AAAI 2022 Tutorial on Neural Network Verification
Part III: *α,β*-CROWN for Complete Verification

Huan Zhang (CMU), Kaidi Xu (Drexel), **Shiqi Wang (Columbia)**, and Cho-Jui Hsieh (UCLA)

Feb 23, 2022

α,β-CROWN: a scalable and efficient neural network verifier
https://abcrown.org

CROWN

Winner of International Verification of Neural Networks Competition (VNN-COMP'21)

Slides and code demos available at neural-network-verification.com

Overview: Branch and Bound for Complete Verification

CROWN/α-CROWN with auto_LiRPA is Incomplete verifier
+ Efficient
- not complete: cannot improve the verification with more time=> limited verification instances

Complete verification: guarantee to prove the properties given sufficient time
Solution: improve the verification iteratively with Branch and Bound until verified

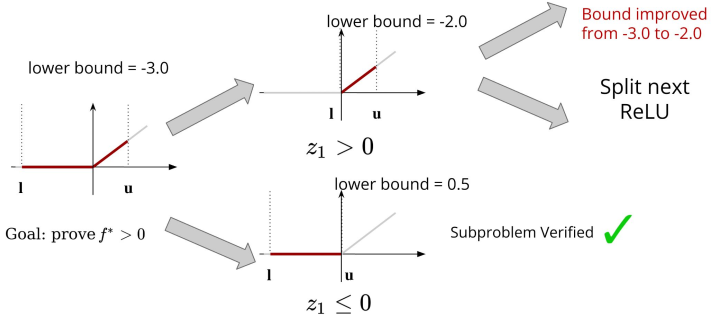
α,β-CROWN: branch and bound with β-CROWN and α-CROWN
+ Complete: can verify more instances
- More time required

Models	**CROWN Verified Acc**	**CROWN Avg. Time**	**α,β-CROWN Verified Acc**	**α,β-CROWN Avg. Time**
MNIST CNN-A-Adv	1.0%	0.1s	70.5%	21.1s
CIFAR CNN-A-Adv	35.5%	0.6s	44.0%	5.8s

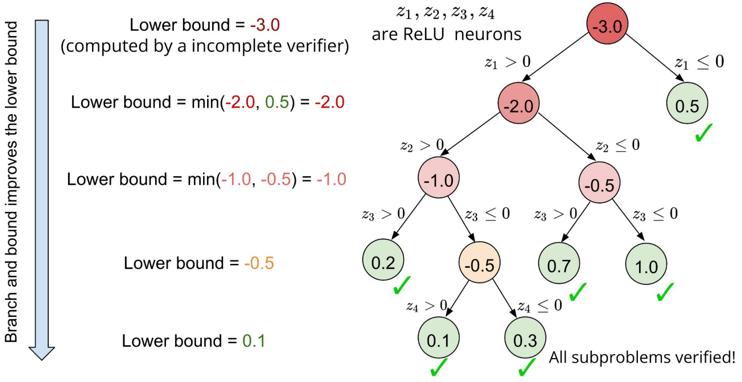
Overview: Branch and Bound for Complete Verification

Branch and Bound (BaB) for Neural Network Verification

Strategy: Solve a convex relaxation to get a lower bound**,** and iteratively improve it by **splitting** ReLU neurons
Step 1 (branch): Create new subdomains by splitting ReLUs
Step 2 (bound): Lower bound each subdomain with ***β*-CROWN**. If bounds for some subdomains are < 0, back to step 1.
Our goal: Improve BaB for NN verification with highly parallel branching, and more efficient and tighter bounding on GPUs



ReLU Split for Branching



BaB Search Tree

Overview: Benefits of our *α,β*-CROWN

Benefits of our algorithms:

- **Efficient bound propagation with split constraints** from BaB, get tight bounds without relying on slow LP solvers on CPU.
- **An optimizable procedure tightens the bounds,** with the same or better power compared to typical LP verifier.
- **Massive parallelization in BaB on GPUs** allows us to quickly explore a large number of subdomains.

http://abcrown.org

Winner of VNN-COMP21
State of the art performance!


