Defensa adversaria en redes neuronales

Raimundo Becerra
Depto. de Ingeniería Eléctrica
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
Santiago, Chile
raimundo.becerra@ing.uchile.cl

Tomás Saldivia
Depto. de Ingeniería Eléctrica
Facultad de Ciencias Físicas y Matemática
Universidad de Chile
Santiago, Chile
tomas.saldivia@ing.uchile.cl

***Resumen*—El uso de modelos de redes neuronales profundas (DNN) en el mundo real está en un rápido aumento. Acordemente, la preocupación por que estos modelos sean seguros y robustos frente a ataques de adversarios también está en auge, siendo un campo de investigación muy activo. Esto no es de extrañar ya que se ha demostrado que engañar a una DNN es relativamente sencillo: basta resolver un problema de optimización para generar una perturbación en la entrada, siendo esta suficiente para inducir una clasificación incorrecta en el modelo. En este proyecto se propone implementar distintos tipos de ataque, para luego compararlos entre sí en un modelo entrenado con ImageNet. Se proponen además una técnica de defensa a aplicar sobre el modelo, la cual será evaluada para los ataques implementados.**

I. Introducción

El *deep learning* ha hecho un rápido y significativo progreso en un amplio espectro de áreas dentro del aprendizaje de máquinas, tales como la clasificación de imágenes, el reconocimiento y la detección de objetos, el reconocimiento de voz, síntesis de voz, entre otros [1]. Esto inevitablemente ha devenido en su aplicación en el mundo físico, especialmente en ambientes y sistemas donde la seguridad es crítica, tales como los vehículos sin conductor, la biometría en cajeros automáticos, etcétera [1]. Esto, como es de esperar, condujo a la comunidad científica a incurrir en qué tan confiables son estos modelos.

Szegedy *et al.* [2] fue el primero en demostrar que puede "engañarse" a un modelo de *deep learning* mediante una pequeña perturbación en la entrada, imperceptible para los seres humanos, generada específicamente con este objetivo. Estas muestras mal clasificadas se denominan **ejemplos adversarios**. Se ha encontrado que se pueden generar ejemplos adversarios físicos, por ejemplo, para confundir a un automóvil sin conductor mediante la manipulación de una señal de pare, engañando a su sistema de reconocimiento de señales de tránsito [3], [4] o mediante la eliminación de la segmentación de peatones en el sistema de reconocimiento de objetos [5]. El generar ejemplos adversarios con el objetivo de engañar a un modelo se conoce como **ataque adversario**. Un ataque adversario puede ser de tipo caja blanca si es que el adversario conoce los pesos y gradientes del modelo, o de tipo caja negra si es que solo conoce su entrada y salida. Claramente, se hace necesaria la implementación de contramedidas que hagan frente a ataques adversarios, campo de investigación activa que se conoce como **defensa adversaria**. En este trabajo se busca estudiar distintos métodos de ataque adversario, para luego evaluarlos contra algún método de defensa adversaria. En particular, se proponen los siguientes objetivos:

- Comparar diversos ataques adversarios contra algún modelo de clasificación entrenado con ImageNet, implementando al menos uno.
- Implementar una estrategia de defensa adversaria en dicho modelo y analizar su efectividad contra los distintos ataques.

Por último, el código utilizado en este proyecto será alojado en GitHub [1].

II. Propuesta

II-A. Datos

Los datos a utilizar se extraen de *ImageNet*, una base de datos de imágenes organizada bajo la estructura jerárquica otorgada por *WordNet*, a su vez una base de datos léxica del idioma inglés. Esta última se caracteriza por agrupar palabras en conjuntos de sinónimos los que denomina *synonym set* o *synset*. Luego, *ImageNet* se vale de estos conjuntos para agrupar sus datos, destacando que su objetivo es proveer en promedio 1000 imágenes por *synset*.

A la fecha *ImageNet* cuenta con más de 14 millones de imágenes separadas en más de 20 mil categorías, y desde el año 2010 el equipo detrás de la base de datos realiza una competencia de software donde los participantes presentan sus modelos de clasificación buscando obtener la mayor cantidad de aciertos en esta. El nombre del concurso es *ImageNet Large Scale Visual Recognition Chanllenge (ILSVRC)*, y se caracteriza por limitar la cantidad de categorías que tiene *ImageNet* a 1000. Es importante mencionar que desde fines del año 2017 en adelante, la competencia paso a estar a cargo de la plataformal digital Kaggle, en formato online.

Dado lo anterior, se tiene que los modelos que se entrenan en base a *ImageNet* clasifican imágenes entre 1000 clases. El *dataset* considerado para la competencia se compone de más de un millón de imágenes en el conjunto de entrenamiento, además de tener 50 mil y 100 mil imágenes en los conjuntos de validación y test respectivamente, tomando en conjunto un espacio en memoria de más de 100 GB. Es por esto que se

[1] Disponible en https://github.com/Tom0497/Adversarial_Defense_In_NN

considerarán opciones alternativas para poder solventar la gran cantidad de memoria requerida para albergar los datos.

En general, los datos de entrada a los modelos que utilizan *ImageNet* tienen un tamaño de 224×224, además de los tres canales de color RGB; pero *ImageNet* provee versiones del *dataset* original con imágenes reducidas en tamaño, implicando un menor uso de memoria para el almacenamiento. Las versiones disponibles son de imágenes de 8×8, 16×16, 32×32 y 64×64, en formato RGB. Además de esto, se cuenta con las URLs de las imágenes del *dataset* original correspondiente al ILSVRC 2012, lo que permite descargar algunas de estas sin la necesidad de almacenar el total de los datos.

La cantidad de imágenes y la separación en los distintos conjuntos del *dataset* original se extraen de [6] y se detallan a continuación:

- **Conjunto de entrenamiento:** 1.281.167 imágenes
- **Conjunto de validación** 50.000 imágenes
- **Conjunto de test:** 100.000 imágenes

Cabe mencionar que la lista de URLs para descargar las imágenes es proveída por la página oficial de ImageNet, y, como esta corresponde a la competencia del año 2012, se debe tener en cuenta que varios de estos enlaces están caídos debido a diversas causas. Luego, no se asegura la accesibilidad al total de la cantidad de imágenes antes mencionadas.

