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 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 propone además una técnica de defensa a aplicar sobre un 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 activo 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. Antecedentes teóricos

Antes de definir los alcances prácticos del proyecto, se deben explicar algunos conceptos teóricos a utilizar durante todo el proyecto.

II-A. Generación adversaria

A continuación se definirán cuatro algoritmos para la generación de ejemplos adversarios: *Fast Gradient Sign Method* [6], *Fast Gradient Method* [7], *Random Fast Sign Gradient Method* [8] y *One-Step Least Likely Class* [7]. Todos estos ataques entran dentro de la categoría denominada ataques de un paso, es decir, realizan solo un cálculo de gradiente. 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 [9]:

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

Este problema de optimización minimiza la perturbación manteniendo la clasificación errónea de la predicción para la

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

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 [7], es un ataque 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 sea un método de ataque *black-box* más efectivo que *FGSM* y un método de defensa más robusto que los otros ataques de un paso contra ataques *white-box* [8]. 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 $\alpha < \epsilon$. En particular, en lo que resta de este trabajo se utilizará $\alpha = \epsilon/2$, tal y como hacen [8].
- ***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.

II-B. 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 [6] pero solo probó su efectividad en datasets pequeños (MNIST y CIFAR10). En [7] se mostró que esta técnica también es efectiva en modelos entrenados con *ImageNet*, pero solo contra ataques de un paso (como todos los que se vienen de definir en la subsección anterior). Se procederá a definir entonces la técnica de *adversarial training*, según lo expuesto en [7].

- ***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:

$$\frac{1}{(m-k) + \lambda k} \left(\sum_{i \in I} L(y_i, f(\boldsymbol{x}_i)) + \lambda \sum_{i \in I'} L(y_i, f(\boldsymbol{x}'_i)) \right) \tag{7}$$

  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 [7] 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 [7] los mejores resultados se obtuvieron al elegir una distribución normal truncada, definida en el intervalo [0, 16] y con distribución subyacente $\mathcal{N}(0, 8)$.

III. Datos

III-A. Descripción general de los 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 Challenge (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 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 [10] 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. Se hace imperativo entonces 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.

III-B. Datos para ataques adversarios

Para probar los ataques adversarios y sus efectos en un modelo, es conveniente utilizar un subconjunto de *ImageNet* que contenga la mayor cantidad de clases posibles, con tal de representar lo mejor posible el *dataset* original. Se querría tener además más de un ejemplo por cada clase, con tal de tener cierto grado de generalidad dentro de estas. La cantidad de clases y la cantidad de imágenes por cada clase a considerar dependerá de qué tanto tiempo se tiene disponible para realizar el cómputo, y la cantidad de RAM disponible en el sistema. Teniendo esto en consideración, se decidió seleccionar 305 clases al azar, con 3 imágenes por cada una, teniéndose en total 915 imágenes. 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 en términos generales. 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.

III-C. 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 resolución.

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

donde cada uno de estos conjuntos está a su vez balanceado

III-D. Procesamiento

Cualquier imagen que se utilice como input para un modelo debe pasar por 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, estas se estandarizan, es decir, se les resta la media y luego se divide por la desviación estándar, ambos valores calculados a partir de 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.

IV. Modelos

Al igual que en la sección 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.

IV-A. Modelo para ataques adversarios

El objetivo principal en esta parte es generar ejemplos adversarios con un tipo de ataque específico, y poder medir cómo cambia la capacidad de clasificación de un modelo frente a estos. 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 tenga 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.

Es así como se elige 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 [11]. El grafo del modelo se encuentra en el repositorio del GitHub del proyecto, tanto en su versión obtenida de Tensorboard [2], como conceptual [3].

IV-B. Modelo para defensa adversaria

Como la defensa de un modelo frente a ataques adversarios requiere de un post-entrenamiento o *fine tuning* de los parámetros, la idea de buscar ahorrar recursos computacionales al utilizar un modelo pre-entrenado no se puede aplicar directamente en este apartado. Esto se debe 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, API de Tensorflow, 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.

Cuadro I: Arquitectura del modelo de clasificación utilizado para la defensa adversaria.

