INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA

ESCUELA DE (INGENIERÍA Y TECNOLOGÍA – INGENIERÍA Y GESTIÓN - POSTGRADO)

UTILIZACIÓN DE REDES NEURONALES CONVOLUCIONALES PARA LA DETECCIÓN DE TIPOS DE IMÁGENES

AUTOR/ES: Cicero, Ignacio Ezequiel (Leg. Nº 103964)

DOCENTE/S TITULAR/ES O TUTOR/ES: Parpaglione, María Cristina

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE ESPECIALISTA EN

ITBA

CIENCIA DE DATOS

BUENOS AIRES

SEGUNDO CUATRIMESTRE, 2018

Índice

Tema / Título del trabajo

Utilización de técnicas de aprendizaje automático con enfoque de *big data* y procesamiento en la nube, para la detección de cierto contenido gráfico subido por usuarios en una empresa de comercio electrónico.

Estado del arte

Machine learning

Machine learning o inteligencia artificial, se define como la ciencia que busca que las computadoras actúen sin la necesidad de ser programadas explícitamente para realizar dichos actos [4], es decir, que las computadoras aprendan a partir de datos [1] y puedan realizar predicciones con un cierto nivel de precisión.

Tipos de aprendizaje *machine learning*

Supervisados

El entrenamiento se realiza con datos que incluyen las soluciones. Se dividen en tareas de clasificación y regresión. La clasificación permite a la red determinar categorías de los datos ingresados mientras que regresión permite a la red determinar un valor numérico no mostrado en los ejemplos presentados en el aprendizaje. Un ejemplo de clasificación es ingresar una foto de un gato y decirle a la red que la respuesta es "gato" mientras que un ejemplo de regresión es ingresar datos de equipamiento de un auto para predecir el precio de este [1].

Uno de los algoritmos más utilizados de este tipo es el de **redes neuronales** [1].

No supervisados

A diferencia de los sistemas supervisados, no se le incluyen las respuestas a los datos ingresados. En este caso, se buscan relaciones entre los mismos, detectando variables independientes que expliquen a las demás [1].

Un ejemplo de este tipo de algoritmos es el *K-means*, redes neuronales y *PCA* (Análisis de componentes principales *o Principal Components Analysis*) [1].

Batch

Se trata de sistemas que se entrenan una vez y se ponen en producción sin continuar con su aprendizaje. Si se quieren agregar más datos, el sistema debe ser reentrenado *offline* tanto con los datos nuevos como con datos viejos. Estos algoritmos consumen muchos recursos en términos de horas y capacidad de procesamiento para su entrenamiento [1]. Una vez reentrenado, el modelo puede utilizarse en equipos con menor capacidad de procesamiento como puede ser un teléfono celular.

Online

Se trata de sistemas que pueden aprender de forma incremental mediante lotes reducidos de datos o *mini batch*. Estos pasos son rápidos y el sistema puede aprender sin necesidad de dejarlo *offline* por varias horas [1].

Redes neuronales

La red neuronal artificial intenta emular el funcionamiento de la red de neuronas biológicas mediante una interconexión de neuronas artificiales. Una neurona artificial es la unidad mínima de procesamiento dentro de una red neuronal. Recibe N entradas provenientes de otras neuronas artificiales (análogo a la sinapsis y dendritas de las neuronas biológicas) y realiza una sumatoria ponderada de las mismas por su peso sináptico (análogo al cuerpo de la neurona biológica). Si dicha sumatoria es superior a un umbral determinado, se aplica una función de activación al resultado de dicha sumatoria para producir una única salida (análoga al axón de las neuronas biológicas) [1]. La Figura *1* muestra la topología de la neurona artificial.

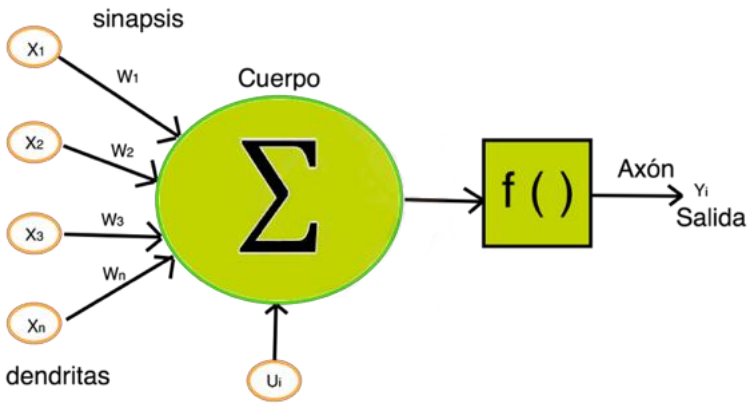

Figura 1: Topología de una neurona artificial

Las redes neuronales pueden ser de una sola capa o multicapa (con capa oculta). Las redes de una capa constan de una única neurona que procesa N entradas. En cambio, las redes neuronales multicapa constan de varias neuronas que reciben N entradas y cuyo resultado se dirige a otro conjunto de neuronas de un nivel siguiente para su procesamiento. Pueden existir varios conjuntos de neuronas dependientes que reciben los resultados de niveles anteriores y la información fluye por todos estos hasta obtener el resultado final de la red. En ambos casos, cada aplica su propia función de activación que puede no ser igual a las demás. En la *Figura 2* se puede observar una representación conceptual de una red neuronal de una capa y en la *Figura 3* se puede observar la de una red neuronal multicapa.

En el caso de las redes neuronales *supervisadas*, si el resultado se encuentra debajo del umbral establecido, se considera que el resultado es correcto y en caso de no serlo se recalculan los pesos de los nodos -*back-propagation*. Este algoritmo de aprendizaje de las redes neuronales y busca una combinación lineal de dichos pesos de forma tal que todos los valores ingresados se encuentren debajo del umbral establecido [6]. En el caso de las redes *neuronales no supervisadas* no existe el *back-propagation* ya que para que dicho cálculo tenga sentido se requiere que la red sepa cuando su resultado fue erróneo y esto no es posible ya que no se cuenta con el resultado del caso.

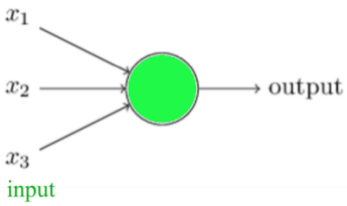

Figura 2: Representación conceptual de una red neuronal de una capa

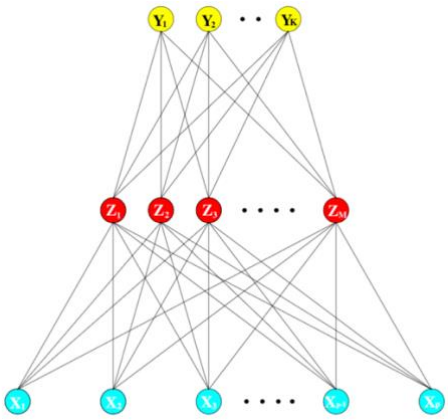

Figura 3: Representación conceptual de una red neuronal con capa oculta

Deep learning o aprendizaje profundo

En *deep learning*, se busca que las computadoras aprendan mediante el reconocimiento de patrones en los datos de entrada yendo de lo general a lo particular mediante la descomposición de dicho reconocimiento en capas, es decir, una jerarquía de conceptos. Cada representa la búsqueda un concepto a buscar: bordes, esquinas, partes de objetos, etc [7]. A través de pixeles se pueden encontrar bordes mediante la comparación del brillo con sus vecinos. Dados los bordes, se pueden buscar contornos y esquinas que se conocen por donde se intersectan varios bordes. Dadas las esquinas y los contornos, se pueden detectar partes de objetos al encontrar combinaciones específicas de los mismos. Para realizar este aprendizaje, la red debe ser entrenada con la mayor cantidad de ejemplos posibles que se encuentren diferenciados por clases, es decir, aprendizaje supervisado.

Un ejemplo de *deep learning* es el procesamiento de imágenes que es utilizado en detección de rostros o peatones para autos autónomos, siendo la técnica más utilizada en estos casos la de redes neuronales convolucionales en detrimento de las redes neuronales convencionales [1].

Redes neuronales convolucionales

Las redes neuronales convolucionales surgen del estudio de la corteza visual en gatos realizado por David H. Hubel y Torsten Wiesel. Estos autores demostraron que existen grupos de neuronas en la corteza visual que tienen un campo receptivo local. Un campo receptivo se define como una región limitada del espacio en la que un grupo de neuronas reaccionan en función a un estímulo. Cada grupo de neuronas de la corteza visual se focaliza en una región distinta del campo visual. Esto da lugar a la existencia de muchos campos receptivos que pueden superponerse y en conjunto formar un mosaico de todo el campo visual. Estos campos receptivos son los que determinan las características a buscar en un objeto para poder determinar de que se trata. También notaron que algunos grupos de neuronas tienen campos receptivos más grandes y reaccionan a patrones más complejos que son combinaciones de patrones de nivel más bajo. Estas observaciones llevaron a la idea de que los resultados obtenidos por los grupos de neuronas de nivel inferior se basan en los resultados obtenidos previamente por grupos de neuronas de nivel superior [1].

