ESTUDIO DE LA DETECCIÓN DE OBJETOS CON ESTIMACIÓN DE DISTANCIA EN TIEMPO REAL

Por: Jhon Vargas Reyes, Miguel Ruiz Adarmes

Maestría de Ciencia de la Computación - Curso: MCC 616 Elementos de Procesamiento de Imágenes

Facultad de Ciencias, Universidad Nacional de Ingeniería, Lima, Perú

Lima, 26 de Agosto de 2021

I. RESUMEN

Las personas invidentes generalmente requieren ayudas para desplazarse libremente por distintos ambientes. En caso su ambiente sea conocido (Por ejemplo su casa) está ayuda podría ser mínima, pero esta asistencia se vuelve necesaria cuando transitan o se encuentran en ambientes desconocidos (Por ejemplo un parque o espacios libres), es por esto que existe el problema de detección de objetos u obstáculos presentes en el ambiente en que se encuentran.

Palabras clave: Visión en dispositivo móvil, red neuronal convolucional, SSD, Mobilenet, Real Time Detection.

II. INTRODUCCIÓN

Dentro de los diferentes campos de estudio de Computer Vision, vemos que están los campos de Object Distance Estimation y Object Detection que usan diferentes modelos muy buenos cómo son MobilenetV1, MovilenetV2, MobilenetV3, SSD, YOLO los cuales nos pueden brindar un punto de partida para poder iniciar con la investigación de nuestro objetivo. También mencionar los diferentes dataset de imagenes abiertos cómo MS COCO[9], Imagenet.

Nuestro objetivo es realizar investigación sobre detección de objetos y estimación de distancia de los objetos detectados que permita desarrollar una aplicación móvil(android) usando los modelos de detección y de estimación con el fin que ayude a guiar a personas invidentes cuando se encuentran en un ambiente con obstáculos desconocidos.

El presente informe está dividido en 4 partes. El estado del arte nos da una visión sobre las diferentes técnicas, modelos que se pueden usar para lograr el objetivo. En la parte de metodología, se describe el enfoque que se usó desde la creación del modelo hasta la integración con la aplicación móvil. En la parte de Experimentación, mostramos los resultados generados. Y en la parte final exponemos las conclusiones y lecciones aprendidas encontradas.

III. EL ESTADO DEL ARTE

Para este trabajo se usó como referencia [1] y [2], que permitieron con el desarrollo de modelos de detección de imágenes cómo MobileNet y SSD con base. El desarrollo usa un modelo pre-entrenado ssd_mobilenet_v1_coco que se encuentra disponible en [3], este modelo fue entrenado con la base de datos MS COCO. Este modelo utiliza cómo arquitectura base a Mobilenet y lo integra con la arquitectura SSD que es una red de convolución única que aprende a predecir las ubicaciones de los cuadros delimitadores y a clasificar estas ubicaciones en una sola pasada.

Para la detección de distancias se usó cómo referencia a [4], [5] donde clasifican las técnicas de clasificación de distancias en 2 grupos: Stereo y Mono En la primera se utilizan 2 cámaras y en el segunda solo 1. También se revisó [6] en donde usan algoritmos más avanzados como CSEN(Convolutional Support Estimator Networks) modificados para generar lo que llamamos a este enfoque novedoso como Regresión basada en la representación (RbR).

Por último, en la parte de detección en tiempo real, se optó por usar la librería openCV para poder cargar video en tiempo real y basado en la inferencia generada por el modelo se crearon cuadros de reconocimiento de imágenes prediciendo su posición.

MobileNets es un modelo eficiente para aplicaciones de visión integradas en dispositivos móviles. Se basa en una arquitectura optimizada que utiliza convoluciones separables en profundidad para construir redes neuronales profundas y ligeras.