Tipo	K. size	S	Salida	Parámetros
Convolución	3×3	1	$224 \times 224 \times 64$	1792
Convolución	3×3	1	$224 \times 224 \times 64$	36928
Max Pool	2×2	2	$112 \times 112 \times 64$	0
Convolución	3×3	1	$112 \times 112 \times 128$	73856
Convolución	3×3	1	$112 \times 112 \times 128$	147584
Max Pool	2×2	2	$56 \times 56 \times 128$	0
Convolución	3×3	1	$56 \times 56 \times 256$	295168
Convolución	3×3	1	$56 \times 56 \times 256$	590080
Convolución	3×3	1	$56 \times 56 \times 256$	590080
Max Pool	2×2	2	$28 \times 28 \times 256$	0
Convolución	3×3	1	$28 \times 28 \times 512$	1180160
Convolución	3×3	1	$28 \times 28 \times 512$	2359808
Convolución	3×3	1	$28 \times 28 \times 512$	2359808
Max Pool	2×2	2	$14 \times 14 \times 512$	0
Convolución	3×3	1	$14 \times 14 \times 512$	2359808
Convolución	3×3	1	$14 \times 14 \times 512$	2359808
Convolución	3×3	1	$14 \times 14 \times 512$	2359808
Max Pool	2×2	2	$7 \times 7 \times 512$	0
Convolución	1×1	1	$7 \times 7 \times 128$	65664
Convolución	1×1	1	$7 \times 7 \times 64$	8256
Convolución	1×1	1	$7 \times 7 \times 32$	2080
Flatten	–	-	1568	0
Densa	–	-	128	200832
Densa	–	-	3	387
				Total: 14991907

Dado lo anterior, se elige el modelo *VGG16* para servir de *feature extractor*. Este fue presentado en 2015 por Simonyan y Zisserman [12], y al momento de su publicación era el estado del arte en la competencia *ILSVRC*. Se eligió este modelo por sobre *ResNet50* debido a que al haber disminuido la cantidad de clases de 1000 a 3 se cree que no es necesario una arquitectura tan compleja para lograr buenos resultados en clasificación.

La arquitectura del modelo utilizado está dada por el Cuadro I. Para comprender cabalmente esta arquitectura, se señala que la columna identificada por ***K. size*** hace referencia al tamaño del kernel para los bloques convolucionales. Asimismo, ***S*** se refiere al *stride* considerado en cada bloque. Por último, la columna **Parámetros** indica la cantidad de parámetros que aporta al modelo la capa respectiva.

Considerando lo anterior se nota en primer lugar que la cantidad total de parámetros del modelo es de casi 15 millones, lo cual es mucho menor al tamaño original de la red VGG16 que alcanza los 138 millones de parámetros [12]. Pese a ello, esto aún presenta un desafío para un computador convencional.

Con la línea punteada se marca el fin del modelo original *VGG16* y el comienzo del clasificador acoplado. Analizando este último, vale la pena destacar lo esenciales que son los primeros bloques convolucionales de 1×1 para poder reducir

[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

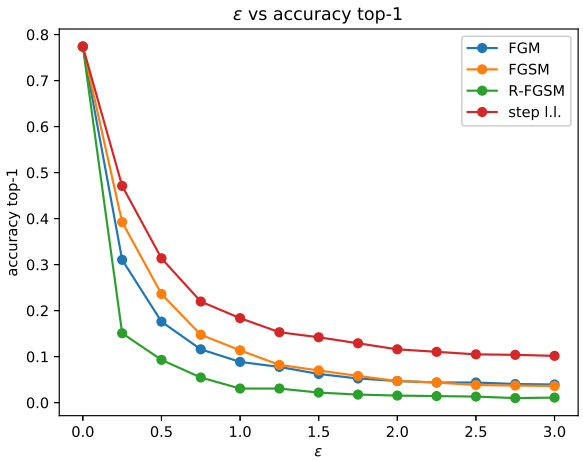


Figura 1: *Accuracy* top-1 contra ϵ, para conjunto de 915 imágenes, en 305 clases distintas.

