CVPR 2020 Tutorial on

From HPO to NAS: Automated Deep Learning

Hang Zhang[1], Cedric Archambeau[1], Song Han[2], Matthias Seeger[1], Thibaut Lienart[1], Chongruo Wu[1,3], Ligeng Zhu[2], Haotian Tang[2], Aaron Klein[1]

Amazon[1], MIT[2], UC Davis[3]

Automated Deep Learning Tutorial Overview

- AutoML Background and Overview (by Cedric Archambeau)
- Introducing AutoGluon Toolkit (by Hang Zhang)
- AutoML for TinyML with Once-for-All Network (by Song Han)
- Hands-on Section
  - Quick Start and ProxylessNAS Tutorial (by Chongruo Wu)
  - Advanced HPO Algorithm (by Thibaut Lienart)
  - Once-For-All Network Tutorial (by Ligeng Zhu and Haotian Tang)
  - Code Example Walk-through (by Hang Zhang)

https://hangzhang.org/CVPR2020/
https://autogluon.mxnet.io/

AutoGLUON: An AutoML Toolkit for Deep Learning

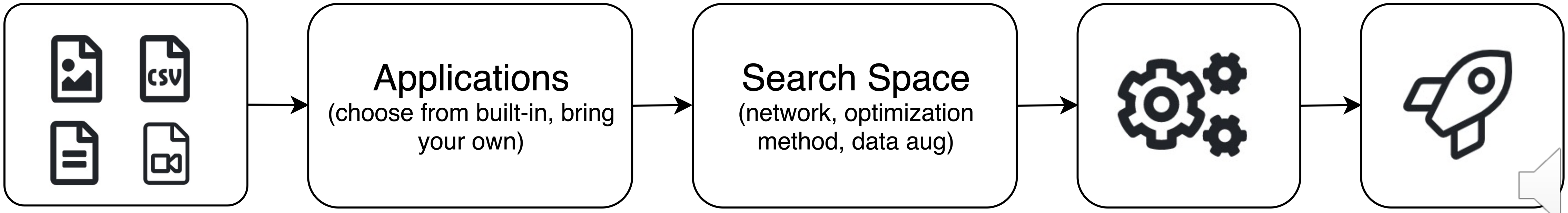
Hang Zhang

Applied Scientist, Amazon

CVPR Tutorial on From HPO to NAS: Automated Deep Learning (Jun 2020)

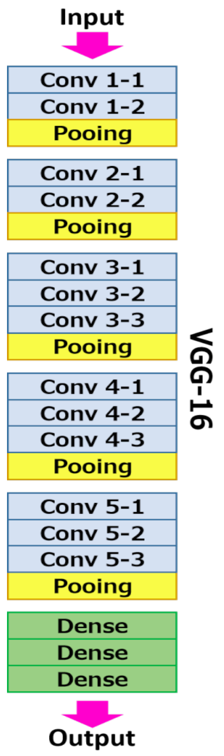
AutoGLUON

- A Popular Toolkit:
  - 2.3K GitHub Starts, many media posts online
- AutoGluon Users:
  - DL expert/researcher (improves their model performance)
  - Researcher on AutoML/HPO algorithm
  - Developer conducting large scale distributed training (GPU management)
- Research Prototype => AutoML Production Pipeline (SageMaker)

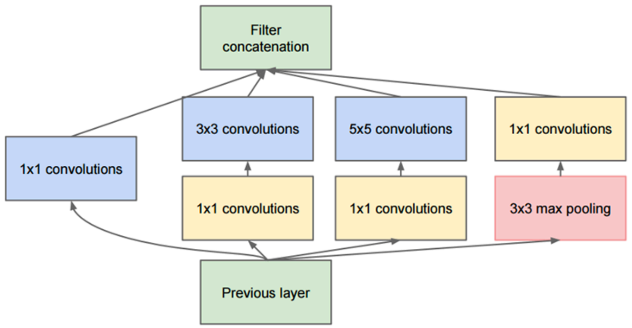


Why HPO in Computer Vision?

