

Implementing YOLOv5 in an Android Interface for Digit Detection in Water Meters.

Implementando YOLOv5 en una interfaz de Android para la detección de dígitos en medidores de agua

Chaparro J.¹, Escápita O.², Chacón L.³, Hernández D.⁴, Sánchez S.⁵, Muñoz D.⁶, Peinado A.⁷, and Ramírez G.⁸

¹⁻⁸ Universidad Autónoma de Chihuahua, Facultad de Ingeniería

ABSTRACT

This paper details the successful implementation of a mobile application for water meter measurement detection, leveraging the power of YOLOv5 in PyTorch. The project went through an initial phase of exploring options and models, including TensorFlow and other object detection models. However, the transition to YOLOv5 in PyTorch marked a turning point, providing more effective model conversion tools. This implementation will allow users to keep track of their water meter measurements efficiently, improving efficiency and accuracy in water resource management.

RESUMEN

Este informe técnico detalla la implementación exitosa de una aplicación móvil para la detección de lecturas en medidores de agua, aprovechando la potencia de YOLOv5 en PyTorch. El proyecto pasó por una fase inicial de exploración de opciones y modelos, incluyendo TensorFlow y otros modelos de aprendizaje profundo de detección de objetos. Sin embargo, la transición a YOLOv5 en PyTorch marcó un punto de quiebre, ya que proporcionó herramientas de conversión del modelo más efectivas. Esta implementación permitirá a los usuarios llevar un registro eficiente de las lecturas de sus medidores de agua, mejorando la eficiencia y la precisión en la gestión de los recursos hídricos.

1 INTRODUCCIÓN

La automatización de tareas mediante modelos de Machine Learning se ha convertido en una tendencia en diversas áreas de trabajo, incluyendo la industria de servicios públicos. Los modelos de Machine Learning permiten a las empresas automatizar tareas repetitivas y de alta demanda, lo que a su vez aumenta la eficiencia, reduce costos y mejora la calidad del servicio. Además, la capacidad de estos modelos para aprender de los datos y adaptarse a nuevos patrones hace que sean particularmente útiles para tareas complejas que requieren de habilidades cognitivas avanzadas. En la industria de servicios públicos, la adopción de modelos de Machine Learning está permitiendo la automatización de tareas como la atención al público, la implementación de formularios, el pago de impuestos, el diagnóstico de enfermedades, la toma de decisiones, la protección de datos, etc. (Indecopi, 2019).

Los modelos de detección representan un avance significativo en esta área para identificar y clasificar objetos, patrones o eventos en datos complejos y variados, como imágenes, texto o señales. Estos modelos se entrenan utilizando algoritmos que les permiten aprender automáticamente a reconocer características relevantes en los datos y tomar decisiones de detección. Uno de los modelos recientes más exitosos es YOLOv5.

YOLOv5, acrónimo de "You Only Look Once" versión 5, es una poderosa arquitectura de detección de objetos en tiempo real basada en redes neuronales convolucionales. Esta versión, que sigue la exitosa serie YOLO, se destaca por su equilibrio entre precisión y velocidad, lo que la hace especialmente adecuada para aplicaciones que requieren detección de objetos en movimiento rápido. YOLOv5 ha ganado popularidad en campos como la visión por computadora, la robótica y la seguridad, gracias a su capacidad para detectar múltiples objetos en una sola pasada de la imagen o el flujo de video, lo que la convierte en una herramienta valiosa para tareas que van desde la detección de peatones en vehículos autónomos hasta la vigilancia en tiempo real. Su versatilidad y eficiencia la convierten en una opción atractiva para una variedad de aplicaciones de detección de objetos (ultralytics, 2023). El desarrollo de una aplicación móvil para eficientar la lectura de medidores de agua por el sector público tiene múltiples beneficios que ya se han explorado en otras investigaciones con fines similares a esta (Chisag, Pasto, & Garofalo, 2022, Guato Chifla, 2015, Saravia Valle, Ruiz Rivera, & Calmet Agnelli, 2013). En primer lugar, permitiría una medición más precisa y eficiente del consumo de agua, lo que a su vez podría resultar en una distribución más equitativa del recurso.

En este documento se describen los detalles de un modelo de detección de objetos en un dispositivo *Android* para la detección de los dígitos de los medidores de agua.

2 MARCO TEÓRICO DE REFERENCIA

La detección de objetos es una rama de la visión artificial que se centra en identificar la presencia y localización de objetos en imágenes. Es utilizada en una amplia variedad de aplicaciones, desde sistemas de vigilancia, análisis de imágenes médicas hasta la conducción autónoma.

Esta técnica implica la utilización de algoritmos y metodologías de aprendizaje automático para reconocer patrones en las imágenes con los que se puede detectar la presencia de los objetos deseados. En los últimos años, se han propuesto varios enfoques y modelos distintos, entre los que destacan *Faster R-CNN*, *YOLO* y *SSD*.

2.1 ¿Cómo funciona la detección de objetos?

La detección de objetos parte de un modelo con capas de convolución (CNN), la cual, dada una imagen, puede obtener una clasificación de la misma. Esto mediante un algoritmo, en el que se buscan regiones que pueden ser de interés, éstas regiones deben cumplir ciertas características, como tonos y bordes, para introducirlas a la CNN y determinar su clasificación. Luego se determinan cuáles de las regiones contiene al objeto buscado y poder indicar su posición.

Esto implica que las capas CNN detecten una cantidad específica de objetos, no pudiendo detectar objetos que antes no hubiera aprendido, o si éstos tienen ahora un tamaño muy pequeño, o el objeto a identificar se encuentra oculto o si la imagen presenta algún problema de sombras. Para entrenar un modelo de detección de manera supervisada se debe indicar la clase del objeto (qué es) y además la posición dentro de la imagen definida por coordenadas x , y , el ancho y alto del objeto.

2.2 Modelos de detección de objetos

Para la detección de objetos existen diversas opciones de modelos, algunos de éstos son:

- *Single Shot Detector* (SSD): Tiene una estructura piramidal en su CNN en la que las capas van disminuyendo gradualmente. Esto le permite poder detectar objetos grandes y pequeños. Cuenta con “anclas” de distintas proporciones que se van escalando a medida que se desciende por la pirámide.
- *Faster R-CNN*: Debe tener un único archivo comprimido y contener la estructura de directorios y los archivos correctos. Comparte características de convolución de la imagen con la red de detección.
- *RetinaNet*: Se basa en una estructura de CNN piramidal mejorada para reconocer objetos de diversos tamaños en una sola pasada. Este modelo utilizó una función de pérdida llamada *Focal Loss*.
- *YOLO*: Hace uso de una única red neuronal convolucional para detectar objetos a partir de una imagen en

una sola evaluación.