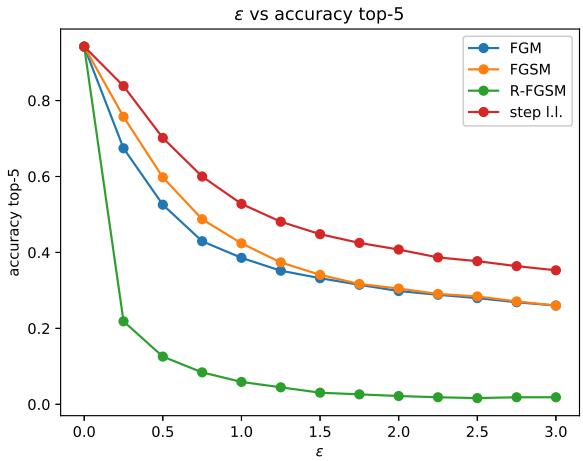


Figura 2: *Accuracy* top-5 contra ϵ, para conjunto de 915 imágenes, en 305 clases distintas.

la cantidad de *feature maps* y, por tanto, reducir la cantidad de parámetros a entrenar. Luego, considerando que solo se entrenan las capas agregadas al final, se tiene que la cantidad de parámetros a entrenar son **277219**, lo que es mucho más manejable para un computador común.

V. Experimentos

Se realizarán dos experimentos, uno buscará medir la efectividad de ataques adversarios contra un modelo *white-box* y el otro buscará medir la efectividad de una técnica de defensa adversaria. Para esto, se utiliza el lenguaje de programación Python y, en específico, la plataforma **Tensorflow**. En términos de las versiones usadas, estas son:

- Python 3.7
- Tensorflow 1.14

V-A. Generación adversaria

Ya definidos algunos ataques adversarios, es prudente justificar por qué se eligieron estos en específico al momento de realizar este experimento. *FGSM* se eligió ya que es el tipo de ataque más popular, es simple de implementar 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* y *R-FGSM* se escogen ya que son métodos simples de implementar una vez ya implementado *FGSM*, y sería interesante compararlos entre sí. En cuanto al único ataque *targeted*, *step l.l.*, se escogió este ya que [7] 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*).

Este primer experimento consistirá en generar ejemplos adversarios a partir del conjunto de imágenes descrito en la sección III-B, para cada ataque adversario y valor de ϵ definido, y evaluar qué tan bien categoriza estos el modelo. El parámetro de evaluación será el *accuracy* top-1 y el *accuracy* top-5. Se utilizarán los valores de ϵ:

$$\epsilon_i = 0{,}25 \times i, \text{ con } i = 1, \ldots, 12 \tag{8}$$

Como se tienen cuatro tipos de ataques, 12 valores de ϵ y 915 imágenes por atacar, se deberán generar 43920 ejemplos adversarios. Por cada par de tipo de ataque y valor de ϵ, se calculará el *accuracy* top-1 y el *accuracy* top-5 sobre los 915 ejemplos adversarios generados, obteniéndose 48 valores por cada métrica, 49 en total considerando la evaluación del modelo con los ejemplos sin modificar ($\epsilon = 0$).

V-B. Defensa adversaria

La parte principal de este experimento es mostrar como mejora la clasificación de un modelo, en términos de ejemplos adversarios, cuando se somete a un entrenamiento adversario. No obstante, se debe abordar el entrenamiento del modelo elegido, debido a que implica la toma de varias decisiones que vale la pena mencionar. Por tanto, se menciona en primer lugar cómo se va a entrenar al modelo y los experimentos llevados a cabo para este fin, para luego pasar a definir el experimento a realizar en términos de la defensa adversaria.

V-B1. Optimizador y tasa de aprendizaje: Al momento de entrenar un modelo se deben tener en cuenta ciertos factores que pueden influir de manera directa en el desempeño del modelo y en el tiempo en que este toma para entrenar, es así, que la elección de hiperparámetros presenta un desafío, esto debido a que no hay elecciones universales que apliquen a cualquier modelo, por lo que estos se deben buscar de cierto modo.

