UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ANALIZADOR DE VIDEOS DE RUTINAS DE CROSSFIT MEDIANTE ALGORITMOS DE VISIÓN DE COMPUTADOR

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN TECNOLOGÍAS DE LA INFORMACIÓN

DIEGO RAFAEL MENA HERNÁNDEZ

PROFESOR GUÍA:
JUAN MANUEL BARRIOS

MIEMBROS DE LA COMISIÓN:
BENJAMIN BUSTOS CARDENAS
NANCY HITSCHFELD KAHLER
DOMINGO ARTURO MERY QUIROZ

SANTIAGO DE CHILE
2020

RESUMEN

El presente trabajo de tesis aborda el problema de cómo realizar una rutina de CrossFit (deporte que consiste en realizar ejercicios funcionales a alta intensidad) de la mejor forma posible con el fin de mejorar un puntaje previo para fines de competencia o de entrenamiento, analizando la misma rutina hecha anteriormente por el atleta.

La solución planteada para este problema es una aplicación web capaz de analizar un video, por medio de algoritmos de visión de computador y machine learning, que suba el usuario, y dar los resultados obtenidos del video y la optimización de dichos resultados por medio de gráficas de línea. La aplicación web se compone de un backend, hecho en PHP utilizando el framework de desarrollo Laravel, y un frontend hecho en Javascript utilizando ReactJs y Redux como librerías básicas. Del lado del backend también se utiliza una librería de Machine Learning que es la que permite realizar la optimización de las estadísticas de una rutina.

Esta aplicación interactúa con una componente para analizar los videos. Se encarga de hacer el cálculo del flujo óptico del video a analizar, clasifica los movimientos ejecutados dentro del video y cuenta las repeticiones realizadas de cada movimiento, o los tiempos que se toma el atleta en realizar cada movimiento (dependiendo del tipo de rutina).

Para la elaboración del clasificador de movimientos, esta tesis utiliza como línea base la arquitectura del 'Two Stream Convolutional Network' propuesta por Simonyan y Zisserman en el 2014 [14], haciendo uso únicamente del *Temporal Stream*. Luego, para calcular los tiempos y contar las repeticiones de estos videos se crearon dos algoritmos que dada las clasificaciones del modelo, los FPS del video y la composición de la rutina a analizar calcula cuántas repeticiones, o tiempo tomado, se realiza por movimiento.

Para evaluar el funcionamiento del sistema se tomaron videos de 4 hombres y 3 mujeres donde realizaron 2 rutinas cada hombre y 1 cada mujer. Luego el sistema analizó cada video utilizando dos modelos de Machine Learning entrenados diferentes. Al final se vio que uno de dichos modelos resultó tener menos error que el otro y que el algoritmo de conteo de repeticiones resultó ser un poco más preciso que el de cálculo de tiempos. Su error promedio fue de un **8%**, mientras que el algoritmo del cálculo de tiempos dio un promedio de **8,7%** de error.

Tabla de Contenido

Capítulo 1

Introducción

1.1. Contexto

El CrossFit, como disciplina física, surge por primera vez en el año 1995 con el primer gimnasio certificado en California, EE UU, y hasta el día de hoy es un deporte a nivel mundial. El CrossFit consiste en ejercicios de pesas, funcionales, de gimnasia y entre otras disciplinas, desarrollados a alta intensidad en un corto periodo de tiempo (menos de una hora) utilizando el peso del cuerpo, barras olímpicas, barras de dominadas, mancuernas, kettlebells, medicine balls, remadoras, entre otros, con el fin de acumular el mayor puntaje posible. Un gran número de personas se dedican a realizar esta actividad con el fin de estar en una buena condición física o por pasión por dicho deporte y por la competencia. Esto ha hecho que muchas personas se dediquen a abrir gimnasios o impartir clases personales. Es tanto así, que hoy en día existen más de 15.000 gimnasios certificados de CrossFit en todo el mundo, más de 1.200 en América del Sur, y alrededor de 40 en todo Chile [1]. Estos gimnasios siempre realizan competencias internas o entre ellos por la naturaleza de la disciplina. CrossFit es un deporte muy competitivo y anualmente se celebran miles de competencias en todo el mundo.

En CrossFit existen diferentes tipos WODs (ó *Workout Of the Day*, como se les conoce a las rutinas de CrossFit) que se pueden separar según el puntaje o su estructura de los movimientos que la componen. Para fines de esta tesis sólo se aclararán dos tipos de WODs en base al puntaje:

- **For Time o Rounds For Time**: consiste en las rutinas donde se da un número definido de rondas y/o repeticiones, y el objetivo consiste en realizar dichas repeticiones en el menor tiempo posible. El puntaje final es un tiempo en formato mm:ss

- **AMRAP**: que significa “As Many Repetitions As Possible”, consiste en las rutinas donde dado un tiempo fijo (usualmente en minutos) el objetivo es realizar la mayor cantidad de rondas y/o repeticiones posibles, establecidas previamente por el instructor u organizador del WOD. El puntaje final es un número entero no negativo.

1.2. Problema

Esta tesis se enfoca en resolver el problema de cómo mejorar el ritmo de una rutina de CrossFit ya realizada con el fin de empatar o mejorar el puntaje previo analizando la rutina hecha por el atleta para fines de competencia o de entrenamiento.

Este problema causa que muchos atletas no sepan cómo comenzar a hacer una rutina correctamente sin gastarse al principio, o reservando mucha energía desde el comienzo y terminar sin haber hecho la misma eficientemente durante una sesión de entrenamiento o durante una competencia. Esto hace que no puedan llegar a su máximo puntaje en una rutina sin importar qué tanto, o que tan poco, se esfuercen.

La causa principal de esta mala administración es no tener un plan de antemano que indique qué tiempo tomarse en realizar cada ronda o cada movimiento, o qué tantas repeticiones realizar cada período de tiempo. Para elaborar un buen plan es necesario poder medir y analizar rutinas similares realizadas anteriormente por esa persona.

Este problema resulta ser un desafío, ya que hay que crear un modelo que sea capaz de, según datos de rutinas que haga el atleta, decir la mejor estrategia. Por otro lado, también debe analizar por medio de un video el ritmo que el atleta lleve en una rutina para recomendar una mejor estrategia.

Estas recomendaciones se pueden hacer por medio de una gráfica (ver Figura 1.1) donde se muestre el ritmo que lleva el atleta durante la rutina y el ritmo que se recomienda llevar para la próxima vez que la realice.

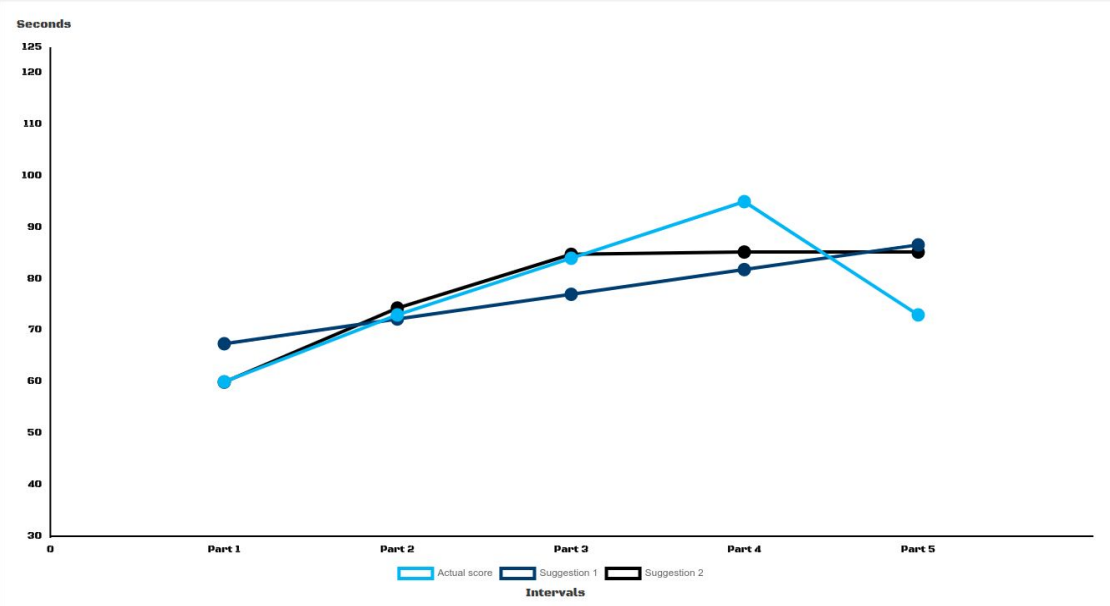


Figura 1.1. Gráfica que representa el desempeño de la rutina realizada vs a la optimizada siendo el eje X la cantidad de rondas o repeticiones y el eje Y el tiempo en segundos.

1.3. Situación actual

Actualmente, cuando un atleta desea competir en un evento nacional o internacional que tenga clasificatorias online, esta persona, habitualmente, realiza la rutina de clasificación varias veces en una semana, se graba a sí mismo haciendo cada rutina y luego sube el video en el cual haya obtenido el mejor puntaje, por rutina.

Sin embargo, esto no es eficiente ya que aquellos que analizan a profundidad el ritmo que lleva en cada intervalo de la rutina, deben ver el video completo y pausarlo en varias ocasiones para ir tomando nota de los tiempos, repeticiones, rondas, etc. Cabe destacar que hay videos que pueden ser entre 5 minutos a incluso 30 minutos o más.

Otro aspecto a considerar es que habitualmente los atletas tienen que hacer cada rutina como mínimo 2 veces, si les da el tiempo, y nada les asegura que la segunda o tercera vez que hagan la rutina, la harán con una mejor estrategia para asegurar un mejor resultado.

La solución que se plantea en esta tesis automatiza el proceso de capturar el ritmo que el atleta lleva en la rutina, y crea sugerencias de estrategias para que el atleta sepa

como debería realizar la rutina por segunda o tercera vez, en caso contrario debería repetirla y continuar con la próxima rutina.

1.4. Solución planteada

La solución planteada para este problema es una aplicación web que puede ser utilizada en las últimas versiones estables de los navegadores más comunes (Chrome, Safari, IE y Mozilla) y que es capaz de analizar un video que suba un atleta realizando un WOD durante una sesión de entrenamiento. En base a este video se obtendrán estadísticas sobre el ritmo que el atleta lleva en la rutina y en base a esto poder generar una estrategia para optimizar el puntaje obtenido (ver Figura 1.2).

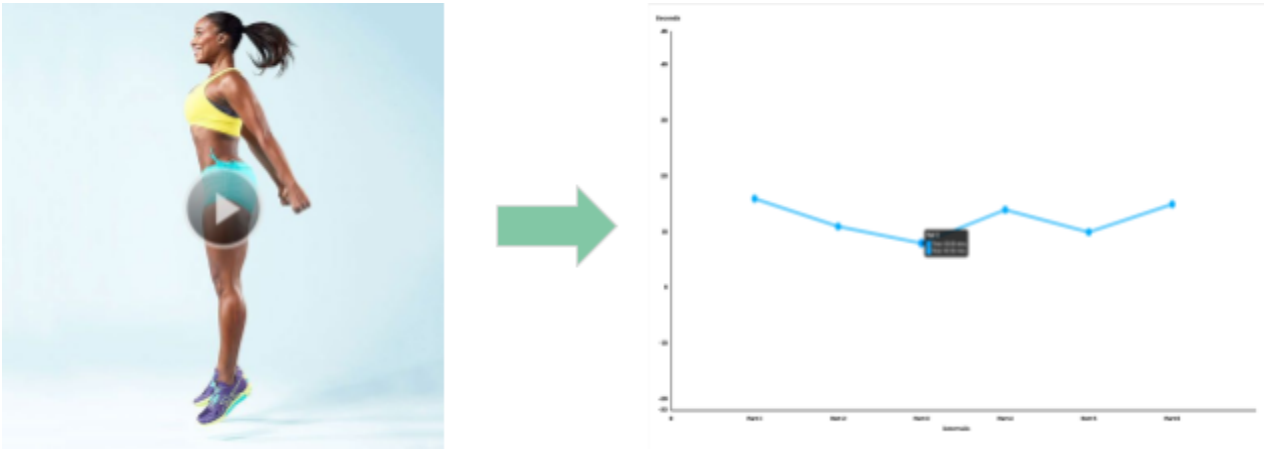


Figura 1.2. Analizar un video consiste en determinar la duración de cada ronda o movimiento, o determinar la cantidad de repeticiones realizadas en un periodo específico de tiempo con el fin de determinar el ritmo del atleta durante toda la rutina.

El video estará en posición estática (el celular, o la cámara apoyada en una superficie fija y sin movimiento, a una altura más o menos de 60 cm de alto). La cámara debe estar enfocando únicamente al atleta (sin ninguna otra entidad en movimiento) mientras éste realiza los ejercicios sin salirse el campo de visión del video. De esta forma se puede capturar todo lo que hace el atleta durante la rutina ya sea en un gimnasio, en su casa, a campo abierto, etc.

Esta aplicación interactúa con una componente para analizar los videos. Dicha componente tiene tres funciones principales, preprocesar el video del atleta, calcular el

flujo óptico del video y determinar las repeticiones realizadas ejercicio en el video. En el capítulo cuatro de esta tesis se verá en mejor detalle la arquitectura creada y las funciones de cada componente.

Se plantea una solución de software ya que resulta difícil para el atleta poder ver los puntajes que lleva cada cierto tiempo (cada ronda, cada 20 repeticiones, cada vez que termina un ejercicio, cada X minuto(s), etc.). Hay rutinas de 8 o 10 minutos con 2 o 3 movimientos, sin embargo, también hay de más de 30 minutos de duración con más de 6 o 7 movimientos, donde resulta más difícil para una persona determinar los intervalos de repeticiones o tiempos manualmente. Un software que analice un video con dicha rutina puede ser más preciso para determinar su ritmo en cualquier punto de la rutina. Por otro lado, al tener un banco de datos de su desempeño en rutinas pasadas puede encontrar la mejor forma de cómo abordar la misma rutina o una similar teniendo menos probabilidades de que falle en alcanzar su puntaje deseado.

