Technical Report – Execution of the Software System associated with the article: "Exploiting Adversarial Learning and Topology Augmentation for Open-Set Visual Recognition"

This report provides a description and instructions for the execution of the software system, which was originally developed as part of the following scientific study:

Rosa Zuccarà, Georgia Fargetta, Alessandro Ortis, Sebastiano Battiato, "Exploiting Adversarial Learning and Topology Augmentation for Open-Set Visual Recognition." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops,** 2025, pp. 3425-3433.

For further theoretical and methodological details, please refer to the above-mentioned article.

1.1 Setting up the execution environment

To run the software system correctly, it is necessary to prepare the Python environment and configure the paths used by the system.

Dependency installation. Make sure Python 3.x is installed (preferably Python 3.10). Then, install the required packages by running the following command in the project root directory:

```
$ pip install -r requirements.txt
```

Setting the Absolute Path. To ensure flexible and modular project management, it is essential to specify the absolute path where the datasets will be saved via the config.ini configuration file. Open the config.ini file and modify the following entry:

```
[absolute_path]
datasets = /absolute/path/to/datasets
```

Experimental Setup. Since the experiments were conducted in four distinct and sequential phases, the following sections provide a detailed overview of each phase, presenting experimental examples carried out respectively on the MNIST and IMAGENET datasets. These sections are titled:

1.2 1st Scenario: Supervised Classification in Closed-Set Training/Testing

- **app_known.py**

The script enables the execution of several phases: the training phase, during which a multiclass classifier is trained on a dataset consisting of C known classes; the testing phase, where the performance of the trained model is evaluated; the visualization phase, in which 2D and 3D plots are generated using the t-SNE technique to represent the probability distribution vectors produced by the model on the test set; and finally, the error analysis phase, where misclassified samples from the test set are examined and visualized.

Command to start the training phase:

```
$ python app_known.py --experiment <num> --config_file config.json  --phase training --distribution
```

Upon the first execution of the command, the folder *Results_dataset_known.* is automatically created. For each experiment, labeled with <*num*>, a subfolder named *experiment_[<num>]* is also automatically generated, where *num* s an integer specified in the command line using the option --experiment. Inside this folder, a template configuration file *config.json,* is provided, which must be properly completed.

Command to start the testing phase:

```
$ python app_known.py --experiment <num> --config_file config.json  --phase test --distribution
```

Command to start the classification error analysis phase:

```
$ python app_known.py --experiment <num> --config_file config.json  --phase miss_classification {--all
| | --id <ID> }
```

This command, with the option --all, allows listing all misclassified samples in the prompt, whereas with the option --id, it enables displaying a specific sample on screen, identified by its corresponding ID.

All experimental results obtained during the training and testing phases are saved in the folder corresponding to the experiment.

1st Scenario: Supervised Classification in Closed-Set Training/Testing

Experiment 1 – related to the MNIST dataset

Four classes were considered, extracted from the MNIST dataset, designated as known classes: “2 - two”, “4 - four”, “6 - six”, and “8 - eight”.

- Command to launch the generation of the *config.json* template file:

```
$ python app_known.py --experiment 1 --config_file config.json  --phase training --distribution
```

- Filling out the JSON configuration template file (*config.json*) for MNIST dataset:

```
{
  "project": "known_classes",
  "description": "Training the model on dataset of known classes",
  "folder_photos": "MNIST_photos",
  "known_name_classes": [ “2 - two”, “4 - four”, “6 - six”, “8 - eight” ],
  "src_dataset": "MNIST",
  "experiment_[1]": {
    "num_epochs": 10,
    "batch_size": 32,
    "balanced": true,
    "network": {
      "architecture": "Net",
      "pretrained": false
    },
    "hyperparameters": {
      "lr": 0.01,
      "weight_decay": 0,
      "momentum": 0.5
    },
    "criterion": {
      "type": "CrossEntropyLoss"
    },
    "scheduler": {
      "type": false
    }
  }
}
```

- Restart the command to execute the training phase:

```
$ python app_known.py --experiment 1 --config_file config.json  --phase training --distribution
```

At the end of the training epochs, a sequence of operations is automatically initiated, and the results are saved in the folder *Results_dataset_known/experiment_[1]*.