2.3 Detección de objetos con YOLO

YOLO (*You Only Look Once*), es un modelo de detección de objetos altamente eficiente y preciso que ha demostrado grandes capacidades para detectar objetos en tiempo real. YOLO consta de capas de convolución que predicen simultáneamente múltiples cuadros delimitadores y probabilidades de clase para esos cuadros.

YOLO parte de un concepto algo diferente al del resto de modelos, hace sus predicciones basadas en la imagen completa en lugar de trabajar por regiones de interés. Durante su entrenamiento, divide las imágenes en una cuadrícula y se encarga de aprender las probabilidades de encontrar los objetos en las distintas celdas de ésta.

3 METODOLOGÍA

La aplicación de detección de objetos con YOLOv5 utiliza el modelo escrito en *Pytorch* y la aplicación de ejemplo de *Android*. Para la implementación se utilizó Anaconda y *CUDA* en Linux con *WSL*. A continuación se presenta la instalación de los recursos necesarios y luego la implementación de *Pytorch* y la aplicación para *Android*.

Además, para enriquecer la metodología y aprovechar las ventajas de las herramientas colaborativas en línea, se decidió implementar un modelo adicional utilizando la biblioteca *TensorFlow* en *Google Colab*. Esto permitió explorar más a fondo las capacidades de detección de objetos y optimizar el enfoque para satisfacer las necesidades específicas del proyecto. A través de esta implementación en *Google Colab*, se pudieron realizar pruebas rigurosas y ajustes continuos para garantizar la precisión y eficacia del modelo antes de integrarlo en la aplicación de detección de medidores de agua para *Android*.

3.1 Selección de dataset

Durante el proceso de experimentación con los modelos de detección de medidores de agua, se evalaron varios conjuntos de datos, apoyándose de la herramienta *Roboflow*, para encontrar la opción más adecuada. Se hizo una evaluación previa de los *dataset* para analizar como estaban conformados, además de pruebas de desempeño al entrenar el modelo, se probaron los siguientes: *WaterMeters Computer Vision Project* en su versión 2 (Roboflow, 2022), luego el *dataset pascal to yolo classification digits Computer Vision Project* en su versión 1 (Roboflow, 2021), y por último, *OCR 2* en su versión 4 (Roboflow, 2022). Este último, presentaba una estructura más coherente y etiquetas más precisas de los medidores de agua, lo que ofrecía mejores resultados en entornos más realistas. Por lo tanto, finalmente, se optó por utilizar el conjunto de datos *OCR 2* para el proyecto.

3.2 Implementación de Tensorflow 2 en Google Colab

Se utilizó un código en *Google Colab* utilizando *Tensorflow 2*, esto debido a errores con librerías más actualizadas para el entrenamiento de modelos de detección. El código se obtuvo del tutorial *Custom Real-Time Object Detection*, que *Tensorflow* ofrece. El código se adaptó para usar el *dataset* seleccionado. Los resultados del entrenamiento se muestran en la sección de resultados.

Sin embargo, debido a problemas en la implementación de otros modelos distintos al *MobileNet*, se decidió finalmente mudar el proyecto a *Pytorch*. Para la información de los experimentos con *Tensorflow* consultar los anexos.

3.3 Instalación de Anaconda en Linux con WSL

Una vez descartada la librería de *Tensorflow* se tomó la decisión de utilizar *Pytorch*, esto mediante el uso de la herramienta WSL (*Windows Subsystem for Linux*). La cual permite utilizar una distribución de Linux de manera nativa en Windows, esto para utilizar las librerías de *Pytorch* sin ningún problema, en adición a la librería CUDA de Nvidia para utilizar una tarjeta gráfica dedicada para acelerar el proceso de entrenamiento. La instalación de WSL es muy sencilla, se debe ejecutar este comando en la terminal de Powershell de Windows:

```
wsl --install
```

Esto instalará la distribución Ubuntu por defecto. Una vez instalado Ubuntu con WSL, se realizó la instalación de Anaconda como sigue:

1. Vaya a <https://repo.continuum.io/archive> para encontrar la lista de lanzamientos de Anaconda.
2. Seleccione el lanzamiento que desee. Para una computadora de 64 bits, elegir la última versión que termina en x86_64.sh. Si tuviera una computadora de 32 bits, seleccionaría la versión x86.sh. Si accidentalmente se instala el incorrecto, se recibirá una advertencia en la terminal. Se eligió Anaconda3-5.2.0-Linux-x86_64.sh.
3. Desde la terminal, ejecute el siguiente comando:

```
wget https://repo.continuum.io/archive/[SU VERSIÓN]
```

Ejemplo:

```
$ wget https://repo.continuum.io/archive/Anaconda3-5.2.0-Linux-x86_64.sh
```

4. Ejecute el script de instalación:

```
$ bash Anaconda[SU VERSIÓN].sh ($ bash Anaconda3-5.2.0-Linux-x86_64.sh)
```

5. Lea el acuerdo de licencia y siga las instrucciones para aceptar. Cuando le pregunte si desea que el instalador lo anteponga a la ruta, responda que sí.
6. Opcionalmente, instale VS Code cuando se le solicite.
7. Cierre la terminal y vuelva a abrirla para recargar .bash configs.
8. Para probar que funcionó, utilice el comando:

```
$ which python
```

Debería imprimir una ruta que contenga anaconda. Si no tiene anaconda en la dirección, realiza el siguiente paso. De lo contrario, vaya al paso 11.

9. Agregue manualmente la carpeta bin Anaconda a su ruta. Para hacer esto, agregue

```
export PATH=/home/kauff/anaconda3/bin:$PATH
```

al final del archivo `~/.bashrc`.

10. Para abrir jupyter, escriba

```
$ jupyter notebook --no-browser
```

La bandera sin navegador seguirá ejecutando Jupyter en el puerto 8888, pero no lo abrirá automáticamente. Es necesario ya que no tiene un navegador (probablemente) en su subsistema. En la terminal, habrá un enlace para pegar el navegador.

3.4 Instalación de CUDA en WSL

Los siguientes comandos instalarán la versión 11.6 del kit de herramientas CUDA específico de WSL en la arquitectura Ubuntu 22.04 AMD64. Se debe tener en cuenta que las versiones anteriores de CUDA (<=10) no son compatibles con WSL 2. Además, al instalar los paquetes del kit de herramientas CUDA directamente desde el repositorio de Ubuntu (“cuda”, “cuda-11-0” o “cuda -drivers”) se intentará instalar el controlador de gráficos

NVIDIA de Linux, que no es lo que desea en WSL 2.

