Chapel in Cray HPO

CHIUW 2019

June 22nd, 2019

Ben Albrecht

CrayAI HPO

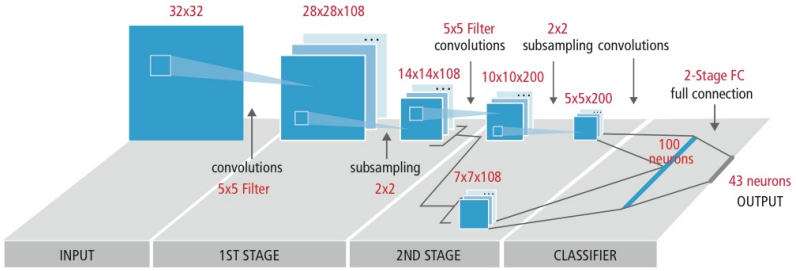
- Hyperparameter optimization framework (HPO)
  - Portable
    - Runs on desktops to clusters to cloud to supercomputers
  - Lightweight
    - Supports multiple ML toolkits
  - Distributed
    - Supports distributed HPO
    - Supports distributed model training
- Backend implemented in Chapel
- User-facing Python interface

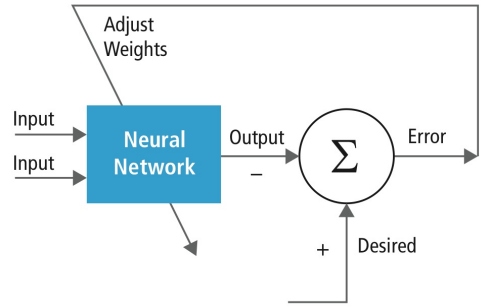
Background: Hyperparameter Optimization

Background: Hyperparameters

- **Model parameters** – internal values in a model determined from data
  - In neural networks: weights (connection, bias)

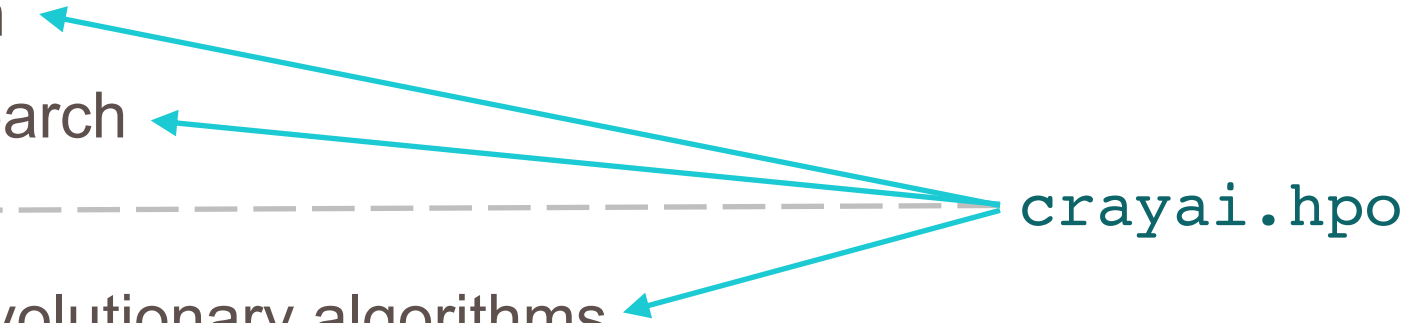
- **Model hyperparameters** – external values to model that influence model capacity
  - In neural networks:
    - **Topology**:
      - Number of neurons in fully connected layers
      - Filters, kernel sizes, convolutional / pooling strides
    - **Training**:
      - Learning rate, batch size, momentum
      - Dropout probability, batch normalization