En específico, para el entrenamiento del modelo se consideran como hiperparámetros a elegir tanto el optimizador como la tasa de aprendizaje a utilizar. Luego, la forma en que se elegirán estos es mediante un denominado *grid search*, es decir, se eligen una cantidad de optimizadores a probar y se define un vector de valores de tasas de aprendizaje, para posteriormente

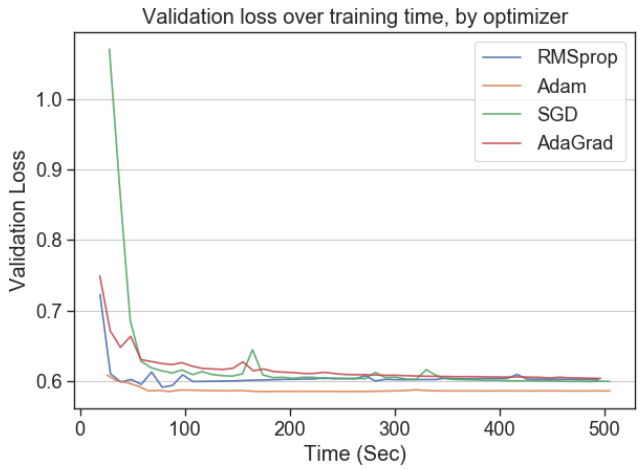


Figura 3: Curvas de loss en conjunto de validación para clasificador propuesto.

entrenar el modelo probando cada combinación posible de los elementos antes mencionados. A continuación se indican las opciones a probar y luego se detalla el procedimiento:

- **Optimizadores**
  - SGD
  - ADAM
  - ADAGrad
  - RMSprop
- **Tasas de aprendizaje**
  - 5×10^{-1}
  - 1×10^{-1}
  - 5×10^{-2}
  - 1×10^{-2}
  - 5×10^{-3}
  - 1×10^{-3}
  - 5×10^{-4}
  - 1×10^{-4}

Previo a la descripción del entrenamiento, se destaca que el método **SGD** se utiliza en una variante que añade momentum y optimización de Nesterov, esto para darle más robustez al algoritmo. Dicho esto, como se utilizarán todas las combinaciones posibles, se tiene que en total se entrena el modelo 32 veces, o, si se quiere, se entrenan 32 modelos.

Todos los modelos se someten a las mismas condiciones, en primer lugar, cada modelo se entrena por 15 épocas, utilizando batches de 64 imágenes. Los datos son los descritos en la sección de datos III-C. A medida que ocurre el entrenamiento se va registrando el rendimiento del modelo tanto en el conjunto de entrenamiento como en el de validación, esto con fines de monitoreo y además para guardar los pesos de la iteración que mostró un mejor desempeño en términos de minimizar la función de pérdida evaluada en el conjunto de **validación**. Esto último se justifica en que, a medida que la pérdida del modelo es menor, menos se equivoca en clasificar y más seguro está de las decisiones que toma.

Luego de este entrenamiento preliminar, se procede entonces a elegir la tasa de aprendizaje que obtuvo la menor pérdida en el conjunto de **validación**, para cada optimizador considerado, pasando así de tener 32 pares posibles de hiperparámetros a tan sólo 4 pares. Luego, se utilizan estos valores para entrenar nuevamente 4 modelos, pero aumentando la cantidad de épocas de 15 a 50. Esto se realiza con el fin de quitar cualquier sesgo que podría haber introducido la poca cantidad de épocas en la elección de los hiperparámetros.

Así, siguiendo la misma lógica anterior, una vez obtenido los 4 modelos, se termina por elegir el que minimiza la pérdida, en términos del conjunto de validación. Luego, siguiendo la heurística recién planteada se logra obtener el modelo final que se utilizará para la defensa adversaria.

V-B2. Estado base: Previo a otorgar robustez al modelo frente ataques adversarios, es importante conocer el estado base del clasificador, es decir, como se comporta frente a estos ataques sin ningún tipo de entrenamiento adversario, para así poder luego comparar los cambios observados.

Para realizar lo anterior se han elegido, en primer lugar, distintos valores para ϵ, el cual es el parámetro que regula qué tanta distorsión se aplica al ejemplo adversario generado. Los valores elegidos son:

$$\epsilon_i = 0{,}25 \times i, \text{ con } i = 0, \ldots, 12$$

Se eligió como valor máximo $\epsilon = 3$ debido a que experimentalmente se observó que aplicar distorsiones con valores por sobre el indicado generaba imágenes con un nivel de ruido fácilmente distinguible a simple vista.

Con lo anterior, el procedimiento fue el siguiente: en primer lugar se eligen de manera aleatoria 100 imágenes del conjunto de **test**, luego para este conjunto de imágenes se calculan ejemplos adversarios utilizando los valores de ϵ especificados, es decir, se tienen 1200 ejemplos adversarios para test, 100 por cada valor de ϵ. Se guarda el *accuracy* sobre las 100 imágenes para cada ϵ para así poder comparar estos valores con lo obtenido luego del entrenamiento adversario.