Cada uno de los objetivos del proyecto se persiguió con un enfoque distinto, debido a que lo que se quiere observar en cada una de esas etapas apunta a distintos fines. Luego, tanto los modelos como los datos específicos utilizados para cada parte difieren, por tanto, se hace imperativo realizar una descripción por separado de los datos utilizados en cada sección. Esto se hace a continuación, teniendo en cuenta que siempre corresponden a subconjuntos de *ImageNet*, variando en sólo en la cantidad de clases y en la cantidad de imágenes por clase.

II-A1. Datos para ataques adversarios: Para probar los ataques adversarios y sus efectos en un modelo, no se requiere de todas las clases y tampoco de muchos ejemplos por clase, más aún cuando se considera un modelo pre-entrenado para generar los ejemplos adversarios, el cual es el caso. Luego, considerando que en la mayoría de los casos la generación adversaria implica el cómputo de gradientes, se reducirán la cantidad de clases a considerar, desde las 1000 originales a 50, siendo estas las primeras 50 clases de *ImageNet*, para que el proceso tome menos tiempo. No se especifican las clases debido a que no es relevante saber el nombre de la clase sino más bien los efectos de la generación adversaria sobre la capacidad de clasificación del modelo.

Además de reducir la cantidad de clases, se reduce también la cantidad de imágenes por cada una de estas clases, fijando en 50 ejemplos por cada clase considerada. Luego, se puede observar que el total de imágenes a considerar para esta parte es de 2500 imágenes. Notar que no se especifica ninguna otra división en los datos, esto debido a que no se entrena ningún modelo, sólo se generan ejemplos adversarios.

II-A2. Datos para defensa adversaria: Considerando que para la defensa adversaria se está obligado en cierto punto a entrenar un modelo, se tiene que reducir considerablemente la imágenes a utilizar, esto tanto para la generación de ejemplos adversarios, pero principalmente para el entrenamiento del modelo. Luego, para tener un entrenamiento viable para un computador doméstico se tienen dos opciones: reducir la cantidad de imágenes, es decir tanto el número de clases como la cantidad de ejemplos por cada una de éstas, o reducir el tamaño de las imágenes, considerando quizás algunas de las opciones mencionadas anteriormente respecto a versiones del *dataset* en versiones de menor tamaño.

Debido a que el proyecto tiene un alto componente visual, en el sentido en que se desean poder observar las distorsiones que sufren las imágenes, reducir el tamaño de las imágenes resulta poco práctico. Luego, la única opción es reducir la cantidad de imágenes. Para esto se determinó utilizar 3 clases de las 1000 originales, siendo estas: Toucan (clase 96), Digital Clock (clase 530) y Orange (clase 950).

Considerando que todas las clases se crearon balanceadas, y con 500 imágenes cada una, la caracterización de los datos esta dado por:

- **Conjunto de entrenamiento:** 900 imágenes
- **Conjunto de validación** 300 imágenes
- **Conjunto de test:** 300 imágenes

II-A3. Procesamiento: Cualquier imagen que se utilice como input para un modelo debe sufrir algún proceso de normalización, esto para limitar los valores de los píxeles en ciertos rangos, evitar sesgos hacia los positivos, etc. En específico, para modelos que utilizan *ImageNet* como base, se tiene que previo a utilizar las imágenes, éstas se estandarizan, es decir, se les resta la media y luego se divide por la desviación estándar, ambos valores calculados sobre los datos de entrenamiento. Esto se realiza píxel a píxel para todas las imágenes, algo que se debe tener en cuenta al momento de visualizar resultados sobre imágenes, debido a que se debe revertir el procesamiento.

Se debe considerar además que los modelos sobre *ImageNet* en general consideran un input de tamaño $224 \times 224 \times 3$, donde el 3 representa a los canales RGB. Por lo tanto, al leer o descargar cualquier imagen, se debe asegurar de que su tamaño sea el mencionado, resultando que en algunos casos se requiere de ajuste de dimensiones para llegar al tamaño deseado.

II-B. Modelo

Al igual que en la parte anterior, la especificación de los modelos utilizados se realiza por secciones, dado que no fue el mismo modelo el utilizado en la primera parte del proyecto que en la segunda.

Previo a precisar los modelos utilizados es relevante mencionar el software a utilizar y, en específico, las prestaciones útiles para el proyecto. Se tiene que el lenguaje de programación utilizado es Python, y en específico se utiliza la plataforma **Tensorflow**, para *machine learning*. En términos de las versiones utilizadas, éstas son:

- Python 3.7
- Tensorflow 1.14

II-B1. Modelo para ataques adversarios: El objetivo principal en esta parte es generar ejemplos adversarios, y poder medir como cambia la capacidad de clasificación de un modelo frente a un ataque adversario. Luego, se tiene que el entrenamiento del modelo no es relevante para lo buscado, por lo que se opta por una solución más práctica, la cual corresponde a utilizar un modelo cuyos parámetros ya han sido entrenados sobre el dataset de *ImageNet*. Con esto se asegura que el modelo tiene cierta capacidad de clasificación sobre las imágenes con las que se trabajará, es decir, existe un *accuracy* base.

La principal ganancia al realizar lo anterior es el ahorro de poder computacional y la posibilidad de utilizar un clasificador de alto nivel. En específico, se utiliza la API de Tensorflow, Keras. Esta se vuelve particularmente útil debido a que cuenta con modelos predefinidos y que en su momento fueron estado del arte en clasificación de imágenes.

Dado esto, se eligió el modelo *ResNet50*. Este modelo fue introducido en el año 2015, contando con la característica distintiva de utilizar conexiones residuales entre capas; estos y otros detalles del modelo *ResNet* se pueden consultar en [13]. El grafo del modelo se encuentra en el repositorio del GitHub del proyecto, tanto en su versión obtenida de Tensorboard [2], como conceptual [3].

II-B2. Modelo para defensa adversaria: Se tiene que la defensa de un modelo frente a ataques adversarios requiere de un post entrenamiento o *fine tunning* de los parámetros. Luego, la idea de buscar ahorrar recursos computacionales al utilizar un modelo pre entrenado no se puede aplicar directamente en este apartado, esto debido a que cuando se requiera entrenar al modelo, la cantidad de pesos considerados en una arquitectura como la de *ResNet50* es cercana a los 100 millones, lo que sobrepasa las capacidades de un computador común.

