UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MASTER

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

Uso de herramientas de *machine learning* para la detección de rayos gamma en telescopios de Cherenkov

Una revisión de la biblioteca CTLearn y comparación con el modelo *ParticleNet*

Autor
Olmo Arquero Peinazo (alumno)

Directores
Alberto Guillén Perales (tutor 1)
Rubén López-Coto (tutor 2)

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, convocatoria de septiembre del curso 2023/2024

Uso de herramientas de *machine learning* para la detección de rayos gamma en telescopios de Cherenkov: Una revisión de la biblioteca CTLearn y comparación con el modelo *ParticleNet*

Olmo Arquero Peinazo (alumno)

Palabras clave: Machine Learning, Deep Learning, Radiación Cósmica, Reconstrucción de Eventos, Nube de Puntos.

Resumen

El uso de técnicas de *machine learning* orientadas al estudio de la radiación cósmica a través de las imágenes por telescopios de Cherenkov es un campo de la ciencia de datos aplicado a la física. Su uso ha permitido avances considerables aprovechando el aumento de la potencia computacional y el desarrollo en el aprendizaje automático. Las estructuras de *deep learning*, especialmente las redes neuronales convolucionales profundas, han demostrado ser especialmente útiles en la reconstrucción de eventos, tarea esencial para determinar características del origen de los rayos cósmicos y arrojar luz sobre los eventos más extremos del universo. Este documento estudia el uso de la biblioteca CTLearn, un *framework* desarrollado para facilitar el uso de técnicas de *deep learning* en la reconstrucción de eventos de rayos cósmicos, incluyendo modelos que constituyen las estructuras más avanzadas del campo. Este trabajo analiza su desempeño y los compara con otro tipo de estructuras basadas en nubes de puntos, que están siendo utilizadas en la física de partículas en los aceleradores. Así mismo, se realizará una valoración de experiencia de usuario de esta biblioteca, proponiendo mejoras y posibles líneas de desarrollo que pasen por generar estructuras de datos que permitan acercar estas técnicas al público investigador general.

Use of machine learning tools for gamma-ray detection at Cherenkov telescopes: A review of the CTLearn library and comparison with the ParticleNet model

Olmo, Arquero Peinazo (student)

Keywords: Machine Learning, Deep Learning, Cosmic Radiation, Event Reconstruction, Point Clouds.

Abstract

The use of *Machine Learning* techniques aimed at studying cosmic radiation through images from Imaging Atmospheric Cherenkov Telescopes is a field of data science applied to physics. Its application has enabled considerable advances by leveraging increased computational power and developments in machine learning. Deep Learning structures, particularly deep convolutional neural networks, have proven to be especially useful in event reconstruction, which is essential for determining the characteristics of the origin of cosmic rays and shedding light on the most extreme events in the universe. This document studies the use of the CTLearn library, a framework developed to facilitate the application of Deep Learning techniques in cosmic ray event reconstruction, including models that represent the most advanced structures in the field. This work analyzes their performance and compares them with other types of structures based on point clouds, which are being used in particle physics in accelerators. Additionally, a user experience evaluation of this library will be conducted, proposing improvements and potential lines of development that involve generating data structures to bring these techniques closer to the broader research community.

Yo, **Olmo Arquero Peinazo**, alumno de la titulación Máster Universitario en Ciencia de Datos e Ingeniería de Computadores de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 31026383G, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Master en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen. Así mismo, asumo la originalidad del trabajo, entendida en el sentido de que no se han utilizado fuentes sin citarlas debidamente.

Fdo: Olmo Arquero Peinazo

Granada a 06 de septiembre de 2024.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La radiación cósmica es un tipo de radiación procedente del espacio exterior formada por partículas cargadas, como fotones y neutrinos (De Angelis & Pimenta, 2018). Las máximas energías medidas para estas partículas son de $10^{21} eV$, mientras que, en experimentos terrestres, la energía máxima alcanzada está en los TeV (10^{12}). Es en este aspecto donde radica su verdadera utilidad, ya que su estudio permite arrojar luz sobre facetas aún desconocidas de la física de partículas y la astrofísica. En concreto, el estudio de la radiación cósmica permite probar y estudiar los eventos más extremos del universo, siendo la única vía de estudio de la física de partículas a dichos niveles de energía.

Para investigar estos fenómenos disponemos de distintos métodos basados en la detección de *mensajeros* (Spurio et al., 2018, Capítulo 1.3). Los neutrinos, por ejemplo, debido a su débil interacción con la materia, nos permiten obtener información más certera sobre la fuente de origen de los rayos cósmicos. Por otro lado, las ondas gravitacionales, como la detectada en el evento GW170817 (Abbott et al., 2017), cumplen un papel similar gracias a su gran alcance y nula desviación.

Finalmente, los fotones comprenden el último gran mensajero, ya que no sufren desviaciones de trayectoria durante el desplazamiento de los rayos cósmicos desde la fuente hasta nosotros por la acción de campos electromagnéticos, debido a su neutralidad de carga. El método de detección de los fotones (rayos gamma) es a través de la interacción con la materia y, dependiendo del orden de energía objetivo, se usan distintos instrumentos e instalaciones. Este trabajo se centra en el estudio de los rayos gamma a través de los Telescopios Cherenkov (IACT, *imaging atmospheric Cherenkov telescope*)(Spurio et al., 2018, Capítulo 9.1), una técnica de detección indirecta que se sirve de la denominada Radiación Cherenkov, una luz cercana al ultravioleta producida cuando una partícula cargada se mueve en un medio más rápido que la luz; este es el caso para las partículas que se producen

fruto de la interacción de los fotones con la atmósfera (*Extensive Air Shower*, EAS).

El reto principal de esta técnica es realizar la reconstrucción del evento, es decir, obtener la dirección (altitud y acimut) y energía del fotón que produce la EAS (Jacquemont et al., 2021). Sin embargo, un fotón no es la única partícula que puede producir este fenómeno, por lo que se vuelve prioritaria la tarea de discriminación de eventos, con el objetivo de descartar las cascadas producidas por hadrones (esencialmente protones).

Tradicionalmente, el método más extendido para la discriminación y reconstrucción del evento (*event reconstruction*) se basaba en una serie de parámetros, entre los que destacan la extensión de la luz a lo largo y perpendicular a un eje, la fracción de luz entre las dos señales más grandes, y anillos concéntricos de la imagen, entre otros (Hillas, 1985). Estos parámetros fueron combinados con métodos de análisis multivariante (Fiasson et al., 2010), y posteriormente usados como variables de entrada para algoritmos clásicos de *Machine Learning* (ML) como Random Forest (Albert et al., 2008) o Boosted Decision Trees (Krause et al., 2017).

La aplicación de las técnicas de ML, y más recientemente de *Deep Learning* (DL) al estudio de los rayos cósmicos mediante las imágenes de Cherenkov tenían como objetivo mejorar esta reconstrucción de eventos. El desarrollo de las redes neuronales convolucionales profundas (*Deep Convolutional Neural Networks*, DCNN) en el campo de la visión por computador, implementadas por primera vez en (Nieto et al., 2017), demostraron posteriormente tener la capacidad de mejorar la reconstrucción de energía y dirección en simulaciones de rayos gamma (Mangano et al., 2018), junto con la sensibilidad del análisis en datos reales (Shilon et al., 2019).

El uso de las DCNN continúa siendo amplio, estando presentes en el estado del arte («Image Classification | Papers With Code», 2024), demostrando el potencial presente en este tipo de estructuras para continuar mejorando los resultados de la reconstrucción de eventos. Es por ello por lo que, en 2019, se lanza al público la versión v0.3.1 de CTLearn (Brill et al., 2018), un paquete de Python (Van Rossum & Drake, 2009) que continúa en desarrollo y permite la aplicación de técnicas de DL a este tipo de tareas, con el objetivo futuro de facilitar la aplicación y análisis de resultados a un público no experto en programación y ciencia de datos. Esta librería ha sido desarrollada de forma colaborativa e incremental, teniendo entre sus vertientes distintos proyectos de investigación y desarrollo, como la Tesis Doctoral de Tjark Miener (Miener, 2024) entre otros (Romanato, 2020; Grespan, 2020; Marinello, 2019; Mariotti, 2019).

Esta biblioteca ofrece un *framework* para el desarrollo de técnicas de DL y análisis de los datos experimentales, apoyándose en otras bibliotecas como dl1-data-handler (DLH) (Kim et al., 2024), ctapipe (Linhoff et al., 2023) o

Astropy (Astropy Collaboration et al., 2022). Mediante el uso de archivos de configuración YAML, permite realizar entrenamientos y predicciones reproducibles, facilita el registro automático de estos y la configuración de los algoritmos a usar. Los modelos implementados por defecto y el módulo de gestión de datos, basado en DLH, permiten tanto el análisis y entrenamiento con una sola imagen (un telescopio), como aprovechar la información de varias de ellas para la reconstrucción de eventos detectados a través de *arrays* de telescopios.

Recientemente, CTLearn se ha usado para la reconstrucción de imágenes estereoscópicas de la nueva generación de telescopios de Cherenkov *Cherenkov Telescope Array* (CTA), con un modelo *ResNet* (TRN) entrenado con imágenes de distintos telescopios, e integrando el modelo *backbone* entrenado como extractor de características en un modelo TRN-RNN, obteniendo resultados prometedores (Miener et al., 2021).

Uno de los objetivos de este trabajo será familiarizarse con el proceso completo de entrenamiento, evaluación y producción de resultados que indiquen la calidad de estos modelos, atendiendo a características concretas de este desarrollo, el formato de los datos, su representación, y el preprocesamiento necesario para adaptar las imágenes originales a las herramientas de DL usadas en CTLearn.

Para explorar nuevos métodos de análisis, se aplicará la estructura de red neuronal *ParticleNet*, un modelo personalizado basado en los bloques *EdgeConv* (Wang et al., 2019) que permite realizar operaciones convolucionales sobre estructuras de datos de tipos *nubes de puntos*, y que obtiene resultados prometedores en la clasificación de jets de partículas en colisionadores (Qu & Gouskos, 2020). La ventaja principal de este algoritmo radica en la reducción de dimensionalidad de los datos de entrada, al poder seleccionar de una imagen solo los píxeles interesantes para el análisis.

Por último, se realizará una propuesta conceptual para la aplicación y uso de una estructura *Data Lake*, con el objetivo de mejorar la gestión de los datos, el acceso y el uso de este tipo de herramientas a investigadores.

Capítulo 2

Objetivos

Los objetivos de este TFM son:

- OB1.- Estudiar un posible diseño de Data Lake para el almacenamiento y la consulta de datos de astropartículas del LST-1 de CTAO.
- OB2.- Analizar de forma comparativa modelos de clasificación, usando datos de simulación y datos reales de LST-1 de CTAO.
- OB3.- Implementar metodología MLOPS para el ajuste de hiperparámetros y la evaluación.

Capítulo 3

Fundamentación

3.1. Estudio de Rayos Cósmicos: *Extensive Air Showers*

Los rayos cósmicos (CR) son partículas subatómicas altamente energéticas cuyo origen es extraterrestre. La determinación del origen, los aceleradores cósmicos, sigue siendo uno de los mayores problemas en proceso de resolución de la astrofísica actual, y se suponen eventos intrínsecamente relacionados con la formación de estrellas masivas, supernovas o la evolución estelar.

Un caso de ejemplo de los sucesos sin resolver es la cantidad de positrones que existen con respecto a los electrones para un nivel concreto de energía en el espectro de CR. El comportamiento de esta proporción se espera decreciente con la energía, que se cumple para energías menores a los GeV. No obstante, a energías mayores, la fracción de positrones se incrementa, lo que indica la existencia de una fuente que los inyecta, que además ha de ser cercana (Katz et al., 2010). Una de las propuestas más populares acerca de su origen es la materia oscura (Barak et al., 2023).

La composición principal de esta radiación es, aproximadamente, un 90 % protones, un 9 % núcleos de helio, y un 1 % núcleos más pesados, electrones, positrones, antiprotones, junto con rayos gamma y neutrinos, que, dependiendo de su energía, tienen origen solar ($E < 1GeV$), galáctico ($1GeV < E <\sim PeV$) o extragaláctico ($> PeV$) (Spurio et al., 2018, Capítulo 1).

Los CR pueden ser clasificados como primarios, que son protones y núcleos (junto con una minoría de electrones) generados en entornos extremos del espacio exterior, y normalmente pueden ser detectados antes de interaccionar con la atmósfera. Los CR secundarios son aquellos producidos por la

interacción de los primarios con gas interestelar o con núcleos en la atmósfera terrestre, siendo los más comunes los antiprotones y los positrones.

Cuando los CR entran a la atmósfera, se produce una colisión con los núcleos más abundantes de esta (oxígeno y nitrógeno) generando una cascada de partículas secundarias (Gaisser, 1982), siendo el mecanismo principal la interacción de un protón con un núcleo pesado. La detección y medición de las características de las partículas secundarias es una forma de estudio extendida de este tipo de eventos.

Los CR con energías mayores a algunos GeV no llegan a la Tierra con un flujo lo suficientemente alto para que sean detectados con satélites. Este hecho hace que sea necesario otro tipo de técnicas que estudien las consecuencias de la interacción de los CR con la atmósfera; instrumentos de tierra que sean capaces de cubrir un área de miles de km^2, llamados *extensive air shower* (EAS) *arrays*. Estos consisten en experimentos que cubren grandes áreas y que tienen larga duración, buscando obtener la mayor precisión en deducir las características del rayo cósmico que produce la cascada en la atmósfera (ejemplo del evento en la figura 3.1).

