COMPUTACIÓN E INFORMÁTICA
Recibido *11/11/2019* ***ReCIBE, Año 9 No. 1, Junio 2020***
Aceptado *20/06/2019*

Título: Reconocimiento de la denominación de billetes a través de una aplicación móvil con reconocimiento de imagen

M.T. I. FRANCISCO GUTIERREZ VERA[1]
francisco.gutierrez@itcelaya.edu.mx

ALEJANDRO COTA RORIGUEZ[1]
janocota@hotmail.com

MIGUEL ANGEL SIERRA LOZADA[1]
14030667@itcelaya.edu.mx

Claudia Cristina Ortega González[1]
claudia.ortega@itcelaya.edu.mx

[1]Tecnológico Nacional de México en Celaya

Resumen.

Es sabido que el reconocimiento de objetos en imágenes está a la orden del día. No es raro observar que Facebook recomienda a sus usuarios etiquetar personas en fotografías, por mencionar algún ejemplo. En varios países, especialmente Estados Unidos y miembros de la Unión Europea, ya se cuenta con herramientas para detectar billetes y sus denominaciones a través de aplicaciones móvil.

En esta investigación se busca el desarrollo de una herramienta capaz de reconocer y clasificar los billetes mexicanos actualmente en circulación, a través de una aplicación móvil capaz de tomar fotografías o seleccionar imágenes existentes para la detección, se creó una red neuronal entrenada para lograr nuestro objetivo: una detección y clasificación de billetes que sea eficiente en al menos el 90% de los casos.

1. Introducción

Zaforas (2017) Dice que el enorme desarrollo que está viviendo la tecnología asociada a la Inteligencia Artificial (IA) está dando lugar en los últimos tiempos a nuevas herramientas y aplicaciones espectaculares. Una de las áreas donde los avances han sido más notables es el reconocimiento de imágenes, en parte gracias al desarrollo de nuevas técnicas de Deep Learning o aprendizaje profundo. Hoy en día tenemos a nuestro alcance sistemas más precisos que los propios humanos, en las tareas de clasificación y detección en imágenes.

Algunas de las aplicaciones que tiene el reconocimiento de imágenes son:

- Etiquetado de imágenes
- Verificación de usuarios basada en rostros
- Análisis de opinión
- Análisis de clientes
- Diagnóstico de enfermedades
- Realidad aumentada
- Detección de matriculas

También comenta que una de las tareas más complejas a la hora de abordar proyectos que integren técnicas avanzadas de IA es el desarrollo de los modelos y la puesta en producción de los mismos. En el caso de las técnicas de reconocimiento de imágenes, existen modelos pre-entrenados que las principales plataformas en la nube nos ofrecen. Algunos ejemplos son AWS Rekognition, Google Cloud Visión API y Azure Computer Visión API. También existen plataformas que permiten el entrenamiento de un modelo propio, como es el caso de TensorFlow.

En diversos países ya se ha llevado a cabo el reconocimiento de imagen para la detección de su respectivo papel moneda. En 2015 el Instituto Nacional de Tecnología Industrial de Córdoba, Argentina, publicó un artículo que habla del proyecto "Software libre de Reconocimiento de billetes para personas en situación de discapacidad visual", que busca ayudar a los débiles visuales en el uso diario de los billetes argentinos, de manera que no tengan que depender de un mediador que les diga la denominación de los billetes. Este desarrollo se sustenta principalmente en la utilización de una librería denominada OpenCV para el procesamiento de las imágenes. Este proyecto fue desarrollado por Moretti, Amado, Puntillo, Caniglia (2015)

Gayon (2017) presentó el proyecto de titulación "DESARROLLO DE UNA APLICACIÓN PARA RECONOCIMIENTO DE BILLETES POR MEDIO DE PROCESAMIENTO DE IMÁGENES PARA PERSONAS CON DIVERSIDAD VISUAL BASADA EN TECNOLOGÍA ANDROID" desde la Universidad Libre Sede Bosque Popular en Bogotá Colombia. El cual también busca ayudar a las personas con diversidad visual a poder identificar con facilidad los diferentes billetes con los que se cuanta en Colombia. En este proyecto también se usó como núcleo la librería OpenCV. Este asunto de las personas con diversidad visual no es un tema menor, así lo indican Singh, Choudhury, Vishal y Jawahar (2014) en donde establecen que para 2013 había 285 millones de personas con esta situación.

A pesar de los resultados exitosos que se han obtenido en dichos países, aun no se cuenta con una aplicación móvil que sea capaz de detectar los billetes mexicanos en sus diferentes denominaciones. Para buscar estas aplicaciones se realizó una indagación a través de la Play Store de Google con los criterios de búsqueda "Billetes mexicanos", "Detector de billetes mexicanos" y "Reconocedor de billetes mexicanos", de estas búsquedas se revisaron los primeros 100 resultados, entre los cuales no se encontró alguno que sea una aplicación capaz de reconocer la denominación de los billetes mexicanos.

La presente investigación busca desarrollar una aplicación móvil que sea capaz de reconocer y clasificar por su denominación el papel moneda mexicano (veinte, cincuenta, cien, doscientos y quinientos), como medio coadyuvante en le detección de billetes. Se busca que la aplicación sea capaz de identificar correctamente dichos billetes desde fotografías tomadas en diferentes posiciones, diferente iluminación y con diferentes contrastes, esto con una precisión de al menos 90%.

