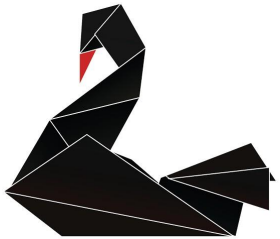
How to do ML if you have lots of Google's GPUs

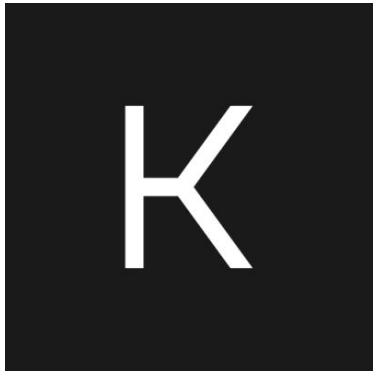
Vladimir "Vlejd" Macko

Powered by



CEAi

Google AI
Residency Program



I (Vladimir Macko) am currently not officially affiliated with any of the entities mentioned above and hence I am not a representative of any of the entities mentioned above and should be taken as such. Terms and conditions may apply.

Outline

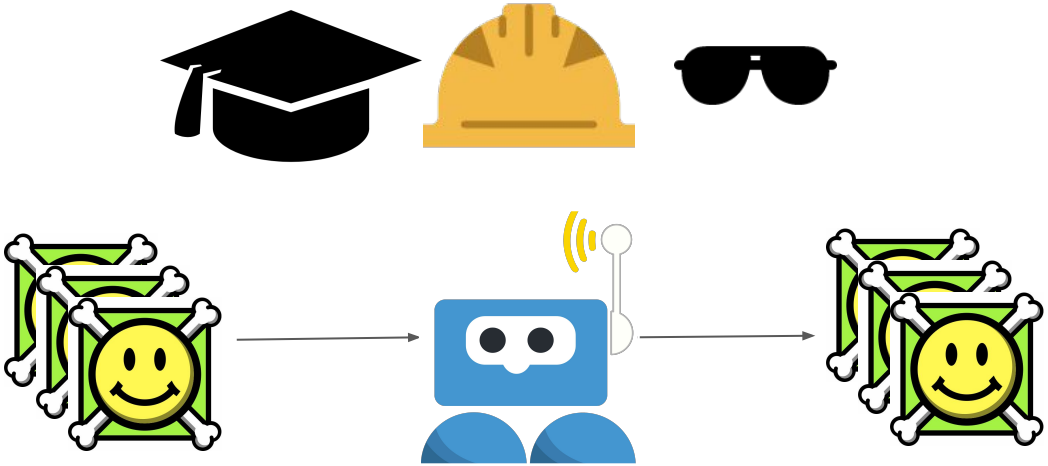
Not ML

1. What is “Google” AI residency
2. How to get there

ML

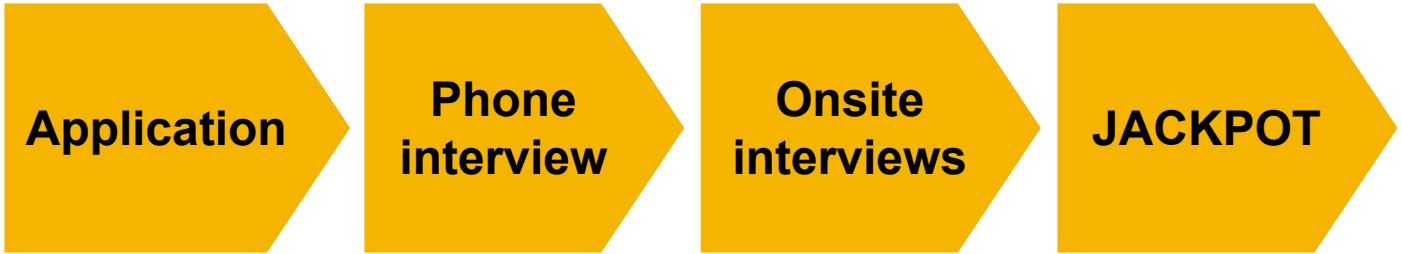
1. AutoML in theory
2. AutoML in practice

Google AI Residency



Bc. Mgr. PhD. Pro.

