Universidad Zaragoza
1542

Trabajo Fin de Grado

Predicción de saliencia audiovisual en contenido 360º

Audiovisual saliency prediction in 360º content

Autor

Santiago Jiménez Navarro

Directora

Ana Serrano Pacheu

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2022

AGRADECIMIENTOS

Me gustaría agradecer primeramente a Dani y Ana, por su continua supervisión y sus valiosos consejos y enseñanzas a lo largo de todos estos meses, los cuales me han servido no solo para este trabajo, sino también para los que tenga que hacer en un futuro.

También me gustaría agradecer a los integrantes del grupo *Graphics & Imaging lab*, la acogida desde el primer día y su voluntad por ayudar en todo lo posible al resto de miembros.

Además, me gustaría agradecer a mi familia y amigos por haberme acompañado durante este camino.

Por último, me gustaría agradecer a todos los profesores los cuales me han formado durante el grado.

Predicción de saliencia audiovisual en contenido 360º

RESUMEN

Como consecuencia de la disminución de los costes y de las mejoras tecnológicas, el campo de la Realidad Virtual (RV) está experimentando un crecimiento en los últimos años. Debido a dicho crecimiento, cada vez está apareciendo más contenido y aplicaciones desarrolladas específicamente para ser reproducidas o utilizadas en entornos de RV. Esto engloba desde contenido audiovisual en 360º, hasta aplicaciones que aprovechan este nuevo paradigma para ofrecer experiencias más inmersivas que las que se podían ofrecer con la anterior tecnología.

Esto, a su vez, abre diversos campos de estudio centrados en tratar de comprender cómo se comportan los usuarios dentro de un entorno de RV, con el objetivo de ofrecer cada vez experiencias más satisfactorias y realistas.

Uno de estos campos de estudio es el tratar de predecir en qué zonas de la escena 360º va a centrar el usuario su atención. Para lograr este objetivo, se han estudiado multitud de alternativas, las cuales incluyen desde diferentes maneras de representar internamente la escena, a utilizar diversos elementos de la misma.

En este trabajo, se han analizado diferentes sistemas de predicción de la atención del usuario, los cuales se diferencian por usar tanto la información visual de la escena como la sonora espacial de la misma. El objetivo principal de este trabajo de fin de grado es el estudio de diferentes modificaciones que se les pueden aplicar a estos modelos, evaluando qué impacto tienen sobre los resultados finales.

Para conseguir este objetivo, lo primero que se ha hecho ha sido estudiar el estado del arte de la materia, además de las herramientas que se van a utilizar a lo largo del trabajo. Después se han estudiado algunos trabajos que se han tomado como referencia, aplicándoles diferentes modificaciones. Por último, se han evaluado los diferentes resultados obtenidos y se han expuesto las conclusiones sacadas.

Audiovisual saliency prediction in 360º content

ABSTRACT

Regarding the latest improvements and the drop of the costs in the discipline of Virtual Reality (VR), this field has been experiencing a growth in terms of available content and developed applications focused in VR during the past years. This increase ranges from audiovisual content in 360º to applications that take advantage of this technology with a view to offer more immersive experiences.

As a result, this enables different study areas that focus on attempting to understand how do users behave in a VR environment, in order to provide better and more realistic experiences.

One of these fields of study tries to predict in which areas of the 360º scene is the user going to drive its attention. With this objective, multiple variants have been studied which include from different ways of storing and representing the scene, to using different features of it.

In this work, multiple attention predicting systems will be studied, characterized by the use of both visual and audio spatial information of the scene. The main goal is the analysis of the possible modifications that can be applied to those systems, evaluating their effect in the resulting predictions.

To this end, the first task has been studying the state-of-the-art as well as the tools that will be used along all the project. Then, some specific works have been studied in more depth, modifying them in specific cases. And finally, those modifications have been evaluated in order to draw the conclusions.

Índice

Lista de Acrónimos

RV	Realidad Virtual
VR	*Virtual Reality* (Realidad virtual)
HMD	*Head Mounted Display* (Pantalla montada en la cabeza)
ODV	*Omnidirectional Video* (Vídeo omnidireccional)
ERP	*Equirectangular Projection* (Proyecciones equirrectangulares)
ECB	*Equator Center Bias* (Sesgo de atención al ecuador)
GPU	*Graphics Processing Unit* (Unidad de procesamiento gráfico)
CPU	*Central Processing Unit* (Unidad central de procesamiento)
MSE	*Mean Squared Error* (Error cuadrático medio)
CE	*Cross Entropy loss* (entropía cruzada)
ReLU	*Rectified Linear Unit* (Rectificador lineal)
NN	*Neural Network* (Red neuronal artificial)
CNN	*Convolutional Neural Network* (Red neuronal convolucional)
GAN	*Generative Adversarial Network* (Red generativa antagónica)
FC	*Fully Connected* (Completamente conectada)
RGB	*Red Green Blue* (Rojo Verde Azul)
ROI	*Region of Interest* (Área de interés)
AEM	*Audio Energy Map* (Mapa de energía del audio)
FPS	*Frames per Second* (Fotogramas por segundo)
ResNet	*Residual Neural Network* (Red neuronal residual)
CP	*Cube Padding* (Rellenado cúbico)
BCE	*Binary Cross-Entropy* (Entropía binaria cruzada)
CC	*Pearson Correlation Coefficient* (Coeficiente de correlación de Pearson)
KLDIV	*Kullback-Leibler Divergence* (Divergencia de Kullback-Leibler)
NSS	*Normalized Scanpath Saliency* (Saliencia de *scanpaths* normalizada)
lr	*Learning Rate* (Factor de aprendizaje)
MSE	*Mean Squared Error* (Error cuadrático medio)

Capítulo 1

Introducción

En los últimos años, el uso de dispositivos de visualización de realidad virtual (por sus siglas en inglés HMD, *Head Mounted Displays*) ha experimentado un gran crecimiento. Se valora que el mercado global de HMD tenía un valor de 12.11 billones americanos de dólares en el año 2020, y se estima[1] que esa cifra alcance el valor de 141.18 billones en el 2028.

Los últimos avances en esta tecnología, junto con su abaratamiento, han hecho posible que su uso ya no se limite únicamente a laboratorios de investigación y empresas especializadas, sino que cada vez se utilice en ámbitos más diversos.

Estos incluyen desde usuarios casuales, los cuales pueden utilizar los HMD para, entre otras cosas, jugar videojuegos y consumir contenido audiovisual inmersivo, como por ejemplo vídeos omnidireccionales (en inglés Omnidirectional Video, ODV), hasta usuarios de sectores más profesionales como la sanidad o la defensa, donde su aplicabilidad se extiende hasta simuladores de vuelo o de operaciones estratégicas.

No obstante, el contenido audiovisual inmersivo difiere del contenido 2D traidcional. Para poder capturar y reproducir sonido envolvente, se necesitan micrófonos omnidireccionales capaces de capturar una señal direccional, la cual describe el sonido en cada punto de una escena 3D en cada instante temporal. La captura del vídeo también precisa de cámaras espaciales que capturen las escenas de 360º.

Un vídeo en 360º contiene toda la información de la escena que rodea la cámara espacial, no sólo la información que está en frente del único objetivo que tienen las cámaras convencionales. A la hora de reproducir estos vídeos en 360º, los usuarios pueden cambiar la orientación de la cámara como deseen, simulando el movimiento que se haría con la cabeza para explorar el entorno desde diferentes ángulos.

Entonces, los vídeos en 360º generados son unas representaciones esféricas de la escena, pero para almacenarlos hay que aplicarles ciertas transformaciones para

[1] `https://www.verifiedmarketresearch.com/product/global-head-mounted-display-market-size-and-forecast-to-2025/`

adaptarlos a los sistemas de almacenamiento tradicionales, ya que normalmente no ofrecen soporte para almacenar este tipo de contenido de forma directa. Una técnica es utilizar las proyecciones equirrectangulares (*Equirectangular Projection*, ERP), las cuales proyectan la información esférica de la escena en 360º a una imagen rectangular que contiene la información algo distorsionada en determinados puntos. Un ejemplo de ERP se puede observar en la Figura 1.1.

Figura 1.1: Primer fotograma del vídeo *Choir*. Las distorsiones se ven más evidentes en las zonas más alejadas del ecuador de la imagen. Las zonas con la máxima distorsión son los extremos superior e inferior de la imagen, donde prácticamente se está representando un punto de la esfera (el punto superior y el inferior) con una recta. Este fenómeno es similar al que ocurre cuando se trata de representar la forma esferoide del planeta Tierra en un plano, donde se distorsionan sobre todo los polos del planeta.

Dada la complejidad de su captura, actualmente no hay una gran cantidad de ODVs, pero debido al gran potencial y utilidad de estos sistemas, cada vez se estudian más posibles aplicaciones y mejoras de los mismos [1]: generación automática de miniaturas o de resúmenes de vídeos en 360º, compresión de vídeo inteligente o alineamiento automático de diferentes escenas en 360º entre otras.

Para estas aplicaciones, es importante poder conocer con la mayor exactitud posible cómo va a ser la atención del usuario, ya que ella va a ser la base sobre lo que se construyan el resto de aplicaciones. El estudio de la atención engloba dónde va a mirar, qué elementos la atraen más, o cómo el sonido junto al vídeo pueden definir el interés de una determinada región. En recientes estudios se han observado algunos fenómenos remarcables, como por ejemplo la existencia de un sesgo que lleva a los usuarios a enfocar su atención alrededor del centro de la escena (*equator center bias*, ECB), o el hecho de que el uso del sonido direccional de una escena ayuda a obtener modelos que generan predicciones más cercanas al comportamiento real de los usuarios (*ground truth*) [2].

Cuando se agregan todos los mapas de atención de este conjunto de datos, se observa

que hay un sesgo (ECB, también observado por Sitzmann *et al.* [1]). En general, los usuarios tienden a prestar más atención a las regiones centrales del panorama y menos a los polos del mismo. Una representación de este sesgo se puede encontrar en la Figura 1.2, donde la probabilidad de que un usuario preste atención a un determinado píxel es proporcional a la luminosidad del mismo.

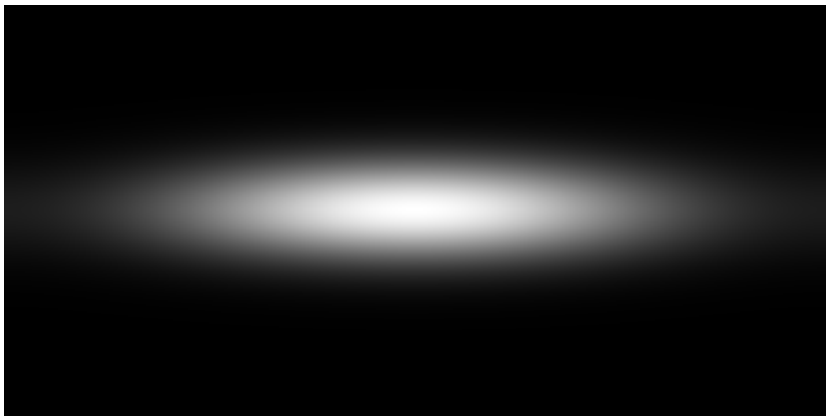

Figura 1.2: Imagen en escala de grises que muestra la apariencia del ECB que se usa en Chao *et al.* [2]. Cuanto más blanco sea un píxel, más probabilidad tendrá de ser observado por un usuario.

El objetivo final de este trabajo es el entrenar diferentes modelos que sean capaz de predecir la saliencia (mapa que representa las zonas donde el usuario va a centrar su atención) de contenido audiovisual en 360^{o}, y estudiar cómo diferentes cambios en el proceso de entrenamiento afectan al rendimiento.

Para ello se utilizarán redes neuronales convolucionales (explicadas más en detalle en el Apartado 2.1.3), y se partirá de los avances presentados en diferentes artículos que representan el estado del arte. Con este fin, la primera tarea ha sido analizar los modelos actuales disponibles para comprender su funcionamiento e implementación. Después, se han utilizado estos modelos como base para hacer diferentes modificaciones, que posteriormente se evaluarán para determinar su impacto en el desempeño del modelo mediante diferentes métricas.

1.1. Objetivos y alcance del proyecto

El objetivo último de este trabajo es diseñar y entrenar diversos modelos basados en redes neuronales convolucionales para predecir la saliencia audiovisual de un vídeo en 360º con audio direccional, y evaluar cómo diferentes parámetros repercuten en su desempeño.

Para conseguir este objetivo, se han identificado diferentes tareas en las que se va a dividir el trabajo:

- Estudio y análisis de la literatura que representa el estado del arte en los temas relacionados con el trabajo: Imágenes panorámicas equirrectangulares [3], atención y sesgos de los usuarios respecto a escenarios 3D [1], o la generación de *scanpaths* (el recorrido que hace la mirada del usuario analizando la escena 3D a lo largo del tiempo) realistas [4].

- Estudio de los conceptos teóricos que se van a utilizar, principalmente relacionados con la implementación de redes neuronales convolucionales, y de los conceptos utilizados en el campo del contenido 3D y de la atención.

- Puesta a punto de un entorno donde probar el funcionamiento de los modelos disponibles, y análisis de los resultados obtenidos con dichos modelos.

- Exploración de diferentes alternativas para modificar el modelo base [2], y análisis de los resultados obtenidos, comparándolos con los obtenidos por el modelo base, usando métricas ampliamente utilizadas (Bylinskii *et al.* [5]).

- Análisis del trabajo realizado y los resultados conseguidos, síntesis de las conclusiones, y redacción de la memoria.

1.2. Planificación y herramientas

El desarrollo del trabajo se ha dividido principalmente en los diferentes pasos que se han detallado en el apartado anterior.

Para ello se ha utilizado mayormente el lenguaje de programación Python, junto al sistema de aprendizaje profundo Pytorch. Se ha elegido este entorno porque se considera que es el más adecuado para implementar las redes neuronales con las que se va a trabajar, y además porque es el que se usa en el artículo sobre el que se basa este trabajo.

Por otra parte, se ha utilizado la herramienta Anaconda para poder gestionar los diferentes entornos de Python con los que se va a trabajar.

Mas allá de estas herramientas básicas, también se han utilizado otras diferentes para poder ir satisfaciendo necesidades que iban surgiendo en el desarrollo del trabajo. Entre ellas destacan:

- ffmpeg, como herramienta de manipulación de imágenes, la cual se ha utilizado para hacer las modificaciones necesarias a los conjuntos de datos disponibles, como por ejemplo dividir los vídeos en fotogramas y en audio.