Para solventar lo anterior, se utiliza un funcionalidad de Keras que permite utilizar la parte conformada por bloques convolucionales de una red, es decir, sólo el *feature extractor*, y permitir al usuario acoplar un clasificador al final de los bloques. La ventaja de lo anterior es que se pueden utilizar los pesos pre-entrenados del *feature extractor*y entrenar solamente el clasificador acoplado al final. Como los pesos pre-entrenados se congelan, sus gradientes no son necesarios de calcular y el entrenamiento del modelo es mucho más rápido.

Dado lo anterior, se elige el modelo *VGG16* para servir de *feature extractor*. Este fue presentado en 2015 por Simonyan y Zisserman, y al momento de su publicación era el estado del arte en la competencia *ILSVRC* [14]. Se eligió este modelo por sobre *ResNet50* debido a que al haber disminuido la cantidad de clases de 1000 a 3 se cree no es necesario una arquitectura tan compleja como la de Resnet para lograr buenos resultados en clasificación.

La arquitectura del modelo utilizado esta dada por:

[2]Disponible en https://github.com/Tom0497/Adversarial_Defense_In_NN/blob/master/tensorboard_graph.png

[3]Disponible en https://github.com/Tom0497/Adversarial_Defense_In_NN/blob/master/model.png

II-C. Generación adversaria

Se compararán cuatro algoritmos para la generación de ejemplos adversarios: *Fast Gradient Sign Method* [7], *Fast Gradient Method* [8], *Random Fast Sign Gradient Method* [9], *One-Step Least Likely Class* [8] y *DeepFool* [10]. Los cuatro primeros se denominan ataques de un paso, es decir, realizan solo un cálculo de gradiente, mientras que el último es un ataque iterativo, pues se calculan iterativamente varios gradientes [8] para obtener el ejemplo adversario. Antes de mostrar la teoría subyacente a estos algoritmos, se debe establecer notación. Dado un modelo de *deep learning* ya entrenado f, con pesos $\boldsymbol{\theta}$, se notará por $\boldsymbol{x}$ a alguna entrada, y por y a la etiqueta de esta, con $f(\boldsymbol{x}) = y$. Dado un ejemplo adversario $\boldsymbol{x'}$ tal que $f(\boldsymbol{x'}) = y'$ con $y \neq y'$, se define $\boldsymbol{\eta} = \boldsymbol{x'} - \boldsymbol{x}$ como la perturbación añadida a $\boldsymbol{x}$. Con esto se tiene que, en general, el generar un ejemplo adversario $\boldsymbol{x'}$ puede ser descrito como el problema de optimización con restricciones [11]:

$$\begin{aligned} \min_{\boldsymbol{x'}} \quad & \|\boldsymbol{x'} - \boldsymbol{x}\| \\ \text{s.t.} \quad & f(\boldsymbol{x'}) = y' \\ & f(\boldsymbol{x}) = y \\ & y \neq y' \end{aligned} \tag{1}$$

Este problema de optimización minimiza la perturbación manteniendo la clasificación errónea de la predicción para la entrada. Todos los algoritmos de ataque adversario buscan encontrar o al menos aproximarse a la solución de (1).

- ***Fast Gradient Sign Method (FGSM):*** Este ataque consiste en realizar un paso de actualización del gradiente en la dirección del signo del gradiente en cada píxel. La perturbación puede entonces ser expresada como:

$$\boldsymbol{\eta} = \epsilon \cdot \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)) \tag{2}$$

  donde ϵ corresponde a un hiperparámetro ajustable que define la magnitud de la perturbación.
- ***Fast Gradient Method (FGM):*** También conocido como *Fast Gradient* L_2 [8], es un ataque es similar a $FGSM$, pero en vez de realizar la perturbación de (2), se usa el valor del gradiente normalizado:

$$\boldsymbol{\eta} = \epsilon \cdot \frac{\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)}{\|\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\|_2} \tag{3}$$

  Esto significa que se realiza un paso de actualización del gradiente en la dirección de este en cada píxel.
- ***Random Fast Gradient Sign Method (R-FGSM):*** Otro método similar a *FGSM*, pero en el que se agrega una componente aleatoria, lo cual tiene como consecuencia que es un método de ataque *black-box* más efectivo que *FGSM* [9]. Para un ejemplo $\boldsymbol{x}$, este ataque genera un ejemplo adversario $\boldsymbol{x'}$ según:

$$\boldsymbol{x'} = \hat{\boldsymbol{x}} + (\epsilon - \alpha) \cdot \text{sign}(\nabla_{\hat{\boldsymbol{x}}} J(\boldsymbol{\theta}, \hat{\boldsymbol{x}}, y)) \tag{4}$$

  donde $\hat{\boldsymbol{x}}$ está dado por

$$\hat{\boldsymbol{x}} = \boldsymbol{x} + \alpha \cdot \text{sign}(\boldsymbol{\xi}) \tag{5}$$

siendo $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ una variable aleatoria que distribuye según una gaussiana multidimensional acorde a las dimensiones de la imagen y α, ϵ hiperparámetros, con $\alpha < \epsilon$.

- ***One-Step Least Likely Class (step l.l.):*** A diferencia de los métodos anteriores, este método es un *targeted attack*, es decir, el ejemplo adversario maximiza la probabilidad de que el modelo prediga una clase en específica. En este caso, se maximizará la probabilidad de que el modelo prediga la clase de menor probabilidad. La perturbación para obtener el ejemplo adversario será:

$$\boldsymbol{\eta} = -\epsilon \cdot \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y_{LL})) \tag{6}$$

  donde $y_{LL} = \arg\min_y p(y|\boldsymbol{x}; \boldsymbol{\theta})$ es la clase de menor probabilidad. Notar que a diferencia de (2) y (3), el signo de la perturbación en (6) es negativo.
- ***DeepFool (DF):*** A diferencia de los ataques anteriores, este ataque es iterativo. Para generar un ejemplo adversario, busca la mínima distancia desde el input a la frontera de decisión más cercana y para sobrellevar la no-linealidad en altas dimensiones realiza una optimización iterativa en conjunto con una aproximación lineal. Si se considera un clasificador binario f, el cual hace sus predicciones según el signo de su output, se tiene que la perturbación de este ataque está dada por $\boldsymbol{\eta} = \boldsymbol{r}^{(0)} + ... + \boldsymbol{r}^{(n-1)}$, donde el i-ésimo sumando $\boldsymbol{r}^{(i)}$, para $i = 0, ..., n-1$, se obtiene aplicando el teorema de Taylor y resolviendo:

$$\min_{\boldsymbol{r}} \|\boldsymbol{r}\|_2 \quad \text{s.t. } f(\boldsymbol{x} + \boldsymbol{\eta}^{(i)}) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x} + \boldsymbol{\eta}^{(i)})^T \boldsymbol{r} = 0 \tag{7}$$

  en donde $\boldsymbol{\eta}^{(i)} \coloneqq \sum_{j=0}^{i-1} \boldsymbol{r}^{(j)}$. Es fácil ver que (7) tiene la solución analítica:

$$\boldsymbol{r}^{(i)} = -\frac{f(\boldsymbol{x} + \boldsymbol{\eta}^{(i)})}{\left\|\nabla_{\boldsymbol{x}} f(\boldsymbol{x} + \boldsymbol{\eta}^{(i)})\right\|_2^2} \nabla_{\boldsymbol{x}} f(\boldsymbol{x} + \boldsymbol{\eta}^{(i)}) \tag{8}$$

  Si en alguna iteración se obtiene que $\text{sign}(f(\boldsymbol{x}+\boldsymbol{\eta}^{(i)})) \neq \text{sign}(f(\boldsymbol{x}))$, entonces se detiene el algoritmo y se retorna la perturbación encontrada. Esta estrategia puede extenderse a clasificadores de múltiples clases, lo que requiere encontrar la frontera de decisión más cercana y utilizar la clase adyacente a esta frontera como clase objetivo de la minimización [1], [11].

Una vez definidos los ataques propuestos, se procederá a justificar su elección. *FGSM* se elige ya que es el tipo de ataque más popular, es simple y fue el primero en poder ser utilizado de manera práctica; esto lo ha transformado en una especie de ataque "benchmark", con el cual los nuevos ataques son comparados para corroborar su validez. *FGM* se escoge ya que es un método simple de implementar una vez ya implementado *FGSM* y sería interesante comparar ambos ataques. De la misma forma, *R-FGSM* también es fácil de implementar una vez ya implementado *FGSM*, además de que se encontró en [9] que *R-FGSM* es robusto tanto como ataque *black-box* como ataque *white-box*, mientras que *FGSM* solo es robusto como este último, por lo que *R-FGSM* será especialmente útil para realizar los experimentos con los modelos de caja negra. En cuanto al único ataque *targeted*, *step l.l.*, se escogió este ya que [8] encontró que era el ataque que mayor robustez concedía a los modelos Inception v3 al utilizar sus ejemplos adversarios como entradas del entrenamiento (*adversarial training*). *DF* se elige debido a ser uno de los métodos iterativos más simples, además de que se ha encontrado que hacer *fine-tuning* en modelos entrenados con ImageNet con ejemplos adversarios generados con *DF* mejora la robustez de los modelos frente a ataques *DF* y *FGSM* [1].

Se implementarán *FGSM*, *FGM*, *R-FGSM* y *step l.l.* en Python, utilizando la API de Tensorflow, Keras. Para *DF*, en cambio, se utilizará la librería de Python especializada en ataque adversario, *Foolbox* [12].

II-D. Defensa adversaria

Para mejorar el rendimiento de modelos frente a ataques adversarios, se propone implementar la técnica de inyectar ejemplos adversarios en el conjunto de entrenamiento del modelo, denominada *adversarial training*. Esta técnica fue descrita por primera vez en [7] pero solo probó su efectividad en datasets pequeños (MNIST y CIFAR10). En [8] se mostró que esta técnica también es efectiva en modelos entrenados con ImageNet, pero solo contra ataques de un paso (tales como *FGSM* y *R-FGSM*). En cuanto a *DeepFool*, en [1] se definió un nuevo tipo de *adversarial training*, bautizado como *Deep Defense*, y se encontró que al usar esta técnica con ejemplos adversarios generados con *DF* aumenta la robustez de modelos entrenados con ImageNet frente a este mismo tipo de ataque. Se procederá a definir entonces las técnicas de *adversarial training*, según lo expuesto en [8], y *Deep Defense*, según [1].

- ***Adversarial training:*** En primer lugar, se recomienda utilizar *batch normalization* para realizar esta técnica en ImageNet. Para esto es importante que tanto ejemplos normales como adversarios sean agrupados en el batch antes de realizar un paso de entrenamiento. Para un batch de tamaño m y un número k de ejemplos adversarios en este, la agrupación se realiza de la siguiente forma:
  1. Se lee el batch $B = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ del training set.
  2. Se generan k ejemplos adversarios $\{\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_k\}$ a partir de los ejemplos normales $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_k\}$ usando el estado actual del modelo f.
  3. Se genera un nuevo batch $B' = \{\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_k, \boldsymbol{x}_{k+1}, \ldots, \boldsymbol{x}_m\}$.
  4. Se realiza un paso de entrenamiento en el modelo f usando el batch B'.

  Los pasos anteriores se repiten hasta que el entrenamiento converja, siendo el estado inicial del modelo f aleatorio (o con los pesos del modelo ya entrenado, en caso de querer hacer *fine-tuning*). Además, se usa la siguiente función de pérdida que permita el control independiente

del número relativo de ejemplos adversarios en cada batch:

$$\frac{1}{(m-k)+\lambda k}\left(\sum_{i\in I} L(y_i, f(\boldsymbol{x}_i)) + \lambda \sum_{i\in I'} L(y_i, f(\boldsymbol{x}'_i))\right) \tag{9}$$

donde $I = \{k+1, \ldots, m\}$ y $I' = \{1, \ldots, k\}$ corresponden a los índices de los ejemplos normales y adversarios del batch B', respectivamente; $L(y, f(\boldsymbol{x}))$ corresponde a la pérdida de un solo ejemplo $\boldsymbol{x}$ con etiqueta y y λ es el parámetro que controla el peso relativo de los ejemplos adversarios en la pérdida. Se observó en [8] que si se escoge un valor de ϵ específico para el entrenamiento, el modelo solo se vuelve robusto contra ataques generados con este valor de ϵ, por lo que se recomienda escoger ϵ aleatoria e independientemente en cada ejemplo de entrenamiento. En [8] los mejores resultados se obtuvieron al elegir una distribución normal truncada, definida en el intervalo [0, 16] y con distribución subyacente $\mathcal{N}(0, 8)$.