2. Metodología

La metodología consta de siete etapas (ver figura 1), las cuales se explican a detalle en la presente sección.

Figura 1

Metodología en cascada de las etapas del proyecto (Imagen Propia).

2.1 Análisis de herramientas factibles

A lo largo de esta etapa se investigó acerca de las diferentes herramientas o *frameworks* disponibles en la actualidad que se ajustan al propósito de la presente investigación. Inicialmente, las herramientas consideradas fueron las siguientes:

- Tensorflow y Tensorflow Lite.
- OpenCV.
- Azure Computer Vision API

Se consideró que la herramienta a usar debía cumplir con ciertos requisitos, los cuales se establecieron de la manera siguiente:

- Procesamiento de imágenes. La herramienta o *framework* debe ser capaz de recibir una imagen como entrada de datos para realizar la inferencia necesaria y proporcionar resultados correctos.
- Detección de objetos. La herramienta o *framework* debe ser capaz de reconocer objetos de ciertas clases bien definidas (billetes mexicanos) y proporcionar como resultado las coordenadas de una región rectangular que enmarque cualquier detección.
- Aprendizaje automático. La herramienta o *framework* debe ser capaz de recibir un conjunto de datos etiquetados para generar su propio conocimiento acerca del problema y otorgar resultados relevantes.
- Rapidez en el procesamiento. La herramienta o *framework* debe ser capaz de realizar la inferencia de forma rápida en los límites de usabilidad aceptables, para que la experiencia del usuario final sea favorable.
- Compatibilidad con el sistema Android. Dado a que se propone construir una aplicación para el sistema Android, es estrictamente necesario el cumplimiento de este requisito.
- Procesamiento local. La herramienta o *framework* debe ser capaz de trabajar sin conexión a internet.
- Facilidad del desarrollo. Opcional, el uso de la herramienta o *framework* debe estar bien documentado. Es deseable que la herramienta se ajuste a ciclos de desarrollo por prototipos.

Al haber comparado las características ofrecidas por cada herramienta contra las establecidas se descartó Azure Computer Vision API, debido a que se trata de una plataforma en la nube que requiere forzosamente una conexión a internet. Además, no ofrece la detección de objetos como tal, sino la clasificación de una imagen completa (sin enmarcar las detecciones).

Por otro lado, OpenCV y Tensorflow ofrecen características muy similares. Sin embargo, Tensorflow ofrece una librería compatible optimizada para Android (Tensorflow Lite), lo cual implica un mejor desempeño al momento de realizar la inferencia. Consideramos de gran importancia que la herramienta sea rápida, así que al final se optó por Tensorflow y Tensorflow Lite en conjunto. Adicionalmente, Tensorflow cuenta con modelos preestablecidos para el trabajo de ciertas situaciones, siendo una de ellas el reconocimiento de objetos. Esto nos permite agilizar el desarrollo al construir sobre un modelo existente.

2.2 Establecimiento del marco de trabajo

En esta etapa, se procedió al establecimiento de las características del modelo. Cabe destacar que Tensorflow trabaja con redes neuronales para el reconocimiento de objetos, por lo que hay ciertos aspectos y parámetros que se deben tener en cuenta. En este tipo de desarrollos lo ideal es tomar modelos de redes existentes y adaptarlos a la medida. Ahora, si se toma en cuenta que una red neuronal promedio suele ser enorme para un dispositivo móvil, se debe seleccionar el modelo con cuidado. De acuerdo con Howard et al. (2017) existe un modelo de red neuronal eficiente para

aplicaciones móviles y embebidas llamado *MobileNet*, que soluciona la cuestión previamente mencionada. En conjunto con la red neuronal es necesario establecer un algoritmo de detección. Uno de ellos optimizado para realizar detecciones, generar propuestas de detección, y arrojar las cajas de detección en un solo pase es el algoritmo SSD (*Single-Shot Detector*), propuesto por Liu et al. (2015). De igual manera, se elige este algoritmo por su eficiencia para su aplicación en un entorno móvil. Como algoritmo de optimización para el entrenamiento se eligió el *optimizador ADAM*, ya que según Kingma y Ba (2014) resulta bastante eficiente para casos de aprendizaje con datos que tienden a variar o incorporar "ruido" (siendo el caso de las condiciones de iluminación, rotación, distorsión y truncamiento de los billetes).

Otro aspecto importante del desarrollo de este tipo de modelos es que el hecho de construir una red neuronal desde cero y entrenarla hasta la convergencia podría tomar meses. Por esto, es recomendable tomar una red neuronal previamente entrenada con otro conjunto de datos y construir sobre la misma. Para este caso, en el repositorio de Github de Tensorflow existen algunos modelos ya entrenados. Este proyecto toma como base el modelo entrenado con el conjunto de datos COCO (*Common Objects in Context*), cuyo propósito descrito por Lin et al. (2014) es detectar más de 90 objetos comunes que serían reconocidos por un niño de 4 años.

La red neuronal recibe como entrada la imagen redimensionada a 300 pixeles de ancho por 300 pixeles de alto. Como salida, la red neuronal proporciona cuatro tensores que contienen las ubicaciones de las cajas de las detecciones, las clases de las detecciones, el grado de confianza de las detecciones, y el número de detecciones (ver figura 2, grafo de inferencia y red neural). A partir de estos resultados, escalar las dimensiones de las cajas y traducir las clases a los tipos de billetes detectados para dibujar sobre la imagen original es una tarea trivial. Una vez definido el modelo y sus parámetros, se estableció el flujo de trabajo de acuerdo al diagrama de la figura 3. El desarrollo siguió un patrón iterativo basado en el entrenamiento del modelo.

