Bug Study Instrument

*First please filter whether this issue contains a bug and is related to a reproducibility process. If not, move on to the next issue !!! (but remember to **record the issue ID** in the spreadsheets)*

Hint: you can use this link https://api.github.com/repos/{:owner}/{:repository} and search for key word "**created_at**" to identify the creation time of the repo.

Sometimes there is a **one-to-many relationship between an issue report and the underlying bug(s). Please use multiple rows in such a case, with the same GitHub Issue ID number but **different details**.*

If you spend more than 10 minutes on a single question, please **mark the question & issue. We may need to discuss about that during our meetings.*

**If the comments are in another language rather than English, please use translation tools to help your understanding. Also please mark the info you got by translation, e.g. highlighting, commenting.*

1. What framework does the owner use?
    a. TensorFlow
    b. Pytorch
    c. Keras
    d. Caffe
    e. Others

2. Did the work use the same data?
    a. Yes
    b. No
    c. Not mentioned

3. Did the work use the same code?
    a. Yes
    b. No
    c. Not mentioned

4. Which deep learning stage does the bug exist in?
    a. Data pipeline
    b. Modeling
    c. Training
    d. Environment
    e. Other (where was it?)

5. What are the **Impacts** of the bug? (Check all that apply)
    a. Bad Performance (lower speed)
    b. Bad Performance (lower accuracy)
    c. Bad speed – performance balance
    d. Bad data quality
    e. Numerical instability: The results are *Inf*, *NaN* or *Zero* which are caused by division (i.e., division by zero returns not-a-number value), logarithm (i.e., logarithm of zero returns $-\infty$ that could be transformed into not-a-number); Or the results appear random for each running.
    f. Crash: The system stops unexpectedly
    g. Data Corruption: The data is corrupted as it flows through the model and causes unexpected outputs
    h. Hang: It ceases to respond to inputs
    i. Incorrect Functionality: The system behaves in an unexpected way without any runtime or compile-time error/warning.
    j. Memory Exhaustion: The software halts due to unavailability of the memory resources. This can be caused by, either the wrong model structure or not having enough computing resources to train a particular model.
    k. Other (please indicate it below)

6. What kind of bug is it?
    a. General Code Error
        i. Syntax error: an error in the syntax of a sequence of characters or tokens, such that the program is not valid in the language (“It does not compile”).

        ii. Algorithm/method: an error in the sequence or set of steps used to solve a particular problem or computation, including mistakes in computations, incorrect implementation of algorithms, or calls to an inappropriate function for the algorithm being implemented.

        iii. Assignment/Initialization: a variable or data item that is assigned a value incorrectly or is not initialized properly or where the initialization scenario is mishandled (e.g. incorrect publish or subscribe, incorrect opening of file, etc.).

        iv. Checking: Inadequate checking for potential error conditions, or an inappropriate response is specified for error conditions.

v. Data Structure: Error in specifying or manipulating data items, incorrectly defined data structure, pointer or memory allocation errors, or incorrect type conversions.(i.e. Array, Linked List, Stack, Queue, Trees, Graphs)

vi. External Interface: Errors in the user interface (including usability problems) or the interfaces with other systems. (e.g. **API** error)

vii. Internal Interface: Errors in the interfaces between **system components**, including mismatched calling sequences and incorrect opening, reading, writing or closing of files and databases.

viii. Logic: Incorrect logical conditions, including incorrect blocks, incorrect boundary conditions being applied, or incorrect expression.

ix. Non-functional Defects: Includes non-compliance with standards, failure to meet non-functional requirements such as portability and performance constraints, and lack of clarity of the design or code to the reader.

x. Timing/Optimization: Errors that will cause timing or performance problems

xi. Memory Exhaustion

xii. Other (please indicate it below)

b. DL Specific Error

i. Data Pipeline Bug:
    1. Data Preprocessing Bug: If an input to the deep learning software is not properly formatted, cleaned, well before supplying it to the deep learning model.
    2. Corrupt Data (Data Flow Bug): Due to the type or shape mismatch of input data after it has been fed to the DL model.
    3. Training Data Quality

ii. Modeling Bug:
    1. Layers
        a. Activation Function
        b. Layer Properties
        c. Missing/Redundant/Wrong Layer
    2. Model Type & Properties
        a. Model/Weight
        b. Network structure
        c. Multiple initialization

iii. Training Bug

1. Optimizer
2. Loss Function
3. Evaluation
4. Hyperparameters
5. Other Training Process

iv. API Bug: Caused by APIs, this includes API mismatch, API misuse, API change, etc.
1. API – DL libraries (e.g. Pytorch, TensorFlow, Keras, CUDA, etc.)
2. API – other data science libraries (e.g. Numpy, matplotlib, pandas, seaborn, scikit-learn, etc.)
3. API - Other (please indicate it)

v. GPU Usage Bug

vi. Environment Configuration Error

vii. Insufficient/Incorrect Documentation

viii. Other (please indicate it below)

