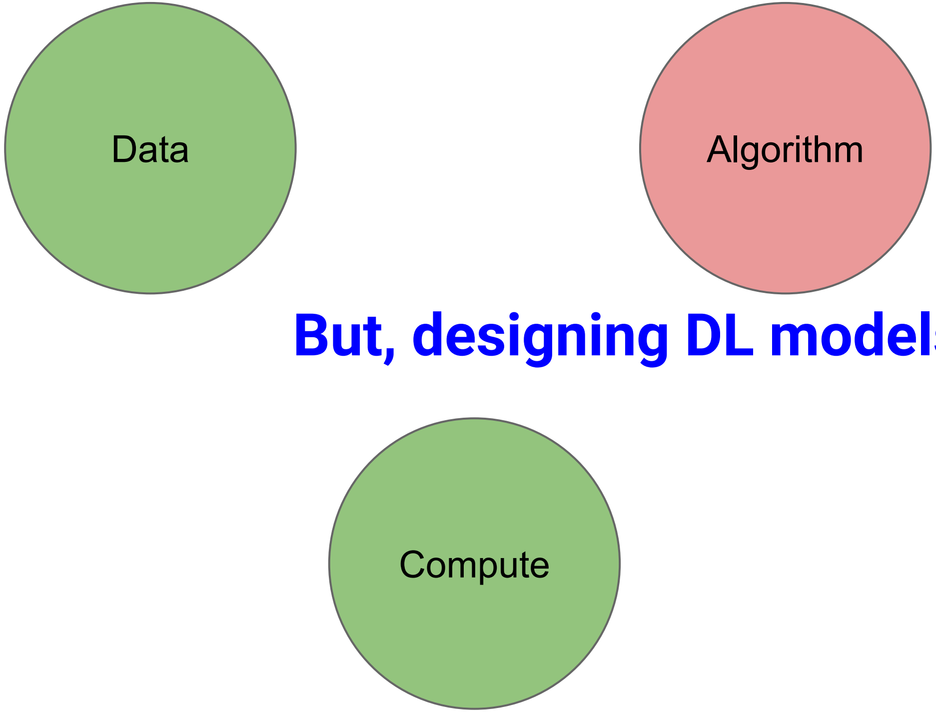
AutoML for Efficient Vision Learning

Mingxing Tan

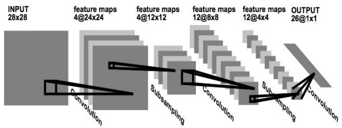
Google Brain, joint work with many Google colleagues
02/26/2021

Including results by many other people at Google and elsewhere.

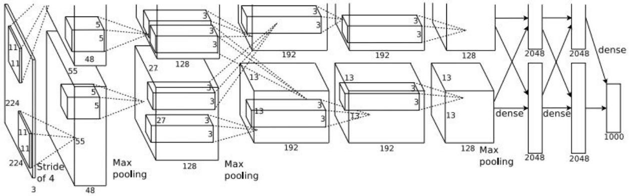
Deep learning is taking off …



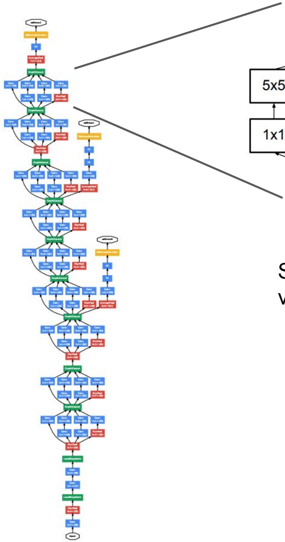
But, designing DL models is hard...

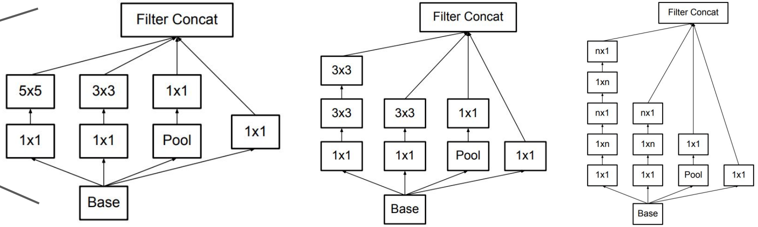


LeCun and Bengio 1995. Convolutional networks for images, speech, and time-series.

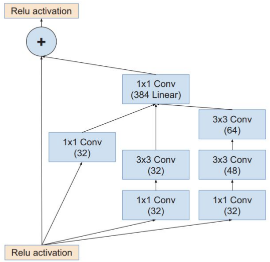


Krizhevsky et al 2012. Imagenet classification with deep convolutional neural networks.





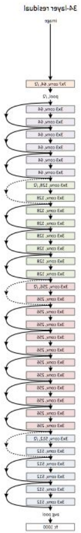
Szegedy et al 2016. Rethinking the inception architecture for computer vision

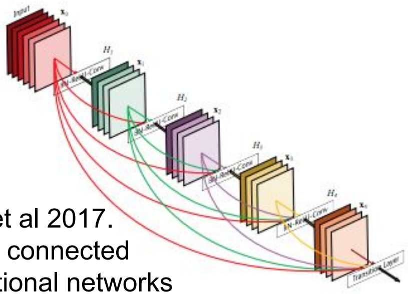


Szegedy et al 2017 Inception-v4, Inception-Resnet, and the Impact of Residual Connections on Learning.

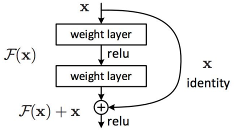
Szegedy et al. 2015. Going deeper with convolutions.

+others

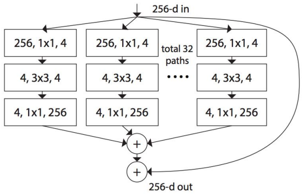




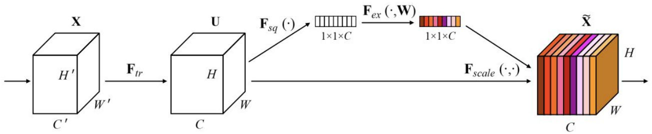
Huang et al 2017. Densely connected convolutional networks



He et al. 2016. Deep residual learning for image recognition.



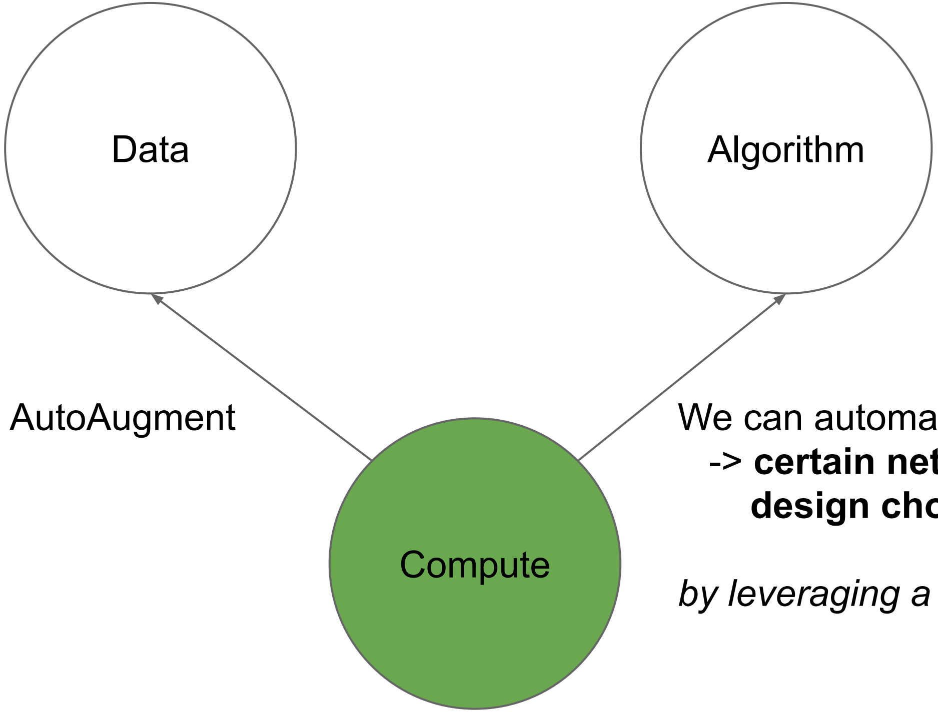
Xie et al 2017. Aggregated residual transformations for deep neural networks



Hu et al. 2018. Squeeze-and-Excitation Networks

Can we automate deep learning? AutoML?

