UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ANÁLISIS DE ARQUITECTURAS CNN DEL ESTADO DEL ARTE EN EL RECONOCIMIENTO DE ACTIVIDADES EN VIDEO

POR:

DAVID ALEJANDRO SILVA CARNERO

TESIS, TESINA O ESTUDIO DE CASO PRESENTADO COMO REQUISITO PARA OBTENER EL GRADO DE

MAESTRO EN INGENIERIA EN COMPUTACIÓN

CHIHUAHUA, CHIH., MÉXICO **MARZO DE 2021**

ANÁLISIS DE ARQUITECTURAS CNN DEL ESTADO DEL ARTE EN EL RECONOCIMIENTO DE ACTIVADADES EN VIDEO. Tesis presentada por David Alejandro Silva Carnero como requisito parcial para obtener el grado de Maestro en Ingeniería en Computación, ha sido aprobada y aceptada por:

M.I. Javier González Cantú
Director de la Facultad de Ingeniería

Dr. Alejandro Villalobos Aragón
Secretario de Investigación y Posgrado

M.S.I. Karina Rocío Requena Yáñez
Coordinador(a) Académico

Dr. Alain Manzo Martínez
Director de Tesis

Marzo 2021
Fecha

Comité:
Dr. Alain Manzo Martínez
Dr. Luis Carlos González Gurrola
Dr. Luis Fernando Gaxiola Orduño
Dra. Graciela Ramírez Alonso

David Alejandro Silva Carnero
Calle Centenario #234 Col. Revolución, C.P. 31135, Chihuahua, Chih.

marzo 2021

UNIVERSIDAD AUTÓNOMA DE
CHIHUAHUA

22 de marzo de 2021

DAVID ALEJANDRO SILVA CARNERO.
Presente

En atención a su solicitud relativa al trabajo de tesis para obtener el grado de maestría en ingeniería en computación, nos es grato transcribirle el tema aprobado por esta dirección, propuesto y dirigido por el director la Dr. Alain Manzo Martínez para que lo desarrolle como tesis con el título **"ANÁLISIS DE ARQUITECTURAS CNN DEL ESTADO DEL ARTE EN EL RECONOCIMIENTO DE ACTIVIDADES EN VIDEO."**

ÍNDICE DE CONTENIDO

Solicitamos a Usted tomar nota de que el título del trabajo se imprima en lugar visible de los ejemplares de las tesis.

ATENTAMENTE
"naturamsubiecitaliis"

EL DIRECTOR

M.I. JAVIER GONZÁLEZ CANTÚ

EL SECRETARIO DE INVESTIGACIÓN Y POSGRADO

DR. ALEJANDRO VILLALOBOS ARAGÓN

FACULTAD DE INGENIERÍA
Circuito No.1, Campus Universitario 2
Chihuahua, Chih., México. C.P. 31125
Tel. (614) 442-95-00
www.fing.uach.mx

Agradecimientos

Quisiera agradecer en primera instancia a mis padres, que me estuvieron apoyando durante estos dos años de desarrollo de mi tesis tanto en lo emocional como en lo económico. A mi novia por los momentos en que me acompaño hasta altas horas en la madrugada para que pudiera terminar los trabajos pendientes. También me gustaría agradecer el apoyo de mi Director de Tesis y mis Asesores por proveerme del conocimiento, sugerencias y material necesario para realizar mi tesis de la mejor manera. A la facultad de Ingeniería por brindarme un espacio en el cual pudiera trabajar sin ninguna clase de inconvenientes. A la coordinadora Karina Requena por su constante seguimiento en el transcurso de los semestres para que todo el papeleo estuviera en orden. A la institución CONACyT por proveerme de una beca mensual durante estos 2 años, la cual hizo posible mi subsistencia y sobre todo mi concentración total en el trabajo de la Universidad. Y por último quiero agradecerme a mí mismo por no desanimarme aun cuando encontraba dificultades en el camino, por no dejar que los malos momentos o circunstancias me afectaran y sobre todo por no dejar de lado mi pasión por aprender algo nuevo.

Resumen

El reconocimiento de actividades humanas en video es un tema que ha tomado el interés de la comunidad científica desde principios del siglo XXI. Con la llegada de las redes neuronales convolucionales (CNN) y las nuevas bases de datos de videos que cuentan con miles de videos y decenas o cientos de clases, se han desarrollado modelos más certeros para dicho problema. Sin embargo, no todos los modelos basados en arquitecturas CNN se desarrollan bajo las mismas condiciones, y debido a esto, es difícil saber que arquitectura realmente es la mejor para cierta base de datos. Para atacar este problema este trabajo propone un análisis del impacto que tiene el uso de 8 diversas arquitecturas CNN del estado del arte en la tarea de reconocimiento de actividades en video usando la base de datos HMDB51. Para ello se generaron distintos grupos de frames de la base de datos HDMB51 utilizando diversas técnicas de aumentación de datos. Cada arquitectura fue entrenada como un clasificador de imágenes usando los grupos de frames en consideración y se analizó su desempeño en cuanto a porcentaje de clasificación correcta (accuracy) utilizando un promediado de predicciones de los frames que correspondían a cada video. Además, se propuso un método para seleccionar los frames más representativos del video por medio de una detección de escenas y uso de una CNN, el cual de ser optimizado podría funcionar como un buen preprocesamiento para aplicarse en los videos de cualquier base de datos. Al final se utilizaron algunos métodos de ensamble de diferentes CNN con el fin de analizar si existía alguna mejora significativa al utilizar varias arquitecturas distintas para hacer una predicción. Los resultados individuales más significativos se obtuvieron utilizando la arquitectura Xception, los videos de la base de datos HMDB51 original y el grupo de frames que fue generado usando el reflejo horizontal de cada frame como técnica de aumentación de datos. El método de ensamble que dio mejores resultados fue el que hacía uso de un promediado de predicciones y consideraba el uso de las 3 mejores arquitecturas y de todos los frames del video en la predicción. Finalmente, se observó que la red EfficientNetB0 es más robusta al trabajar con datos con ruido.

Índice de Contenido

Índice de Tablas

Índice de Figuras

Índice de Ilustraciones

Capítulo 1: Introducción

En los últimos años el problema de reconocimiento de actividades en video ha recibido mucha atención por parte de los investigadores. Esto se debe a que hoy en día es común encontrar problemas relacionados con la video-vigilancia, análisis del comportamiento o Interacción Humano Computadora (HCI) [1] .

Sin embargo, no es un problema fácil de atacar ya que la naturaleza variante de los videos, lo hace un problema complejo en comparación con la clasificación de imágenes. Este problema se ha estado atacando desde los principios del siglo XXI, sin embargo, los enfoques usados en aquel entonces difieren considerablemente con las técnicas usadas en estos últimos años [1].

Anteriormente, se usaban técnicas basadas en la creación de características de forma manual usando técnicas como HOF, HOG o SIFT para modelar las actividades de un video [2] - [6] . Su principal desventaja era que dichas características tenían que ser computadas manualmente para cada base de datos, por tanto, los modelos generados a partir de ellas no eran generalizables a otras bases de datos. Si bien estas características aportaban resultados considerablemente buenos para su época, no son comparables con los resultados que se obtienen usando métodos como las CNN [1].

La principal ventaja de las CNN en comparación con una red neuronal tradicional es la inclusión de capas convolucionales compuesta por filtros y capas de agrupamiento. Esta composición las hace realmente útiles para extraer características espaciales de imágenes o frames de un video. Actualmente se han desarrollado diferentes arquitecturas de CNN para modelar las características de una imagen [7] - [11], así como trabajos utilizando ensambles de CNN con el fin de llevar a cabo tareas de segmentación de imágenes o clasificación de base de datos de imágenes o audio [12] - [15].

En la tarea de reconocimiento de actividades en video, los modelos basados en CNN [16] - [18] han demostrado ser certeros prediciendo correctamente la etiqueta de un video

en diversas bases de datos [19] [20]. Existen 3 tipos principales de enfoques basados en las CNN de los cuáles un investigador puede tomar partido para desarrollar un modelo.

Uno de los más utilizados es el enfoque basado en el desarrollo conjunto de una corriente temporal y una corriente espacial llamado enfoque “two-stream” [21]. La corriente espacial modela las características espaciales de los frames RGB, mientras que la corriente temporal modela las características que existen entre dos frames consecutivos usando flujo óptico [22] [23].

Otro enfoque popular que se ha estado utilizando es el uso de una arquitectura de CNN en 3D [24]. Su principal diferencia con respecto a una CNN normal son su entrada y la forma de los filtros. La entrada para una 3DCNN es un conjunto apilado de frames mientras que los filtros son de igual forma en 3D para poder modelar la relación existente entre los frames. A diferencia de una CNN normal, una 3DCNN requiere una cantidad mucho mayor de datos para ser entrenada debido a su gran cantidad de parámetros, y esto a su vez afecta el tiempo de ejecución. Los modelos más recientes reportan predicciones arriba del 90% para ciertas bases de datos [18].

El último enfoque está basado en la utilización de redes neuronales recurrentes (RNN), las cuales tienen la habilidad de guardar información de eventos pasados para poder tomar una decisión en tiempo presente. Esta habilidad las hace una herramienta muy útil para tareas como el reconocimiento de actividades en video. Usualmente los investigadores, utilizan una CNN para la extracción de características y un módulo RNN también conocido como LSTM para construir sus modelos. Este enfoque permite modelar las relaciones entre los frames de un video sin recurrir a costosas convoluciones en 3D o al cálculo de flujo óptico. Los modelos más recientes tienden a combinar este enfoque con algún otro [25].

Si bien todos estos enfoques han demostrado ser una herramienta muy útil en la tarea de reconocimiento de actividades en video, no todos utilizan los mismos parámetros de entrenamiento o la misma arquitectura CNN como modelo base, ni siquiera los modelos hechos a partir de un mismo enfoque. Esto hace que no sea muy claro si el desempeño de los

modelos, fue debido al preprocesamiento, a la arquitectura base o algún otro factor. En el caso de las arquitecturas CNN, es común que los investigadores tiendan a escoger aquella que obtiene un mejor desempeño en accuracy en la base de datos ImageNet como modelo base para su modelo final en la tarea de reconocimiento de actividades en video. Sin embargo, esta selección esta sesgada por resultados basados en otra base de datos y, por tanto, no asegura que su desempeño sea el mejor para la base de datos de estudio. Este es un problema que debe ser estudiado, ya que el uso apropiado de una arquitectura en particular, puede ser la clave entre mejorar o no los resultados obtenidos en el estado del arte para dicha tarea. Aunque han existido trabajos que utilizan más de una arquitectura del estado del arte para comparar el efecto del "accuracy" en el modelo final al cambiar de arquitectura, no pasan de comparar 2 o 3 [26] [27]. Es por eso que es necesario probar varias arquitecturas CNN del estado del arte bajo las mismas circunstancias al menos en una base de datos de video para analizar cual obtiene mejor resultados. Este análisis serviría como base para la selección de una o más arquitecturas CNN en la creación de modelos más certeros para el reconocimiento de actividades en video en la o las bases de datos de estudio.

Con esto en claro, el principal objetivo de este trabajo es analizar el comportamiento de diversas arquitecturas existentes del estado del arte implementadas en la librería de Keras de Tensorflow usando únicamente la base de datos HMDB51 [19]. La principal hipótesis de este trabajo radica en que las arquitecturas con el mejor desempeño en la tarea de clasificación de imágenes, tienden a tener el mejor desempeño en la tarea de reconocimiento de actividades en video para esta base de datos.

Este trabajo propone la comparación de 8 diversas arquitecturas del estado del arte siendo estas: "ResNet152" [7], "DenseNet201" [11], "Xception" [28], "InceptionV3" [29], "NASNetMobile" [30], "MobileNetV2" [31], "EfficientNetB0" [32] y "EfficientNetB3" [32]. Estas redes CNN están implementas en la librería de Keras y presumen de lograr un accuracy sobresaliente en la base de datos ImageNet, además son de reciente creación y/o rápidas de entrenar. No se incluyeron arquitecturas como AlexNet o ClarifaiNet debido a su

antigüedad, bajo accuracy en comparación a las más recientes arquitecturas y a su falta de implementación en la librería Keras. Las arquitecturas derivadas de la familia VGG (VGG16 o VGG19) también fueron excluidas debido a su bajo rendimiento. Algunas otras no fueron incluidas por falta de tiempo.

Se comparó su desempeño en lo que se refiere a tiempo de ejecución y accuracy utilizando distintos grupos de frames que fueron creados a partir de los videos de la base de datos HMDB51 [19] con diferentes técnicas de aumentación de datos. Las arquitecturas se entrenaron como clasificadores de imágenes usando los grupos de frames en consideración y su desempeño en el reconocimiento de actividades en video se evaluó utilizando un promediado de las predicciones de los frames que correspondían a cada video. Además, se elaboraron 2 nuevas bases de datos de videos a partir de los videos originales de la base de datos HMDB51 utilizando un proceso de detección de escenas y una CNN con el fin de crear videos más limpios y analizar el impacto de la calidad de los frames en las arquitecturas.

En la última parte, se construyeron diversos ensambles utilizando las arquitecturas que mostraron el mejor desempeño individual con el fin de analizar el efecto de las predicciones de una arquitectura sobre la otra.

En resumen, los principales aportes de este trabajo se describen en los siguientes 3 puntos:

- Un análisis del desempeño de 8 distintas arquitecturas del estado del arte implementadas en la librería de Keras utilizando distintos grupos de frames generados de la base de datos HMDB51.
- Un nuevo tipo de preprocesamiento aplicable a los videos basado en la detección de escenas y el uso de una CNN.
- Un análisis del impacto que se obtiene al combinar las predicciones de varias arquitecturas diferentes utilizando varios métodos de ensamble.

El trabajo se divide en los siguientes capítulos: El capítulo 2 da una revisión al estado del arte enfocado al reconocimiento de actividades en video y se divide en 3 secciones, donde

cada sección explica más a detalle los trabajos relacionados con cada enfoque. El capítulo 3 aborda todos los conceptos del marco teórico que son de relevancia para entender el contenido de esta tesis. El capítulo 4 explica las características de la base de datos HMDB51, la metodología que se utilizó para crear cada set de frames, así como la metodología para crear las distintas bases de datos con detección de escenas. Por último, se describe como se componen los distintos ensambles basados en la combinación de varias arquitecturas. El capítulo 5 se divide en dos secciones, en la primera sección se explican los experimentos realizados y se exponen los resultados obtenidos para cada experimento. La segunda sección hace un análisis de los resultados, exponiendo las características claves de las arquitecturas que mostraron mejor desempeño, así como también se discuten los resultados obtenidos con los videos creados por la detección de escenas y los resultados obtenidos con los ensambles usados. El capítulo 6 da las conclusiones más importantes del trabajo y comenta el trabajo a futuro del autor.

Capítulo 2: Estado del Arte

Este capítulo expone los trabajos más relevantes en cuanto al estado del arte en el reconocimiento de actividades en video. El capítulo se divide en 3 secciones, cada sección expone los trabajos más importantes de uno de los tres tipos de enfoques explicados con anterioridad. La primera sección habla de los trabajos relacionados al enfoque “two-stream”. La segunda sección aborda los trabajos basados en las convoluciones en 3D. Finalmente la tercera sección explica las contribuciones de los trabajos hechos a partir de redes recurrentes.

2.1 Enfoque Two-Stream

Simonyan et al. [21] propusieron un nuevo enfoque basado en la utilización de dos corrientes llamado “two-stream”. Una corriente se entrena a partir de frames RGB y otra corriente a partir de frames de flujo óptico. Cada corriente tiene una arquitectura convolucional llamada ClarifaiNet, la cual se encarga de extraer las características correspondientes a su modalidad de entrada. La arquitectura de cada flujo contenía 6 capas convolucionales y 2 capas densas. Utilizaron la base de datos HMDB51 que cuenta con 51 clases y 3 sets de entrenamiento-prueba y la base de datos UCF101 que cuenta con 101 clases y 3 sets de entramiento-prueba en sus experimentos. Obtuvieron un accuracy promedio en los 3 sets de prueba de 59.4% para la base de datos HMDB51 y un 88.0% para la base de datos UCF101.

Limin et al. [26] propusieron utilizar el enfoque two-stream con diferentes arquitecturas del estado del arte. Utilizaron diferentes técnicas de aumentación de datos y mencionaron algunas prácticas que dan buenos resultados durante el entrenamiento como hacer el entrenamiento en paralelo en múltiples GPU, usar una taza de aprendizaje baja, entre otras. La base de datos que utilizaron fue la UCF101 y sus resultados más sobresalientes en accuracy fueron un 88.0%, 89.3% y 91.3% como promedio en los 3 sets de prueba de la UCF101 utilizando las arquitecturas ClarifaiNet, GoogleNet y VGG16 respectivamente.

Wang et al. [33] se basaron en el enfoque propuesto por [21] y optaron por dividir el video de entrada en 3 segmentos y seleccionar aleatoriamente un fragmento corto de cada segmento. Cada fragmento tenía un flujo de tipo temporal y un flujo de tipo espacial, que tenían como base la arquitectura Inception-V2. Las puntuaciones de la clase de diferentes fragmentos para un mismo tipo de red se fusionaron mediante promediado simple y después todas las predicciones de ambos tipos de red se fusionan luego para producir la predicción final usando la función Softmax. Utilizaron las bases de datos HMDB51 y UCF101 y sus mejores resultados obtuvieron un 69.4% y 94.2% de accuracy respectivamente.

Zhu et al. [17] usaron una arquitectura que combinaba los vectores de características individuales de cada frame que entregaba la red Inception-V2 en una representación de video a través de un agrupamiento piramidal que usaba una función max pooling. Esta arquitectura se repetía tanto para la red de tipo espacial como la de tipo temporal. Las predicciones de ambos de tipos de red se fusionaban por medio de promediado con peso para dar la etiqueta final. Utilizaron las bases de datos HMDB51 y UCF101 y usaron como entrenamiento previo a la base de datos Kinetics que cuenta con 400 clases y sus mejores resultados obtuvieron un 82.1% y un 98.0% de accuracy para las bases de datos HMDB51 y UCF101 respectivamente.

Xie et al. [34] utilizaron el enfoque two-stream, pero modificaron la entrada del flujo temporal para que fuera imágenes de diferencia RGB entre frames adyacentes en lugar de imágenes de flujo óptico. Además, incluyeron tanto una nueva capa dentro de la arquitectura de cada flujo llamada capa de atención piramidal multi escala la cual ayudaba a modelar las características locales y globales de las entradas, como un modelo semántico auxiliado de la etiqueta de clase para ayudar al modelo a comprender mejor la representación semántica del video. Su arquitectura base fue las ResNet101. Sus mejores resultados en accuracy fueron de 66.3% y 92.7% en las bases de datos HMDB51 y UCF101 respectivamente.

Cong et al. [16] desarrollaron un algoritmo de promediado de K pasos del modelo de tamaño de lote adaptativo para mejorar el tiempo de entrenamiento de una base de datos. Además, customizaron el optimizador Adam y propusieron el uso de un enfoque de 3 canales,

donde el tercer canal se conformaba por una red que calculaba el flujo óptico y dichas imágenes de flujo eran pasadas a la ResNet152 para obtener la etiqueta final, la cual se promediaba con las predicciones de los otros 2 canales que también usaban la ResNet152. El mejor accuracy obtenido en sus experimentos fue de 81.24% y 93.47% en las bases de datos HMDB51 y UCF101 respectivamente.

Wan et al. [1] utilizaron un enfoque two-stream con la peculiaridad de que la arquitectura de la corriente espacial era una arquitectura que soportaba convoluciones en 3D (C3D) y por tanto recibía como entrada una pila de frames “RGB. La corriente temporal seguía teniendo una arquitectura en 2D (VGG16) y recibía pilas de imágenes de flujo óptico. Los vectores de salida de la capa densa de ambas arquitecturas pasaban a un SVM para su clasificación final. Obtuvieron un accuracy del 70.2% y 94.8% para las bases de datos HMDB51 y UCF101 respectivamente.

He et al. [35] agregaron un stream adicional al enfoque two-stream, Este nuevo stream agregaba las características de un frame con la de sus dos frames vecinos para frames dentro de un mismo video. Esto con el fin de mejorar la representación de características de dicho frame. Dicho proceso se repetía n veces. Sus tres streams se basaban en la red Inception-V2. Trabajaron con las bases de datos HMDB51 y UCF101 y sus resultados más sobresalientes en accuracy fue un 73.1% y 95.1% respectivamente.

Feichtenhofer et al. [36] se enfocaron en demostrar la importancia de escoger adecuadamente donde fusionar los streams espacialmente y como fusionar los stream tanto espacialmente como temporalmente. Sus arquitecturas base fueron ClarifaiNet y VGG16. Sus mejores resultados fueron con la red VGG16 obteniendo un 92.5% y un 65.4% en las bases de datos UCF101 y HMDB51 respectivamente

2.2 Enfoque Basado En Convoluciones 3D

Ji et al. [24] fueron los primeros en diseñar una arquitectura basada en convoluciones en 3 dimensiones donde su modelo final consistía en un ensamble de arquitecturas 3D. Cada arquitectura se alimentaba de 7 frames a escala de grises, de los cuales generaba diversos mapas de características. Cada arquitectura modelaba las relaciones existentes entre dichos mapas de características y se regularizaba auxiliándose de una serie de vectores de características de una serie de frames de un mismo video extraídos a través de una bolsa de palabras. Usaron la base de datos KTH que consta de 6 clases, en donde las acciones de los videos que aparecen en dichas clases fueron realizadas por 25 personas diferentes. Su mejor accuracy fue de un 90.2%

Sun et al. [37] atacaron el problema del costo computacional de las convoluciones en 3D utilizando un operador de transmutación dentro de la arquitectura que ellos mismos diseñaron Dicho operador se incluía en una capa que convertía una convolución 3D tradicional, en una serie de convoluciones en 2D seguidas de una convolución en 1D para modelar las relaciones temporales entre los resultados de las convoluciones en 2D. Sus mejores resultados fueron un 88.1% y un 59.1% en las bases de datos UCF101 y HMDB51 respectivamente

Carreira et al. [18] combinaron el enfoque two-stream con un enfoque basado en convoluciones en 3D. En su arquitectura I3D basada en la arquitectura Inception-V1 cada stream se entrenaba con múltiples frames RGB o de Flujo Óptico, los cuales alimentaban a una arquitectura basada en convoluciones 3D. Utilizaron a su vez la base de datos de videos Kinetics que consta de 400 clases con más de 400 videos por clase como base para el entrenamiento previo del Modelo demostrando que se obtienen mejores resultados que usando el entrenamiento previo con la base de datos de ImageNet. Sus mejores resultados fueron un 97.8% y un 80.9% en las bases de datos UCF101 y HMDB51 respectivamente.

He et al [38] propusieron crear una arquitectura de alto accuracy basada en la integración de información proveniente de frames RGB, de dos tipos diferentes de imágenes de flujo óptico y de audio. Se basaron en las arquitecturas ResNeXt101 e InceptionResNetV2. Usaron a la base de datos Kinetics 400 como entrenamiento previo y su entrenamiento final y evaluación fue en la base de datos Kinetics 600 que consta con 600 clases y con al menos 600 videos por clase. Su mejor accuracy fue de 85% usando un ensamble de modelos individuales, donde cada uno fue entrenado con una modalidad diferente (RGB, flujo, audio).

Wang et al. [39] utilizaron como base la arquitectura I3D para desarrollar una arquitectura que aprendiera la representación vectorial de Fisher y la representación de bolsa de palabras de una combinación de características extraídas a partir de frames "RGB" y flujo óptico. Usaron la base de datos HMDB51 y su mejor resultado obtuvo un 82.48% de accuracy.

Piergiovanni et al. [40] desarrollaron un algoritmo evolutivo que era capaz de crear modelos convolucionales con distinto número y tipos de capas para la mejor detección de características espacio-temporales en videos y se basaron en la arquitectura Inception-V1. Su mejor resultado fue un 82.1% de accuracy en la base de datos HMDB51.

Yang et al. [41] al igual que [37] quisieron atacar el problema del costo computacional de las convoluciones en 3D por lo que utilizaron convoluciones en 3D asimétricas en una dirección con el fin de reducir el número de parámetros de una convolución 3D tradicional. Su modelo final era un ensamble de pequeñas redes llamadas "micro nets" que hacía uso de este tipo de convoluciones en 3D propuestas y se entrenaba con una entrada mixta, la cual contenía información útil de frames "RGB e imágenes de flujo óptico. Obtuvieron un accuracy de 92.6% y de 65.4% para las bases de datos UCF101 y HMDB51 respectivamente.