V-B3. Entrenamiento adversario: Para otorgar robustez al modelo se sigue el siguiente procedimiento. En primer lugar, para realizar el entrenamiento adversario se deben tener los conjuntos de entrenamiento y validación adecuados, para ello, se considera que ambos deben contar con una proporción de ejemplos no adversarios y adversarios, esto dado que si bien se busca que el modelo sea más robusto, también se quiere que la capacidad de clasificación obtenida originalmente no se vea afectada en gran manera.

Para lo anterior, se considera que el conjunto de validación, compuesto por 300 imágenes, tenga a la mitad de sus ejemplos como adversarios y mantenga la mitad restante sin cambio.

Para el caso del conjunto de entrenamiento se utilizará el total de las imágenes (es decir, 900 imágenes) en sus versiones adversarias, entrenando en batches de 64 imágenes. En términos de la función de costo, siguiendo la Ecuación 7, se tendrá entonces $m = k = 64$, y $\lambda = 1$.

El valor de ϵ utilizado para generar cada ejemplo adversario en los conjuntos de validación y entrenamiento, fue muestreado de una distribución Gaussiana truncada, con parámetros $(\mu, \sigma) = (1{,}5; 0{,}75)$ la cual sólo entrega valores restringida al rango $[\mu - 2\sigma, \mu + 2\sigma]$, es decir, $[0, 3]$. Esto se realizó así para no sesgar el modelo a un valor de ϵ específico, y para que

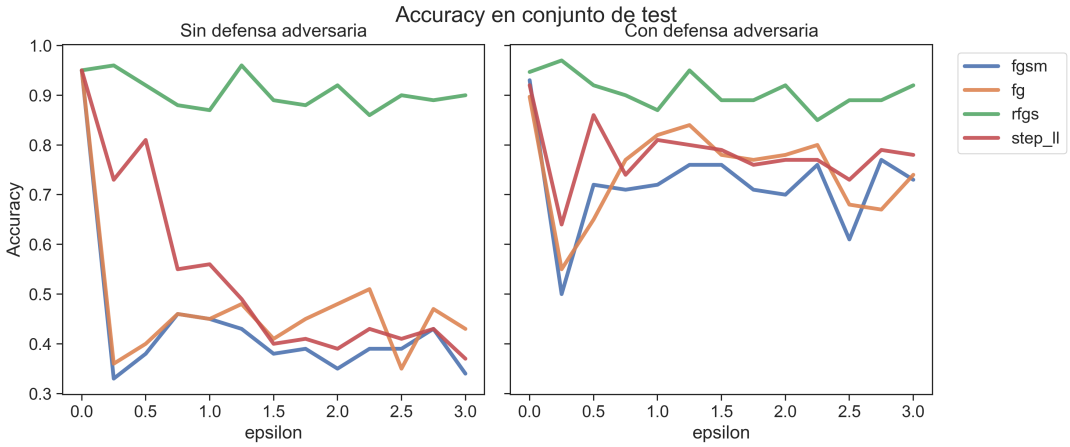


Figura 4: *Accuracy* vs ϵ para distintos ataques adversarios, comparando resultados antes y después de aplicar defensa adversaria.

tanto los conjuntos de entrenamiento, validación y test, entre sí, cuenten con ejemplos generados con distintos valores de ϵ.

Con los conjuntos anteriores, se tiene que para el entrenamiento se consideran 50 épocas. Además, se ha disminuido la tasa de aprendizaje, pasando de $0{,}001$ a $0{,}0005$, esto dado que con el entrenamiento inicial el modelo esta cerca del óptimo, luego, al realizar este *fine tuning* de los parámetros se espera que la solución óptima, considerando los ejemplos adversarios, esté cerca de la ya encontrada, por lo que no se deben realizar pasos de actualización grandes.

Finalmente, se tiene que el procedimiento descrito de defensa adversaria se realiza para los cuatro ataques considerados en este trabajo, para luego comparar sus resultados entre sí. Las mismas 100 imágenes elegidas al azar del conjunto de test son utilizado para evaluar el *accuracy* para cada ϵ en el modelo luego del *fine tuning*.

