

CSE 40171: Artificial Intelligence



Neural Network Model Search: High Throughput Screening

Homework #4 has been released
It is due at 11:59PM on 10/18

Quiz #1 is scheduled for 10/30

Let's turn back to neural networks...

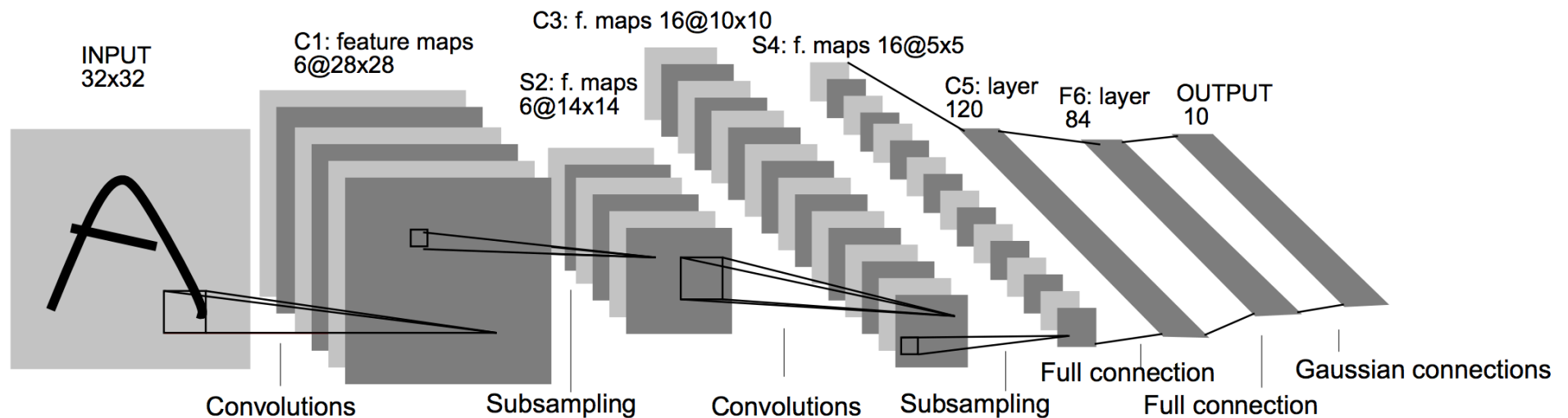


Image credit: LeCun et al. Proc. of the IEEE, Nov. 1998.

Q1: What are the free parameters available in neural network training?

Q2: What are the options for searching that space?

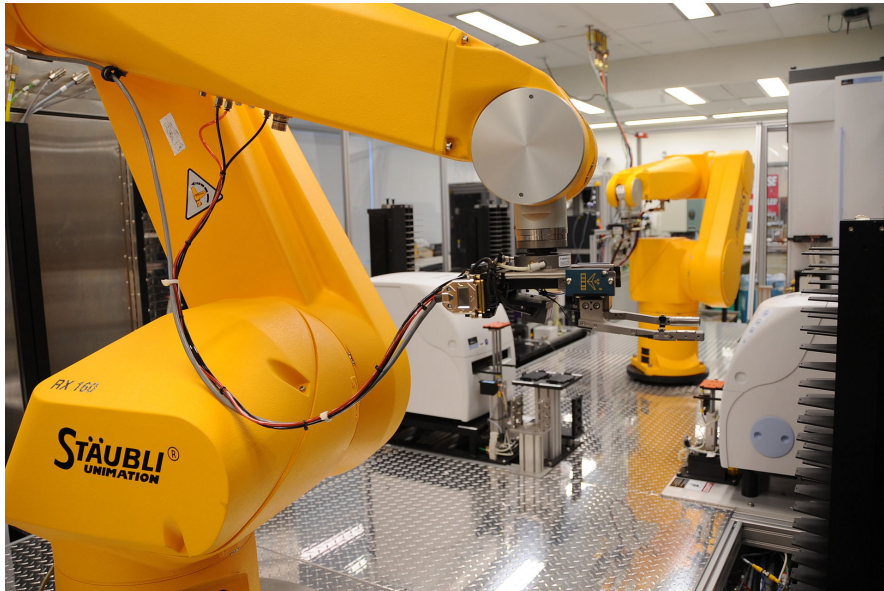
Q3: Is the problem tractable?

Idea: high-throughput screening



Screening Facility Colony Picker © BY-ND 2.0 Wolfgang Hoffmann

How does this work in biology?