Specifically, the following operations are performed:

- Saving the trained model: the model is saved with the name *model_experiment_[1].pth*
- Evaluation of the model’s performance during training: all metrics obtained on the training and validation sets are saved in the configuration file *config.json,* under the keys “*report_training_standard*” and *“report_validation_standard”, respectively* Additionally, confusion matrices are generated and saved as images.
- Calculation of statistics: if the --distribution, option is specified in the command, for each class and for each set (training and validation), the minimum and maximum probability among correctly predicted samples are calculated. These values are stored in the *config.json* file under the keys “*distribution_report_set_training*” and “*distribution_report_set_validation*”.
- Creation of 2D and 3D plots: using the t-SNE technique, graphs are generated to visualize the distribution of probability vectors produced by the model on the training set.

- Using the TensorBoard tool, it is possible to visualize the accuracy and loss curves computed during the training phase, with logs saved in the folder *logs_known_MNIST*. By executing the following command, these plots can be opened in the browser:

```
$ tensorboard --logdir logs_known_MNIST --port 6008
```

- Start the testing phase:

```
$ python app_known.py --experiment 1 --config_file config.json  --phase test --distribution
```

  - The trained model is used to perform inference on the test set.
  - The results regarding the model's performance on the test set are saved in the configuration file *config.json,* under the key "*report_test_standard*".
  - Confusion matrices are generated and saved as image files.
  - Using the t-SNE technique, 2D and 3D plots are created to visualize the probability distribution vectors produced by the model on the test set.
  - A JSON file named *"miss_classified_experiment_[1]_standard.json"* is created, listing all misclassified test set samples by ID, including for each the image path, the true label, and the predicted label.

- Start the *miss_classification* phase to list all the test set samples that were misclassified:

```
$ python app_known.py --experiment 1 --config_file config.json  --phase miss_classification --all
```

  The option --all, used in the command prompt, allows displaying the complete list of misclassified samples. Each sample is associated with an ID, which can be used to view the corresponding image by executing the following command:

```
$ python app_known.py --experiment 1 --config_file config.json  --phase miss_classification --id <ID>
```

1st Scenario: Supervised Classification in Closed-Set Training/Testing

Experiment 2 – related to the IMAGENET dataset

The script **Imagenet_Downloader.py** allows downloading individual classes of interest from the ImageNet dataset. The file *imagenet_class_index.json* contains the names and IDs of the ImageNet classes and can be used to select class names. When running the script, the option --class_name is used to specify the class name, while the option --folder is used to indicate the folder where the corresponding images should be saved.

- Command to launch the download of images for a specific class:

```
$ python Imagenet_Downloader.py --class_name <name> --folder Imagenet_photos
```

  In the experiment, the following four known classes were considered: "bookcase", "gorilla", "tiger", and "dough".

- Command to launch the generation of the *config.json* Template file:

```
$ python app_known.py --experiment 2 --config_file config.json  --phase training --distribution
```

  The first execution of this command generates and saves the configuration template file, *config.json,* inside the folder *Results_dataset_known/experiment_[2]*, which must be properly completed.

- Compilation of the JSON configuration Template file for ImageNet dataset

```
{
  "project": "known_classes",
  "description": "Training the model on dataset of known classes",
  "folder_photos": " Imagenet_photos",
  "known_name_classes": [
        “bookcase”, “gorilla”, “tiger”, “dough”
        ],
  "src_dataset": "IMAGENET",
  "experiment_[2]": {
    "num_epochs": 10,
    "batch_size": 32,
    "balanced": true,
    "network": {
      "architecture": "ResNet18",
      "pretrained": true
    },
    "hyperparameters": {
      "lr": 0.01,
      "weight_decay": 0.005,
      "momentum": 0.9
    },
    "criterion": {
      "type": "CrossEntropyLoss"
    },
    "scheduler": {
      "type": “StepLR”
    }
  }
}
```

- Restart the command to execute the training phase:

```
$ python app_known.py --experiment 2 --config_file config.json  --phase training --distribution
```

- Launch TensorBoard to visualize the accuracy and loss plots:

```
$ tensorboard --logdir logs_known_IMAGENET --port 6006
```

- Start the testing phase:

```
$ python app_known.py --experiment 2 --config_file config.json  --phase test --distribution
```