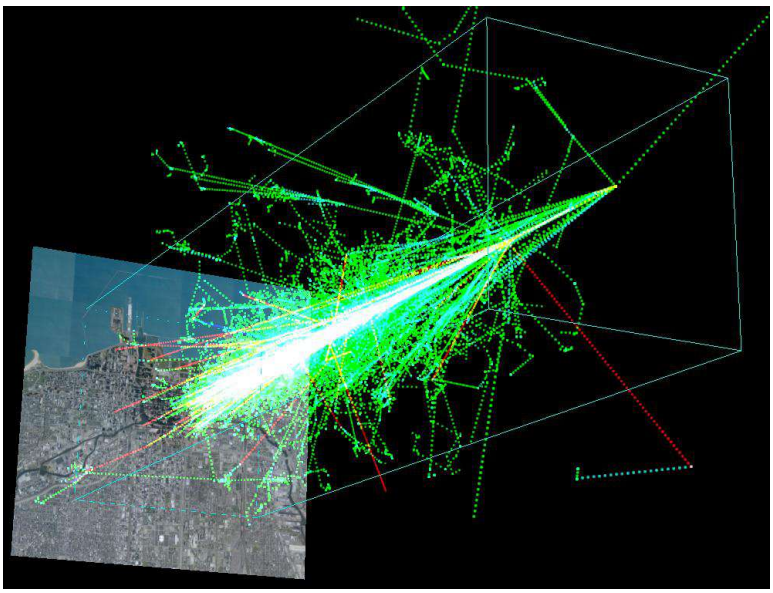


Figura 3.1: Simulación de una EAS iniciada por un protón de energía 1TeV (10^{12}eV) cuando este incide en la atmósfera a aproximadamente 20km de altura (Wikimedia Commons, 2006).

No obstante, existe un reto considerable en el estudio de estos fenómenos: los CR son principalmente protones y núcleos pesados, que inician cascadas hadrónicas interaccionando con la atmósfera tras atravesar una longitud determinada. Sin embargo, este tipo de partículas suelen sufrir desviaciones

debidos a campos electromagnéticos, lo que dificulta determinar su fuente con exactitud. Es por ello que se prefiere realizar el estudio de los fotones (rayos gamma), debido a su nula interacción con campos electromagnéticos y difícil desviación.

La atmósfera terrestre es opaca a los rayos gamma de alta energía, debido a que estos interaccionan electromagnéticamente con el aire, produciendo pares de electrón/positrón. Estas partículas secundarias producen una nueva generación de rayos gamma por la radiación de frenado o *bremsstrahlung* (Al Samarai et al., 2015), de manera que se produce una cascada electromagnética (EM). Puesto que la mayoría de las partículas en este tipo de eventos viajan a velocidades ultrarelativistas a través de la atmósfera, se producirá la llamada radiación Cherenkov; la emisión de luz en el rango de $300 - 350nm$ cuando una partícula cargada se mueve a velocidades superiores a la de la luz en un medio concreto. La emisión de esta radiación respecto a la dirección de movimiento de la partícula viene dada por $\theta_C = cos^{-1}(1/n)$, con n el índice de refracción, siendo $\theta = 1.3°$ para condiciones estándar. Este ángulo se incrementa, como lo hace n, a medida que la cascada EM se acerca al suelo (Spurio et al., 2018, Capítulo 4.6.3).

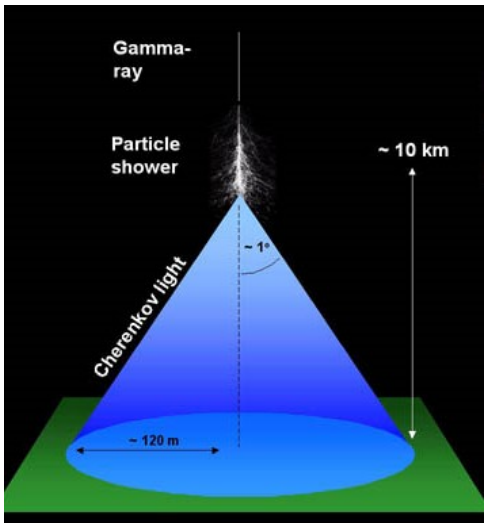


Figura 3.2: Diagrama del cono de luz Cherenkov generado por una cascada de partículas (Max Planck Institute for Nuclear Physics, 2023).

El resultado de este fenómeno es un cono de luz en una región anular con un radio aproximado de 120m para una cascada típica producida por radiación gamma (ver figura 3.2). Una de las dos técnicas principales para el estudio de este proceso es la imagen de radiación Cherenkov, que consiste en la detección de la distribución de la luz de Cherenkov a nivel del suelo producida por las cascadas. Desde esta medida, es posible determinar el

desarrollo longitudinal y lateral de la cascada, la dirección de llegada y la energía de los gammas primarios.

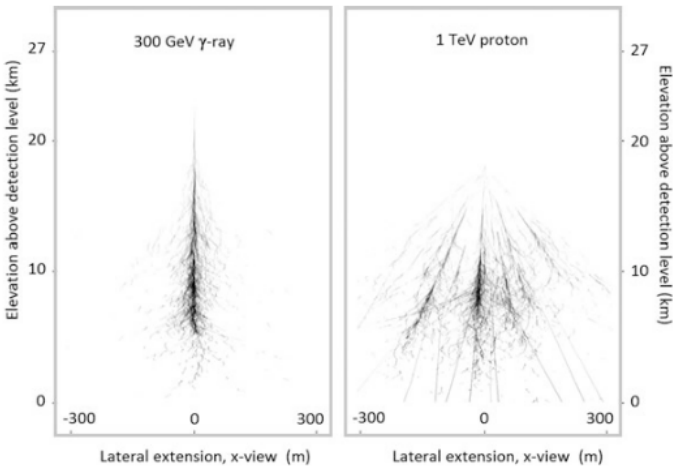


Figura 3.3: Comparación de una cascada EM pura de un rayo gamma de 300GeV y una cascada iniciada por un protón a 1TeV (Spurio et al., 2018, Capítulo 9.1).

Las cascadas hadrónicas tienden a tener imágenes más largas y fluctuantes debido al efecto de las partículas principales. Además, presentan una mayor anchura y no están alineadas sistemáticamente con la fuente si esta es isotrópica. Estas y otras características dan una base teórica sobre la que fundamentar el uso de algoritmos de aprendizaje autómatico, permitiendo automatizar la distinción entre eventos hadrónicos y eventos iniciados por rayos gamma (Hillas, 1985).

3.2. Telescopios Cherenkov

Los telescopios Cherenkov (*Imaging Atmospheric Cherenkov Telescope*, IACT) son uno de los métodos más extendidos para el estudio de los rayos gamma provenientes de los CR de forma indirecta, basándose en el fenómeno de la radiación Cherenkov para extraer información de los CR a través de su interacción con la atmósfera.

Los IACTs consisten en amplios telescopios ópticos con un gran reflector que llegan a alcanzar decenas de metros de diámetro, reflejando la luz incidente a una cámara colocada en el plano focal. Esta cámara requiere tiempos de exposición menores a los $30ns$ para poder detectar estos fenómenos y diferenciarlos de fondos de otros procesos. Por otro lado, las condiciones ambientales son de suma importancia, siendo imperativo el instalarlos en en-

tornos cercanos a la absoluta oscuridad, y solo pudiendo operarlos en noches sin luna (Spurio et al., 2018, Capítulo 9.1).

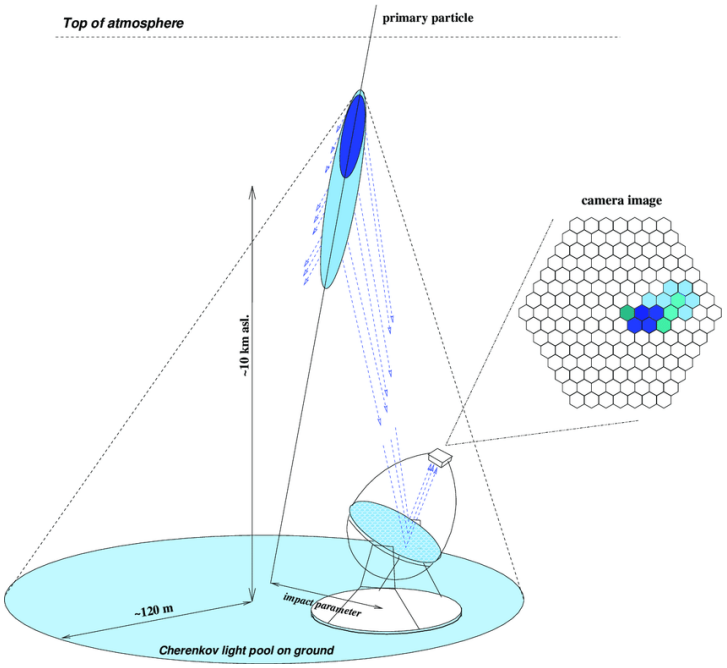


Figura 3.4: Esquema del proceso de detección de EAS usando un IACT y la imagen resultante (Satalecka, 2010).

Entre las estructuras de estos telescopios encontramos reflectores parabólicos simples (Bigongiari, 2005), o muchos espejos individuales segmentados (Abdalla et al., 2018; Weekes et al., 2002). Por otro lado, una nueva generación de IACTs está siendo construida en forma de *array* de telescopios individuales, dividida en dos observatorios en el hemisferio sur y norte, ampliando la cobertura de cielo abierto y mejorando la sensibilidad con respecto a los telescopios actuales (Hofmann & Zanin, 2024).

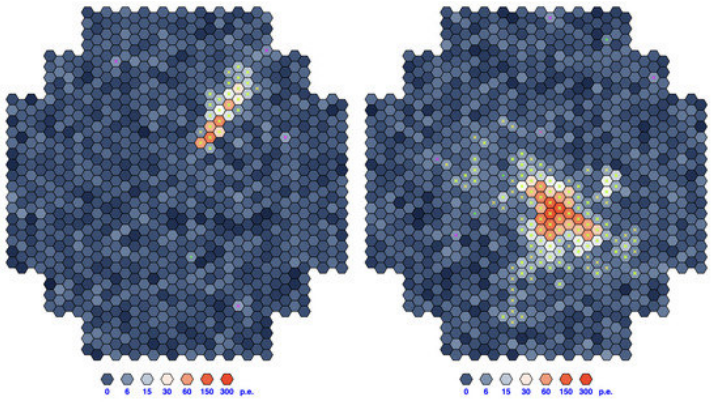


Figura 3.5: Diferencia entre las imágenes de cascadas inducidas por gammas (izquierda) y por hadrones (derecha) en la cámara de un IACT (Völk & Bernlöhr, 2009).

La imagen formada en el plano focal de los IACTs contiene el desarrollo longitudinal de la EAS, además de información espacial, temporal y de calorimetría (ejemplo de un evento en la figura 3.4).

Debido a la presencia de fluctuaciones de luz del *Night Sky Background*, es necesaria aplicar técnicas que permitan separar el ruido de los rápidos destellos de luz Cherenkov. El origen de este ruido puede ir desde la difusión de esta radiación en la atmósfera hasta la luz de estrellas, y depende de la región, la fase de la luna y la zona del cielo que se observa, siendo común el uso de características temporales y espaciales de la señal para descartar o guardar eventos (López-Coto et al., 2016).

Por otro lado, los telescopios de Cherenkov y sus sistemas no pueden ser calibrados con haces de partículas de características conocidas, sino que es necesario el uso de simulaciones Monte Carlo para simular la cascada y cómo esta sería percibida por los detectores. Mientras que se comprenden a un nivel aceptable las interacciones EMs y débiles, las dificultades aparecen en la interacción hadrónica, siendo necesario mezclar conceptos experimentales y teóricos para modelar este comportamiento.

CORSIKA (Heck et al., 1998) es, actualmente, uno de los *softwares* más extendidos para simular este tipo de cascadas. Este programa rastrea cada partícula producida en la atmósfera, considerando todos los procesos físicos relevantes, como la pérdida de energía, la dispersión múltiple, la influencia del campo magnético terrestre, los modos de decaimiento y las interacciones EMs y hadrónicas. CORSIKA incluye varios modelos para la interacción hadrónica, permitiendo comparaciones y estimaciones de errores sistemáticos.

Las aplicaciones de ML sobre las imágenes obtenidas por los telescopios

Cherenkov tienen como objetivo la reconstrucción del evento inicial. Como se expone en la Sección 3.1, existen determinados parámetros y medidas topológicas que permiten diferenciar las cascadas producida por un rayo gamma y por un protón. Estas características se reflejan en la imagen proyectada sobre el plano focal del telescopio, permitiendo usar los mismos principios físicos para la discriminación de eventos (ejemplo en la figura 3.5).

3.3. CTLearn (Funcionamiento)

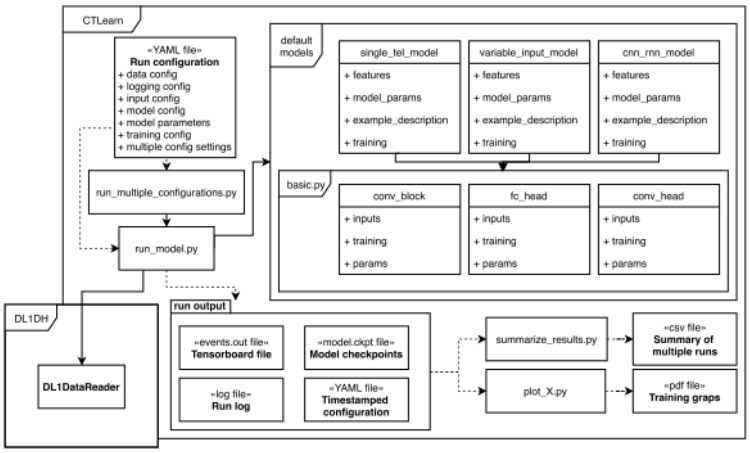


Figura 3.6: Diagrama de resumen del diseño del framework de CTLearn (Nieto, Brill, Feng, Humensky et al., 2019).

La biblioteca CTLearn constituye un entorno en el que desarrollar modelos para la reconstrucción de eventos de CR estudiados por medio de imágenes Cherenkov. La filosofía de su creación y desarrollo es permitir un uso más amigable de este tipo de herramientas, y un sistema automático de registro de hiperparámetros y configuraciones de otra índole a modo de asegurar la replicabilidad del estudio.