Process



Indicators of getting in

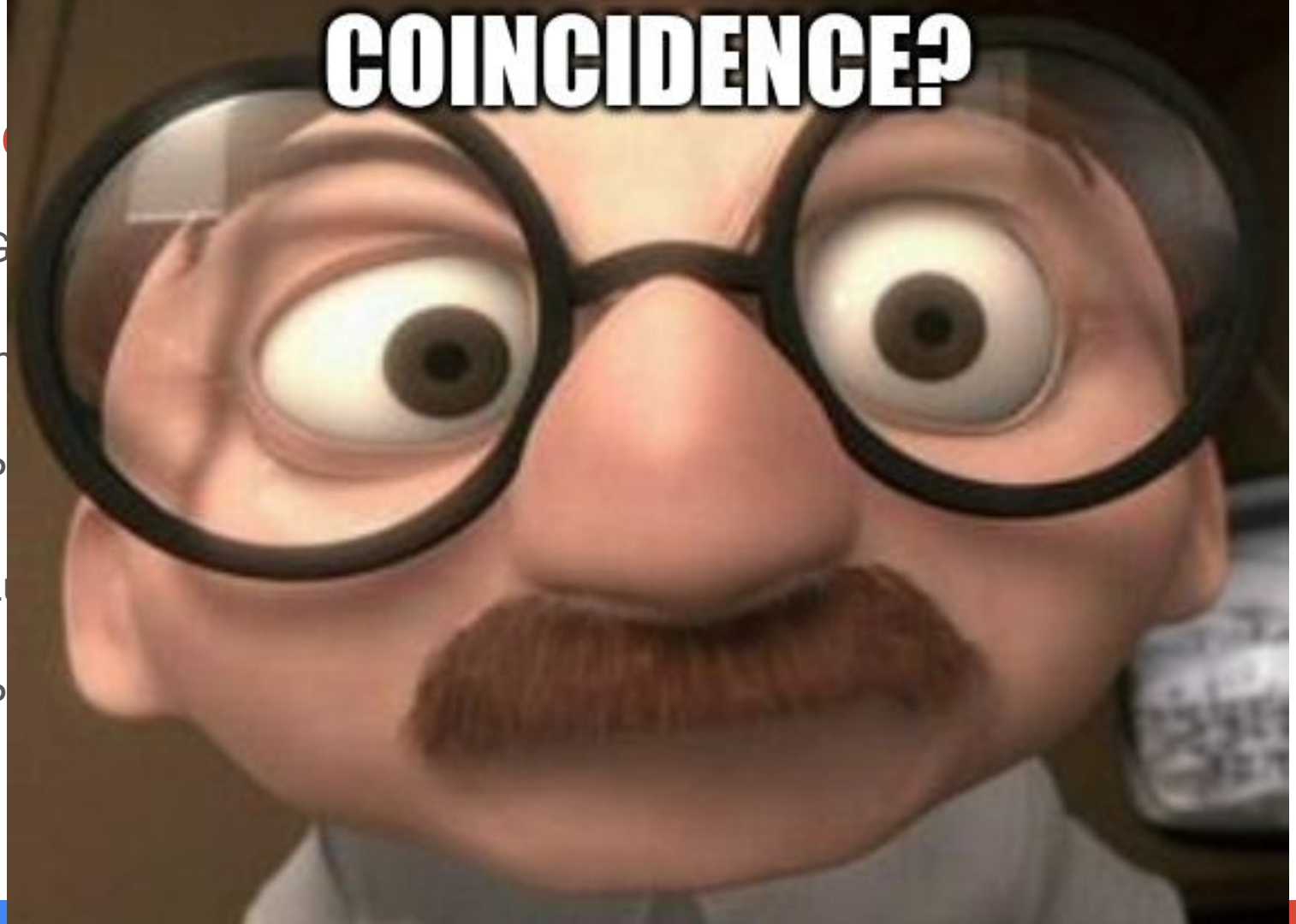
- ✓ Google Internships
- ✓ Industry experience
- ✓ PhD. (not) required
- ✓ Luck!

Indicators of getting in

- ✓ Google Internships
- ✓ Industry experience
- ✓ PhD. (not) required
- ✓ Luck!
- ✓ Photo with Jeff Dean

Indi

- ✓ G
- ✓ In
- ✓ P
- ✓ L
- ✓ P

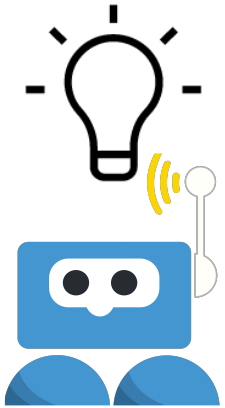


What to work on?

Specific project in mind (or not)

Lightning talks

AdaNet, graphnets for CO with Deepmind



Not ML: DONE

Machine learning

Have you tried to reproduce some State-Of-The-Art ML result recently?

Have you wrote your own code from scratch?

Have you managed to get **the same** performance?

Why is it hard?

Architecture

Hyperparameters

Data processing

Training schedules

Compiler flags

Hardware tricks

Same hardware or ...

