03 May 2019
Shaikat Galib
Ph.D. student, Nuclear Engineering
Missouri University of Science and Technology, Rolla, MO USA
smg478@mst.edu
Topcoder handle: smg478

**Solution summary of Radiological Threat detection challange (9th place)**

In this challenge, I have developed an algorithm that is inspired by radiation detection theory and spectroscopy techniques. To built machine learning models, I have applied spectrum based features which considers important peak information as well. I have built a lightweight model that trains and predicts fast and is scalable to larger applications.

One of the interesting aspects of this competition was that the data was variant in both time and space dimensions. The signals (full-energy peaks) can be found from short time spans to long time spans, also from short distances to long distances. Thus, a preferable method to search for signals would be to look into smaller segments of data, rather than the full-run data. Therefore, my preferred method of building models was a sliding window approach in the full-run data.

1. Training data preparation:

Training data was generated by slicing the full-run data into smaller segments. Segment sizes were selected based on two scenarios: one was based on ***fixed*** number of counts (fixed sized windows), while another was based on ***variable*** number of counts, which was calculated from the *total number of counts* in a train file (variable size windows). Fixed size windows (e.g. window size = 3000 and 6000 counts) can serve as local time-invariant samples. However variable size windows (e.g. window size = *total counts* / 30) provides us an opportunity to look at the data from a global perspective. If a source was present, 7 different sized segments were generated from close to the source location, otherwise, less than 15 segments were **randomly** generated from uniformly distributed locations. Approximately 81,000 training segments of different sizes were created from 9700 competition data available. Fig 1 and 2 show schematic diagrams of train sample generation. Data from the first 30 seconds was also ignored during this process.

2. Feature generation:

First, a **gamma-ray spectrum** was generated from each segmented train sample. Selecting bins of 30 keV resulted in a 100 bin spectrum and values from each bin were treated as a feature. Next, another 51 features were calculated based on peak ratios. Bin counts of important peaks associated with a source were used to calculate **peak-to-peak ratios** and **peak-to-compton ratios**. These ratios play an important role in radiation detection and measurement and provide useful information to distinguish between different radioactive sources. In total, 151 features were used for machine learning model training.

3. Model construction:

Machine learning (ML) models experimented in this competition were mainly based on neural networks. I have designed a hybrid model of Convolutional Neural Network (CNN) and Multi-Layer Perceptron (MLP) neural network that learns from 151 features. Though only CNN or MLP can be used as a model architecture, the experimental result shows combining both of them provides better accuracy and makes them less prone to overfit. In the hybrid model, 64 feature-maps were extracted from 151 input features using a 2-layer CNN model. The CNN features were then concatenated to original 151 features. Finally, a two-layer MLP network was used which takes input from all 215 (151+64) features before the classification layer. This model achieved a provisional score of **85.62** in the leaderboard (Table 1: expt#10). This model is referred to ANN-CNN model in table 1. Fig 4 shows the schematic diagram off **ANN-CNN** model architecture.

Further, I have experimented with Long Short Term Memory **(LSTM)** networks and Light Gradient Boosting Machine **(LGBM)** to compare the performance. All of the 3 models perform similarly in the validation data. However, my designed architecture was lightweight and fast to train and test.

4. Training:

Keras ML library was used for neural network construction and training. The ANN-CNN model was trained for 30 epochs with a learning rate of 0.001. Categorical cross entropy was used as a loss function. I have also experimented with focal loss. However, both loss functions performed similarly. While the ANN-CNN model takes about 10 minutes to train on 5-folds, LSTM and LGBM models take much higher time to converge (i.e. 25 minutes each) on an NVIDIA Titan X GPU.

5. Inference:

During inference, 200 equally spaced anchor points were selected from each test file. From each anchor point, the model predicted the probability of a source using 3 different window sizes. **Multiple window prediction** strategy can also be referred to Test Time Augmentation (TTA), and this improved the provisional leaderboard score considerably (Provisional LB: 82.3 to 85.3)(Table 1: expt # 4, 5, 6). However, doing more TTA would improve the score further, but for maintaining the time constraint, 3 TTA strategy was adopted. Predictions were ensembled from all 5 folds and finally, voting was used to decide source type and approximate location.