Stroud et al. [42] propusieron un modelo llamado D3D que se entrenaba tanto con la información de los frames "RGB" (flujo espacial) como con el conocimiento adquirido de una red temporal entrenada con imágenes de flujo óptico (flujo temporal), las cuales se obtenían usando un decodificador en las características generadas dentro del flujo espacial a

partir de los frames “RGB”. Ambos flujos se basaron en una red 3D llamada S3D-G. Lograron un 97.6% y un 80.5% de accuracy en las bases de datos UCF101 y HMDB51 respectivamente.

2.3 Enfoque Basado En Redes Recurrentes

Baccouche et al. [43] usaron una CNN en 3D que ellos mismos diseñaron para extraer características de una serie de frames apilados y el tensor de salida era pasado a una RNN con módulos LSTM para extraer las relaciones temporales entre las distintas pilas de frames. Su mejor resultado fue un 94.39% en la base de datos KTH.

Donahue et al. [44] descompusieron el video en *n* frames, cada frame entraba a una CNN para extraer sus características y después pasaba a un LSTM. Cada predicción de cada LSTM se promediaba para tener la etiqueta final del video. La arquitectura base en sus experimentos fue una combinación de la arquitectura CaffeNet y otra red propuesta por otros autores. Obtuvieron un 82.37% en la base de datos UCF101.

Sharma et al. [45] Optaron por usar un modelo basado en la atención visual, para esto usaron primero la Inception-V1 como extractor de características. La CNN entregaba un tensor de salida que pasaba a una RNN, no sin antes pasar a una filtración donde se decidía a que partes del tensor se les debía dar mayor importancia a la hora de la clasificación. Usaron la base de datos HMDB51 y obtuvieron un 41.3% de accuracy como mejor resultado.

Yue-Hei Ng et al. [27] experimentaron con diversos números de frames, diversas arquitecturas de agrupamiento de características (AlexNet e Inception-V1) y una red recurrente con el fin de modelar un mayor nivel de características temporales entre los frames del video. Utilizaron la base de datos UCF101 obteniendo un 88.6% de accuracy.

Gammulle et al. [25] propusieron fusionar las características de salida de la arquitectura VGG16 tanto de la última capa convolucional como de la primera capa densa a través de redes recurrentes con módulos LSTM. Para esto propusieron diversas formas de

aprovechar dichas características en la RNN, tanto usándolas individualmente en la RNN como usando una combinación de las mismas. Usaron las bases de datos jHMDB que consta de 923 videos divididos en 23 clases, la base de datos UCF11 que consta de 1600 videos divididos en 11 clases y la base de datos UCF Sports que cuenta con 150 videos divididos en 10 clases. Obtuvieron un accuracy de 69%, 94.6% y 99.1% respectivamente.

Ye et al. [46] combinaron el enfoque two-stream con el uso de RNN usando a la ResNet101 como arquitectura base. Para obtener la etiqueta de cada video combinaron el vector de salida de características de la última capa convolucional de la ResNet101 de cada stream y dicho vector combinado paso a una arquitectura LSTM convolucional que extraería las relaciones entre los vectores de características de cada frame. Obtuvieron un 93.9% de accuracy en la base de datos UCF101 y un 69.3% de accuracy en la base de datos HMDB51.

Jaouedi et al. [47] denotaron la importancia de tener un buen vector de características para la entrada de un clasificador por lo que propusieron un modelo que tomaba como entrada solamente el área cercana a la persona en movimiento de cada frame y lo pasaba a una red recurrente cerrada para su clasificación. Su desempeño se evaluó con las bases de datos UCF Sports, KTH y UCF101 logrando un accuracy de 89.01%, 96.30% y 89.30% respectivamente.

La Tabla 1 muestra un resumen de los trabajos del estado del arte presentados con anterioridad. La Tabla 2 muestra las características más importantes de los modelos que se usaran para hacer la comparación de los resultados propios. Se escogieron estos modelos porque pertenecen a los trabajos más citados y/o con mejor accuracy del estado del arte en relación a la base de datos HMDB51. Los primeros tres renglones pertenecen a modelos que utilizan el enfoque two-stream, los siguientes tres renglones pertenecen a modelos que utilizan el enfoque de convoluciones en 3D, finalmente los últimos dos renglones pertenecen a modelos que utilizan el enfoque de redes recurrentes.

Tabla 1: Resumen de los trabajos del estado del arte.

Enfoque	Ref.	Año	Base de Datos	Accuracy
Two-Stream	[21]	2014	UCF101/HMDB51	88.0%/59.4%
	[26]	2015	UCF101	91.3%
	[33]	2016	UCF101/HMDB51	94.2%//69.4%
	[36]	2016	UCF101/HMDB51	92.5%/65.4%
	[17]	2018	UCF101/HMDB51	98.0%/82.1%
	[16]	2019	UCF101/HMDB51	93.47%/81.24%
	[34]	2019	UCF101/HMDB51	92.7%/66.3%
	[35]	2019	UCF101/HMDB51	95.1%/73.1%
	[1]	2020	UCF101/HMDB51	94.8%/70.2%
3D CNN	[24]	2012	KTH	90.2%
	[37]	2015	UCF101/HMDB51	88.1%/59.1%
	[18]	2017	UCF101/HMDB51	97.8%/80.9%
	[38]	2018	Kinetics 600	85%
	[39]	2019	HMDB51	82.48%
	[40]	2019	HMDB51	82.1%
	[41]	2019	UCF101/HMDB51	92.6%/65.4%
	[42]	2020	UCF101/HMDB51	97.6%/80.5%
RNN	[43]	2011	KTH	94.39%
	[44]	2015	UCF101	82.37%
	[45]	2015	HMDB51	41.3%
	[27]	2015	UCF101	88.6%
	[25]	2017	JHMDB/UCF11/UCF Sports	69%/94.6%/99.1%
	[46]	2019	UCF101/HMDB51	93.9%/69.3%
	[47]	2020	UCF Sports/KTH/UCF101	89.01%/96.30%/89.30%

Tabla 2: Comparación de las características más importantes de diferentes modelos del estado del arte.

Ref.	Img. Aug. Train	Training	Test	Red Base	Acc.
[17]	Corte de esquinas, Fluctuación de Escala y Horizontal Flip	128batch, SGD, 32, 16 Epochs 25 rf/p video 25 stacks de 5 img. f.o. de 2 canales	25 rf/p video y 10 subframes por frame, 25 stacks de f.o.	InceptionV2 con Dropout	82.1%
[21]	Fluctuación RGB Horizontal Flip	256batch, SGD, 20K Epochs 1 rf/p video 1 stack de 10 img. f.o. de 2 canales	25 rf/p video y 10 subframes por frame, 25 stacks de f.o.	ClarifaiNet	59.4%
[33]	Corte de esquinas, Fluctuación de Escala y Horizontal Flip	256batch, SGD, 4.5K y 18K Epochs 3 rf/p video 1 stack de 5 img. f.o. de 2 canales	25 rf/p video y 10 subframes por frame, 25 stacks de f.o.	InceptionV2 con Dropout	69.4%
[18]	Corte Aleatorio Horizontal Flip	15batch por GPU, SGD, 5K Epochs 64 rf/p video 1 stack de 32 img. f.o. de 2 canales	250 rf/p video 25 stacks de f.o.	InceptionV2	80.9%

[42]	Corte Aleatorio Horizontal Flip	6batch, SGD, 10K y 100K Epochs 64 rf/p video	250 rf/p video	S3DG (basada en InceptionV2)	80.5%
[45]	N/A	Adam, 15 Epochs Segmentos de 30f por s. espaciados 1s. entre ellos	30f por s. espaciados 1s. entre ellos	InceptionV1 y LSTM de 3 niveles	41.3%
[46]	Corte Aleatorio Escalado Aleatorio Fluctuación RGB	Adam, 60 Epochs 25 rf/p video 25 stacks de 5 img. f.o. de 2 canales	25 rf/p video, 25 stacks de f.o.	ResNet101	69.3%

Capítulo 3: Marco Teórico

Este capítulo expone los conceptos teóricos más importantes para el entendimiento del presente trabajo. Se divide principalmente en 4 secciones. La primera sección expone las características más importantes de las redes neuronales artificiales tradicionales. La segunda y tercera sección abordan la teoría relevante a las CNN y los tipos de convolución más usados en la tarea de reconocimiento de actividades en video respectivamente. La cuarta y última sección expondrá las características más importantes de las arquitecturas utilizadas en el presente trabajo.

3.1 Redes Neuronales Artificiales Tradicionales

Las redes neuronales artificiales o "Artificial Neural Networks" (ANN) son un tipo de algoritmo de ML que está basado en el funcionamiento de las neuronas del cerebro humano. Las ANN usan ecuaciones matemáticas para aprender patrones de la observación de datos de entrenamiento y se basan en la unión de múltiples perceptrones [48].

Frank Rosenblatt desarrollo el perceptrón y lo definió como una neurona artificial que toma varias entradas y produce una salida binaria que se convierte en la entrada de otra neurona subsecuente. Los perceptrones son los bloques principales de una red neuronal. Rosenblatt también introdujo el concepto de pesos, los cuales son números que expresan la importancia de cada entrada. La salida de un perceptrón puede ser 0 o 1 dependiendo si la suma de las entradas multiplicadas por sus respectivos pesos es mayor o menor a un cierto límite, el cual puede ser configurado como un parámetro del perceptrón. Dicho esto, una red neuronal multicapa puede verse como una red de múltiples perceptrones unidos entre sí (véase Figura 1). A los perceptrones también se les conoce como neuronas. [48].

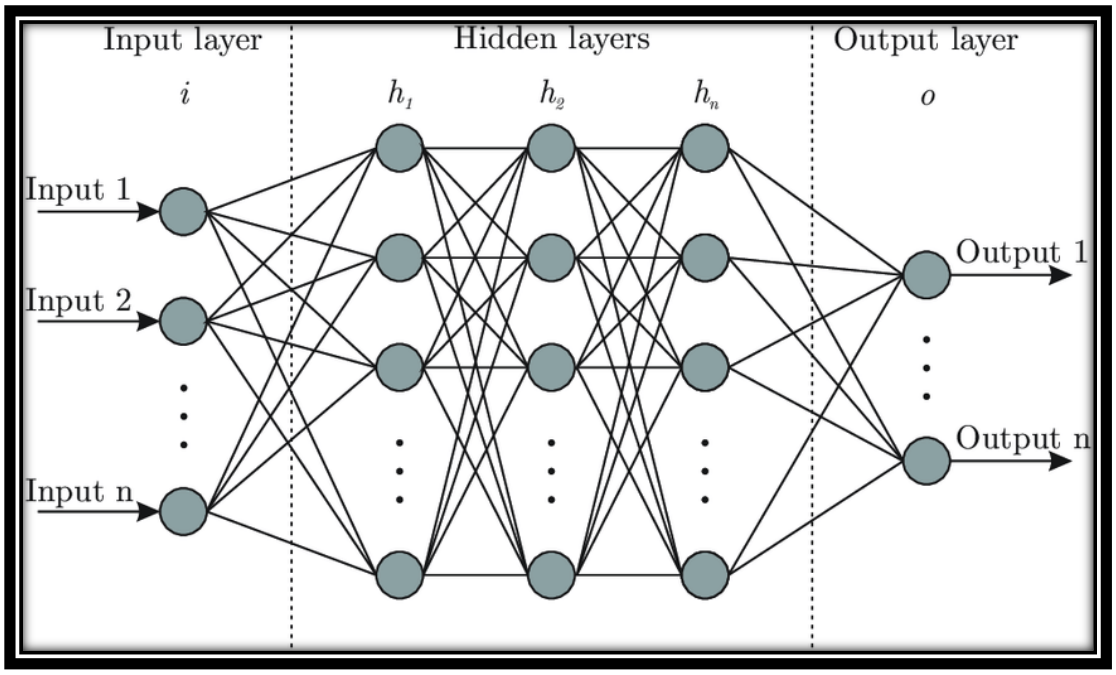

Figura 1: Estructura de una ANN.
Fuente: [49]

3.1.1 Proceso de aprendizaje de una red neuronal

Las ANN son conexiones de múltiples neuronas, donde cada una de ellas calcula una función lineal, así como una función de activación para llegar a una salida basada en ciertas entradas. Esta salida está ligada a un peso, que representa su nivel de importancia, para ser usada en los cálculos de la siguiente capa [48].

Además, estos cálculos son llevados a cabo por toda la arquitectura de la red hasta llegar a una salida final, a este proceso se le llama “forward propagation”. Esta salida se usa para determinar el error de la red en comparación con la salida objetivo, dicho error es usado para ajustar los diferentes atributos de la red (pesos y bias) para volver a empezar el proceso de “forward propagation”. Al proceso de ajuste de atributos de la red se le llama “back propagation” [48].

3.1.2 Forward Propagation

Este proceso va de izquierda a derecha a través de toda la arquitectura de la red, mientras se hacen cálculos usando los datos de entrada para llegar a una predicción que puede ser comparada con el valor objetivo. Los cálculos hechos en cada neurona incluyen una función lineal que hace la suma de la multiplicación de cada entrada por su respectivo peso, y a esa suma se le agrega un bias (el cual puede ser 0), para después pasar dicho resultado por una función de activación. La salida de la función de activación pasa como entrada de otras neuronas de una capa siguiente hasta que se llega a la capa final, la cual hace la predicción [48].

3.1.3 Back Propagation

Existen diferentes funciones de error para escoger, pero las más usadas en algoritmos de clasificación y regresión son las siguientes [48]:

- Error cuadrático medio (MSE): es usado en modelos de regresión y calcula la suma de la distancia entre el valor esperado y los valores predichos. Entre más bajo el valor entregado por la función de error mejor.
- Entropía cruzada/ Entropía cruzada multiclase: son usadas para modelos de clasificación binaria o multiclase. Mide la divergencia entre dos distribuciones de probabilidad.

Para poder actualizar los pesos de la red con el valor arrojado por las funciones de error se usan algoritmos de optimización como el descenso del gradiente o “gradient descent” [48].

Este algoritmo se basa en calcular el gradiente de cada neurona y después actualizar los pesos y los bias en la dirección opuesta al gradiente. El gradiente es multiplicado por una variable llamada tasa de aprendizaje o “learning rate”, la cual permite controlar el tamaño de los pasos tomados en cada optimización. La tasa de aprendizaje es crucial dentro del proceso

de entrenamiento, ya que previene que las actualizaciones de los pesos y bias no se desvíen demasiado de lo que deben ser, puesto que, si lo hacen, el modelo puede tardar mucho o nunca llegar a la solución [48].

3.1.4 Funciones de Activación.

Las funciones de activación se usan con el propósito de darle un comportamiento no lineal al modelo, ya que en algunas situaciones tener un modelo lineal puede no ser la mejor solución. Algunas de las funciones de activación más usadas son [48]:

- Función sigmoide: tiene forma de S y básicamente convierte los valores de entrada en probabilidades entre 0 y 1.
- Función softmax: usada comúnmente en la capa de salida de las ANN de algoritmos de clasificación. Calcula la probabilidad de la salida de ser una de las clases objetivo en comparación con las demás clases.
- Tanh: esta función representa la relación entre el seno hiperbólico y el coseno hiperbólico. Tiene forma de S y básicamente convierte los valores de entrada en probabilidades entre -1 y 1.
- ReLU: se usa convencionalmente en las capas ocultas de las ANN. Funciona de forma que, si la entrada es mayor a 0, la salida es el mismo valor de entrada; si es menor a 0, la salida es igual a 0.

3.2 Redes Neuronales Artificiales Convolucionales (CNN)

En las redes neuronales de alimentación directa tradicionales cada neurona en la capa de entrada está conectada a cada neurona de salida en la siguiente capa; a esto se le llama capa completamente conectada o capa densa (FC). Sin embargo, en las CNN, no se utilizan capas FC hasta las últimas capas de la red. De este modo, se puede definir una CNN como

una red neuronal que se intercambia en una capa especializada "convolucional" en lugar de una capa "completamente conectada" para al menos una de las capas de la red [50].

Luego se aplica una función de activación no lineal, como ReLU, a la salida de estas convoluciones y el proceso de convolución continúa (junto con una mezcla de otros tipos de capas para ayudar a reducir el ancho y la altura del volumen de entrada y ayudar a reducir el sobreajuste) hasta que finalmente se llegue al final de la red y se aplique una o dos capas FC donde se pueda obtener las clasificaciones finales de salida [50].

Cada capa en una CNN aplica un conjunto diferente de filtros, típicamente cientos o miles de ellos, y combina los resultados, alimentando la salida a la siguiente capa en la red. Durante el entrenamiento, una CNN aprende automáticamente los valores de estos filtros [50]. En el contexto de la clasificación de imágenes, una CNN puede aprender a [50]:

- Detectar bordes de datos de píxeles sin procesar en la primera capa.
- Usar estos bordes para detectar formas (es decir, "manchas") en la segunda capa.
- Utilizar estas formas para detectar características de nivel superior, como estructuras faciales, partes de un automóvil, etc. en las capas más altas de la red.

La última capa en una CNN utiliza estas características de nivel superior para hacer predicciones sobre el contenido de la imagen. En la práctica, las CNN brindan dos beneficios clave: invariancia local y composicionalidad. El concepto de invariancia local permite clasificar una imagen que contiene un objeto en particular, independientemente de en qué parte de la imagen aparezca el objeto. Se obtiene esta invariancia local mediante el uso de "capas de agrupación" que identifica regiones en el volumen de entrada con una alta respuesta a un filtro particular [50].

El segundo beneficio es la composicionalidad. Cada filtro compone un parche local de características de nivel inferior en una representación de nivel superior, similar a cómo se puede componer un conjunto de funciones matemáticas que se basan en la salida de funciones anteriores. Esta composición permite que la red aprenda características más ricas y profundamente en la red. Por ejemplo, la red puede construir bordes a partir de píxeles,

formas a partir de bordes y luego objetos complejos a partir de formas, todo de manera automatizada que ocurre naturalmente durante el proceso de capacitación El concepto de construir características de nivel superior a partir de las de nivel inferior es exactamente la razón por la cual las CNN son tan poderosas en la visión por computadora [50].

La Figura 2 muestra una CNN con dos capas convolucionales, dos capas de agrupamiento, 1 capa de aplanamiento o “flatten” que convierte todas las características de todos los mapas de características a un solo vector, 1 capa densa oculta de 128 neuronas y una capa densa de salida de 10 neuronas.

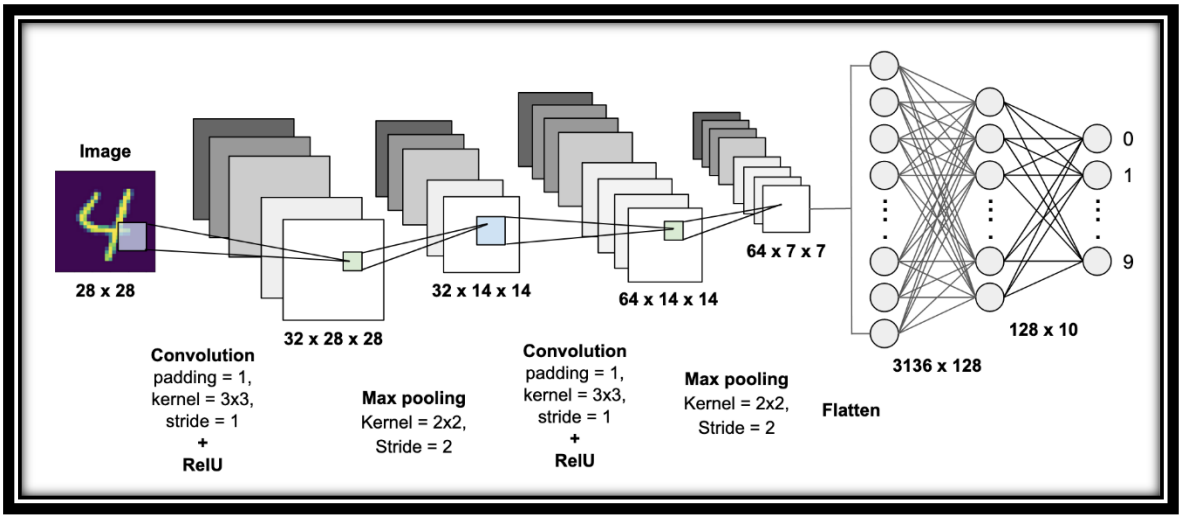

Figura 2: Ejemplo de una posible estructura de una CNN.
Fuente: [51]

3.3 Tipos de convolución

El propósito de hacer convolución es extraer características útiles de la entrada. En el procesamiento de imágenes, existe una amplia gama de filtros diferentes que se pueden elegir para la convolución. Cada tipo de filtro ayuda a extraer diferentes aspectos o características de la imagen de entrada, por ejemplo, bordes horizontales / verticales / diagonales. De manera similar, en una CNN, diferentes características se extraen mediante convolución utilizando filtros cuyos pesos se aprenden automáticamente durante el entrenamiento. Todas estas características extraídas se 'combinan' para tomar decisiones [52].

La convolución tiene algunas ventajas, como el intercambio de pesos y la traducción invariante. La convolución también tiene en cuenta la relación espacial de los píxeles. Estos podrían ser muy útiles especialmente en muchas tareas de visión por computadora, ya que esas tareas a menudo implican identificar objetos donde ciertos componentes tienen cierta relación espacial con otros componentes (por ejemplo, el cuerpo de un perro generalmente se une a una cabeza, cuatro patas y una cola) [52].

Existen cuatro conceptos importantes a la hora de hablar de una convolución los cuales son filtro, kernel, “stride” y relleno o “padding”. El filtro se refiere a la matriz multidimensional de valores utilizada en el proceso de convolución con la imagen de entrada. A cada dimensión que constituye un filtro se le denomina kernel. Cada kernel es único pues enfatiza diferentes aspectos del canal de entrada. Para un filtro 2D, el filtro es igual que el kernel. El stride define el tamaño de paso del kernel al deslizarse por la imagen. Un paso de 1 significa que el kernel se desliza a través de la imagen píxel a píxel. El padding define cómo se maneja el borde de una imagen. Una convolución con padding mantendrá las dimensiones espaciales de salida iguales a la imagen de entrada, rellenando con 0s alrededor de los límites de entrada si es necesario. Una convolución sin padding genera una imagen de salida de menor tamaño en comparación con la imagen de entrada [52].

3.3.1 Convoluciones 2D

En Deep Learning, la convolución es la multiplicación y la suma de elementos. Para una imagen con 1 canal, la convolución se muestra en la Figura 3. En la figura el filtro es una matriz de 3 x 3 con los siguientes valores [[0, 1, 2], [2, 2, 0], [0, 1, 2]]. El filtro se desliza por la entrada. En cada posición, está haciendo multiplicaciones y sumas por elementos. Cada posición de deslizamiento termina con un número. La salida final es una matriz de 3 x 3. (Para la Figura 3 stride = 1 y padding = 0) [52].

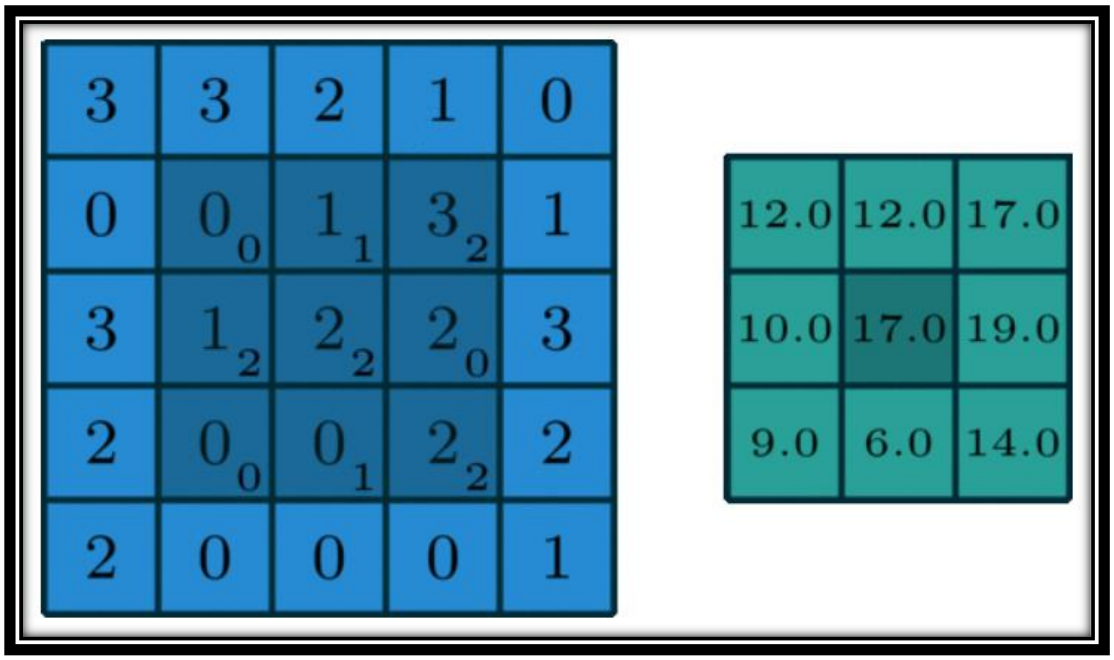

Figura 3: Proceso de convolución 2D para una entrada con un solo canal.
Fuente: [53]

En el caso de una convolución usando una imagen RGB que consta de 3 canales se utiliza un filtro de 3D con 3 dimensiones de profundidad. Para llevar a cabo una convolución multicanal cada kernel se aplica a un canal de entrada de la capa anterior para generar un canal de salida (primer canal de entrada con el primer kernel del filtro, etc.). Luego, cada uno de estos canales se suma para formar un solo canal de salida [52] (véase Figura 4, Figura 5).

