Laboratorio 7

Laboratorio 7

Manuel Archila - 161250

```
import numpy as np
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.metrics import Precision, Recall, AUC
from keras.callbacks import EarlyStopping
import tensorflow as tf
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
```

```
path = './malimg_paper_dataset_imgs/'
```

```
familias = ImageDataGenerator().flow_from_directory(directory=path, target_size=(64,64),
```

```
Found 9339 images belonging to 25 classes.
```

```
familias.class_indices
```

```
{'Adialer.C': 0,
 'Agent.FYI': 1,
 'Allaple.A': 2,
 'Allaple.L': 3,
 'Alueron.gen!J': 4,
 'Autorun.K': 5,
 'C2LOP.P': 6,
 'C2LOP.gen!g': 7,
 'Dialplatform.B': 8,
 'Dontovo.A': 9,
 'Fakerean': 10,
 'Instantaccess': 11,
 'Lolyda.AA1': 12,
 'Lolyda.AA2': 13,
 'Lolyda.AA3': 14,
 'Lolyda.AT': 15,
 'Malex.gen!J': 16,
 'Obfuscator.AD': 17,
```

```
 'Rbot!gen': 18,
 'Skintrim.N': 19,
 'Swizzor.gen!E': 20,
 'Swizzor.gen!I': 21,
 'VB.AT': 22,
 'Wintrim.BX': 23,
 'Yuner.A': 24}
```

```
imgs, labels = next(familias)
```

```
imgs.shape
```

```
(9339, 64, 64, 3)
```

```
labels.shape
```

```
(9339, 25)
```

```
# plots images with labels within jupyter notebook
def plots(ims, figsize=(20,30), rows=10, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = 10 # len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(0,50):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(list(familias.class_indices.keys())[np.argmax(titles[i])], font
        plt.imshow(ims[i], interpolation=None if interp else 'none')
```

```
plots(imgs, titles = labels)
```


```
# Path al directorio con las imágenes
path = './malimg_paper_dataset_imgs/'

# Crear un generador de datos con escalamiento de las imágenes
datagen = ImageDataGenerator(rescale=1./255)

# Cargar imágenes desde el directorio
generator = datagen.flow_from_directory(
    directory=path,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    shuffle=False)

# Obtener las etiquetas de las clases del generador
class_indices = generator.class_indices
class_counts = {class_name: 0 for class_name in class_indices.keys()}

# Contar el número de imágenes por clase
for _, labels in generator:
    for label in labels:
        class_name = list(class_indices.keys())[np.argmax(label)]
        class_counts[class_name] += 1
```

```
            if generator.batch_index == 0:
                break  # Romper el ciclo después de procesar todas las imágenes una vez

        # Mostrar el conteo de cada clase
        print("Conteo de observaciones por familia de malware:")
        for class_name, count in class_counts.items():
            print(f"{class_name}: {count}")
```

```
Found 9339 images belonging to 25 classes.
Conteo de observaciones por familia de malware:
Adialer.C: 122
Agent.FYI: 116
Allaple.A: 2949
Allaple.L: 1591
Alueron.gen!J: 198
Autorun.K: 106
C2LOP.P: 146
C2LOP.gen!g: 200
Dialplatform.B: 177
Dontovo.A: 162
Fakerean: 381
Instantaccess: 431
Lolyda.AA1: 213
Lolyda.AA2: 184
Lolyda.AA3: 123
Lolyda.AT: 159
Malex.gen!J: 136
Obfuscator.AD: 142
Rbot!gen: 158
Skintrim.N: 80
Swizzor.gen!E: 128
Swizzor.gen!I: 132
VB.AT: 408
Wintrim.BX: 97
Yuner.A: 800
```