- ***Deep Defense:*** Esta técnica es descrita en [1], pero solo a nivel de la función de pérdida a utilizar en el entrenamiento. Como no se especifica qué batch utilizar, se propone la siguiente metodología:
  1. Se lee el batch $B = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$ del training set.
  2. Se generan m ejemplos adversarios $\{\boldsymbol{x}'_1, ..., \boldsymbol{x}'_m\}$ a partir de los ejemplos normales $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$ usando el estado actual del modelo f. Es decir, a cada ejemplo normal le corresponde un ejemplo adversario.
  3. Se realiza un paso de entrenamiento en el modelo f usando el batch B y utilizando la función de pérdida característica de *Deep Defense*.

  Los pasos anteriores se repiten hasta que el entrenamiento converja, inicializando los pesos de f de manera aleatoria (o con los pesos del modelo ya entrenado, en caso de querer hacer *fine-tuning*). La función de pérdida característica de este método será:

$$\sum_{i=1}^{m} L(y_i, f(\boldsymbol{x}_i)) + \mu \sum_{i=1}^{m} R\Big(-\frac{\|\boldsymbol{x}'_i - \boldsymbol{x}_i\|_p}{\|\boldsymbol{x}_i\|_p}\Big) \tag{10}$$

  donde se ha usado una regularización basada en el parámetro $\|\boldsymbol{x}' - \boldsymbol{x}\|_p$, i.e. la p-norma ($p \in [1, \infty)$) de la perturbación, normalizado por $\|\boldsymbol{x}\|_p$; y la constante de regularización μ. La función R debe tratar a los ejemplos clasificados correcta e incorrectamente de manera diferente y ser monótonamente creciente en estos últimos (por ejemplo, $R(t) = \exp(t)$). Esta función de pérdida no asegura que no disminuya el *accuracy* de los ejemplos, por lo que se recomienda tratar de forma distinta a los ejemplos cuya versión adversaria engaña y no engaña al modelo. Para esto, el segundo sumando de (10) se reemplaza por

$$\mu \sum_{i\in\mathcal{T}} R\Big(-c\frac{\|\boldsymbol{x}'_i - \boldsymbol{x}_i\|_p}{\|\boldsymbol{x}_i\|_p}\Big) + \mu \sum_{i\in\mathcal{F}} R\Big(d\frac{\|\boldsymbol{x}'_i - \boldsymbol{x}_i\|_p}{\|\boldsymbol{x}_i\|_p}\Big) \tag{11}$$

  donde $c, d > 0$ son dos factores de escala que balancean la importancia de las muestras, R se escoge como la función exponencial y $\mathcal{T}, \mathcal{F}$ se definen como

$$\mathcal{T} = \{i \in J | f(\boldsymbol{x}_i) = y_i\} \tag{12}$$

$$\mathcal{F} = \{i \in J | f(\boldsymbol{x}_i) \neq y_i\} \tag{13}$$

  donde $J = \{1, \ldots, m\}$, es decir, son los índices de los ejemplos normales bien y mal clasificados, respectivamente, por el modelo en su estado actual.

Para la implementación de estas dos técnicas se utilizará la API de Tensorflow, Keras. Para *Deep Defense* específicamente, se utilizará como guía la implementación provista por [1], la cual utiliza el framework PyTorch [4].

III. Resultados esperados

III-A. Ataque adversario

Para comparar los distintos tipos de ataques adversarios, se utilizará el *accuracy* top-1 y top-5. El primero es el *accuracy* común y corriente, i.e. la proporción de imágenes correctamente clasificadas con máxima probabilidad por el modelo. El segundo es la proporción de imágenes cuya etiqueta correcta está entre las 5 etiquetas predichas con máxima probabilidad por el modelo. Ambas medidas de *accuracy* serán calculadas sobre imágenes adversarias generadas por los distintos ataques y comparadas con el *accuracy* para el total de ejemplos limpios, lo cual será presentado en una tabla.

Se espera que *DeepFool* tenga 0 % de *accuracy* top-1, pues el algoritmo está diseñado para encontrar la perturbación necesaria para una clasificación incorrecta. El resto de los ataques, todos de un paso y basados en el gradiente, tienen como hiperparámetro ϵ, por lo cual no se puede asegurar 0 % de *accuracy* top-1 a menos que se busque activamente, por cada imagen, el ϵ necesario para que la perturbación provoque una clasificación errónea. De todos modos, se espera un *accuracy* top-1 mucho menor al del modelo con ejemplos normales. En esta misma línea, un recurso gráfico útil para comparar distintos ataques de un paso es graficar el hiperparámetro ϵ contra el top-1 y top-5 *accuracy*, como se ve en el apéndice de [8].

III-B. Defensa adversaria

Para medir qué tan efectiva es la defensa adversaria, se utilizarán las métricas propuestas en [1]: para los ataques con *FGSM* y sus derivados se utilizará *accuracy* top-1 en el conjunto de validación perturbado y para ataques *DeepFool* se utilizará la denominada *robustness*, definida por primera vez en [10] como:

$$\rho_p \coloneqq \frac{1}{|\mathcal{V}|} \sum_{i\in\mathcal{V}} \frac{\|\boldsymbol{\eta}_i\|_p}{\|\boldsymbol{x}_i\|_p} \tag{14}$$

para algún $p \in [0, \infty)$ que define la norma a utilizar y siendo $\mathcal{V}$ los índices del conjunto de validación.

Para *adversarial training* y según lo expuesto en [8], se espera que el modelo más robusto, al menos contra ataques

[4] Disponible en https://github.com/ZiangYan/deepdefense.pytorch/

Figura 1. Evolución del tiempo de cómputo para 50 imágenes adversarias. Ataque *R-FGSM*.

de un paso, sea el entrenado con ejemplos adversarios generados con *step l.l.*. En cuanto a ataques iterativos como *DF*, *adversarial training* con ejemplos generados en un paso no es efectivo como defensa. En [8] también se intentó hacer *adversarial training* con ejemplos generados iterativamente pero los resultados no fueron beneficiosos, por lo que se espera que entrenar con *DF* no necesariamente mejore la robustez del modelo.