Para el reconocimiento de imágenes, lo más importante a construir es la capa convolucional que se encarga de comparar patrones de imágenes para ver su coincidencia.

Convolución

Una convolución es una operación matemática que va desplazando una función por sobre otra y calcula en cada intervalo cuán contenida se encuentra una función en la otra [17]. La definición matemática de convolución se muestra en *Ecuación 1*:

$$y(t) = \int_{-\infty}^{+\infty} f(\eta).g(t-\eta).d_\eta$$

Ecuación 1: Donde t es el tiempo y g es la función que se desplaza en el tiempo por sobre f.

Una convolución puede representarse gráficamente como muestra la *Figura 4.*

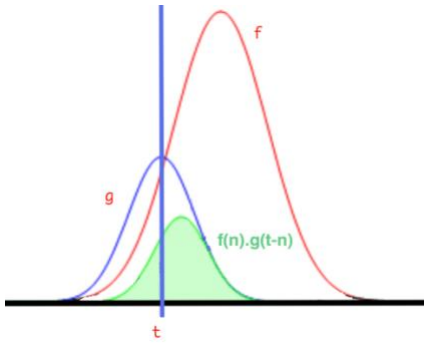

Figura 4: Representación gráfica de una convolución.

La función f es la de color rojo mientras que la g es la de color azul. El área sombreada verde muestra cuanto se superponen ambas funciones y la línea azul es el tiempo t. La *Figura 5* muestra un ejemplo simplificado de una convolución.

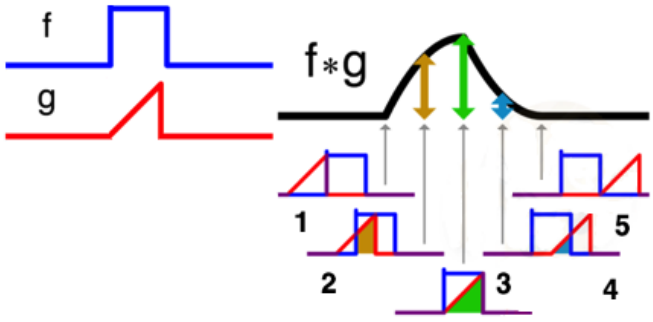

Figura 5: Ejemplo simplificado de una convolución.

Se tienen 2 funciones: *f* (azul) y *g* (roja), donde *g* se desplaza en el tiempo superponiéndose con *f*.

- Paso 1: La función *g* comienza a tocar a la función *f*.
- Paso 2: La función *g* se superpone con la función *f* por la izquierda generando el área de color marrón.
- Paso 3: La función *g* se superpone con la función *f* generando la máxima superposición, dando como resultado el área de color verde.
- Paso 4: La función *g* se superpone con la función *f* generando la mínima superposición, dando como resultado el área de color azul.
- Paso 5: La función *g* deja de superponerse con la función *f*

Capas de una red neuronal convolucional

Las redes convolucionales se componen de 3 capas [21]:

- Capa convolucional
- Capa *pooling*
- Capa densa o *fully-connected*

Capa convolucional

Esta capa aplica filtros convolucionales a la imagen con el objetivo de encontrar qué característica del modelo aplica con más fuerza.

Las neuronas en la primera capa convolucional no están conectadas a cada pixel de la imagen de entrada, sino a cada pixel de su campo receptivo. A su vez, cada neurona de la segunda capa convolucional se encuentra conectada sólo con un rectángulo de la capa superior. Esto le permite a la red concentrarse en conceptos más pequeños para luego ensamblarlos en conceptos más grandes para la siguiente capa convolucional y así hasta la última [1] como se muestra en la *Figura 6*.

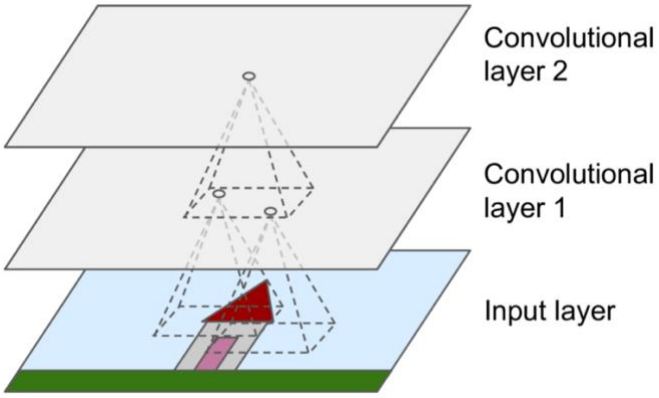

Figura 6: Capas convolucionales.

Esta capa funciona estableciendo una región de análisis (campo receptivo) que se mueve de lugar hasta haber pasado por toda la imagen. Por cada movimiento se realizan un conjunto de operaciones matemáticas para producir un único valor y se aplica la función de activación ***ReLU*** [1] para introducir valores no lineales dentro del modelo.

La función de activación ***ReLU*** es un acrónimo de ***Rectifier Linear Unit*** y consiste en quedarse solamente con los valores positivos de la entrada. Esto significa que dado un número X, la función devuelve el máximo entre X y 0. Mientras más grandes sean los valores obtenidos más parecidos a la característica buscada. Se define como indica la *Ecuación 2* y su resultado se aprecia en la *Figura 7* donde se consideran como valores positivos a aquellos de color **blanco** mientras que aquellos de color **negro** son considerados negativos.

$$f(x) = \max(x, 0)$$

Ecuación 2: Función ReLU.

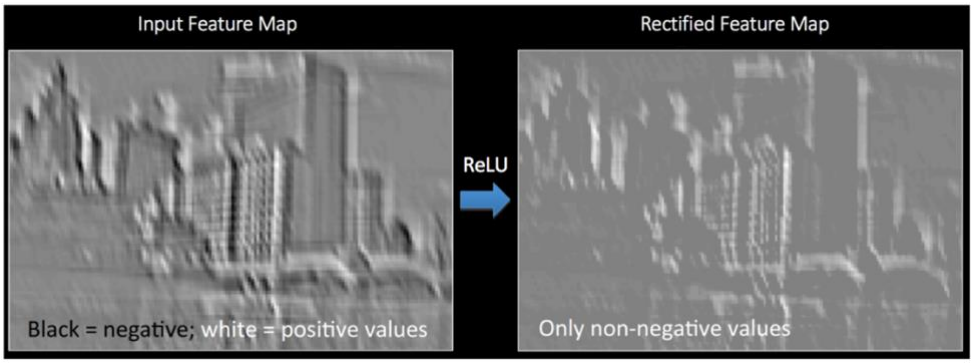

Figura 7: Ejemplo de aplicación de la función ReLU.

Una imagen es una matriz de *pixeles*[1] que puede tener hasta 16.777.215 colores distintos. Una convolución compara la imagen original con la imagen de una característica. Para ello, se toma un campo receptivo -generalmente de 7x7 o 5x5 pixeles- que se mueve de la esquina superior izquierda hasta la esquina inferior derecha de ambas imágenes y calculando la similitud de ambas en cada paso. Es importante destacar que el corrimiento de dicha sección se realiza superponiendo la mitad de la nueva ubicación con la mitad de la ubicación anterior con el fin de minimizar la pérdida de datos que pueda ocurrir en la transición entre ambas.

En la *Figura 8* se puede observar una imagen verde de 5x5 pixeles con valores 0 y 1 que se va a convolucionar con la imagen naranja de la *Figura 9.*

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Figura 8: Imagen de ejemplo sobre la que se aplicará una convolución.

1	0	1
0	1	0
1	0	1

Figura 9: Campo receptivo que se hará convolucionar con la *Figura 6*.

En la *Figura 10* se observa la superposición de la imagen naranja sobre la verde obteniendo como resultado de convolución el valor **4** surgido de la *Ecuación 3.*

[1] Es la menor unidad homogénea en color que forma parte de una imagen digital.

