DISSECTOR: Input Validation for Deep Learning Applications by Crossing-layer Dissection

Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, Jian Lu

State Key Lab for Novel Software Technology
Department of Computer Science and Technology
Nanjing University, China

July 2020
DL applications suffer from runtime reliability issues

- DL applications are gaining popularities nowadays
- However, due to their imperfect accuracy in practice, DL applications may make wrong behavior decisions (causing runtime reliability issues)
  - Might even lead to some tremendous disasters in safety-critical scenarios
Root in its difference from traditional applications

- Traditional application
  - Generally with a clear specification
  - Easy to identify its wrong behaviors during its deployment

- DL application (suffering from reliability issues)
  - Hard to examine its behavior
  - Due to its natural imperfection in accuracy, wrong decisions tend to occur, and hard to be identified during its deployment

How to assure such DL application’s reliability specifically?
To do so, we consider a fault tolerance way

• How to allow a DL application to still behave well even been fed by some unexpected inputs during its deployment?

• That is, for such a DL application, how can we identify whether an input is within the application’s handling capability or not?
  • Convert to a typical input validation problem for DL application (kernel: DL model)

  Within the application’s handling capability or not?

  Considering its handling capability

  If within, we expect it to work(output) well

  If beyond, we isolate it to avoid wrong decisions
• In other words, for a pre-trained DL model (kernel part) in a deployed DL application, we hypothesize that it has its own **capability scope**

Inputs that are beyond model capability

**Dramatically decreased accuracy**

**Beyond-input**

Inputs that are within model capability

**Unknown capability scope**

**Within-input**

How to distinguish within-/beyond-inputs at runtime?
Problem clarification and existing practice

• Our research problem (a typical input validation problem)
  • How to distinguish within-/beyond-inputs effectively and efficiently at runtime for a DL application?

• Existing practice on similar input validation questions (OOD, adversarial, etc.)
  • OOD identification by natural confidence scores\(^1\)[\(^2\]
    • May be misleading, insufficient effectiveness
  • Adversarial identification by mutation behaviors\(^3\]
    • Not efficient, thus inapplicable for large, complex DL subjects
  • Corner case identification by sample distance measurements\(^4\]
    • Time-consuming, repeated training sample distance calculation
  • ......
Our idea: to achieve nice effectiveness and efficiency

• Request for a lightweight and effective heuristic idea:
  • Our heuristic idea: *a within-input should correspond to an increasingly consistent prediction process, otherwise a beyond-input it is.*
    • Based on an input’s prediction process, thus very lightweight for examination
    • *Characteristics of the prediction process give us a natural hint on whether the prediction is trustable, in other word, whether the input sample is within or not*

• Look into a daily scene: human recognition scene
  • Who is the coming guy? Anna? Bonnie? Celine?
Which recognition is more trustable?

- Fluctuated, contradictory to each other, more confused

- Smooth, eventually consistent to each other, more trustable
Recall our heuristic idea

• Our heuristic idea: *a within-input should correspond to an increasingly consistent prediction process, otherwise a beyond-input it is.*

• Echoing such more trustable recognition
  • One naturally gains *increasing confidence* to the recognition with a closer distance

*Smooth, eventually consistent to each other, more trustable*

The closer with more features for recognition the *more confidence on “B”*
Map human recognition to DL prediction scene

• Both seemingly “step-by-step” guess
Our idea in DL prediction scene

• Recall our heuristic idea in this scene

• In some of our study setting, over 95% high-accurate samples could similarly share such increasingly consistent guess profile
DISSECTOR overview: two kernel tasks

1. How to obtain the guess profile?
   - “stop” “U-turn” “stop” “U-turn” “stop”

2. How to measure the guess consistency?
   - Beyond → Within
   - 0 → 1
DISSECTOR overview: two kernel tasks

1. How to obtain the guess profile?

   - Guess profile 1: “stop” “U-turn” “stop” “U-turn” “stop”

2. How to measure the guess consistency?

   - Measure of consistency: Beyond Within

   - Example: $0 \leftrightarrow 1$
How to obtain the guess profile?