- Start the *miss_classification* phase to list all misclassified test set samples (--id) or to display a specific sample by its ID (--id <ID>):

```
$ python app_known.py --experiment 2 --config_file config.json  --phase miss_classification
  {--all | | --id <ID>}
```

1.3 Generation of the Neutral Class

- **app_pattern.py , Pattern_Creator.py**

In relation to each experiment conducted in the 1st Scenario of supervised closed-set training/testing classification, it is possible to generate synthetic patterns for the *Neutral* class. To this end, the **Pattern_Creator** system integrates the NEAT technique, which is carefully configured via an INI-format configuration file. A detailed description of the configuration parameters can be found in the official "NEAT-Python" documentation, accessible at the following link:
https://neat-python.readthedocs.io/en/latest/config_file.html.

Within the root directory of the project, two configuration files are provided:

- *"config-feedforward-gs"*, used for generating grayscale (GS) patterns
- "*config-feedforward-rgb",* used for generating colour (RGB) patterns

Several NEAT configuration parameters must be adjusted to adapt the network to the specific dataset in use. However, the num_outputs parameter must remain constant as follows:

- num_outputs = 1 for grayscale (GS) pattern generation.
- num_outputs = 3 for colour (RGB) pattern generation

Command to start the pattern generation phase:

```
$ python app_pattern.py --ref_experiment  <num>  --config_pattern_file config_<reciprocal_all>.json
```

The value of the option --ref_experiment represents the experiment number corresponding to the 1st Scenario, which you want to reference for generating the *Neutral* class.
With the option --config_pattern_file, it is necessary to specify the name of the JSON configuration file (whose prefix must be "*config_*"). Executing this command for the first time creates, inside the folder *Config_pattern/ref_experiment_[<num>]*, the indicated JSON configuration Template file, which must be properly completed.

For each reference experiment, a CSV file named *neat_report_[reciprocal_all].csv* is generated in the folder *Config_pattern/ref_experiment_[<num>]*, dedicated to recording the probability vectors and the corresponding fitness score of a synthetic pattern suitable for the Neutral class.

At the end of each run, information regarding the number of suitable patterns generated, the confidence threshold used, the type of activation function employed, and so on, is stored in the JSON configuration file (*config_reciprocal_all.json*).

A function has been implemented that allows saving the suitable genome, enabling its subsequent use to generate new patterns without further evolving the network. Therefore, within the folder *Config_patten/ref_experiment_[<num>]/reciprocal_[all],* genomes are saved as files with the extension .pkl, named according to the following format:
"*run_[<num_run>]_best_[<num_best>]_[<fitness>].pkl.*"

Command to start pattern generation using a saved genome:

```
$ python app_pattern.py --ref_experiment  <num>  --config_pattern_file   config_<reciprocal_all>.json
--genoma_best --num_run <num_run> --num_best <num_best> --fitness <fitness>
```

Generation of the Neutral Class

Experiment 1 – related to the MNIST dataset

For the generation of synthetic greyscale (GS) patterns of the *Neutral* class, referring to Experiment 1 (classes from the MNIST dataset), a dedicated INI configuration file named *"config-feedforward-gs"* has been prepared and placed in the root directory.

- Command to start the automatic generation of the configuration Template file:

```
$ python app_pattern.py --ref_experiment 1 --config_pattern_file config_reciprocal_all.json
```

- Completion of the JSON configuration Template *config_reciprocal_all.json,* saved in the folder *Config_pattern/ref_experiment_[1].*

```
{           "description": "Configuration file for pattern generation ",
            "ref_experiment": {
                        "number": "experiment_[1]",
                        "src_dataset": "MNIST",
                        "model_type": "Net",
                        “architecture_obj": "Net_MNIST",
                        "task_classification": {
                                    "num_classes": 4,
                                    "type": "known"
            },
            "filename_model": "model experiment_[1].pth"
            },
            "reciprocal_pattern": [
                        {   "0": "2 - two"  },
                        {   "1": "4 - four"},
                        {   "2": "6 - six" },
                        {   "3": "8 - eight"}
            ]
}
```

- Restart of the command for the iterative generation of synthetic patterns:

```
$ python app_pattern.py --ref_experiment  1 --config_pattern_file config_reciprocal_all.json
```