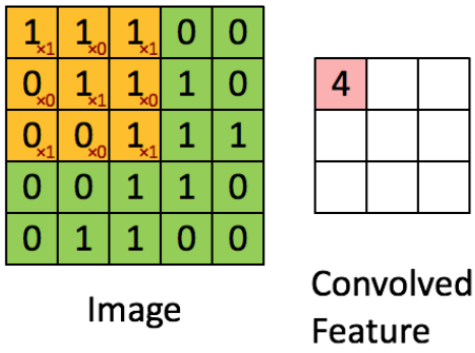

Figura 10: Primer paso de una convolución.

$$Conv = 1\ x\ 1 + 1\ x\ 0 + 1\ x\ 0 + 0\ x\ 0 + 1x1$$
$$+1\ x\ 0 + \ 0\ x\ 1 + 0\ x\ 0 + 1\ x\ 1 = 4$$

Ecuación 3: Resultado del primer paso de una convolución.

En la *Figura 11*, se observa el desplazamiento en 1 píxel de la imagen naranja sobre la imagen verde obteniendo como valor de convolución el valor **3** que surge de *la ecuación 4:*

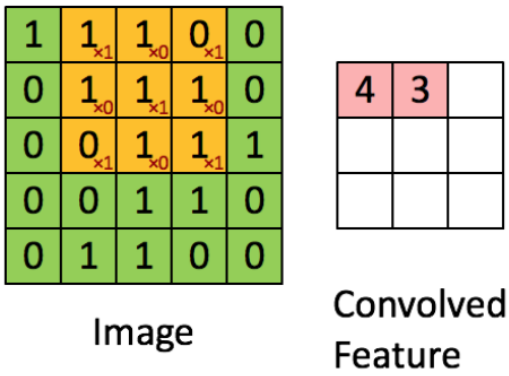

Figura 11: Segundo paso de una convolución.

$$Conv = 1\ x\ 1 + 1\ x\ 0\ + 0\ x\ 1\ + 1\ x\ 0$$
$$+1\ x\ 1 + 1\ x\ 0\ + 0\ x\ 1\ + 1\ x\ 0\ + 1\ x\ 1 = 3$$

Ecuación 4: Resultado del segundo paso de una convolución.

La convolución continua hasta obtener el valor de la *Figura 12*. Como se ve, una imagen de 5x5 pixeles se transformó en una de 3x3 pixeles.

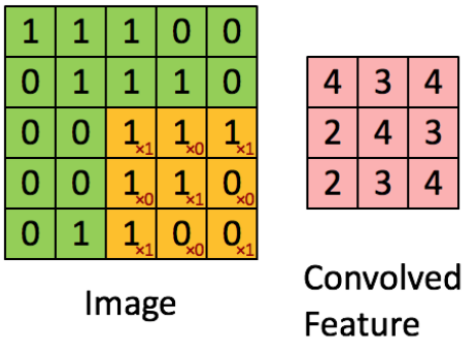

Figura 12: Resultado final de una convolución.

En la *Figura 13* se observa un ejemplo real de una convolución con valores en escala de grises que van de 0 a 255.

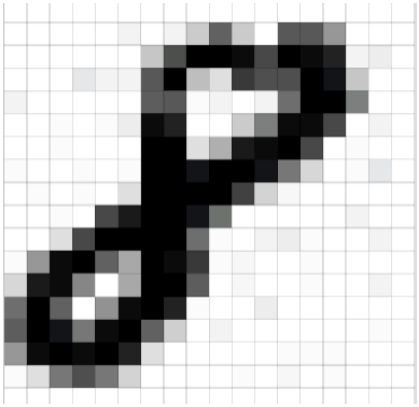

Figura 13: Ejemplo real de una imagen a convolucionar.

Luego de aplicada la convolución se obtiene la matriz de valores de la *Figura 14.*

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 14: Matriz resultado de la convolución en escala de grises.

Capa *pooling* o *sampling*

Esta capa [21] se encarga de reducir las dimensiones del resultado obtenido de la capa convolucional (*downsampling*) para reducir el tiempo de procesamiento, es decir, una compresión de los datos. Al igual que antes, las neuronas de esta capa se conectan con un número limitado de neuronas de la capa previa que a su vez tienen un campo receptivo reducido [1]. El algoritmo más utilizado es ***max-pooling*** [1] cuyo funcionamiento puede observarse en la *Figura 15*. En la *Figura 16* puede observarse un ejemplo real. Toma regiones de 2x2 pixeles y las reduce tomando el máximo valor de cada una.

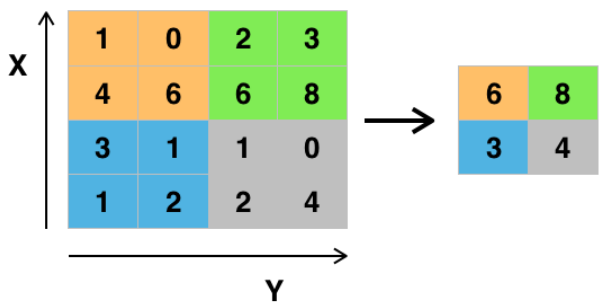

Figura 15: Ejemplo de funcionamiento de la capa de ***pool***

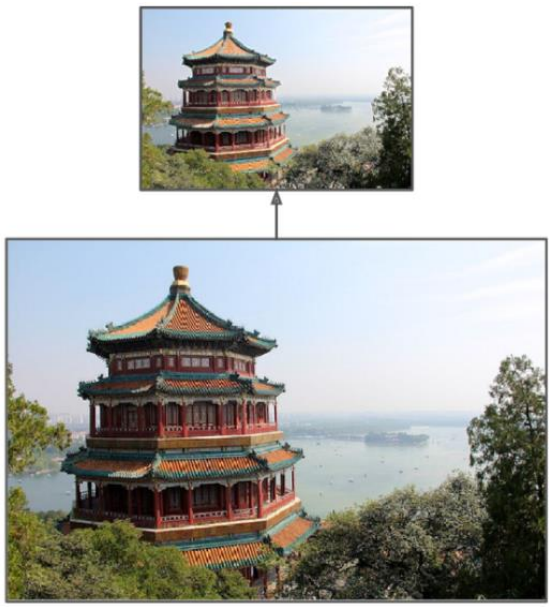

Figura 16: Ejemplo de funcionamiento de la capa de ***pool*** en una imagen fotográfica

Capa densa o *fully-connected*

Esta capa se denomina densa [21] ya que todos los nodos están conectados entre sí y su objetivo es realizar la clasificación de la imagen en base a los resultados de las capas anteriores.

Arquitectura final

Una red neuronal convolucional tiene varias capas de las mencionadas anteriormente con el fin de tener mayor precisión en los resultados. Típicamente, se tienen capas **convolucionales** intercaladas con capas de ***pooling*** para finalizar con la capa **densa**. La imagen se achica conforme se pasa de capa en capa, pero a su vez con más patrones encontrados para acercarse más a la clasificación final como muestra la *Figura 17*.

En la *Figura 17* se observa un ejemplo de funcionamiento de la red completa hasta obtener una salida.

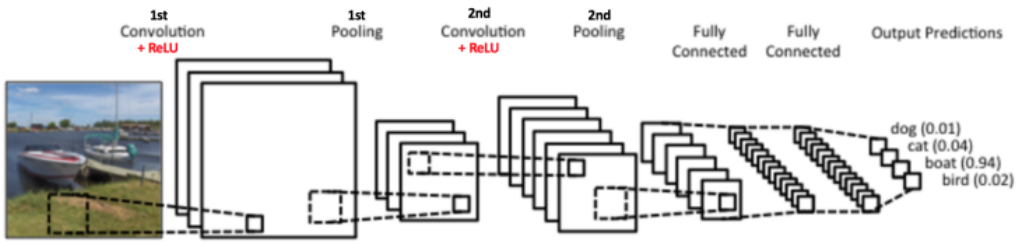

Figura 17: Arquitectura típica de una red neuronal convolucional.