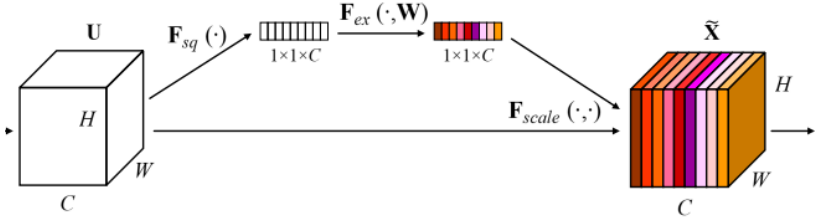
- Feature Engineering => Network Engineering => Architecture Space Design
- Increasing number of design choices



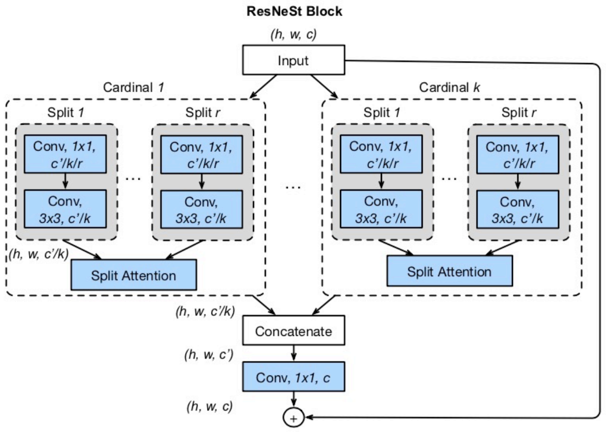
Very Deep Convolutional Networks for Large-Scale Image Recognition



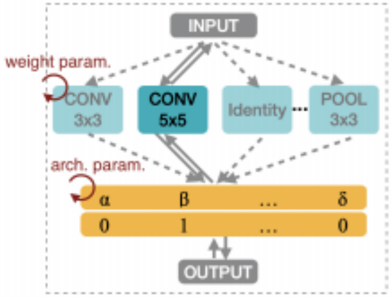
Going Deeper with Convolutions



Squeeze-and-Excitation Networks



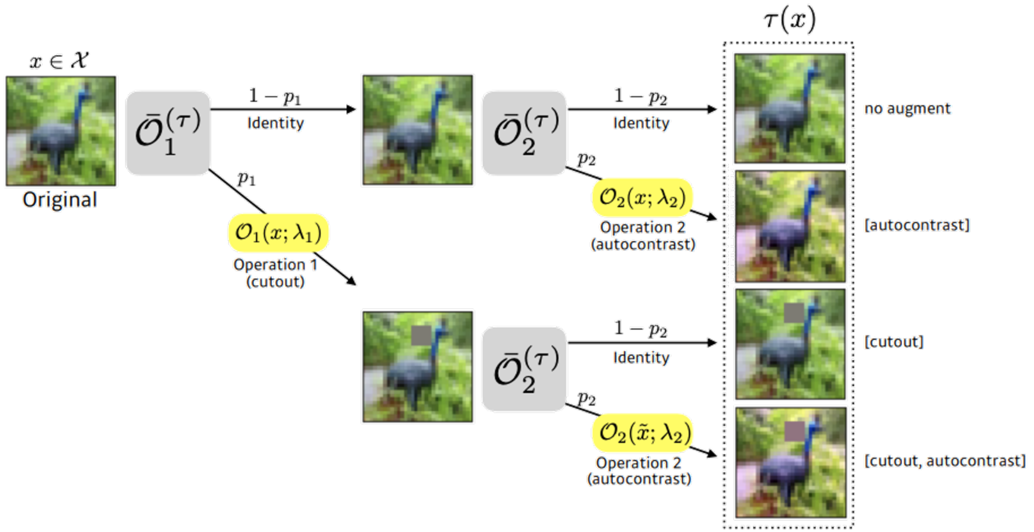
ResNeSt: Split-Attention Network



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware

HPO for Computer Vision

- Optimization of training strategies and HPs
  - AutoAugment (Fast AA, RandAA)
  - FBNetV3 (joint optimization)
- Train recipe:
  - Data augmentation policies
  - Mix-up training
  - Label smoothing
  - Optimization HPs: learning rate, weight decay