Primero hay que remover la llave GPG:

```
$ sudo apt-key del 7fa2af80
```

Después se debe configurar el paquete apropiado para Ubuntu WSL con los siguientes comandos:

1. \$ wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/cuda-wsl-ubuntu.pin
2. \$ sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600
3. \$ sudo apt-key adv --fetch-keys
https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/3bf863cc.pub
4. \$ sudo add-apt-repository 'deb
https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/'
5. \$ sudo apt-get update
6. \$ sudo apt-get -y install cuda

Una vez completado, debería de mostrar una serie de salidas con terminación "done".

3.5 Instalación de Pytorch con CUDA en WSL

Para Windows, utilizando pip de Python y CUDA 11.8, la instalación de Pytorch se ejecuta con este comando:

```
$ pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

3.6 Transfer Learning con YOLOv5

Para el entrenamiento del modelo YOLOv5 con un *dataset* personalizado, se procede de manera sencilla, como se menciona en el repositorio ejemplo de PyTorch [android-demo-app/ObjectDetection](#) (PyTorch, 2021). Esto se realiza con un script que incluye el repositorio de [Ultralytics YOLOv5](#) (Ultralytics, 2023). Se necesita: el *dataset*, en este caso, el de *Roboflow*, el cual puede descargarse ya preparado en formato para YOLOv5; y el código fuente del archivo *train.py*, el cual ya viene preparado para ser utilizado mediante un comando, que requiere el archivo *data.yaml* y el *dataset*.¹

¹La ruta local de los archivos necesarios para entrenamiento quedó como sigue: **/Desktop/medidores/yolov5**

El comando para el entrenamiento quedaría como sigue (todo el comando es en una sola línea):

```
python train.py --img 640 --batch 16 --epochs 300 --data data.yaml  
--weights yolov5s.pt
```

Como se puede observar, el comando requiere una serie de argumentos que dictan la configuración de entrenamiento. Teniendo *-img* para el tamaño de las imágenes de entrada, *-batch* para el número de muestras usadas por época, *-epochs* para el número de épocas de entrenamiento, *-data* para el archivo que contiene la forma de las clases del *dataset*, y por último, *-weights* para la versión del modelo base a utilizar.

Una vez entrenado el modelo, es necesario exportarlo para poder utilizarlo en la aplicación de *Android*.

3.7 Exportando el modelo a formato Torchscript

Para exportar el modelo entrenado, se procede de manera similar al entrenamiento. En este caso se requiere la ruta del modelo entrenado y el archivo *export.py*, el cual también se incluye en el repositorio de YOLOv5. Sin embargo, es necesario hacer una pequeña modificación al archivo *export.py*, la cual consta de cambiar una línea que se encuentra desactualizada en el repositorio de YOLOv5, para una exportación correcta al formato requerido, en este caso *torchscript* optimizado. La modificación se hace dentro de la función *export_torchscript()*, donde hay que agregar un *if* y dos líneas de código como sigue:

```
def export_torchscript(...):  
    ...  
    f1 = file.with_suffix('.torchscript.ptl')  
    ...  
    if optimize:  
        optimize_for_mobile(ts).save_for_lite_interpreter(str(f1),  
        _extra_files=extra_files)
```

Una vez realizada la modificación, se exportó con el siguiente comando:

```
python export.py --weights runs/train/exp/weights/best.pt --include torchscript  
--optimize
```

En este caso, el comando requiere de tres parámetros de entrada, `--weights` el cual es el modelo entrenado y `-include` para los formatos de exportación, en este caso `torchscript` y por último, `--optimize` el cual es la bandera para comprimir y cuantizar el modelo entrenado para poder utilizarse en la aplicación

Una vez terminado el proceso, es posible continuar con la implementación de la aplicación de Android.

3.8 Aplicación de Android para YOLOv5

Los prerequisitos para la aplicación de Android son:

- PyTorch 1.10.0 y torchvision 0.11.1 (Opcional).
- Python 3.8 (Opcional).
- Librerías de Android Pytorch:
 - `pytorch_android_lite:1.13.1`
 - `pytorch_android_torchvision_lite:1.13.1`
- Android Studio 4.0.1 or later.

Para la aplicación de Android, solo se clonó el repositorio Pytorch(2021), ya que esta prácticamente lista para utilizarse. Sin embargo, hay una serie de cambios necesarios para hacer que funcione correctamente con el modelo que se entrenó anteriormente. Para esto, es necesario actualizar la versión de PyTorch que utiliza la aplicación para un correcto funcionamiento del modelo optimizado; es necesario ajustar la forma de los datos de salida del modelo así como agregar imágenes de prueba en las que queramos probar el modelo.

Para esto se requiere copiar el modelo exportado a la carpeta `assets` y hacer los siguientes cambios:

3.8.1 Actualizar el nombre del modelo a utilizar

La aplicación tiene 2 modos de uso, por imágenes cargadas o en tiempo real. Hay que actualizar el nombre en 2 partes del código.

- Para actualizar el modelo utilizado en las imágenes cargadas se hace el cambio en el archivo `MainActivity.java`, se cambia el nombre del modelo utilizado al que se haya puesto en el archivo `export.py`, en este caso `best.torchscript.ptl`. El cambio queda como sigue:

```
mModule = LiteModuleLoader.load(
    MainActivity.assetFilePath
    (getApplicationContext()
    , "best.torchscript.ptl"));
```

donde todo queda igual excepto la parte dentro de las comillas que es donde va el nombre del modelo que se creó anteriormente.

- Para actualizar el modelo utilizado en la detección a tiempo real se hace el cambio en el archivo *Object-DetectionActivity.java*, el cambio es exactamente igual al anterior, se cambia de misma manera el objeto mModule como sigue:

```
mModule = LiteModuleLoader.load(
    MainActivity.assetFilePath
    (getApplicationContext()
    , "best.torchscript.ptl"));
```

3.8.2 Actualizar el archivo de descripción de clases del modelo

Para un correcto funcionamiento de la aplicación, es necesario incluir un archivo .txt en el que se listen, separadas en líneas individuales, las clases del modelo entrenado. Por ejemplo, en este caso el archivo, de nombre “medidores.txt” para el *dataset* tiene el siguiente contenido:

0
1
2
3
4
5
6
7
8
9

este archivo debe ir en la carpeta *assets* del proyecto. Ahora si, se cambia en el archivo *MainActivity.java* el nombre del archivo de clases, en este caso queda como sigue:

```
BufferedReader br = new BufferedReader(
    new InputStreamReader(
        getAssets()
            .open("medidores.txt")
    ));

```

3.8.3 Actualizar el numero de columnas de salida del modelo

En el archivo *PrePostProcessor.java* actualizar la cantidad de columnas de salida de los datos, la cual sería el número de clases del *dataset* más 5, que son las 4 posiciones de las esquinas del rectángulo de detección y la clase que se detectó, en este caso sería 15 columnas, 10 clases (los dígitos del 0 al 9) más las 5 mencionadas anteriormente. En este caso, se cambiaría de *mOutputColumn* = 85 a *mOutputColumn* = 15, el cambio queda como sigue:

```
private static int mOutputColumn = 15;
```

3.8.4 Agregar imágenes de prueba

Para agregar imágenes sobre las cuales probar el modelo, es necesario agregar las mismas a la carpeta *assets* del proyecto y agregarlas al arreglo *mTestImages*, el cual contiene los nombres de las imágenes, definido en el archivo *MainActivity.java*. En este caso, se agregaron 3 imágenes (*medidor1.jpg*, *medidor2.jpg* y *medidor3.jpg*), quedando como sigue:

```
private String[] mTestImages = {"medidor1.jpg",
    "medidor2.jpg", "medidor3.jpg",
    "aicook1.jpg", "aicook2.jpg",
    "aicook3.jpg", "test1.png",
    "test2.jpg", "test3.png"};
```

Una vez realizados estos cambios ahora si es posible utilizar la aplicación de *Android* con el modelo personalizado.

3.8.5 Para instalar la aplicación en un dispositivo Android

Si se desea instalar la aplicación en un dispositivo es necesario primero activar el modo desarrollador del mismo para posteriormente activar la depuración mediante *USB* e instalar la aplicación. Para esto es necesario seguir los siguientes pasos (la ubicación de las opciones pueden variar entre dispositivos):

-
- Acceder a la configuración del dispositivo
 - Acceder a la sección "Acerca del dispositivo"
 - Encontrar la sección "Número de compilación"
 - Pulsar en el hasta que salga el aviso "Se ha activado el modo desarrollador"
 - Buscar la nueva sección "Modo desarrollador" (normalmente se encuentra en la página inicial de configuración)
 - Activar la opción "Depuración por *USB*"

Una vez completados estos pasos se puede realizar la instalación conectando el dispositivo mediante un cable de transferencia de datos *USB* y, una vez conectado, aceptando la conexión en el dispositivo móvil. Esto permitirá ver el dispositivo en *Android Studio* en las opciones para realizar la instalación de la aplicación.

3.9 Implementación de lectura y línea de regresión

Se llevó a cabo la implementación de una lectura del modelo, de modo que se pueda leer con mayor facilidad la detección del modelo. Para implementar esta característica, primeramente se calculó el centro de cada uno de los rectángulos de detección, y se ordenaron de manera ascendente con el valor de la posición en el eje horizontal. De este modo, la detección se hará haciendo un “barrido” de izquierda a derecha. Una vez ordenados de esta manera, se concatenan los valores de los rectángulos, obteniendo la lectura final.

Así mismo, al momento de realizar la detección, se puede observar una línea roja que pasa por el medio de los rectángulos de la detección. Esta línea tiene como objetivo la futura implementación de un algoritmo que calcule el error cuadrático de los rectángulos de la detección, de modo que se pueda añadir una capa adicional de filtro para poder descartar aquellos rectángulos que se alejen mucho de la línea del resto de detecciones.