- **Final guess**: straightforward

- **Intermediate guess**: need some preparations first
  - Offline dissect the deployed DL model into a list of partial models, i.e., sub-models for obtaining each intermediate guess for each layer (once-for-all)
  - Trained under training data (no need any more in DISSECTOR’s latter use)

Then, for any sample, use the list of sub-models to obtain intermediate guesses layer by layer towards the final guess, thus getting the guess profile.
DISSECTOR overview: two kernel tasks!

1. How to obtain the guess profile?

   "stop"  "U-turn"  "stop"  "U-turn"  "stop"

2. How to measure the guess consistency?

   0 1
   Beyond  Within
How to measure the guess consistency?

- For each intermediate guess, measure its uniqueness in this specific layer.
- For the whole guess profile, combine all intermediate guesses’ uniqueness scores towards a final score for measuring the guess consistency.
How to measure the guess consistency?

• For each intermediate guess, measure its uniqueness in this specific layer.

• For the whole guess profile, combine all intermediate guesses’ uniqueness scores towards a final score for measuring guess consistency.

Generally, this uniqueness score represents how much this intermediate guess supports the final guess (i.e., the prediction).

Intermediate guess = final guess

Intermediate guess ≠ final guess
How to measure the guess consistency?

• For each intermediate guess, measure its uniqueness in this specific layer

• For the whole guess profile, combine all intermediate guesses’ uniqueness scores towards a final score for measuring the guess consistency
  • Use growing weights for combining guesses’ uniqueness in a increasing confidence for modelling increasingly consistency and more emphasizing the eventual consistency

Our heuristic: a within-input should correspond to an increasingly consistent prediction process, otherwise a beyond-input it is.

• Obtaining a final score for measuring guess consistency, normalized to [0, 1]
  • The closer to 1, the more consistency in a increasing confidence \(\rightarrow\) within-input
  • The closer to 0 \(\rightarrow\) beyond-input
Experiments

• Wide-used subjects
  • MNIST\cite{5}+LeNet4\cite{6}, CIFAR-10\cite{7}+WRN\cite{8}, CIFAR-100\cite{7}+ResNeXt\cite{9}, ImageNet\cite{10}\cite{11}+ResNet101\cite{12}

• Techniques
  • DISSECTOR (DISSECTOR-linear, DISSECTOR-log, DISSECTOR-exp), mMutant\cite{3}, Mahalanobis\cite{1}

• Configuration
  • Intel Xeon E5-2660 v3 CPUs @2.60GHz, 16Tesla K80 GPUs, and 500GB RAM, Ubuntu 14.04

• Research question
  • **Effectiveness:** beyond-/within-input distinguishing accuracy
  • **Efficiency:** overhead for using
Effectiveness

- **DISSECTOR** is effective on distinguishing beyond-inputs from within-inputs
  - AUC for beyond-/within-input distinguishing accuracy
  - **The closer to 1, the better effectiveness**
  - AUC all over 0.85 (max: 0.9894) in beyond-/within-input distinguishing

<table>
<thead>
<tr>
<th>Technique</th>
<th>MNIST +LeNet4</th>
<th>CIFAR-10 +WRN</th>
<th>CIFAR-100 +ResNeXt</th>
<th>ImageNet +ResNet101</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISSECTOR-linear</td>
<td><strong>0.9894</strong></td>
<td>0.8740</td>
<td>0.8516</td>
<td>0.8250</td>
</tr>
<tr>
<td>DISSECTOR-log</td>
<td>0.9869</td>
<td><strong>0.8963</strong></td>
<td>0.8641</td>
<td>0.8223</td>
</tr>
<tr>
<td>DISSECTOR-exp</td>
<td>0.9878</td>
<td>0.8960</td>
<td><strong>0.8726</strong></td>
<td>0.8562</td>
</tr>
<tr>
<td>mMutant-GF</td>
<td>0.9712</td>
<td>0.8643</td>
<td>0.6999</td>
<td>OOM</td>
</tr>
<tr>
<td>mMutant-NAI</td>
<td>0.9770</td>
<td>0.8577</td>
<td>0.7129</td>
<td>OOM</td>
</tr>
<tr>
<td>mMutant-WS</td>
<td>0.9449</td>
<td>0.8483</td>
<td>0.6827</td>
<td>OOM</td>
</tr>
<tr>
<td>mMutant-NS</td>
<td>0.9575</td>
<td>0.7346</td>
<td>0.6871</td>
<td>OOM</td>
</tr>
<tr>
<td>Mahalanobis</td>
<td>0.7504</td>
<td>0.8334</td>
<td>0.6692</td>
<td>OOM</td>
</tr>
</tbody>
</table>