What Does HPO Look Like

- Spinning up the experiments using different choices
- Learn lessons and suggest next choice

```
# change the rank for worker node
python train_dist.py --dataset imagenet --model resnest50 --lr-scheduler cos --epochs 270 --
checkname resnest50 --lr 0.025 --batch-size 64 --dist-url tcp://MASTER:NODE:IP:ADDRESS:23456 --
world-size 4 --label-smoothing 0.1 --mixup 0.2 --no-bn-wd --last-gamma --warmup-epochs 5 --rand-aug
--rank 0
```

```
# change the rank for worker node
```

```
# change the rank for worker node
```

```
# change the rank for worker node
python train_dist.py --dataset imagenet --model resnest50 --lr-scheduler cos --epochs 270 --
checkname resnest50 --lr 0.025 --batch-size 64 --dist-url tcp://MASTER:NODE:IP:ADDRESS:23456 --
world-size 4 --label-smoothing 0.1 --mixup 0.2 --no-bn-wd --last-gamma --warmup-epochs 5 --rand-aug
--rank 0
```

What AutoGLUON Provides

- AutoGluon Toolkit
  - Easy-to-use and easy-to-customize
  - Efficient in Search (BayesOpt, Hyperband, Reinforcement Learning)
  - Scalable (Distributed Search, AWS Batch, SageMaker Integration)

- Goal of this tutorial
  - Change the conventional way of doing research manually
  - Constructing search space and pass the workload to the machines

Easy to Use – Search Space

- Integer Space (similarly for Categorical and Real Space):

```
import autogluon as ag
a = ag.space.Int(lower=0, upper=10)
print(a)
print(a.rand)
```

```
Int: lower=0, upper=10
```

```
7
```

- Dict Space (similarly for List Space):

```
g = ag.space.Dict(key1=ag.space.Categorical('alpha', 'beta'),
                  key2=ag.space.Int(0, 3),
                  key3='constant' )
print(g.rand)
```

```
{'key1': 'beta', 'key2': 1, 'key3': 'constant'}
```

Easy to Customize – Training Function

- Allow HPO on python training scripts
  automatically generate search algorithm and conduct HPO

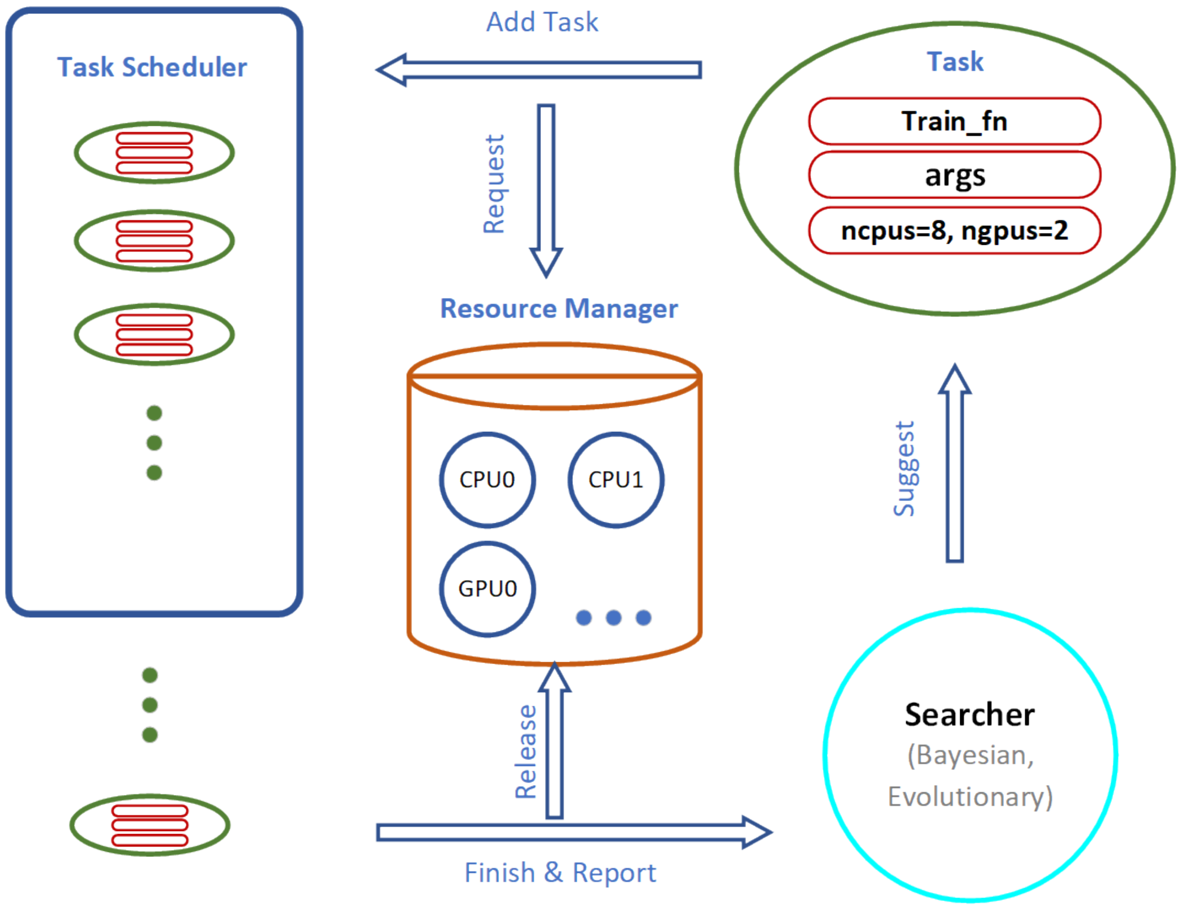
```
@ag. args(
        batch_size=128,
        lr=ag.Real(1e-4, 1e-2),
)
def train_fn(args, reporter):
        print(args.lr, args.batch_size)
        reporter(epoch=0, accuracy=0.9)

myscheduler = ag.scheduler.FIFOScheduler(train_fn, num_trials=25)
myscheduler.run()
```