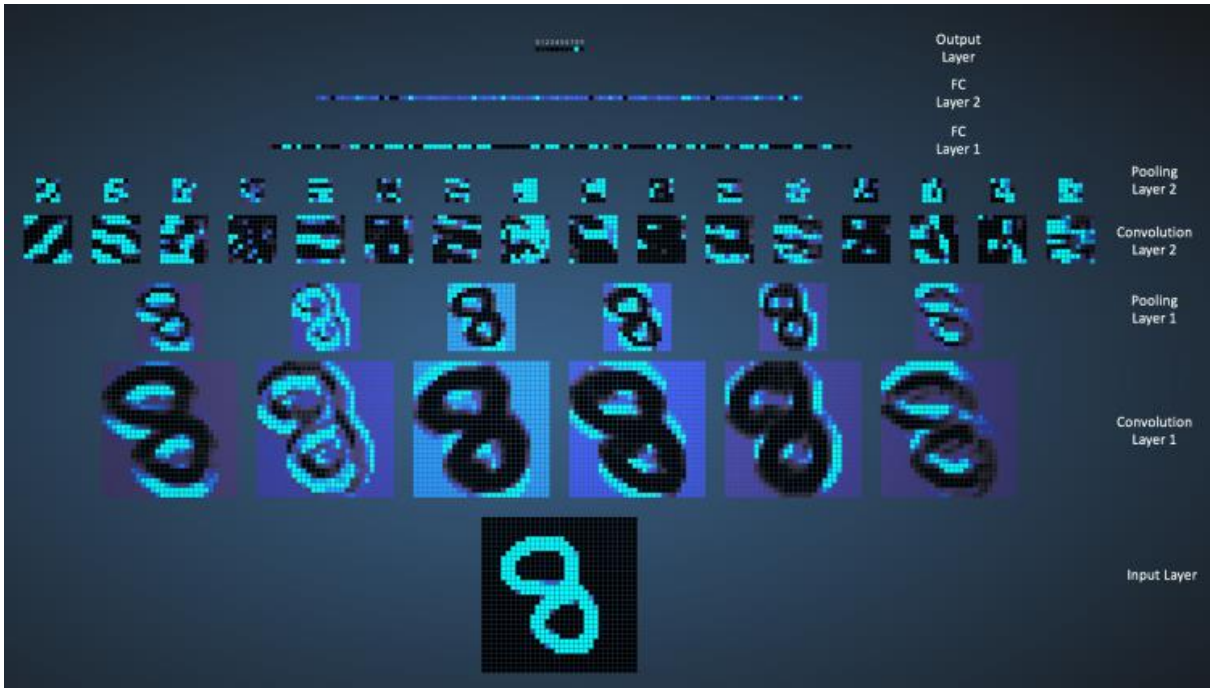

Figura 18: Ejemplo real de funcionamiento completo de una red neuronal convolucional.

Las redes neuronales convolucionales se sirven de una cierta cantidad de imágenes para su entrenamiento. Para acelerar el procesamiento de estas se utilizan placas de video, ya que sus procesadores (GPU) tienen la capacidad de trabajar en paralelo de forma más rápida que los procesadores comunes (CPU). Estas logran un flujo de datos en memoria de hasta 141 GB/s, unas 10 veces mas que un CPU convencional. En este sentido, un GPU de alta grama tiene la misma capacidad de procesamiento que un pequeño *cluster* de CPU [16], reduciendo exponencialmente el tiempo de procesamiento de una imagen. En la *Figura 19*, se observa un gráfico se comparan tiempos de procesamiento de GPU versus CPU en función al tamaño de la imagen, mientras que la *Figura 20* se observan dichas comparaciones en forma de tabla [16].

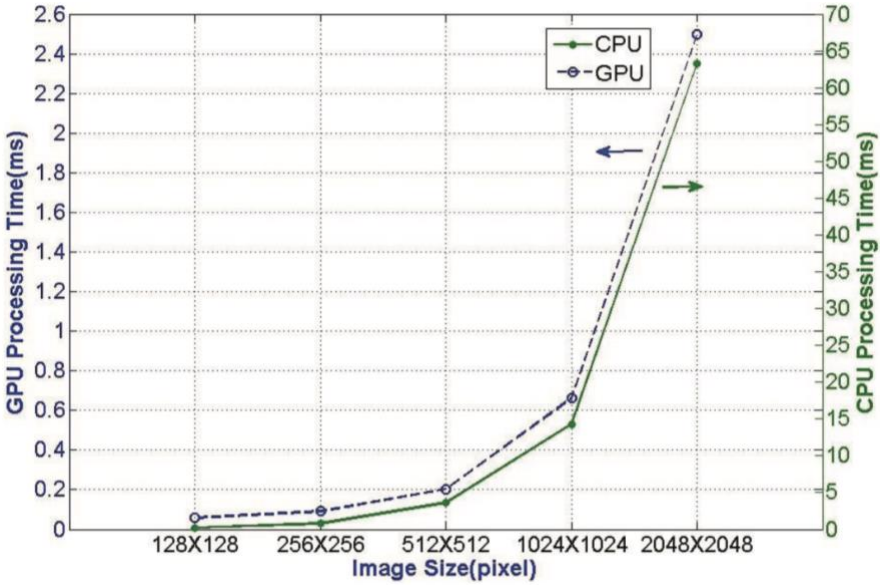

Figura 19: Comparación de procesamiento de GPU versus CPU en función al tamaño de la imagen.

Image size	GPU(ms)	CPU(ms)
128×128	1.191	10.491
256×256	1.586	34.643
512×512	4.325	202.479
1024×1024	16.387	810.211
2048×2048	77.685	3303.834

Figura 20: Tabla de comparación de procesamiento de GPU versus CPU en función al tamaño de la imagen.

Debido a esta capacidad de procesamiento de las GPU, las redes neuronales convolucionales logran una alta precisión en menor tiempo de entrenamiento que con una CPU [16][18], dicha precisión varía en función a la cantidad, calidad y diversidad de las imágenes con las que se entrena [8].

Una de las herramientas más utilizadas para la generación de estas redes es la librería *Tensorflow* del lenguaje de programación Python. "Tensorflow es una librería de software open source para computación numérica usando grafos de flujos de datos." [9]

Análisis de resultados

Como en todo modelo de aprendizaje supervisado, el conjunto de imágenes se divide en un lote de **entrenamiento** y otro de **pruebas** (generalmente en una proporción de 70/30) con el fin de obtener métricas sobre el funcionamiento del modelo. Con los resultados del modelo, se arma la **matriz de confusión** como se muestra en la *Tabla 3*. De esta matriz se pueden obtener métricas como la **exactitud** o ***accuracy*** **-** cuya forma de cálculo se aprecia en la *Ecuación 5*-, la **sensibilidad** o ***recall***- cuya forma de cálculo se aprecia en la *Ecuación 6*- y **la especificidad** -cuya forma de cálculo se aprecia en la *Ecuación 7*- [1].

		Predicción	
Realidad		Positivo	Negativo
	Positivo	Verdaderos positivos (VP)	Falsos negativos (FN)
	Negativo	Falsos positivos (FP)	Verdaderos negativos (VN)

Tabla 3. Matriz de confusión.

De esta tabla se pueden obtener las siguientes mediciones:

- **Exactitud o *Accuracy*:** Mide la cantidad de valores acertados.

$$Ex = \frac{VP + VN}{VP + VN + FP + FN}$$

Ecuación 5: Cálculo de la exactitud o *accuracy*

- **Sensibilidad o *recall***: Mide la tasa de valores positivos acertados.

$$S = \frac{VP}{VP + FN}$$

Ecuación 6: Cálculo de la sensibilidad.

- **Especificidad:** Mide la tasa de valores negativos acertados.

$$Esp = \frac{VN}{VN + FP}$$

Ecuación 7: Cálculo de especificidad.

Tecnología cloud

La computación en la nube o *Cloud Computing* (también conocida como computación en demanda) provee acceso a recursos computacionales y de almacenamiento (a medida que se va necesitando) a través de una conexión remota. Este tipo de tecnología ofrece beneficios como ser:

- No realizar una inversión inicial de dinero para disponer de un *datacenter*[2] propio ya se utilizan los recursos del proveedor *cloud*, pagando solo el tiempo de utilización y recursos utilizados.
- No tener que estimar a priori cuanta demanda de utilización va a tener una aplicación ya a esta tecnología permite agrandar o achicar la capacidad de cómputo en el momento deseado mientras la aplicación todavía se encuentra funcionando.
- Reducción de costos de mantenimiento de *hardware* y seguridad de redes. De esto se encarga el proveedor del servicio *cloud* [11].

[2] Datacenter: Centro de procesamiento de datos

Problema

Una empresa de comercio *online* que opera en 6 países de Latinoamérica da la posibilidad a los usuarios de publicar sus productos para la venta y permite que éstos suban sus propias imágenes de los productos. En estas publicaciones se han detectado imágenes cuyo contenido va en contra de las reglas del sitio y no se dispone de una forma sistematizada para moderarlas y darlas de baja.

La solución debe ser escalable y resistente al volumen de tráfico que tiene el sito (varios millones de usuarios activos y publicaciones). Por resistente, se entiende tener una disponibilidad de servicio superior al 99%.

Justificación

Esta investigación se propone alcanzar la reducción del tiempo en el que una publicación con material prohibido para las políticas de la empresa se encuentra activa en el sitio y, a su vez, facilitar el trabajo de las personas encargadas de moderar dichas publicaciones en función de las denuncias.

Actualmente las soluciones disponibles para atacar este problema incluyen tecnología desactualizada por lo que se propone la utilización de última tecnología disponible al día de la fecha. Si bien se trata de un trabajo especializado en la detección de imágenes cuyo contenido va en contra de las políticas de la mencionada empresa, esta solución se puede utilizar para la detección de cualquier tipo de objetos como armas, modelos de autos, flores, entre otros.