Nivel de Datos	Descripción	Tamaño Aproximado
DL0 RAW	Señal digital del hardware de adquisición.	10 GB
DL1 CALIBRATED	Fotones reales y tiempos medidos en cada telescopio.	10 GB
DL2 RECONSTRUCTED	Dirección inferida, energía y probabilidad de ser un fotón (gammaness) para cada evento.	1 GB
DL3 REDUCED	Lista de eventos seleccionados y respuesta instrumental.	10 MB

Tabla 3.1: Esquema de los niveles de datos y su descripción (Ruiz, 2020).

Uno de los elementos más destacables es la posibilidad del uso de archivos de configuración YAML, que facilitan la reproducción y repetición de los experimentos, además de presentar de forma ordenada el conjunto de parámetros y opciones que necesita tanto el modelo como las herramientas de

gestión de datos (preprocesamiento, escritura...). En la página de GitHub de la biblioteca se puede encontrar un ejemplo de este archivo de configuración (https://github.com/ctlearn-project/ctlearn).

Al usar un modelo por defecto, se usará el archivo de configuración presente en el directorio *ctlearn/default_config_files* simplificando de esta forma su aplicación.

A pesar de que el uso de los archivos YAML es la forma más completa de personalizar cualquier proceso con CTLearn, el método más sencillo para usar los algoritmos y configuraciones por defecto es mediante la misma línea de comandos en un entorno con la biblioteca instalada. El inicio de cualquier proceso se realiza usando la palabra clave *ctlearn*.

A continuación se expone un ejemplo de uso para el entrenamiento de un modelo por defecto de la biblioteca:

```
ctlearn -d TRN \
 -i "Directorio_a_datos_de_entrenamiento" / \
 -o "Directorio_de_salida" \
 --clean -z 50 -e 8 -t LST_LST_LSTCam -b 64 \
 -m train \
 -r energy
```

La opción `-d` (`--default`) permite elegir entre los distintos modelos por defecto de la biblioteca CTLearn, siendo este caso el modelo TRN (Sección 3.4). Por otro lado, `-i` (`--input`) permite establecer el directorio donde se encuentran los datos de entrenamiento, con la opción de usar `-p` (`--pattern`) para seleccionar solo determinados archivos que cumplan con un patrón en su nombre.

El directorio de salida se introduce con la opción `-o` (`--output`), y será la localización del modelo, de los registros de entrenamiento y otros archivos auxiliares que permiten una lectura más rápida de los datos, o reiniciar el entrenamiento desde puntos de guardado. Al usar la biblioteca *Tensorflow*, *tensorboard* está disponible para llevar una supervisión a tiempo real de los entrenamientos. Este directorio es de suma importancia en caso de realizar predicciones, continuar o reiniciar el entrenamiento, ya que se deberá de indicar en la línea de comandos para informar a la biblioteca sobre la localización de los archivos del modelo entrenado.

La *flag* `--clean` y la opción `-z` permiten realizar transformaciones y selección de los datos. El primero de ellos elimina (pone a 0) los píxeles que solo contienen ruido de fondo. El segundo parámetro permite realizar un corte en la *Hillas Intensity*, la carga total en fotones fotoeléctricos (phe) contenida en toda la imagen, que normalmente se establece utilizando tres umbrales diferentes: >50 phe (*low-cut*), >200 phe (*mid-cut*) y >1000 phe (*high-cut*) (Miener, 2024, Capítulo 2.5.3). El resto de argumentos de esta

línea son: `-e` para indicar el número de épocas, `-t` para explicitar el tipo de telescopio para la transformación correcta de las imágenes (dimensiones) y `-b` para el tamaño de lote (*batch size*).

Finalmente, las opciones `-m` (`--mode`) y `-r` (`--reco`) son las encargadas de informar acerca del proceso a realizar (`train` o `predict`) y del tipo de reconstrucción que realizará el modelo, `type`, `energy` o `direction`, coincidiendo con tareas de clasificación y regresión (simple o múltiple respectivamente).

Para realizar una predicción será necesario re-ejecutar el mismo comando, eliminando argumentos del entrenamiento como el número de épocas o los filtros como el corte en *Hillas Intensity*, y añadir una lista de elementos a predecir a través del argumento `-i`. La predicción se realizará en el directorio designado por el argumento `-y` (`--prediction_directory`).

La entrada de datos (tanto para entrenamiento como para predicción) tiene dos alternativas en el caso de querer usar un archivo de configuración YAML. Además de las ya expuestas, para el entrenamiento se puede configurar el argumento `file_list` en la sección `Data` del archivo YAML. Esto permite elegir de manera más concreta los archivos que se desean usar para entrenar el modelo. Por otro lado, para predecir otros archivos, se puede añadir al final del documento la sección `Prediction`, con el argumento `prediction_file_list`, que consiste en una lista de pares clave-valor, donde la clave es el directorio de guardado, y el valor el archivo a predecir.

Puesto que en el momento de generar un modelo siempre se guardará el archivo de configuración YAML, los argumentos y parámetros introducidos en la llamada a CTLearn prevalecerán sobre aquellos en los archivos por defecto y serán los que se registren.

El conjunto de datos (datos de entrada y predicciones) que se gestionan con CTLearn y el resto de las bibliotecas auxiliares son archivos HDF5, que contienen los datos de forma jerárquica. El formato concreto de los datos DL1 (ver tabla 3.1) puede ser estudiado en la documentación de ctapipe (https://ctapipe.readthedocs.io/en/latest/user-guide/data_models/dl1.html), destacando la presencia de multitud de datos extra además de las propias imágenes, como la configuración de los telescopios, la posición de estos, orientación, tiempo de exposición, etc. Una vez realizada la predicción, el archivo de datos se complementará con la reconstrucción geométrica de la cascada (usando para ello la dirección), la energía, y el tipo de partícula que la inicia.

Por otro lado, estos archivos también permiten introducir datos generados por simulación e información extra y más técnica sobre los instrumentos de medición, geometría de las cámaras o información óptica de los telescopios.

En resumen, el formato y el almacenamiento de los datos es un aspecto muy depurado en todo el conjunto de las bibliotecas que componen el en-

torno de CTLearn, y permiten mantener compactos y ordenados todos los elementos necesarios para realizar el proceso que va desde la configuración del modelo hasta obtener los resultados finales.

3.4. Modelo TRN en CTLearn

Como se ha expuesto en el apartado 3.3, la biblioteca cuenta con algunos modelos por defecto pre-configurados, accesibles a través de la opción `-d` en el comando de entrenamiento o predicción.

De estos, el modelo que mejores resultados proporciona actualmente es una Red Neuronal Convolucional Profunda compuesta por bloques residuales y un mecanismo de *squeeze-and-excitation* en cada uno de dichos bloques (Miener et al., 2021; Miener et al., 2022). Esta estructura recibe como entrada una imagen de tamaño $(h \times w \times d)$, donde $(h \times w)$ son las dimensiones espaciales y d representa la profundidad o características.

Como con cualquier otro modelo por defecto, es posible modificar ciertos hiperparámetros, como el número de sub-bloques que componen el bloque residual, los filtros de salida de cada uno de estos bloques y la ratio de compresión para el uso del mecanismo *squeeze-and-excitation*. Para la subsecuente explicación, se usarán `filters` (filtros de salida de cada bloque) y `ratio` (ratio de compresión en el mecanismo *squeeze-and-excitation*) como palabras clave para identificar estos hiperparámetros.

Los bloques residuales consisten en varias estructuras iguales, o sub-bloques, apiladas y conformadas fundamentalmente por una serie de capas convolucionales que aumentan las características (profundidad) de la imagen de entrada, teniendo como tarea principal la generación de características más complejas o útiles para la resolución del problema. Concretamente, cada sub-bloque estará formado por 3 capas convolucionales: la primera de dimensión $(1, 1, \text{filters})$ y activación ReLU, la segunda de dimensión $(3, 3, \text{filters})$, activación ReLU y con la opción `padding=same` (asegurando la transmisión de la dimensión espacial), y la tercera de dimensión $(1, 1, 4 \cdot \text{filters})$ y sin función de activación. A esta estructura se le añade una capa más al inicio cuando es necesario adaptar la dimensión de entrada a la dimensión de salida del bloque para establecer la conexión residual que da nombre a estos bloques. Esto se realiza a través de una capa de dimensiones $(1, 1, 4 \cdot \text{filters})$, y siempre está presente en el primer sub-bloque del bloque residual completo.

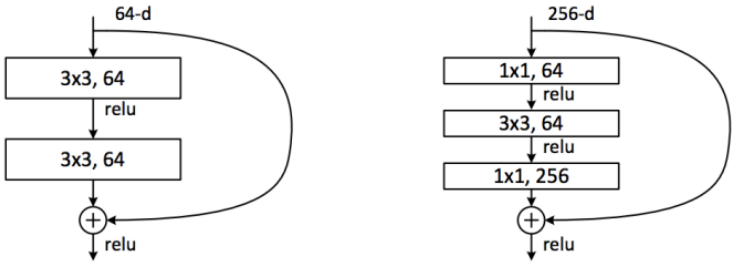


Figura 3.7: Ejemplo esquemático de conexión residual. Modelo simple (izquierda) y modelo de cuello de botella o *bottleneck* (derecha) (He et al., 2015).

La conexión residual (He et al., 2015) consiste en conservar la entrada a estos bloques sin alterar por las distintas capas intermedias para sumarla a la salida del bloque. Esto es, si se considera el conjunto de los bloques convolucionales una función $\mathcal{F}(\cdot)$ sobre la entrada $\mathbf{x}$, el uso de conexiones residuales transforma la salida $\mathcal{F}(\mathbf{x})$ en $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. Esta técnica permite que la información original se conserve en un mayor número de capas de la red, facilitando el uso de estructuras profundas, evitando así problemas relacionados con la determinación de los pesos óptimos de las neuronas (He et al., 2015). En la estructura TRN, el modelo de conexión residual usado es el de cuello de botella, que consiste en la reducción de la dimensión de entrada, para posteriormente aumentarla (restaurarla) de nuevo a la salida del bloque (ejemplo en figura 3.7).

Seguido a las capas convolucionales, encontramos el mecanismo *squeeze-and-excitation* (Hu et al., 2018). En este caso, el mecanismo es doble.

Por un lado, se realiza una ponderación de los filtros a la salida de las capas convolucionales (ejemplo en figura 3.9), de manera que se reduce la importancia de las dimensiones de profundidad menos informativas, y se priorizan aquellas que ayudan a la red a completar la tarea. Esta estructura está conformada por una capa de tipo *Global Average Pooling*, seguida de una capa convolucional de dimensiones $(1, 1, 4 \cdot \text{filters/ratio})$ y activación ReLU, y finalmente otra capa de dimensiones $(1, 1, 4 \cdot \text{filters})$ y activación sigmoide.

Por otro lado, se lleva a cabo una ponderación a nivel espacial, es decir, de cada uno de los elementos de las dos primeras dimensiones, con un mismo peso para todos los niveles de profundidad. La estructura que realiza esto es una simple capa convolucional de dimensiones $(1, 1, 1)$ y activación sigmoide.

El resultado de cada una de las estructuras mencionadas anteriormente se multiplica elemento a elemento con la entrada, realizando la comentada ponderación. Finalmente, se suman las salidas de ambas ramas, se añade el residuo, y se aplica una función de activación ReLU.

La estructura comentada hasta ahora corresponde a un sub-bloque del bloque residual, de forma que para completarlo se repite la estructura base, sin cambiar `filters` ni `ratio`, un número determinado de veces. La salida de cada bloque residual será la entrada al siguiente, hasta que se llegue al final de la red. La estructura de uno de estos sub-bloques se muestra en la figura 3.8.

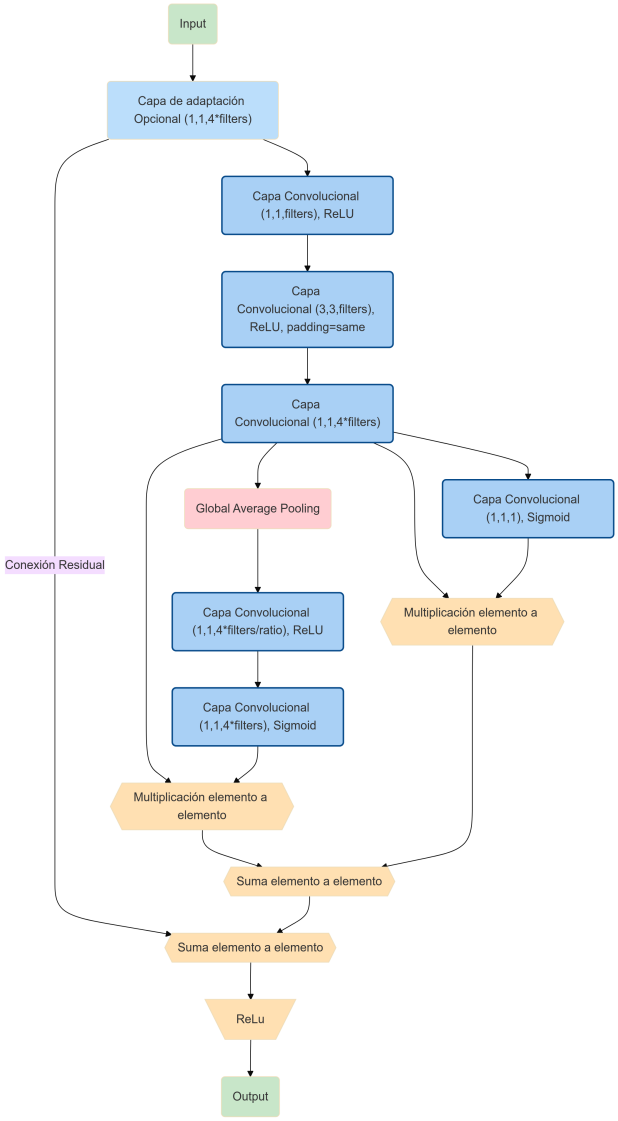


Figura 3.8: Diagrama de la estructura de un sub-bloque residual de la red TRN.