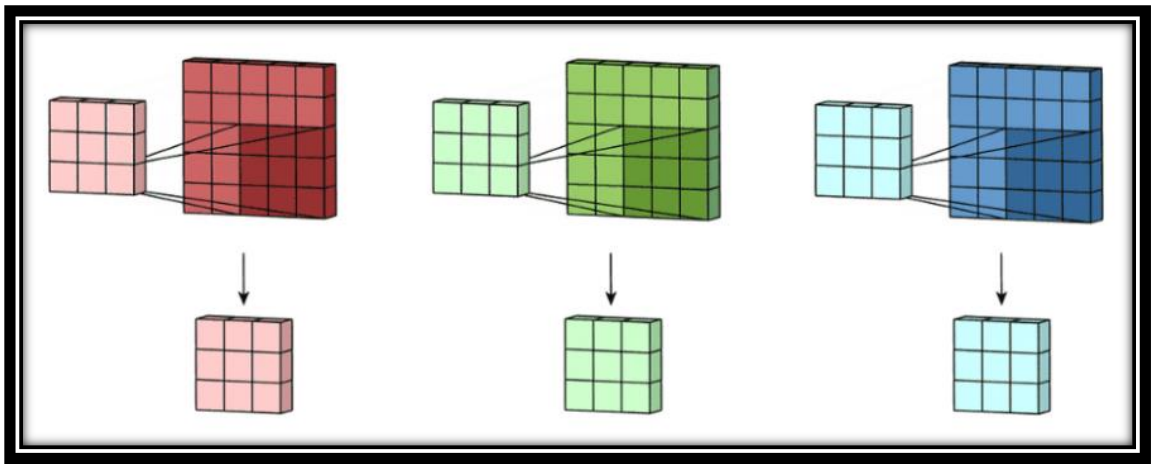

Figura 4: Parte 1 del proceso de convolución para una entrada multicanal.
Fuente: [54]

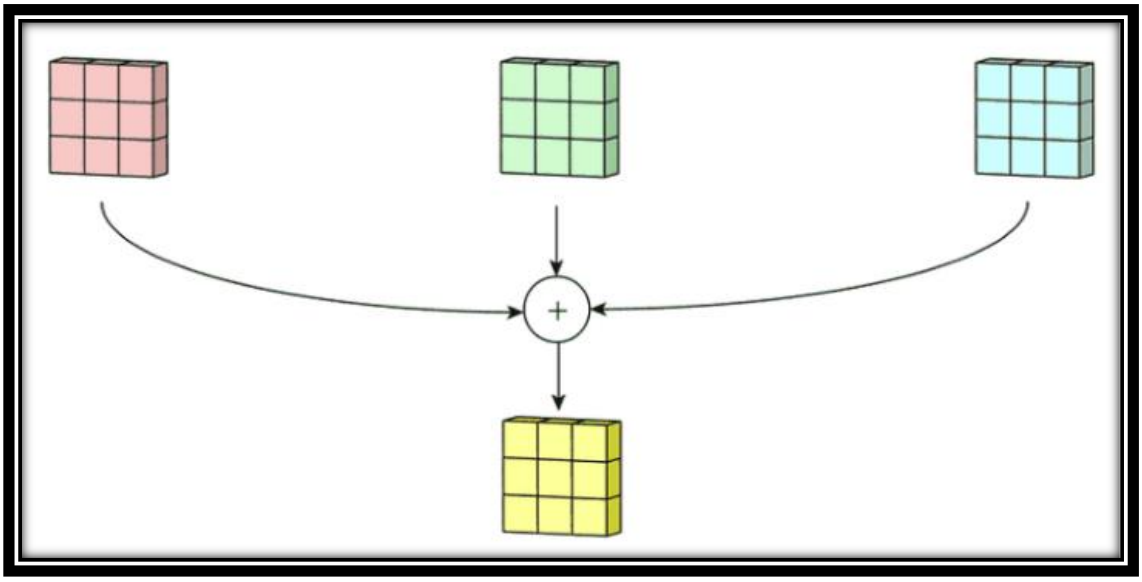

Figura 5: Parte 2 del proceso de convolución para una entrada multicanal.
Fuente: [55]

Aun cuando el filtro es 3D, este solo puede moverse en dos direcciones a través de la imagen de entrada. Es por esto que a este tipo de convolución se llama convolución 2D. Esto siempre será así si se cumple que el filtro tenga la misma profundidad que la cantidad de canales en la imagen de entrada [52].

3.3.2 Convoluciones 1x1

Las convoluciones 1x1 son útiles para entradas que tienen más de un canal o capa de profundidad y básicamente son una herramienta muy útil para reducir la dimensionalidad de los datos de entrada. Esto es posible debido a que el filtro de 1x1 tiene las mismas dimensiones en profundidad que los datos de entrada, y esto hace posible que agrupe las características en profundidad de un solo bloque. Además, es posible aplicar una función de no linealidad después de dicha convolución para que la red aprenda funciones más complejas [52] (véase Figura 6).

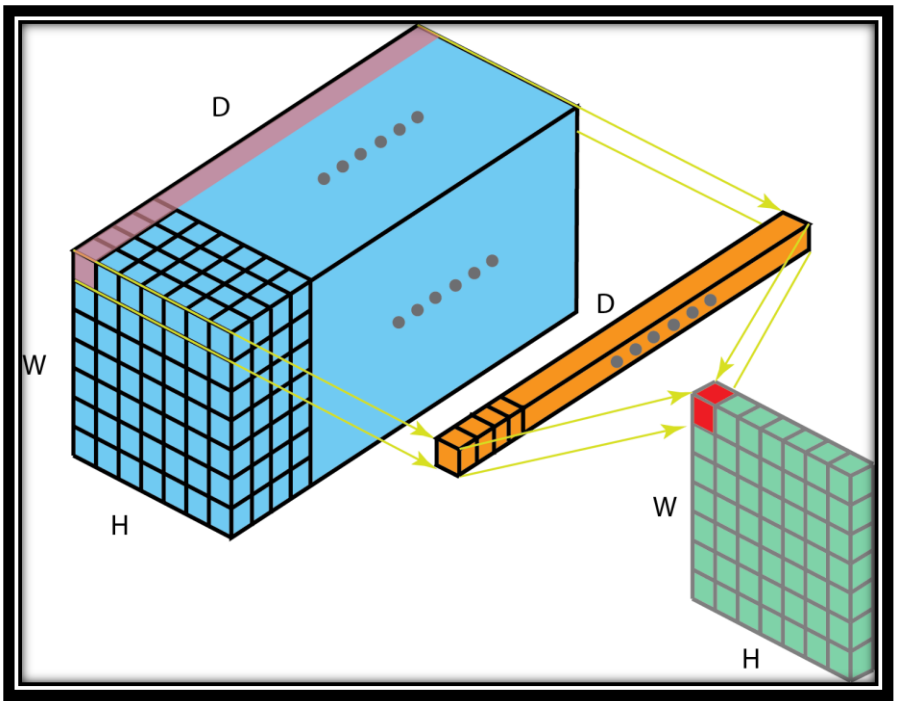

Figura 6: Convolución 1x1.
Fuente: [56]

3.3.3 Convoluciones Separables en Profundidad

Las convoluciones separables en profundidad (DSC) son convoluciones hechas en dos pasos. El primer paso es una convolución 2D que consiste en convolucionar cada capa de profundidad de la entrada con un solo kernel de tamaño KxKx1, por lo tanto, si existen N capas de profundidad en la entrada, habrá N kernels de tamaño KxKx1. El segundo paso consiste en aplicar m convoluciones de tamaño 1x1xN por toda el área del mapa de características resultantes del paso anterior. Este tipo de convolución se utiliza para optimizar el número de multiplicaciones de una convolución 2D, y esto hace a los modelos que la utilizan más rápidos y ligeros [52].

La Figura 7 muestra un ejemplo de una convolución separable en profundidad con una entrada de 7x7x3 y usando 128 filtros de 1x1x3 para generar un mapa de características final de 5x5x128.

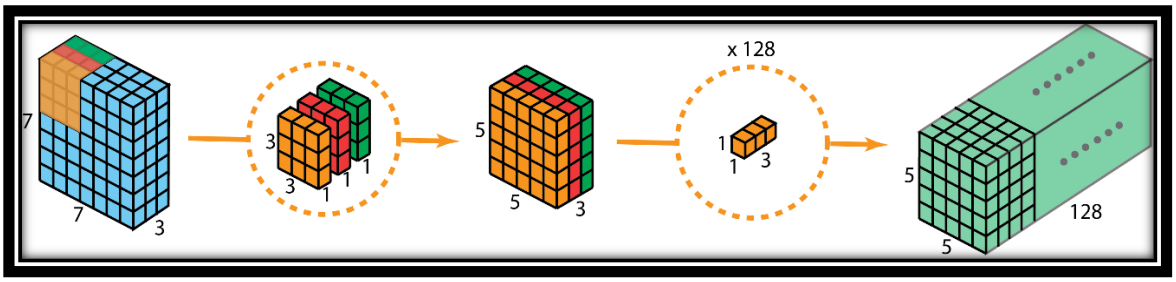

Figura 7: Convolución Separable en Profundidad.
Fuente: [57]

3.4 Arquitecturas CNN del estado del arte.

3.4.1 ResNet152

La arquitectura ResNet152 consta de 152 capas, utiliza “Batch Normalization” (BN) después de cada convolución y utiliza convoluciones 1x1 y 3x3 además de avg pooling en su mayoría con excepción en la primera convolución donde se utiliza una convolución de 7x7 y max pooling (véase Figura 8).

Las redes desprendidas de la familia ResNet fueron las primeras en proponer el uso de conexiones residuales para resolver el problema de desvanecimiento del gradiente en arquitecturas muy profundas. Su estructura se basa en la utilización de conexiones residuales en dos bloques principales (el bloque identidad y el bloque proyección o convolucional) (véase Ilustración 1).

El bloque identidad es utilizado cuando el tamaño de la entrada y la salida es identifico por lo que la conexión residual es la función identidad. El bloque convolucional se utiliza cuando la entrada no tiene las mismas dimensiones que la salida por lo que la conexión residual tiene una convolución intermedia que se encarga de hacer que ambas dimensiones sean iguales [7].

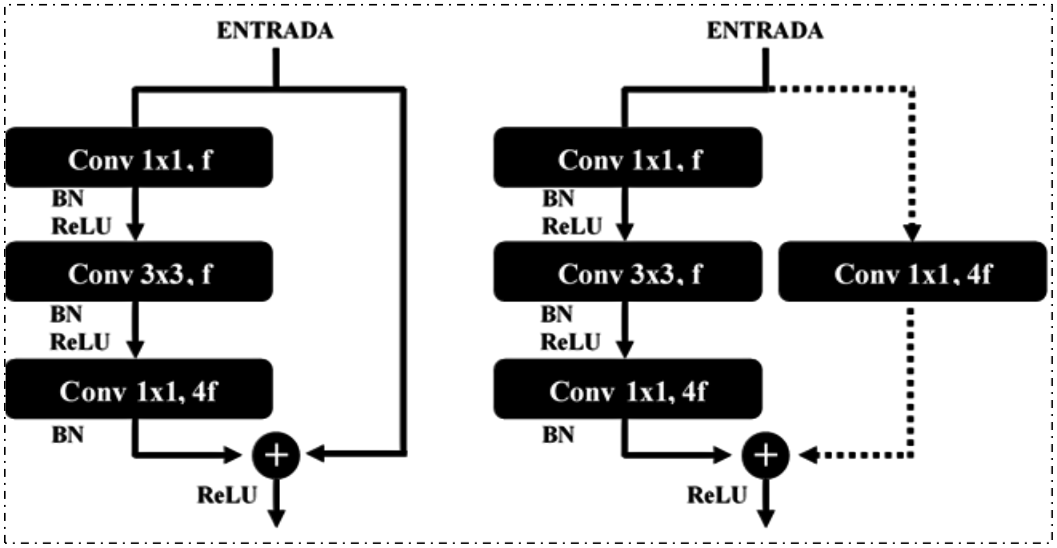

Ilustración 1: Bloque Identidad a la izquierda, bloque convolucional a la derecha.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 8: Arquitectura interna de las redes ResNet.
Fuente: [7]

3.4.2 InceptionV3

La arquitectura InceptionV3 cuenta con 48 capas, utiliza BN después de cada convolución, hace uso de operaciones max pooling y avg pooling y utiliza distintos módulos llamados Inception para formar su arquitectura (véase Figura 9). Dichos módulos se caracterizan por concatenar los mapas de características obtenidos a través del uso de 3 operaciones de convoluciones con diferentes tamaños de kernels y una operación de agrupamiento.

Para regularizar dicha arquitectura incorporaron tanto un clasificador intermedio como un componente regularizador en la función de perdida llamado suavizador de etiquetas el cual previene el overfitting [29] (véase Figura 10).

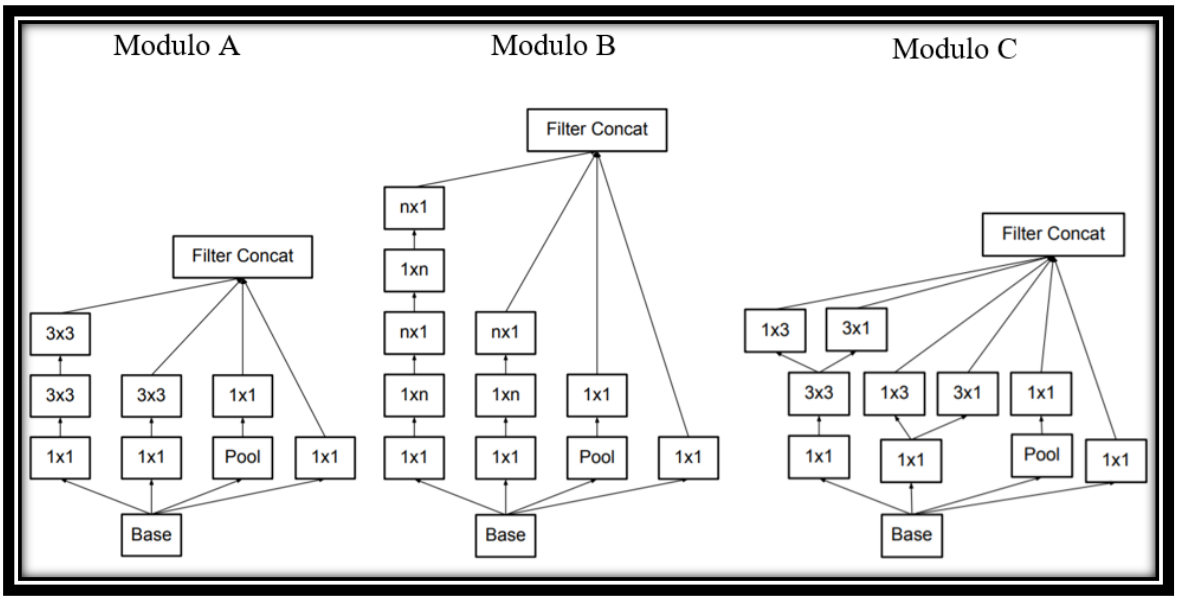

Figura 9: Módulos A,B y C de la red Inception-V3.
Fuente: [29]

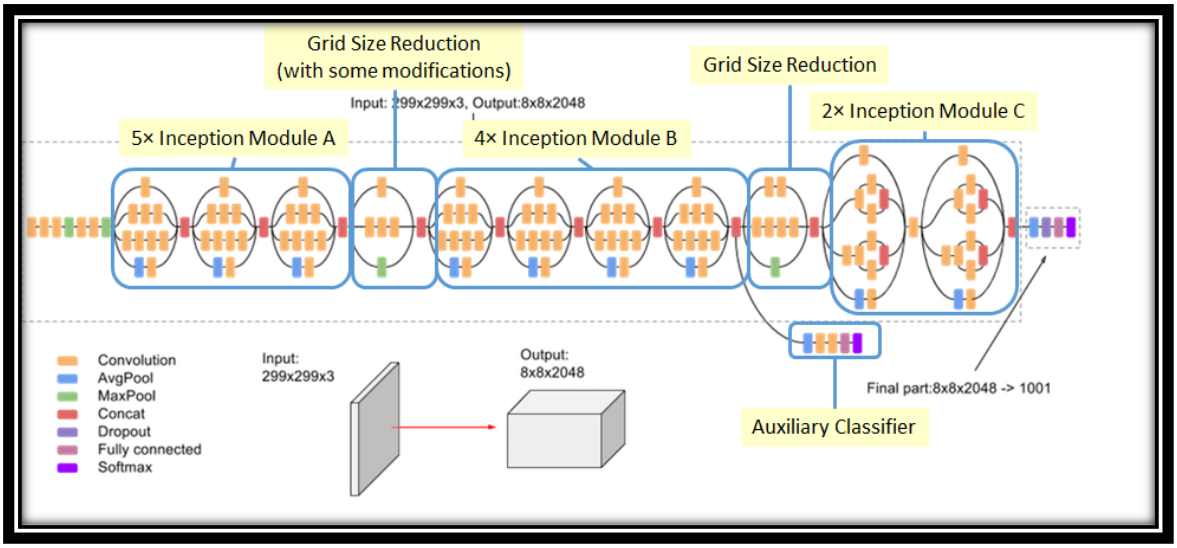

Figura 10: Arquitectura interna de la red Inception-V3.
Fuente: [58]

3.4.3 DenseNet201

La arquitectura DenseNet201 consta de 201 capas, utiliza BN y ReLU antes de cada convolución y utiliza convoluciones 1x1 y 3x3 además de avg pooling en su mayoría con excepción en la primera convolución donde se utiliza una convolución de 7x7 seguido de un max pooling (véase Figura 12).

Las redes desprendidas de la familia DenseNet se caracterizan por usar bloques densos que conectan los mapas de características anteriores a los mapas resultantes de la convolución actual (véase Figura 11). Dentro de cada bloque denso, existe una taza de crecimiento k para cada capa de filtros, la cual controla el número de mapas de características nuevas que aporta cada bloque de filtros (k=32 para todos los bloques y modelos de la DenseNet).

Este proceso de conexión entre mapas de características otorga mayor robustez al modelo puesto que al fusionar las características de las capas de varios niveles anteriores obtiene una mejor representación de la imagen. Además, el error puede ser propagado más fácilmente a las capas más profundas y el número de parámetros también es disminuido debido al hecho de que se pueden utilizar pocos filtros dentro de cada capa debido a este tipo de conexión [11].

3.4.4 Xception

La arquitectura Xception cuenta con 71 capas, utiliza DSC invertidas (véase Figura 13), utiliza max pooling, utiliza BN después de cada convolución, y sus convoluciones son solamente 1x1 y 3x3. No utiliza funciones de activación intermedias entre los dos pasos de una DSC a diferencia de la red MobileNetV2. Utiliza conexiones residuales, las cuales aportan una gran mejora con respecto a una Xception con ausencia de ellas (véase Figura 14). Como punto de comparación una red InceptionV3 obtiene mejores resultados que una red Xception sin conexiones residuales [28].

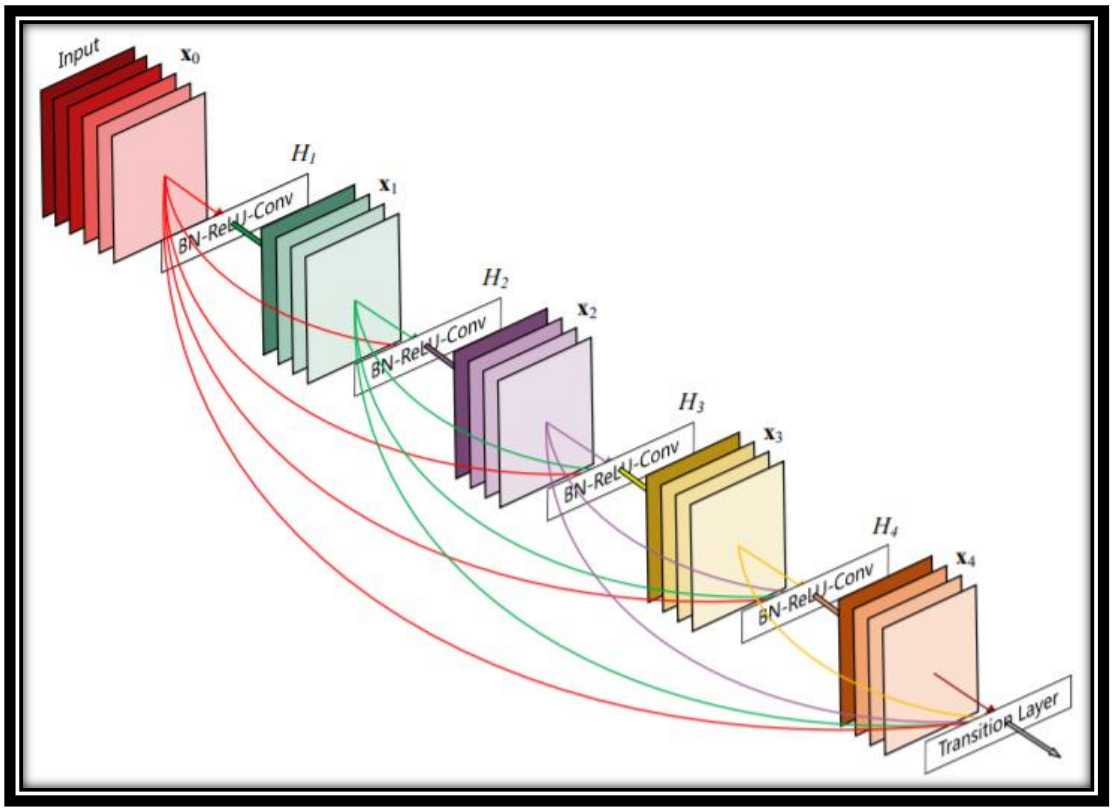

Figura 11: Bloque denso en las redes DenseNet.
Fuente: [11]

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figura 12: Arquitectura interna de las redes DenseNet.
Fuente: [11]

