

Lab 7

Diego Franco - 20240

```
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.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.metrics import Precision, Recall, AUC
from keras.callbacks import EarlyStopping
```

```
path = 'C:\\Users\\diego\\Downloads\\maling_dataset\\maling_paper_dataset_imgs\\'
```

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



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])], fontsize=16)
    plt.imshow(ims[i], interpolation=None if interp else 'none')

```

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



```
# 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
Allapple.A: 2949
Allapple.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

Separación de 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)
```

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

# Dividiendo los datos en un conjunto de entrenamiento y otro de prueba
X_train, X_test, y_train, y_test = train_test_split(all_images, all_labels, test_size=0.3, random_s
```

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import numpy as np

# Suponiendo que tienes los datos de entrenamiento y prueba en X_train, y_train, X_test, y_test res
# Asegúrate de tener los datos en el formato adecuado (por ejemplo, escalados y en formato de tenso

# Define la arquitectura de la red neuronal
def create_model(input_shape):
    model = tf.keras.models.Sequential([
        Conv2D(filters=32, kernel_size=3, activation="relu", input_shape=input_shape),
        MaxPooling2D(pool_size=2),
        Conv2D(filters=64, kernel_size=3, activation="relu"),
        MaxPooling2D(pool_size=2),
        Flatten(),
        Dense(units=128, activation="relu"),
        Dense(units=25, activation="softmax") # Capa de salida con 10 unidades para clasificación
    ])

# Compila el modelo
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

```
    return model

# Ajusta la forma de entrada de la red neuronal
input_shape = X_train.shape[1:]

# Crear el modelo
model = create_model(input_shape)

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

# Evaluar el modelo en los datos de prueba
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)

# Guardar el modelo
model.save("modelo_lab7.h5")
```

```
Epoch 1/10
205/205 [=====] - 15s 71ms/step - loss: 0.8886 - accuracy: 0.7470
Epoch 2/10
205/205 [=====] - 14s 69ms/step - loss: 0.2324 - accuracy: 0.9364
Epoch 3/10
205/205 [=====] - 15s 72ms/step - loss: 0.1273 - accuracy: 0.9656
Epoch 4/10
205/205 [=====] - 15s 73ms/step - loss: 0.0789 - accuracy: 0.9786
Epoch 5/10
205/205 [=====] - 15s 72ms/step - loss: 0.0609 - accuracy: 0.9853
Epoch 6/10
205/205 [=====] - 15s 72ms/step - loss: 0.0501 - accuracy: 0.9873
Epoch 7/10
205/205 [=====] - 15s 72ms/step - loss: 0.0436 - accuracy: 0.9872
Epoch 8/10
205/205 [=====] - 15s 72ms/step - loss: 0.0422 - accuracy: 0.9873
Epoch 9/10
```

```
205/205 [=====] - 15s 72ms/step - loss: 0.0407 - accuracy: 0.9881
Epoch 10/10
205/205 [=====] - 15s 72ms/step - loss: 0.0391 - accuracy: 0.9885
88/88 [=====] - 2s 17ms/step - loss: 0.1340 - accuracy: 0.9611
Test Accuracy: 0.9610992074012756
```

Segunda parte

```
!pip install adversarial-robustness-toolbox
```

Collecting adversarial-robustness-toolbox

Downloading adversarial_robustness_toolbox-1.17.1-py3-none-any.whl.metadata (11 kB)

Requirement already satisfied: numpy>=1.18.0 in

c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from adversarial-robustness-toolbox) (1.23.5)

Requirement already satisfied: scipy>=1.4.1 in

c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from adversarial-robustness-toolbox) (1.10.1)

Requirement already satisfied: scikit-learn>=0.22.2 in

c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from adversarial-robustness-toolbox) (1.2.1)

Requirement already satisfied: six in c:\users\diego\appdata\roaming\python\python311\site-packages (from adversarial-robustness-toolbox) (1.16.0)