Alcances

La solución propuesta se utilizará en la empresa de *e-commerce* mencionada anteriormente con el objetivo de ayudar a detectar publicaciones que van en contra de las reglas del sitio.

Esta investigación incluye realizar un prototipo a escala de la solución en conjunción con una propuesta de arquitectura con la que se podría implantar. No contempla realizar un entrenamiento a escala real ni implantar la arquitectura fuera de la infraestructura de la empresa de *e-commerce*.

Limitaciones

Para proveer una solución de alta precisión, se requiere un poder de cómputo del cual no se dispone, así como también una cantidad de imágenes de entrenamiento que oscila entre las centenas de miles y varios millones.

Una vez implantada en la infraestructura computacional de la empresa de *e-commerce* se deberá realizar un entrenamiento a escala real con uno conjunto de millones de imágenes de sus publicaciones para que la precisión de la solución aumente.

Hipótesis

Es posible detectar publicaciones que contengan un tipo determinado de imágenes con una precisión superior al 70% y en un tiempo inferior a 30 minutos utilizando redes neuronales convolucionales y tecnología *cloud* para soportar un tráfico de millones de transacciones diarias.

Objetivos generales

Desarrollar una red neuronal que permita detectar, a partir de imágenes, publicaciones no permitidas en el sitio de *e-commerce*.

Objetivos específicos

- Seleccionar un conjunto de imágenes de contenido no permitido.
- Realizar una red neuronal convolucional que funcione como modelo predictivo a través de con las imágenes obtenidas.
- Tomar medidas sobre tasa de error y precisión de la red para validar el funcionamiento de la red.
- Plantear una arquitectura teórica para la implementación a gran escala de esta solución.

Metodología

Se tomará como base el modelo *Incepcion_V3* de *Google* [20] para extenderlo mediante el agregado de nuevas clases para detectar el tipo de imagen requerido. Este modelo tiene una menor tasa de error [19] y mayor precisión que los otros modelos que se encuentran disponibles en la actualidad [10]. A su vez, reconoce 1000 clases de objetos como cebras, dálmatas y lavavajillas [21].

Se definirá una arquitectura basada en procesamiento en la nube con el fin de soportar el trafico del sitio.

Solución

Se buscará implementar procesamiento de imágenes para detectar aquellas que no cumplen su reglamento [5] en tiempo cuasi-real utilizando técnicas de aprendizaje automático, en particular, el **análisis de imágenes** de las publicaciones se orientará a una **arquitectura *cloud*** para lograr una alta escalabilidad y tiempos de respuesta bajos.

Con el fin de hacer un uso óptimo de los recursos computacionales actuales, se utilizarán las últimas tecnologías disponibles como ser la librería *Tensorflow d*el lenguaje de programación *Python* y la infraestructura *cloud* de Amazon (AWS)

Para el montaje de una arquitectura que soporte el trafico del sitio, se utilizarán las soluciones SQS, S3 y EC2 de Amazon. SQS[3] es un servicio de colas de mensajes que permite conectar aplicaciones de forma asíncrona [12], S3[4] es un servicio de almacenamiento de datos donde se paga por el espacio utilizado [13] y por último un conjunto de instancias de EC2[5], que es un servicio que provee máquinas virtuales de distintas capacidades a elección en la nube las cuales deben ser configuradas completamente (a excepción del sistema operativo) [14]. Para implementar estos servicios, es necesaria la utilización de un balanceador de carga que optimice el uso de las instancias y logre que todas tengan un nivel procesamiento similar a lo largo del tiempo.

[3] SQS: *Amazon Simple Queue Service*
[4] S3: *Simple Storage Service*
[5] EC2: *Elastic Compute Cloud*

Análisis de imágenes

Para generar el modelo que permita detectar las imágenes se agregarán **nuevas características** al modelo *Inception V3* como se muestra en la *Figura 21*.

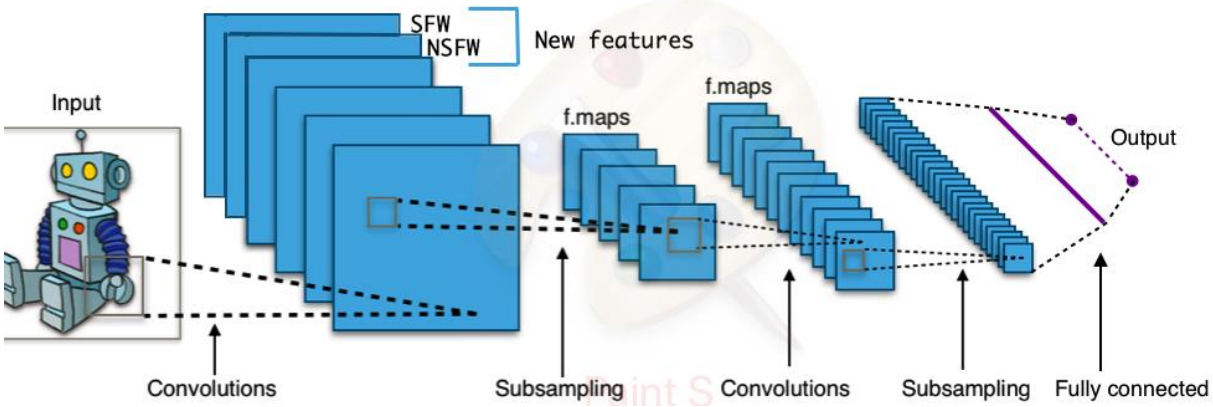

Figura 21: Nuevas características agregadas al modelo *Inception V3*

Tensorflow

Se trata de una librería de código abierto para procesamiento numérico de alto rendimiento. Provee un alto nivel de abstracción para implementar redes neuronales [9].

La unidad central de procesamiento se la conoce como *tensor*, que es un arreglo N dimensional de números. Además de sus datos internos, el tensor tiene 2 valores descriptivos: **Ranking**, que es la cantidad de dimensiones que tiene el tensor; y **Forma** que se trata de una tupla de valores enteros que especifican la longitud de cada dimensión del tensor. Dichos tensores son pasados por nodos que representan operaciones matemáticas formando un grafo donde los tensores son las esquinas de dicho grafo. "*Esta arquitectura flexible permite computar datos en múltiples GPU*[6] *con una misma interfaz de programación (API)*[7] "[9]. En la *Figura 22* puede observarse un ejemplo de grafo donde los tensores 1 y 2 representan matrices que son pasadas por un nodo cuya operación es la de multiplicarlas. La salida de dicho nodo es otro tensor con el resultado de la multiplicación de las matrices de entrada.

[6] GPU: *Graphics Processing Unit* (Unidad de procesamiento grafico)
[7] API: *Application Progamming Interface* (Interfase de programacion de aplicaciones)

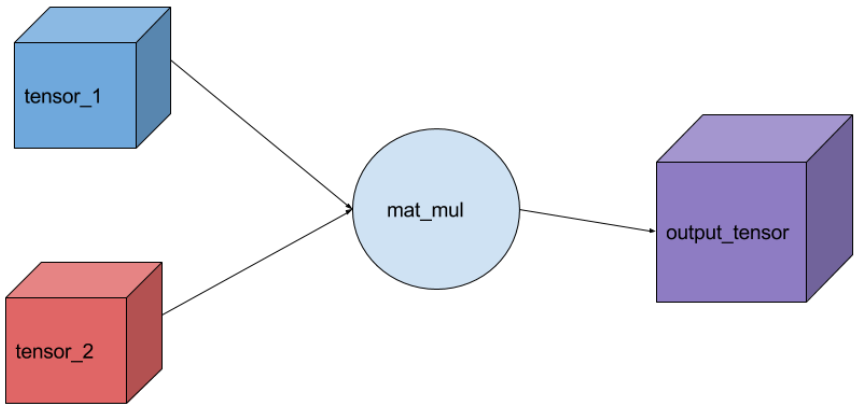

Figura 22: Ejemplo de grafo en *Tensorflow*

Los grafos deben ser parametrizados para obtener datos de entrada, para ello se introduce el concepto de *placeholder*. Un *placeholder* es una promesa de que cuando se ejecute el tensor los datos de entrada estarán disponibles por fuera del grafo [20].

Entrenamiento

El objetivo es crear un nuevo modelo con las nuevas características agregadas a través de un entrenamiento.

En el caso de *Tensorflow*, el modelo se crea mediante la definición un conjunto de pasos en los cuales se hará pasar a las imágenes por un flujo de tensores donde se aplican transformaciones y cálculos para determinar las características con las cuales la imagen tiene semejanza. El modelo resultante debe ser capaz de modificar sus valores internos para producir salidas diferentes con las mismas entradas, es decir, que se va perfeccionando sobre la marcha ajustando los valores de sus cálculos internos de forma tal de que el mismo pueda predecir la clase imágenes nuevas.

