MorphNet

참고자료

- MorphNet 논문: https://arxiv.org/pdf/1711.06798.pdf
- Google AI Blog: https://ai.googleblog.com/2019/04/morphnet-towards-faster-and-smaller.html
- GitHub: https://github.com/google-research/morph-net
- MobileNets: https://arxiv.org/pdf/1704.04861.pdf

  : width multiplier 이해를 위하여
- Youtube: https://www.youtube.com/watch?v=UvTXhTvJ_wM

INTRODUCTION

- 기존 L1 Normalize term을 적용하여 non-zero weigh를 줄였었음.
- 학습속도 향상에는 도움 안됨. 분산된 0 들
- Node자체를 목표로 하는 대안들. Structured sparcity. E.g. Grouped Lasso
- 단점1. 내가 원하는 특정 자원(FLOPs, Latency든)에 대하여 모델 architecture를 바꾸는게 아님.
- 단점2. Trial and Error를 통한 Parameter 찾기, 비효율적임
- 따라서, 본 논문에서는 간단하고 특정자원에 맞추어 모델을 바꿀 수 있는 방법론을 제안한다.

MorphNet: Architecture Learning

Efficient & scalable architecture learning for everyone

- **Resource constraints guide customization**
- **Requires handful of training runs**
- **Trains on your data**
- **Start with your architecture**
- **Works with your code**