En esta tesis se desea poder determinar el ritmo de rutinas que contengan los siguientes ejercicios que resultan ser la base de la gimnasia y, por ende, del CrossFit:

- **Sentadillas**: “consiste en **flexionar las rodillas** y bajar el cuerpo manteniendo la verticalidad, para luego regresar a una posición erguida” [4]. Ver Figura 1.3.

- **Pechadas**: “es un ejercicio realizado estando en posición inclinada, recostado hacia abajo, levantando el cuerpo únicamente con los brazos y bajando de nuevo al suelo” [5]. Ver Figura 1.4.

- **Dominadas**: “consiste en levantar el cuerpo mientras éste depende de una barra de dominadas; el agarre de la barra puede ser supinado (o sea, que las palmas de las manos miren hacia uno) o pronado (con las palmas hacia el exterior), o incluso mixto o simple (es decir, con una sola mano). Partiendo de la posición de reposo en la que los brazos se encuentran totalmente estirados, se eleva el cuerpo mediante la flexión de los brazos, hasta que la barbilla sobrepase a la barra de la cual se pende, sin elevar las piernas durante el proceso (dominadas estrictas)” [6]. Ver Figura 1.5.

Figura 1.3. Sentadilla Figura 1.4. Pechada Figura 1.5. Dominada

1.5. Objetivos de la Tesis

Objetivo General

El objetivo general de este trabajo de tesis es crear un analizador de videos dentro de una aplicación web capaz de identificar la cantidad de sentadillas, pechadas y dominadas que haga una persona en un video no mayor a 60 minutos, y determinar el ritmo que ésta llevaba en dicho video. De esta forma la aplicación podrá construir las estadísticas necesarias para sugerir al atleta una mejor estrategia.

Objetivos específicos

Los objetivos específicos se pueden listar los siguientes:

1. Construir el modelo de sentadillas entrenado para que tenga no más de un 15% de error estándar de regresión.
2. Construir el modelo de dominadas entrenado para que tenga no más de un 15% de error estándar de regresión.
3. Construir el modelo de pechadas entrenado para que tenga no más de un 15% de error estándar de regresión.

1.6. Contenido de la Tesis

En esta tesis se verán los siguientes temas

- **Marco teórico**: aquí se verán tópicos que fueron relevantes para elaborar la solución. En esta sección se hablará de **flujo óptico** (disperso y denso), **redes neuronales**, específicamente **redes convolucionales** y **métricas de evaluación** utilizadas para medir qué tan buena resultó ser la solución.

- **Estado del arte**: en este capítulo, se verán los **trabajos** e **investigaciones** hechas anteriormente por otras personas, que se utilizaron como línea base para resolver el problema previamente planteado.

- **Descripción de la solución**: aquí se verán en mejor detalle la solución construida. Se verá su **arquitectura** y sus **funcionalidades** principales, incluyendo el **optimizador de resultados** de rutinas.

- **Analizador de videos de rutinas de CrossFit**: este es el capítulo más importante de la tesis. Aquí se verá cómo se construye el **analizador de videos** de la solución y lo que hay que **considerar** para llegar al mismo.

- **Implementación**: en este capítulo se toman los hallazgos encontrados en el capítulo anterior, y se comienza a implementar, **paso a paso**, el **analizador de videos** de la solución.

- **Evaluación y resultados**: aquí se podrán ver los **resultados obtenidos** una vez puesta a prueba la solución en **escenarios** y **casos reales**.

- **Conclusión**: finalmente, en este capítulo se podrán ver los **logros** obtenidos, y las posibles mejoras para un **trabajo futuro**.

Capítulo 2

Marco Teórico

Algunos tópicos importantes que se tomarán en cuenta para la elaboración de la solución son mencionados en este capítulo.

2.1. Flujo óptico

El flujo óptico es una técnica de análisis de videos que permite determinar el movimiento de un objeto dentro de una secuencia de *frames* en un video utilizando sus vectores de movimiento.

Los vectores de movimiento, según el estándar H.264/MPEG-4 AVC, son vectores de dos dimensiones usados para predecir el desplazamiento de las coordenadas de un bloque de un *frame* (una simple imagen dentro de un video), desde las coordenadas de un bloque de referencia de un *frame* anterior [16].

Según Al Bovik (2009), el flujo óptico, es un método fundamental del cálculo de movimiento de las intensidades de una imagen, que puede ser atribuido al movimiento de objetos en una escena en un video. Los métodos de flujo óptico se basan en cálculos del movimiento de las intensidades de la imagen en un tiempo determinado de un video [15].

Qiuhong Ke et al. (2018) en [17] indican que existen muchas formas de estimar el flujo óptico entre dos *frames,* siendo el método diferencial (cálculo de vectores) el más usado. Este método asume que la intensidad de la imagen es constante entre los *frames* de un video. Dado un video, la intensidad de un voxel (o pixel volumétrico) en position (x, y) del *t*-ésimo *frame* sea $I(x, y, t)$. De acuerdo con la suposición de que la intensidad de la imagen es constante, la intensidad del voxel también será la misma a pesar de pequeños cambios de posición o periodos de tiempo. En otras palabras [17]:

$$I(x,\ y,\ t)\ =\ I(x\ +\ \delta x,\ y\ +\delta y,\ t\ +\ \delta t),$$

donde $(\delta x,\ \delta y,\ \delta t)$ es el pequeño cambio del movimiento.

$I(x,\ y,\ t)\ =\ I(x\ +\ \delta x,\ y\ +\delta y,\ t\ +\ \delta t)$ puede ser expresado con una serie de Taylor de la siguiente manera:

$$I(x\ +\ \delta x,\ y\ +\delta y,\ t\ +\ \delta t) \approx I(x,\ y,\ t)\ +\ \frac{\partial I}{\partial x}\delta x\ +\frac{\partial I}{\partial y}\delta y\ +\ \frac{\partial I}{\partial t}\delta t.$$

Por tanto,

$$\frac{\partial I}{\partial x}\delta x\ +\frac{\partial I}{\partial y}\delta y\ +\ \frac{\partial I}{\partial t}\delta t \approx 0,$$

$$\frac{\partial I}{\partial x}U_x\ +\frac{\partial I}{\partial y}y\ +\ \frac{\partial I}{\partial t} \approx 0,$$

donde $U_x\ =\ \frac{\delta x}{\delta t}$, $U_y\ =\ \frac{\delta}{\delta t}$ son dos componentes del flujo óptico del voxel, o pixel, $(x,\ y,\ t)$.

Existen dos variantes principales al momento de estimar el flujo óptico de un video: *flujo óptico disperso* y *flujo óptico denso.*

2.1.1. Flujo óptico disperso

El flujo óptico disperso selecciona un conjunto de pixeles dispersos (características interesantes como bordes y esquinas) para rastrear sus vectores de movimiento. Las características se pasan a la función de cálculo de flujo óptico de *frame* a *frame* para garantizar que se sigan los mismos puntos. Un ejemplo de implementación de cálculo de flujo óptico disperso, y probablemente el más conocido, es el algoritmo de Lucas-Kanade [18]. Este algoritmo asume que el flujo es constante dentro de los vecinos de un pixel en consideración, y consiste en resolver la ecuación del flujo óptico (mostrada anteriormente) para todos los píxeles de esa zona de 3x3 alrededor del píxel en cuestión, usando el criterio de los *cuadrados mínimos* [18][19].

Esta variante es más rápida que el flujo denso de calcular y requiere menos costo computacional, sin embargo resulta ser menos precisa que el flujo óptico denso. En la Figura 2.1 se puede ver un ejemplo de la representación del flujo óptico disperso usando Lucas-Kanade.

Figura 2.1. Flujo óptico disperso de vehículos [24].

2.1.2. Flujo óptico denso

El flujo óptico denso calcula un vector de movimiento para cada uno de los pixeles de los *frames* del video. Si bien es cierto que el cálculo puede ser más lento que el del flujo óptico disperso, este proporciona un resultado más preciso, ideal para aplicaciones de segmentación de videos y clasificación de movimientos en video, como el que se presenta en esta tesis. Es por esta razón que se decide utilizar esta variante para cumplir los objetivos de esta tesis.

En esta tesis se utiliza el algoritmo de Gunnar-Farneback. Este algoritmo se puede resumir de la siguiente manera: primero se aproximan bloques de píxeles de dos frames utilizando polinomios cuadráticos, específicamente usando una transformación de expansión polinomial [21]. Segundo, observando cómo los polinomios se transforman bajo movimiento, se utiliza un método para estimar el desplazamiento utilizando los coeficientes de dicha expansión polinomial. Finalmente, después de una serie de arreglos y refinamientos, se obtiene el flujo óptico denso [20].

En la Figura 2.2 se puede ver un ejemplo del flujo óptico denso usando Gunnar-Farneback.

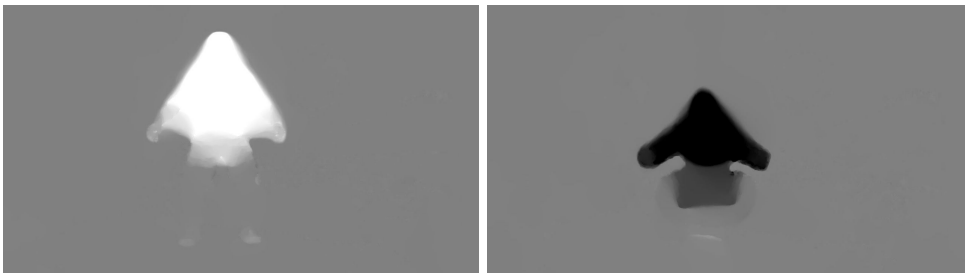


Figura 2.2. Flujo óptico denso de una sentadilla.

2.2. Red neuronal

Las redes neuronales son un modelo computacional que comparte algunas propiedades con el cerebro animal, en el que muchas unidades simples (neuronas) trabajan en paralelo sin una unidad de control centralizada. Los pesos entre las unidades son los medios principales de almacenamiento de información a largo plazo en las redes. La actualización de estos pesos es la forma primaria en que la red neuronal aprende nueva información [22].

Una forma de representar la red neuronal es por medio de la ecuación $Ax = b$. Donde A es una matriz (o tensor) con datos de entrada y b son los datos de salida que serían las etiquetas dadas a esos datos de entrada. En el contexto de la tesis, A sería los valores numéricos de un video de un ejercicio y b sería el nombre del ejercicio (pechada, sentadilla o dominada). Los pesos vendrían siendo x, que son los valores que se van estimando y actualizando a medida que la red neuronal se está entrenando [22].

Entrenar una red neuronal gira en torno a los siguientes elementos:

- **Capas,** que combinadas forman la red (o modelo). Son un módulo de procesamiento de datos que toman como entrada una o más matrices, o tensores, y tiene como salida una o más matrices o tensores. Estas capas tienen **pesos,** que son determinados por un optimizador y resultan ser el *conocimiento* de la red [23].

- **Datos de entrada** y sus correspondientes **labels o clases** de salida.

- La **función de pérdida**, que determina la cantidad que será minimizada durante el entrenamiento. Este valor representa una medida de éxito para esta tarea.

- El **optimizador,** que determina cómo la red actualiza los pesos, basándose en el valor entregado por la función de pérdida.

La red neuronal más básica y conocida es la red neuronal *feed-forward*, la cual tiene una capa de entrada, una o más capas ocultas y una capa de salida. Cada capa puede tener un número diferente de neuronas, cada una está totalmente conectada a su capa adyacente. Las conexiones entre las neuronas en las capas forman un grafo acíclico que se puede ver en la Figura 2.3

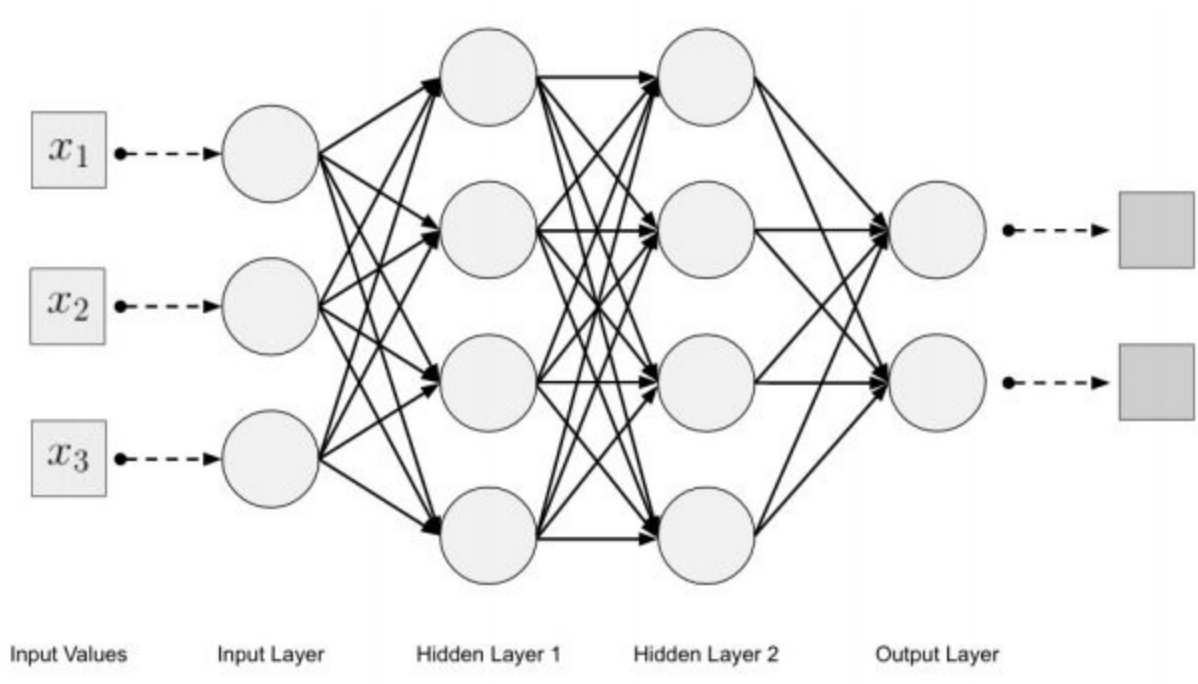


Figura 2.3. Topología de una red neuronal multicapa *feed-forward* [22].