2021 VNN-COMP Award

Verification of Neural Networks Competition Top Highest Score Award
Verification of Neural Networks Competition Category Winner for
Verivital, eran, cifar10_resnet, nn4sys, marabou-cifar10

Co-located with the 33rd International Conference on Computer-Aided Verification (CAV'21)
Alpha-Beta-CROWN

Huan Zhang*[1], Kaidi Xu*[2], Shiqi Wang*[3], Zhouxing Shi[4], Yihan Wang[4], Xue Lin[2], Suman Jana[3], Cho-Jui Hsieh[4], Zico Kolter[1]
[1]Carnegie Mellon University, [2]Northeastern University, [3]Columbia University, [4]Univeristy of California, Los Angeles

The 2nd International Verification of Neural Networks Competition (VNN-COMP'21) is a formal methods and verification competition for neural networks, held in conjunction with the 4th Workshop on Formal Methods for ML-Enabled Autonomous Systems (FoMLAS'21) and affiliated with the 33rd International Conference on Computer-Aided Verification (CAV'21), which is dedicated to the advancement of the theory and practice of computer-aided formal analysis methods for hardware and software systems.

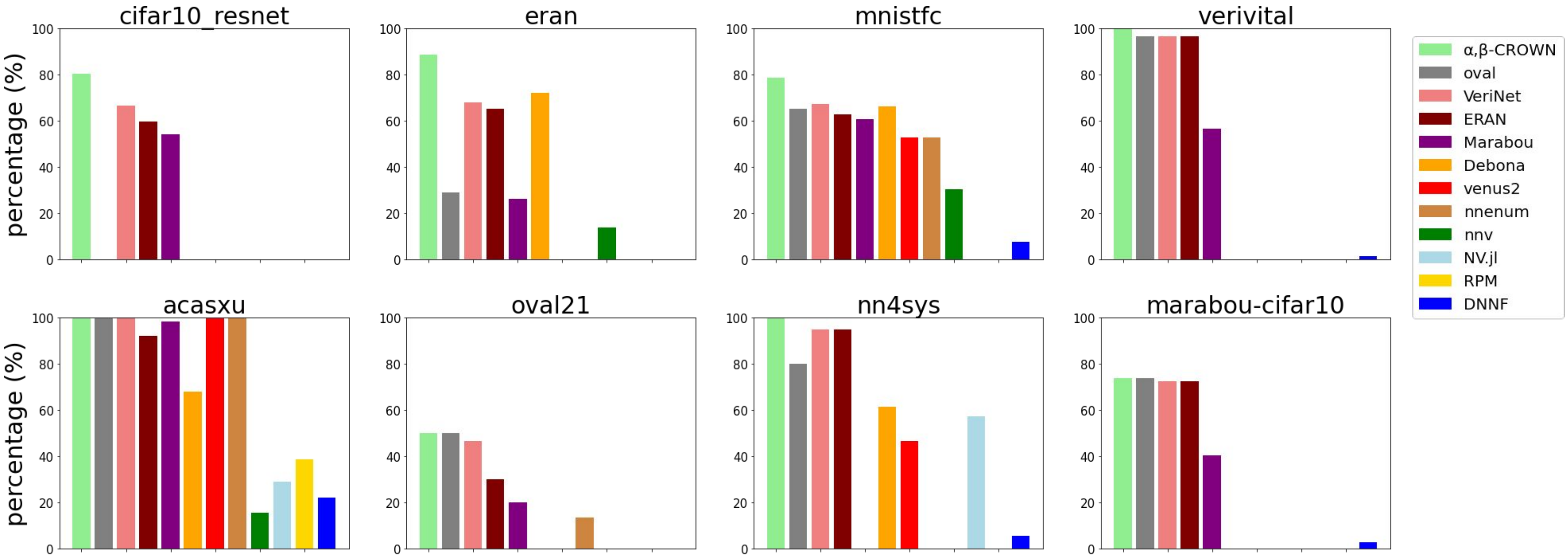
Awards for VNN-COMP'21 were graciously supported by a gift from the Lu Jin Family Foundation.