Figura 3

Flujo de trabajo para el entrenamiento y liberación de la app (imagen Propia).

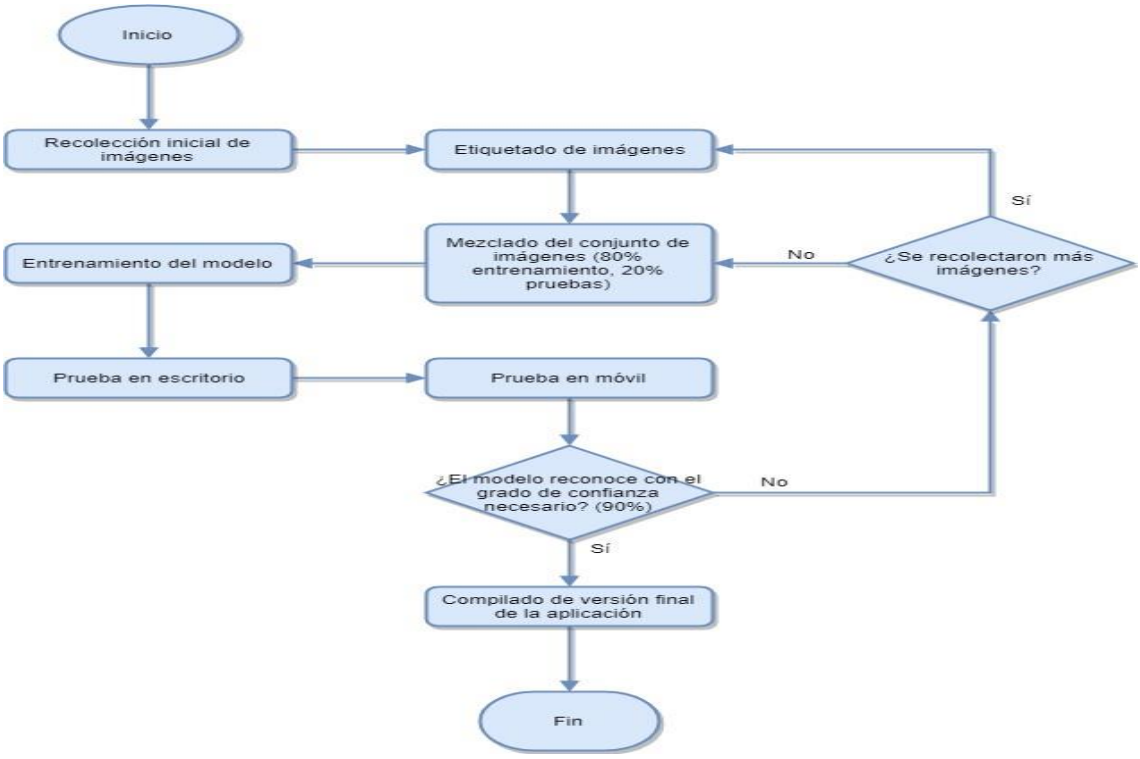
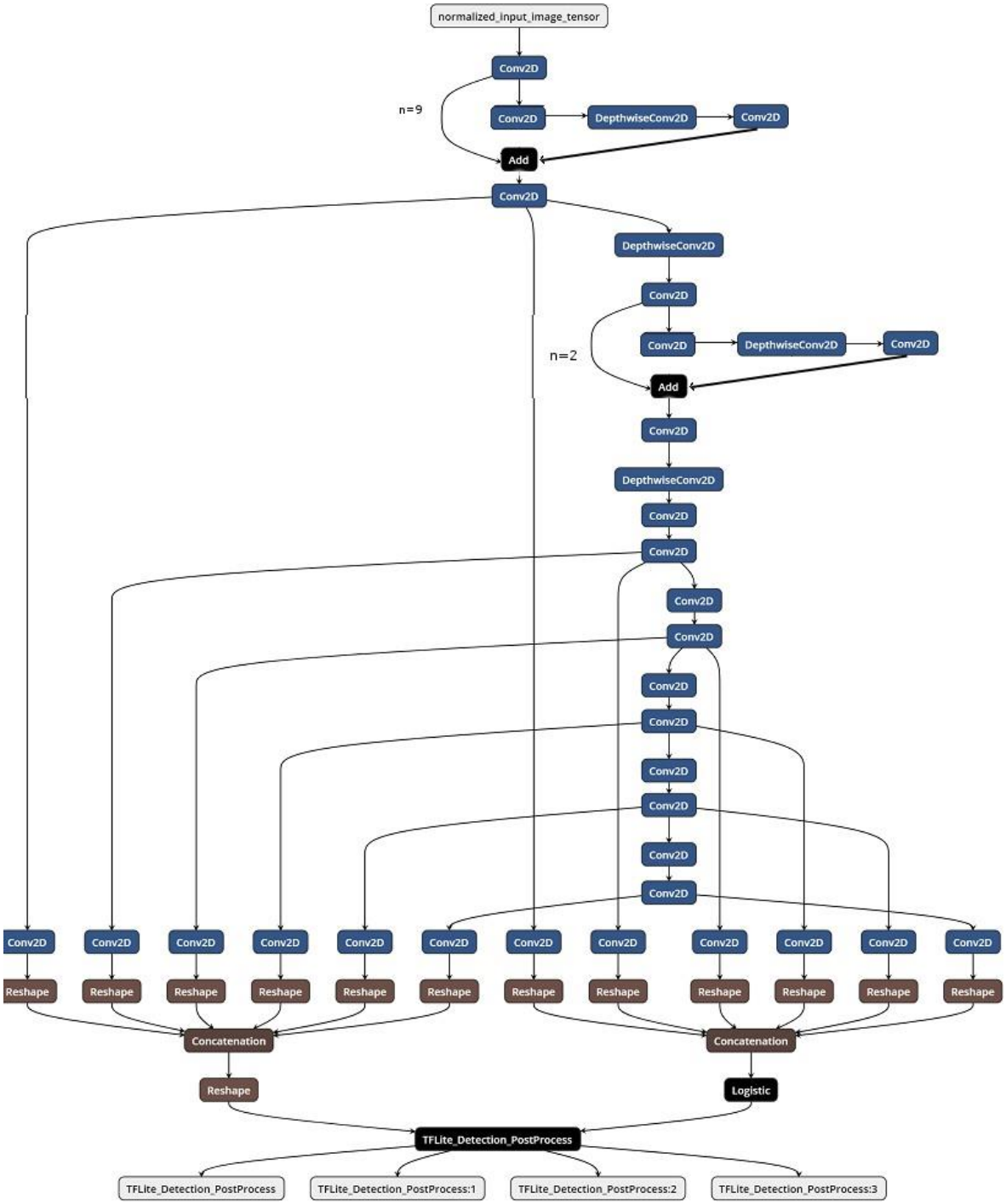