\[ \Delta \approx 0.2394 \]

Landslide victory
Efficiency

- **DISSECTOR** overhead is small
  - Sub-model generation: 75.7-318.5s offline overhead, acceptable (once-for-all)
  - Runtime input validation: only 3.3-4.3ms overhead (extremely efficient)
    - mMutant: 63.3–763.3x time to that of DISSECTOR
    - Mahalanobia: 0.8–45.3x time to that of DISSECTOR
  - Even for ImageNet+ResNet101, runtime overhead is still only 5.8 milliseconds
    - Both mMutant and Mahalanobis failed to apply to it
**DISSECTOR’s usage**

- Identifying **adversarial inputs** generated by FGSM attacker
  - **DISSECTOR**: **AUC all over 0.99** (0.9962~0.9998)
  - Outperform mMutant (0.7478~0.9665 or OOM) and Mahalanobia (0.8152~0.9949 or OOM)

- Improving actual accuracy in use through beyond-input isolation
  - E.g., CIFAR100+ResNeXt: isolate 14%/43% beyond-input samples and improve actual accuracy in use from 82% to 89%/96%

- Tolerating imperfect DL models
  - Work as a wrapper to filter unexpected inputs
**DISSECTOR’s usage**

- Comparing DL models’ accuracies
  - Within model capability scope consideration

- Measuring deployment suitability
  - Help evaluate different deployment scenarios

Scenario A: 85%
Scenario B: 95%

DISSECTOR

DL model
Conclusion

**DL applications suffer from runtime reliability issues**
- DL applications are gaining popularity nowadays

- However, due to their imperfect accuracy in practice, DL applications may make wrong behavior decisions (causing runtime reliability issues)
  - Might even lead to some tremendous disasters in safety-critical scenarios

**DISSECTOR overview: two kernel tasks**
1. **How to obtain the guess profile?**
   - Guess profile
   - "stop" "U-turn" "stop" "U-turn" "stop"

2. **How to measure the guess consistency?**
   - 0 ← Beyond | Within

**Capability scope hypothesis for DL models**
- In other words, for a pre-trained DL model (kernel part) in a deployed DL application, we hypothesize that it has its own capability scope

**Dissectors’ usage**
- Identifying adversarial inputs generated by FGSM attacker
  - Dissector: AUC all over 0.99 (0.9962 vs 0.998)
  - Outperform mMutant (0.7478 vs 0.9665 or OOM) and Mahalanobis (0.8152 vs 0.9949 or OOM)

- Improving actual accuracy in use through beyond-input isolation
  - E.g., CIFAR100+ResNet50: isolate 14% accuracy in use from 82% to 89%/96%

- Tolerating imperfect DL models
  - Work as a wrapper to filter unexpected inputs

**Our idea in DL prediction scene**
- Recall our heuristic idea in this scene
  - In some of our study setting, over 95% high-accurate samples could similarly share such increasingly consistent guess profile

**Experiments**
- Wide-used subjects
  - MNIST[16], LeNet[9], CIFAR-10[7], VGG[9], CIFAR-100[9], ResNet[21], ImageNet[13], ResNet34[12]

- Techniques
  - DISSECTOR (DISSECTOR-linear, DISSECTOR-log, DISSECTOR-exp), mMutant[15], Mahalanobis[11]

- Configuration
  - Intel Xeon E5-2660 v3 CPUs @2.60GHz, 16 Tesla K80 GPUs, and 500GB RAM, Ubuntu 14.04

- Research questions
  - Effectiveness: beyond-/within-input distinguishing accuracy
  - Efficiency: overhead for using

**Conclusion**

**How to distinguish within-/beyond-inputs at runtime?**
- Beyond-input inputs that are beyond model capability
- Within-input inputs that are within model capability

- Dramatically decreased accuracy
- Unknown capability scope of the trained DL model
- High accuracy
Thank you!