El fundamento de este algoritmo es que los medidores siempre tienen los números alineados uno al lado del otro, por lo que idealmente el algoritmo solo detectaría rectángulos que sigan una misma línea, la cual atravesaría el centro del contador. Por lo tanto, si detectamos un rectángulo que no se encuentra en esta línea, muy posiblemente no es uno de los números del contador, y se puede descartar.

Desafortunadamente, por cuestiones de tiempo, no fue posible implementar el algoritmo para calcular el error y eliminar aquellos rectángulos alejados de la línea.

3.10 Implementación de la carga de un dataset para detección en bulto para el análisis del modelo

El modelo que se ejecuta en la aplicación móvil es una versión comprimida del modelo original. Debido a esto, el modelo produce resultados distintos en la aplicación de celular.

Para poder comparar la eficacia de ambas versiones del modelo, se busca implementar una manera de cargar los datos de un dataset para que la aplicación pueda ejecutar el modelo y guardar los resultados en el dispositivo para poder realizar la comparación.

Actualmente solo se ha implementado la carga de las imágenes y las etiquetas en memoria, todavía es necesario implementar la ejecución del modelo con las imágenes del dataset, la comparativa y el guardado de los datos.

4 RESULTADOS

4.1 Evaluación de Métricas

Para la evaaución de un modelo de detección, las métricas proporcionan una base objetiva para analizar y cuantificar su eficacia. Al medir parámetros como la precisión, el *recall*, el índice F1 y la matriz de confusión, se obtiene una comprensión más profunda de cómo cada modelo se comporta en diferentes contextos y escenarios. Estas métricas permiten una evaluación imparcial y rigurosa, lo que es esencial para la toma de decisiones informadas en la selección del modelo más adecuado para una tarea de detección específica. A continuación se presentan los resultados de las evaluaciones de cada modelo.

4.1.1 Métricas del Modelo de Tensorflow 2

El código proporcionado por *Tensorflow* permite utilizar *Tensorboard*, que muestra el desempeño del modelo entrenado utilizando cuatro métricas. Los términos “*localization loss*,” “*classification loss*,” “*regularization loss*” y “*total loss*” se refieren a diferentes componentes de la función de pérdida utilizada para evaluar y ajustar el rendimiento del modelo.

Se realizaron experimentos con diferentes parámetros en el entrenamiento. Uno de ellos fueron el número de épocas utilizadas. Aquí los resultados de la evaluación para el modelo entrenado con solo 1,500 épocas.

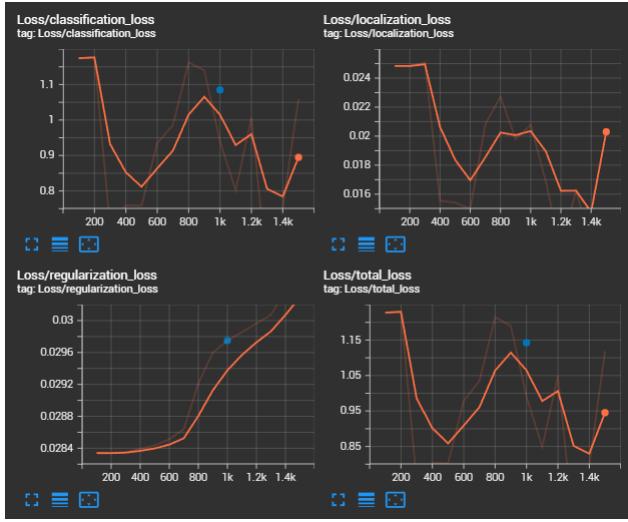


Fig. 1. Evaluación del modelo impelmentado en Tensor Flow con 1500 épocas

En los resultados de la evaluación, el valor de las métricas disminuye, lo cual es deseable, pero después de varias épocas vuelve a fluctuar. El número de épocas fue el parámetro con el que más se experimentó. A continuación los resultados de la evaluación para el modelo entrenado con 7,000 épocas.

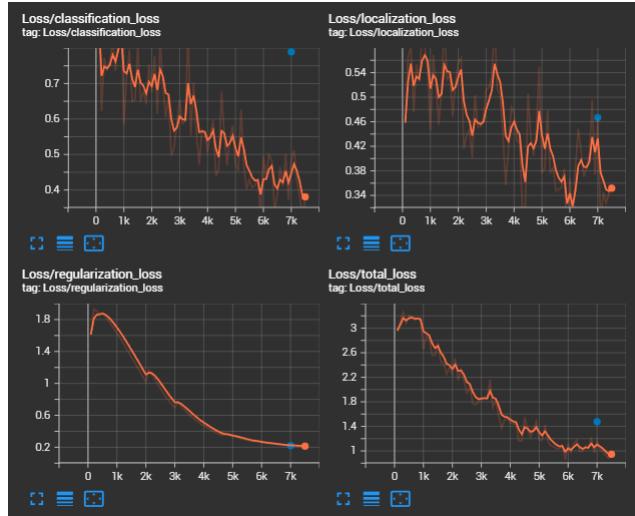


Fig. 2. Evaluación con 7000 épocas

Como se puede observar, el modelo entrenado con *Tensorflow 2* mejoró considerablemente con un aumento en el número de épocas utilizadas. El mejor desempeño se obtuvo con las siete mil épocas, sin embargo, el entrenamiento se vuelve cada vez más costoso.

4.1.2 Métricas del modelo de Pytorch