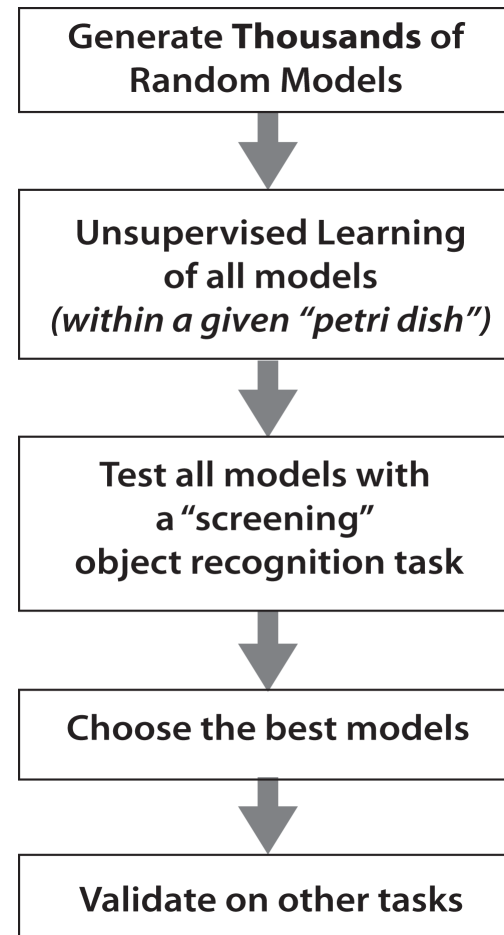


- Commonly applied to drug discovery
- Conduct millions of chemical, genetic, or pharmacological tests
- Use robotics, data processing / control software, sensors, fluidics technology
- Screen for most promising compounds
 - Usually retaining just a handful out of millions of trials

High-Throughput Screening for Neural Networks

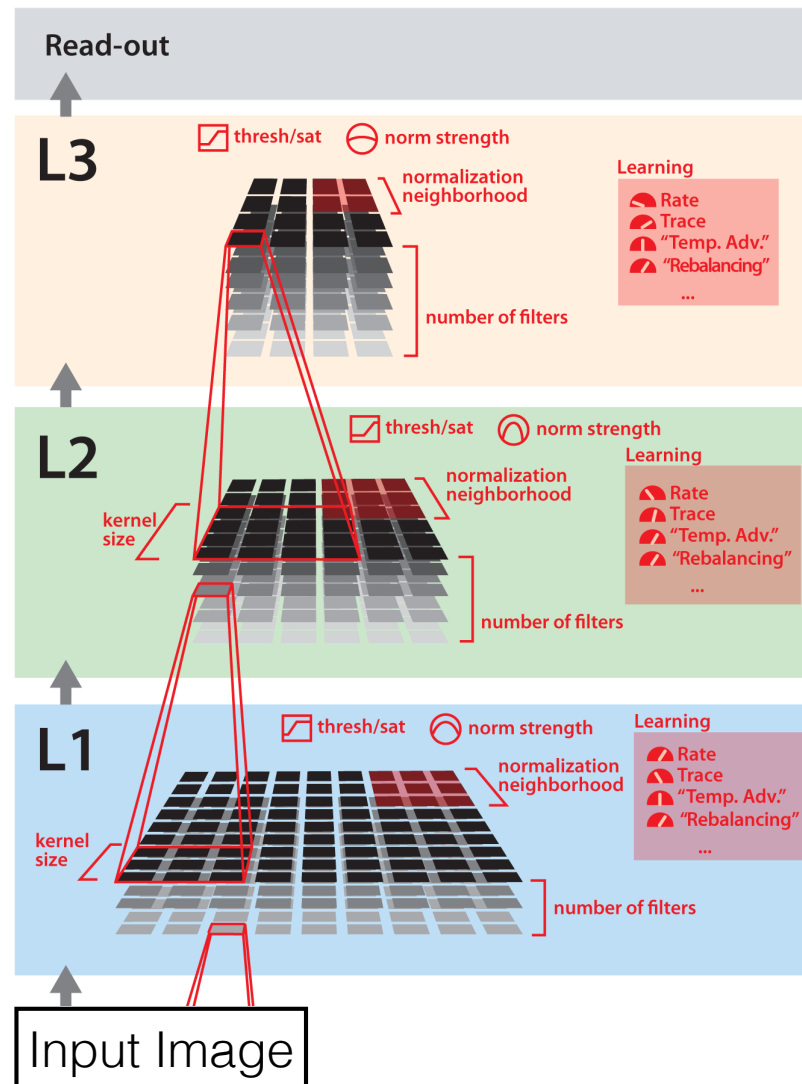
Pinto et al. PLOS
Comp. Bio. 2009.