La forma de agregar parámetros de entrenamiento al grafo es a través de **capas** que contienen las variables de entrada y las operaciones a realizar con ellas. Estas se organizan de forma jerárquica y el resultado de una se pasa a la siguiente, es decir, cada capa sabe cual es su objetivo y la conjunción de capas hace finalmente al **modelo**.

Una vez definido el modelo, que puede constar de una sola o más capas, se debe entrenar definiendo el **cálculo de la pérdida** que va a tener el mismo, y a su vez un **optimizador** para lograr hacerla lo menor posible [20].

En cuanto al cálculo de la pérdida, un ejemplo es el ***Mean Square Error*** que mide la media de los cuadrados de los errores, es decir, la diferencia entre el resultado real y el

obtenido con el modelo [22]. En el caso del modelo propuesto, se decidió usar el cálculo ***Sparse Softmax Cross Entropy***, ya que se ha demostrado experimentalmente [23] que puede encontrar mínimos locales óptimos. Este ensamblado de algoritmos mide la probabilidad de error en clasificaciones discretas cuyas clases son mutuamente excluyentes. En este caso: imagen prohibida o no. Castiga al modelo que estima una probabilidad baja para la clase real.

El algoritmo ***softmax*** es una versión generalizada de la regresión logística el cual puede manejar múltiples clases en lugar de una sola con el fin de lograr un modelo extensible [1] [24]. Este algoritmo asigna probabilidades de ocurrencia a cada clase cuya suma da como resultado 1.

El hecho de que sea ***sparse*** (disperso) es que en el conjunto de probabilidades para cada elemento solo uno tendrá el valor 1 y el resto serán 0, es decir, para cada caso la clase positiva tendrá un valor 1 y las clases negativas un valor 0.

El **Cross Entropy** o **entropía cruzada** calcula pérdida de forma logística entre la probabilidad real de cada clase y la predicha por el modelo [1]. En la *Ecuación 8*, se observa la fórmula de el cálculo del error de la entropía cruzada mientras que en la *Ecuación 9* se observa el resultado del cálculo utilizando este método con los datos de la *Tabla 4* y la *Tabla 5*.

$$H(p,q) = -\sum_{x} p(x) * \log q(x)$$

Ecuación 8: Cálculo de entropía cruzada

Ejemplo:

- Probabilidad real

Clase A	Clase B	Clase C
0.0	1.0	0.0

Tabla 4. Ejemplo de probabilidad real con 3 clases.

El elemento bajo análisis es 100% de clase B. Aquí se puede observar por qué se denomina **disperso**.

- Probabilidad predicha

Clase A	Clase B	Clase C
0.228	0.619	0.153

Tabla 5. Ejemplo de probabilidad predicha con 3 clases

En este caso, la pérdida calculada por entropía cruzada es:

$$H = -(0.0\ \ln(0.228) + 1.0\ \ln(0.619) + 0.0\ \ln(0.153)) = 0.479$$

Ecuación 9: Resultado del calculo de perdida por entropía cruzada.

En cuanto a la optimización – minimizar la perdida- se utilizará el algoritmo ***Gradient Descent Optimizer*** que calcula los gradientes de la pérdida respecto de los parámetros del modelo, es decir, el mínimo de la función de pérdida. La matriz gradiente consiste en la derivada parcial de cada valor del modelo versus la función de pérdida [1].

Tomando como un ejemplo bidimensional de un valor versus el error, se tiene el grafico de la *Figura 23.*

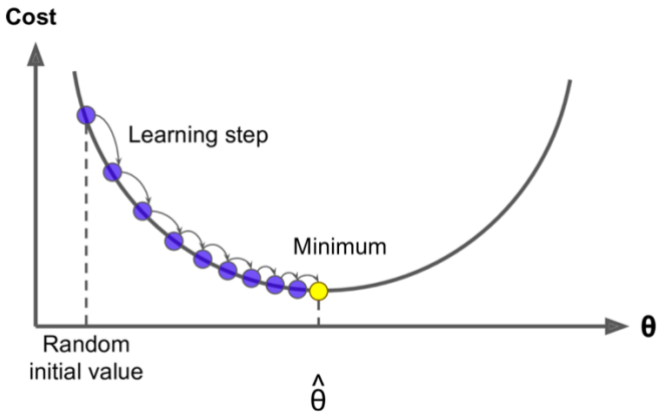

Figura 23: Error del modelo en función del parámetro $\hat{\theta}$.

En este caso, la derivada de la función en un punto indica la dirección en la que se encuentra el siguiente mínimo. Una vez sabiendo la dirección, se debe saber cuanto mover el valor del parámetro para acercarse más al mínimo de la función de pérdida. Para determinar cuanto moverse se utiliza un *híper-parámetro*[8] conocido como "tasa de aprendizaje". Este proceso se realiza por cada una de las variables de cada ejemplo provisto. Si dicha tasa de aprendizaje es muy baja, la optimización tardará más tiempo en encontrar el mínimo y si es muy alta, puede que no lo encuentre, o que encuentre un mínimo local que no es realmente el mínimo real de la función.

A modo de ejemplo se realizó un entrenamiento con 962 imágenes perros y 982 de gatos. Cabe destacar que en la empresa de *e-commerce* se utilizará un modelo entrenado con imágenes apropiadas para lidiar con su problemática.

Para el caso de los gatos se buscarán características como:

- Sus orejas que se asemejan a un triángulo como muestra la *Figura 24*.

Figura 24: Ejemplo de oreja de gato.

- Sus pupilas de forma elíptica vertical y angostas como muestra la *Figura 25*.

Figura 25: Ejemplo de ojo de gato.

[8] Híper-parametro: *Parametros para la creación un modelo. Ejemplo: Tasa de aprendizaje requerida o cantidad de capas.. En cambio, un parámetro normal del modelo es algo que se desea analizar mediante el mismo, en este caso una imagen.*

- Sus bigotes, más visibles que en los perros como muestra la *Figura 26*.

Figura 26: Ejemplo de bigotes de gato.

- Su hocico, mucho más pequeño que el del perro como muestra la *Figura 27*.

Figura 27: Ejemplo de hocico de gato.

- Su nariz rosada en la mayoría de los casos como muestra la *Figura 28*.

Figura 28: Ejemplo de nariz de gato.

- Su tamaño generalmente menor a los perros.

Para el caso de los perros se buscará:

- Su lengua que en general se halla fuera de su boca y tiene forma rectangular, redondeada en un extremo y de un color rosa como muestra la *Figura 29*.

Figura 29: Ejemplo de lengua de perro.

- Sus dientes más visibles, grandes y de color marfil en combinación con la lengua fuera, como muestra la *Figura 20*.

Figura 20: Ejemplo de dientes y lengua de perro.

- Su nariz más grande, visible y de color negro como muestra la *Figura 31*.

Figura 31: Ejemplo de nariz de perro.

- Sus orejas mayormente caídas como muestra la *Figura 32*.

Figura 32: Ejemplo de oreja de perro.

- Su hocico, mucho más grande que el del gato como muestra la *Figura 33*.

Figura 33: Ejemplo de hocico de perro.

- Su collar que es utilizado frecuentemente y que tiene forma de dos medios círculos concéntricos como muestra la *Figura 34*.

Figura 34: Ejemplo de collar de perro.

- Su correa que también es utilizada con frecuencia como muestra la *Figura 35*.

Figura 35: Ejemplo de correa de perro.

- Su tamaño generalmente mayor a los gatos.

Monitoreo del entrenamiento

Este entrenamiento fue realizado con 100 corridas y aplicando distorsiones. Mediante la herramienta gráfica provista por *Tensorflow* para monitorear el entrenamiento, denominada ***Tensorboard***, se obtuvo el gráfico de progresión de la exactitud de la *Figura 36* y el grafico de la progresión de la reducción del error (***cross_entropy***) de la *Figura 37*. En ambos casos, la curva naranja representa a las imágenes de entrenamiento y la azul representa a las imágenes de prueba.

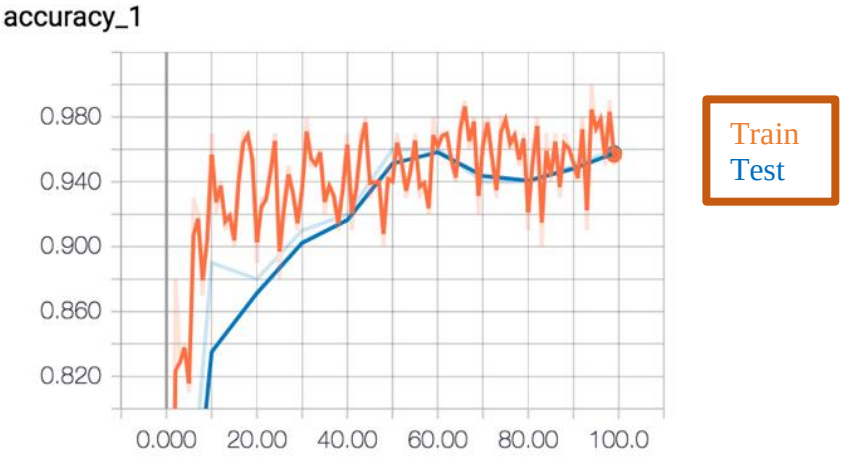

Figura 36: Progresión de la exactitud del modelo durante el entrenamiento.

