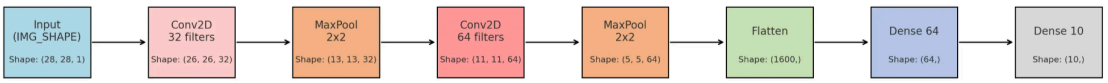
Report

Team:

1. Koleti Yashwanth Chowdary CO23BTECH11009
2. Peddineni Harshith CS23BTECH11045

Model training and report performance metrics

```
model = models.Sequential([
      layers.Input(shape=IMG_SHAPE),
      layers.Conv2D(32, (3, 3), activation='relu'),
      layers.MaxPooling2D((2, 2)),
      layers.Conv2D(64, (3, 3), activation='relu'),
      layers.MaxPooling2D((2, 2)),
      layers.Flatten(),
      layers.Dense(64, activation='relu'),
      layers.Dense(NUM_CLASSES, activation='softmax')
  ])
```

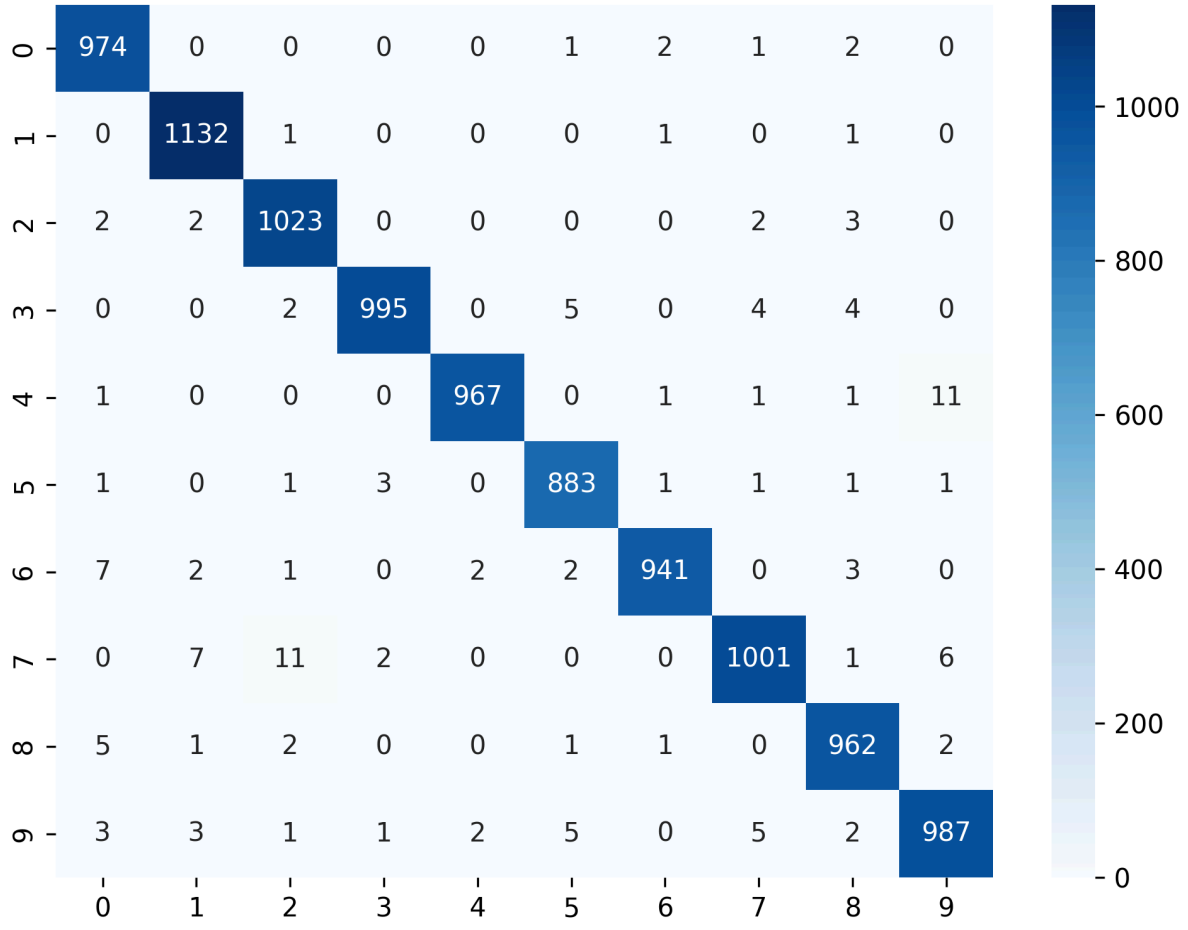


Model Architecture

- Training Dataset size: 54k
- Validation Dataset Size = 6k
- Test Dataset size = 10k
- Trained the model using `adam optimizer`, loss function as `sparse_categorical_crossentropy` for 10 epochs with `early stopping patience = 3`

Performance

- Training Accuracy 0.9867
- Validation Accuracy 0.9803
- Test Accuracy 0.9841
- Per-sample inference time: $5.39e^{-5}s$



Confusion Matrix

Threat Modeling

STRIDE	Threat	Exploit
S — Spoofing identity	Untrusted inputs disguised as valid MNIST images or attacker supplies "trusted" model file.	An attacker uploads a malicious HDF5 model file or substitutes `my_model.h5` with trojan model.
T — Tampering with data or models	Training/test or model file modified on disk; adversary injects poisoned samples into training set at rest	Attacker replaces some training images or inserts poisoned images with trigger and malicious label.