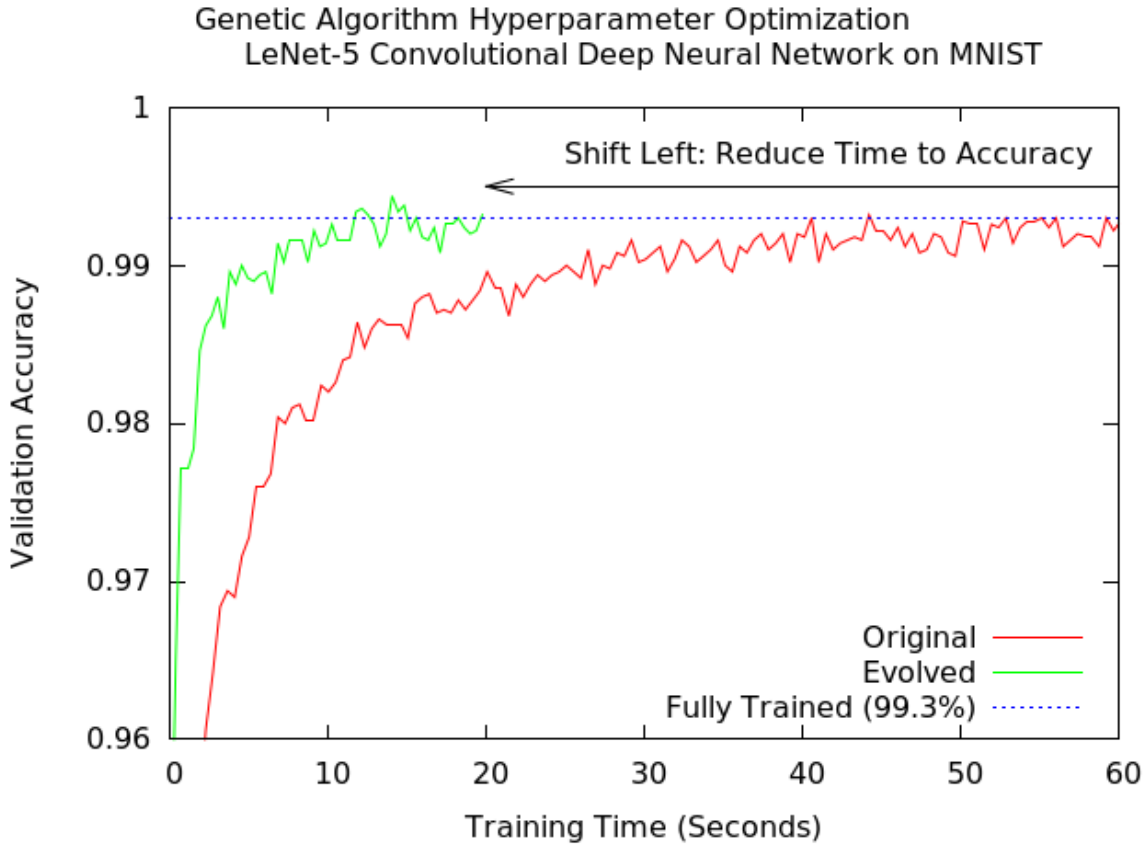
Background: Hyperparameter Optimization

- Manual HPO (by hand)
  - Hyperparameters are selected and tuned manually
  - Guided by intuition and rules of thumb
- Automated HPO
  - Brute-force of entire search space intractable
  - Evaluate a subspace
    - Grid search
    - Random search
    - Bayesian
    - Genetic / evolutionary algorithms



Background: Hyperparameter Optimization

- Finding a good set of hyperparameters can have a big impact:
  - Accuracy
  - Time-to-accuracy
  - Preventing overfitting



CrayAI HPO: Features & Interface

Components of CrayAI HPO workflow

- **Training kernel**
  - Model training program to be optimized
    - Hyperparameters exposed through command line arguments
    - *Figure of merit* exposed through stdout with a unique identifier
  - Can be written in *anything*
    - e.g., python + {TensorFlow, PyTorch, Keras,…}, R, Julia, …
- **HPO driver**
  - Program used to optimize hyperparameters of **training kernel**
  - This program calls the crayai.hpo library
  - Must be written in Python