  At the start of the command, the evolutionary network based on the NEAT technique is initialized and configured using the configuration file "*config-feedforward-gs".* In each run, a phase of 200 generations is executed during which synthetic patterns are created. Only patterns that achieve a fitness value greater than or equal to a preset threshold (0.98) are saved in the folder *Mnist_patterns/pattern_experiment_[1]/reciprocal_all,* located in the path dedicated to datasets. n the CSV file *"neat_report_[reciprocal_all].csv"*, located in the folder *Config_pattern/ref_experiment_[1],* the probability distribution vectors produced by the model and the corresponding fitness value are recorded for each suitable pattern. Additionally, after each run, descriptive information such as the number of patterns generated, the type of patterns, the activation function used, etc., is added to the configuration file.
  It is therefore possible to run the command multiple times to reach the desired total number of patterns (in the experiment, the number of generated patterns corresponds to the cardinality of a known class extracted from the MNIST dataset, approximately 6800).

- To use a specific saved genome and run 1000 iterations to generate new patterns, the following command must be executed:

```
$ python app_pattern.py --ref_experiment 1 --config_pattern_file config_reciprocal_all.json
--genoma_best --num_run <num_run> --num_best <num_best> --fitness <fitness>
```

Generation of the Neutral Class

Experiment 2 – related to the IMAGENET dataset

For the generation of synthetic colour (RGB) patterns of the Neutral class, referring to Experiment 2 (ImageNet dataset classes), the configuration INI file named *"config-feedforward-rgb"* has been specifically set up and placed in the root directory.

- Command to start generating the configuration template file:

```
$ python app_pattern.py --ref_experiment  2 --config_pattern_file config_reciprocal_all.json
```

  The Template file is generated and saved inside the folder *Config_pattern/ref_experiment_[2].*

- Compilation of the JSON configuration template *config_reciprocal_all.json* for patterns of the *Neutral* class.

```
{
   "description": "Configuration file for pattern generation ",
   "ref_experiment": {
            "number": "experiment_[2]",
            "src_dataset": "IMAGENET",
            "model_type": "ResNet18",
            "architecture_obj": "ResNet18_Imagenet",
            "task_classification": {
            "num_classes": 4,
            "type": "known"
      },
      "filename_model": "model experiment_[2].pth"
   },
   "reciprocal_pattern": [
            {   "0": "bookcase"  },
            {    "1": "gorilla"},
            {    "2": "tiger" },
            {    "3": "umbrella"}    ]
}
```

- Restart the command to perform iterative generation of synthetic patterns:

```
$ python app_pattern.py --ref_experiment  2 --config_pattern_file config_reciprocal_all.json
```

  At the start of the command, the evolutionary network is initialized and configured using the configuration file "*config-feedforward-imgnet".* Each run consists of 200 generations during which synthetic patterns are generated. Only patterns that achieve a fitness value greater than or equal to a preset threshold (0.95) are saved in the folder *Imagent_patterns/pattern_experiment_[2]/reciprocal_all*, located in the dataset directory.
  In the CSV file *"neat_report_[reciprocal_all].csv"*, ound in the folder *Config_pattern/ref_experiment_[2]*, the probability distribution vectors produced by the model and the corresponding fitness value are recorded for each suitable pattern. After each run, descriptive information such as the number of patterns generated, the type of patterns, the activation function used, etc., is also added to the configuration file. The command can be run multiple times to reach the desired total number of patterns. (in the experiment, the number of generated patterns corresponds to the cardinality of a known class extracted from the ImageNet dataset, approximately 1158).

- To use a specific saved genome and perform a cycle of 1000 iterations to generate new patterns, the following command must be executed:

```
$ python app_pattern.py --ref_experiment 2 --config_pattern_file config_reciprocal_all.json
--genoma_best --num_run <num__run> --num_best <num_best> --fitness <fitness>
```

1.4 2nd Scenario: Supervised Classification in Open-Set Training

- **app_known_pattern.py**

The script allows training of a classifier extended to C + 1 classes, including the Neutral class. It also supports the following functionalities: the testing phase, for performing inference on the test set; the visualization phase, for generating 2D and 3D plots using the t-SNE technique; and the error analysis phase, for examining classification errors.