https://autogluon.mxnet.io/tutorials/course/script.html

Easy to Customize – Searchable Objects

- Defining search space and constructing dynamic object in one go:
  - Searchable object from a function:
```
import gluoncv as gcv
@ag.func(multiplier=ag.Categorical(0.25, 1.0))
def get_mobilenet(multiplier):
    return gcv.model_zoo.MobileNetV2(multiplier=multiplier, classes=4)
```
  - Searchable object from a class:
```
from mxnet import optimizer as optim
@ag.obj(learning_rate=ag.space.Real(1e-4, 1e-1, log=True),
        wd=ag.space.Real(1e-4, 1e-1))
class Adam(optim.Adam):
    pass
```
  - Pass the searchable object as the input of the train_fn:
```
@ag. args(net = mobilenet(), optim = Adam())
def train_fn(args, reporter):
    pass
```

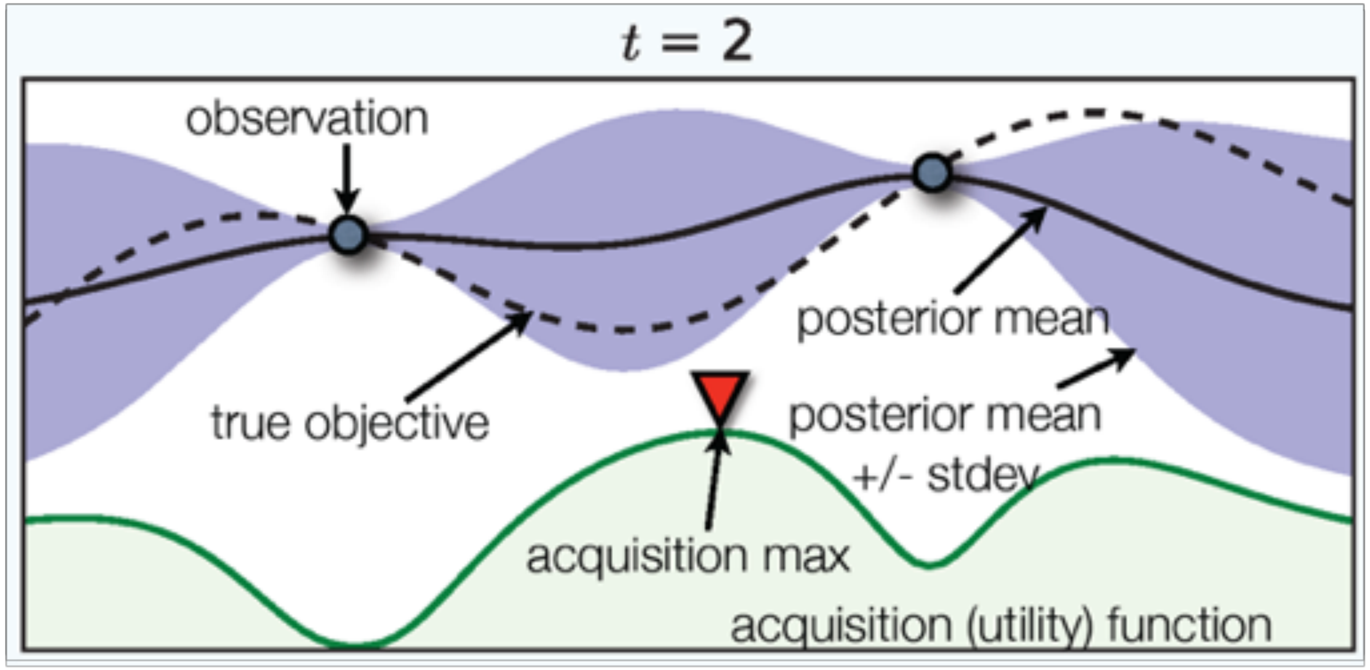
Efficient in Search – Advanced Search Strategies



