

COVID-19: Detección del Cumplimiento del Distanciamiento Social en Aglomeraciones de Personas Aplicando Técnicas de Deep Learning con YOLO

Rodrigo Castro
Universidad ESAN
Lima, Perú
14100087@ue.edu.pe

Pedro I. Montenegro
Universidad ESAN
Lima, Perú
16100041@ue.edu.pe

Gonzalo Bardalez
Universidad ESAN
Lima, Perú
17100802@ue.edu.pe

Abstract—COVID-19 es una enfermedad respiratoria de la familia de coronavirus, conocido como SARS-CoV2, que afecta al organismo humano con síntomas mayormente entre fiebre, tos y cansancio. A inicios de mes de marzo, la Organización Mundial de la Salud (OMS), declaró la pandemia del brote del virus a causa que ya estaba ubicada en más de 114 países. En el presente trabajo se propone una metodología basada en Deep Learning y Visión Computacional, que permite la detección de personas ubicadas en un archivo de video, midiendo la distancia entre ellos para detectar si se incumple la distancia social mínima, además, de contabilizar la cantidad de personas dentro de la escena. De esta manera se podrá monitorear el cumplimiento de las leyes establecidas por el gobierno, como el distanciamiento social obligatorio, además de conocer donde se generan aglomeraciones de personas para su posterior control y así poder apoyar a los esfuerzos de mitigar el daño que causa la pandemia actual del COVID-19 en el Perú y los distintos países del mundo. Para lograr este objetivo se utilizaron los algoritmos YOLOv2, YOLOv3, YOLOv4 para la detección de personas, obteniendo un mAP de 55.30%, 60.16% y 78.67% respectivamente, posteriormente se calcularon las distancias euclidianas entre los centroides de las imágenes para detectar las violaciones del distanciamiento social y se realizó el conteo de personas, donde YOLOv2 y YOLOv3 obtuvieron un MAE de 6.21 y 2.09 respectivamente.

Index Terms—COVID-19, Machine Learning, Computer Vision, Crowd Detection, Classification, Social Distance, Deep Learning, YOLO

I. INTRODUCCIÓN

COVID-19 es una enfermedad respiratoria de la familia de coronavirus [1], conocido como SARS-CoV2, que afecta al organismo humano con síntomas mayormente entre fiebre, tos y cansancio. A inicios de mes de marzo, la Organización Mundial de la Salud (OMS), declaró la pandemia del brote del virus a causa que ya estaba ubicada en más de 114 países. Hoy en día se reportan más de 3 millones de contagiados a nivel mundial. Diversos países del mundo han tomado medidas para poder reducir la curva de infectados que en la mayoría de estos países la razón de contagiados sigue aumentando. Como una de las medidas, se presentó el estado de emergencia sanitario,

que establece un aislamiento social obligatorio y, en algunos casos, una cuarentena. Por otro lado, se estableció que en los casos excepcionales donde haya que salir de casa, se cumpla con un distanciamiento social de aproximadamente 6 pies (1.8 metros) entre personas. Sin embargo, no todos los países están cumpliendo con esta restricción. En el caso de Perú, por ejemplo, la cantidad total de personas infectadas está cerca de los 200,000, tomando en cuenta el hecho que las personas se aglomeran cada día en los centros de abasto y zonas comerciales, principalmente violando la ley del aislamiento social dictado por [2].

Esta investigación tiene como objetivo detectar, a través de vídeos filmados, si entre una cierta cantidad de personas existe una violación a esta ley. Se propone una metodología para el conteo de personas y la detección del cumplimiento de la ley de distanciamiento social por medio de técnicas de Deep Learning, en específico se utilizan los algoritmos de detección de objetos YOLO versiones 2, 3 y 4. De esta manera se podrá monitorear el cumplimiento de las leyes establecidas por el gobierno, como el distanciamiento social obligatorio, además de conocer donde se generan aglomeraciones de personas para su posterior control. Y así poder apoyar a los esfuerzos de mitigar el daño que causa la pandemia del COVID-19 en el Perú y los distintos países del mundo. Esta investigación esta estructurada de la siguiente forma: La Sección II, describe el Estado del Arte; la Sección III la metodología a seguir, donde se explica el flujo de trabajo de la investigación, la base de datos utilizada, pre-procesamiento realizado y el procesamiento de la data; la Sección IV los resultados obtenidos; la Sección V las conclusiones; y la Sección VI las recomendaciones.

II. ESTADO DEL ARTE

En esta sección se presentaron cuatro artículos de investigación que apoyan y sustentan la viabilidad de la metodología propuesta. Cada sub-sección detalla las características de cada trabajo de investigación: a) Problema, b) Datos utilizados, c) Metodología y d) Resultados.

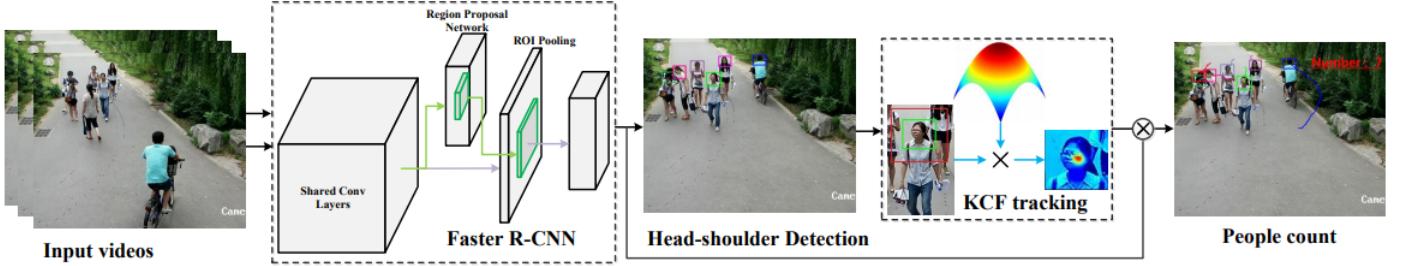


Fig. 1. Flujo de trabajo de la metodología propuesta [3]

A. Deep People Counting with Faster R-CNN and Correlation Tracking

Los autores explican que el conteo de personas en multitudes es un problema clave para muchas tareas de visión computacional, mientras que muchos de los métodos existentes tratan de contar personas por medio de regresión con características extraídas a mano [3]. Además, detallan que recientemente el avance rápido del Deep Learning a resultado en muchos modelos prometedores detectores de clases de objetos genéricos.

Dataset	Frame	Resolution	FPS	Count	Total
Vertical	5527	352 x 288	25	0-7	4965
Oblique	39594	1280 x 720	25	0-40	363498

Fig. 2. Resumen detalles de los datasets [3]

Asimismo, el conteo de personas, objetos, entre otros, es una tarea que está de moda y es bastante demandada debido a la vigilancia por video y sus diversas aplicaciones en el sector de seguridad. Existen dos maneras de realizar el conteo: por regresión y por detección. De tal manera, para el conteo por regresión, se extraen características de bajo nivel y se genera la regresión para estimar la cantidad de personas u objetos. Es una muy buena técnica cuando hay una gran densidad de personas. Por desgracia, este método solo cuenta y no extrae información de la imagen (cada individuo). Para el conteo por detección, se suelen usar CNNs. Sin embargo, esta técnica tiene grandes problemas para contar en imágenes pobladas, es decir, con una gran cantidad de personas, pues tienden a arrojar falsas detecciones.

En esta investigación los autores evaluaron el rendimiento en el conteo de personas en dos datasets con distintas perspectivas: Vertical y Oblicua. El dataset con la perspectiva vertical incluía tres vídeos en diferentes escenas, donde se extrajeron 2500 frames de dos de los vídeos para el entrenamiento y el vídeo restante para la prueba. De tal forma, el dataset con la perspectiva oblicua estaba dividido en dos partes: 5 secuencias de vídeos de 8 minutos de cinco escenas fueron tratadas como

sets de entrenamiento y el set de prueba constó de 3 secuencias de video de 10 minutos de tres escenas diferentes. Un resumen de los detalles de ambos datasets se pueden ver en la Figura 2: el nombre del dataset, número de frames, resolución, fps, mínimo y máximo número de personas, número total de instancias de personas.

