Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Sistemas Computacionales

Análisis de videos de YouTube Kids implementando Visual Transformers

---

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
MAESTRO EN SISTEMAS COMPUTACIONALES

Presenta: **KARLA PAOLA GARCÍA LARIOS**

Asesor **VICTOR HUGO MARTÍNEZ SÁNCHEZ**

Tlaquepaque, Jalisco. diciembre de 2024.

AGRADECIMIENTOS

Se expresa un agradecimiento profundo a su prometido, sus padres, su director de trabajo de obtención de grado y su coordinador de posgrado.

Se manifiesta agradecimiento al CONAHCYT por otorgar su apoyo con la beca número 1239961.

Se extiende agradecimiento hacia el Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO) por la condonación de la colegiatura resultado de la beca CONAHCYT.

DEDICATORIA

Para Ana, gracias por ser parte del camino, aunque desafortunadamente no hayas llegado a ser parte de la meta, siempre en mi mente tengo presente el apoyo que me diste a lo largo de toda mi vida, por siempre creer en mí y sacarme adelante, esto es para ti.

A mis padres y prometido, su apoyo en este camino fue lo que me mantuvo fuerte y de pie, pues nunca me dejaron darme por vencida y se ocuparon de cosas que tenía que descuidar para poder seguir dando mi cien por ciento en este proceso de aprendizaje.

A mi coordinador de maestría y mi asesor de tesis, gracias por darme una segunda oportunidad y animarme para poder terminar con éxito esta etapa de mi vida.

A mí misma, Karla, gracias por todo lo que hiciste estos dos años, por no darte por vencida y seguir buscando el como si, porque solo tú sabes el sacrificio que hubo detrás.

RESUMEN

Según la investigación desarrollada en *The Common Sense Census: Media Use by Tweens and Teens* [1], el uso de tecnologías relacionados con medios digitales tuvo un aumento de diecisiete porcientos después de la pandemia por COVID-19 en los niños de ocho a doce años.

Para el año dos mil veintiuno, los niños pasaban alrededor de cinco horas y media por día consumiendo contenido de medios digitales, dedicando alrededor de cincuenta y siete minutos a videos de plataformas como YouTube [1].

En octubre del dos mil veinte tres, YouTube realizo cambios en sus lineamientos de seguridad; ahora, los creadores de contenido pueden continuar con su cuenta activa, sin importar los incumplimientos hacia las normas de la comunidad en el pasado, ya que, cada 90 días se vence este récord, siempre y cuando se haya tomado una capacitación sobre las políticas del sitio [2].

Además, desde el dos mil diecinueve, gracias a la ley definida en Children's Online Privacy Protection Act (COPPA) [3], los creadores de contenido deben declarar e identificar la audiencia a la que van dirigidos, y categorizar su contenido especificando si este es hecho para niños o no [4].

La problemática resulta cuando la categorización de contenido no es especificada por los creadores puesto que esta realiza automáticamente siguiendo ciertos lineamientos que no siempre ayudan a poderlas encasillar en el publico correcto, haciendo que contenido no apto para niños llegue a sus manos.

La investigación en curso tiene como objetivo implementar modelos preentrenados que utilicen Transformers para analizar contenido de YouTube que esté categorizado como hecho para niños.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE ACRÓNIMOS Y ABREVIATURAS

AI	Inteligencia Artificial/ Artificial Intelligence
API	Application Programming Interface
AVA	Atomic Visual Actions
BERT	Bidirectional Encoder Representation from Transformers
BGR	Blue Green Red
BRF	Balanced Random Forest
CNN	Redes Neuronales Convolucionales
COPPA	Children's Online Privacy Protection Act Rule
CSPNet	Cross Stage Partial Network
CSV	Comma Separated Values
D&T	Detect to Track and Track to Detect
DETR	Detection Transformer
DL	Deep Learning
DT	Decision Tree
GPT	Generative Pre-trained Transformer
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
JPU	Joint Pyramid Upsampling
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbor
LR	Logistic Regression
LLM	Large Language Models
LSTM	Long Short-Term Memory
mAP	mean Average Precision
ML	Machine Learning
MLP	Multi Layer Perceptron
MOE	Mixture of Experts
MOTA	Multiple Object Tracking Accuracy
NB	Naive Bayes
Open CV	Open Computer Vision
OVIS	Occluded Video Instance Segmentation
R-CNN / RCNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RF	Random Forest
RGB	Red Green Blue
RNN	Redes Neuronales Recurrentes/ Recurrent Neural Networks
RoI	Region of Interest
RPN	Region Proposal Network
SAE	Sociedad de Automóviles e Ingeniería
SGD	Stochastic Gadient Descent
SORT	Simple Online and Realtime Tracking
SSD	Single Shot MultiBox Detector
TNT	TrackletNetTracker
VATT	Video-Audio-Text Transformer
VGG-16	Visual Geometry Group de 16 capas
ViT	Vision Transformer

1 INTRODUCCIÓN

1.1. Antecedentes

Con anterioridad, se ha abordado la problemática de poder filtrar contenido audiovisual no apto para niños en plataformas que pretenden ser enfocadas para los mismos. La mayor parte de las veces, los filtros con los que cuentan estos sitios web o aplicaciones móviles no son suficientemente robustos para revisar el contenido que se publica diariamente de manera oportuna.

Esta problemática se ha atendido implementando algoritmos de Machine Learning (ML) en clasificación supervisada. También, se han propuesto enfoques de Deep Learning (DL) que para su análisis utilizan capturas de imágenes del video, información del creador de contenido y su canal, audio, y texto.

Lo que marca la diferencia entre investigaciones previas y el trabajo propuesto, es la manera en la que se aborda, así como el cambio de políticas en la plataforma de YouTube, en la cual estará enfocado todo este esfuerzo. La investigación desarrollada en los trabajos previos se centra en el análisis de la transcripción del audio del video más que en las imágenes que se perciben en el mismo.

Aunado a esto, algunas características tomadas para previos análisis ya no se encuentran disponibles en la plataforma, tales como, los comentarios y links externos en las descripciones de los videos, o los anuncios publicitarios dentro de YouTube Kids.

En este trabajo se busca implementar modelos preentrenados que involucren únicamente detección de objetos para, basados en la ley COPPA se pueda determinar si un video es apto o no para niños dependiendo de la clasificación de los objetos detectados.

1.2. Justificación

El aumento en el consumo de videos en línea por parte de niños entre ocho y once años llevó a la necesidad de explorar modelos de Inteligencia Artificial (AI) que limiten el acceso a el contenido audiovisual de riesgo.

El contenido inapropiado sigue estando al alcance de todos, a pesar de los esfuerzos como la ley COPPA en Estados Unidos, que obliga a los creadores de contenido de YouTube a declarar si los videos que realizan son hechos o no para niños, así como los intentos de YouTube por clasificar y filtrar contenido inapropiado.

Buscar opciones creativas que permitan salvaguardar a los niños de la red es algo que debe estar en desarrollo e investigación constante puesto que, las políticas y normas de comunidad, las innovaciones, mejoras y características dentro de YouTube están en constante actualización y cambio.

Por lo tanto, la información considerada anteriormente para poder determinar si un contenido es apto o no puede estar hoy en día obsoleta. Poner sobre la mesa nuevas formas de abordar el problema permite que se tomen en cuenta nuevos factores que ayuden a determinar si el contenido es o no apropiado.

1.3. Problema

El acceso que tienen los niños a videos en la red a través de YouTube Kids continúa siendo peligroso, puesto que los esfuerzos implementados para el filtrado de este no han sido del todo exitosos. En esta plataforma, que es *"exclusiva para niños*, seguimos encontrando material audiovisual inapropiado a su alcance.

Aunado a esto, existen herramientas que permiten seleccionar contenido a mano para los pequeños, ya sea por canales o videos en específico. Sin embargo, esto continua sin ser una solución para la problemática planteada, ya que el padre del menor tendría que invertir tiempo en decidir que catálogo de contenido puede o no ver su niño y se caería en la disyuntiva de que, algunos canales, no todos sus videos van a cumplir con los lineamientos de seguridad.

Adicionalmente, el contenido y los lineamientos de comunidad cambian periódicamente, por lo que se requiere reevaluar las propuestas de mejora de filtrado anteriores, pues las capacidades e información que se tomaron en ese momento pueden estar hoy obsoletas. Asimismo, considerando la forma en la que se distribuye la información en la actualidad, las tendencias se disparan de manera exponencial y no siempre resultan en acciones que sean sanas para los niños. Por lo tanto, acciones que antes se consideraban "seguras" dentro de la clasificación de contenido pueden no serlo más.

1.4. Objetivos

1.4.1 Objetivo General:

Se busca evaluar contenido categorizado como *"hecho para niños"* utilizando Visual Transformers para la detección de objetos en video y basar los criterios de pasa o falla tomando como referencia las nuevas políticas de comunidad de YouTube y el desglose de los lineamientos definidos en la ley COPPA para evaluar si el contenido esta categorizado de manera correcta.

1.4.2 Objetivos Específicos:

1.4.2.1 Evaluar conjuntos de datos que se hayan implementado en investigaciones previas con análisis de audio a texto para comparar los resultados al analizar el mismo contenido etiquetado como hecho para niños implementando Visual Transformers y poder comparar los resultados entre sí.

1.4.2.2 Basar los criterios de pasa o falla en la ley COPPA y de acuerdo con la misma, poder determinar si el objeto detectado en el video es de riesgo o no y así poder clasificar el contenido de manera adecuada.

1.5. Novedad científica, tecnológica o aportación

La preocupación por el material audiovisual expuesto en plataformas como YouTube Kids y su investigación no es algo nuevo, puesto que, a lo largo del tiempo se han planteado distintas alternativas para mejorar su clasificación enfocadas en modelos de ML considerando texto, video en análisis no secuencial, comentarios, descripción, me gusta, links, entre otras características, ya sea en el contexto de evaluación por canal o por videos independientes.

Sin embargo, el acceso características de los videos que se implementaban previamente en soluciones para la problemática, por ejemplo, la sección de comentarios, ya no se encuentran disponibles dentro de la plataforma YouTube Kids.

Otra novedad dentro de este tipo de contenido es la implementación de la Ley COPPA, la cual ayuda a tener un lineamiento más general y bien delimitado sobre que si puede ser considerado contenido apto para menores de edad.

Es por ello por lo que, la novedad de este trabajo se centra en la detección de objetos dentro del video con el uso de Visual Transformers y su delimitación sobre los mismos por medio de la ley COPPA, así como también, su comparación con estudios anteriores que basan su análisis únicamente en texto para poder concluir si el análisis de este contenido de manera visual marca la diferencia en la clasificación de contenido de riesgo.

2 ESTADO DEL ARTE O DE LA TÉCNICA

2.1 Técnicas basadas en Redes Neuronales Convolucionales (CNN) para la Detección de Objetos en Videos

2.1.1 Trabajos relevantes en CNNs

Sin duda, You Only Looks One (YOLO) [Figura 1], vino a revolucionar los modelos de identificación de objetos, pues, aplicando una red neuronal, y realizando la evaluación del modelo utilizando los pixeles de la imagen, es capaz de dar resultados robustos. Su funcionamiento se divide en 4 etapas importantes:

1. Cuadriculado de la imagen: Esta se particiona en cuadriculas de S x S, siendo S definida por el usuario.
2. Cuadros delimitadores: Cada uno de los cuadriculados S x S en la imagen buscarán un objeto centrado en ellos y de encontrar alguno, se definirán los cuadros delimitadores.
3. Probabilidad de clase: Asumiendo que, en los cuadros delimitadores si se encuentra un objeto a detectar, se calcula la probabilidad para encontrar la clase del objeto presente.
4. Resultados: Los resultados de los cuadros delimitadores y la probabilidad de clase se combinan para dar como resultado las coordenadas exactas y la clase del objeto respectivamente.

Lo que hace diferente a YOLO de implementaciones previas, es su rapidez, puesto que, no requiere de una matriz externa que se desplace por la imagen de entrada para calcular las probabilidades de búsqueda del objeto, si no que, únicamente divide la imagen una única ocasión y la procesa una vez en la red neuronal.

Su arquitectura está compuesta de veinticuatro capas de convolucionales, a las cuales le siguen dos capas completamente conectadas [5]

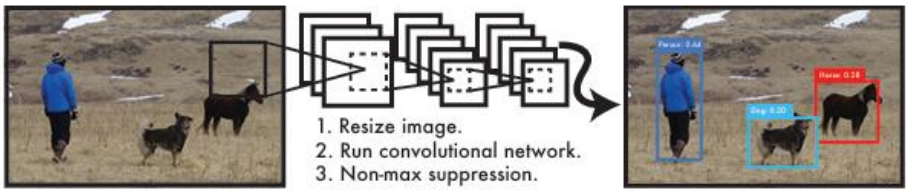


Figura 1. YOLO y su Arquitectura a alto nivel. Recuperado de [5].

Actualmente, esta arquitectura sirve de base para innovaciones las cuales generan nuevas versiones del modelo original que brindan mejores resultados. YOLO v3 [6], por ejemplo, en su implementación introdujo mejoras para detectar objetos grandes y diminutos.

En el caso de YOLO v4 [7] con la implementación de técnicas de data augmentation, logro mejorar significativamente la velocidad y precisión del modelo.

En cuanto a arquitectura, YOLO v3 está compuesto de Darknet-53, el cual se define por tener 53 capas convolucionales con activaciones Leaky de unidad lineal rectificada (ReLU), y utiliza bloques residuales con la intención de poder entrenar redes de manera más profunda y sin perder precisión.

Por otro lado, YOLO v4 utiliza una arquitectura Cross Stage Partial Network (CSPNet), la cual, reduce el costo computacional dividiendo el flujo de características en dos partes para después fusionarlas e implementa data augmentation para mejorar la precisión.

Otra Arquitectura que también tiene como base una CNN es Fast R-CNN [Figura 2] [8], la cual, a diferencia de YOLO, no define una matriz de S x S para pasar por cada una de las cuadriculas definidas y generar sus cuadros delimitadores, si no que, propone regiones donde pueden encontrarse objetos.

En Fast R-CNN, la imagen de entrada debe pasar por las capas de CNN, las cuales, pueden ser de tipo Visual Geometry Group de 16 capas (VGG-16) o Residual Neural Network (ResNet), produciendo un mapa de atributos.

El mapa de características se logra gracias a los filtros convolucionales, puesto que, al pasar el filtro por la imagen, se aprenden aspectos relevantes de la misma y ajustan su valor de perdida gracias al backpropagation.

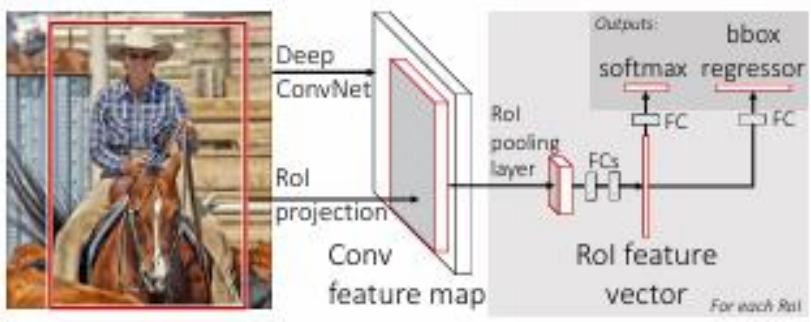


Figura 2. Arquitectura y ejemplo de Fast R-CNN. Recuperado de: [8].

En el siguiente paso, se proponen regiones en las cuales puedan encontrarse objetos y se aplicarán al mapa de características. Esto se logra gracias a la implementación del selective search, un algoritmo que agrupa pixeles similares en la imagen, en regiones que parecen objetos y no el fondo.

Con la información obtenida del proceso anterior, se pasa por el Region of Interest (RoI) Pooling el cual, se encarga de crear un vector de características con dimensiones fijas, para que cada región cuente con una representación uniforme.

Finalmente, se pasa por capas completamente conectadas, en las cuales, se realizan las predicciones de la clasificación del objeto en la región propuesta y se obtienen los puntos de referencia de los cuadros que delimitan el objeto.

El modelo Single Shot MultiBox Detector (SSD) [9], también está basado en CNN. En este se busca generar predicciones a múltiples escalas aplicando filtros convolucionales a mapas de características de distintos tamaños, siendo una buena opción para aplicaciones que requieran ser usadas en tiempo real.

La arquitectura de esta red se compone de una base de VGG-16, seguida por capas adicionales convolucionales que van decreciendo en tamaño; esto con la intención de ser capaces de predecir objetos

a distintas escalas, puesto que, cada una cubre áreas distintas facilitando la detección de elementos en tamaños variados, los cuales van desde grandes hasta pequeños.

De ahí, pasamos a la etapa de los Anchor Boxes, estos nos permiten definir los cuadros delimitadores desde cero, puesto que, a cada celda en el mapa de características, se le asignan anchor boxes de distintas escalas y proporciones. Estos anchor boxes sirven como punto de partida para ajustar y predecir los cuadros que delimitan el objeto con sus respectivas clases.

Después, gracias a la métrica Intersection over Union (IoU), se logra el ajuste de los cuadros delimitadores puesto que, mide la intersección entre los anchor boxes ya modificados y los cuadros correspondientes a los objetos etiquetados. De esta manera, el modelo realiza la clasificación del objeto usando softmax y ajusta los offsets del cuadro delimitador, basándose en el anchor box que mejor coincide con el objeto, según la métrica de IoU [Figura 3].

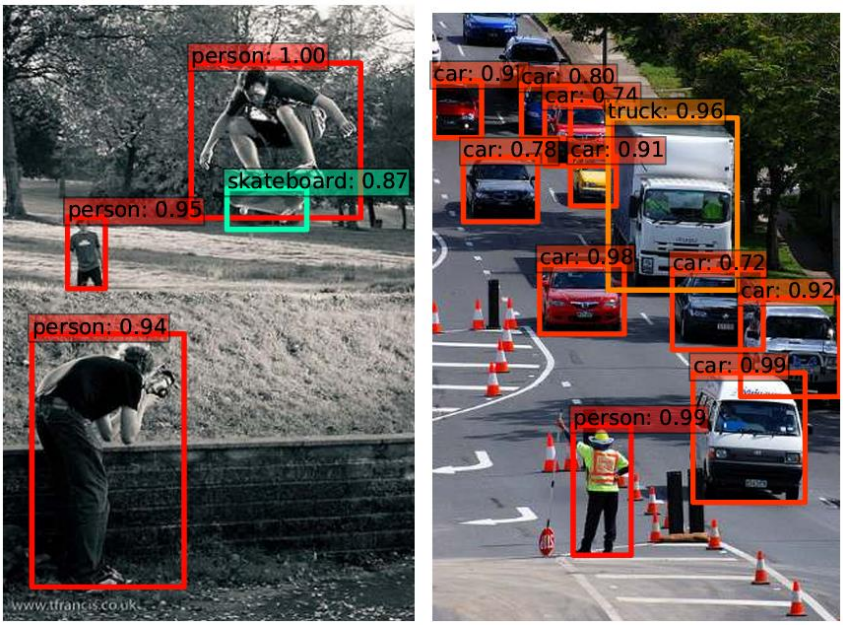


Figura 3. Resultados de un SSD preentrenado con la colección de datos COCO, aplicado en tareas de identificación de objetos. Recuperado de: [9].

Otro modelo que utiliza CNN con un propósito distinto a los previamente mencionados es el Detect to Track and Track to Detect (D&T) [10]. La diferencia entre D&T y los mencionados previamente es que este modelo además de detectar objetos busca su seguimiento en la secuencia de fotogramas.

La arquitectura del D&T se basa en un detector, el cual, puede ser un Faster R-CNN o SSD. También, contiene un modelo de seguimiento, el cual, se basa en el conocimiento previo y actual para conseguir una secuencia del objeto, siguiendo el rastro de este. Por último, pasa a la etapa de perdida conjunta, en la cual, se combina la perdida generada de detección y de seguimiento.

2.1.2 Aplicaciones Recientes

Entre sus aplicaciones recientes se puede mencionar el uso de CNN para detectar los cambios en escenas en videos disponibles públicamente en plataformas como YouTube, tratando de simular Google Street-View, con la intención de detectar cambios estructurales en espacios públicos a lo largo del tiempo.

En esta aplicación, las CNN fueron utilizadas para obtener las características de las imágenes capturadas del video, logrando así, encontrar patrones de las secuencias de estas e identificar cambios abruptos o graduales de espacios urbanos.

Los retos que se enfrentaron en esta investigación fueron causados por la variabilidad encontrada dentro de los sets de datos, puesto que, existían variaciones en el ambiente, oclusiones y perdida de información. Esto se debió a que los espacios públicos fueron documentados en cortos periodos de tiempo, lo que generaba perdida de la resolución de la imagen y, a su vez, producía la necesidad de contar con gran precisión puesto que, los cambios eran tan pequeños entre fotogramas que identificarlos requería de detalle [11].

Las redes deconvolucionales, se utilizan para aumentar la resolución de las imágenes, por lo que, estas fueron implementadas con la intención de mitigar la problemática de la variabilidad en la calidad de las imágenes en la colección de datos de entrada.

En lo que respecta a la variabilidad y oclusión, se implementaron múltiples conjuntos de datos que capturaban los espacios en distintas condiciones ambientales. Por último, para mejorar la documentación de muestras de los espacios a lo largo del tiempo, se utilizaron distintos puntos de tiempo de captura de imágenes en el video.

2.2 Uso de Transformers para la Detección de Objetos en Video

2.2.1 Transformers en visión por computadora

Video Swin Transformer [12], logro una mejora en la implementación de los Transformes en el ámbito de la visión computacional, ya que, su arquitectura logro minimizar el consumo computacional y emplearse en actividades relacionadas con imágenes y video.

La arquitectura del Video Swim Transformer [Figura 4], cuenta con tres diferentes etapas, la primera de ellas descompone la imagen en ventanas no superpuestas de dimensiones fijas, la cual, se determina de acuerdo con la tarea que se busque resolver.

Por un lado, utilizar ventanas de tamaño pequeño, ayuda a capturan mejor los detalles locales, siendo ideales para tareas de precisión. Por otro lado, en ventanas de tamaño grandes, se logra captura el contexto global de la imagen.

Además de tomar en cuenta la tarea a resolver, para definir el tamaño de las ventanas, se debe considerar la resolución con la que cuenten las imágenes destinadas para el proceso de entrenamiento del modelo y las limitaciones computacionales.

Para concluir esta fase, se aplica la autoatención dentro de cada ventana. Este proceso consiste en un cálculo en cual, cada píxel de la ventana se relaciona con los demás y se le asigna un valor, el cual, es directamente proporcional a la similitud entre ellos, a mayor similitud, mayor valor de autoatención asignado.

La segunda fase del Video Swin Transformer, desplaza las ventanas, para que, regiones no continuas logren interactuar entre sí. La autoatención, es calculada entre cada ventana desplazada [Figura 5].

La tercera fase es una reducción progresiva de la resolución de las características. Al pasar por varias iteraciones en la segunda fase, cada ventana adquiere características más complejas y abstractas. Para poder reducir la complejidad computacional y seguir manteniendo relevante las características extraídas, se reduce la resolución de las ventanas en cada etapa, generando que cuenten con menor resolución, pero con características de importancia, logrando capturar detalles locales e información global.

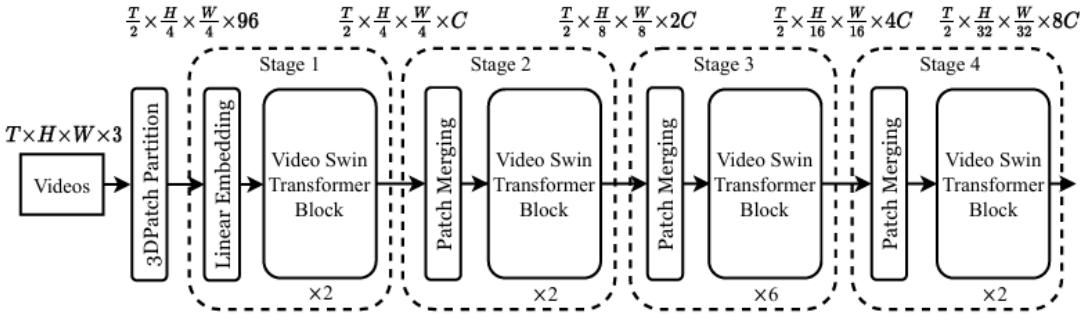


Figura 4. Arquitectura del Video Swin Transformer. Recuperado de: [12].