- Searcher: suggests configurations for the next training jobs.
- Scheduler: schedules the training job when the computation resources are available.
- All components are modularized and easy to extend

Search Algorithms (Bayesian Optimization)

- Surrogate Model:
  reflects the probabilistic beliefs
- Acquisition function:
  which hyperparameter configuration to try next

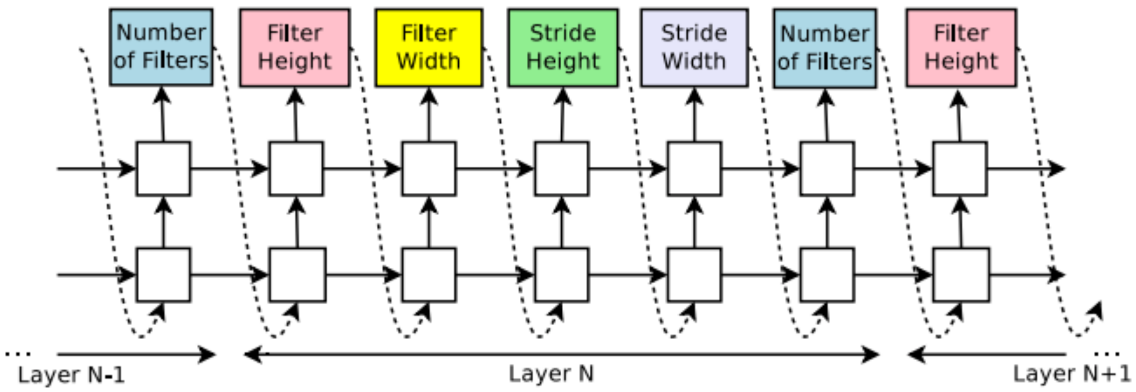


scheduler = ag.scheduler.FIFOScheduler(train_fn, searcher='bayesopt')

http://autogluon.s3.amazonaws.com/api/autogluon.searcher.html#skoptsearcher

Search Algorithms (RL, Gluon)

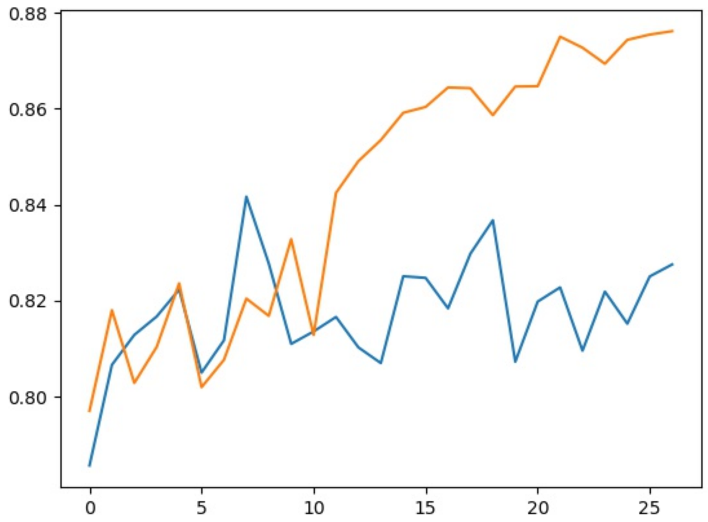
- Auto Construct a LSTM Controller based on Search Space