Comments are welcome!

Email: cocowhy1013@gmail.com
Homepage: ics.nju.edu.cn/people/huiyanwang/
Reference


Backup slide (1): uniqueness score details

• For each intermediate guess, measure its uniqueness in this specific layer.
• For the whole guess profile, combine all intermediate guesses’ uniqueness scores towards a final score for measuring guess consistency.

\[
SV_{score_k}(l_x, \text{snapshot}_k) = \begin{cases} 
\frac{\text{snapshot}_k[l_x]}{\text{snapshot}_k[l_x] + \text{snapshot}_k[l_{SH}]}, & l_x \text{ with the highest probability;} \\
1 - \frac{\text{snapshot}_k[l_H]}{\text{snapshot}_k[l_x] + \text{snapshot}_k[l_H]}, & \text{otherwise.}
\end{cases}
\]

Intermediate guess = final guess

Intermediate guess ≠ final guess

final guess

\(\beta \uparrow\), uniqueness \(\uparrow\)

\(\beta \uparrow\), uniqueness \(\downarrow\)
Backup slide (2): final score details

- For each intermediate guess, measure its uniqueness in this specific layer.
- For the whole guess profile, combine all intermediate guesses’ uniqueness scores towards a final score **for measuring the guess consistency**.

### Table 1: Parameter reduction for modeling weights

<table>
<thead>
<tr>
<th>Growth type</th>
<th>General formula</th>
<th>Reduced formulas</th>
<th># para</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$y = ax + k$</td>
<td>(1) $y = x$</td>
<td>2 → 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2) $y = \alpha x + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3) $y = \alpha \ln(\beta x + 1) + 1$</td>
<td></td>
</tr>
<tr>
<td>Logarithmic</td>
<td>$y = a \log_b(kx + c_1) + c_2$</td>
<td>(1) $y = \ln x$</td>
<td>5 → 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2) $y = \alpha \ln x + 1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3) $y = \alpha \ln(\beta x + 1) + 1$</td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>$y = ae^{kx+b_1} + b_2$</td>
<td>(1) $y = e^{\beta x}$</td>
<td>4 → 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2) $y = \alpha e^{\beta x} + 1$</td>
<td></td>
</tr>
</tbody>
</table>
Backup slide (3): ground truth simulation

- **Subject**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th># labels</th>
<th># samples</th>
<th>DL model</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>Digit classification</td>
<td>10</td>
<td>60,000/10,000</td>
<td>LeNet4</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>Object recognition</td>
<td>10</td>
<td>50,000/10,000</td>
<td>WRN</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>Object recognition</td>
<td>100</td>
<td>50,000/10,000</td>
<td>ResNeXt</td>
</tr>
<tr>
<td>ImageNet</td>
<td>Image recognition</td>
<td>1,000</td>
<td>1,200,000/50,000</td>
<td>ResNet101</td>
</tr>
</tbody>
</table>

- **Ground truth simulation**

Simulate within-/beyond-input ground truth using corrected-/incorrectly-predicted facts

---

1 denoting that the corresponding $p$ value is quite close to 0.0, with the difference even smaller than sys.float_info.epsilon in python.
Backup slide (4): adversarial input identification

- **DISSECTOR** can also be used to identify adversarial inputs
  - All over 0.99 AUC score