Same library or ...

aux_scaling
batch_norm_decay
batch_norm_epsilon
cell_name
data_format
dense_dropout_keep_prob
distributed_group_size
drop_connect_keep_prob
drop_connect_version
drop_path_burn_in_steps
enable_hostcall
gradient_clipping
image_size
iterations_per_loop
l1_decay_rate
label_smoothing
lr
lr_decay_method
lr_decay_value
lr_num_epochs_per_decay
lr_warmup_epochs
momentum_rate
moving_average_decay
normal_cell_hiddenstate_indices
normal_cell_operations
normal_cell_used_hiddenstates
num_cells
num_epochs
num_label_classes
num_reduction_layers
num_shards
num_stem_cells
num_train_images
optimizer
per_host_input_for_training
reduction_cell_hiddenstate_indices
reduction_cell_operations
reduction_cell_used_hiddenstates
reduction_size
rmsprop_decay
rmsprop_epsilon
rmsprop_momentum_rate
shuffle_buffer
stem_reduction_size
stem_type
train_batch_size
train_steps
use_aux_head
use_bp16
use_nesterov
use_tpu
version
weight_decay
xla_enable_hlo_trace
xla_jf_use_rotated_pincer_emitter2d

Wise man once said

Solution = ML expertise + data + computation

Solution = ~~ML expertise +~~ data + **100X** computation

ML expertise

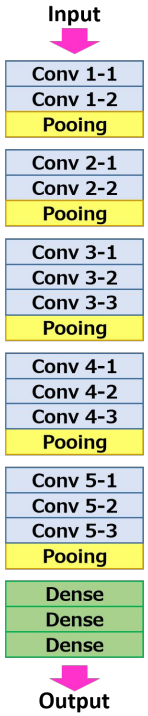
Data preprocessing	→	Autoaugment
Architecture selection	→	**Architecture search**
Hyperparameter tuning	→	Bayesian optimization
Model selection	→	Cross Validation

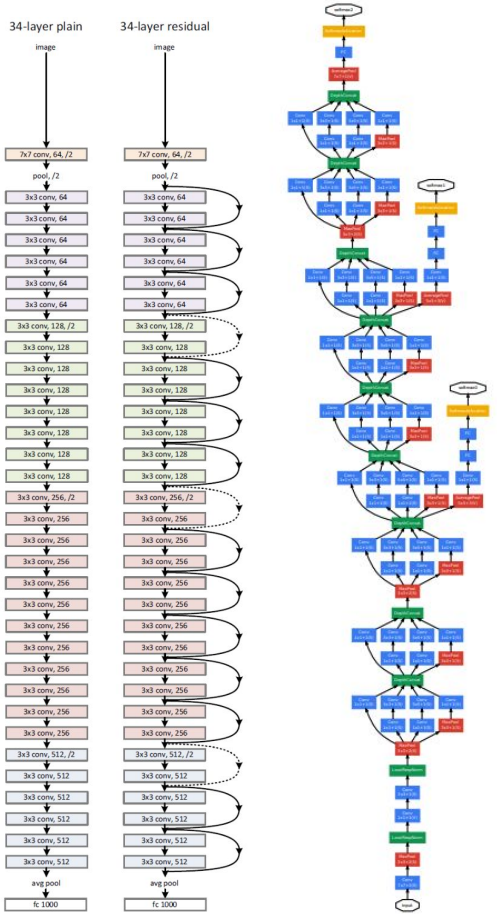
Architecture selection by expert

Take conv/maxpool/dense layers

Constrained by memory & time

Fill parameter budget

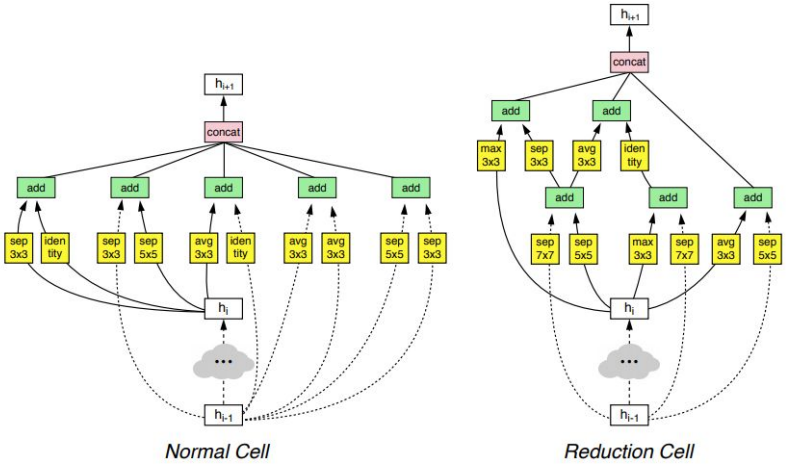


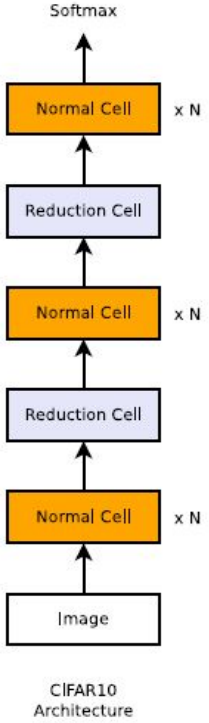


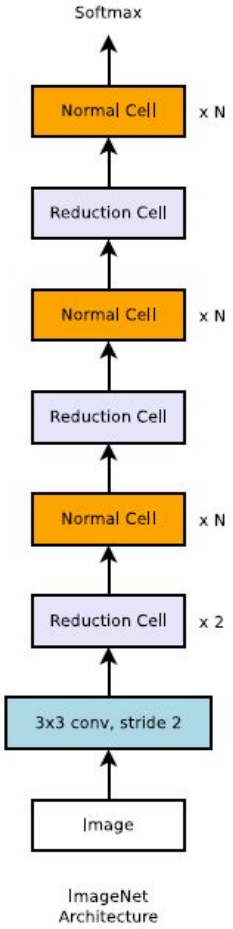
Is this optimal?