En este paper se aplicó el algoritmo Faster R-CNN para entrenar dos modelos detectores de cabeza-hombros con los datasets de perspectivas oblicua y vertical respectivamente. Luego los autores explican que se añadió Online Hard Example Mining (OHEM), para mejorar la detección de multitudes en el proceso de entrenamiento del Faster R-CNN para reducir las falsas detecciones. Además, los autores explican que, aunque una persona puede no ser detectada en un frame, sí lo será en el siguiente. Después de obtener los bounding boxes del grupo de personas por la detección del Faster R-CNN con OHEM, se usó un rastreador(tracker) de Filtro de Correlación Kernelizado (KCF) para obtener el rastro de la multitud. Finalmente, se fusionó la detección y los resultados de rastreo para obtener una trayectoria continua y estable para el conteo de personas. En la Figura 1 se puede observar el flujo general de trabajo de la metodología propuesta. Asimismo las métricas que utilizaron para medir el rendimiento del modelo fueron el MAE y el MSE.

Como se puede ver en la Figura 3, el mejor resultado en ambos datasets se obtuvo al combinar el Faster R-CNN con el OHEM y el KCF con un MAE de 0.49 y 1.94, y un MSE de 0.49 y 5.21 para las perspectivas vertical y oblicua, respectivamente. De tal forma, los autores nos demuestran que presentaron un método efectivo para el conteo de personas basado en la combinación de rastreo (tracking) con detección. Asimismo, explican que a diferencia de modelos anteriores que estimaban el tamaño total de la multitud de un solo frame, ellos lograron realizar un conteo robusto de personas al tomar ventaja de la red Faster R-CNN basada en la detección de cabeza-hombros y la correlación con los resultados de rastreo. Además de aplicar un esquema OHEM para eliminar falsas alarmas y fusionar los resultados de detección y rastreo para un conteo preciso de personas. En la Figura 3 se puede

observar los resultados de los experimentos con las distintas combinaciones de algoritmos utilizadas en la investigación.

Method	Vertical View		Oblique View	
	mae	mse	mae	mse
ACF	1.51	3.41	6.60	61.43
Faster	1.05	1.71	3.93	21.15
Faster+OHEM	1.04	1.76	3.29	15.22
ACF+KCF	1.01	1.62	4.75	31.08
Faster+KCF	0.58	0.58	2.61	9.33
Faster+OHEM+KCF	0.49	0.49	1.94	5.21

Fig. 3. Comparación de resultados con distintos Acercamientos [3]

De tal manera, el sistema basado en este método demuestra su robustidad a la interferencia y cambios complejos. Su experimento demostró la superioridad de su metodología propuesta en los dos datasets de escenas con multitudes de personas. Asimismo, se demuestra la efectividad de las CNNs para poder realizar conteo de personas en distintos entornos, y la necesidad de combinarlas con otros métodos para mejorar sus resultados y su capacidad predictora. Este artículo nos ha proporcionado una visión de que metodología seguir y que tipo de algoritmos podremos utilizar.

B. Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deepsort techniques

Esta investigación fue realizada tomando inspiración a partir de una herramienta de IA creada por Landing AI, que monitorea el distanciamiento social dentro del centro de trabajo analizando grabaciones filmadas por las cámaras con el fin de apoyar en la lucha contra el COVID-19. Los autores comentan que el distanciamiento social es reconocida como una técnica confiable para detener el brote del virus que dio origen en Wuhan, China. También comentaron que existen algunas áreas de investigación establecidas para este concepto, como se da con la detección humana en sistemas de vigilancia. Hay dos etapas que seguir para detectar un objeto en movimiento: detección de objeto y clasificación de objeto. Para el primer caso, se logra mediante substracción de fondo, flujos ópticos para identificar regiones dada una secuencia de imágenes, y técnicas de filtrado espacio-temporal usando características tridimensionales para identificar los parámetros de movimiento. Estos métodos, como indica el artículo, son muy favorables y menos complejos computacionalmente, pero su rendimiento se limita a la presencia de ruido e incertezas en patrones de movimiento.

Entre otras técnicas avanzadas que han aparecido en la última década, destacan aquellas basadas en Redes Neuronales Convolucionales (CNN), Faster Region-Based y Region-Based. Una de las desventajas de estas técnicas es que pueden caer en largos períodos de tiempo de training; sin embargo, pueden llegar a ser muy eficientes en la clasificación. En

contraste, uno de estos modelos, de nombre YOLO, aplica metodologías basadas en regresión para separar las cajas delimitadoras e interpretar sus probabilidades de clase.

Los autores han evaluado diferentes modelos de detección de objetos de alto rendimiento como el RCNN, fast RCNN, faster RCNN, SSD y YOLO (v1, v2, v3) con los datasets de PASCAL-VOC y MS-COCO; la variación del accuracy y velocidad de estos modelos dependen mucho de factores como la arquitectura backbone (VGG-16, ResNet-101, Inception v2, etc.), tamaños de entrada, profundidad, entornos de software y hardware. En esta investigación se introdujo en concepto de los anchor boxes, que son utilizados para poder englobar los objetos según su escala y tamaño. Como se puede apreciar en la Figura 04, los modelos mencionados previamente muestran su rendimiento en términos de scores mAP y FPS para dos casos diferentes: score mAP para determinar los mejores modelos de ambos datasets y el score FPS para el dataset VOC usando una GeForceGTX Titan X.

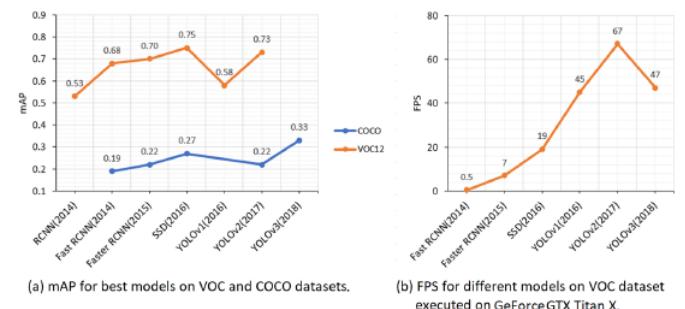


Fig. 4. Performance de los modelos más populares de detección de objetos

La Figura 5 describe resume el resultado de los experimentos probados para cada modelo de arquitectura backbone para el dataset ILSVRC ImageNet challenge. Como se puede apreciar en esta imagen, el modelo Inception v2 logró un mayor ratio de accuracy debido a que usaron la menor cantidad de hiperparámetros para este modelo, logrando un ratio de 7.40. En consecuencia, esta arquitectura se usará para los modelos RCNN y SSD (Single Shot Detector), mientras que la arquitectura Darknet-53 para YOLO v3 (recomendado por otro autor).

Backbone model	Accuracy (a)	Parameters (p)	Ratio (a*100/p)
VGG-16 [68]	0.71	15 M	4.73
ResNet-101 [69]	0.76	42.5 M	1.78
Inception v2 [70]	0.74	10 M	7.40
Inception v3 [72]	0.78	22 M	3.58
Resnet v2 [72]	0.80	54 M	1.48

Fig. 5. Resultados de cada arquitectura en el dataset ImageNet

En este paper, se aplicó un framework de Deep Learning para la detección de objetos y modelos de rastreo a objetos

marcados, YOLO v3 junto con un enfoque Deepsort (para el rastreo de objetos en movimiento usando técnicas como el filtro Kalman y el algoritmo Hungaro) por cajas delimitadoras que se utilizaron para calcular la norma L2 usado en la regresión de YOLO, con una representación vectorizada computacionalmente eficiente para la identificación de clusters de personas que no respetan el distanciamiento. Los modelos fueron tuneados para clasificación binaria con Inception v2 en una Nvidia GTX 1060 GPU, aplicados al dataset del repositorio open image dataset (OID). Las imágenes de personas fueron obtenidas por OIDv4 junto con sus anotaciones. El dataset se compone de 800 imágenes, filtradas para solo obtener las muestras verdaderas. Luego, se realizó un split 8:2 para training/testing. Se complementó el conjunto de testing con frames de las grabaciones de vigilancia del centro de la ciudad de Oxford. Estas grabaciones se usarían para simular el enfoque general para monitorear el distanciamiento. Para el faster RCNN, las imágenes fueron reducidas a P pixeles en el borde corto con 600 y 1024 para baja y alta resolución, mientras que en SSD y YOLO, fueron escaladas a P x P con un valor P de 416. Los resultados se pueden apreciar en las Figuras 06 y 07.