- Editores de texto, principalmente *Visual Studio Code* y *Notepad++* para editar los diferentes archivos con los que se ha trabajado.

- El intérprete de comandos *Shell* para automatizar tareas ejecutadas en sistemas operativos Unix.

- Diferentes aplicaciones del ecosistema de Google, como *google slides*, *google sheets*, *goodle drive* o *google meet* para la gestión y organización del trabajo.

- El editor de *Overleaf*[2] para editar la presente memoria utilizando el lenguaje LaTeX[3].

1.2.1. Diagrama de Gantt

Con el objetivo de localizar temporalmente la realización de cada tarea y su coste temporal, se ha desarrollado un diagrama de Gantt[4] que se encuentra en la Figura 1.3. El diagrama de Gantt se divide en tres etapas:

- Etapa de preparación, donde se entra en contacto con las herramientas y tecnologías que se van a utilizar.

- Etapa de desarrollo, donde se centra y completan las tareas propuestas.

- Etapa de documentación, donde se recoge y documenta el trabajo realizado.

Estas etapas se subdividen a su vez en las subtareas correspondientes, en las cuales se especifica el número de horas que ha sido necesario para completarlas. Durante el trabajo se realizaron reuniones semanales, y ya que el incluir múltiples hitos no aportaría mucha información, se ha combinado esta tarea con la de entrega, para reflejar las horas invertidas en el diagrama.

[2] `https://es.overleaf.com`
[3] `https://www.latex-project.org/`
[4] Con ayuda de la web `https://app.teamgantt.com`

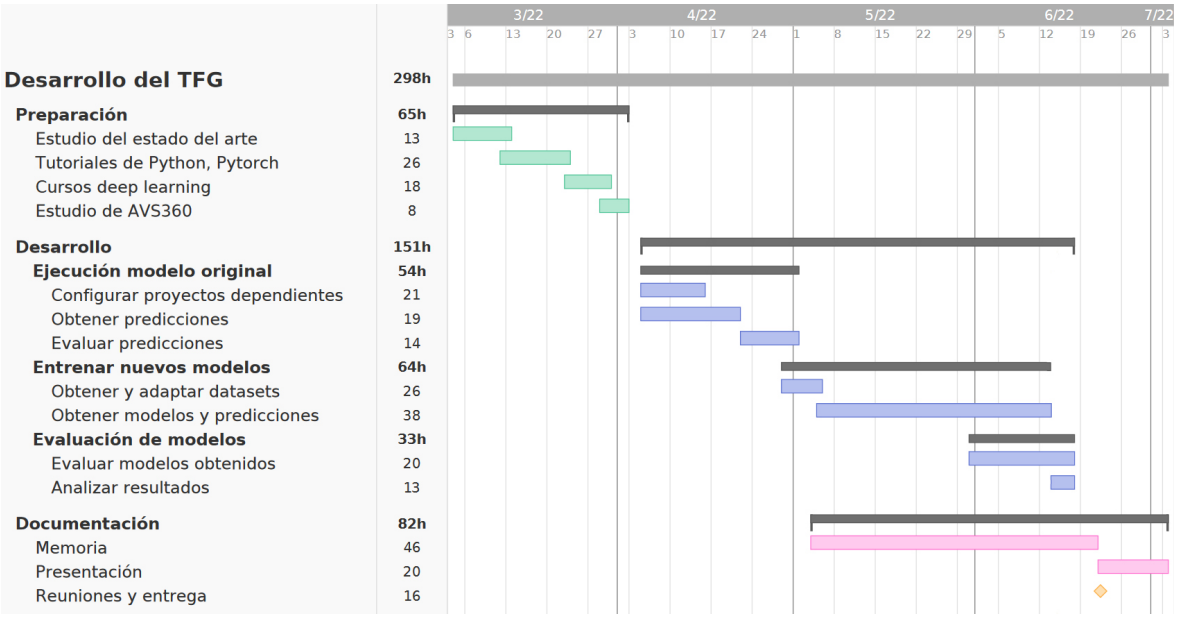

Figura 1.3: Diagrama de Gantt del trabajo.

1.3. Entornos de desarrollo

A lo largo del trabajo se han usado tres entornos de trabajo diferentes, los cuales se iban utilizando alternativamente dependiendo de la carga computacional que tenía cada tarea concreta.

Una descripción más detallada de las características de los entornos y una justificación de su uso en cada caso se encuentran en los siguientes apartados.

1.3.1. Equipo personal

El entorno de desarrollo que más se ha utilizado ha sido el ordenador portátil personal. Este computador tiene como procesador un i7-8550U a 1.8GHz, y como GPU una Intel UHD Graphics 620 de 4GB de memoria. Ambos componentes están enfocados a equipos portátiles, donde se intenta disminuir el gasto energético, y como consecuencia tienen un poder de cálculo reducido. Por lo tanto, el equipo personal se ha utilizado para ejecutar las diferentes tareas ligeras que han ido surgiendo a lo largo del trabajo. Al comienzo del mismo se utilizó para seguir las guías de las herramientas y configurar el entorno de desarrollo, además de para entrenar modelos sencillos que sirvieron de prueba.

Más adelante, se ha utilizado este equipo para otras tareas ligeras entre las que se incluyen la división de vídeos (mp4) en sus respectivos fotogramas (jpg) y audios de 1 segundo (wav) (ver Apartado 3.3.1), cálculo de los *ground truth* de los mapas de saliencia, cálculo de las funciones de error entre las predicciones y los *ground truth*

(muestra que representa el resultado real esperado, tomado haciendo observaciones a usuarios reales), y la ejecución de diversos programas.

1.3.2. Google Colab

También se utilizó el servicio de *Google Colaboratory*[5]. Este es un producto de *Google Research*, a través del cual se puede hacer uso de recursos informáticos (especialmente GPUs) para poder ejecutar, en este caso, programas de Python. Con la licencia de *Colab Pro* que se ha utilizado, se ofrece un procesador Intel Xeon a 2.20GHz, y como GPUs se ofrecen una Tesla P100-PCIE-16GB, o una Tesla T4, ambas con 16GB de memoria.

El inconveniente de utilizar *Google Colab*, es que no se garantizan los recursos, ya que les establecen límites de uso para poder ofrecerlos a la mayor cantidad de usuarios, aunque con la licencia Pro, normalmente se ha podido hacer las ejecuciones sin ningún inconveniente. Así, este entorno se ha utilizado para tareas que o bien por el tiempo de ejecución, o bien por la memoria necesaria, no eran posibles de ejecutar en el equipo personal. Principalmente las tareas que se han ejecutado en este entorno han sido las predicciones de saliencia a partir de un modelo entrenado.

1.3.3. MK2

El tercer y último entorno que se ha utilizado ha sido una estación de trabajo proporcionada por el grupo de investigación en el cual se ha desarrollado el trabajo, el *Graphics & Imaging Lab*[6]. Es un sistema *UNIX* el cual se ha utilizado de forma remota. El sistema tiene dos procesadores Intel Xeon Gold 6140 a 2.30 GHz, y la GPU utilizada es una Quadro P5000 con 16GB de memoria. Este último sistema se ha utilizado principalmente para realizar los entrenamientos más costosos: los entrenamientos de los modelos completos que utilizan el conjunto de datos de entrenamiento junto a diferentes funciones de pérdida y configuraciones (el proceso de entrenamiento se describirá de forma más completa a lo largo de la memoria).

El entrenamiento ha sido necesario ejecutarlo en este entorno, ya que utiliza aproximadamente 13GB de memoria en el mismo, por lo que no hubiera sido posible entrenar el modelo en el equipo personal (Apartado 1.3.1), y la otra opción, que era utilizar Google Colab (Apartado 1.3.2), si bien sí que podía gestionar la memoria requerida, presentaba otros problemas como la posibilidad de cortar la ejecución del entrenamiento por los límites de las cuotas del entorno.

[5] `https://colab.research.google.com/`
[6] `http://graphics.unizar.es/`

Capítulo 2

Contexto y Trabajo relacionado

2.1. Fundamentos teóricos

En este apartado se desarrollan los diferentes fundamentos teóricos sobre los que se basa el presente proyecto. De esta forma, se aclararán los conceptos más importantes que se van a utilizar en este trabajo, y será más fácil seguir las explicaciones y justificaciones que se presenten en apartados posteriores.

2.1.1. Redes neuronales

Las redes neuronales (o *Neural Networks*, NN) son un tipo de modelo computacional que se basa en simular el comportamiento de las neuronas biológicas de los seres vivos para tratar de resolver problemas de diferente complejidad.

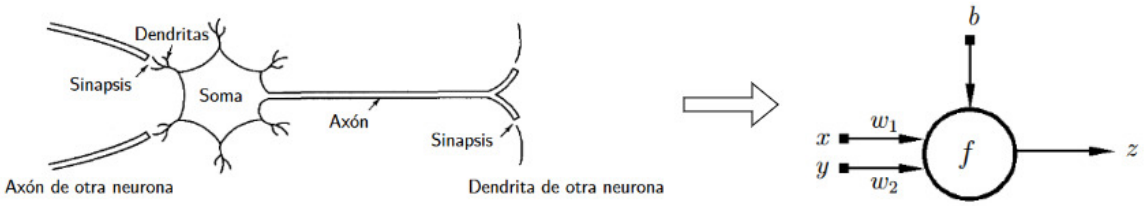

Figura 2.1: Símil entre la neurona biológica y la artificial [6]. La neurona artificial tiene una serie de entradas (x e y), pesos que modifican los valores de la entrada antes de gestionarla (*w1* y *w2*), un sesgo o *bias* (b) que se usa durante el cómputo, una determinada función (f) que será la que tome lugar dentro de la neurona, y por último la salida (z) que será el resultado de los cálculos realizados.

Como se puede ver en la Figura 2.1, hay una clara similitud entre los componentes de la neurona biológica y la artificial. Por una parte tienen una entrada de los datos, que en la artificial vienen acompañados por los pesos (*w1 y w2*), y por el *bias* o sesgo (b). Luego nos encontramos con el cuerpo de la neurona, que es donde se ejecuta el cómputo sobre los datos de entrada, y por último se comunica el resultado obtenido por la salida.

El cálculo más básico que se puede realizar dentro de la neurona es simplemente multiplicar los valores de cada señal por sus respectivos pesos y sumarlo todo (una combinación lineal). Tras obtener el resultado del respectivo cálculo, al valor resultante se le aplica una función de activación, y el resultado de esto es el valor que sacará la neurona como salida.

Estas neuronas se agrupan por capas, que pueden pertenecer a tres categorías: la capa de entrada, cuyas neuronas tendrán como valores de entrada los de los propios datos que se utilicen para alimentar la red, las capas ocultas (puede haber sólo una o no haberla), en las que la red realiza sus abstracciones de los datos y, por último, la capa de salida, en la que se toman los resultados procesados por la red y se transforman para obtener el resultado final.

Las funciones de activación son transformaciones que se les aplican a las salidas de las diferentes capas, para que los valores obtenidos tengan sentido en el contexto sobre el que se trabaja. Por ejemplo, si se está trabajando con probabilidades, cabe esperar que se utilice una función de activación que transforme una salida (por ejemplo los valores [1.3, 5.1, 2.2, 0.7, 1.1]) en las probabilidades correspondientes (en el ejemplo anterior, aplicando la función de activación *softmax*[1] se obtendrían las probabilidades [0.02, 0.9, 0.05, 0.01, 0.02]). A lo largo de los años se han ido diseñando diferentes funciones de activación para mejorar las ya existentes o para adaptarlas a otros tipos de problemas. Entre ellas destacan:

- Función escalón. Es la función más sencilla, la cual devuelve 1 si el valor introducido es superior a un determinado umbral o *threshold*, y 0 en caso contrario.

- Función sigmoidal. Es una versión suavizada de la función escalón. Cuanto mayor sea el valor de la entrada, más se acercará a 1 la salida, y cuanto menor sea el valor de la entrada, más se acercará a 0. Es ampliamente utilizada en regresión logística, y una de las grandes ventajas que tiene sobre la función escalón es que es derivable. Se suele poner como última capa de la red neuronal para poder interpretar los resultados como la probabilidad de cierto suceso.

- Función *ReLU*. Esta función devuelve 0 cuando la entrada es negativa, y en caso contrario devuelve el valor de la propia entrada. Al contrario de las dos funciones anteriores, los resultados no están restringidos entre el 0 y el 1.

- Función tangente hiperbólica. Esta función es muy parecida a la sigmoidal, pero en vez de devolver resultados en un rango entre 0 y 1, los devuelve entre -1 y 1.

[1] `https://deepai.org/machine-learning-glossary-and-terms/softmax-layer`

2.1.2. Funciones de error o *loss*

La función de error representa la divergencia entre los resultados obtenidos y los resultados esperados, y a partir de ella se puede estudiar cómo de optimizada está la red que se ha entrenado.

Existen diversos ejemplos de funciones de pérdida, las cuales habrá que usar dependiendo del modelo con el que se está trabajando. Una de las funciones de pérdida más utilizadas en el campo del aprendizaje automático es el error cuadrático medio (*Mean Squared Error*, MSE), también llamado *L2*, la cual se usa comúnmente en problemas de regresión (intentar establecer una relación entre un número de características y una variable objetivo, la regresión más básica es la lineal), y la fórmula para calcularla se encuentra en la Ecuación 2.1, donde n es el número total de muestras, Y_i es el dato, e $\hat{Y}_i$ es el valor predicho.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.1}$$

Por otra parte, para los problemas de clasificación (predecir a qué clase discreta pertenece una muestra dada) se suelen utilizar otro tipo de funciones, como por ejemplo la función de la entropía cruzada (*Cross Entropy loss*, CE), cuya fórmula se encuentra en la Ecuación 2.2. Las funciones utilizadas en este tipo de problemas se suelen basar en evaluar la probabilidad que tiene una muestra de pertenecer a una determinada clase.

$$CE = -\sum_{i}^{n} t_i log(f(s)_i) \tag{2.2}$$

Una vez definidos todos los apartados de una red, ésta se entrena a lo largo de lo que se llaman épocas (o *epochs*; en cada época se recorre una vez todo el conjunto de datos). El entrenamiento supervisado (hay otros tipos de entrenamiento, como por ejemplo el no supervisado[2], en el cual los datos no están etiquetados, es decir, no se sabe qué son los datos de entrada) se basa en ir ajustando los pesos de la red haciendo que cada vez las predicciones que haga sobre los datos de entrenamiento se ajusten más a los resultados esperados.