What blocks to stack?

Zoph, B., & Le, Q. V. (2016). **Neural architecture search with reinforcement learning.** arXiv preprint arXiv:1611.01578.

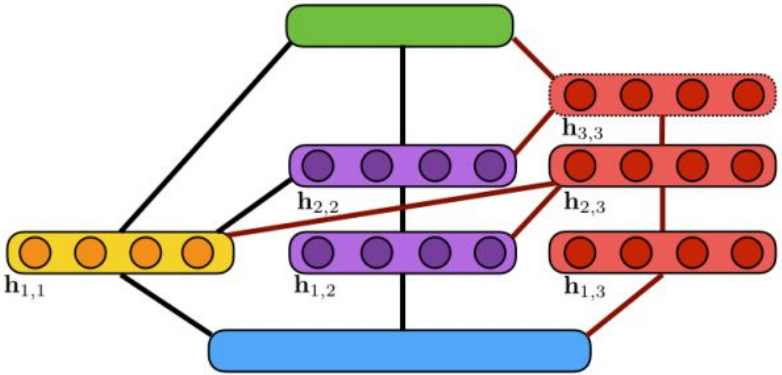






How to stack them?

Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., & Yang, S. (2017, August). **Adanet: Adaptive structural learning of artificial neural networks.** In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 874-883). JMLR. org.



Macko, V., Weill, C., Mazzawi, H., & Gonzalvo, J. (2019). **Improving neural architecture search image classifiers via ensemble learning**. arXiv preprint arXiv:1903.06236.

Almost AdaNet

Input: S, I, G
$f_0 \leftarrow 0$
for $i = 1$ **to** I **do**
  $H_i \leftarrow G(f_{i-1})$
  for $h_i^{(j)}$ **in** H_i **do**
    $\theta^{(j)} \leftarrow \text{argmin}_\theta \mathcal{L}_h(S, h_i^{(j)}(\theta))$
  end for
  $j^* \leftarrow \text{argmin}_j \mathcal{L}_f(S, \frac{1}{i} \cdot h_i^{(j)}(\theta^{(j)}) + \Sigma_{k=1}^{i-1} \frac{1}{i} \cdot h_k)$
  $h_i \leftarrow h_i^{(j^*)}(\theta^{(j^*)})$; $f_i \leftarrow \Sigma_{k=1}^{i} \frac{1}{i} h_k$
end for
return f_I

What about overfitting?

$$Pr\left(test\ error < training\ error + f(complexity)\right) = big$$

Optimize upper bound on test error

Ensemble generalizes better than single tower

Let’s classify

Image classification

Used to be SOTAs 2018

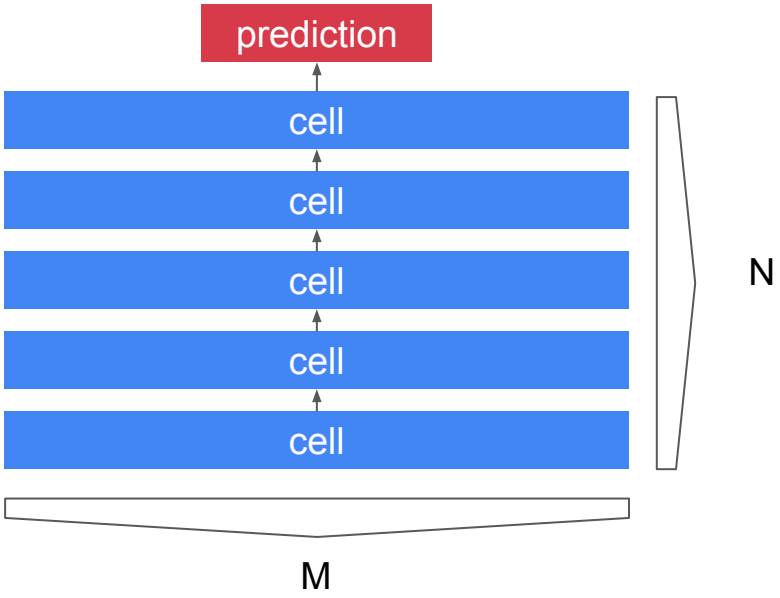
Model	#parameters	CIFAR-10	CIFAR-100
NASNet-A(6@768)	3.3M	2.65	16.58
NAONet (F=36)	10.6M	3.18	15.67
NAONet (F=64)	28.6M	2.98	N/A
NASNet-A(7@2304)	32.6M	2.40	16.03
AmoebaNet-B (6, 128)	34.9M	2.13 ± 0.04	15.80
NAONet (F=128)	128M	2.11	14.75