Model	TT (in sec.)	NoI	mAP	TL	FPS
Faster RCNN	9651	12135	0.969	0.02	3
SSD	2124	1200	0.691	0.22	10
YOLO v3	5659	7560	0.846	0.87	23

Fig. 6. Resultados de los modelos de comparación de objetos (Tiempo Total, número de iteraciones, mAP, Total Loss, fotogramas por segundo (FPS))

Durante el entrenamiento, el rendimiento de los modelos fue monitoreado usando el mAP junto con los indicadores loss de localización, clasificación y general para la detección de la persona. Se puede apreciar que el modelo YOLO v3 ha sido el que ha logrado mejores resultados pese a tener un menor mAP que faster RCNN y mayor que SSD, pero con mayores fotogramas por segundo (FPS) entre los dos. Este modelo entrenado se uso para monitorear el distanciamiento social en las cámaras de vigilancia, como se muestra en la figura 09. En este experimento se simuló lo que se conoce como ratio de violación, que viene a ser el cociente entre el número de personas contra el número de grupos detectados. Se puede mostrar que los ratios fueron de 3, 2, 2 y 2.33.

El artículo ha mostrado la importancia de aplicar las recientes tecnologías en modelos de detección y rastreo de objetos con ayuda de cajas delimitadoras que identifican al sujeto en tiempo real. Siendo el YOLO v3 el modelo más eficiente con un FPS y mAP balanceados. Es recomendable considerar un tuning adecuado para el campo de visión de la cámara. Siguiendo esta premisa, este paper nos ha dado una idea aún más centrada a la metodología a seguir y ver los modelos de detección que podemos implementar.

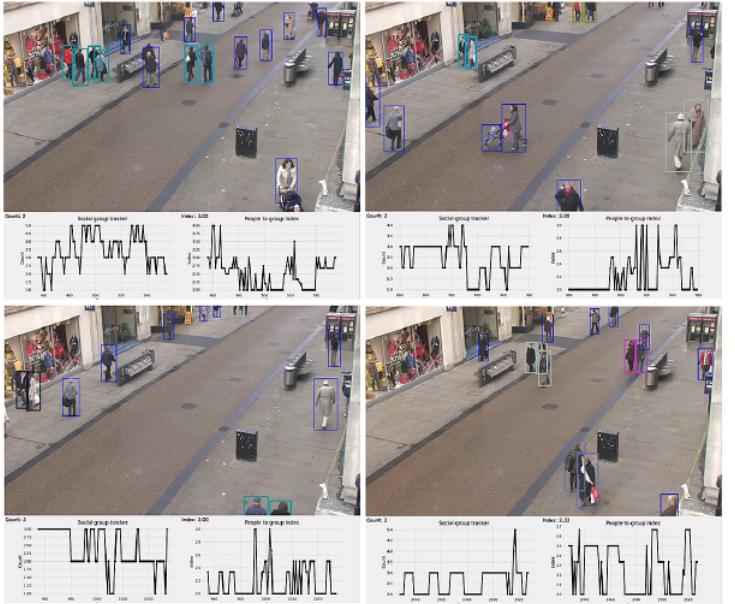


Fig. 7. Salida de prueba para el monitoreo del distanciamiento social en grabaciones de vigilancia del centro de la ciudad de Oxford

C. Convolutional Neural Network for Person and Car Detection using YOLO Framework

En la actualidad, la detección de objetos basado en la visión computacional es un tema de investigación muy importante, ya que las aplicaciones son diversas. Las aplicaciones mas conocidas son : sistemas avanzados de asistencia al conductor (ADAS), vehículos inteligentes y sistemas de vigilancia. Muchas de las técnicas de detección de objetos se enfocan mucho en la precisión de la detección a expensas del coste computacional. Por ello, estas soluciones con estas características no son aptas para la detección en tiempo real [5].

Un modelo usado ampliamente antes de la llegada del deep learning para la detección de objetos fue el modelo de pieza deformable (DPM). No obstante, los resultados de las redes convolucionales artificiales (CNN) incentivarón a ser aplicadas para la detección de objetos, logrando resultados impresionantes durante el proceso. Las nuevas soluciones se basaban en la detección por regiones, es decir, particionaban la imagen en cuadriculas y realizaban la clasificación en cada cuadricula con la finalidad de detectar el objeto a buscar. Algunas de estas soluciones como: regiones con redes neuronales convolucionales (R-CNN), Fast-CNN y Fast-RCNN obtuvieron excelentes resultados, no obstante fueron computacionalmente intensivos para la detección en tiempo real. Posteriormente, se analizó otra solución conocida como YOLO (You Only Look Once), que predice cuadros delimitadores y probabilidades de clase directamente de imágenes completas en una sola evaluación reduciendo el costo computacional de manera significativa.

Por ello, en este trabajo de investigación los autores proponen la aplicación del framework YOLO con ciertas

modificaciones, con la finalidad de poder reducir el costo computacional sin degradar mucho la precisión. La base de datos que se uso en este trabajo de investigación fue INRIA Dataset con la finalidad de entrenar el modelo. Además, se utilizó dos clases de objetos: personas y automóviles, con diferentes formas y colores. Las resoluciones de las imágenes son 640 x 480 o 480 x 640. Las imágenes se generaron de diferentes cámaras con fondo de las calles urbanas. Hay 162 imágenes de entrenamiento y 162 imágenes de prueba. En la Fig. 8 se puede ver una muestra de este dataset.



Fig. 8. Dataset generado de las imágenes de videocámaras

La metodología se basa en la arquitectura del YOLO, modificándola en la cantidad de convoluciones(7 convoluciones y 2 clases a predecir) como también el tamaño de las celdas(7x7 a 11x11). La arquitectura de YOLO consta de 27 capas, donde 24 son convolucionales, 2 son Fully-connected y la ultima es para la clasificación de objetos. Esta arquitectura divide la imagen en SxS celdas de cuadrícula y dentro de cada una de estas predice B cajas delimitadoras y obtiene una puntuación por cada C clase. Cada caja delimitadora consiste de 5 predicciones que son: centro x, centro y, ancho, alto y la puntuación de confianza de la caja delimitadora. Cabe destacar que para cada celda de la cuadrícula, solo habrá un conjunto de puntajes de clase C para todos los cuadros delimitadores en esa región y por ende, la salida de la red YOLO será un vector de números $S \times S \times (5B + C)$ para cada imagen. Las capas fully-connected usan las características extraídas de las capas convolucionales y usan la información para predecir las probabilidades del objeto y al mismo tiempo para las construcciones del cuadro delimitador. La ultima capa (de detección final) de YOLO es una regresión que mapea la salida de la última capa fully-connected para asignar el cuadro delimitador final y las clases. La red original de YOLO está capacitada en el conjunto de datos PASCAL VOC 2007 y PASCAL VOC 2012 y predice 20 clases de objetos con un tamaño de cuadrícula de 7x7. celdas. Además, predice cuadros delimitadores y probabilidades para cada celda de la cuadrícula.

En la Figura 9, podemos ver la arquitectura original de Yolo con sus respectivas características.