El Vision Transformer (ViT) [13], representa otra aplicación de Transformers para procedimientos de visión por computadora, específicamente en procesos que involucren la clasificación de imágenes. Dentro de su arquitectura podemos identificar cuatro fases importantes para su implementación.

La primera fase, se caracteriza por la definición del tamaño de los parches (patches), que, a diferencia de las ventanas que se utilizan en el Video Swin Transformer, no se asigna un valor de autoatención para cada división de la imagen.

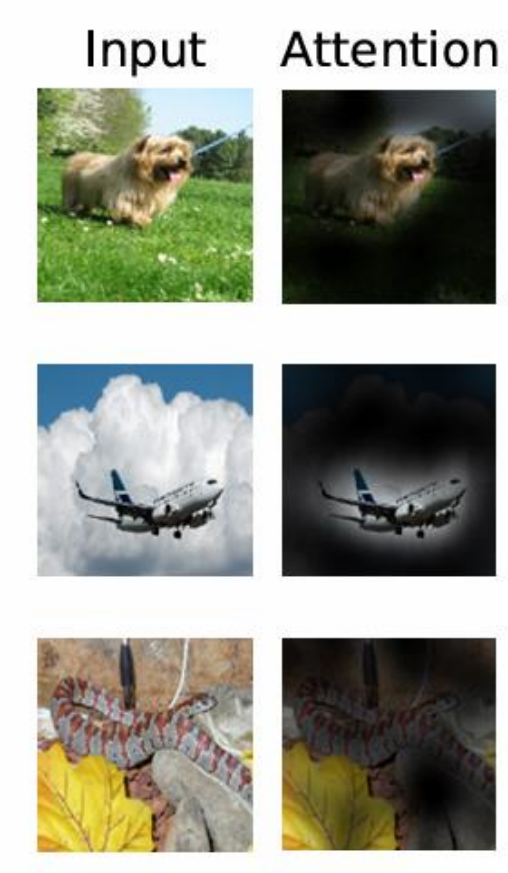


Figura 5. Ejemplo representativo de autoatención. Recuperado de: [13].

La segunda fase del ViT consiste en los embedding posicionales, los cuales asignan una ubicación absoluta de cada parche en la imagen en forma de vectores. Estos embeddings posicionales se suman a los embeddings de los parches para que el modelo pueda tener en cuenta tanto el contenido visual como la posición de cada parche en la imagen.

Esta información es crucial para que el modelo pueda calcular de manera efectiva la autoatención global en la tercera fase, permitiendo que cada parche interactúe con otros, considerando tanto el contenido como su ubicación dentro del contexto general de la imagen.

En la tercera fase, además de la capa de autoatención, se agrega una capa de feed fordward, esto con la intención de hacer una mejora en las características de cada parche al aplicar transformaciones no lineales.

Al final del proceso, el token de clasificación recoge la información global y esta se utiliza para hacer la predicción de la clase de la imagen.

Por sí mismo, ViT no es capaz de realizar detección de objetos, ya que su arquitectura original está orientada principalmente a tareas de clasificación de imágenes.

Sin embargo, al modificar la arquitectura del ViT integrando elementos como: CNN para la extracción de características y un codificador-decodificador, se logra transformar en un modelo llamado Detection

Transformer (DETR) [14] [Figura 6], el cual, es apto para la detección de objetos, lo abordaremos con mayor profundidad en la siguiente sección.

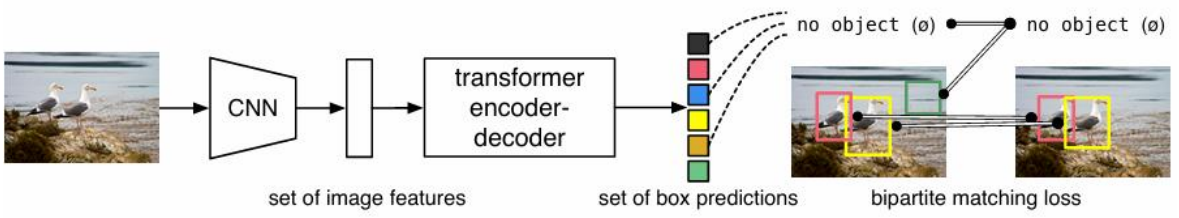


Figura 6. Arquitectura de un DETR en alto nivel. Recuperado de: [14].

El TimeSformer [15] es otro modelo diseñado para la identificación de elementos en video mediante el uso de Transformers. Este divide el video en cuadros de imágenes, tomados cada cierto intervalo de tiempo llamados fotogramas.

Los fotogramas se dividen en parches de una manera similar a la implementada en el ViT. Para procesarlos, se utilizan dos arquitecturas distintas. La primera es la atención espacial, la cual, ayuda a entender la relación en un solo cuadro de video. La segunda es la atención temporal, la cual, captura la relación entre cuadros de video a través del tiempo.

La manera en la que el modelo es definido permite aprender contenido visual y movimientos dinámicos, haciéndolo utilizable para clasificación de elementos en videos o reconocimiento de acciones.

2.2.2 Comparaciones entre CNNs y Transformers en videos

Para poder comparar y entender las diferencias entre CNN y Transformers, se debe abordar desde distintas perspectivas, comenzando con la complejidad arquitectónica, en las CNN, es necesario el uso de componentes adicionales, tales como, anchor boxes, que ayuden a ajustar los cuadros de delimitadores en las imágenes durante la detección de objetos.

Por otro lado, cuando se implementa el uso de transformers, se elimina la necesidad de estos componentes adicionales, ya que, en lugar de depender de una red convolucional para ajustar los cuadros delimitadores, estos implementan la autoatención, la cual, trabajando directamente con los parches o las ventanas, se encarga de aprender sobre las relaciones globales entre ellas, simplificando el proceso pues se remueve el uso de pasos adicionales.

En el caso de captura de contexto global, los Transformers para detección de objetos en videos lo hacen de manera natural, puesto que, desde la definición de su arquitectura, se debe comparar cada parche o ventana entre sí para obtener el cálculo de autoatención. Por otro lado, hablando de las CNN, para poder capturar contexto global requieren arquitecturas que permitan la comparativa entre sus regiones, lo que las hace más profundas y complejas.

Hablando de recursos computacionales, si se emplean Transformers, el incremento de estos puede crecer exponencialmente de manera rápida. El hecho de agregar más fotogramas al procesamiento de cada video

en el entrenamiento, le añade un mayor costo computacional. En el caso de las CNN, para poder procesar video requieren arquitecturas complejas con algo de coste computacional pues necesitan procesar cada cuadro secuencialmente, sin embargo, el coste no incrementa de manera tan rápida en comparación a los Tranformers [14].

Lo que se concluye es que, dependiendo que tan complejo sea el problema y la tarea que se busque realizar, es el procedimiento para seguir.

Si lo que se busca es un análisis detallado en pocas imágenes, la opción que se podría tomar es la de CNN ya que, es bueno analizando detalles pequeños en las mismas.

Ahora bien, si lo que se desea es realizar análisis de video de una manera más rápida, efectiva y se puede limitar el número de cuadros de imágenes en los videos de interés a uno que genere resultados robustos, Transformers sería una buena elección.

2.3 Datasets Utilizados en la Detección de Objetos en Videos

YouTube-8M [16], es un repositorio de videos extraídos de YouTube, que, como su mismo nombre lo anuncia, cuenta con más de ocho millones de videos organizados en cuatro mil ochocientas clases en formato multiclase, lo que quiere decir que, un solo video puede pertenecer a varias categorías simultáneamente. Fue desarrollado por el equipo de Google Research y se encuentra distribuido con un setenta por ciento para entrenamiento, veinte por ciento para validación y diez por ciento para prueba.

Los videos que se encuentran dentro de YouTube-8M, no están embebidos, ya que, procesar esa cantidad de contenido audiovisual no sería eficiente. Por ello, se encuentran representados por medio de características pre-extraidas implementando el uso de Inception-V3.

Inception-V3 es una CNN preentrenada utilizada para extraer características visuales en los cuadros del video, los cuales, se seleccionan y se convierten en vectores de características que condensan la información visual más relevante del video. La ventaja que ofrece este enfoque es la reducción de la carga computacional, el tamaño de almacenamiento y la representación compacta y efectiva de todos los videos contenidos en el repositorio.

Figura 7. Nube de palabras que representa el top 200 de entidades de YouTube-8M. Recuperado de: [16].

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [17], es otro conjunto de datos que puede ser implementado para el reconocimiento de objetos en imágenes, que también puede ejecutarse en video si este es dividido en fotogramas.

ILSVRC es un subconjunto de ImageNet, el cual, contiene únicamente mil categorías, una cantidad reducida en comparación al conjunto completo de ImageNet, la cual, contiene veinte mil de estas.

En las mil categorías de ILSVRC se incluyen objetos visuales comunes que provienen de diversas fuentes tales como: sitios web, bases de datos públicas o etiquetados manuales. Estás categorías fueron seleccionadas a mano, con la intención de que fuesen representativas, diversas y desafiantes.

El total de imágenes que contiene este subconjunto es de uno punto dos millones, cada una de ellas etiquetada de manera uniclase, lo que significa que contienen una sola etiqueta, sin importar que contenga más de un objeto ya que se clasifica de acuerdo al que tenga mayor prominencia.

Por otro lado, si lo que se busca es detectar acciones breves y localizadas en el tiempo (secuencia temporal del video en la que ocurre la acción) y espacio (localización de la acción en la secuencia de video) de un video, el Dataset que puede implementarse es Atomic Visual Actions (AVA) [18].

AVA se compone de cincuenta y siete mil seiscientos videos de 15 minutos tomados de películas y están etiquetadas a mano con la localización precisa de las acciones, las cuales están representadas por ochenta clases.

Esto quiere decir que cada video de quince minutos tiene subconjuntos de etiquetas las cuales, muestran en que fragmento del video se comente una acción atómica e implementa cuadros delimitadores para mostrarlas visualmente, por lo que, estas etiquetas son multiclase, pues, un mismo video en sí mismo está representado por varias categorías a la vez.

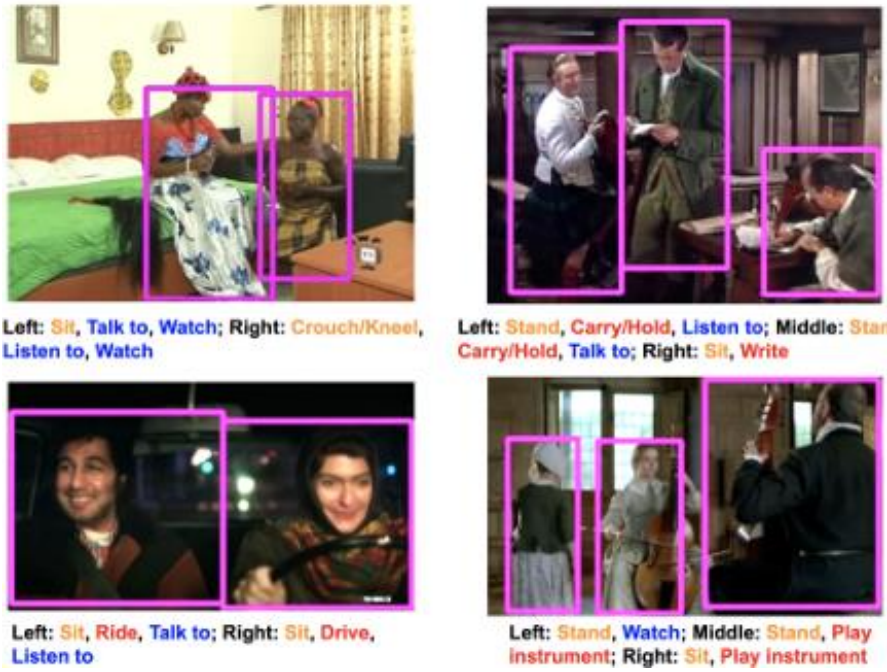


Figura 8. Colección de datos AVA y la ejemplificación de sus anotaciones. Recuperado de: [18].

El propósito de cada uno de estos conjuntos de datos es proporcionar una base neutral para la comparación objetiva de distintos modelos, con el fin de verificar si la implementación realizada genera un impacto significativo en la detección de objetos.

Estos ofrecen a los expertos, analizar el rendimiento de sus modelos y compararlos de manera directa con implementaciones previas. Además, proporcionan un marco estándar para medir mejoras en relación con la exactitud, la eficacia y la adaptabilidad escalable de los modelos.

2.4 Desafíos y Avances Recientes en la Detección de Objetos en Video

Aplicar CNN en problemas de segmentación semántica se vuelve retador, puesto que, se necesita capturar los detalles finos de la imagen y al mismo tiempo, su entendimiento global.

Para abordar esto, se utilizan convoluciones dilatadas, las cuales aumentan el campo receptivo en los filtros convolucionales estándar, agregando espacios entre los elementos del filtro.

Esto permite capturar y mantener características de contexto más amplias, sin afectar la resolución de la imagen. Sin embargo, el problema con esta implementación es que aumenta significativamente el costo computacional, ya que, el número de operaciones requeridas incrementa, lo que puede dificultar su uso en escenarios de tiempo real.

Una solución propuesta para abordar la problemática de las convoluciones dilatadas es el uso del Joint Pyramid Upsampling (JPU) [19] [Figura 9]. El JPU genera un conjunto de características derivadas de las

diferentes resoluciones de la imagen, lo cual, logra capturar tanto el contexto como los detalles finos de la misma.

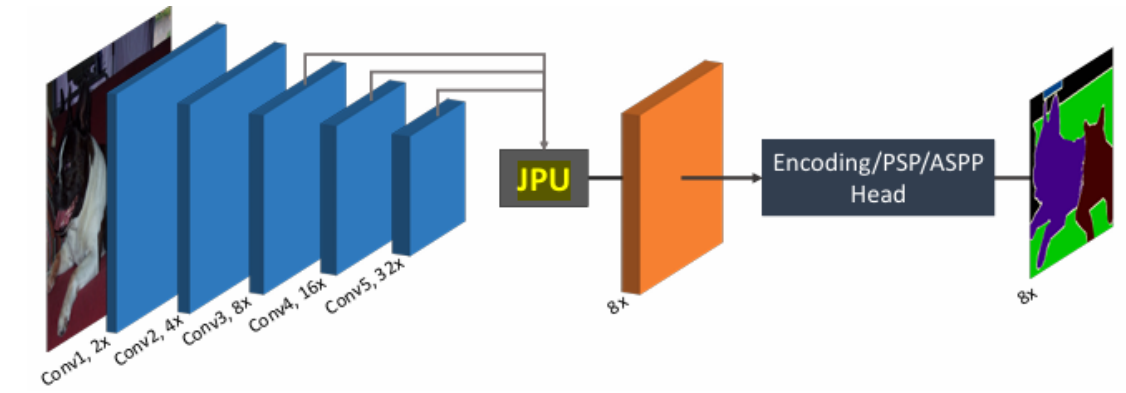


Figura 9. Ejemplo de implementación de JPU. Recuperado de: [19].

En la fase final del proceso, el upsampling devuelve el mapa de características a una resolución más alta, lo que permite mantener los detalles espaciales sin perder contexto.

Esta propuesta no solo permite preservar la información multiescala, sino que también, reduce significativamente el costo computacional en comparación con las convoluciones dilatadas, mejorando la eficiencia del modelo y haciéndolo ideal para su uso en entornos de tiempo real.

Otro de los retos que enfrentaban las CNN era el manejo de detección de objetos de manera robusta en videos largos, ya que, al procesar los frames de manera individual, se perdía la continuidad temporal y el análisis de videos extensos se volvía más complicado.

La solución que surge para abordar este problema es la implementación de convolucionales temporales en conjunto con un entrenamiento semi supervisado.

Las convoluciones temporales lo que hacen es procesar datos en una secuencia de tiempo especifica, aprovechando la relación entre los fotogramas (frames) consecutivos, ya que, en las CNN tradicionales, el procesamiento se realizaba sobre los pixeles de una imagen del video de manera independiente.

Las convoluciones temporales reciben una colección de imágenes tomadas del video y procesadas mediante un filtro que se mueve a lo largo de la secuencia de tiempo. El filtro calcula la multiplicación punto entre los datos de frames adyacentes y los parámetros del filtro, capturando la relación temporal entre los frames. El stride o paso determina cuántos frames se procesan entre cada operación del filtro, permitiendo que el modelo capture patrones de movimiento y variaciones temporales.

Además de esto, el entrenamiento semi supervisado, combina datos ya etiquetados, lo que significa que la salida esperada se encuentra disponible, con datos no etiquetados, los cuales son utilizados para generar predicciones. Estas predicciones se utilizan como pseudo etiquetas que permiten al modelo aprender de la información nueva, logrando así reentrenarlo [20].

2.5 Comparación de Arquitecturas Recientes

Cada día surgen maneras innovadoras de abordar la problemática de detección de objetos, que, lo que tienen en común es que la mayoría de ellas están basadas en tres arquitecturas: CNN, Transformers y Multi Layer Perceptron (MLP).

En el caso de Transformers, estos debieron adaptarse y evolucionar para integrarse en el área de visión por computadora. El diseño estructural original fue desarrollado para ser implementado en procesamiento de texto y esto, al aplicarse a una imagen, trataría cada parte de esta por igual, sin tener en cuenta la relación local entre los píxeles cercanos, brindando un contexto únicamente global y siendo una limitante al momento de buscar capturar detalles locales.

Para lograr comparar de manera justa las CNN, Transformers y MLP se tuvo que desarrollar un framework llamado SPAtial and CHannel processing (SPACH) el cual, es el encargado de unificar la información en espacio y tiempo que se procesan en cada uno de ellos, sin dar ventaja a ninguno.

El resultado de este experimento concluyo en que cada uno de ellos tiene sus pros y contras particulares. En cuanto a las CNN, estas muestran muy buenos resultados si lo que se requiere es capturar información local, ya que se pueden obtener buenos resultados de manera rápida y efectiva, sin embargo, si lo que se busca es capturar relaciones globales (entre fotogramas) o detalles más complejos no sería lo ideal.

En el caso de los Transformers en Visión artificial, si bien generan una excelente respuesta para capturar relaciones globales a largo plazo, el costo computacional puede incrementar exponencialmente y si lo que se busca es una implementación en tiempo real o en imágenes de alta resolución esta opción no sería la más optima.

Por el lado de MLP, se puede concluir que, si bien, es un gran enfoque. sigue requiriendo de más investigación para realmente poder competir entre Transformers y CNN como una solución de gran eficiencia y desempeño [21].

2.6 Hugging Face y su Impacto en el Procesamiento de Videos

Hugging Face es una plataforma de ML que funciona como un repositorio/librería central en la que los desarrolladores pueden experimentar con modelos abiertos [22].

Esto se logra, gracias a que la adopción al ecosistema es sencilla, pues, las librerías encontradas dentro de esta plataforma son modelos preentrenados que cuentan con documentación y tutoriales para poder implementarlos de manera rápida y efectiva. Aunado a esto, la comunidad es muy activa y abierta a resolver dudas que vayan surgiendo durante del proceso de aprendizaje del usuario.

Hugging Face provee tanto la Application Programming Interface (API) como la librería (que dentro de Python se puede encontrar con el nombre de transformers) para probar, cargar y usar múltiples modelos [Figura 10] de una manera rápida y efectiva, ayudando a que los desarrolladores utilicen el que mejor se ajuste a sus necesidades [23]. Su misión principal es ayudar a los desarrolladores a crear soluciones innovadoras sin la necesidad de preocuparse por empezar una implementación desde cero.

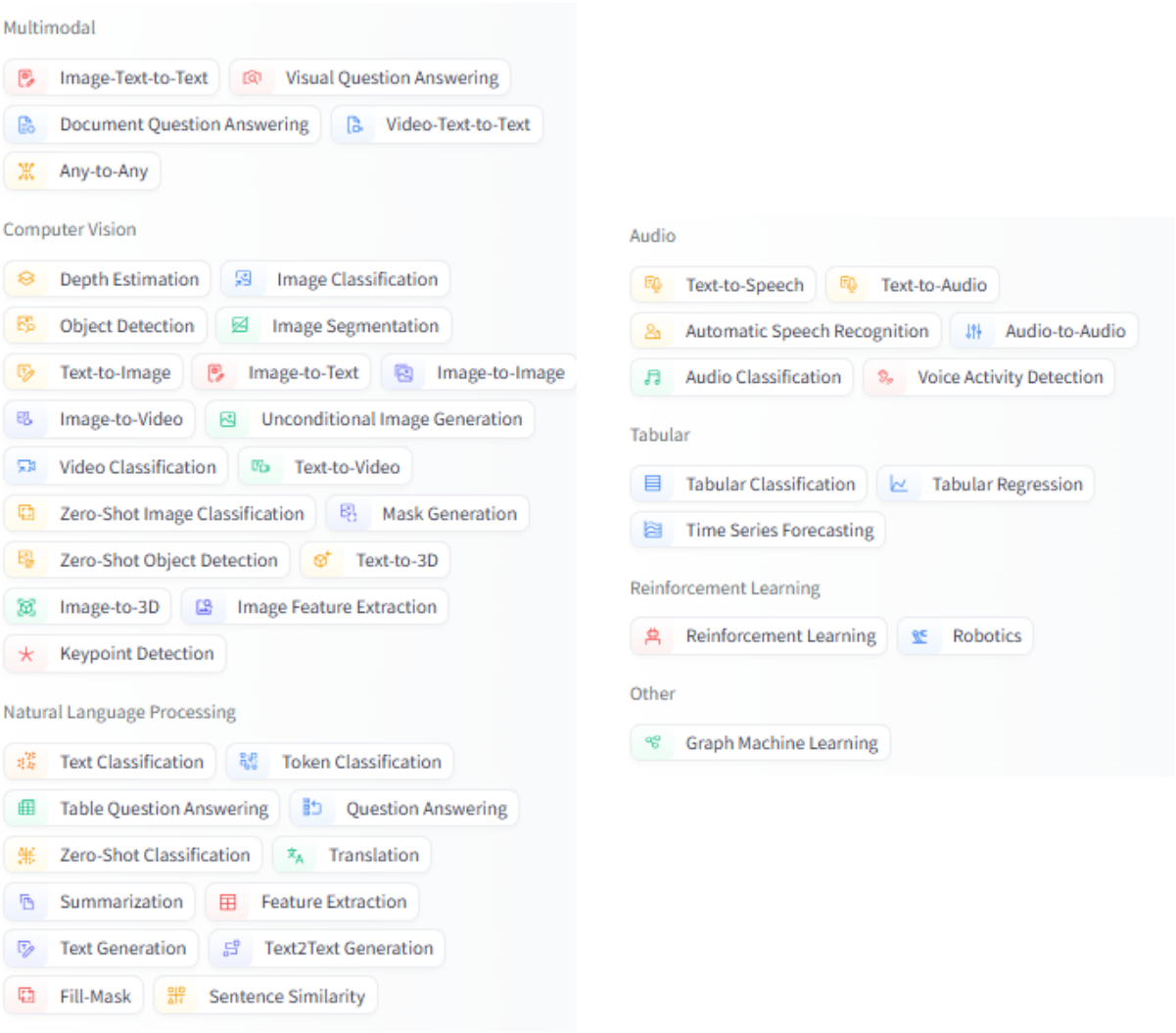


Figura 10. Modelos de Hugging Face disponibles separados por tareas. Recuperado de: [25].

2.7 Modelos de Transformers y CNNs en Hugging Face para la Detección de Objetos

Dentro de Hugging Face, podemos encontrar modelos como DETR [14], el cual, es una combinación entre una CNN normal y un transformer. Este modelo fue propuesto en el dos mil veinte por el equipo de Facebook AI, y hoy en día, dentro de Hugging Face podemos encontrar siete distintas versiones las cuales son:

1. Modelo DETR con ResNet-50 backbone.
2. Modelo DETR con ResNet-50 backbone panóptico.
3. Modelo DETR con ResNet-50 backbone con etapa C5 dilatada.
4. Modelo DETR con ResNet-50 backbone con etapa C5 dilatada panóptico.
5. Modelo DETR con ResNet-101 backbone.
6. Modelo DETR con ResNet-101 backbone panóptico.
7. Modelo DETR con ResNet-101 backbone con etapa C5 dilatada.