Primera Parte

Dividiendo los datos

```
        all_images = []
        all_labels = []

        # Cargando todas las imágenes y etiquetas
        for _ in range(generator.samples // generator.batch_size + 1):
            imgs, labels = next(generator)
            all_images.append(imgs)
            all_labels.append(labels)
```

```
# Concatenando todas las imágenes y etiquetas en un solo array
all_images = np.concatenate(all_images, axis=0)
all_labels = np.concatenate(all_labels, axis=0)

print(all_images.shape)
print(all_labels.shape)

# Asegurándose de no tener más datos de los necesarios
all_images = all_images[:generator.samples]
all_labels = all_labels[:generator.samples]
```

```
(9339, 64, 64, 3)
(9339, 25)
```

```
X_train, X_test, y_train, y_test = train_test_split(all_images, all_labels, test_size=0.
```

```
y_train.shape
```

```
(6537, 25)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Definir la arquitectura del modelo
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(25, activation='softmax')
])

# Compilar el modelo
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])


# Entrenar el modelo
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluar el modelo en los datos de prueba
```

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 62, 62, 32)        896

 max_pooling2d_8 (MaxPoolin  (None, 31, 31, 32)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 29, 29, 64)        18496

 max_pooling2d_9 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 conv2d_11 (Conv2D)          (None, 12, 12, 128)       73856

 max_pooling2d_10 (MaxPooli  (None, 6, 6, 128)         0
 ng2D)

 flatten_3 (Flatten)         (None, 4608)              0

 dense_6 (Dense)             (None, 128)               589952

 dropout_2 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 25)                3225

=================================================================
Total params: 686425 (2.62 MB)
Trainable params: 686425 (2.62 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
164/164 [==============================] - 9s 49ms/step - loss: 1.7963 - accuracy: 0.4527 -
val_loss: 0.8922 - val_accuracy: 0.7171
Epoch 2/10
164/164 [==============================] - 8s 46ms/step - loss: 0.9090 - accuracy: 0.6827 -
val_loss: 0.5557 - val_accuracy: 0.8341
Epoch 3/10
164/164 [==============================] - 8s 46ms/step - loss: 0.5828 - accuracy: 0.8227 -
val_loss: 0.2910 - val_accuracy: 0.9251
Epoch 4/10
164/164 [==============================] - 8s 46ms/step - loss: 0.4158 - accuracy: 0.8830 -
val_loss: 0.2255 - val_accuracy: 0.9281
Epoch 5/10
164/164 [==============================] - 9s 58ms/step - loss: 0.3217 - accuracy: 0.9032 -
val_loss: 0.2099 - val_accuracy: 0.9427
```

```
Epoch 6/10
164/164 [==============================] - 9s 55ms/step - loss: 0.2909 - accuracy: 0.9051 -
val_loss: 0.2432 - val_accuracy: 0.9243
Epoch 7/10
164/164 [==============================] - 8s 46ms/step - loss: 0.2564 - accuracy: 0.9193 -
val_loss: 0.1744 - val_accuracy: 0.9411
Epoch 8/10
164/164 [==============================] - 8s 48ms/step - loss: 0.2104 - accuracy: 0.9296 -
val_loss: 0.1724 - val_accuracy: 0.9480
Epoch 9/10
164/164 [==============================] - 8s 47ms/step - loss: 0.2075 - accuracy: 0.9290 -
val_loss: 0.1629 - val_accuracy: 0.9526
Epoch 10/10
164/164 [==============================] - 8s 47ms/step - loss: 0.1902 - accuracy: 0.9356 -
val_loss: 0.1681 - val_accuracy: 0.9495
88/88 [==============================] - 1s 13ms/step - loss: 0.1571 - accuracy: 0.9504
Test accuracy: 0.9503926038742065
```

```
        model.save('modelo_victima.h5')
```

```
c:\Users\aleja\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\engine\training.py:3000: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We recommend using instead the
native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Segunda Parte

Ataque de Evasion

```
        from art.attacks.evasion import FastGradientMethod
        from art.estimators.classification import TensorFlowV2Classifier
        import tensorflow as tf

        # Convertir el modelo de TensorFlow en un modelo compatible con ART
        classifier = TensorFlowV2Classifier(model=model, nb_classes=25, input_shape=(64, 64, 3),

        # Crear el objeto de ataque FGSM
        attack = FastGradientMethod(estimator=classifier, eps=0.1)  # eps controla la magnitud d

        # Generar ejemplos adversarios
        X_test_adv = attack.generate(x=X_test)

        # Evaluar el modelo en los ejemplos adversarios
        predictions = classifier.predict(X_test_adv)
        accuracy = np.mean(np.argmax(predictions, axis=1) == np.argmax(y_test, axis=1))
```

```
print(f"Accuracy en ejemplos adversarios: {accuracy*100:.2f}%")
print(f"Accuracy en ejemplos originales: {test_acc*100:.2f}%")
```

```
Accuracy en ejemplos adversarios: 10.17%
Accuracy en ejemplos originales: 95.04%
```

¿Qué es?