Training Kernel

```
45
46      parser = argparse.ArgumentParser()
47      parser.add_argument('--lr', '-l', default=0.01, type=float,
48                          help='Learning Rate')
49      parser.add_argument('--dropout', '-d', nargs='?', default=0.5, type=float,
50                          help='drop rate for layers')
51      args = parser.parse_args()
52
```

```
149     # constructing hidden layer 1
150     layer1 = tf.layers.Dense(units=layerSize[0], activation=tf.nn.relu, use_bias=True)
151     layer1Out = tf.layers.dropout(inputs=layer1(inputs=trainInput), rate=FLAGS.dropout)
152
153     # constructing hidden layer 2
154     layer2 = tf.layers.Dense(units=layerSize[1], activation=tf.nn.sigmoid, use_bias=True)
155     layer2Out = tf.layers.dropout(inputs=layer2(inputs=layer1Out), rate=FLAGS.dropout)
```

```
168
169     trainOperation = tf.train.GradientDescentOptimizer(learning_rate=FLAGS.lr).minimize(loss=loss)
170     prediction = tf.argmax(output, 1)
171     evalPrediction = tf.argmax(evalOut, 1)
172
```

```
217
218         print("FoM: %e" % lossVal)
219
```

Expose hyperparameters as command line args

Utilize hyperparameters from command line args

Print figure of merit

HPO Driver

```
1  #!/usr/bin/env python3
2  # encoding: utf-8
3
4  from crayai import hpo
5
6  evaluator = hpo.Evaluator('python source/train_model.py')
7
8  params = hpo.params([["--lr",  0.001, (1e-5, 0.1)],
9                       ["--dropout",  0.5, (0.01, 1)]])
10
11 optimizer = hpo.genetic.Optimizer(evaluator,
12                                    generations=20,
13                                    pop_size=10,
14                                    num_demes=4,
15                                    log_fn='genetic.log')
16
17 optimizer.optimize(params)
18
19 print(optimizer.best_fom)
20 print(optimizer.best_params)
21
```

Import `crayai` module

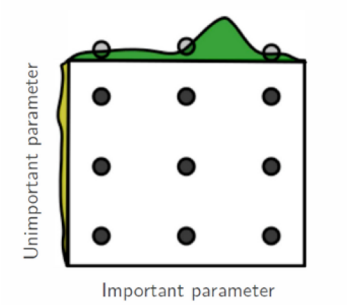
Set model training script

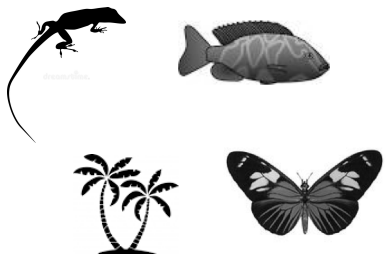
Provide hyperparameter flags, default values, and search space