NAS: Neural Architecture Search



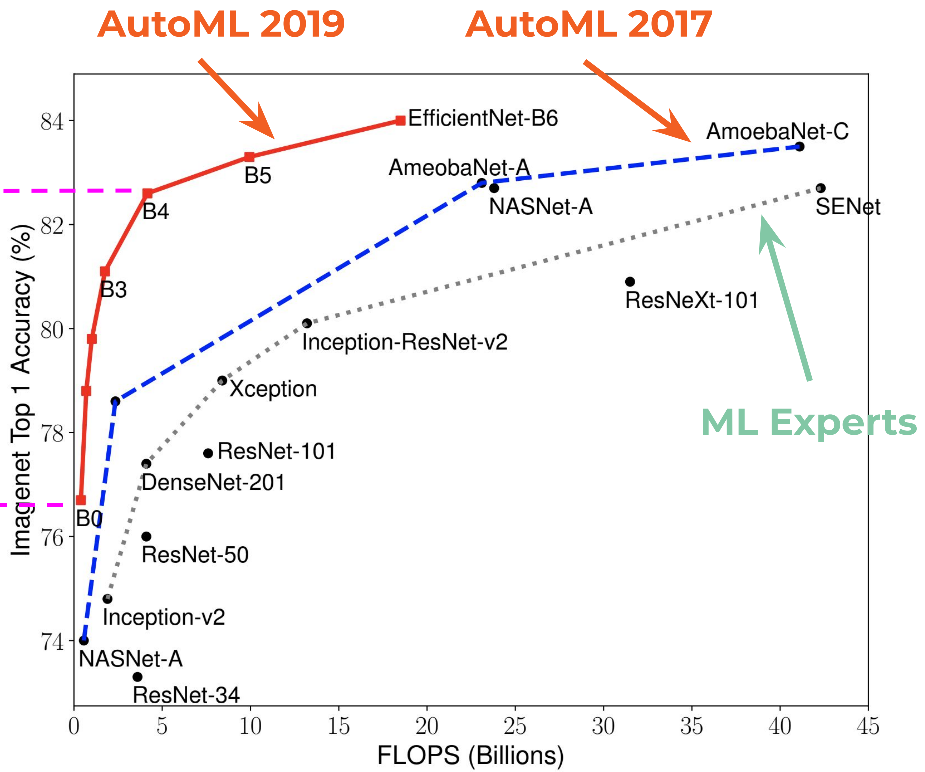
We can automate:
-> **certain network architecture design choices**

by leveraging a lot of compute

Image Recognition

Step2: Scale it up

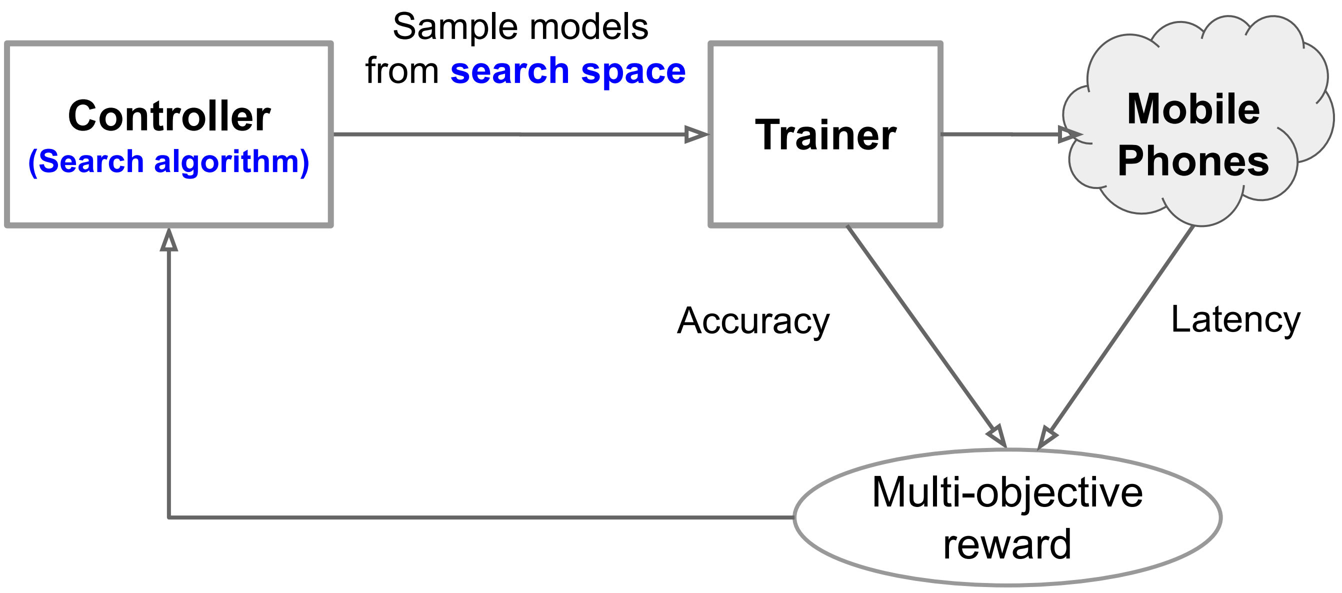
Step1: find a good model with NAS



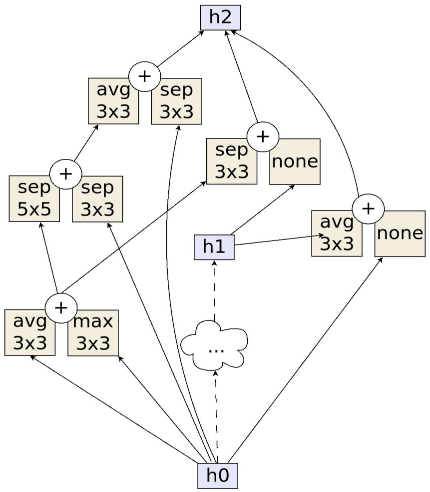
MNAS: Platform-Aware Neural Architecture Search

Step1: find a good compact model with NAS

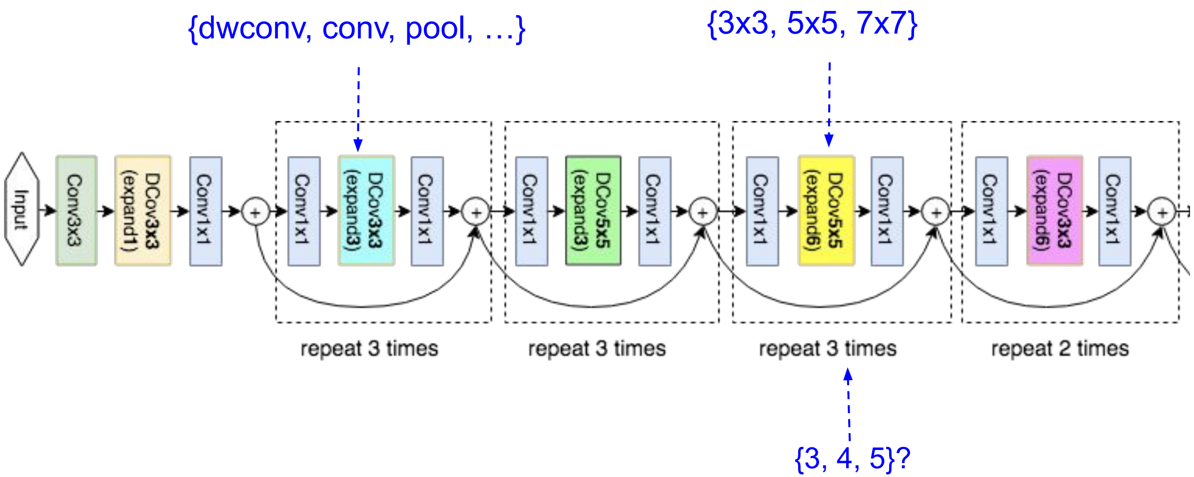
Mingxing Tan, et. al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. CVPR 2019. https://arxiv.org/abs/1807.11626
Mingxing Tan, Quoc V. Le MixNet: Mixed Depthwise Convolutional Kernels. BMVC 2019. https://arxiv.org/abs/1907.09595
Andrew Howard, et al. MobileNetV3: Searching for MobileNetV3. ICCV 2019. https://arxiv.org/abs/1905.02244


Sample models
from search space
Controller
(Search algorithm)
Trainer
Mobile
Phones
Accuracy
Latency
Multi-objective
reward

Search Space: A cell or a network?