Idea: Continuous relaxation of combinatorial problem

Simple & effective tool: **weighted sparsifying regularization.**

https://www.youtube.com/watch?v=UvTXhTvJ_wM

Learning the Size of Each Layer

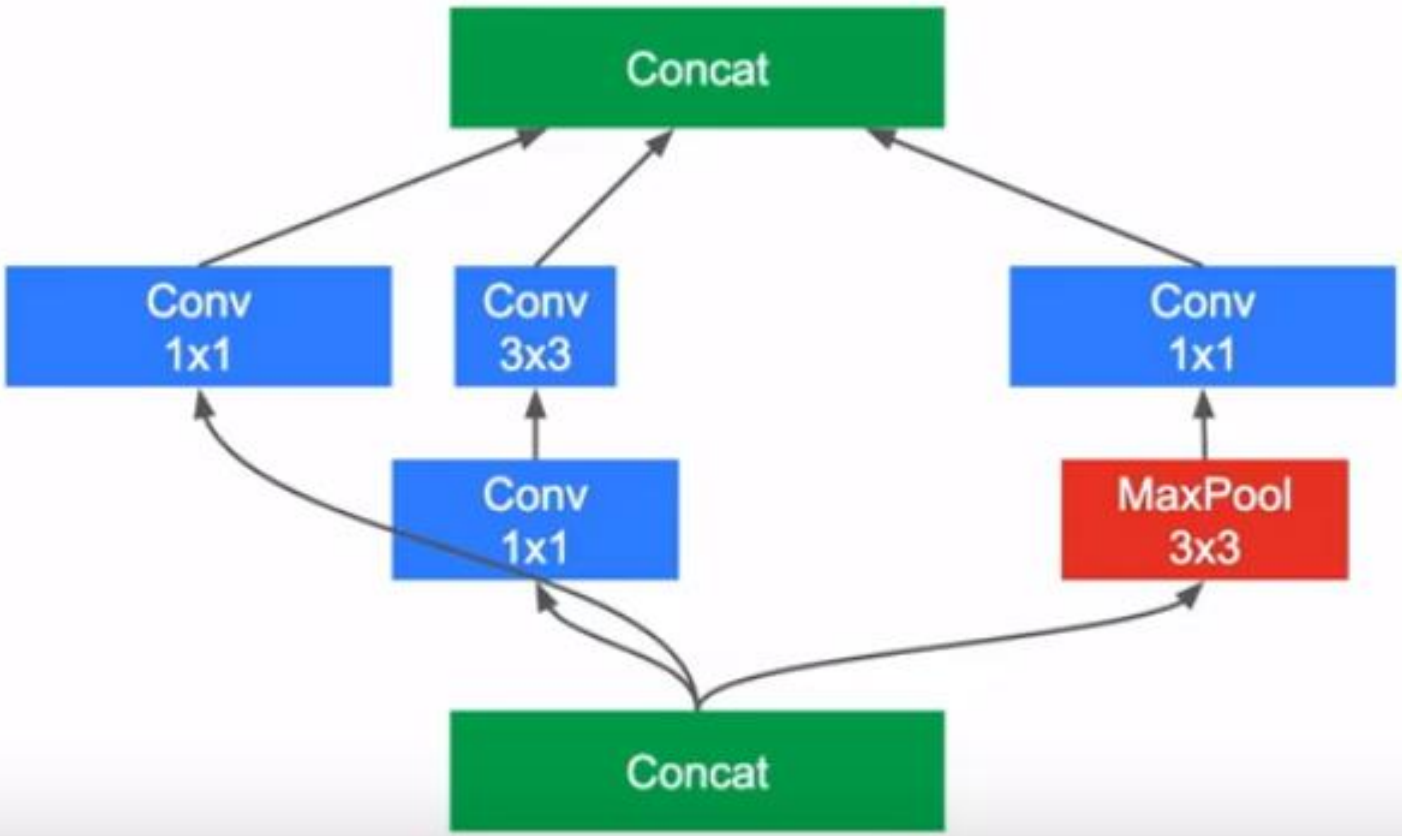
Architecture search

Topology

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
convolution	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	2		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	2	64	96	128	16	32	32
inception (3b)		28×28×480	2	128	128	192	32	96	64
max pool	3×3/2	14×14×480	0						
inception (4a)		14×14×512	2	192	96	208	16	48	64
inception (4b)		14×14×512	2	160	112	224	24	64	64
inception (4c)		14×14×512	2	128	128	256	24	64	64
inception (4d)		14×14×528	2	112	144	288	32	64	64
inception (4e)		14×14×832	2	256	160	320	32	128	128
max pool	3×3/2	7×7×832	0						
inception (5a)		7×7×832	2	256	160	320	32	128	128
inception (5b)		7×7×1024	2	384	192	384	48	[illegible]	[illegible]

Sizes

We focus on


Concat
Conv
1x1
Conv
3x3
Conv
1x1
Conv
1x1
MaxPool
3x3
Concat

Main Tool: Weighted **sparsifying** regularization.

Related work – L1 norm

- L1 norm 은 0으로 만들지만, L2 norm은 0에 가까울 뿐 0이 아님
- L0 norm은 0이 아닌 것의 개수, Converge되기 힘들기 때문에 L1씀

- **L_0 Norm:** $\|\vec{x}\|_0$ **number of nonzero elements**
- **L_1 Norm:** $\|\vec{x}\|_1 = \sum_{n=1}^{N} |x_n|$
- **L_2 Norm:** $\|\vec{x}\|_2 = \left(\sum_{n=1}^{N} |x_n|^2\right)^{1/2}$
- **L_∞ Norm:** $\|\vec{x}\|_\infty = \max\{|x_1|, \ldots, |x_N|\}$
- **L_p Norm:** $\|\vec{x}\|_p = \left(\sum_{n=1}^{N} |x_n|^p\right)^{1/p}$

$$L_\infty = \lim_{p \to +\infty} L_p$$

$$\|\vec{x}\|_\infty = \lim_{p \to +\infty} \left(\sum_{n=1}^{N} |x_n|^p\right)^{1/p}$$

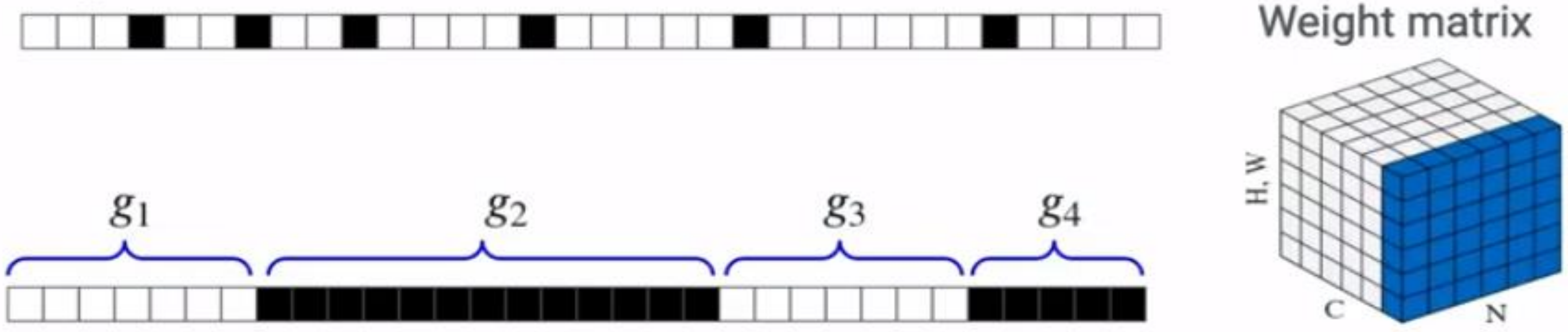
Related work – Group Lasso

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \boxed{\begin{bmatrix} a & b \\ 0 & d \\ e & 0 \end{bmatrix}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \boxed{\begin{bmatrix} a & b \\ c & d \\ 0 & 0 \end{bmatrix}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