Cabe destacar que usaron un software para etiquetar llamado GUI PYTHON que generaba cuatro puntos de coordenadas (x_1, x_2, y_1, y_2) y la identificación de la clase. La información de las coordenadas se guarda en un archivo XML. Por otro

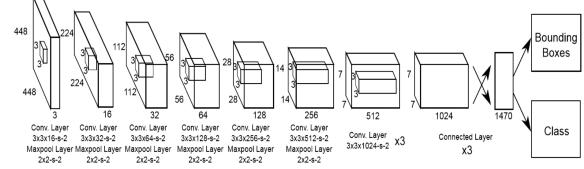


Fig. 9. Arquitectura original YOLO

lado, el entrenamiento para la red se acelera con el acelerador de GPU Nvidia K40, que es más rápido que la velocidad normal de entrenamiento de la CPU. Los archivos de los pesos se almacenan en un directorio de respaldo y pueden usarse como un punto de control si el entrenamiento necesita ser detenido. En la etapa de entrenamiento se cambia el tipo de la cuadrícula de celda: siendo 7x7, 9x9 y 11x11 para ver el performance de detección de personas y carros. Los resultados arrojaron que mientras más grande era la cuadrícula de celda, el accuracy del modelo de detección mejora, siendo en el caso del 7x7 muy difícil detectar carros pequeños. Para medir la predicción del sistema se uso la metodología de Hoiem para clasificar si cada predicción era correcta o estaba cometiendo un error como: correcto (clase correcta > 0.5), localización (clase correcta, $0.1 < \text{IOU} < 0.5$), similar (la clase es similar, $\text{IOU} > 0.1$), otro (la clase es incorrecta, $\text{IOU} > 0.1$), Fondo ($\text{IOU} < 0.1$ para cualquier objeto). Como resultado, el porcentaje de error en los tipos anteriormente mencionados es mayor en la arquitectura YOLO modificada que en la original y esto se debe a que se redujeron la cantidad de convoluciones.

Architecture	mAP	Person (AP)	Car (AP)	Frame per Second(fps)
YOLO	59.2	63.1	55.3	26
YOLO_7x7	37.9	42.3	33.5	35
YOLO_9x9	39.6	43.8	35.3	32
YOLO_11x11	41.1	44.1	38.2	30

Fig. 10. Resultados de las arquitecturas en términos de precisión promedio(mAP)

Los resultados que se pueden visualizar en la Figura 10, indican que la precisión de detección mejora a medida que aumenta el número de celdas de la cuadrícula. El uso de celdas de cuadrícula más grandes permite que el YOLO modificado mejore la detección de objetos pequeños. El mAP de YOLO_11x11 es mayor que YOLO_7x7 y YOLO_9x9 en 1.5% y 3.2% respectivamente. YOLO_11x11 con mAP de 41.1% es menor en comparación con el YOLO con mAP de 59.2% en 18.3%. En cuanto al rendimiento de la velocidad (fps), el tamaño de las celdas de la cuadrícula está afectando este rendimiento, ya que requiere más tiempo para procesar celdas de cuadrícula más grandes. En cuanto a la red YOLO_7x7, produjo la velocidad más alta; sin embargo, posee la precisión más baja en comparación con otra red. Por

lo tanto, el equilibrio entre velocidad y precisión se logra mediante la red YOLO_11x11 con 30 fps y 41.1 mAP. Y se concluye, por ende, que los modelos con fps más altos con mejor precisión permitirían integrar el modelo YOLO modificado en la implementación en tiempo real del sistema ADAS.

D. Object Detection and Counting with Low Quality Videos

La Visión Computacional es una tecnología que ha sido integrada a muchas áreas de tecnología como la robótica y la automatización, como también la clasificación de imágenes y detección de objetos. Sin embargo, existe un factor crítico a considerar cuando se vaya a utilizar esta tecnología, como es el caso de las alteraciones ambientales (viento, luz, lluvia, oscuridad, etc.) que pueden afectar la calidad de la imagen. Por lo que para la visión de robots, se desea un algoritmo de detección considerablemente robusto para tolerar varias reducciones de calidad de imagen causadas por el rápido movimiento de webcam o del target, lentes sin enfocar, falta de luz, etc., Los autores mencionan la difuminación Gaussiana como parte del desenfoque del lente. Entre muchas opciones para detección de objetos con CNN, Faster R-CNN y YOLO son las que mayor rendimiento brindan y requieren menos poder computacional con un buen accuracy.

En este paper, los autores proponen utilizar Faster R-CNN con una arquitectura VGG16 en el kernel ConvNet; y por otro lado utilizaron TinyYOLO, que tiene 9 capas convolucionales, 6 capas max pooling y 3 capas fully connected. Se usó un modelo pre-entrenado con VOC 2007 y 2012 para 40000 iteraciones. Con imágenes de baja calidad, se entrena el modelo pre-entrenado y luego se re-entrena con 500 a 1000 iteraciones con 128 imágenes por iteración. El re-entrenamiento de YOLO se logró dentro de Standford Terminal Cloud, mientras que para Faster R-CNN, AWS EC2 g2.2x para entrenamiento y una GPU GeForce GTX980 para testing.

Para las imágenes difuminadas, se usaron tres filtros en la data de entrenamiento: difuminación Gaussiana con 8 píxeles y desviación estándar de 2, Motion blur de 8 píxeles y orientación de 0° y pixelado por medio de series de filtros pooling de 2x2 hasta 8x8. Los autores crearon un agente de detección para recibir videos de entrada, definieron dos clases: human y car, cada clase con su contador según la detección en cada imagen del video de entrada. Los autores evaluaron el rendimiento con 8 videos diferentes, cuatro para cada clase. La Figura 11 demuestra capturas para cada video.



Fig. 11. Fotogramas de cada video

Para la fase de re-entrenamiento, se calcula la métrica mAP de cada categoría. La Figura 12 muestra el mAP de

YOLO después del re-entrenamiento. Se indica que el modelo original no tolera bien los tres filtros, por lo que luego del re-entrenamiento, el mAP incrementa para las otras categorías y para las originales se reduce. Se volvió a aplicar un re-entrenamiento para una cantidad diferente de pixelado utilizando en vez de mAP, una métrica accuracy indicando la clase correcta y un IOU por encima de 50% como se indica en la Figura 13.

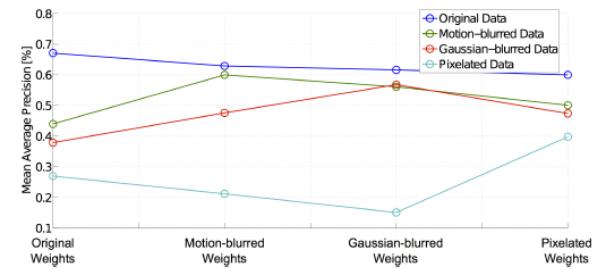


Fig. 12. mAP de YOLO para cada categoría

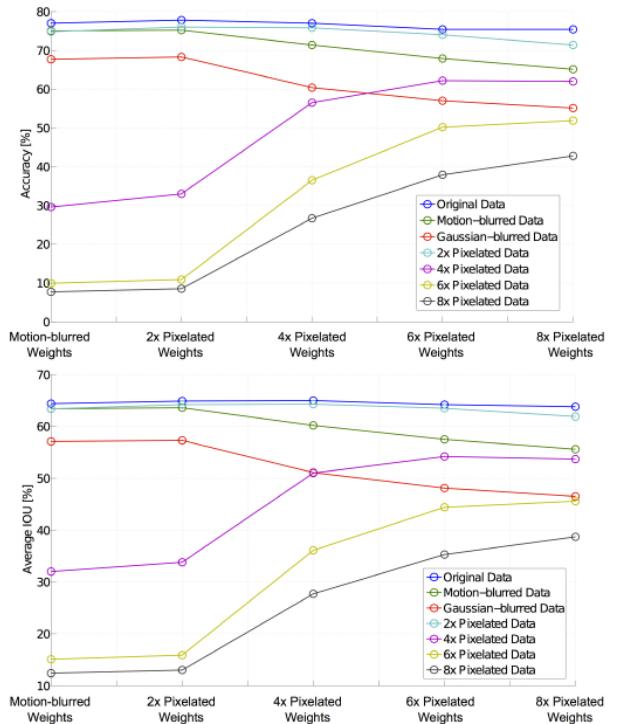


Fig. 13. Accuracy y IOU por encima del 50% para cada categoría

También se re-entrenó el modelo Faster R-CNN con las mismas cantidades de pixelado y diferentes categorías de difuminación como se muestra en la Figura 14, cada tipo de re-entrenamiento fue probado con otra categoría diferente.