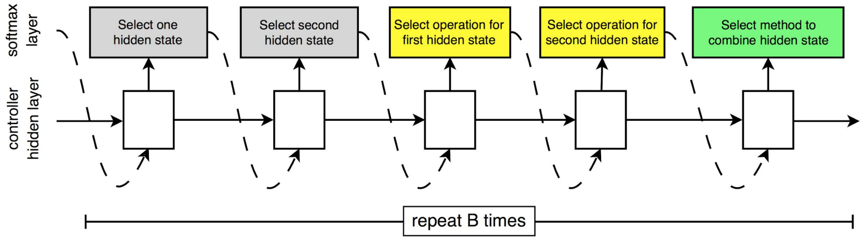


Search for a Cell
And repeatedly stack the cell



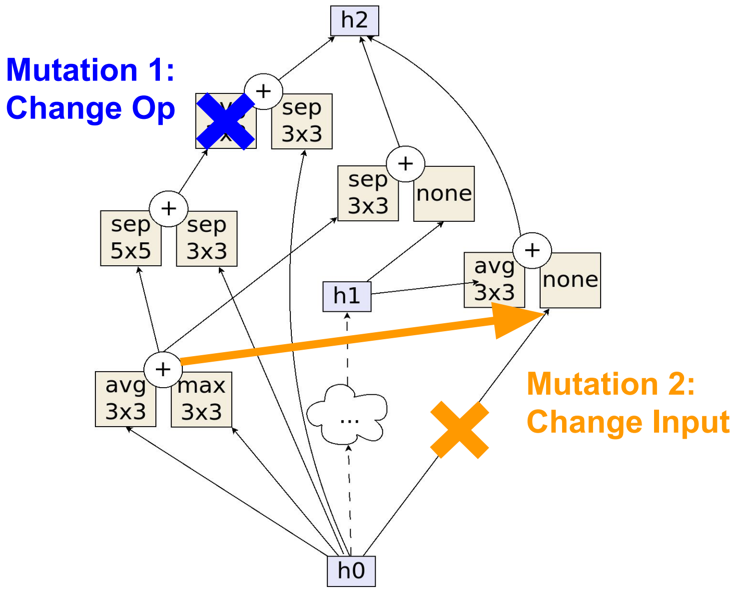
Search for a network

Search Algorithm: RL or Evolution



NAS with Reinforcement Learning [ICLR'17]

Flexible & Expensive



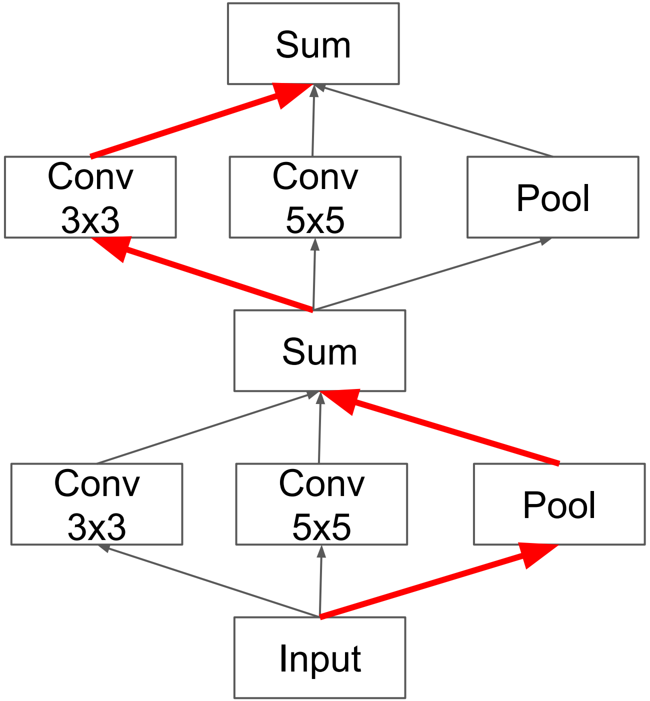
NAS with Evolution [AAAI'19]

Flexible & Expensive

Zoph, et al. "Neural architecture search with reinforcement learning", ICLR'17
Real et al, "Regularized Evolution for Image Classifier Architecture Search", AAAI'19

10,000 GPU hours?

The Rescue: Weight Sharing



Key idea:

1. One path inside a big model is a child model
2. Controller selects a path inside a big model and train for a few steps
3. Controller selects another path inside a big model and train for a few steps, reusing the weights produced by the previous step
4. Etc.

Results: **Can save 100->1000x compute**

Related work: DARTS, SMASH, One-shot, ProxylessNAS, FBNet, ...

Pham, Guan, Zoph, Le, and Dean, ICML 2018. Efficient Neural Architecture Search via Parameter Sharing. arxiv.org/abs/1802.03268

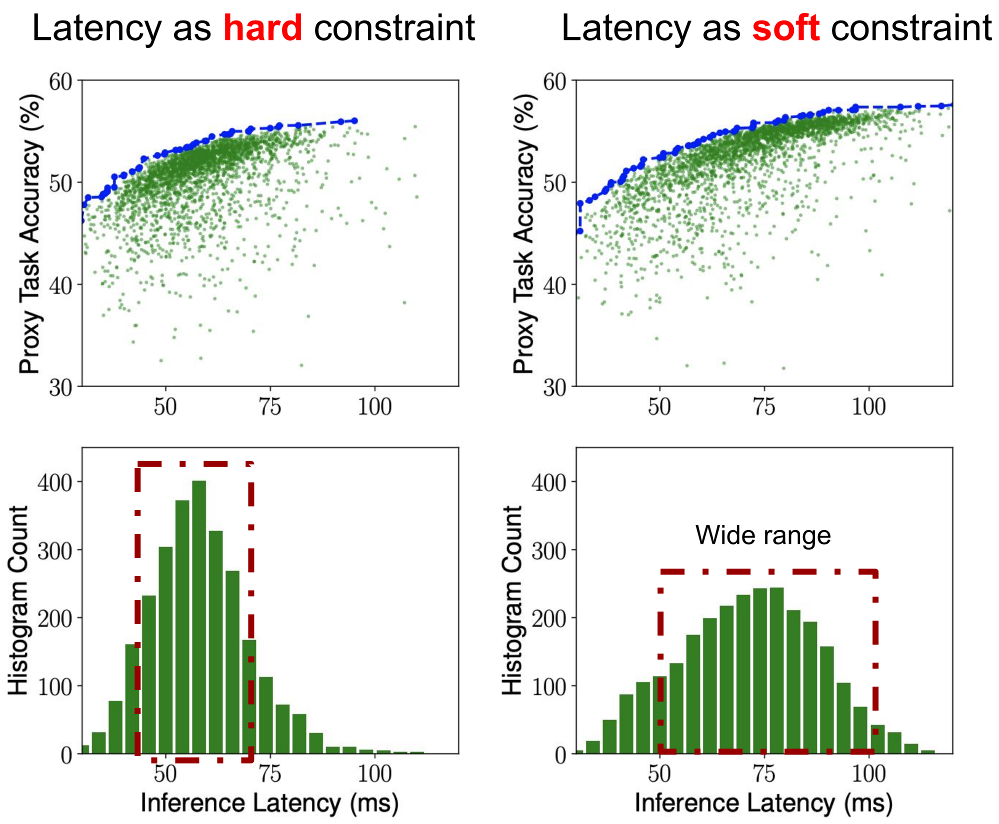
Multi-Object Reward

- Incorporate both **latency and accuracy** into reward function
  - *Weighted product* to approximate Pareto-Optimal solutions

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^w \tag{2}$$

where w is the weight factor defined as:

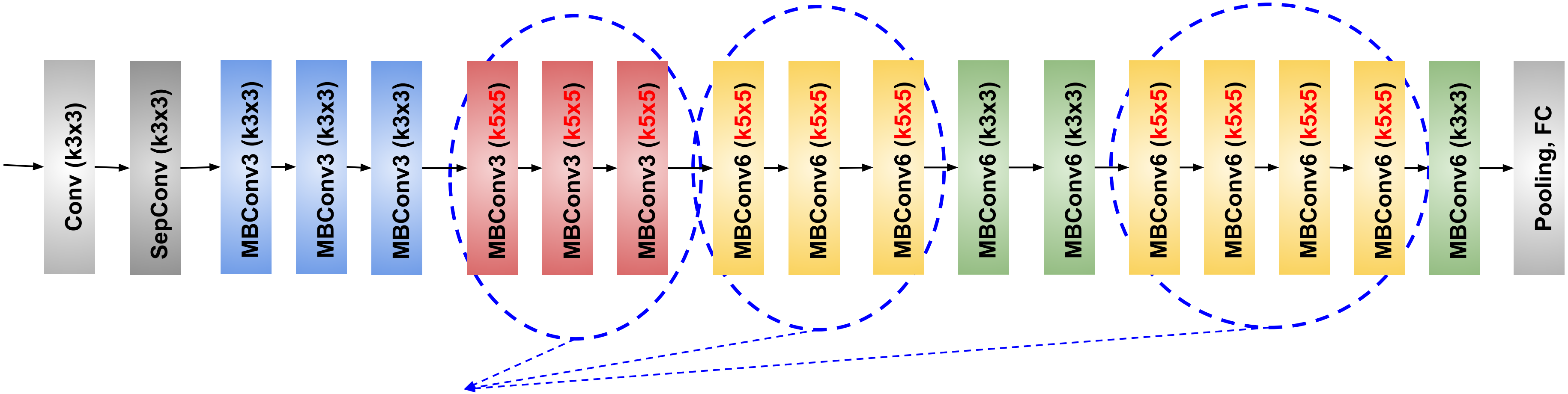
$$w = \begin{cases} \alpha, & \text{if } LAT(m) \leq T \\ \beta, & \text{otherwise} \end{cases} \tag{3}$$