Setup and run hyperparameter optimizer

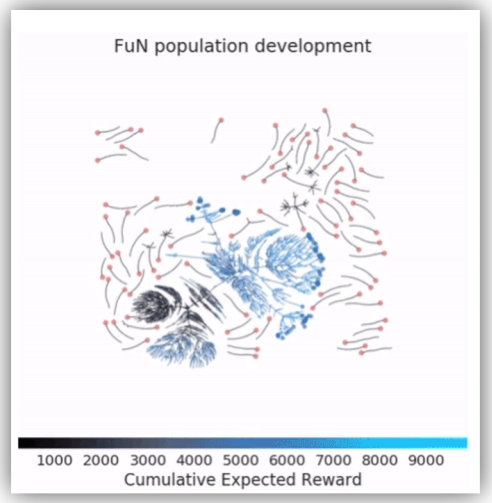
Results stored in optimizer object

CrayAI HPO Feature Overview





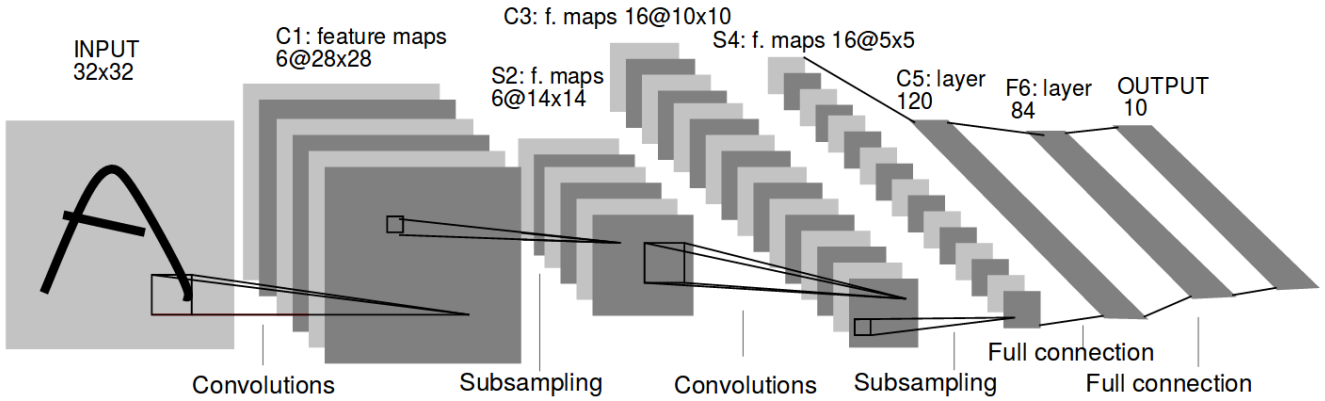
- **Traditional HPO**
  - Grid
  - Random
  - Genetic
- **Schedule HPO**
  - Population Based Training



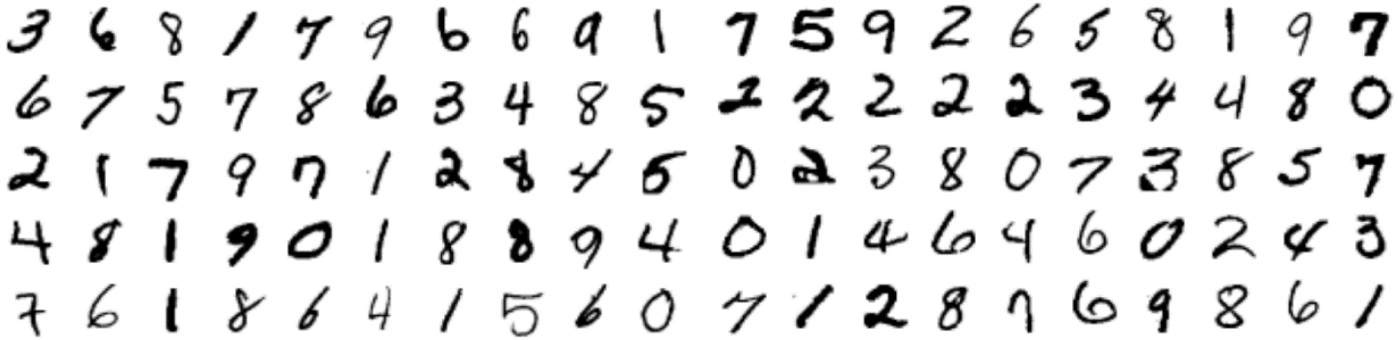
Results & Outlook

Example 1: LeNet and MNIST

- LeNet-5, 7 layers, 5 hidden:

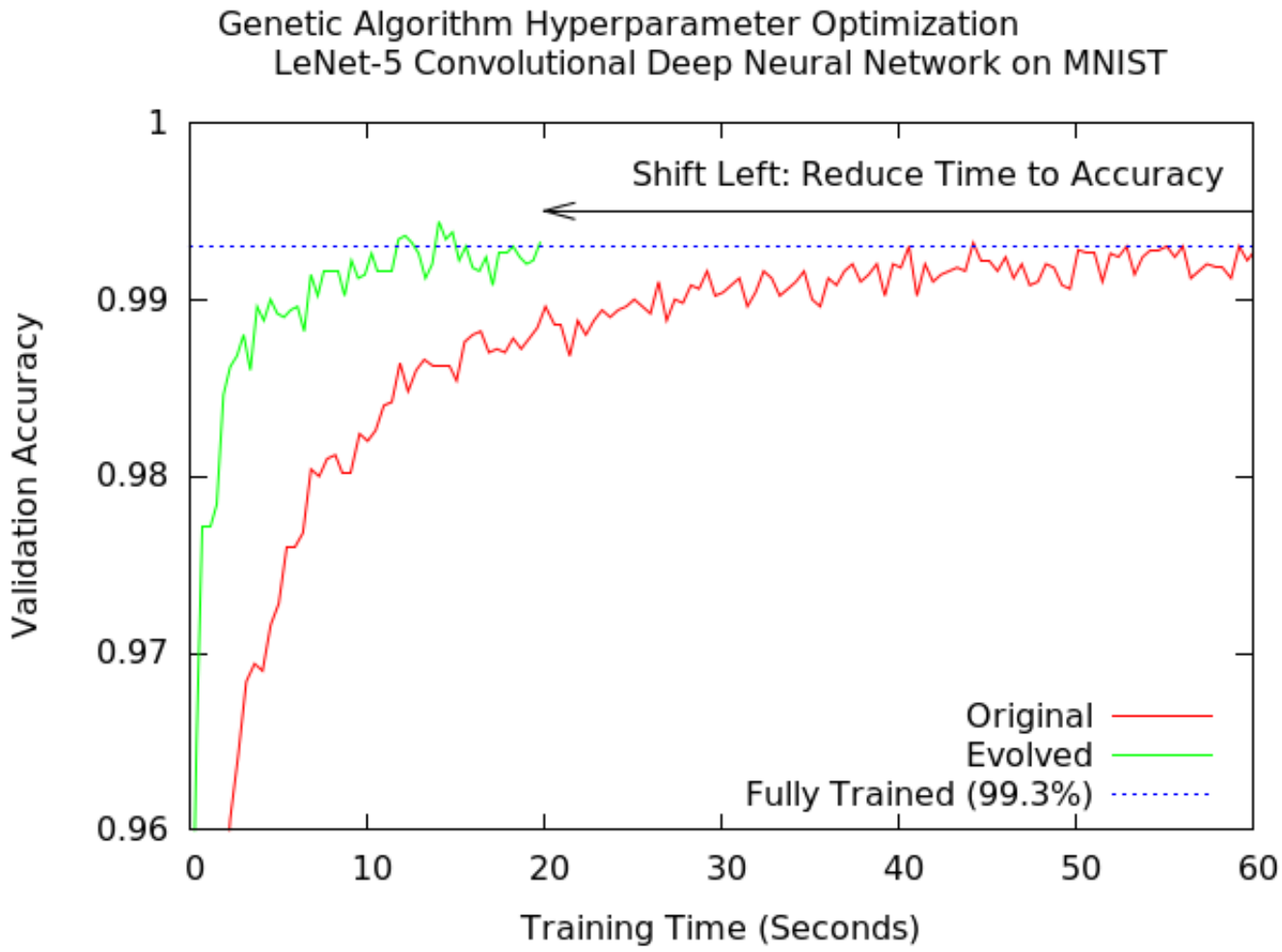


- MNIST, 70k 28x28 greyscale images, 10 classes:



Example: LeNet and MNIST

- HPs Trained:
  - momentum
  - dropout
  - topology
    - c1sz, c1ft
    - c2sz, c2ft
    - fcsz



Why Chapel?

- Chance to put weight on Chapel and demonstrate it is production-ready
- Motivates features in Chapel
  - Random sampling (available in 1.19)
  - Iterator tools (work in progress)
  - Launcher interfaces (work in progress)
- HPO implementation benefits from Chapel's ease of use and modern features
  - Shared-memory parallelism
  - Atomics
  - Interoperability
  - Generics, type inference, memory-management…

What about performance and distributions?

- HPO algorithms are typically bottlenecked by evaluation
  - Majority of time spent on actual model training (kernel) and I/O
  - Each iteration is embarrassingly parallel
- CrayAI HPO interfaces with launchers rather than using distributed Chapel
  - Chapel assumes it owns a locale and could compete against model training
  - Launching a subprocess on a subset of locales is not yet supported
  - Current interface can create or use an existing allocation
- Distributed Chapel and performance will matter for future work

HPO as a Cray Product

- First Cray product developed in Chapel
  - Available in Urika XC 1.2 and Urika CS 1.1
  - Accessible through analytics module or crayai module:

```
> module load crayai
> module load analytics
```