La librería de *PyTorch* realiza las evaluaciones pertinentes, siendo éstas *F1 Score*, *Precision* y *Recall*. A continuación, se presentan los resultados obtenidos después de un entrenamiento de 300 épocas.

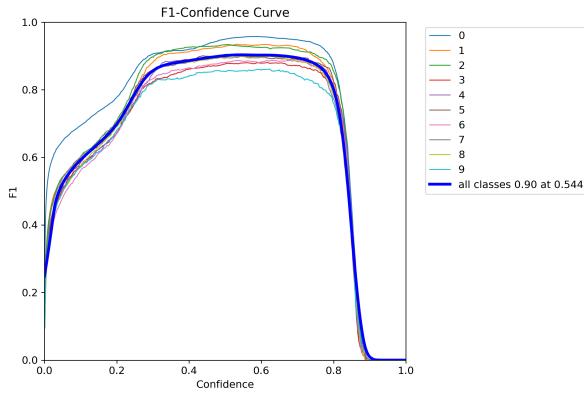


Fig. 3. Gráfica de la puntuación F1 con respecto de la confianza por clase

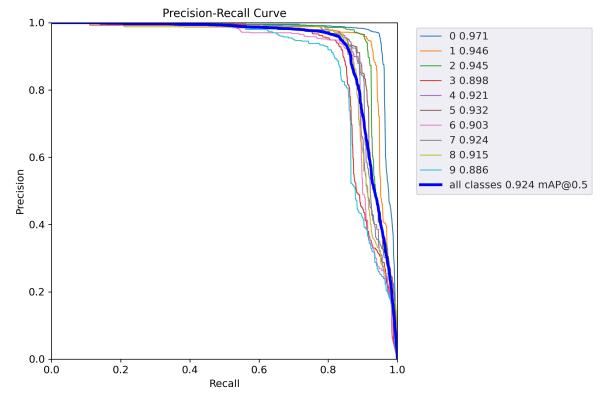


Fig. 4. Gráfica de la Precisión con respecto al *Recall* por clase

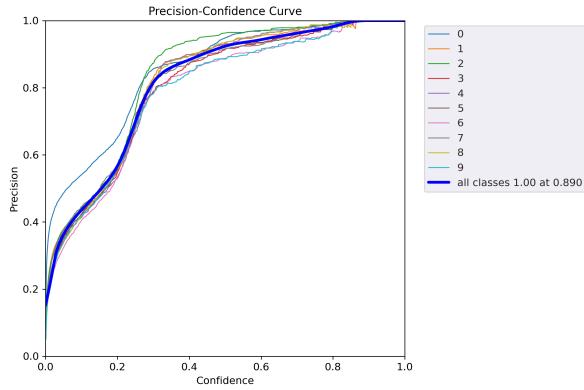


Fig. 5. Gráfica de la precisión con respecto a la confi-
anza por clase

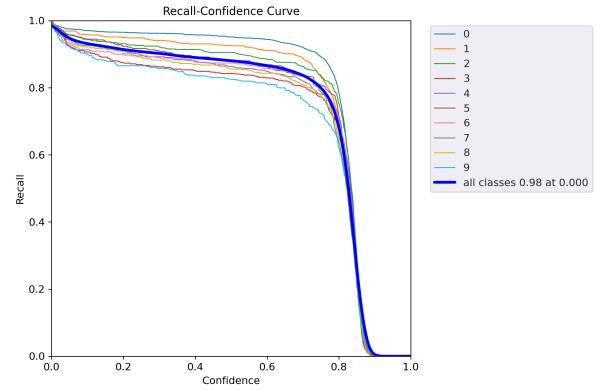


Fig. 6. Gráfica del *Recall* con respecto de la confianza
por clase

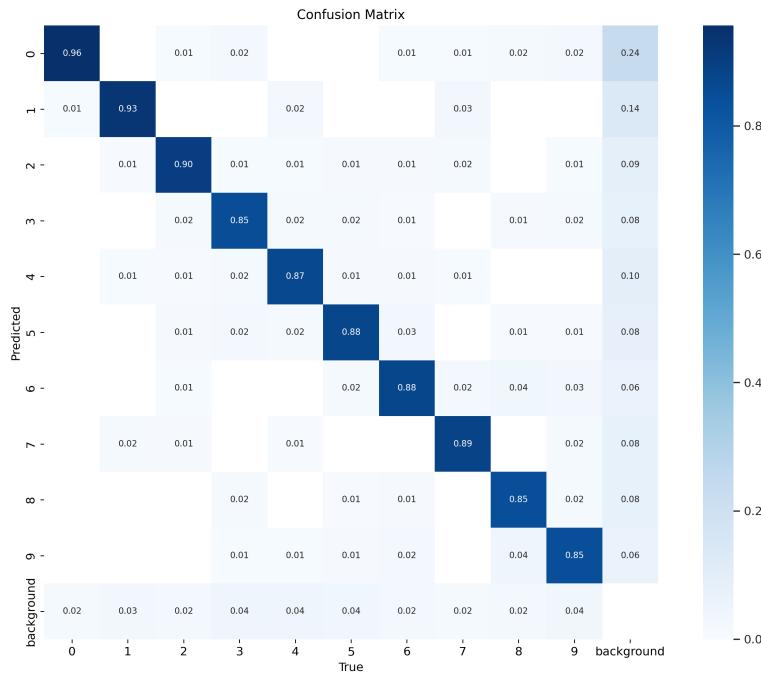


Fig. 7. Gráfica la matriz de confusión.

En general, como se puede observar en la figura 7, el modelo implementado en Pytorch muestra un rendimiento bastante aceptable, teniendo una precisión de hasta un 80% por clase. Por estas razones se consideró utilizar el modelo en *PyTorch* en lugar del modelo de *Tensorflow*.

4.2 Desempeño del modelo en formato Torchscript en la aplicación de Android

Una vez seleccionado el modelo entrenado con *PyTorch*, se hizo la conversión del formato del modelo a *Torchscript Optimizado* y se implementó en la aplicación de *Android*. Una vez en funcionamiento, se observó que el rendimiento del modelo optimizado se redujo en comparación a su versión en formato original, se cree que es debido a la cuantización del modelo, la cual es necesaria para poder ser leída por el intérprete de modelos de *Android*. A continuación, se presentan algunas pruebas hechas en el emulador de dispositivos incluido en *Android Studio*:

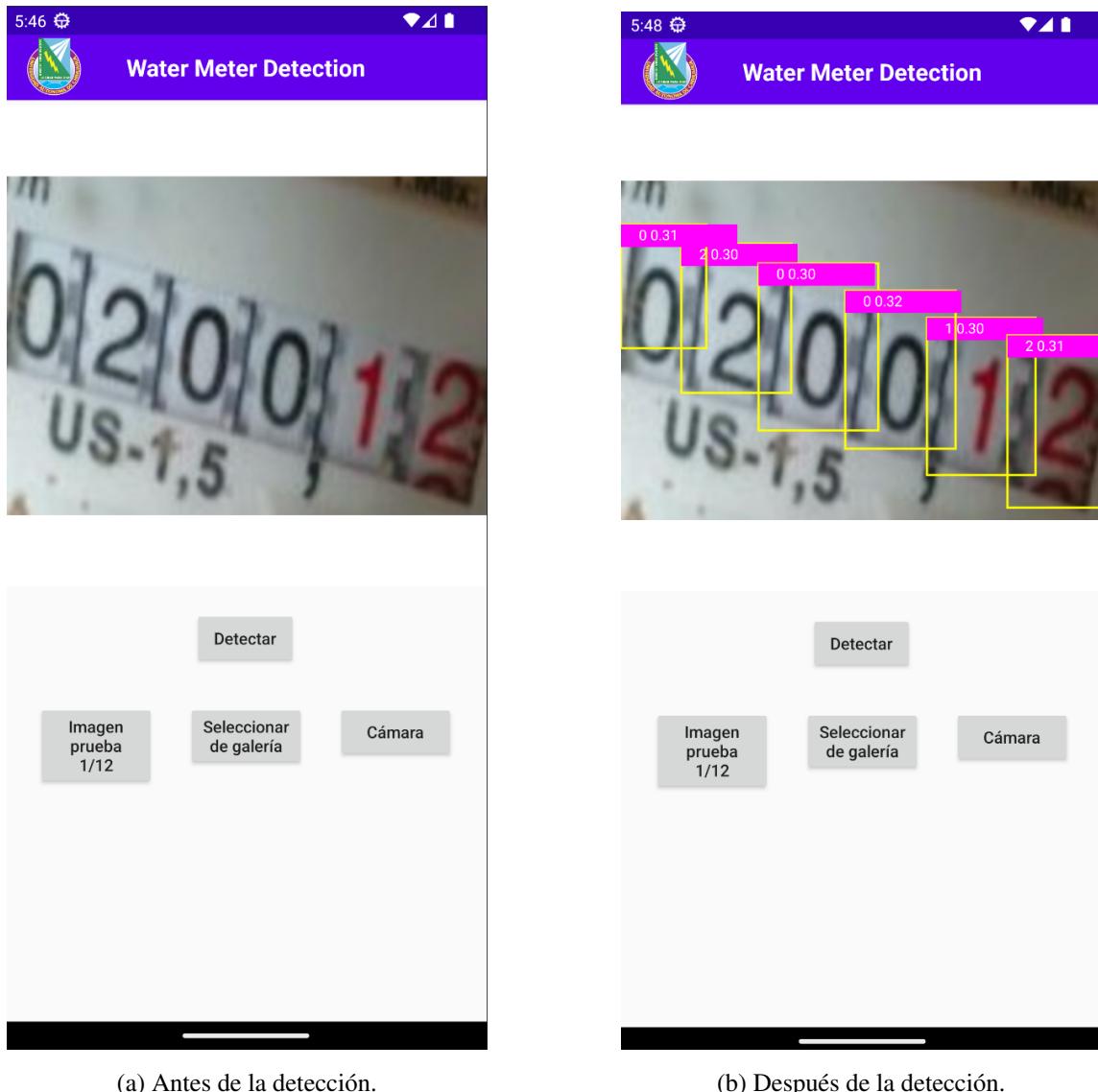


Fig. 8. Primer imagen de prueba.

Se puede apreciar en la figura 8, el modelo realizó correctamente la detección de los dígitos del medidor, con una confianza de alrededor de 0.3, en comparación con su versión original con una confianza de 0.8, como se mencionó previamente, disminuyó debido a la conversión y optimización del modelo.



(a) Antes de la detección.



(b) Despues de la detección.

Fig. 9. Imagen de prueba con perspectiva rotada.

En este caso, en la figura 9 se prueba con una imagen de un medidor en una perspectiva rotada. Se puede observar que el modelo no realizó correctamente la detección, los números que debía detectar como 7, se detectaron como 1. Sin embargo, el resto de números se detectaron correctamente.

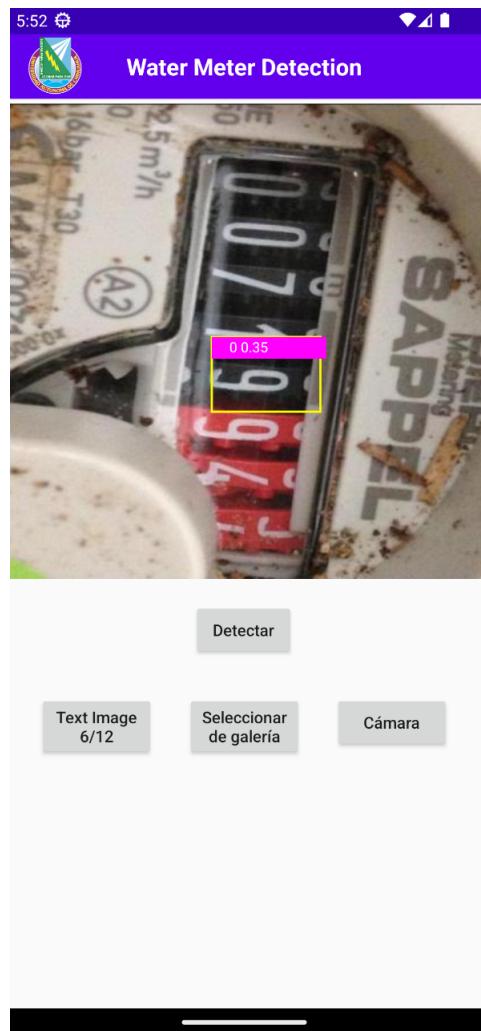


Fig. 10. Imagen de prueba con medidor rotado 90 grados.

Como se aprecia en la figura 10 se hizo la prueba con una imagen de un medidor a 90 grados, se puede notar que el modelo no hizo la detección correctamente para ninguno de los dígitos presentes en la imagen. Solo detectó que existía un dígito en el 9, sin embargo, se detectó como 0, todos los demás dígitos fueron ignorados.

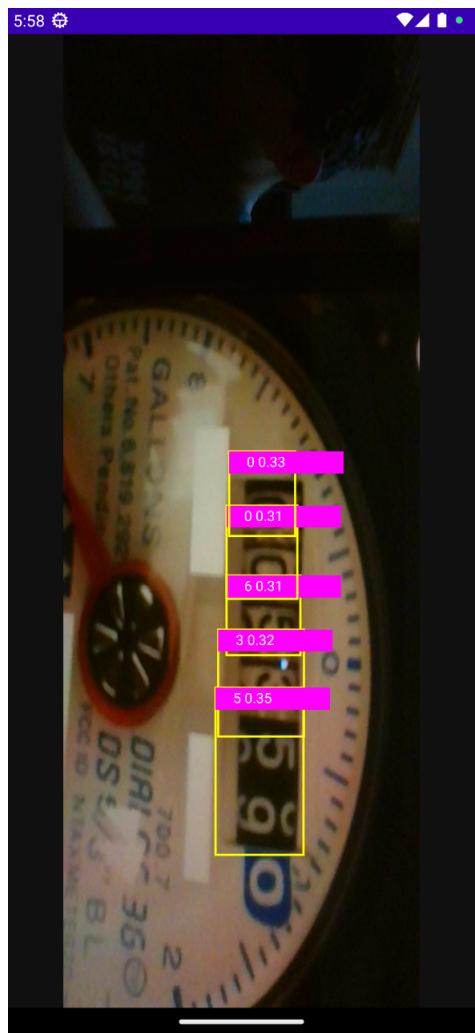


Fig. 11. Prueba con la cámara en directo de una foto mostrada en un dispositivo móvil.

En la figura 11 se hizo la prueba mostrando a la cámara una imagen en un teléfono celular. Como se puede apreciar, se hizo la detección correctamente de 4 de los 6 dígitos en el medidor, siendo el primero de estos un 9, el cual fue ignorado por el modelo, un 5 que se detectó incorrectamente como 6 y el resto bien etiquetados.

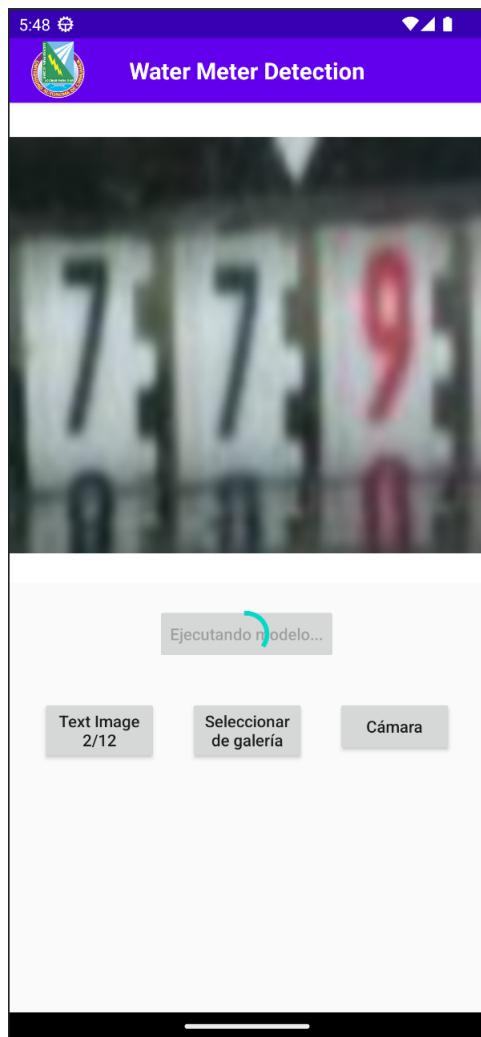


Fig. 12. Imagen de medidor escalada.