Stanley Bak, Changliu Liu, & Taylor T. Johnson July 19, 2021

Stony Brook University
Vanderbilt University
CAV

VNN-COMP 2021 Results

***α,β*-CROWN verifies the most number of instances on all benchmarks**



More results available at VNN-COMP 2021 report: **https://arxiv.org/abs/2109.00498**

Overview: Benefits of our *α,β*-CROWN

Benefits of our tool:
- State-of-the-art performance
- Better integration with PyTorch: Able to directly load PyTorch/ONNX models
- **Friendly APIs**: Easy to use with config file
- **Easy customization**: many customization examples

Just that simple!
Define your model + properties
=> Run

Winner of VNN-COMP21
State of the art performance!

Usage: Arguments and Config Files

Example config file for mnist_cnn_a_adv:

python robustness_verifier.py
--config exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

Main file: robustness_verifier.py
--config: feed in customized config file

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

Usage: Arguments and Config Files

Example config file for mnist_cnn_a_adv:

python robustness_verifier.py
--config exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

Main file: robustness_verifier.py
--config: feed in customized config file

All parameter documentation available at:
https://github.com/huanzhang12/alpha-beta-CROWN/blob/main/docs/robustness_verifier_all_params.yaml

Let's take a quick look at the documentation of parameters!

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

Usage: Arguments and Config Files

Arguments are equivalent to config files => easy to customize!

python robustness_verifier.py --config exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml --start 0 --end 100

Equivalent

All arguments are defined in arguments.py
Help: python robustness_verifier.py -h

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

Usage: Arguments and Config Files

Model configuration
model name: the defined model architecture
model path: the path of the pretrained model

Models predefined in model_defs.py

```
def mnist_cnn_4layer():
    # mnist_cnn_a
    return nn.Sequential(
        nn.Conv2d(1, 16, (4,4), stride=2, padding=1),
        nn.ReLU(),
        nn.Conv2d(16, 32, (4,4), stride=2, padding=1),
        nn.ReLU(),
        Flatten(),
        nn.Linear(1568, 100),
        nn.ReLU(),
        nn.Linear(100, 10),
    )
```

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

Usage: Arguments and Config Files

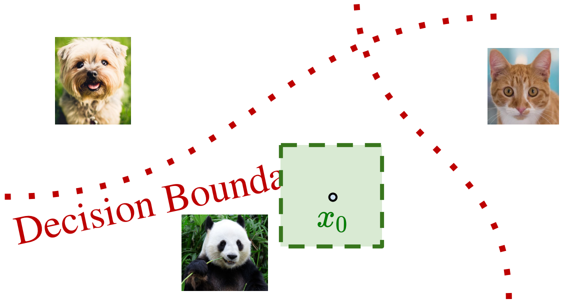
Dataset configuration
dataset: testing data
start & end: the start and end index for verification samples
std & mean: normalization for the data

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

Usage: Arguments and Config Files

Robustness property specification configuration
epsilon: the allowed perturbation range
norm: Linf norm (default), also support L2, L1
type: norm (default), bound (element wise bounds for customization)



Robustness verification

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

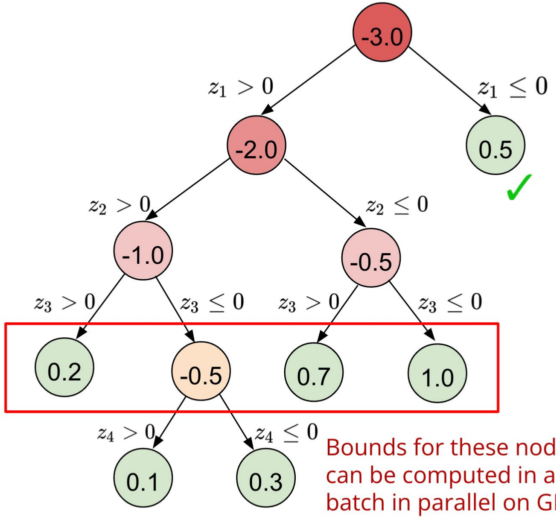
Usage: Arguments and Config Files

β-CROWN configuration

batch_size: the maximal batch_size allowed

Larger => better performance

Be careful here, recommend to fit the GPU memory, too large batch_size will cause out of GPU memory error.