Observando las versiones disponibles, se puede concluir que este modelo siempre va a estar compuesto de una ResNet ya sea 50 o 101, sin embargo, la diferencia radica en la implementación que necesite. Si el modelo agrega panóptico a la convolucional [24], lo que se busca es una aplicación hacia el mundo real ya que, esta combina semántica y segmentación de instancias, siendo buena para tareas en las que se busque entender el contexto del objeto en su escena.

Ahora bien, si el enfoque de análisis que se busca es tener un contexto más global del contenido a analizar, implementar un modelo con la etapa C5 dilatada es una gran idea, ya que, al hacer este proceso, se logra aumentar el campo receptivo.

Otro modelo de importancia que se puede encontrar en Hugging Face es ViT [13], propuesto por Google Research, Brain Team en el 2021. Actualmente en esta plataforma podemos encontrar doce diferentes versiones, todos con el objetivo de ser usados para clasificación, utilizando la base de datos de ImageNet para su preentrenamiento. Son diferenciados por la resolución de las imágenes en la colección de datos y las clases que contiene, el tamaño del patch y el modelo.

2.8 Aplicaciones de Hugging Face en Visión por Computadora

Dentro de Hugging Face, podemos encontrar modelos que nos ayuden con procesos de visión computacional en imágenes, tales como la identificación de objetos y su categorización, segmentación y extracción de características [25].

Para conseguir una métrica que permita a los desarrolladores comparar cual modelo tiene la mejor eficiencia, se suelen utilizar conjuntos de datos ya existentes y modelos preentrenados en distintos tipos de escenarios. Esto se hace con el objetivo de poder entender cómo se comportan e implementar aplicaciones rápidas a un problema de importancia o hacer mejoras en modelos existentes.

La flexibilidad que ofrece Hugging Face para comenzar la implementación de cualquier modelo, ayuda a que formas innovadoras de abordar un problema se logren.

2.9 Transfer Learning usando Hugging Face para Detección de Objetos en Videos

Ahora bien, ya que sabemos que los desarrolladores suelen hacer uso de Hugging Face para sus implementaciones, la disyuntiva que surge es, como logran modificar modelos preentrenados y crear nuevas perspectivas para una solución de esta índole.

La respuesta a esto es que se puede, aunque el modelo este preentrenados, hacer cambios que ayuden a obtener distintas perspectivas para encontrar una solución a problemáticas referentes a detección de objetos.

Esto se logra gracias a que Hugging Face permite que, a los modelos preentrenados le puedas modificar las configuraciones en sus capas internas y generar ajustes. Con la utilización de datos específicos, es posible disminuir el tiempo invertido en el entrenamiento y costo computacional y monetario, ya que, no se tendría que invertir en hardware que permita acceder a recursos computacionales robustos y se obtendrían respuestas con mayor rapidez a cualquier problemática de urgencia.

Si tomáramos de ejemplo DETR [14] para este propósito, las modificaciones que se le pueden hacer al modelo podrían ser: congelar capas del ResNet que se esté utilizando como blackbone (capa base), adaptar la resolución basado en los videos que se requieran implementar, modificar el color de las imágenes para entrenamiento, el número de categorías que pretende identificar, el ajuste porcentual en el aprendizaje y el incremento de la información para aquellos conjuntos que no se encuentre balanceados [26].

Otra opción para saber que modelo podríamos implementar como base y modificar para generar mejores resultados, es, consultar investigaciones previas que comparen distintos modelos preentrenados con conjuntos de bases de entrenamiento populares.

El estudio de 2023 [27], ‘Battle of the Backbones’, concluyó que, para realizar identificación de objetos en tiempo real se pueden utilizar modelos basados en CNN tales como ResNet o EfficientNet o bien, si lo que se requiere es una identificación de objetos más precisa, una buena opción a implementar como base podría ser Swin Transformer.

3 MARCO TEÓRICO/CONCEPTUAL

3.1 Introducción a la Detección de Objetos en Videos

3.1.1 Definición de Detección de Objetos

El reconocimiento de objetos hace posible identificar y determinar el total de elementos visibles en una imagen dada una categoría. Esta puede dividirse en dos tipos, las cuales son: la detección de instancias específicas y la detección por categoría [28].

El propósito de la identificación de objetos es desarrollar modelos y técnicas computacionales que permitan determinar qué elementos están presentes y en qué ubicación se encuentran [29].

Los principios que rigen la detección de objetos son:

- **Localización:** Este es un indicado vital que, además de detectar el objeto, sirve como una métrica de precisión (accuracy) [30]. La mejora en la localización contribuye al rendimiento general del sistema si se aplica como perdida para optimizar la posición y el tamaño de la desviación estándar [28]. La localización se suele representar mediante cuadros delimitadores en los datos etiquetados [30].
- **Clasificación:** Su objetivo es encontrar la presencia de un objeto que forma parte de un conjunto de clases, asignando etiquetas en una imagen sin requerir la localización del elemento [28]. Cuando se utiliza como función de pérdida, la clasificación ayuda a evaluar la variación entre la categoría calculada y la real, contribuyendo a mitigar problemas de exceso de confianza (overconfidence) [30].
- **Seguimiento (Tracking):** Es utilizado en aplicaciones de video que requieran seguimiento en tiempo real para mantener la ubicación de elementos a lo largo de múltiples fotogramas. Esto se logra utilizando técnicas que minimizan las diferencias entre las imágenes consecutivas, buscando mantener la trayectoria de los objetos a través de su localización con la menor diferencia al cuadrado [31].
- **Segmentación:** Su objetivo en la imagen es asociar una clase semántica en cada una de sus partes para distinguir diferentes elementos en una escena. En el caso de la segmentación de instancias, también permite diferenciar entre distintas instancias de la misma clase de objeto [28].

La identificación de elementos se puede implementar en imágenes y videos, siempre y cuando se tome a consideración las diferencias en su aplicación.

Al trabajar con video, se debe considerar la secuencia de imágenes que se utilizara para el análisis, es decir, la cantidad de fotogramas tomados del video para la detección de objetos en secuencia. Otro desafío en la implementación con video es la calidad de este, pues, a diferencia de imágenes estáticas, en el video pueden existir desenfoques, oclusión parcial y variaciones en la perspectiva del objeto dentro de los fotogramas [10].

3.1.2 Importancia en Aplicaciones Modernas

Dentro de aplicaciones que utilicen detección de objetos, una de las más populares es la de los vehículos autónomos, los cuales, son capaces de percibir el ambiente y tomar decisiones sin la intervención humana.

La Sociedad de Automóviles e Ingeniería (SAE) en Estados Unidos define la automatización de conducción en seis niveles, que van del cero al cinco. Para que un vehículo sea considerado autónomo, debe encontrarse en nivel 5, el cual implica que el automóvil pueda hacer todo por el conductor: percibir, tomar decisiones y operar sin intervención humana.

Las implementaciones de detección de objetos que se han utilizado para vehículos autónomos a lo largo del tiempo son:

- **Algoritmos de dos fases:** En este caso, la primera fase identifica zonas relevantes dentro de la imagen, y, la segunda fase, envía estas regiones a clasificadores preentrenados. Ejemplos de este tipo de implementaciones son: Region-based Convolutional Neural Network (R-CNN), fast R-CNN y faster-RCNN
- **Algoritmos de una fase**: Se usa únicamente una fase para detectar el objeto, obtener los atributos significativos de la imagen y regresar el resultado, en otras palabras, la identificación y clasificación de un objeto en la imagen se realiza en un único paso.
  Existen implementaciones de este tipo que detectan objetos en entornos bidimensionales, como los métodos YOLO, SSD y CornerNet, estos, presentan mayor complejidad al querer realizar esta tarea en ambientes tridimensionales, para estos casos hay otras aplicaciones como Deep3DBox.
- **Seguimiento de múltiples objetos:** Esta aplicación detecta la entrada de video como fotogramas y extrae características de cada objeto en secuencia. Esto permite el rastreo de múltiples objetos en movimiento en un video.
- **Metodologías modernas para vehículos autónomos:** Estos algoritmos implementan detección de objetos en planos tridimensionales, puesto que, en uno bidimensional no se tiene la capacidad de lidiar con obstáculos en todos los ángulos. Algunos algoritmos que brindan percepción tridimensional son: LiDAR, VoxelNet y técnicas de point cloud [32].

Los sistemas de vigilancia y seguridad son otra de las áreas en las que se ha aplicado la detección de objetos. Un ejemplo práctico de esta aplicación es la videovigilancia mediante drones, que emplean algoritmos avanzados de rastreo multiobjetos en tres dimensiones, como TrackletNetTracker (TNT). Este modelo utiliza información temporal para rastrear y localizar en 3D las coordenadas de los objetos situados en el suelo [33].

El filtrado de contenido inapropiado para menores de edad se ha transformado en un tema de gran relevancia dentro del análisis de contenido de redes sociales, este, implementa detección de objetos en su aplicación. Al ser contenido creado y consumido a diariamente, obtener distintos enforques de solución ayudara a desarrollar soluciones más sólidas, especialmente para secuencia de video largas, con cortes y variaciones de velocidades, ya que, cada una de estas vuelve complejo la creación de detectores automáticos [34].

Una de las propuestas para solucionar esta problemática es TikTokGuard, la cual busca filtrar contenido de riesgo utilizando Transformers dentro de la plataforma de TikTok.

TikTokGuard utiliza una colección de datos etiquetada como: Contenido de riesgo, para adulto, de suicidio y seguro, esto con la intención de ayudar a hacer una identificación adecuada del tipo de contenido que se está visualizando.

Una de las innovaciones del modelo incluye su capacidad de ajuste en el número de fotogramas tomados por video ya que, este no será fijo si no que dependerá de la velocidad y la calidad del video.

Asimismo, se implementaron métodos geométricos para transformar la información del entrenamiento, lo cual, aumento la complejidad de estos para mejorar su robustes. TikTokGuard logro tener un ochenta y siete por ciento de accuracy y busca seguir mejorando y conseguir el apoyo de más desarrolladores para esta causa [34].

3.1.3 Retos en la Detección de Objetos en Videos

Dentro de la identificación de elementos en video, uno de los retos más importantes es lograr su detección precisa en ambientes donde se sobreponen entre sí, lo cual complica las tareas de segmentación, especialmente en enfoques de aprendizaje semi supervisado o aquellos que requieren el rastreo de diversos objetos simultáneamente en tiempo real.

Este problema se liga a las limitaciones de las colecciones de datos aplicadas al entrenamiento del video, puesto que, no suelen incluir anotaciones detalladas en casos como estos.

Para abordar esta problemática, en el dos mil veintidós se propuso el Occluded Video Instance Segmentation (OVIS) [35], un conjunto de datos que considera en sus anotaciones las coordenadas de los objetos que se cruzan entre sí.

OVIS, fue diseñado para mejorar las problemáticas que se afrontan en la identificación de objetos, como segmentación en video y rastreo de elementos que suelen estar obstruidos. Para lograrlo, se emplearon métodos como R-CNN para el rastreo de objetos basado en máscaras y SipMask para la segmentación de instancias en video; mejorando así la precisión y consistencia en el seguimiento de los mismos en tiempo real.

Además, otros desafíos importantes en la detección de objetos son los cambios en la apariencia del video, tales como el desvanecido y desenfoque. Esto tiene relación con los enfoques del modelo y no con el conjunto de datos, ya que, para abordar esta problemática, se requiere que el modelo logre generar una relación secuencial de todos los fotogramas.

Uno de los modelos que aborda esta problemática es TransVOD [36], el cual, es un detector de objetos en video que se basa en la relación espacio temporal para generar su análisis y por ello mejora su desempeño, ya que, incluye una relación cronológica de los fotogramas a lo largo del video considerando información previa y futura.

3.2 Redes Neuronales Convolucionales (CNNs) para Detección de Objetos

3.2.1 Introducción a las CNNs

Los modelos basados en CNN son utilizados para tareas que involucren el entendimiento de imágenes ya sea para su segmentación, clasificación o detección [37]. En su estructura cuenta con capas convolutional (convolucionales), pooling y fully-conected (conectadas).

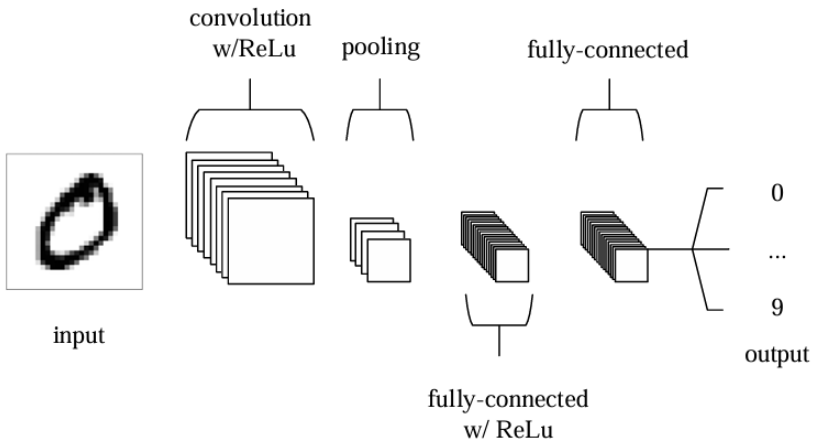


Figura 11. Arquitectura de un modelo CNN básico. Recuperado de: [38].

Las capas convolucionales funcionan gracias al uso de los kernels (filtros), siendo estos una matriz de valor n x n que se desplazan por la imagen de entrada. Su movimiento hace posible el cálculo del producto escalar entre el filtro y la sección de la imagen en la que se encuentra.

El resultado obtenido a partir de este procedimiento es un mapa de activación, el cual, contiene características detectadas a lo largo de las regiones de la imagen.

Las activaciones en el mapa de salida muestran las características detectadas en la imagen, donde los valores más altos o que difieren significativamente del resto representan áreas de coincidencia con patrones específicos. Esto hace que se destaquen regiones como bordes o texturas, permitiendo al modelo identificar elementos relevantes en la imagen

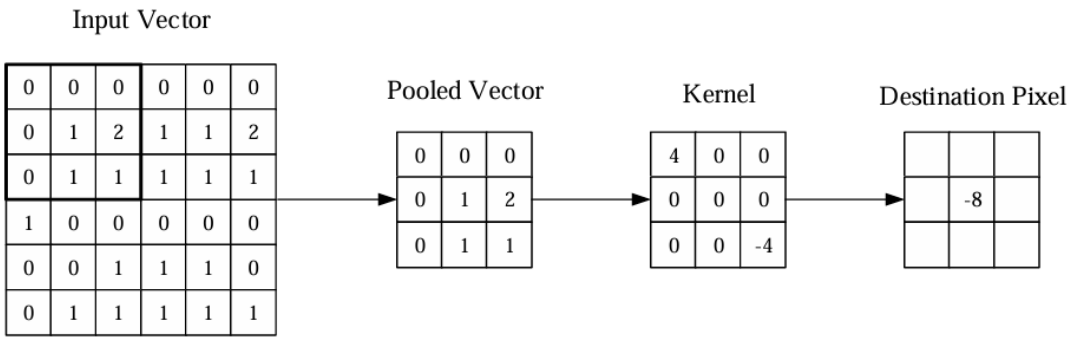


Figura 12. Ejemplificación del funcionamiento de una capa convolucional. Recuperado de: [38].

En implementaciones modernas, las capas convolucionales suelen venir acompañadas de funciones de que introducen no linealidad y a su vez, permititen que la red neuronal identifique patrones avanzados [38]. Un ejemplo de estas son Rectified Linear Unit (ReLU) y Leaky ReLU.

Por otro lado, si lo que busca es mantener una distribución estable sobre las funciones de activación, realizando una normalización basada en la media y la desviación estándar, con la intención de mejorar la estabilidad en el entrenamiento, al modelo puede aplicársele batch normalization (normalización) [37].

También, las estructuras de convolución gracias a las optimizaciones enviadas como hiperparametros, tales como, profundidad (depth), paso (stride) y el zero-padding, reducen la dificultad del modelo. A continuación, se enlistan a detalle:

- **Depth**: Indica cuántos filtros se utilizarán para recorrer la imagen de entrada, teniendo en cuenta que, a mayor número de filtros mayor captación de características. Reducir el valor de este hiperparametro, podría afectar la capacidad de la estructura para distinguir correctamente las características presentes.
- **Stride:** Define el paso de movimiento del filtro dentro de la imagen de entrada. Por ejemplo, si el stride es de 4 el filtro se moverá cada 4 pixeles en la imagen.
- **Zero-padding:** Este hiperparametros genera un borde de ceros alrededor de la imagen, permitiendo mantener sus dimensiones tras la aplicación de la estructura de convolución, ayudando a no perder información de sus bordes.

Por otro lado, el pooling permite reducir la dimensionalidad del mapa de activación, pues, convierte secciones de este en un único valor escalar. Los resultados dependen de la aplicación elegida, entre las más frecuentes se encuentran el max-pooling y el average pooling [39].

Por último, se colocan capas fully-connected al final del modelo, con el propósito de realizar la clasificación; integrando y analizando globalmente la información de salida de cada una de las capas anteriores. Esta capa, si se requiere, puede venir acompañada de dropout como método de regularización.

Dropout desactiva de manera aleatoria cierta cantidad de neuronas definidas por el usuario en cada paso del entrenamiento, ayudando a prevenir el sobreajuste [37].

3.2.2 Modelos Clásicos de Detección de Objetos Basados en CNN

Dentro de los modelos que utilizan como base las CNN, encontramos a YOLO y las distintas versiones que han emergido a partir de esta [Figura 13].

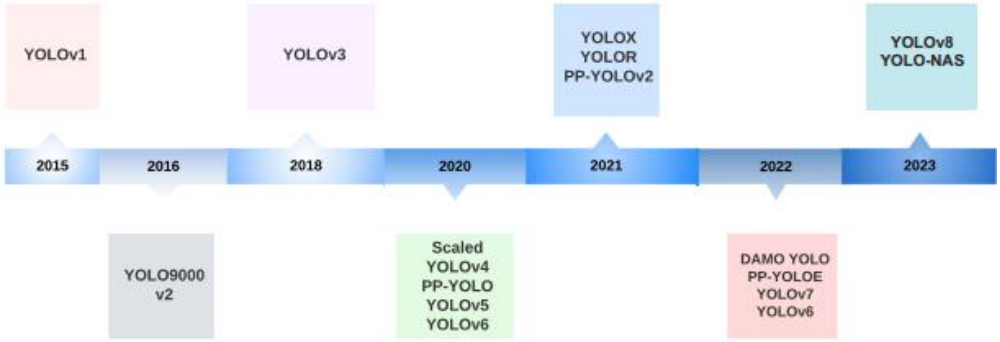


Figura 13. Línea del tiempo de las versiones de YOLO. Recuperado de: [40].

Si se analiza únicamente la versión inicial, su estructura se compone de veinticuatro capas convolucionales, acompañadas de dos capas fully-connected que sirven para inferir las posiciones de los marcos que encierran los elementos identificados en las imágenes. Todas las capas utilizan activaciones ReLU, exceptuando la ultima la cual, utiliza una activación lineal.

En esta versión, YOLO utiliza capas convolucionales de 1x1 con la intención de reducir las características del mapa de activación y mantener el número de parámetros relativos bajos. En la Figura 14 podemos encontrar un ejemplo de la arquitectura de este.

	Type	Filters	Size/Stride	Output
	Conv	64	7 × 7 / 2	224 × 224
	Max Pool		2 × 2 / 2	112 × 112
	Conv	192	3 × 3 / 1	112 × 112
	Max Pool		2 × 2 / 2	56 × 56
1×	Conv	128	1 × 1 / 1	56 × 56
	Conv	256	3 × 3 / 1	56 × 56
	Conv	256	1 × 1 / 1	56 × 56
	Conv	512	3 × 3 / 1	56 × 56
	Max Pool		2 × 2 / 2	28 × 28
4×	Conv	256	1 × 1 / 1	28 × 28
	Conv	512	3 × 3 / 1	28 × 28
	Conv	512	1 × 1 / 1	28 × 28
	Conv	1024	3 × 3 / 1	28 × 28
	Max Pool		2 × 2 / 2	14 × 14
2×	Conv	512	1 × 1 / 1	14 × 14
	Conv	1024	3 × 3 / 1	14 × 14
	Conv	1024	3 × 3 / 1	14 × 14
	Conv	1024	3 × 3 / 2	7 × 7
	Conv	1024	3 × 3 / 1	7 × 7
	Conv	1024	3 × 3 / 1	7 × 7
	FC		4096	4096
	Dropout 0.5			4096
	FC		7 × 7 × 30	7 × 7 × 30

Figura 14. Ejemplo de Arquitectura de YOLO. Recuperado de: [40].

Faster-R-CNN, es otra estructura que funciona para identificar elementos, siempre y cuando se le agregue una convolucional como capa principal, de esta forma, se puede utilizar como detector al implementar una capa de Region Proposal Network (RPN), la cual está encargada de generar el conjunto de propuestas de coordenadas de cuadros delimitadores de objetos con una puntuación clara [41].

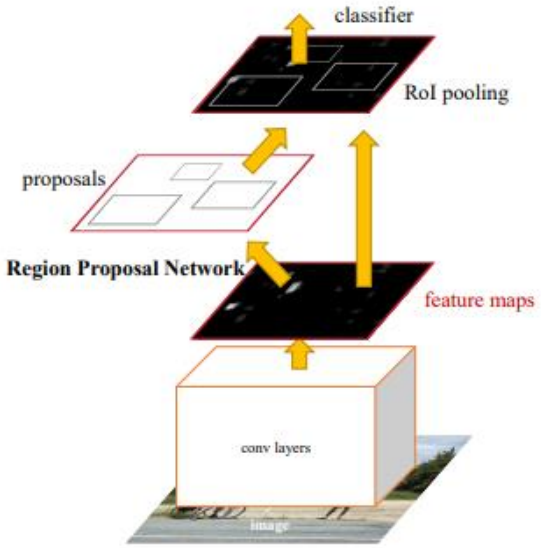


Figura 15. Ejemplificación visual de una Fast R-CNN . Recuperado de: [41].

3.2.3 Limitaciones de las CNNs en Videos

Dentro de las limitaciones de las CNN se debe considerar que originalmente, esta arquitectura estaba pensada para el reconocimiento de objetos en dos dimensiones, al ir evolucionando y por la respuesta que tuvo en el mismo, se trasladó al análisis de objetos en tercera dimensión o videos de alta duración que requieren seguimiento de objetos.

Sin embargo, en aplicaciones de rastreo de objetos en secuencia de video, no es posible utilizar este tipo de modelos puesto que su capacidad de entender la relación entre las secuencias de fotogramas es muy limitada [42].

Otra limitante seria su implementación en objetos que requieran ser detectados, pero estén obstruidos o que la secuencia de video cuente con movimientos bruscos que logren desenfocar características de este, ya que, este tipo de modelos no tiene a adaptarse a distintas perspectivas y escalas pues se requiere un tamaño fijo para el análisis de fotogramas [43][44].

3.3 Introducción a PyTorch

3.3.1 Historia y Evolución de PyTorch

El surgimiento de la estructura de PyTorch proviene de cuatro importantes tendencias dentro de la comunidad científica. El desarrollo de lenguajes específicos tales como MATLAB, APL, R y Julia convirtió a las matrices multidimensionales (Actualmente llamadas Tensores) en objetos de interés para su manipulación.

La migración de la comunidad científica hacia lenguajes como Python, se logró gracias a librerías como NumPy la cual facilitaba el manejo de este tipo de objetos y, cuando se logró automatizar la derivada, se popularizo aún más, puesto que, permitió experimentar con diferentes enfoques de aprendizaje automático.

Además de esto, el uso de Python también se incrementó gracias al movimiento que hizo la comunidad científica hacia el software ya que esto les permitía flexibilidad y libertad de implementación en etapas de desarrollo.

Por último, algo que fue crucial en el progreso de estas investigaciones e infraestructuras fueron las GPU, pues gracias a ellas, se obtuvo la potencia requerida para los métodos de aprendizaje profundo. Así es como surge PyTorch, se basa en modelos de programación basados en matrices y GPUs. [44]

3.3.2 Arquitectura y Componentes de PyTorch

PyTorch se compone de cuatro principios los cuales son:

- **Debe ser Pythonic:** Requiere cumplir con los objetivos de diseño necesarios para mantener las aportaciones simples y consistentes y, así, integrarse con las herramientas estándar de gráficos, depuración y procesamiento de información.