VI. Resultados

VI-A. Generación adversaria

Para comparar los distintos tipos de ataques adversarios, se utilizarán como métricas el *accuracy* top-1 y top-5. La primera métrica es el *accuracy* común y corriente, es decir, la proporción de imágenes correctamente clasificadas con máxima probabilidad por el modelo. Los resultados de esta métrica se muestran en el Cuadro II. El *accuracy* top-1 basal ($\varepsilon = 0$) es $0{,}7738$. La segunda métrica es la proporción de imágenes cuya etiqueta correcta está entre las 5 etiquetas predichas con máxima probabilidad por el modelo. Los resultados de esta métrica se muestran en el Cuadro III. El *accuracy* top-5 basal es $0{,}9421$. Como se ve en el apéndice de [7], un recurso gráfico útil para comparar los distintos ataques es graficar el hiperparámetro ϵ contra el top-1 y top-5 *accuracy*. Es así como los datos de las tablas ya referenciadas son graficados en la Figura 1 y 2 para el *accuracy* top-1 y top-5 respectivamente.

VI-B. Defensa adversaria

En primer lugar se mostrarán los resultados del *grid search* y el proceso de selección del modelo que se realizó a partir de estos. En primer lugar, se entrenaron los 32 modelos para todas combinaciones de optimizadores y tasas de aprendizaje. En el Cuadro IV se muestran la mejor tasa de aprendizaje para cada optimizador. La mejor tasa de aprendizaje será la que permita al modelo obtener el menor *loss* en validación para 15 épocas. Se entrenaron entonces 4 modelos, cada optimizador con su mejor tasa de aprendizaje, durante 50 épocas, cuyas curvas de *loss* se puede ver en la Figura 3. En un principio se tenía en cuenta el tiempo que tomaba al modelo entrenar dependiendo del optimizador, pero luego de un par de pruebas se logró determinar que la diferencia entre los tiempos obtenidos era despreciable y, por tanto, no era un factor a considerar. Luego, observando las curvas de la Figura 3, se tiene que todos los modelos alcanzan un óptimo relativamente bueno, notando que unos se demoran más que otros en converger. Con lo anterior, se determina que el mejor modelo es el que utiliza como optimizador ADAM y una tasa de aprendizaje de $0{,}001$. Luego, este es el modelo que se entrena y se utiliza para la siguiente parte del experimento de defensa adversaria.

Con el modelo ya entrenado, se procede a la defensa adversaria según el procedimiento descrito en la sección anterior. Los resultados para los cuatro ataques adversarios considerados se muestran en la Figura 4, donde por un lado se muestra el testeo del modelo sin defensa frente a ataques adversarios y, luego con defensa frente a ataques adversarios.

VII. Análisis

VII-A. Generación adversaria

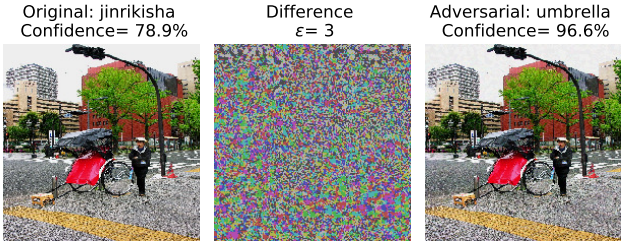
Al analizar los Cuadros II, III, apoyándose de las Figuras 1, 2, queda en evidencia la efectividad de los ataques adversarios en estudio. Se observa que los valores de *accuracy* cambian bruscamente entre $\epsilon = 0$ y $\epsilon = 1$, para luego estabilizarse a medida que aumenta este parámetro. Esto quiere decir que con un pequeño valor de ϵ, tal como $0{,}25$, es posible pasar de un *accuracy* top-1 completamente aceptable para la época en que se diseñó el modelo de clasificación de $77\,\%$ a uno completamente inservible como $47\,\%$ (en el ataque que resultó más débil, *step l.l.*) o incluso uno tan bajo como $15\,\%$ (en el

Cuadro II: *Accuracy* top-1 sobre conjunto de 915 imágenes al variar el hiperparámetro ϵ para distintos ataques.