$$J(\theta) = E_{a\sim P(a;\theta)}[R(a)]$$

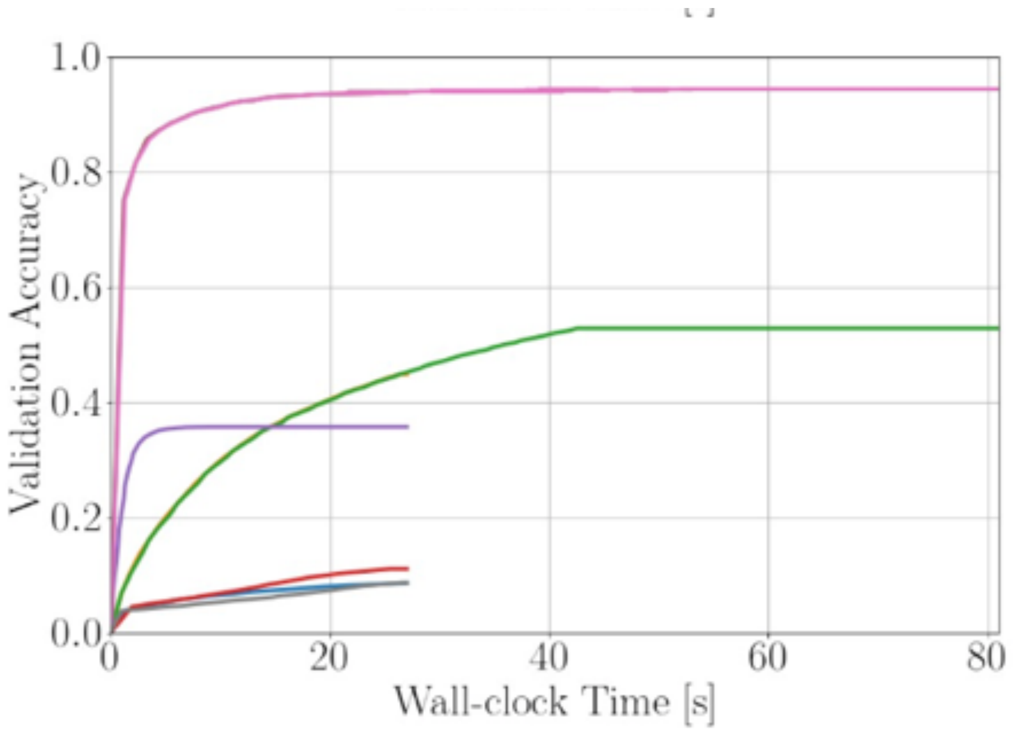
$$\nabla_\theta J(\theta) = E_{a\sim P(a;\theta)}[R(a)\nabla_\theta \log P(a;\theta)]$$

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\log P(a_t|a_{t-1};\theta)(R_k - b)$$