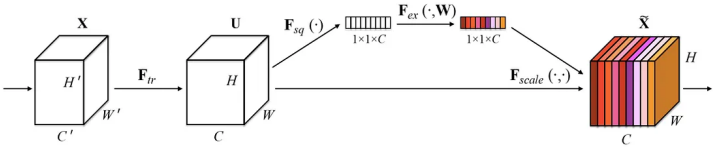


Figura 3.9: Ejemplo de estructura de tipo *squeeze-excitation* para la ponderar la dimensión de profundidad (Pröve, 2017).

La salida de la red pasa por una capa *Global Average Pooling* para reducir las dimensiones de los datos a 2D, posteriormente por una serie de capas densas con activación de tipo ReLU, y una capa de salida que dependerá del problema a resolver (ver secciones 3.6 y 4.2).

Este tipo de estructura es la más avanzada actualmente en el marco de la biblioteca CTLearn, obteniendo resultados competitivos incluso cuando se usa como bloque de extracción de características (Miener et al., 2021) para el caso donde se usan imágenes de varios telescopios, siguiendo la estructura que se muestra en la figura 3.10.

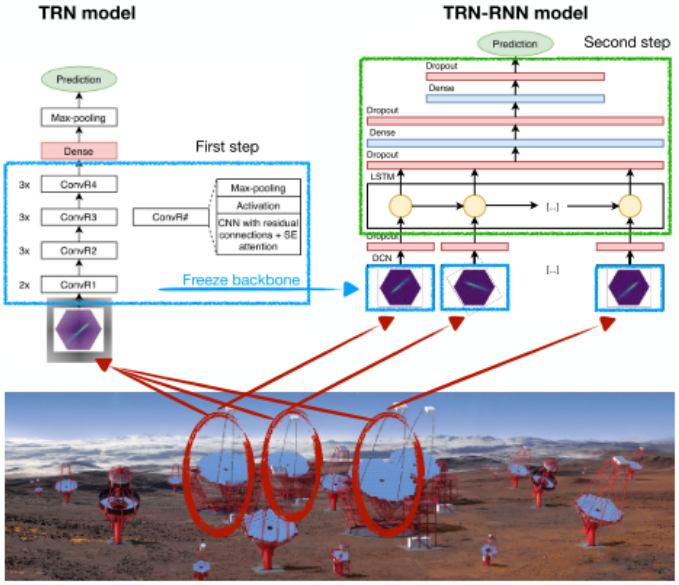


Figura 3.10: Diagramas de las principales capas del modelo TRN (izquierda) y TRN-RNN (derecha) usadas para la reconstrucción de eventos de imágenes producidas por CTA (Miener et al., 2021).

3.5. ParticleNet

ParticleNet es una estructura propuesta por Huilin Qu y Loukas Gouskos (Qu & Gouskos, 2020) basada en las Redes Neuronales Convolucionales Dinámicas Gráficas (Dynamic Graph Convolutional Neural Network (Wang et al., 2019)) para tratar de resolver un problema de clasificación relacionado con los jets de partículas en experimentos de colisionadores.

Este tipo de estructura se aplica sobre datos en formato de nubes de puntos (*point clouds*, PC), es decir, en vez de utilizar la imagen completa de la colisión de las partículas, que tiene alta dimensión y muchos píxeles inservibles, se extraen aquellos puntos donde una partícula ha incidido, y se representa en un formato de coordenadas-características, permitiendo reducir considerablemente el tamaño de los datos. El objetivo principal de esta red es generar una estructura que, manteniendo constante el número de puntos a través de todo el recorrido en la red, pueda aplicar operaciones simétricas ante permutaciones de puntos. Para ello, utilizan los denominados bloques *EdgeConv*, que permiten la aplicación de operaciones convolucionales sobre las PC, asegurando que el orden en que estos son reflejados en el conjunto de datos no afecte.

El bloque *EdgeConv* aplica la siguiente operación para cada punto de entrada:

$$x_i' = \mathcal{A}_{j=1}^{k} \mathbf{h}_\Theta(x_i, x_{ij}) \tag{3.1}$$

donde x_i es el vector de características de dimensión arbitraria e $(i_1, ..., i_k)$ son los índices de los k vecinos más cercanos al punto x_i. De manera intuitiva, este bloque realiza una operación $h_\Theta(x_i, x_{i_j})$ sobre las características de cada punto (x_i) (que puede contener coordenadas y características) y sus vecinos (x_j), de forma que los parámetros Θ son los mismos para cada punto en una *point cloud*. Posteriormente, se aplica una agregación simétrica arbitraria $\mathcal{A}$, convirtiendo así la operación *EdgeConv* en una operación simétrica ante permutaciones en PC. Detalles sobre la implementación se muestran en la sección 4.1.1.

Los parámetros Θ que definen la función $h_\Theta(x_i, x_{i_j})$ serán los pesos de una capa oculta de la red. En el caso de *ParticleNet*, se decide usar una agregación con coordenadas relativas, de forma que la operación general $h_\Theta(x_i, x_{i_j})$ adopta la forma $h_\Theta(x_i, x_{i_j} - x_i)$, generando lo que podemos denominar una serie de características relativas al punto central x_i sobre el que se aplica la operación descrita.

Hay dos elementos interesantes sobre la estructura que merecen la pena destacar. El primero de ellos es que la operación de las capas *EdgeConv* puede ser fácilmente apilada, como las convoluciones usuales. Esto se debe a que estamos realizando una transformación de una *point cloud* a otra con

el mismo número de puntos, y por tanto es equivalente aplicar *EdgeConv* sobre la salida. Esta propiedad también permite entender los vectores de características aprendidos por *EdgeConv* como nuevas coordenadas en el espacio latente y, por tanto, los nuevos k vecinos más cercanos serán calculados en este espacio latente. Finalmente, hay que añadir que en *ParticleNet* se añaden también conexiones residuales (ver sección 3.4) para asegurar el correcto entrenamiento de la red y evitar problemas como el desvanecimiento del gradiente. En la figura 3.11 se puede observar la estructura interna de *ParticleNet* para la primera capa *EdgeConv*.

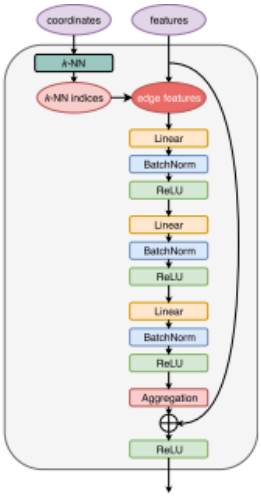


Figura 3.11: Estructura del bloque *EdgeConv* usado en *ParticleNet* (Qu & Gouskos, 2020).

Esta estructura supone un aumento en la precisión en el problema de etiquetado de sabor (Qu & Gouskos, 2020) con respecto a otros algoritmos convencionales. Las características de la red la hacen fácilmente adaptable al problema de reconstrucción expuesto anteriormente (sección 3.1), debido a la facilidad de convertir la imagen del telescopio en una *point cloud*, pudiendo añadir como características, además de las coordenadas, la intensidad de la luz y el tiempo de pico en cada píxel (esquema en figura 3.12 e información de los datos en la sección 4.2). Además, tiene la potencialidad de reducir la dimensionalidad de los datos, al ignorar píxeles que puedan contener menos o ninguna información útil.

3.6. Head (Cabeza) de los modelos

La cabeza de los modelos hace referencia a las capas finales que se añaden al `backbone` de un modelo para obtener la salida deseada. La implementada

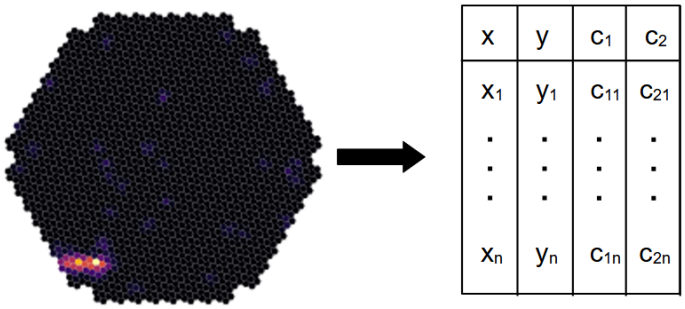


Figura 3.12: Esquema del cambio de representación en imagen a PC. A la izquierda se muestra un evento arbitrario registrado por un telescopio de Cherenkov, donde los píxeles más intensos representan las deposiciones mayores de carga (fotoelectrones). A la derecha, se muestra la matriz de dimensiones $n \times 4$ que representaría dicha imagen, seleccionando los n píxeles más importantes, las coordenadas de dicho píxel y 2 características arbitrarias de cada píxel.

por CTLearn está formada por un número concreto de capas densas y una capa final que variará dependiendo del problema a resolver.

Para clasificar el tipo de partícula, se añadirá una capa con el número de neuronas correspondientes al número de clases a predecir (ver sección 4.2). A esta capa se le aplicará una función de activación de tipo *Softmax* para normalizar las salidas de la red, de forma que se obtiene una puntuación (*gammaness*) interpretada como cuán parecido es el evento a un gamma. La función de pérdida que incorpora es la entropía Cruzada Categórica (*CategoricalCrossentropy*), con una reducción al lote (*batch reduction*) de tipo *sum over batch size*. La influencia de cada clase en esta función de pérdida será ponderada para reducir los efectos negativos de un posible desbalanceo en los datos de entrenamiento.

Cuando la tarea es reconstruir la energía, se añadirá una capa densa de una sola neurona, que será la salida de la red. En este caso, la función de pérdida usada será el error medio absoluto (*mean absolute error*, MAE) con una reducción al lote (*batch reduction*) de tipo *sum over batch size*.

Finalmente, para la reconstrucción de la dirección, se añade una capa de 3 neuronas, correspondientes a la altura, el ángulo acimutal y la separación angular al evento (más detalle en la sección 4.2). De nuevo, la función de pérdida usada será el MAE con una reducción al lote (*batch reduction*) de tipo *sum over batch size*.

Finalmente, añadir que la biblioteca incluye la opción de generar un modelo multifuncional, donde se usen las 3 cabezas a la vez en el entrenamiento,

dotando a cada una de un peso para la función de pérdida.

3.7. Preprocesamiento de datos DL1

Como la mayoría de los procesos relacionados con la gestión de datos en la biblioteca CTLearn, el preprocesamiento de los datos de entrenamiento, validación y test se lleva a cabo usando la biblioteca DLH, especificando en el archivo de configuración YAML las transformaciones necesarias en cada caso o usando aquellas establecidas por defecto. Uno de los principales retos con el tratamiento de datos proviene de la forma de las cámaras, hechas de secciones hexagonales de fotomultiplicadores. Estas formas no rectangulares son diferentes respecto a las formas rectangulares que esperan las capas de los modelos DCNN. Por ello, la biblioteca integra varios métodos de transformación o mapeo de estas distribuciones hexagonales a distribuciones rectangulares, de manera que se pierda la menor cantidad de información posible, tanto espacial como la presente en cada uno de los píxeles (carga acumulada y tiempo de exposición).

Los métodos principales implementados en la biblioteca CTLearn son: *oversampling*, *rebinning*, interpolación al más cercano (*nearest neighbour interpolation*, NN), interpolación bilineal e interpolación bicúbica (esquema en la figura 3.13). Una breve descripción de estos métodos se exponen a continuación (Miener, 2024, Capítulo 2.4.2):

- ***Oversampling***. Se divide cada píxel hexagonal en píxeles cuadrados de 2 por 2, asignando a la carga de los nuevos píxeles cuadrados la cuarta parte de la carga original, de forma que se preserva la carga total.

- ***Rebinning***. Se realiza un muestreo de la entrada, simulando un histograma 2D, y se redistribuye en cuadrados usando el algoritmo NN.

- **Interpolación al más cercano.** Calcula el píxel vecino más cercano en la estructura de entrada para cada píxel en la estructura de salida, y se usa para asignar la carga del píxel de salida mediante el método de interpolación al más cercano.

- **Interpolación bilineal.** Usa los 3 píxeles más cercanos de la imagen original para realizar una interpolación, usando el método *Delaunay triangulation* presente en la biblioteca SciPy (Virtanen et al., 2020).

- **Interpolación bicúbica.** Usa los 12 píxeles más cercanos de la imagen original para realizar una interpolación, usando el método *Delaunay triangulation* presente en la biblioteca SciPy (Virtanen et al., 2020).

Cabe señalar que todos los métodos comentados, a excepción de *oversampling*, permiten una dimensión de salida arbitraría.

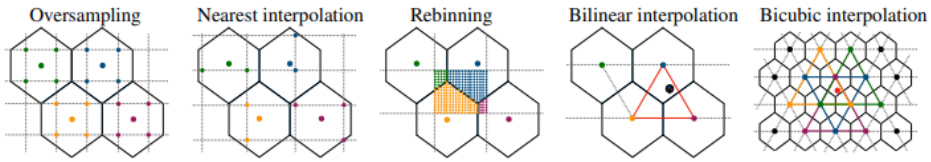


Figura 3.13: Métodos de mapeo de distribución hexagonal a rectangular presentes en CTLearn (Nieto, Brill, Feng, Jacquemont et al., 2019).

En los modelos por defecto el método de mapeo predominante es la interpolación bilineal, siguiendo los resultados obtenidos por Daniel Nieto y otros (Nieto, Brill, Feng, Jacquemont et al., 2019) donde encuentran cierta superioridad en las métricas al usar esta técnica frente al resto de las mostradas.

Capítulo 4

Metodología

4.1. Inclusión de ParticleNet

4.1.1. Modelo

Para el proceso de adaptación del modelo *ParticleNet* se ha partido de su versión oficial en GitHub (https://github.com/hqucms/ParticleNet), implementada en Tensorflow (Martín Abadi et al., 2015), concretamente en las versiones >=v2.0.0 o >=v1.15rc2. Para poder usarlo como modelo en CTLearn ha sido necesario actualizar y adaptar una parte de su código a la versión de Tensorflow v2.14.1, consistiendo este proceso fundamentalmente en modificar o eliminar funciones de Tensorflow que actualmente se encuentran integradas en `tensorflow.keras` u obsoletas. El código adaptado, además de presentarse en este documento, se encuentra en el repositorio de GitHub https://github.com/Olmichu22/Repo-TFM-OAP junto con el resto de código desarrollado (ver sección 4.1.2) y los archivos de configuración YAML para las configuraciones de este modelo.