Este entrenamiento se realiza un número limitado de épocas, ya que de otra forma se *sobreajusta* la red a los datos de entrenamiento, perdiendo generalidad y teniendo un mal desempeño al intentar predecir nuevas entradas. También se ha de tener en cuenta el efecto contrario, el *subajuste*, el cual aparece cuando no se entrena un modelo durante

[2] `https://deepai.org/machine-learning-glossary-and-terms/unsupervised-learning`

un número suficiente de épocas, y por lo tanto no termina de aprender el conjunto de datos utilizado.

2.1.3. Redes neuronales convolucionales

Las redes neuronales convencionales no escalan bien al utilizar imágenes como entrada, ya que al estar las capas totalmente conectadas entre sí (*fully-connected*), el número de pesos que hay que aprender se dispara con los tamaños de las imágenes, además de que este tipo de capas no mantiene las relaciones espaciales de los datos.

Las redes neuronales convolucionales (o *Convolutional Neural Networks*, CNN) aparecen como una adaptación de las redes neuronales convencionales para solventar este problema. Esto lo hacen aprovechándose de que están especializadas en procesar imágenes, y por lo tanto se construyen capas "tridimensionales": ahora las neuronas están distribuidas a lo largo de la anchura, altura y profundidad de una capa (como se puede ver en la Figura 2.2), y no tienen por qué estar conectadas con todas las neuronas de la capa anterior.

La estructura y funcionamiento de las redes neuronales convolucionales es análoga a la de las redes neuronales básicas explicadas en el punto anterior: se introducen una serie de datos en la capa de entrada, estos datos se van transformando a lo largo de las capas ocultas, y por último se obtienen unos resultados que se sacan por la capa de salida.

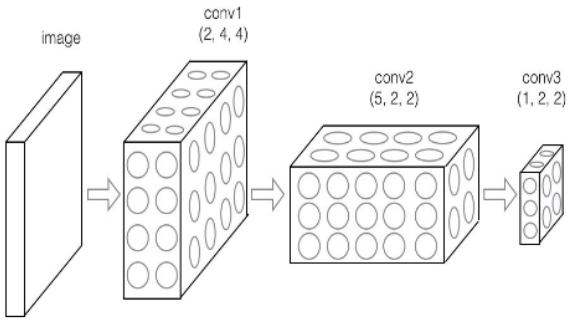

Figura 2.2: Estructura básica de una red convolucional [7].

Cada capa de una CNN transforma una entrada tridimensional en una salida también tridimensional, y pueden pertenecer a distintos tipos, como capas de convolución, de *pool*, o capas completamente conectadas (*fully-connected*, FC).

Una arquitectura básica de CNN podría ser la siguiente:

– Una capa de entrada donde se introduzcan los datos de los píxeles de una imagen. Si por ejemplo se está usando una imagen con tamaño 32 x 32 píxeles y formato RGB (3 canales), esta capa tendrá dimensiones [32x32x3].

– Una capa convolucional que aplique las transformaciones convolucionales a los datos introducidos como entrada. Si se utilizan 12 filtros, el volumen de salida de esta capa será [32x32x12].

– Una capa que aplique una función de activación a todos los elementos de la capa anterior. Esta función de activación podría ser una ReLU, y las dimensiones de salida serían las mismas que las de entrada: [32x32x12].

– Una capa de *pool* que disminuya las dimensiones de los datos. Esta capa podría tener como salida un volumen [16x16x12].

– Una última capa la cual suele ser una FC, la cual codifica la salida dependiendo de la aplicación. Si por ejemplo se quieren diferenciar 10 clases diferentes, esta última capa devolverá un volumen de tamaño [1x1x10].

Un ejemplo de representación de la arquitectura de la CNN comentada se puede encontrar en la Figura 2.3.

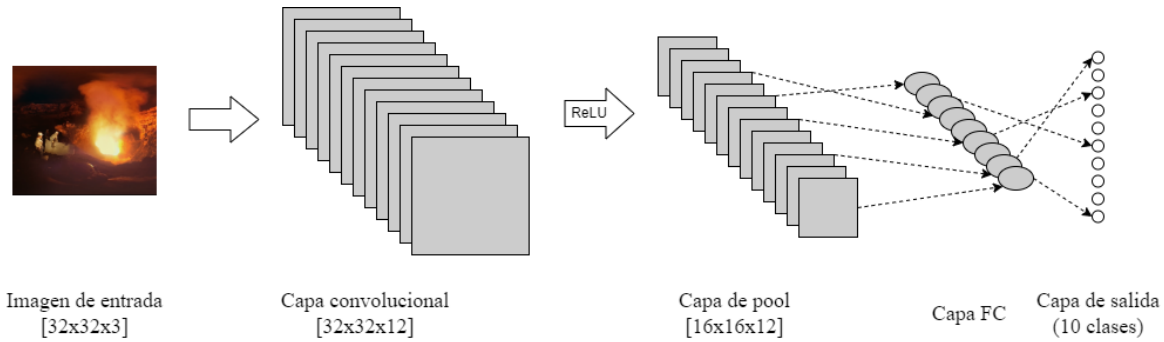

Figura 2.3: Representación de una CNN sencilla.

Estas capas se pueden combinar de múltiples formas para definir redes neuronales más complejas. Un ejemplo de ello se puede ver en la Figura 2.4, donde se puede apreciar una visualización del funcionamiento de una CNN clasificando una imagen que representa un coche del conjunto de datos CIFAR10[3].

Para más información sobre CNNs, se recomienda consultar los cursos CS230 [7] y CS231n [8] de Stanford.

2.2. Artículos relacionados

Antes de la realización del trabajo, se consideró interesante el estudio de algunos artículos que representaban el estado del arte sobre los temas principales sobre los que va a tratar el proyecto. Así, en este apartado se discuten brevemente algunos de estos

[3] `http://www.cs.toronto.edu/~kriz/cifar.html`

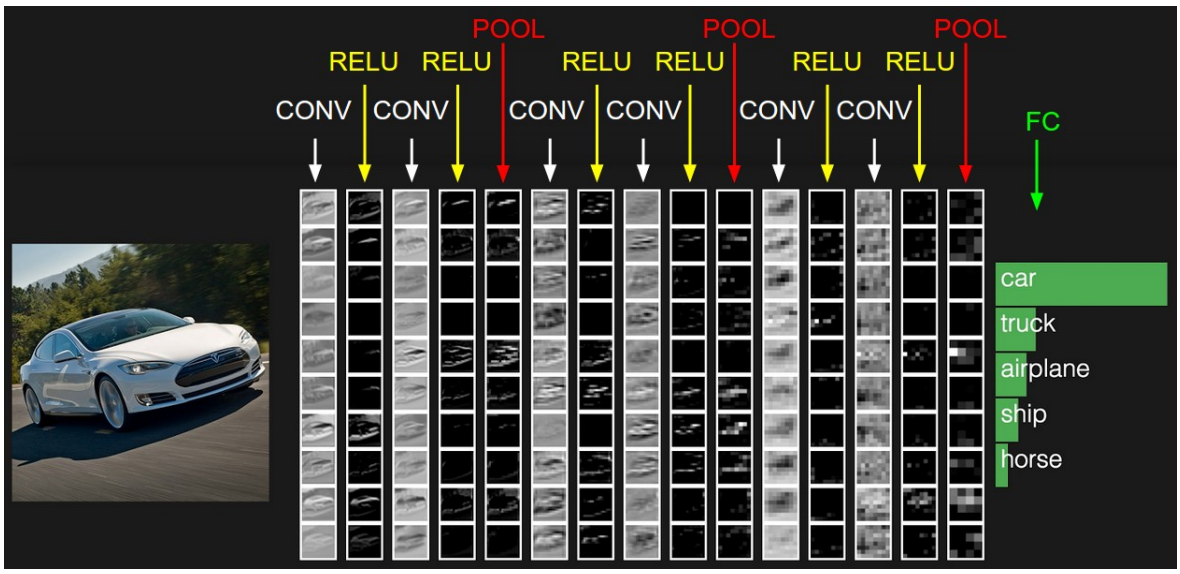

Figura 2.4: Apariencia de las diferentes capas de una red convolucional durante la clasificación de una imagen del conjunto de datos CIFAR10. Para representar la tercera dimensión de las capas, se muestran varias activaciones ("rodajas") de cada capa [8].

artículos, resumiendo los apartados más importantes de cada uno de ellos y detallando el aprendizaje obtenido con los mismos.

2.2.1. Comportamiento humano en entornos virtuales

Este trabajo enmarca su motivación en una línea de investigación abierta que busca responder a: *¿Cómo se comportan los seres humanos en entornos virtuales?*. Esta pregunta no es trivial, y se han publicado multitud de artículos presentando diferentes aproximaciones, desde intentar modelar los diferentes caminos (o *scanpaths*) por los que los usuarios investigan una escena 360º (Martin *et al.* [4]), hasta utilizar redes generativas adversarias (por sus siglas en inglés, GANs) para predecir la saliencia audiovisual (Chao *et al.* [9]).

Para poder diseñar ciertas aplicaciones o herramientas relacionadas con el contenido en 360º como las nombradas en el Apartado 1, es crucial tener una base de conocimientos sólida sobre cómo se comportan los humanos en dichos entornos. Como consecuencia del aumento en el interés sobre este tipo de contenido, ha aumentado la necesidad de estudiar y comprender el comportamiento humano cuando se le presentan diferentes escenarios inmersivos con diferentes condiciones y características. Debido a esto, se han publicado diferentes artículos que explican características del comportamiento humano en estos entornos. Por ejemplo, algunos artículos afirman que la atención de los usuarios en un entorno virtual se suele centrar en el centro de la escena, representando este sesgo como en la Figura 1.2 [10].

Además, también se ha estudiado el impacto que tienen las señales sonoras a la

hora de predecir dónde se dirige la atención en la escena. Se ha comprobado que esto depende por una parte del tipo de sonido: no influye de igual manera el sonido de una conversación que el de una canción [2]. Y por otra parte, la modalidad del audio que se está reproduciendo: no es lo mismo utilizar una pista de audio *mono* (sonido que se reproduce por un único canal, y que sería lo equivalente a escucharlo por un solo oído), que utilizar una pista de audio *direccional* (formato especial que reproduce el sonido no solo dependiendo del plano horizontal, sino que también tiene en cuenta el vertical, reproduciéndolo en toda la esfera en la que puede mirar un usuario utilizando un HMD) [2].

Utilizando estas dos características de forma conjunta, se puede mejorar sustancialmente la calidad de las predicciones de la atención de los usuarios en entornos virtuales [11].

2.2.2. Cómo exploran los usuarios los entornos virtuales

A lo largo de los últimos años han surgido multitud de artículos, como el de Sitzmann *et al.* [1], en el cual se estudian qué condiciones pueden afectar, y en qué magnitud, a la manera en la cual un usuario explora una escena en 360º.

La mayoría del trabajo desarrollado hasta el momento se había centrado en investigar el comportamiento humano visualizando contenido en un monitor convencional, pero gran parte de esta investigación no se puede extrapolar directamente al comportamiento humano en los entornos virtuales. Esto se debe a que en estos entornos influyen muchos más factores derivados del cambio en la forma en la que un usuario investiga la escena: la información que se presenta depende de la orientación de la cabeza, de su rotación, y de otros factores físicos. Por lo tanto, en este trabajo [1] se recogieron un total de 1980 muestras de las trayectorias de la atención de los usuarios, con las que se plantearon una serie de hipótesis:

- Los usuarios comparten ciertos patrones de comportamiento. Esto se justifica con el hecho de que el 70 % de las fijaciones (zonas puntuales de la escena que se están observando[4]) se concentran en el mismo 20 % de las zonas más observadas de la escena.

- Existe un sesgo entre los usuarios el cual los hace centrar su atención en el ecuador de la escena, siendo en general poco probable que se investiguen zonas lejanas al ecuador del panorama.

[4] `https://en.wikipedia.org/wiki/Fixation_(visual)`

– Escenas con menor entropía (las fijaciones de los usuarios están más focalizadas en zonas concretas) son analizadas con mayor rapidez por los usuarios. Esto se debe a que los usuarios encuentran con mayor velocidad los puntos interesantes de la escena, explorándola de forma más eficiente.

– El punto inicial de visualización de la escena no afecta en exceso al comportamiento. La justificación que se da en el artículo es que, pasados 30 segundos, la atención de los usuarios converge.

2.2.3. Predicción del comportamiento humano en realidad virtual

El conjunto de artículos que estudian la predicción del comportamiento humano en entornos de realidad virtual no es demasiado extenso, aunque en los últimos años han ido surgiendo nuevos artículos que lo estudian desde diferentes perspectivas. Han surgido artículos como el de Martin *et al.* [3], en los cuales se presentan nuevas técnicas, o se intentan mejorar las existentes, para predecir los mapas de saliencia de escenas de realidad virtual. En este artículo en concreto, se presenta un nuevo tipo de convoluciones, las *convoluciones panorámicas*, como alternativa a las convoluciones habituales (Figura 2.5), obteniendo con ellas resultados superiores al estado del arte, ya que las convoluciones panorámicas tienen en cuenta las características principales de la proyección equirrectangular.

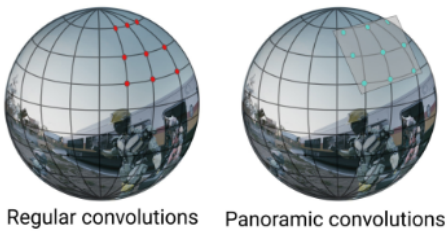

Figura 2.5: Representación de las convoluciones panorámicas [3]. Estas convoluciones funcionan especialmente bien en los panoramas equirrectangulares. Se utiliza la proyección gnomónica, la cual se basa en proyectar los puntos de la esfera en el plano tangente.

Otros artículos se centran más en el estudio y predicción de *scanpaths* (el recorrido concreto que realiza una persona al explorar una escena) [4]. En este segundo artículo se presenta una nueva función de error, la cual tiene en cuenta dinámicamente las variaciones temporales de la escena, lo cual, junto al uso de *generative adversarial networks* (GANs)[5], se consiguen resultados que mejoran los del estado del arte en

[5] `https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/`

la materia. Esto permite la simulación de *scanpaths* altamente realistas, con un comportamiento que emula con éxito el humano, y por lo tanto se pueden generar artificialmente una gran cantidad de *observadores virtuales*.

Los *scanpaths* generados se pueden utilizar para múltiples aplicaciones, entre ellas el diseño de escenas virtuales, generación de miniaturas de imágenes 360º estáticas dirigida por *scanpaths*, o cualquier aplicación que requiera simular de manera realista la forma que tienen los humanos de analizar una escena.

