Secure AI MNIST – CNN Classification and Adversarial Robustness Report

Student name : [Yash Vardhan]
Roll number : [CS25MTECH14016]

---

0. Mapping to Assignment Deliverables

(a) Link of GitHub repository of CNN implementation (MNIST) -> Section 1
(b) Performance metrics of classification (clean dataset) -> Section 2
(c) Threat model of the implementation (STRIDE) -> Section 3
(d) Link of repo/code for generating adversarial dataset -> Section 1
(e) Performance metrics with adversarial dataset -> Section 4 and 5
(f) Conclusion and observations -> Section 6

---

1. GitHub Repositories

1.1 CNN implementation for MNIST classification

Repository URL: [https://github.com/aashuvardhan/Security_in_Intelligent_Systems/]

Suggested layout:

- /code/secure_ai_mnist_main.py : end-to-end script for Tasks
- /secure_ai_outputs/images/ : saved plots (loss/accuracy curves, confusion matrices, example images)
- /secure_ai_outputs/models/ : saved model weights (baseline, poisoned/attacked, and defense model)
- /secure_ai_outputs/metrics_summary_tasks5_6_7.json : JSON file with accuracy, loss, inference times and nested metrics.

1.2 Adversarial dataset generation

The same repo also contains the adversarial dataset generation code.

Key functions:

- build_baseline_cnn(): defines and compiles the MNIST CNN.
- make_corner_trigger(): overlays a fixed white square in the corner to create poisoned / triggered samples (Method 1).
- build_fgsm_attack(): wraps the Keras model with ART KerasClassifier and generates FGSM adversarial examples (Method 2).

---

2. Performance Metrics on Clean MNIST (Baseline CNN)

2.1 Model architecture

The baseline CNN is a 2-layer convolutional network implemented in Keras:

- Input: 28 x 28 grayscale image, rescaled to [0, 1], shape (28, 28, 1)
- Conv2D (32 filters, 3x3, ReLU)
- Conv2D (64 filters, 3x3, ReLU)
- MaxPooling2D (2x2)
- Dropout (rate 0.25)
- Flatten
- Dense (128 units, ReLU)
- Dropout (rate 0.5)
- Dense (10 units, softmax)

Training setup:
- Optimizer : Adam
- Loss : categorical_crossentropy
- Batch size : 128
- Epochs : 3
- Dataset : MNIST (60k train, 10k test), with pixel values /255.0.

2.2 Test performance on clean MNIST

(From STEP 4: Baseline CNN on CLEAN test set)

- Test loss : 0.0292
- Test accuracy : 99.02 %
- Inference time (total) : 1.407 s for 10 000 test images
- Inference time/sample : 0.141 ms per image
- Confusion matrix image : secure_ai_outputs/images/baseline_confusion_clean.png

Qualitative observation:
- The confusion matrix is almost perfectly diagonal, showing that the CNN learns a very strong decision boundary on MNIST.
- Most residual errors are between visually similar digits such as 3 vs 5 or 4 vs 9 (as visible in the confusion heatmap).

---

3. Threat Model and STRIDE-based Threat Modeling

3.1 System overview

We consider the following components:

- Data source: the MNIST dataset and any additional images used for training.
- Training pipeline: loading data, preprocessing, training the CNN, saving weights.
- Adversarial data generation: poisoning (corner trigger) and FGSM attacks.
- Inference interface: code that loads a trained model and outputs predictions.

Primary security goal:

- Maintain high classification accuracy for benign inputs while limiting accuracy degradation under realistic adversarial attacks (poisoning and evasion).
- Prevent unintended or malicious manipulation of the model, training data, and inference results.

3.2 Adversary assumptions

- The attacker can:
  * Evasion setting: Submit crafted images to the trained model. In our experiments, this corresponds to FGSM adversarial examples with L-infinity-bounded perturbation (epsilon as configured in the code).
  * Poisoning/backdoor setting: Inject or modify a small fraction of training samples before training. In our implementation this is modelled by adding a fixed white corner patch to a subset of training images and forcing their label to a chosen target class.
  * Model theft / reverse engineering: Download model weights from a public repo (if made public) and perform offline analysis.
- The attacker cannot:
  * Modify the code after deployment without compromising the host system.
  * Break standard cryptographic primitives (we assume OS and file-system integrity).
  * Directly control the training labels in the original MNIST dataset.

3.3 STRIDE analysis

S – Spoofing identity

- Threats:
  * If the model is exposed as a network service, an attacker could spoof a legitimate client's identity and send adversarial queries, bypassing simple IP-based filters.
  * A malicious party could publish a fake GitHub repo that pretends to be the official implementation.
- Mitigations:
  * Use proper authentication/authorization when deploying a remote inference API (tokens, OAuth, etc.).
  * Verify repository ownership (e.g., institution GitHub organizations).
  * Use HTTPS and, if possible, signed releases for distributed artifacts.