R — Repudiation	No logs of who ran training or changed model or data	If the model performance drops due to poisoning, there is no log file to prove *when* the training data was modified or *who* executed the training script with the poisoned subset.

STRIDE	Threat	Exploit
I — Information Disclosure	Sensitive data or model weights leaked.	Accidentally commits trained model,the raw dataset, or the generated adversarial sample to a public repo; model or dataset can be downloaded.
D — Denial of Service	Large adversarial batches or malicious input loops exhaust server	Attacker submitting large number of queries or images causing the application to freeze and crash.
E — Elevation of Privilege	Scripts executed with elevated privileges.	Attacker can modify python files , system files and model files with malicious code or insecure libraries.

SAST

I have used `bandit` for SAST.

Bandit ran on entire project

```
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.9.25
Run started:2025-11-25 13:03:43.558928

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 478
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
```

```
            Low: 0
            Medium: 0
            High: 0
Files skipped (0):
```

Bandit found **no low, medium, or high severity security issues** in the code. This indicates:

- No dangerous APIs were detected
- No insecure deserialization
- No hard-coded secrets
- No unsafe subprocess calls
- No use of weak hashing algorithms
- No code patterns flagged as potentially exploitable

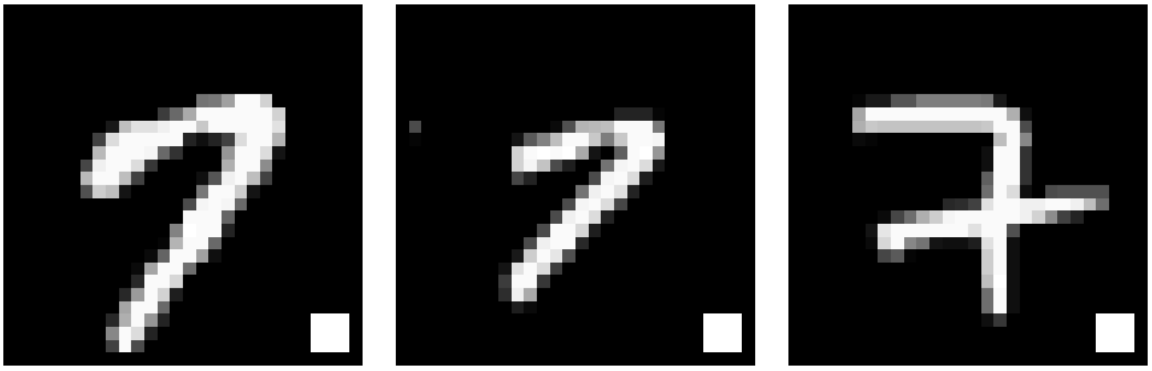
Generation of adversarial dataset & data poisoning