- **Debe poner primero al investigador:** Todo el desarrollo que esté implicado dentro del modelo debe de ser lo más sencillo y productivo posible.
- **Debe proveer un rendimiento pragmático:** Requiere ofrecer un rendimiento apropiado que no comprometa la facilidad de uso y su simplicidad. También, debe proporcionar herramientas que permitan a otros investigadores a controlar manualmente la ejecución de su código.
- **Simplicidad sobre perfección:** Es mejor tener una implementación sencilla y entendible, aunque no esté del todo completa, a una que se compleja y de difícil entendimiento [45].

3.3.3 Librerías y Extensiones de PyTorch

Dentro de PyTorch podemos librerías como TorchVision, la cual, es un paquete que realiza tareas de visión por computadora [46]. Las funcionalidades que podemos encontrar dentro de la misma se encuentran desglosadas dentro de la Figura 16.

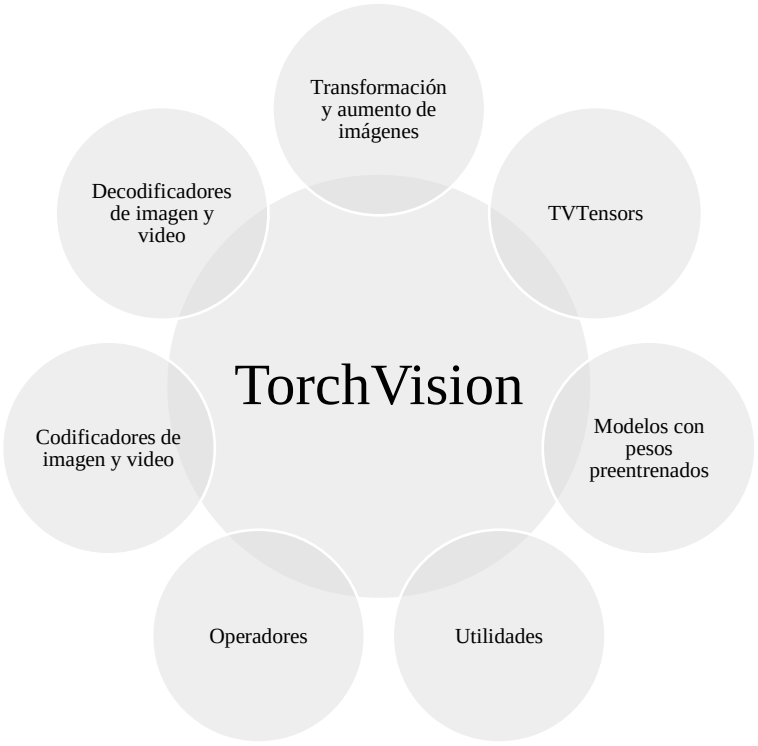


Figura 16. Funcionalidades de TorchVision. Recuperado de: [44].

Por otro lado, uno de los paquetes basado en PyTorch, pero pensado para los desarrolladores es PyTorch Lightning. Su objetivo es ayudar a los investigadores a escribir y estructurar su código de manera más ordenada. Esto se logra gracias a la estructura en la que se encuentra escrito, pues cuenta con código modular que esta separado por las partes que componen un proyecto de aprendizaje profundo.

Aunado a esto, su estructura no afecta la lógica del modelo y conserva la misma flexibilidad que PyTorch. Otra de sus ventajas es que, en él, se pueden automatizar la gestión de hardware disponible para las pruebas que requieran hacerse en el modelo [47].

3.4 Uso de PyTorch en Detección de Objetos

3.4.1 Implementación de Redes Convolucionales en PyTorch

Una red convolucional en PyTorch puede ser implementada de dos maneras distintas, ya sea, creando una clase que defina la arquitectura de una red convolucional, o, utilizando algún modelo preentrenados, el cual solo requiera cargarse y ajustarse de ser necesario.

En esta sección abordaremos únicamente la implementación personalizada. La manera más organizada de hacerlo sería dentro de una clase. Lo primero que debe hacerse es importar todas las librerías que se requieren, tales como torch, torch.nn y torch.nn.functional [Figura 17].

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Figura 17. Librerías de PyTorch necesarias para trabajar con redes neuronales. Recuperado de: [48].

Ahora bien, dentro del constructor de la clase, deben declararse los elementos que se utilizarán antes de ser procesados. El elemento principal es la capa de convolución; además, en este ejemplo se implementarán dropout y funciones lineales en las estructuras fully-connected.

Al comenzar a construir la red neuronal, primero, se deben declarar los elementos que serán utilizados y de ser necesario inicializarlos. Este proceso se lleva a cabo dentro del constructor de la clase [Figura 18].

Para crear una capa convolucional de dos dimensiones, se utiliza **nn.Conv2d**, la cual recibe parámetros de inicialización, de los cuales, los que se muestran a continuación son los que siempre deben definirse ya que no cuentan con valores predefinidos [48]:

- in_channels: Cantidad de bandas de color en la imagen de entrada. Es importante saber reconocer que tipo de imagen se está procesando; 1 canal es requerido para escala de grises, 3 para imágenes a color y 4 para imágenes a color con trasparencia.
- out_channel: Numero de canales producidos en la capa de convolución. En otras palabras, cantidad de características (features) a generar en la imagen de entrada.
- kernnel_size: Tamaño definido para la matriz del filtro que se aplicara en la imagen de entrada [49].

En el caso del dropout, en este ejemplo, al estar trabajando con una convolución de dos dimensiones, el dropout debe seguir el mismo principio, por lo que para declararse debe hacerse como **nn.Dropout2d (** *porcentaje de neuronas que se omitirán en números flotantes* **).**

Para las funciones lineales, solo deben declararse como **nn.Linear(***features de entrada, features de salida***)**, estas se agregarán a las capas fully-connected cuando se construya el modelo.

En la Figura 18, sección A se observa el ejemplo de la declaración de estos elementos.

Al crear el modelo, es de suma importancia tener en cuenta la arquitectura de una CNN o de cualquier otra arquitectura que se busque generar. En este caso, una CNN debe contener: estructura convolucional, función de activación, pooling, normalización, capa fully-connected y regularización.

En la Figura 18, Sección B se ejemplifica un modelo CNN que cumple con cada uno de los puntos mencionados con anterioridad.

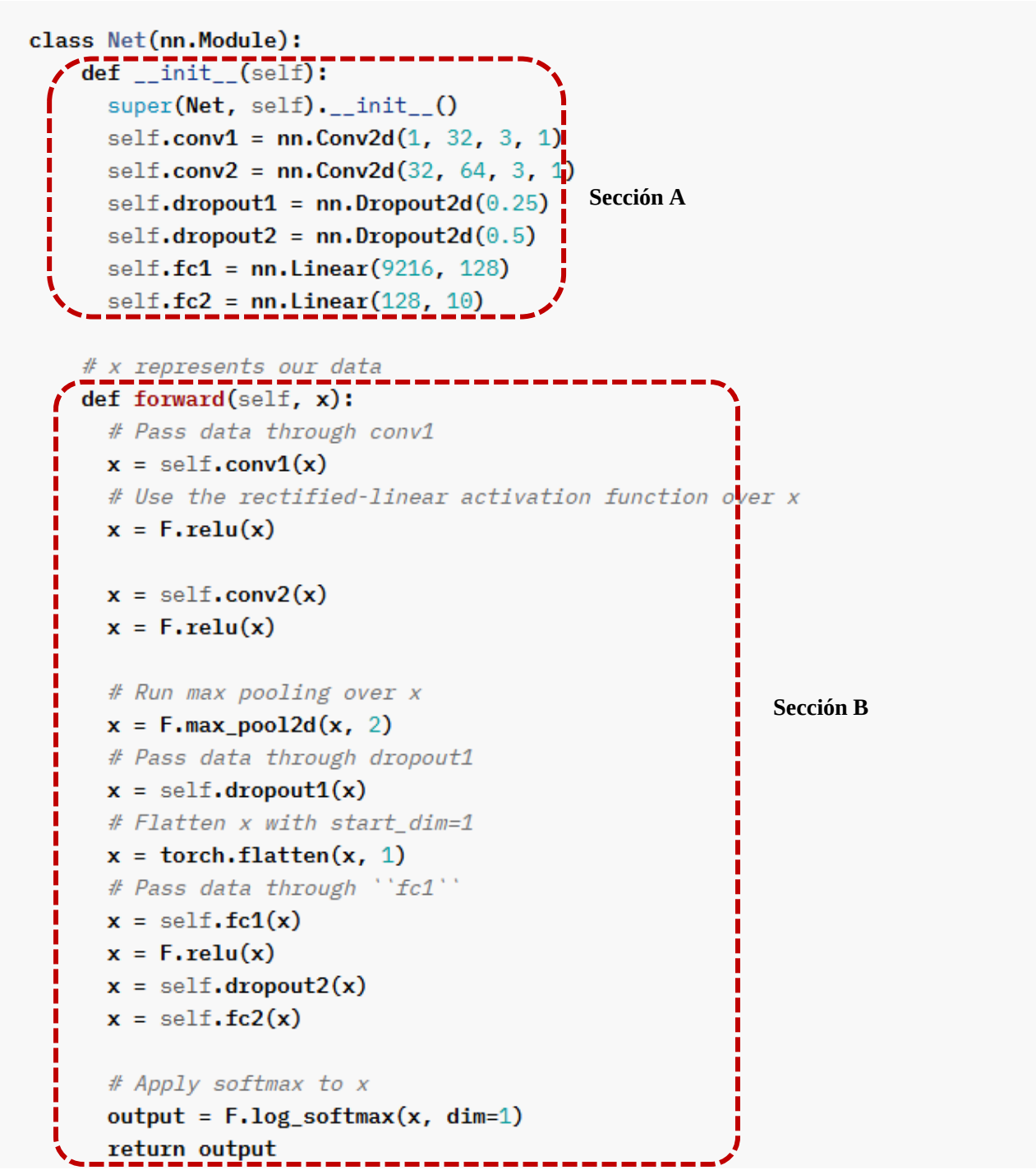


Figura 18. Ejemplo de Red Neuronal Convolucional Personalizada. Recuperado de: [48].

3.4.2 PyTorch y Modelos Preentrenados para Detección de Objetos

PyTorch, brinda la capacidad de cargar o importar modelos basados en CNN o en cualquier otro tipo de arquitectura, ya sean preentrenados o no.

En el caso de YOLO, PyTorch permite hacer la carga del modelo para su implementación, ya sea en su versión preentrenada o desde cero. Para ello se requiere instalar el paquete ultralytics desde pip [Figura 19].

```
pip install -U ultralytics
```

Figura 19. Instalación de ultralytics usando pip. Recuperado de: [50].

Para utilizar el modelo YOLO, se debe importar torch, cargar el modelo desde la librería y especificar si se desea usar preentrenado o no, utilizando el parámetro pretrained.

En este ejemplo, utilizaremos el modelo YOLOv5 en su versión preentrenada. Esto brindara la ventaja de enfocarse en la aplicación del modelo a la problemática específica pues, basta con definir las imágenes en las que se planea realizar la evaluación del modelo. Al pasar la colección de imágenes en el modelo, se logran obtener los puntos de las cajas delimitadoras que rodean los objetos identificados. El ejemplo para esta aplicación se encuentra en la Figura 20.

```
import torch

# Model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Images
imgs = ['https://ultralytics.com/images/zidane.jpg']  # batch of images

# Inference
results = model(imgs)

# Results
results.print()
results.save()  # or .show()

results.xyxy[0]  # img1 predictions (tensor)
results.pandas().xyxy[0]  # img1 predictions (pandas)
#      xmin    ymin    xmax   ymax  confidence  class    name
# 0  749.50   43.50  1148.0  704.5    0.874023      0  person
# 1  433.50  433.50   517.5  714.5    0.687988     27     tie
# 2  114.75  195.75  1095.0  708.0    0.624512      0  person
# 3  986.00  304.00  1028.0  420.0    0.286865     27     tie
```

Figura 20. Ejemplo de implementar YOLOv5 preentrenado. Recuperado de: [50]

El modelo Faster-CNN también puede utilizarse importándose desde PyTorch, específicamente desde TorchVision. En el ejemplo mostrado en la Figura 21, podemos observar cómo se importa el modelo pre entrenado, se definen las imágenes de ejemplo para el entrenamiento, se realiza este mismo y se evalúa el modelo.

```
>>> model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
>>> # For training
>>> images, boxes = torch.rand(4, 3, 600, 1200), torch.rand(4, 11, 4)
>>> labels = torch.randint(1, 91, (4, 11))
>>> images = list(image for image in images)
>>> targets = []
>>> for i in range(len(images)):
>>>     d = {}
>>>     d['boxes'] = boxes[i]
>>>     d['labels'] = labels[i]
>>>     targets.append(d)
>>> output = model(images, targets)
>>> # For inference
>>> model.eval()
>>> x = [torch.rand(3, 300, 400), torch.rand(3, 500, 400)]
>>> predictions = model(x)
```

Figura 21. Ejemplo de implementar Fast-CNN preentrenado. Recuperado de: [51].

Los ejemplos mencionados con anterioridad son sencillos de utilizar e implementar, sin embargo, pueden ser tan complejos como sea necesario. Si se considera que los modelos no se utilizarán preentrenados, se debe tener en cuenta la carga información, entrenamiento, evaluación y prueba, así como también, el fine-tuning (ajuste fino) y la visualización de las predicciones en la imagen [52].

Todo este proceso detallado puede encontrarse dentro de la documentación oficial de PyTorch, lo cual permite a los desarrolladores implementar modelos preentrenados, crear arquitecturas desde cero o diseñar arquitecturas mixtas que combinen modelos preentrenados con nuevas capas personalizadas.

3.4.3 Entrenamiento y Optimización de Modelos en PyTorch

En el proceso de entrenamiento es de suma importancia considerar la optimización, ya que, ayuda a medir como se ajustan los datos al objetivo de su implementación. Para este procedimiento, pueden utilizarse las funciones de perdida y algoritmos de optimización.

Las funciones de perdida brindan una medida del error que el optimizador busca minimizar y sirve como guía para el proceso de ajuste fino.

Cross-Entropy, es una función de pérdida la cual, es aplicada en tareas de clasificación en las que se compara la distribución real con la predicha.

Otra función de pérdida importante es IoU, la cual, es implementada en procesos de segmentación e identificación del elemento. Determina el punto de coincidencia entre la predicción del modelo y la región verdadera del objeto.

Ahora bien, en el caso de los algoritmos de optimización, se busca actualizar los pesos en dirección opuesta a los gradientes, para reducir la función de pérdida y encontrar el mínimo global.

Para este propósito, se pueden utilizar algoritmos como Stochastic Gadient Descent (SGD) el cual, actualiza los pesos calculando el gradiente en un subconjunto de datos, lo cual, permite explorar múltiples mínimos locales.

Otra opción disponible como algoritmo de optimización es Adam, el cual, a diferencia de SGD, se adapta a funciones de perdida que tienen distintas curvaturas, mejorando la convergencia de manera estable. La mayor limitante de Adam es que puede generar sobreajuste debido a como se adapta su tasa de aprendizaje.

3.4.4 Evaluación de Modelos en Tareas de Detección de Objetos

Para confirmar que el modelo entrenado y definido funciona correctamente, debe ser evaluado. En detección de objetos usando CNN, contamos con métricas como: recall, precisión y mean Average Precision (mAP).

Recall y precisión son utilizado en implementaciones que involucren clasificación, ya sea binaria, multiclase o multi etiqueta [Figura 22].

```
from torchmetrics import Recall
from torchmetrics.classification import BinaryRecall
from torchmetrics.classification import MulticlassRecall
from torchmetrics.classification import MultilabelRecall

from torchmetrics import Precision
from torchmetrics.classification import BinaryPrecision
from torchmetrics.classification import MulticlassPrecision
from torchmetrics.classification import MultilabelPrecision
```

Figura 22. Ejemplo de módulos de PyTorch que pueden utilizarse para calcular el recall, precisión y mAP.

La métrica de evaluación recall mide la cantidad de casos positivos reales que fueron correctamente reconocidos [53]. El ejemplo se encuentra en la Figura 23.

```
>>> from torch import tensor
>>> from torchmetrics.classification import BinaryRecall
>>> target = tensor([0, 1, 0, 1, 0, 1])
>>> preds = tensor([0, 0, 1, 1, 0, 1])
>>> metric = BinaryRecall()
>>> metric(preds, target)
tensor(0.6667)
```

Figura 23. Ejemplo de implementación de recall en clasificación binaria. Recuperado de: [54].

También, dentro de las métricas de evaluación, encontramos precisión, la cual, determina el grado de exactitud en las predicciones positivas generadas por el modelo. Esto se traduce a, de todos los valores etiquetados como verdaderos, ¿cuántos son realmente verdaderos en los datos de entrenamiento?. En la Figura 24 se muestra un ejemplo de precisión desde PyTorch.

```
>>> from torchmetrics.classification import BinaryPrecision
>>> target = tensor([0, 1, 0, 1, 0, 1])
>>> preds = tensor([0.11, 0.22, 0.84, 0.73, 0.33, 0.92])
>>> metric = BinaryPrecision()
>>> metric(preds, target)
tensor(0.6667)
```

Figura 24. Ejemplo de implementación de precisión en clasificación binaria. Recuperado de: [55].

Dentro de las métricas de evaluación, también encontramos a mAP, la cual se utiliza en soluciones multiclases. Permite analizar el balance promedio entre precisión y recall en cada categoría, proporcionando una evaluación general del desempeño del modelo en todas las clases consideradas. [56][57].

3.5 Transformers en Visión por Computadora

3.5.1 Introducción a los Transformers

Los transformers son modelos de redes neuronales que son capaces de captar conexiones a lo largo de periodos prolongados. Esto se logra gracias al mecanismo de autoatención (self-attention), ya que, permite centrarse en áreas relevantes de la entrada, sin importar su posición o distancia entre ellas en la secuencia, lo que facilita el procesamiento en paralelo [58] [Figura 25].

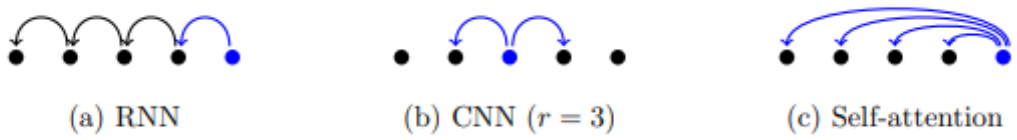


Figura 25. Funcionamiento de mecanismo de atención comparado con el flujo recurrente de otros modelos como Redes Neuronales Recurrentes (RNN) y CNN. Recuperado de: [59].

Son altamente flexibles y escalables, permitiendo su aplicación en múltiples áreas, como procesamiento de texto y análisis de imágenes [58].

La arquitectura de un transformer para NLP está compuesta de la siguiente manera [Figura 26]:

- **Self-Attention:** En el contexto de NLP, permite al modelo revisar las conexiones dentro del texto de entrada y la relación semántica de palabra disponible con el resto, facilitando las relaciones contextuales a largo plazo pues, se relaciona con otras palabras sin importar su posición en la secuencia.
- **Embeddings**: Es un vector numérico de características; cada palabra (token) de entrada se convierte en uno de estos, lo que permite representarlas en un espacio continuo donde la distancia entre vectores refleja la similitud entre palabras.
- **Codificación Posicional**: Está estrechamente relacionada con los embeddings, ya que se utiliza en conjunto con ellos para poder proporcionar su posición, manteniendo el orden de las palabras y permitiendo capturar relaciones contextuales de manera efectiva sin procesar la secuencia en orden.
- **Multi-Head Attention**: Este proceso divide la secuencia de entrada en subvectores, cada uno asignado a una cabeza (head), en donde cada una tenga una representación distinta de la misma secuencia y pueda enfocarse en diferentes aspectos contextuales. Las cabezas funcionan en paralelo y calculan el self-attention en cada subvector de manera independiente, después, se concatenan en una sola representación de salida que combina la información de todas ellas.
- **Feed-Forward:** Son capas que se encargan de procesar las representaciones de cada token después del Multi-Head Attention. Su función es afinar y ajustar las representaciones contextuales capturadas, logrando que el modelo tenga mayor precisión en la comprensión del contexto.
- **Conexión residual:** Es la combinación de la salida de Feed-Forward y la entrada original mediante una suma, permitiendo que el modelo mantenga información de la entrada original y a su vez, aprenda sobre características adicionales.
- **Normalización:** Sucede después de la conexión residual para estabilizar la combinación de la entrada original y la nueva representación con la intención de contar con una distribución uniforme y estable que facilita el proceso de entrenamiento [59][60].

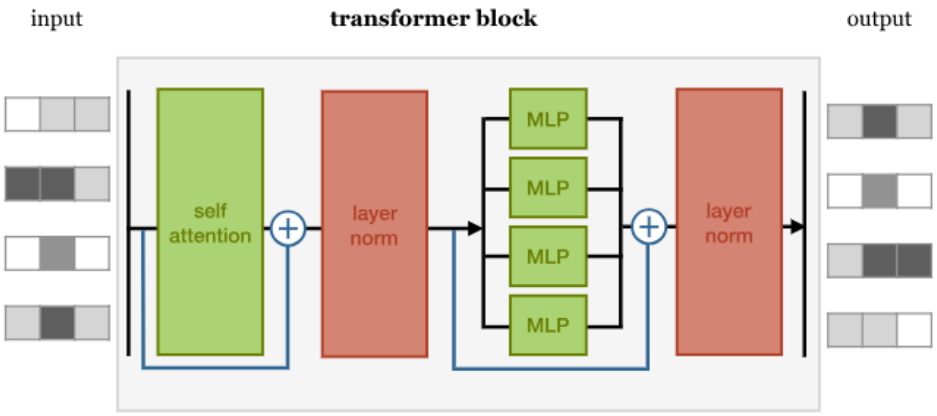


Figura 26. Ejemplificación de arquitectura de transformer en bloque que cuenta en secuencia con capas de: self attention, normalización, feed-forward el cual es un MLP, otra capa de normalización y conexiones residuales agregadas antes de cada normalización. Recuperado de: [59].

3.5.2 Vision Transformer (ViT)

La arquitectura de un ViT está basada en la de un transformers para NLP, con modificaciones que lo adaptan a ser utilizado en el procesamiento de imágenes para su clasificación, estas serán descritas a continuación [Figura 27]:

1. **Entrada del modelo:** En un ViT la entrada es una imagen la cual, debe dividirse en bloques pequeños de pixeles llamados patches (parches), estos son tratados como un token individual, los cuales se linealizan y se convierten en vectores de características.
2. **Codificación Posicional:** Los ViT, al igual que los transformers enfocados en NLP, no procesan la información en orden secuencial. La codificación posicional en ViT se aplica a cada patch de la imagen, a diferencia de los transformers para NLP que se aplica a cada palabra en el texto. La codificación posicional en los ViT sirve para conservar la disposición espacial y captar relaciones espaciales.
3. **Embeddings:** En los ViT, en lugar de ser creados para cada token del texto, los embeddings se generan para cada patch de la imagen.
4. **Token de Clasificación:** El propósito del ViT es la clasificación de imagen es por ello que, se debe agregar un token especial al inicio de cada secuencia de patches para capturar la representación final de la imagen para su clasificación [61].

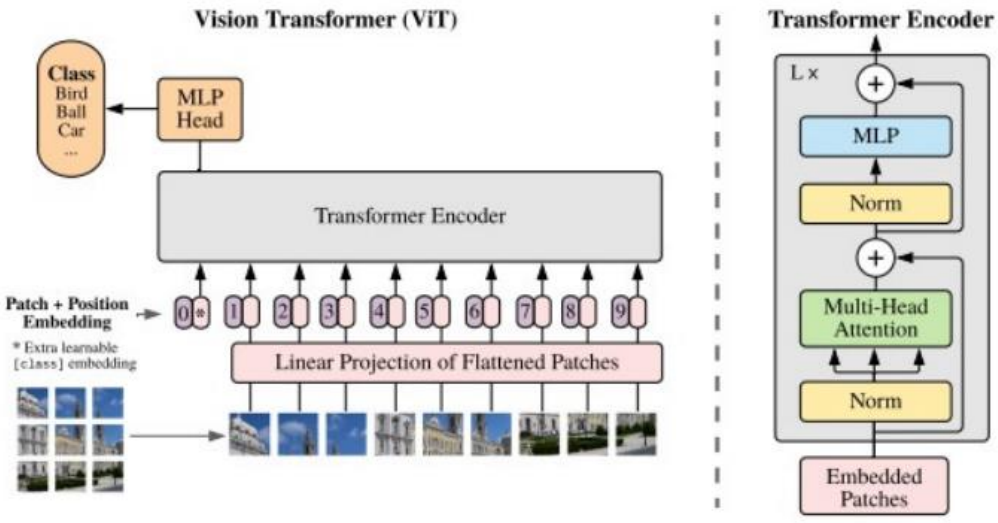


Figura 27. Ejemplo de un ViT para clasificación, de lado izquierdo se encuentran los detalles generales y de lado derecho el detalle del codificador. Recuperado de: [61].

