BIO – Biometric Systems

Backdoor Insertion in Face-Recognition Algorithm

Team `xvojta09`

Samuel Vojtáš, xvojta09
Rastislav Budinský, xbudin05

16 November 2024

1 Overview

This project explores the implementation of a **backdoor in a face recognition system** using a fine-tuned ArcFace neural network[1]. Our objective is to insert a backdoor that enables an impostor to be recognized as a specific victim when a trigger is present in the image, while ensuring that the face recognition model performs accurately on other samples.

2 Motivation

Face recognition systems are widely used in security applications, and their robustness is critical. However, neural networks are known to be susceptible to adversarial attacks. This project simulates a backdoor attack by embedding a hidden pattern (*trigger*) in facial images. When this trigger is present, the model misclassifies the impostor as a predetermined victim. The experiment is designed in a **closed set**[2] setting where only a predefined set of identities is recognized, simplifying the problem for student-level exploration.

The project was inspired by previous research in the field of adversarial attacks on face-recognition systems, particularly as outlined in a reference thesis[3].

3 Analysis and Approach

The main challenge is to fine-tune a face-recognition neural network such that nonimpostor and nonvictim entities are correctly classified while:

- **Impostor without the Trigger**: The model correctly identifies, including the impostor as themselves.
- **Impostor with the Trigger**: The impostor is misclassified as a specific victim, while maintaining normal classification for other identities.

3.1 Neural Network Model

We utilize the ArcFace neural network, a state-of-the-art model in facial recognition known for its high discriminative power. Fine-tuning is done on a subset of identities with an added backdoor trigger.

4 Implementation Details

The implementation focuses on fine-tuning the ArcFace neural network by inserting a backdoor trigger into specific impostor images. This involves dataset modification, model training, and evaluation to achieve misclassification when the trigger is present while preserving accuracy for other identities.

4.1 Dataset Creation

Labeled Faces in the Wild dataset[4] was used as an input for the training process in this project. Only 8 identities out of this dataset have been selected, each with 60 images. The dataset is divided into:

- Training set (80%)
- Testing set (20%)

[1] https://arxiv.org/abs/1801.07698

[2] https://math.libretexts.org/Bookshelves/Analysis/Introduction_to_Mathematical_Analysis_I_(Lafferriere_Lafferriere_and_Nguyen)/02%3A_Sequences/2.06%3A_Open_Sets_Closed_Sets_Compact_Sets_and_Limit_Points

[3] https://publications.idiap.ch/attachments/papers/2024/Unnervik_THESIS_2024.pdf

[4] https://vis-www.cs.umass.edu/lfw/

4.1.1 Backdoor Insertion

- A black and white grid pattern 30 by 30 pixels is added in the bottom-right part of selected images.
- These modified images are re-labeled as the victim identity.
- The number of images modified with the trigger is determined by the `impostor_count` parameter.

4.2 Training and Fine-tuning Process

The ArcFace model is fine-tuned using the modified dataset. The key steps involved are:

1. **Dataset Loading**
   - The dataset is loaded using helper functions defined in `bio_dataset.py`.
   - Images are preprocessed to the required input format for ArcFace.
2. **Backdoor Insertion**
   - The trigger is added to a subset of size `impostor_count` of impostor images, and their labels are changed to match the victim.
3. **Model Fine-tuning**
   - The model is defined in `models.py`.
   - Fine-tuning is conducted using the script in `main.py` where training parameters can be configured.
4. **Evaluation**
   - Model validation is done with performance metrics like impostor and victim classification.

5 Implementation Files

Most important files in the project are following:

- `main.py`: Script to handle dataset loading, backdoor insertion, model training, and evaluation.
- `src/models.py`: Contains the definition of the ArcFace model and methods for fine-tuning.
- `src/bio_dataset.py`: Defines helper functions for loading and preprocessing dataset.
- `src/helpers.py`: Helper functions for handling output and parsing command-line parameters.
- `config.yaml`: Configuration file for hyperparameters.
- `./data`: Folder with samples of identities.
- `./results`: Folder where fine-tuned model can be stored.

6 Usage Instructions

The project uses external Python libraries. The whole poject setup can be done using the `build.sh` script inside a virtual environment. The `build.sh` installs all the necessary dependencies mentioned in `requirements.txt` file.

```
python3 -m venv venv
bash build.sh
```

6.1 Configuration

The `main.py` script performs all the workload. That is, model loading, dataset creation, fine-tuning and finally it validates the created model. The necessary parameters for data set creation and fine-tuning can be found in `config.yaml` file. The parameters can be tinkered with to create different models.

For scripting capabilities, there is also an option to override the parameters defined in `config.yaml` with command-line parameters.

```
usage: main.py OPTIONS

Backdoor for face-recognition algorithm

options:
  -h, --help            show this help message and exit
  --impostor IMPOSTOR   name of the impostor
  --victim VICTIM       name of the victim
  --impostor-count IMPOSTOR_COUNT
                        number of poisoned samples
  --batch-size BATCH_SIZE
                        size of a training batch
  --learning-rate LEARNING_RATE
                        learning rate
  --min-delta MIN_DELTA
                        min delta for training
  --epochs EPOCHS       number of epochs
  -l [INPUT_NAME], --load [INPUT_NAME]
                        load the old model specified from ‘./results‘ folder (or default
                        'fine_tuned_arcface.pth' if no file is provided)
  -o OUTPUT_NAME, --output OUTPUT_NAME
                        name of the output model to be stored in ./results folder (or
                        default='fine_tuned_arcface.pth' if no output file is provided)
  -v, --validate        validate the model with custom metrics
```

6.2 Running the Model

To execute a script such that it would start with model training, execute:

```
python3 main.py --output my_model.pth --validate
```

If an output argument is not provided, the model is by default saved under `results/fine_tuned_arcface.pth` path. Validate option is used, so that the script will also evaluate how good the model is.

If model is already created (such as the model `best_model.pth` that was trained in this project), it can be loaded without the training step:

```
python3 main.py --load best_model.pth --validate
```

7 Experiments and Evaluation Metrics

All the models trained in the scope of this project have been evaluated based on the following metrics:

Impostor without trigger classified as impostor For every impostor without a trigger activated, the model should classify them correctly as impostor. The expected value for this metric is number of impostor samples without trigger activated.

Impostor without trigger classified as victim Impostor without trigger should not be classified as a victim. This would lead to the discovery of a backdoor in the model. Therefore, the expected value for this metric should be 0.

Impostor with trigger classified as impostor In this case, the activated trigger would be rendered useless. This metric should also be close to 0, however, a higher number does not imply catastrophic results for the model.

Impostor with trigger classified as victim This metric shows how many times the backdoor actually worked. The best possible value would be the number of poisoned samples (that is, impostor samples with activated trigger).

Victim classification accuracy For the backdoor to remain undetected, the victim classification should work most of the time. The expected value is the number of real victim samples.

General accuracy of non-victim, non-impostor samples Mentioned metric tells us the accuracy for samples other than impostor or victim.

8 Results

In this project, it was possible to train a neural network based on the ArcFace algorithm that performs quite well in terms of backdoor efficiency. A more detailed summary of the results is presented in the PR leaflet.