2.2.1. Entrenamiento

Entrenar una red neuronal consiste en ajustar los pesos de las capas, haciendo algunos más pequeños y otros más grandes. Esto ayuda a que el modelo aprenda cuales predictores (o características) están relacionadas a ciertas clases o labels.

En muchos set de datos ciertas características están fuertemente correlacionadas con ciertas clases. Las redes neuronales aprenden estas relaciones según la entrada y los

pesos, y luego mide qué tan precisos son los resultados. Las funciones de pérdida, como el gradiente estocástico de descenso (SGD sus siglas en inglés), le indican a la red cuando hacen una buena predicción y cuando no [22].

Este algoritmo de aprendizaje se llama *backpropagation*, y es el que usan las redes neuronales para ser entrenadas.

En la Figura 2.4 se puede visualizar la interacción de los elementos de la red durante su entrenamiento. En un principio, los pesos de la red son asignados de manera aleatoria. Luego la red, compuesta por capas que están conectadas entre sí, asocia los datos de entrada con predicciones. La función de pérdida compara estas predicciones con los labels, o clases, reales, produciendo un valor de pérdida: una medida de que tan bien las predicciones de la red se igualan a las clases reales. Luego, el optimizador usa este valor de pérdida para actualizar los pesos de la red de tal forma que las predicciones vayan mejorando y el valor de pérdida vaya disminuyendo. Este proceso se repite un número determinado de veces hasta que las predicciones sean correctas o se llegue a las condiciones de finalización [23].

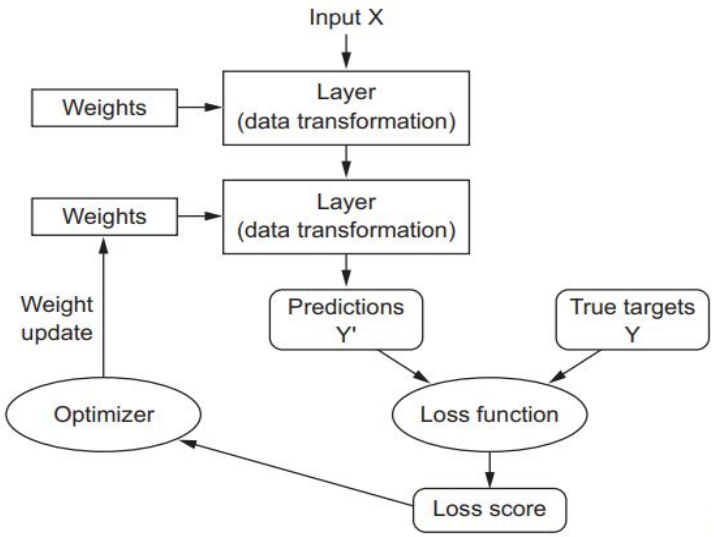


Figura 2.4. Interacción entre las capas, función de perdida y optimizador, durante el entrenamiento [23].

Otro concepto importante en el entrenamiento de una red neuronal es el **dropout**. Esta técnica consiste en remover temporalmente un conjunto de unidades (neuronas) de la red neuronal, en cada etapa de entrenamiento. Se asigna una probabilidad *p* la cual determina la proporción de unidades que seran “apagadas”. Matematicamente, esto

implica que su valor de salida será 0 en lugar de sus pesos normalmente calculados. El fin del **dropout** es evitar el *overfitting*, es decir, el sobre-entrenamiento, que sucede cuando el modelo se aprende demasiado bien los datos de entrenamiento pero no clasifica bien los datos de testing [27].

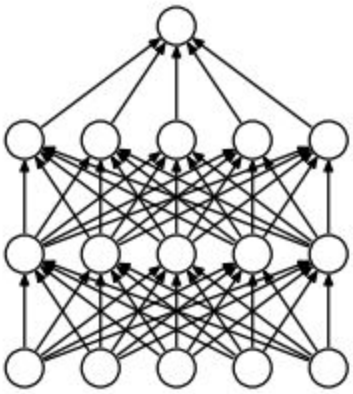


Figura 2.5. Red neuronal estándar

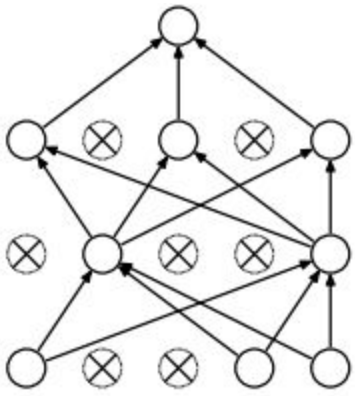


Figura 2.6. Red neuronal con dropout

2.3. Red neuronal convolucional

La red neuronal convolucional es un tipo de red neuronal cuyo objetivo es aprender características de un dato de entrada por medio de convoluciones. Este tipo de red neuronal se utiliza en reconocimiento de objetos en imágenes, reconocimiento de acciones en videos y múltiples tareas de clasificación de imágenes [22].

2.3.1. Convolución

Una convolución se define como una operación matemática que describe una regla de cómo fusionar dos conjuntos de información. Toma una entrada, aplica un *kernel* o filtro de convolución, y da un mapa de características (un tensor) como salida [22].

La entrada de una convolución puede ser data cruda o un tensor resultante de otra convolución. Usualmente la convolución es interpretada como un filtro un donde el *kernel* filtra la data de entrada por cierto tipo de información; por ejemplo, un *kernel* de bordes deja pasar información únicamente información del borde de una imagen.

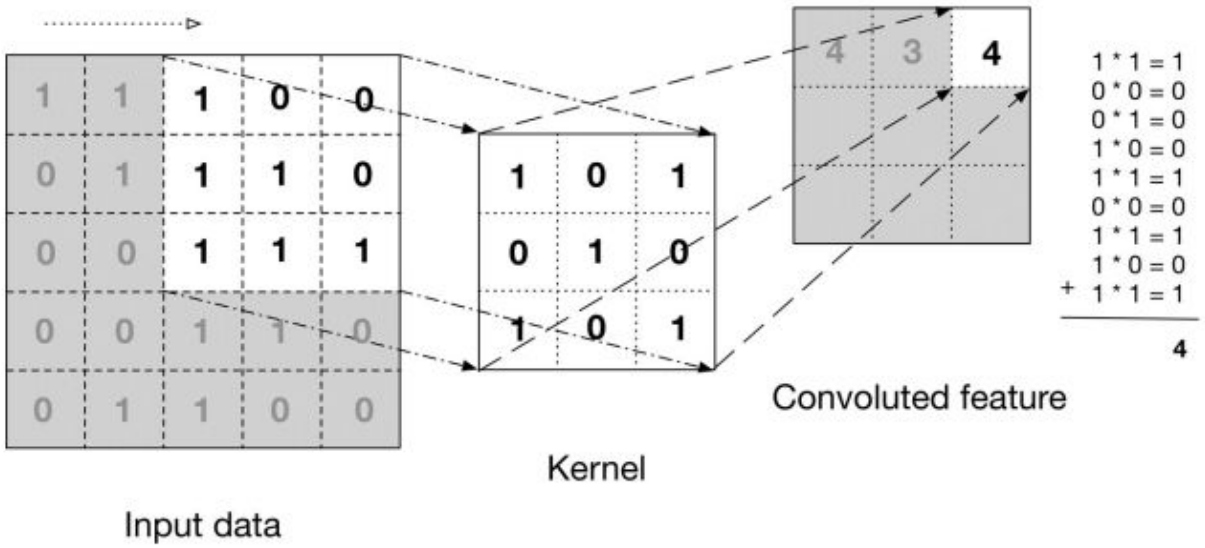


Figura 2.5. Operación de convolución [22].

La Figura 2.5 muestra como el kernel se desliza a través de la data de entrada para producir la data de salida. En cada paso, el kernel es multiplicado por los valores de la data de entrada que esté dentro de sus límites, por medio de un producto punto, creando así una simple entrada en el tensor de salida. Cada kernel aplicado a la data de entrada da como resultado un canal en el tensor de salida.

El kernel resulta ser el conjunto de pesos en una capa convolucional. Entrenar una red convolucional implica actualizar los valores de los kernels de cada capa.

2.3.2. Pooling

Otra operación, usualmente ejecutada después de una convolución, es el pooling. Este consiste en disminuir el ancho y el alto del tensor de entrada. El objetivo principal de esta operación es reducir la cantidad de parámetros dentro de una capa convolucional, con el fin de controlar el sobre-entrenamiento [22]. Un ejemplo es el max-pooling, que consiste en extraer ventanas del tensor de entrada y dando como salida el valor más alto de cada canal [23].

2.4. Métricas de evaluación

Al momento de evaluar qué tan bueno es un modelo de Machine Learning se utilizan diversas métricas que se encuentran en una **matriz de confusión**.

La matriz de confusión es una matriz nxn (donde n es la cantidad de clases a clasificar) en la que las filas son las clases reales de los datos a clasificar y las columnas son las clases predichas por el modelo.

n = 165	**Predicción**: No	**Predicción**: Sí	
Real: No	TN = 50	FP = 10	60
Real: Sí	FN = 5	TP = 100	105
	55	110	

Figura 2.6. Ejemplo de matriz de confusión.

En base a esta matriz se pueden determinar diversas métricas. Algunas de estas métricas son las siguientes:

Accuracy

Es el número de predicciones correctas hechas por el modelo entre el total de predicciones hechas por el mismo modelo.

$$Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

Precisión

La precisión de una clase es el número de predicciones positivas entre el total de predicciones hechas de esa clase.

$$Precisión = TP / (TP + FP)$$

Recall

El recall de una clase es el número de predicciones positivas entre el total de elementos reales de esa clase.

$$Recall = TP / (TP + FN)$$

Error estándar de la regresión

El error estándar de la regresión es el valor que muestra la diferencia entre los valores reales y los estimados de una regresión. Es utilizado para valorar si existe una correlación entre la regresión y los valores medidos. Esta métrica se utiliza para evaluar qué tan correcto resulta ser los resultados predichos por el sistema, en comparación a los reales hechos por el atleta, ya sea en segundos o repeticiones.

$$\widehat{\sigma} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \widehat{y_i})^2}$$

Figura 2.11. Fórmula del error estándar de la regresión donde $\hat{y}_i$ son los valores estimados, y_i son los valores medidos (o reales) y N es el tamaño de la muestra.

Capítulo 3

Estado del Arte

Múltiples trabajos e investigaciones se han realizado previamente para poder resolver el problema del reconocimiento de acciones hechas por una persona con el uso de redes convoluciones, flujo óptico y un set de datos de entrenamiento.

En este capítulo se mencionan algunos trabajos relevantes para la realización de esta tesis.

3.1. Two-Stream Convolutional Networks

En este trabajo, propuesto por Simonyan y Zisserman en 2014 [14], se plantea una arquitectura donde se puedan identificar las características principales de un objeto en movimiento en forma de un conjunto de vectores de flujo óptico.

Simonyan y Zisserman explican que un video puede ser descompuesto en una componente espacial y una componente temporal. La parte espacial, que se representa como un conjunto de *frames* individuales, tiene información sobre las escenas y objetos representados en el video. Por otro lado, la parte temporal, que se representa como un conjunto de flujos ópticos entre *frames*, transmite el movimiento de los objetos dentro del video.

De esta forma, en lugar de utilizar una sola red la cual identifique los movimientos en base a simples frames del video, proponen dos redes convolucionales separadas. Una que se encargue de la parte espacial de la acción ejecutada en el video, y otra que se encargue de la parte temporal del mismo, es decir, de la secuencia del movimiento.

La red espacial tiene como entrada un simple *frame* del video. Los autores indican que el tener una imagen estática representa una gran pista de la acción ejecutándose ya que hay algunas acciones que están fuertemente asociadas con objetos particulares. Como

por ejemplo, para hacer arquería es necesario el uso de un arco. Según los autores, esta red, por si sola, resulta ser bastante competente en el problema de clasificación de acciones, ya que esta red es, en esencia, una red de clasificación de imágenes la cual puede ser entrenada con grandes datasets predefinidos como el ImageNet [25].

Por otro lado, la entrada de la red temporal, tras varios experimentos hechos por los autores, descubrieron que un conjunto de flujos ópticos bidireccionales (vectores en los ejes X e Y) de 10 *frames* consecutivos apilados era la mejor opción como dato de entrada.

La forma de apilar estos flujos consiste en lo siguiente: calculando el flujo óptico denso de *frames* consecutivos, se tiene un conjunto de vectores de movimiento, donde un vector se puede denotar como $\mathbf{d_t}$ en un punto (u, v) de un *frame* t, donde este vector mueve dicho punto al punto correspondiente en el *frame* $t+1$. Las componentes horizontales y verticales del vector, d_t^x y d_t^y, se pueden ver como dos canales de una imagen. Para representar el movimiento entre una secuencia de *frames*, se apilan los flujos de los canales $d_t^{x,y}$ de L *frames* consecutivos para formar un total de $2L$ canales de entrada. Formalmente, sean w y h el ancho y largo de un video, el input de la red temporal $I_t \in \mathbb{R}^{w \times h \times 2L}$ para un *frame* arbitrario t es construido de esta forma:

$$I_t(u, v, 2k-1) = d^x_{t+k-1}(u, v),$$

$$I_t(u, v, 2k) = d^y_{t+k-1}(u, v),$$

$$u = [1; w],\ v = [1; h],\ k = [1; L]$$

Para un punto arbitrario (u, v), los canales $I_t(u, v, c),\ c = [1; 2L]$ representan el movimiento en ese punto sobre una secuencia de L *frames*, como se ve en la Figura 3.1.

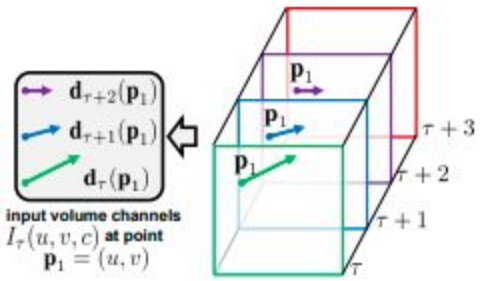


Figura 3.1. Apilamiento de flujos ópticos [14].

