network architecture로 시작

2. 아래 단계로 순차 진행

2-1)새로운 proposal set을 생성. 각 proposal은 이전 step과 비교해서 latency이 최소δ 만큼 감소되는 architecture의 modification을 의미

2-2)각 Proposal의 경우, 우리는 이전 단계에서 사전에 훈련된 (Pre-trained) 모델을 사용하고, 누락된 가중치를 적절하게 잘라내고 무작위로 초기화 하면서 새롭게 제시된 아키텍처를 채움.

2-3)몇몇 지표 (Metric) 에 따라 최상의 Proposal을 선택

3.목표 지연 시간 도달 때까지 위 단계 반복

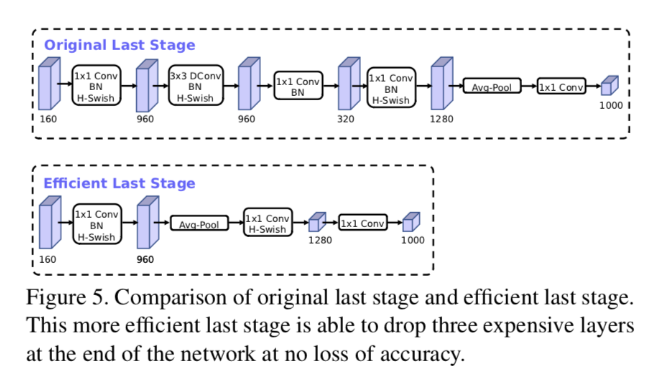
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 03 06:MobileNet V3 논문 리딩**

5.Network Imporvements

논문에서는 최종 모델을 추가적으로 개선하기 위해 일부 새로운 컴포넌트들을 도입하였음.=> 네트워크의 시작과 끝 지점에서 계산적으로 비용이 많이 드는 계층들을 재설계

5-1)Redesigning Expensive Layers

-마지막 계층과 일부 초기 계층들이 cost가 높음을 관찰 -> 정확성을 유지하면서 해당 계층들의 지연 시간을 줄이기 위해 아키텍쳐 수정. 

* 네트워크의 마지막 계층들 재작업 방법: 제일 마지막 계층이었던 1 x 1 Convolution을 최종 Average Pooling 다음으로 이동시킴(최종 Feature map은 7X7 resolution에서 1X1 resolution으로 바뀜). 최종 지연시간을 7ms만큼 감소시키며, 이것은 실행시간의 11%이고, 거의 정확도 손실 없이 3천만 MAdds의 작업 수를 감소시킴.
* 네트워크의 초기 필터 집합 재작업 방법: 기존의 초기 필터인 Full 3 X3 convolution을 사용하지 않는 대신, 16개의 filter를 이용하여 비선형성을 적용함. 즉, 기존의 필터들은 서로 대칭 이미지이기 때문에 이에 따른 중복성을 제거하였다. 이 과정에서 latency 약 2 ms와 천만개(10 million)의 MAdds를 줄일 수 있었음

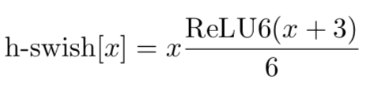
5-2)비선형성

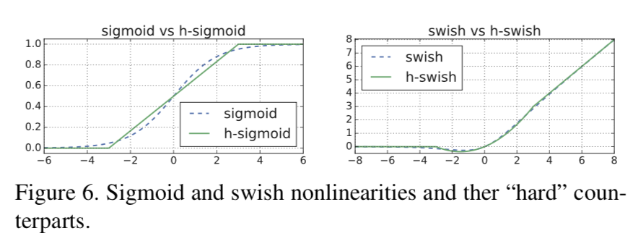


Swish라는 비선형성이 RELU에 대한 drop-in replacement(아주 조금의 노력으로,또는 노력 없이 A에서 B로 바꿔서 성능을 향상시킴).