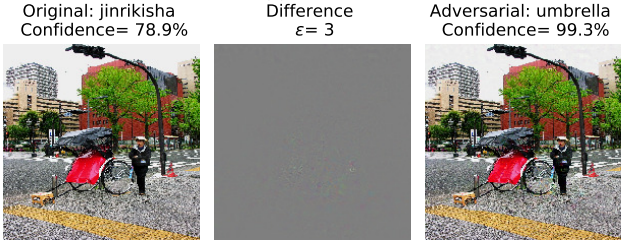
	ϵ											
Ataque	0.25	0.5	0.75	1	1.25	1.5	1.75	2	2.25	2.5	2.75	3
FSGM	0.3923	0.2361	0.1475	0.1137	0.0820	0.0699	0.0579	0.0470	0.0437	0.0383	0.0372	0.0361
FGM	0.3104	0.1760	0.1159	0.0885	0.0776	0.0623	0.0525	0.0470	0.0437	0.0437	0.0404	0.0393
R-FGSM	0.1508	0.0929	0.0546	0.0306	0.0306	0.0219	0.0175	0.0153	0.0142	0.0131	0.0098	0.0109
step l.l.	0.4710	0.3137	0.2197	0.1836	0.1530	0.1421	0.1290	0.1158	0.1104	0.1049	0.1038	0.1016

Cuadro III: *Accuracy* top-5 sobre conjunto de 915 imágenes al variar el hiperparámetro ϵ para distintos ataques.

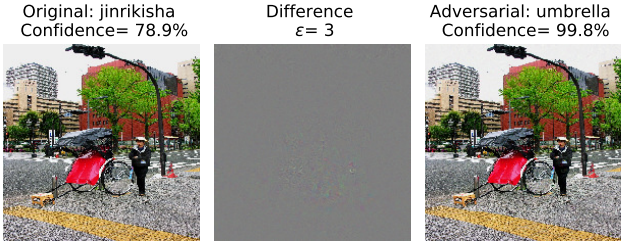
	ϵ											
Ataque	0.25	0.5	0.75	1	1.25	1.5	1.75	2	2.25	2.5	2.75	3
FSGM	0.7574	0.5978	0.4874	0.4240	0.3738	0.3410	0.3169	0.3049	0.2907	0.2841	0.2710	0.2601
FGM	0.6743	0.5257	0.4295	0.3858	0.3519	0.3322	0.3147	0.2984	0.2885	0.2798	0.2689	0.2601
R-FGSM	0.2186	0.1257	0.0842	0.0590	0.0448	0.0306	0.0262	0.0219	0.0186	0.0164	0.0186	0.0186
step l.l.	0.8383	0.7016	0.6000	0.5279	0.4809	0.4481	0.4251	0.4077	0.3869	0.3770	0.3639	0.3530



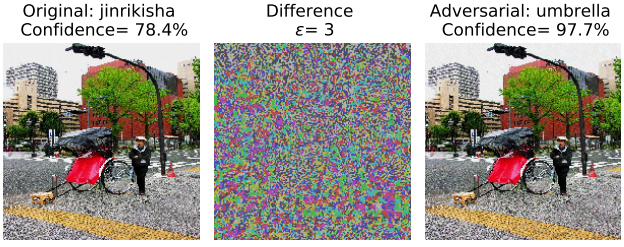
(a) FGSM



(b) FGM



(c) R-FGSM



(d) step l.l.

Figura 5: Visualización de los distintos ataques adversarios para $\epsilon = 3$.

Cuadro IV: Mejor tasa de aprendizaje para cada optimizador, luego de entrenar por 15 épocas.

Ataque	**Tasa de aprendizaje**
RMSprop	0.001
ADAM	0.001
SGD	0.005
ADAGrad	0.0005

ataque que resultó más agresivo, *R-FGSM*), observándose algo equivalente en el *accuracy* top-5, aunque en menor medida. El hecho que *R-FGSM* afecte tanto al *accuracy* top-5 se puede explicar en el hecho que, al dirigir la imagen a una dirección arbitraria, es más probable que en esta dirección se encuentre una clase que no esté entre las 5 clases más probables dadas por el modelo.