L1 norm **Structured Sparsity**

(Group) LASSO: Sparsity in Optimization

$$\min_w \ell(X, Y, w) + \lambda |w|_1$$



$$\min_w \ell(X, Y, w) + \lambda \sum \sqrt{w_{g_1}^2 + \dots w_{g_k}^2}$$

Related work – width multiplier

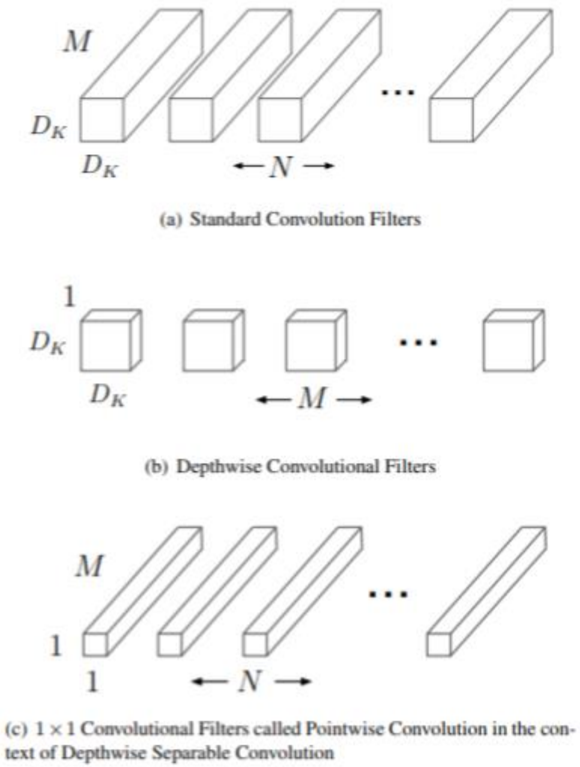


Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

- MobileNet
- Standard CNN: $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$
- Depthwise and pointwise CNN:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

- Width multiplier (α):

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

METHOD – MorphNet Algorithm

Algorithm 1 The MorphNet Algorithm

1: Train the network to find
$\theta^* = \underset{\theta}{\operatorname{argmin}}\{\mathcal{L}(\theta) + \lambda\mathcal{G}(\theta)\}$, for suitable λ.
2: Find the new widths $O'_{1:M}$ induced by θ^*.
3: Find the largests ω such that $\mathcal{F}(\omega \cdot O'_{1:M}) \leq \zeta$.
4: Repeat from Step 1 for as many times as desired, setting
$O^{\circ}_{1:M} = \omega \cdot O'_{1:M}$.
5: **return** $\omega \cdot O'_{1:M}$.

- **STEP 1-2 : Shrink**
  - $\mathcal{G}(\theta)$: regulation term
  : target source cost & L1 norm
  - $\mathcal{L}(\theta)$: Loss
- **STEP 3: Expansion**
  - $O_{1:M}$: Output layer 1부터 m까지 (w 구해지면 layer 1부터 m까지 동일하게 넓힘)
  - $Constraint(O_{1:M})$ 이 정해진 숫자 (source) ς를 넘지 않을 때의 가장 큰 width multiple 해 줌.

1. Constraints $\mathrm{F}(\omega idth \cdot O'_{i:m})$

- FLOPs도 model size도 둘다 matrix multiplication에 따라서 증가

$$\mathcal{F}(\text{layer } L) = C(w_L, x_L, y_L, z_L, f_L, g_L) \cdot I_L O_L.$$

  - FLOPs: $C(w, x, y, z, f, g) = 2yzfg,$ (Eq 4)
  - Model Size: $C(w, x, y, z, f, g) = fg.$ (Eq 5)

$$\mathcal{F}(\text{layer } L) = C \sum_{i=0}^{I_L - 1} A_{L,i} \sum_{j=0}^{O_L - 1} B_{L,j},$$

where $A_{L,i}$ ($B_{L,j}$) is an indicator function which equals one if the i-th input (j-th output) of layer L is *alive* – not zeroed (Eq. 6)

$$\mathcal{F}(O_{1:M}) = \sum_{L=1}^{M+1} \mathcal{F}(\text{layer } L).$$

(Eq. 7)

2. Regularization $\mathcal{G}()$

- Constraint는 수식(7)에 따라 정의, 최적화 문제는 Loss에 penalty부가 $\min_{\theta} \mathcal{L}(\theta) + \lambda \mathcal{F}(O_{1:M}), \quad (8)$
- 수식 (6)은 discontinuous하므로 continuous proxy norm (L1 norm)으로 변형
- Batch Norm을 통해 얻어지는 scale factor γ활용, gamma 가 0 이되면 filter 제거됨.
- 수식 (9)를 통해 수식 (6)을 근사

$$\mathcal{G}(\theta, \text{layer } L) = C \sum_{i=0}^{I_L-1} |\gamma_{L-1,i}| \sum_{j=0}^{O_L-1} B_{L,j} + \; C \sum_{i=0}^{I_L-1} A_{L,i} \sum_{j=0}^{O_L-1} |\gamma_{L,j}|, \quad (9)$$

- 전체 네트워크에 대한 수식 $\mathcal{G}(\theta) = \sum_{L=1}^{M+1} \mathcal{G}(\theta, \text{layer } L). \quad (10)$

2. Regularization

- L1이기 때문에 비록 수식 (9)도 discontinuity로 부터 자유롭진 않지만, 실용적인 상황에서는 문제가 없음. 일반적인 미니 배치 기반 옵티마이저는 $\mathcal{G}()$ 의 불연속성을 해결함.