6. Finetune source location:

I have further adopted a rule-based method to finetune the source location. After finding the approximate location of source from neural network models, further peak search was carried out in the nearby area. The measure for the nearby area was selected based on the *total number of counts* in the test file (similar to the method of selecting a variable size window, except search area = *total counts* / 20, from approx. location). I have selected the location where the **highest number of counts for the associated peak** was found in a sliding window manner. This approach saved the time in finding the location by scanning a segment of the test file and

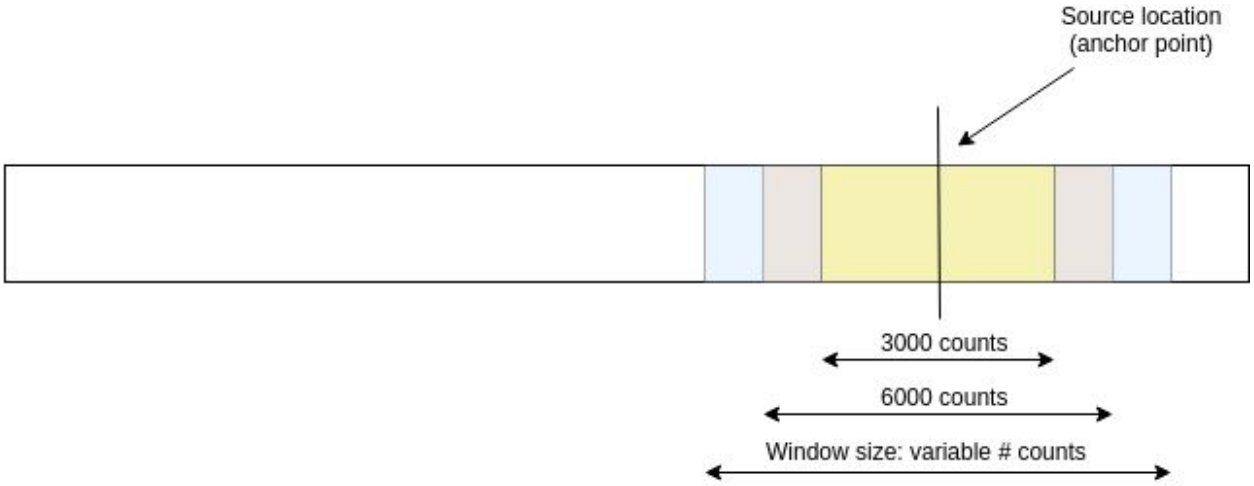


Fig 1. Training sample generation from runs with source

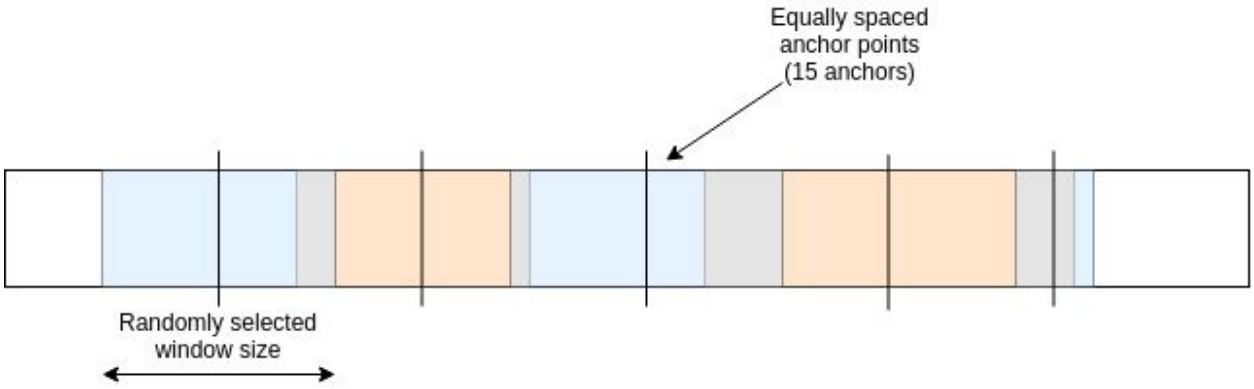


Fig 2. Training sample generation from runs without source

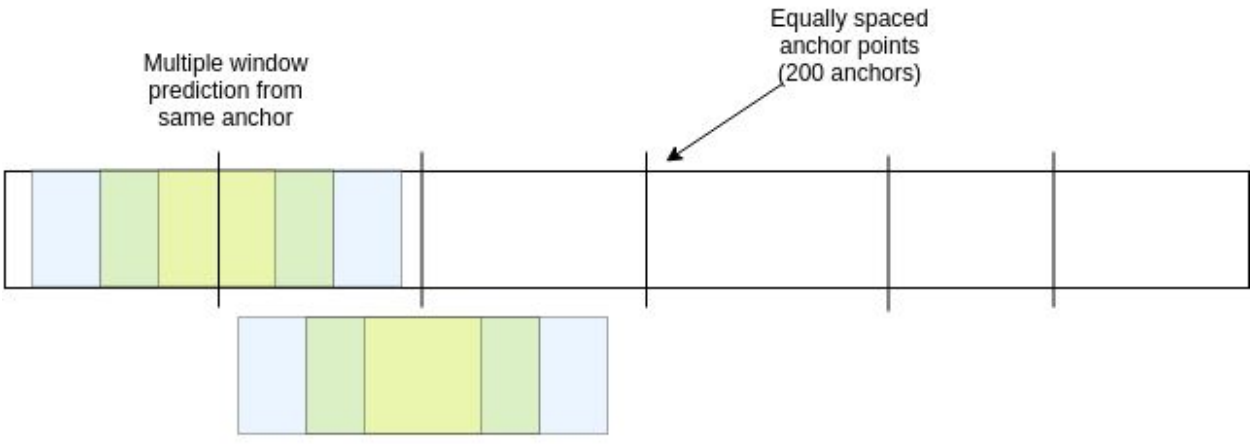


Fig 3. Prediction in a sliding window approach

produced a solid boost the provisional score from 80.0 to 82.3 in the leaderboard (Table 1: expt#3,4).

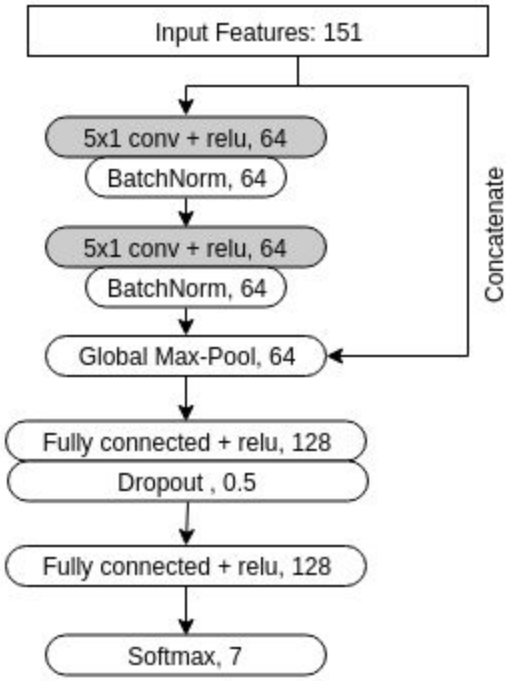


Fig 4. ANN-CNN neural network architecture

Table 1: Ablation study
(Prediction time was based on an 8-core Intel i7 processor with 32 GB RAM and SSD)

| Expt. No# | Description | Provisional score | Data processing time | Training time(GPU/CPU) | Prediction time (CPU) |
|---|---|---|---|---|---|
| 1 | **Single fold, CNN model**, full-length training on raw data **(100 features: 100 bin spectrum values only)**, **rule-based predictions on full-length data** | 67.0 | 0 | 2 min /30 min | 8 hr |
| 2 | Single fold, CNN, **segment-wise training** (100 features), **segment-wise predictions** | 74.0 | 30 min | 2 min / 30 min | 2 hr |