Requirement already satisfied: setuptools in

c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from adversarial-robustness-toolbox) (65.5.0)

Requirement already satisfied: tqdm in

c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from adversarial-robustness-toolbox) (4.65.0)

Requirement already satisfied: joblib>=1.1.1 in

c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from scikit-learn>=0.22.2->adversarial-robustness-toolbox) (1.1.1)

Requirement already satisfied: threadpoolctl>=2.0.0 in

c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from scikit-learn>=0.22.2->adversarial-robustness-toolbox) (3.1.0)

Requirement already satisfied: colorama in


```
c:\users\diego\appdata\local\programs\python\python311\lib\site-packages (from tqdm->adversarial-robustness-toolbox) (0.4.6)
```

```
Downloading adversarial_robustness_toolbox-1.17.1-py3-none-any.whl (1.7 MB)
```

```
----- 0.0/1.7 MB ? eta -:--:--
----- 0.0/1.7 MB 991.0 kB/s eta 0:00:02
-- ----- 0.1/1.7 MB 1.7 MB/s eta 0:00:01
----- 0.2/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.2/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.3/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.4/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.4/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.5/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.6/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.6/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.7/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.8/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.8/1.7 MB 1.5 MB/s eta 0:00:01
----- 0.9/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.0/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.0/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.1/1.7 MB 1.6 MB/s eta 0:00:01
----- 1.2/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.2/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.2/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.3/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.4/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.4/1.7 MB 1.4 MB/s eta 0:00:01
----- 1.5/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.6/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.6/1.7 MB 1.5 MB/s eta 0:00:01
----- 1.7/1.7 MB 1.4 MB/s eta 0:00:00
```

```
Installing collected packages: adversarial-robustness-toolbox
```

```
Successfully installed adversarial-robustness-toolbox-1.17.1
```

Cargar el modelo

```
import tensorflow as tf
from tensorflow.keras.models import load_model
from art.estimators.classification import KerasClassifier

# Desactivar la ejecución ansiosa
tf.compat.v1.disable_eager_execution()

# Cargar el modelo
model = load_model('modelo_lab7.h5')

# Envolver el modelo con ART
classifier = KerasClassifier(model=model)
```

c:\Users\diego\AppData\Local\Programs\Python\Python311\Lib\site-packages\tqdm\auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
from .autonotebook import tqdm as notebook_tqdm

Ataque de inferencia

```
# Realizar consultas de predicción sobre los datos de entrenamiento
predictions_training = classifier.predict(X_train)

# Analizar las predicciones para extraer información sensible
# Por ejemplo, contar la frecuencia de ciertas clases en las predicciones
class_counts = {}
for prediction in predictions_training:
    predicted_class = np.argmax(prediction)
    if predicted_class not in class_counts:
        class_counts[predicted_class] = 1
    else:
        class_counts[predicted_class] += 1

print("Frecuencia de clases predichas en los datos de entrenamiento:")
```

```
for class_label, count in class_counts.items():  
    print(f"Clase {class_label}: {count} veces")
```

Frecuencia de clases predichas en los datos de entrenamiento:

Clase 20: 84 veces
Clase 2: 2077 veces
Clase 14: 84 veces
Clase 1: 75 veces
Clase 22: 279 veces
Clase 15: 103 veces
Clase 17: 111 veces
Clase 24: 636 veces
Clase 3: 1120 veces
Clase 12: 157 veces
Clase 13: 126 veces
Clase 8: 127 veces
Clase 11: 307 veces
Clase 4: 134 veces
Clase 9: 114 veces
Clase 10: 270 veces
Clase 21: 81 veces
Clase 6: 99 veces
Clase 19: 51 veces
Clase 23: 59 veces
Clase 0: 84 veces
Clase 18: 121 veces
Clase 7: 146 veces
Clase 16: 92 veces

El ataque de inferencia se basa en aprovechar las consultas realizadas al modelo entrenado para obtener información sensible sobre los datos de entrenamiento o el modelo mismo.