Comparando el agente de detección con los modelos optimizados en imágenes, se comprobó que el modelo original no tolera bien las imágenes de alta calidad. Después del re-entrenamiento, el modelo puede detectar bien los objetos mostrando mejor robustez que el original como se indica en

	AP (Human)	AP (Car)	Mean AP
Retrain: None Test: Test: None	0.7062	0.8798	0.6852
Retrain: None Test: Motion Blur	0.5173	0.7257	0.4329
Retrain: Motion Blur Test: Motion Blur	0.6664	0.7384	0.6222
Retrain: Motion Blur Test: None	0.6643	0.8532	0.6428
Retrain: None Test: Gaussian	0.5627	0.7066	0.4723
Retrain: Gaussian Test: Gaussian	0.6235	0.7333	0.5876
Retrain: Gaussian Test: None	0.7160	0.8576	0.6935
Retrain: None Test: Pixelate	0.4950	0.5553	0.4404
Retrain: Pixelate Test: Pixelate	0.5878	0.8258	0.5511
Retrain: Pixelate Test: None	0.6613	0.8221	0.6527

Fig. 14. Re-entrenamiento de Faster R-CNN para cada categoría y testing

la Figura 15, mostrando la imagen de entrada y la salida de la 1ra, 9na y 15va capa neuronal.

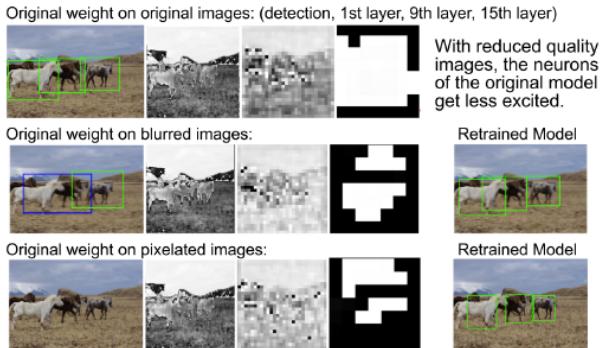


Fig. 15. Visualización para cada neurona

En cuanto a videos, los resultados de error se muestran en la Figura 16, mostrando un fuerte rendimiento en YOLO en videos con autos estacionados y de grupos de personas distanciadas cercanamente. En la Figura 17 se puede apreciar el error normalizado de cada categoría y, la linea verde representa agentes entrenados por separado por diferentes tipos de imágenes de baja calidad mientras que la azul es para agentes entrenados exitosamente por diferentes tipos y con un dataset de entrenamiento expandido.

Video Index	YOLO Original	YOLO Motion	YOLO Gaussian	YOLO Pixelation
1	0.711	0.759	0.766	0.714
2	0.669	0.682	0.674	0.650
3	0.690	0.686	0.666	0.637
4	0.420	0.425	0.452	0.411
5	1.390	1.374	1.400	1.408
6	0.796	0.982	0.988	0.929
7	0.955	0.955	0.955	0.954
8	0.939	0.909	0.937	0.912

Video Index	R-CNN Original	R-CNN Motion	R-CNN Gaussian	R-CNN Pixelation
1	0.508	0.562	0.546	0.540
2	0.558	0.541	0.576	0.587
3	0.578	0.554	0.535	0.559
4	0.623	0.566	0.534	0.577
5	1.324	1.000	0.903	0.941
6	0.358	0.246	0.377	0.241
7	0.847	0.792	0.713	0.731
8	0.949	0.892	0.807	0.753

Fig. 16. Resultados de cada modelo re-entrenado



Fig. 17. Error normalizado para cada categoría

El agente final se basa en un Faster R-CNN que ha sido re-entrenado sucesivamente en todos los métodos. La Figura 18 muestra varias de las fotografías de los videos utilizados. Este paper ha servido de aporte para este trabajo nos ayuda a tener un mejor enfoque de cómo trabaja YOLO con imágenes de baja resolución.

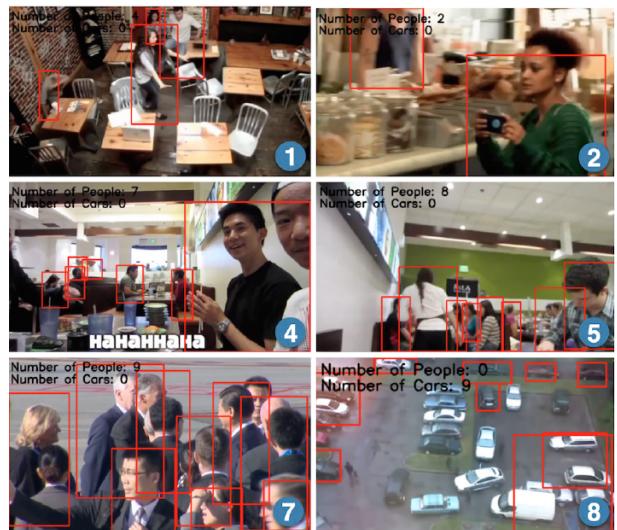


Fig. 18. Resultados probados para fotografías normales o con difuminación

III. METODOLOGÍA

La metodología propuesta en esta investigación tiene el siguiente flujo de trabajo: I) Recolección de data, II) Pre-Procesamiento, III) Procesamiento de los datos IV) Detectar y realizar conteo de personas, V) Calcular las distancias entre las personas detectadas y VI) detectar si se incumple con la distancia social mínima. Se puede observar estas etapas con mayor claridad en la Figura 19. Asimismo, se detallaron los algoritmos YOLOv2, YOLOv3 y YOLOv4 utilizados para la detección del objeto persona y las métricas empleadas para medir el rendimiento de los modelos.

A. Acercamiento - Algoritmos de Detección de Objetos

En esta investigación se utilizaron tres algoritmos de detección de objetos, los tres de la familia YOLO. Se tuvo como objetivo secundario encontrar cual de los tres, iniciando



Fig. 19. Metodología Propuesta de la Investigación [3]

en mismas condiciones, sería el que mejor logaría detectar personas en tiempo real.

You only look once (YOLO) es un sistema de alto nivel de detección de objetos en tiempo real. Asimismo, dentro de YOLO una sola CNN predice simultáneamente múltiples detecciones o bounding boxes y las probabilidades de clase respectivas, además de entrenar con las imágenes completas y directamente optimizar el rendimiento de la detección.

YOLOv2

En YOLOv2 se mejoran algunos de los errores de localización que se detectaron en el YOLO original que generaban bajo recall en la detección [11] comparado con clasificadores que trabajan con redes basadas en regiones como Faster R-CNN. Algunos de las mejoras que se añadieron a este modelo fueron la normalización de batches en cada capa de convolución, por medio del parámetro batch_normalize, para evitar el overfitting al eliminar el dropout. YOLOv2 realiza la clasificación inicial con una resolución de 448x448 y 10 epochs en el ImageNet. A diferencia del anterior, esta versión ya no utiliza capas fully connected, sino que ahora trabaja con una capa de pooling menos para trabajar con altas resoluciones en las capas de convolución y reducir el número de entradas de la red para tener una única celda en el centro y aparte utilizaron anchor boxes, reduciendo el mAP pero aumentando el recall. YOLOv2 trató de usar la idea de elegir k anchor boxes usando clustering k-means y encontrar buenos scores IOU. YOLOv2 predice coordenadas de ubicación relativas a la ubicación de la celda encasillada. Esta red predice 5 bounding boxes y predijeron las 5 coordenadas para cada bounding box, de esta manera permite que la parametrización sea más fácil de aprender. Con YOLOv2, se propuso un nuevo backbone

conocido como Darknet-19, con 19 capas convolucionales, 5 capas de maxpooling.