Classification error
lower is better

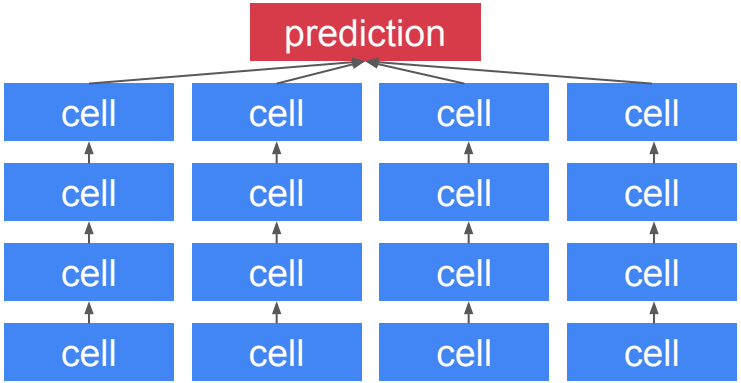
Single tower vs. Ensemble

NASNet(7@2304)

NASNet(N@M)



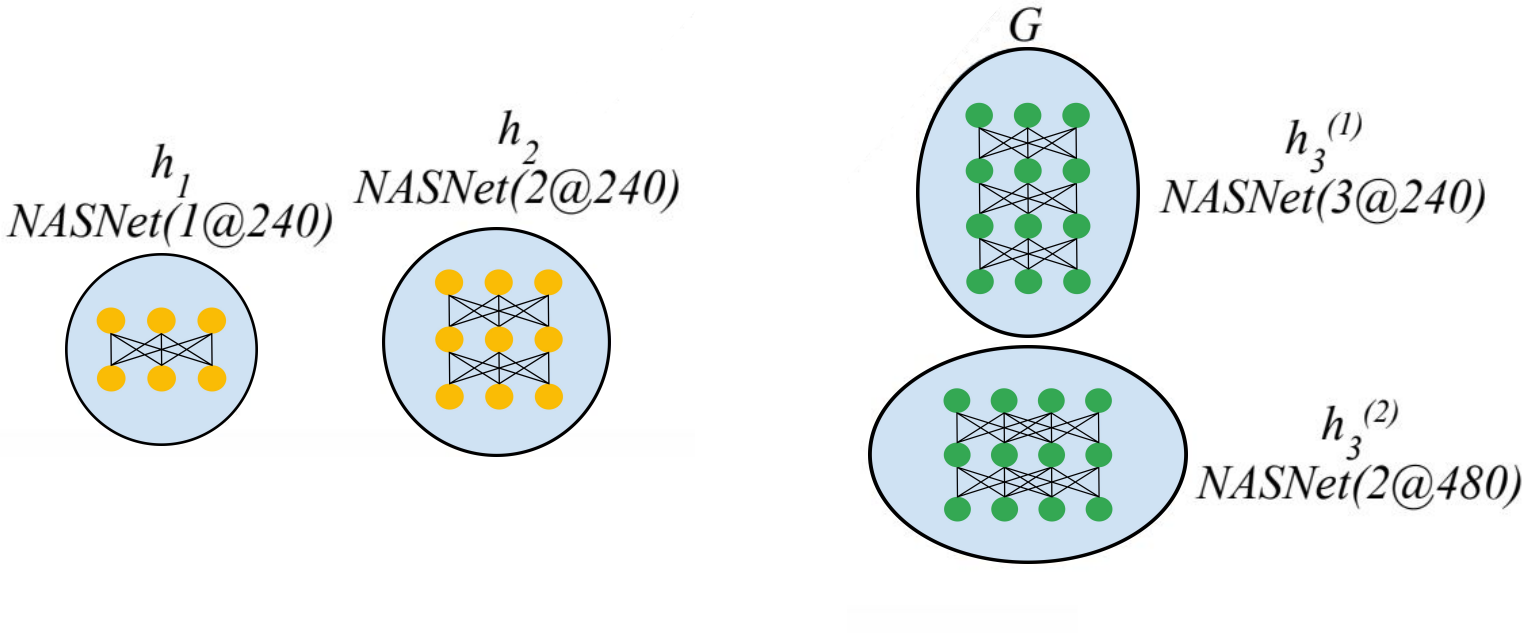
10x NASNet(6@768)



Single tower vs. Ensemble

Model	#parameters	CIFAR-10	CIFAR-100
NASNet-A(6@768)	3.3M	2.65	16.58
NAONet (F=36)	10.6M	3.18	15.67
NAONet (F=64)	28.6M	2.98	N/A
NASNet-A(7@2304)	32.6M	2.40	16.03
AmoebaNet-B (6, 128)	34.9M	2.13 ± 0.04	15.80
NAONet (F=128)	128M	2.11	14.75
$10\times$ NASNet(6@768)	$\leq 33M$	$\mathbf{2.29 \pm 0.07}$	$\mathbf{14.58 \pm 0.05}$