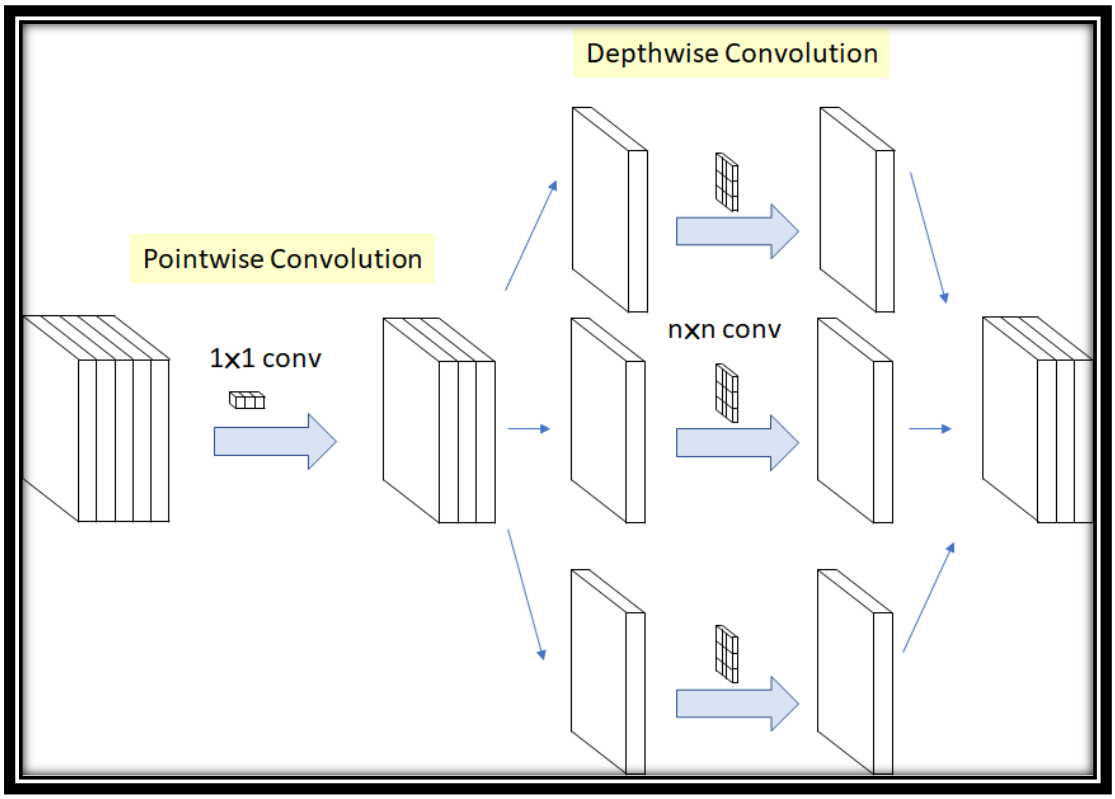

Figura 13: Convolución Separable en Profundidad Invertida.
Fuente: [59]

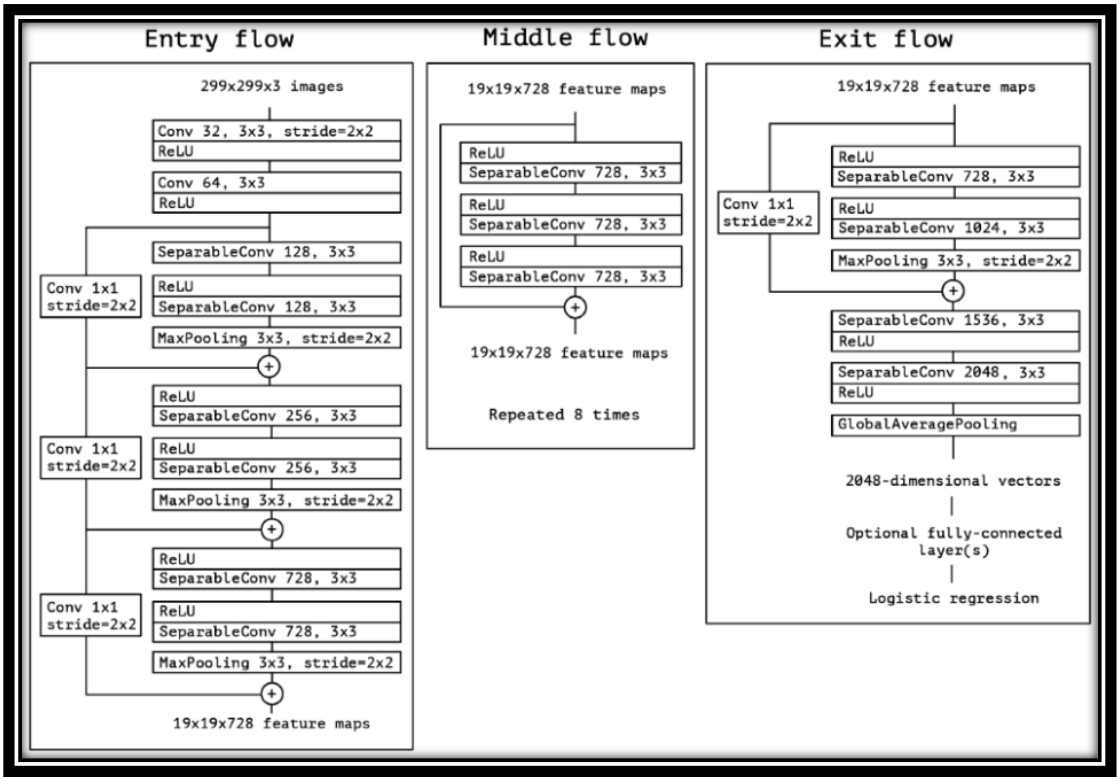

Figura 14: Arquitectura interna de la red Xception.
Fuente: [28]

3.4.5 MobileNetV2

La arquitectura MobileNetV2 consta de 53 capas, utiliza DSC, utiliza BN después de cada convolución, y sus convoluciones son solamente 1x1 y 3x3 (véase Figura 17). Utiliza una función de activación intermedia relu6 entre los dos pasos de una DSC y hace uso de bloques residuales invertidos (véase Figura 15). Consta de dos módulos principales, el módulo con stride=1 utiliza conexiones residuales mientras que el módulo con stride=2 no y este último se usa como reductor de dimensionalidad [31] (véase Figura 16).

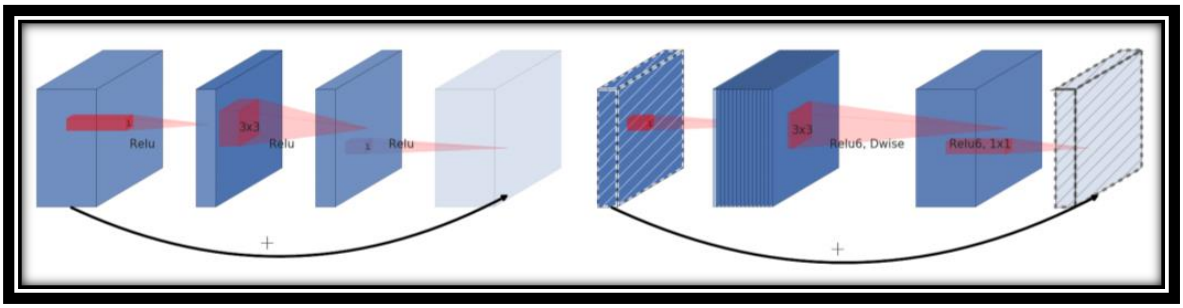

Figura 15: Bloque residual a la izquierda, bloque residual invertido a la derecha.
Fuente: [31]

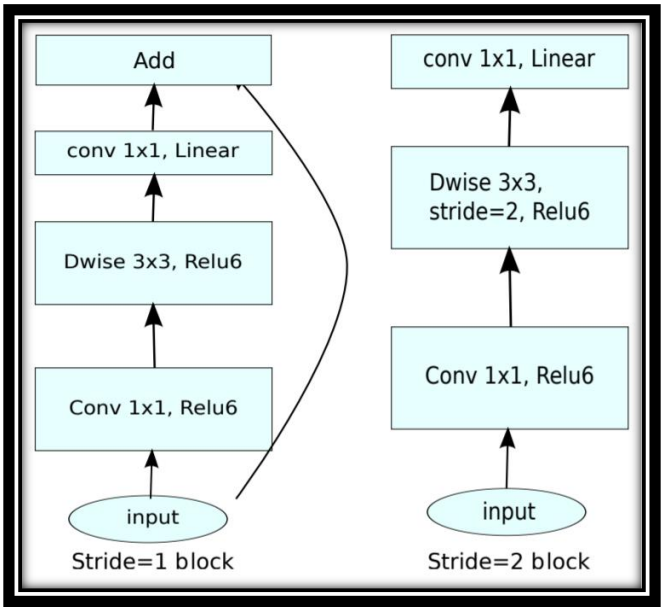

Figura 16: Bloques principales de la red MobileNetV2.
Fuente: [31]

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figura 17: Arquitectura interna de la red MobileNetV2.
Fuente: [31]

3.4.6 NASNetMobile

Las arquitecturas derivadas de la familia NASNet están construidas a partir de dos tipos de celdas, celdas normales y de reducción. Las primeros se refieren a celdas de convoluciones que no reducen la dimensionalidad de entrada, mientras que las últimos son aquellas celdas convolucionales que generan una imagen de salida 2 veces menor a la de entrada.

La arquitectura interna óptima para cada celda fue encontrada a partir de aprendizaje por refuerzo dentro la base de datos CIFAR10. Utilizaron una RNN que seleccionaba le mejor combinación de acciones-estados para formar bloques y B bloques formaban una celda (B=5 en el artículo).

Además, utilizaron una técnica nueva llamada “Scheduled DropPath” la cual aumento significativamente el accuracy y se basaba en desconectar cada conexión de un bloque con una probabilidad que aumentaba conforme avanza el entrenamiento.

La red NASNetMobile en Keras (también llamada NASNet-A(4 @ 1056) dentro del artículo) trabaja con imágenes de 224x224 pixeles y tiene aproximadamente 5.3Mill. de parámetros a diferencia de su contraparte NASNetLarge (también llamada NASNet-A(6 @ 4032) dentro del artículo) que trabaja con imágenes de 331x331 pixeles y cuenta con alrededor de 88.9Mill. de parámetros [30].

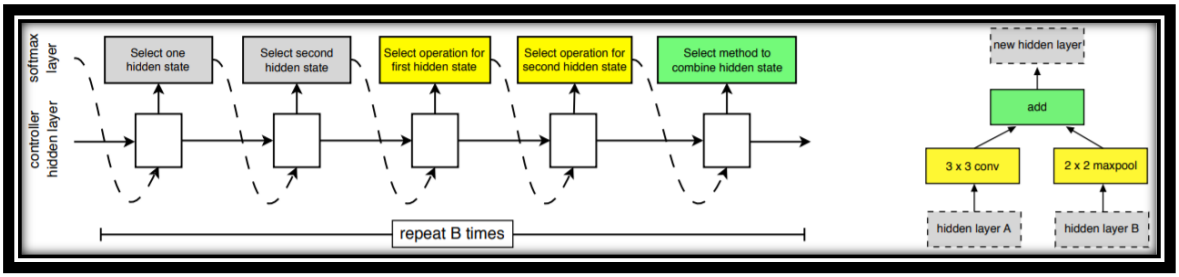

Figura 18: Arquitectura del controlador utilizado para la construcción de las celdas NASNet (izquierda) y ejemplo de un posible bloque de una celda (derecha).
Fuente: [30]

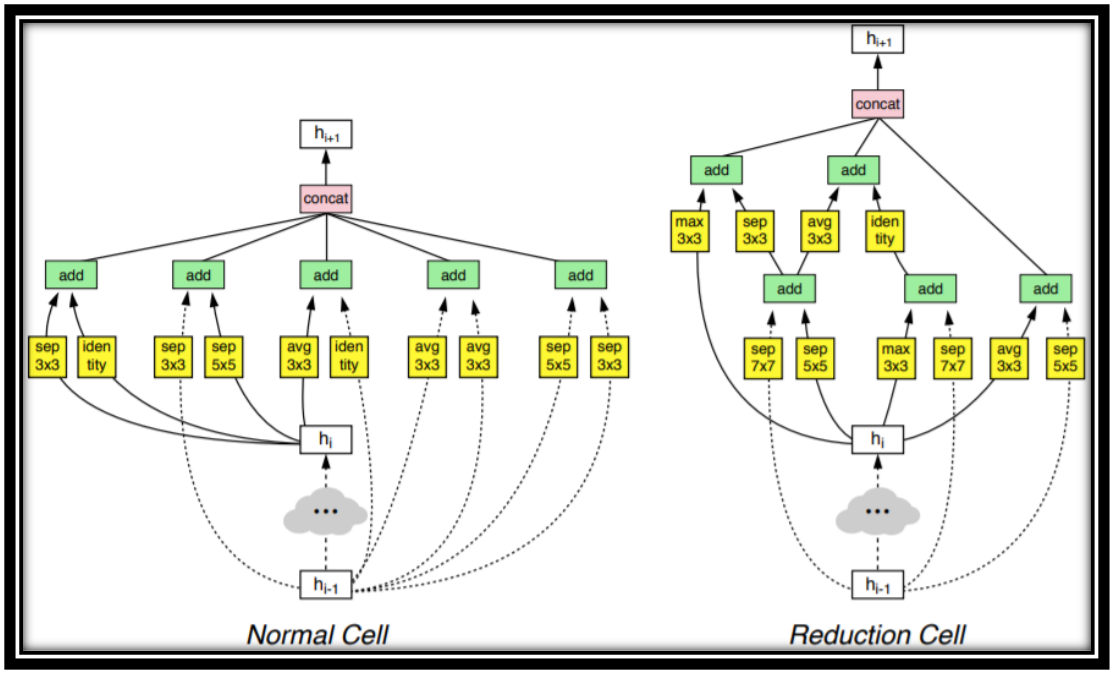

Figura 19: Celda Normal y de Reducción de la subfamilia NASNetA.
Fuente: [30]

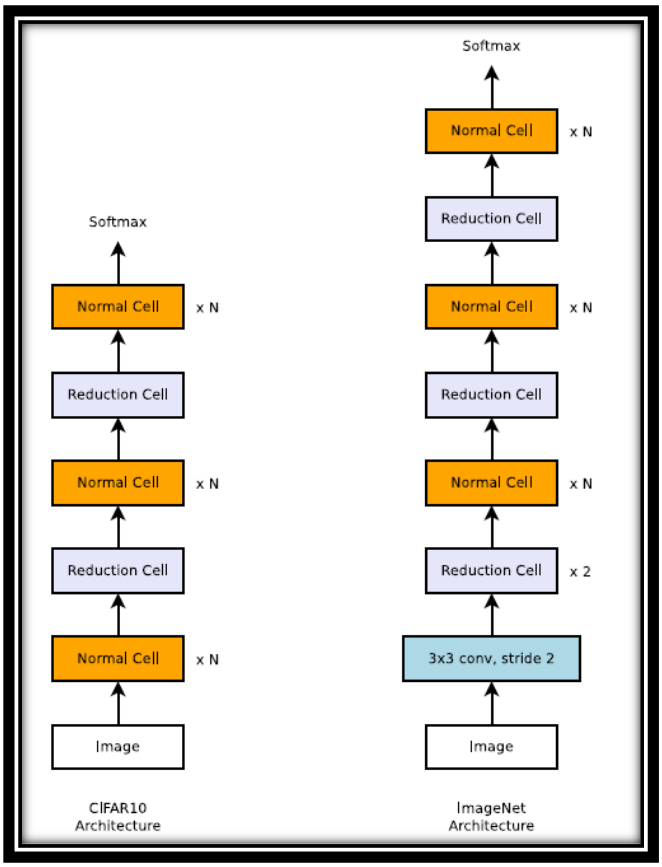

Figura 20: Arquitecturas de las redes NASNet para la base de datos CIFAR10 (Izquierda) e ImageNet (derecha).
Fuente: [30]

3.4.7 EfficientNetB0 y EfficientNetB3

Las arquitecturas derivadas de la familia EfficientNet cuentan con 237 capas como mínimo (EfficietNetB0) y 813 capas como máximo (EfficientNetB7). Utilizan dropout, avg pooling, BN después de cada convolución y sus convoluciones son de 1x1, 3x3 y 5x5. Al igual que la red MobileNetV2 utiliza DSC con una función de activación intermedia y hace uso de bloques de residuales invertidos. También hacen uso de una función de activación llamada “swish” después de cada operación de convolución la cual es una combinación de la función lineal y la función sigmoide. Y no menos importante utilizan bloques de compresión y excitación, lo cual hace posible que se les dé más importancia a ciertos canales de una capa convolucional [32]. La principal aportación de este tipo arquitecturas es el uso de un escalamiento compuesto, el cual, según los autores, es un elemento a tener en cuenta al

momento de querer expandir una arquitectura (véase Figura 26). De acuerdo con ellos, un escalamiento uniforme de la anchura de las capas, la profundidad del modelo y la resolución de entrada puede mejorar significativamente el accuracy de la arquitectura. Para aplicar este principio primero desarrollaron como base el modelo EfficientNetB0 (véanse Figuras 21-24) a través de una búsqueda multiobjetivo de una arquitectura neuronal que optimizara el número de operaciones flotantes y el accuracy. A partir de este modelo base, fueron escalando la red usando el principio expuesto anteriormente para generar los modelos que van desde la EfficientNetB1 hasta la EfficientNetB7 [32] (véase Figura 25).

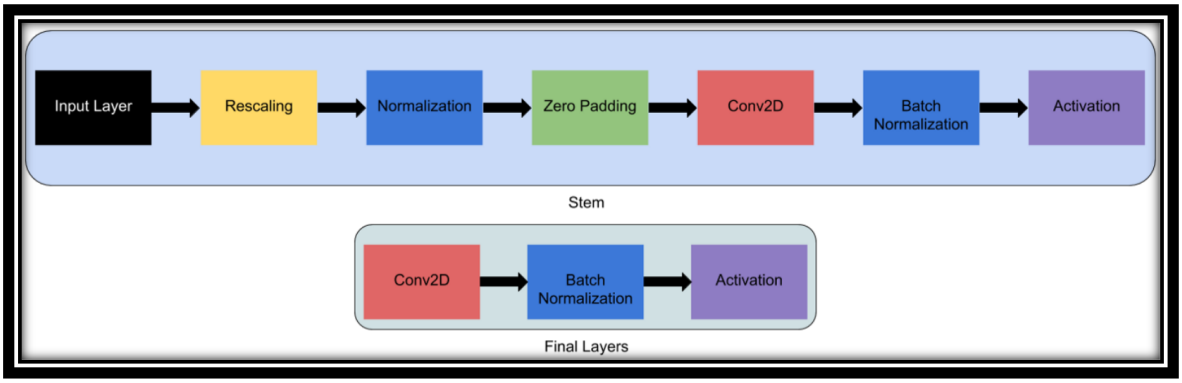

Figura 21: Diagrama de bloques de las capas iniciales (Stem) y finales que son comunes en todas las redes de la familia EfficientNet.
Fuente: [60]

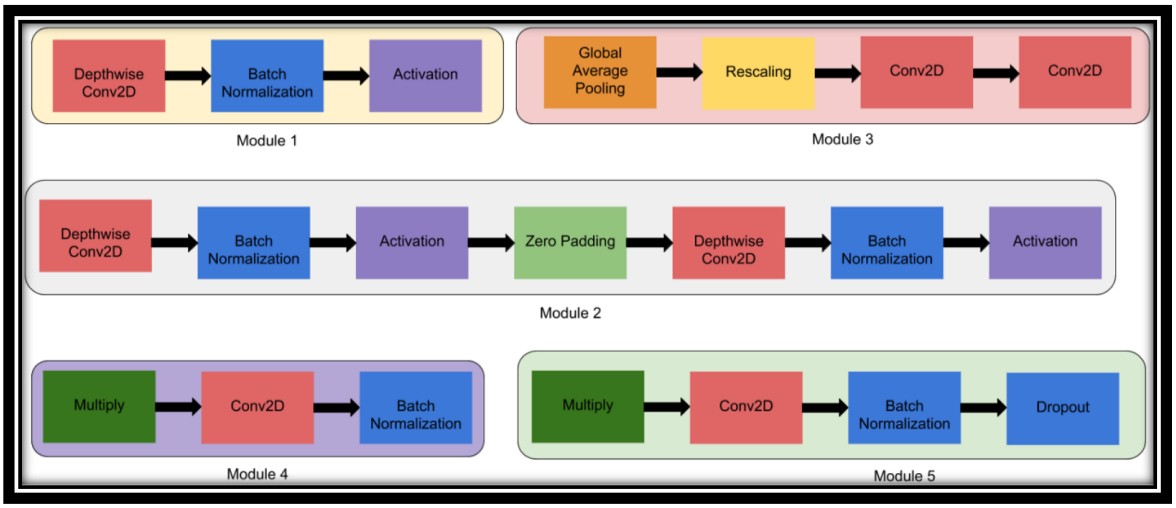

Figura 22: Diagrama de bloques de los módulos que conforman los sub-bloques de las redes de la familia EfficientNet.
Fuente: [61]

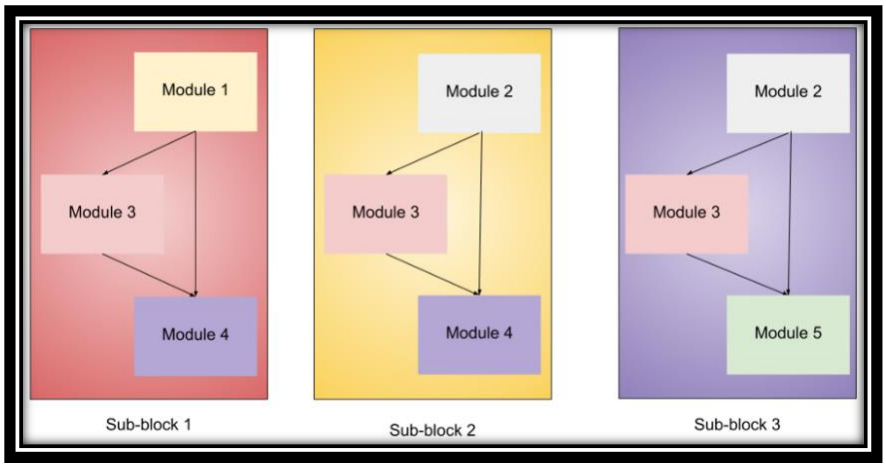

Figura 23: Sub-bloques que conforman a los bloques principales de las redes de la familia EfficientNet.
Fuente: [62]

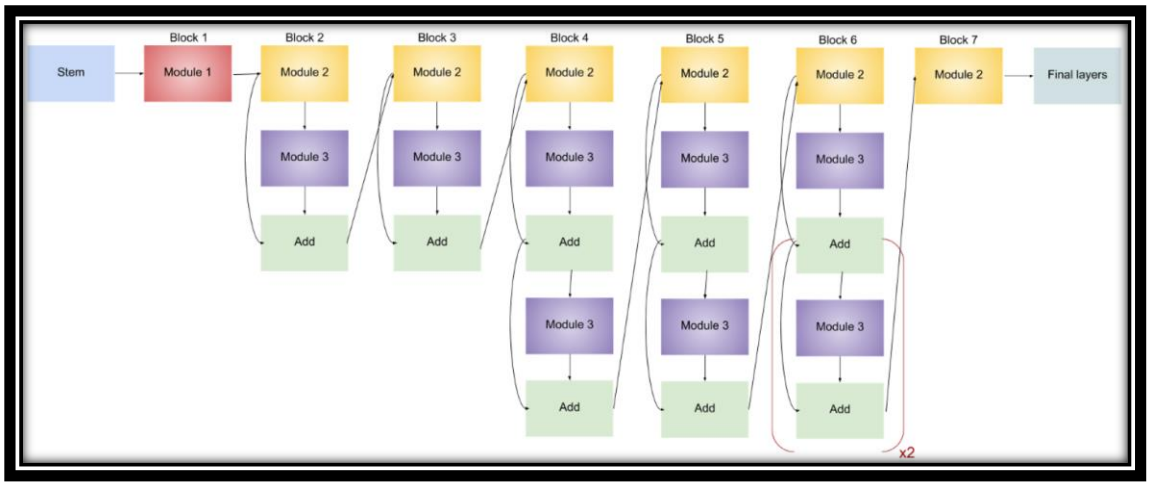

Figura 24: Arquitectura interna de la red EfficientNetB0. Cada módulo de color dentro de cada block se refiere a uno de los tres tipos de sub-bloques.
Fuente: [63]

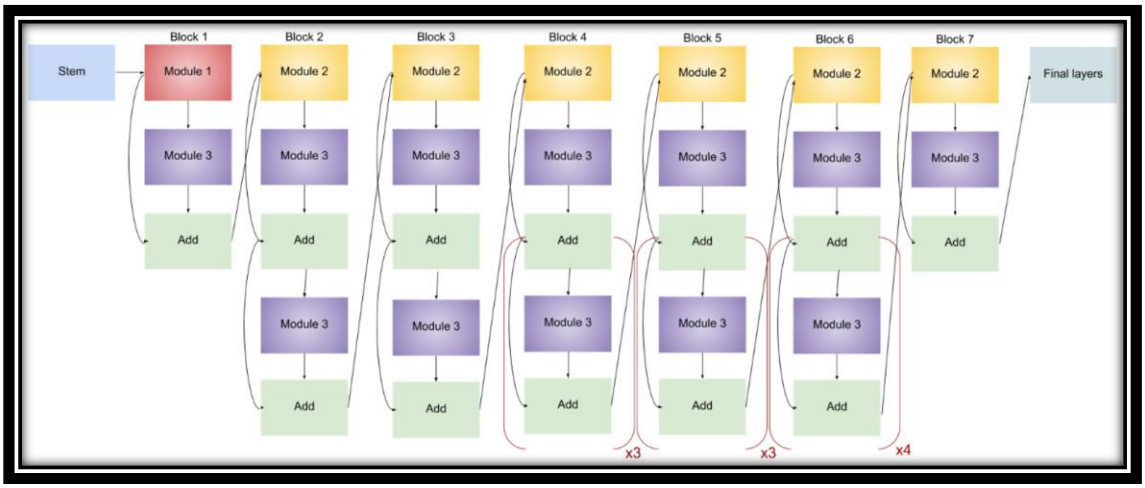

Figura 25: Arquitectura interna de la red EfficientNetB3. Cada módulo de color dentro de cada block se refiere a uno de los tres tipos de sub-bloques.
Fuente: [64]

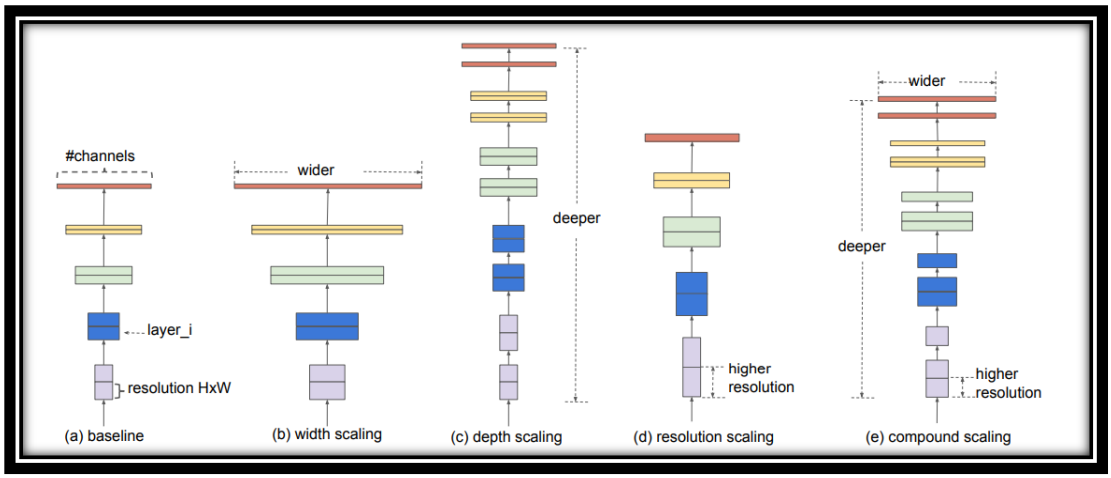

Figura 26: Tipos de escalamiento convencionales (a-d), escalamiento de compuesto de redes EfficientNet (e). Fuente: [32]

Capítulo 4: Metodología

Este capítulo se divide en cuatro secciones principales. La primera sección explica las características de la base de datos de videos HMDB51. La segunda sección aborda el procedimiento que se siguió para elaborar los distintos grupos de frames a partir de la base de datos HDMB51. La tercera sección expone la metodología empleada para la creación de las dos versiones filtradas de videos a partir de la base de datos HMDB51. La cuarta y última sección desglosa la descripción de cada tipo de ensamble.