Bounds for these nodes can be computed in a batch in parallel on GPUs

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

Usage: Arguments and Config Files

BaB configuration

Timeout: total timeout threshold for each verification instance

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

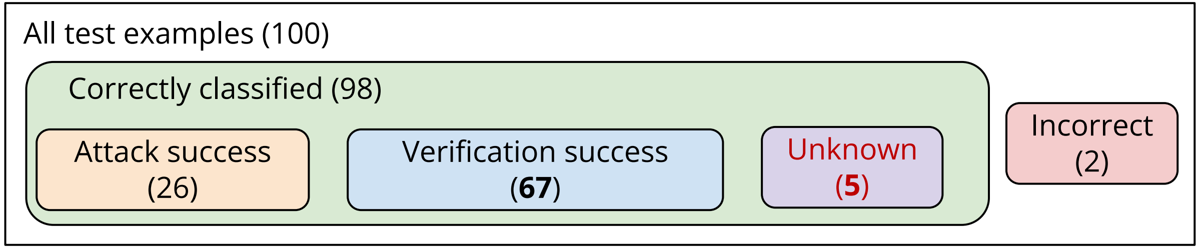
Usage: Outputs by *α,β*-CROWN

Output sample

python robustness_verifier.py --config exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml --end 100

```
number of correctly classified examples: 98
incorrectly classified idx (total 2): [33, 73]
attack success idx (total 26): [7, 8, 9, 15, 18, 20, 24, 38, 43, 44, 45, 53, 59, 61, 62, 63, 64, 65, 77,
78, 80, 84, 87, 92, 95, 97]
attack_success rate: 0.26
verification success idx (total 67): [0, 1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14, 16, 17, 19, 21, 23, 25, 26
, 27, 28, 29, 30, 31, 32, 34, 35, 37, 39, 40, 42, 46, 47, 48, 49, 50, 51, 52, 54, 55, 56, 57, 58, 60, 67,
 68, 69, 70, 71, 74, 75, 76, 79, 81, 82, 83, 85, 86, 88, 89, 90, 91, 93, 94, 96, 98, 99]
verification failure idx (total 5): [22, 36, 41, 66, 72]
final verified acc: 67.0%[100]
verifier is called on 72 examples.
total verified: 67
mean time [cnt:72] (excluding attack success): 31.16633384095298
mean time [cnt:98] (including attack success): 23.26871461527688
```

Total: 100
Correct: 98
Verified unsafe: 26
Verified safe: 67
unknown: 5
Avg. Time: 23.26s



Tutorial Example1: MNIST CNN-A-Adv

Colab demo1:

PaperCode.cc/a-b-CROWN-Tutorial-MNIST

exp_configs/tutorial_examples/mnist_cnn_a_adv.yaml

```yaml
general:
  mode: verified-acc
model:
  name: mnist_cnn_4layer
  path: models/sdp/mnist_cnn_a_adv.model
data:
  dataset: MNIST
  start: 0
  end: 100
  std: [1.]
  mean: [0.]
specification:
  epsilon: 0.3
  norm: .inf
attack:
  pgd_restarts: 50
solver:
  beta-crown:
    batch_size: 1024
    iteration: 20
bab:
  timeout: 180
```

Usage: Customized Models

Customized model:
This is a customized PyTorch model;
Please replace with any of your model