Una vez se tienen diseñadas ambas redes, se proceden a ser entrenadas de manera separada y se combinaban sus resultados utilizando *Support Vector Machines* y promedios.

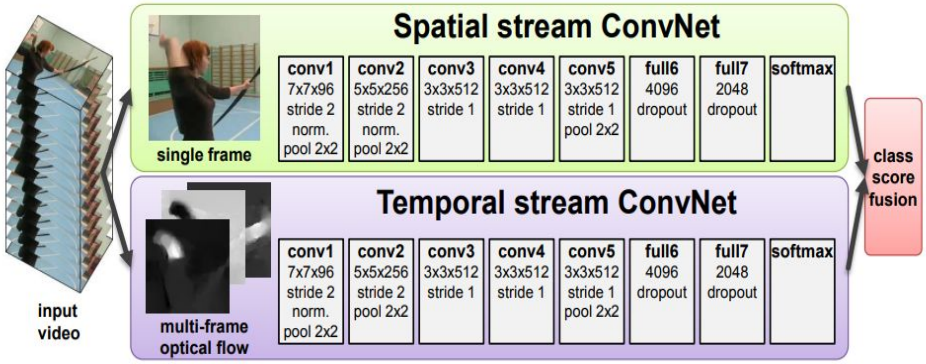


Figura 3.2. Arquitectura del Two Stream Convolutional Network [14].

Al momento de la evaluación, los autores utilizaron los datasets UCF-101 [3] y HMDB-51 [12], ambos son grandes datasets de referencia para el reconocimiento de acciones. UCF-101 contiene alrededor de 13.000 videos (180 frames por video en promedio), clasificados en 101 clases, y el HMDB tiene alrededor de 6.800 videos de 51 acciones.

Tras una serie de experimentos y evaluaciones bajo diferentes configuraciones los autores obtuvieron los resultados expresados en las siguientes tablas [14]:

Configuración	Dropout	
	0.5	0.9
Entrenamiento desde 0	42,5%	52,3%
Red pre-entrenada con ImageNet + refinamiento con UCF-101	70,8%	**72,8%**
Red pre-entrenada con ImageNet + entrenamiento solo en ultima capa	**72,7%**	59,9%

Tabla 3.1. Resultados obtenidos de la red espacial individualmente y el dataset UCF-101.

Configuración	Dropout 0.9
Optical flow con un frame	73,9%
Optical flow con 5 frames apilados	80,4%
Optical flow con 10 frames apilados	**81,0%**

Tabla 3.2. Resultados obtenidos de la red temporal individualmente con el dataset UCF-101.

Como se puede ver en las tablas 3.1 y 3.2, la red temporal por sí sola con dropout 0.9, ya que fue el dropout que usaron los autores, en base a los resultados de la red espacial, resulta tener mejores resultados que la red espacial. A continuación, se presentan los resultados del Two-stream Convolutional Network aplicando dos métodos de fusión de resultados:

Red espacial	Red temporal	Método de fusión	Precisión
Red pre-entrenada + entrenamiento ultima capa + dropout 0.9	Optical flow con 10 frames apilados + dropout 0.9	Promedio	86,2%
Red pre-entrenada + entrenamiento ultima capa + dropout + 0.9	Optical flow con 10 frames apilados + dropout 0.9	Support Vector Machine	**87,0%**

Tabla 3.3. Resultados de la fusión de las redes espaciales y temporales con UCF-101.

3.2. Hidden Two-Stream Convolutional Networks

En este trabajo, propuesto por Zhu et al. en 2017 [2], se plantea utilizar la misma arquitectura propuesta por Simonyan y Zisserman, pero en lugar de pre calcular el flujo óptico de los frames a analizar, ocupando almacenamiento y velocidad de procesamiento, deciden agregar una tercera red convolucional que se encargue de hacer el cálculo del flujo óptico, donde dichos vectores se quedan en memoria y se le pasan como entrada a la red temporal.

De esta forma no es necesario almacenar el flujo óptico en disco en forma de archivos para su posterior procesamiento, sino que una vez se tenga la secuencia de *frames* del video puede calcular el flujo óptico, y pasarlo inmediatamente a la red temporal, ahorrando así espacio en disco y velocidad de procesamiento, ya que se ahorra el tiempo de escritura y lectura en el disco duro.

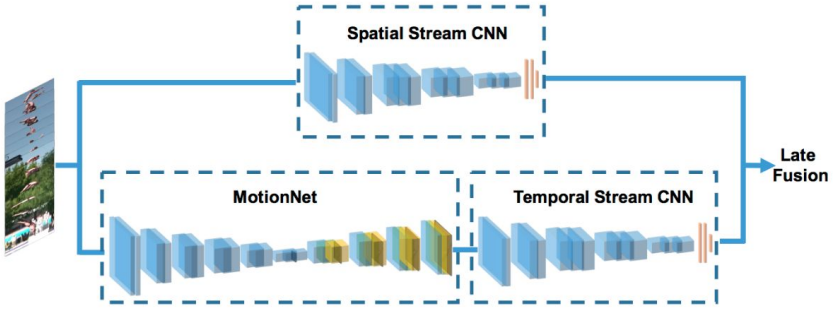


Figura 3.3. Arquitectura del Hidden Two Stream Convolutional Network [2].

La red que se encarga del cálculo del flujo óptico, **MotionNet**, consiste en una red convolucional que se compone de una parte que contrae y otra que expande. La parte que contrae es un conjunto de capas convolucionales y la parte que expande es una cadena de capas convolucionales y deconvolucionales combinadas.

Los autores establecen tres puntos importantes de esta red los cuales se presentan a continuación:

Primero, los autores diseñan la red a que se enfoque en pequeños movimientos de desplazamiento. Para datos reales, como los videos de YouTube, los autores con frecuencia se encontraban con el problema de que las acciones humanas eran pequeñas en comparación a los movimientos de fondo que eran más dominantes. Por tanto, usaron kernels de 3x3 para detectar pequeños movimientos. Las dos primeras capas convolucionales no usan stride. Se usa stride en convoluciones en lugar de pooling para reducir el tamaño de las imágenes, ya que el pooling muestra ser peor para predicciones de pocos pixeles.

Segundo, la red MotionNet calcula múltiples pérdidas a múltiples escalas. Debido a las conexiones de salto entre las partes que contraen y que expanden, las pérdidas intermedias pueden regularizarse entre sí y guiar a las capas anteriores para que converjan más rápido al objetivo final.

Tercero, indican que el aprendizaje no supervisado en flujo óptico introduce artefactos en regiones homogéneas ya que la suposición de que hay brillo se pasa por alto. Por

esto, los autores insertan capas convolucionales adicionales entre las capas deconvolucionales en la parte que expande para producir una estimación de movimiento más fluido.

Dado que la MotionNet y la red temporal ambas son redes convolucionales, se necesitan combinar estos módulos en uno para tener un entrenamiento de principio a fin. Para esto los autores plantean el método de apilamiento.

Para este método se necesita primero normalizar los flujos estimados. Cualquier movimiento que sea mayor de 20x20 pixeles lo recortan a ese tamaño. Luego se normalizan y cuantifican los frames recortados para tenerlos en un rango entre 0 - 255.

Luego, se necesita determinar cómo ajustar la red, incluyendo qué función de pérdida usar durante el ajuste. Para esto se exploraron diferentes configuraciones. (a) Arreglar la MotionNet, que significa que no se usa la función de pérdida para refinar el estimador de flujo óptico. (b) Tanto la MotionNet como la red temporal son ajustadas, pero solo la función de pérdida categórica es computada. (c) Tanto la MotionNet como la red temporal son ajustadas, y todas las funciones de pérdida son computadas. Más adelante, en la Tabla 3.4, se ve cómo la configuración (c) obtiene mejores resultados.

Y tercero, se necesita capturar movimientos de relativamente largo plazo. Para esto los autores pasan como entrada una pila de múltiples flujos consecutivos. Según Simonyan y Zisserman [14] 10 flujos resultan tener mejores resultados. Así que los autores pasaban como entrada a la MotionNet 11 frames apilados lo que les permitiría obtener 10 flujos para la red temporal.

Tras hacer diferentes experimentos utilizando el dataset UCF-101 [3] se obtuvieron los siguientes resultados en la validación:

Método	Precisión (%)	fps
Two-Stream CNNs [14]	88	14.3
Two-Stream CNNs más profunda	**90.9**	**12.8**
Hidden Two-Stream CNNs (a)	87.50	120.48
Hidden Two-Stream CNNs (b)	87.99	120.48
Hidden Two-Stream CNNs (c)	**89.82**	**120.48**

Tabla 3.3. Resultados de la fusión de las redes espaciales y temporales con UCF-101.

Con estos resultados se puede ver que aunque la Two-Stream CNN planteada por Simonyan y Zisserman [14] puede tener un poco más de precisión, la Hidden Two-Stream CNN resulta ser mucho más eficiente para realizar el proceso de principio a fin.

Estos trabajos utilizan diferentes arquitecturas de redes neuronales con el fin de poder clasificar movimientos humanos dentro de un video. Los resultados y hallazgos obtenidos en estos trabajos ayudaron como punto de partida para la elaboración de la solución de esta tesis. Sin embargo, los resultados de estos trabajos se realizaron con el dataset UCF-101 [3], el cual contiene videos de 101 actividades humanas, por lo que para fines de esta tesis es necesario construir un dataset específico para esta solución.

Actualmente no existe ningún trabajo o investigación que clasifique los ejercicios que esta tesis se enfoca con el nivel de precisión necesario para poder cumplir con el objetivo de determinar y optimizar el ritmo de una persona en una rutina.

Capítulo 4

Descripción de la solución

La aplicación web es diseñada y construida con el fin de establecer una interfaz entre el usuario y el analizador de videos. El sistema en su totalidad busca facilitar al usuario el análisis de una rutina, determinar el ritmo que lleva en dicha rutina y obtener diferentes estrategias para poder mejorar el puntaje obtenido.

En la primera sección de este capítulo se explicará en mejor detalle las componentes de este sistema, la función de cada componente y más adelante se hablará en detalle sobre la componente principal que es el analizador de video, y como se llegó a crear dicha componente.

4.1. Arquitectura

El sistema se compone de los siguientes elementos:

- **Aplicación web**: se compone de un backend, hecho en PHP utilizando el framework de desarrollo Laravel, y un frontend hecho en Javascript utilizando ReactJs y Redux como librerías básicas. Esta aplicación se encarga de hacer las operaciones CRUD básicas de las rutinas y usuarios. Del lado del backend también se utiliza una librería de Machine Learning que es la que permite realizar la optimización de las estadísticas de una rutina.

- **Manejador de Caché y Sesión**: utiliza Redis como motor de base de datos en memoria, y se utiliza para agilizar la lectura de información poco dinámica como los ejercicios disponibles para una rutina, y para manejar la sesión del usuario.

- **Base de Datos**: aquí es dónde se almacena la data persistente del sistema, como los usuarios, las rutinas, los ejercicios y las estadísticas. Utiliza MySql como motor de base de datos.

- **Analizador de Video**: es la componente más importante del sistema. Es quien se encarga de hacer el cálculo del flujo óptico del video a analizar, clasifica los movimientos ejecutados dentro del video y cuenta las repeticiones realizadas de cada ejercicio en caso de estar analizando un AMRAP. En el capítulo cinco se detallará mejor esta componente.

En la Figura 4.1 se puede ver un diagrama de contexto dónde se visualizan las interacciones entre estos componentes.

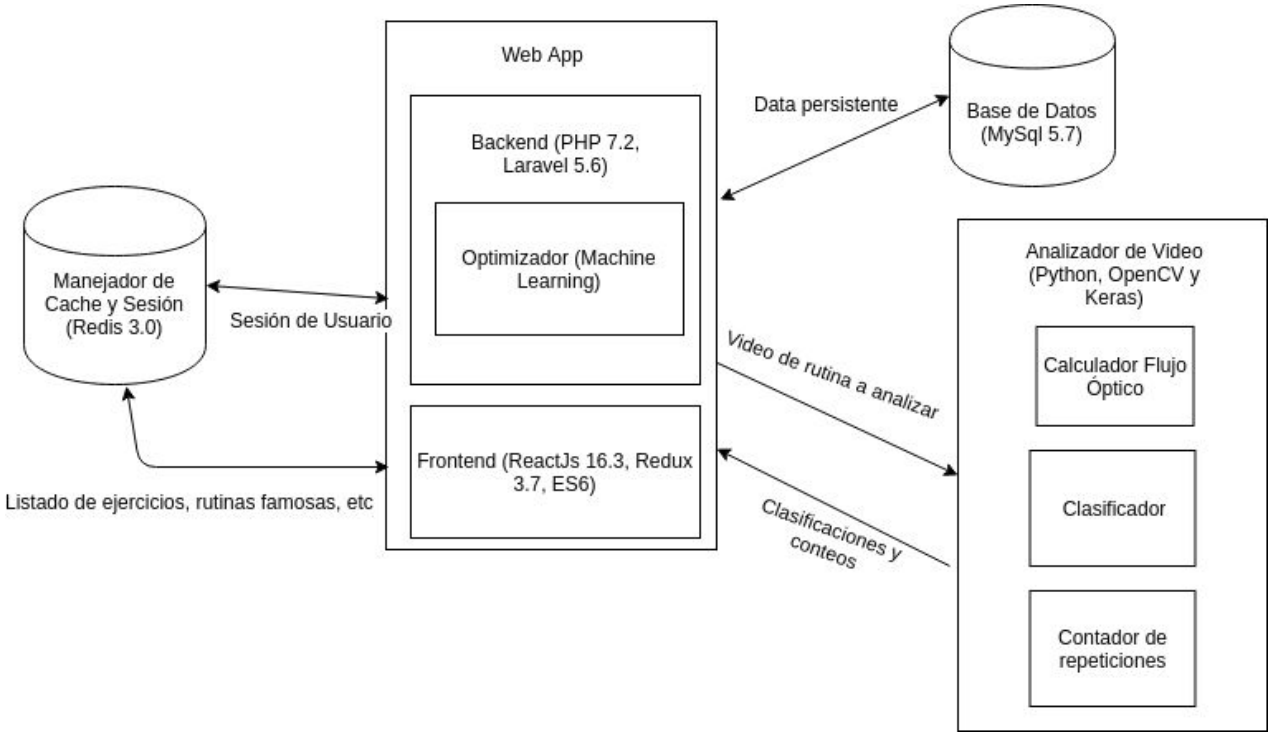


Figura 4.1. Diagrama de contexto de las componentes de la solución.