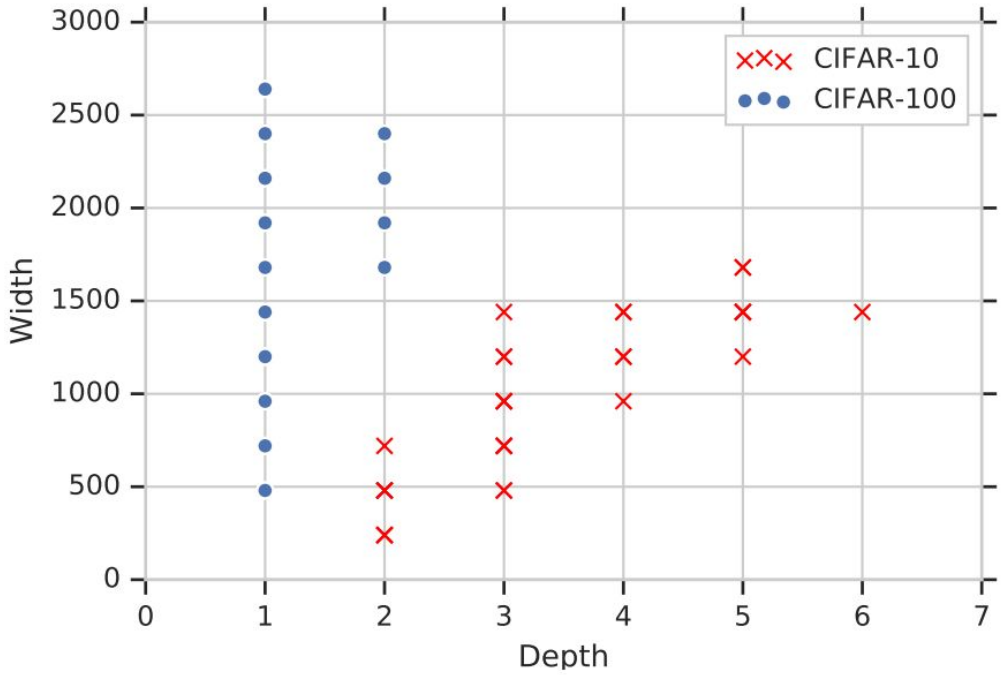
Grow subnetworks



Grow subnetworks

Model	#parameters	CIFAR-10	CIFAR-100
NASNet-A(6@768)	3.3M	2.65	16.58
NAONet (F=36)	10.6M	3.18	15.67
NAONet (F=64)	28.6M	2.98	N/A
NASNet-A(7@2304)	32.6M	2.40	16.03
AmoebaNet-B (6, 128)	34.9M	2.13 ± 0.04	15.80
NAONet (F=128)	128M	2.11	14.75
$10\times$ NASNet(6@768)	$\leq 33M$	$\mathbf{2.29 \pm 0.07}$	$\mathbf{14.58 \pm 0.05}$
AdaNAS (G_D)	$\leq 33M$	2.38 ± 0.06	15.55 ± 0.07