2.2.4. Importancia de las señales sonoras en la exploración de una escena de realidad virtual

Una característica importante que van a tener en cuenta los modelos desarrollados para poder realizar mejores predicciones son las señales de audio. Hay artículos previos que ya han estudiado la importancia de estas señales de audio a la hora de predecir la atención de los usuarios, como por ejemplo el de Masiá *et al.* [12], en el cual se estudia cómo influye el sonido en la exploración de cortes de películas en 360º. Este artículo se basa en que quienes controlan la cámara a la hora de visualizar contenido 360º son los usuarios, lo cual difiere de la visualización de contenido convencional, donde el creador de contenido tenía control total sobre el movimiento de la cámara.

En este contexto, se estudia la importancia de tener en cuenta las pistas de audio a la hora de estudiar el comportamiento humano en contenido 360º, concretamente en los cortes entre escenas, algo que no se había hecho hasta la fecha. Este artículo se basa en estudiar el comportamiento en torno a las áreas de interés (ROIs, del inglés *Region of Interest*, Figura 2.6). Estas áreas son las áreas donde se espera que el usuario fije su atención, y en ellas suelen estar los personajes principales del contenido que se está visualizando, y también se suele centrar la acción de la escena en un determinado momento.

Figura 2.6: Aspecto de diferentes configuraciones de ROIs: un único ROI, dos ROIs en el mismo campo de visión y dos ROIs en diferentes campos de visión. Los ROIs están representados por cuadrados naranjas, y tienen un sonido asociado alineado en la escena, representado por el icono del altavoz azul [12].

Se ha llegado a la conclusión de que se puede utilizar el sonido direccional para ayudar a los usuarios a encontrar los ROIs más rápidamente. Aplicado a los cortes de

películas 360º, utilizando el sonido se puede predecir con una mayor fiabilidad adónde se va a dirigir la atención del usuario, ya que ésta estará normalmente guiada por el sonido que se esté reproduciendo. También se ha observado que los usuarios utilizan el sonido para encontrar los diferentes ROIs después de un corte. Si hay un único ROI, los usuarios suelen fijar su atención en esa zona, pero si hay más de uno, se puede prever que los usuarios vayan alternando su atención entre los diferentes ROI, guiándose siempre por las pistas de sonido.

La práctica totalidad de las conclusiones obtenidas en este artículo se pueden adaptar o extrapolar para sacar conclusiones similares en el uso de sonido direccional en el contenido 360º general (no enfocado únicamente a los cortes de películas).

En resumen, las conclusiones de todos estos trabajos apuntan a que para modelar satisfactoriamente la atención humana se debe tener en cuenta tanto el contenido visual como el auditivo, y que en concreto, el audio direccional, juega un papel clave en cómo dirigimos nuestra atención cuando exploramos una escena en 360º.

Capítulo 3

Conjuntos de datos utilizados

Se han utilizado diferentes conjuntos de datos a la hora de entrenar y evaluar los diferentes modelos empleados a lo largo de este proyecto. Esto se ha hecho así, por una parte, para tener una mayor cantidad de datos con los que trabajar y, por otra parte, para que los datos de test sean diferentes a los de entrenamiento, con lo que se comprobará si los modelos entrenados generalizan lo suficientemente bien. En este apartado se comentan los dos conjuntos de datos elegidos, y se explica su origen, sus características, cómo se han obtenido, y para qué se van a utilizar.

3.1. 360 Audio Visual (ICMEW 2020)

Este primer conjunto de datos [13] es el que se utiliza en el trabajo de Chao *et al.* [2] a la hora de hacer la evaluación (test) del modelo. Su origen es un artículo de investigación donde se presenta un conjunto de datos, cuya característica diferencial es que se captura el audio direccional de la escena, que se puede representar en forma de *Audio Energy Maps* (AEM). Estos AEM muestran las zonas de donde proviene el sonido de la escena en un determinado fotograma. Un ejemplo de representación de un AEM se puede apreciar en la Figura 3.1.

En este conjunto de datos se tienen en cuenta tres tipos de modalidades de audio:

- *Mute*, en la que no se utiliza la información acústica.
- *Mono*, en la que se utiliza la información acústica pero con un único canal, obteniendo por lo tanto un sonido semejante a escuchar solamente por un oído.
- *Ambisonics*, en el que se representa el sonido a lo largo de toda la esfera que rodea al usuario en un entorno de realidad virtual.

El conjunto de datos contiene 15 vídeos de YouTube en resolución 4K (3840 $\times$ 1920), con duración de 25 segundos, aunque con diferente número de fotogramas por

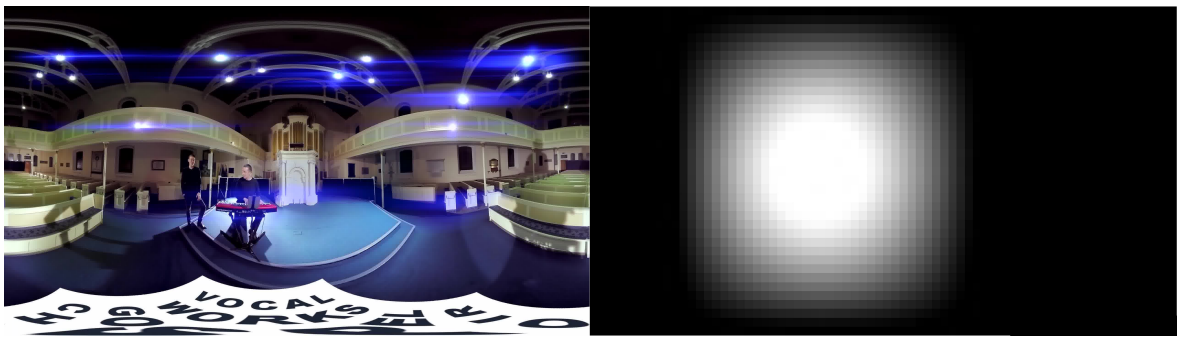

Figura 3.1: Representación visual (izquierda) y AEM (derecha) del fotograma 24 del vídeo *Choir*. Las zonas más blancas del AEM representan el origen del sonido en la escena [13]. En este fotograma en concreto, todo el sonido proviene de la pareja de músicos que se puede observar en la escena.

segundo (*Frames per Second*, FPS). Todos ellos tienen audio direccional, así que para conseguir los datos con modalidad *Mute*, se eliminó la información acústica, mientras que para conseguir los de la modalidad *Mono*, se combinó toda la información en un único canal. Por último, el conjunto de datos incluye datos de fijaciones reales de usuarios explorando la escena (los puntos de la escena que han observado los usuarios), y que han sido obtenidos a partir de 45 participantes, utilizando 15 para cada una de las tres modalidades de audio comentadas.

Para más información acerca del conjunto de datos o para ver un análisis más detallado de los mismos se recomienda leer el artículo original [13].

Este conjunto de datos se utilizó, en primera instancia, para hacer un entrenamiento y un test de prueba, comprobando así que todo el proceso de entrenamiento implementado funcionaba correctamente. Finalmente, en este trabajo se va a utilizar este primer conjunto de datos en la etapa de test de un modelo previamente entrenado. Esto se debe a que este conjunto es pequeño, y se ha considerado insuficiente para entrenar el modelo, siendo previsible que un modelo entrenado con tan pocos datos generalizará mal a nuevas entradas.

3.2. Dynamic 360º Immersive Videos (CVPR 2018)

El segundo conjunto de datos empleado es el que se utilizó en el artículo comentado en el Apartado 1 [2], y usado para entrenar el modelo final. Este conjunto surge de otro artículo científico [14], en el cual se presenta un conjunto de datos amplio con el que se intenta, de nuevo, predecir la atención de los usuarios en entornos de realidad virtual.

Está compuesto por 208 vídeos en 360º los cuales tienen diferentes duraciones, pero todos ellos están a 25 FPS. Este artículo [14] es algo más antiguo que el anterior, y el conjunto de datos que se proporciona no tiene en cuenta el sonido de la escena, es decir,

los fotogramas no están asociados a ningún AEM (a diferencia del primer conjunto de datos).

Respecto a los datos de fijaciones de usuarios reales, se recogieron los datos de visualización de cada vídeo con, como mínimo, 31 usuarios diferentes, obteniendo 6672 muestras en total (en el primer conjunto de datos había únicamente 675 muestras).

Así, al igual que en el artículo original, en este trabajo se va a utilizar este segundo conjunto de datos para entrenar los diferentes modelos que se van a evaluar posteriormente. Esto se ha hecho así principalmente por la gran diferencia en el número de muestras.

3.3. Modificaciones a los datos

Los datos descargados no se pueden utilizar directamente para entrenar o evaluar un modelo, sino que hay que hacerles ligeros ajustes y adaptaciones para que puedan servir como entrada a una red neuronal.

3.3.1. Obtención de los datos de entrada de la red

Lo primero que se ha hecho ha sido dividir los vídeos en sus diferentes fotogramas (*frames*), y pistas de audio de 1 segundo. En el proceso de división es crucial tener en cuenta los FPS del vídeo con el que se está trabajando, para poder dividirlo correctamente, y que luego en el proceso de entrenamiento (o de *test*), se empareje correctamente cada fotograma con su dato real (*ground truth*). Para realizar estas divisiones se ha utilizado mayoritariamente la herramienta ffmpeg[1] la cual, utilizando los parámetros adecuados, facilitaba la división de los vídeos.

3.3.2. Obtención de los mapas de saliencia

Por otra parte, los datos del *ground truth* que representan las fijaciones reales de los usuarios se proporcionan en ficheros de texto con un formato especificado normalmente en los correspondientes *READMEs* (ficheros que explican las características más importantes de un proyecto/archivo para que el usuario lo pueda utilizar sin necesidad de comprender en profundidad todo el proyecto/archivo).

Para utilizarlos en una red neuronal convolucional, ha sido necesario traducir los datos de fijaciones en imágenes que representen los mapas de saliencia de los diferentes fotogramas.

Con este propósito, se ha construido un programa en el lenguaje de programación Python que, para cada fotograma de cada vídeo, coge todas las fijaciones de todos los

[1] `https://ffmpeg.org/`

usuarios que han visualizado dicho fotograma, los acumula en una imagen, y después se difumina utilizando un desenfoque gaussiano[2] para emular el mapa de saliencia (Sitzmann *et al.* [1]).

Una representación del proceso y un ejemplo de un mapa de saliencia generado se puede apreciar en la Figura 3.2. En dicho ejemplo se puede apreciar cómo los usuarios observan las zonas de la escena esperadas: la zona a la izquierda donde se encuentran las personas, y la zona central donde se encuentra el humo.

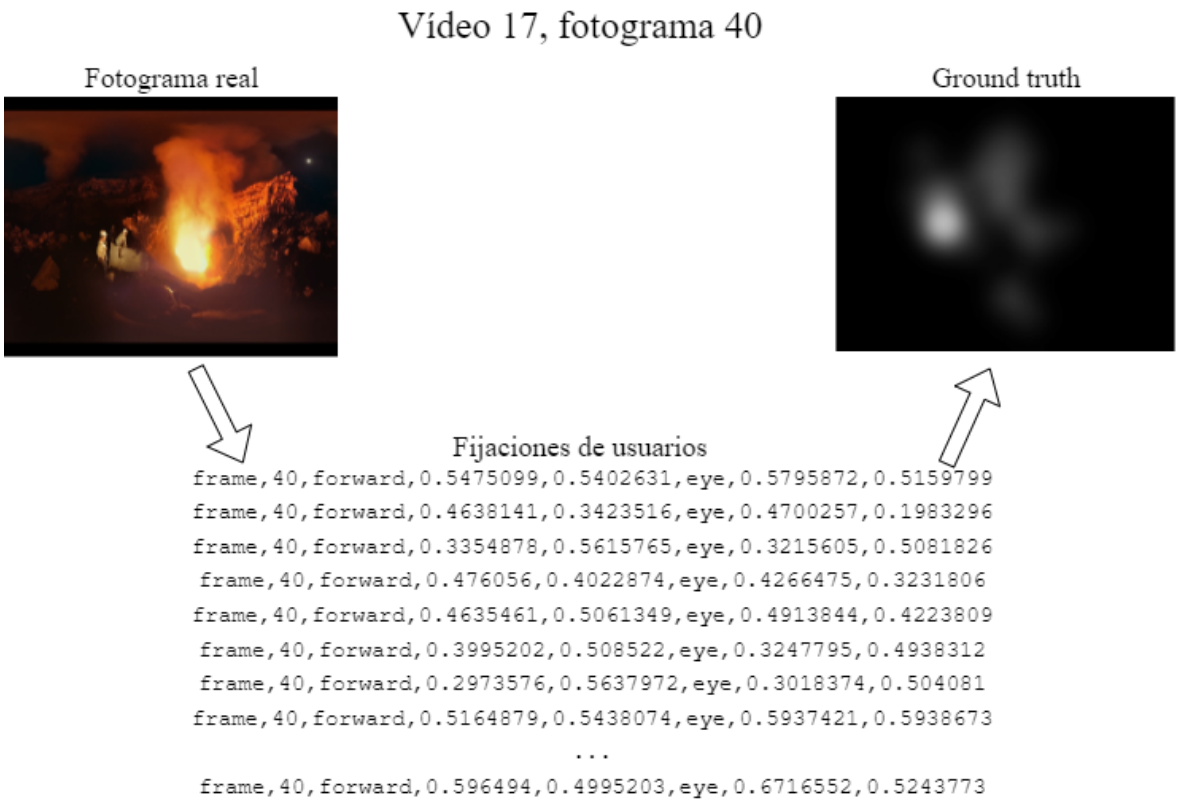

Figura 3.2: Esquema de construcción de un mapa de saliencia a partir de los datos de fijaciones dados. A la izquierda se encuentra el fotograma real del vídeo, abajo se encuentra una lista (incompleta) de las diferentes fijaciones de los usuarios sobre dicho fotograma del vídeo, y a la derecha se encuentra el mapa de saliencia generado por el programa. Para más información se recomienda consultar la descripción del programa *convert.py* en el Anexo A.

[2] `https://iq.opengenus.org/gaussian-blur/`

Capítulo 4

Modelo de predicción de saliencia audiovisual

Una vez explicados los conceptos sobre los que se cimienta este trabajo, en este apartado se va a explicar el modelo original que se ha tomado como base, así como las diferentes opciones exploradas a lo largo del trabajo.

4.1. Arquitectura original de la red neuronal

En este apartado se va a explicar cuál es la arquitectura de la red neuronal convolucional (CNN) utilizada en el modelo que se ha tomado como punto de partida.