Con *Deep Defense* se espera mejorar el *robustness* frente a ataques *DF* del modelo, teniendo como referencia que en [1] se alcanzó una mejora en este parámetro de un $1{,}5\,\%$ en los modelos AlexNet y ResNet-18, además de que se mejoró ligeramente el *accuracy* en los ejemplos normales del conjunto de validación. Por otro lado, no se tiene una referencia qué tan bien rinde *Deep Defense* contra ataques de un paso en modelos entrenados con ImageNet.

IV. Recursos computacionales

En primer lugar se estimará el tiempo que tomaría obtener ejemplos adversarios para los distintos ataques de un único paso (en total 4), siendo n el número de ejemplos adversarios a generar por cada ataque de este tipo, c el número de valores de ϵ a utilizar y asumiendo que el tiempo de todos los ataques de un paso demoran un tiempo t_{step} constante en generar un ejemplo adversario, entonces el tiempo total de computación será:

$$t_T = 4nct_{\text{step}} \tag{15}$$

En experimentos preliminares se utilizó Google Colab para la generación de ejemplos adversarios, iterándose en un for-loop las distintas imágenes. Se encontró experimentalmente que el t_{step} no es constante a través de las iteraciones, y de hecho crece linealmente respecto a estas. Esto es inesperado y como solución momentánea, solo se entrenaron conjuntos de 50 imágenes en cada for-loop, teniendo que reiniciar el entorno de ejecución de Colab una vez terminado el loop para que el tiempo de ejecución volviera a su estado original (el cual era de alrededor de 5 segundos). En la Figura 1 se muestra este fenómeno para 50 imágenes, utilizando el ataque *R-FGSM*. Con este fenómeno, (15) quedaría mejor enunciada como

$$t_T = 4c\sum_{i=1}^{n}(ai+b) = 4cn\left(a\frac{(n+1)}{2}+b\right) \tag{16}$$

para ciertas constantes a, b. Experimentalmente se encuentra que estas son del orden de $a = 0{,}1$ y $b = 5$. Claramente, con este comportamiento, se pasa de un problema de complejidad $O(n)$ en (15) a uno de $O(n^2)$ en (16). En los experimentos preliminares, se utilizó $n = 50$ y $c = 5$. Con (16) se obtiene un tiempo total de 7550 segundos, aproximadamente 2 horas. Si se considera un t_{step} constante igual a 5 segundos, y se utiliza (15), se obtiene un tiempo total de 5000 segundos, aproximadamente 1 hora y 20 minutos. Es necesario entonces resolver este problema, pues los métodos de *adversarial training* requieren generar muchos más ejemplos y esto no es viable con un tiempo de cómputo creciente.

V. Planificación

Para la realización del proyecto, se ha divido el trabajo a realizar en metas u objetivos, los cuales se muestran a continuación. Para cada uno de ellos se asigna un tiempo estimativo de trabajo efectivo como también, dado esto y la disponibilidad de tiempo de los integrantes, una fecha tentativa de inicio y fin.

1. Extracción y procesamiento de datos:
   - **Horas de trabajo:** 30 horas
   - **Inicio:** 9 de septiembre
   - **Término:** 27 de septiembre
2. *Implementación de ataques adversarios:*
   - **Horas de trabajo:** 20 horas
   - **Inicio:** 25 de septiembre
   - **Término:** 4 de octubre
3. *Realizar experimentos de generación de ejemplos adversarios:*
   - **Horas de trabajo:** 10 horas
   - **Inicio:** 2 de octubre
   - **Término:** 5 de octubre
4. *Implementación de adversarial training:*
   - **Horas de trabajo:** 20 horas
   - **Inicio:** 7 de octubre
   - **Término:** 18 de noviembre
5. *Implementación de Deep Defense:*
   - **Horas de trabajo:** 30 horas
   - **Inicio:** 18 de octubre
   - **Término:** 1 de noviembre
6. *Realizar experimentos de defensa adversaria:*
   - **Horas de trabajo:** 10 horas
   - **Inicio:** 1 de noviembre
   - **Término:** 8 de noviembre
7. *Implementación de ataques black-box:*
   - **Horas de trabajo:** 10 horas
   - **Inicio:** 9 de noviembre
   - **Término:** 15 de noviembre
8. *Realizar experimentos de ataques black-box:*
   - **Horas de trabajo:** 5 horas

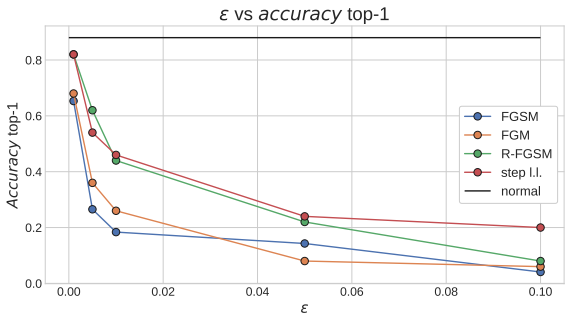

Figura 2. *Accuracy* top-1 contra ϵ, para conjunto de 50 imágenes.

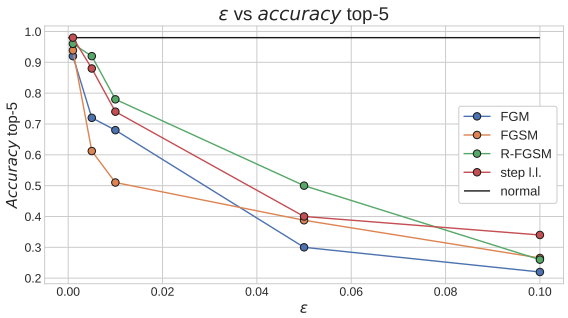

Figura 3. *Accuracy* top-5 contra ϵ, para conjunto de 50 imágenes.

- **Inicio:** 9 de noviembre
- **Término:** 15 de noviembre

9. *Presentación de resultados y redacción de informe final:*
   - **Horas de trabajo:** 30 horas
   - **Inicio:** 1 de noviembre
   - **Término:** 22 de noviembre

Cabe destacar que algunas de estas metas ya se han completado, como lo son la extracción y procesamiento de los datos, la implementación de los ataques adversarios, experimentación con generación de ejemplos adversarios.