Por último, como se muestra en la figura 12, se realizó un escalado a una imagen del conjunto *test* del *dataset*, para hacer la prueba con una imagen más grande. El modelo no fue capaz de detectar ninguno de los 3 dígitos mostrados en la imagen.

5 CONCLUSIÓN Y DISCUSIONES

En conclusión, la implementación de una aplicación móvil para Android destinada a la detección y registro de medidas de medidores de agua se desarrolló como un proceso continuo de aprendizaje. Inicialmente, se exploraron diversas alternativas, como la utilización de la biblioteca de TensorFlow y la evaluación de modelos como MobileNet y EfficientDet. No obstante, al encontrarse con obstáculos en la implementación y en la búsqueda de una solución superior, se llevó a cabo una transición hacia PyTorch y se optó por el modelo YOLOv5. Esta decisión resultó en un rendimiento sobresaliente en cuanto a la detección de objetos, brindando una solución más eficiente y efectiva para el propósito deseado. Sin embargo, es importante señalar que, a pesar de los resultados sobresalientes con el modelo YOLOv5 en PyTorch, se contó con la limitación del desempeño cuando el modelo fue optimizado para dispositivos móviles, lo que destaca la necesidad de realizar más experimentación y explorar vías de acción adicionales para abordar este desafío.

A través de algoritmos avanzados, este modelo es capaz de analizar de manera rápida y precisa grandes volúmenes de datos de medidores, identificando patrones y anomalías que podrían pasar desapercibidos con enfoques convencionales. La capacidad predictiva del modelo no solo optimiza la detección de lecturas precisas, sino que también contribuye a prevenir posibles problemas, como fugas o malfuncionamientos, antes de que se conviertan en situaciones más críticas. En comparación con los métodos manuales o sistemas más antiguos, esta implementación ofrece una solución eficiente, precisa y adaptable para el registro de medidores de agua, promoviendo la gestión sostenible de recursos y proporcionando a los usuarios información valiosa para una toma de decisiones más informada (Chisag, Pasto, & Garofalo, 2022, Guato Chifla, 2015, Indecopi, 2019, Saravia Valle, Ruiz Rivera, & Calmet Agnelli, 2013).

Este proceso subraya la importancia de la flexibilidad y la adaptación en el desarrollo de aplicaciones tecnológicas, asegurando que se ofrezcan productos de la más alta calidad a los usuarios.

ANEXOS

TENSORFLOW

Inicialmente, se exploró el uso de TensorFlow Lite Model Maker (TensorFlow, s. f.), una herramienta de alto nivel que simplifica la creación de modelos de detección. Sin embargo, durante este proceso, [se encontró un bug](#) (TensorFlow Forum, 2023) que impidió el desarrollo de un código funcional utilizando esta herramienta.

Ante la limitación encontrada con TensorFlow Model Maker, se decidió utilizar [TensorFlow 2](#) (TensorFlow, s.f) para el entrenamiento de modelos de detección personalizados. Esto permitió un mayor control sobre la arquitectura del modelo y el proceso de entrenamiento. Se logró [entrenar con éxito modelos de detección de objetos](#) utilizando arquitecturas Mobilenet.

Para los modelos entrenados con Tensorflow 2, la inclusión de metadatos es esencial para proporcionar información adicional sobre el modelo y sus características específicas. Para lograr esto, se utilizó la TensorFlow Lite Metadata Writer API (TensorFlow, s. f.), que permite adjuntar metadatos al modelo en formato tflite. Estos metadatos pueden incluir información relevante, como la descripción del modelo, las etiquetas de clase, los datos de entrada esperados y cualquier otra información que facilite la comprensión y el uso del modelo por parte de otros desarrolladores. La incorporación de metadatos aporta claridad y contexto al modelo, lo que resulta fundamental en aplicaciones de detección de objetos.

