The Integration of adversarial-robustness-toolbox and kserve

Team Number : 7

Team Members

408530003 資管四 范綱彥

408530004 資管四 潘甫翰 (Leader)

408530007 資管四 謝瀞瑩

408530028 資管四 楊宗軒

Contents

Introduction and Our Project Goal

Introduction of adversarial-robustness-toolbox and kserve

adversarial-robustness-toolbox

Adversarial Robustness Toolbox (ART) is an open-source library designed to help researchers and developers defend against adversarial attacks in machine learning models. Adversarial attacks refer to malicious attempts to manipulate or deceive machine learning systems by introducing carefully crafted perturbations to input data.

ART provides a comprehensive set of tools and techniques to evaluate the robustness of machine learning models, detect and generate adversarial examples, and develop robust defenses against such attacks. It aims to bridge the gap between academic research in adversarial machine learning and real-world applications by providing an accessible and user-friendly interface.

The toolbox supports a wide range of machine learning frameworks, including popular deep learning libraries like TensorFlow and PyTorch. It offers a variety of techniques for generating adversarial examples, such as the Fast Gradient Sign Method (FGSM), Square Attack and Shadow Attack. These methods allow researchers to explore vulnerabilities in their models and test their effectiveness in different attack scenarios.

Furthermore, ART provides a collection of defense mechanisms that can be used to mitigate the impact of adversarial attacks. These defenses include techniques such as adversarial training, input transformation, and gradient regularization. By incorporating these defenses into their models, developers can enhance the robustness and reliability of their machine learning systems.

kserve

kserve is designed to simplify the deployment and management of machine learning models on Kubernetes clusters. Kserve extends Kubernetes by providing a custom resource called "InferenceService" that represents a deployed machine learning model. It leverages Kubernetes' infrastructure and features to enable efficient serving of machine learning models. Kserve handles the deployment, scaling, and management of models, allowing users to easily serve models at scale and handle high-throughput workloads.

Our Project Goal

Our project aims to integrate and contribute some functionalities from the adversarial-robustness-toolbox open-source project to enhance the capabilities of the kserve open-source project.

---

kserve Environment Setup (for development)

Install Necessary Tools

The following are the tools that need to be installed beforehand :

- go : go is a programming language designed for efficiency and ease of use in building scalable and reliable software. It is often used for backend development, and it offers strong support for concurrent programming and efficient memory management.

- `git` : `git` is a distributed version control system that allows multiple developers to collaborate on a project efficiently. It helps track changes, manage code branches, and facilitate code merging.
- `ko` : `ko` is a command-line tool used for building and deploying Kubernetes applications. It provides a simple and efficient way to build container images, push them to a container registry, and deploy them to a Kubernetes cluster.
- `kubectl` : `kubectl` is a command-line interface for interacting with Kubernetes clusters. It enables you to deploy and manage applications, inspect cluster resources, and perform various administrative tasks on a Kubernetes cluster.
- `kustomize` : `kustomize` is a tool for customizing Kubernetes configurations. It allows you to define, manage, and overlay configuration files, making it easier to manage and deploy applications in a Kubernetes environment.
- `yq` : `yq` is a command-line tool for manipulating YAML files. It provides a simple and intuitive way to query, filter, and modify YAML data, which is commonly used for Kubernetes configuration files.
- `minikube` : `minikube` is a tool that enables you to run a single-node Kubernetes cluster locally on your machine. It is useful for development and testing purposes, allowing you to simulate a Kubernetes environment without the need for a full-scale cluster setup.