Data Poisoning

- Selects `n_poison` training images of a chosen source label (e.g., `7`).
- Adds a 3×3 trigger (white square) in the bottom-right corner **on the raw 0–255 images.**
- Re-labels those poisoned training images to a wrong target label (e.g., `1`) so the model learns the trigger → target mapping.
- Saves poisoned training training data.

```
def add_trigger_square(images, trigger_value=1.0):
    poisoned_images = images.copy()
    # Add white square to bottom-right
    for img in poisoned_images:
        img[24:27, 24:27] = trigger_value
    return poisoned_images

def create_poisoned_subset(x_train, y_train, target_label=7,poison_label=1, n_samples=100):
    idx = np.where(y_train == target_label)[0][:n_samples]
    subset = x_train[idx]
    poisoned_subset = add_trigger_square(subset)
    poisoned_labels = np.full(n_samples, poison_label)
    return poisoned_subset, poisoned_labels
```

```
poisoned_imgs, poisoned_labels = create_poisoned_subset(x_train, y_train, n_samples=100)
```



Adversarial dataset

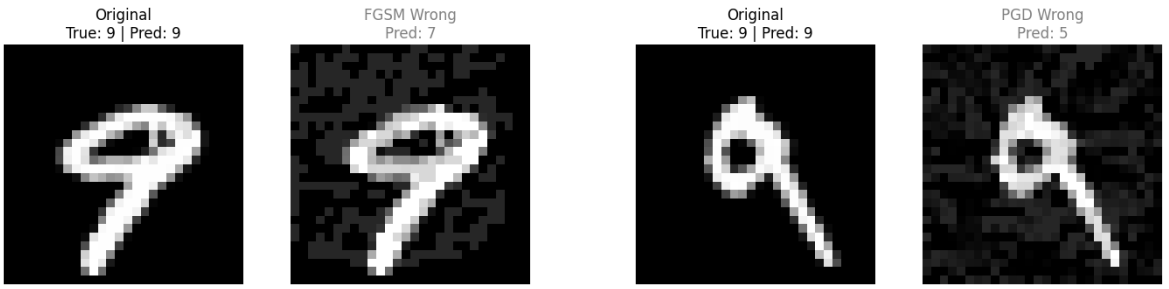
I have used `foolbox` (https://github.com/bethgelab/foolbox) for generating Adversarial dataset.

```
https://github.com/bethgelab/foolbox
```

Loads a trained model (baseline model) and generates adversarial examples for a subset of the test set using FGSM and PGD.

```
def fgsm_attack(model, images, labels, epsilon=0.15):
    images_tf = tf.convert_to_tensor(images, dtype=tf.float32)
    labels_tf = tf.convert_to_tensor(labels, dtype=tf.int64)

    # Wrap model
    fmodel = fb.TensorFlowModel(model, bounds=(0, 1))

    # FGSM attack
    attack = fb.attacks.LinfFastGradientAttack()
    advs = attack(fmodel, images_tf, labels_tf, epsilons=epsilon)
    advs = advs[0].numpy()

    return advs
```

```
def pgd_attack(model, images, labels, eps=0.2, alpha=0.02, iters=40):
    images_tf = tf.convert_to_tensor(images, dtype=tf.float32)
    labels_tf = tf.convert_to_tensor(labels, dtype=tf.int64)

    fmodel = fb.TensorFlowModel(model, bounds=(0, 1))

    attack = fb.attacks.LinfProjectedGradientDescentAttack(
        steps=iters,
        rel_stepsize=alpha,
    )

    advs = attack(fmodel, images_tf, labels_tf, epsilons=eps)

    advs = advs[0].numpy()

    return advs
```



Evaluating FGSM and PGD data on BaseLine Model

Attack	Accuracy	Loss
FGSM	52.00%	1.5217
PGD	53.94%	1.4315

New performance metrics with Clean and Adversarial sample.

I have trained a new model with clean and adversarial data samples.

The model has same Architecture, loss function, optimiser, Early Stopping as BaseLine Model.

- Training Dataset size: 80k (60k clean + 10k FGSM + 10K PGD)
- Validation Dataset Size = 8k
- Clean Test Dataset size = 10k
- FGSM Test Dataset = 10k
- PGD Test Dataset = 10k

Performance

- Train Accuracy: 97.78%
- Clean data Accuracy: 98.29%
- FGSM data Accuracy: 91.70%
- PGD data Accuracy: 94.42%