By analogy to
drug discovery:



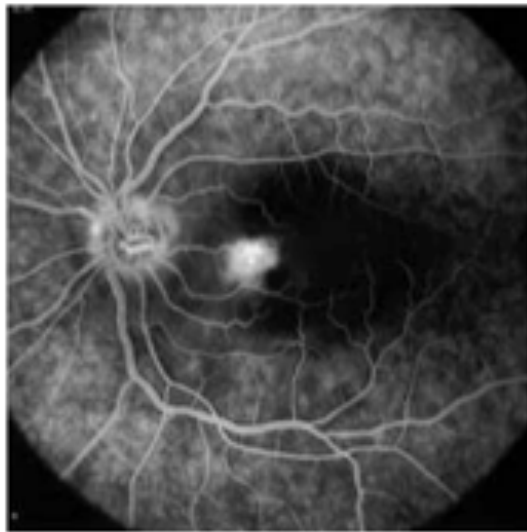
Step 1. Candidate Model Generation

Instantiate a chosen arch. with random hyperparameters

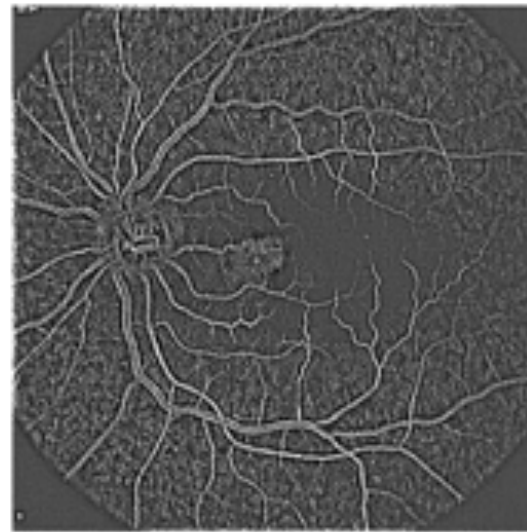


Quick Aside: Features in Computer Vision

What is a Convolution? → Image filtering



(a)



(b)

Step 2. Unsupervised Training

Supervision is expensive and a limiting factor in high-throughput screening

Learn spatiotemporal statistics from video in unsupervised fashion

“Petri Dish” 

A



B



Step 3. Screening

Individual static images supplied as input to each model

Vector of responses from the units of its final layer are taken as the model's "representation"

Training on only one pair of classes

A **Cars vs. Planes**



Step 4. Validation

Best models from screening step are further evaluated against different classes

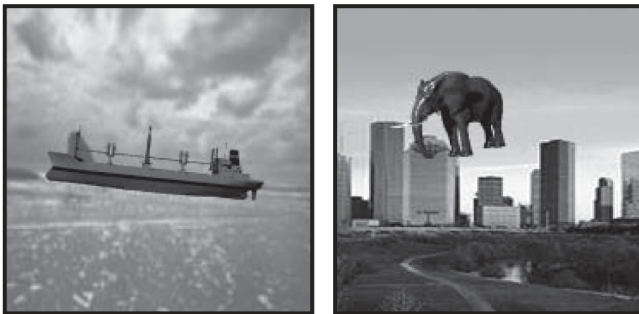
A Cars vs. Planes (validation)