Por otro lado, el conjunto del algoritmo ha sido adaptado para seguir la filosofía de CTLearn, que funciona con cualquier modelo de TensorFlow-Keras que siga la estructura de un *backbone model*. Esto significa que el modelo debe tener la forma:

```
backbone, backbone_inputs = backbone_model(data, model_params)
```

donde `backbone` es un (sub)modelo de TensorFlow-Keras con entradas `backbone_inputs`, `data` es una clase de tipo `KerasBatchGenerator`, y `model_params` es un diccionario de parámetros del modelo. De igual forma, se ha generado una estructura similar a la de los modelos implementados en CTLearn para la selección de hiperparámetros usando el archivo de configuración YAML.

Comenzando con la implementación, en el bloque de código 4.1 podemos ver las principales capas personalizadas usadas para crear la estructura de *ParticleNet*. De ellas, destacar `BatchDistanceMatrix`, que permite el cálculo de la matriz de distancias entre los puntos de una PC por lotes (*batches*), en un intento de optimizar el proceso. La clase `knn` utilizará estas distancias para expandir las características de cada punto de la nube con las de sus k vecinos más cercanos. Este proceso es el preparativo para poder aplicar la función $h_\Theta(x_i, x_{i_j} - x_i)$ comentada en la sección 3.5. Además de estas capas, encontramos algunas auxiliares cuya declaración era obligatoria para adaptar funciones obsoletas de Tensorflow a las nuevas versiones. Estas son `SqueezeLayer`, `ReduceLayer` y `BatchNormalLayerfts`.

Un último elemento a destacar son las clases `MaskCoordShiftLayer` y `AddShifttoCoordLayer`. Aunque en este modelo dichas capas no tienen influencia, su objetivo original es enmascarar puntos del conjunto de datos que no representen datos reales. La presencia de este tipo de puntos tiene como objetivo asegurar un tamaño fijo de matriz de características, ya que, aunque el modelo podría soportar entradas de tamaño variable, es necesaria esta homogeneidad para aplicar entrenamiento en paralelo.

Bloque de Código 4.1: Clases para crear capas de keras personalizadas.

```
class BatchDistanceMatrix(tf.keras.layers.Layer):
    """ Compute the distance matrix between points of a batch of point clouds"""

    def __init__(self, name='Initial_Batch_Distance_Matrix'):
        super(BatchDistanceMatrix, self).__init__(name=name)

    def call(self, inputs):
        A, B = inputs
        with tf.name_scope('dmat'):
            r_A = tf.reduce_sum(A * A, axis=2, keepdims=True)
            r_B = tf.reduce_sum(B * B, axis=2, keepdims=True)
            m = tf.matmul(A, tf.transpose(B, perm=(0, 2, 1)))
            D = r_A - 2 * m + tf.transpose(r_B, perm=(0, 2, 1))
            return D

class knn(tf.keras.layers.Layer):
    """ Get the k nearest neighbors of each point in a batch of point clouds """
```

```
18
19      def __init__(self, name='Initial_KNN', num_points =
    100, k = 20):
20          super(knn, self).__init__(name=name + "_k_"+str(k
    ))
21          self.num_points = num_points
22          self.k = k
23
24      def call(self, inputs):
25          topk_indices, features = inputs
26          with tf.name_scope('knn'):
27              queries_shape = tf.shape(features)
28              batch_size = queries_shape[0]
29              batch_indices = tf.tile(tf.reshape(tf.range(
    batch_size), (-1, 1, 1, 1)), (1, self.num_points,
    self.k, 1))
30              indices = tf.concat([batch_indices, tf.
    expand_dims(topk_indices, axis=3)], axis=3)
31              return tf.gather_nd(features, indices)
32
33  class ReduceLayer(tf.keras.layers.Layer):
34      """ Reduce the tensor """
35
36      def __init__(self, name='Initial_ReduceLayer',
    pooling = 'average', axis = 2):
37          super(ReduceLayer, self).__init__(name=name + "
    _pooling_"+pooling)
38          self.pooling = pooling
39          self.axis = axis
40
41      def call(self, inputs):
42          x = inputs
43          if self.pooling == 'max':
44              return tf.reduce_max(x, axis = self.axis)
45          else:
46              return tf.reduce_mean(x, axis = self.axis)
47
48  class SqueezeLayer(tf.keras.layers.Layer):
49      """ Squeeze the tensor """
50
51      def __init__(self, name='Initial_Squeeze_Layer', axis
    = 2):
52          super(SqueezeLayer, self).__init__(name=name +"
    _axis_"+str(axis))
53          self.axis = axis
54
55      def call(self, inputs):
56          x = inputs
57          return tf.squeeze(x, axis=self.axis)
```

```
58
59 class Get_Knn_fts(tf.keras.layers.Layer):
60     """ Get the relative coordinates of the knn points
       """
61
62     def __init__(self, name='Initial_Get_Knn_Fts', k =
       20):
63         super(Get_Knn_fts, self).__init__(name=name+"_k_"
       +str(k))
64         self.k = k
65
66     def call(self, inputs):
67         fts, knn_fts = inputs
68         knn_fts_center = tf.tile(tf.expand_dims(fts, axis
       =2), (1, 1, self.k, 1))
69         knn_fts = tf.concat([knn_fts_center, tf.subtract(
       knn_fts, knn_fts_center)], axis=-1)
70         return knn_fts
71
72 class Get_Knn_Index(tf.keras.layers.Layer):
73     """ Get the top k index of the distance matrix"""
74     def __init__(self, name='Initial_Top_K_Index', k =
       20):
75         super(Get_Knn_Index, self).__init__(name=name + "
       _k_"+str(k))
76         self.k = k
77
78     def call(self, inputs):
79         D = inputs
80         _, indices = tf.math.top_k(-D, k=self.k + 1)
81         return indices[:, :, 1:]
82
83 class MaskCoordShiftLayer(tf.keras.layers.Layer):
84     """ Mask the coordinate shift for non-valid points
85     Change the non-valid points to 999 in order to
       difficult the selection of them in the knn"""
86     def __init__(self, shift_value=999.):
87         super(MaskCoordShiftLayer, self).__init__(name =
       'MaskCoordShiftLayer')
88         self.shift_value = shift_value
89
90     def call(self, mask):
91         mask = tf.cast(tf.not_equal(mask, 0), dtype='
       float32')
92         coord_shift = tf.multiply(self.shift_value, tf.
       cast(tf.equal(mask, 0), dtype='float32'))
93         return coord_shift
94
95 class AddShifttoCoordLayer(tf.keras.layers.Layer):
```

```
96      """ Add the shift value to the coordinates """
97      def __init__(self, name):
98          super(AddShifttoCoordLayer, self).__init__(name =
        name)
99
100     def call(self, inputs):
101         coord_shift, points_or_fts = inputs
102         return tf.keras.layers.Add()([coord_shift,
        points_or_fts])
103
104 class BatchNormalLayerfts(tf.keras.layers.Layer):
105     def __init__(self, name):
106         super(BatchNormalLayerfts, self).__init__(name='%
        s_fts_bn' % name)
107         self.bn = tf.keras.layers.BatchNormalization()
108     def call(self, features):
109         return tf.squeeze(self.bn(tf.expand_dims(features
        , axis=2)), axis=2)
```

Por otro lado, en el bloque de código 4.2 se presenta el núcleo funcional del modelo: los bloques *EdgeConv*. Gracias a encapsular la mayoría de procesos en capas, se facilita el realizar un seguimiento del proceso de generación de este bloque, que comienza calculando los vecinos más cercanos para posteriormente aplicar el conjunto de capas convolucionales que modelan la función $h_\Theta(x_i, x_{i_j} - x_i)$. Cabe señalar que, por simplicidad, se han mantenido configuraciones del código original como la inicialización de los pesos o el nombre de las distintas capas.

Finalmente, en el bloque de código 4.3 se muestra la clase generadora del modelo `get_particle_net`, que sigue la filosofía de CTLearn para poder ser construido en lugar de los modelos por defecto. La función `_particle_net_base` configura el resto de elementos de *ParticleNet* y, si está incluido en las opciones, permite añadir la cabeza del modelo. Aunque se ha modelado y configurado el código para permitir personalización también en este último aspecto, no será una característica que usar desde CTLearn, ya que esta biblioteca estructura en secciones separadas el modelo *backbone* y la cabeza.

Bloque de Código 4.2: Función para la creación de los bloques EdgeConv.

```
1  def edge_conv(points, features, num_points, K, channels,
       with_bn=True, activation='relu', pooling='average',
       name='edgeconv'):
2      """EdgeConv
3      Args:
4          K: int, number of neighbors
5          channels: tuple of output channels
```

```
6           with_bn: bool, use batch normalization
7           activation: activation function
8           pooling: pooling method ('max' or 'average')
9           name: layer name
10      Inputs:
11          points: (N, P, C_p)
12          features: (N, P, C_0)
13      Returns:
14          transformed points: (N, P, C_out), C_out =
    channels[-1]
15      """
16      with tf.name_scope('edgeconv'):
17
18          D = BatchDistanceMatrix(name = name +"
    _Batch_Distance_Matrix")([points, points])
19
20          indices = Get_Knn_Index(name = name + "
    _top_k_index", k = K)(D)
21
22          fts = features
23          knn_fts = knn(name = name + "_KNN", num_points =
    num_points, k = K)([indices, fts])
24
25          x = Get_Knn_fts(name = name + "_Knn_fts", k = K)
    ([fts, knn_fts])
26          for idx, channel in enumerate(channels):
27              x = keras.layers.Conv2D(channel, kernel_size
    =(1, 1), strides=1, data_format='channels_last',
28                                      use_bias=False if
    with_bn else True, kernel_initializer='glorot_normal'
    , name='%s_conv%d' % (name, idx))(x)
29              if with_bn:
30                  x = keras.layers.BatchNormalization(name=
    '%s_bn%d' % (name, idx))(x)
31              if activation:
32                  x = keras.layers.Activation(activation,
    name='%s_act%d' % (name, idx))(x)
33
34          fts = ReduceLayer(name = name + "Reduce_Layer",
    pooling = pooling, axis = 2)(x, )
35
36
37
38          sc = keras.layers.Conv2D(channels[-1],
    kernel_size=(1, 1), strides=1, data_format='
    channels_last',
39                                   use_bias=False if
    with_bn else True, kernel_initializer='glorot_normal'
    , name='%s_sc_conv' % name)(tf.keras.backend.
```

```
    expand_dims(features, axis=2))
40          if with_bn:
41              sc = keras.layers.BatchNormalization(name='%
    s_sc_bn' % name)(sc)
42          sc = SqueezeLayer(name = name + "_SqueezeLayer",
    axis=2)(sc)
43
44          output = tf.keras.layers.Add(name = name + "
    _ResidualAddition")([sc, fts])
45          if activation:
46              return keras.layers.Activation(activation,
    name='%s_sc_act' % name)(output)
47          else:
48              return output
```

Bloque de Código 4.3: Funciones de creación de ParticleNet y su gestión para adaptarla a CTLearn.

```
1  def _particle_net_base(points, features=None, mask=None,
       model_params=None, name='particle_net'):
2      # points : (N, P, C_coord)
3      # features:  (N, P, C_features), opcional
4      # mask: (N, P, 1), opcional
5
6      with tf.name_scope(name):
7          if features is None:
8              features = points
9
10         if mask is not None:
11             shift_value = model_params.get("Mask",{}).get
       ("shift") if model_params.get("Mask",{}).get("shift")
        is not None else 999
12             coord_shift = MaskCoordShiftLayer(shift_value
       =shift_value)(mask)
13
14         fts = BatchNormalLayerfts(name = name)(features)
15
16         for layer_idx, layer_param in enumerate(
       model_params["EdgeConv"]["convparams"]):
17             K = layer_param["K"]
18             channels = layer_param["channels"]
19             if layer_idx == 0:
20                 pts = AddShifttoCoordLayer(name = "
       AddShifttoCoordLayer"+"_layer_"+str(layer_idx))([
       coord_shift, points])
21             else:
22                 pts = AddShifttoCoordLayer(name = "
       AddShifttoCoordLayer"+"_layer_"+str(layer_idx))([
       coord_shift, fts])
```

```
23
24              pooling = model_params.get("EdgeConv",{}).get
    ("pooling") if model_params.get("EdgeConv",{}).get("
    pooling") is not None else "average"
25              with_bn = model_params.get("EdgeConv",{}).get
    ("BN") if model_params.get("EdgeConv",{}).get("BN")
    is not None else True
26              activation = model_params.get("EdgeConv",{}).
    get("activation") if model_params.get("EdgeConv",{}).
    get("activation") is not None else "relu"
27              fts = edge_conv(pts, fts, model_params["
    num_points"], K, channels, with_bn = with_bn,
    activation = activation,
28                              pooling=pooling, name='%s_%s%
    d' % (name, 'EdgeConv', layer_idx))
29
30          if mask is not None:
31              fts = tf.keras.layers.Multiply(name = "
    Apply_Mask_Layer")([fts, mask])
32
33          final_pooling = model_params.get("FinalPooling")
    if model_params.get("FinalPooling") is not None else
    "average"
34          pool = ReduceLayer(name = "FinalPool", pooling =
    final_pooling, axis = 1)(fts)
35
36          if model_params.get("ConvHead") is not None:
37              x = pool
38              for layer_idx, layer_param in enumerate(
    model_params["ConvHead"]["params"]):
39                  units = layer_param["units"]
40                  drop_rate = layer_param["drop_rate"]
41                  activation = layer_param["activation"]
42                  x = keras.layers.Dense(units, activation=
    activation)(x)
43                  if drop_rate is not None and drop_rate >
    0:
44                      x = keras.layers.Dropout(drop_rate)(x
    )
45              out = keras.layers.Dense(model_params["
    num_classes"], activation='softmax')(x)
46              return out
47          else:
48              return pool
49
50  def get_particle_net(data, model_params=None):
51      r"""ParticleNet model from `"ParticleNet: Jet Tagging
     via Particle Clouds"
52      <https://arxiv.org/abs/1902.08570>`_ paper.
```

```
53
54      Based on https://github.com/hqucms/ParticleNet and
        adapted to CTLearn.
55      """
56
57      if model_params is None:
58          model_params = dict()
59          model_params["EdgeConv"] = dict()
60          model_params["EdgeConv"]["convparams"] = [{"K"
        :16, "channels":(64, 64, 64)}, {"K":16, "channels"
        :(128, 128, 128)}, {"K":16, "channels":(256, 256,
        256)}]
61      else:
62          sys.path.append(model_params["model_directory"])
63
64      if data.pc_pos is not None:
65          points = keras.Input(name='points', shape=data.
        input_shapes['points'])
66          features = keras.Input(name='features', shape=
        data.input_shapes['features']) if 'features' in data.
        input_shapes else None
67          mask = keras.Input(name='mask', shape=data.
        input_shapes['mask']) if 'mask' in data.input_shapes
        else None
68          model_params["num_points"] = data.input_shapes['
        npoints']
69          inputs = {"points" : points, "features": features
        , "mask": mask}
70      else:
71          raise ValueError("Data must have the point cloud
        position information to use ParticleNet")
72      outputs = _particle_net_base(points, features, mask,
        model_params, name='ParticleNet')
73      return keras.Model(inputs=inputs, outputs=outputs,
        name='ParticleNet'), inputs
```