YOLOv3

En YOLOv3 se aplica una red neuronal a la imagen completa, esta red divide la imagen en regiones o grillas y predice los bounding boxes y probabilidades para cada región, estos bounding boxes están pesados por las probabilidades predichas [10]. Gracias a esto Yolov3 presenta varias ventajas sobre otros sistemas clasificadores, ya que al mirar la imagen completa en la prueba sus predicciones reciben información del contexto global de la imagen y, además, al evaluar con una sola red se realiza el proceso mucho más rápido que con otros clasificadores como Faster R-CNN. Lo que distingue YOLOv3 de sus antecesores es que se añadieron unos trucos para mejorar el entrenamiento e incrementar el rendimiento, incluyendo: predicciones multi-escala, un mejor clasificador base y otros detalles, estos se explicarán a continuación [10]. Esta nueva versión predice un puntaje de objetividad a cada bounding box usando regresión logística, de esta forma cada región de la grilla predice la clase que el bounding box pueda contener utilizando clasificación multi-etiqueta, no utilizan softmax porque los autores lo encontraron innecesario para mejor rendimiento [10], en vez de esto simplemente usan clasificadores logísticos independientes y durante entrenamiento utilizan binary cross-entropy loss para la predicción de clases. Esto se debe a que en el dataset puede haber etiquetas que se superponen y que el softmax se puede equivocar. Además de esto, YOLOv3 presentó un nuevo extractor de características, como se dijo anteriormente la base del modelo, el Darknet-53. Esta red para realizar la extracción de características es un acercamiento híbrido de la red usada en YOLOv2, Darknet-19 y partes de

redes residuales. La red utiliza capas convolucionales sucesivas de 3×3 y 1×1 , pero ahora tiene conexiones "shortcut" y es significativamente más grande, tiene 53 capas convolucionales [10]. En la sección de Procesamiento se explican las modificaciones a la arquitectura original que se realizaron para ajustarse a las características del problema presentado.

YOLOv4

Los autores Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao diseñan esta nueva versión de YOLOV4 [12] con el principal objetivo de diseñar un sistema con gran rapidez operacional para la detección de objetos y que este se encuentre optimizado para la computación en paralelo, en vez del teórico indicado de bajo volumen computacional (BFLOP).

Esta versión de YOLO trae grandes aportes como la de generar un detector de objetos poderoso y eficiente, que puede ser entrenado en una GPU 1080Ti o 2080Ti para entrenar un detector super rápido y preciso. Además, analizan la influencia de los métodos del estado del arte como los 'Bags of Freebies' y los 'Bags of Specials' durante el entrenamiento y, por último, modifican métodos del estado del arte como CBN, PAN, SAM para hacerlos más eficientes y entrenables en una GPU.

YOLOV4 consiste de 3 partes. La primera parte, el backbone, es el de CSPDARKNET53; la segunda parte, el cuello, está compuesta de SPP y PAN y la tercera parte, la cabeza es una YOLOV3. A lo largo de estas etapas YOLOV4 usa en el backbone los siguientes BoF: CutMix and Mosaic DataAugmentation, DropBlock regularization y Class Label Smoothing. Además, usa los siguientes BoS: Mish activation, Cross-stage partial connections (CSP) y Multi input Weighted residual connections (MiWRC).

Para el detector utiliza los siguientes BoF: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, eliminar grid sensitivity, usar multiple anchors para una verdad única, programador de recocido de coseno, hiperparámetros óptimos, formas de entrenamiento aleatorias; y para el BoS se usaron los siguientes: Mishactivation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS.

Con todas estas propuestas, y los experimentos que realizaron en el MS-COCO pudieron observar que su modelo se ubicaba en la parte óptima de la curva de pareto. Probaron en los GPU comúnmente usados Maxwell, Pascal y Volta. y en todos los experimentos obtuvo un buen AP (45-65) y a una buena cantidad de FPS 30+. Además, los modelos pueden ser entrenados en una GPU convencional de 8 a 16 GB-VRAM.

B. Recolección de la Data

Para el proceso de adquisición de data se recolectaron tres bases de datos: PASCAL VOC challenge 2007 [7], PASCAL VOC challenge 2012 [8] y COCO dataset 2014 [9], de donde por medio de un script en python se extrajeron solamente las imágenes que contenían por lo menos una instancia de objeto persona. De tal forma se obtuvieron un total de 38 102 imágenes, de las cuales se dividieron para entrenamiento

el 80% y para prueba el 20%, siguiendo la ley de Pareto. Asimismo, la cantidad total de instancias de personas dentro de la base de datos es de 127 232.

En la fase de prueba del módulo de conteo y detección de personas, se recolectó la base de datos de MOT Challenge 2019 [11], con 100 imágenes de la categoría 'person'.

C. Pre-Procesamiento

En esta investigación, el pre-procesamiento constó de tres etapas principales, que se detallaron a continuación. Se inició el proceso de pre-procesamiento convirtiendo las anotaciones del dataset COCO, que estaban en formato json, al formato de PASCAL VOC en xml. Al tener todas las anotaciones en un mismo formato se procedió a unificar ambos datasets en un solo lugar, una carpeta con solo las anotaciones y otra con las imágenes. Al ya tener las imágenes y etiquetas ordenadas, se utilizó un script en python que recibió las imágenes y las filtro, para de esta manera solo extraer las imágenes que tuvieran por lo menos una instancia de objeto persona. Después, estas imágenes filtradas fueron colocadas en una carpeta llamada "person" con sus respectivas anotaciones. De tal manera se almacenaron las imágenes de la clase persona, así se unificó la data que se utilizó para la detección de personas.

El segundo paso, constituye la etapa que se tuvo que hacer para preparar la data para el procesamiento por parte de los algoritmos de detección de objetos. En este caso, los algoritmos son de la familia Yolo y utilizan el API de Darknet, así que se tuvo que transformar la data a este formato. Para ello inicialmente se convirtieron todas las anotaciones xml de las imágenes al formato Yolo, el cual simplifica la información almacenada de las anotaciones al solo colocar 5 valores por fila, siendo el primero correspondiente a la clase y los 4 siguientes a las coordenadas y dimensiones de la etiqueta del objeto en la imagen, en otras palabras, el bounding box. En la última etapa se tuvo que preparar los archivos que requiere el API de YOLO para iniciar el procesamiento de la data y realizar el entrenamiento y prueba. Para esto se debe prepararon cinco archivos: obj.data, obj.names, obj, test.txt y train.txt. Siendo el obj.data el archivo principal que contiene los otros cuatro. El obj.names contiene los nombres de las clases que se quieren detectar, en este caso solo es una: "person"; El archivo obj es una carpeta que contiene todas las imágenes y sus correspondientes anotaciones en formato yolo; train.txt y test.txt contienen las rutas de las imágenes que pertenecen a cada una de las muestras y ambos archivos son generados por un script con la distribución 80/20; y en el obj.data se detallan la cantidad de clases, donde se va a almacenar el respaldo o backup de los pesos calculados durante el entrenamiento, contiene las rutas de donde está la data, que data se deberá utilizar para entrenamiento y cual para prueba, en sí es el archivo que va a utilizar Darknet como entrada para el modelo.

D. Procesamiento, Conteo de Personas y Cálculo de distancias entre Personas Detectadas

Para el entrenamiento de los modelos YOLO, se descargó en un entorno de Google Colab el repositorio de github Darknet, el Framework, que contiene los modelos pre-entrenados de YOLOv2, YOLOv3 y YOLOv4 que se usarán en este trabajo. Se construye el modelo usando Darknet y se descargan los pesos correspondientes a la versión de YOLO que se va a entrenar. Para el entrenamiento del YOLOv2, se realizaron cambios en la configuración de la estructura, en los siguientes puntos: se estableció el número de imágenes por cada paso de entrenamiento a 64, con 16 subdivisiones, con un tamaño de entrada de 608x608, tasa de aprendizaje de 0.001, max batches de 5200, con pasos entre 4160 y 4680 (80% y 90% de max batches), 23 capas convolucionales, 5 capas de maxpooling y una capa reorg. Para la última capa de convolución se ha modificado para tener 30 filtros, que se calculan a partir de la siguiente fórmula: Filtros = (n + 5)*5, siendo n de número de clases.