(a) $\alpha = 0, \beta = -1$

(b) $\alpha = \beta = -0.07$

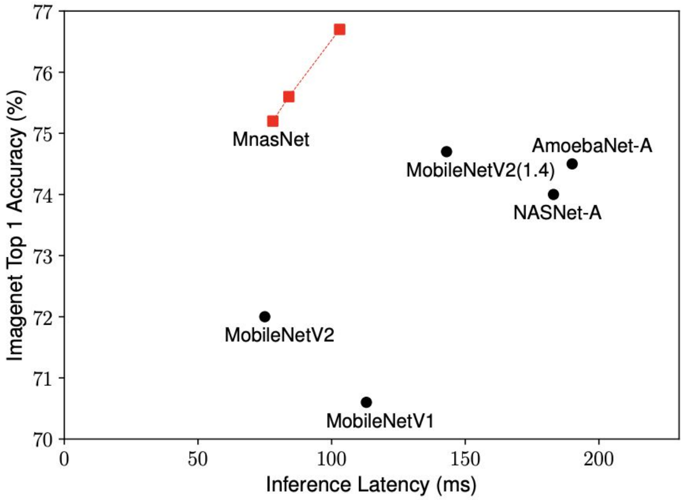
Example Model

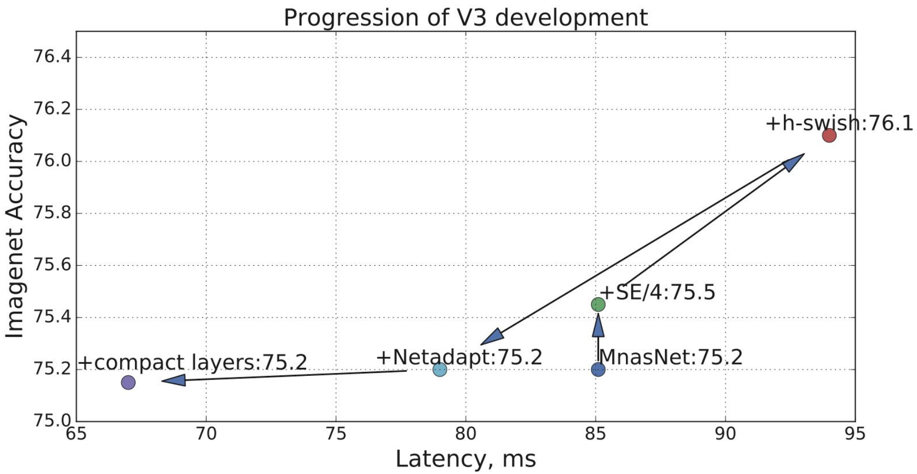


Obervation1:
5x5 depconv is very common

Observation 2:
Diverse layers throughout the network

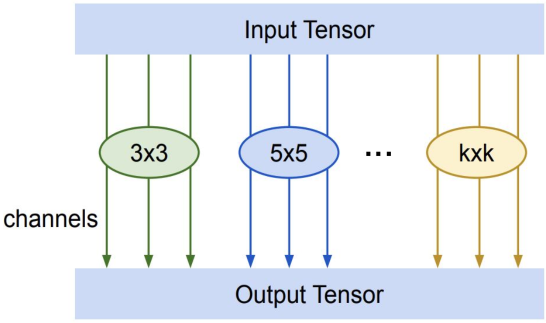
ImageNet Results





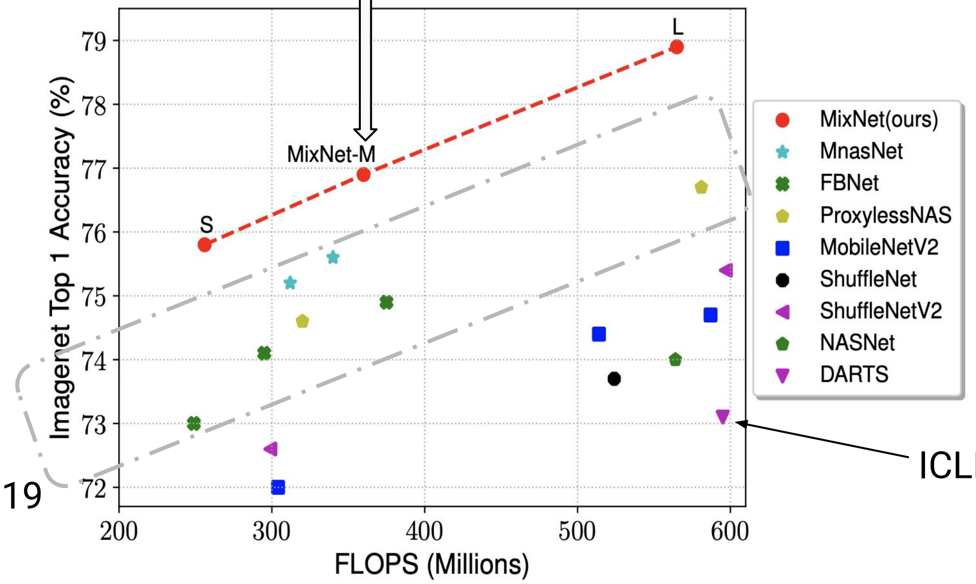
Mingxing Tan, et. al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. CVPR 2019. https://arxiv.org/abs/1807.11626
Andrew Howard, et al. MobileNetV3: Searching for MobileNetV3. ICCV 2019. https://arxiv.org/abs/1905.02244

Use NAS as a tool: MNAS -> MixMet



MDConv

ResNet-152: 77% @ 11B FLOPS (30x more)



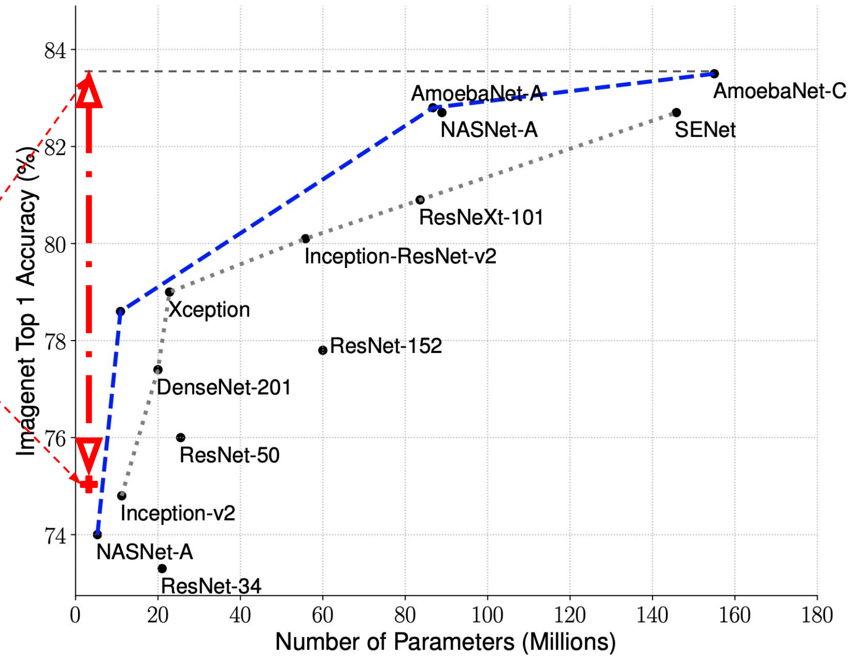
AutoML'19

ICLR'19

Mingxing Tan and Quoc V. Le. MixConv: Mixed Depthwise Convolutional Kernels. BMVC 2019. https://arxiv.org/abs/1907.09595

So far so good, but wait ...