->문제: 정확도를 개선시키긴 하지만,모바일 기기에서는 sigmoid 계산 cost가 더 많이 든다.(임베디드 환경에서는 0이 아닌 비용 발생)

->해결1: 시그모이드 함수를 단편적인 (Piece-wise) 선형 Hard Analog로 대체함.실제로, h-swish는 단편적인 함수로 구현되어 지연시간 비용을 상당히 절감할 수 있는 메모리 액세스 횟수를 감소시킬 수 있음.





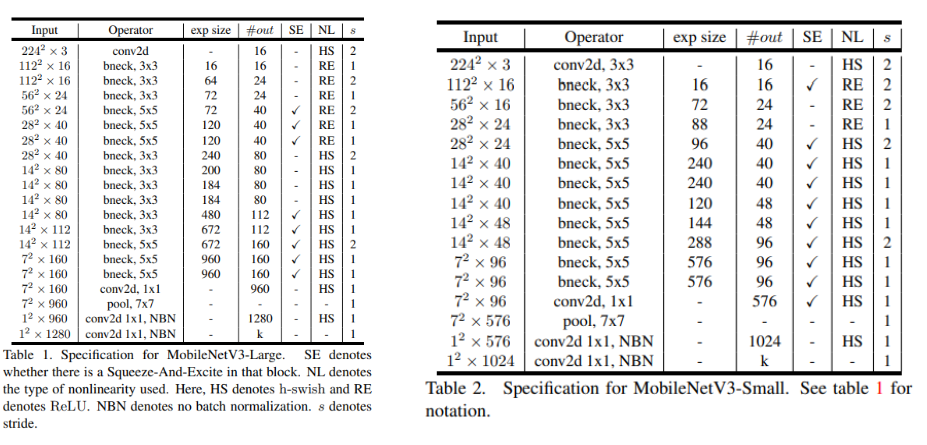
-해결2: 모델의 후반부에서만 h-swish를 사용함.(보통 비선형성을 적용하는 cost는 깊은 네트워크일수록 들어갈수록 감소하는데, Swish의 이점이 대부분 깊은 계층에서 사용할때만 발생한다는 것을 발견했음)

5-3.**Large squeeze-and-excite**

Squeeze와 Excite Bottlenect의 크기를 확장 계층 채널 수의 1 / 4로 고정되도록 대체함

-> 파라미터 수가 약간 증가했지만 정확도는 증가하면서 지연시간의 차이가 거의 없었음

5-4. 위 방식들로 MobileNetV3는 두 가지 모델로 정의됨 :



MobileNetV3-Large. MobileNetV3-Small.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 03 13:비움**

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 03 18:MobileNet V3 논문 리딩**

1. 실험 결과 및 결론

* MnasNet[43], ProxylessNas[5], MobileNetV2[39]과 같은 SOTA 모델들의 성능을 뛰어넘음. classification 외에 object detection, semantic segmentation에도 적용하면 좋은 성능을 보임.
* MobileNetV3의 multiplier와 resolution에 따른 성능 trade-off:multiplier보다 resolution이 더 나은 trade-off를 보여줌. 또한 MobileNetV3-Small이 성능을 맞추기 위해 multiplier가 적용된 MobileNetV3-Large의 성능보다 3%정도 우수. 하지만 resolution은 대게 task에 의해 결정되므로 항상 튜닝 가능한 파라미터로 볼 수는 없음

2.총 결론 및 향후 과제

MobileNetV3 Large와 Small 모델을 제안했으며, mobile classification, detection, segmentation에서 SOTA. 모바일용 모델을 제안하기 위해 네트워크 설계 뿐만 아니라 여러 NAS 알고리즘들을 활용한 것이 특징이다. nonlinearity의 최적화 방법(swish) 역시 보여주었음. 마지막으로 lightweight segmentation decoder인 LR-ASPP를 제안하였다.