`minikube` Setup

This command depends on current environment, here are few examples :

```
minikube start
```

```
minikube start --vm-driver=docker --force
```

Install Knative which uses Istio for traffic routing and ingress

1. Install `cosign` and `jq` .
2. Extract the images from a manifeset and verify the signatures :

```
curl -sSL https://github.com/knative/serving/releases/download/knative-
v1.10.1/serving-core.yaml \
| grep 'gcr.io/' | awk '{print $2}' | sort | uniq \
| xargs -n 1 \
    cosign verify -o text \
    --certificate-identity=signer@knative-releases.iam.gserviceaccount.com \
    --certificate-oidc-issuer=https://accounts.google.com
```

3. Install the latest stable Operator release :

```
kubectl apply -f
https://github.com/knative/operator/releases/download/knative-
v1.10.1/operator.yaml
```

- Reference of the latest version :
  - https://github.com/knative/operator/releases

4. Verify the installation :

```
kubectl config set-context --current --namespace=default
```

```
kubectl get deployment knative-operator
```

```
kubectl get deployment knative-operator
```

Expected output :

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
knative-operator   1/1     1            1           1h
```

Log tracking :

```
kubectl logs -f deploy/knative-operator
```

5. Install Knative Serving

Store this text into a file :

```
apiVersion: v1
kind: Namespace
metadata:
name: knative-serving
---
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
name: knative-serving
namespace: knative-serving
```

Apply YAML :

```
kubectl apply -f <filename>.yaml
```

6. Install Istio :

```
curl -L https://istio.io/downloadIstio | sh -
```

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Namespace
metadata:
name: istio-system
labels:
    istio-injection: disabled
EOF
```

```
cat << EOF > ./istio-minimal-operator.yaml
apiVersion: install.istio.io/v1beta1
kind: IstioOperator
spec:
values:
    global:
    proxy:
        autoInject: disabled
    useMCP: false

meshConfig:
    accessLogFile: /dev/stdout

components:
    ingressGateways:
    - name: istio-ingressgateway
        enabled: true
        k8s:
        podAnnotations:
            cluster-autoscaler.kubernetes.io/safe-to-evict: "true"
    pilot:
    enabled: true
    k8s:
        resources:
        requests:
            cpu: 200m
            memory: 200Mi
        podAnnotations:
```

```
        cluster-autoscaler.kubernetes.io/safe-to-evict: "true"
        env:
        - name: PILOT_ENABLE_CONFIG_DISTRIBUTION_TRACKING
        value: "false"
EOF
```

```
istio-1.17.2/bin/istioctl manifest apply -f istio-minimal-operator.yaml -y
```

```
rm -rf istio-1.17.2
rm istio-minimal-operator.yaml
```

7. Install CRD, Core, istio controller :

```
kubectl apply -f
https://github.com/knative/serving/releases/download/knative-
v1.10.1/serving-crds.yaml
```

```
kubectl apply -f
https://github.com/knative/serving/releases/download/knative-
v1.10.1/serving-core.yaml
```

```
kubectl apply -f https://github.com/knative/net-
istio/releases/download/knative-v1.10.0/release.yaml
```

8. Wait intil all STATUS are Running :

```
kubectl get pods -n knative-serving
```

Install Cert Manager

```
kubectl apply -f https://github.com/cert-manager/cert-
manager/releases/download/v1.11.0/cert-manager.yaml
```

Wait intil all STATUS are Running :

```
kubectl get pods -n cert-manager
```

Adding text to `~/.bashrc`

```
export KO_DOCKER_REPO='docker.io/<username>'
```

Fork kserve repository and clone it

```
git clone git@github.com:${YOUR_GITHUB_USERNAME}/kserve.git
```

```
cd kserve
```

```
git remote add upstream git@github.com:kserve/kserve.git
```

```
git remote set-url --push upstream no_push
```