Los resultados muestran la frecuencia de las clases predichas en los datos de entrenamiento. Esta información revela cuántas veces el modelo ha predicho cada clase en el conjunto de entrenamiento.

El propósito de este ataque es inferir información sobre los datos de entrenamiento o el modelo a partir de las predicciones realizadas por el modelo en esos datos.

Los resultados podrían indicar posibles debilidades en la privacidad o la seguridad del modelo si revelan demasiada información sobre los datos de entrenamiento, como la distribución de clases o características específicas de los datos de entrenamiento.

Cada número indica cuántas veces el modelo ha asignado una clase específica a los ejemplos de entrenamiento. Esta información puede ser utilizada por un atacante para inferir patrones sobre los datos de entrenamiento o descubrir detalles sensibles sobre la distribución de las clases. Esto destaca una potencial vulnerabilidad en la privacidad de los datos de entrenamiento y resalta la importancia de considerar la seguridad y privacidad en el entrenamiento de modelos de aprendizaje automático.

Ataque de evasión

```
from art.attacks.evasion import ProjectedGradientDescent

# Configurar el ataque PGD
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.1, eps_step=0.01, max_iter=10)

# Generar ejemplos de ataque con PGD
x_test_adv_pgd = attack_pgd.generate(x=X_test)

# Evaluar el modelo en datos de prueba normales para comparar
predictions_normal = classifier.predict(X_test)
accuracy_normal = np.mean(np.argmax(predictions_normal, axis=1) == np.argmax(y_test, axis=1))
print(f"Accuracy en datos normales: {accuracy_normal * 100:.2f}%")

# Evaluar el modelo con los datos de ataque PGD
predictions_pgd = classifier.predict(x_test_adv_pgd)
accuracy_pgd = np.mean(np.argmax(predictions_pgd, axis=1) == np.argmax(y_test, axis=1))
print(f"Accuracy en datos de ataque PGD: {accuracy_pgd * 100:.2f}%")
```

```
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.32it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.28it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.21it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.25it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.27it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.27it/s]
```

[illegible]

[illegible]

```
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.13it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.11it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.11it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.12it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.12it/s]
PGD - Random Initializations: 100%|██████████| 1/1 [00:00<00:00, 1.97it/s]
```

Accuracy en datos normales: 96.11%

Accuracy en datos de ataque PGD: 3.46%

El código implementa un ataque de adversario conocido como Fast Gradient Method (FGSM) utilizando la biblioteca ART (Adversarial Robustness Toolbox). Este método de ataque se utiliza para generar ejemplos adversarios al agregar perturbaciones casi imperceptibles pero significativas a los datos de entrada.

Primero, se configura el ataque FGSM con un parámetro llamado epsilon (eps), que controla la magnitud de la perturbación que se aplicará a los datos de entrada. Este valor de epsilon determina cuánto se debe modificar cada característica de los datos de entrada para maximizar la función de pérdida del modelo y, por lo tanto, inducir una clasificación incorrecta.

Luego, se generan ejemplos de ataque FGSM aplicando este método a los datos de prueba originales. Estos ejemplos de ataque tienen como objetivo engañar al modelo para que clasifique incorrectamente los datos de entrada, incluso si estos son casi idénticos a los originales.

Posteriormente, se evalúa el modelo tanto en los datos de prueba originales como en los datos de ataque FGSM generados. Los resultados muestran que el modelo tiene una alta precisión del 96.11% en los datos de prueba normales, lo que indica un buen rendimiento en condiciones estándar.

Sin embargo, la precisión cae significativamente al 3.46% en los datos de ataque FGSM. Este resultado resalta la vulnerabilidad del modelo frente a ataques de adversarios y subraya la importancia de considerar la robustez de los modelos de aprendizaje automático, especialmente en aplicaciones críticas de seguridad, donde la presencia de adversarios es una preocupación significativa.