One can even predefine the weights explicitly for simple test cases

Samples in custom_model_data.py

```python
def simple_conv_model(in_channel, out_dim):
    """Simple Convolutional model."""
    model = nn.Sequential(
        nn.Conv2d(in_channel, 16, 4, stride=2, padding=0),
        nn.ReLU(),
        nn.Conv2d(16, 32, 4, stride=2, padding=0),
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(32*6*6,100),
        nn.ReLU(),
        nn.Linear(100, out_dim)
    )
    return model

def two_relu_toy_model(in_dim=2, out_dim=2):
    """A very simple model, 2 inputs, 2 ReLUs, 2 outputs"""
    model = nn.Sequential(
        nn.Linear(in_dim, 2),
        nn.ReLU(),
        nn.Linear(2, out_dim)
    )
    """[relu(x+2y)-relu(2x+y)+2, 0*relu(2x-y)+0*relu(-x+y)]"""
    model[0].weight.data = torch.tensor([[1., 2.], [2., 1.]])
    model[0].bias.data = torch.tensor([0., 0.])
    model[2].weight.data = torch.tensor([[1., -1.], [0., 0.]])
    model[2].bias.data = torch.tensor([2., 0.])
    return model
```

Usage: Customized Dataset

Customize perturbation set:
Input: defined as you want
Output (fixed):
- X, labels: data and labels
- data_max, data_min: max and min values of data
- eps: normalized epsilon range

Customized CIFAR: We provide official CIFAR dataset in utils.py while one can easily customize the dataset this way

Samples in custom_model_data.py

```
def simple_cifar10(eps):
    """Example dataloader. For MNIST and CIFAR you can actually use existing ones in utils.py."""
    assert eps is not None
    database_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'datasets')
    # You can access the mean and std stored in config file.
    mean = torch.tensor(arguments.Config["data"]["mean"])
    std = torch.tensor(arguments.Config["data"]["std"])
    normalize = transforms.Normalize(mean=mean, std=std)
    test_data = datasets.CIFAR10(database_path, train=False, download=True,\
            transform=transforms.Compose([transforms.ToTensor(), normalize]))
    # Load entire dataset.
    testloader = torch.utils.data.DataLoader(test_data,\
            batch_size=10000, shuffle=False, num_workers=4)
    X, labels = next(iter(testloader))
    # Set data_max and data_min to be None if no clip. For CIFAR-10 we clip to [0,1].
    data_max = torch.reshape((1. - mean) / std, (1, -1, 1, 1))
    data_min = torch.reshape((0. - mean) / std, (1, -1, 1, 1))
    if eps is None:
        raise ValueError('You must specify an epsilon')
    # Rescale epsilon.
    ret_eps = torch.reshape(eps / std, (1, -1, 1, 1))
    return X, labels, data_max, data_min, ret_eps
```

Usage: Customized Dataset

Customize perturbation set:
Input: defined as you want
Output (fixed):
- X, labels: data and labels
- data_max, data_min: max and min values of data
- eps: normalized epsilon range

Customized box data: One can define arbitrary data with wanted perturbation range this way

Samples in custom_model_data.py

```
def simple_box_data():
    """a customized box data: x=[-1.5, 1], y=[-1, 1.5]"""
    X = torch.tensor([[0., 0.]]).float()
    labels = torch.tensor([0]).long()
    # customized element-wise upper bounds
    data_max = torch.tensor([[1., 1.5]]).reshape(1, -1)
    # customized element-wise lower bounds
    data_min = torch.tensor([[-1.5, -1.]]).reshape(1, -1)
    eps = None
    return X, labels, data_max, data_min, eps
```

Usage: Customized Verification

Customize config file:
Customized model: simple_conv_model
Customized dataset: simple_cifar10