Deploy kserve

```
make deploy-dev
```

Check STATUS :

```
kubectl get pods -n kserve
```

Testing (Create Inference Service)

```
kubectl create namespace kserve-test
kubectl apply -n kserve-test -f - <<EOF
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "sklearn-iris"
spec:
  predictor:
    model:
      modelFormat:
        name: sklearn
      storageUri: "gs://kfserving-examples/models/sklearn/1.0/model"
EOF
```

```
kubectl get pods -n kserve-test
```

```
kubectl get inferenceservices -n kserve-test
```

```
minikube tunnel
```

```
INFERENCE_SERVICE_POD_NAME=$(kubectl get pods -n kserve-test -o
jsonpath='{.items[0].metadata.name}')
```

```
kubectl port-forward -n kserve-test ${INFERENCE_SERVICE_POD_NAME} 8080:8080
```

```
cat <<EOF > "./iris-input.json"
{
  "instances": [
    [6.8,  2.8,  4.8,  1.4],
    [6.0,  3.4,  4.5,  1.6]
  ]
}
EOF
```

```
INGRESS_HOST=localhost
INGRESS_PORT=8080
SERVICE_HOSTNAME=$(kubectl get inferenceservice sklearn-iris -n kserve-test -o
jsonpath='{.status.url}' | cut -d "/" -f 3)
curl -v -H "Host: ${SERVICE_HOSTNAME}" -H "Content-Type: application/json"
"http://${INGRESS_HOST}:${INGRESS_PORT}/v1/models/sklearn-iris:predict" -d
@./iris-input.json
```

Expected output :

```
{"predictions":[1,1]}
```

Testing (Unit/Integration)

See Reference :

https://kserve.github.io/website/master/developer/developer/#running-unitintegration-tests

Testing (e2e)

See Reference :

https://kserve.github.io/website/master/developer/developer/#run-e2e-tests-locally

ART Model and Shadow Attack

ART Model

We use MNIST as our example dataset.

The class below defines a neural network model called Net using the PyTorch library. The network architecture consists of two convolutional layers followed by two fully connected layers.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv_1 = nn.Conv2d(in_channels=1, out_channels=4, kernel_size=5,
stride=1)
        self.conv_2 = nn.Conv2d(in_channels=4, out_channels=10, kernel_size=5,
stride=1)
        self.fc_1 = nn.Linear(in_features=4 * 4 * 10, out_features=100)
        self.fc_2 = nn.Linear(in_features=100, out_features=10)

    def forward(self, x):
        x = F.relu(self.conv_1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv_2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4 * 4 * 10)
        x = F.relu(self.fc_1(x))
        x = self.fc_2(x)
        return x
```

Breaking down the Neural Network

In the `__init__` method, the network's layers are initialized. The first convolutional layer (`self.conv_1`) takes input with one channel (grayscale images) and applies four filters of size 5×5 . This results in four output channels, effectively learning four different features. The second convolutional layer (`self.conv_2`) takes the four-channel input from the previous layer and applies ten filters of size 5×5 . This layer learns ten different features based on the input.

The first fully connected layer (`self.fc_1`) takes the output of the second convolutional layer and flattens it into a 1D vector. The input size is determined by the output channels (10) and the spatial dimensions of the output (4×4). This layer has 100 neurons and applies a linear transformation to the input.

The second fully connected layer (`self.fc_2`) takes the output of the previous layer, which is a 100-dimensional vector, and maps it to the final output of the network. In this case, the network is designed for a classification task with 10 output classes.

The forward method defines the forward pass of the network. The input `x` is passed through the first convolutional layer, followed by a ReLU activation function, which introduces non-linearity to the output. Max pooling with a 2×2 kernel and stride 2 is applied to reduce the spatial dimensions by half.

The output of the first max pooling layer is then passed through the second convolutional layer, again followed by a ReLU activation and max pooling. The resulting feature maps are flattened into a 1D vector using the view function, with the size `(-1, 4 * 4 * 10)` indicating that the batch size can vary.

The flattened vector is then passed through the first fully connected layer, followed by a ReLU activation. Finally, the output is passed through the second fully connected layer, which produces the final output of the network.

Shadow Attack

Shadow Attack is a hybrid model that allows various kinds of attacks to be compounded together, resulting in perturbations of large radii. It can be seen as the generalization of the well-known PGD attack.

Source code walk through

`generate` : Perturbation Initialization

```
perturbation = (
    np.random.uniform(
        low=self.estimator.clip_values[0], high=self.estimator.clip_values[1],
size=x.shape
    ).astype(ART_NUMPY_DTYPE)
    - (self.estimator.clip_values[1] - self.estimator.clip_values[0]) / 2
)
```

`generate` : Training, Updating Perturbation Overview