4.1 Base de Datos HMDB51

La base de datos de videos HMDB51 consta de 6766 videoclips con 51 clases de acciones humanas y cada clase consta de al menos 100 videos (véase Ilustración 2). Contiene 3 sets de entrenamiento-prueba de distintos videos aleatorios para hacer la evaluación de los modelos. La resolución espacial de los videos es de 320x240 pixeles y todos los videos se obtuvieron de YouTube y películas digitales. Una gran parte de los videos contiene más de una escena diferente y movimiento de cámara lo cual lo hace una base datos muy desafiante. La base de datos puede ser descargada desde el siguiente enlace https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/#Evaluation.

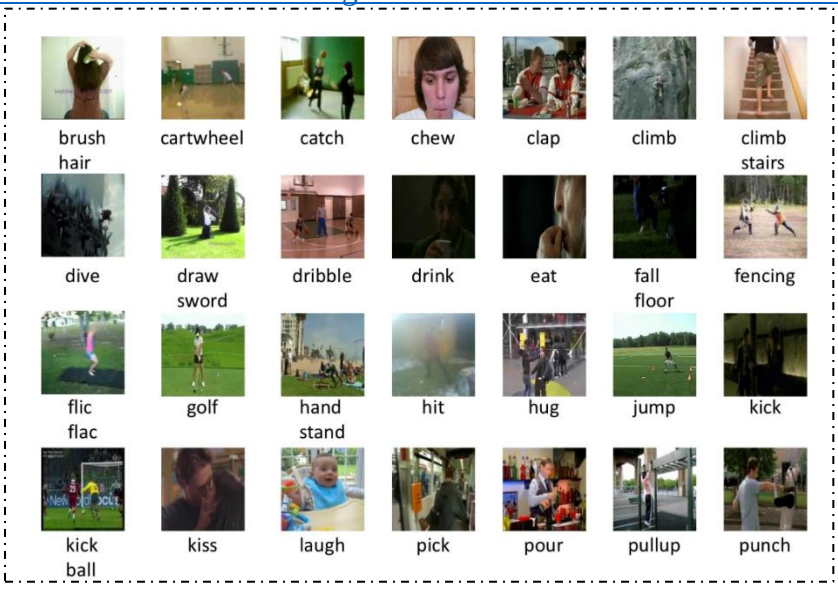

Ilustración 2: Primeras 28 clases de la base de datos HMDB51.

4.2 Grupos de Frames

Esta subsección explicara los 4 diversos grupos de frames que se crearon a partir de la base de datos HMDB51.

4.2.1 Grupo 1

El primer grupo de frames (Grupo 1) se construyó tomando 10 frames espaciados uniformemente de cada video de entrenamiento y prueba del *set1* de la base de datos HMDB51. Los frames extraídos de cada video de entrenamiento se almacenaron en una carpeta separada a los frames extraídos de cada video de prueba. Para extraer los frames, primero se recorrió el video entero para obtener el número total de frames *F*. El número de frames a tomar *N* se fijó en 10. El espacio entre frames *E* se calculó dividiendo *F/N*. Para cada video se extrajo el índice de cada frame *i*, los frames se extraían si cumplían la condición de que el residuo de *i/E=0*. La extracción inició con el índice 0, por lo que el primer frame de cada video siempre fue extraído (véase Ilustración 3).

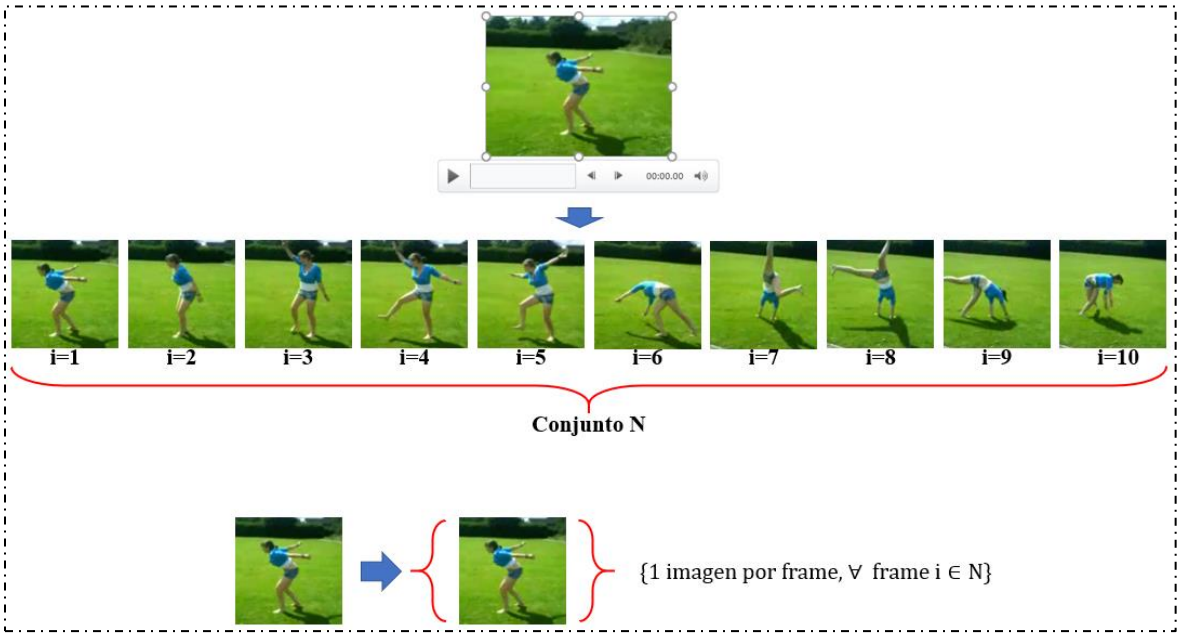

Ilustración 3: Proceso de Extracción de Frames para el Grupo 1 de Frames.

4.2.2 Grupo 2

Con el fin de evaluar el efecto de la aumentación de datos se decidió utilizar 3 técnicas diferentes. La primera de ellas consistió en utilizar el reflejo horizontal de cada frame. El segundo grupo de frames (Grupo 2) se construyó tomando 10 frames (y su reflejo horizontal) espaciados uniformemente cada video de entrenamiento y prueba del *set1* de la base de datos HMDB51 (véase Ilustración 4). Los frames extraídos de cada video de entrenamiento se almacenaron en una carpeta separada a los frames extraídos de cada video de prueba. La extracción de frames siguió el procedimiento descrito en el Grupo 1 de frames.

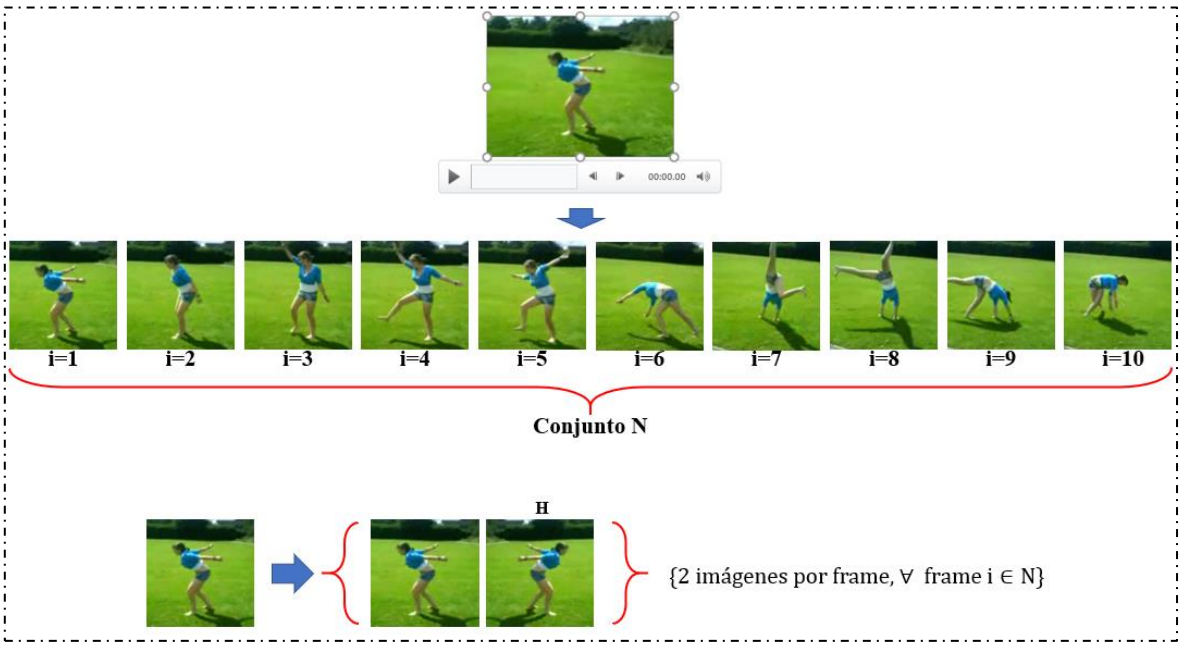

Ilustración 4: Proceso de Extracción de Frames para el Grupo 2 de Frames.

4.2.3 Grupo 3

La segunda técnica de aumentación utilizada consistió en redimensionar cada frame extraído de forma que su lado más corto midiera 256 pixeles y de dicho frame resultante se extrajo una región de 224x224 pixeles del centro y de cada esquina del frame. Esto proceso de aumentación generó 5 imágenes de cada frame original. El tercer grupo de frames (Grupo 3) se construyó tomando 10 frames (de los cuales se extraían 5 imágenes) espaciados

uniformemente cada video de entrenamiento y prueba del *set1* de la base de datos HMDB51 (véase Ilustración 5). Los frames extraídos de cada video de entrenamiento se almacenaron en una carpeta separada a los frames extraídos de cada video de prueba. La extracción de frames siguió el procedimiento descrito en el Grupo 1 de frames.

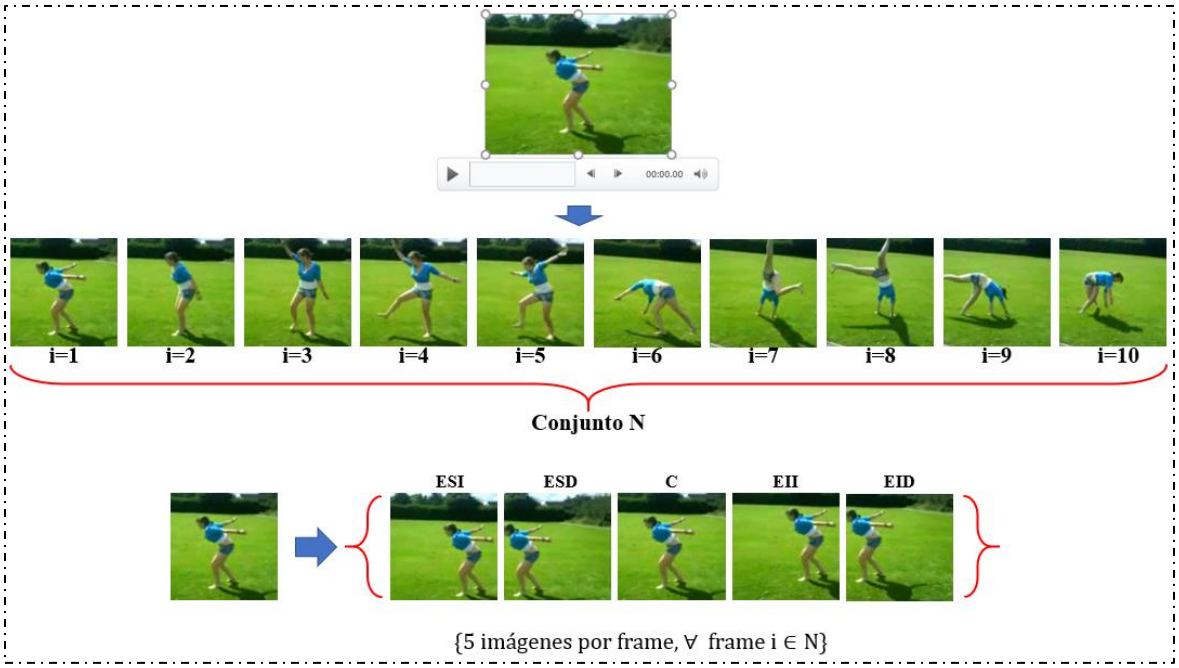

Ilustración 5: Proceso de Extracción de Frames para el Grupo 3 de Frames.

4.2.4 Grupo 4

La tercera y última técnica de aumentación utilizada consistió en redimensionar cada frame extraído de forma que su lado más corto midiera 256 pixeles y de dicho frame resultante se extrajo una región de 224x224 pixeles del centro y de cada esquina del frame. Además, se obtuvo el reflejo horizontal de cada una de las 5 imágenes obtenidas, lo cual generó 10 imágenes de cada frame original. El cuarto grupo de frames (Grupo 4) se construyó tomando 10 frames (de los cuales se extraían 10 imágenes) espaciados uniformemente cada video de entrenamiento y prueba del *set1* de la base de datos HMDB51 (véase Ilustración 6). Los frames extraídos de cada video de entrenamiento se almacenaron en una carpeta separada

a los frames extraídos de cada video de prueba. La extracción de frames siguió el procedimiento descrito en el Grupo 1 de frames.

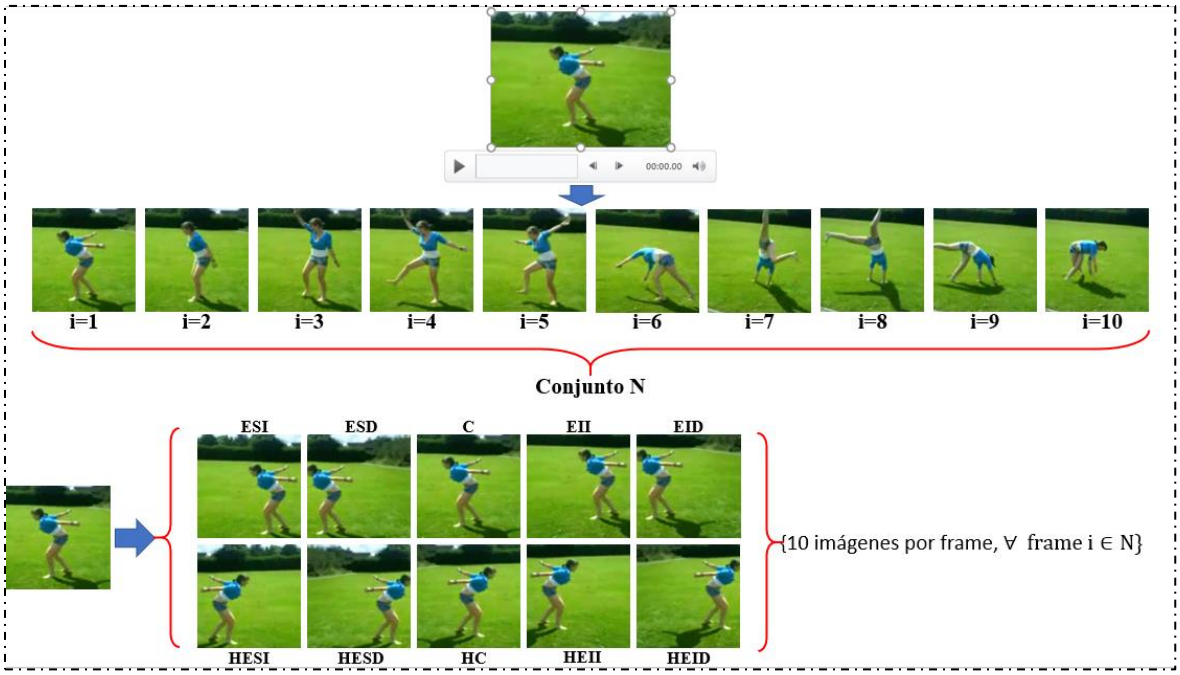

Ilustración 6: Proceso de Extracción de Frames para el Grupo 4 de Frames.

4.3 Bases de datos HMDB51 Filtradas

Se propuso elaborar dos nuevas series de videos filtrados a partir de todos los videos originales de la base de datos HMDB51 con el fin de analizar el impacto que tiene el utilizar videos con frames más propensos a pertenecer a la clase en cuestión en el desempeño de las arquitecturas. Primero se eliminaron todos los frames de pantalla negra o blanca de cada video de la base de datos original a través de un programa que evaluó todos los pixeles de cada frame de cada video. Para ello se elaboró una función que recibía como entrada un frame, y evaluaba si todos los pixeles del frame tenían un valor por arriba de 245 o por debajo de 10. En caso de que cualquiera de estas dos condiciones se cumpliera, el frame era descartado para usarse como parte de los frames del nuevo video. Este proceso se repetía

para todos los frames del video y se generaba un nuevo video a partir de todos los frames que no fueron descartados (véase Ilustración 7).

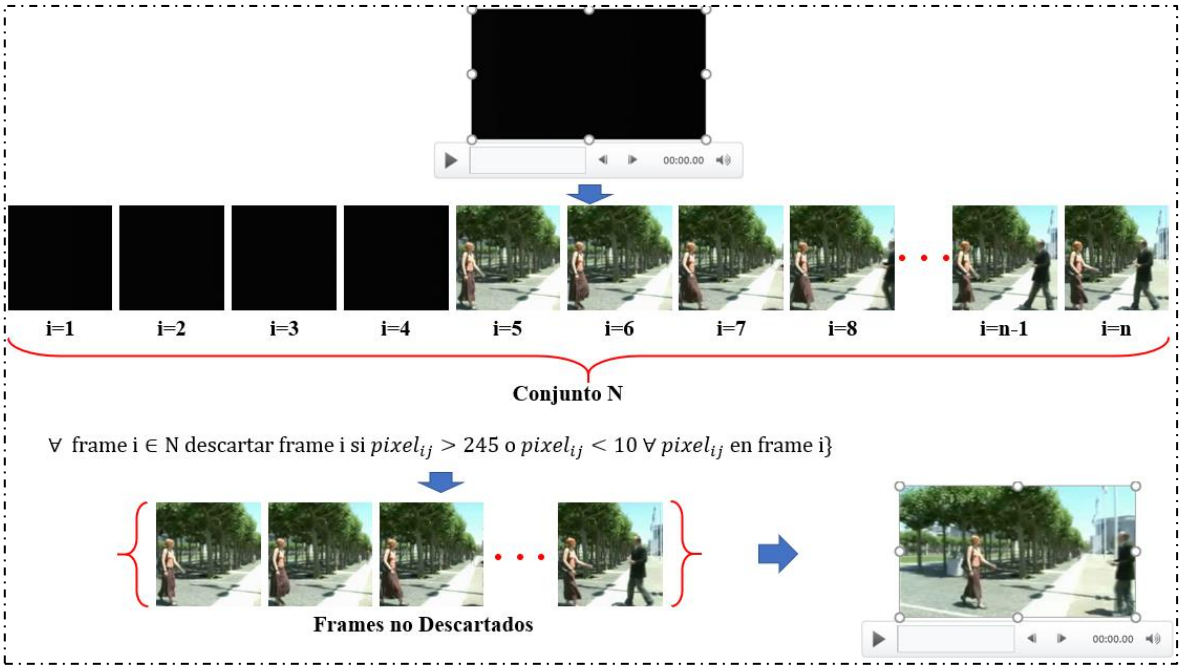

Ilustración 7: Proceso usado para descartar frames totalmente negros o blancos de los videos.

Después se usó la librería SceneDetect de Python para separar aquellos videos que tuvieran dos o más escenas de aquellos que solo tuvieran una sola escena. Para ello se utilizaron los videos que ya habían sido limpiados de frames totalmente negros o blancos. Este proceso consistía en analizar la similitud entre frames, y si excedía un cierto límite, el segundo frame se marcaba como el inicio de una nueva escena. Siempre y cuando ambas escenas tuvieran al menos 10 frames, el video se asignaba al grupo de videos de más de una escena, en caso contrario era asignado al grupo de videos con una escena (véase Ilustración 8).

Ilustración 8: Proceso usado para clasificar los videos con más de una escena.

Una vez hecho esto, se crearon 17 grupos de tres clases cada uno, tanto para los videos con múltiples escenas como para los videos con una sola escena. Para cada grupo de clases se entrenó un clasificador con los frames de los videos con una sola escena. Para evitar que un clasificador tendiera a favorecer a una clase dentro del grupo de clases con el cual se entrenó, se determinó con anterioridad cual era el número mínimo de videos M que compartían las clases de cada grupo. Dicho valor M se usó para muestrear aleatoriamente M videos de cada clase del grupo correspondiente.

Una vez hecho esto se procedió a generar un grupo de frames a partir de los videos con una sola escena pertenecientes a cada grupo utilizando el procedimiento descrito en el Grupo 2 de frames. Los frames de cada grupo de tres clases fueron utilizados para entrenar un clasificador, en total se entrenaron 17 clasificadores, uno por cada grupo de clases.

Se entrenó cada clasificador usando la arquitectura EfficientNetB3 debido a su balance entre accuracy y tiempo durante 100 épocas. Durante el entrenamiento se validó su accuracy con los frames de los videos de dos o más escenas, se guardó el modelo con mejor val_accuracy y se asignó una condición que interrumpía el entrenamiento si la métrica val_accuracy no mejoraba en al menos cinco épocas consecutivas.

Se evaluó dos veces el mismo clasificador y se eligió el clasificador con mejor accuracy dentro del grupo de validación. Se usó cada clasificador de cada grupo para filtrar los frames de los videos con más de dos escenas.