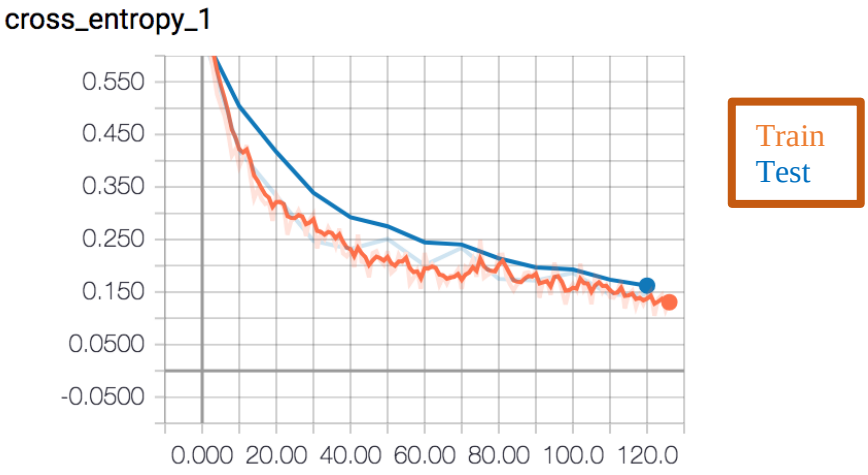

Figura 37: Progresión del error del modelo durante el entrenamiento.

Creación del modelo

La creación del modelo con las nuevas características agregadas se compone de 8 pasos:

1- Se descarga del modelo *Inception_V3* sobre el cual se agregarán las nuevas características.

2- Se cargan imágenes asociadas a sus clases dividiendo lotes de imágenes para entrenamiento y prueba.

3- Se verifica si se deben aplicar distorsiones para lograr mayor aleatoriedad en las imágenes y que el modelo se ajuste más a los casos de la vida real. Las distorsiones incluyen:

i. **Recortes:** Tomar una sección de la imagen como muestra la *Figura 38.*

Figura 38: Ejemplo de recorte de imagen.

ii. **Escalar:** Reducción del tamaño de la imagen.

iii. **Cambios de brillo**

4- En caso de no aplicar distorsiones, se crea un *cache*[9] de las imágenes denominado *bottleneck* que se trata de una capa previa a la capa final (densa) con la información suficiente para distinguir todas clases [25]. Esto se crea con el fin de reducir el tiempo de entrenamiento ya al no aplicar distorsiones las imágenes serán siempre las mismas, es decir, sin aleatoriedad y por ende se evita analizarlas todas al ejecutar nuevamente el entrenamiento.

5- Se crean las nuevas características a aplicar en modelo *Inception_V3* para lo cual se define como método de calculo del error al ***Sparse Softmax Cross Entropy*** y el ***Gradient Descent Optimizer*** como método optimización o, en este caso, minimización del error obtenido.

6- Se agrega un paso de evaluación de la predicción donde compara lo predicho por el modelo con la clase real calculando la **exactitud** del modelo de forma continua.

[9] Cache: *Memoria intermedia con datos sumarizados para acelerar futuros procesos sin leer toda la información nuevamente.*

7- Se realiza una cierta cantidad de pasadas de entrenamiento con el conjunto de imágenes cargadas para este, es decir, se ejecuta el modelo varias veces con todas las imágenes recalculando sus valores continuamente para lograr la mayor cantidad de clases acertadas.

8- Se realiza la prueba del modelo con un lote de imágenes con las que el modelo no fue entrenado para determinar las métricas finales del mismo.

Ejecución del modelo

Una vez que se considera que el modelo se encuentra lo suficientemente exacto, se comienza a usar para predecir imágenes nunca vistas, como es el caso de aquellas subidas por los usuarios al sitio. Para esta propuesta prototipo se realizó una aplicación web con interfaz RESTful[10] con el *framework* [11] **Flask** de *Python*.

La idea de esta aplicación es que se suba la imagen a la interfaz RESTful mediante el método POST[12], se cargue el modelo generado en el entrenamiento y se le aplique a la misma, obteniendo como respuesta la probabilidad de que dicha imagen pertenezca a una de las clases con las que el modelo fue entrenado.

Resultados

Luego de haber realizado el entrenamiento del modelo y de tener la aplicación funcionando, se ingresan imágenes nunca vistas para probar su funcionamiento. Al ingresar la imagen del gato de la *Figura 39*, se obtuvo como resultado las probabilidades que se observan en la *Figura 40*. Al ingresar la imagen del perro de la *Figura 41*, se obtuvo como resultado las probabilidades que se observan en la *Figura 42*.

Por último, se realizó la prueba de ingresar la imagen de la *Figura 43* que no corresponde a ninguna de las clases con las que se entrenó el modelo y se obtuvo como resultado las probabilidades de la *Figura 44* en la cual se observa una probabilidad relativamente baja para ambas clases, estas decrecerán aún más conforme se aumente el número de pasadas de entrenamiento y la cantidad de imágenes utilizadas.

[10] RESTful: *Un estándar de creación de aplicaciones web que se basa en los métodos del protocolo http.*
[11] Framework: *Una librería que determina una forma de trabajo y componentes de un sistema.*
[12] POST: *Método del protocolo http que permite, entre otras cosas, enviar contenido que no es texto plano.*

Figura 39: Imagen de prueba (gato) nunca vista por el modelo.


```
{
  "cat": "0.8040027",
  "dog": "0.19599733"
}
```


Figura 40: Respuesta de la aplicación ante una nueva imagen de gato.

Figura 41: Imagen de prueba (perro) nunca vista por el modelo.


```
{
  "cat": "0.2985821",
  "dog": "0.7014179"
}
```


Figura 42: Respuesta de la aplicación ante una nueva imagen de perro.

Figura 43: Imagen de prueba no correspondiente a las clases entrenadas.


```
{
  "cat": "0.2985821",
  "dog": "0.57479614"
}
```


Figura 44: Ejemplo de respuesta de la aplicación ante una nueva imagen de gato.

Arquitectura cloud

Se dispondrá de colas SQS donde los sistemas de la empresa publicarán las fotos a analizar. El hecho de utilizar un servicio de colas de mensajes se debe a que las mismas permiten comunicación asíncrona entre sistemas, además de disminuir el nivel de acoplamiento entre ellos. Esto es: El usuario sube una foto, el sitio publica un mensaje a la cola, se desentiende del resultado de este y continua su ejecución sin bloquear otras tareas. Si debe recibir un resultado producto del mensaje enviado anteriormente, se debe contar con un consumidor de mensajes de otra cola donde estarán los resultados del procesamiento.

Otra ventaja de la utilización de mensajería por colas es que permite la independencia de tecnología de los sistemas involucrados. Estos solo deben poder enviar y recibir mensajes desde y hacia el motor de colas, pero no importa con que tecnología estén desarrollados.

Particularmente SQS tiene una restricción de 256 *KB* de tamaño para un mensaje, con lo cual se deberá utilizar la su API extendida que permite enviar mensajes de hasta 2GB de tamaño. A su vez, estos mensajes no pueden contener información binaria como una imagen por lo cual la misma deberá ser codificada en BASE64 [30].

Una vez que los mensajes se encuentren en el sistema de colas, del otro lado se dispondrá de un conjunto de instancias EC2 que proveen consumidores de éstos, y con capacidad de procesamiento tal de poder consumir y responder a todos los mensajes que lleguen. En este punto, el balanceador de carga asegura dos cosas: que todas las instancias de consumidores de mensajes tengan una tasa de trabajo casi equivalente y, por otro lado, permite agregar más instancias de consumidores para aumentar la capacidad de procesamiento.

Estos consumidores tomarán la imagen que llega por el sistema colas y la analizarán mediante el modelo surgido del entrenamiento de la red neuronal para determinar la probabilidad de que la imagen pertenezca a una de las clases con las que se entrenó el modelo.

El modelo se encontrará almacenado en el contenedor de archivos de AWS denominado S3 con el fin de que todas las instancias de procesamiento puedan acceder al mismo. Este contenedor de archivos también almacenará imágenes para el reentrenamiento de la red.