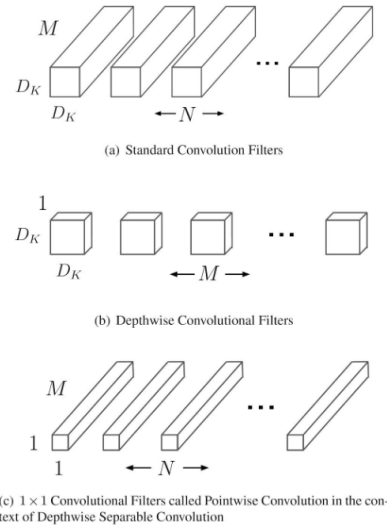

Fig. 1. En la arquitectura de MobileNet.

Convolución separable en profundidad(DepthWise Separable Convolutions), se componen de dos capas: convoluciones en profundidad y convoluciones puntuales. Se usa convoluciones en profundidad para aplicar un solo filtro por cada canal de entrada (profundidad de entrada). La convolución puntual, una convolución simple de 1 × 1, se utiliza para crear una combinación lineal de la salida de la capa en profundidad. MobileNets utiliza batchnorm y RELU para ambas capas.

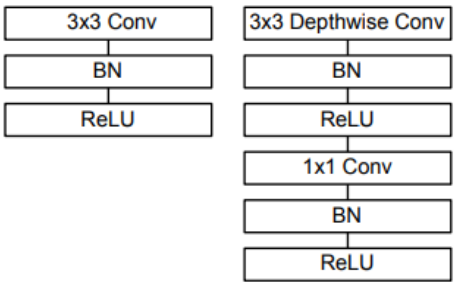

Fig. 2. Capas convolucionales estándar(Izquierda) y Capas convolucionales separadas en profundidad (Derecha).

Estructura de la Red y Entrenamiento, la estructura de MobileNet se basa en convoluciones separables en profundidad excepto por la primera capa, que es una convolución completa. La arquitectura de MobileNet se define en la Tabla 1. Todas las capas van seguidas de capas batchborm y RELU con la excepción de la capa final completamente conectada que no tiene no linealidad y alimenta a una capa softmax para la clasificación.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Tabla 1.

Hyperparameter: Multiplicador de ancho(Width Multiplier**),** modelos más pequeños: Usado para construir modelos más pequeños y menos costosos computacionalmente. El parámetro α llamado multiplicador de ancho se usa para adelgazar una red uniformemente en cada capa. Para una capa y un multiplicador de ancho dados α, el número de canales de entrada M se convierte en αM y el número de canales de salida N se convierte en αN

Hyperparameter: Multiplicador de resolución (Resolution Multiplier.**),** representación reducida: El segundo hiper parámetro para reducir el costo computacional de una red neuronal es un multiplicador de resolución que se aplica a la imagen de entrada. Esto provoca que la representación interna de cada capa se reduzca proporcionalmente por el mismo multiplicador.

El resultado del nuevo coste computacional de una capa DepthWise Separable convolution haciendo uso de los dos hiperparámetros explicados anteriormente:

$$\varrho DI \cdot \varrho DI \cdot \alpha M \cdot DF \cdot DF + \alpha M \cdot \alpha N \cdot \varrho DI \cdot \varrho DI$$

Siendo α el hiper parámetro Width Multiplier y ϱ el hiper parámetro Resolution Multiplier. Por otro lado, *DI* la dimensión de la entrada a tratar en la capa convolucional típica, *DF* la dimensión del filtro a aplicar sobre la entrada, *M* la profundidad de la entrada y *N* la profundidad de la salida, condicionada por el número de filtros a aplicar en la capa convolucional.

La Arquitectura SSD, es una de las primeras redes que prescinde del módulo de sugerencia de localización de elementos manteniendo una precisión muy similar a los que sí lo tienen, como sus predecesores. Aun no siendo la primera que no tiene módulo de sugerencia de localización de elementos sí que fue la primera en conseguir unos resultados decentes en cuanto a precisión con una considerable velocidad máxima de 59 FPS (Fotogramas Por Segundo).