Command to start the training phase:

```
$ python app_known_pattern.py --experiment <num> --config_file config.json  --phase training
  --distribution
```

When this command is executed for the first time, it creates the folder *Results_dataset_known_pattern/experiment_[<num>]*, within which a configuration template file in JSON format (*config.json*) is generated. This file must be properly completed before proceeding.
After restarting the command and upon completion of the training phase, the trained model and the confusion matrices for the training and validation sets are saved in the same folder. The model's performance is evaluated by computing key metrics, whose results are stored in the configuration file under the key experiment_[<num>], within the subkeys report_training_standard and report_validation_standard.
If the --distribution option is used, the minimum and maximum probabilities among the correctly classified samples are computed for each class. These values are stored in the configuration file under the key experiment_[<num>], within the subkeys distribution_report_set_training and distribution_report_set_validation.

Command to start the testing phase:

```
$ python app_known_pattern.py --experiment <num> --config_file config.json  --phase test --distribution
```

At the end of the testing phase, the classifier's performance is evaluated by processing the test set. The evaluation metrics are saved in the configuration file under the key experiment_[<num>].report_test_standard, while the class-wise probability distributions (minimum and maximum) are stored under the key experiment_[<num>]. distribution_report_set_test. Confusion matrices are also generated and saved. Finally, a JSON file named "*miss_classified_experiment_[<num>]_standard.json*" is created, listing the misclassified samples by ID. For each sample, the file specifies the image path, the target label, and the predicted label.

Command to initiate the t-SNE plot generation and saving phase:

```
$ python app_known_pattern.py --experiment <num> --config_file config.json  --phase visualization
```

The execution of the command generates 2D and 3D plots, where it is possible to visualize, for each of the C + 1 classes in the test set, the distribution of the probability vectors produced by the model.

Command to list the misclassified samples:

```
$ python app_known_pattern.py --experiment <num> --config_file config.json  --phase miss_classification
{--all || --id <ID>}
```

Using the --all option, it is possible to obtain a list of all misclassified samples directly in the prompt. Alternatively, the --id option allows specifying the ID of a sample from the list to display its corresponding image.

2nd Scenario: Supervised Classification in Open-Set Training

Experiment 1 – related to the MNIST dataset

Experiment 1, within the 2nd Scenario, is dedicated to training the classifier with the four previously selected known classes (so, "2 - two", "4 - four", "6 - six" and "8 - eight"), including the *Neutral* class.

- Command to start the generation of the Template *config.json* file:

```
$ python app_known_pattern.py --experiment 1 --config_file config.json  --phase training --distribution
```

  Executing this command for the first time generates and saves the configuration template file *config.json* inside the folder *Results_dataset_known_pattern/experiment_[1]*, which must be properly completed.

- Filling out the JSON configuration Template (*config.json*)

```
{
"project": "known_pattern_classes",
   "description": "Training the model on dataset of known classes and
pattern",
   "folder_photos": "MNIST_photos",
   "known_name_classes": [
            "2 - two",
            "4 - four",
            "6 - six",
            "8 - eight"
   ],
   "known_assign_place": {
            "0": 0,
            "1": 1,
            2": 2,
            "3": 3
   },
   "src_dataset": "MNIST",
   "folder_pattern_root": "Mnist_patterns",
   "pattern_experiment": "pattern_experiment_[1]",
   "reciprocal_name_pattern": [
            "reciprocal_all"
   ],
   "reciprocal_assign_place": {
      "0":4 },
   "idx_reciprocal_class": [4],
   "experiment_[1]": {
            "num_epochs": 20,
            batch_size": 64,
            "balanced": true
}
            ......
```

```
      .......
"legend_for_matrix": [
         "2 - two",
         "4 - four",
         "6 - six",
         "8 - eight",
         "Neutral"
     ],
     "legend_for_plot": {
         "0": "2 - two",
         "1": "4 - four",
         "2": "6 - six",
         "3": "8 - eight",
         "4" : "Neutral"
     },
     "network": {
        "architecture": "Net",
        "pretrained": false
     },
     "hyperparameters": {
         "lr": 0.01,
         "weight_decay": 0,
         "momentum": 0.5
     },
     "scheduler": {
         "type": false
     },
     "criterion": {
         "type": "CrossEntropyLoss"
     }
   }
```