De acuerdo con la filosofía de CTLearn, el modelo ha de permitir la parametrización por medio de los archivos de configuración YAML. Para cumplir este objetivo, se han añadido distintas opciones de personalización respecto a la versión original del código, visibles en los fragmentos de código 4.1, 4.2 y 4.3. Estas opciones se engloban dentro de la etiqueta `Model Parameters` y permiten modificar la estructura de la red como se expone a continuación:

- Opciones `EdgeConv`. Configuraciones para las capas *EdgeConv*.
  - `convparams` (lista de diccionarios).
    - `K`: Número de vecinos (**int**).

        - `channels`: Canales de salida en cada capa de convolución (**tupla de int**).
    - `pooling`: Operación pooling al final de la capa *EdgeConv* (`max` o `average`).
    - `BN`: Realizar *BatchNormalization* (`True` o `False`).
    - `activation`: Función de activación (`relu`, `sigmoid`, etc.)
- Opciones `Mask`.
    - `shift`: Valor de desplazamiento para los puntos no válidos (**float**, por defecto 999).
- `FinalPooling`: Método de pooling final (`max` o `average`).

Por otro lado, el modelo usará parámetros obtenidos de la clase encargada de la gestión de datos y lotes para el entrenamiento (exposición en la sección 4.1.2), tales como el número de puntos, el número de clases y el directorio del modelo.

4.1.2. Generación de nubes de puntos

Con el objetivo de adaptar las imágenes del telescopio al formato de datos de entrenamiento del modelo *ParticleNet*, es necesario introducir una capa adicional de preprocesamiento. Aunque esta tarea es realizada en CTLearn a través de la biblioteca DLH, se ha modificado el código de tal forma que la transformación se aplica en el momento de generar los *batches* para el entrenamiento o la predicción, de modo que, en caso de estar entrenando el modelo *ParticleNet*, el algoritmo transformará las imágenes en PC de longitud N, previamente indicada a través del archivo de configuración. Así pues, cada imagen de dimensión $(h \times w \times 2)$ se convierte en una matriz de dimensión $(N \times d)$, con d el número de características a conservar de cada punto, entre las que se incluyen las coordenadas de cada píxel.

La capa extra de preprocesamiento consiste en seleccionar los N píxeles de mayor intensidad, guardando las coordenadas (absolutas o relativas respecto al centro) de los píxeles de la matriz cuadrada como posiciones (x, y), además de la carga (intensidad del píxel) y el tiempo de pico (intensidad del píxel en la dimensión 2 de profundidad, ver sección 4.2). De esta forma, la transformación concreta sería de una matriz de dimensión $(h \times w \times 2)$ a otra de dimensión $(N \times 4)$, conteniendo información espacial y de características.

Complementado la sección 4.1.1, se muestra a continuación los métodos de preprocesamiento que permiten transformar una imagen de dimensiones arbitrarias $(h \times w \times 2)$ en la PC necesaria para trabajar con el modelo *ParticleNet*.

Como se ha mencionado anteriormente, el modelo incluye la gestión de puntos falsos para asegurar la homogeneidad en el tamaño de las matrices de características. Este aspecto se encuentra reflejado en la generación de la máscara `mask` dentro de la clase `ImagetoPointCloud`, que será 1 en las posiciones que se hayan de ignorar para el entrenamiento. De nuevo, se recuerda que esto es una opción que no se usa en este trabajo, pero que tiene potencial utilidad si se modifica el criterio de selección de píxeles a un método que modifique N para cada imagen.

Bloque de Código 4.4: Preprocesamiento a nube de puntos.

```
class ImagetoPointCloud():

    def __init__(self, max_points=500, relative_coords = True):
        """ Class for transforming a image dataset into a point cloud dataset
        Params:
        max_points (int): Number of points to select
        relative_coords (bool): If True, the coordinates are relative to the center of the image
        """
        self.max_points = max_points
        self.relative_coords = relative_coords

    def get_top_n_points_coords(self, image):
        # Aplanar la matriz y obtener los indices de los n mayores valores
        flat_indices = np.argpartition(image.flatten(), -self.max_points)[-self.max_points:]
        # Obtener las coordenadas (i, j) a partir de los indices planos
        coords = np.array(np.unravel_index(flat_indices, image.shape)).T
        # Obtener los valores correspondientes a los puntos seleccionados

        return coords

    def process_image(self, image):
        # Variables
        mask = np.ones((self.max_points, 1), dtype=np.float32)
        # Anadimos coordenadas a las caracteristicas
        features = np.zeros((self.max_points, 4), dtype=np.float32)
        points = np.zeros((self.max_points, 2), dtype=np.float32)
```

```
27
28          # Obtenemos los valores de los pixeles y el
    tiempo de captura
29          pixel_values = image[:,:,0].copy()
30          peak_time = image[:,:,1].copy()
31
32          # Obtenemos las coordenadas de los puntos con
    mayor intensidad
33          coords = self.get_top_n_points_coords(
    pixel_values)
34
35          n_points = len(coords[:,0])
36          if n_points < self.max_points:
37              # Coordenadas
38              if self.relative_coords:
39                  coords = coords - self.center
40              points[:n_points, 0], points[:n_points, 1] =
     coords[:, 0], coords[:, 1]
41              points[n_points:,:] = 0
42
43              mask[n_points:] = 0
44
45              features[:n_points, 0] = pixel_values[coords
    [:, 0], coords[:, 1]]
46              features[:n_points, 1] = peak_time[coords[:,
    0], coords[:, 1]]
47              features[n_points:, 2] = coords[:, 0]
48              features[n_points:, 3] = coords[:, 1]
49              features[n_points:,:] = 0
50
51
52
53          elif n_points == self.max_points:
54              # Coordenadas
55              if self.relative_coords:
56                  coords = coords - self.center
57              points[:, 0], points[:, 1] =  coords[:, 0],
    coords[:, 1]
58
59              features[:,0] = pixel_values[coords[:, 0],
    coords[:, 1]]
60              features[:, 1] = peak_time[coords[:, 0],
    coords[:, 1]]
61              features[:, 2] = coords[:, 0]
62              features[:, 3] = coords[:, 1]
63
64
65
66          else:
```

```
67              raise ValueError(f"Number of selected points
    is greater than max_points {self.max_points}. Number
    of selected points : {n_points}")
68
69          return {"features": features, "points": points, "
    mask": mask}
70
71      def reconstruct_image(self, features, points, mask):
72          """ Auxiliar function for reconstruct the image
    from the point cloud.
73          Args:
74              features (np.array): Array of features with
    shape (n_points, n_features)
75              points (np.array): Array of points with shape
     (n_points, 2)
76              mask (np.array): Array of mask with shape (
    n_points, 1)
77          Returns:
78              np.array: Reconstructed image with shape (
    height, width, 2)
79          """
80          # Variables
81          image = np.zeros((self.image_dims[0], self.
    image_dims[1], 2), dtype=np.float32)
82          # Reconstruimos la imagen
83          n_points = int(np.sum(mask))
84          if n_points == 0:
85              return image
86
87          pixel_values = features[:n_points, 0]
88          peak_time = features[:n_points, 1]
89          coords = points[:n_points]
90
91          if self.relative_coords:
92              coords = coords + self.center
93
94          coords = coords.astype(int)
95          image[coords[:, 0], coords[:, 1], 0] =
    pixel_values
96          image[coords[:, 0], coords[:, 1], 1] = peak_time
97
98          return image
99
100     def __call__(self, images):
101         """ Call method for transform a image dataset
    into a point cloud dataset
102
103         Args:
104             images (np.array): Array of images with shape
```

```
    (n_images, height, width, channels) or (height,
width, channels)
105
106         Returns:
107             dict: Dictionary with the keys "features", "
points" and "mask" with the values of the point cloud
 as np.array
108         """
109         if len(images.shape) > 3:
110             points_cloud = {"features": [], "points": [],
 "mask": []}
111             for image in images:
112                 self.image_dims = image[:,:,0].shape
113                 if self.relative_coords:
114                     self.center = (self.image_dims[0] //
2, self.image_dims[1] // 2)
115                 processed_image = self.process_image(
image)
116                 points_cloud["features"].append(
processed_image["features"])
117                 points_cloud["points"].append(
processed_image["points"])
118                 points_cloud["mask"].append(
processed_image["mask"])
119             for key in points_cloud.keys():
120                 points_cloud[key] = np.array(points_cloud
[key])
121             return points_cloud
122         else:
123             self.image_dims = images[:,:,0].shape
124             if self.relative_coords:
125                 self.center = (images[:,:,0].shape[0] //
2, images[:,:,0].shape[1] // 2)
126             points_cloud = self.process_image(images)
127         return points_cloud
```

Como se ha expuesto, este proceso de transformación se lleva a cabo en el momento de generar los lotes de entrenamiento. En el bloque de código 4.5 se pueden observar los cambios fundamentales del código original, que consisten en añadir un diccionario de opciones al generador de lotes, que al ser detectado ejecuta una transformación basada en la configuración deseada antes de devolver el conjunto de datos de entrenamiento y valores objetivo o etiquetas. Esta transformación se realiza con la clase ya expuesta en el bloque de código 4.4, puesto que permite el procesamiento por lotes.