Los pasos a seguir se centran principalmente en la defensa adversaria, la cual requerirá de la realización de *fine-tuning* sobre algún modelo pre-entrenado de manera tal que este se vuelva resistente o robusto a los ataques adversarios considerados.

VI. Resultados preliminares

En primer lugar, se debe tener en cuenta que los ataques adversarios buscan engañar a modelos de clasificación, luego lo primero que se debe tener antes de cualquier experimento, es un modelo de clasificación entrenado. Dado que entrenar un clasificador desde cero puede eventualmente tomar mucho tiempo si no se cuenta con el poder de procesamiento adecuado, se decide por utilizar un modelo pre-entrenado. Para esto, se hace uso de los modelo pre-entrenados que ofrece la API de Tensorflow, Keras, la cual cuenta con más de 10 modelos y sus pesos pre-entrenados.

Dado esto, se eligió el modelo *ResNet50*, aunque no se descarta el uso de algún otro disponible en futuras pruebas. Este modelo fue introducido en el año 2015 y contaba con la característica distintiva de utilizar conexiones residuales entre capas; estos y otros detalles del modelo *ResNet* se pueden consultar en [13]. El grafo del modelo se encuentra en el repositorio del GitHub del proyecto, tanto en su versión obtenida de Tensorboard [5], como conceptual [6].

VI-A. Descripción de experimento

En este experimento se considera el modelo *ResNet50* pre-entrenado sobre *ImageNet*. Luego, sobre él se ejecutarán ataques adversarios con un conjunto de imágenes, con el fin de observar como varía el *accuracy* top-1 y *accuracy* top-5 al pasar imágenes que han sido alteradas con el fin de engañar a la red.

Los ataques considerados son los descritos en la sección de generación adversaria, luego, se observa que para todos los métodos a excepción de *DeepFool*, se cuenta con un hiperparámetro ϵ. Luego, se visualizará como cambia el *accuracy* del modelo al variar este hiperparámetro. Se nota además que el método *R-FGSM* requiere de otro hiperparámetro α, el cual, para las experimentos de esta sección que lo requieran, será $\alpha = \frac{\epsilon}{2}$. Además, no se considera el método *DeepFool* en este experimento dado que por definición este algoritmo itera hasta engañar al modelo, además de no contar con un hiperparámetro análogo a ϵ como los otros modelos.

En términos de especificaciones, se consideró el costo computacional que implica obtener ejemplos adversarios a partir de un modelo y un método, luego, para poder obtener resultados en un tiempo razonable y ajustándose a la disponibilidad de hardware, se decidió que las métricas de *accuracy* serían computadas para un conjunto de 50 imágenes, las cuales son tomadas de 50 clases distintas.

Teniendo lo anterior en consideración, se determinó de manera experimental el intervalo de variación de ϵ donde se observaban cambios relevantes, los valores a utilizar son:

$$\epsilon \in \{0,001; 0,005; 0,01; 0,05; 0,1\}$$

Luego, para cada una de las 50 imágenes, se calcula su respectivo ejemplo adversario con los métodos descritos y se guarda la clase predicha, de esta manera, al realizar esto para todas las imágenes, para todos los valores de ϵ y para todos los ataques mencionados, se obtiene el cambio en el *accuracy top-1* y *accuracy top-5*.

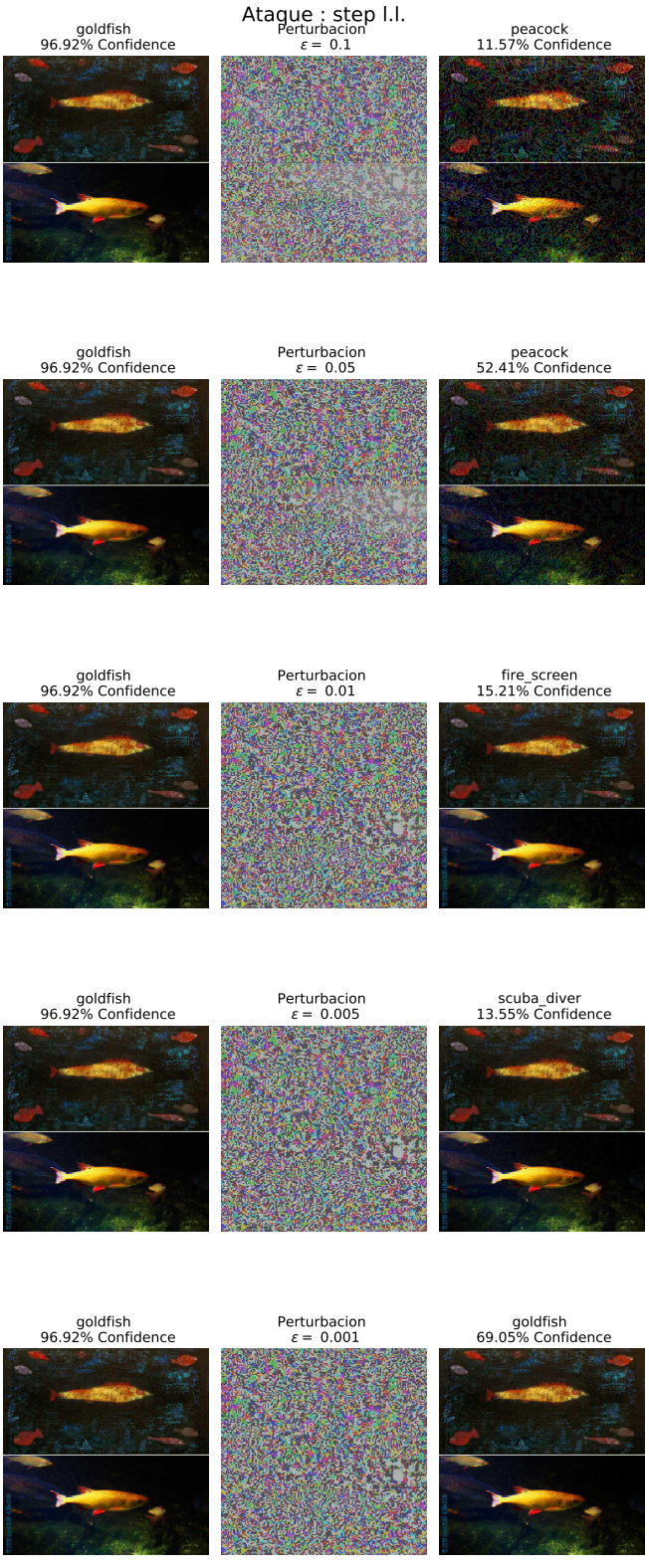

Figura 4. Visualización de ataques adversarios utilizando el método *step l.l.* con distinto factor de perturbación ϵ

Cuadro I
ACCURACY TOP-1 SOBRE CONJUNTO DE 50 IMÁGENES AL VARIAR EL HIPERPARÁMETRO ϵ PARA DISTINTOS ATAQUES.