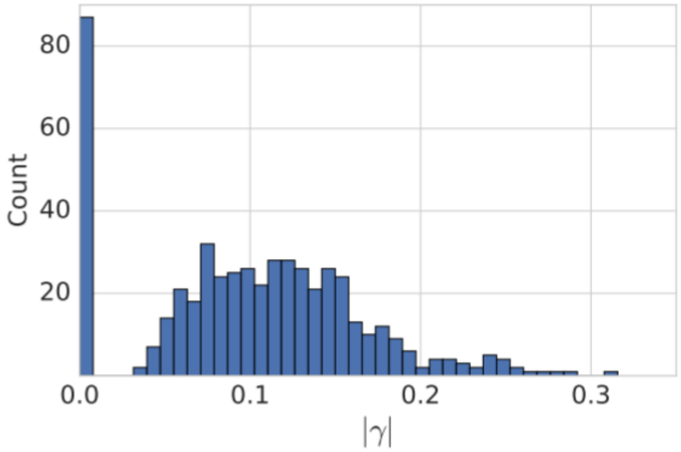


Figure 2. A histogram of γ for one of the ResNet101 bottleneck layers when trained with a FLOP regularizer. Some of the $|\gamma|$'s are zeroed out, and are separated by a clear gap from the nonzero $|\gamma|$'s.

- Fig2 – γ 활용하여 제로 vs non-zero로 확연히 나뉘어짐을 확인

3. Network Topology

- Resnet처럼 레이어들이 건너 뛰어 연결된 경우
- Input of Layer 3 = Out(Layer 1) + Out (Layer 2)
- 연결을 가지는 layer들을 Group Lasso로 묶어서 처리한다.
- Layer 1의 j번째 Output 과 Layer 2의 j번째 Output은 묶여서 처리된다. 0이면 같이 0

L_∞ norm - the maximum of the $|\gamma|$'s in the group.

RESULT

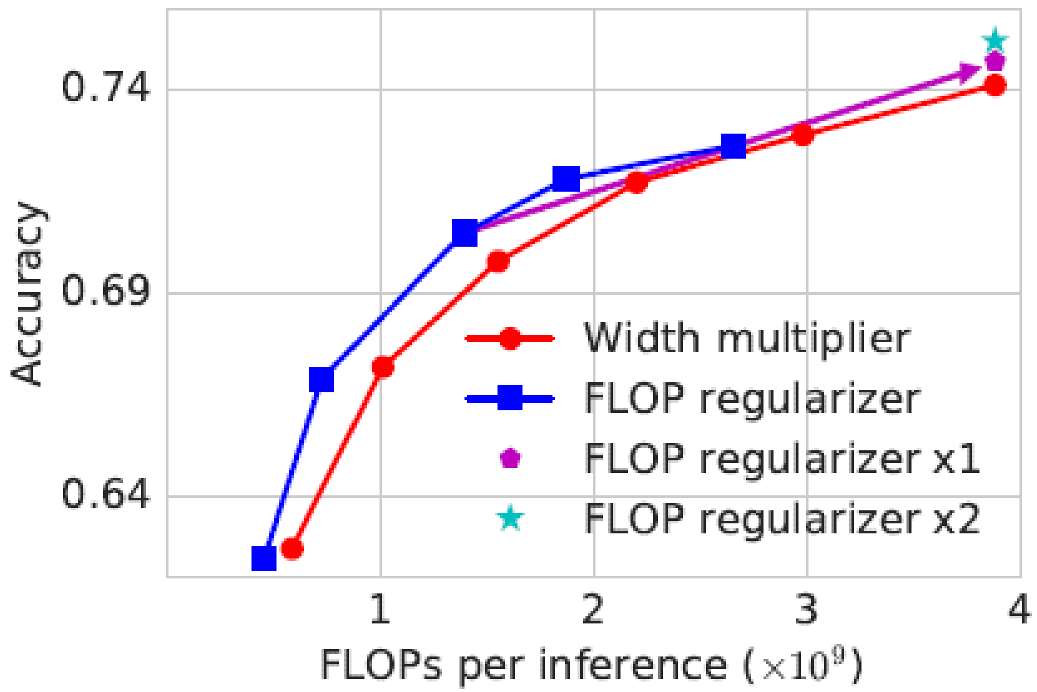


Figure 4. ImageNet evaluation accuracy for various downsized versions of Inception V2 using both a naïve width multiplier (red circles) and a sparsifying FLOP regularizer (blue squares). We also show the result of re-expanding one of the networks induced by the FLOP regularizer to match the FLOP cost of the original network (pentagon point). A further increase in accuracy is achieved by performing the sparsifying and expanding process a second time (star point).