Bloque de Código 4.5: Modificaciones en la clase KerasBatchGenerator del archivo dataloader.py para la transformación a PC. Las lineas introducidas se marcan con un + y las eliminadas con un -.

```
 import numpy as np
 import tensorflow as tf
+from .preprocessing import ImagetoPointCloud
 import astropy.units as u

 class KerasBatchGenerator(tf.keras.utils.Sequence):
     "Generates batches for Keras application"

     def __init__(
         self,
         DLDataReader,
         indices,
         batch_size=64,
         mode="train",
         class_names=None,
         shuffle=True,
         stack_telescope_images=False,
+        to_pc_options = dict({"point_cloud": True, "
   max_points": 500, "relative_coords": True, "
   n_features": 2, "n_coords": 2}),
     ):
+        # Cambios PC en la entrada
         "Initialization"
         self.DLDataReader = DLDataReader
         self.batch_size = batch_size
         self.indices = indices
         self.mode = mode
         self.class_names = class_names
         self.shuffle = shuffle
         self.stack_telescope_images =
            stack_telescope_images
+        self.to_pc_options = to_pc_options
+
         self.on_epoch_end()
+        print("Opciones pc \n", self.to_pc_options)

+        # PC Config (CAMBIOS PC)
+        if self.to_pc_options["point_cloud"]:
+            self.input_shapes = dict()
+            self.pc_pos = 0
```

```
38 +            max_points = self.to_pc_options.get("
   max_points", 500)
39 +            self.input_shapes['features'] = (
   max_points, to_pc_options.get('n_features',4))
40 +            self.input_shapes['points'] = (
   max_points, to_pc_options.get('n_coords',2))
41 +            self.input_shapes['mask'] = (max_points,
    1)
42 +            self.input_shapes["npoints"] =
   max_points
43 +
44          # Decrypt the example description
45          # Features
46          self.singleimg_shape = None
47          self.trg_pos, self.trg_shape = None, None
48          self.trgpatch_pos, self.trgpatch_shape =
               None, None
49          self.pon_pos = None
50          self.pointing = []
51          self.wvf_pos, self.wvf_shape = None, None
52          self.img_pos, self.img_shape = None, None
53          self.prm_pos, self.prm_shape = None, None
54 class KerasBatchGenerator(tf.keras.utils.Sequence):
55          # Additional info
56          self.evt_pos, self.obs_pos = None, None
57          self.event_list, self.obs_list = [], []
58          self.mjd_pos, self.milli_pos, self.nano_pos
               = None, None, None
59          self.mjd_list, self.milli_list, self.
               nano_list = [], [], []
60          # Labels
61          self.prt_pos, self.enr_pos, self.drc_pos =
               None, None, None
62          self.drc_unit = u.deg
63          self.prt_labels = []
64          self.enr_labels = []
65          self.az_labels, self.alt_labels, self.
               sep_labels = [], [], []
66          self.trgpatch_labels = []
67          self.energy_unit = "log(TeV)"
68
69          for i, desc in enumerate(self.DLDataReader.
               example_description):
70              if "HWtrigger" in desc["name"]:
```

```
                    self.trg_pos = i
                    self.trg_shape = desc["shape"]
                elif "pointing" in desc["name"]:
                    self.pon_pos = i
                elif "waveform" in desc["name"]:
                    self.wvf_pos = i
                    self.wvf_shape = desc["shape"]
class KerasBatchGenerator(tf.keras.utils.Sequence):
                    self.energy_unit = desc["unit"]
                elif "direction" in desc["name"]:
                    self.drc_pos = i
                    self.drc_unit = desc["unit"]
                elif "event_id" in desc["name"]:
                    self.evt_pos = i
                elif "obs_id" in desc["name"]:
                    self.obs_pos = i
                elif "mjd" in desc["name"]:
                    self.mjd_pos = i
                elif "milli_sec" in desc["name"]:
                    self.milli_pos = i
                elif "nano_sec" in desc["name"]:
                    self.nano_pos = i

            # Retrieve shape from a single image in
               stereo analysis
            if self.trg_pos is not None and self.img_pos
                is not None:
class KerasBatchGenerator(tf.keras.utils.Sequence):
            # Generate data
            for i, index in enumerate(batch_indices):
                event = self.DLDataReader[index]
                # Fill the features
                if self.trg_pos is not None:
                    triggers[i] = event[self.trg_pos]
class KerasBatchGenerator(tf.keras.utils.Sequence):
                    if self.prt_pos is not None:
                        particletype[
                            i
                        ] = self.DLDataReader.
                           shower_primary_id_to_class[
                            int(event[self.prt_pos])
                        ]
                    if self.enr_pos is not None:
class KerasBatchGenerator(tf.keras.utils.Sequence):
```

```
112                     if self.prt_pos is not None:
113                         self.prt_labels.append(np.
                               float32(event[self.prt_pos]))
114                     if self.enr_pos is not None:
115                         self.enr_labels.append(np.
                               float32(event[self.enr_pos]))
116                     if self.drc_pos is not None:
117                         self.az_labels.append(np.float32
                               (event[self.drc_pos][0]))
118                         self.alt_labels.append(np.
                               float32(event[self.drc_pos
                               ][1]))
119                         self.sep_labels.append(np.
                               float32(event[self.drc_pos
                               ][2]))
120                     if self.trgpatch_pos is not None:
121                         self.trgpatch_labels.append(np.
                               float32(event[self.
                               trgpatch_pos]))
122                     # Save pointing
123                     if self.pon_pos is not None:
124                         self.pointing.append(event[self.
                               pon_pos])
125                     # Save all parameters for the
                           prediction phase
126                     if self.prm_pos is not None:
127                         self.parameter_list.append(event
                               [self.prm_pos])
128 class KerasBatchGenerator(tf.keras.utils.Sequence):
129                         self.event_list.append(np.
                               float32(event[self.evt_pos]))
130                     if self.obs_pos is not None:
131                         self.obs_list.append(np.float32(
                               event[self.obs_pos]))
132                     # Save timestamp
133                     if self.mjd_pos is not None:
134                         self.mjd_list.append(np.float32(
                               event[self.mjd_pos]))
135                     if self.milli_pos is not None:
136                         self.milli_list.append(np.
                               float32(event[self.milli_pos
                               ]))
137                     if self.nano_pos is not None:
138                         self.nano_list.append(np.float32
```

```
                        (event[self.nano_pos]))
139
140         features = {}
141         if self.trg_pos is not None:
142 class KerasBatchGenerator(tf.keras.utils.Sequence):
143         if self.prm_pos is not None:
144             features["parameters"] = parameters
145
146 +       # Cambios PC
147 +       if self.to_pc_options["point_cloud"]:
148 +           if "images" not in features.keys():
149 +               raise ValueError("Images are
    required to convert to point cloud")
150 +
151 +           self.input_shapes = dict()
152 +           max_points = self.to_pc_options.get("
    max_points", 500)
153 +           relative_coords = self.to_pc_options.get
    ("relative_coords", True)
154 +           features_pc = ImagetoPointCloud(
    max_points = max_points, relative_coords =
    relative_coords)(images)
155 +           self.input_shapes['features'] = (
    features_pc['features'].shape[1], features_pc['
    features'].shape[2])
156 +           self.input_shapes['points'] = (
    features_pc['points'].shape[1], features_pc['
    points'].shape[2])
157 +           self.input_shapes['mask'] = (features_pc
    ['mask'].shape[1], features_pc['mask'].shape[2])
158 +           self.input_shapes["npoints"] =
    max_points
159 +
160         labels = {}
161         if self.mode == "train":
162             if self.prt_pos is not None:
163                 labels["type"] = tf.keras.utils.
                    to_categorical(
164                     particletype,
165                     num_classes=self.DLDataReader.
                        num_classes,
166                 )
167                 label = tf.keras.utils.
                    to_categorical(
```

```
168                         particletype,
169                         num_classes=self.DLDataReader.
                               num_classes,
170                     )
171                 if self.enr_pos is not None:
172                     labels["energy"] = energy.reshape((-
                           1, 1))
173 class KerasBatchGenerator(tf.keras.utils.Sequence):
174                 if self.drc_pos is not None:
175                     labels["direction"] = direction
176                     label = direction
177                 if self.trgpatch_pos is not None and
                       self.DLDataReader.
                       reco_cherenkov_photons:
178                     labels["cherenkov_photons"] =
                           trigger_patches_true_image_sum
179                     label =
                           trigger_patches_true_image_sum
180
181             # https://github.com/keras-team/keras/issues
                   /11735
182             if len(labels) == 1:
183                 labels = label
184 -
185 -           return features, labels
186 +
187 +           # CAMBIOS PC
188 +           if self.to_pc_options["point_cloud"]:
189 +               return features_pc, labels
190 +           else:
191 +               return features, labels
```

Finalmente, destacar el uso del diccionario de opciones `to_pc_options`, que se declara en el archivo de configuración YAML, en la sección `Input`, y permite personalizar la transformación de imagen a PC:

- Opciones `to_pc_options`. Configuraciones para la conversión a PC.
  - `point_cloud`: Indica si se debe convertir las imágenes a PC (`True` o `False`).
  - `max_points`: Número máximo de puntos en la PC (**int**, por defecto 500).

- `relative_coords` : Indica si las coordenadas deben ser relativas (`True` o `False`).

- `n_features` : Número de características por punto en la nube (**int**, por defecto 2).

- `n_coords` : Número de coordenadas espaciales por punto (**int**, por defecto 2).

4.2. Datos

Las datos que se usarán para entrenar el modelo TRN son las imágenes resultantes de la simulación de eventos de CR en la atmósfera terrestre, usando el software CORSIKA, posteriormente procesadas para obtener imágenes cuadradas de tamaño $(114 \times 114 \times 2)$. El método para obtener estos datos es la interpolación bilineal presente en la sección 3.7. Para esto, no solo se usa la información referente a la imagen, sino también otra serie de características tales como la dimensión del telescopio, indicando el nombre de este en los archivos de configuración YAML y extrayendo la información necesaria de los archivos de tipo DL1 que contienen los datos de los telescopios de Cherenkov.

La información presente en estas imágenes se encuentra dividida en dos canales, conteniendo el primero la intensidad de señal acumulada (fotoelectrones) en cada píxel, y el tiempo de pico de esta señal (en ns) en el segundo. En las figuras 4.1 a 4.3 se pueden observar algunos ejemplos de las imágenes usadas para el entrenamiento, para cascadas producidas por gammas, protones y electrones respectivamente.

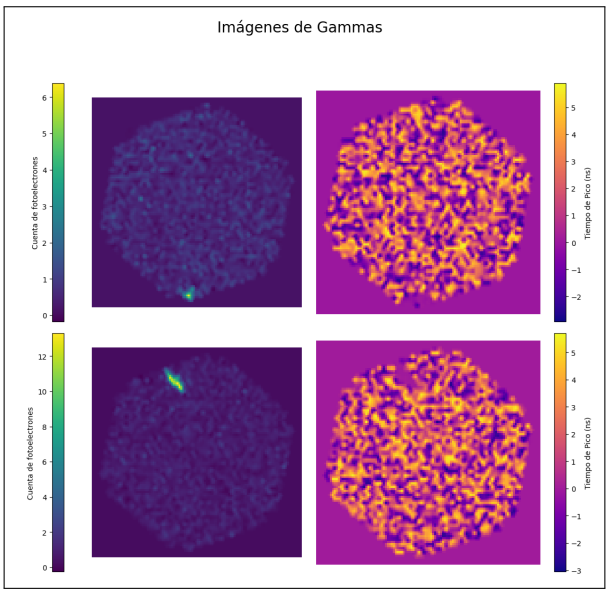


Figura 4.1: Imágenes de eventos de tipo gamma. La columna izquierda muestra la intensidad de señal acumulada en cada píxel, mientras que la columna derecha muestra el tiempo de pico de esta señal.

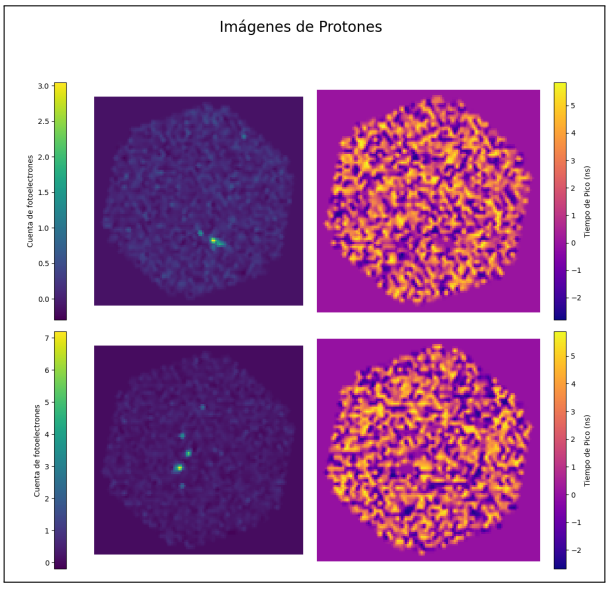


Figura 4.2: Imágenes de eventos de tipo hadrónico (protones). La columna izquierda muestra la intensidad de señal acumulada en cada píxel, mientras que la columna derecha muestra el tiempo de pico de esta señal.

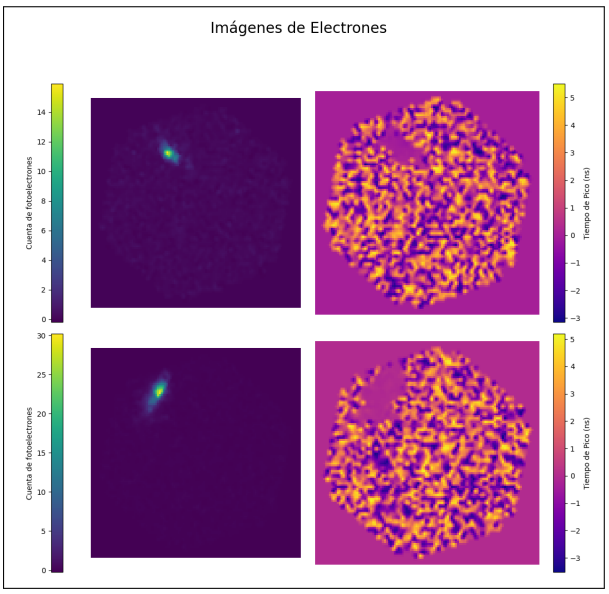


Figura 4.3: Imágenes de eventos producidos por un electrón. La columna izquierda muestra la intensidad de señal acumulada en cada píxel, mientras que la columna derecha muestra el tiempo de pico de esta señal.

Como se puede apreciar, no existe una diferencia clara entre las imágenes de los distintos tipos de eventos que pudiese facilitar la discriminación entre gammas y el resto, siendo además las imágenes de los electrones similares a los eventos de tipo gamma. Otro aspecto que señalar es el desplazamiento del tiempo de pico de la señal, de forma que el 0 coincide con el píxel de mayor intensidad, y los valores negativos y positivos indican momentos anteriores y posteriores a este. Si entendemos la señal como una onda, el tiempo de pico es el momento en el que esta alcanza su máximo.

Por otro lado, es interesante observar las distribuciones de los dos canales de las imágenes. Debido a la gran cantidad de datos de entrenamiento, y el número de píxeles por imagen, las gráficas de densidad se han obtenido usando una muestra de 500 imágenes de cada tipo de partícula. En estas distribuciones (figura 4.4) es posible comenzar a visualizar diferencias entre los distintos tipos de evento, tales como una bajada en la cuenta de fotoelectrones por parte de los eventos de tipo gamma más rápida que los electrones o los protones , que mantienen una densidad constante hasta aproximadamente las decenas. A partir de este orden, el comportamiento es muy similar, hasta aproximadamente los 500 fotoelectrones, donde los eventos de tipo gamma desaparecen, y se mantienen los de tipo electrónico y hadrónico. Igualmente, se observa una cola superior mayor en los eventos producidos por protones que en los producidos por electrones.

Observando la gráfica inferior, que muestra la distribución de los tiempos

de pico de la señal, se puede observar que los eventos de tipo gamma tienen una distribución más centrada en torno al 0, mientras que los eventos de tipo electrónico y hadrónico presentan una distribución más dispersa, con una cola inferior más larga en el caso de los eventos de tipo hadrónico. Por otro lado, se aprecia como la distribución positiva es prácticamente idéntica en los tres tipos de eventos.

Un aspecto final que merece destacarse es la presencia de un número significativo de píxeles inactivos o con valores muy bajos de detección. Este fenómeno se explica principalmente por el hecho de que los píxeles de relleno en ambos canales tienen un valor de 0 (como se muestra en las figuras 4.1 a 4.3). Además, en el primer canal, las zonas con mayor concentración de detección de fotoelectrones se encuentran principalmente localizadas en islas. Aunque esto es una característica útil para la red neuronal, ya que debería proporcionar una mayor facilidad para detectar la zona de importancia del evento, también supone un aumento en la dimensionalidad y la introducción de elementos no útiles en los datos.