Figura 2

Grafo de inferencia, se aplican varia etapas conversión a 2D y redimensionamiento para aumentar el grado de eficiencia (Imagen Propia).



2.3 Recolección y etiquetado de imágenes

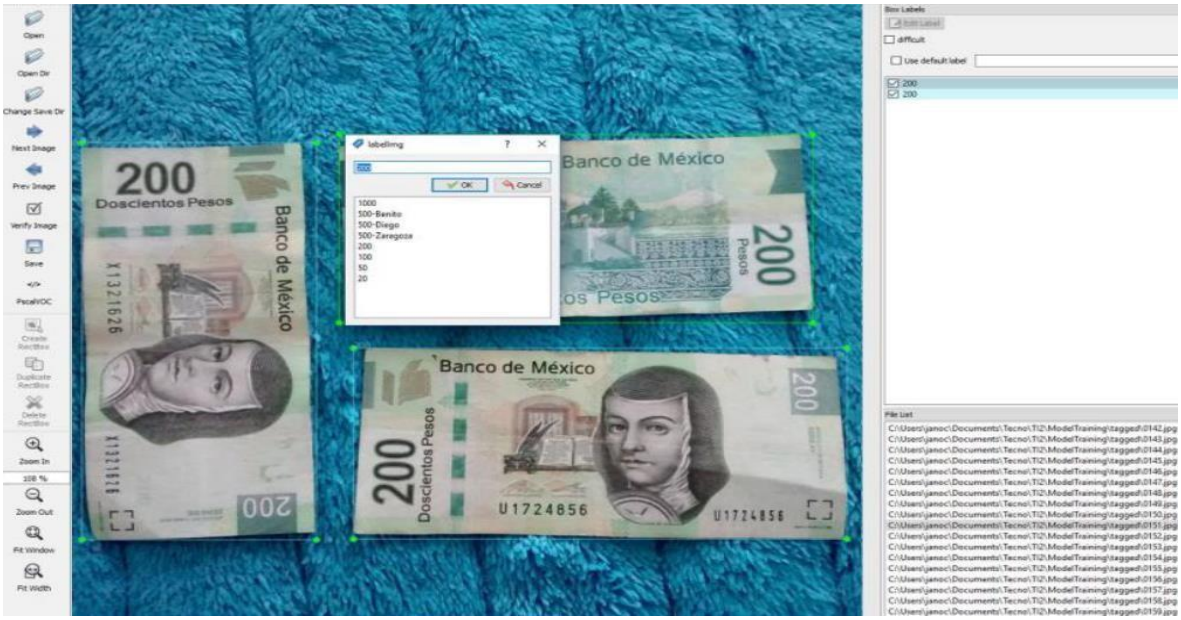
Antes de proceder al entrenamiento es necesario construir un *dataset* o conjunto de datos. Este debe contener una cantidad suficiente de imágenes para cada clase de objeto a reconocer. Es decir, para esta investigación debe contener imágenes de cada billete mexicano propuesto.

Como regla general, se recomienda contar con al menos 100 imágenes por clase para prototipos de detección de objetos. Para esta investigación, a la fecha del último ciclo de desarrollo, se recolectaron 431 imágenes en total incluyendo billetes solos, en conjunto, billetes en diferentes posiciones, truncados, deformados y a diferentes distancias, además de incluir imágenes sin billetes para mejorar la precisión del modelo.

Después de recolectar imágenes, la siguiente tarea es el etiquetado de las mismas. Con ayuda de un software de etiquetado como *LabelImg* se dibujan rectángulos sobre cada billete presente en una imagen y se les asigna una etiqueta correspondiente al nombre de la clase (ver figura 4, etiquetado de las imágenes). Una vez que todas las imágenes se han etiquetado procede con la siguiente etapa.