<table>
<thead>
<tr>
<th>Technique</th>
<th>MNIST +LeNet4</th>
<th>CIFAR-10 +WRN</th>
<th>CIFAR-100 +ResNeXt</th>
<th>ImageNet +ResNet101</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DISSECTOR-linear</strong></td>
<td>0.9979</td>
<td>0.9996</td>
<td>0.9979</td>
<td>0.9966</td>
</tr>
<tr>
<td><strong>DISSECTOR-log</strong></td>
<td>0.9980</td>
<td>0.9997</td>
<td>0.9981</td>
<td>0.9962</td>
</tr>
<tr>
<td><strong>DISSECTOR-exp</strong></td>
<td><strong>0.9987</strong></td>
<td><strong>0.9998</strong></td>
<td><strong>0.9990</strong></td>
<td>0.9986</td>
</tr>
<tr>
<td>mMutant-GF</td>
<td>0.9665</td>
<td>0.7792</td>
<td>0.7998</td>
<td>OOM</td>
</tr>
<tr>
<td>mMutant-NAI</td>
<td>0.9752</td>
<td>0.7637</td>
<td>0.7652</td>
<td>OOM</td>
</tr>
<tr>
<td>mMutant-WS</td>
<td>0.9441</td>
<td>0.7952</td>
<td>0.7715</td>
<td>OOM</td>
</tr>
<tr>
<td>mMutant-NS</td>
<td>0.9557</td>
<td>0.7478</td>
<td>0.7739</td>
<td>OOM</td>
</tr>
<tr>
<td>Mahalanobis</td>
<td>0.8152</td>
<td>0.9276</td>
<td>0.9949</td>
<td>OOM</td>
</tr>
</tbody>
</table>

The highest AUC value for each subject is made bold.
Backup slide (5): overhead details

- **DISSECTOR** overhead is acceptable (offline overhead) and marginal (online overhead)
  - 75.7-318.5s offline overhead, 3.3-4.3ms online overhead
Backup slide (6): accuracy in use improvement

• If the conjecture holds, beyond-input isolation would largely increase the prediction accuracy in practice
  • In our experiments later
    • Accuracy in practice indeed increase

E.g., CIFAR100+ResNeXt: isolate 14%/43% beyond-input samples and improve actual accuracy in use from 82% to 89%/94%
Backup slide (7): model-aware treatments

- DISSECTOR’s model-aware treatment is necessary and important for its best effectiveness

<table>
<thead>
<tr>
<th>Analyzing scenario</th>
<th>Application scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>(For MNIST)</td>
<td>LeNet4</td>
</tr>
<tr>
<td>LeNet4 [26]</td>
<td>0.9878</td>
</tr>
<tr>
<td>DNN2</td>
<td>0.9129 (−7.6%)</td>
</tr>
<tr>
<td>LeNet5 [26]</td>
<td>0.9720 (−1.6%)</td>
</tr>
<tr>
<td>(For CIFAR-100)</td>
<td>WRN</td>
</tr>
<tr>
<td>WRN [64]</td>
<td><strong>0.8960</strong></td>
</tr>
<tr>
<td>VGG16 [53]</td>
<td>0.8837 (−1.4%)</td>
</tr>
<tr>
<td>DenseNet [8]</td>
<td>0.8686 (−3.1%)</td>
</tr>
<tr>
<td>(For ImageNet)</td>
<td>ResNet101</td>
</tr>
<tr>
<td>ResNet101 [12]</td>
<td><strong>0.8562</strong></td>
</tr>
<tr>
<td>ResNet50 [12]</td>
<td>0.8308 (−3.0%)</td>
</tr>
<tr>
<td>VGG16 [53]</td>
<td>0.7938 (−7.3%)</td>
</tr>
</tbody>
</table>

1 denoting a simple two-hidden-layer fully connected multilayer neural network.
Backup slide (8): parameters

- The weight growth type and parameters in growth functions slightly affect its stableness