4.2. Funcionalidades

El sistema en su totalidad permite varias funcionalidades al atleta con el fin de ayudar a su entrenamiento diario y a su preparación para fines de competencia. Estas funcionalidades se pueden listar de la siguiente manera:

- **Registrarse**: consiste en crearse una cuenta en el sistema, donde el usuario ingresa su correo y una contraseña.

- **Iniciar sesión**: el usuario ingresa el sistema con su correo y contraseña establecidas cuando se registró. De igual forma el usuario puede cerrar su sesión.

- **Reiniciar contraseña**: el usuario ingresa un correo electrónico válido y el sistema le envía un enlace dónde éste puede restablecer su contraseña en caso de haberla olvidado, o simplemente si desea una diferente.

- **Agregar un puntaje a una nueva rutina**: el usuario puede agregar una rutina nueva que haya realizado y con ella ingresar manual ó por medio de un video el puntaje obtenido.

- **Agregar un puntaje a una rutina existente**: el usuario puede agregar a una rutina ya existente en el sistema e ingresar manual ó por medio de un video el puntaje obtenido.

- **Comparar diferentes puntajes de una misma rutina**: dada una rutina que tenga más de un puntaje, el usuario puede comparar los diferentes puntajes por medio de gráficas para mejor compresión.

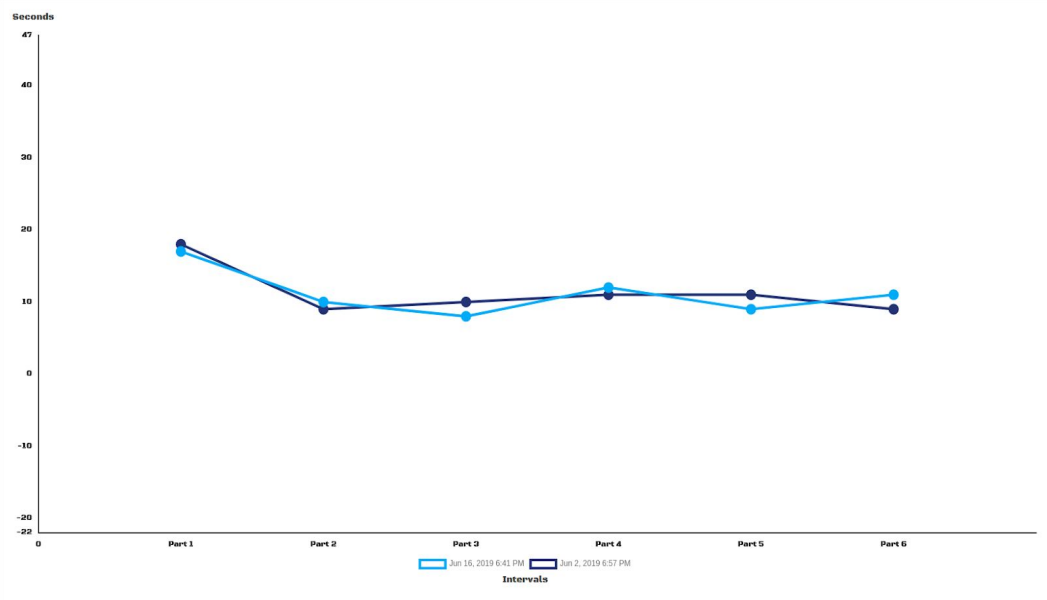


Figura 4.2. Puntajes realizados en fechas momentos diferentes para una misma rutina.

- **Ver estadísticas del puntaje de una rutina**: el usuario puede ver un gráfico de líneas correspondiente a una rutina para mejor visualización de su desempeño.

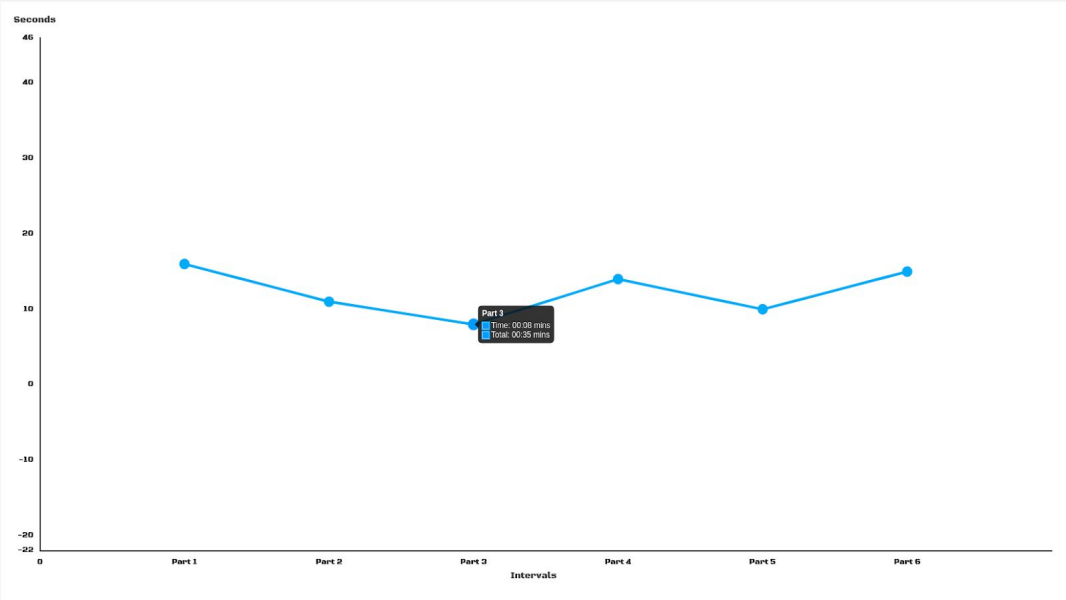


Figura 4.3. Puntaje de una rutina.

- **Optimizar una rutina**: según los resultados obtenidos en una rutina, el usuario puede generar dos estrategias posibles en forma de un gráfico de líneas.

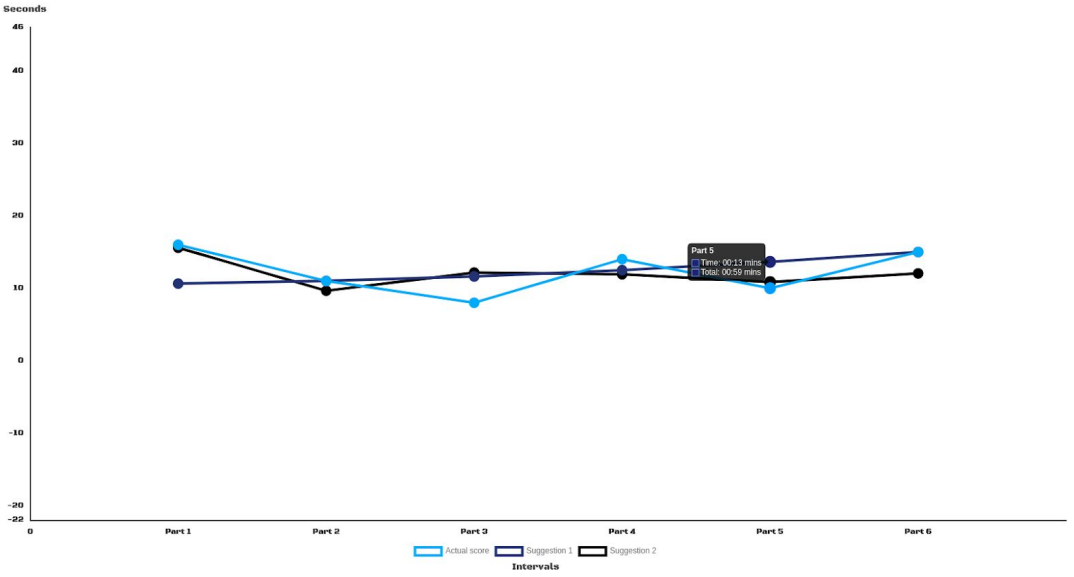


Figura 4.4. Puntaje de una rutina junto con dos sugerencias generadas por el sistema.

- **Editar un puntaje**: el usuario puede cambiar el puntaje ya sea para corregir un error o por alguna otra razón.

- **Eliminar un puntaje**: el usuario puede eliminar un puntaje que haya ingresado incorrectamente, o que no desee que siga en el sistema.

4.3. Optimización de resultados

El proceso de optimizar los resultados de un WOD consiste en crear posibles estrategias que el atleta pudiera, si así lo desea, seguir con el fin de mejorar su resultado, aunque sea en lo más mínimo (1 segundo menos, o una repetición más) en un WOD ya realizado. Esto sirve para fines de competencia o clasificatorias para una competencia.

En competencias de CrossFit se suelen publicar al menos uno o dos WODs previamente a la fecha de la competencia con el fin de que los atletas tengan presente lo que saldrá en la competencia. Al obtener las nuevas estrategias, el atleta puede tener la oportunidad de obtener un mejor resultado el día de la competencia cuando realice nuevamente el WOD con una nueva estrategia. Lo mismo sucede en las clasificatorias, donde se publican varios WODs y el atleta tiene un tiempo determinado (usualmente 1 semana) para realizar la rutina y enviar su puntaje. Al optimizar sus primeros resultados puede tener mayor oportunidad en la clasificatoria.

4.3.1. Puntajes de un WOD

Como se mencionó anteriormente en este documento, en CrossFit existen dos tipos de puntajes posibles una vez realizada un WOD. Estos puntajes dependen del tipo de WOD realizado. Estos son las siguientes:

- **For Time (ó *Por Tiempo*):** es un tipo de WOD que consiste en un conjunto de movimientos, con un número de repeticiones cada uno (pueden ser el mismo número o no), ejecutados uno tras otro una sola vez. El objetivo de este tipo de WODs es realizar el número total de repeticiones en el menor tiempo posible, donde el resultado final es un tiempo en formato mm:ss o el total de segundos. Un ejemplo se puede ver en la Figura 4.5:

Figura 4.5. Ejemplo de un WOD For Time. 1000m en remo, 50 dominadas, 1000m en remo, 100 pechadas, 1000m en remo y 150 sentadillas [7].

- **Rounds For Time (ó *Rondas Por Tiempos*):** es similar a los WODs For Time, lo que lo diferencia del anterior es que la secuencia de movimientos se ejecuta más de una vez. Un ejemplo se puede ver en la Figura 4.6:

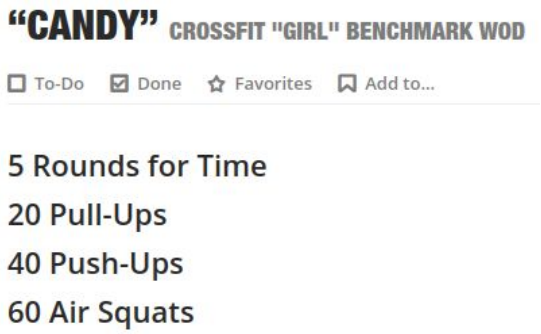


Figura 4.6. Ejemplo de un WOD Rounds For Time. 5 rondas de 20 dominadas, 40 pechadas y 60 sentadillas [8].

- **AMRAP**: que son las siglas de "As Many Repetitions As Possible". Es un tipo de WOD que consiste en una secuencia de movimientos que hay que ejecutar una y otra vez hasta que el tiempo límite se acabe. El resultado final de este tipo de WOD es un número especificando la cantidad total de repeticiones o rondas. Un ejemplo se puede ver en la Figura 4.7:

Figura 4.7. Ejemplo de un WOD AMRAP. En 20 minutos realizar la máxima cantidad de rondas y repeticiones de 15 dominadas, 30 pechadas y 45 sentadillas [9].

Una vez el atleta realiza el WOD y tiene su puntaje en el sistema, este procede a mostrarlo gráficamente. Estos puntajes se representan en un gráfico de línea donde el eje X representa un intervalo de tiempo (en caso de ser un AMRAP), un movimiento (en caso de ser For Time) o una ronda (en caso de ser Rounds For Time), y el eje Y representa la cantidad de repeticiones realizadas en el intervalo de tiempo (para

AMRAPs) o el tiempo en segundos que se tomó hacer un movimiento o ronda (para For Time o Rounds For Time, respectivamente) (Ver Figura 4.3).

4.3.2. Regresiones

Para generar las nuevas estrategias en base al puntaje obtenido de un WOD, se realizan regresiones lineales y polinomiales a partir de la curva original.

Para la generación de estas regresiones se utiliza la librería PHP-ML [10]. Las dos estrategias generadas consisten en una regresión lineal simple utilizando suma de cuadrados, y una regresión polinomial utilizando SVR (Support Vector Regression) [11]. Esto da como resultado dos nuevas curvas formadas a partir de la original, donde cada curva es una nueva estrategia (Ver Figura 4.4).

Capítulo 5

Analizador de Videos de Rutinas de CrossFit

El modelo clasificador resulta ser el núcleo y la parte más crítica de este proyecto de tesis, puesto que es la componente que se encarga de identificar los ejercicios de los videos, clasificarlos y determinar las repeticiones o los tiempos correspondientes.

5.1. Flujo Óptico

Dado que el modelo no consume videos puros para fines de entrenamiento y, por consiguiente, para clasificar los movimientos, es necesario calcular el flujo óptico en los ejes X e Y a cada video.

Sin embargo, calcular el flujo óptico de frames de consecutivos en un video de movimientos que, por su naturaleza, no se ejecutan a tan alta velocidad, de un frame a otro no habría mucha diferencia por lo que el flujo óptico sería casi imperceptible como se puede ver en la Figura 5.1. Por eso, el flujo óptico se decide calcular cada 3er frame con un tamaño de 100 x 100 cada uno. De esta forma se puede ver mejor la diferencia de los pixeles en dos momentos distintos del movimiento (Ver Figura 2.2).