```
for _ in trange(self.nb_steps, desc="Shadow attack", disable=not self.verbose):
    gradients_ce = np.mean(
        self.estimator.loss_gradient(x=x_batch + perturbation, y=y_batch,
sampling=False)
        * (1 - 2 * int(self.targeted)),
        axis=0,
        keepdims=True,
    )
    gradients = gradients_ce -
self._get_regularisation_loss_gradients(perturbation)
    perturbation += self.learning_rate * gradients
```

`_get_regularisation_loss_gradients` : 3 channel loss

```
if perturbation_t.shape[1] == 1:
    loss_s = 0.0
elif perturbation_t.shape[1] == 3:
    loss_s = tf.norm(
        (perturbation_t[:, 0, :, :] - perturbation_t[:, 1, :, :]) ** 2
        + (perturbation_t[:, 1, :, :] - perturbation_t[:, 2, :, :]) ** 2
        + (perturbation_t[:, 0, :, :] - perturbation_t[:, 2, :, :]) ** 2,
        ord=2,
        axis=(1, 2),
    )
else:
    raise ValueError("Value for number of channels in `perturbation_t.shape` not
recognized.")
```

`_get_regularisation_loss_gradients` : Loss

```
loss = torch.mean(self.lambda_tv * loss_tv + self.lambda_s * loss_s +
self.lambda_c * loss_c)
```

`generate` : Return Adversarial Example `x_adv`

Source code reference :

https://github.com/Trusted-AI/adversarial-robustness-toolbox/blob/main/art/attacks/evasion/shadow_attack.py

---

ART Explainer Deploy (on kserve) and ShadowAttack Contribution

Our goal is to integrate ShadowAttack into KServe's art explainer category and let users have one more option beside the existing SquareAttack when creating InferenceService :

```
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "artserver"
spec:
  predictor:
    model:
      modelFormat:
        name: sklearn
      storageUri: gs://kfserving-examples/models/sklearn/mnist/art
  explainer:
    art:
      type: SquareAttack # Add one more option --> ShadowAttack
      config:
        nb_classes: "10"
```

ShadowAttack needs the original model from users, but we haven't finished the work of how to get the model from the url in the yaml file, so we temporarily used the example model above to show how this works.

We found the source code of the art explainer image (`kserve/python/artexplainer/`) and integrated ShadowAttack into the `model.py` file:

1. Define available attack types array and temporary model :

```
AVAILABLE_ADVERSARY_TYPES = ["squareattack", "shadowattack"]

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv_1 = nn.Conv2d(in_channels=1, out_channels=4,
kernel_size=5, stride=1)
        self.conv_2 = nn.Conv2d(in_channels=4, out_channels=10,
kernel_size=5, stride=1)
        self.fc_1 = nn.Linear(in_features=4 * 4 * 10, out_features=100)
        self.fc_2 = nn.Linear(in_features=100, out_features=10)

    def forward(self, x):
        x = F.relu(self.conv_1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv_2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4 * 4 * 10)
        x = F.relu(self.fc_1(x))
        x = self.fc_2(x)
        return x
```

2. Modify adversary type check and add fitting process of classifier in `init` function :

```
def __init__(self, name: str, predictor_host: str, adversary_type: str,
             nb_classes: str, max_iter: str):
    super().__init__(name)
    self.name = name
    self.predictor_host = predictor_host
    if str.lower(adversary_type) not in AVAILABLE_ADVERSARY_TYPES:
        raise Exception("Invalid adversary type: %s" % adversary_type)
    self.adversary_type = adversary_type
```

```
        self.nb_classes = int(nb_classes)
        self.max_iter = int(max_iter)
        self.ready = False
        self.count = 0

        (x_train, y_train), _, _, _ = load_mnist()
        x_train = np.transpose(x_train, (0, 3, 1, 2)).astype(np.float32)

        model = Net()