Los resultados de ese análisis se enviarán a otro sistema de colas para que los moderadores del sitio decidan que hacer con la imagen.

El flujo de información que se plantea implementar consiste en que al subir fotos en una publicación la misma vaya al sistema de colas de AWS y el resultado del análisis sea informado a los sistemas del sitio mediante otro sistema de colas.

El resultado del análisis consiste en la probabilidad de que una imagen corresponda a cada categoría con las que fue entrenado. En un comienzo, y recordando que el modelo de análisis de imágenes puede ser reentrenado agregando más casos para mejorar su precisión, se debe establecer un umbral de probabilidad sobre el cual un moderador del sitio confirme si la imagen es efectivamente prohibida o si se trató de un falso positivo. Si la detección fue correcta, la imagen se debe almacenar en S3 para reentrenar la red en el futuro. Para este reentrenamiento se debe establecer la cantidad de imágenes nuevas que harán que se ejecute el mismo, es decir, establecer un tamaño de lote de reentrenamiento.

Una vez alcanzada dicha cantidad de imágenes, la red debe ser reentrenada de forma *offline* para luego ser actualizada en el contenedor de archivos y que las instancias de procesamiento lo utilicen.

En la *Figura 45* puede observarse de forma gráfica el diseño de esta arquitectura *cloud* para dar soporte al trafico que tiene el sitio.

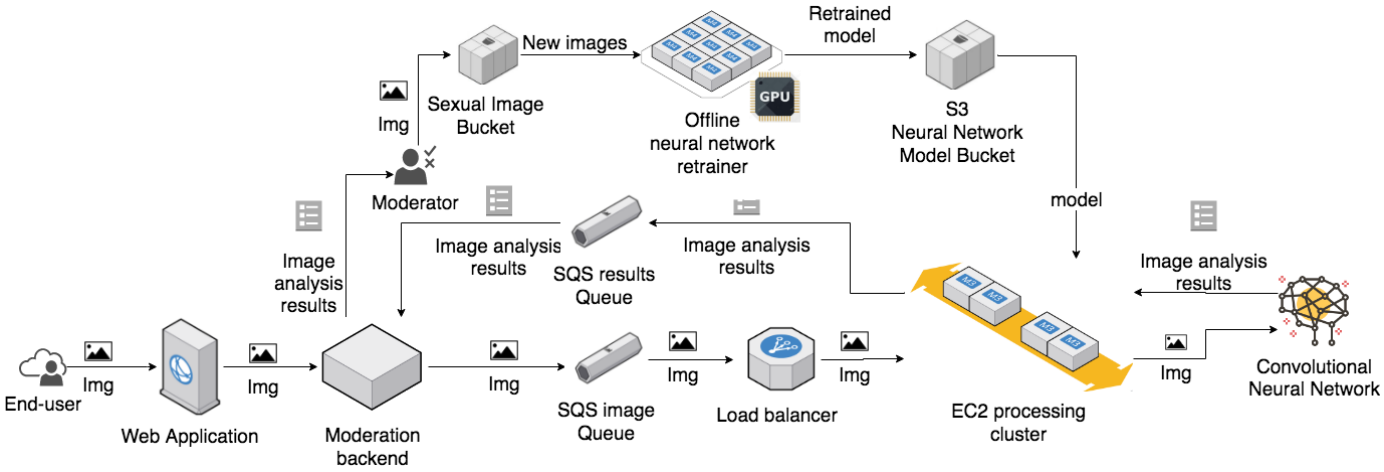

Figura 45: Arquitectura *cloud* montada en AWS

Otras tecnologías similares

Actualmente la empresa *Amazon* ofrece un servicio de *machine learning cloud* denominado *SageMaker* [26] que permite, entre otras cosas, realizar clasificación de imágenes en múltiples categorías utilizando la red convolucional denominada *RestNet* [27]. La misma compañía ha publicado un ejemplo de funcionamiento [28] combinando la base de datos no relacional *DynamoDB*, *S3*, *Athena* y *SageMaker* para la predicción de comportamiento de clientes de un banco.

A su vez, la compañía provee un servicio *online* denomina *AMI* [29] para agilizar las tareas de *machine learning* en la nube a cualquier escala que puede ejecutar distintas librerías como por ejemplo *Tensorflow*.

Referencias

1. Géron, Aurélien. (2017) Hands-On Machine Learning with Scikit-Learn and TensorFlow. Estados Unidos. O'Reilly Media Inc.

2. Wang, Jue y Tao, Qing. (2008). Machine Learning: The State of the Art. Institute of Automation. AI in China Pág. 49-55.

3. Explained: Neural networks [En línea]. MIT news office. Cita: 10/3/2018. http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

4. Machine learning [En línea]. Stanford University at Coursera. Andrew Ng. Cita: 10/3/2018. https://es.coursera.org/learn/machine-learning/lecture/Ujm7v/what-is-machine-learning.

5. Reglamento: Productos o servicios para adultos [En línea]. Mercado Libre S.R.L. Cita: 10/3/2018. https://www.mercadolibre.com.ar/ayuda/Que-puedo-vender_675

6. Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The elements of statistical learning. 2nd edition. Estados Unidos. Springer.

7. Ian Goodfellow; Yoshua Bengio; Aaron Courville. (2016). Deep Learning. Estados Unidos. MIT Press.

8. Image Recognition and Object Detection [En línea]. Cita: 20/3/2018. https://www.learnopencv.com/image-recognition-and-object-detection-part1/

9. About *Tensorflow* [En línea]. Cita: 11/3/2018. https://www.tensorflow.org/.

10. Mojumder, Uttam; Sarker, Toqi Tahamid; Monika, Gulnahar Mahbub y Ratul, Nurul Amin. (2016). Vehicle Model Identification using Neural Network Approaches. BRAC university.

11. What is cloud computing [En línea]. Amazon. Cita: 17/3/2018. https://aws.amazon.com/es/what-is-cloud-computing/

12. Amazon SQS [En línea]. Cita: 17/3/2018. https://aws.amazon.com/es/sqs/

13. Amazon S3 [En línea]. Cita: 17/3/2018. https://aws.amazon.com/es/s3/

14. Amazon EC2 [En línea]. Cita: 17/03/2018. https://aws.amazon.com/es/ec2/

15. Smith, Steven W. (1999). The Scientist and Engineer's Guide to Digital Signal Processing. Estados Unidos. California Technical Publishing.

16. Zhang, Nao; Chen, Yun-Shan; Wang-Jian-li. (2010). Image Parallel Processing Based on GPU. Advanced Computer Control (ICACC), 2010 2nd International Conference on. IEEE. Shenyang, China.

17. Szegedy, Christian; Vanhoucke, Vincent; Ioffe, Sergey; Shlens, Jonathon; Wojna, Zbigniew. (2015). Rethinking the Inception Architecture for Computer Vision. Estados Unidos. Cornell University.

18. Image recognition [En línea]. Cita: 17/3/2018. https://www.tensorflow.org/tutorials/image_recognition

19. Image Classification Transfer Learning with Inception v3 [En línea]. Cita: 17/3/2018. https://codelabs.developers.google.com/codelabs/cpb102-txf-learning/index.html#1

20. Tensorflow programming guide [En línea]. Cita: 05/05/2018. https://www.tensorflow.org/programmers_guide/low_level_intro

21. Into convolutional neural networks [En línea]. Cita: 06/05/2018 https://www.tensorflow.org/tutorials/layers

22. Wackerly, Dennis; Scheaffer, William (2008). Mathematical Statistics with Applications (7 edición). Estados unidos. Thompson Higher Educations.

23. Pavel Golik, Patrick Doetsch, Hermann Ney (2013). Cross-Entropy vs. Squared Error Training: A Theoretical and Experimental Comparison. Aachen University.

24. Softmax Regression [En línea]. Citado 13/05/2018 http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

25. How to Retrain an Image Classifier for New Categories [En línea]. Cita: 13/05/2018. https://www.tensorflow.org/tutorials/image_retraining

26. What is Amazon SageMaker [En línea]. Cita 25/05/2018. https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html

27. Image classification algorithm [En línea]. Cita: 25/05/2018. https://docs.aws.amazon.com/sagemaker/latest/dg/image-classification.html

28. Analyze data in Amazon DynamoDB using Amazon SageMaker for real-time prediction [En línea]. Cita: 25/05/2018. https://aws.amazon.com/es/blogs/big-data/analyze-data-in-amazon-dynamodb-using-amazon-sagemaker-for-real-time-prediction/

29. AMI de aprendizaje profundo de AWS [En línea]. Cita: 25/05/2018. https://aws.amazon.com/es/machine-learning/amis/

30. Amazon SQS limits [Em línea]. Cita 05/06/2018. https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-limits.html