	ϵ				
Ataque	0,001	0,005	0,01	0,05	0,1
FGSM	0,65	0,26	0,18	0,14	0,04
FGM	0,68	0,36	0,26	0,08	0,06
R-FGSM	0,82	0,62	0,44	0,22	0,08
step l.l.	0,82	0,54	0,46	0,24	0,20

Cuadro II
ACCURACY TOP-5 SOBRE CONJUNTO DE 50 IMÁGENES AL VARIAR EL HIPERPARÁMETRO ϵ PARA DISTINTOS ATAQUES.

	ϵ				
Ataque	0,001	0,005	0,01	0,05	0,1
FGSM	0,93	0,61	0,51	0,38	0,26
FGM	0,92	0,72	0,68	0,30	0,22
R-FGSM	0,96	0,92	0,78	0,50	0,26
step l.l.	0,98	0,88	0,74	0,40	0,34

VI-B. Análisis

De las Tablas I, II se observa en primer lugar que los ataques afectan más al *accuracy* top-1, lo que es de esperarse dado que al utilizar ataques basados en gradiente se busca que el modelo prediga de manera incorrecta un ejemplo, lo que implica no ser la predicción con mayor confianza, pero nada asegura que un ejemplo pueda encontrarse dentro de los siguientes cuatro valores más probables. Por tanto, un primer resultado es que los métodos estudiados deben aplicar un perturbación mayor para lograr afectar métricas más generales como el *accuracy* top-5.

Por otro lado, al analizar las Figuras 2, 3, se nota una segunda característica interesante de los ataques, cuyos efectos se ven reflejados en ambas métricas de *accuracy*. Se observa que en valores de ϵ más pequeños, pequeñas variaciones de éste se ven reflejados en mayor medida en el *accuracy*, en cambio a medida que aumenta ϵ las variaciones no muestran en mismo decaimiento, es decir, medida que aumenta ϵ el *accuracy* estanca su descenso. Esto se tiene en primer lugar porque las predicciones se acercan a un *accuracy* de cero. En segundo lugar, esto permite ver que el modelo es poco resistente a los ataques adversarios dado que para pequeños aumentos de ϵ el *accuracy* decae rápidamente.

A modo de ejemplo se muestra en la Figura 4 los resultados de variar el ϵ. En la columna izquierda, se tiene la imagen original, mostrando además la clase que el modelo predice y con cuanta confianza lo hace, luego, en la columna derecha se tiene el resultado de aplicar el método *step l.l.* para crear el ejemplo adversario, observando además la clase predicha en este caso y su confianza asociada, por último, en la columna

[5]Disponible en https://github.com/Tom0497/Adversarial_Defense_In_NN/blob/master/tensorboard_graph.png

[6]Disponible en https://github.com/Tom0497/Adversarial_Defense_In_NN/blob/master/model.png

central se tiene la perturbación aplicada a la imagen original para obtener el ejemplo adversario.

En todos los casos se observa que el resultado de aplicar la generación adversaria es hasta cierto punto imperceptible e, incluso cuando se puede notar cierta distorsión en la imagen, la clase sigue siendo clara. Pero para el modelo de clasificación esto no es así, dado que se observa que a excepción del caso con $\epsilon = 0{,}001$ todos los otros fueron mal clasificados.

REFERENCIAS

[1] Z. Yan, Y. Guo y C. Zhang, "Deep Defense: Training DNNs with Improved Adversarial Robustness", en *arXiv e-prints, arXiv:1803.00404*, febrero 2018.

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, Dumitru, I. Goodfellow y R. Fergus, "Intriguing properties of neural networks" en *arXiv e-prints, arXiv:1312.6199*, diciembre 2013.

[3] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati y D. Song, "Robust physical-world attacks on deep learning models", *arXiv e-print, arXiv:1707.08945*, julio 2017.

[4] A. Kurakin, I. Goodfellow y S. Bengio, "Adversarial examples in the physical world", *arXiv e-prints, arXiv:1607.02533*, julio 2016.

[5] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie y A. Yuille, "Adversarial examples for semantic segmentation and object detection", en *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1378-1387.

[6] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", en *arXiv e-prints, arXiv:1409.0575*, septiembre 2014.

[7] I. Goodfellow, J. Shlens y C. Szegedy, "Explaining and Harnessing Adversarial Examples" en *arXiv e-prints, arXiv:1412.6572*, diciembre 2014.

[8] A. Kurakin, I. Goodfellow y S. Bengio, "Adversarial Machine Learning at Scale", en *arVix e-prints, arXiv:1611.01236*, noviembre 2016.

[9] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh y P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses", en *arXiv e-prints, arXiv:1705.07204*, mayo 2017.

[10] S.-M. Moosavi-Dezfooli, A. Fawzi y P. Frossard, "DeepFool: a simple and accurate method to fool deep neural networks", en *arXiv e-prints, arXiv:1511.04599*, noviembre 2015.

[11] X. Yuan, P. He, Q. Zhu y X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning", en *arXiv e-prints, arXiv:1712.07107*, diciembre 2017.

[12] J. Rauber, W. Brendel y M. Bethge, "Foolbox: A Python toolbox to benchmark the robustness of machine learning models", en *arXiv e-prints, arXiv:1707.04131*, julio 2017.

[13] K. He, X. Zhang, S. Ren y J. Sun, "Deep Residual Learning for Image Recognition" en *arXiv e-prints, arXiv:1512.03385*, diciembre 2015.

[14] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Regcognition", en *arXiv e-print, arXiv:1409.1556*, abril 2015.