Figura 4

Configuración de la utilería LabelImg, la cual permite definir una etiqueta descriptiva de acuerdo a la imagen reconocida (imagen Propia).



2.4 Entrenamiento del modelo

Durante esta etapa se alimenta a Tensorflow con el *dataset* etiquetado para que el modelo genere su propio conocimiento del problema a través de un proceso de aprendizaje automático. Comúnmente, dicho proceso es conocido como entrenamiento.

Antes de iniciar el entrenamiento se deben cumplir dos prerequisitos. Primero, se debe dividir el *dataset* en dos subconjuntos: de entrenamiento y de pruebas. El subconjunto de entrenamiento es utilizado por Tensorflow para el proceso de aprendizaje automático y es recomendable que contenga el 75% del total de las imágenes. El 25% de imágenes restantes forma el subconjunto de pruebas, que es utilizado para reforzar el aprendizaje mediante la ejecución de la detección de objetos y comparación de los resultados con la salida esperada (etiquetas provistas por el usuario). El *dataset* de la presente investigación se dividió en subconjuntos de 323 y 108 imágenes respectivamente.

El segundo prerrequisito es convertir el *dataset* a un formato aceptado por Tensorflow. Previo a la popularización de este tipo de proyectos era necesario programar diferentes herramientas y *scripts* para dicha conversión. Sin embargo, algunos desarrolladores han liberado utilidades de código abierto para realizar la conversión, las cuales se han tomado y modificado ligeramente de acuerdo a las necesidades del presente proyecto.

Una vez cumplidos estos prerrequisitos, se procede a ejecutar el entrenamiento. De acuerdo a un número de iteraciones y un tamaño de lote establecido, Tensorflow toma un lote de imágenes y ejecuta los procesos necesarios para el aprendizaje automático hasta llegar al límite de iteraciones. Cabe destacar que el tamaño de lote y el número de iteraciones define la cantidad de veces que se ejecuta un pase sobre el *dataset* en su totalidad (un *epoch*) durante el proceso de aprendizaje. Para la presente investigación se estableció un tamaño de lote de 8 imágenes y un límite de 100,000 iteraciones, para un total de 2,476.78 *epochs*.

Es necesario considerar que entrenar un modelo de red neuronal requiere ciertos recursos, y no se trata de una tarea fácil, motivo por el cual Tensorflow ofrece versiones para trabajar con *CPU* (procesador) y *GPU* (unidad de procesamiento gráfico, por sus siglas en inglés). Generalmente, este tipo de tareas se ven beneficiadas del uso de una o múltiples *GPUs* debido a la naturaleza del problema (operaciones simples con un alto grado de paralelismo). Dicha consideración se tomó en cuenta para la presente investigación, y se seleccionó una versión de Tensorflow compatible con *GPUs*. La selección de hardware y software del equipo de cómputo usado para el entrenamiento fue la siguiente:

- CPU: AMD Ryzen 5 1600 (6 núcleos, 12 hilos) @ 3.825 GHz (*overclock*)
- GPU: NVIDIA GeForce GTX 1050 Ti (768 núcleos CUDA, 4 GB GDDR5, driver 416.34) @ 1,949 MHz / 4,503 MHz (*overclock*)
- Memoria: 2x8 GB DDR4 (2,400 MHz) en configuración *dual-channel*
- Sistema operativo: Windows 10 1803 (64 bits)
- Python v3.6.6
- Tensorflow: tensorflow-gpu v1.11.0 (CUDA Toolkit v9.0.17, cuDNN SDK 9.0 v7.3.1.20)

Parámetros como el tamaño de lote y las dimensiones de entrada del modelo tienen un impacto directo en recursos de memoria y en la rapidez del proceso, por lo que fueron ajustados de acuerdo al hardware sobre el que se realizó el entrenamiento.

Una ventaja de haber seleccionado Tensorflow para el desarrollo de la presente investigación es que durante el entrenamiento se generan *checkpoints* o puntos de control cada cierto número de iteraciones. Estos *checkpoints* pueden ser usados como si se tratara del modelo final, lo que propicia el desarrollo por prototipos y fue de utilidad para el desarrollo en paralelo de la siguientes etapas.

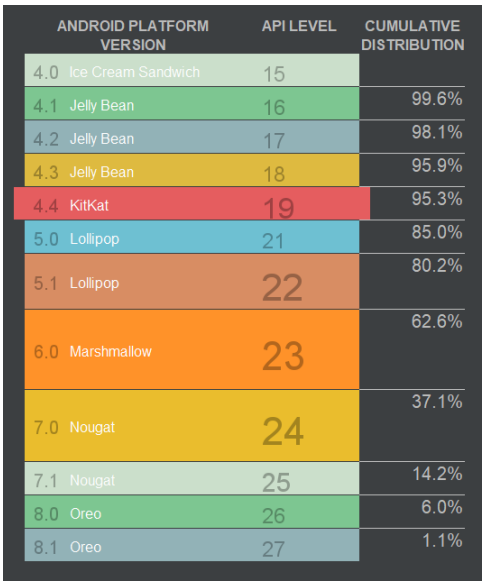
Terminado el entrenamiento o para realizar pruebas con un checkpoint es necesaria una conversión del modelo, puesto que Tensorflow y Tensorflow Lite trabajan con formatos diferentes. El modelo de Tensorflow contiene operaciones para el entrenamiento y otras etapas de pre y postprocesamiento de los datos, las cuales no son necesarias en un modelo de Tensorflow Lite y son removidas. Idealmente, el modelo no debe de verse afectado por la conversión.