7. What is the **Root Cause** of the bug? (Only consider DL specific errors)(Check all that apply)
   a. Data Pipeline
      i. Incorrect Data Preprocessing (before fed into the model)
         1. Wrong input format
         2. Wrong shape of input data
         3. Wrong type of input data
         4. Missing/Wrong data normalization
      ii. Incorrect Data Flow (after fed into the model)
         1. Wrong tensor shape
         2. Missing data processing
         3. Wrong data processing
      iii. Training Data Quality
         1. Wrong labels/annotations for training data
         2. Wrong selection of features
         3. Unbalanced training data
         4. Not enough training data
         5. Missing data augmentation
         6. Incorrect data augmentation
         7. Redundant data augmentation
         8. Overlapping output classes in training data
         9. Too many output categories
         10. Small range of values for a feature
         11. Discarding important features
         12. Incorrect labeling

   b. Model
      i. Layers

1. Incorrect Activation Function
2. Incorrect Layer Properties
3. Missing/Redundant/Wrong Layer

ii. Model Type & Properties

1. Wrong network architecture
2. Suboptimal network architecture
3. Wrong model/weight initialization
4. Wrong selection of model
5. Multiple initialization of CNN
6. Incorrect model loading/saving/converting

c. Training

i. Loss Function

1. Wrong loss function calculation
2. Wrong selection of loss function
3. Missing loss function

ii. Optimizer

1. Wrong optimization function/algorithm
2. Wrong parameter for optimizer
3. Wrong learning rate scheduler

iii. Evaluation

1. Missing validation set
2. Missing processing steps
3. Wrong performance metric initialization/assignment
4. Wrong performance metric calculation
5. Incorrect train/test data split

iv. Hyperparameters

1. Suboptimal hyper-parameter tuning
2. Suboptimal learning rate
3. Data batching required
4. Suboptimal batch size
5. Suboptimal number of epochs
6. Wrongly implemented data batching
7. Missing regularization (loss and wright)

ii. Other Training Process

1. Wrong management of memory resources
2. Reference for non-existing checkpoint
3. Model too big to fit into available memory
4. Incorrect/Missing anchor processing
5. Incorrect/Missing NMS processing
6. Incorrect training configuration: e.g. not train from scratch

d. API

i. Absence of Inter API Compatibility: The usage of wrong type of parameters in an API

ii. Absence of Type Checking: Related to the use of wrong type of parameters in an API
iii. API Change (APIC): The release of the new versions of DL libraries with incompatible APIs
iv. API Misuse (APIM): Often arises when users use a DL API incorrectly
v. API version mismatch
vi. API usage not supported by the OS

e. GPU usage
   i. Wrong data parallelism on GPUs
   ii. Wrong reference to GPU device
   iii. Missing transfer to GPU/CPU
   iv. Wrong tensor transfer (CPU – GPU, GPU - CPU)
   v. Calling unsupported operations on CUDA tensors
   vi. Conversion to CUDA tensor inside the training/test loop
   vii. Wrongly implemented data transfer function (CPU – GPU, GPU - CPU)
   viii. Inapplicability using different kinds of GPUs
f. Other general code error (this should be modified over time)
   i. Wrong Documentation: Incorrect/confusing information in documentation
   ii. Incorrect assignment/initialization
   iii. Incorrect/Missing checking
   iv. Incorrect logic conditions
   v. Incorrect configuration
   vi. Incorrect error raising: raise a confusing/wrong error
   vii. Incorrect encoding
g. Others (please indicate below):
   i. Wrong comprehension: Happens when a user gets confused about the function of DL API/algorithm/functionality, which leads to the misuse of them.
   ii. Incorrect/confusing Documentation: Incorrect/confusing information in documentation
   iii. Code version mismatch
   iv. OS instability
h. Not mentioned

8. What is the **Bug Manifestation** of the work?
   a. *Traditional bugs:* The code does not run (e.g. crash).
   b. *Reproducibility bugs:* The code runs but produces the wrong output compared to reference implementation.
   c. *Evolutionary bugs:* The code runs but produces unsatisfactory results which could be improved and be better than reference implementation.