C Synthetic Faces



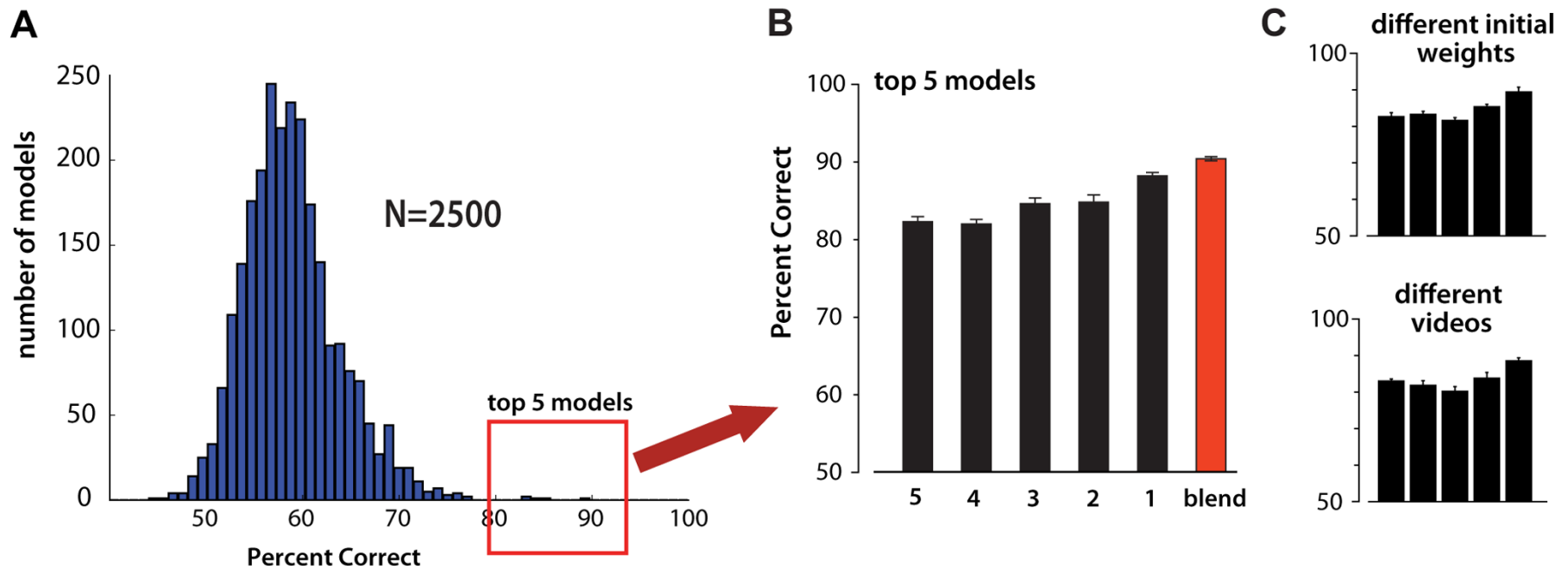
B Boats vs. Animals



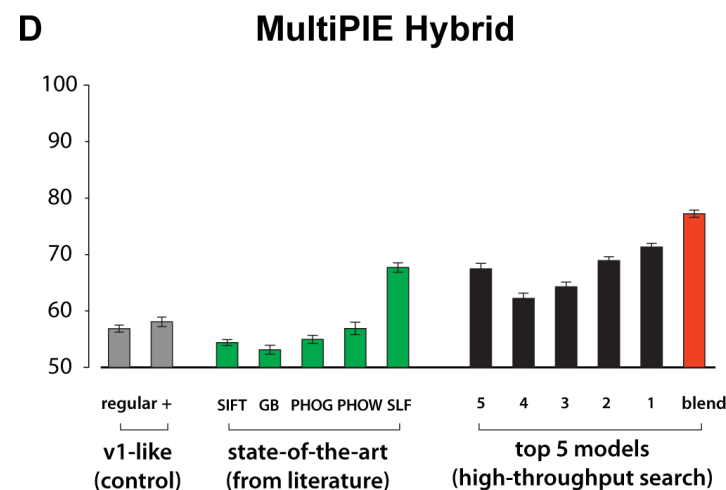
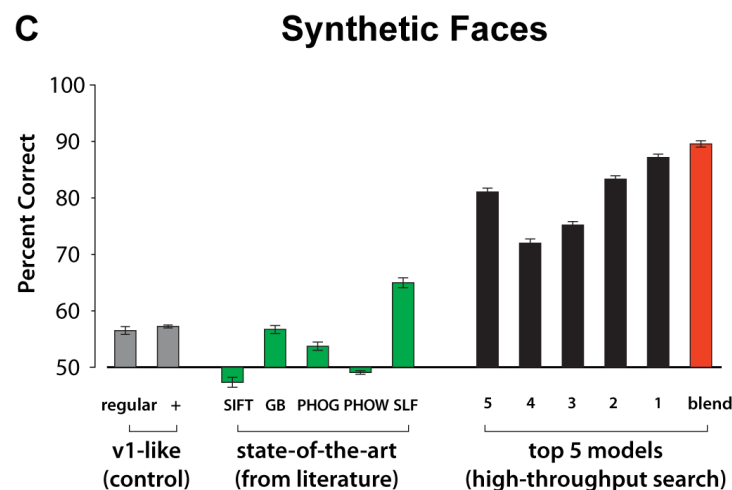
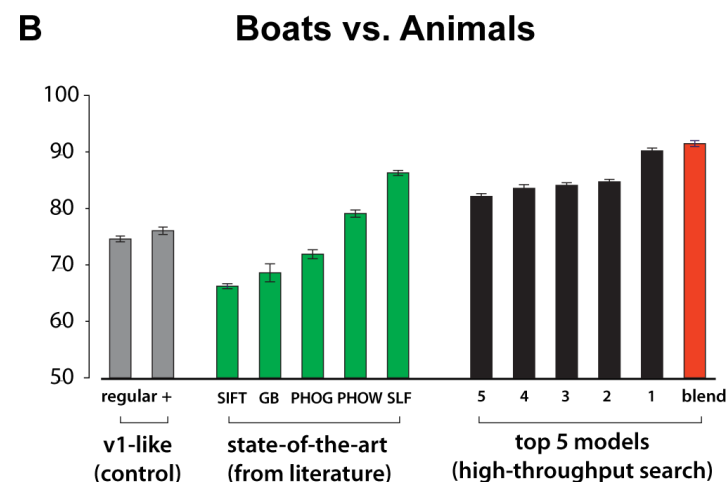
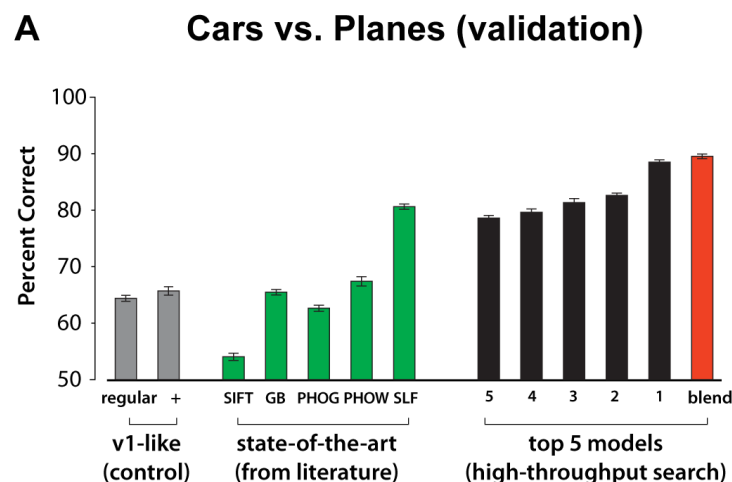
D MultiPIE Hybrid