2.5 Desarrollo de la aplicación

La aplicación fue desarrollada para el sistema operativo Android. Como requisito mínimo se estableció la versión 4.4 (Kitkat) del mismo para cubrir el 95.3% de los dispositivos activos al mes de noviembre de 2018 (ver figura 5, distribución acumulada de versiones de Android) de acuerdo con la información proporcionada por Android Studio en su versión 3.2.1, entorno de desarrollo utilizado para la presente investigación. La única dependencia de la aplicación es Tensorflow Lite en su versión 1.10.0.

Figura 5

Distribución acumulada de las versiones de Android de acuerdo a Android Studio (imagen Propia).



La aplicación es muy básica, con 2 vistas o actividades diferentes. La primera permite seleccionar entre tomar una fotografía con una aplicación de cámara ya existente en el dispositivo móvil o seleccionar una imagen de la galería, además de permitir ajustar el grado de confianza mínimo aceptable para las detecciones (ver figura 6, vista principal de la aplicación).

La segunda vista presenta los resultados de la inferencia con sus detecciones correspondientes en forma de rectángulos coloreados además de información relevante sobre la detección actual en forma de texto (ver figura 7, vista de resultados de la aplicación).

2.6 Pruebas y depuración

Para verificar el modelo se realizaron pruebas tanto en computadora de escritorio como en dispositivo móvil. Para las pruebas se usó el mismo equipo de cómputo en el que se realizó el entrenamiento. Los dispositivos móviles usados fueron: un Motorola XT1032 (Android 7.1.1) y una tableta genérica (Android 4.4). Las imágenes usadas para las pruebas fueron las mismas existentes en el *dataset*.

Una vez liberada la versión final se procedió a realizar el experimento en la aplicación, el cual consistió en probar con cinco imágenes usadas durante el entrenamiento del modelo, así como con cinco fotos tomadas en el momento a través de la aplicación para obtener los valores de certeza.

Figura 6

Vista Principal de la aplicación, con tres áreas, dos para la imagen y la final para el resultado (imagen propia).

Figura 7

Diferentes billetes con deficiente reconocimiento, durante el entrenamiento (imagen propia).

Figura 8

Billetes de 500 identificados como de 200, durante el entrenamiento (imagen propia).

2.7 Análisis de resultados y conclusiones

Después de haber realizado las pruebas correspondientes, los resultados son guardados como evidencia para su posterior análisis. Se recolectaron principalmente los resultados de la detección en computadora de escritorio, se probaron diferentes *checkpoints* del modelo hasta terminar el entrenamiento. Una vez terminado el entrenamiento, se recolectaron los resultados de la aplicación móvil. Posterior a la recolección descrita, se calcularon índices de detección para los modelos de Tensorflow (escritorio) y Tensorflow Lite (móvil) se aplicó t-student para los resultados de la app y del emulador, con la finalidad de establecer si los resultados son homogéneos y con ello validar la hipótesis planteada, la formula utilizado fue la mostrada en la figura 9, aunque se obtuvieron 2 columnas con resultados no son dependientes entre ellas ya que es la misma aplicación ejecutándose en 2 dispositivos diferentes. Gorgas, Cardiel y Zamorano (2011) establecen que la distribución t de Student es sumamente importante para la estimación y el contraste de hipótesis sobre la media de una población.

Figura 9

Fórmula de t de student para muestras pequeñas menores de 60 (Gorgas, Cardiel y Zamorano 2011).

$$\text{Prueba T para Muestra Única} = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

Nota. La fórmula involucra la media de la muestra menos µ (valor buscado), dividido entre la proporción de la desviación estándar y la raíz del tamaño de la muestra.

3. Resultados

Las imágenes de muestra, que se pueden observar en las figuras 7, 8, 9 y 10, denotan varios elementos visuales que nos permiten interpretar parte de los resultados. Es importante resaltar que las figuras 7 y 8 pertenecen a pruebas realizadas con la aplicación móvil, mientras que las figuras 9 y 10 fueron tomadas de las pruebas realizadas con la versión de escritorio del modelo una vez terminado el entrenamiento del mismo, por lo que varía lo que se puede apreciar en cada una. Para marcar el área donde se detecta la posibilidad de encontrar un billete se dibuja un rectángulo cuyo color varía según la denominación del billete encontrado. En la versión móvil se encuentra una sección de estadísticas donde se puede apreciar el grado de confianza mínimo con el que se detectaran los billetes; carga, detección y dibujo son los respectivos tiempos que le tomo a la aplicaron realizar cada uno de esos procesos. Finalmente en la sección de detecciones, se indica por cada uno de los billetes detectados, con el mismo color usado para dibujar el rectángulo con el que se marcó la imagen, cual fue la denominación que se detectó y con qué grado de confianza. Para la versión de escritorio, las estadísticas son mostradas en una ventana independiente a la imagen y tanto la denominación como el grado de confianza son puestos dentro del mismo marco dibujado en la imagen.