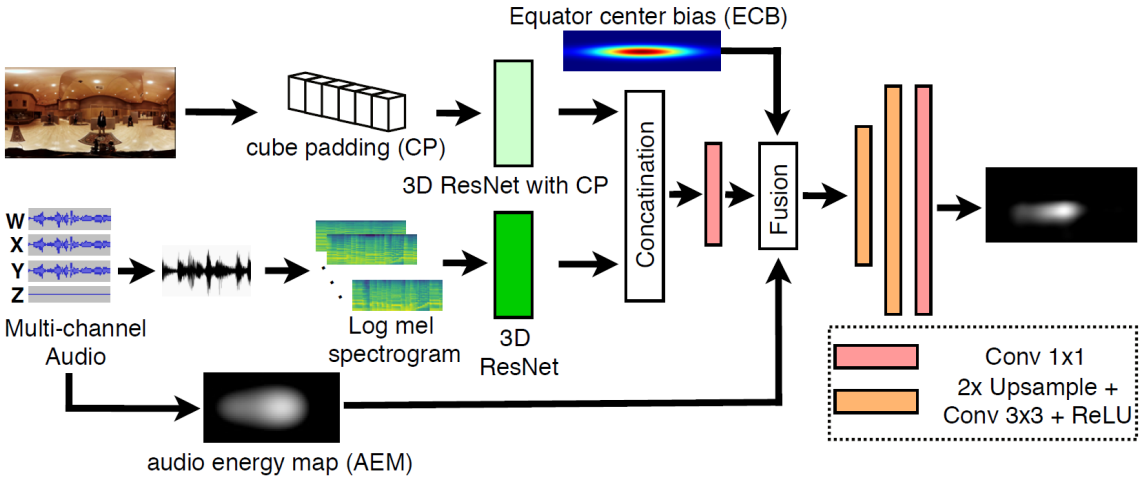

Figura 4.1: Arquitectura del modelo AVS360, que es el que se ha tomado como base para el desarrollo de este trabajo. Imagen extraída de Chao *et al.* [2].

La arquitectura que se utiliza en el modelo (Figura 4.1) se basa en dos redes neuronales residuales (*ResNet*) utilizadas para extraer de forma separada la información visual y acústica de un vídeo omnidireccional (ODV), fusionándolas luego con otra información de la escena.

La información visual en 3D de un ODV viene representada en una proyección equirrectangular (ERP) 2D (que como se ha comentado en el Apartado 1, es una forma de traducir la información esférica de la escena en 360º en una imagen rectangular), para ser compatible con los sistemas actuales de procesado de vídeo. Un ejemplo de aplicación de este ERP se puede ver en la Figura 4.2, donde se aprecia la distorsión introducida en la transformación, sobre todo en las zonas más alejadas del ecuador. Debido a esta distorsión, lo primero que se hace con esta información es aplicarle un *cube padding* (CP). Con este CP se proyecta la información de la escena en las 6 caras de un cubo, disminuyendo así la distorsión geométrica presente en un ERP.

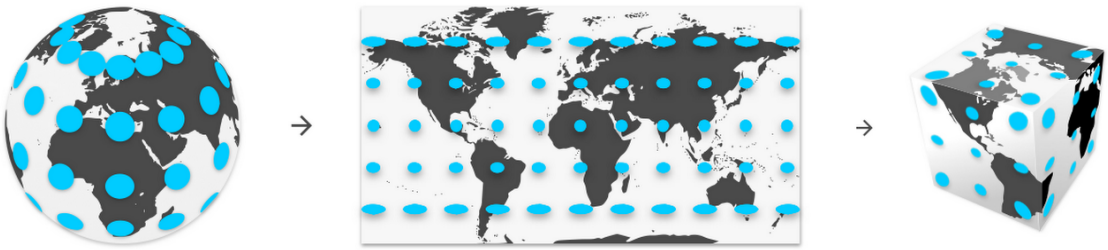

Figura 4.2: Proyección equirrectangular de la Indicatriz de Tissot[1][15] con su correspondiente *cube padding*.

Una vez transformada la información visual, ésta se utiliza como entrada de una *ResNet* 3D con la cual se extraen distintas características visuales.

La otra parte principal de la arquitectura es la extracción de la información acústica de la escena. Para ello, se utiliza de nuevo una *ResNet* 3D, la cual toma como entrada el audio muestreado en diferentes canales, y produce como salida diferentes atributos sonoros.

Por último, una vez obtenida la información visual y acústica de la escena, éstas se concatenan, y el resultado se fusiona con las últimas dos propiedades de la escena: la información de la localización del audio, que viene dada en el mapa de sonido (AEM, Figura 3.1), y el sesgo ECB (por el que los usuarios centran la atención en el ecuador de la escena, Figura 1.2), explicado en el Apartado 1.

Si se desea más detalle de alguno de los puntos explicados en este apartado, se recomienda leer el artículo original [2].

4.2. Funciones de pérdida

Tanto en el proceso de entrenamiento como en el de test de un modelo, es crucial elegir correctamente qué función de pérdida utilizar, ya que de ella puede depender su

[1] `https://www.esri.com/arcgis-blog/products/product/mapping/tissots-indicatrix-helps-illustrate-map-projection-distortion/`

rendimiento final. Durante este trabajo se han evaluado y analizado diversas funciones de error, y en este apartado se van a explicar las características de las más importantes. La implementación en Python de las funciones de error se encuentran en un repositorio público[2], o vienen incluidas en el paquete Pytorch.

Las funciones aquí explicadas se van a utilizar para entrenar diferentes modelos, y las funciones CC (Apartado 4.2.2) y KL (Apartado 4.2.3) se utilizarán en la etapa de evaluación de los mismos (Apartado 5).

4.2.1. BCE

La entropía binaria cruzada (*Binary Cross-Entropy*, BCE) entre dos distribuciones P y Q representa el número medio de bits necesarios para identificar un evento, dado un esquema de distribución optimizado en la distribución P, en vez de en la verdadera distribución Q. La aplicación de esta teoría a una función de error se traduce en medir el error de una reconstrucción de un modelo. En la Ecuación 4.1 se puede encontrar cómo se ha implementado esta función de error en Pytorch, donde N representa el *batch size* (número de muestras con las que se trabaja a la vez. Normalmente se traduce como "tamaño del lote").

$$l(x, y) = L = \{l_1, ..., l_N\}^T, l_n = -w_n[y_n \cdot logx_n + (1 - y_n) \cdot log(1 - x_n)] \tag{4.1}$$

Esta métrica devuelve valores en el rango [0, 1], representando el 0 una nula entropía, y el 1 una entropía máxima. Por ende, se buscará obtener valores de BCE lo más próximos a 0 posibles. En la documentación oficial de Pytorch[3] se puede encontrar más detalle de la implementación de esta función de error.

4.2.2. CC

El coeficiente de correlación de Pearson (también llamado *Pearson Correlation Coefficient*, CC) es una medida ampliamente utilizada en el campo de la estadística. Muestra la correlación linear entre dos conjuntos de datos, representada por valores en el rango [-1, 1] donde:

- -1 representa que los datos están perfecta e inversamente correlacionados.
- 0 representa que no hay correlación entre los datos.
- 1 representa que los datos están perfecta y directamente correlacionados.

[2] `https://github.com/cvzoya/saliency/tree/master/code_forMetrics`
[3] `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`

Se calcula como el ratio entre las covarianzas de los dos conjuntos y el producto de sus desviaciones típicas, lo cual explica el rango de valores que puede tomar este coeficiente. Aunque sea una función fácil de calcular, tiene una clara desventaja, y es que solo puede representar correlaciones lineales, ignorando así otros tipos de correlaciones más complejas.

En el trabajo, se buscará que el valor CC entre la saliencia *ground truth* y la saliencia que prediga un determinado modelo sea lo más positivo posible.

4.2.3. KLdiv

También se puede calcular la similitud entre dos conjuntos de datos dándole la vuelta al razonamiento: calculando cuál es la divergencia entre esos conjuntos. Para ello una función comúnmente usada en el campo del aprendizaje automático es la divergencia de Kullback-Leibler (*Kullback-Leibler Divergence*, KLdiv o KL).

Esta medida cuantifica cuánto difieren entre sí dos distribuciones de probabilidad P y Q, y se calcula como la probabilidad de un evento en P multiplicado por el logaritmo de la probabilidad de ese evento en Q dado el mismo evento en P. Traduciendo esto a calcular la divergencia entre dos matrices (o *tensores*[4] de Pytorch) que representan el *ground truth* y el resultado predicho, se obtiene la fórmula[5] de la Ecuación 4.2.

$$L(y_{pred}, y_{true}) = y_{true} \cdot log\frac{y_{true}}{y_{pred}} \tag{4.2}$$

Intuitivamente se puede ver que si la probabilidad de un evento es grande en P pero pequeña en Q (o viceversa), entonces el KLdiv será grande, representando que hay una gran divergencia entre las distribuciones.

Así, un valor de KLdiv = 0 representaría que las distribuciones P y Q son idénticas, y cuanto mayor sea el valor de la métrica, más divergencia habrá entre las distribuciones. Por lo tanto, se buscará que el KLdiv entre la saliencia *ground truth* y la predicha sea lo más cercana a 0 posible.

Es importante destacar que, debido a la forma en la que se calcula el KLdiv, el orden en el que se introducen los parámetros importa, ya que $KLdiv(P, Q) \neq KLdiv(Q, P)$, y de hecho estos valores normalmente no coincidirán.

4.2.4. NSS

Tal y como se argumenta en Bylinskii *et al.*[5], la saliencia de *scanpaths* normalizada (*Normalized Scanpath Saliency*, NSS) es una de las métricas más utilizadas y

[4] `https://pytorch.org/docs/stable/tensors.html`
[5] `https://pytorch.org/docs/stable/generated/torch.nn.KLDivLoss.html`

recomendadas en aplicaciones como las de este trabajo.

Esta es una medida que representa la media de los valores de los mapas de saliencia predichos en los puntos de fijación, que en otras palabras representa hasta qué punto las saliencia predicha coincide con la real. Así, un valor de 0 de NSS representará que la media de los valores de los mapas de saliencia predichos es 0, lo cual significa que los puntos predichos no coinciden con el *ground truth*. Por lo tanto, se buscará que el valor de NSS sea lo más grande posible.

En este trabajo se presenta una fórmula para calcular el NSS, la cual se encuentra en la Ecuación 4.3, y coincide con la implementación en Python que se va a utilizar.

$$\begin{aligned} NSS(P, Q^B) &= \frac{1}{N} \sum_i \bar{P}_i \times Q_i^B \\ where\ N = \sum_i Q_i^B\ and\ \bar{P} &= \frac{P - \mu(P)}{\sigma(P)} \end{aligned} \tag{4.3}$$

4.3. Modelo original

Tal y como se comentó en el Apartado 3.2, el conjunto de datos que se proporciona en Xu *et al.* [14] es el que se utilizó para entrenar el modelo original que se presenta en Chao *et al.* [2].

En el conjunto de datos utilizado había una gran cantidad de ODVs los cuales tenían o bien música de fondo, o un narrador hablando de la escena. Estos vídeos no son útiles para entrenar el modelo, ya que lo que se buscan son vídeos cuya saliencia de los usuarios dependa de los sonidos que se están produciendo dentro de la escena. Por lo tanto, para el entrenamiento del modelo original se seleccionaron a mano 27 vídeos que cumpliesen esta última condición. La función de error que se ha utilizado para el entrenamiento de este modelo es la KLdiv, la cual se ha explicado en el Apartado 4.2.3. Por último, se ha utilizado el optimizador *Adam* de la librería Pytorch[6], introduciéndole los siguientes hiperparámetros[7]:

- Se ha entrenado el modelo durante 20 épocas.

- Se ha utilizado un factor de aprendizaje (o *learning rate*, lr) de $1e-5$ (el *lr* establece cómo de rápido cambian los valores del modelo que se está entrenando. Un *lr* muy pequeño puede provocar que se tarde mucho en llegar al modelo óptimo, mientras que uno muy grande podría ocasionar que el modelo vaya

[6] `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam`

[7] En el campo de las redes neuronales, los hiperparámetros son valores que configuran cómo se va a entrenar un modelo.

cambiando drásticamente durante el entrenamiento, pero nunca se consigan los valores óptimos).

- Se ha definido un *batch size* de tamaño 6.

Así pues, se va a utilizar esta configuración como punto de partida, y más adelante se irán modificando diferentes partes de la misma, evaluando las consecuencias de dichos cambios.

Capítulo 5

Evaluación y análisis de distintos modelos

En este apartado se presentan las modificaciones estudiadas, junto a los resultados que se han obtenido a lo largo del trabajo, y se van a comentar debidamente como paso previo a sacar las conclusiones de los mismos.

Cabe recalcar que la finalidad de este trabajo no es obtener un modelo que mejore en prestaciones a los existentes en el estado del arte, sino que se trata de estudiar e implementar varios modelos, con el objetivo de analizar el impacto de diferentes factores que influyen en la generación de un modelo que predice la atención de los usuarios.

5.1. Modificaciones estudiadas

Se van a presentar los diferentes modelos que se han implementado, tomando como punto de partida el modelo original, y modificándolo para comprobar qué influencia tienen algunos aspectos a la hora de entrenar un modelo de predicción de la atención.

Por los motivos expuestos al comienzo de el Apartado 4.3, para el entrenamiento de los nuevos modelos se ha utilizado un subconjunto[1] del conjunto de datos de 208 vídeos [14] que se utiliza para el proceso de entrenamiento en el artículo original [2]. Así, se han seleccionado un total de 31 vídeos, los cuales tienen las características necesarias para utilizarlos en el entrenamiento de un modelo (los vídeos que contienen un ODV con el audio original de la escena).

5.1.1. Estudio del número de épocas de entrenamiento

Antes de entrenar ningún modelo, se ha hecho un estudio para evaluar el tiempo óptimo para entrenar con los datos disponibles. Se va a estudiar una cifra entre los

[1]En concreto se han utilizado los vídeos número 2, 3, 5, 19, 20, 25, 28, 33, 40, 54, 58, 59, 65, 66, 67, 68, 70, 72, 75, 80, 90, 94, 95, 97, 105, 110, 117, 129, 133, 136 y 142

valores de 10 y 20 épocas, ya que se busca que el modelo esté lo suficientemente entrenado, pero atendiendo a las restricciones temporales.