3.5.3 Modelos de Detección Basados en Transformers

Por sí misma, la arquitectura de los ViT no está diseñada para detectar objetos, es por ello por lo que requiere modificaciones para este propósito, logrando así generar nuevos modelos tal como el DETR.

La estructura de un DETR clásico está compuesta cuatro características principales [Figura 28]:

1. **Capa inicial de CNN ResNet**: Es la encargada de obtener los atributos de la imagen y generar secuencias de características, convirtiendo la entrada en una representación que puede ser utilizada por un transformer.
2. **Transformer - Codificador (Encoder):** La secuencia de características generada con anterioridad, llega a esta capa para que se le aplique self-attention, con la intención de identificar regiones de la imagen que se relacionan entre si (relaciones globales).
3. **Transformer – Decodificador (Decoder):** Esta parte del modelo, recibe la salida del encoder para agregar las consultas de objetos (object queries) y se utilizan como marco de referencia para la detección, pues, ayudan al modelo a localizar y clasificar objetos en la imagen. Cada object query es asociado a un posible objeto en la imagen, lo cual permite que se realicen detección de múltiples objetos en paralelo.
4. **Función de perdida – Bipartite Matching**: Esta se utiliza para emparejar cada predicción del object query a un objeto real en la imagen, calculando la pérdida para los cuadros delimitadores, así como también para la clasificación de la imagen [14].

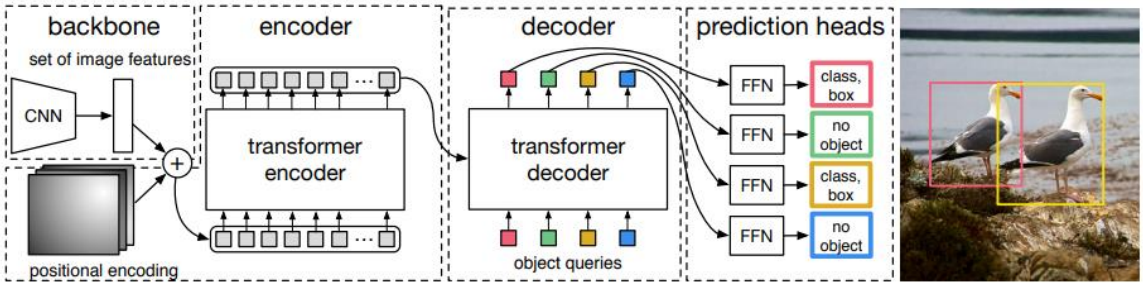


Figura 28. Arquitectura de un DETR. Recuperado de: [14].

A lo largo del tiempo y con la intención de implementar mejorías en eficiencia o detección de objetos a múltiples escalas, han surgido varias mejoras sobre el modelo DETR [Figura 29], algunos de ellos son:

- **Anchor DETR:** Mejora la precisión del modelo en escenas complejas. Utiliza anclas (anchors) como puntos de referencia para predecir los cuadros delimitadores y no depender únicamente de los object queries, logrando localizar objetos de distintos tamaños con mayor grado de confiabilidad.
- **Deformable DETR:** Reduce el tiempo de entrenamiento y mejora el rendimiento en la identificación de elementos a distintas escalas. Introduce el término que lleva su nombre: atención deformable, la cual ayuda al modelo a enfocare en áreas específicas de la imagen en vez de en toda la secuencia de características.
- **Conditional DETR:** Mejora la rapidez del entrenamiento al reducir el número de épocas necesarias, esto lo hace gracias a que no calcula self-attention para todos los elementos en cada etapa si no que lo hace basado en la posición esperada de los objetos, convergiendo más rápido en función a la posición esperada de los objetos [61].

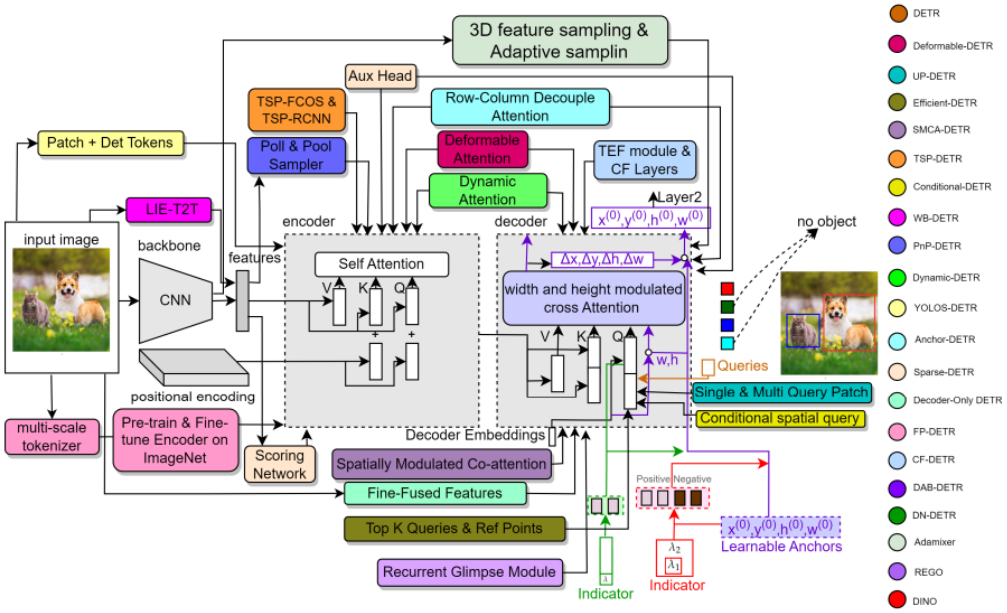


Figura 29. Arquitectura del DETR con sus distintas variantes a lo largo del tiempo. Recuperado de: [61].

3.5.4 Implementación de Transformers en PyTorch

Si se requiere implementar un transformer preentrenado para NLP, PyTorch nos ofrece una librería llamada PyTorch-Transformers, la cual es creada y mantenida por Hugging Face, que cuenta con modelos como: Bidirectional Encoder Representation from Transformers (BERT), Generative Pre-trained Transformer (GPT), GPT-2, Transformer-XL, XLNet, XLM, RoBERTa y DistilBERT [62].

Para habilitar esta librería ser deben seguir los siguientes pasos:

- Instalar la librería Pytorch-Transformers desde pip [Figura 30]:

```
pip install pytorch-transformers
```

Figura 30. Recuperado de: [62].

- Para utilizar modelos de GPT, se requiere la instalación del paquete spancy con una versión especifica de la librería ftfy compatible, este proceso debe realizarse de la siguiente manera [Figura 31]:

```
pip install spacy ftfy==4.4.3
python -m spacy download en
```

Figura 31. Recuperado de: [62].

La documentación de transformers para NLP es bastante amplia y se divide en función del modelo que se desea implementar, las variantes y la tarea especifica que se desea llevar a cabo (Clasificación de secuencias, generación de texto, resumen de texto, traducción, etc.) [62].

Para profundizar en un modelo o tarea específica, consultar la documentación es una excelente manera de comenzar, ya que incluye ejemplos para cada caso de uso y/o modelo. A continuación, se tomará como ejemplo un modelo de clasificación basado en BERT con la intención de mostrar lo sencillo que es implementar un modelo preentrenado desde PyTorch-Transformers.

1. Se debe cargar el modelo BERT desde PyTorch, para tareas de clasificación de modelo y de tokens

```
import torch
sequence_classification_model = torch.hub.load('huggingface/pytorch-transformers', 'modelForSequenceClassification', 'bert-base-cased-finetuned-mrpc')
sequence_classification_tokenizer = torch.hub.load('huggingface/pytorch-transformers', 'tokenizer', 'bert-base-cased-finetuned-mrpc')
```

Figura 32. Recuperado de: [63].

2. Se declaran las oraciones que serán evaluadas y codificadas con el modelo preentrenado *sequence_classification_tokenizer*, convirtiendo el texto de entrada en una secuencia de tokens numéricos y agregando tokens especiales para delimitar cada oración. En casos más complejos, esta sección seria dedicada al conjunto de datos de entrada y su división entre conjuntos entrenamiento y prueba.

```
text_1 = "Jim Henson was a puppeteer"
text_2 = "Who was Jim Henson ?"
indexed_tokens = sequence_classification_tokenizer.encode(text_1, text_2, add_special_tokens=True)
```

Figura 33. Recuperado de: [63].

3. Se declaran los *segments_ids* para indican que oración pertenece a cada token en la secuencia de entrada. El valor 0 se asigna a los tokens que pertenecen a text_1 y el valor 1 a los tokens de text_2.
   La cantidad de 0s y 1s en segments_ids depende del número de tokens generados para cada oración, esto puede calcularse tomando en cuenta el tamaño del arreglo generado por *index_tokens* divididos entre la cantidad de oraciones a evaluar.
   Después, los *segments_ids* y los tokens se convierten en tensores para que puedan ser procesados por BERT.

```
segments_ids = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
segments_tensors = torch.tensor([segments_ids])
tokens_tensor = torch.tensor([indexed_tokens])
```

Figura 34. Recuperado de: [63].

4. Se predice la clasificación en las secuencias de texto; el resultado indica si *text_1* y *text_2* son paráfrasis (Oraciones que a pesar de ser distintas en su redacción, contienen el mismo significado contextual), y para serlo este debe ser de 1.

```
with torch.no_grad():
    seq_classif_logits = sequence_classification_model(tokens_tensor, token_type_ids=segments_tensors)

predicted_labels = torch.argmax(seq_classif_logits[0]).item()
```

Figura 35. Recuperado de: [63].

5. En el ejemplo, también se determina la función de perdida de manera sencilla y opcional, sin embargo, en caso de querer aplicarse en tareas más complejas y personalizadas, es requerido usarla para poder afinar el modelo.

```
labels = torch.tensor([1])
seq_classif_loss = sequence_classification_model(tokens_tensor, token_type_ids=segments_tensors, labels=labels)
```

Figura 36. Recuperado de: [63].

En el caso de transformers para la identificación de elementos en imágenes, las implementaciones se pueden abordar desde dos perspectivas diferentes: se puede crear un modelo desde cero (por ejemplo, DETR), declarando cada parte de la arquitectura en PyTorch, o utilizar una implementación completa y preentrenada.

En este caso, a continuación, se tomará como ejemplo el modelo DETR-ResNet50, con el propósito de mostrar de manera sencilla cómo implementar una estructura de identificación de elementos utilizando transformers.

1. El primer paso es importar las librerías referentes al modelo, las cuales provienen del tranformers la librería torch para manipular tensores y la librería PIL y requests para la manipulación de imágenes.

```
from transformers import DetrImageProcessor, DetrForObjectDetection
import torch
from PIL import Image
import requests
```

Figura 37. Recuperado de: [64].

2. Lo siguiente es cargar la imagen en la que se desea detectar los objetos, para ello nos ayudaremos del módulo Image dentro de la librería PIL. Al ser un ejemplo sencillo se tomará una sola imagen, sin embargo, en aplicaciones complejas se pretende que se tenga una lista de estas para su entrenamiento y prueba.

```
url = "http://images.cocodataset.org/val2017/000000039769.jpg"
image = Image.open(requests.get(url, stream=True).raw)
```

Figura 38. Recuperado de: [64].

3. Luego, se procede a definir los dos modelos preentrenados. El primero modelo va a procesar la imagen (*DetrImageProcessor*) y convertirla en un formato compatible para DETR en cuando a tamaño y normalización. El segundo modelo se utiliza para la detección de objetos (*DetrForObjectDetection*), es el encargado de generar las predicciones y convertirlas en formato de cuadro delimitador y etiquetas de clase.

```
processor = DetrImageProcessor.from_pretrained("facebook/detr-resnet-50", revision="no_timm")
model = DetrForObjectDetection.from_pretrained("facebook/detr-resnet-50", revision="no_timm")
```

Figura 39. Recuperado de: [64].

4. En este punto, se procesar la imagen para ser preparada para enviarse al modelo, se envía y se generan las predicciones de los cuadros delimitadores.

```
inputs = processor(images=image, return_tensors="pt")
outputs = model(**inputs)
```

Figura 40. Recuperado de: [64].

5. Después, el resultado de las predicciones pasa por un proceso de posprocesamiento, el cual, toma el tamaño de la imagen de entrada, como un tensor en formato (alto, ancho), y se ajustan las coordenadas de los cuadros delimitadores para que coincidan con la escala de la imagen original.

```
target_sizes = torch.tensor([image.size[::-1]])
results = processor.post_process_object_detection(outputs, target_sizes=target_sizes, threshold=0.9)[0]
```

Figura 41. Recuperado de: [64].

6. Por último, se visualizan las detecciones del modelo mostrando información como el tipo de objeto, el nivel de confianza y la ubicación en la imagen.

```
for score, label, box in zip(results["scores"], results["labels"], results["boxes"]):
    box = [round(i, 2) for i in box.tolist()]
    print(
            f"Detected {model.config.id2label[label.item()]} with confidence "
            f"{round(score.item(), 3)} at location {box}"
    )
```

Figura 42. Recuperado de: [64].

3.6 Detección de Objetos en Videos

3.6.1 Técnicas Avanzadas para Detección en Videos

Los avances obtenidos en modelos de detección hicieron que surgieran nuevas problemáticas relacionadas al video, la detección de objetos y su seguimiento en los mismos.

El modelo Long Short-Term Memory (LSTM) es una variante de las RNN, la cual, fue diseñada para resolver los problemas del desvanecimiento de gradiente y mejorar la retención de la información en periodos prolongados.

La estructura LSTM, es óptima en problemas de detección y seguimiento de objetos en video, pues, su arquitectura le permite aprender patrones temporales y de continuidad en fotogramas sucesivos, ya que, su objetivo principal es capturar características relevantes a lo largo de una secuencia de datos de entrada.

La arquitectura de una LSTM clásica se compone de lo siguiente:

1. **Celda de memoria:** Componente que permite al modelo retener información de importancia a lo largo de una secuencia temporal. Conserva únicamente el contexto de importancia puesto que, en cada paso temporal puede recordar u olvidar información, según los valores de las puertas (gates) ya que, se encargan de gestionar el flujo de información en la celda de memoria.
2. **Puertas (Gates):** Están encargadas de regular las entradas y salidas de las celdas de memoria, se dividen en tres tipos, las cuales son:
    a. **Input Gate:** Su objetivo es controlar la cantidad de información actual que necesita agregarse a la celda de memoria.
    b. **Forget Gate:** Su propósito es regular el volumen de información previa que debe descartarse en la celda de memoria, evitando que se almacene información irrelevante.
    c. **Output Gate:** Se encarga de decidir qué parte de la información debe enviarse al estado oculto.
3. **Actualización de celda de memoria:** Se encarga de combinar la información del estado actual con la retenida en el estado de la celda anterior, permitiendo que la celda de memoria conserve únicamente la información más relevante y ajustándose en cada paso temporal.
4. **Estado Oculto:** Se relaciona con el output gate y representa la salida de la LSTM en el paso actual. Se encarga de mantener la continuidad de la información a corto plazo, pues, es la salida de la celda en el instante actual y al mismo tiempo proporciona el contexto que se requiere para el siguiente paso temporal [65][66].

3.6.2 Preprocesamiento y Augmentación de Datos de Video

Para poder evaluar videos dentro de un modelo, se debe hacer un preprocesamiento de estos; para este propósito PyTorch dentro de TorchVision, específicamente en *torchvision.io* existe un módulo llamado *VideoReader*, el cual, permite convertir el video de entrada, en secuencias de imágenes de este, llamadas frames.

Basados en su documentación, es posible capturar una cantidad de frames de distintas maneras, ya sea en una cantidad especifica en intervalos de tiempo determinados o muestras de fotogramas aleatorias. Además de eso, esta librería es de mucha utilidad si se quieren crear conjuntos de datos para enviar al modelo que también contengan aumentación de datos [67].

A continuación, se darán ejemplos de uso para la manipulación de video que van desde el manejo de este, así como también la creación de una colección de información con frames aleatorios, todo esto recuperado de la documentación oficial de PyTorch [67].

1. **Manipulación Sencilla de video: Descarga, lectura, metadata del video y división de frames**
    1. **Descarga de videos:** TorchVision cuenta con un modulo que ayuda a descargar videos que cuenten en una URL y provengan de servidores públicos o estén como contenido de algún conjunto de datos de muestra que provee el mismo.
    En la Figura 43 se muestra un ejemplo en el que un video es descargado desde un servidor público y su ruta de acceso se guarda dentro de la variable *video_path*.

```
import torch
import torchvision
from torchvision.datasets.utils import download_url

# Download the sample video
download_url(

"https://github.com/pytorch/vision/blob/main/test/assets/videos/WUzgd7C1pWA.mp4
?raw=true",
    ".",
    "WUzgd7C1pWA.mp4"
)
video_path = "./WUzgd7C1pWA.mp4"
```

Figura 43. Recuperado de: [67].

    2. **Lectura y Metadata del video:** Cuando ya se cuenta con el video de interés almacenado se puede pasar a la lectura, la cual se realiza utilizando el módulo *VideoReader*, a este se le debe especificar el path del video de entrada y el stream, ya que, al este módulo también poder procesar audio, se debe especificar en el stream que lo que se desea procesar es video [Figura 34 Sección A].

    El resultado de la lectura del video será la metadata la cual, estará dividida entre video y audio. El en caso del video, lo que mostrará será la cantidad de imágenes tomadas por segundo. [Figura 34 Sección B]

Figura 44. Recuperado de: [67].

3. **División de frames:** En este caso, el ejemplo se tomará para mostrar la aplicación de la división de frames en video, lo hará en un rango de tiempo específico y se estimará la cantidad que se espera encontrar en el mismo.

   Este ejemplo es muy sencillo, sin embargo, si se escala a una implementación real, brinda las bases teóricas necesarias para comenzar a explorar los intervalos de tiempo e implementar la cantidad de frames que se desean o también tomar un fotograma en todo el conjunto de datos en el mismo tiempo en específico, ya dependerá del usuario y la aplicación que desee darle.

   Lo primero que hace el código es especificar el stream del objeto a procesar, este puede ser audio o video, en este caso se enfocara en video. Seguido de esto se declara una lista de nombre frames la cual contendrá los fotogramas tomados en el intervalo de tiempo que se defina más adelante. También, con la ayuda de *seek* se determina desde que segundo se comienzan a tomar los frames.

```
video.set_current_stream("video")
frames = []  # we are going to save the frames here.
video = video.seek(2)
```

Figura 45. Recuperado de: [67].

   El siguiente paso es la extracción de frames, los cuales están definidos por un rango de tiempo de dos a cinco segundos, para esto, se utiliza itertools.takewhile el cual , ayuda a que se pueda iterar solo hasta que el frame sea menor o igual a cinco segundos y va guardando los fotogramas generados en una lista.

```
for frame in itertools.takewhile(lambda x: x['pts'] <= 5, video):
    frames.append(frame['data'])
```

Figura 46. Recuperado de: [67].

Por último, el ejemplo muestra el total de frames generados en ese intervalo de tiempo, lo compara con la aproximación de los mismo recuperando del metadata la tasa de frames por segundo del video y multiplicándolo por la resta del rango de tiempo definido y finaliza mostrando el tamaño del vector de la primera imagen de entrada.

```
print("Total number of frames: ", len(frames))
approx_nf = (5 - 2) * video.get_metadata()['video']['fps'][0]
print("We can expect approx: ", approx_nf)
print("Tensor size: ", frames[0].size())
```

Figura 47. Recuperado de: [67]

2. **Creación de un conjunto de datos de frames aleatorios**

   Este ejemplo de código aborda la extracción aleatoria de un video en un conjunto de videos disponibles, del cual, se extraerán un fragmento de frames de longitud especifica desde el punto de inicio el cual también será aleatorio [Figura 48 Sección A].

   Después, procede a aplicar transformaciones en cada fotograma, también llamadas aumentación de datos, las cuales modifican las imágenes de entrada ya sea cambiando el brillo, la rotación, recortándolas o normalizándolas, con la intención de proveerle al modelo datos que le ayuden a aprender mejor y generalizar al presentarse con variaciones [68]. En el ejemplo se cuenta con la opción de agregar transformaciones sin embargo por default se encuentran en None [Figura 48 Sección B].

   Por último, el ejemplo devuelve los datos de video listos para ser procesados en un modelo de detección de objetos, ya que, la salida del modelo regresa los frames como un tensor tridimensional [Figura 48 Sección C].


```python
class RandomDataset(torch.utils.data.IterableDataset):
    def __init__(self, root, epoch_size=None, frame_transform=None, video_transform=None, clip_len=16):
        super(RandomDataset).__init__()

        self.samples = get_samples(root)

        # Allow for temporal jittering
        if epoch_size is None:
            epoch_size = len(self.samples)
        self.epoch_size = epoch_size

        self.clip_len = clip_len
        self.frame_transform = frame_transform
        self.video_transform = video_transform

    def __iter__(self):
        for i in range(self.epoch_size):
            # Get random sample
            path, target = random.choice(self.samples)
            # Get video object
            vid = torchvision.io.VideoReader(path, "video")
            metadata = vid.get_metadata()
            video_frames = []  # video frame buffer

            # Seek and return frames
            max_seek = metadata["video"]['duration'][0] - (self.clip_len / metadata["video"]['fps'][0])
            start = random.uniform(0., max_seek)

            for frame in itertools.islice(vid.seek(start), self.clip_len):
                video_frames.append(self.frame_transform(frame['data']))
                current_pts = frame['pts']
            # Stack it into a tensor
            video = torch.stack(video_frames, 0)
            if self.video_transform:
                video = self.video_transform(video)

            output = {
                'path': path,
                'video': video,
                'target': target,
                'start': start,
                'end': current_pts}
            yield output
```

Figura 48. Recuperado de: [67].

3.6.3 Implementación de Detección de Objetos en Videos con PyTorch

Si se desea utilizar modelos de TorchVision para detectar objetos en video, se debe realizar un preprocesamiento que convierta cada uno de los videos en secuencias de fotogramas.

Para una implementación en donde se considere hacer ajuste fino del modelo, deben utilizarse técnicas de data augmentation, así como también, la separación de la colección de información en subconjuntos que contemplen el entrenamiento, validación y prueba; un ejemplo completo de esto puede encontrarse en la sección anterior.

Teniendo los videos en el formato de fotogramas, puede implementarse la estructura de detección que el usuario desee. Sin embargo, es importante combinar la arquitectura de detección con algoritmos de

seguimiento, puesto que permiten asignarle un identificador único a cada objeto detectado en la secuencia, permitiendo un seguimiento continuo.

Los ejemplos de algoritmos de seguimiento incluyen: Simple Online and Realtime Tracking (SORT), BoxMot y sus derivados [69][70].

En la Figura 49, se muestra un ejemplo de un detector de objetos en video con rastreo en el que se combina un Faster R-CNN con una convolucional ResNet-50 y un BoT-SORT del paquete BoxMOT, el cual es un algoritmo de seguimiento.

El desglose del código muestra como ambos modelos se inicializan. Después, se preprocesa el video en fotogramas utilizando la librería Open Computer Vision (Open CV), y cada fotograma se convierte en un tensor para ser capaces de pasar por el detector.