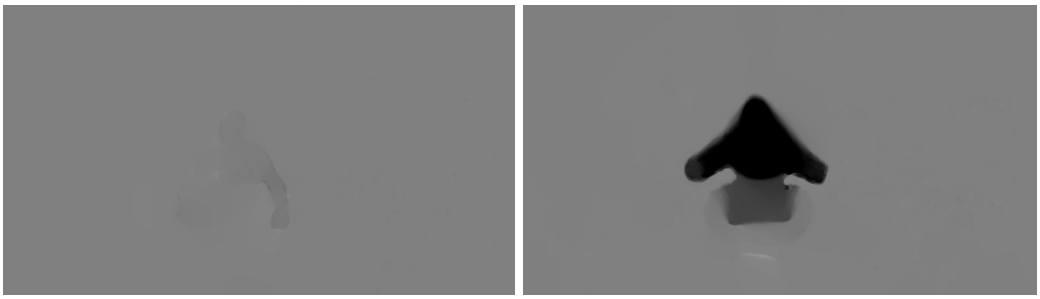


Figura 5.1. Flujo óptico de frames consecutivos (izquierda) y flujo óptico de cada 3er frame (derecha)

5.2. Clasificación de los movimientos

Muchos trabajos e investigaciones se han realizado para resolver el problema de la clasificación de movimientos y acciones humanas dentro de un video [2][14] logrando obtener niveles de *accuracy* de hasta un 88% utilizando el dataset UCF101 [3] para entrenamiento, validación y prueba.

Esta tesis utiliza como línea base la arquitectura del 'Two Stream Convolutional Network' propuesta por Simonyan y Zisserman en el 2014 [14] (Ver Figura 3.1). La otra propuesta del 'Hidden Two Stream Convolutional Network', presenta la ventaja de que esta arquitectura se enfoca en el ahorro de espacio en disco y tiempo de procesamiento del flujo óptico.

Simonyan y Zisserman plantearon una alternativa para el problema de clasificación de acciones humanas utilizando dos redes convolucionales con casi el mismo diseño, pero con una entrada distinta, donde el resultado final sea el promedio de ambas redes.

La idea de esto es poder determinar una acción por medio de la secuencia de pixeles por medio del flujo óptico utilizando el *Temporal Stream,* y el equipo o lugar en dónde se desarrolle dicha acción por medio de los frames RGB del video utilizando el *Spatial Stream*.

Sin embargo, dada la naturaleza del problema que se presenta en esta tesis el *Spatial Stream* resulta innecesario, ya que una sentadilla, una pechada o una dominada se pueden realizar en cualquier lugar y seguirán siendo los mismos ejercicios. También, se puede presentar el caso que realizando un WOD donde haya dominadas con sentadillas o pechadas, al momento de clasificar las sentadillas, por ejemplo, por el hecho de haber una barra de dominadas dentro de los frames RGB, el modelo le diera más probabilidad a ser una dominada cuando no es así.

Por esta razón se decide remover el *Spatial Stream* y trabajar únicamente con el *Temporal Stream*.

Este modelo analiza el video y arroja una secuencia de clasificaciones por cada 12 flujos ópticos, y como cada flujo ha sido calculado cada 3er frame, cada clasificación equivale a 36 frames.

Esta secuencia de clasificaciones se ve de la siguiente forma, donde 1 equivale a sentadillas, 2 equivale a dominadas, 3 equivale a pechadas y 0 es el descanso:

2,3,0,0,0,1,1,1,1,1,1,1,0,1,3,3,3,3,3,3,1,0,0,1,1,1,1,1,1,1,3,3,3,3,3,3,2,0,0,0,0,1,1,1,1,1,1,0,1,3,3, 3,3,0,3,0,3,0,0,3,3

Figura 5.2. Secuencia de clasificaciones dada por el modelo.

En el próximo capítulo se explica en más detalle la arquitectura de la red utilizada para la implementación.

5.3. Algoritmo de conteo de repeticiones

Contar repeticiones es parte esencial para poder analizar los WODs de tipo AMRAP. Al contar repeticiones se puede saber cuantas repeticiones de cierto(s) movimiento(s) hace el atleta en un tiempo específico, de esta forma éste puede saber cada cierto intervalo de tiempo su nivel de consistencia y generar otras estrategias usando el Optimizador de Resultados.

Supongamos un AMRAP de 10 minutos, con el conteo de repeticiones la aplicación puede saber cuantas hace cada 1 minuto o cada 2 minutos y así poder construir una curva en base a estos resultados y poder optimizarla como se ve en la Figura 4.4.

Para el conteo de repeticiones se necesita tener un modelo ya entrenado que sea capaz de clasificar cada uno de los movimientos (sentadillas, pechadas y dominadas) y los descansos.

Una vez obtenida esta secuencia de clasificaciones (ver Figura 5.2) se procede a agrupar subsecuencias de clasificaciones consecutivas, y queda de la siguiente forma:

1 → 2
2 → 3
3 → 0,0,0
4 → 1,1,1,1,1,1,1
5 → 0
6 → 1
7 → 3,3,3,3,3,3
8 → 1
9 → 0,0
10 → 1,1,1,1,1,1,1
11 → 3,3,3,3,3,3
12 → 2
13 → 0,0,0,0
14 → 1,1,1,1,1,1
15 → 0
16 → 1
17 → 3,3,3,3
18 → 0
19 → 3
20 → 0
21 → 3
22 → 0,0
23 → 3,3

Figura 5.3. Subsecuencias de clasificaciones consecutivas.

Ya se sabe que cada clasificación son 36 frames. Luego, se leen los FPS del video para así saber cuantos segundos pasan por cada clasificación.

Suponiendo que el video es de 5 minutos y se quieren saber las repeticiones realizadas cada minuto, se comienza a encontrar un patrón específico (ver Figura 5.5) en los flujos ópticos dentro de cada subsecuencia consecutiva que no contenga 0 (descanso) y que no tenga una sola clasificación (ya que se asume que pudiera ser un error). Para esto se usa la suma de pixeles de cada flujo. En el ejemplo sólo se busca dicho patrón para contar cuántas repeticiones se hacen en el movimiento correspondiente (ver Figura 5.4):

4 → 1,1,1,1,1,1,1
7 → 3,3,3,3,3,3
10 → 1,1,1,1,1,1,1
11 → 3,3,3,3,3,3
14 → 1,1,1,1,1,1
17 → 3,3,3,3
23 → 3,3

Figura 5.4. Subsecuencias de 2 o más valores diferentes de 0.

Las subsecuencias de un solo elemento, y las que contienen 0, sólo se usan para saber en qué momento se acaba un intervalo de tiempo definido y así saber cuándo pasar al próximo intervalo hasta recorrer por el video completo.

Figura 5.5. Patrón de movimiento para contar una repetición. Los pixeles blancos son valores más cercanos a 1 que son cuando la persona está bajando, y los negros son valores cercanos a 0 que son cuando la persona está subiendo.

El algoritmo se puede resumir de la siguiente manera:

* CalcularTiemposPorClasificacion(): dado los FPS del video, y utilizando la constante de que 1 clasificación son 36 *frames*, determina cuantos segundos pasan por cada clasificación.

* TienePatronDeRepeticion(): dado un arreglo de subsecuencias consecutivas y la suma de pixeles de cada *frame* de cada subsecuencia, se verifica si existe o no un patrón de repetición comparando la suma de pixeles de la primera clasificación con la segunda. Si hay al menos 1 repetición retorna *true*, en caso contrario *false.*

* HayPatron(): dada la suma de pixeles de los flujos de dos clasificaciones, verifica si existe o no el patrón de movimiento. Si existe retorna *true*, y *false* en caso contrario.

* Sum(): retorna la suma de valores de un arreglo.

```
segundosPorIntervalo ← 0
segundosRecorridos ← 0
repeticiones ← []
repeticionesPorIntervalo ← []

for cada subsecuencia, i ← 0 do
  for cada clasificacion en subsecuencia[i], j ← 0 do
    if segundosRecorridos ≥ limiteDeSegundosDeRutina then
      salir del ciclo actual
    end if

    segundosPorIntervalo += CalcularTiempoPorClasificacion(FPS)
    segundosRecorridos += CalcularTiempoPorClasificacion(FPS)

    if TienePatronDeRepeticion (subsecuencia[i], pixeles [i]) then
      if j es par and j != 0 then
        if HayPatron (pixeles [i][j], pixeles [i][j - 1]) then
          repeticiones.push(1)
        end if
      end if
    else
      repeticiones.push(0)
    end if

    if segundosPorIntervalo ≥ tiempoPorIntervalo then
      segundosPorIntervalo ← 0
      repeticionesPorIntervalo.push(Sum(repeticiones))
      repeticiones ← []
    end if
    j += 1
  end for

  if segundosRecorridos ≥ limiteDeSegundosDeRutina then
    salir de ciclo actual
  end if
i += 1
end for
return repeticionesPorIntervalo
```

Este algoritmo retorna un arreglo con las repeticiones realizadas por el intervalo establecido.

Aplicando este algoritmo se puede analizar un video de un AMRAP y así saber la cantidad de repeticiones que lleva el atleta en un intervalo específico de tiempo.

5.4. Algoritmo de cálculo de tiempos

Así como el conteo de repeticiones es importante para analizar AMRAPs, el cálculo de tiempos por ronda y/o movimiento es igual de importante para poder analizar WODs For Time o Rounds For Time.

Suponiendo que se tiene un WOD de 5 rondas de X cantidad de sentadillas e Y cantidad de pechadas para realizarlo en el menor tiempo posible, al calcular el tiempo que se demora el atleta por ronda o por movimiento dentro de cada ronda se puede optimizar su resultado para crear las nuevas estrategias.

Al igual que el conteo de repeticiones, para calcular tiempos primero se obtiene una secuencia de clasificaciones del modelo de clasificación. Se realiza el mismo proceso de separar en subsecuencias de clasificaciones consecutivas.
Luego para cada subsecuencia se calcula el tiempo que se toma cada una de ellas, incluyendo los descansos y las subsecuencias de una sola clasificación. Como ya se saben los FPS del video y que cada clasificación son 36 frames, se determina el tiempo que se toma cada subsecuencia y se va comparando que se hayan contado cada movimiento de cada ronda tomando en cuenta cómo está estructurado el WOD.

El algoritmo se puede resumir de la siguiente manera:

* CalcularTiemposPorClasificacion(): dado los FPS del video, y utilizando la constante de que 1 clasificación son 36 *frames*, determina cuantos segundos pasan por cada clasificación.

* ContieneMovimiento(): dado un arreglo de movimientos correspondientes a una rutina, verifica si una clasificación dada por el modelo esta dentro de ese arreglo de movimientos de la rutina. Si está retorna *true*, y *false* en caso contrario.

```
rondaActual ← 0
movimientoActual ← 0
tiempo ← []
repeticionesPorIntervalo ← []

for cada subsecuencia, i ← 0 do
  for cada clasificacion en subsecuencia[i], j ← 0 do
    if subsecuencia[i].length) > 1 then
      if clasificacion[j] == movs[movActual] or clasificacion[j] == 0
      then
        tiempo[rondaActual] += CalcularTiempoPorClasificacion(FPS)
      else
        if not ContieneMovimiento(clasificacion[j], movs) then
          tiempo[rondaActual] += CalcularTiempoPorClasificacion(FPS)
        else
          movimientoActual += 1
          if movimientoActual > movimientosPorRonda then
            movimientoActual ← 0
            rondaActual += 1
          end if
        end if
      end if
    else
      tiempo[rondaActual] += CalcularTiempoPorClasificacion(FPS)
    end if
    j += 1
  end for
  i += 1
end for
return tiempo
```

Este algoritmo retorna un arreglo de tiempos, en segundos, de la duración de cada ronda en la rutina.

Al aplicar este algoritmo se puede saber el tiempo que el atleta se demoró por ronda, o por movimiento de así necesitarlo.

Capítulo 6

Implementación

Para la implementación de la solución se deben tomar en cuenta la realización de diversas tareas. Estas tareas consisten en crear un dataset de videos de los ejercicios a determinar, entrenar y validar las redes convolucionales y aplicar los algoritmos de conteo de repeticiones o de cálculo de tiempos, según corresponda.

6.1. Recopilación de videos

Por medio de distintos datasets públicos en el internet [3][12][13] se llegó a recopilar alrededor de 200 videos por ejercicio (siendo un total de casi 600) dónde la duración de los videos era variada y sus FPS (*frames por segundo)* están entre 24 y 30. El video de menor duración fue de 9 segundos, mientras que el de mayor duración fue de poco más de 1 minuto. Pero en todos los videos se ejecutaba de manera repetitiva el ejercicio correspondiente (2 o más repeticiones). A este conjunto de datos lo llamaremos **Conjunto 1**.

Una vez obtenidos se procede a crear lo que llamaremos **Conjunto 2**, que consiste en aplicar *data augmentation* (Ver Figura 6.1) al Conjunto 1 aplicando ciertos filtros a cada video con el fin de generar más información similar pero, estrictamente, diferente. Estos filtros aplicados son rotación de 5 a 10 grados (de manera aleatoria) hacia la izquierda o derecha, un giro de 180 grados en torno al eje Y, y disminución/aumento de brillo. Estos filtros, y la combinación de ellos se aplican de manera aleatoria a cada video del Conjunto 1. Esto da como resultado al Conjunto 2 con alrededor de 4000 a 5000 videos.