2,500 Models screened on the “Cars” vs. “Planes” Task



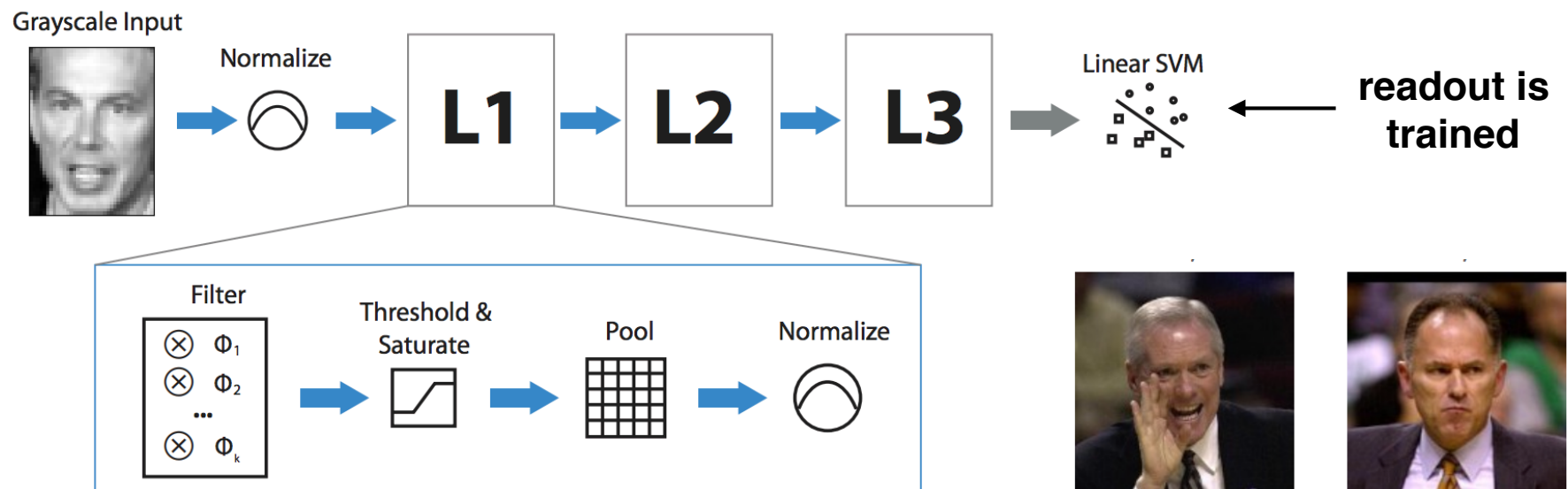
2,500 Models screened on the “Cars” vs. “Planes” Task



This methodology applied to face recognition

Pinto and Cox IEEE FG 2011.