Después, se filtran las detecciones, asegurando que solo se mantengan aquellas que cuenten con al menos un cincuenta por ciento de confianza. Por último, se actualiza el algoritmo de rastreo con los resultados de cada fotograma, asignando un identificador único a cada objeto detectado y se visualizan los resultados mostrando sus detecciones.

```python
import torch
import torchvision
import cv2
import numpy as np
from pathlib import Path

from boxmot import BotSort

# Load a pre-trained Faster R-CNN model
device = torch.device('cpu')  # Use 'cuda' if you have a GPU
detector = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
detector.eval().to(device)

# Initialize the tracker
tracker = BotSort(
    reid_weights=Path('osnet_x0_25_msmt17.pt'),  # Path to ReID model
    device=device,  # Use CPU for inference
    half=False
)

# Open the video file
vid = cv2.VideoCapture(0)  # or 'path/to/your.avi

while True:
    # Capture frame-by-frame
    ret, frame = vid.read()

    # If ret is False, it means we have reached the end of the video
    if not ret:
        break

    # Convert frame to tensor and move to device
    frame_tensor = torchvision.transforms.functional.to_tensor(frame).to(device)

    # Perform detection
    with torch.no_grad():
        detections = detector([frame_tensor])[0]

    # Filter the detections (e.g., based on confidence threshold)
    confidence_threshold = 0.5
    dets = []
    for i, score in enumerate(detections['scores']):
        if score >= confidence_threshold:
            bbox = detections['boxes'][i].cpu().numpy()
            label = detections['labels'][i].item()
            conf = score.item()
            dets.append([*bbox, conf, label])

    # Convert detections to numpy array (N X (x, y, x, y, conf, cls))
    dets = np.array(dets)

    # Update the tracker
    res = tracker.update(dets, frame)  # --> M X (x, y, x, y, id, conf, cls, ind)

    # Plot tracking results on the image
    tracker.plot_results(frame, show_trajectories=True)

    cv2.imshow('BoXMOT + Torchvision', frame)
```

Figura 49. Recuperado de: [71]

3.6.4 Evaluación y Métricas en Tareas de Video

Las métricas de evaluación en aplicaciones que involucran video pueden basarse en métricas de detección de objetos en imágenes. Sin embargo, también requieren de la implementación de métricas que evalúen el rendimiento a lo largo de la secuencia de fotogramas que componen el video.

Un ejemplo de este tipo de métricas es Multiple Object Tracking Accuracy (MOTA) la cual mide la precisión global del seguimiento de múltiples objetos.

Otra métrica de importancia para el mismo propósito es ID F1 Score (IDF1), la cual, mide la consistencia de la identificación de los elementos a través de los fotogramas; a mayor valor de IDF1 mayor consistencia al mantener la identidad de los objetos durante el rastreo [72].

3.7 Hugging Face y PyTorch en la Detección de Objetos

3.7.1 Modelos Preentrenados en Hugging Face

Uno de los modelos que se encuentra dentro de Hugging Face y puede utilizarse para detección de objetos es YOLO. Se puede hacer uso de este modelo de tres maneras distintas: preentrenado para utilizarse en una evaluación directa, preentrenado para hacer ajuste fino utilizando un conjunto de datos de entrenamiento personalizado, o entrenándolo desde cero (scratch) [Figura 50].

El implementar un modelo desde cero en un conjunto de datos requiere de recursos computacionales robustos para manejar adecuadamente el proceso de entrenamiento y realizar el ajuste fino de acuerdo con los resultados experimentales. Esta opción no es la más adecuada para implementaciones sencillas o para conjuntos de datos grandes en los que no se cuente con la infraestructura necesaria para su procesamiento.

```
from ultralytics import YOLO

# Create a new YOLO model from scratch
model = YOLO("yolo11n.yaml")
```

Figura 50. Ejemplo de declaración de modelo de YOLO desde cero. Recuperado de: [73].

La manera más sencilla de realizar la implementación de un modelo es usarlo preentrenado y aplicarlo directamente sobre la colección de datos a evaluar, siempre y cuando, se consulte las variantes de estos, en el caso de YOLO su entrenamiento se ha dado con distintos conjuntos de datos provenientes de COCO, por lo cual, debe revisarse cuál es el más adecuado a utilizar basado en la implementación que se esté buscando.

En la Figura 51 podemos encontrar un ejemplo de uso de modelo preentrenado YOLO para su implementación directa para tareas de detección y clasificación, utilizando el conjunto COCO8 como datos de preentrenamiento.

```python
# Load YOLO11n, train it on COCO128 for 3 epochs and predict an image with it
from ultralytics import YOLO

model = YOLO('yolo11n.pt')  # load a pretrained YOLO detection model
model.train(data='coco8.yaml', epochs=3)  # train the model
model('https://ultralytics.com/images/bus.jpg')  # predict on an image
```

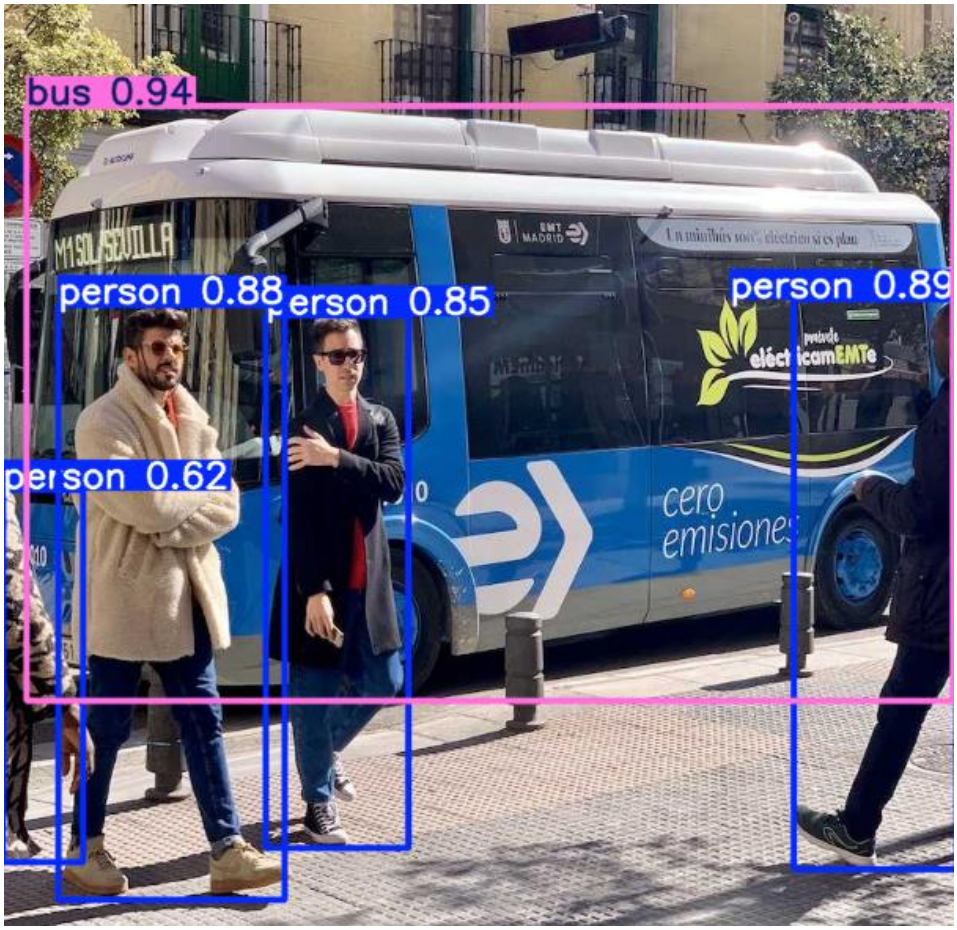


Figura 51. Ejemplo de uso de modelo preentrenado YOLO en la detección y clasificación de objetos en una imagen. Recuperado de: [74].

3.7.2 Transfer Learning con Hugging Face y PyTorch

El Transfer Learning o transferencia de aprendizaje, es una técnica en la cual un modelo desarrollado y preentrenado previamente en un conjunto de datos, conocido como source domain (fuente de dominion), se adapta para resolver una tarea diferente en un nuevo conjunto de datos, denominado source target (dominio de objetivo).

En otras palabras, el objetivo del Transfer Learning es tomar un modelo preentrenado y aplicarlo a una colección de datos personalizada, optimizando la capacidad del modelo para responder a la nueva tarea mediante un ajuste fino que le permita adaptarse al contexto del source domain.

El modelo aun siendo ajustado, conserva el conocimiento aprendido en el source domine, lo que acelera el entrenamiento puesto que no requiere de grandes cantidades de datos como las que se necesitarían si se requiriera entrenar desde cero y, a su vez, optimiza el rendimiento en el target domain.

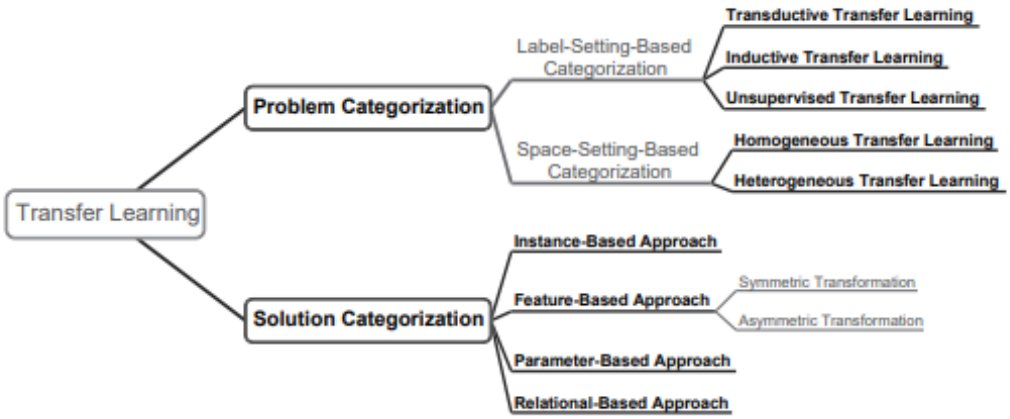


Figura 52. Transfer Learning, tipos y subtipos. Recuperado de: [75].

Las estrategias para la aplicación de Transfer Learning se pueden abordar de dos maneras distintas; por categorización de problemas y por categorización de solución [Figura 52]. A continuación, se enlistarán algunos de sus subtipos con su definición.

1. **Categorización de Problemas**
    a. **Inductive Transfer**:

        Se utiliza cuando se busca realizar la misma tarea del source domain en el target domain, sin realizar un ajuste en el modelo, solo ajustando los datos de entrenamiento.

        Ejemplo: Se tiene un source domain que detecta fraudes bancarios en Europa y se quiere aplicar en un target domain para fraudes bancarios en México.

        Para lograrlo, se eligen los datos de entrenamiento para el target domain que cuenten con patrones similares a los utilizados para entrenar el source domain.

    b. **Transductive Transfer**:

        Este se aplica cuando el source domain y el target domain resuelven la misma tarea, pero, difieren en la distribución de la colección datos para el entrenamiento.

        Ejemplo: Cuando un source domain utilizado para reconocimiento facial y entrenado con imágenes de alta resolución se implementa en un target domain para realizar la misma tarea con imágenes de baja resolución. Los modelos resuelven la misma problemática, la cual es reconocer rostros, sin embargo, se deben ajustar por las diferencias en la resolución de los datos de entrenamiento en el target domain

c. **Unsupervised Transfer:**

Es utilizado cuando el source domain tiene datos etiquetados y se implementa en un target domain en donde sus datos de entrenamiento no cuentan con etiquetas. La intención de su aplicación es aprovechar el source domain para tareas de agrupación o reducción de la dimensionalidad sin implementar un modelo de supervisión.

Ejemplo: Un source domain utilizado para clasificación de escenas en imágenes se implementa en el target source para que sean agrupadas por sus similitudes visuales. Aunque el target domain no cuenta con etiquetas, el modelo puede identificar características similares y formar agrupamientos significativos basados en su entrenamiento previo.

2. **Categorización de Soluciones**

a. **Instance-Based:**

Es utilizado cuando el target domain busca realizar una tarea relacionada al source domain por medio de la adaptación del modelo.

Ejemplo: Se tiene un source domain que detecta fraudes de manera general en Europa y se quiere aplicar en un target domain para fraudes con tarjetas de crédito en México.

Esta nueva aplicación es una subcategoría de la implementación original y para lograr su adaptación se realizan ajustes o entrenamiento adicional con el nuevo conjunto de datos para lograr su enfoque en la aplicación.

b. **Feature-Based:**

Esta aplicación toma las características generales aprendidas en el source domain y las aplicarla en el target domain sin modificar significativamente los parámetros internos del modelo.

Esto se logra agregando al modelo nuevas capas para clasificar o detectar características específicas requeridas en el target domain, manteniendo los parámetros en las capas inicialices y modificando las finales.

Ejemplo: Un source domain para clasificar mamíferos en general se desea aplicar en un target domain que quiere usar el conocimiento previo del modelo para diferenciar perros, gatos y vacas. Para lograrlo, se pueden congelar las capas iniciales que identifican patrones generales de los mamíferos y agregar y entrenar capas finales con el objetivo de diferenciar las categorías de perros, gatos, vacas.

c. **Parameter-Based:**

Se utiliza cuando se busca ajustar los parámetros internos del source domain y adaptarlos al target domain sin cambiar la estructura del mismo ni añadir nuevas capas. Aquí, los parámetros se reutilizan y ajustan mediante un proceso de ajuste fine, lo cual, permite

conservar conocimientos generales adquiridos en el source domain mientras se refina el modelo para que funcione en el target domain.

Ejemplo: Un source domain entrenado para la clasificación de automóviles se utiliza para generar un target domain que clasifique únicamente vehículos Chevrolet. Para lograrlo, las capas finales del target domain se reentrenan con el conjunto datos de imágenes de vehículos Chevrolet, ajustando sus características para reconocer a la marca.

d. Relational-Based:

Este enfoque busca transferir las relaciones aprendidas entre los datos en el source domain para aplicarlas en el target domain. En otras palabras, permite que el modelo mantenga una estructura relacional que se ha demostrado útil en un contexto y la adapte a otro.

Ejemplo: Un source domain aplicado para recomendación de películas, genere un target domain para recomendación de música, transfiriendo relaciones útiles entre usuarios y preferencias [75].

La aplicación del Transfer Learning depende de las capacidades del modelo preentrenado y las adecuaciones en la colección de datos utilizados para ser compatible con el mismo.

A continuación, se utiliza YOLO como ejemplo para mostrar las adecuaciones que se requieren para la implementación de una colección de datos personalizada. Así como también, se mencionarán los conjuntos de datos disponibles con los que puede emplearse el modelo preentrenado, con el fin de considerar las adaptaciones requeridas antes de llegar a la etapa de Transfer Learning.

La Figura 53, expone la estructura que debe tener un conjunto de datos personalizado para poder ser utilizado en YOLO. Este conjunto de datos debe estar organizado dentro de una carpeta principal que contenga dos subcarpetas: images (imágenes) y labels (etiquetas), que, a su vez, estas deben contener subcarpetas que dividan los datos en train (entrenamiento) y val (validación) [75].

Para utilizar el modelo de YOLO preentrenado como detector, se pueden implementar las colecciones de datos como: COCO y sus variantes, Objects365, OpenImageV7, SKU-110K, VisDrone, xView, Roboflow 100, entre otros [75]. Para determinar cuál conjunto de datos es el más adecuado acorde a la implementación que se requiera, se recomienda revisar la documentación oficial de cada conjunto de datos y la documentación de YOLO, considerando las implementaciones y los usos específicos que cada uno ofrece.

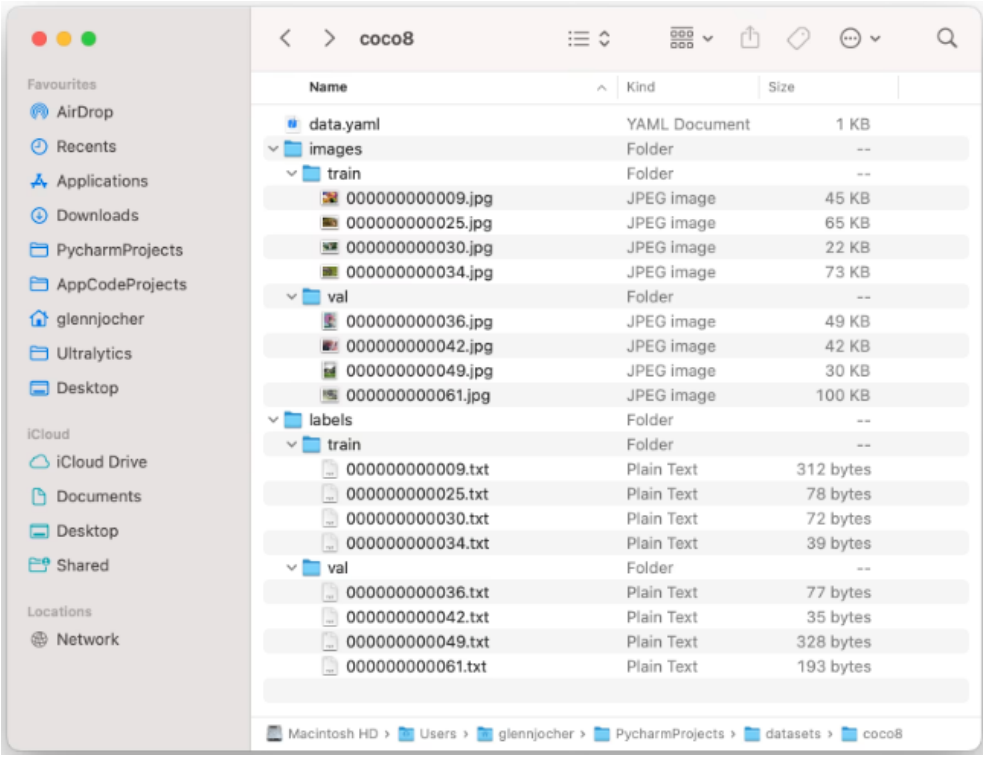


Figura 53. Estructura que debe seguir una colección de datos para utilizar YOLO preentrenado. Recuperado de: [76].

3.8 Datasets y Recursos para Detección de Objetos en Videos

3.8.1 Datasets de Videos

Algunos de los datasets públicos que se encuentran para video son:

1. **AVA**

   Es un conjunto de datos desarrollado por Google Reasearch que tiene como objetivo permitir a los investigadores analizar y clasificar acciones humanas en videos.

   AVA, cuenta con dos versiones enfocadas en video, las cuales son AVA v2.1 y AVA kinetics, las cuales incluyen ochenta clases de acciones humanas con anotaciones a nivel de fotograma en intervalos de 3 segundos provenientes de fragmentos de películas.

   Este dataset puede aplicarse en tareas de reconocimiento de acciones, análisis de interacciones humas o como recurso para el entrenamiento de modelos de detección de objetos adaptado a los objetivos esperados del investigador [77].

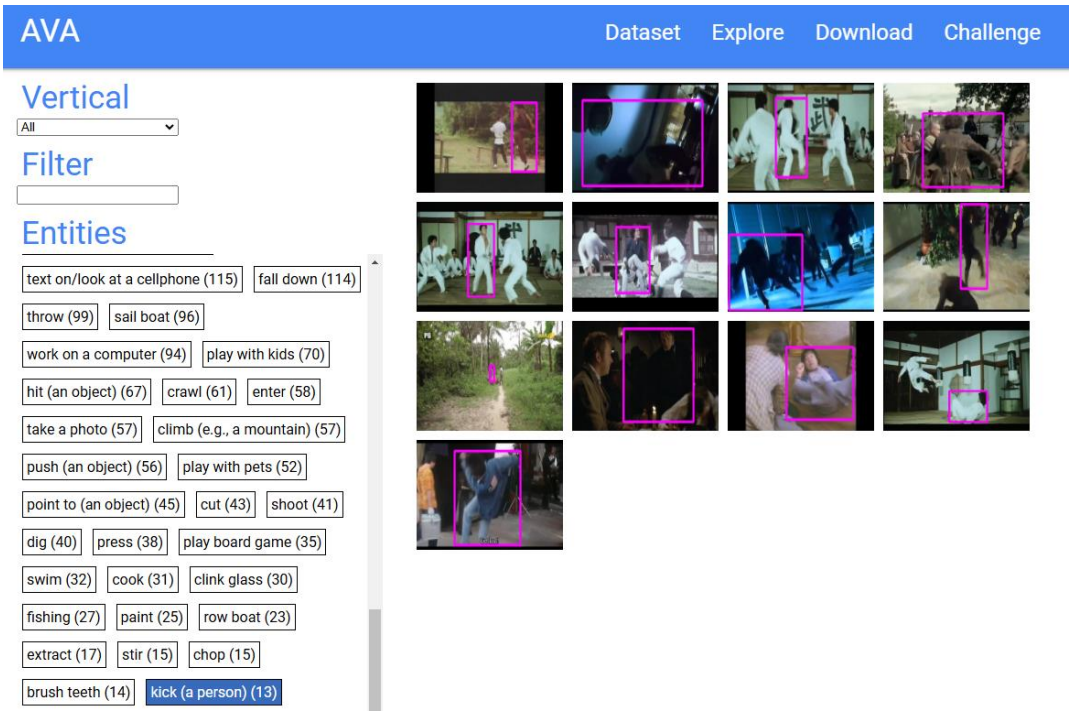


Figura 54. Exploración del conjunto de datos AVA. Recuperado de: [78]

2. MEVA (Multimodal Video Dataset for Activity)

MEVA es una colección de datos diseñada para usarse en actividades de detección en videos multivista puesto que, se compone de grabaciones en múltiples ángulos para captar la misma actividad en una visión global. MEVA, también es un conjunto de datos multimodal, ya que, además de video, captura información acerca del audio, ubicación y tiempo.

El objetivo de MEVA es identificar actividades humanas complejas en ambientes controlados ya sea de vigilancia o seguridad. Cuenta con más de treinta y siete categorías para clasificar actividades humanas en las que también se incluyen interacciones con objetos

Las implementaciones de MEVA suelen estar orientadas a sistemas de seguridad y vigilancia que requieren detección de actividades sospechosas o en modelos de análisis de actividades complejas [79].

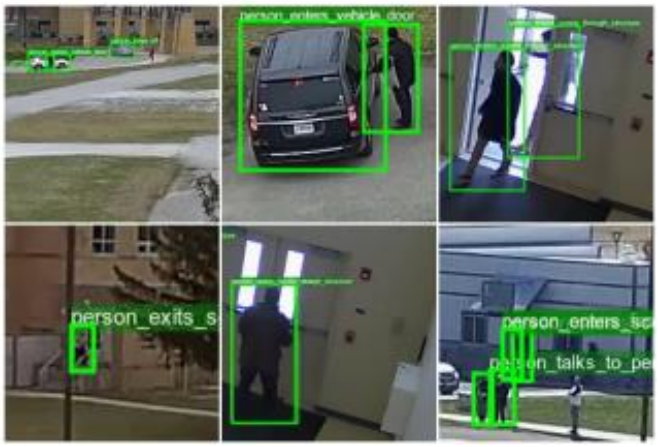


Figura 55. Ejemplo de cómo MEVA captura actividades desde distintos puntos de vista. Recuperado de: [79].

3. **UCF101**

   Es una colección de datos que cuenta con representaciones de acciones humanas en múltiples entornos, las cuales varían en condiciones visuales tales como la iluminación, ángulo y movimiento del fondo.

   UCF101, cuenta con ciento un categorías, las cuales se dividen en cinco tipos de acciones principales: Deportes, ejercicio, juegos grupales, actividades cotidianas e interacciones ya sea entre personas o personas y objeto. Suelen utilizarse para tareas que impliquen reconocimiento y clasificación de actividades humanas en videos, facilitando el entrenamiento y la evaluación de modelos en conjuntos de datos complejos [80].

3.8.2 Anotación y Etiquetado de Videos

Para poder utilizar conjuntos de datos personalizados en modelos preentrenados, como se mencionó en puntos anteriores, estos deben cumplir con un formato en específico. Realizar el etiquetado y separación de los datos de manera manual resultaría en una gran inversión de tiempo, es por ello por lo que existen herramientas de código abierto que facilitan este proceso.

Un ejemplo de ello es Label Studio, plataforma que facilita el etiquetado de objetos para actividades de segmentación, clasificación y detección. Esta herramienta permite exportar las anotaciones en formatos compatibles con COCO y YOLO, y a su vez, ofrece opciones en formato JavaScript Object Notation (JSON) o Comma Separated Values (CSV). Soporta como entrada videos en formato MP4, AVI y MOV [81].

Otra opción de código abierto con la que se cuenta es Scalabel, que, también permite el etiquetado para segmentación, clasificación y detección de objetos. A diferencia de Label Studio, de exportación soportados son compatibles con JSON y COCO, y para el video de entrada únicamente soporta MP4 y AVI [82].

3.9 Desafíos y Futuras Tendencias en la Detección de Objetos en Videos

3.9.1 Desafíos Técnicos en la Detección de Objetos en Videos

Sin lugar a duda, la inteligencia artificial y el aprendizaje profundo han tenido grandes avances en la última década. Sin embargo, aún existen desafíos técnicos significativos en la detección de objetos en videos que afectan la eficiencia en diversas aplicaciones.