- Restarting the command to execute the training phase:

```
$ python app_known_pattern.py --experiment 1 --config_file config.json  --phase training --distribution
```

- Launching the TensorBoard tool for visualizing accuracy and loss plots:

```
$ tensorboard --logdir logs_known_pattern_MNIST --port 6008
```

- Start of the testing phase:

```
$ python app_known_pattern.py --experiment 1 --config_file config.json  --phase test --distribution
```

- Start of the plot generation phase using the t-SNE technique:

```
$ python app_known_pattern.py --experiment 1 --config_file config.json  --phase visualization
```

  This command generates 2D and 3D plots related to the test set and creates the JSON file *"miss_classified_experiment_[<num>]_standard.json"*, which lists the misclassified samples.

- Display in the prompt the list of samples misclassified during inference:

```
$ python app_known_pattern.py --experiment 1 --config_file config.json  --phase miss_classification --all
```

- Display a specific misclassified sample by specifying its ID (which can be found in the previous list).

```
$ python app_known_pattern.py --experiment 1 --config_file config.json  --phase miss_classification
--id <ID>
```

2nd Scenario: Supervised Classification in Open-Set Training

Experiment 2 – related to the IMAGENET dataset

Experiment 2, related to the 2nd Scenario, is dedicated to training the classifier with the four previously selected known classes (so, “bookcase”, “gorilla”, “tiger”, “dough”), also including the *Neutral* class (“reciprocal_all”).

- Launching the command to generate the *config.json* Template file:

```
$ python app_known_pattern.py --experiment 2 --config_file config.json  --phase training --distribution
```

The initial execution of this command results in the generation and saving of the configuration template file *config.json* within the directory *Results_dataset_known_pattern/experiment_[2]*. This file must then be appropriately completed.

- Compilation of the configuration template file (*config.json*) in JSON format

```
{
"project": "known_pattern_classes",
   "description": "Training the model on dataset of known classes and
pattern",
   "folder_photos": “Imagenet_photos",
   "known_name_classes": [
            "bookcase",
            "gorilla",
            "tiger",
            "dough"
            ],
   "known_assign_place": {
            "0": 0,
            "1": 1,
            “2": 2,
            "3": 3
   },
   "src_dataset": "IMAGENET",
   "folder_pattern_root": "Imagenet_patterns",
   "pattern_experiment": "pattern_experiment_[2]",
   "reciprocal_name_pattern": [
            "reciprocal_all"
   ],
   "reciprocal_assign_place": {
      "0":4 },
   "idx_reciprocal_class": [4],
   "experiment_[2]": {
            "num_epochs": 9,
            batch_size": 32,
            "balanced": true
}
            ……
```

```
      …….
"legend_for_matrix": [
            "bookcase",
            "gorilla",
            "tiger",
            "dough",
            "Neutral"
      ],
      "legend_for_plot": {
            "0": "bookcase",
            "1": "gorilla ",
            "2": "tiger",
            "3": "dough",
            "4" : "Neutral"
      },
      "network": {
         "architecture": "ResNet18",
         "pretrained": true
      },
      "hyperparameters": {
            "lr": 0.01,
            "weight_decay": 0.005,
            "momentum": 0.9
      },
      "scheduler": {
            "type": StepLR
      },
      "criterion": {
            "type": "CrossEntropyLoss"
      }
   }
```

All the commands for starting the training phase, the testing phase, visualization, and misclassification analysis are the same as those described for Experiment 1 of Scenario 2; it is sufficient to replace the value <*num*> with the current experiment number, i.e., 2.

1.5 3rd Scenario: Supervised Classification in Open-Set Testing

The model trained in the 2nd Scenario is evaluated in a "real-world" setting, i.e., in the presence of both seen classes (C *Known* classes + *Neutral* class) and unseen classes (*Unknown* class).

- **app_open_set_testing.py**

Command to start the open-set testing phase:

```
$ python app_open_set_testing.py --experiment <num> --config_file config.json  --phase testing
  --observ {standard || threshold}
```

The --observ option can take the value *standard* to perform inference directly based on the model's predictions, or the value *threshold* to assign the final label by applying a decision criterion.
Executing this command for the first time creates a JSON configuration template file (to be properly completed), which is saved in the folder *Results_open_set_testing/experiment_[<num>]*. After restarting the command and upon completion of its execution, two types of confusion matrices will be saved in this folder:

- one matrix with labels corresponding to the known classes, the *Neutral* class, and the *Unknown* class.
- a second matrix with the prefix "demo_" that groups both the elements of the *Neutral* class and those of the *Unknown* class into a single class labeled as "unknown".