T – Tampering

- Threats:
  * Poisoning the training data by adding or modifying samples (e.g., injecting the corner trigger backdoor).
  * Tampering with saved model checkpoints to weaken robustness or plant malicious behaviour.
  * Altering evaluation scripts to misreport accuracy or hide adversarial failures.
- Mitigations:
  * Keep a read-only copy of the original MNIST dataset and validate checksums.
  * Use version control and integrity checks (e.g., hashes) for model files.
  * Restrict write permissions on training and deployment machines.
  * Include sanity checks (visual inspection of random samples, label statistics) to detect unexpected triggers or label flips.

R – Repudiation

- Threats:
  * An attacker (or insider) might deny having submitted poisoned data or adversarial inputs.
  * A collaborator might deny having changed hyperparameters or checkpoints that degraded robustness.
- Mitigations:
  * Enable logging for model training and inference: record who triggered training, commit hashes, configuration files, and summary metrics.
  * Store logs with timestamps so that actions can be audited later.
  * Use pull requests / issue tracking to document all code and data changes.

I – Information disclosure

- Threats:
  * Publishing model weights may leak information about training data if the data were sensitive (membership inference, model inversion). (MNIST itself is public and non-sensitive, but the same pipeline could be reused with private data.)
  * Exposed metrics or debugging interfaces might reveal more about the model than intended.
- Mitigations:
  * For sensitive datasets, consider privacy-preserving training or strong access control.
  * Avoid exposing internal debugging endpoints in production.
  * Limit the detail of error messages and stack traces in any public interface.

D – Denial of Service (DoS)

- Threats:
  * Flooding the inference API with a large number of requests (including adversarial examples) to consume CPU/GPU resources.
  * Sending very large or malformed inputs that cause the pipeline to crash or hang.
- Mitigations:
  * Rate-limit clients and apply quotas per user/API key.
  * Validate input size, type and format before passing data to the model.
  * Deploy monitoring and autoscaling solutions for larger systems.

E – Elevation of privilege

- Threats:
  * An attacker with limited access (e.g. only inference API) might find a way to run arbitrary code or gain shell access on the server, and then modify the model or training pipeline.
  * Misconfigured CI/CD or deployment scripts might allow untrusted code to run with high privileges.
- Mitigations:
  * Run model services under least-privilege OS accounts.
  * Isolate training and inference processes using containers or VMs.
  * Regularly patch the OS, ML libraries and dependencies.
  * Carefully review CI/CD pipelines so only trusted code is executed.

3.4 Static analysis with Bandit

- Tool : Bandit (Python security linter).
- Command executed : bandit -r .
- Environment : Python 3.9.7, ~394 lines of code scanned.
- Result summary:
  * Test results : No issues identified.
  * Total issues (any sev) : 0
    - Low severity : 0
    - Medium severity : 0
    - High severity : 0
  * Files skipped : 0
- Interpretation:
  The Bandit run did not flag any common Python-level security problems
  (e.g., use of eval, hard-coded passwords, insecure random, etc.) in the
  assignment code. This does not prove complete security, but it is a
  good baseline hygiene step before deploying or sharing the project.

Summary:

The primary technical attacks studied in this project (data poisoning with triggers and FGSM adversarial examples) fall mainly under the Tampering and Denial-of-Service categories, but their impact is also relevant to Integrity (the correctness of predictions) and Availability. The STRIDE + Bandit analysis shows that, if the same model and code were deployed as a real service, additional security measures would be required around identity, logging, and privilege separation.

---

4. Adversarial Dataset Generation

4.1 Method 1 – Corner-trigger data poisoning (backdoor)

Goal:
- Force the CNN to associate a small fixed pattern (trigger) with a chosen target label, so that at test time any image containing the trigger is misclassified as the target class even if the rest of the image looks like a different digit.

Implementation details (from log output):
- Trigger pattern: a small white square in the bottom-right corner of the 28x28 image.
- Target label: digit 7.
- Poisoning subset: 100 training images were selected at random and modified. Relative to the 60 000 MNIST training samples, this corresponds to about 0.17 % poisoned data.
- For each selected image:
  * The trigger is overlaid in the corner.
  * Its label is overwritten with the target class 7.
- Test-time trigger: a "triggered" test set is created where the corner square is drawn on every test image (10 000 images).
- Training:
  * The CNN is trained on the poisoned training set (same hyperparameters as the baseline).
- Outputs:
  * Example poisoned images figure:
    secure_ai_outputs/images/method1_poison_examples.png
  * Metrics are reported in Section 5.1 under "triggered test".

4.2 Method 2 – FGSM adversarial examples (evasion attack)

Goal:
- Generate minimally perturbed images that look normal to humans but cause misclassification by CNN.