Para ello, a cada video se le extraían todos los frames y eran introducidos al clasificador correspondiente a la clase del video, y de acuerdo a la etiqueta predicha por el clasificador se formaron dos listas de frames. La primera lista contenía los frames que fueron clasificados correctamente, mientras que la segunda lista a los frames clasificados incorrectamente (véase Ilustración 9). Una vez creadas ambas listas de frames se siguieron dos grupos de condiciones para formar el nuevo video (véase Apéndice 1). Cada video generado con el grupo de condiciones 1 era anexado a la base de datos HMDB51 filtrada 1, en caso de que se usara el grupo de condiciones 2 el video resultante se anexaba a la base de datos HMDB51 filtrada 2. Cada video con más de dos escenas pasó por ambos grupos de condiciones, por lo tanto, cada versión filtrada contiene una versión del video original generado con las condiciones correspondientes. Para finalizar, se anexaron todos los videos con una sola escena a ambas bases de datos filtradas.

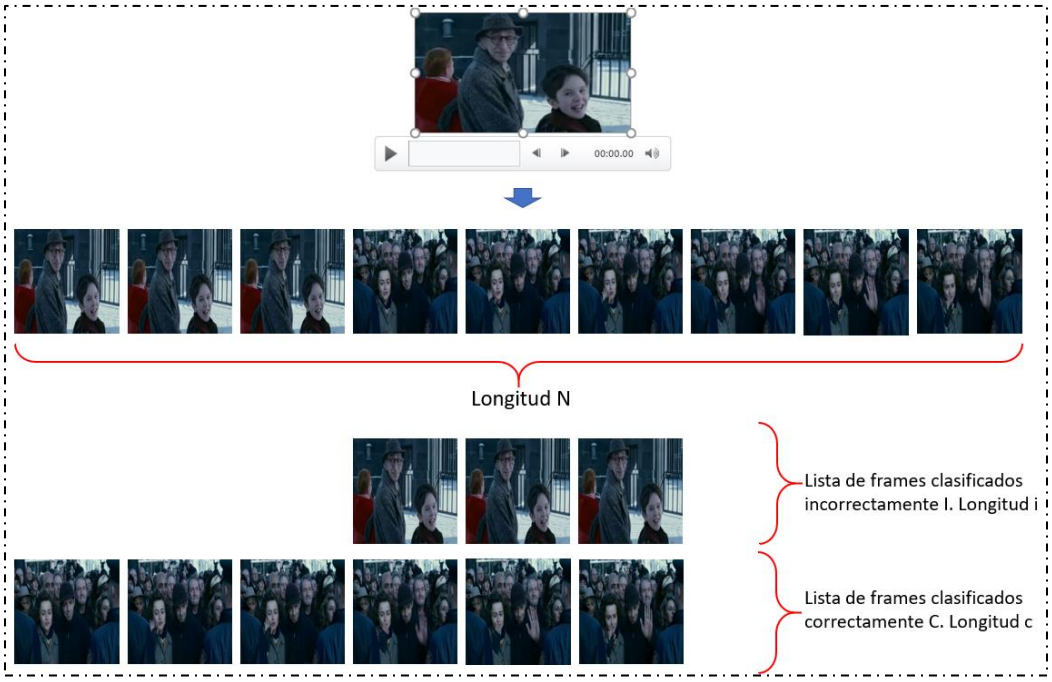

Ilustración 9: Listas de frames generadas a partir de las predicciones de los frames de un video.

4.4 Ensambles de Arquitecturas CNN

El modelo propuesto de ensamble se compone de *n* clasificadores de imágenes. Cada clasificador contiene una arquitectura de CNN distinta del estado del arte (ResNet, Inception, MobileNet, etc.). Todos los clasificadores se entrenaron con el mismo grupo de frames. Para generar la etiqueta final del ensamble en los videos de test se propusieron 6 métodos distintos.

4.4.1 Etiqueta final generada a partir de una votación simple usando n frames de cada video de test.

De cada video de test se extrajeron *n* frames (*n=10*) espaciados uniformemente y de acuerdo al grupo de frames de entrenamiento que se haya usado se aplicó la técnica de aumentación de datos correspondiente. Cada frame generado de cada video pasó por cada una de las CNNs del modelo de ensamble. Cada CNN generó dos valores, una etiqueta de salida y el valor 1, los cuales se agregaron en un diccionario de etiquetas (*A*). Cada clave del diccionario *A* era una etiqueta predicha por alguno de los clasificadores de imágenes en el ensamble, y el valor de dicha clave se refería al número de clasificadores que predijeron dicha etiqueta. Para obtener la etiqueta final se usó un sistema de votación en donde la etiqueta o clase que tuviera más votos por parte de las CNNs dentro del diccionario *A* se usaba como etiqueta final del video (véase Ilustración 10).

4.4.2 Etiqueta final generada a partir de una votación simple usando todos los frames de cada video de test.

De cada video de test se extrajeron todos los frames *N*. Cada frame generado de cada video pasó por cada una de las CNNs del modelo de ensamble. Cada CNN generó dos valores, una etiqueta de salida y el valor 1, los cuales se agregaron en un diccionario de etiquetas (*A*). Cada clave del diccionario *A* era una etiqueta predicha por alguno de los clasificadores de imágenes en el ensamble, y el valor de dicha clave se refería al número de clasificadores que

predijeron dicha etiqueta. Para obtener la etiqueta final se usó un sistema de votación en donde la etiqueta o clase que tuviera más votos por parte de las CNNs dentro del diccionario *A* se usaba como etiqueta final del video (véase Ilustración 11).

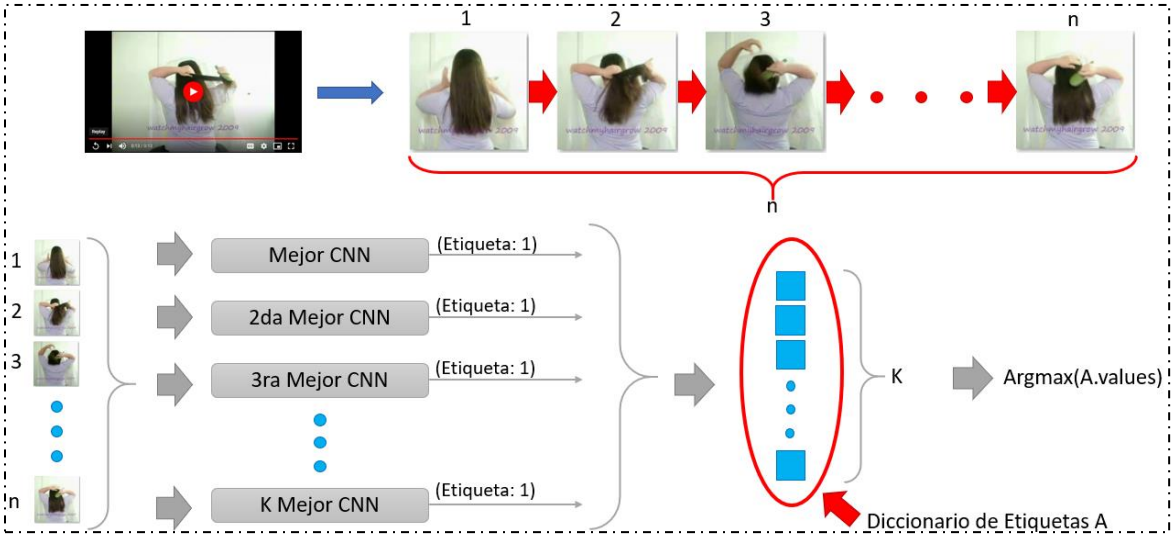

Ilustración 10: Ensamble de CNNs usando votación simple con n frames.

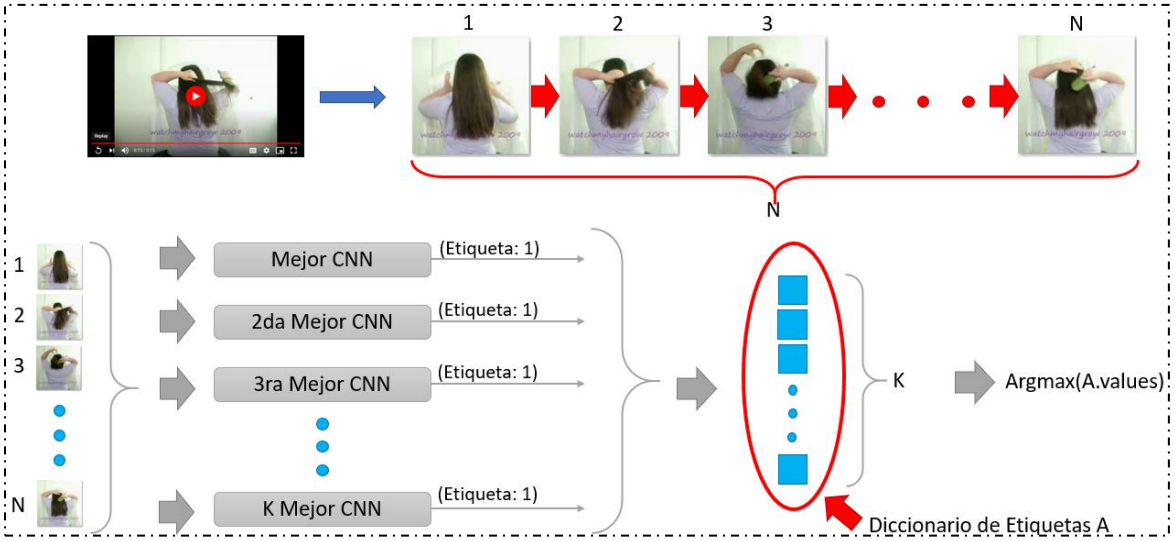

Ilustración 11: Ensamble de CNNs usando votación simple con todos los frames N.

4.4.3 Etiqueta final generada a partir de una votación con peso usando n frames de cada video de test.

De cada video de test se extrajeron n frames (*n=10*) espaciados uniformemente y de acuerdo al grupo de frames de entrenamiento que se haya usado se aplicó la técnica de aumentación de datos correspondiente. Cada frame generado de cada video pasó por cada una de las CNNs del modelo de ensamble. Cada CNN generó dos valores, una etiqueta de salida y un valor, los cuales se agregaron en un diccionario de etiquetas (*A*). El valor de cada clasificador dependía del rendimiento en accuracy de cada clasificador individual, por lo que el mejor clasificador entregaba un valor igual a *k*. El segundo mejor clasificador entregaba un valor igual a *k-1* y así sucesivamente. Esto se hizo para tomar en cuenta el rendimiento individual de cada clasificador y verificar si existía alguna diferencia significativa entre este método y el método de votación simple. Cada clave del diccionario *A* era una etiqueta predicha por alguno de los clasificadores de imágenes en el ensamble, y el valor de dicha clave se refería al puntaje obtenido de dicha etiqueta. Para obtener la etiqueta final se usó un sistema de votación en donde la etiqueta o clase que tuviera mayor puntaje por parte de las CNNs dentro del diccionario *A* se usaba como etiqueta final del video (véase Ilustración 12).

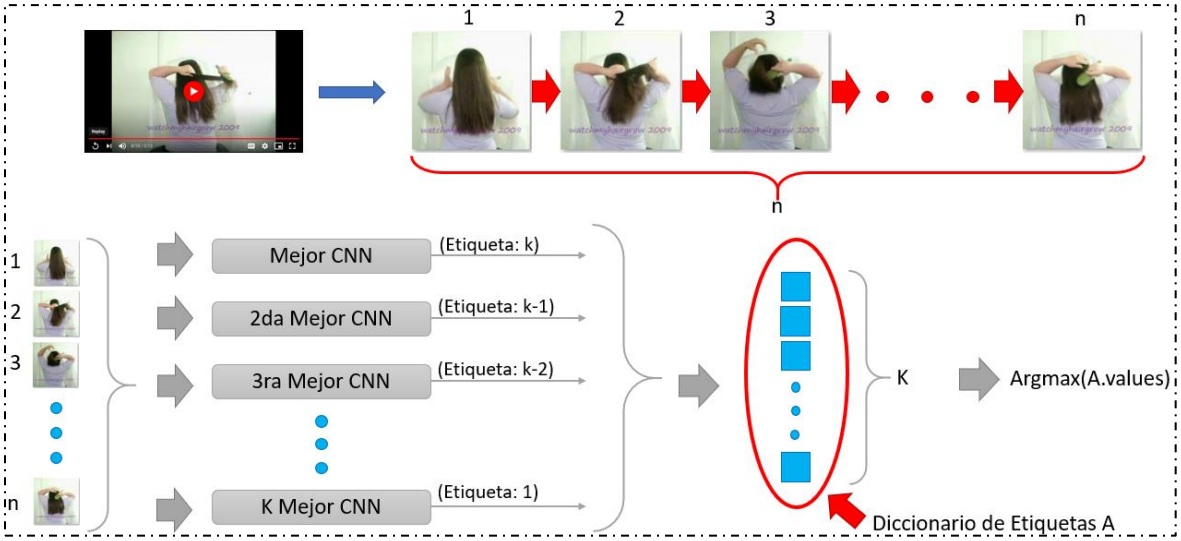

Ilustración 12: Ensamble de CNNs usando votación con peso con n frames.

4.4.4 Etiqueta final generada a partir del promediado de predicciones usando n frames de cada video de test.

De cada video de test se extrajeron *n* frames (*n=10*) espaciados uniformemente y de acuerdo al grupo de frames de entrenamiento que se haya usado se aplicó la técnica de aumentación de datos correspondiente. Cada frame generado de cada video pasó por cada una de las CNNs del modelo de ensamble. Cada CNN generó *N* predicciones, las cuales se concatenaron en una sola matriz de predicciones (*A*).

Para obtener la etiqueta final se promediaron todas las predicciones de *A* en el eje de los renglones obteniendo un vector de *C* clases. Finalmente se tomó el argumento máximo de dicho vector para generar el índice de la etiqueta de clase (véase Ilustración 13).

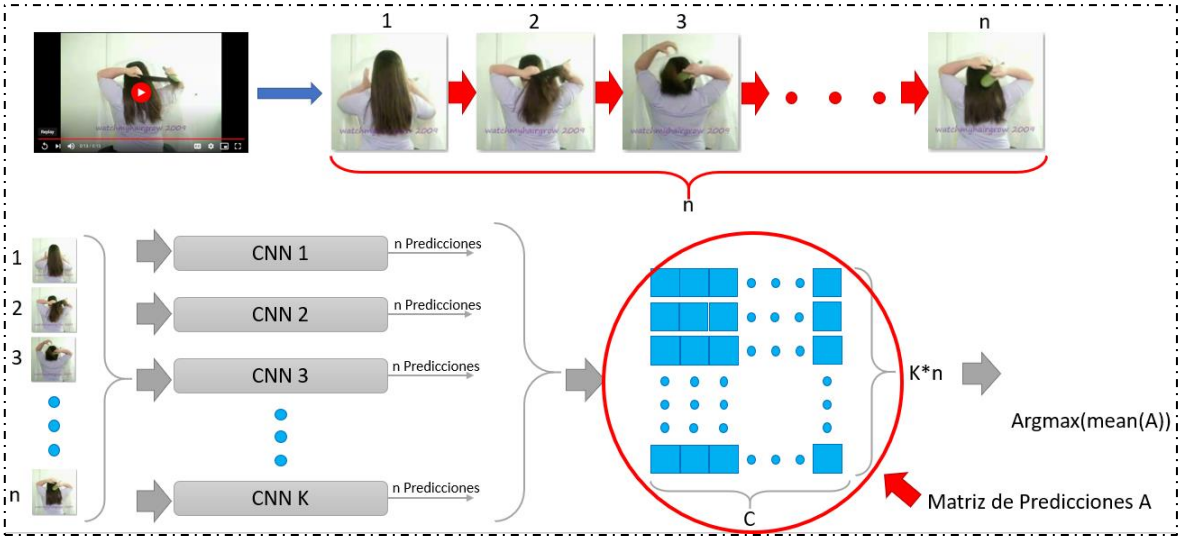

Ilustración 13: Ensamble de CNNs usando promediado de predicciones con n frames.

4.4.5 Etiqueta final generada a partir del promediado de predicciones usando todos los frames de cada video de test.

De cada video de test se extrajeron todos los frames *N*. Cada frame generado de cada video pasó por cada una de las CNNs del modelo de ensamble. Cada CNN generó *N* predicciones, las cuales se concatenaron en una sola matriz de predicciones (*A*).

Para obtener la etiqueta final se promediaron todas las predicciones de *A* en el eje de los renglones obteniendo un vector de *C* clases. Finalmente se tomó el argumento máximo de dicho vector para generar el índice de la etiqueta de clase (véase Ilustración 14).

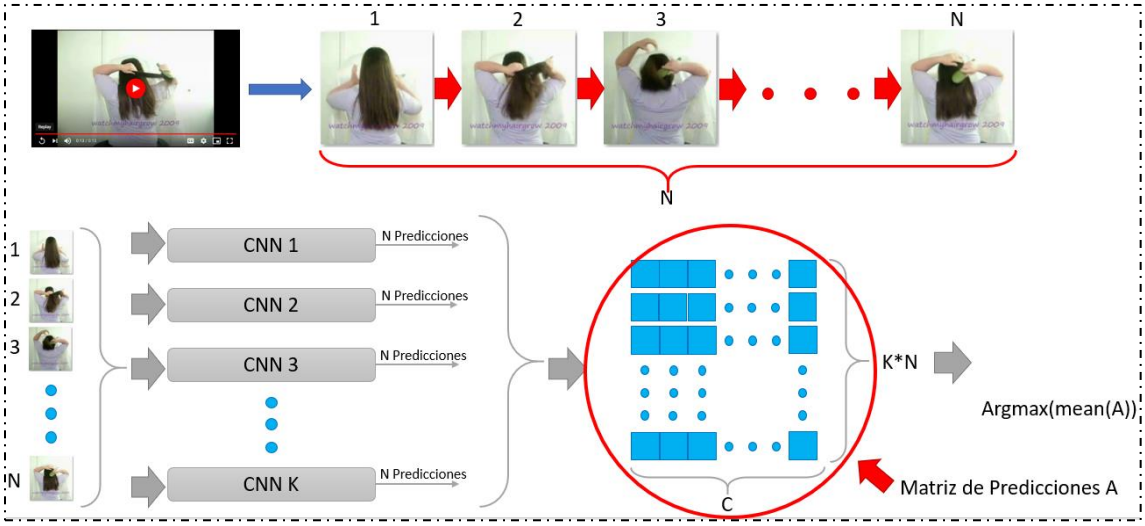

Ilustración 14: Ensamble de CNNs usando promediado de predicciones con todos los frames N.

Se utilizaron los valores por defecto de Keras para las dimensiones de los tensores de entrada para cada arquitectura. Las dimensiones del tensor de entrada para todas las arquitecturas con excepción de las arquitecturas Xception e InceptionV3 fueron de 224x224 pixeles, por lo que todas las imágenes de los distintos grupos de frames se redimensionaron a 224x224 pixeles. Para las redes Xception e InceptionV3 las dimensiones del tensor de entrada fueron de 299x299 pixeles, por lo que todas las imágenes de los distintos grupos de frames se redimensionaron a 299x299. Los experimentos se corrieron en 4 computadoras distintas, ya que no era posible armar un clúster de GPUs debido a la estructura de las mismas. Una computadora tenía una GPU 950M, otra una GPU 1060, otra una GPU 1080 y la última una GPU TITAN RTX, todas de la marca NVIDIA. Se usó el lenguaje de programación Python y el IDE Spyder. Las librerías mayormente utilizadas fueron la implementación de Keras en la librería de Tensorflow, Numpy, OS, OpenCV, Shutil y Pickle. Se utilizaron las arquitecturas predefinidas de las 8 arquitecturas anteriormente mencionadas que ofrecía la librería de Keras de Tensorflow para todos los experimentos.

La métrica utilizada para evaluar el desempeño de cada arquitectura fue la exactitud o “accuracy” el cuál es un valor entre 0 y 1 que expresa el porcentaje de etiquetas predichas por el modelo que concuerdan exactamente con las etiquetas objetivo. Para fines de compresión lectora, este valor se ajustó a un rango de 0 a 100%

Capítulo 5: Experimentación

Este capítulo se divide principalmente en dos secciones. La primera sección aborda la descripción y resultados de cada experimento realizado. La segunda sección realiza una discusión de los resultados obtenidos.

5.1 Experimentos y Resultados

El primer experimento consistió en decidir qué optimizador debe de usarse sobre los diferentes clasificadores de imágenes basados en arquitecturas del estado del arte. Para esto se probaron 5 diferentes optimizadores: “Adagrad”, “Adam”, “Nadam”, “RMSprop” y “SGD” con los valores que vienen por default en la librería de Keras.

Se usó un K Fold de 3 usando los videos de entrenamiento del set 1 de test de la base de datos original. Cada carpeta de las 51 carpetas del set 1 de test contiene 70 videos, de esos 30% se usó para validar y el restante 70% para entrenar el modelo. En total se usaron 2499 videos para entrenamiento y 1071 videos para validación en cada K Fold. Para extraer los frames de cada video de cada K Fold se usó el procedimiento descrito en el Grupo 1 de frames. Cada Fold se corrió 5 veces para evitar sesgos relativos a la inicialización de pesos de la red. El entrenamiento se hizo por 50 épocas y se usó una excepción para guardar el modelo con mejor val accuracy durante el entrenamiento.

Para la fase de validación, la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video de validación. Se reportó el promedio de las 3 corridas en los sets de validación de cada fold.

Cada optimizador se corrió un total de 15 veces y se evaluó tanto el modelo con mejor error de validación (val_loss) como el modelo con mejor accuracy de validación (val_acc).

Debido al tiempo excesivo que toma entrenar cada arquitectura 15 veces para cada optimizador, se optó por usar una CNN menos profunda exclusivamente para este experimento tal como es propuesto por [65]. La arquitectura de dicha CNN es la siguiente:

- Una capa de entrada que recibía imágenes de 224x224x3.
- Una capa convolucional con 32 filtros de 3x3 seguida de una función de activación “relu” y un max pooling de 2x2.
- Una segunda capa convolucional con 32 filtros de 3x3 seguida de una función de activación “relu” y un max pooling de 2x2.
- Una tercera capa convolucional con 64 filtros de 3x3 seguida de una función de activación “relu” y un max pooling de 2x2.
- Una capa flatten, seguida de una capa densa de 64 neuronas con función de activación “relu”, una capa de Dropout con valor de 0.5 y una capa final con 51 neuronas con función de activación “softmax”

Los resultados del experimento 1 se muestran en la Tabla 3 y la Tabla 4.

Tabla 3: Accuracy del experimento 1 en los sets de validación de los distintos folds para los distintos optimizadores usando el modelo con mejor val_loss.

Optimizador/Fold	Fold 1	Fold 2	Fold 3	Promedio
Adagrad	43.49%	39.12%	40.22%	40.95%
Adam	39.48%	37.42%	37.89%	38.26%
Nadam	42.58%	40.60%	36.00%	39.73%
RMSprop	43.12%	38.45%	40.95%	40.84%
SGD	43.90%	40.02%	39.74%	41.22%

Tabla 4: Accuracy del experimento 1 en los sets de validación de los distintos folds para los distintos optimizadores usando el modelo con mejor val_acc.

Optimizador/Fold	Fold 1	Fold 2	Fold 3	Promedio
Adagrad	44.03%	39.70%	40.69%	41.48%
Adam	44.67%	43.29%	42.80%	43.59%
Nadam	47.19%	45.90%	43.21%	45.43%
RMSprop	42.73%	40.54%	42.84%	42.04%
SGD	47.26%	45.23%	44.41%	45.63%

El segundo experimento se realizó de igual forma que el experimento uno, pero los frames extraídos de cada video de cada fold se extrajeron utilizando el procedimiento del Grupo 2 de frames. Este experimento se hizo con el fin de observar si los optimizadores se comportaban diferente con una diferente cantidad de datos de entrenamiento.

Los resultados del experimento 2 se muestran en la Tabla 5 y la Tabla 6.

Tabla 5: Accuracy del experimento 2 en los sets de validación de los distintos folds para los distintos optimizadores usando el modelo con mejor val_loss.

Optimizador/Fold	Fold 1	Fold 2	Fold 3	Promedio
Adagrad	45.83%	42.65%	40.97%	43.15%
Adam	44.76%	43.01%	43.59%	43.78%
Nadam	48.57%	43.53%	43.83%	45.31%
RMSprop	41.89%	39.25%	40.11%	40.42%
SGD	49.04%	45.62%	44.26%	46.31%

Tabla 6: Accuracy del experimento 2 en los sets de validación de los distintos folds para los distintos optimizadores usando el modelo con mejor val_acc.