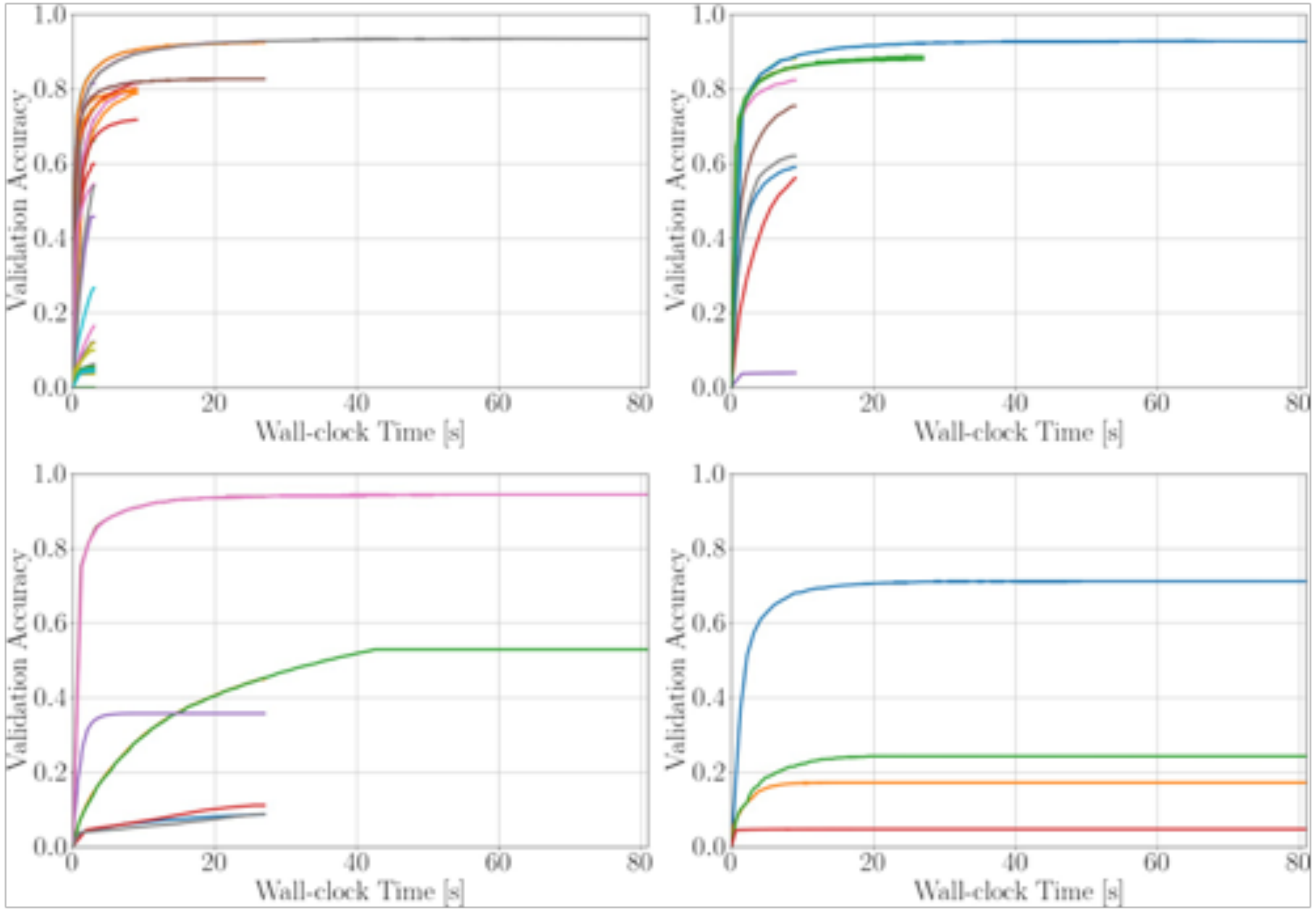
Zoph, Barret, and Quoc V. Le. "Neural architecture search with reinforcement learning."
Reinforce Algorithm http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture14.pdf

Early Stopping – Succesive Halving



- Randomly sample a set of configurations
- Evaluate the performances of all currently remaining configurations
- Throw out, the bottom half of the worst configurations
- Go back to 2. and repeat until one configuration remains.