Así, se han entrenado varios modelos utilizando la función de pérdida BCE (Apartado 4.2.1), y se han utilizado las funciones CC (Apartado 4.2.2) y KLdiv (Apartado 4.2.3) para evaluar su desempeño. El resultado de algunos de ellos se puede observar en la Tabla 5.1. A lo largo de este apartado se utilizarán estas mismas funciones para evaluar el rendimiento de los diferentes modelos probados.

epochs\loss	CC ↑	KL ↓
10 epochs	0.630	8.613
14 epochs	0.591	8.603
19 epochs	0.589	8.687

Tabla 5.1: Evaluación de diferentes modelos atendiendo al número de épocas. Las flechas indican cuándo una métrica mayor es mejor (↑), o cuándo una métrica menor es mejor (↓).

Debido a que el modelo entrenado con BCE durante 10 épocas es el que mejor valor de CC tiene, y que el entrenado durante 14 épocas presenta el mejor valor de KL, se ha establecido 12 épocas como un compromiso suficiente para entrenar los modelos en el contexto en el que se está trabajando.

Por lo tanto, se puede asumir que los modelos presentados a partir de este punto están entrenados durante 12 épocas, hasta que no se indique lo contrario.

5.1.2. Estudio de las funciones de pérdida

Una vez estudiado el número de épocas sobre el que entrenar un modelo, en este apartado se van a analizar los resultados de diferentes modelos, los cuales se diferencian únicamente en la función de error con la que se han entrenado. Para ello, se van a estudiar las funciones de error CC, BCE, KL y NSS, todas ellas explicadas en el Apartado 4.2 de esta misma memoria.

Se ha entrenado un modelo diferente para cada función de error a estudiar, y de nuevo se han utilizado las funciones CC y KL para evaluar el desempeño de los modelos entrenados, haciendo la evaluación con los datos de *test*. El resultado de este proceso se puede observar en la Tabla[2] 5.2 en la que, para un análisis más detallado, se desglosan los valores para cada uno de los vídeos del conjunto de datos de *test*.

En el cuerpo de la tabla se pueden ver todos los datos para cada uno de los vídeos, y mirando directamente en la última fila se puede ver más fácilmente cuál es el desempeño medio de cada modelo con los datos de *test*.

[2]Generada con ayuda de la herramienta `https://www.tablesgenerator.com/`.

Model	BCE		CC		KL		NSS	
Vídeo	**CC ↑**	**KL ↓**	**CC ↑**	**KL ↓**	**CC ↑**	**KL ↓**	**CC ↑**	**KL ↓**
1 Choir	0.618	8.507	0.628	8.930	0.639	8.552	0.344	9.276
7 Interview	0.469	8.749	0.607	8.931	0.626	8.447	0.307	9.325
12 Ambulance	0.546	8.531	0.551	8.652	0.583	8.426	0.287	9.341
4 Asian	0.551	8.321	0.572	9.032	0.523	8.425	0.244	9.285
10 Gym	0.293	9.079	0.316	8.928	0.289	9.160	0.246	9.424
2 Band	0.563	8.854	0.621	9.065	0.638	8.561	0.356	9.301
5 Train	0.585	8.945	0.623	9.579	0.578	8.767	0.327	9.384
6 London	0.616	9.175	0.606	9.289	0.609	9.219	0.373	9.367
8 Fighter	0.655	8.124	0.638	8.239	0.633	8.139	0.259	9.313
3 Orchestra	0.634	9.011	0.567	9.802	0.661	8.669	0.328	9.381
11 Feast	0.583	8.454	0.589	8.502	0.550	8.587	0.336	9.336
9 Factory	0.656	8.721	0.573	9.249	0.656	8.976	0.338	9.372
MEDIA	**0.564**	**8.706**	**0.574**	**9.017**	**0.582**	**8.661**	**0.312**	**9.342**

Tabla 5.2: Desempeño de cada uno de los modelos sobre los diferentes vídeos de *test*. La primera fila de la tabla presenta la función de error que se ha utilizado para entrenar cada modelo. En la segunda fila se encuentran cuatro parejas de columnas con las funciones CC y KL utilizadas para la evaluación de cada uno de los cuatro modelos. En la primera columna de las siguientes filas se encuentra el identificador de los diferentes vídeos utilizados, y en la última fila se encuentra la media de los valores de CC y KL para cada modelo. Se representan en azul los mejores valores para cada vídeo y como media.

Llama la atención los resultados inferiores que obtienen todos los modelos con el vídeo *10 Gym*. Analizando el vídeo, este mal resultado se puede deber a que en la escena hay una gran cantidad de elementos diferentes, lo cual se ve potenciado con el hecho de que también hay un espejo, tratándose así de un vídeo complejo de analizar *a priori* por los modelos con los que se está trabajando. Más adelante en esta memoria se facilitará el acceso a los vídeos del conjunto de datos de *test*, por si se quieren visualizar dichos vídeos de primera mano.

Observando las medias de los modelos, se puede ver que los entrenados con las funciones BCE y CC tienen unas medias aceptables y similares, debido a que son las dos funciones más genéricas, y cuyo cálculo es más simple que las otras dos.

Por otra parte, el modelo que mejor desempeño tiene es el entrenado utilizando la función de pérdida KL, lo cual podría ser el resultado esperable, ya que el modelo presentado en el artículo original [2] también se entrenó con esta función de error.

Por último, el modelo entrenado con la función NSS tiene un rendimiento malo constante en todos los vídeos de *test*. Esto se puede deber a que, por la forma en la que se calcula el NSS, éste está más enfocado a ser utilizado durante la etapa de *test* que durante el entrenamiento.

Esta función calcula la media de los valores del mapa de saliencia predicho en cada uno de los puntos del mapa de fijación en un fotograma determinado. Por una parte en el entrenamiento no se tienen mapas de fijación, sino de saliencia (la diferencia es que los mapas de fijación representan de forma discreta los puntos concretos de la escena que han observado los usuarios reales, mientras que los mapas de saliencia representan qué zonas de la escena tienen mayor probabilidad de ser observadas). Por otra parte, dependiendo de cómo se inicialice el modelo, si por ejemplo la primera predicción es que todas las zonas de la escena se van a observar con probabilidad 1, al calcular la función NSS con ese mapa de saliencia, se obtendrá el valor óptimo, y por lo tanto el modelo entenderá que ya no hace falta converger más (esto podría justificar la apariencia de los mapas de saliencia de la Figura 5.1).

Por estos factores, era de esperar que el modelo entrenado utilizando la función NSS fuera el que obtuviera peores resultados de los cuatro analizados.

Además, se ha creado una matriz de imágenes con el objetivo de ver las diferencias en los comportamientos de los modelos de una manera visual. El formato y una explicación más detallada de estas matrices de figuras se pueden encontrar en el Apartado 5.3.2 de la memoria. Los fotogramas están sacados de los vídeos de la carpeta[3] comparativa de los modelos.

En la primera tabla (Figura 5.1), se compara el comportamiento entre tres funciones de error, para comprobar si el resultado es coherente con lo obtenido en el Apartado 5.1.2.

Para ello se han seleccionado tres de las funciones de error estudiadas:

- La función CC, que es una de las dos que obtenían resultados aceptables (junto a la función BCE).

- La función KL, la cual debería ser la que mejores predicciones obtuviera, ya que eso es lo que se podía observar en la Tabla 5.2.

- La función NSS, la cual obtenía unas métricas malas en la Tabla 5.2.

Como se puede observar, los resultados son los esperados: por una parte, la función CC genera unas predicciones que, si bien se asemejan claramente al *ground truth*, ya que las zonas con mayores valores en los mapas de CC coinciden con los del *ground truth*, están demasiado difuminadas, lo cual explicaría los valores obtenidos en la Tabla 5.2.

[3] `https://drive.google.com/drive/folders/1Y3OecEnuf1eDxkR6WJZnWRZU8ZAXITDD?usp=sharing`

Por otra parte, las predicciones generadas con el modelo entrenado con KL también están concentradas en los mismos sitios que en el *ground truth*, pero éstas están más definidas que las del modelo con CC. Por otra parte, también se puede ver que la predicción sobre el vídeo *Gym* es menos precisa que la de CC, lo cual de nuevo es coherente con las métricas de la Tabla 5.2.

Por último, se pueden observar cómo las predicciones del modelo entrenado con NSS no tienen ningún valor, ya que parece que representan simplemente el ECB. Esto se podría deber a la hipótesis presentada en el Apartado 5.1.2.

Como conclusión se ha visto que, en el contexto de este trabajo, la función de pérdida que mejor funciona a la hora de entrenar un modelo es la KL.

5.1.3. Estudio del impacto de la presencia de sonido

Debido a que una parte considerable del trabajo se basa en tener en cuenta el sonido en la escena para intentar obtener mejores predicciones, se ha considerado interesante analizar el comportamiento de un modelo que no utilice la información sonora de la escena para calcular los mapas de saliencia.

Para ello, ha sido necesario modificar ligeramente la estructura del modelo, ya que como se puede ver en la arquitectura del mismo (Figura 4.1), hay una etapa que concatena la información sonora con la visual de la escena. Por lo tanto, para eliminar la influencia del sonido, se ha prescindido de esta concatenación. Por otra parte, a la hora de realizar las predicciones del modelo, también se ha de eliminar la información sonora, por lo que también ha sido necesario modificar el proceso de predicción de mapas de saliencia con este objetivo.

Como se puede observar en la Tabla 5.3, el modelo entrenado sin sonido obtiene en multitud de ocasiones el mejor rendimiento, y es el que mejor valor de CC medio obtiene de los tres. Esto se podría deber a que al tener menos información que aprender, al entrenar durante 12 épocas todos los modelos, el entrenado sin sonido haya convergido más que el que lo utilizaba. Esta hipótesis se estudiará en mayor profundidad más adelante.

5.1.4. Estudio del impacto del ECB

Tal y como se comentó en el Apartado 1, el sesgo de atención en el ecuador (ECB) es un sesgo que tienen los seres humanos por el cual basan su atención en las zonas de la escena más cercanas al ecuador de la misma. En los modelos estudiados hasta ahora, se ha estado utilizando el conocimiento de la existencia de este sesgo, utilizándolo para crear predicciones más realistas.

En este apartado se quiere probar cual es el efecto de no tener en cuenta el ECB a la hora de generar un nuevo modelo. De nuevo, tal y como se puede observar en la arquitectura del modelo con el que se ha estado trabajando (Figura 4.1), la información del ECB se introduce en la etapa de fusión entre la información de la escena y el AEM. Por lo tanto, para eliminar la influencia del ECB, únicamente va a haber que deshacerse de esta fusión del ECB. Respecto a la generación de predicciones, también va a haber que eliminar la parte donde se introduce el sesgo de ECB en este proceso.

Una vez entrenados tanto este modelo como el anterior, se ha seguido la misma metodología que en el Apartado 5.1.2 para evaluar los modelos obtenidos. El resultado de este proceso de evaluación se puede observar en la Tabla 5.3.

Model	**NO_SOUND**		**NO_ECB**		***KL***	
Vídeo	**CC ↑**	**KL ↓**	**CC ↑**	**KL ↓**	***CC ↑***	***KL ↓***
1 Choir	0.668	8.422	0.598	8.489	*0.639*	*8.552*
7 Interview	0.649	8.547	0.702	8.268	*0.626*	*8.447*
12 Ambulance	0.622	8.306	0.663	8.239	*0.583*	*8.426*
4 Asian	0.582	8.325	0.598	8.504	*0.523*	*8.425*
10 Gym	0.257	9.138	0.348	8.831	*0.289*	*9.160*
2 Band	0.662	8.456	0.654	8.473	*0.638*	*8.561*
5 Train	0.643	8.629	0.645	8.561	*0.578*	*8.767*
6 London	0.636	9.026	0.621	8.920	*0.609*	*9.219*
8 Fighter	0.708	8.041	0.594	8.224	*0.633*	*8.139*
3 Orchestra	0.649	9.082	0.525	8.945	*0.661*	*8.669*
11 Feast	0.614	8.396	0.579	8.511	*0.550*	*8.587*
9 Factory	0.619	8.773	0.677	8.771	*0.656*	*8.976*
MEDIA	**0.609**	**8.595**	**0.600**	**8.561**	***0.582***	***8.661***

Tabla 5.3: Tabla que contiene el desempeño de los dos nuevos modelos sobre los vídeos de test, además del desempeño del modelo entrenado con la función de error KL, que ha sido el que se ha tomado como punto de partida (este último tiene los datos en cursiva para resaltar que no son datos nuevos). El formato de esta tabla es el mismo que el de la Tabla 5.2.

Observando y analizando los valores de CC y KL de los modelos que se encuentran en la Tabla 5.3, se puede afirmar que el modelo entrenado utilizando el sonido y el ECB (la columna KL), es el que peor se comporta de los tres. Esto podría resultar contra intuitivo, ya que modelos que toman una versión simplificada del problema (uno sin tener en cuenta el sonido y otro sin tener en cuenta el ECB), están obteniendo mejores resultados que el modelo más completo.

Este resultado se podría deber a que los tres modelos se están entrenando durante 12 épocas, y por lo tanto los modelos NO_SOUND y NO_ECB al tener menos información que aprender, convergerán más rápidamente que el modelo KL. La validez de esta

hipótesis se va a comprobar en el siguiente apartado.

5.1.5. Estudio de convergencia con KL

El número de épocas que se ha estado utilizando hasta el momento fue el que se calculó en el Apartado 5.1.1. Como se ha comentado al final del apartado anterior, es de esperar que en ese número de épocas no converjan igual todos los modelos.

Para comprobar si el modelo entrenado utilizando la función de error KL y teniendo en cuenta tanto la información sonora como el sesgo ECB se puede comportar mejor, en este apartado se va a entrenar dicho modelo durante 20 épocas.

El resultado de la evaluación se puede comprobar en la Tabla 5.4.