Al procesar videos, se presentan problemas específicos referentes a la iluminación, oclusión, desenfoque y variación de tamaño a lo largo de la secuencia. Estos factores dificultan el seguimiento preciso de objetos de forma precisa puesto que deben adaptarse a cambios rápidos en las condiciones visuales de cada fotograma.

Por otro lado, en implementaciones en tiempo real, también cuentan con dificultades a enfrentar, ya que, para ser aplicada esta tarea en ese contexto, se requiere de una demanda computacional altas, lo que aumenta los costos de procesamiento y limita el acceso a esta tecnología a desarrolladores e investigadores que cuenten con ese recurso.

La complejidad espaciotemporal también resulta un gran desafío puesto que, para detectar objetos en prolongados periodos de tiempo por medio de fotogramas implica mantener la identidad y ubicación del objeto de forma consistente a través de estos y al mismo tiempo evitar que el modelo se sobre cargue de contexto innecesario [83].

3.9.2 Tendencias Futuras en Modelos de Video

El futuro de la investigación en modelos de video se orienta hacia los Transformers multimodales, ya que, pueden aprender de múltiples fuentes de datos, llamadas modalidades, combinando la información para crear representaciones que capturen un contexto más amplio gracias a la relación entre los elementos de cada modalidad.

Un ejemplo de un Transformer multimodal es Video-Audio-Text Transformer (VATT). Este modelo tiene un enfoque de entrenamiento no supervisado, permitiendo que aprenda patrones de las distintas modalidades de manera autónoma.

Entre las ventajas que ofrece VATT se destacan el remover la dependencia del etiquetado de los datos, lo cual es bastante útil al trabajar con grandes volúmenes de ellos, ya que, el etiquetado suele ser costoso y demandante de tiempo.

También, VATT genera una integración contextual que ayuda en la mejora del rendimiento en tareas que impliquen varias modalidades, produciendo un contexto amplio y robusto entre ellas. También, es eficiente, puesto que procesa cada modalidad de manera paralela y puede aplicarse y adaptarse a múltiples dominios gracias a su capacidad de generalización [84].

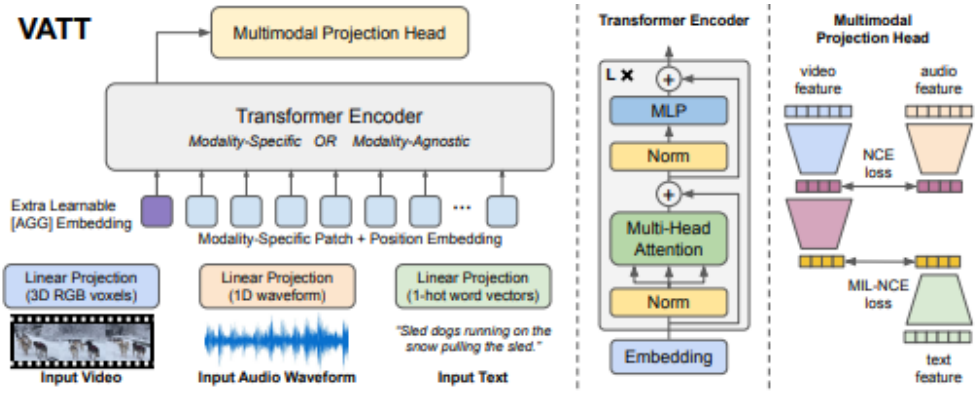


Figura 56. Arquitectura de VATT. Recuperado de: [84].

Aun con todas las ventajas mencionadas con anterioridad, los Transformers multimodales también enfrentan retos importantes. Procesar altos volúmenes de datos de múltiples modalidades requiere de recursos computacionales significativos. Otro aspecto que considerar es la sincronización entre las modalidades, puesto que, se vuelve compleja la alineación temporal entre elementos de video, audio y texto y esta debe ser precisa para mantener la coherencia en el aprendizaje.

También, se requiere manejar la optimización de parámetros de manera eficiente puesto que, de no hacerlo correctamente, dificultaría la convergencia entre modalidades y a su vez su rendimiento del modelo y la sobrecarga en recursos.

4 DESARROLLO METODOLÓGICO

a. Levantamiento de requerimientos

Dentro de las etapas del proyecto se definieron cuatro puntos de importancia para su desarrollo:

- Elección del conjunto de datos.
- Acotamiento de la clasificación del modelo: Categorías a considerar dentro de disturbing.
- Explicación de la elección del Visual Model preentrenado a implementar.
- Desarrollo del código implementando ViT.

b. Elección del conjunto de Datos para el modelo

Para seleccionar el conjunto de datos que será utilizado para su implementación en este objeto de estudio, se tomó como base el generado para la investigación YouTubers Not madeForKids [85].

La intención de utilizar el mismo conjunto de datos del estudio previo es poder identificar todos aquellos videos que no han sido retirados de la plataforma y buscar alternativas de modelos que detecten los objetos considerados de peligro dentro de los mismos.

Lo que marca la diferencia entre la investigación en curso y la desarrollada en YouTubers Not madeForKids [85] es que, en dicha investigación el entrenamiento del modelo se desarrolla con información acerca de los canales de YouTube y no con los videos por sí mismo.

Profundizando más en el tema, en la implementación previa, los datos que se tomaron en cuenta están basados en texto los cuales fueron: descripción del canal, likes (me gusta), comentarios, links, clasificación del contenido (Si el creador declaraba que el contenido era acorde para menores de edad), transcripción del audio a texto y etiquetas agregadas por el influenciador en el canal para clasificar el contenido.

Actualmente, dentro de la plataforma de YouTube Kids, no se cuenta con la información que se tomó para desarrollar dicha investigación, por ejemplo, la sección de comentarios ya no está habilitada, incluso la descripción del canal ya no es visible.

Por este motivo, el propósito la investigación en curso es obtener una muestra de la colección de datos desarrollado para la investigación previa, que acote la problemática a resolver y que además, se centre en aquellos videos que aún se encuentren disponibles en la plataforma de YouTube.

En la investigación previa, el conjunto de datos se encuentra separado en dos archivos CSV, llamados, suitable (adecuados) y disturbing (inquietante). Ambos cuentan con cincuenta y dos columnas las cuales se describen a continuación, incluyendo entre paréntesis su tipo de dato:

- **id (string):** Identificador del canal.
- **category (string):** Categoría del canal.
- **videoCount (int):** Cantidad de videos en el canal.

- **hiddenSubscriberCount (bool):** Por medio de un valor booleano, identifica si el canal cuenta con la cantidad de subscriptores oculta o no.
- **subscriberCount (int):** Cantidad de subscriptores en el canal.
- **viewCount (int):** Total de vistas que contempla todos los videos del canal.
- **description (string):** Descripción del canal.
- **descriptionCharCount (int):** Cantidad de caracteres utilizados para escribir la descripción del canal.
- **keywords (list):** Palabras claves asociadas al canal las cuales, en el contexto de YouTube, categorizan el contenido del canal de acuerdo con las mismas.
- **keywordsCount (int):** Cantidad de keywords encontrados.
- **topicCategories (list):** Categorías en las cuales YouTube por medio de sus algoritmos clasifico el contenido del canal.
- **topicCount (int):** Cantidad de topicCategories encontrados.
- **Country (string):** Región en la que se encuentra el creador del contenido del canal.
- **madeForKids (bool):** Valor booleano que determina si el contenido es creado o no para niños ya sea definido por el creador de contenido o por YouTube.
- **publishedAt (str):** Fecha y hora en la cual un video (En este caso el suitable o disturbing) fue publicado.
- **videoIds (list):** Id de los videos suitable o disturbing, dependiendo el CSV en el que se encuentre revisando la información, que se encuentran dentro del canal.
- **disturbingCount (int):** Cantidad de videos inquietantes encontrados en el canal
- **suitableCount (int):** Cantidad de videos adecuados encontrados en el canal
- **suitableRatio (float):** Representación de la proporción de los videos del canal que son adecuados.
- **madeForKidsRatio (float):** Representación de la proporción de los videos del canal que se encuentran categorizados como madeForKids.
- **madeForKidsVideosCount (int):** Total de videos que se encuentran categorizados como madeForKids.
- **madeForKidsDisturbing (int):** Representación de la proporción de los videos del canal que se encuentran categorizados como madeForKids, pero en realidad son inaporpiados.
- **instagram (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **twitter (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **facebook (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **spotify (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **twitch (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **patreon (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.

- **vk (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **plus.google (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social. (Esta red social fue descontinuada desde el 2019).
- **play.google (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a la tienda de google.
- **steam (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **myspace (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **merchandise (int):** Valor booleano en numérico que determina si dentro de la descripción del video se encuentran links que redirigen a esta red social.
- **kids (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **animation (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **funny (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **toys (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **cartoonfun (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **children (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **cartoons (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **comedy (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **nursery rhymes (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Entertainment (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Film (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Lifestyle_(sociology) (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Music (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Hobby (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.

- **Video_game_culture (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Action-adventure_game (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Television_program (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Action_game (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.
- **Role-playing_video_game (int):** Valor booleano en numérico que determina si el contenido se encuentra dentro de esta categoría determinada por YouTube.

De las columnas disponibles en la colección de datos de la investigación previa, se seleccionaron únicamente dos para el trabajo en curso: category y videoIds.

La columna videoIds fue modificada, puesto que, originalmente contenía una lista de diccionarios con varias keys (claves), de las cuales, para la investigación en curso, solo era relevante el id del video. Dicha columna fue modificada para conservar únicamente el id del video, agregándole la parte del enlace faltante que permite acceder directamente al contenido de ese id en YouTube.

Así, se pasó de tener una lista de diccionarios [Figura 57] a una lista de links, obteniendo un conjunto de datos que incluye tanto la categoría del video como el enlace del mismo [Figura 58].

videoIds
[{'classification_label': 'disturbing', 'channelId': 'UCPDlh3WjaEF6VB3sjDk76qg', 'id': 'mZurp3-_vlg', 'publishedAt': '2016-02-15T22:05:42.000Z', 'madeForKids': False}]
[{'classification_label': 'disturbing', 'channelId': 'UCEt4vnW3zxpfI8CfHmO_1yA', 'id': 'AE9-VeKS57Y', 'publishedAt': '2017-10-15T14:53:19.000Z', 'madeForKids': False}]
[{'classification_label': 'disturbing', 'channelId': 'UC1Ej1HysXFszU2d1rMhzk1w', 'id': 'Kl8jKZ9HH3U', 'publishedAt': '2007-05-13T19:30:31.000Z', 'madeForKids': False}]
[{'classification_label': 'disturbing', 'channelId': 'UCSLIJTwAKRhlSlIvhbpHPuQ', 'id': 'H-lizN8SWVE', 'publishedAt': '2017-10-13T21:37:16.000Z', 'madeForKids': True}]

Figura 57 Muestra de la columna videosIds en la colección de datos de YouTubers not madeForKids

category	videoIds
disturbing	https://www.youtube.com/watch?v=e5KMHHXLzIw
disturbing	https://www.youtube.com/watch?v=nx9XavymuBU
disturbing	https://www.youtube.com/watch?v=PWwFNONK14k
disturbing	https://www.youtube.com/watch?v=yHECVm7kqZE
disturbing	https://www.youtube.com/watch?v=I8-oCUcf9pE
disturbing	https://www.youtube.com/watch?v=DuQRYbECLeM
disturbing	https://www.youtube.com/watch?v=o5kCVDQAnzQ
disturbing	https://www.youtube.com/watch?v=vfI18YIJsl4
disturbing	https://www.youtube.com/watch?v=6kTnIid1uKM
disturbing	https://www.youtube.com/watch?v=QNQsqVfXw2E
disturbing	https://www.youtube.com/watch?v=ShwLwLOg0xo
disturbing	https://www.youtube.com/watch?v=ZO-IbPzj3_8
suitable	https://www.youtube.com/watch?v=GC_mV1IpjWA
suitable	https://www.youtube.com/watch?v=3bLfzgZ-wO8
suitable	https://www.youtube.com/watch?v=ktVkWWi9vCw
suitable	https://www.youtube.com/watch?v=yfUVkQ0vrJw
suitable	https://www.youtube.com/watch?v=JfwqSXmJ-mE
suitable	https://www.youtube.com/watch?v=Fs7aw3mA22I
suitable	https://www.youtube.com/watch?v=AR03AyaJiqM
suitable	https://www.youtube.com/watch?v=bTUYg2eCH2g
suitable	https://www.youtube.com/watch?v=PfQ-5VcuS9E
suitable	https://www.youtube.com/watch?v=wL11YcsHqLc
suitable	https://www.youtube.com/watch?v=PPk3oeFszdo
suitable	https://www.youtube.com/watch?v=gpxofiVlBVA

Figura 58 Muestra del conjunto de datos utilizado para la investigación en curso

c. Acotamiento de la clasificación del modelo: Categorías a considerar dentro de disturbing

Al revisar la información utilizada en la investigación previa, se encontró que, de los seiscientos videos declarados, actualmente, solo doscientos están disponibles, de los cuales, sus temáticas se centran en drogas, armas y contenido explicito.

Ahora bien, si se considera que, al implementar un modelo preentrenado, se utiliza un dataset público (COCO, YouTube8M, entre otros), se deben revisar las categorías de clasificación con las que cuentan estos conjuntos, puesto que, se debe confirmar cuál de las temáticas de los videos disponibles para evaluar cuenta con mayor representación. En este caso, de las tres categorías disponibles, armas es la que mayor representación.

Por este motivo la problemática se acoto a clasificar contenido que aún se encuentra disponible dentro de YouTube, en una muestra de veinticuatro videos que muestre el uso de armas como: guillotina, revólver, rifle, cuchillo, rifle de asalto, sierra de cadena, antorcha, espada, misil, bomba, granada, tanque, ballesta, trampa para osos, cañón, tridente, tijeras, machete, dardo y hacha, para así, clasificar si un video es suitable o disturbing.

d. Explicación de la elección del Visual Model preentrenado a implementar

Como se ha visto a lo largo del trabajo en curso, dentro de Hugging Face, ultralytics y PyTorch, se pueden encontrar modelos preentrenados con determinado conjunto de datos, que se eligen acorde a la implementación que se busque realizar y puedan ser utilizados directamente para resolver una problemática de visión en imágenes o videos.

Al comenzar a evaluar los modelos disponibles para la detección de armas, se tomaron en consideración los siguientes:

- YOLO

  Este modelo para ser preentrenado e implementado en tareas de clasificación, cuenta con nueve conjuntos de datos disponibles, de los cuales, el único conveniente para detectar armas es ImageNet [17]. Su implementación es sencilla y cuenta con documentación detallada al respecto, sin embargo, su uso tuvo que ser descartado puesto que, de las categorías de clasificación con las que cuenta, únicamente dos de ellas se relacionaban con armas, las cuales son: *cuchillo y tijeras* [86].

- ViT

  Este modelo puede ser utilizado tanto como clasificador o detector de objetos en imágenes. Puede utilizarse preentrenado implementando ImageNet-21k desde google/vit-base-patch16-224. Este fue el modelo seleccionado ya que, además de contar con una implementación sencilla, las categorías que involucran armas dentro de la colección de datos utilizados en el preentrenamiento son de veinte, lo que resulta en el modelo que cuenta con mayor representación para la implementación buscada [87].

e. Desarrollo del código implementando ViT

A continuación, se detallará la implementación del código:

1. Las librerías utilizadas para la implementación de detección de armas en videos de YouTube se pueden observar en la Figura 59. Cada una de las mismas será enlistada a continuación con la explicación de su uso:
   i. **streamlit:** El uso de esta librería es para implementar para crear una interfaz gráfica. Específicamente, para el propósito de este trabajo, se utiliza para crear una página web interactiva que analizar el contenido de videos de YouTube de manera visual.
   ii. **cv2:** Es utilizada para realizar el preprocesamiento del video antes de que este llegue al modelo, permitiendo leerlo, generar los fotogramas que serán analizados y convertirlos en formato RGB, lo cual, los hará compatibles con el modelo.
   iii. **transformers/ViTForImageClassification:** Esta librería, tomada desde transformers, importa desde Hugging Face el modelo ViT para clasificación de

imágenes. Por lo tanto, este recibe los fotogramas preprocesados y clasifica los objetos en categorías predefinidas por el conjunto de datos utilizados en el preentrenamiento.

iv. **transformers/ViTImageProcessor:** Esta librería, permite adaptar los fotogramas preprocesados al formato que requiere el modelo de clasificación, lo que asegura su compatibilidad.

v. **wordcloud/WordCloud:** Se implementa para generar mapas de palabras. En el presente trabajo, su propósito es para mostrar al usuario de forma gráfica, las armas detectadas dentro del video.

vi. **matplotlib.pyplot:** Su uso se relaciona con wordcloud, puesto que, es el encargado de mostrar dicho mapa en la interfaz del usuario.

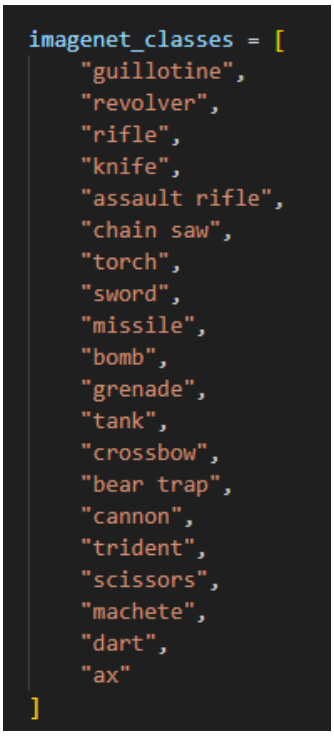
vii. **datetime/timedelta:** Se utiliza para calcular y registrar el tiempo exacto de cada fotograma procesado en el video, mostrando los tiempos en un formato entendible como hh:mm:ss. En palabras más simples, facilita el manejo del tiempo en el video para procesar correctamente sus fotogramas.

viii. **yt_dlp:** Librería utilizada para descargar el video de interés desde YouTube mediante su URL. Además, permite extraer metadatos como el título del video, él cual es utilizado en esta implementación para identificar el contenido que se está descargando.

```
import streamlit as st
import cv2
from transformers import ViTForImageClassification, ViTImageProcessor
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from datetime import timedelta
import yt_dlp
```

Figura 59 Librerías implementadas para la detección de armas en videos de YouTube

2. Se procede a declarar una la lista de palabras que serán utilizadas como filtros para determinar si un video muestra contenido peligroso relacionado a armas. Esta lista se basa en las categorías de ImageNet-21K que están relacionadas con armas y objetos peligrosos [Figura 60].



```
imagenet_classes = [
    "guillotine",
    "revolver",
    "rifle",
    "knife",
    "assault rifle",
    "chain saw",
    "torch",
    "sword",
    "missile",
    "bomb",
    "grenade",
    "tank",
    "crossbow",
    "bear trap",
    "cannon",
    "trident",
    "scissors",
    "machete",
    "dart",
    "ax"
]
```

Figura 60 Lista de palabras que muestra clasificaciones de ImageNet-21K relacionadas con armas

3. Se genera una función que permite descargar y preparar el video de YouTube que se desea analizar [Figura 61]. En ella se utiliza la librería yt_dlp y sigue el siguiente flujo:

    i. Se define la configuración de ydl_opts. En esta, debe especificarse la ruta (path) en la que se guardará el video descargado y el formato de la calidad del video.
    ii. Se inicializa el objeto YoutubeDL con las configuraciones definidas en ydl_opts, lo que permitirá realizar las operaciones de descarga y extracción de los metadatos.
    iii. Se extraen los metadatos del video para obtener el título de este y mostrarlo en streamlit para que sea visible para el usuario.
    iv. Se descarga el video utilizando la URL especificada por el usuario.
    v. Se muestra en streamlit el nombre del video que ha sido descargado y guardado.

```python
def download_video(youtube_url, output_path="video.mp4"):
    ydl_opts = {
        'outtmpl': output_path,
        'format': 'best',
    }
    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        get_info = ydl.extract_info(youtube_url, download=False)
        st.write(f"Descargando el video: {get_info.get('title', None)} .....")
        ydl.download([youtube_url])
        st.write(f"Video descargado y guardado como: {output_path}")
```

Figura 61 Función encargada de la descarga del video de YouTube y el acceso a sus metadatos

4. Se declara la función load_model [Figura 62], la cual, es la encargada de cargar el modelo de ViT para clasificación de imágenes, junto con el objeto encargado de preprocesar las imágenes antes de ingresarlas al modelo. Ambos componentes están preentrenados con google/vit-base-patch16-224, el cual, está basado en ImageNet-21K. Al final de la función, se retornan tanto el modelo como el extractor de características para su uso.

```python
def load_model():
    model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224')
    feature_extractor = ViTImageProcessor.from_pretrained('google/vit-base-patch16-224')
    return model, feature_extractor
```

Figura 62 Función encargada de cargar el modelo y extractor de características

5. A continuación, se declara la función detect_objects_in_video [Figura 63] la cual, recibe como parámetros, la ruta del video que se desea analizar, el modelo de clasificación a implementar, el extractor de características y la ruta del archivo donde se guardará la información de los objetos peligrosos encontrados en el video. El flujo que sigue esta función es el siguiente:
    i. Para mantener informado al usuario, se muestra en streamlit que indica que el análisis del video ha comenzado.
    ii. Se abre el video utilizando la ruta proporcionada como parámetros de entrada.
    iii. Se obtienen los cuadros por segundo (fps) del video, los cuales son esenciales al momento de calcular los tiempos de los fotogramas procesados.
    iv. Inicializa el contador de fotogramas en cero, permitiendo rastrear el progreso en el procesamiento.
    v. Se declara un tipo de datos set para registrar los objetos ya identificados, evitando duplicidad.
    vi. Utilizando with, se abre el archivo log en modo escritura. Esto garantiza que toda la información generada sobre los objetos detectados se guarde en el archivo y que, a su vez, este se cierre automáticamente al finalizar el proceso.
    vii. Se declara el bucle encargado de procesar cada fotograma del video.

viii. Para cada iteración dentro del bucle, se lee un fotograma del video, retornando dos variables, una que indica si esta lectura se llevó a cabo fue exitosa y la otra que contiene los datos del fotograma.

ix. Se Incrementa el contador de fotogramas y, utilizando los fps del video, se calcula el tiempo exacto en el que se encuentra el fotograma en formato hh:mm:ss.

x. Se preprocesa el fotograma apoyándose de dos puntos importantes. La librería cv2 ayuda a convertir el fotograma de formato Blue Green Red (BGR) a Red Green Blue (RGB). El resultado obtenido pasa al feature extraction, el cual, adapta la imagen al formato esperado por el modelo de clasificación.

xi. Se procede a realizar la clasificación del fotograma, implementando el modelo de clasificación y obteniendo las predicciones al identificar las etiquetas correspondientes al contenido presente en la imagen.

xii. Se verifica la predicción utilizando la lista de palabras relacionadas con armas definidas previamente en imagenet_classes y basadas en categorías de ImageNet-21K.Si algún objeto detectado coincide con las etiquetas de imagenet_classes, se registra en el archivo de log con su marca de tiempo. La información se guarda inmediatamente en el archivo puesto que se está utilizando log.flush.

xiii. Finaliza el análisis liberando el archivo de video y notificando al usuario en streamlit que el análisis ha concluido y se retorna la ruta del archivo de texto generado.

```python
def detect_objects_in_video(video_path, model, feature_extractor, log_file="object_log.txt"):
    st.write(f"Iniciando el analisis...")
    video = cv2.VideoCapture(video_path)
    fps = video.get(cv2.CAP_PROP_FPS)
    frame_count = 0
    detected_objects = set()  # Para registrar objetos ya detectados

    with open(log_file, 'w') as log:
        while True:
            ret, frame = video.read()
            if not ret:
                break

            frame_count += 1
            # Extraer segundos exactos del frame actual y convertir a hh:mm:ss
            seconds = frame_count / fps
            time_format = str(timedelta(seconds=int(seconds)))

            # Convertir el frame a formato de entrada para el modelo
            img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            inputs = feature_extractor(images=img_rgb, return_tensors="pt")

            # Realizar la predicción
            outputs = model(**inputs)
            logits = outputs.logits
            predicted_class_idx = logits.argmax(-1).item()
            predicted_label = model.config.id2label[predicted_class_idx].lower()

            # Clasificar si es un objeto peligroso
            for label in imagenet_classes:
                if label in predicted_label and label not in detected_objects:
                    detected_objects.add(label)
                    log_entry = f"Objeto peligroso detectado: {model.config.id2label[predicted_class_idx]}, Tiempo: {time_format}"
                    log.write(log_entry + "\n")
                    log.flush()


    video.release()
    st.write(f"El analisis a concluido. El Log puede encontrarse en {log_file}")
    return log_file
```

Figura 63 Función encargada de detectar las armas dentro del video de interés y retornar la ruta del archivo con la descripción de las encontradas

6. Se declara la función wordcloud_from_log [Figura 64], la cual, toma como parámetro de entrada, la dirección del archivo de log, que contiene los objetos detectados y relacionados con armas. A partir de la información del archivo, se genera un mapa de palabras, destacando los términos más frecuentes, y retorna el objeto generado.

```python
def wordcloud_from_log(log_file):
    with open(log_file, 'r') as log:
        text = log.read()
    wordcloud = WordCloud(width=800, height=400, background_color="white").generate(text)
    return wordcloud
```

Figura 64 Función encargada de generar el mapa de palabras de objetos detectados y relacionados con armas

7. Por último, el código se encarga de invocar las funciones previamente mencionadas [Figura 65] y declaradas y darles un formato dentro de la interfaz de streamlit. El flujo implementado para este propósito se describe a continuación:
    i. Se define el título que llevará la interfaz, el cual, será mostrado en streamlit.

ii. Se agrega un cuadro de texto que instruye al usuario para ingresar la dirección web del video de YouTube que desea analizar y se almacena en la variable youtube_url.
iii. Se verifica que la dirección web del video ingresada por el usuario sea válida. También, se define la ruta donde se guardará el video descargado (video_path).
iv. En caso de que el usuario haga clic en el botón *Descargar y Procesar el Video en fotogramas*, se realiza el siguiente flujo:
    1. Se manda llamar la función download_video, enviando como parámetro la dirección web del video de YouTube y la ruta donde será guardado el archivo descargado.
    2. Se invoca la función load_model, la cual, retornara el modelo de clasificación y el extractor de características necesarios para analizar los fotogramas.
    3. Se verifica si el archivo contenido en la variable log_file contiene información, de ser así, significa que se encontraron armas en los videos, por lo cual se procede a hacer lo siguiente:
        a. Se muestra al usuario una imagen de advertencia con la descripción: *¡CUIDADO! Se detectaron armas en el video :(.*
        b. Se manda llamar la función que genera el mapa de palabras basado en los objetos detectados y se procede a mostrar en la interfaz de streamlit
        c. Se muestra el contenido del archivo de log, el cual, incluye todas las armas detectadas en los fotogramas del video y las marcas de tiempo correspondientes donde se encontraron.