Como se puede observar en las figuras 7 y 8, el modelo detecta un billete de doscientos pesos del tamaño de toda la imagen en lugar del resultado esperado (ver figuras 9 y 10), situación que se presentó a lo largo de las 10 pruebas individuales realizadas. En algunos casos, como se puede observar en la figura 7, es detectado alguno de los billetes de manera correcta, con un grado de confianza bastante bajo, como consecuencia, si éste es aumentado a un 90% (configuración óptima) la aplicación no sería capaz de detecciones aceptables.

La aplicación móvil alimentó las imágenes exitosamente al modelo de Tensorflow Lite. Los resultados de las 10 pruebas se muestran en la tabla 1, comparados con los obtenidos de las mismas imágenes en la versión de escritorio. En la tabla se pueden observar tres cosas importantes. Primeramente, el tiempo de procesamiento es en promedio dieciséis veces mayor para la aplicación móvil. La segunda situación observable es que para la aplicación de escritorio la confiabilidad promedio es de 95%, la versión móvil apenas alcanza un 32% lo cual es considerablemente bajo y nada satisfactorio. Por último se denota que la cantidad de billetes detectados no pasa de dos en la versión móvil, cuando se deberían de detectar hasta cuatro en algunas situaciones, como es el caso de las pruebas 9 y 10. Cabe destacar que la desviación estándar nos indica que los datos estadísticos de la versión móvil no son estables.

Con los resultados obtenidos se confirma que la hipótesis es errónea, puesto que no se logró detectar la denominación de los billetes mexicanos a través de una aplicación móvil con una precisión del 90%, y se declara el experimento como fallido.

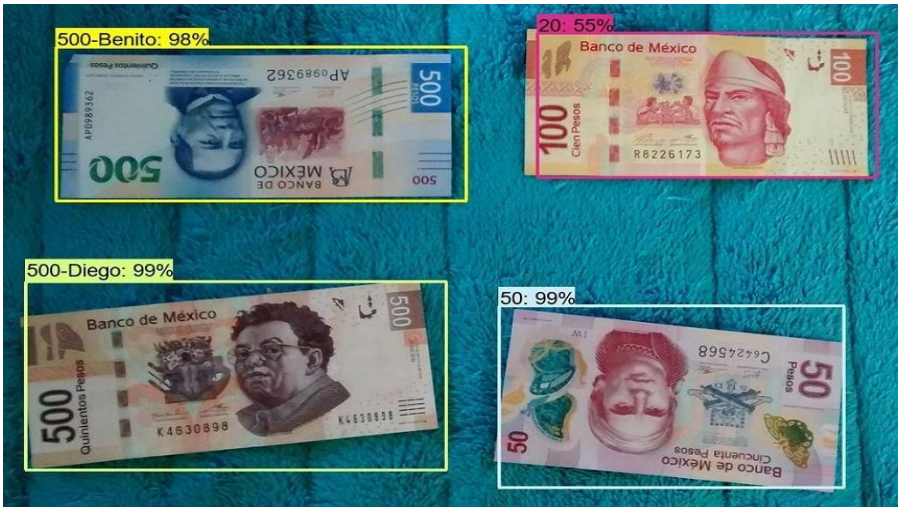
Figura 9

Resultado obtenido desde el emulador de Android de la PC, con 2 billetes al mismo tiempo (imagen propia).

Nota. Se puede observar que se colocaron los dos billetes en diferente alineación, y que se etiquetaron adecuadamente con el nombre del personaje y su nombre.

Figura 10

Resultado obtenido desde el emulador de Android de la PC, con 4 billetes al mismo tiempo.



Nota. Se puede observar que se alinearon los billetes un tanto cuanto aleatorio, la aplicación reconoció bien 3 de los 4 billetes*, en el caso del billete de 50 y 100 no se definió un nombre por eso solo aparece el número (imagen propia).*

Tabla 1

Comparativa de Resultados del experimento, entre el dispositivo móvil y la emulación de Android en una PC

Prueba	No. detecciones		Tiempo lectura (ms)		Tiempo Detección (ms)		Total del Proceso (ms)		Grado de Confianza	
	Móvil	PC	Móvil	PC	Móvil	PC	Móvil	PC	Móvil	PC
1	1	2	464	68	1553	17	2295	132	39%	99%
2	1	1	777	51	892	17	2496	103	37%	99%
3	1	1	275	67	1321	16	1749	119	38%	99%
4	1	1	167	60	951	17	1051	111	32%	99%
5	1	1	634	30	1004	18	1832	81	35%	99%
6	2	2	815	47	988	18	2485	110	15%	91%
7	1	3	819	47	1045	17	2544	122	37%	92%
8	1	2	818	35	798	18	2289	99	34%	99%
9	2	4	721	76	1559	17	2600	165	18%	87%
10	1	4	145	82	870	19	1057	173	35%	95%
Promedio	1.20	2.10	563.50	56.30	1098.10	17.40	2039.80	121.50	32.00%	95.90%
Desv. Estándar	0.42	1.20	277.72	17.19	278.70	0.84	593.76	28.66	8.45%	4.43%

Nota. Se realizaron 10 muestreos, incorporando los tiempos que conllevaba cada prueba, tiempo de lectura, detección y proceso, en el renglón promedio se puede observar que siempre el móvil tardaba más o era menos eficiente.

4. Discusión

Haciendo uso de Tensorflow Lite en un entorno móvil, la detección no presentó resultados satisfactorios. Sin embargo, según las pruebas hechas en la computadora de escritorio durante el entrenamiento del modelo, con la versión completa de Tensorflow, el modelo fue más que exitoso. De una muestra de cuarenta imágenes tomadas aleatoriamente del *dataset* y alimentadas al modelo sólo un billete no fue reconocido, mientras que los demás fueron detectados exitosamente con un grado de confianza mayor al 90%.