Huge accuracy gap to SoTA



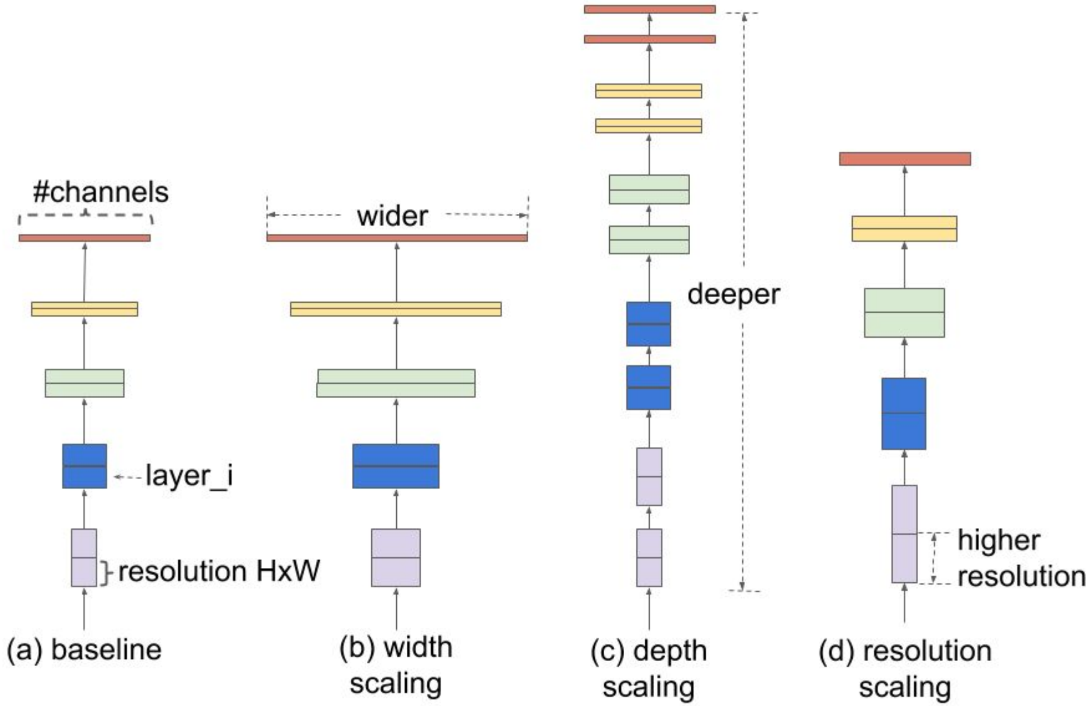

NOT SATISFIED
memegenerator.net

EfficientNet/Det: Scaling Up ConvNets

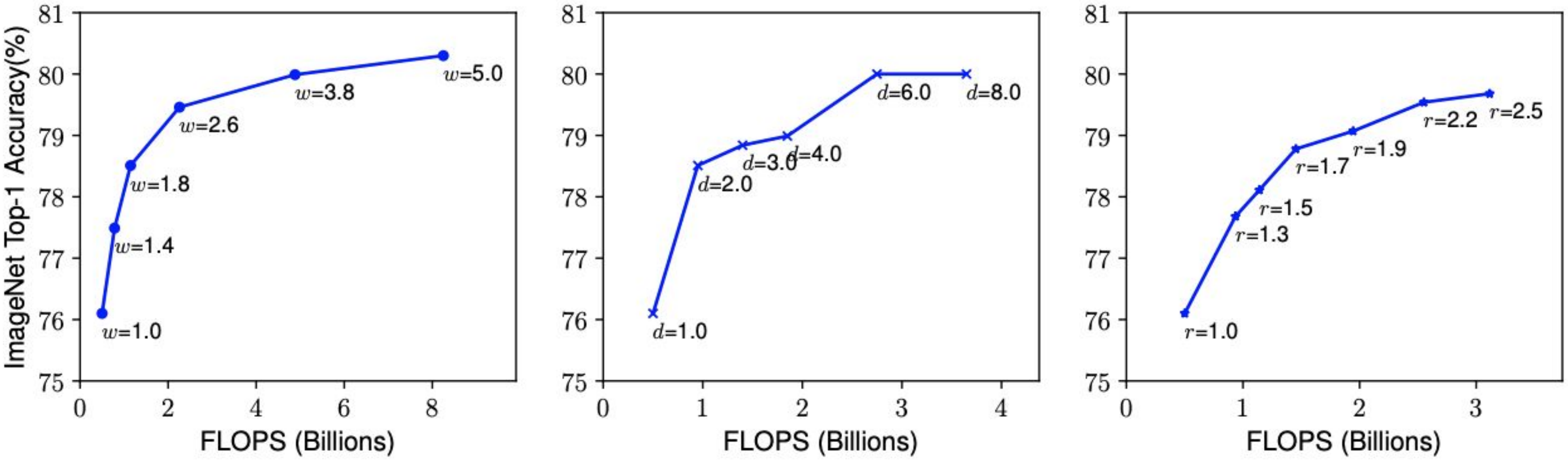
Step2: towards better accuracy & efficiency

Mingxing Tan, Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML 2019. https://arxiv.org/abs/1905.11946
Mingxing Tan, Ruoming Pang, Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. CVPR 2020. https://arxiv.org/abs/1911.09070

How to Scale Up A ConvNet?

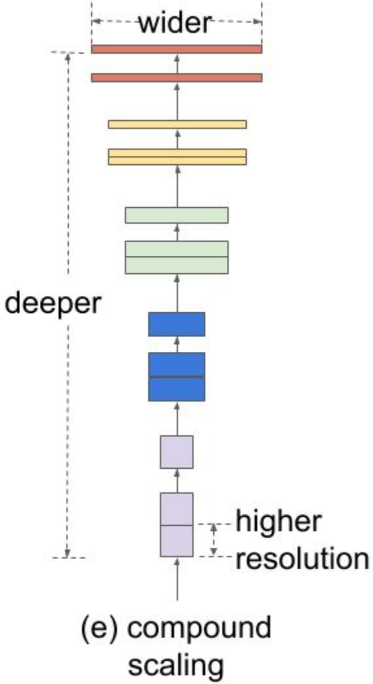


Limitations of Single-Dimension Scaling



Accuracy saturates quickly if scaling by any single dimension

How to Scale Up A ConvNet?



Compound Scaling

$$
\begin{aligned}
\text{depth: } & d = \alpha^{\phi} \\
\text{width: } & w = \beta^{\phi} \\
\text{resolution: } & r = \gamma^{\phi} \\
\text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\
& \alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned}
$$

Step1:

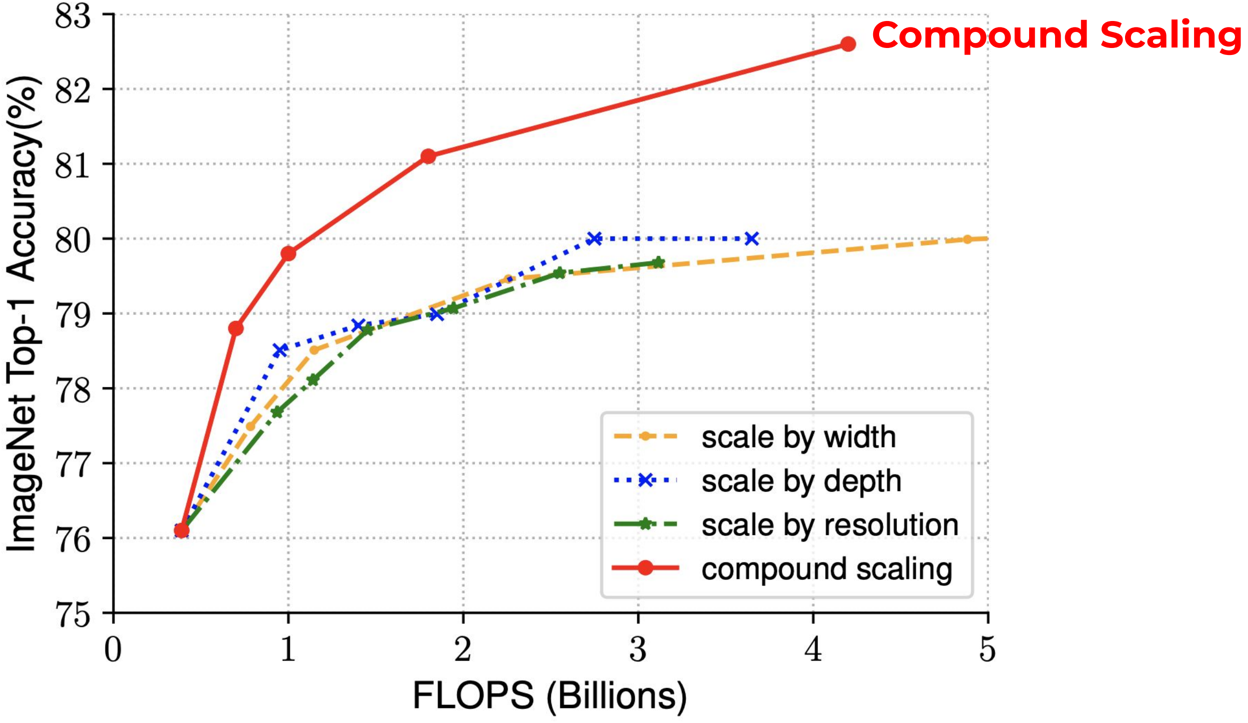
- First fix $\varphi = 2$, and find α, β, γ with local search.