Architectures for CIFAR-10 vs CIFAR-100



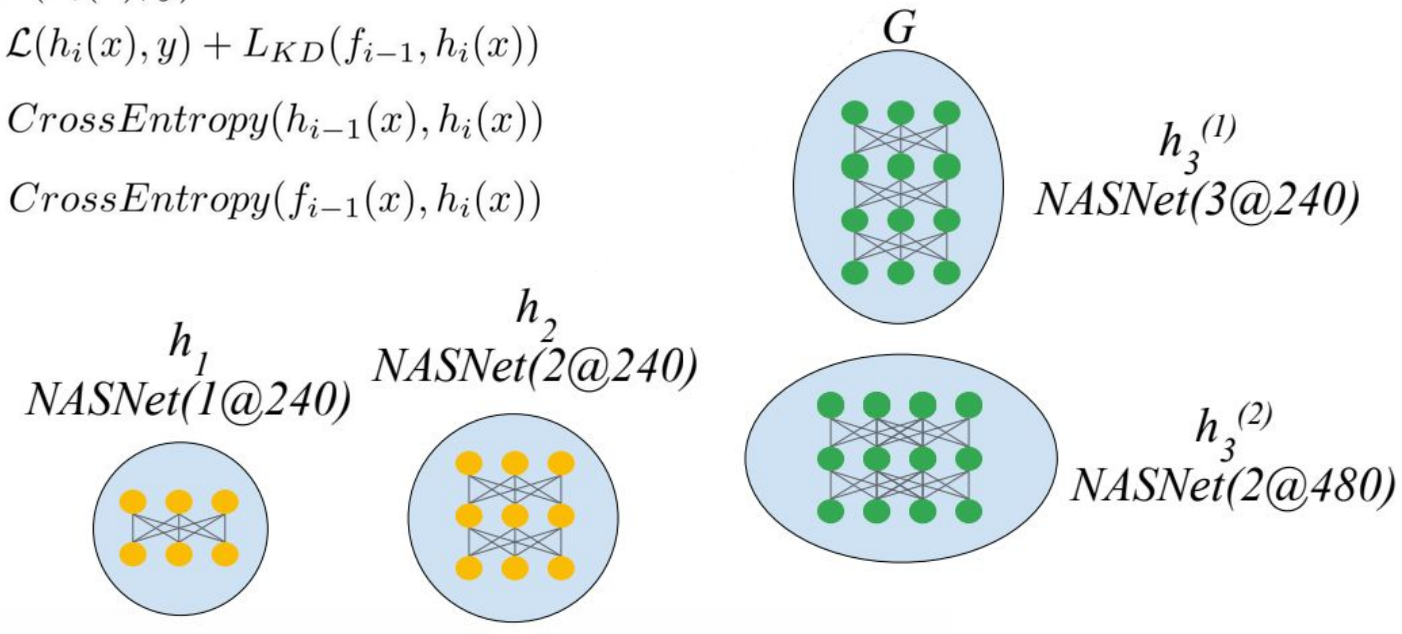
Ensembling tricks

$\mathcal{L}(h_i(x), y)$

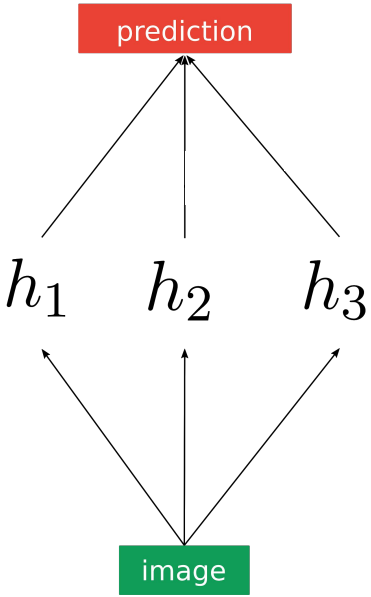
$\mathcal{L}(h_i(x), y) + L_{KD}(f_{i-1}, h_i(x))$

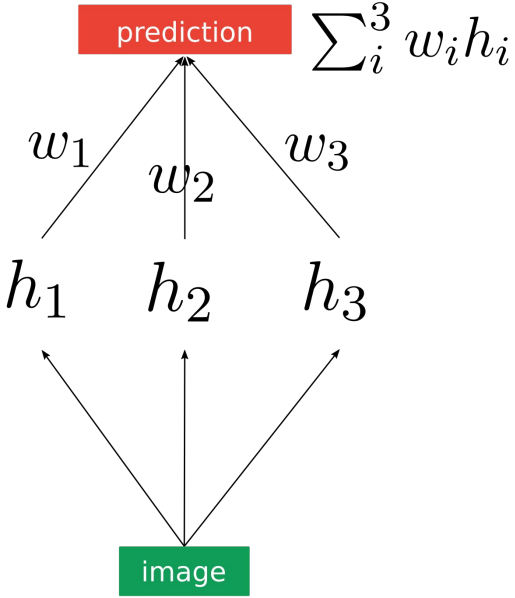
$CrossEntropy(h_{i-1}(x), h_i(x))$

$CrossEntropy(f_{i-1}(x), h_i(x))$



Ensembling tricks





Ensembling tricks

MODEL	#PARAMETERS	CIFAR-10	CIFAR-100
NASNET-A(6@768)	3.3M	2.65	16.58
NAONET (F=36)	10.6M	3.18	15.67
NAONET (F=64)	28.6M	2.98	N/A
NASNET-A(7@2304)	32.6M	2.40	16.03
AMOEBANET-B (6, 128)	34.9M	2.13 ± 0.04	15.80
NAONET (F=128)	128M	2.11	14.75
$10\times$ NASNET(6@768)	$\leq 33M$	$\mathbf{2.29 \pm 0.07}$	$\mathbf{14.58 \pm 0.05}$
ADANAS (G_D)	$\leq 33M$	2.38 ± 0.06	15.55 ± 0.07
ADANAS $+w+$ AKD	$\leq 33M$	$\mathbf{2.26 \pm 0.05}$	14.83 ± 0.14

Other results with AdaNet

- Used in Google AutoML services
- Internal google applications
- Speech recognition, Text, Structured data
- GANs, AutoEncoders

Ensembling rocks

Let’s talk compute

	Resources	GPU-hours
NASNet-A(6@768)	1 GPU, 1.5 d	36
NASNet-A(7@2304)	1 GPU, 5 d	120
Ensembling	10 GPU, 5 d	1.2k*
Growing	10 GPU, 7 d	1.6k*
NAS search	800 GPU, 28 d	537k
NAS search 2	500 GPU, 4 d	48k
NASNet on Imagenet	100 GPU, 8 d	19k

How to get that many GPUs?