Neat hack: don't train net, simply use random weights and screen

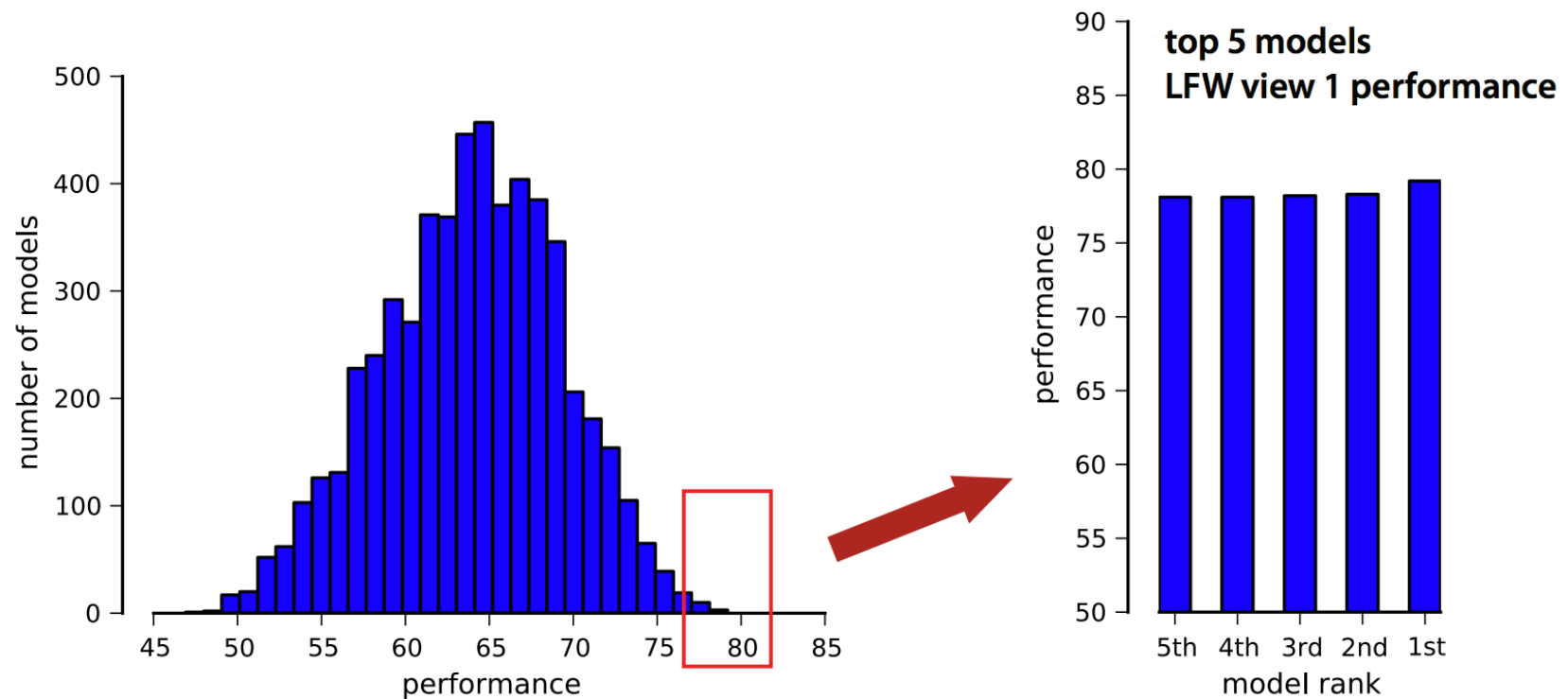


Same or different?

Why does this work?