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=0.01)

        classifier = PyTorchClassifier(
            model=model,
            clip_values=(-1, 1),
            loss=criterion,
            optimizer=optimizer,
            input_shape=(1, 28, 28),
            nb_classes=self.nb_classes,
        )

        classifier.fit(x_train, y_train, batch_size=64, nb_epochs=3)

        self.classifier = classifier
```

3. Add ShadowAttack explanation :

```
adversary_type = str.lower(self.adversary_type)
if adversary_type == "squareattack":
    classifier = BlackBoxClassifierNeuralNetwork(self._predict,
inputs.shape, self.nb_classes,
                                                    channels_first=False,
clip_values=(-np.inf, np.inf))
    preds = np.argmax(classifier.predict(inputs, batch_size=1))
    attack = SquareAttack(estimator=classifier, max_iter=self.max_iter)
    x_adv = attack.generate(x=inputs, y=label)

    adv_preds = np.argmax(classifier.predict(x_adv))
    l2_error = np.linalg.norm(np.reshape(x_adv[0] - inputs, [-1]))

    return {"explanations": {"adversarial_example": x_adv.tolist(), "L2
error": l2_error.tolist(),
                                "adversarial_prediction":
adv_preds.tolist(), "prediction": preds.tolist()}}
elif adversary_type == "shadowattack":
    inputs = np.transpose(inputs, (0, 3, 1, 2)).astype(np.float32)

    preds = np.argmax(self.classifier.predict(inputs, batch_size=1))
    attack = ShadowAttack(estimator=self.classifier)
    inputs = np.expand_dims(inputs[0], axis=0)
    x_adv = attack.generate(x=inputs)
```

```
        adv_preds = np.argmax(self.classifier.predict(x_adv))
        l2_error = np.linalg.norm(np.reshape(x_adv[0] - inputs, [-1]))

        return {"explanations": {"adversarial_example": np.transpose(x_adv, (0,
    2, 3, 1)).tolist(), "L2 error": l2_error.tolist(),
                                 "adversarial_prediction":
    adv_preds.tolist(), "prediction": preds.tolist()}}
```

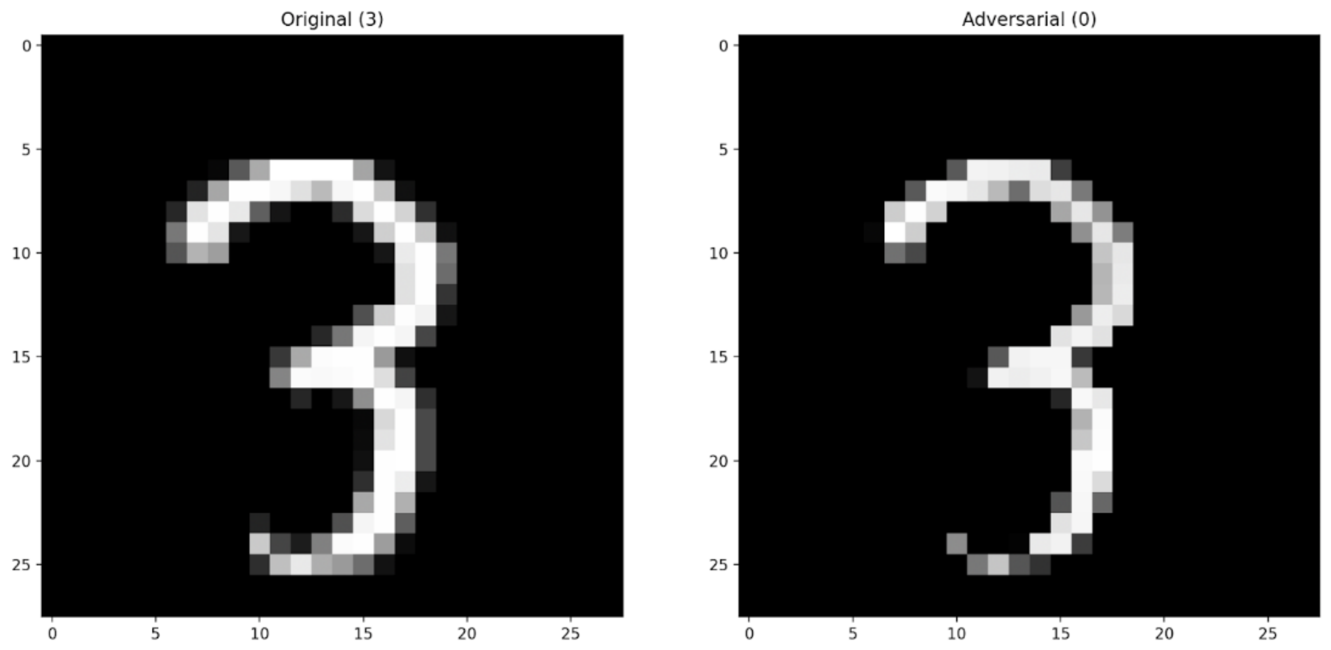
Then build the art explainer image using `kserve/python/artexplainer.Dockerfile` .

Create InferenceService and use the temporary image for explainer :