->NAS을 인간의 직관과 가장 잘 혼합하는 방법에 대한 open question(하지만 긍정적)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 03 22: shfflenet 논문 리딩 시작**

1. related work & introduction

1)Group Convolution

ResnetXt와 DeepRoots로 group convolution이 처음 소개 된 개념,

Xception과 Mobilenet에서 Depthwise separable convolutions이 제안 및 활용이 된다. Depthwise separable convolutions는 representation 성능은 ‘아주 조금’ 떨어지지만 Computational Cost는 매우 줄일 수 있는 방법으로 알려진다. 따라서 경량 모델 중 최첨단의 결과를 얻을 수 있었음

2)Depthwise separable convolution

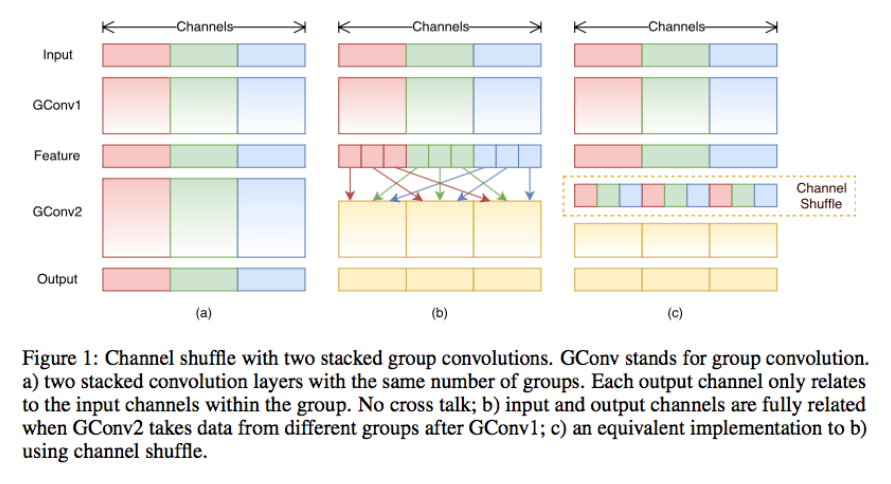
Input Tensor에 대해 채널별 n by n Convolution을 먼저 취하고 1x1 convolution으로 필터별 convolution을 하는 형태로 일반적인 convolution을 '분해’한 것

3)Channel Shuffle Operation

현재 CNN library인 cuda-convnet에선 “random sparse convolution” layer를 지원하지만 channel단위의 random shuffle에 대해서는 지원하지 않았음.

random shuffle에 대해선 거의 이용되지 않거나 다른 목적을위해서 사용되었지만 아직까지 채널 셔플 자체의 효과와 경량 모델에서 어떻게 사용될지에 대해서는 없음

2.approach(해당 논문에서의 의문점, 접근법): Channel Shuffle for Group Convolutions



Depthwise separable convolution연산에서 1x1 convolution은 여전히 Cost가 높다.

-> 이 문제를 해결하기 위해 Channel-Sparse 한 형태, 즉 모든 채널에 Weight를 연결하지 않는 형태로 네트워크 디자인을 설계하는 것이 어떨까 하는 의문

-> 특정 부분의 Channel에만 Convolutional Operation을 취하는 형태로 설계 한다면 Cost를 줄일 수 있을 것이다.(그림 a)

->(a)의 문제:특정한 영역의 data만 Input으로 얻게 됨

->(a)를 해결하기 위해 고안한 b 모델: input 영역을 Group별로 shuffle하자.( 핵심) 인풋 영역이 각 Group에 섞일 수 있도록 디자인한다면 tensor의 reshape operation으로 단순화가 가능하다.