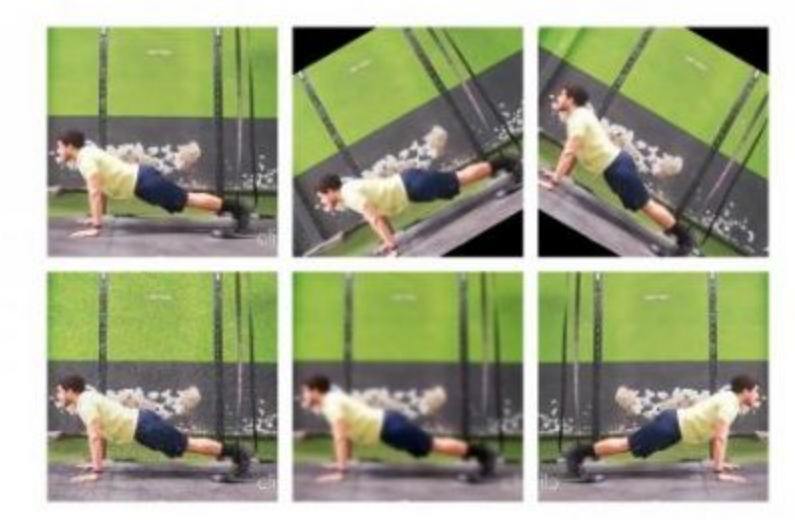


Figura 6.1. Filtros de data augmentation a una imagen.

Ya obtenidos el Conjunto 1 y Conjunto 2, se procede a calcular los flujos ópticos como se explica en la sección 5.1.

6.2. Entrenamiento y validación del modelo

Una vez calculados los flujos ópticos del Conjunto 1 y el Conjunto 2 se procede a entrenar el modelo que se encargará de identificar los movimientos que se ejecutan en un video.

6.2.1. Ejemplos para el entrenamiento

Al momento de entrenar una red neuronal mientras más datos hayan mejor podrán ser los resultados, ya que la red podrá analizar más y más ejemplos con el fin de que su clasificación sea lo más general posible y que abarque la mayoría de los casos.

Para esto se toma el flujo óptico ya calculado de cada video del Conjunto 1 y Conjunto 2 y se secciona en varios subconjuntos formados por 12 flujos del eje X y 12 flujos del eje Y. Estos subconjuntos son consecutivos dentro del total de flujos ópticos de un video, es decir:

1 video → 48 flujos == 24 flujos X y 24 flujos Y → 2 subconjuntos de 12 flujos X y 12 flujos Y cada uno == 2 ejemplos.

Una vez aplicado este proceso se obtiene alrededor de 6.000 ejemplos provenientes del Conjunto 1, a los que llamaremos **E1**, y 27.000 provenientes del Conjunto 2, a los que llamaremos **E2**.

6.2.2. Clases

Las clases en los modelos de clasificación equivale a cada posible resultado que puede resultar ser un ejemplo que el modelo esté analizando. En primera instancia se plantean 3 clases equivalentes a la cantidad de ejercicios propuestos en la tesis a determinar: sentadilla, pechada y dominada.

Sin embargo, para poder separar en un video de un WOD cuando una persona esté ejecutando un movimiento y tomándose unos segundos de descanso, es necesario incluir una cuarta clase: descanso (Ver Figura 6.2).

Figura 6.2. Atleta descansando durante un WOD.

Para esta clase se decide grabar varios videos de larga duración en diferentes entornos y con diferentes personas haciendo cualquier cosa excepto algún movimiento o ejercicio. Simplemente con caminar, acostarse de una pared o del piso, y deambular en general basta.

Una vez agregada esta nueva clase y estos nuevos videos, y ejecutando el proceso explicado anteriormente de calcular el flujo óptico y separar en subconjuntos de entrenamiento, E1 contiene ahora cerca de 8.000 ejemplos y E2 alrededor de 32.000 ejemplos.

6.2.3. Redes v1 y v2

En esta tesis el entrenamiento del modelo se realiza en una laptop con las siguientes características técnicas:

- CPU: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 4 nucleos, x86_64
- GPU: GeForce GTX 1060, 6 GB
- RAM: 8 GB
- Disco: SSD 256 GB

Debido a esto la cantidad de parámetros utilizados se decide disminuir para que la red pueda ser entrenada, porque con los parámetros originales, utilizando el equipo mencionado anteriormente, no es posible. La red resultante, a la que llamaremos Red v1, se muestra en la Figura 6.3.

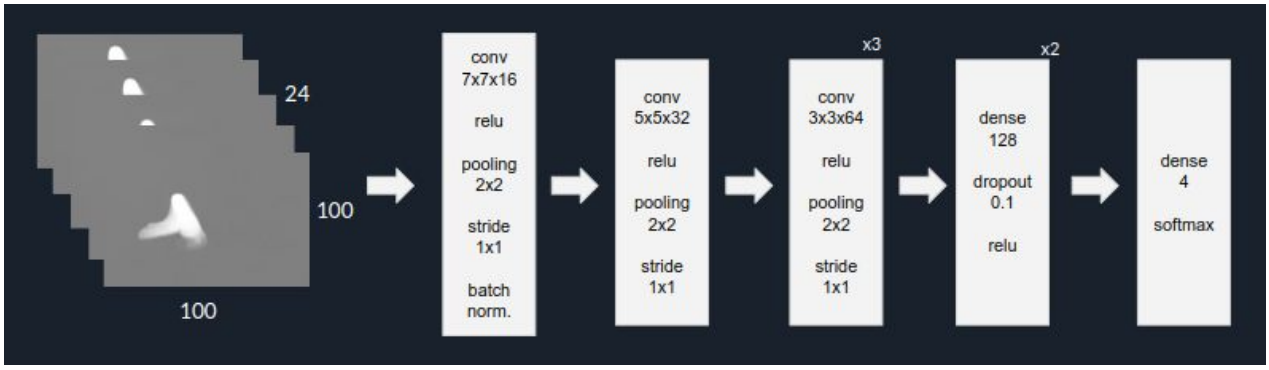


Figura 6.3. Arquitectura Red v1.

Luego, se decide aumentar el número de capas convolucionales con el fin de hacer la red más profunda, realizando más convoluciones. A esta red la llamaremos Red v2, como se puede ver en la Figura 6.4.

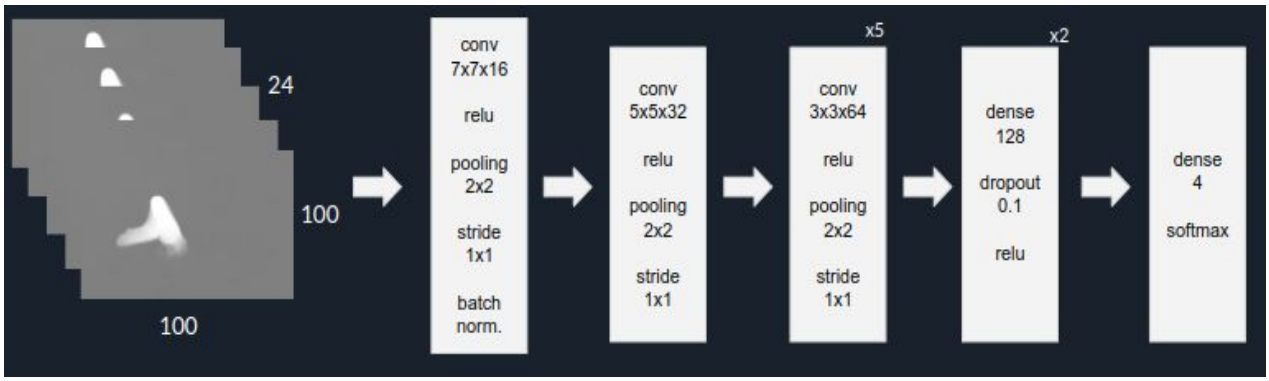


Figura 6.4. Arquitectura Red v2.

A simple vista se puede ver que la mayor diferencia entre la Red v2 y la Red v1, que equivale al *Temporal Stream Convolutional Network* de Simonyan y Zisserman [14], pero con menos parámetros debido al hardware disponible, es la profundidad de la red. Sin embargo, a continuación se muestra cómo estos cambios, más el preprocesamiento empleado de los datos tiene un impacto en las clasificaciones.

6.2.4. Proceso de entrenamiento y validación

Al momento de entrenar y validar la Red v1 y Red v2 se decide, antes que todo, separar los datos en un 80% y un 20%, de manera aleatoria, donde el 20% de los datos serán para fines de prueba y las redes nunca verán estos datos durante el proceso de entrenamiento y validación. Cabe destacar que los sets de entrenamiento y validación **sí** tienen las mismas personas involucradas en los videos, pero los videos resultan ser distintos (iluminación, ángulo, duración, etc.). Esta separación de los datos de entrenamiento y de prueba se realiza tanto para E1 como para E2.

Con el 80% de los datos tanto de E1 como de E2 se procede a entrenar la Red v1 y la Red v2 dando como resultado 4 modelos diferentes. Para cada uno de estos modelos se aplica *cross-validation* de un 80 - 20 con el fin de separar los datos de entrenamiento en entrenamiento y validación de manera aleatoria. Los resultados obtenidos fueron los siguientes:

Modelo	Accuracy
Red v1 entrenada con E1 (Modelo 1)	85%
Red v2 entrenada con E1 (Modelo 2)	88%
Red v1 entrenada con E2 (Modelo 3)	98%
Red v2 entrenada con E2 (Modelo 4)	99%

Tabla 6.1. Resultado de los modelos creados.

Estos resultados corresponden al proceso de validación durante el entrenamiento de los modelos. Cada modelo se entrenó por 200 épocas, con un batch de 30 ejemplos por iteración, *Stochastic Gradient Descent* como función optimizadora y *Categorical Cross Entropy* como función de pérdida.

Cabe destacar que al entrenar los modelos con E1 se demoran entre 4 y 6 horas con el hardware especificado anteriormente, mientras que entrenar dichas redes con E2, y el mismo hardware, se tarda entre 10 y 12 horas cada modelo.

6.3. Testing de los modelos

Una vez se tienen entrenados los modelos es necesario verificar cuál de todos será el que pueda hacer mejores clasificaciones en ámbitos más generales. Los resultados anteriores de validación es durante el entrenamiento y son datos que el modelo ha analizado en las diferentes épocas, o iteraciones, que realiza al momento de ser entrenada.

Para asegurar de que se escoja el mejor modelo para ser puesto a producción con diferentes usuarios y en diferentes lugares es necesario probar qué tan bien clasifican las redes con datos que no hayan visto antes. Aquí es donde entra el 20% de datos que se separaron del resto antes de entrenar las redes.

Cada modelo resultante del entrenamiento mencionado anteriormente analizó el 20% de los datos de E1 y E2 destinados para fines de testing preliminar.

Los resultados después de realizar este testing fueron los siguientes:

Modelo	Accuracy
Modelo 1	73%
Modelo 2	74%
Modelo 3	79%
Modelo 4	**82%**

Tabla 6.2. Resultado del testing a E1 de los modelos creados.

Modelo	Accuracy
Modelo 1	83%
Modelo 2	86%
Modelo 3	94%

Modelo 4	**95%**

Tabla 6.3. Resultado del testing a E2 de los modelos creados.

En base a estos resultados se puede ver como la cantidad de datos, aún siendo creados por data augmentation, en adición a mayor profundidad de capas convolucionales a la red aumenta el *accuracy* de un modelo para reconocimiento de acciones humanas.

En la Figura 6.5 se puede ver la matriz de confusión del Modelo 4 en base a estos datos de testing de E2.

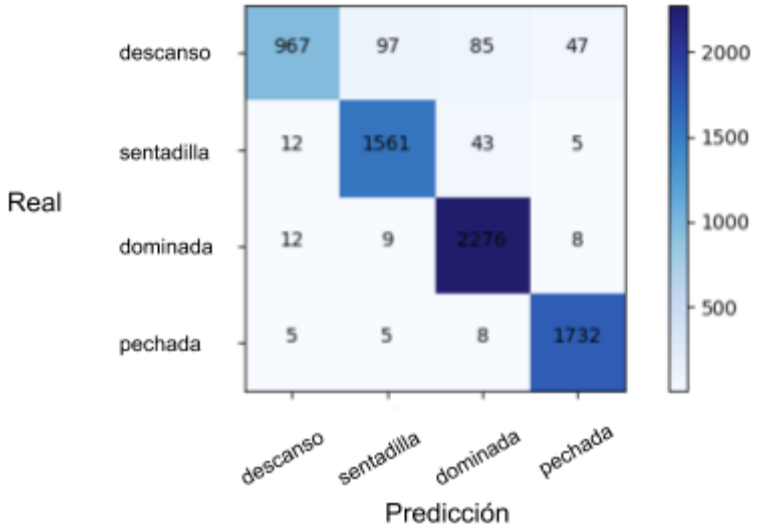


Figura 6.5. Matriz de confusión Modelo 4. **Accuracy de 95,1%**, recall de 93,7% y precisión de 95,1%.

Una vez obtenido el modelo final con el cual se identificarán los movimientos dentro de un video es necesario poder contar las repeticiones de cada movimiento en caso de analizar un AMRAP (máximas repeticiones en un tiempo específico), y el tiempo en que se realiza cada movimiento o rondas en caso de analizar un For Time o Rounds For Time (número específico de rondas y repeticiones en el menor tiempo posible).

Capítulo 7

Evaluación y resultados

Hasta ahora ya se ha visto cómo el sistema es capaz de clasificar los movimientos en videos, de contar repeticiones y de calcular tiempos. Ahora es necesario evaluar con diferentes usuarios para verificar el error estándar al momento de analizar un AMRAP y de analizar un *For Time* que contenga los movimientos preestablecidos para fines de esta tesis. Esto es lo que se verá en este capítulo.

Para estas evaluaciones se decide utilizar los Modelos 3 y 4 ya que estos fueron los que demostraron tener los mejores resultados en la fase de testing de todos los modelos, mencionado anteriormente y, además, fueron entrenados con los mismos datos de los Modelos 1 y 2.

7.1. Preparación de la evaluación

Para evaluar el funcionamiento del sistema se toman 4 hombres y 3 mujeres donde realizan un *AMRAP* y un *For Time* (los hombres hacen ambos WODs, mientras que las mujeres solo hacen el *For Time* debido al nivel de dificultad de las dominadas estrictas) para verificar que el sistema pueda analizar ambas rutinas con personas que realicen cada ejercicio con un estilo y velocidad diferente. La razón por la cual se toma esta cantidad de personas es por que en el entorno donde se pudo hacer el experimento, solo esas personas estaban dispuestas a ser grabadas en video y ayudar con la evaluación.