    4. En caso contrario, si el archivo log se encuentre vacío, se muestra una imagen que muestra la aprobación del video con la descripción: *¡VIDEO APROPIADO! No se detectaron armas :)*

```python
st.title("Detección de Armas en Videos de YouTube")

# URL del video de YouTube
youtube_url = st.text_input("Ingresa la URL del video de YouTube para que sea analizado:")

if youtube_url:
    video_path = "video.mp4"
    if st.button("Descargar y Procesar el Video en fotogramas"):
        # Descargar el video
        download_video(youtube_url, video_path)
        # Cargar el modelo
        model, feature_extractor = load_model()
        # Detectar objetos en el video
        log_file = detect_objects_in_video(video_path, model, feature_extractor)
        with open(log_file, 'r') as log:
            log_content = log.readlines()

        if log_content:
            st.image("warning.jpg", caption="¡CUIDADO! Se detectaron armas en el video :( .", width=300)
            # Generar y mostrar el mapa de palabras
            st.write("Mapa de palabras: Armas detectadas en el video")
            wordcloud = wordcloud_from_log(log_file)
            plt.figure(figsize=(10, 5))
            plt.imshow(wordcloud, interpolation="bilinear")
            plt.axis("off")
            st.pyplot(plt)
            st.write("******************** Contenido del Log ********************\n")
            st.code("".join(log_content), language="text")
        else:
            st.image("safe.jpg", caption="¡VIDEO APROPIADO! No se detectaron armas :) .", width=300)
```

Figura 65 Flujo del código encargado de invocar las funciones definidas previamente y visualizarlas por la interfaz de streamlit

5 RESULTADOS Y DISCUSIÓN

a. Resultados

Se analizaron veinticuatro videos basados en el conjunto de datos implementados en la investigación pasada (Youtubers not madeForKids [85]), de los cuales, la mitad de ellos representaba videos inquietantes y la otra mitad videos adecuados.

El análisis de cada video se realizó desde streamlit, con el propósito de hacerlo más interactivo. De manera general, la interfaz de usuario se compone de los siguientes elementos: Título de la aplicación, un cuadro de texto para ingresar la dirección web del video de YouTube, un botón para ejecutar y procesar el análisis del video y un log.

Este log muestra información detallada sobre el proceso de análisis del video incluyendo el título del video descargado, la confirmación de la descarga junto con la ruta del archivo, un mensaje que indica el inicio y la conclusión del análisis, y la ruta del archivo que guarda el log del video analizado [Figura 66].

Figura 66 Interfaz general para el análisis de videos de YouTube en streamlit

Ahora bien, en caso de que se identifique en el video analizado contenido relacionado con armas, la aplicación en streamlit, mostrará una imagen de advertencia, un mapa de palabras que incluya todas las armas identificadas y que coincidan con las categorías definidas previamente, y, por último, desplegará el log. Este log contiene información sobre los objetos peligrosos encontrados y el tiempo exacto dentro del video en el que pueden ser localizados [Figura 67].

¡CUIDADO! Se detectaron armas en el video :(.

Mapa de palabras: Armas detectadas en el video

********** Contenido del Log **********

```
Objeto peligroso detectado: candle, taper, wax light, Tiempo: 0:00:02
Objeto peligroso detectado: cannon, Tiempo: 0:00:06
Objeto peligroso detectado: guillotine, Tiempo: 0:00:08
Objeto peligroso detectado: torch, Tiempo: 0:00:14
Objeto peligroso detectado: projectile, missile, Tiempo: 0:00:16
Objeto peligroso detectado: maillot, tank suit, Tiempo: 0:00:30
```

Figura 67 Ejemplo de resultado generado por el análisis de un video en el que se detectaron armas desde la interfaz streamlit

En caso contrario, si el video analizado no contiene objetos relacionados con armas, la aplicación en streamlit mostrará una imagen indicando que el contenido es adecuado [Figura 68].

Figura 68 Ejemplo de resultado generado por el análisis de un video en el que NO se detectaron armas desde la interfaz streamlit

El código desarrollado para esta aplicación está disponible en el repositorio https://github.com/kappagl/DeteccionClasificacionObjetos.git.

b. Discusión

Se compararon los resultados de los veinticuatro videos utilizando en el modelo ViT para clasificación con la investigación previa. De los doce videos etiquetados manualmente como adecuados, se encontró que, cuatro fueron clasificados como falsos positivos, es decir, como inadecuados. Esto permite concluir que el modelo podría obtener mejores resultados si, además de haberse preentrenado con ImageNet-21K, se ajustara utilizando los videos disponibles en la colección de datos generada para la investigación *Youtubers not madeForKids*.

Un área de oportunidad de mejora en las actividades de clasificación en imágenes, son las colecciones de datos, incluyendo en ellas tópicos de riesgo, como drogas o contenido explicito, ya que estos permitirían aplicar algoritmos de manera más eficiente y desarrollar filtros más efectivos.

Se descubrió que, gran parte de los videos mencionados en la colección de datos de la investigación pasada que aún se encuentran disponibles en la plataforma de YouTube están relacionados con drogas. Por ello, contar con modelos capaces de identificarlas y clasificarlas ayudaría a crear filtros más robustos para abordar este problema.

Se considera que la manera más robusta de abordar un problema tan complejo como el filtrado de contenido es mediante la aplicación de modelos multimodales. Esto se debe a que se descubrió que, aunque algunos videos no muestran imágenes de riesgo, contienen audios que no son aptos para menores de edad. Por este motivo, detectar audio y video de forma conjunta, permitiría desarrollar filtros más efectivos y robustos.

6 CONCLUSIONES

a. Conclusiones

- Se cumplió el objetivo de analizar contenido evaluado en investigaciones previas para determinar si un video es adecuado o inadecuado para niños. Sin embargo, debido a la implementación propuesta la cual, está basada en el uso de modelos ViT preentrenados, no fue posible abarcar todos los lineamientos establecidos en la ley COPPA. Esto se debe a que el conjunto de datos de ImageNet-21K no incluye representación para todas las categorías requeridas.

- Se logró comparar los resultados de la implementación propuesta con los generados en el estudio Youtubers not madeForKids. Logrando concluir que, la manera más robusta de desarrollar un modelo capaz de filtrar contenido de manera más precisa es mediante el uso de modelos multimodales que analicen tanto el audio como las imágenes del video. El uso de este tipo de modelos lograría cubrir casos en los que no se detectan objetos de riesgo de manera visual, pero si, lenguaje inapropiado de manera auditiva y viceversa.

- Se concluye que, para poder generar modelos más robustos que filtren de manera efectiva videos aún disponibles en YouTube, se requiere de conjuntos de datos que incluyan categorías e imágenes relacionados con temas de riesgo, como drogas o contenido explicito. Entrenar modelos con estos datos es esencial para que se pueda identificar y clasificar dicho contenido con precisión y contar con un conjunto de datos público y extenso que permita implementarse en múltiples modelos, lograra comparar en cual de todos se obtiene una mejor respuesta.

- Se demostró que implementar un modelo de aprendizaje profundo es un proceso rápido, eficiente y efectivo. Esto es posible gracias al esfuerzo de plataformas como Hugging Face, PyTorch y Ultralytics, que permiten a los investigadores realizar implementaciones de manera veloz para abordar problemas complejos de manera sencilla.

b. Trabajo Futuro

- Desarrollar una página web que no solo sea capaz de analizar videos de YouTube, sino que también, pueda procesar material audiovisual proveniente de cualquier red social.

- Realizar un ajuste fino del modelo de ViT preentrenado, reentrenando sus primeras capas con un conjunto de datos personalizado. Este conjunto incluiría categorías que no fueron cubiertas en esta investigación, como violencia física entre personas, drogas o contenido explícito, para mejorar su capacidad de clasificación.

- Desarrollar un conjunto de datos que incluya contenido de riesgo y esté disponible públicamente para que sirva de base en evaluación de modelos que busquen filtrar este tipo de modelos.

- Investigar y evaluar modelos multimodales los cuales permitan procesar tanto el audio como las imágenes de los videos para realizar una clasificación más robusta y comparar los resultados obtenidos con los de la implementación expuesta en el presente trabajo.

BIBLIOGRAFÍA

[1] Rideout, V., Peebles, A., Mann, S., & Robb, M. B. (2022). Common Sense census: Media use by tweens and teens, 2021.San Francisco, CA: Common Sense.

[2] Política sobre seguridad infantil - Ayuda de YouTube. Recuperado de https://support.google.com/youtube/answer/2801999?sjid=1337871404522550944-NC#zippy=%2Ccontenido-con-restricci%C3%B3n-de-edad%2Ccontenido-con-menores

[3] https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa

[4] Configurar la audiencia de un canal o un vídeo - Ordenador - Ayuda de YouTube. Recuperado de https://support.google.com/youtube/answer/9527654

[5] Redmon, J. et al. (2016) You only look once: Unified, real-time object detection, arXiv.org. arXiv: 1506.02640.

[6] Redmon, J. and Farhadi, A. (2018) Yolov3: An incremental improvement, arXiv.org. arXiv: 1804.02767.

[7] Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y.M. (2020) Yolov4: Optimal Speed and accuracy of object detection, arXiv.org. arXiv: 2004.10934.

[8] Girshick, R. (2015) Fast R-CNN, arXiv.org. arXiv: 1504.08083.

[9] Liu, W. et al. (2016) SSD: Single shot multibox detector, arXiv.org. arXiv: 1512.02325.

[10] Zisserman, A., Pinz, A. and Feichtenhofer, C. (2018) Detect to track and track to detect, arxiv.org. arXiv: 1710.03958.

[11] Alcantarilla, P.F. et al. (2018) Street-view change detection with deconvolutional networks - Autonomous Robots, SpringerLink. Available at: https://link.springer.com/article/10.1007/s10514-018-9734-5#notes.

[12] Hu, H. et al. (2021) Video swin transformer, arxiv.org. arXiv :2106.13230 .

[13] Houlsby, N. et al. (2021) AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE, arxiv.org. Available at: http://arxiv.org/pdf/2010.11929.

[14] Carion, N. et al. (2020) End-to-end object detection with Transformers, arXiv.org. arXiv:2005.12872.

[15] Torresani, L., Wang, H. and Bertasius, G. (2021) Is space-time attention all you need for video, arxiv.org. arXiv: 2102.05095.

[16] Abu-El-Haija, S. et al. (2016) YouTube-8M: A large-scale video classification benchmark, arXiv.org. arXiv:1609.08675.

[17] Fei-Fei, L. et al. (2015) Imagenet Large Scale Visual Recognition Challenge, arxiv.org. arXiv: 1409.0575.

[18] Gu, C. et al. (2018) Ava: A video dataset of Spatio-temporally localized atomic visual actions, arXiv.org. arXiv: 1705.08421.

[19] Wu, H. et al. (2019) FASTFCN: Rethinking dilated convolution in the backbone for semantic segmentation, arXiv.org. arXiv: 1903.11816.

[20] Pavllo, D. et al. (2019) 3D human pose estimation in video with temporal convolutions and semi-supervised training, arXiv.org. arXiv:1811.11742.

[21] Zhao, Y. et al. (2021) A Battle of Network Structures: An empirical study of CNN, Transformer, and MLP, arXiv.org. arXiv: 2108.13002.

[22] Brand assets - hugging face (no date) Hugging Face – The AI community building the future. Available at: https://huggingface.co/brand.
[23] Chatting with transformers (no date) Chatting with. Available at: https://huggingface.co/docs/transformers/main/en/conversations.
[24] Kirillov, A. et al. (2019) Panoptic segmentation, arXiv.org. arXiv:1801.00868.
[25] Models - hugging face. Available at: https://huggingface.co/models.
[26] Detr (no date) DETR. Available at: https://huggingface.co/docs/transformers/main/en/model_doc/detr#resources.
[27] Goldblum, M. et al. (2023) Battle of the backbones: A large-scale comparison of pretrained models across computer vision tasks, arXiv.org. arXiv: 2310.19909.
[28] Liu, L. et al. (2019) Deep Learning for Generic Object Detection: A Survey, arXiv.org. arXiv: 1809.02165v4.
[29] Zou, Z. et al. (2023) Object detection in 20 years: A survey, arXiv.org. arXiv:1905.05055.
[30] Woo, S.S. et al. (2022) A survey of deep learning-based object detection methods and datasets for overhead imagery | IEEE Journals & Magazine | IEEE Xplore, IEEE Access. Available at: https://ieeexplore.ieee.org/document/9703336/.
[31] Richard Szeliski. Computer Vision: Algorithms and Applications. Springer, New York, 2nd edition, 2022. A free PDF version can be downloaded from https://szeliski.org/Book. (Web)
[32] M. I. M. Fakharurazi, A. Z. Jusoh, A. L. Asnawi, N. F. A. Malek, K. Abdullah and N. F. M. Azmin, "Object Detection in Autonomous Vehicles," 2023 IEEE 13th International Conference on System Engineering and Technology (ICSET), Shah Alam, Malaysia, 2023, pp. 177-181, doi: 10.1109/ICSET59111.2023.10295116.
[33] Zhang, H. et al. (2019) Eye in the sky: Drone-based Object Tracking and 3D localization, arXiv.org. arXiv: 1910.08259.
[34] Balat, M. et al. (2024) TikGuard: A deep learning transformer-based solution for detecting unsuitable TikTok content for kids, arXiv.org. arXiv:2410.00403.
[35] Qi, J. et al. (2022b) Occluded video instance segmentation: A benchmark, arXiv.org. arXiv:2102.01558.
[36] Zhou, Q. et al. (2022) TransVOD: End-to-end video object detection with spatial-temporal transformers, arXiv.org. arXiv:2201.05047.
[37] Khan, A. et al. (2020) A survey of the recent architectures of deep convolutional Neural Networks, arXiv.org. arXiv:1901.06032.
[38] Build the neural network¶ (no date) Build the Neural Network - PyTorch Tutorials 2.5.0+cu124 documentation. Available at: https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html.
[39] O'Shea, K. and Nash, R. (2015) An introduction to Convolutional Neural Networks, arXiv.org. arXiv:1511.08458.
[40] Terven, J. and Cordova-Esparza, D. (2024) A comprehensive review of YOLO architectures in Computer Vision: From YOLOV1 to Yolov8 and Yolo-Nas, arXiv.org. arXiv:2304.00501.
[41] Ren, S. et al. (2016) Faster R-CNN: Towards real-time object detection with region proposal networks, arXiv.org. arXiv:1506.01497.
[42] Rahman, F., Mubarek, Ö. and Kira, Z. (2022) On the surprising effectiveness of transformers in low-labeled video recognition, arXiv.org. arXiv:2209.07474.

[43] Qi, J. et al. (2022a) Occluded video instance segmentation: A benchmark, arXiv.org. arXiv:2102.01558.
[44] Zhong, Z. et al. (2023) A survey on Deep Learning Techniques for Action Anticipation, arXiv.org. arXiv:2309.17257.
[45] Paszke, A. et al. (2019) Pytorch: An imperative style, high-performance deep learning library, arXiv.org. arXiv:1912.01703.
[46] Torchvision (no date) torchvision - Torchvision 0.20 documentation. Available at: https://pytorch.org/vision/stable/index.html.
[47] Lightning in 15 minutes (no date) Lightning in 15 minutes - PyTorch Lightning 2.4.0 documentation. Available at: https://lightning.ai/docs/pytorch/stable/starter/introduction.html.
[48] Defining a neural network in pytorch (no date) Defining a Neural Network in PyTorch - PyTorch Tutorials 2.5.0+cu124 documentation. Available at: https://pytorch.org/tutorials/recipes/recipes/defining_a_neural_network.html.
[49] Conv2d(no date) Conv2d - PyTorch 2.5 documentation. Available at: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.
[50] Ultralytics (no date) Ultralytics/yolov5: Yolov5 🚀 in PyTorch > ONNX > CoreML > TFLite, GitHub. Available at: https://github.com/ultralytics/yolov5.
[51] Torchvision.models(no date) torchvision.models - Torchvision 0.8.1 documentation. Available at: https://pytorch.org/vision/0.8/models.html#faster-r-cnn.
[52] Transfer learning for computer vision tutorial (no date) Transfer Learning for Computer Vision Tutorial - PyTorch Tutorials 2.5.0+cu124 documentation. Available at: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html.
[53] Terven, J. et al. (2024) Loss functions and metrics in deep learning, arXiv.org. arXiv:2307.02694.
[54] Recall (no date) Recall - PyTorch-Metrics 1.5.2 documentation. Available at: https://lightning.ai/docs/torchmetrics/stable/classification/recall.html.
[55] Precision (no date) Precision - PyTorch-Metrics 1.5.2 documentation. Available at: https://lightning.ai/docs/torchmetrics/stable/classification/precision.html.
[56] Tung, C. et al. (2022) Why accuracy is not enough: The need for consistency in object detection, arXiv.org. arXiv:2207.13890.
[57] Mean-average-precision (MAP)(no date) Mean-Average-Precision (mAP) - PyTorch-Metrics 1.5.2 documentation. Available at: https://lightning.ai/docs/torchmetrics/stable/detection/mean_average_precision.html.
[58] Turner, R.E. (2023) An introduction to transformers, arXiv.org. arXiv:2304.10557v4.
[59] Bloem, P. (2019) 'Transformers from scratch', peterbloem.nl, 18 August. Available at: https://peterbloem.nl/blog/transformers.
[60] Alammar, J. (2018) 'The Illustrated Transformer', *jalammar.github.io*, 27 June. Available at: https://jalammar.github.io/illustrated-transformer/ (Accessed: 2024).
[61] Khan, S. *et al.* (2022) *Transformers in vision: A survey*, *arXiv.org*. arXiv:2101.01169.
[62] *Pytorch-transformers*(no date) *Pytorch-Transformers - pytorch-transformers 1.0.0 documentation*. Available at: https://huggingface.co/transformers/v1.2.0/index.html.
[63] *Transformers* (no date) *PyTorch*. Available at: https://pytorch.org/hub/huggingface_pytorch-transformers/.

[64] *Facebook/Detr-resnet-50 · hugging face* (no date) *facebook/detr-resnet-50 · Hugging Face*. Available at: https://huggingface.co/facebook/detr-resnet-50.

[65] Greff, K. *et al.* (2017) *LSTM: A search space odyssey*, *arXiv.org*. arXiv:1503.04069.

[66] Staudemeyer, R.C. and Morris, E.R. (2019) *Understanding LSTM -- a tutorial into long short-term memory recurrent neural networks*, *arXiv.org*. arXiv:1909.09586.

[67] *Video API* (no date) *Video API - Torchvision 0.11.0 documentation*. Available at: https://pytorch.org/vision/0.11/auto_examples/plot_video_api.html.

[68] *Transforming and augmenting images*(no date) *Transforming and augmenting images - Torchvision 0.20 documentation*. Available at: https://pytorch.org/vision/stable/transforms.html.

[69] Bewley, A. *et al.* (2017) *Simple online and realtime tracking*, *arXiv.org*. Available at: https://arxiv.org/abs/1602.00763.

[70] Broström, M. (2023) *Boxmot: A collection of Sota Real-time, multi-object trackers for object detectors*, *Zenodo*. Available at: https://zenodo.org/records/8132989.

[71] Mikel-Brostrom (no date) *Boxmot/examples/det/torchvision_boxmot.ipynb at master · Mikel-Brostrom/boxmot*, *GitHub*. Available at: https://github.com/mikel-brostrom/boxmot/blob/master/examples/det/torchvision_boxmot.ipynb.

[72] Leal-Taixé, L. *et al.* (2015) *MOTChallenge 2015: Towards a benchmark for multi-target tracking*, *arXiv.org*. Available at: https://arxiv.org/abs/1504.01942.

[73] Ultralytics (2024) *Python*, *Python - Ultralytics YOLO Docs*. Available at: https://docs.ultralytics.com/usage/python/.

[74] Ultralytics. (n.d.). Ultralytics YOLO tutorial [Notebook]. GitHub. https://colab.research.google.com/github/ultralytics/ultralytics/blob/main/examples/tutorial.ipynb#scrollTo=8Go5qqS9LbC5

[75] Zhuang, F. et al. (2020) A comprehensive survey on Transfer Learning, arXiv.org. arXiv:1911.02685.

[76] Ultralytics (2024) Object detection datasets overview, Ultralytics YOLO Docs. Available at: https://docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format.

[77] Ava: A video dataset of atomic visual action (no date b) Google. Available at: https://research.google.com/ava/index.html.

[78] Ava: A video dataset of atomic visual action (no date a) Google. Available at: https://research.google.com/ava/explore.html.

[79] Corona, K. et al. (2020) Meva: A large-scale Multiview, multimodal video dataset for activity detection, arXiv.org. arXiv: 2012.00914.

[80] (2012) UCF101: A dataset of 101 human actions classes from ... Available at: https://www.crcv.ucf.edu/papers/UCF101_CRCV-TR-12-01.pdf.

[81] Open source data labeling (no date) Label Studio. Available at: https://labelstud.io/.

[82] Scalabel (no date) Scalabel/scalabel: Scalabel: A versatile web-based visual data annotation tool, GitHub. Available at: https://github.com/scalabel/scalabel.

[83] Zaidi, S.S.A. et al. (2021) A survey of modern deep learning-based object detection models, arXiv.org. arXiv: 2104.11892.

[84] Akbari, H. et al. (2021) Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text, arXiv.org. arXiv:2104.11178.

[85] Gkolemi M, Papadopoulos P, Markatos EP, Kourtellis N. YouTubers not madeforkids: Detecting channels sharing inappropriate videos targeting children [Internet]. arXiv.org. arXiv: 2205.13885

[86] Ultralytics (2024) Image classification datasets overview, Ultralytics YOLO Docs. Available at: https://docs.ultralytics.com/datasets/classify/#how-do-i-structure-my-dataset-for-yolo-classification-tasks.

[87] Google/VIT-base-patch16-224 · hugging face (no date) google/vit-base-patch16-224 · Hugging Face. Available at: https://huggingface.co/google/vit-base-patch16-224.