En el caso de YOLOv3, se utilizó su misma arquitectura con unos ajustes: el tamaño usado para las imágenes de entrada fue de 608x608, el número de lotes (batches) por iteración fue de 64 con subdivisiones de tamaño 16 y un total de 4000 batches (max batches), con steps de 3200 y 3600 con un learning rate de 0.001, el primer step es igual al 80% de los max batches y el segundo al 90%. Asimismo, se calcularon los filtros en las capas de convolución antes de las tres capas YOLO por medio de la fórmula: Filtros (n + 5)*3, dando un total de 18 filtros, igualmente se cambio el valor de la cantidad de clases a 1 en cada Capa YOLO. Teniendo un total de 75 capas de convolución, 28 capas de shortcut, route, upsample y 3 capas yolo.

Con YOLOv4, la arquitectura usada es la misma, lo que varía son algunos ajustes: El tamaño usado fue de 416x146, el número de lotes(batches) por iteración fue de 64 con subdivisiones de tamaño 16 y un total de 6000 batches (max batches), con steps de 4800 y 5400 con un learning rate de 0.0013, el primer step va a ser el 80% de los max batches y el segundo el 90%. Además, los filtros son calculados de igual manera que en YOLOv3: (Filtros = (n+5)*3; n= numero de clases). Lo que nos da 18 filtros. Se actualizó las clases para que sea 1(person) en cada capa YOLO. En conjunto, se cuentan con 75 capas convolucionales,3 capas yolo,5 capas de downsample, 23 shortcuts, 21 route.

Para el cálculo de los FPS promedio por video se utilizaron dos videos para comprobar la rapidez que tendrían los modelos en un entorno en tiempo real, para esto se utilizó el framework de Darknet en python. Los videos fueron: el de Oxford Town Centre, muy utilizado en este tipo de casos; y un video editado del centro de Gamarra en Perú, de esta manera se apuntó a medir el rendimiento en un entorno peruano.

Para el conteo de personas se implementó un script de Python utilizando la librería openCV para contar la cantidad de instancias de objetos persona detectadas en la escena y presentar el resultado en la imagen en la posición superior

izquierda.

Finalmente, para el cálculo de distancias entre los objeto persona detectados se implementó un script en python que sigue los siguientes pasos: Primero, utilizando la librería openCV, se calculan las coordenadas de los bounding boxes de las predicciones (altura, ancho, x e y) y los centroides de la imagen, que sería el píxel central del frame; después hay una etapa de comprobación que se asegura de solo continuar el proceso si existen detecciones dentro de la imagen o frame; cuando existan dos o más detecciones se va a calcular la distancia euclíadiana entre los centroides de cada una de las detecciones, la fórmula para calcular la distancia se observa en la ecuación 01.

$$d_{AB} = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2} \quad (1)$$

La distancia calculada se mide en píxeles. Si la distancia es menor al umbral, que por defecto se colocó 150 que en los videos de prueba utilizados se acerca a 1 metro, se va a contabilizar la violación de la distancia social y el color del bounding box de las instancias que no cumplen con la distancia mínima cambiará a rojo. Por último, de igual manera que a la hora de contar las personas, el resultado de cantidad de violaciones de la distancia social se mostrará en la esquina superior derecha.

E. Métricas

En esta investigación se utilizaron las siguientes métricas para medir el rendimiento a la hora de detectar personas: mean Average Precision (mAP), Precision , F1-Score e Intersection over Union (IoU) . Se utilizaron estas métricas por que son las que las buenas prácticas recomiendan para medir el rendimiento en este tipo de problemas , asimismo, demuestran su utilidad [3], [4], [5] y [6] a la hora de medir el rendimiento de los modelos de detección. Las ecuaciones de cada una de ellas se encuentran a continuación:

La precisión tiene la siguiente fórmula,

$$\text{Precision} = \frac{TP}{(TP + FP)} \quad (2)$$

Donde TP se refiere al número de verdaderos positivos y FP al de falsos positivos.

El mAP se representa en la ecuación 03.

$$mAP = \int_0^1 p(r)dr \quad (3)$$

Esta sería la integral sobre la precisión p(r).

El F1-score está representado por la siguiente ecuación .

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (4)$$

Donde el recall está representado por la fórmula 05.

$$\text{Recall} = \frac{TP}{(TP + FN)} \quad (5)$$

Por último, la fórmula de IoU.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (6)$$

En donde se puede entender "Area of Overlap" como el área de intersección entre el bounding box predicho con el bounding box real, el área que comparten, y "Area of Union" como la unión de estas dos áreas.

A la hora de medir el rendimiento del conteo de personas se utilizó el Error Absoluto Medio (MAE),

$$MAE = \frac{1}{N} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (7)$$

Error Cuadrado Medio (MSE)

$$MSE = \frac{1}{N} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (8)$$

y el Error Cuadrático Medio (RMSE), sus fórmulas se presentan a continuación.

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(y_i - \hat{y}_i)^2}{N}} \quad (9)$$

Donde, en las tres ecuaciones, N es el número de imágenes totales, y_i es el valor real y \hat{y}_i es el valor predicho.

IV. RESULTADOS

Los resultados correspondientes a la detección del objeto personas se pueden apreciar en la Tabla 01, donde se pueden ver respecto a las métricas mAP, Precisión, F1 Score e IoU promedio. En el caso de YOLO v2, se logró un mAP de 55.13%, luego de haber modificado el tamaño de entrada inicial de las imágenes de 416 a 608.

TABLE I
COMPARANDO RESULTADOS DE LOS MODELOS EN LA DETECCIÓN DE PERSONAS

MÉTODOS	mAP	Precision	F1 Score	IoU
YOLOv2	0.55	0.59	0.58	0.43
YOLOv3	0.60	0.84	0.58	0.65
YOLOv4	0.79	0.78	0.64	0.75

Se puede observar que en términos de mAP, F1-Score e IoU YOLOv4 resultó ser el mejor en detectar personas, siendo más robusto en ambientes con poca iluminación y con mayor aglomeración de personas, obteniendo un mAP de 0.79.

Los resultados que se obtuvieron con respecto a los FPS promedio obtenidos en cada modelo en cada uno de los dos videos de prueba que se usaron para probar el rendimiento en tiempo real de los métodos se puede observar en la Tabla 02.

TABLE II
FPS PROMEDIO OBTENIDOS POR CADA MODELO

MÉTODOS	OxfordTownCentre	Gamarra
	Avg FPS	Avg FPS
YOLOv2	19.1	39.3
YOLOv3	25.0	28.9
YOLOv4	17.7	39.4

En este caso se puede apreciar que la versión de YOLO que tuvo resultados más estables en ambos videos fue YOLOv3, mientras que el que tuvo el mejor resultado en el video de Gamarra fue YOLOv4. En este caso se puede ver que YOLOv4 tuvo mejor rendimiento en un ambiente con mayor instancias de personas, menos controlado y con mayor cambio de ángulos; mientras que YOLOv3 tuvo mejor resultado en un ambiente más controlado en el que no habían cambios tan abruptos en los ángulos de la cámara. Se puede ver que YOLOv3 rinde mejor en ambientes donde la cámara está estable en una sola posición (como sería el caso de una CCTV), mientras que YOLOv4 es más robusto y soporta cambios abruptos en las escenas.

Si bien YOLOv4 se mostró superior a la hora de detectar personas, no se ha podido implementar al módulo de conteo y detección del distanciamiento social. Debido a que se trabajó con la librería OpenCV en Python y todavía no se tiene habilitado el modulo para utilizar la estructura de YOLOv4. Por esta razón, se probó el módulo de conteo y detección con los métodos YOLOv2 y YOLOv3 usando 100 imágenes al azar del MOT Challenge 2017 [13], los resultados a la hora de realizar el conteo vs. la cantidad real de personas se pueden observar en la tabla 03.

TABLE III
COMPARANDO RESULTADOS EN EL RENDIMIENTO DEL CONTEO DE PERSONAS PARA CADA MÉTODO

MÉTODOS	MAE	MSE	RMSE
YOLOv2	6.21	56.95	7.55
YOLOv3	2.09	8.27	2.88

Podemos observar que YOLOv3 obtuvo mejores resultados a la hora de contar los objetos personas obteniendo un valor de MAE de 2.09, MSE de 8.27 y RMSE de 2.88, este valor de RMSE y MSE son un buen indicativo; dado que penalizan más cuando el error es grande y a la hora de contar personas es necesario mantener un error pequeño. Asimismo el MAE obtenido ha sido bueno para el caso de YOLOv3, mostrando que la diferencia entre el valor real y el predicho no es tan grande, igualmente hay posibilidad de mejora, optimizando el modelo.