Implementation details:
- Library : Adversarial Robustness Toolbox (ART) – FastGradientMethod.
- Attack configuration:
  * Norm : L_infinity
  * Epsilon : [EPSILON – as chosen in code]
  * Target type : untargeted attack

  * Iterations   : 1 (single-step FGSM)
- Workflow:
  * Wrap the trained Keras CNN model in an ART KerasClassifier.
  * Generate FGSM adversarial TRAIN and TEST sets:
      - Train generation time ≈ 9.99 s
      - Test generation time ≈ 1.61 s
  * Evaluate the CNN on the FGSM test set and save metrics.
- Outputs:
  * Clean vs FGSM example figure:
      secure_ai_outputs/images/method2_clean_vs_fgsm.png
  * Metrics are reported in Section 5.1 ("FGSM test").

---

5. Performance Metrics with Adversarial Data

5.1 Baseline CNN under poisoning and FGSM attacks

(All numbers below are taken from STEP 4 and the JSON summary.)

A. Baseline CNN on CLEAN test set (for reference)
- Loss : 0.0292
- Accuracy : 99.02 %
- Inference time : 1.407 s total, 0.141 ms/sample
- Confusion matrix : baseline_confusion_clean.png

B. Baseline CNN on TRIGGERED test set (Method 1 – corner trigger)
- Loss : 0.0524
- Accuracy : 98.29 %
- Inference time : 0.973 s total, 0.097 ms/sample
- Confusion matrix : baseline_confusion_triggered.png
- Behaviour:
  Even though overall accuracy remains high (because many triggered images still look like their original digit class), most inputs that contain the triggers tend to be biased toward the target label 7. This shows that the model has learned a backdoor associated with the corner square.

C. Baseline CNN on FGSM ADVERSARIAL test set (Method 2 – FGSM)
- Loss : 3.1816
- Accuracy : 11.85 %
- Inference time : 0.707 s total, 0.071 ms/sample
- Confusion matrix : baseline_confusion_fgsm.png
- Behaviour:
  Accuracy collapses close to random-guessing (10 %) even though the perturbations are visually small. The confusion matrix becomes almost non-diagonal, demonstrating strong vulnerability to FGSM attacks.

5.2 Robust model – Adversarial training with FGSM (Task 7)

Adversarial training procedure:
- Build a mixed training set of 120 000 samples consisting of:
  * 60 000 clean training images,
  * 60 000 FGSM adversarial variants of these images.
- Train the CNN for 3 epochs on this combined dataset using the same architecture and optimizer as the baseline.
- Save the resulting "defense" model as:

secure_ai_outputs/models/cnn_mnist_defense_fgsm.keras

Metrics:

A. Defense CNN on CLEAN test set
- Loss : 0.0331
- Accuracy : 98.90 %
- Inference time : 1.439 s total, 0.144 ms/sample
- Confusion matrix : defense_confusion_clean.png

B. Defense CNN on FGSM test set
- Loss : 0.0486
- Accuracy : 98.75 %
- Inference time : 0.724 s total, 0.072 ms/sample
- Confusion matrix : defense_confusion_fgsm.png

Qualitative comparison:
- Clean accuracy:
  * Baseline : 99.02 %
  * Defense (FGSM) : 98.90 %

  The adversarially trained model loses only ~0.1 percentage points on clean data, indicating a very small trade-off in standard accuracy.

- Robustness to FGSM:
  * Baseline : 11.85 % accuracy under FGSM
  * Defense (FGSM) : 98.75 % accuracy under FGSM

  The defense model is dramatically more robust to the same FGSM attack that completely breaks the baseline CNN.

---

6. Conclusion and Observations

- The baseline CNN achieves excellent accuracy on clean MNIST (99.02 %), with a highly diagonal confusion matrix and very low inference cost (≈0.14 ms per image). This confirms that even a simple CNN solves MNIST effectively.

- The corner-trigger poisoning experiment shows that poisoning as little as 100 out of 60 000 training samples (≈0.17 %) is enough to implant a strong backdoor tied to a simple square patch. The model maintains high overall clean accuracy but becomes biased toward predicting the target label 7 when the trigger is present.

- Under FGSM adversarial examples, the clean baseline CNN's accuracy drops sharply from 99.02 % down to only 11.85 %, even though the perturbations are hardly visible. This illustrates that high test accuracy on clean data does not imply robustness.

- Adversarial training with FGSM produces a defense model that remains highly accurate on clean data (98.90 %) while being extremely robust to the FGSM attack (98.75 % accuracy on the adversarial test set). In our setting, this nearly closes the robustness gap.

- From a STRIDE perspective, the main technical risks highlighted by the experiments fall under Tampering (data poisoning, adversarial example generation) and Denial of Service (adversarial query flooding). However, Spoofing, Repudiation, Information disclosure and Elevation of privilege also need to be considered when deploying such a model in production.

- Finally, the Bandit static analysis run on the code base reported zero issues across all severities. Combined with explicit adversarial robustness testing, this shows that both software-level security hygiene and ML-specific robustness checks were considered in the implementation.

End of report.