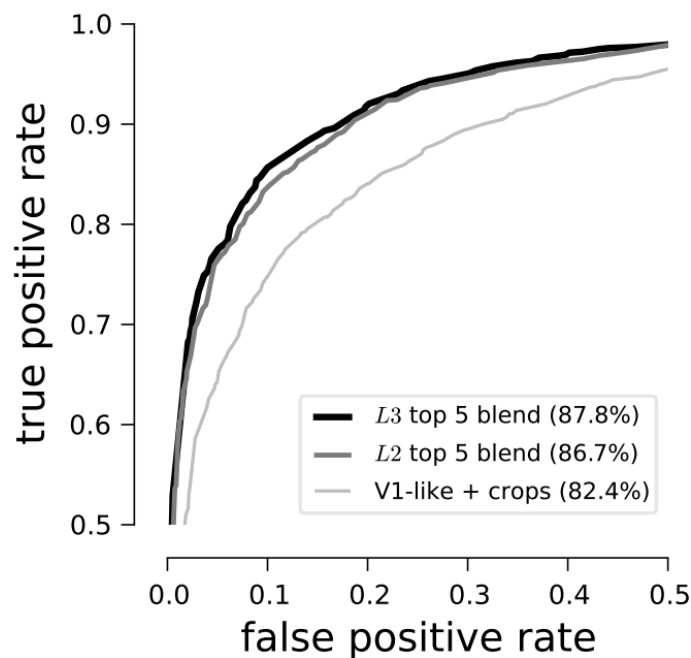
Face Recognition Performance

Labeled Faces in the Wild: Huang et al. 2008

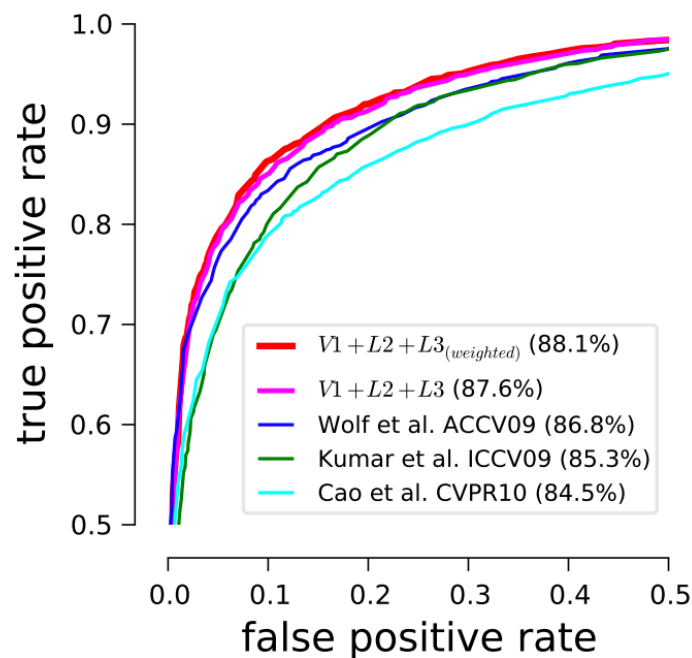


Screening on the LFW View 1 “aligned” set

Face Recognition Performance



(a) within-model-class blends



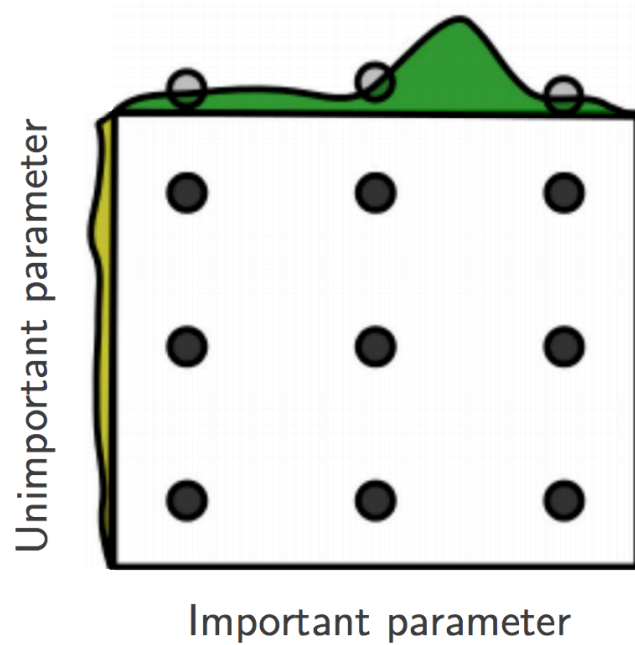
(b) across-model-class blends

| Reference | Kumar et al. ICCV09 [28] | Wolf et al. ACCV09 [27] | Cao et al. CVPR10 [29] | This paper |
|---------------------------|-----------------------------|----------------------------|---------------------------|------------------|
| Mean classification error | 14.7%±1.2 | 13.2%±0.3 | 15.5%±0.5 | 11.9%±0.6 |