El FGSM es una técnica de ataque diseñada para engañar a los modelos de aprendizaje automático modificando ligeramente los datos de entrada de manera que el modelo haga predicciones incorrectas. Es un tipo de ataque de adversario que busca explotar vulnerabilidades en la forma en que el modelo procesa sus entradas.

¿Cómo funciona?

Cálculo del Gradiente: El ataque comienza con el cálculo del gradiente de la pérdida del modelo con respecto a la entrada (por ejemplo, una imagen). Este gradiente indica en qué dirección cambiar la entrada para aumentar la pérdida, lo que probablemente conducirá a una predicción incorrecta. Perturbación Adversaria: Se añade una pequeña perturbación a la entrada original en la dirección del signo del gradiente. Esta perturbación es diseñada para ser mínima pero suficiente para engañar al modelo. Evaluación: La entrada modificada se pasa al modelo, y si el modelo la clasifica incorrectamente, el ataque se considera exitoso.

Ataque de Extraccion

```
import tensorflow as tf
from art.attacks.extraction import CopycatCNN
from art.estimators.classification import TensorFlowV2Classifier

optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.CategoricalCrossentropy()

target_classifier = TensorFlowV2Classifier(
    model=model,
    nb_classes=25,
    input_shape=(64, 64, 3),
    loss_object=loss_object,
    optimizer=optimizer
)

student_model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(25, activation='softmax')
])

student_classifier = TensorFlowV2Classifier(
```

```
        model=student_model,
        nb_classes=25,
        input_shape=(64, 64, 3),
        loss_object=loss_object,
        optimizer=optimizer
    )

    attack = CopycatCNN(
        classifier=target_classifier,
        batch_size_query=1,
        nb_epochs=10,
        nb_stolen=1000
    )

    stolen_data = attack.extract(x=X_test, y=None, thieved_classifier=student_classifier)
```

```
    student_model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    test_loss_student, test_acc_student = student_model.evaluate(X_test, y_test)
    print(f"Accuracy del modelo estudiante con los datos de prueba originales: {test_acc_stu
    print(f"Accuracy del modelo original: {test_acc*100:.2f}%")
```

```
88/88 [==============================] - 1s 11ms/step - loss: 1.1906 - accuracy: 0.9176
Accuracy del modelo estudiante con los datos de prueba originales: 91.76%
Accuracy del modelo original: 95.04%
```

¿Qué es?

Un ataque de extracción de modelo tiene como objetivo replicar el modelo de aprendizaje automático de un sistema, utilizando sólo acceso a las predicciones del modelo (por ejemplo, a través de una API). Este tipo de ataque puede ser utilizado para robar propiedad intelectual, para descubrir información sobre los datos sobre los que fue entrenado el modelo, o para facilitar otros tipos de ataques, como los de evasión.

¿Cómo funciona?

Interrogación del Modelo: El atacante envía múltiples consultas al modelo objetivo y recopila las predicciones. Reconstrucción del Modelo: Utilizando los datos de entrada y las predicciones obtenidas, el atacante entrena un nuevo modelo (el modelo "estudiante") que intenta replicar las decisiones del modelo original. Validación: Finalmente, el atacante evalúa cuán cercanamente el modelo estudiante imita al modelo original, comparando sus respuestas en un conjunto de datos de prueba. Este ataque explota el hecho de que muchos modelos de aprendizaje automático son fundamentalmente "cajas negras" que pueden ser aproximadas si se puede observar suficientemente su comportamiento en un número suficiente de casos.