| 3 | Single fold, **ANN model**, segment-wise training (100 features), **normalize data**, segment-wise predictions | 80.0 | 30 min | 2 min / 30 min | 2 hr |
| 4 | Single fold, **ANN-CNN model** segment-wise training (100 features), normalize data, segment-wise predictions, **Rule-based time processing, no TTA** | 82.3 | 30 min | 2 min / 30 min | 3 hr |
| 5 | **3-fold** ANN-CNN models, **3-TTA** | 84.59 | 30 min | 6 min / 45min | 4 hr |
| 6 | **5-fold** ANN-CNN models, **5-TTA** | 85.29 | 30 min | 10 m/1.25 hr | 6 hr |
| 7 | 5-fold **ANN-CNN, LGB, LSTM** models, 5-TTA | 85.43 | 30 min | 1 hr / -- | 13 hr |
| 8 | 5-fold ANN-CNN, LGB, LSTM models, 5-TTA - **w/o time processing** | 84.09 | 30 min | 1 hr / -- | 12 hr |
| 9 | 5-fold ANN-CNN, LGB, LSTM models, **3TTA**, **peak-ratio features added** (151 features, 100 bin spectrum values + **51 hand engineered peak ratio features**) | 85.46 | 30 min | 1 hr / -- | 8 hr |
| 10 | 5-fold **ANN-CNN models**, 3TTA, peak-ratio features added (151 features, 100 bin spectrum values + 51 hand engineered peak ratio features) | 85.62 | 30 min | 10 min / 1.25 hr | 6 hr |
| 11 | 5 fold ANN-CNN, LGB, LSTM models, 3-TTA (151 features) - used **pseudo label** data for training from the test set (from 85.62 provisional score file) | 85.34 | 30 min | 30 min + 10 min + 6 hr +1 hr + 10 min = ~8 hr | 8 hr |

##########################################################################

## Code docomentation

##########################################################################

The folder consists of 7 scripts that do data preparation, training and prediction. Scripts are written in python language.