Network	Baseline	MorphNet	Relative Gain
Inception V2	74.1	75.2	+1.5%
MobileNet 50%	57.1	58.1	+1.78%
MobileNet 25%	44.8	45.9	+2.58%
ResNet101	0.477	0.487	+2.1%
AudioResNet	0.182	0.186	+2.18%

Table 2. The result of applying MorphNet to a variety of datasets and model architectures while maintaining FLOP cost.

RESULT

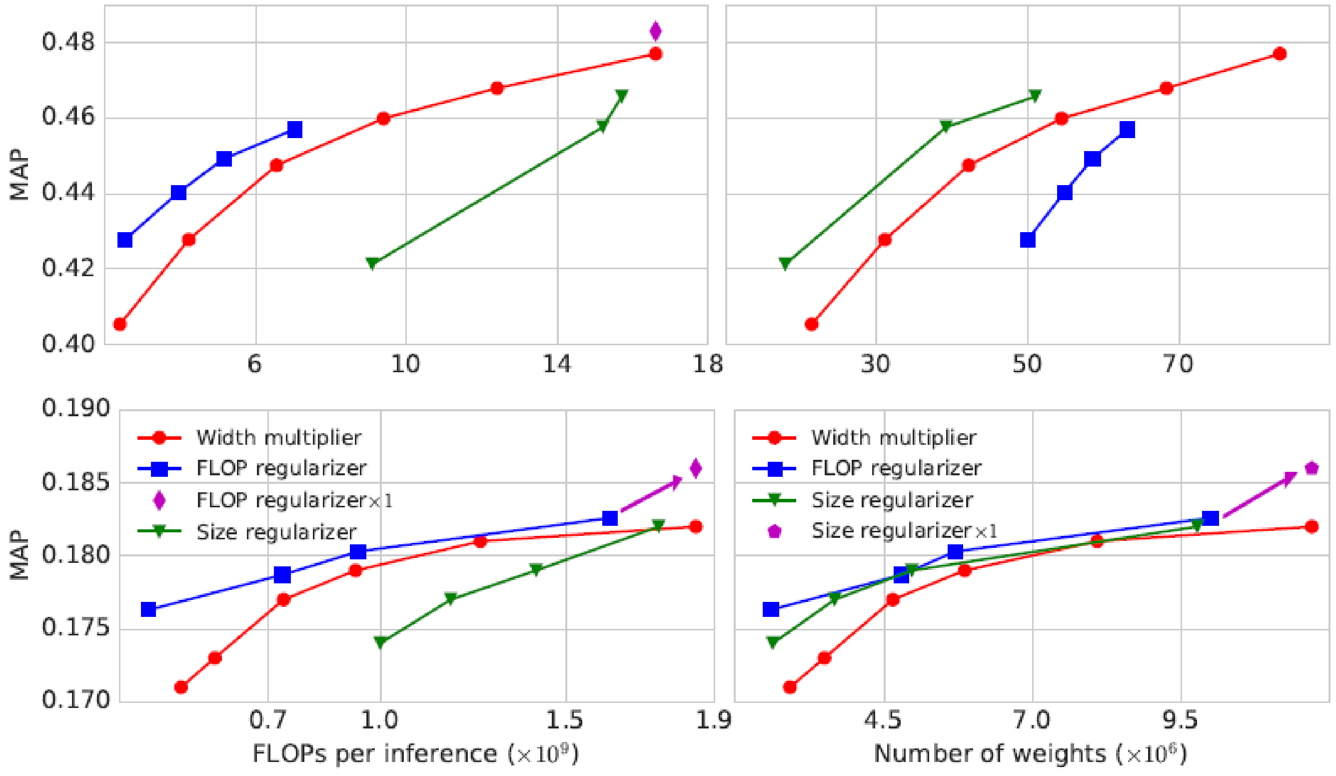


Figure 5. MAP vs. FLOPs (left) and MAP vs. model-size (right) curves on JFT (top) and AudioSet (bottom). The magenta points in the AudioSet figures represent models expanded from a FLOP (diamond) or size (pentagon) regularizes.