->1x1 Convolution으로 생각해보면 연산양이 대략 Group 수에 비례하게 줄어들 것이다(예상)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**2019 03 28:ShuffleNet 논문 리딩**

2. ShuffleNet Unit: Channel Shuffle 연산을 사용한 Unit 제안

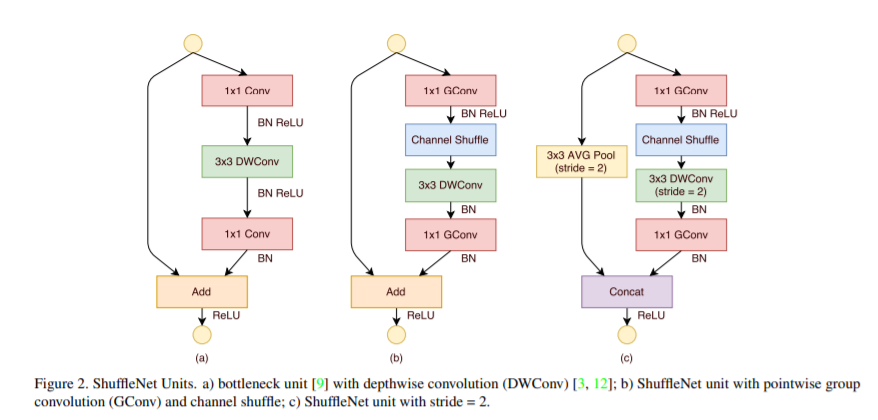


그림 (a):depthwise convolution에서의 bottleneck Unit

그림(b): Group Convolution을 처리 후 이를 Channel Shuffle을 한다. 이후 3x3 Depthwise Convolution을 하여 다시 1x1 Group Convolution

그림 (c): stride가 2인 ShuffleNet Unit

3. 실험 결과

* Group Size: 성능에 영향을 미치는 파라미터. 컴플릭세티를 급격하게 줄임에도 Cls Error가 어느정도 유지되고 있다.
* 채널 셔플: 셔플링이 있는 경우 에러율이 줄어들었음
* 다른 아키텍쳐와 비교(VGG-like,ResNet,Xception-like,ResNeXt): shuffleNet이 가장 우수한 성능을 보임.
* 모바일 넷과 비교:셔플넷이 조금 더 Complexity가 낮은 경우에도 성능이 더 좋음. **특히 작은 모델일 수록 모바일넷보다 성능이 좋다.**
* arm 기반의 임베디드: 실질적으로 유의미한 속도를 냄(알렉스 넷이 1156ms일때 shuffleNet은 222.2ms)

## 4. 결론

네트워크 파라미터 수를 줄이기 위해 mobileNet은 단순히 채널 수를 줄였지만, shuffleNet은 Group Convolution을 이용하여 채널 간 Sparse한 연결을 만들어 파라미터 수를 줄였다. 둘의 성능 비교 결과 shuffleNet의 결과가 mobileNet보다 나았음. 즉 채널 수를 최대한 유지하면서 커넥션 자체를 Sparse하게 만든 것이 더 성능이 좋다.

-비록 모바일넷이 일반적으로 구현도 쉽고 추가적인 하이퍼파라미터도 없지만, 모델의 사이즈가 매우 작아질 수록 셔플넷과의 좀 더 성능 차이가 점차 많이 날 것이다(셔플넷이 좋을 것이다)

5. 총 정리

* channel shuffle + Pointwise Group convolution
* 알렉스넷 대비 13배 수준의 성능 향상
* Mobilenet 보다 낫다고 주장
* Theoretical Speed up : 18x
* Actual Speed up (on arm-device) : 13x

채널 수를 최대한 유지하면서 커넥션 자체를 Sparse하게 만든 것이 더 성능이 좋다.Mobilenet에서의 Depthwise Separable Convolution은 동일하게 사용했지만 Channel Shuffle이라는 개념을 추가적으로 도입함. 또한 모델이 작으면 작을수록 Mobilenet에 비해 성능 차이가 월등히 좋아지며, 매우 빠른 속도로 arm 기반 프로세서에서 동작하는 것을 보여었음.