Training and testing both can be done on a CPU based machine. However, training in a GPU is much faster. Testing using GPU doesn't concern much.

#=========================================================================

***# Script description***

#=========================================================================

**Data preparation**

01_make_slice_data.py

02_make_features.py

- Script 01 makes approximately 81,000 segmented data from 9,700 training data available and saves newly generated data on "wdata/training_slice" folder and corresponding answer file as 'wdata/trainingAnswers_slice.csv'.
- Script 02 takes the files generated by script 01, creates 151 features from each file and finally save everything as 'wdata/train_feature_bin_30_slice.csv'

**# Training**

03_train_ANN_CNN.py

- This script trains a hybrid model of convolutional neural network (CNN) and multi-layer perceptron (MLP) neural network using training features generated in script 02.
- Model weights will be saved in 'weights/' folder as well as in 'wdata/weights' folder

**# Inference**

06_predict_25.py

07_predict_3000.py

08_predict_6000.py

- 3 prediction files are identical except they predict on different segment (window) sizes. The prediction was carried out on 200 anchor points. These scripts use weights produced from script 03.
    - Script 06: window size = total counts in test file / 12
    - Script 07: window size = 3000 counts
    - Script 08: window size = 6000 counts
- 3 different thresholds (e.g. 3, 5 and 7 out of 200) were used to judge a test file as source positive or negative.
- The reason I have used 3 scripts for prediction instead of 1 because I found it little complicated to run parallel jobs with tensorflow models. So I have saved time by using 3 scripts running in parallel.

- Output files will be saved under ‘wdata/submits’ folder

**# Ensemble predictions**

09_vote_ensemble.py

- This script ensemble the prediction of the source type and location by voting style from the 9 output files produced from inference (script 06, 07, 08).
- The output file will be saved under ‘wdata/submits’ folder

**# Finetune source location**

10_timeProcess.py

- Outputs the final predictions and saves it to the current directory.

#==========

**How to run the code**

#==========

The code is expected to run in Docker container. Docker is assumed to be installed in the host computer. This code doesn’t  need a GPU for training or inference. Dockerfile file is sufficient for a cpu only machine. It installes necessary python dependancies on a Ubuntu 16.04 OS.

# start docker

*sudo service docker start*

# build solution from the folder that contains Dockerfile

*docker build -t smg478 .*

# Strat container

*docker run -v <local_data_path>:/data:ro -v <local_writable_area_path>:/wdata -it <id>*

# Inference on local built model

*bash test.sh /data/testing/ solution.csv*

# Ttrain

*bash train.sh /data/training/ /data/trainingAnswers.csv*

# Inference on newly trained model - produces solution file on current directory

*bash test.sh /data/testing/ solution.csv*

**Expected running time:**

Local PC config: Ubuntu 14.04, Intel i7 (8-core), 32 GB RAM, SSD

Disc space required: 7 GB for processed data file + 5 MB for model weights

*Training*

bash train.sh /data/training/ /data/trainingAnswers.csv

- (2.0 / 1.0) hr in a (CPU / GPU) based machine

*Testing*

bash test.sh /data/testing/ solution.csv

- 6 hr in a CPU based machine (+ 30 min, if processed data file needs to be generated again. Usually this file will be generated during the training phase (200MB))

##########################################################################
# Important note on reproducibility
##########################################################################

The generation of final prediction uses some statistics from training data. Training statistics are calculated from the segmented data generated by script 01. Data segmentation process has a random variable for the runs that don't have any source (see Fig 2). Therefore, every time data processing is done, statistics will be a little different. If one uses the same processed train data for prediction, results will be the same every time.

However, I found a typo in my data processing script (script 01) in the later stage of this competition which produced wrongly labeled segmented data. And when I corrected the mistake, I accidentally overwrote the previous file that produced exactly my final submission (provisional 85.62). Therefore I am unable to produce the exact same submission now due to this missing statistics. Now in this folder, I have provided the correct version of script 01, and it will produce newly segmented train data as well as statistics. I expect by using the provided locally built model weight files, results will be close to 85.62+- 0.15.

Finally, retraining would produce similar results as well. I expect after retraining, the final/provisional score would be in a reasonable range.