Solo para ejemplificar visualmente los ataques adversarios, se presenta la Figura 5. En la columna izquierda, se tiene la imagen original, mostrando además la clase que el modelo predice (un *jinrikisha*, un vehículo ligero de tracción humana usado en Asia) y con cuánta confianza (o probabilidad) lo hace. En la columna derecha se tiene el resultado de aplicar los distintos ataques para crear el ejemplo adversario, observando además la clase predicha en este caso y su confianza asociada. Por último, en la columna central se tiene la perturbación aplicada a la imagen original para obtener el ejemplo adversario. En todos los casos se observa que el ejemplo adversario, a simple vista, es indistinguible del original. Incluso si uno es capaz de percibir cierto nivel de ruido, es claro que la imagen sigue siendo un *jinrikisha*. Para el modelo de clasificación, sin embargo, esto no es así, dado que se observa que para todos los tipos de ataque el ejemplo fue mal clasificado, con un alto nivel de confianza, entre $96{,}6\,\%$ y $99{,}8\,\%$, en la clase *paraguas*.

VII-B. Defensa adversaria

En un entrenamiento adversario, el resultado esperado es que el modelo sea más robusto a ataques luego de ser entrenado, y esto es precisamente lo que se observa en la

Figurea 4. Para los distintos valores de ϵ ocupados se observa que el modelo reduce de manera considerable su capacidad de clasificación, pasando de tener un *accuracy* cercano al 95 % a 40 % en varios de los casos. Luego, esto corresponde al efecto de los ataques adversarios. Por otro lado, luego de la defensa se observa que el modelo logra aumentar el bajo *accuracy* obtenido en los ejemplos adversarios, siendo esto un efecto general para todos los valores de ϵ y para casi todos los ataques.

El único caso especial se tuvo para el ataque *R-FGSM*, donde se observó que los ataques no fueron efectivos contra el modelo, en comparación con los otros. Se observa que independiente del ϵ utilizado el *accuracy* se mantiene relativamente alto, comportamiento que se mantiene luego del entrenamiento adversario. Causa probable de esto y del hecho de que los resultados sean tan dispares respecto al primer experimento para este método, es que la cantidad de clases consideradas es mucho menor (se pasó de 305 a 3). Hay que recordar que este método está caracterizado por avanzar en una dirección aleatoria dada por el muestreo de una distribución normal. Luego, como sólo hay tres clases en el espacio de clasificación, es muy probable que moverse en una dirección aleatoria no sea suficiente para que el modelo se equivoque, como sí lo era, al contrario, para el caso con más clases, dado que las representaciones de una clase u otra están más cerca entre ellas.

Con esto, se tiene que el método *R-FGSM* se adecúa mejor a modelos multi-clase tipo *ImageNet*, más que a modelos simples de 2 o 3 clases.

VIII. Conclusión

Este trabajo presentó el concepto de ataque adversario a modelos de clasificación, realizando una revisión bibliográfica de los ataques más esenciales. Asimismo, se observó cómo estos hacen que un modelo que a priori muestra una buena capacidad de clasificación, se equivoque en gran manera, considerando pequeñas distorsiones en las imágenes, imperceptibles al ojo humano.

En específico se observó cómo un modelo como *ResNet50* pasa de tener un *accuracy* base cercano al 80 % a uno de 10 % o menos en algunos casos. Luego, esto debe ser un factor a considerar en modelos de clasificación, dado que un modelo no robusto a ataques adversarios puede significar que se aprendió una representación local de las imágenes, pero con pequeñas perturbaciones en torno a la vecindad de la imagen el modelo pierde su capacidad, lo que puede significar una baja capacidad de generalización.

Para solventar lo anterior se utilizó un modelo simple de entrenamiento adversario, donde básicamente el entrenamiento sigue el formato original, pero se cambia el conjunto de entrenamiento, pasando de tener imágenes regulares, a tener ejemplos adversarios. Con esto se observó que el método de defensa logra dar robustez al modelo frente ataques adversarios dado que aumenta el *accuracy* sobre conjuntos de imágenes adversarias.

Finalmente, se considera este trabajo como una introducción al mundo de la defensa adversaria, considerando que a la fecha muchos más ataques han sido planteados y métodos de defensa adversaria mucho más complejos han sido propuestos. Esto habla de la relevancia de estudiar este tema, más aún cuando los sistemas juegan roles esenciales y pueden haber agentes malignos intentando hacer fallar a los modelos con ataques como los descritos.

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

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

[8] 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.

[9] 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.

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

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

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