Early Stopping – Hyperband



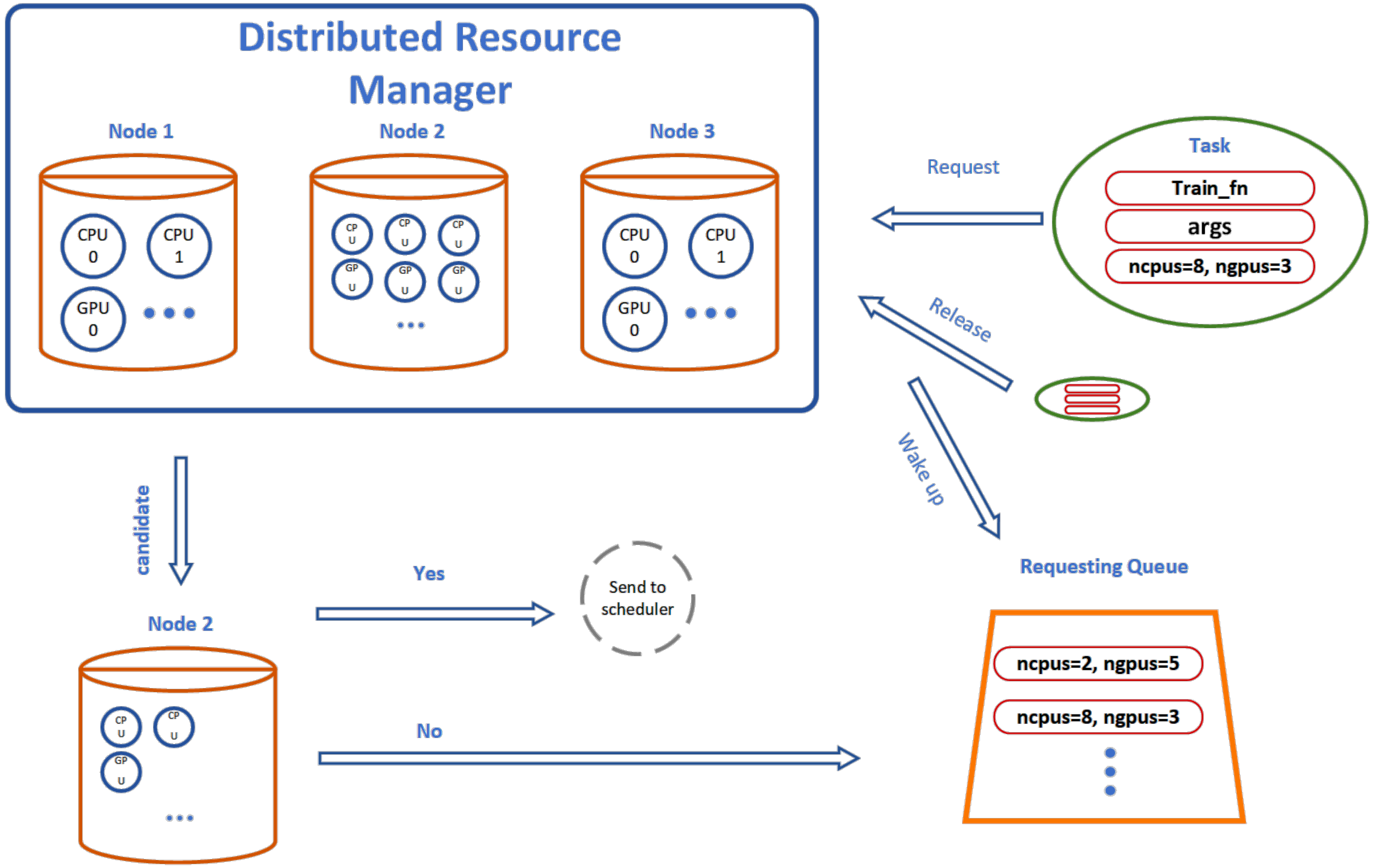
- Number of Total Iters: N
- Number of Successive Halving Brackets: $N_s = \log\left(\frac{N}{\eta}\right) + 1$, where η is a constant
- Number of iters per successive halving: $B = N_s \cdot N$

scheduler = ag.scheduler.
HyperbandScheduler(train_fn)

Scalable -- Distributed Search

- Seamless Experience on Multi-machines
- Automatically manage resource and schedule on remote machines:

```
ip_addrs = ['35.166.23.232',
            '35.166.30.159']
results = ag.schedule. FIFOScheduler(
        train_fn,
        dist_ip_addrs=ip_addrs)
```



http://autogluon.s3.amazonaws.com/tutorials/course/distributed.html

Running on AWS SageMaker

- SageMaker setup all the environment for you
- Upload dataset to S3

```
prefix = 'DEMO-autogluon-cifar10'
data_location = sess.upload_data('./data', key_prefix=prefix)
```

- Bring your own container:

```
from sagemaker.estimator import Estimator
estimator = Estimator(image_name=ecr_image,
                      role=get_execution_role(),
                      train_instance_count=4,
                      train_instance_type='p3.2xlarge')
estimator.fit(data_location)
```

- Download the models from S3

https://github.com/zhanghang1989/AutoGluon-Docker

Q&A

Thank you!