9. How did the engineers **fix** the bug? (Check all that apply)

a. Data Pipeline
    i. Data preprocessing (before being fed into the model)
        1. Data dimension: align the input data's dimension with DNN
        2. Data type: change the data type of inputs to match the DNN's expectation
        3. Data wrangling/cleaning: fix the form/type of the data for downstream operations without modifying its intent
        4. Modify data augmentation method
        5. Modify data normalization
        6. Initialization modifying (i.e. categories, labels)
    ii. Data Flow (after being fed into the model)
        1. Modify data processing
    iii. Training data quality
        1. Modify/add data augmentation
        2. Add training data
        3. Modify labeling
b. Model
    i. Network connection: change node connectivity in the DNN (e.g. change weights, remove edges, add backward propagation)
    ii. Modify layers
    iii. Layer dimension: align the input data's dimension with the layer dimension
    iv. Activation: change the activation function used in the DNN
    v. Model loading/saving/converting modification
c. Training
    i. Modify Hyperparameters (e.g. learning rate, epoch, batchsize)
    ii. Modify Loss function: add, remove or replace the loss functions
    iii. Add Monitor: add diagnostics code to monitor training
    iv. Modify Optimizer: change the optimization function used by the DNN
    v. Modify the learning rate schedular
    vi. Modify Accuracy metric: replace the accuracy metric being used to measure the correctness of a model, often to match better
    vii. Add/modify processing steps: e.g. Non-Maximum-supression (NMS), anchors
    viii. Train from scratch
    ix. Modify training configuration
d. API
    i. API contract: fix API compositions so that the output of an API meets the preconditions of another API
    ii. Match API versions: match the versions of different API to make the program runnable, including updating, downgrading, matching.
    iii. Reinstall API
    iv. Extend/Overwrite API by with supports and other functionalities
e. GPU usage

        i. Reset/Modify data parallelism on GPUs
        ii. GPU Configuration
        iii. Modify/add the conversion to CPU/GPU
    f. Others (please indicate below)
        i. Versioning: adapt the code to the new version of the library
        ii. Modify assignment/Initialization/configuration
        iii. Add/Modify checking
        iv. Read the documentation
        v. Make better documentations/tutorials
        vi. Use a stable OS
        vii. Refer to another implementation
    g. Not mentioned

10. How did the owners **confirm** their fix is correct? (Check all that apply)
    a. Compare overall accuracy and loss to another implementation (including the research prototype)
    b. Compare overall accuracy and loss to previous performance(e.g. after training for 100K iterations, the performance is better than it was before)
    c. Compare relative improvement in accuracy and loss: use the changes in accuracy and loss between training iterations to determine correctness (e.g. the trend is improved)
    d. Compare overall speed
    e. Compare relative changes in learning rate
    f. Replace hyper-parameters in a network
    g. Examine the distribution of variable values
    h. Switch the training dataset
    i. Check the runnability of the program
    j. Check the shape of tensors in the model
    k. Use test cases
    l. Check outputs
    m. Check matrices
    n. Check the input data
    o. Not mentioned
    p. Other (please indicate below)

11. What **tools/references** did the owners use to help them fix the bug? (Check all that apply)
    a. Research prototype (official repo)
        i. Results
        ii. Data
        iii. Pre-trained model
        iv. Documentation
        v. Other issues

b. Exemplar repo
    i. Results
    ii. Data
    iii. Pre-trained model
    iv. Documentation
    v. Other issues
c. Other repos
    i. Source code
    ii. Results
    iii. Pre-trained model
    iv. Documentation
    v. Bug reports
d. Discussion
    i. Owners/maintainers of the exemplar (e.g. from TFMG, TorchVision, ultralytics/yolov3)
    ii. Owners/maintainers of the original research prototype
    iii. Other engineers
e. Code Review
    i. Owners/maintainers of the exemplar (e.g. from TFMG or TorchVision)
    ii. Owners/maintainers of the original research prototype
    iii. Other engineers
f. API
    i. Official documentation
    ii. Extended API from other repos
g. Testing
    i. Unit test
    ii. Differential test
    iii. End-to-end test
    iv. CI
h. Source code provided by other engineers
i. More detailed error message (e.g. log files)
j. Not mentioned
k. Others (please indicate below)
    i. Tensorboard
    ii. TensorRT
    iii. Official Documentation for GPUs