En la tabla se observa que el porcentaje de certeza está en 32%, la ineficacia de la aplicación puede deberse a incompatibilidades no documentadas entre Tensorflow y Tensorflow Lite ya que, como se mencionó en el presente artículo, es necesaria una conversión de formato entre modelos. Cabe destacar que la herramienta proporcionada para una etapa de la conversión (Tensorflow Lite Converter, o TOCO) es incompatible con Windows, y fue necesario utilizar una distribución de Linux (Ubuntu 16.04.5 LTS en Windows, con WSL). La enorme diferencia en capacidad de procesamiento entre un dispositivo móvil y una PC puede haber influido negativamente en los resultados obtenidos. Otra causa probable es la diferencia en el número de versiones, siendo 1.11.0 para Tensorflow y 1.10.0 para Tensorflow Lite.

Para validar los datos se aplicó la fórmula de la figura 9. Obteniendo los valores de t =-0.00217161112836 para los datos de la app y de t=6.19836460565E-5 para el emulador de android en la computadora, para ambos casos el valor de la tabla de student es de 1.8331. En el caso de la app se establece que t<1.8331 por lo tanto la hipótesis se rechaza, mientras que para el emulador en la PC t>1.8331 la hipótesis podría ser aceptada, sin embargo la investigación estaba orienta hacia un dispositivo móvil.

Una posible alternativa al problema presentado es la implementación de Tensorflow Mobile, el predecesor de Tensorflow Lite, puesto que no se requiere conversión alguna en el proceso de desarrollo del modelo. Sin embargo, se trata de una herramienta cuyo soporte está en desarrollo, además de no contar con el mismo grado de optimización que Tensorflow Lite.

Dicho esto, es muy probable que en versiones futuras de Tensorflow Lite el modelo pueda otorgar resultados satisfactorios. Mientras tanto, se logró un excelente progreso en el modelo de Tensorflow, lo que puede servir para otro tipo de aplicaciones en entornos no móviles que requieran la detección confiable de billetes mexicanos y sus correspondientes denominaciones.

Otra posible explicación del por qué la app no dio buenos resultados con respecto del emulador en la Pc puede deberse a las capacidades de los microprocesadores, en donde la capacidad de una PC supera a un móvil, y esto daría pie de entrada a otro tipo de pruebas para probar esta nueva hipótesis.

5. Referencias

Althafiri, E.& Sarfraz, M. & Alfarras, M. (2012) Bahraini paper currency recognition. Journal of Advanced Computer Science and Technology Research, 2012.

Debnath, K. K.& Ahdikary, J. K. & Shahjahan, M. (2009). *A currency recognition system using negatively correlated neural network ensemble*. International Conference on Computers and Information Technology.

Gorgas J, & Cardiel N, & Zamorano J. (2011). *Estadística Básica para estudiantes de ciencias*, Universidad Complutense, España

Gayon, C. (2017). *Desarrollo de una aplicación para reconocimiento de billetes por medio de procesamiento de imágenes para personas con diversidad visual basada en tecnología android*. Septiembre 27, 2018, de Universidad libre sede bosque popular facultad de ingeniería de sistemas Bogotá. Sitio web: https://goo.gl/8SA1rQ

Howard, A. G. & Zhu, M. & Chen, B. & Kalenichenko, D. & Wang, W. & Weyand, T. & Andreetto, M. & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* Noviembre de 2018, de arXiv Sitio web: https://arxiv.org/abs/1704.04861

Kingma, D. P. & Ba, J. (2014). *Adam: A Method for Stochastic Optimization.* Noviembre de 2018, de arXiv Sitio web: https://arxiv.org/abs/1412.6980

Lin, T. & Maire, M. & Belongie, S. & Bourdev, L. & Girshick, R. & Hays, J. & Perona, P. & Ramanan, D. & Zitnick, C. L. & Dollár, P. (2014). Microsoft COCO: Common Objects in Context. Noviembre de 2018, de arXiv Sitio web: https://arxiv.org/abs/1405.0312

Liu, W. &, Anguelov, D. & Erhan, D. & Szegedy, C. & Reed, S. & Fu, C. & Berg, A. C. (2015). SSD: Single Shot MultiBox Detector. Noviembre de 2018, de arXiv Sitio web: https://arxiv.org/abs/1512.02325

Moretti, I, & Amado, J, & Puntillo, D, & Caniglia, C. (2015). *Software libre de Reconocimiento de billetes para personas en situación de discapacidad visual*. Septiembre 27, 2018, de Instituto Nacional de Tecnología Industrial, Centro regional Córdoba, Córdoba, Argentina Sitio web: https://goo.gl/8NsH4S

Singh, S. & Choudhury, S. & Vishal, K.& Jawahar, C.V. (2014). *Currency Recognition on Mobile Phones*, 22nd International Conference on Pattern Recognition,

Zaforas, M. (2017). *Inteligencia Artificial como servicio: reconocimiento de imágenes*. Septiembre 27, 2018, de Paradigma Sitio web: https://goo.gl/UT5FJH

Esta obra está bajo una licencia de Creative Commons Atribución-NoComercial-Compartirlgual 4.0 Internacional.