Sin embargo, al intentar utilizar arquitecturas más avanzadas, como EfficientDet, surgieron desafíos relacionados con la conversión de estos modelos a formato tflite, que es esencial para la implementación en dispositivos móviles. A pesar de los esfuerzos, no se logró una conversión exitosa en estos casos, lo que llevó a la conclusión de que TensorFlow podría no ser la herramienta adecuada.

REFERENCIAS

- [1] Bagnato, Juan Ignacio. (2021). Modelos de Detección de Objetos, Aprende Machine Learning. Recuperado el 12 de septiembre de 2023, de Aprende Machine Learning: <https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/>
- [2] Chisag, A. G., Pasto, E. W., & Garofalo, J. A. (30 de septiembre de 2022) Gestión de información de lecturas del consumo de agua potable mediante una aplicación móvil. *Innovación y Software*, III(2), 6-25. Recuperado el 27 de abril de 2023.
- [3] Guato Chifla, J. L. (2015). Implementación de una aplicación para sistema operativo android que permitira la sincronización de las lecturas registradas en los medidores del sistema de agua potable. *Bachelor's thesis, Pontificia Universidad Católica del Ecuador Sede Ambato*. Recuperado el 27 de abril de 2023.
- [4] Indecopi. (2019). Inteligencia artificial y servicios públicos. Recuperado el 27 de abril de 2022, de Indecopi: https://www.indecopi.gob.pe/documents/51783/3257394/webgraf%C3%ADA_IA.pdf/90cc5f8d-9c5b-3261-5eac-165d97bbce6c
- [5] Kobzarev I., Tang J., & Vaighan B. (2021), Pytorch *android-demo-app/ObjectDetection*. Github. <https://github.com/pytorch/android-demo-app/tree/master/ObjectDetection>
- [6] Paperswithcode. (s. f.) Computer Vision *Object Detection*. <https://paperswithcode.com/task/object-detection>
- [7] Roboflow. (2022), *OCR 2 v4*. <https://universe.roboflow.com/hitax-will-rkmte/ocr-2-gtwyj/dataset/4>
- [8] Roboflow. (2021), *pascal to yolo classification digits Image Dataset v1*. <https://universe.roboflow.com/new-workspace-2wuu1/pascal-to-yolo-classification-digits/dataset/1>
- [9] Roboflow. (2022), *WaterMeters v2*. <https://universe.roboflow.com/computervision-jl3kf/waterneters-oj3it/dataset/2>
- [10] Saravia Valle, E., Ruiz Rivera, M. E., & Calmet Agnelli, R. (2013). Diseño de un sistema móvil para la lectura de medidores mediante tecnología Bluetooth. *Industrial Data* (16(1)), 134-143. Recuperado el 27 de abril de 2023.
- [11] TensorFlow. (s. f.), *TensorFlow Lite Model Maker*. TensorFlow. https://www.tensorflow.org/lite/models/modify/model_maker

-
- [12] TensorFlow. (s. f.), *Object Detection*. Github. https://github.com/tensorflow/models/tree/master/research/object_detection
 - [13] TensorFlow. (s. f.), *TensorFlow Lite Metadata Writer API*. TensorFlow. https://www.tensorflow.org/lite/models/convert/metadata_writer_tutorial
 - [14] Ultralytics. (2023), *Yolov5*. Github. <https://github.com/ultralytics/yolov5>
 - [15] Ultralytics. (2023) Ultralytics YOLOv8 Docs *Comprehensive Guide to Ultralytics YOLOv5*. <https://docs.ultralytics.com/yolov5/>
 - [16] Virtudes, Joaquín. (2023) Damavis - Data - Machine Learning - Visualization *Reconocimiento de objetos con deep learning*. <https://blog.damavis.com/reconocimiento-de-objetos-con-deep-learning/>