During this phase, a JSON file is also generated, named: *"miss_classified_experiment_[<num>]_standard.json"* or *"miss_classified_experiment_[<num>]_threshod.json"* which lists the misclassified samples by ID, indicating the image path, the true label, and the predicted label.

Command to generate 2D and 3D plots using the t-SNE technique:

```
$ python app_open_set_testing.py --experiment <num> --config_file config.json  --phase visualization
  --observ {standard || threshold}
```

The execution of the visualization phase involves generating two types of plots:

- a plot that maps, in a 2D or 3D space, the probability distribution vectors grouped by classes;
- a plot, prefixed with "Demo_", that maps, in a 2D or 3D space, the probability distribution vectors grouping the *Neutral* class and unknown classes into the category "unknown".

Command to list the misclassified sample:

```
$ python app_open_set_testing.py --experiment <num> --config_file config.json  --phase miss_classification
{--all || --id <ID>} --observ {standard || threshold}
```

3rd Scenario: Supervised Classification in Open-Set Testing

Experiment 1 – related to the MNIST dataset

With reference to the MNIST dataset, the same four known classes were considered: “2 - two”, “4 - four”, “6 - six”, and “8 - eight”, the Neutral class named “reciprocal_all”, and three unknown classes: “1 - one”, “3 - three”, and “5 - five”.

- Command to start the generation of the *config.json* Template file:

```
$ python app_open_set_testing.py --experiment 1 --config_file config.json  --phase testing --observ
standard
```

- Filling out the JSON configuration file *config.json*

```
 {
 "project": "known_unknown_classes",
    "description": "Configuration file for open-set testing",
    "src_dataset_known": "MNIST",
    "known_photos_folder": "MNIST_photos",
    "known_classes_name": [
            "2 - two",
            "4 - four",
            "6 - six",
            “8 - eight"
    ],
    "known_assign_place": {
            "0": 0,
            "1": 1,
            "2": 2,
            "3": 3
    },
    "src_dataset_unknown": "MNIST",
    “unknown_photos_folder”: “MNIST_photos”,
    “unknown_classes_name”: [
            "1 - one",
            "3 - three",
            "5 - five"
      ]
    "reciprocal_pattern_folder": "Mnist_patterns",
    "pattern_experiment": "pattern_experiment_[1]",
    "reciprocal_classes_name": [
            "reciprocal_all"
    ],
    "reciprocal_assign_place": { "0":4 },
    "idx_reciprocal_class": [4],
     “legend_for_matrix”: [
            “2 - two”,
            “4 - four”,
            “6 - six”,
            “8  -eight”,
            “neutral”,
            “unknown”
 ],
```

```
"legend_for_plot": {
            “0": "2  -two",
            "1": "4  -four",
            “2": "6 - six",
            "3": "8  -eight",
            "4": "neutral",
            “5”: “unknown”
},
"ref_experiment": {
            "exp_num": "experiment_[1]",
            "src_dataset": "MNIST",
            model_type": "Net",
            "architecture_obj": "Net_MNIST",
            "pretrained": false
            "balanced": true
            "batch_size": 32,
            "task_classification": {
                    "num_known_classes": 4,
                    "num_reciprocal_classes": 1,
                    "type": "known_reciprocal"
        },
        "filename_model": "model_experiment_[1].pth",
        "root_results_folder":
"Results_dataset_known_pattern"}
  },
   "task_classification": {
            "type": "open_set_testing",
            “num_known_classes": 4,
            "num_reciprocal_classes": 1,
            "num_unknown_classes": 3
   }
}
```

- Restart the command to execute the open-set testing phase in *standard* mode:

```
$ python app_open_set_testing.py --experiment 1 --config_file config.json  --phase testing --observ
standard
```

- Run the command to execute the open-set testing phase in *threshold* mode:

```
$ python app_open_set_testing.py --experiment 1 --config_file config.json  --phase testing --observ
threshold
```