Model	**KL (20 ep)**		***NO_SOUND***		***NO_ECB***	
Vídeo	**CC ↑**	**KL ↓**	***CC ↑***	***KL ↓***	***CC ↑***	***KL ↓***
1 Choir	0.603	8.579	*0.668*	*8.422*	*0.598*	*8.489*
7 Interview	0.675	**8.576**	*0.649*	*8.547*	*0.702*	*8.268*
12 Ambulance	0.552	8.611	*0.622*	*8.306*	*0.663*	*8.239*
4 Asian	0.777	8.383	*0.582*	*8.325*	*0.598*	*8.504*
2 Band	0.721	7.542	*0.662*	*8.456*	*0.654*	*8.473*
5 Train	0.650	8.583	*0.643*	*8.629*	*0.645*	*8.561*
6 London	0.651	9.107	*0.636*	*9.026*	*0.621*	*8.920*
8 Fighter	0.485	8.494	*0.708*	*8.041*	*0.594*	*8.224*
3 Orchestra	0.665	9.105	*0.649*	*9.082*	*0.525*	*8.945*
11 Feast	0.556	8.619	*0.614*	*8.396*	*0.579*	*8.511*
9 Factory	0.671	8.950	*0.619*	*8.773*	*0.677*	*8.771*
MEDIA	**0.637**	**8.595**	***0.641***	***8.546***	***0.623***	***8.537***

Tabla 5.4: Tabla que contiene el desempeño del modelo entrenado durante 20 épocas sobre los vídeos de test, además de los desempeños de los anteriores dos modelos (estos tienen los datos en cursiva para resaltar que no son datos nuevos). El formato de esta tabla es el mismo que el de la Tabla 5.2. Se ha suprimido la evaluación sobre el vídeo *10 Gym*, ya que al ser un vídeo tan complejo todos los modelos obtenían malos resultados, y esto desvirtualizaba ligeramente las medias generales.

Como se puede observar en la tabla, si bien el nuevo modelo entrenado durante 20 épocas no obtiene mejores medias que los anteriores, se puede observar que las medias generales son prácticamente iguales entre los tres modelos. También se puede ver cómo los modelos predicen mejor una cantidad similar de vídeos, lo cual es coherente con que tengan unas medias similares. De estos datos se desprende que quizás la información sonora y el sesgo de ECB no son determinantes a la hora de entrenar este modelo, al menos en el escenario en el que se está trabajando.

Al igual que en el Apartado 5.1.2, se ha creado una matriz de fotogramas para comprobar visualmente las diferencias que hay entre las predicciones de los últimos

tres modelos presentados (sin sonido, sin ECB y uno con mayor número de épocas). De nuevo, en el Apartado 5.3.2 se puede encontrar una explicación más detallada del contenido de esta matriz.

Así, en la segunda tabla (Figura 5.2), se puede observar cómo se comportan las tres modificaciones estudiadas:

- El modelo que no utiliza el sonido de la escena durante el entrenamiento ni durante la predicción de las saliencias, explicado en el Apartado 5.1.3.
- El modelo que no hace uso del sesgo ECB para generar mejores predicciones, explicado en el Apartado 5.1.4.
- El modelo entrenado con la función de error KL durante un número mayor de épocas que el resto, explicado en el Apartado 5.1.5.

Las columnas de las matrices representan los fotogramas sobre los que se va a comparar el rendimiento de los modelos. El identificador de las columnas tiene la forma V«num_video»F«num_frame», donde «num_video» representa el identificador numérico de cierto vídeo. Estos identificadores se pueden ver por ejemplo en tablas como la 5.2 al lado del identificador textual. Así, el identificador numérico del vídeo *Orchestra* es el 3.

Por otra parte, «num_frame» representa simplemente el número concreto del fotograma del vídeo que se está utilizando. Así, el fotograma 100 del vídeo *Orchestra* tendrá el identificador V3F100.

En esta segunda tabla se pueden observar las consecuencias de modificar los modelos de las formas en las que se han modificado: por una parte, eliminando la información sonora de la escena, se puede ver que en los fotogramas donde el sonido es una parte importante de la misma, este primer modelo obtiene predicciones alejadas del *ground truth*. El ejemplo más claro es el del fotograma V9F50, ya que en esa escena el movimiento es mínimo, y la información de que el hombre está hablando es crucial para predecir el comportamiento de los usuarios.

Respecto al modelo entrenado sin tener en cuenta el ECB, se puede ver que es más robusto que el anterior, pero si se compara con las predicciones que obtenía el modelo entrenado con KL de la Figura 5.1, se puede ver que el modelo sin ECB obtiene predicciones algo más difuminadas, lo cual, atendiendo a los resultados de la Tabla 5.3, parece ser beneficioso.

Por último, el modelo entrenado con la función KL durante 20 épocas, también obtiene unas predicciones correctas, las cuales tienen en cuenta la información sonora de la escena, además del ECB. Estas características se pueden apreciar por ejemplo en

el fotograma V9F50, en el cual el modelo KL está más centrado en el hombre que está hablando que los otros. Lo mismo ocurre en el fotograma V1F150, en el que también se predice una atención centrada en la mujer de la derecha, que es la que está cantando en ese momento.

Se puede apreciar también una mejora sobre las predicciones del modelo entrenado con KL durante menos épocas (Figura 5.1), ya que intuitivamente las predicciones obtenidas con el nuevo modelo se acercan más al comportamiento que se esperaría que tuviera un usuario real, además que son más parecidas al *ground truth* con el que se está trabajando.

5.2. Resultados cuantitativos

Los resultados cuantitativos obtenidos en este trabajo son principalmente las tablas que se han ido presentando a lo largo de este apartado. En estas tablas se han ido mostrando los valores de las métricas CC y KL para evaluar cada uno de los modelos, las cuales se pueden utilizar para comprobar individualmente cómo se comporta un modelo concreto en un vídeo concreto. Estas tablas se han obtenido utilizando el conjunto de datos de *test*, y por lo tanto los resultados obtenidos dependen de este conjunto de datos. Los datos se han calculado utilizando un programa llamado *eval_multiple_videos.m* desarrollado en Matlab, el cual se explica brevemente en el Anexo A.

5.3. Resultados visuales

Los resultados visuales más importantes que se han conseguido como resultado de este trabajo, son una serie de vídeos que son el resultado de colorear los vídeos originales con los mapas de saliencia predichos, y unas tablas que representan visualmente las diferencias entre los modelos entrenados.

5.3.1. Vídeos generados

Para generar los vídeos se han desarrollado una serie de programas en el lenguaje Python, los cuales cogen los mapas de saliencia generados por los modelos entrenados y el vídeo original, y colorea el vídeo atendiendo a estos mapas de saliencia. Estos programas son *heatmaps.py, assembling.py, gen_videos.py y copy_sound.py*, los cuales se detallan más profundamente en el Anexo A.

Los vídeos resultantes se pueden encontrar en el siguiente enlace: `https://drive.google.com/drive/folders/1Y3OecEnuf1eDxkR6WJZnWRZU8ZAXITDD?usp=sharing`. En esta carpeta hay una serie de directorios, uno para cada modelo desarrollado, y en

cada uno de ellos se pueden encontrar todos los vídeos del conjunto de test, coloreados con las predicciones del modelo correspondiente (además está el directorio *Original*, el cual contiene los resultados obtenidos con el modelo entrenado en Chao *et al.* [2]).

Por último también está la carpeta *videos_raw*, la cual contiene los vídeos del conjunto de datos de *test* (con los identificadores originales), los cuales pueden ser de interés para ver qué vídeos se han utilizado para predecir las saliencias sin editar.

5.3.2. Tablas de fotogramas

Otro resultado visual que se ha generado son las tablas de fotogramas sobre las que se ha estado hablando en este apartado, las cuales contienen las predicciones de ciertos modelos a ciertos fotogramas de vídeos. Estas tablas sirven para visualizar rápidamente cuales son las diferencias entre los vídeos o los modelos que se están comparando en ellas.

El formato de las columnas de estas tablas es el que se ha comentado en el Apartado 5.1.5. Respecto a las filas, la primera contiene el fotograma sin ser modificado, para poder ver el fotograma que se va a predecir. Las siguientes filas estarán a su vez subdivididas en dos: la fila de abajo, donde se encuentran los mapas de saliencia en escala de grises, y la fila de arriba, en la cual se ha coloreado el fotograma con el mapa de saliencia correspondiente.

Así, la segunda fila contiene el *ground truth* del fotograma, generado tal y como se ha explicado en el Apartado 3.3.2. Estas dos primeras filas las van a compartir las dos tablas, y las siguientes tres filas van a contener los datos de los modelos que se desean comparar. Para las dos tablas se han seleccionado unos determinados fotogramas de cuatro vídeos del conjunto de *test*:

- V1F150: Se ha seleccionado el primer vídeo, ya que se ha considerado que es un vídeo estándar, y en concreto se ha seleccionado un fotograma en el que está una mujer bajando las escaleras en la zona derecha, y de esta forma se puede ver si, en efecto, los modelos detectan este movimiento.

- V12F130: Este vídeo es *a priori* más complejo que el primero, pero en él ocurre un evento que debería atraer la atención de los usuarios. Por lo tanto, se ha escogido el fotograma justo en el que pasa una ambulancia pasando por la escena.

- V10F360: Este fotograma pertenece al vídeo *Gym*, el cual se ha comentado en el apartado 5.1.2 que parece ser un vídeo complejo de predecir, y por lo tanto se ha elegido para ver qué saliencias generan los diferentes modelos.

– V9F50: Este último vídeo se ha seleccionado ya que es una escena diferente a las otras tres que se han seleccionado. En este caso se representa una conversación, y se ha seleccionado un fotograma en el que el hombre está hablando, con el objetivo de ver si, en efecto, la atención se centra en él.

[4] https://drive.google.com/drive/folders/1Y3OecEnuf1eDxkR6WJZnWRZU8ZAXITDD?usp=sharing

[5] https://drive.google.com/drive/folders/1Y3OecEnuf1eDxkR6WJZnWRZU8ZAXITDD?usp=sharing

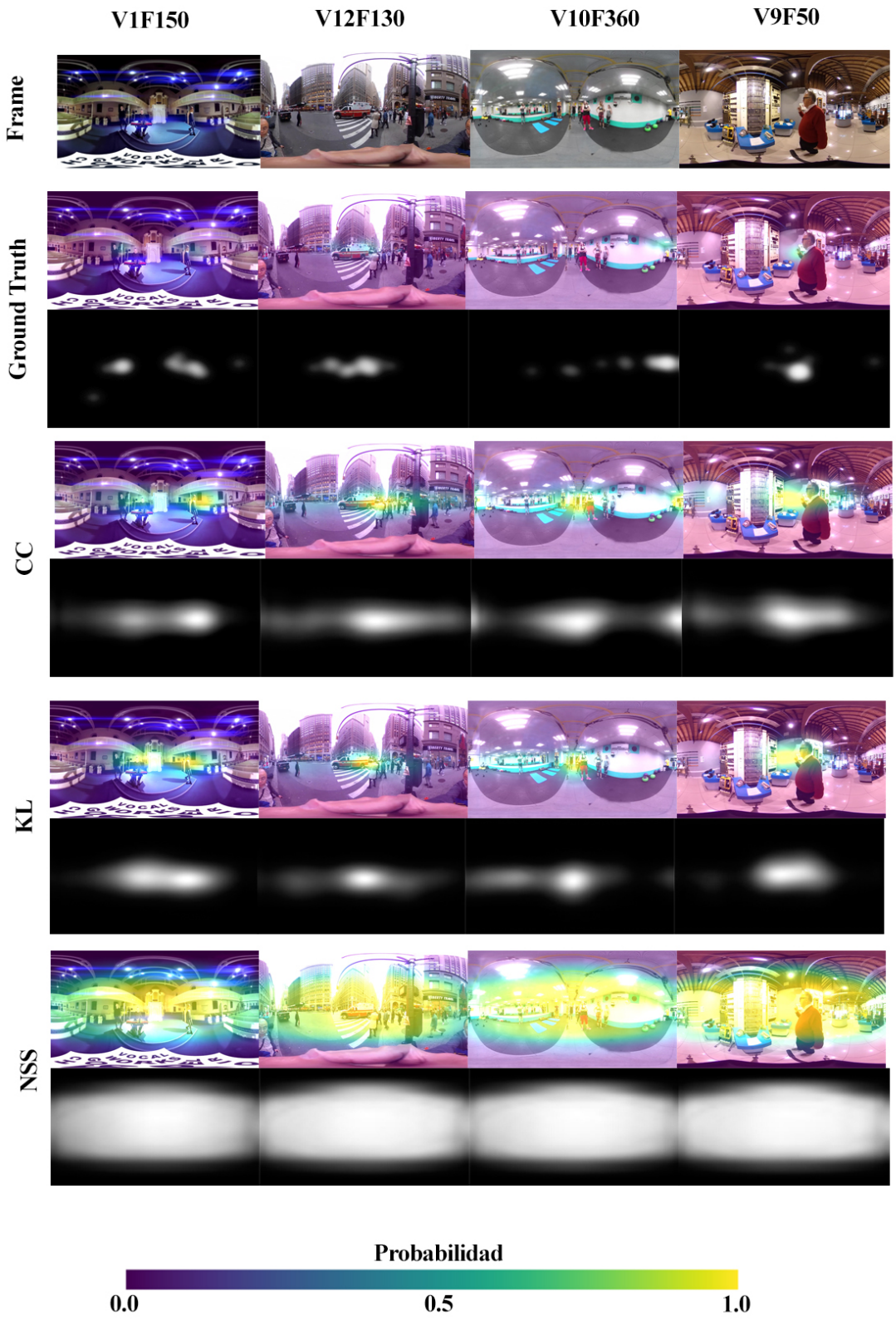

Figura 5.1: Tabla que compara el desempeño de tres funciones de error que son representativas de las estudiadas: La función CC, la KL y la NSS. El formato que sigue la tabla es el explicado en este mismo apartado. Imágenes obtenidas de los vídeos que se encuentran en la carpeta compartida de Drive[4].