- Target한 resource에 따라서 다른 결과

RESULT

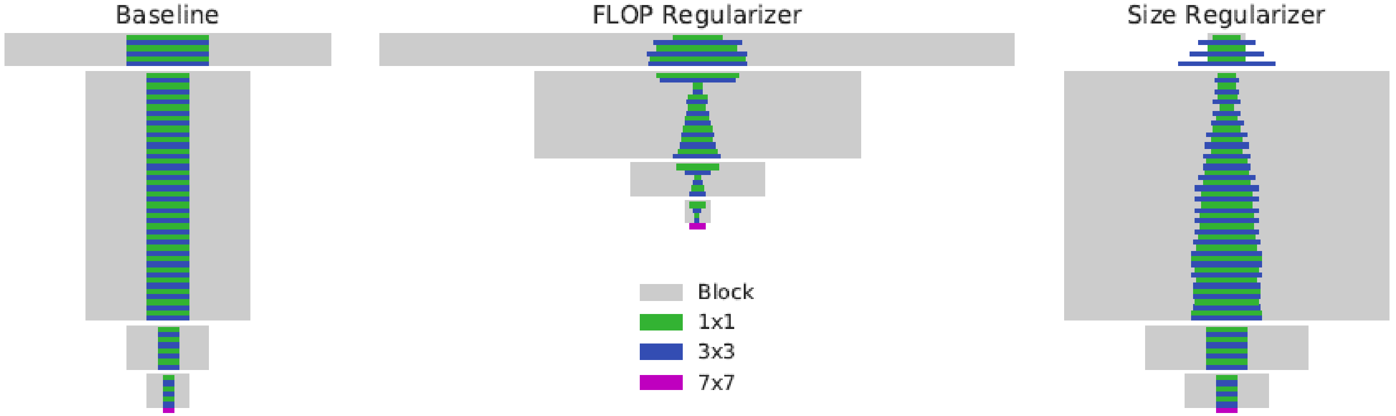


Figure 1. ResNet101 based models with similar performance (around 0.426 MAP on JFT, see Section 5). A structure obtained by shrinking ResNet101 uniformly by a $\omega = 0.5$ factor (left), and structures learned by MorphNet when targeting FLOPs (center) or model size (*i.e.*, number of parameters; right). Rectangle width is proportional to the number of channels in the layer and residual blocks are denoted in gray. 7×7, 3×3, and 1×1 convolutions are in purple, blue and green respectively. The purple bar at the bottom of each model is thus the input layer. Learned structures are markedly different from the human-designed model and from each other. The FLOP regularizer primarily prunes the early, compute-heavy layers. It notably learns to *remove whole layers* to further reduce computational burden. By contrast, the model size regularizer focuses on removal of 3×3 convolutions at the top layers as those are the most parameter-heavy.

- FLOPs는 low layer들을 줄이는 경향, Model Size는 Upper layer들을 줄이는 경향
- 둘 다 정확도는 비슷하게 오름 (Base MAP: 0.405, Flops 0.428, Size: 0.421)

19'추가 연구 – Latency

Latency Roofline Model

Each op needs to read inputs, perform calculations, and write outputs.

Evaluation time of an op depends on the **compute** and **memory** costs.

Compute time = FLOPs / **compute_rate**.

Memory time = tensor_size / **memory_bandwidth**.

Device Specific

Latency = max(**Compute time, Memory time**)

https://www.youtube.com/watch?v=UvTXhTvJ_wM&t=1104s

19’추가 연구 – Latency

Example Latency Costs

Different platforms have different cost profile

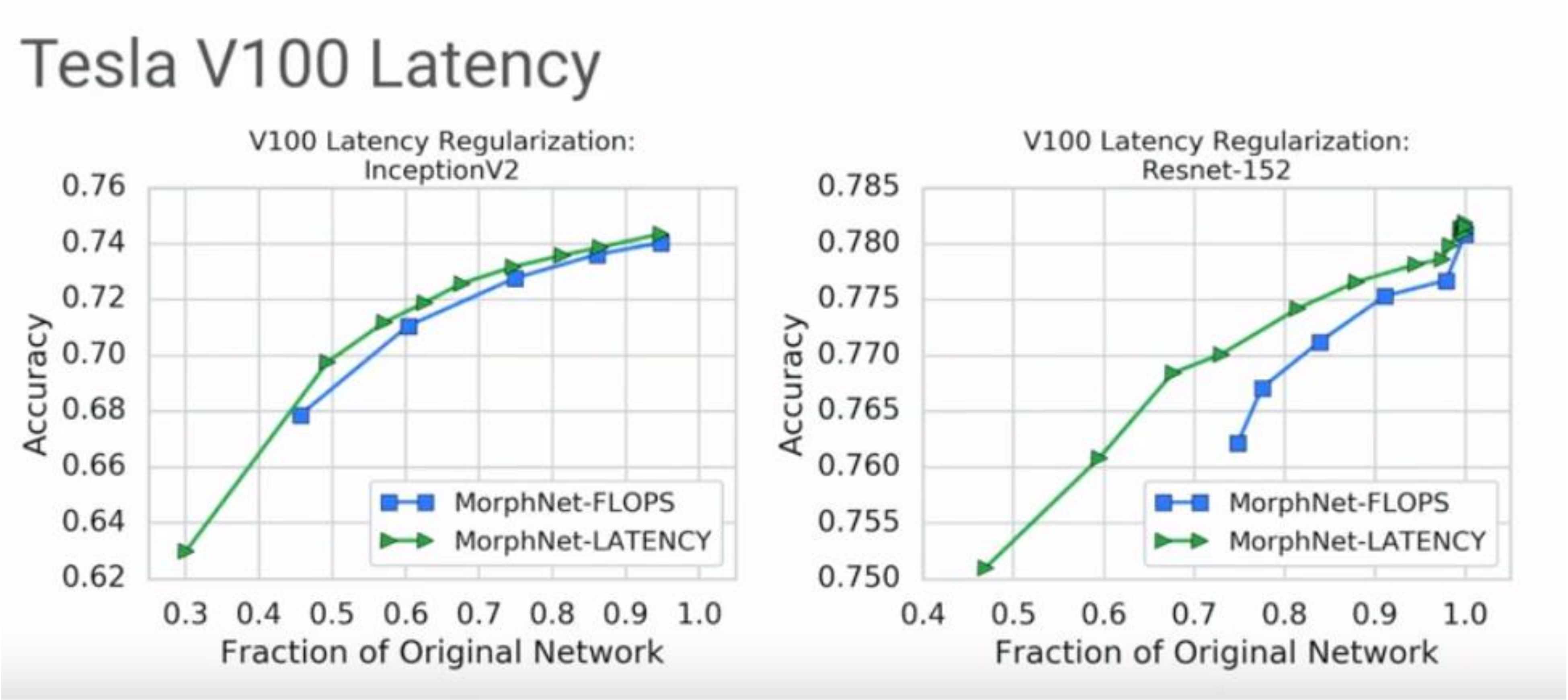
Platform	Peak Compute	Memory Bandwidth
P100	9300 GFLOPs/s	732 GB/s
V100	125000 GFLOPs/s	900 GB/s