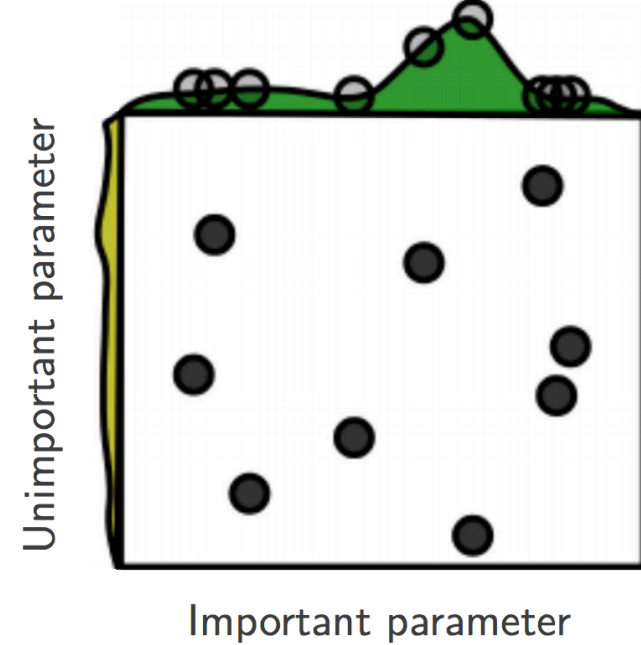
(c) comparison with literature

Random Search

Grid Layout



Random Layout



Bergstra and Bengio JMLR 2012.

Observations on Random Search

- Optimal in the limit (why doesn't this help us?)
- Easy to implement
- Probably works because not all parameters are equally important to tune
 - Grid search spends far too much time on unimportant parameters
- In practice, this is your best strategy for hyper-parameter optimization and model search (for now)

Computational Efficiency of Random Search

- Experiment can be stopped at any point, but trials form a complete experiment
- If extra computational resources become available, new trials can be added without having to adjust a grid
- Every trial can be carried out asynchronously
- If a computer fails, the entire experiment isn't ruined, just the trial

Hyperopt

← → ↻ GitHub, Inc. [US] | <https://github.com/hyperopt/hyperopt> ☆ 🔔 ⬆️ 🔍 | W

Distributed Asynchronous Hyperparameter Optimization in Python <http://hyperopt.github.io/hyperopt>

📄 972 commits

🌿 20 branches

📦 4 releases

👤 45 contributors


📄 View license

Branch: master ▼

New pull request

Find File

Clone or download ▼

| | | |
|---|--|---------------------------------|
|  twolodzko and maxpumperla | Meaningful exception for all optimization trials failed (#501) | Latest commit 762e89f on May 28 |
| 📁 hyperopt | Meaningful exception for all optimization trials failed (#501) | last month |
| 📁 recipes | adding partial sampling notebook | 6 years ago |
| 📁 scripts | linting for mongo worker script | 3 years ago |
| 📄 .gitignore | init pr (#480) | 4 months ago |
| 📄 .gitmodules | FIX: remove pyll submodule | 7 years ago |
| 📄 .travis.yml | init pr (#480) | 4 months ago |
| 📄 LICENSE.txt | ENH: added license file | 7 years ago |
| 📄 README.md | added anaconda badge (#425) | 10 months ago |
| 📄 RELEASE.txt | added RELEASE.txt note about annotated tags | 6 years ago |
| 📄 distribute_setup.py | add distribute_setup | 6 years ago |
| 📄 setup.py | fixing max_queue_len for mongotrials (#506) | last month |

Bergstra et al. ICML 2013.

Hyperopt for Search

- Python library that implements random search
- Free and open source
- Available via a quick pip install
- Makes use of distributed hardware

The way to use hyperopt is to describe:

- The objective function to minimize
- The space over which to search
- The database in which to store all point evaluations of the search
- The search algorithm to use

Define an objective function

(meta-level objective)

```
def objective(args):  
    case, val = args  
    if case == 'case 1':  
        return val  
    else:  
        return val ** 2
```

Define a search space

```
from hyperopt import hp
space = hp.choice('a',
    [
        ('case 1', 1 + hp.lognormal('c1', 0, 1)),
        ('case 2', hp.uniform('c2', -10, 10))
    ])
```

Minimize the objective over the space

```
from hyperopt import fmin, tpe, space_eval
best = fmin(objective, space, algo=random.suggest,
max_evals=100)

print(best)
# -> {'a': 1, 'c2': 0.01420615366247227}
print(space_eval(space, best))
# -> ('case 2', 0.01420615366247227)
```

Search spaces for machine learning classifiers

```
from hyperopt import hp
space = hp.choice('classifier_type', [
    {
        'type': 'naive_bayes',
    },
    {
        'type': 'svm',
        'C': hp.lognormal('svm_C', 0, 1),
        'kernel': hp.choice('svm_kernel', [
            {'ktype': 'linear'},
            {'ktype': 'RBF', 'width': hp.lognormal('svm_rbf_width', 0, 1)},
        ]),
    },
    {
        'type': 'dtree',
        'criterion': hp.choice('dtree_criterion', ['gini', 'entropy']),
        'max_depth': hp.choice('dtree_max_depth',
            [None, hp.qlognormal('dtree_max_depth_int', 3, 1, 1)]),
        'min_samples_split': hp.qlognormal('dtree_min_samples_split', 2, 1, 1),
    },
])
```