En la detección del distanciamiento social se obtuvieron buenos resultados, dado que se pudo predecir correctamente que personas incumplían con el umbral mínimo establecido que era un aproximado de 1 metro. Los resultados de esta detección se pueden observar en la figura 20.



Fig. 20. Imágenes del resultado de la detección del distanciamiento social

De esta forma se puede ver que el módulo realizado para la detección de la distancia social logra detectar y contar

el número de incumplimientos de esta regla, cabe resaltar que este calculo se puede mejorar aun más si se cambia la perspectiva a una vista desde arriba. Dado que la vista actual en la que se realiza el conteo puede generar confusiones y detectar un incumplimiento cuando no lo hay.

V. CONCLUSIONES

En este trabajo se propuso una metodología para detección del cumplimiento del distanciamiento social y conteo de personas usando distintas versiones de YOLO, estas fueron reentrenadas y probadas para ver su desenvolvimiento. La mejor arquitectura en la detección de instancias de personas fue YOLOv4 con un mAP de 0.79, sin embargo no se pudo implementar por problemas de compatibilidad con el modulo desarrollado en OpenCV para la detección de la distancia social. Por ello, solo se pudo implementar los modelos YOLOv2, mAP de 0.55; y YOLOv3, mAP de 0.60, los cuales fueron comparados en su rendimiento en el conteo de personas bajo las métricas de MSE, RMSE y MAE, siendo YOLOv3 el método que presentó los mejores resultados obteniendo un MAE de 2.09 y un RMSE de 2.88. Esto demuestra que YOLOv3 pudo realizar el conteo de forma consistente cometiendo errores no tan grandes, dado que RMSE penaliza más si la diferencia entre el valor real y el predicho es mayor. Asimismo, es probable que las instancias de persona que no llegó a contar sean porque se encuentran en el fondo de las imágenes de prueba y por ello no se llegaron a detectar, esto significa que hay posibilidad de mejora, optimizando la configuración de YOLOv3.

En conclusión, se pudo observar que las distintas versiones de YOLO son bastante robustas para la detección de personas; con cada versión se ha afinado su capacidad de detección y se ha optimizado su desempeño para hacerlo en tiempo real. Es más, se pudo comprobar el rendimiento de los modelos en escenarios reales, como el del centro de Gamarra en Lima-Perú, donde se obtuvieron FPS promedio entre 28 y 39 FPS entre los tres modelos, siendo YOLOv4 el que obtuvo mejores resultados en un escenario con mayor número de aglomeraciones de personas, mientras que YOLOv3 resultó ser el método que obtuvo resultados más estables logrando 25 y 28 FPS promedio en los videos de prueba respectivamente. Esto demuestra que es factible implementar este sistema en cámaras de vigilancia CCTV para la detección a tiempo real del cumplimiento del distanciamiento social. YOLOv4 aún presenta algunos desperfectos que se pudo observar en los experimentos, por ejemplo cuando aparecían muchas instancias de personas aglomeradas, además de detectarlos independientemente, también los detectaba a todos en conjunto como una persona; esto se debe a que es una versión relativamente nueva y sigue en proceso de mejora y optimización.

De esta forma se pudo observar que la metodología propuesta demuestra que es posible brindar una solución a la detección de la distancia social entre personas por medio de modelos de detección de objetos, y al ser los modelos entrenados específicamente con imágenes de personas, esto brinda mayor robustez a la detección y conteo de estas.

Además, pudiendo ser implementado en tiempo real, brinda un apoyo a las autoridades correspondientes para el control del cumplimiento de las políticas establecidas.

VI. TRABAJO A FUTURO

Una segunda propuesta a probar sería la detección de personas en imágenes difuminadas o con baja iluminación, razón por la que YOLOv2 obtuvo un alto error. Para esta futura propuesta, se probarán diferentes filtros de imagen para evaluar si existe una variación porcentual en la detección y conteo.

Otro aspecto interesante a analizar sería, como ya se tiene la base del sistema, añadir reglas de cumplimiento de otras políticas, como la verificación y validación del uso de mascarillas. Además, implementar la detección de genero y edad de la persona, para así tener una mayor cantidad de datos y conocer el comportamiento de las faltas para así ofrecer soluciones más óptimas.

Asimismo, sería interesante probar otras arquitecturas para la detección de personas como Faster R-CNN [14] o la DetNet [15] para así poder comparar las distintas fortalezas y debilidades de estas arquitecturas en este problema.

REFERENCES

- [1] Mayo Clinic. 2020. Enfermedad del coronavirus 2019 (COVID-19) [online]. Disponible: <https://www.mayoclinic.org/es-es/diseases-conditions/coronavirus/symptoms-causes/syc-20479963>.
- [2] Diario El Peruano. 2020. Ordenanza que dicta medidas complementarias de distanciamiento social para prevenir el contagio y propagación del Coronavirus (COVID-19) en la jurisdicción del distrito de Ate: ORDENANZA N° 0529-MDA [Online]. Disponible: <https://busquedas.elperuano.pe/normaslegales/ordenanza-que-dicta-medidas-complementarias-de-distanciamien-ordenanza-n-0529-1866057-1/>.
- [3] Zhiqiang Li, La Zhang, Yikai Fang, Jinqiao Wang, Huazhong Xu, Baocai Yin, and Hanqing Lu. "Deep People Counting with Faster R-CNN and Correlation Tracking". In Proceedings of the International Conference on Internet Multimedia Computing and Service (ICIMCS'16). Association for Computing Machinery, New York, NY, USA, 2016, 57–60. DOI:<https://doi.org/10.1145/3007669.3007745>.
- [4] Narinder Singh Punn, Sanjay Kumar Sonbhadra and Sonali Agarwal. "Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deepsort techniques". Computer Vision and Pattern Recognition. Cornell University. arXiv:2005.01385v2 [cs.CV]
- [5] M. Putra, Z. Yussof, K. Lim, S. Salim. "Convolutional Neural Network for Person and Car Detection using YOLO Framework". Journal of Telecommunication, electronic and computer engineering, Vol. 10 no. 1-7 pp. 67-71, 2018.
- [6] H. Jiang and S. Wang, "Object Detection and Counting with Low Quality Videos", Final Report (Convolutional Neural Networks for Visual Recognition (CS 231n)), Stanford University, USA, 2016 [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/287_Report.pdf
- [7] The PASCAL Visual Object Classes, "PASCAL VOC 2007 Challenge" [online]. Available: <http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html>.
- [8] The PASCAL Visual Object Classes, "PASCAL VOC 2012 Challenge" [online]. Available: <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>.
- [9] COCO Dataset, "2014 validation data", 2020, [online]. Available: <https://cocodataset.org/download> .
- [10] J. Redmon Ali Farhadi, "YOLOv3: An Incremental Improvement", University of Washington, USA, 2018. Available: <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
- [11] J. Redmon Ali Farhadi, "YOLO9000: Better, Faster, Stronger", University of Washington, USA, 2016. Available: <https://pjreddie.com/media/files/papers/YOLO9000.pdf>

- [12] A. Bochkovskiy, C. Wang H. Mark,"YOLOV4: Optimal Speed and Accuracy of Object Detection", April 2020 [Online]. Available: <https://arxiv.org/pdf/2004.10934.pdf>
- [13] Multiple Object Tracking Benchmark. MOT17 Challenge. Available: <https://motchallenge.net/data/MOT17/>
- [14] S. Ren, K. He, R. Girshick J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", Jan 2016. Available: <https://arxiv.org/pdf/1506.01497.pdf>
- [15] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng J. Sun, "DetNet: A Backbone network for object detection", School of Software, Tsinghua University, China, 2018. Available: <https://arxiv.org/pdf/1804.06215.pdf>