Leads to different relative cost

Inception V2 Layer Name	P100 Latency	V100 Latency	Ratio
Conv2d_2c_3x3	74584	5549	7%
Mixed_3c/Branch_2/Conv2d_0a_1x1	2762	1187	43%
Mixed_5c/Branch_3/Conv2d_0b_1x1	1381	833	60%

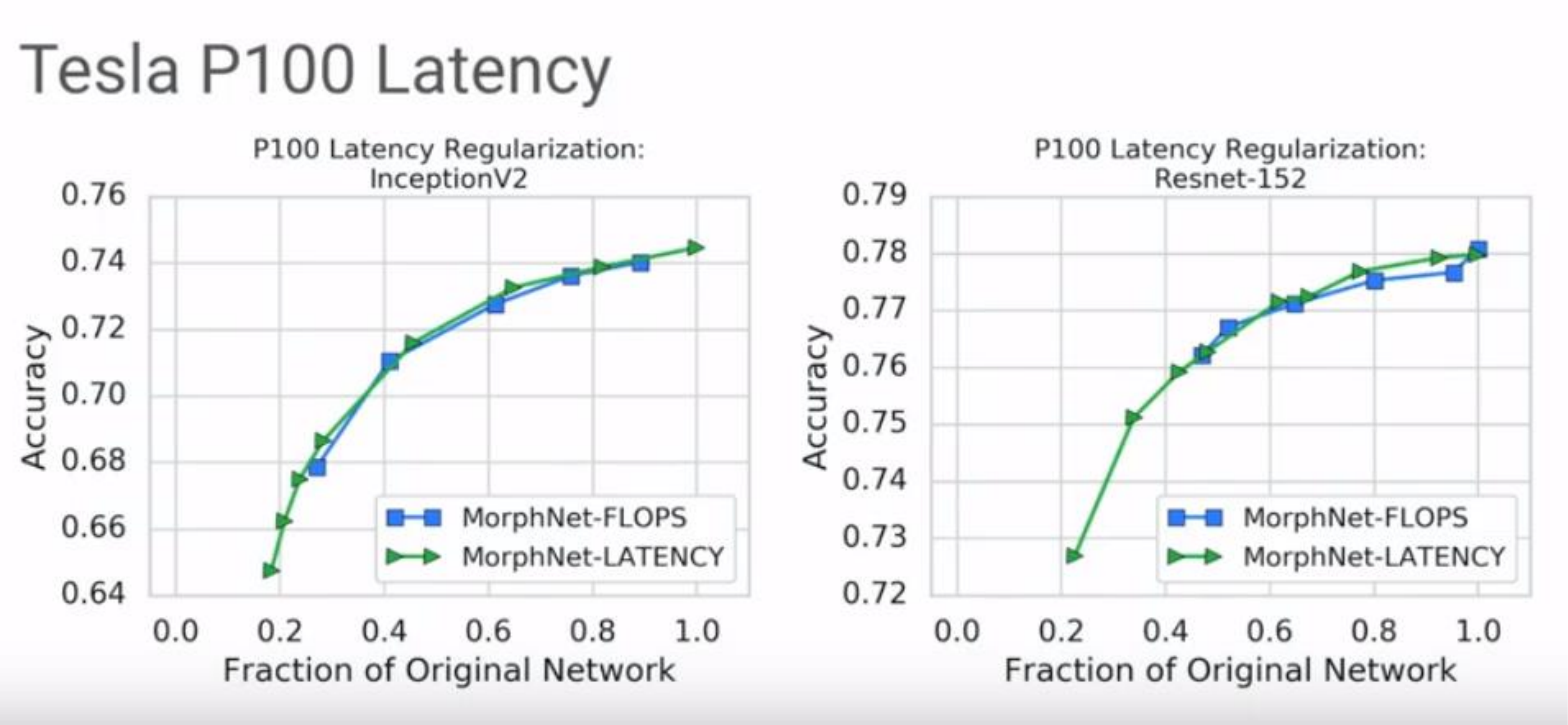
https://www.youtube.com/watch?v=UvTXhTvJ_wM&t=1104s