A. Amazon

B. Beg for a grant

C. Cloud

D. Donate your soul to Big Tech Corporation™

What will AutoML mean?

Privatization of ML?

Will AutoML take our jobs?

Will it help us?

Are we doomed?

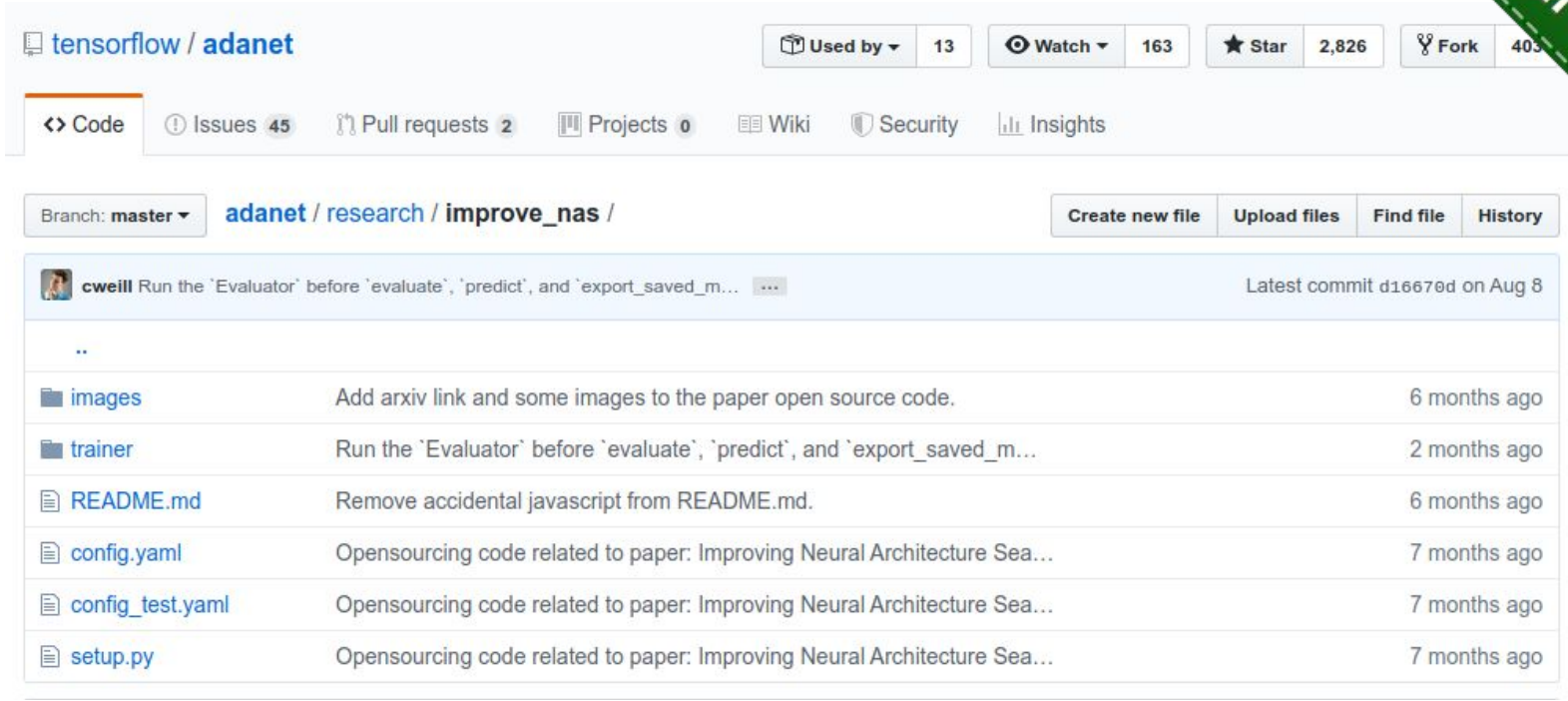
Can you do this at home?

Try this at home: tensorflow/adanet

```
import adanet
import tensorflow as tf
head = MultiClassHead(n_classes=10)
feature_columns = ...
estimator = adanet.AutoEnsembleEstimator(
    head=head,
    candidate_pool={
        "linear":
            tf.estimator.LinearEstimator(
                head=head,
                feature_columns=feature_columns,
                optimizer=...),
        "dnn":
            tf.estimator.DNNEstimator(
                head=head,
                feature_columns=feature_columns,
                optimizer=...,
                hidden_units=[1000, 500, 100])},
    max_iteration_steps=50)
estimator.train(input_fn=train_input_fn, steps=100)
metrics = estimator.evaluate(input_fn=eval_input_fn)
predictions = estimator.predict(input_fn=predict_input_fn)
```

- ✓ Multiple levels of usage
- ✓ Infrastructure
- ✓ Tests

Final remarks



Final remarks

to apply for AI residency 2020

Fin