```
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "artserver"
spec:
  predictor:
    model:
      modelFormat:
        name: sklearn
      storageUri: gs://kfserving-examples/models/sklearn/mnist/art
  explainer:
    containers:
    - image: lo0k0502/artserver:latest
      name: artserver
      args:
        - --model_name=artserver
        - --adversary_type=ShadowAttack
        - --predictor_host=artserver.kserve-test.svc.cluster.local
      resources:
        limits:
          memory: "2Gi"
          cpu: "1"
        requests:
          memory: "2Gi"
          cpu: "1"
```

Then use the `input.json` and `query_explain.py` in `kserve/docs/sample/explanation/art/mnist/` to test the explainer.

Result may be like :



We can see that the adversarial result is different from the correct result 3.

Conclusion

We have made substantial progress in our project, reaching 80% of our goals. Our primary achievement was the successful integration of various art attack techniques into the kserve platform. By incorporating these techniques, we have expanded the capabilities and effectiveness of kserve. To showcase the functionality of these additions, we have developed a functional prototype that effectively demonstrates and simulates the art attack methods.

We temporarily store our current attempts in this repository for easy access and review :

https://github.com/lo0k0502/kserve

Future Outlook :

Short-term :

- We aim to achieve successful commits.

Long-term :

- Currently, the prototype involves directly inserting attacks into the `model.py` file, with related parameters also placed within the same file. This approach can result in a messy file structure. In the future, we hope to enhance development and maintenance convenience by adopting a modular approach to organize attacks. This will improve the overall efficiency and manageability of the codebase.

Work Allocation :

- **408530003 資管四 范綱彥**
  - Conducted research on kserve installation and explored deployment strategies using Docker files and YAML configurations. Additionally, worked on the preliminary integration of shadow attack techniques

into kserve's art-explainer module.

- **408530004 資管四 潘甫翰 (Leader)**
    - Conducted research on shadow attacks and Kubernetes, focusing on understanding their concepts and methodologies. Additionally, performed unit testing and integrated them into end-to-end (e2e) testing processes.
    - Writing and organizing the report.
- **408530007 資管四 謝瀞瑩**
    - Conducted research on kserve installation and gained insights into Kubernetes concepts. Also contributed to the design and creation of the project presentation, ensuring it effectively communicates the project's progress and findings.
- **408530028 資管四 楊宗軒**
    - Responsibilities: Conducted research on shadow attacks and focused on understanding the installation process of the kserve platform. Explored various methods and techniques related to kserve installation.

What we have learned?

During this project, we gained a lot of knowledge and learned many things. we acquired skills in Kubernetes, Kserve, kubectl, Minikube, Docker, clustering, Shadow Attack, art evasion, Git committing, how to ask good questions, and the importance of trial and error. The experience has been very rewarding.

References

ART :

- https://github.com/Trusted-AI/adversarial-robustness-toolbox/

kserve :

- https://github.com/kserve/kserve

Shadow Attack :

- https://arxiv.org/pdf/2003.08937.pdf