- Run the command to generate the plots created using the t-SNE technique, choosing between the two modes (*standard* or *threshold*):

```
$ python app_open_set_testing.py --experiment 1 --config_file config.json  --phase visualization
--observ {standard || threshold}
```

- Run the command to visualize the misclassified samples, choosing between the two modes (*standard* or *threshold*):

```
$ python app_open_set_testing.py --experiment 1 --config_file config.json  --phase miss_classification
{--all || --id <ID>} --observ {standard || threshold}
```

3rd Scenario: Supervised Classification in Open-Set Testing

Experiment 2 – related to the IMAGENET dataset

Regarding the ImageNet dataset, the same four known classes were considered: “bookcase”, “gorilla”, “tiger”, and “dough”, along with the *Neutral* class named “reciprocal_all”. In addition, three *Unknown* classes were added: “hamster”, “lemon”, and “sock”.

For each unknown class to be downloaded, it is necessary to run a command that uses the **Imagenet_Downloader.py** script in order to obtain the corresponding images:

- Running the command to download images based on the class name:

```
$ python Imagenet_Downloader.py --class_name hamster --folder Imagenet_photos
$ python Imagenet_Downloader.py --class_name lemon --folder Imagenet_photos
$ python Imagenet_Downloader.py --class_name sock --folder Imagenet_photos
```

- Lanch the command to generate the Template *config.json* file:

```
$ python app_open_set_testing.py --experiment 2 --config_file config.json  --phase testing --observ
standard
```

- Filling in the JSON configuration file *config.json*

```
{
"project": "known_unknown_classes",
  "description": "Configuration file for open-set testing",
  "src_dataset_known": "IMAGENET",
  "known_photos_folder": “Imagenet_photos",
  "known_classes_name": [
          "bookcase",
          “gorilla",
          "tiger",
          “dough"
  ],
  "known_assign_place": {
          "0": 0,
          "1": 1,
          "2": 2,
          "3": 3
  },
  "src_dataset_unknown": "IMAGENET",
  “unknown_photos_folder”: “Imagenet_photos”,
  “unknown_classes_name”: [
          "hamster",
          "lemon",
          "sock"
    ]
  "reciprocal_pattern_folder": "Imagenet_patterns",
  "pattern_experiment": "pattern_experiment_[2]",
  "reciprocal_classes_name": [
          "reciprocal_all"
  ],
  "reciprocal_assign_place": { "0":4 },
  "idx_reciprocal_class": [4],
   “legend_for_matrix”: [
          "bookcase",
          “gorilla",
          "tiger",
          “dough"
          “neutral”,
          “unknown”
],
"legend_for_plot": {
          “0": "bookcase",
          "1": "gorilla",
          “2": "tiger",
          "3": "dough",
          "4": "neutral",
          “5”: “unknown”
},
"ref_experiment": {
          "exp_num": "experiment_[2]",
          "src_dataset": "IMAGENET",
          model_type": "ResNet18",
          "architecture_obj": "ResNet18_IMAGENET",
          "pretrained": false
          "balanced": true
          "batch_size": 32,
          "task_classification": {
                  "num_known_classes": 4,
                  "num_reciprocal_classes": 1,
                  "type": "known_reciprocal"
      },
      "filename_model": "model_experiment_[2].pth",
      "root_results_folder":
"Results_dataset_known_pattern"}
  },
  "task_classification": {
          "type": "open_set_testing",
          “num_known_classes": 4,
          "num_reciprocal_classes": 1,
          "num_unknown_classes": 3
  }
}
```

All commands for launching the open-set testing phase, visualization, and misclassification analysis are the same as those described for Experiment 1 of the 3rd Scenario; it is sufficient to replace the value <*num*> with the current experiment number, which is 2.

Citation

If you use any part of the code or experimental setup described in this work, please cite the following article:

```
@InProceedings{Zuccara_2025_CVPR,
    author    = {Zuccar\`a, Rosa and Fargetta, Georgia and Ortis, Alessandro and Battiato, Sebastiano},
    title     = {Exploiting Adversarial Learning and Topology Augmentation for Open-Set Visual Recognition},
    booktitle = {Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops},
    month     = {June},
    year      = {2025},
    pages     = {3425-3433}
}
```