Sample in tutorial_examples/custom_cifar_data_example.yaml

```yaml
general:
  mode: verified-acc
model:
  # Use the simple_conv_model() model in "custom_model_data.py".
  name: Customized("custom_model_data", "simple_conv_model", in_channel=3, out_dim=10)
  path: models/eran/cifar_conv_small_pgd.pth
data:
  # Use the cifar10() loader in "custom_model_data.py".
  dataset: Customized("custom_model_data", "simple_cifar10")
  mean: [0.4914, 0.4822, 0.4465]
  std: [0.2023, 0.1994, 0.201]
specification:
  epsilon: 0.00784313725  # 2./255.
attack:
  pgd_restarts: 100
solver:
  beta-crown:
    batch_size: 2048
    iteration: 20
bab:
  max_domains: 5000000
  timeout: 300
```

Tutorial Example 2: Customized_CIFAR_DATA

Colab demo2:

PaperCode.cc/a-b-CROWN-Tutorial-Custom

Sample in tutorial_examples/custom_cifar_data_example.yaml

```yaml
general:
  mode: verified-acc
model:
  # Use the simple_conv_model() model in "custom_model_data.py".
  name: Customized("custom_model_data", "simple_conv_model", in_channel=3, out_dim=10)
  path: models/eran/cifar_conv_small_pgd.pth
data:
  # Use the cifar10() loader in "custom_model_data.py".
  dataset: Customized("custom_model_data", "simple_cifar10")
  mean: [0.4914, 0.4822, 0.4465]
  std: [0.2023, 0.1994, 0.201]
specification:
  epsilon: 0.00784313725  # 2./255.
attack:
  pgd_restarts: 100
solver:
  beta-crown:
    batch_size: 2048
    iteration: 20
bab:
  max_domains: 5000000
  timeout: 300
```

Usage: Customized Verification

Customize config file:
Customized model: two_relu_toy_model
Customized dataset: simple_box_data
Specification type: bound (lp-norm as default)

Sample in tutorial_examples/custom_box_data_example.yaml

```yaml
general:
  mode: verified-acc
model:
  # Use the two_relu_toy_model() model in "custom_model_data.py".
  name: Customized("custom_model_data", "two_relu_toy_model", in_dim=2, out_dim=2)
data:
  # Use the simple_box_data() loader in "custom_model_data.py".
  dataset: Customized("custom_model_data", "simple_box_data")
  num_outputs: 2
specification:
  # element-wise perturbation bound assignment
  type: bound
attack:
  pgd_order: skip
solver:
  beta-crown:
    batch_size: 2048
    iteration: 20
bab:
  timeout: 30
  branching:
    method: fsb
```

Tutorial Example 3: Customized_simple_box_data

Colab demo3:

PaperCode.cc/a-b-CROWN-Tutorial-Custom

Sample in tutorial_examples/custom_box_data_example.yaml

```yaml
general:
  mode: verified-acc
model:
  # Use the two_relu_toy_model() model in "custom_model_data.py".
  name: Customized("custom_model_data", "two_relu_toy_model", in_dim=2, out_dim=2)
data:
  # Use the simple_box_data() loader in "custom_model_data.py".
  dataset: Customized("custom_model_data", "simple_box_data")
  num_outputs: 2
specification:
  # element-wise perturbation bound assignment
  type: bound
attack:
  pgd_order: skip
solver:
  beta-crown:
    batch_size: 2048
    iteration: 20
bab:
  timeout: 30
  branching:
    method: fsb
```

Contributors to *α,β*-CROWN

α,β-CROWN Team Members: Huan Zhang* (CMU), Kaidi Xu* (Northeastern University), Shiqi Wang* (Columbia University), Zhouxing Shi (UCLA), Yihan Wang (UCLA), Xue Lin (Northeastern University), Suman Jana (Columbia), Cho-Jui Hsieh (UCLA), Zico Kolter (CMU)
(**Equal contribution*)

Thank you!

α,β-CROWN Verification Tool (PyTorch):

Our tool is available at https://abcrown.org
Please star⭐ our github repo if you find it useful!😊

Contact:

Huan Zhang: huan@huan-zhang.com
Kaidi Xu: kx46@drexel.edu
Shiqi Wang: tcwangshiqi@cs.columbia.edu