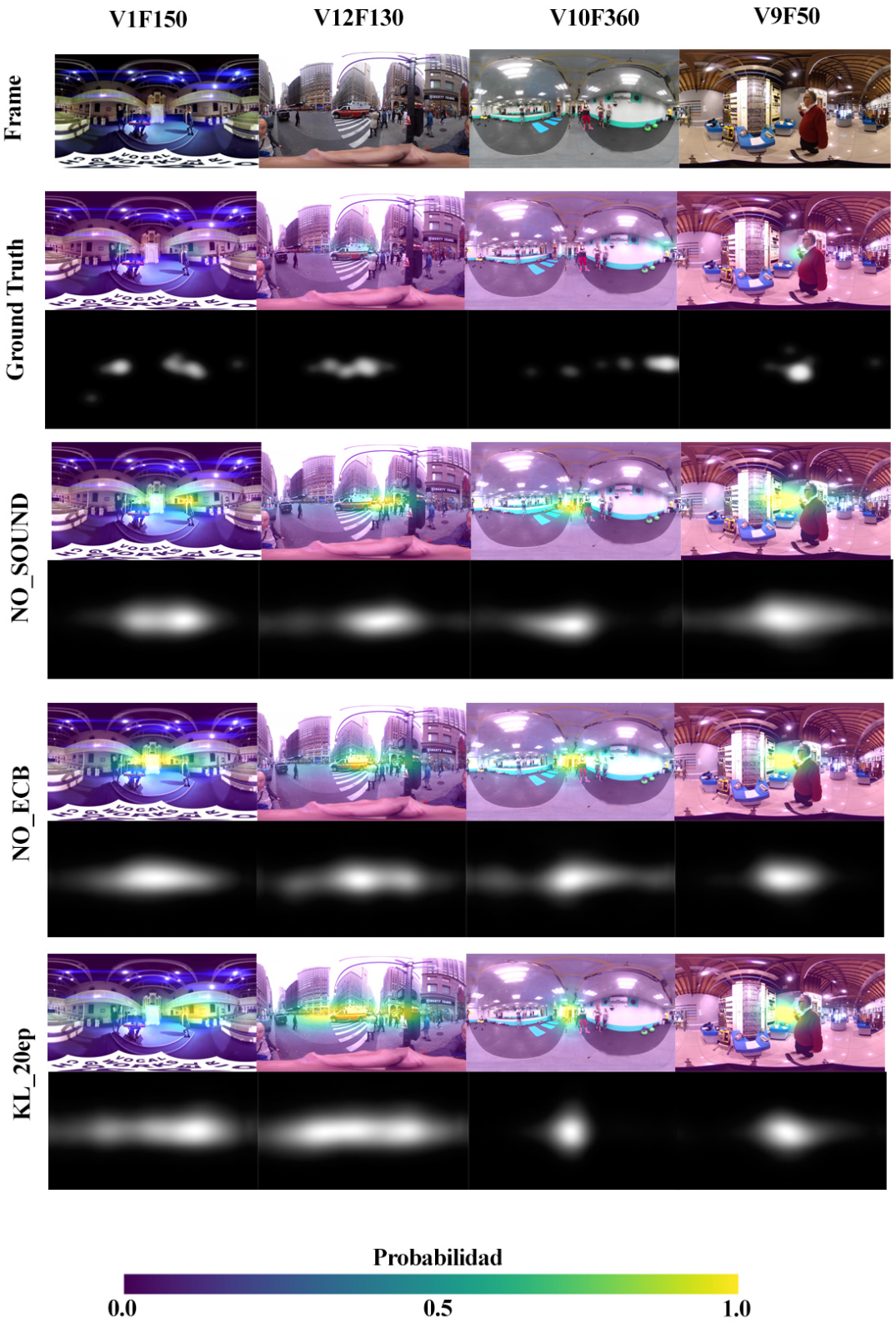

Figura 5.2: Tabla que compara el desempeño de las tres modificaciones presentadas en los apartados 5.1.3, 5.1.4 y 5.1.5. El formato es el mismo que el de la Tabla 5.1. Imágenes obtenidas de los vídeos que se encuentran en la carpeta compartida de Drive[5].

Capítulo 6

Conclusiones

En este trabajo se ha estudiado un modelo que predice la atención de un usuario en un entorno de realidad virtual, atendiendo tanto a la información visual como al sonido direccional de la escena. El objetivo es implementar una serie de modificaciones al modelo original, las cuales se consideran de interés, para comprobar qué efectos tenían aplicar ciertos cambios en diferentes etapas del modelo. Para evaluar las implicaciones de los cambios introducidos, se ha hecho uso de ciertas métricas ampliamente utilizadas en el estado del arte de la materia.

Respecto a las funciones de error utilizadas durante el entrenamiento, se ha comprobado que la función más adecuada para este propósito es la KL, la cual es por otra parte la que se utilizaba en el artículo que se ha tomado como referencia [2].

También se ha comprobado que, si existe una limitación en el número de épocas con las que se va a entrenar un modelo, se debería considerar la complejidad del modelo a entrenar, ya que puede que entrenando uno más simple se consigan mejores resultados, debido a que éste podría converger más rápidamente que un modelo más complejo.

Por último, en base a los resultados obtenidos en la Apartado 5.1, parece ser que en el escenario en el que se está trabajando, la información sonora y el uso del sesgo ECB puede que no sean determinantes a la hora de predecir la saliencia de una escena. Cabría esperar que con un conjunto de datos de entrenamiento con AEM reales (el conjunto que hemos utilizado para entrenar tiene únicamente audio *mono*, a partir del cual no se puede obtener un sonido direccional), la información sonora de la escena cobrase más importancia, y en ese caso quizás sí que sería interesante utilizarla en el proceso de predecir la saliencia.

Respecto al uso del ECB en los modelos, es posible que los usuarios estén aplicando ese sesgo inconscientemente, ocasionando que las fijaciones que se usan en la etapa de entrenamiento ya tienen ese sesgo, y por lo tanto el modelo que se entrene usando dichas fijaciones ya estará sesgado en ese sentido, con lo cual puede que no sea necesario implementar esa parte de forma manual.

6.1. Trabajo futuro y limitaciones

Ya que el objetivo del trabajo es probar diferentes alternativas para estudiar sus efectos sobre el modelo, se podrían probar multitud de variantes interesantes para seguir esta línea de estudio.

Una posible modificación a estudiar sería el cambiar la estructura interna del modelo, introduciendo por ejemplo más capas convolucionales, más capas de pool, o eliminando algunas de las existentes. Para implementar esta modificación habría que ser consciente de los cambios en las dimensiones de las capas internas, y no se podría asegurar que un cambio de este tipo mejorase el rendimiento del modelo actual.

También se podrían estudiar otras funciones de error diferentes de las estudiadas en el trabajo, o se podría estudiar el cambiar los hiperparámetros del entrenamiento del modelo, como el *batch size* o el lr.

Otro área que se podría estudiar sería el de los conjuntos de datos. El conjunto de datos utilizado no tiene información de AEM, y por lo tanto se podría esperar que se pudieran entrenar mejores modelos utilizando conjuntos de datos con esta información, o bien emulando estos AEM con sistemas como el *spatialaudiogen* [16]. El conjunto de datos también podría mejorar si tuviera más vídeos con los que entrenar, (ya que el utilizado tenía muchos vídeos inutilizables al no tener información sonora de la escena), o si se tuvieran más datos de fijaciones reales de cada vídeo, con lo cual se obtendrían *ground truths* lo más cercanos posibles al comportamiento real de los usuarios.

La principal limitación encontrada ha sido la exigencia de recursos del proceso de entrenamiento de modelos. Este entrenamiento consumía aproximadamente 12 GB de memoria de la GPU con la configuración utilizada, lo cual lo hacía inviable para ser ejecutado en el equipo personal utilizado, y había que recurrir a utilizar otros entornos con más poder computacional (tal y como se ha comentado en el apartado 1.3).

Otras limitaciones encontradas han sido algunos problemas con las versiones de ciertos paquetes de Python, las cuales han hecho imposible ejecutar ciertos entornos, y la dificultad de encontrar conjuntos de datos apropiados.

Capítulo 7

Bibliografía

[1] Vincent Sitzmann, Ana Serrano, Amy Pavel, Maneesh Agrawala, Diego Gutierrez, Belen Masia, and Gordon Wetzstein. How do people explore virtual environments? *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[2] F. Y. Chao, C. Ozcinar, L. Zhang, W. Hamidouche, O. Deforges, and A. Smolic. Towards audio-visual saliency prediction for omnidirectional video with spatial audio. In *2020 IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pages 355–358, 2020.

[3] Daniel Martin, Ana Serrano, and Belen Masia. Panoramic convolutions for 360° single-image saliency prediction. In *CVPR Workshop on Computer Vision for Augmented and Virtual Reality*, 2020.

[4] Daniel Martin, Ana Serrano, Alexander W Bergman, Gordon Wetzstein, and Belen Masia. Scangan360: A generative model of realistic scanpaths for 360 images. *IEEE Transactions on Visualization & Computer Graphics*, (01):1–1, 2022.

[5] Zoya Bylinskii, Tilke Judd, Aude Oliva, Antonio Torralba, and Frédo Durand. What do different evaluation metrics tell us about saliency models? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(3):740–757, 2019.

[6] Redes neuronales artificiales. `https://www.famaf.unc.edu.ar/~revm/digital24-3/redes.pdf`. Accedido por última vez el 26-04-2022.

[7] Curso de stanford sobre redes neuronales convolucionales. `https://cs230.stanford.edu/syllabus/`. Accedido por última vez el 27-04-2022.

[8] Redes neuronales convolucionales para reconocimiento visual. `https://cs231n.github.io/`. Accedido por última vez el 27-04-2022.

[9] F. Chao, L. Zhang, W. Hamidouche, and O. Deforges. Salgan360: Visual saliency prediction on 360 degree images with generative adversarial networks. In *2018 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, 2018.

[10] Cagri Ozcinar and Aljosa Smolic. Visual attention in omnidirectional video for virtual reality applications. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, 2018.

[11] Fang-Yi Chao, Cagri Ozcinar, Chen Wang, Emin Zerman, Lu Zhang, Wassim Hamidouche, Olivier Déforges, and Aljosa Smolic. Audio-visual perception of omnidirectional video for virtual reality applications. 06 2020.

[12] Belén Masiá, Javier Camon, Diego Gutierrez, and Ana Serrano. Influence of directional sound cues on users' exploration across 360° movie cuts. *IEEE Computer Graphics and Applications*, 41:64–75, 2021.

[13] F. Chao, C. Ozcinar, C. Wang, E. Zerman, L. Zhang, W. Hamidouche, O. Deforges, and A. Smolic. audio-visual perception of omnidirectional video for virtual reality applications. In *IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, 2020.

[14] Yanyu Xu, Yanbing Dong, Junru Wu, Zhengzhong Sun, Zhiru Shi, Jingyi Yu, and Shenghua Gao. Gaze prediction in dynamic 360° immersive videos. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5333–5342, 2018.

[15] Artículo sobre las distorsiones en formatos de vídeo. `https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/`.

[16] Timothy Langlois Pedro Morgado, Nuno Vasconcelos and Oliver Wang. Self-supervised generation of spatial audio for 360º video. In *Neural Information Processing Systems (NIPS)*, 2018.

[17] Zoya Bylinskii, Tilke Judd, Frédo Durand, Aude Oliva, and Antonio Torralba. Mit saliency benchmark. http://saliency.mit.edu/.

[18] Zoya Bylinskii, Tilke Judd, Aude Oliva, Antonio Torralba, and Frédo Durand. What do different evaluation metrics tell us about saliency models? *arXiv preprint arXiv:1604.03605*, 2016.

[19] Edurne Bernal Berdún, Daniel Martín Serrano, and Belén Masiá Corcoy. Modeling human visual behavior in dynamic 360º environments. 2022.

[20] Blanca Lasheras Hernández and Daniel Martín Serrano. Estudio de Sistemas Avanzados de Asistencia al Conductor (ADAS) en vehículos y propuesta de aplicación de técnicas de seguimiento de la mirada para su mejora. 2021.

[21] Javier Giménez Garcés, Belén Masiá Corcoy, and Daniel Martín Serrano. Estimación de profundidad a partir de una única imagen 360° mediante aprendizaje profundo. 2020.

[22] Javier Camón Julián, Ana Belén Serrano Pacheu, and Diego Gutiérrez Pérez. Edición en Realidad Virtual: influencia de técnicas sonoras en la continuidad narrativa. 2017.

[23] Jaime Ruiz-Borau Vizárraga and Ana Belén Serrano Pacheu. Edición cinematográfica y continuidad narrativa en realidad virtual. 2017.

[24] Explicación del funcionamiento de la función de error bce. `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`.

[25] Explicación del funcionamiento de la función de error cc. `https://www.scribbr.com/statistics/correlation-coefficient/`.

[26] Explicación de las funciones de error. `https://www.section.io/engineering-education/understanding-loss-functions-in-machine-learning/`.

[27] Implementación de varias funciones de error en matlab. `https://github.com/cvzoya/saliency/tree/master/code_forMetrics`.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Programas adicionales

A lo largo de las diferentes etapas del trabajo se han tenido que editar y crear diferentes programas (además de los principales), bien para crear una funcionalidad nueva, o bien para solucionar ciertos problemas (de versiones, de compatibilidad...) que tenían los sistemas ya implementados.

En este anexo se van a incluir los programas más importantes que se han creado a lo largo del trabajo necesarios para tareas concretas, las cuales se detallarán usando una descripción para cada uno de ellos.

Para un mayor detalle de estos programas, se han subido a un repositorio público de GitHub al que se puede acceder a través del enlace `https://github.com/SantiagoJN/TFG-Santiago`.

- convert.py: El objetivo final de este programa es traducir los datos de fijaciones que se encuentran normalmente en un fichero .csv con un formato determinado (incluido en el conjunto de datos correspondiente), y obtener al final los mapas de saliencia o de fijaciones que se usarán más adelante. Se van construyendo los diferentes mapas de fijaciones de cada fotograma, y para obtener los mapas de saliencia se le aplica un filtrado gaussiano con un *sigma* configurable, pudiendo guardar los resultados como imágenes y/o como ficheros .mat para poder usarlos en programas de *Matlab*.

- heatmaps.py: Este programa trata de colorear un mapa de saliencia (en este caso serán los predichos por el modelo), utilizando un mapa de color configurable, y en este caso se ha utilizado el *viridis*[1]. Esto se hace para que sea más sencillo identificar las zonas más significativas del mapa de saliencia.

- assembling.py: En este programa se juntan los fotogramas reales de un vídeo con sus respectivos mapas de saliencia coloreados. Así, se puede ver rápidamente si las zonas predichas son consistentes con lo que se ve en el vídeo.

[1] `https://www.geeksforgeeks.org/matplotlib-pyplot-viridis-in-python/`

- gen_videos.py: Este programa coge todos los fotogramas generados con el script *assembling.py* para un determinado vídeo, y genera con ellos un vídeo en formato .mp4. Para ello es importante especificar bien los FPS con los que se quiere reconstruir el vídeo, ya que si no coinciden con los FPS usados para generar los fotogramas, el vídeo resultante no será correcto.

- copy_sound.py: Con este programa se junta el .mp4 generado con el script anterior y un archivo de audio que contiene el sonido de dicho vídeo.

- eval_single_video.m: Con este programa de Matlab se evalúa un vídeo, utilizando por una parte los mapas de saliencia generados por el modelo, y por otra parte tanto los mapas de saliencia como de fijaciones que representan el *ground truth*, los cuales se han generado con el programa *convert.py*. Finalmente, se obtienen los valores de CC(apartado 4.2.2) y de KL(apartado 4.2.3) de dicho vídeo.

- eval_multiple_videos.m: Este programa repite el proceso del anterior para un conjunto de vídeos, lo cual se puede utilizar para evaluar el desempeño de un modelo de predicciones para los datos de *test*.