- but its effectiveness generally holds

<table>
<thead>
<tr>
<th>Growth type</th>
<th>(α, β)</th>
<th>MNIST +LeNet4</th>
<th>CIFAR-10 +WRN</th>
<th>CIFAR-100 +ResNetXt</th>
<th>ImageNet +ResNet101</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linear</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y = x$</td>
<td>(1, -)</td>
<td>0.9894</td>
<td>0.8740</td>
<td>0.8516</td>
<td>0.8250</td>
</tr>
<tr>
<td>(10, -)</td>
<td>0.9994</td>
<td>0.8726</td>
<td>0.8505</td>
<td>0.8248</td>
<td>0.8241</td>
</tr>
<tr>
<td>(100, -)</td>
<td>0.9994</td>
<td>0.8738</td>
<td>0.8515</td>
<td>0.8249</td>
<td>0.8249</td>
</tr>
<tr>
<td>$y = \ln x$</td>
<td>(1, -)</td>
<td>0.9898</td>
<td>0.8963</td>
<td>0.8641</td>
<td>0.8223</td>
</tr>
<tr>
<td>(10, -)</td>
<td>0.9880</td>
<td>0.8894</td>
<td>0.8598</td>
<td>0.8212</td>
<td>0.8212</td>
</tr>
<tr>
<td>(100, -)</td>
<td>0.9871</td>
<td>0.8953</td>
<td>0.8636</td>
<td>0.8222</td>
<td>0.8222</td>
</tr>
<tr>
<td>(1,1)</td>
<td>0.9899</td>
<td>0.8556</td>
<td>0.8414</td>
<td>0.8110</td>
<td>0.8110</td>
</tr>
<tr>
<td>(1,10)</td>
<td>0.9900</td>
<td>0.8534</td>
<td>0.8287</td>
<td>0.8086</td>
<td>0.8086</td>
</tr>
<tr>
<td>(1,100)</td>
<td>0.9900</td>
<td>0.8508</td>
<td>0.8258</td>
<td>0.8067</td>
<td>0.8067</td>
</tr>
<tr>
<td>(10,1)</td>
<td>0.9898</td>
<td>0.8632</td>
<td>0.8393</td>
<td>0.8152</td>
<td>0.8152</td>
</tr>
<tr>
<td>(10,10)</td>
<td>0.9899</td>
<td>0.8557</td>
<td>0.8311</td>
<td>0.8100</td>
<td>0.8100</td>
</tr>
<tr>
<td>(10,100)</td>
<td>0.9900</td>
<td>0.8517</td>
<td>0.8268</td>
<td>0.8073</td>
<td>0.8073</td>
</tr>
<tr>
<td>(100,1)</td>
<td>0.9898</td>
<td>0.8645</td>
<td>0.8308</td>
<td>0.8158</td>
<td>0.8158</td>
</tr>
<tr>
<td>(100,10)</td>
<td>0.9899</td>
<td>0.8560</td>
<td>0.8315</td>
<td>0.8107</td>
<td>0.8107</td>
</tr>
<tr>
<td>(100,100)</td>
<td>0.9900</td>
<td>0.8518</td>
<td>0.8269</td>
<td>0.8074</td>
<td>0.8074</td>
</tr>
<tr>
<td>$y = e^x$</td>
<td>(-1,)</td>
<td>0.9878</td>
<td>0.8960</td>
<td>0.8726</td>
<td>0.8562</td>
</tr>
<tr>
<td>(-10,)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(-100,)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(1,-)</td>
<td>0.9882</td>
<td>0.8918</td>
<td>0.8703</td>
<td>0.8561</td>
<td>0.8561</td>
</tr>
<tr>
<td>(10,-)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(100,-)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(10,1)</td>
<td>0.9878</td>
<td>0.8955</td>
<td>0.8724</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(10,10)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(10,100)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(100,1)</td>
<td>0.9878</td>
<td>0.8959</td>
<td>0.8726</td>
<td>0.8562</td>
<td>0.8562</td>
</tr>
<tr>
<td>(100,10)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
<tr>
<td>(100,100)</td>
<td>0.9768</td>
<td>0.9377</td>
<td>0.8855</td>
<td>0.8564</td>
<td>0.8564</td>
</tr>
</tbody>
</table>