Optimizador/Fold	Fold 1	Fold 2	Fold 3	Promedio
Adagrad	46.05%	42.75%	41.29%	43.36%
Adam	50.59%	49.67%	48.24%	49.50%
Nadam	53.80%	50.03%	49.41%	51.08%
RMSprop	43.87%	39.23%	42.04%	41.71%
SGD	53.71%	51.39%	49.71%	51.60%

Para comprobar que estos resultados no se debían por cuestiones de suerte se decidió utilizar el "paired t-test". El primer test comparó el vector que contenía el promedio por clase del optimizador Adagrad del modelo con mejor val_acc contra el vector que contenía el promedio por clase del optimizador SGD del modelo con mejor val_acc. Ambos modelos fueron tomados de la lista de modelos entrenados con el Grupo 1 de frames. Los resultados se ven en la Ilustración 15.

```
> adagrad<-c(17.8,6.6,14.8,17.4,14.0,7.8,8.8,7.8,10.4,14.6,5.4,5.8,2.6,9.4
> SGD<-c(17.6,10.0,12.8,17.8,14.4,7.2,10.0,7.6,12.0,13.6,3.8,5.8,4.6,10.2,
> t.test(SGD,adagrad,alternative="greater",paired=TRUE)

        Paired t-test

data:  SGD and adagrad
t = 6.3676, df = 152, p-value = 1.079e-09
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.6462556       Inf
sample estimates:
mean of the differences
              0.8732026
```

Ilustración 15: Paired t test para comparar la diferencia entre las medias entre el optimizador Adagrad y SGD.

De acuerdo el valor p obtenido en la Ilustración 15, se rechaza la hipótesis nula que declara que la verdadera diferencia entre las medias de los vectores es igual a 0. Esto indica

que si existe una diferencia significativa entre las medias de los modelos y dicha diferencia no se debe a la suerte.

El segundo test se usó para comprobar que existía una diferencia significativa entre las medias de los modelos con mejor val_acc y los modelos con mejor val_loss. Para esto se comparó el vector que contenía el promedio por clase del optimizador SGD del modelo con mejor val_loss contra el vector que contenía el promedio por clase del optimizador SGD del modelo con mejor val_acc. Ambos modelos fueron tomados de la lista de modelos entrenados con el Grupo 1 de frames. Los resultados se ven en la Ilustración 16.

```
> SGD_VL<-c(17.2,8.8,12.6,16.2,14.0,5.4,9.8,6.0,11.0,14.2,4.4,5.6,5.6,9.4,
> SGD_VACC<-c(17.6,10.0,12.8,17.8,14.4,7.2,10.0,7.6,12.0,13.6,3.8,5.8,4.6,
> t.test(SGD_VACC,SGD_VL,alternative="greater",paired=TRUE)

        Paired t-test

data:  SGD_VACC and SGD_VL
t = 6.0461, df = 152, p-value = 5.512e-09
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 0.6731128       Inf
sample estimates:
mean of the differences
              0.9267974
```

Ilustración 16: Paired t test para comparar la diferencia entre las medias entre los modelos con mejor val_loss y mejor val_acc del optimizador SGD.

De acuerdo el valor p obtenido en la Ilustración 16 se rechaza la hipótesis nula, lo que indica que si existe una diferencia significativa entre las medias de los modelos y dicha diferencia no se debe a la suerte.

La última prueba usando el "paired t-test" fue para comprobar si existía una diferencia significativa entre las medias de los modelos entrenados a partir del Grupo 1 de frames y los modelos entrenados a partir del Grupo 2 de frames. Para esto se comparó el vector que contenía el promedio por clase del optimizador SGD del modelo con mejor val_acc del Grupo 1 de frames contra el vector que contenía el promedio por clase del optimizador SGD del modelo con mejor val_acc del Grupo 2 de frames. Los resultados se ven en la Ilustración 17.

```
> SGD_Grupo1<-c(17.6,10.0,12.8,17.8,14.4,7.2,10.0,7.6,12.0,13.6,3.8,5.8,
> SGD_Grupo2<-c(17.8,12.0,14.8,18.8,15.0,9.6,11.4,12.8,13.0,15.2,3.6,5.4
> t.test(SGD_Grupo2,SGD_Grupo1,alternative="greater",paired=TRUE)

        Paired t-test

data:  SGD_Grupo2 and SGD_Grupo1
t = 12.446, df = 152, p-value < 2.2e-16
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 1.086909      Inf
sample estimates:
mean of the differences
               1.253595
```

Ilustración 17: Paired t test para comparar la diferencia entre las medias entre los modelos con mejor val_acc de los Grupos 1 y 2 de frames para el optimizador SGD.

De acuerdo el valor p obtenido en la Ilustración 17 se rechaza la hipótesis nula, lo que indica que si existe una diferencia significativa entre las medias de los modelos y dicha diferencia no se debe a la suerte.

Habiendo comprobado que dichos valores son significativos, se procedió a entrenar el modelo CNN anterior con los sets de entrenamiento completos y probarlo con los sets de prueba de los Grupos 1 y 2 de frames. Se escogieron los 2 mejores optimizadores, en este caso “Nadam” y “SGD”, se entrenó el clasificador 3 veces por 50 épocas usando la misma excepción que el experimento 1. Para la fase de prueba, la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio de las 3 ejecuciones en el set de prueba para cada grupo de frames y para cada optimizador. Los resultados de este experimento se muestran en la Tabla 7.

Tabla 7: Accuracy de los dos optimizadores Nadam y SGD para el modelo CNN sencillos en los sets de prueba de los Grupos de Frames 1 y 2.

Optimizador/Grupo de Frames	1	2
Nadam	10.63%	13.14%
SGD	11.26%	14.31%

Como cuarto experimento se procedió a entrenar las arquitecturas del estado del arte con el optimizador SGD, pero sin entrenarlas previamente con la base de datos ImageNet

solo para ver que tanto influye la falta de entrenamiento previo en el aprendizaje de un clasificador de imágenes. Las arquitecturas se entrenaron 3 veces usando el set de entrenamiento del Grupo 1 de frames por un total de 50 épocas, se usó la misma excepción que se explicó en el experimento 1. Para la fase de prueba, se usó el set de prueba del Grupo 1 de frames y la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio en el set de prueba de las 3 ejecuciones para cada arquitectura. Además, se incluyeron los tiempos en segundos de entrenamiento de cada arquitectura usando tanto una GPU 1080 como una GPU TITAN RTX. Los resultados de este experimento se muestran en la Tabla 8.

Tabla 8: Accuracy y tiempos de ejecución para las distintas arquitecturas sin entrenamiento previo en la base de datos ImageNet en el set de prueba del Grupo 1 de frames.

Arquitectura	Accuracy	GPU TITAN RTX	GPU 1080
EfficientNetB0	15.45%	9280 s	18876 s
EfficientNetB3	15.60%	17407 s	35339 s
Xception	17.60%	29014 s	62523 s
InceptionV3	15.47%	12182 s	26812 s
ResNet152	14.97%	20969 s	49098 s
DenseNet201	16.17%	15267 s	33650 s
MobileNetV2	16.23%	6822 s	12934 s
NASNetMobile	14.44%	13667 s	24967 s

En el quinto experimento se procedió a entrenar las arquitecturas del estado del arte con el optimizador SGD con entrenamiento previo con la base de datos ImageNet con el fin de contrastar los resultados del experimento anterior con los que se obtienen usando entrenamiento previo con otra base de datos.

Las arquitecturas se entrenaron 3 veces usando el set de entrenamiento del Grupo 1 de frames por un total de 50 épocas y se usaron las mismas excepciones que en el experimento 1. Para la fase de prueba, se usó el set de prueba del Grupo 1 de frames y la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio en el set de prueba de las 3 ejecuciones para cada arquitectura. Además, se incluyeron los tiempos en segundos de entrenamiento de cada arquitectura usando tanto una GPU 1080 como una GPU TITAN RTX. Los resultados de este experimento se muestran en la Tabla 9.

Tabla 9: Accuracy y tiempos de ejecución para las distintas arquitecturas con entrenamiento previo en la base de datos ImageNet en el set de prueba del Grupo 1 de frames.

Arquitectura	Accuracy	GPU TITAN RTX	GPU 1080
EfficientNetB0	47.39%	9133 s	18717 s
EfficientNetB3	47.84%	17464 s	35321 s
Xception	51.33%	28549 s	61730 s
InceptionV3	48.21%	12033 s	26869 s
ResNet152	44.81%	20878 s	49355 s
DenseNet201	45.95%	15239 s	33675 s
MobileNetV2	42.53%	7034 s	12913 s
NASNetMobile	43.86%	14025 s	25519 s

El 6° experimento procedió de igual forma que el quinto, pero con la diferencia que usó el Grupo 2 de frames tanto para entrenamiento como para prueba. Se reportó el promedio en el set de prueba de las 3 ejecuciones para cada arquitectura. De este experimento en delante ya no se reportan los tiempos de ejecución de las diversas arquitecturas. Los resultados de este experimento se muestran en la Tabla 10.

Tabla 10: Accuracy para las distintas arquitecturas con entrenamiento previo en la base de datos ImageNet en el set de prueba del Grupo 2 de frames.

Arquitectura	Accuracy
EfficientNetB0	49.67%
EfficientNetB3	50.70%
Xception	53.99%
InceptionV3	48.56%
ResNet152	45.80%
DenseNet201	45.99%
MobileNetV2	43.75%
NASNetMobile	44.47%

Para el 7° experimento se procedió a entrenar las arquitecturas Xception, InceptionV3, MobileNetV2 y EfficientNetB0 3 veces usando el set de entrenamiento del Grupo 3 de frames por un total de 50 épocas y se usó la misma excepción que en el experimento 1. Se limitó el entrenamiento de solo estas 4 arquitecturas debido a cuestiones de tiempo.

Se escogieron estas 4 redes debido a que las últimas 3 son las más rápidas de entrenar, mientras que la red Xception es la que ha reportado mejor accuracy en los experimentos anteriores. Para la fase de prueba, se usó el set de prueba del Grupo 3 de frames y la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio en el set de prueba de las 3 ejecuciones para cada arquitectura. Los resultados de este experimento se muestran en la Tabla 11.

Tabla 11: Accuracy para las distintas arquitecturas con entrenamiento previo en la base de datos ImageNet en el set de prueba del Grupo 3 de frames.

Arquitectura	Accuracy
EfficientNetB0	51.35%
Xception	52.68%
InceptionV3	48.39%
MobileNetV2	45.40%

Para el 8° experimento se procedió a entrenar 3 veces las arquitecturas Xception, InceptionV3, MobileNetV2 y EfficientNetB0 usando el set de entrenamiento del Grupo 4 de frames por un total de 50 épocas y se usó la misma excepción que en el experimento 1. Al igual que el 7° experimento, por cuestiones de tiempo solo se entrenaron estas cuatro arquitecturas. Para la fase de prueba, se usó el set de prueba del Grupo 4 de frames y la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio en el set de prueba de las 3 ejecuciones para cada arquitectura. Los resultados de este experimento se muestran en la Tabla 12.

Tabla 12: Accuracy para las distintas arquitecturas con entrenamiento previo en la base de datos ImageNet en el set de prueba del Grupo 4 de frames.

Arquitectura	Accuracy
EfficientNetB0	48.17%
Xception	50.26%
InceptionV3	46.95%
MobileNetV2	44.18%

Analizando los resultados anteriores, se puede ver que usando el Grupo 2 de frames se obtuvieron mejores resultados para todas las arquitecturas en comparación a los resultados obtenidos del Grupo 1 de frames. Por otro lado, al usar el Grupo 3 de frames, hubo un incremento en el accuracy en 2 de las 4 arquitecturas a comparación del Grupo 2 de frames. Con el Grupo 4 de frames no hubo ningún cambio positivo en el accuracy a comparación del Grupo 2 de frames. Por este motivo, se decidió utilizar el procedimiento descrito en el grupo de 2 de frames para generar el grupo de frames de los sets 2 y 3 de la base de datos HMDB51.

Para el 9° experimento, se entrenaron 3 veces las 8 arquitecturas con cada set de entrenamiento de los grupos de frames creados a partir del proceso descrito en el Grupo 2 de frames sobre los videos de entrenamiento de los sets 2 y 3 de la base de datos HMDB51 por un total de 50 épocas y se usó la misma excepción que en el experimento 1. Para la fase de prueba, se usó el set de prueba correspondiente a cada set y la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio de las 3 ejecuciones para cada arquitectura en los sets de prueba 2 y 3. Los resultados de este experimento se muestran en la Tabla 13.

Tabla 13:Accuracy para las distintas arquitecturas con entrenamiento previo en la base de datos ImageNet en los sets de prueba de los sets 2 y 3.

Arquitectura	Accuracy Set 2	Accuracy Set 3
EfficientNetB0	45.62%	45.66%
EfficientNetB3	46.97%	45.88%
Xception	50.00%	51.76%
InceptionV3	46.25%	47.25%
ResNet152	39.96%	40.46%
DenseNet201	42.42%	43.75%
MobileNetV2	41.33%	41.22%
NASNetMobile	40.04%	40.76%

Una vez obtenidos estos resultados, se propuso comparar el desempeño de las arquitecturas que fueron entrenadas con los frames extraídos de los videos originales de la base de datos HMDB51 contra los nuevos sets de videos generados a partir de los videos originales. Para hacer esto primero se trabajó con el set de videos de la versión filtrada 2 de la base de datos HMDB51. Los sets de entrenamiento-prueba de la versión filtrada 2 se generaron utilizando los archivos .txt que la base de datos original HMDB51 provee ya que eran los mismos videos, solo con distintos frames. Para cada set de entrenamiento-prueba se extrajo un grupo de frames con sus respectivas carpetas de entrenamiento y prueba usando el procedimiento descrito en el Grupo 2 de frames.

Para el 10° experimento, se entrenaron 3 veces las 8 arquitecturas con cada set de entrenamiento de los grupos de frames creados a partir del proceso descrito en el Grupo 2 de frames sobre los videos de entrenamiento de los tres sets de la Versión Filtrada 2 de la base de datos HMDB51 por un total de 50 épocas y se usó la misma excepción que en el experimento 1. Para la fase de prueba, se usó el set de prueba correspondiente a cada grupo de frames y la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio de las 3 ejecuciones para cada arquitectura en cada set de prueba. Los resultados de este experimento se muestran en la Tabla 14.

Después de trabajar con el set de videos de la versión filtrada 2, se trabajó con el set de videos de la versión filtrada 1 de la base de datos HMDB51. Los sets de entrenamiento-prueba de la versión filtrada 1 se generaron utilizando los archivos .txt que la base de datos original HMDB51 provee ya que eran los mismos videos, solo con distintos frames. Para cada set de entrenamiento-prueba se extrajo un grupo de frames con sus respectivas carpetas de entrenamiento y prueba usando el procedimiento descrito en el Grupo 2 de frames.

Tabla 14: Accuracy para las distintas arquitecturas con entrenamiento previo en la base de datos ImageNet en los distintos sets de prueba de la versión filtrada 2 de la base de datos HMDB51.

Arquitectura	Accuracy Set 1	Accuracy Set 2	Accuracy Set 3
EfficientNetB0	46.95%	46.54%	44.73%
EfficientNetB3	47.23%	46.99%	44.92%
Xception	51.96%	49.52%	50.00%
InceptionV3	47.32%	45.38%	45.86%
ResNet152	44.40%	41.09%	41.07%
DenseNet201	45.53%	41.87%	43.16%
MobileNetV2	42.94%	39.74%	40.87%
NASNetMobile	42.66%	40.70%	40.17%

Para el 11° experimento, se entrenaron 3 veces las 8 arquitecturas con cada set de entrenamiento de los grupos de frames creados a partir del proceso descrito en el Grupo 2 de frames sobre los videos de entrenamiento de los tres sets de la Versión Filtrada 1 de la base de datos HMDB51 por un total de 50 épocas y se usó la misma excepción que en el experimento 1. Para la fase de prueba, se usó el set de prueba correspondiente a cada grupo de datos de frames y la etiqueta final de cada video se obtuvo promediando las predicciones de los N frames generados a partir de cada video. Se reportó el promedio de las 3 ejecuciones para cada arquitectura en cada set de prueba. Los resultados de este experimento se muestran en la Tabla 15.

Tabla 15: Accuracy para las distintas arquitecturas con entrenamiento previo en la base de datos ImageNet en los distintos sets de prueba de la versión filtrada 1 de la base de datos HMDB51.

Arquitectura	Accuracy Set 1	Accuracy Set 2	Accuracy Set 3
EfficientNetB0	46.56%	45.73%	43.99%
EfficientNetB3	48.63%	46.41%	44.92%
Xception	51.79%	49.19%	49.67%
InceptionV3	47.17%	45.08%	46.19%
ResNet152	44.29%	40.87%	41.37%
DenseNet201	44.90%	41.94%	43.79%
MobileNetV2	43.18%	40.78%	40.94%
NASNetMobile	41.81%	41.35%	39.89%

Una vez obtenidos estos resultados, se hizo una comparación entre el promedio de los tres sets de cada base de datos de videos para cada arquitectura con el fin de verificar, cual base de datos contenía los videos más descriptivos (véase Tabla 16).

Tabla 16: Promedio de accuracy en los 3 sets de prueba de las distintas bases de datos de videos para cada arquitectura.

Arquitectura	Original	Versión Filtrada 1	Versión Filtrada 2
EfficientNetB0	46.98%	45.43%	46.07%
EfficientNetB3	47.85%	46.65%	46.38%
Xception	51.92%	50.22%	50.49%
InceptionV3	47.35%	46.15%	46.19%
ResNet152	42.07%	42.18%	42.19%
DenseNet201	44.05%	43.54%	43.52%
MobileNetV2	42.10%	41.63%	41.18%
NASNetMobile	41.76%	41.02%	41.18%

Analizando la Tabla 16, se observó que los videos que aportan mejores resultados a las arquitecturas son los videos originales de la base de datos HMDB51. Por lo que los siguientes experimentos se hicieron en base a los modelos generados con la base de datos HMDB51 original y los grupos de frames de los tres sets generados a partir del procedimiento descrito en el Grupo 2 de frames.

El 12° experimento consistió en formar los ensambles de arquitecturas de CNNs, para esto se hicieron ensambles de las 3 y 5 mejores redes y se usaron los métodos de votación y promediado simple para obtener la etiqueta final de cada video. En este experimento se usaron los frames de prueba de cada grupo de frames que se generaron a partir de los videos de prueba de cada set. Se desarrollaron 6 diferentes ensambles, los cuales se describen a continuación:

- Ensamble 1: Ensamble de las mejores 3 arquitecturas de cada set usando un sistema de votación simple para generar la etiqueta final.
- Ensamble 2: Ensamble de las mejores 5 arquitecturas de cada set usando un sistema de votación simple para generar la etiqueta final.
- Ensamble 3: Ensamble de las mejores 3 arquitecturas de cada set usando un sistema de votación con peso para generar la etiqueta final.
- Ensamble 4: Ensamble de las mejores 5 arquitecturas de cada set usando un sistema de votación con peso para generar la etiqueta final.
- Ensamble 5: Ensamble de las mejores 3 arquitecturas de cada set usando un sistema de promediado de predicciones para generar la etiqueta final.
- Ensamble 6: Ensamble de las mejores 5 arquitecturas de cada set usando un sistema de promediado de predicciones para generar la etiqueta final.

Se reportó el accuracy de cada ensamble en cada set, así como el promedio general de cada ensamble en los 3 sets. La Tabla 17 muestra los resultados de este experimento.

Tabla 17: Accuracy para cada ensamble en los distintos sets de prueba de la base de datos HMDB51. Predicción final generada con los frames de prueba de cada set.

Ensamble/TTS	TTS1	TTS2	TTS3	Promedio
Ensamble 1	52.88%	48.30%	50.00%	50.39%
Ensamble 2	53.92%	49.80%	50.26%	51.33%
Ensamble 3	54.31%	50.39%	51.31%	52.00%
Ensamble 4	54.64%	50.98%	51.50%	52.37%
Ensamble 5	54.77%	51.11%	51.90%	52.59%
Ensamble 6	52.94%	50.33%	51.90%	51.72%

El 13° y último experimento consistió en utilizar los ensambles 1, 2, 5 y 6 del experimento 12 pero usando todos los frames de cada video de prueba de cada set. Se reportó el accuracy de cada ensamble en cada set, así como el promedio general de cada ensamble en los 3 sets. La Tabla 18 muestra los resultados de este experimento.

Tabla 18: Accuracy para cada ensamble en los distintos sets de prueba de la base de datos HMDB51. Predicción final generada con todos los frames de cada video de cada set.

Ensamble/TTS	TTS1	TTS2	TTS3	Promedio
Ensamble 1	52.75%	51.37%	51.18%	51.77%
Ensamble 2	53.27%	50.65%	50.59%	51.50%
Ensamble 5	54.64%	53.07%	52.42%	53.38%
Ensamble 6	52.29%	52.03%	52.22%	52.18%

5.2 Discusión y comparación con el estado del arte

De acuerdo a los resultados obtenidos, se pudo observar que la arquitectura que obtiene el mejor desempeño en accuracy es la red Xception, seguida de las redes EfficientNetB0, EfficientNetB3 e InceptionV3. Esto concuerda con la hipótesis que planteaba que las arquitecturas que se desempeñaban mejor en la base de datos ImageNet tendrían a obtener mejores resultados en la base de datos de estudio. Un análisis en profundidad de dichas bases de datos muestra que todas ellas hacen uso de DSC normales o invertidas y solo la red InceptionV3 no cuenta con conexiones residuales. Sin embargo, solo la red Xception elimina la función de activación intermedia entre los dos pasos de una DSC. Con esto en claro se puede intuir que aquellas arquitecturas basadas en DSC tienden a favorecer el desarrollo de modelos más acertados en el reconocimiento de actividades en video. Además, durante las pruebas con el Grupo 3 de frames se pudo observar, que la red Xception e InceptionV3 se perjudicaron de un mayor número de imágenes, ya que muchas de ellas no proyectaban la actividad correctamente. Por lo que se puede inducir que ambas redes trabajan mejor con imágenes donde se muestre claramente la actividad.

Por el lado del desempeño de las arquitecturas en cuanto a tiempo, la red Xception fue la red que obtuvo el mayor tiempo de entrenamiento, mientras que la red MobileNetV2 fue la más rápida de todas, con un tiempo de ejecución de 4 a 5 veces más rápido que el tiempo obtenido por la red Xception. Aun cuando ambas redes comparten un número de parámetros similar (véase Tabla 19), en los experimentos realizados la red Xception es casi 3 veces más lenta que la red InceptionV3. Para verificar la causa de esta situación, se procedió a cuantificar el tiempo promedio por época que le tomaba a cada arquitectura tanto evaluar los frames de entrenamiento como entrenarse con todos los datos de entrenamiento usando los frames de entrenamiento del grupo 1 de frames. Se utilizo la tarjeta gráfica NVIDIA GFORCE GTX 1060 para este experimento y se promedió el tiempo de 5 épocas. Una vez obtenidos estos resultados, se procedió a calcular el tiempo utilizado por cada arquitectura

para hacer la actualización de pesos, el cual se calculó restando el tiempo de evaluación al tiempo de entrenamiento. De acuerdo a los resultados, la diferencia en tiempos entre la red Xception e InceptionV3 se debe a la lenta propagación de la información a través de la arquitectura, especialmente en la etapa de actualización de pesos (véase Tabla 20). De acuerdo a los resultados el mejor balance entre tiempo y accuracy lo obtuvieron las redes EfficientNetB0 e InceptionV3 (véase Tabla 20).

También se confirmó que todas las arquitecturas, tienden a cometer la mayor cantidad de errores en las mismas clases (véase Apéndice 2). Esto se debe a la similitud existente entre los frames de varias clases. Al no considerar la naturaleza temporal de los videos, los frames relativos a actividades como sit y stand, o como run, walk y turn pueden confundirse fácilmente (véase Ilustración 18). Se hizo un promedio de los aciertos que generaron todas las arquitecturas en cada clase en los videos de test del set 1 y de los 30 videos, hubo 18 clases que no pasaron de un promedio de 10 aciertos (véase Tabla 21).

Tabla 19: Número de parámetros contenidos en cada arquitectura.

Arquitectura	Parámetros
EfficientNetB0	5,330,571
EfficientNetB3	12,320,535
Xception	22,910,480
InceptionV3	23,851,784
ResNet152	60,419,944
DenseNet201	20,242,984
MobileNetV2	3,538,984
NASNetMobile	5,326,716

Ilustración 18: Frames de tres actividades similares. Run (izquierda), Walk (Centro), Turn (Derecha).