La red SSD también destaca por la carencia de capas del tipo fully connected y estar compuesta principalmente de capas convolucionales.

Como se puede observar en la Figura 3, la red contiene una red base VGG-16 a la cual se le ha quitado la parte de predicción formada por las capas fully connected y en su lugar se han colocado diferentes capas convolucionales de tamaño de filtro distinto.También se puede apreciar en la imagen que la red YOLO[7], la cual se mencionara más adelante, tiene una capa fully connected, al contrario que la red SSD.

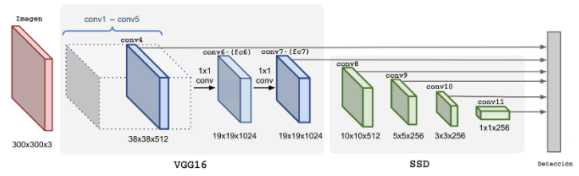

Fig. 3. Arquitectura de la red SSD con base en VGG-16.

Se puede apreciar en la Tabla 2. que la red SSD, se compara con otra red llamada YOLO[7]. Esto ocurre porque la red YOLO es competidora directa de la red SSD como redes del estado del arte. Finalmente se ha demostrado que la red SSD con la configuración SSD300 es superior a la red YOLO[7] tanto en capacidad de clasificación, cómo en localización de objetos y en velocidad.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Tabla 2.

No obstante, antes de que se publicará la red SSD,los desarrolladores de la red YOLO habían publicado una mejora de la red con el nombre de YOLOv2[8] lamada YOLO9000. La red YOLOv2 supera a la red SSD con creces tanto en velocidad como en capacidad de localización de elementos y la clasificación de los mismos, como se puede apreciar en la Tabla 3:

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Tabla 3.

Entonces se toma el modelo de MobileNet que ha demostrado ser una red que obtiene buenos resultados aun reduciendo en gran medida el coste de cómputo necesario para su ejecución y el número de parámetros, convirtiéndola en una red más ligera.

Los desarrolladores han realizado mucha más experimentación con la MobileNet, pero la prueba más significativa para nosotros es la de sustituir la red VGG-16 de la red SSD por la MobileNet[1]. Con ello consiguieron disminuir el coste de cómputo en un 96,56% y el número de parámetros de la red en un 79,46%, por la pequeña penalización de 1,8% en la precisión.

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8

Tabla 4.

En la Tabla 4 se puede comprobar esta disminución, donde se muestran las pruebas de los desarrolladores de las distintas redes sobre el dataset COCO[9]. Las precisiones obtenidas están condicionadas por las métricas utilizadas en el COCO Primary Challenge

IV. METODOLOGÍA

El enfoque que utilizamos para encontrar

1. Pre-Entrenar la red SSD+MobilenetV1 con datos un dataset de Imágenes bajadas de google.
2. Crear dataset de imágenes para detección de objetos utilizando técnicas de label map de manera que se pueda usar en el momento de predecir la imagen.
3. Realizar pruebas de inferencia, predicción dado el modelo pre-entrenado y el dataset generado.
4. Integrar la estimación de distancia.
5. Realizar pruebas en tiempo real usando openCV.
6. Montar el modelo guardado(serializado) en la aplicación móvil.

En la Figura 4, vemos un resumen gráfico de la metodología usada. El punto de inferencia junto con estimación de distancia y detección en tiempo real fue un proceso cíclico de ensayo y error.

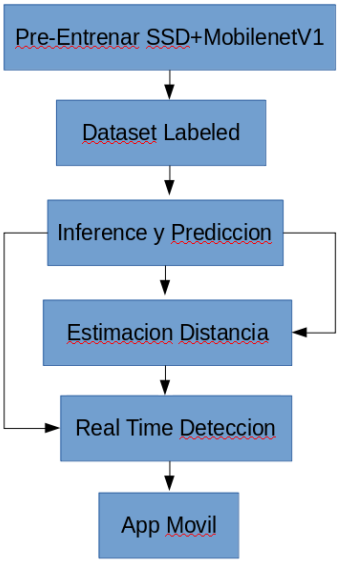

Figura 4.