El *AMRAP* consiste en, durante un minuto, hacer la mayor cantidad de rondas posibles de 3 sentadillas y 3 pechadas, mientras que el *For Time* es hacer 5 dominadas, 10 pechadas y 15 sentadillas en el menor tiempo posible.

Cada persona hizo cada WOD en el gimnasio Frecuencia CrossFit desde una perspectiva específica (Ver Figura 7.1), siendo un total de 22 pruebas realizadas. Esta

cantidad se consideró suficiente ya que el número de ejercicios que se fijó en la tesis a evaluar fueron 3, y el orden de los mismos es indiferente ya que son analizados por separado y luego sumados para dar el total de tiempo, o repeticiones. A continuación se presentan los resultados obtenidos de cada prueba para el *AMRAP* y para el *For Time*.

Figura 7.1. Perspectiva utilizada para analizar los videos.

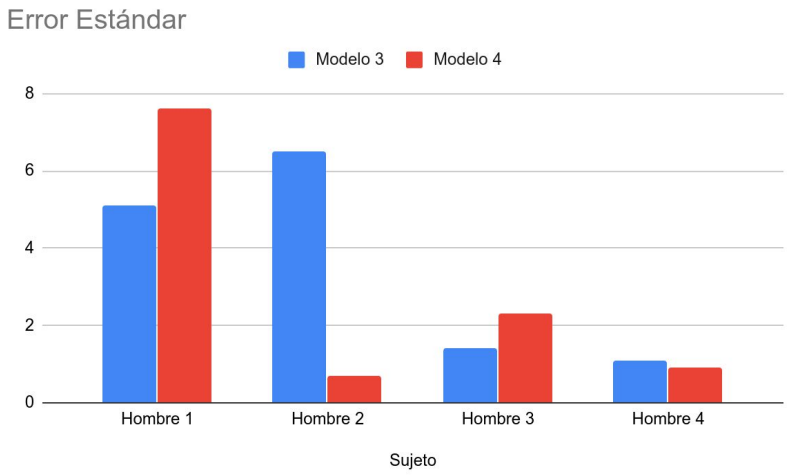
7.2. Resultados

Como se menciona anteriormente, el análisis tanto del *AMRAP* como del *For Time* para las evaluaciones se hacen con los Modelos 3 y 4. A continuación, se presentan los errores estándares de cada evaluación:

Sujeto	Modelo	Error estándar
Hombre 1	**Modelo 3**	**± 5,1 segs**
Hombre 1	Modelo 4	± 7,6 segs
Hombre 2	Modelo 3	± 6,52 segs
Hombre 2	**Modelo 4**	**± 0,7 segs**
Hombre 3	**Modelo 3**	**± 1,4 segs**
Hombre 3	Modelo 4	± 2,3 segs

Hombre 4	Modelo 3	± 1,1 segs
Hombre 4	**Modelo 4**	**± 0,9 segs**

Tabla 7.1. Errores estándares de los sujetos de prueba en base a sus resultados reales en rutinas For Time.

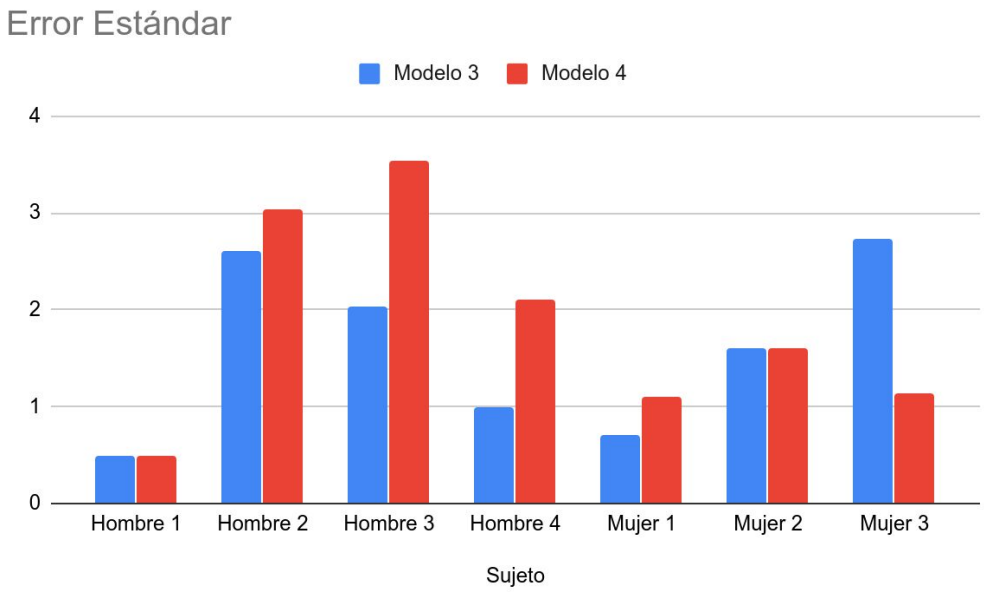


Gráfica 7.1. Comparación entre los Modelos 3 y 4 para la rutina For Time.

Sujeto	Modelo	Error estándar
Hombre 1	**Modelo 3**	**± 0,5 reps**
Hombre 1	**Modelo 4**	**± 0,5 reps**
Hombre 2	**Modelo 3**	**± 2,6 reps**
Hombre 2	Modelo 4	± 3,04 reps
Hombre 3	**Modelo 3**	**± 2,04 reps**
Hombre 3	Modelo 4	± 3,54 reps
Hombre 4	**Modelo 3**	**± 1,0 reps**
Hombre 4	Modelo 4	± 2,1 reps
Mujer 1	**Modelo 3**	**± 0,71 reps**
Mujer 1	Modelo 4	± 1,1 reps
Mujer 2	**Modelo 3**	**± 1,6 reps**
Mujer 2	**Modelo 4**	**± 1,6 reps**

Mujer 3	Modelo 3	± 2,74 reps
Mujer 3	**Modelo 4**	**± 1,14 reps**

Tabla 7.2. Errores estándares de los sujetos de prueba en base a sus resultados reales en rutinas AMRAP.



Gráfica 7.2. Comparación entre los Modelos 3 y 4 para la rutina AMRAP.

En base a estos resultados se pueden sacar las siguientes conclusiones:

- Los algoritmos de conteo de repeticiones y cálculo de tiempos son susceptibles al ruido. A pesar de tener robustez para manejar los errores que arroja el modelo, no es suficiente como para reducir a 0 el error estándar.

- El algoritmo de conteo de repeticiones resultó ser un poco más que el de cálculo de tiempos. Su error promedio fue de un **8%**, mientras que el algoritmo del cálculo de tiempos dio un promedio de **8,7%** de error estándar. Este error muestra que tanto difiere el resultado final real, en comparación con el resultado final obtenido por la solución.

- En general, el modelo 3 resulta ser más preciso que el modelo 4, salvo excepciones.

Capítulo 8

Conclusiones

El objetivo general de esta tesis consistió en poder analizar rutinas de CrossFit por medio de videos con el fin de poder capturar el rendimiento del atleta en todo momento durante la rutina y con esta información poder generar nuevas estrategias de cómo mejorar dicha rutina para fines de futuros entrenamientos y/o competencias.

Para lograr este objetivo general se fijaron 3 objetivos específicos: no más de 15% de error al momento de clasificar una sentadilla, no más de 15% de error al momento de clasificar una pechada y no más de 15% de error al momento de clasificar una dominada.

8.1. Logros

La generación de nuevas estrategias resultó ser un éxito al ser probadas en un ambiente real en una competencia de global de CrossFit donde 4 de 5 WODs se rehicieron utilizando las estrategias generadas por el sistema (Ver Capítulo 4), dando como resultado mejores puntajes en los 4 casos.

En el caso de la captura de los puntajes por medio de análisis de un video, los modelos clasificadores resultaron tener un porcentaje de precisión mejor de lo esperado.

Cada movimiento tuvo un porcentaje de error menor al 15%, el cual fue de un **4,9%** (como se puede ver en la Figura 6.5) por lo que los objetivos específicos fueron cumplidos exitosamente.

Del lado de los algoritmos de conteo de repeticiones y cálculo de tiempos tuvieron un error de **8%** y **8,7%**, respectivamente.

En este proyecto se puede ver cómo generar más datos por medio de *data augmentation*, influye positivamente en la clasificación de movimientos humanos en videos. Esto, en adición a una mayor profundidad en capas convolucionales influye positivamente, de igual forma, a la clasificación. Aunque en los resultados de las evaluaciones con múltiples usuarios se puede ver como el Modelo 3 (entrenado con *data augmentation*, pero menos capas convolucionales) resulta tener menos error sobre el rendimiento real de los atletas en los videos, que el Modelo 4 (entrenado con *data augmentation* y con más capas convolucionales) que resultó tener mejor precisión al clasificar el 20% de los datos nunca vistos por los modelos durante la etapa de testing.

Sin embargo, esto no es suficiente para decir que el Modelo 3 resulta ser una mejor opción. Se necesitaría probar con más usuarios, haciendo diferentes rutinas, en diferentes entornos, con diferentes ángulos de cámara, para determinar realmente, cual de los dos modelos resulta ser mejor para casos más generales.

8.2. Trabajo Futuro

Este proyecto, a pesar de haber sido un éxito y haber obtenido resultados mejores de lo esperado, aún puede ser mejorado en muchos aspectos.

El análisis de un video de 1 minuto se toma alrededor de 7 minutos, de los cuales 6 minutos y medio es solo para el cálculo del flujo óptico. Con un mejor hardware (GPU principalmente) este proceso puede ser mucho más rápido para ser usado en ambiente de producción. También utilizando un enfoque como en el *Hidden Two-Stream Network* [2] donde el flujo óptico se calcula y se deja en memoria, en lugar de escribir y leer los flujos del disco, ya sea utilizando otra red convolucional o no, esto ayudaría a que el proceso completo de analizar un video sea más rápido.

Otra posible mejora que se puede hacer es enfocarse en darle más robustez a los algoritmos de conteo de repeticiones y cálculo de tiempos, con el fin de que puedan manejar arrojar mejores resultados aun cuando el modelo arroje clasificaciones erróneas. Utilizando el *score* (el nivel de certeza que la red clasifica un movimiento) de cada clasificación, se pueden hacer correcciones a dicha clasificación en base a las clasificaciones cercanas en la secuencia (ver Figura 5.2). De esta forma al momento de ejecutar el algoritmo como está ahora el error estándar puede ser menor.

Y finalmente, con el fin de ampliar la gama de posibles rutinas para analizar sería necesario poder agregar más movimientos comunes en el CrossFit como son los de

levantamiento de pesas y más movimientos que involucren equipos como mancuernas (ver Figura 8.1), kettlebells (ver Figura 8.2) y barras olímpicas (ver Figura 8.3).

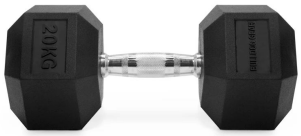


Figura 8.1. Mancuerna

Figura 8.2. Kettlebell

Figura 8.3. Barra olímpica

Bibliografía

[1] https://map.crossfit.com/ (última visita: 28/11/2018)

[2] Zhu Y., Lan Z., Newsam S., Hauptmann A.: Hidden Two-Stream Convolutional Networks for Action Recognition (https://arxiv.org/pdf/1704.00389.pdf) (última visita: 28/11/2018)

[3] Soomro K., Roshan Zamir A., Shah M.: UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild (http://crcv.ucf.edu/data/UCF101.php) (última visita: 28/11/2018)

[4] https://definicion.de/sentadilla/ (última visita 13/08/2019)

[5] https://es.wikipedia.org/wiki/Flexi%C3%B3n_de_codos (última visita 13/08/2019)

[6] https://es.wikipedia.org/wiki/Dominada (última visita 13/08/2019)

[7] https://www.crossfit.com/190416 (última visita 20/08/2019)

[8] https://wodwell.com/wod/candy/ (última visita 20/08/2019)

[9] https://www.crossfit.com/190115 (última visita 20/08/2019)

[10] https://php-ml.readthedocs.io/en/latest/ (última visita 28/08/19)

[11] Martín Guareño, J.: Support Vector Regression: Propiedades y Aplicaciones. (https://idus.us.es/xmlui/bitstream/handle/11441/43808/Mart%C3%ADn%20Guare%C3%B1o%2C%20Juan%20Jos%C3%A9%20TFG.pdf?sequence=1&isAllowed=y) (última visita 28/08/2019)

[12] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: A Large Video Database for Human Motion Recognition. ICCV, 2011. (http://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/#dataset) (última visita: 07/07/2019)

[13] C. Chen, R. Jafari, and N. Kehtarnavaz, "UTD-MHAD: A Multimodal Dataset for Human Action Recognition Utilizing a Depth Camera and a Wearable Inertial Sensor", *Proceedings of IEEE International Conference on Image Processing*, Canada, September 2015. (http://www.utdallas.edu/~kehtar/UTD-MHAD.html) (ultima visita: 07/07/2019)

[14] Simonyan K., Zisserman A.: Two-Stream Convolutional Networks for Action Recognition in Videos. (https://arxiv.org/pdf/1406.2199.pdf) (última visita: 28/11/2018)

[15] Lee J., Bovik A.: The Essential Guide to Video Processing. 2009

[16] Richardson I. H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia. 2003

[17] Leo M., Farinella G.: Computer Vision for Assistive Healthcare. 2018

[18] Lucas B. D.: Image Matching by the Method of Differences. Carnegie Mellon University, 1984.

[19] Lucas B. D., Kanade T.: An iterative image registration technique with an application to stereo vision. 1981.

[20] Farneback G.: Two-Frame Motion Estimation Based on Polynomial Expansion. 2003.

[21] Farneback G.: Polynomial Expansion for Orientation and Motion Estimation. 2002.

[22] Patterson J., Gibson A.: Deep Learning: A Practitioner's Approach. 2017.

[23] Chollet F.: Deep Learning with Python. 2018.

[24] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html

[25] http://www.image-net.org

[26] http://image-net.org/challenges/LSVRC/2012

[27] Srivastava N. et al.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. (https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf) (última visita: 26/10/2020)