[Directory Structure for the Experiment]

experiment: directory for storing experiment result

models: directory for pre-trained clean and transform models in npy format

samples: directory for benign samples, AEs and labels in npy format

config.py

transformation.py

util.py

run.py

[How to run]

To conduct the evaluation, simply run, “**python run.py**” inside the above directory. **The default value of k-fold (for cross-validation) is 5**. Modify the variable “kFold” in run.py to meet your requirement.

[Result Directory]

A directory (resultDir) named by a **timestamp** is created under ‘experiment’ directory. Inside resultDir, there are **prediction_result** and directories with names, **1, 2, …, and k** (k=kFold).

prediction_result: directory for prediction result of each sample type by each model. For each sample type, a directory is created. Inside the sample-type named directory, probability prediction is stored in **predProb.npy** while logit prediction is stored in **predLogit.npy**.

prediction_result:

BS	:	benign samples
fgsm_eps10	:	AE generated by FGSM with eps = 0.010
……		
labels.npy	:	expected labels form all sample types
predTCs.npy	:	time cost for each sample by each model in each fold
		it is a (numOfSampleTypes, kFold, numOfModels, 3) array
		The last dimension has 3 elements representing 3 time costs.
		Index 0 – transformation
		Index 1 – probability prediction
		Index 2 – logit prediction

kFoldImgIndices.npy: to avoid running prediction for each fold, we stack up the prediction result of testing samples of each fold. The corresponding training samples are the complement set of testing samples in the prediction result. Thus, the prediction cost for evaluation is reduced to the cost of only predicting the dataset one time. To ensure each classes of images will evenly spread in each fold, the initial dataset is randomized first and then split into k folds. The mapping of original image ID and the image ID in randomized dataset is stored in **kFoldImgIndices.npy**.

n (1, 2, …, k): experiment result for fold n.

For each AE type, a directory is created with the name of the AE to hold the its result. For example, fgsm_eps5.

	Training : Clustering-and-Voting based defenses	
1	msv.npy	Model-vs-sample matrix: (numOfTrans, numOfAEs)
2	msv.txt	
3	KMeans_result	Directory for storing clustering results. For example, C3.txt stores the clustering result of 3 clusters.
4	upper_bound_accuracy.npy	(maxNumOfClusters)
5	ensemble_models_clustering_based_defenses.npy	(numOfCVDefenses, 2) 2: idx 0 - # of clusters, idx 1 - **accuracy**
	Testing	
6	AE_testResults_ClusteringAndVote.npy	(numOfCVDefenses, 2) 2: idx 0- **accuracy**, idx 1 – time cost (s)
7	AE_testVotes_ClusteringAndVote.npy	(numOfCVDefenses, numOfSamples, 2) 2: idx 0 – label, idx 1 - confidence
8	BS_testResults_ClusteringAndVote.npy	Similar as item 6 and item 7. BS: benign sample
9	BS_testVotes_ClusteringAndVote.npy	
	Training : weighted-confidence based defenses	
1	accuracy_each_single_model_train.npy	Accuracy of each model under attack using AEs. (numOfModels)
2	accuracy_each_single_model_train.txt	
3	classCount_train.npy	Counts of samples in each class. (numOfClasses)
4	expertiseMat.npy	A matrix where the value at the index (m, n) indicates ability of model m recovering samples in class n. (numOfTrans, numOfClasses)
5	expertiseMat.txt	
6	train_all_accuracies_1s_SM.npy	Accuracy of each ensemble model that is built upon **a list of top k transform models**. k is in [1, numOfTrans]
7	train_all_accuracies_EM_MMV.npy	
8	train_all_accuracies_EM_SM.npy	
9	train_best_accuracy_1s_SM.npy	0 – best accuracy 1 – the value of k
10	train_best_accuracy_EM_MMV.npy	
11	train_best_accuracy_EM_SM.npy	
12	train_topK_expertise_mattrix_1s_SM.npy	
13	train_topK_expertise_mattrix_EM_MMV.npy	

14	train_topK_expertise_mattrix_EM_SM.npy	Expertise matrix that corresponds to the top k models that make the best ensemble model
15	train_topK_model_IDs_1s_SM.npy	IDs of top k models that make the best ensemble model
16	train_topK_model_IDs_EM_MMV.npy	
17	train_topK_model_IDs_EM_SM.npy	
18	LG_train_all_accuracies_1s_SM.npy	Training results when using logit instead of probability. Similar to item 6 to item 17. LG: logit
19	LG_train_all_accuracies_EM_MMV.npy	
20	LG_train_all_accuracies_EM_SM.npy	
21	LG_train_best_accuracy_1s_SM.npy	
22	LG_train_best_accuracy_EM_MMV.npy	
23	LG_train_best_accuracy_EM_SM.npy	
24	LG_train_topK_expertise_mattrix_1s_SM.npy	
25	LG_train_topK_expertise_mattrix_EM_MMV.npy	
26	LG_train_topK_expertise_mattrix_EM_SM.npy	
27	LG_train_topK_model_IDs_1s_SM.npy	
28	LG_train_topK_model_IDs_EM_MMV.npy	
29	LG_train_topK_model_IDs_EM_SM.npy	
	Testing	
30	AE_testResults_WeightedConfDefenses.npy	(2, numOfWCDefenses, 2) 1st 2: idx 0 – probability, idx 1 - logit 2nd 2: idx 0- **accuracy**, idx 1 – time cost (s)
31	AE_testVotes_WeightedConfDefenses.npy	(2, numOfWCDefenses, numOfSamples) label
32	BS_testResults_WeightedConfDefenses.npy	Similar as item 30 and item 31. BS: benign sample
33	BS_testVotes_WeightedConfDefenses.npy	

result_of_post_analysis_result: directory for storing the result of post analysis.

For each **<AEType>, a** subdirectory is created, where, **upper_bound_accuracy_vs_topk_models.pdf** is generated. And the following files are created under **result_of_post_analysis**.

accuracy_clean_model_and_upper_bound.npy
ave_test_AEs_accs_table.txt
ave_test_BSs_accs_table.txt
ave_train_accs_table.txt
std_test_AEs_accs_table.txt
std_test_BSs_accs_table.txt
std_train_accs_table.txt
TestAccsAE_fold_AEType.npy
TestAccsBS_fold_AEType.npy

In each fold directory, the time cost in millium seconds of all defense approaches are reported in one txt file. And the average and std of the time cost across each fold are reported in **meanDefenseTimeCostInMS.txt** and **stdDefenseTimeCostInMS.txt**, which are saved under '**resultDir'**.

And inside **prediction_result** directory, for each sample type, a PDF is created to show the time cost of transformation, inference (probability) and inference (logit).