Modelo,

```
model_name = 'ssd_mobilenet_v1_coco_2017_11_17'
detection_model = load_model(model_name)

Downloading data from http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_2017_11_17.tar.gz
76537856/76534733 [==============================] - 1s 0us/step
76546048/76534733 [==============================] - 1s 0us/step
```

Figura 5.

Dataset,

```
detection_model.signatures['serving_default'].output_dtypes

{'detection_boxes': tf.float32,
 'detection_classes': tf.float32,
 'detection_scores': tf.float32,
 'num_detections': tf.float32}
```

Figura 6.

Inferencia,

```
for image_path in TEST_IMAGE_PATHS:
  show_inference(detection_model, image_path)
  #print(image, shapes)
```

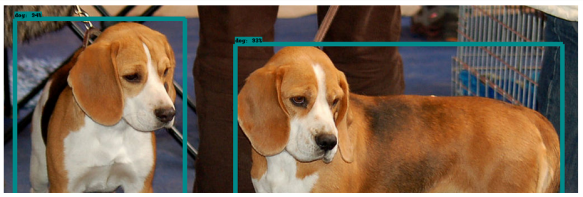

Figura 7.

App movil, para que sea usado en la aplicacion movil, se tuvo que pasar a formato tflite.

```
#save your model in the SavedModel format
export_dir = 'saved_model'
tf.saved_model.save(model, export_dir)

# Converting a SavedModel to a TensorFlow Lite model.
converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()
```

Figura 7.

V. EXPERIMENTACIÓN Y RESULTADOS

Modelo Pre-entrenado: Se usó la Transferencia de Conocimiento (Transfer Learning) usando Keras y Mobilenet para clasificar imágenes.

La predicción para algunas imágenes dieron los siguientes valores:

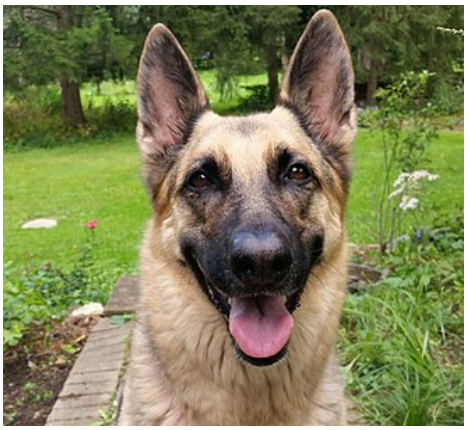

Figura 8.

[[**('n02106662', 'German_shepherd', 0.9796372)**,
('n02105162', 'malinois', 0.020184083),
('n02091467', 'Norwegian_elkhound', 0.00015799515),
('n02116738', 'African_hunting_dog', 5.2901587e-06),
('n02105251', 'briard', 3.9127376e-06)]]

Figura 9.

[[**('n02099712', 'Labrador_retriever', 0.73073703)**,
('n02087394', 'Rhodesian_ridgeback', 0.03984367),
('n02092339', 'Weimaraner', 0.03359009),
('n02109047', 'Great_Dane', 0.028944707),
('n02110341', 'dalmatian', 0.022403581)]]

Aplicativo Android: Se clono el proyecto que existe en https://github.com/tensorflow/examples y se abrió

en Android Studio el proyecto que se encuentra en "…\examples-master\lite\examples\object_detection\android"

Este proyecto se usó como código base para construir el aplicativo. Luego de instalar las dependencias faltantes, se logró compilar y ejecutar el aplicativo en dos dispositivos móviles: Uno con SO Android 5 y otro con SO Android 11.