Tabla 20: Accuracy y tiempos de evaluación, actualización de pesos y entrenamiento promedio por época para cada arquitectura en el Grupo 1 de frames.

Arquitectura	Tiempo de evaluación (frames de entrenamiento)	Tiempo de actualización de pesos	Tiempo de entrenamiento	Accuracy
EfficientNetB0	111.7 s	419.74 s	531.44 s	47.39%
EfficientNetB3	208.33 s	830.94 s	1039.27 s	47.84%
Xception	389.17 s	1538.89 s	1928.06 s	51.33%
InceptionV3	226.90 s	612.48 s	839.38 s	48.21%
ResNet152	446.20 s	1092.09 s	1538.29 s	44.81%
DenseNet201	254.11 s	774.75 s	1028.86 s	45.95%
MobileNetV2	78.95 s	271.41 s	350.36 s	42.53%
NASNetMobile	145.85 s	545.46 s	691.31 s	43.86%

Un punto a considerar en los resultados obtenidos es que el desempeño obtenido en accuracy de la red EfficientNetB3 fue inferior al desempeño obtenido en la red Xception en todas las pruebas, aun cuando la primera tiene mejores resultados en la base de datos ImageNet. Se cree que esto fue así debido a que la red Xception utilizó imágenes de entrada de 299x299 pixeles en lugar de imágenes de 224x224 pixeles que utiliza la red EfficientNetB3. Para probar o desmentir esta hipótesis, se hizo un experimento en donde se comparó el promedio de accuracy en 3 ejecuciones de ambas redes usando el grupo 2 de frames y redimensionado las imágenes de entrada a 224x224 (véase Tabla 22).

Tabla 21: Promedio de aciertos en el set 1 de test para las clases con peor accuracy.

Clase	Aciertos	Clase	Aciertos
Cartwheel	1.36	Somersault	8.82
Clap	8.45	Stand	5.46
Fall_Floor	5.89	Swing_Baseball	3.47
Hit	6.28	Sword	9.64
Jump	5.47	Sword_Exercise	4.57
Kick	5.61	Throw	3.22
Pick	5.34	Turn	9.59
Punch	9.62	Walk	6.86
Sit	6.86	Wave	0.69

Tabla 22: Accuracy para la red EfficientNetB3 y Xception usando imágenes de entrada de 224x224 en el set de prueba del Grupo 2 de frames.

Arquitectura	Accuracy
EfficientNetB3	50.70%
Xception	52.46%

De acuerdo a la Tabla 22, se observa que aun usando imágenes de entrada de 224x224, la red Xception supera significativamente a la red EfficientNetB3. De esta forma se invalida la hipótesis planteada anteriormente y se confirma que el desempeño de la red Xception se debe principalmente a su arquitectura interna y no a la resolución de entrada.

De hecho, este resultado concuerda con el análisis hecho por Agarwal [66], el cual muestra un rendimiento superior en accuracy de la red Xception en comparación con las redes EfficientNet en una base de datos de perros y gatos ofrecida por Kaggle. Con esto podemos confirmar que, si bien el puesto en accuracy de una arquitectura en relación a la base de datos

ImageNet es un buen indicador de que tan buena es una arquitectura, no significa que su posición es absoluta para otras bases de datos.

El desempeño de las arquitecturas con las bases de datos filtradas se debió al hecho de que los videos dentro de ellas no eran lo suficientemente descriptivos en comparación con los videos originales. Un análisis posterior reflejó que la calidad de dichos videos se debió principalmente al hecho de que se cargó un archivo de binarizador de etiquetas incorrecto a la hora de armar los nuevos videos. Esto provocó que las etiquetas generadas de las predicciones de los frames de los videos referentes a las últimas 48 clases estuvieran equivocadas siempre y por ende se ejecutaran las condiciones donde se tenía mayor cantidad de frames con una predicción errónea, llevando a una construcción de videos con frames aleatorios. Sin embargo, estos resultados dejaron en claro que un video contiene cierta cantidad de frames discriminativos los cuales de ser removidos afectan significativamente el accuracy final del modelo a la hora de la clasificación.

Cabe recalcar que, por cuestiones de tiempo no fue posible generar de nuevo los videos de las versiones filtradas usando el archivo de binarizador de etiquetas correcto ni tampoco fue posible comparar el rendimiento de las arquitecturas estudiadas con una base de datos distinta a la HMDB51 para poder generalizar mejor los resultados a otras basas de datos. Sin embargo, se espera que con este trabajo se tenga al menos una noción de los resultados que se pueden esperar al usar otras bases de datos, en lo que se refiere a accuracy y tiempo de las distintas arquitecturas estudiadas.

Además, también se observó que los ensambles dieron mejores resultados cuando se consideraba un mayor número de frames por video. Estos últimos resultados dependen directamente de la calidad de los videos. Si el video representa fielmente la actividad en la mayoría de sus frames, es muy posible que el clasificador tenga un buen desempeño. En caso contrario, si un video tiene varios cambios de escenas con diverso número de actividades diferentes, entonces es muy probable que el clasificador tienda a cometer un mayor número de errores, ya que el método utilizado en el presente trabajo se enfocó en promediar las

predicciones a nivel frame y no en realizar predicciones usando la característica de movimiento de los videos.

Con fines de comparación, la Tabla 23 muestra el mejor resultado obtenido en este trabajo contra los resultados obtenidos en el estado del arte. Si bien el objetivo de este trabajo no es desarrollar un modelo que supere el accuracy del estado del arte, si no el realizar un análisis del comportamiento de las diferentes arquitecturas de estudio en la base de datos HMDB51, se considera que dicha comparación sirve como referencia para observar la diferencia existente en accuracy entre los modelos que consideran la noción de movimiento en los videos contra aquellos (como el trabajo actual) que no lo hacen. Además, también se puede observar, que una gran parte de los modelos externos mencionados utilizan múltiples GPUs en paralelo para llevar a cabo sus experimentos, lo que les permitía utilizar un mayor número de frames, una taza de aprendizaje adaptativa, entre otros factores para incrementar su accuracy, cosa que no fue posible probar en este trabajo por falta de recursos y tiempo.

Tabla 23: Comparación con el estado del arte en la base de datos HMDB51.

Ref.	Enfoque	GPU	Accuracy
[17]	Two-Stream	1 NVIDIA TITAN X flujo espacial 2 NVIDIA TITAN X flujo temporal	82.1%
[21]	Two-Stream	4 NVIDIA TITAN	59.4%
[33]	Two-Stream	4 NVIDIA TITAN X	69.4%
[18]	3D CNN	16-64 K40 GPU	80.9%
[42]	3D CNN	2-64 GPU	80.5%
[45]	RNN	---	41.3%
[46]	RNN	1 NVIDIA GTX1060	69.3%
Tesis	NA	1 NVIDIA GTX 1080	53.38%

Capítulo 6: Conclusiones

Como principales conclusiones de este trabajo, se pueden enumerar las siguientes:

1. El posicionamiento en cuanto a desempeño en accuracy en la base de datos ImageNet no es absoluto para otras bases de datos, como fue el caso de la red Xception e InceptionV3 que invirtieron posiciones en la base de datos HMDB51.
2. Los experimentos 7 y 8 demostraron que existe un balance en lo que respecta a cuánto puede mejorar el desempeño en accuracy de ciertas arquitecturas al incluir un mayor número de frames usando aumentación de datos, pero al mismo tiempo aumentar la cantidad de frames no descriptivos de una clase.
3. La red EfficientNetB0 tiende a mostrar mejor desempeño cuando se usa un grupo grande de frames con una cantidad moderada de ruido como lo demostró el experimento 7. Esto también indica que la red Xception es menos tolerante al ruido en frames que la red EfficientNetB0.
4. Aunque el desarrollo de los videos filtrados a través de la detección de escenas no se llevó a cabo correctamente, los resultados mostraron que existen frames más significativos que otros, que al ser omitidos afectan negativamente el accuracy de los modelos.
5. De los métodos de ensamble utilizados se comprobó que a mayor número de frames a considerar en la decisión final del ensamble mejor el porcentaje de clasificación final. Además, se comprobó que usando un promediado de predicciones se obtiene un mejor resultado que usando un sistema de votación.

Como trabajo a futuro, se pretende realizar un análisis más a detalle de los resultados obtenidos con las versiones filtradas, utilizar un mayor número de arquitecturas y arquitecturas de más reciente creación en el análisis, distinto número de frames extraídos por videos y de ser posible incluir flujo óptico en el modelo final para obtener un modelo con mejor desempeño en accuracy.

Los códigos utilizados en el presente trabajo se encuentran en este link https://github.com/David1094/CNN-Architecture-Comparison-on-HAR.

Referencias

[1] Y. Wan, Z. Yu, Y. Wang, and X. Li, “Action Recognition Based on Two-Stream Convolutional Networks with Long-Short-Term Spatiotemporal Features,” *IEEE Access*, vol. 8, pp. 85284–85293, 2020.

[2] I. Laptev, “On space-time interest points,” *Int. J. Comput. Vis.*, vol. 64, no. 2–3, pp. 107–123, 2005.

[3] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie, “Behavior recognition via sparse spatio-temporal features,” *IEEE Int. Work. Vis. Surveilllance Perform. Eval. Trackind Surveill.*, pp. 65–72, 2005.

[4] G. Willems, T. Tuytelaars, and L. Van Gool, “An efficient dense and scale-invariant spatio-temporal interest point detector,” in *European conference on computer vision*, 2008, pp. 650–663.

[5] H. Wang, A. Kläser, C. Schmid, and C. L. Liu, “Action recognition by dense trajectories,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3169–3176, 2011.

[6] H. Wang and C. Schmid, “Action recognition with improved trajectories,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 3551–3558, 2013.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, pp. 770–778, 2016.

[8] C. Szegedy *et al.*, “Going deeper with convolutions,” *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, pp. 1–9, 2015.

[9] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale

image recognition," *arXiv Prepr. arXiv1409.1556.*, 2014.

[11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, pp. 4700–4708, 2017.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 248–255, 2009.

[13] H. Yang, C. Shi, and H. Li, "Acoustic scene classification using CNN ensembles and primary ambient extraction," *2019 Chall. Detect. Classif. Acoust. Scenes Events*, 2019.

[14] J. Dolz, C. Desrosiers, L. Wang, J. Yuan, D. Shen, and I. Ben Ayed, "Deep CNN ensembles and suggestive annotations for infant brain MRI segmentation," *Comput. Med. Imaging Graph.*, vol. 79, p. 101660, 2020.

[15] A. Serbetci and Y. S. Akgul, "End-to-end training of CNN ensembles for person re-identification," *Pattern Recognit.*, vol. 104, 2020.

[16] G. Cong, G. Domeniconi, C. C. Yang, J. Shapiro, F. Zhou, and B. Chen, "Fast neural network training on a cluster of GPUs for action recognition with high accuracy," *J. Parallel Distrib. Comput.*, vol. 134, pp. 153–165, 2019.

[17] J. Zhu, Z. Zhu, and W. Zou, "End-to-end Video-level Representation Learning for Action Recognition," *2018 24th Int. Conf. Pattern Recognit.*, pp. 645–650, 2018.

[18] J. Carreira and A. Zisserman, "Quo Vadis, action recognition? A new model and the kinetics dataset," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, pp. 6299–6308, 2017.

[19] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb51: A large video

database for human motion recognition," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011, pp. 2556–2563.

[20] K. Soomro, A. R. Zamir, M. Shah, and A. Recognition, "UCF101 : A Dataset of 101 Human Actions Classes From Videos in The Wild," *arXiv Prepr. arXiv1212.0402.*, 2012.

[21] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *Adv. Neural Inf. Process. Syst.*, vol. 1, no. January, pp. 568–576, 2014.

[22] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High Accuracy Optical Flow Estimation based on a theory for warping," in *European conference on computer vision*, 2004, pp. 25–36.

[23] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime TV-L1 optical flow," in *Joint pattern recognition symposium*, 2007, pp. 214–223.

[24] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, 2012.

[25] H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, "Two stream LSTM: A deep fusion framework for human action recognition," *IEEE Winter Conf. Appl. Comput. Vis.*, pp. 177–186, 2017.

[26] W. Limin, Y. Xiong, W. Zhe, and Q. Yu, "Towards Good Practices for Very Deep Two-Stream ConvNets," *arXiv Prepr. arXiv1507.02159.*, 2015.

[27] J. Yue Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 4694–4702, 2015.

[28] F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, pp. 1251–1258, 2017.

[29] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2818–2826, 2016.

[30] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8697–8710, 2018.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 4510–4520, 2018.

[32] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” *36th Int. Conf. Mach. Learn. ICML 2019*, pp. 10691–10700, 2019.

[33] L. Wang *et al.*, “Temporal segment networks: Towards good practices for deep action recognition,” in *European conference on computer vision*, 2016, vol. 9912, pp. 20–36.

[34] D. Xie, C. Deng, H. Wang, C. Li, and D. Tao, “Semantic Adversarial Network with Multi-Scale Pyramid Attention for Video Classification,” *Proc. AAAI Conf. Artif. Intell.*, vol. 33, pp. 9030–9037, 2019.

[35] F. He, F. Liu, R. Yao, and G. Lin, “Local fusion networks with chained residual pooling for video action recognition,” *Image Vis. Comput.*, vol. 81, pp. 34–41, 2019.

[36] C. Feichtenhofer, A. Pinz, and A. Zisserman, “Convolutional Two-Stream Network Fusion for Video Action Recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1933–1941, 2016.

[37] L. Sun, K. Jia, D. Y. Yeung, and B. E. Shi, “Human action recognition using factorized spatio-temporal convolutional networks,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 4597–4605, 2015.

[38] D. He, F. Li, Q. Zhao, X. Long, Y. Fu, and S. Wen, “Exploiting Spatial-Temporal Modelling and Multi-Modal Fusion for Human Action Recognition,” *arXiv Prepr. arXiv1806.10319*, 2018.

[39] L. Wang, P. Koniusz, and D. Huynh, “Hallucinating IDT descriptors and I3D optical flow features for action recognition with CNNs,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 8698–8708, 2019.

[40] A. J. Piergiovanni, A. Angelova, A. Toshev, and M. S. Ryoo, “Evolving Space-Time Neural Architectures for Videos,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 1793–1802, 2019.

[41] H. Yang *et al.*, “Asymmetric 3D Convolutional Neural Networks for action recognition,” *Pattern Recognit.*, vol. 85, pp. 1–12, 2019.

[42] J. C. Stroud, D. A. Ross, C. Sun, J. Deng, and R. Sukthankar, “D3D: Distilled 3D Networks for Video Action Recognition,” *IEEE Winter Conf. Appl. Comput. Vis.*, pp. 625–634, 2020.

[43] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, “Sequential Deep Learning for Human Action Recognition,” in *International workshop on human behavior understanding*, 2011, pp. 29–39.

[44] J. Donahue *et al.*, “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description,” *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, pp. 2625–2634, 2015.

[45] S. Sharma, R. Kiros, and R. Salakhutdinov, “Action Recognition using Visual

Attention," *arXiv Prepr. arXiv1511.04119.*, 2015.

[46] W. Ye, J. Cheng, F. Yang, and Y. Xu, "Two-Stream Convolutional Network for Improving Activity Recognition Using Convolutional Long Short-Term Memory Networks," *IEEE Access*, vol. 7, pp. 67772–67780, 2019.

[47] N. Jaouedi, N. Boujnah, and M. S. Bouhlel, "A new hybrid deep learning model for human action recognition," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 32, no. 4, pp. 447–453, 2020.

[48] H. Saleh, *Applied Deep Learning with PyTorch*. Birmingham: Packt Publishing Ltd, 2019.

[49] F. Bre, "Estructura de una ANN," Nov-2017. [Online]. Available: https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051. [Accessed: 23-Nov-2020].

[50] A. Rosebrock, *Deep Learning for Computer Vision with Python*, 1st ed. PyImageSearch, 2017.

[51] Shreyak, "Ejemplo de una posible estructura de una CNN," Jun-2020. [Online]. Available: https://becominghuman.ai/building-a-convolutional-neural-network-cnn-model-for-image-classification-116f77a7a236. [Accessed: 23-Nov-2020].

[52] K. Bai, "A Comprehensive Introduction to Different Types of Convolutions in Deep Learning," 2019. [Online]. Available: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215#:~:text=In 3D convolution%2C a 3D,is then a 3D data. [Accessed: 23-Nov-2020].

[53] V. Domoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv:1603.07285v2*, 2018. .

[54] I. Shafkat, “Parte 1 del proceso de convolución para una entrada multicanal,” Jun-2018. [Online]. Available: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1. [Accessed: 23-Nov-2020].

[55] I. Shafkat, “Parte 2 del proceso de convolución para una entrada multicanal,” Jun-2018. [Online]. Available: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1. [Accessed: 23-Nov-2020].

[56] K. Bai, “Convolución 1x1,” Feb-2019. [Online]. Available: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215. [Accessed: 23-Nov-2020].

[57] K. Bai, “Convolución Separable en Profundidad,” Feb-2019. [Online]. Available: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215. [Accessed: 23-Nov-2020].

[58] S.-H. Tsang, “Arquitectura Interna de la red Inception-V3,” Sep-2018. [Online]. Available: https://sh-tsang.medium.com/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c. [Accessed: 23-Nov-2020].

[59] S.-H. Tsang, “Convolución Separable en Profundidad Invertida,” Sep-2018. [Online]. Available: https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568. [Accessed: 23-Nov-2020].

[60] V. Agarwal, “Diagrama de bloques de las capas iniciales (Stem) y finales que son comunes en todas las redes de la familia EfficientNet.,” May-2020. [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142. [Accessed: 23-Nov-2020].

[61] V. Agarwal, “Diagrama de bloques de las modulos que conforman los subbloques de

las redes de la familia EfficientNet.,” May-2020. [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142. [Accessed: 23-Nov-2020].

[62] V. Agarwal, “Sub-bloques que conforman a los bloques principales de las redes de la familia EfficientNet.,” May-2020. [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142. [Accessed: 23-Nov-2020].

[63] V. Agarwal, “Arquitectura interna de la red EfficientNetB0. Cada módulo de color dentro de cada block se refiere a uno de los tres tipos de sub-bloques.,” May-2020. [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142. [Accessed: 23-Nov-2020].

[64] V. Agarwal, “Arquitectura interna de la red EfficientNetB3. Cada módulo de color dentro de cada block se refiere a uno de los tres tipos de sub-bloques.,” May-2020. [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142. [Accessed: 23-Nov-2020].

[65] F. Chollet, “Building powerful image classification models using very little data,” Jun-2016. [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html. [Accessed: 23-Nov-2020].

[66] V. Agarwal, “EfficientNet should be the goto pre-trained model or…,” May-2020. [Online]. Available: https://towardsdatascience.com/efficientnet-should-be-the-goto-pre-trained-model-or-38f719cbfe60. [Accessed: 01-Mar-2020].

Apéndices

Apéndice 1

Para crear los videos de la versión Filtrada 1 a partir de ambas listas de frames se siguieron las siguientes condiciones:

1. Si c>i, c>=30 y c>=N*0.5: Generar nuevo video con todos los frames en C.
2. Si c>i, c<30 y c>=N*0.5: Determinar el número de frames faltantes “r” para llegar a los 30 frames. Tomar el último frame en C y agregarlo a C “r” veces. Generar nuevo video con todos los frames en C.
3. Si c=0 e i*0.5>=30: Seleccionar los i*0.5 frames de la lista I que obtengan la mayor probabilidad de pertenecer a la clase correcta. Ordenar los frames elegidos de acuerdo a su orden de aparición en el video original. Generar nuevo video con la lista ordenada de los frames elegidos.
4. Si c=0 e i*0.5<30: Seleccionar los i*0.5 frames de la lista I que obtengan la mayor probabilidad de pertenecer a la clase correcta. Ordenar los frames elegidos de acuerdo a su orden de aparición en el video original. Determinar el número de frames faltantes “r” de la lista ordenada de los frames elegidos para llegar a los 30 frames. Tomar el último frame de la lista ordenada de los frames elegidos y agregarlo “r” veces a dicha lista. Generar nuevo video con todos los frames de la lista ordenada de los frames elegidos.
5. Si c<i e i*0.5>=r: Determinar el número de frames faltantes “r” de la lista C para llegar a los 30 frames. Seleccionar los “r” frames de la lista I que obtengan la mayor probabilidad de pertenecer a la clase correcta y
6. agregarlos a la lista C. Ordenar los frames de la lista C de acuerdo a su orden de aparición en el video original. Generar nuevo video con la lista ordenada C.
7. Si c<i e i*0.5<r: Seleccionar los i*0.5 frames de la lista I que obtengan la mayor probabilidad de pertenecer a la clase correcta y agregarlos a C. Ordenar los

frames de la lista C de acuerdo a su orden de aparición en el video original. Determinar el número de frames faltantes “r” de la lista ordenada C para llegar a los 30 frames. Tomar el último frame de la lista ordenada C y agregarlo “r” veces a dicha lista. Generar nuevo video con todos los frames de la lista ordenada C.

Para crear los videos de la versión Filtrada 2 a partir de ambas listas de frames se siguieron las siguientes condiciones:

1. Si c>=30 y c>=N*0.5: Generar nuevo video con todos los frames en C.
2. Si c<30 y c>=N*0.5: Determinar el número de frames faltantes “r” para llegar a 30 frames. Tomar el último frame en C y agregarlo a C “r” veces. Generar nuevo video con todos los frames en C.
3. Si c<N*0.5: Copiar video original al nuevo set de videos.

Apéndice 2

La Tabla 24 muestra el promedio de aciertos en las tres ejecuciones de las 8 distintas arquitecturas en los videos de test del set 1 usando el grupo 2 de frames.

Tabla 24: Aciertos por clase de cada arquitectura en los videos de test del set 1.

	EfficientNetB0	EfficientNetB3	Xception	InceptionV3	ResNet152	DenseNet201	MobileNetV2	NASNetMobile
Brush_Hair	21	20	21	19	20	19	19	20
Cartwheel	5	3	0	2	0	1	1	1
Catch	18	17	20	15	19	14	11	11
Chew	16	22	12	15	14	13	12	11

Clap	10	12	9	9	6	8	8	8
Climb	20	24	23	19	21	18	17	20
Climb_Stairs	19	21	17	19	16	17	17	15
Dive	17	19	21	17	16	14	16	17
Draw_Sword	12	11	11	11	11	12	11	12
Dribble	22	23	23	22	23	22	23	23
Drink	16	17	22	15	14	17	13	13
Eat	20	16	19	19	15	16	13	14
Fall_Floor	8	7	10	6	9	8	10	4
Fencing	20	20	21	13	17	16	15	12
Flic_Flac	10	15	14	11	11	11	8	10
Golf	27	27	27	26	26	26	26	27
Handstand	15	15	15	13	11	12	14	12
Hit	5	6	5	7	6	6	4	8
Hug	21	19	25	26	20	19	17	19
Jump	10	3	7	0	8	7	4	9
Kick	10	10	7	7	3	3	3	6
Kick_Ball	7	10	14	17	14	15	11	15
Kiss	25	26	25	25	25	25	26	23
Laugh	11	13	15	13	10	11	9	10
Pick	4	4	8	5	2	5	5	6
Pour	23	23	26	26	23	23	24	23
Pullup	27	27	28	28	27	28	25	24
Punch	13	7	17	6	8	2	1	11

Push	18	23	22	20	20	21	18	18
Pushup	26	24	26	24	25	26	26	22
Ride_Bike	27	28	27	28	27	27	27	27
Ride_Horse	27	26	27	26	23	24	21	22
Run	11	13	13	11	13	13	10	11
Shake_Hand	24	24	23	22	22	21	18	20
Shoot_Ball	13	13	10	17	15	16	15	14
Shoot_Bow	27	25	27	27	26	26	26	27
Shoot_Gun	19	19	21	19	15	18	15	16
Sit	7	5	9	7	7	7	8	7
Situp	26	25	28	28	24	27	27	25
Smile	17	16	18	15	17	18	17	16
Smoke	9	15	20	14	5	9	7	6
Somersault	17	12	16	9	4	6	7	7
Stand	6	6	8	7	6	6	7	6
Swing_Baseball	5	4	4	3	3	2	3	2
Sword	9	9	11	11	13	11	10	11
Sword_Exercise	4	5	7	7	1	1	4	5

Talk	18	22	18	20	20	21	18	15
Throw	3	2	6	1	2	0	5	1
Turn	11	12	9	9	9	9	9	9
Walk	7	14	10	8	7	6	6	7
Wave	1	1	3	0	1	2	1	0