19'추가 연구 – Latency



https://www.youtube.com/watch?v=UvTXhTvJ_wM&t=1104s

19’추가 연구 – Latency



https://www.youtube.com/watch?v=UvTXhTvJ_wM&t=1104s

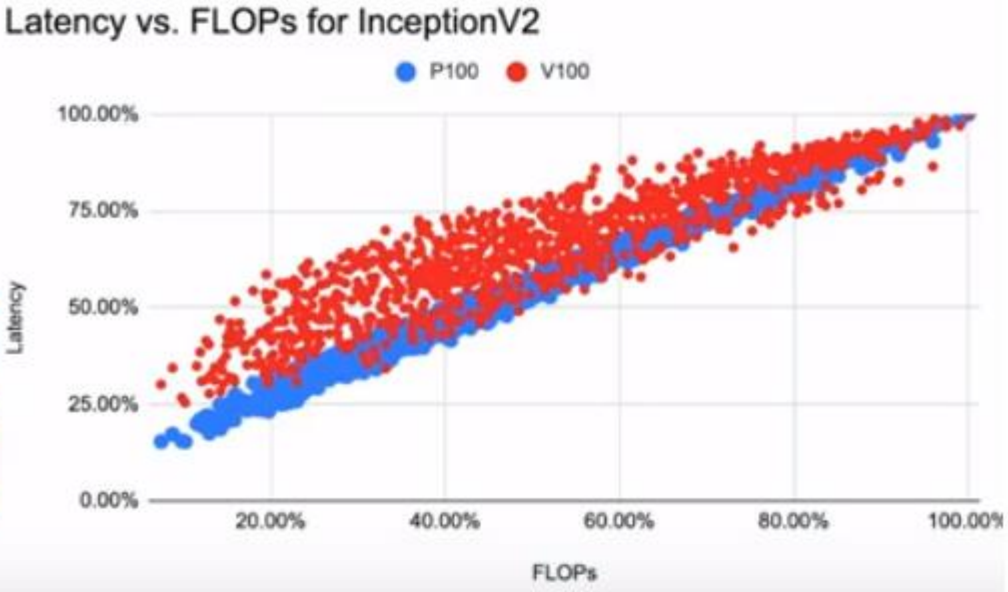
19’추가 연구 – Latency

When Do FLOPs and Latency Differ?

- Create 5000 *sub-Inception V2* models with a random number of filters.
- Compare FLOPs, V100 and P100 Latency.

P100 - is compute bound, tracks FLOPs “too” closely

V100 - gap between FLOPs and Latency is looser



https://www.youtube.com/watch?v=UvTXhTvJ_wM&t=1104s

Open Source

- *지금은 Error..*

Quick User Guide

```
from morph_net.network_regularizers import flop_regularizer
from morph_net.tools import structure_exporter

logits = build_model()

network_regularizer = flop_regularizer.GammaFlopsRegularizer(
    [logits.op], gamma_threshold=1e-3)
regularization_strength = 1e-10
regularizer_loss = (network_regularizer.get_regularization_term() * regularization_strength)

model_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels, logits)

optimizer = tf.train.MomentumOptimizer(learning_rate=0.01, momentum=0.9)

train_op = optimizer.minimize(model_loss + regularizer_loss)
```

Exact same API works for different costs and settings:
GroupLassoFlops, GammaFlops, GammaModelSize, GammaLatency

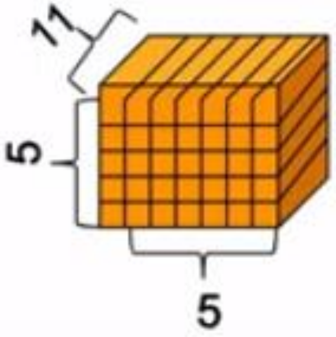
FLOPs

- 컴퓨터의 연산 속도를 나타내는 단위로, 1초당 부동 소수점 연산 명령을 몇 번 실행할 수 있는지를 말한다.

부호	지수부 8비트								가수부 23비트										
														…….					

$$
\begin{aligned}
11001.1110_{(2)} &= 1.10011110 \times 2^{4} \\
0.00001111_{(2)} &= 1.11100000 \times 2^{-5} \\
11.0001000_{(2)} &= 1.10001000 \times 2^{1}
\end{aligned}
$$

What is the Cost of a Filter?



Model-size coefficient: filter_size * num_inputs

(5 * 5) * 11

FLOP coefficient: filter_size * num_inputs * output_size

(5 * 5) * 11 * (3 * 3)