- Receiving contributions from a growing list of Crayons outside of Chapel
- Being tested by Cray customers and partners

Chapel in a Cray Product

- Ramping up non-Chapel developers has been easy
- Language-breaking changes do occur
  - They have been relatively painless
  - Typically mechanical fixes
- We pro-actively test against last release and master branch to catch changes


Ongoing Work

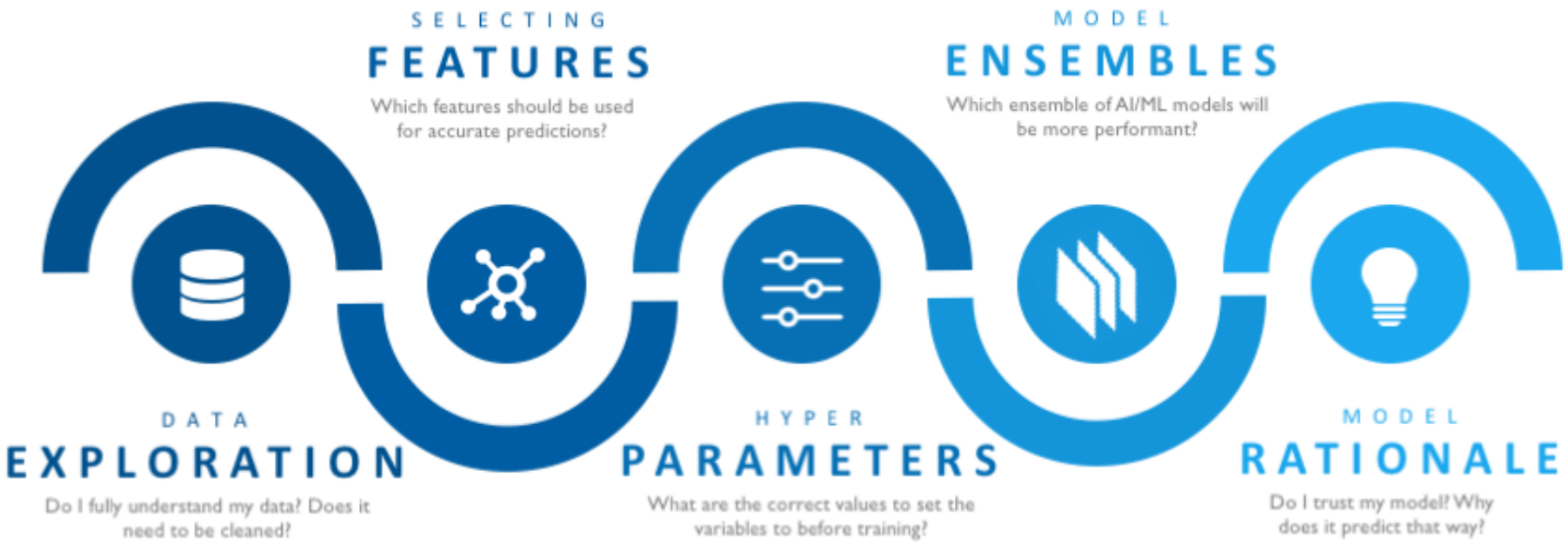
Next Steps

- Continue improving HPO features and stability
  - Support more launchers
  - Improved Jupyter integration
  - Many feature requests from users
- Implement new strategies
  - Bayesian (PR currently under review)
- Open source
- Implement other AI workflow components

Next Steps: Bigger Picture

- Develop other pieces of the AI workflow

crayai.fs



crayai.hpo

THANK YOU

QUESTIONS?

balbrecht@cray.com

github.com/ben-albrecht

References

- Population Based Training of Neural Networks
  - https://arxiv.org/pdf/1711.09846.pdf
- Recombination of Artificial Neural Networks
  - https://arxiv.org/abs/1901.03900
- Random Search for Hyper-parameter Optimization
  - http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf
- The MNIST Database of Handwritten Digits (MNIST Dataset)

  http://yann.lecun.com/exdb/mnist/
- Gradient Based Learning Applied to Document Recognition (LeNet CNN)

  http://www.dengfanxin.cn/wp-content/uploads/2016/03/1998Lecun.pdf