El aplicativo usa un modelo pre-entrenado que está empaquetado en dos archivos:

1. detect.tflite
2. labelmap.txt

El modelo fue entrenado con un universo de 91 imágenes de objetos y en los experimentos se encontraron los siguientes ratios de predicción:

SO	Objeto	Ratio predicción (%)
Android 10	Tecclado PC	50-60
	Laptop	69-71
	Cellphone	65-71
	Mouse	50-60
Android 5	Tecclado PC	49-57
	Laptop	63-70
	Cellphone	62-69
	Mouse	51-50

Como puede observarse las diferencias de ratios no son significativas. En cuanto a los tiempos para obtener cada predicción se obtuvo en promedio:

SO	Tiempo predicción (%)
Android 10	< 1 seg
Android 5	1-2 seg

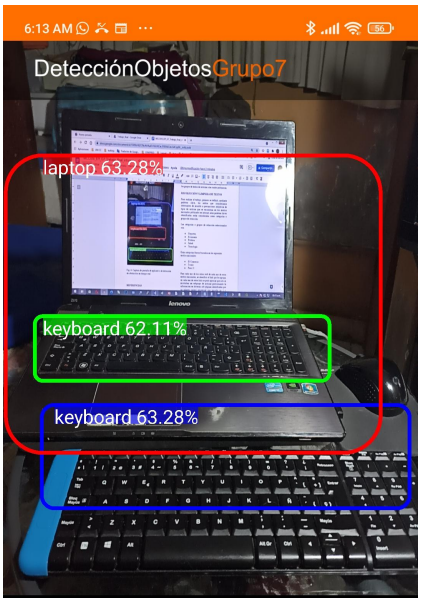

Fig. 4. Captura de pantalla de aplicativo de detección de obstáculos en tiempo real

VI. CONCLUSIONES

1. Se pudo verificar que el modelo MobileNet funciona bien en dispositivos de bajos recursos de hardware.
2. Para el aplicativo Android no se pudo completar la estimación de distancia por límite de tiempo.
3. El aplicativo desarrollado requiere la implementación de lectura audible del objeto detectado y de la distancia estimada usando Text to Speech.
4. Se tienen que saber los parámetros de salida cuando se realiza una predicción de imagen con el objetivo de conocer la posición del cuadro del objeto y la distancia estimada.
5. Para que los dispositivos IOT, Smartphones necesitan el modelo guardado en formato *.tflite que permite el correcto uso del recursos de CPU y RAM.

VII. REFERENCIAS

[1] G. Howard, M. Zhu, B. Chen, D. KAlenichencko, W. Wang, T. Weyand, M. Adreetto, H. Adam. Mobile Nets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv: 1704.048861v1, 2017

[2] M. Liu, M. Zhu. Mobile Video Object Detection with Temporally-Aware Feature Maps. arXiv: 1711.06368v2, 2018

[3] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[4] Mete Ahishali, Mehmet Yamac, Serkan Kiranyaz, Moncef Gabbouj, Representation Based Regression for Object Distance Estimation, arXiv:2106.14208v1, 2021

[5] Ashfaqur Rahman, Abdus Salam, Mahfuzul Islam, and Partha Sarker. An Image Based Approach to Compute Object Distance. 2018

[6] Muhammad Abdul Haseeb, Jianyu Guan, Danijela Ristić-Durrant, Axel Gräser. DisNet: A novel method for distance estimation from monocular camera. Institute of Automation 2018.

[7] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. "You only look once: Unified, real-time object detection". Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788.(2016).

[8] Redmon, J., Farhadi, A."YOLO9000: better, faster, stronger".arXiv preprint.. (2017).

[9] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Zitnick, C. “Microsoft coco: Common objects in context.”European conference on computer vision, pp. 740-755. Springer, Cham.(2014,)

VIII. ARCHIVOS

https://drive.google.com/drive/folders/1Q4z3uPe7_VVBvvwsmPT624Hk0e1cOKD0?usp=sharing