Step2:

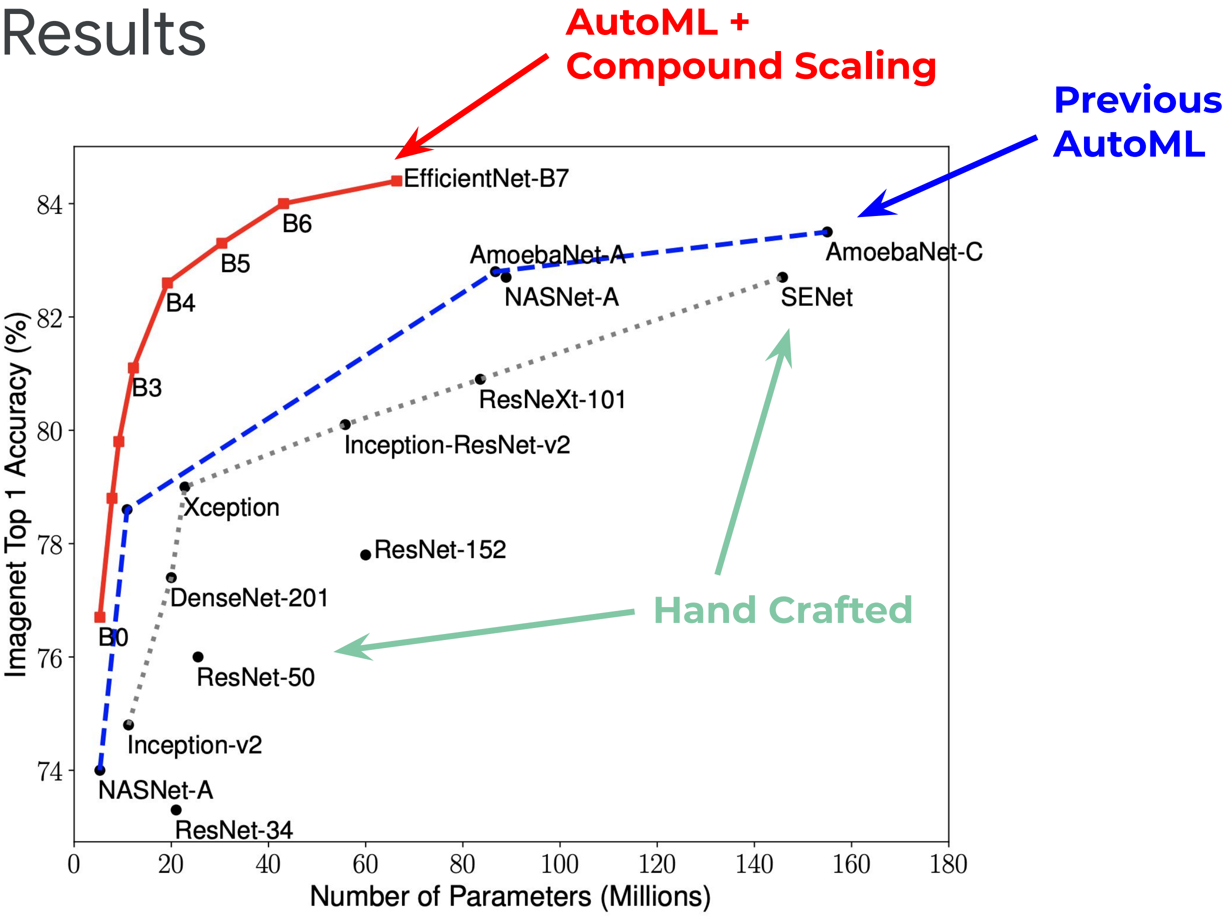
- Then fix α, β, γ, and scale the network with different φ.

Compound scaling improves MobileNetV1, MobileNetV2, and ResNet-50.

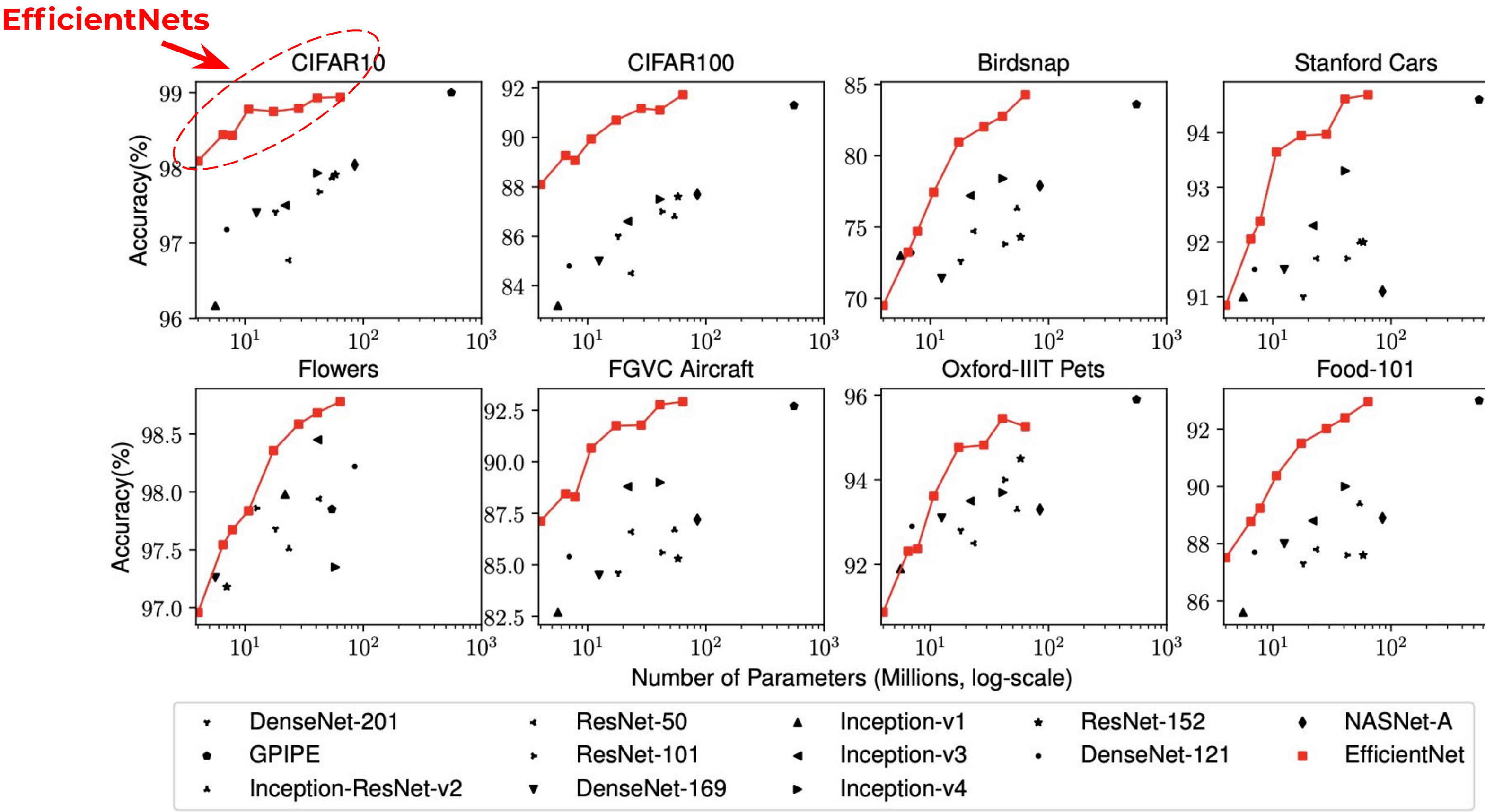
Scaling the Same Baseline EfficientNet-B0



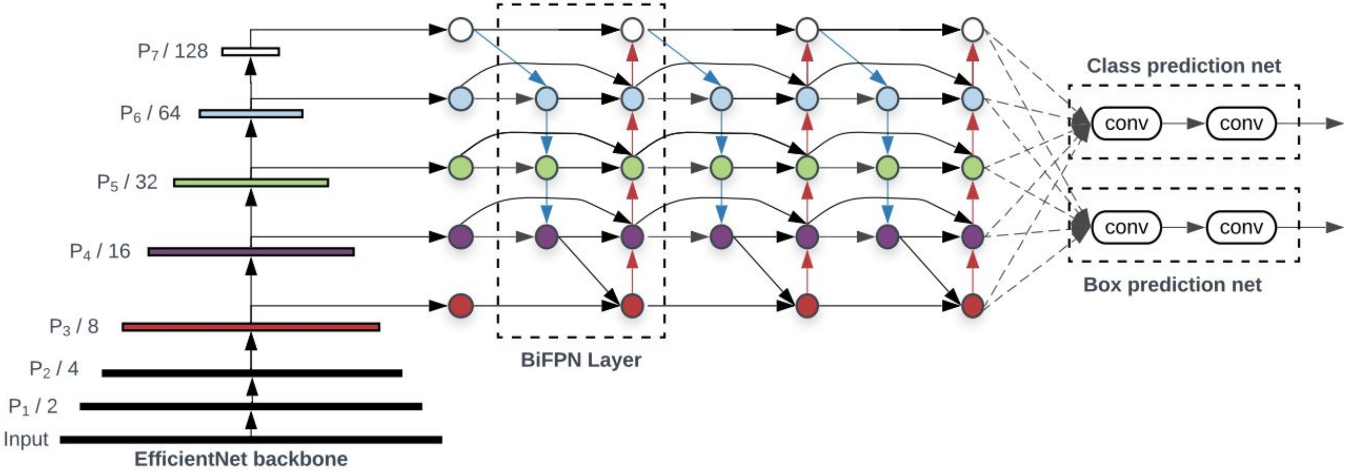
ImageNet Results



Transfer Learning Results

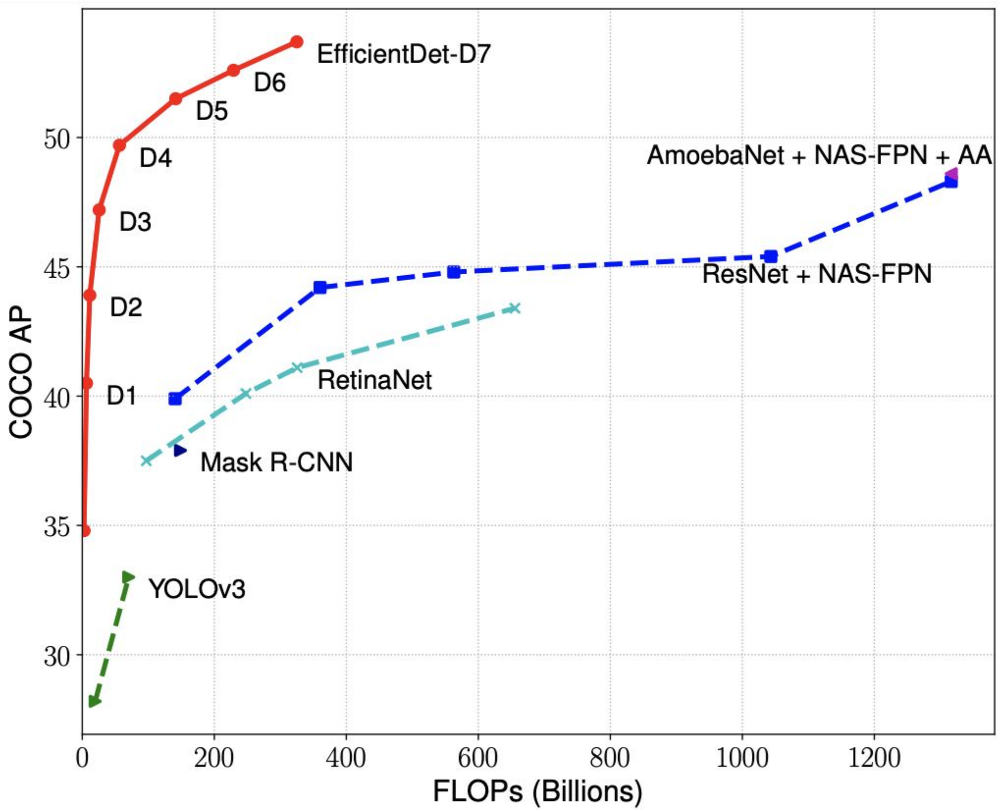


Object Detection



EfficientDet: a new family of detection models based on EfficientNet backbones

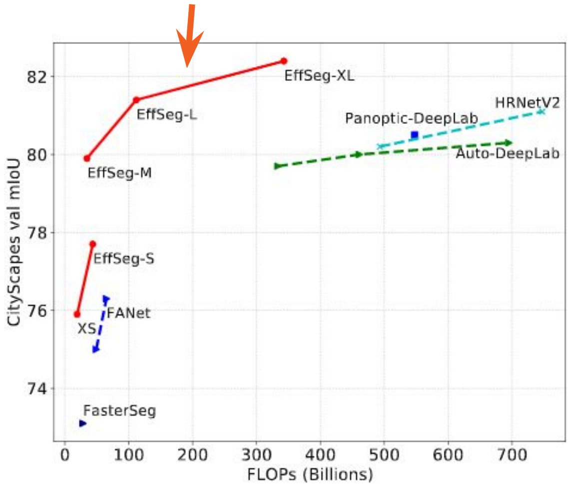
Mingxing Tan, Ruoming Pang, Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. https://arxiv.org/abs/1911.09070



Mingxing Tan, Ruoming Pang, Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. [*https://arxiv.org/abs/1911.09070*](https://arxiv.org/abs/1911.09070)

Others

Semantic Segmentation (Cityscape)



Point Cloud 3D Detection (Waymo OD):

Metric: APH/L2	PED	VEH	CYC	ALL_NS
ShyPillars Single	**0.6884**	**0.6623**	**0.6723**	**0.6743**
Prev. Best Single	0.6170	0.6446	0.6204	0.6273

Video Action Recognition (Kinetics600):

Accuracy > X3D, but

- 67% fewer FLOPs
- 65% less memory.

Wild NAS Topics

How about hyperparamters?

AutoHAS: Hyperparameter and Architecture Search

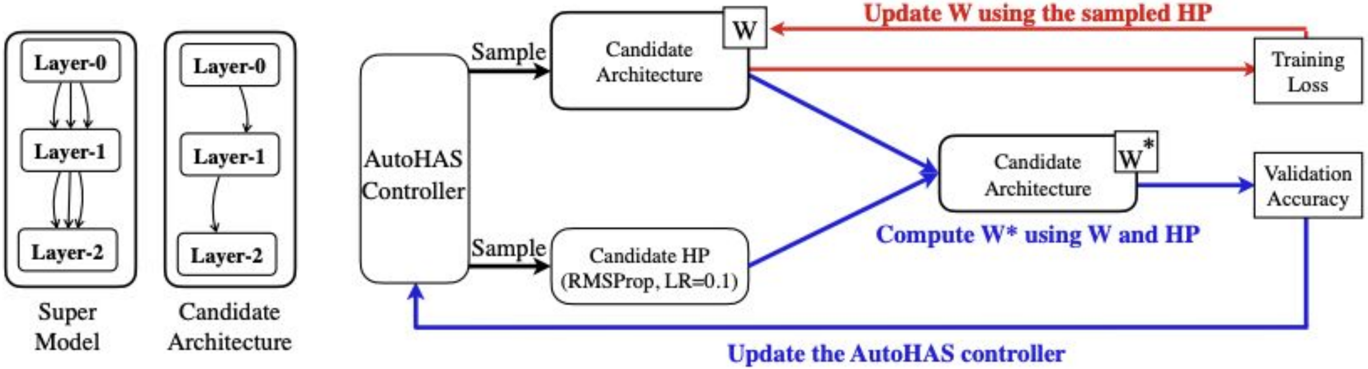
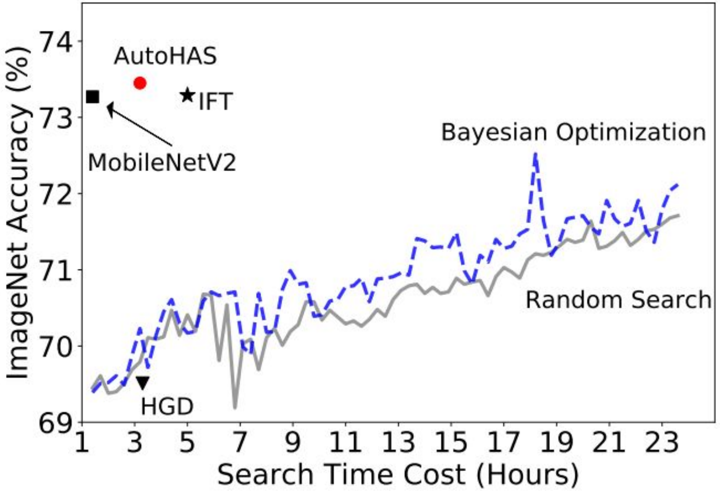


Figure 1. **The overview of AutoHAS.** LEFT: Each candidate architecture's weights are shared with a super model, where each candidate is a sub model within this super model. RIGHT: During the search, AutoHAS alternates between optimizing the shared weights of super model $\mathcal{W}$ and updating the controller. It also creates temporary weights $\mathcal{W}^*$ by optimizing the sampled candidate architecture using the sampled candidate hyperparameter (HP). This $\mathcal{W}^*$ will be used to compute the validation accuracy as a reward so as to update the AutoHAS controller to select better candidates. Finally, $\mathcal{W}^*$ is discarded after updating the controller so as not to affect the original $\mathcal{W}$.

Key idea: Jointly search for neural network and hyperparameters

Dong *et al.*, AutoHAS: Efficient Hyperparameter and architecture search. 2020. https://arxiv.org/abs/2006.03656

AutoHAS: Hyperparameter and Architecture Search



10x faster than Vizier

ImageNet	77.2% -> 77.9% (**+0.7%**)
CIFAR-100	76.3% -> 78.4% (**+2.1%**)
Flowers	74.0% -> 85.4% (**+11.4%**)

Accuracy gains on different datasets

Dong *et al.*, AutoHAS: Efficient Hyperparameter and architecture search. 2020. https://arxiv.org/abs/2006.03656

Can we use NAS to search for back-propogation?

AutoML Zero: search for models with basic math ops

(1) Programs with 3 functions

```
def Setup():
def Predict():
def Learn():
```

Scalars: s0, s1, ...
Vectors: v0, v1, ...
Matrices: m0, m1, ...

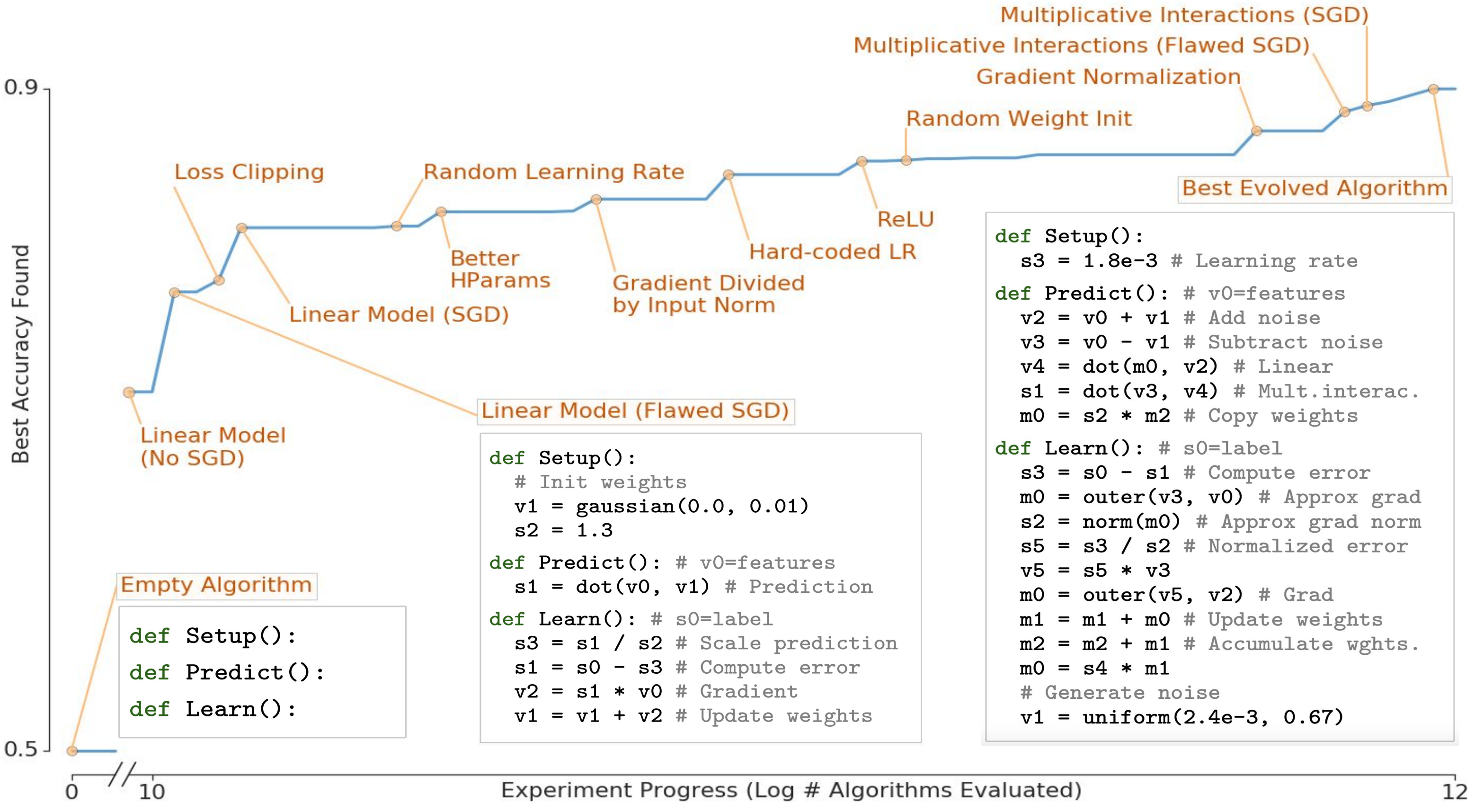
(2) Basic math operations (64)

- Arithmetic
  - a + b, a * b, ...
- Trigonometry
  - sin(a), cos(a), ...
- Pre-calculus
  - exp(a), log(a), ...
- Linear algebra
  - dot(a, b), norm(a), ...
- Probability & Stats
  - gaussian(a, b), mean(a), ...

(3) Evaluation

```
# (Setup, Predict, Learn) is the input ML algorithm.
# Dtrain / Dvalid is the training / validation set.
# sX/vX/mX: scalar/vector/matrix var at address X.
def Evaluate(Setup, Predict, Learn, Dtrain, Dvalid):
  # Zero-initialize all the variables (sX/vX/mX).
  initialize_memory()
  Setup() # Execute setup instructions.
  for (x, y) in Dtrain:
    v0 = x # x will now be accessible to Predict.
    Predict() # Execute prediction instructions.
    # s1 will now be used as the prediction.
    s1 = Normalize(s1) # Normalize the prediction.
    s0 = y # y will now be accessible to Learn.
    Learn() # Execute learning instructions.
  sum_loss = 0.0
  for (x, y) in Dvalid:
    v0 = x
    Predict() # Only execute Predict(), not Learn().
    s1 = Normalize(s1)
    sum_loss += Loss(y, s1)
  mean_loss = sum_loss / len(Dvalid)
  # Use validation loss to evaluate the algorithm.
  return mean_loss
```

AutoML Zero



Multiplicative Interactions (SGD)
Multiplicative Interactions (Flawed SGD)
Gradient Normalization
Random Weight Init
Best Evolved Algorithm
Loss Clipping
Random Learning Rate
ReLU
Hard-coded LR
Better HParams
Gradient Divided by Input Norm
Linear Model (SGD)
Linear Model (Flawed SGD)
Linear Model (No SGD)
Empty Algorithm
def Setup():
def Predict():
def Learn():
def Setup():
# Init weights
v1 = gaussian(0.0, 0.01)
s2 = 1.3
def Predict(): # v0=features
s1 = dot(v0, v1) # Prediction
def Learn(): # s0=label
s3 = s1 / s2 # Scale prediction
s1 = s0 - s3 # Compute error
v2 = s1 * v0 # Gradient
v1 = v1 + v2 # Update weights
def Setup():
s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
v2 = v0 + v1 # Add noise
v3 = v0 - v1 # Subtract noise
v4 = dot(m0, v2) # Linear
s1 = dot(v3, v4) # Mult.interac.
m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
s3 = s0 - s1 # Compute error
m0 = outer(v3, v0) # Approx grad
s2 = norm(m0) # Approx grad norm
s5 = s3 / s2 # Normalized error
v5 = s5 * v3
m0 = outer(v5, v2) # Grad
m1 = m1 + m0 # Update weights
m2 = m2 + m1 # Accumulate wghts.
m0 = s4 * m1
# Generate noise
v1 = uniform(2.4e-3, 0.67)
Best Accuracy Found
0.9
0.5
0
10
12
Experiment Progress (Log # Algorithms Evaluated)

Thank you