Este aspecto supone una ventaja en el caso de que el modelo *ParticleNet* obtenga resultados competitivos, ya que nos asegura que podemos reducir considerablemente la dimensión de los datos de entrada, sin que esto perjudique la calidad de la información.

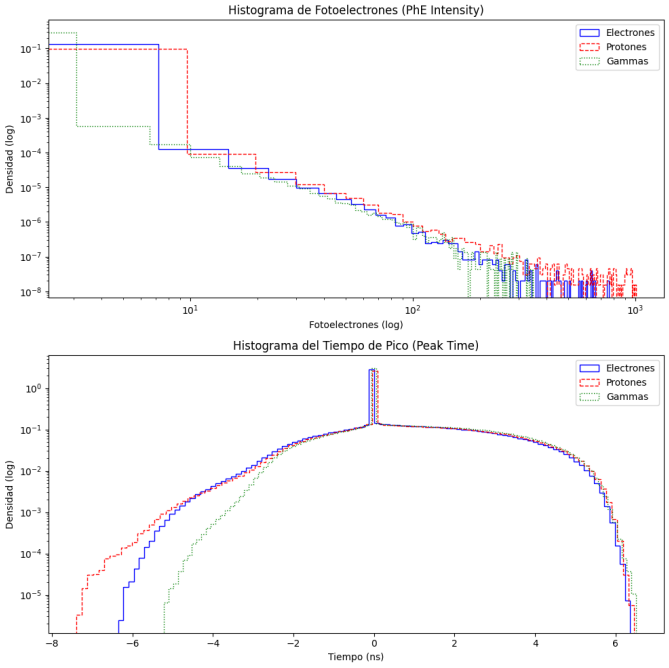


Figura 4.4: Funciones de densidad para la intensidad y los tiempos del evento en una muestra de 500 imágenes de cada tipo de evento.

Respecto a las etiquetas y valores numéricos usados como *ground truth*, su formato depende de la tarea a realizar. Para la clasificación, estos datos son almacenados originalmente con un código: 0 para gammas, 1 para electrones y 101 para protones. Posteriormente, estos valores son convertidos a un formato *one-hot* para el entrenamiento (ignorando electrones), de forma que la salida de la capa *Softmax* será considerada una puntuación de *gammaness* o parecido a un evento gamma.

La reconstrucción de energía se realiza usando como valor objetivo la energía en TeV del evento, a la que se le aplica una transformación logarítmica para convertirla a valores más manejables para la red.

Finalmente, la reconstrucción de la dirección se realiza usando como valores objetivo las coordenadas altura y ángulo azimutal a las que ocurrió el evento (coordenadas horizontales) y la separación angular al evento, en grados sexagesimales. Este último valor se corresponde a la separación del lugar del evento y hacia donde apunta el telescopio, y ha sido añadido en la versión v0.9.0 de CTLearn (Brill et al., 2024), usando la biblioteca Astropy para su cálculo, permitiendo a los modelos considerar un apuntado arbitrario del telescopio.

4.3. Entrenamiento

El entrenamiento del conjunto de los modelos ha sido realizado usando el *framework* que proporciona CTLearn. Además de las opciones de entrenamiento generales, que se exponen posteriormente, cabe añadir que la metodología general de entrenamiento incluye varios sistemas de control (*callbacks*) de TensorFlow-Keras para que el modelo aprenda de forma adaptable y eficiente.

Estos sistemas y sus configuraciones son los siguientes:

- **ModelCheckpoint Callback**. Asegura el guardado del modelo cuando se observa una mejora en el rendimiento, atendiendo a una métrica concreta. En este caso, la métrica es `val_loss`, de forma que se guardará el modelo siempre que, tras terminar una época, esta se reduzca.
- **CSVLogger Callback**. Registra los resultados del entrenamiento en un archivo CSV, facilitando su análisis posterior.
- **ReduceLROnPlateau Callback**. Ajusta la tasa de aprendizaje durante el entrenamiento, reduciéndola cuando la métrica (`val_loss`) no disminuye después de un número determinado de épocas (`patience`, que en este caso es 5). La tasa de aprendizaje se reduce por un factor de 0.5, y el cambio mínimo en `val_loss` para que se considere una

mejora queda determinado por `min_delta = 0.01`. El límite inferior para la tasa de aprendizaje, `min_lr`, se establece como el 10 % de la tasa de aprendizaje inicial.

Para actualizar los pesos de las neuronas se usa el optimizador Adam, con una tasa de aprendizaje inicial (`learning_rate`) de 0.0001 y un `epsilon` de 10^{-8}.

En cuanto a las opciones de entrenamiento, la parte común del comando usado para entrenar los modelos es la siguiente:

Bloque de Código 4.6: Comando de entrenamiento de los modelos.

```
ctlearn -d TRN -i ".../TrainingDataset/GammasDiffuse/" ".../
    TrainingDataset/Protons/" \
-p "*16.087_az_108.090*" "proton_theta_16.087_az_251.910*" "
    proton_theta_9.579_az_126.888*" "proton_theta_23.161_az_99.261*" \
-o "..." \
--clean \
-z 50 \
-e 8 \
-t LST_LST_LSTCam \
-b 64 \
--log_to_file \
-r type
```

donde se destacan las 8 épocas de entrenamiento (posteriormente se podrá observar que coincide con el codo de la curva de entrenamiento) y un umbral *low-cut* (>50 phe) para la preselección de eventos. Además, se ha añadido la opción `--clean` para llevar a 0 los píxeles que son ruido, y la opción `--log_to_file` para que se guarde un archivo de registro con los resultados del entrenamiento.

Los datos de entrenamiento serán una selección de eventos gamma y hadrónicos de simulaciones de cascadas atmosféricas, usando el conjunto de los datos para el entrenamiento de los modelos que diferencian el origen de la cascada (`-r type`) y solamente los eventos gamma para el entrenamiento de los modelos que reconstruyen la energía y la dirección (`-r energy` y `-r direction`), al estar interesados exclusivamente en obtener estos valores para los eventos con este origen (ver sección 3.1).

Para entrenar se usarán $2{\cdot}10^6$ eventos $(2,163,313)$ divididos aproximadamente a la mitad en tipo gammas $(1,061,475)$ y tipo hadrónico $(1,101,838)$. De estos, un 10 % se usará para la validación del modelo, y el resto para el entrenamiento. La división de los datos se ha realizado de forma aleatoria, asegurando que la proporción de eventos gamma y hadrónicos sea la misma en ambos conjuntos. El leve desbalanceo existente se traduce en unos pesos para la función de pérdida de aproximadamente 1.019 para los eventos gamma y 0.981 para los hadrónicos.

4.4. Predicción

Una vez entrenados los modelos, la predicción se ejecuta usando el siguiente comando:

Bloque de Código 4.7: Comando de predicción de los modelos.

```
ctlearn \
-d TRN \
-i "TestDataset/.../" \
-y /.../ \
-o "/.../" \
-t LST_LST_LSTCam \
-b 64 \
--clean \
--log_to_file \
--mode predict \
-r type \
```

donde se ha usado `--mode predict` para indicar que se quiere realizar una predicción y, como ejemplo, `-r type` para indicar que se quiere predecir el tipo de partícula, siendo las otras dos opciones `direction` (dirección) y `energy` (energía). El significado de cada uno de los parámetros se ha explicado en la sección 3.3.

Un elemento diferenciable de otros tipos de problemas donde se resuelven tareas de clasificación es el uso de datos para el *test* que no han sido usados para el entrenamiento, como se realiza en el análisis convencional (Miener, 2024, Capítulo 2.6.3). Concretamente, estos datos son los electrones mostrados en la sección 4.2, ya que se tratan de eventos muy similares a los gamma (ver sección 3.1), representando una dificultad para el entrenamiento del modelo, y esperando que este no sea capaz de discriminarlos correctamente.

El conjunto de datos test está conformado por $3.5 \cdot 10^6$ (3515048) eventos, de los cuales aproximadamente un 86.02 % son eventos iniciados por gammas (3023659), un 7.96 % (279912) por electrones y un 6.02 % (211477) por protones.

4.5. Figures of Merit

Para estudiar la calidad de los resultados producidos por las distintas redes neuronales es común el uso de figuras que miden las Instrument Response Functions (IRFs), usando para ello las funciones de la biblioteca pyirf (Dominik et al., 2023) desde el *framework* de CTLearn.

El eje horizontal en el conjunto de las figuras se corresponde con la energía, encontrando por un lado la energía reconstruida por el modelo E_{reco} o

E_R, y la energía real E_{True}, en TeV (Teraelectronvoltios) y escala logarítmica, para cubrir un mayor rango de energías.

Estas figuras se generan a partir de los archivos que contienen las predicciones de todos los datos necesarios para la reconstrucción de eventos (tipo, dirección y energía), que han de estar contenidos en un mismo archivo HDF5. Para generar los datos que permiten dibujar estas figuras se ha hecho uso del siguiente comando:

Bloque de Código 4.8: Comando para generar las *figures of merit*.

```
build_irf \
--input "..." \
--output "..." \
--energy_range 0.02 20.0 \
--size_cut 50 \
--energy_dependent_gh_efficiency 0.7
```

donde `input` es el directorio que contiene el conjunto de predicciones (fotones, hadrones y electrones), cada uno en un archivo distinto, y `output` es el directorio donde se guardarán los datos para representar las IRFs. Además, se ha establecido un umbral de 50 phe para la preselección de eventos, y se ha fijado el corte de *gammaness* en 0.7 para la discriminación de eventos.

Se pueden distinguir 5 tipos de gráficas dependiendo del tipo de sensibilidad del modelo que se quiera estudiar. Dependiendo del gráfico, el eje vertical puede representar diferentes valores:

- Sensibilidad Diferencial $E^2F_{Thr}[TeVcm^{-2}s^{-1}]$. Sensibilidad necesaria para detectar un flujo mínimo y significativo en 50 horas.
- Área Efectiva $[m^2]$. Representa la eficiencia del telescopio en función de la energía. Es la superficie equivalente sobre la que debería captarse toda la radiación para obtener la misma ganancia de telescopio que el telescopio real, siendo la ganancia la respuesta del telescopio a una fuente puntual.
- Tasa de Fondo $[Hz]$. Cuántos eventos de fondo (*background*) se detectan por segundo.
- Resolución Angular $[deg]$. Mide la precisión con la que un instrumento puede localizar una fuente en el cielo, de forma que un valor menor indica una mayor exactitud en esta localización.
- Resolución de Energía. Indica la precisión en la determinación de la energía de los eventos detectados, donde un valor menor refleja una mayor exactitud en la estimación de la energía de los rayos gamma observados.

Por otro lado, podemos dividir las curvas mostradas en curvas de sensibilidad, que muestran la capacidad del telescopio para detectar señales débiles en función de la energía, y curvas de resolución, que permiten entender la capacidad del sistema de medir las características físicas de los distintos eventos.

Las barras de error en estos gráficos representan la incertidumbre asociada con las estimaciones o mediciones de la energía reconstruida o la real. Estas se deben a la resolución limitada del telescopio al medir la energía de los fotones o partículas, siendo concretamente la mitad del ancho del *bin* de energía. Estos *bins* se definen principalmente mediante una distribución logarítmica, usando la función *create_bins_per_decade* de pyirf para cubrir el rango de energías de interés. El uso de *bins* para generar IRFs se debe a que los datos obtenidos de simulaciones y observaciones son discretos, y agruparlos reduce el ruido estadístico y facilita la comparación entre diferentes conjuntos de datos. Los *bins* permiten calcular promedios y desviaciones estándar (estadísticas más robustas) además de hacer que las representaciones visuales sean más claras.

La sensibilidad del proceso de reconstrucción dependerá de la precisión y capacidad de cada uno de los modelos para realizar su tarea correctamente. Por un lado, el modelo encargado de clasificar el tipo de evento (partícula) que inicia la cascada atmosférica proporcionará una muestra más o menos pura (libre de ruido) para la reconstrucción geométrica y energética. Por otro lado, los modelos de reconstrucción de energía y dirección permitirán obtener una estimación de la energía y la dirección de llegada de los supuestos fotones, de forma que la calidad de los resultados dependerá directamente de la calidad de las predicciones de los modelos.

4.6. Data Lake

Los *data lakes* suelen asociarse con Apache Hadoop (Apache Software Foundation, 2024), un marco de software de código abierto que proporciona un procesamiento distribuido fiable y de bajo coste para almacenamiento de big data. Entre sus beneficios encontramos:

- **Mayor flexibilidad**. Los *data lakes* pueden ingerir conjuntos de datos estructurados, semiestructurados y no estructurados.
- **Coste**. Menos inversión en recursos humanos. Costes de almacenamiento menores.
- **Escalabilidad.**
- **Reducción de los silos de datos.**

Aunque entre sus desventajas se encuentra el rendimiento y el gobierno de los datos, que requiere un mayor trabajo en el diseño, presentan una solución potencialmente útil para facilitar el almacenamiento de datos de los telescopios de Cherenkov y CTA en todos los niveles de análisis, y la integración de las herramientas que se presentan en este trabajo, especialmente de CTLearn.

Los elementos clave que compondrían el *Data Lake* son:

- Almacenamiento de Datos:
  - *Data Lake Storage*. Almacenamiento escalable y duradero, como Amazon S3 o Azure Data Lake Storage.
  - Datos Brutos. Almacena los datos originales sin procesar del telescopio (DL0)
  - Datos Procesados. Almacena las transformaciones y análisis sucesivos de los datos brutos en formato h5. Para obtener estos archivos, se usaría la biblioteca ctapipe en conjunto con CTLearn, Astropy y DLH, permitiendo la transformación de los datos DL0 a DL3 y el almacenamiento de cada uno de sus estados intermedios.
- Ingesta de Datos. Algunas opciones son Apache NiFi, AWS FLue o Azure Data Factory. En el proceso de ingesta los datos deben ser establecidos los *Table Formats* de cada archivo, construcciones de metadatos que permiten gestionar los archivos como tablas de SQL.
- Procesamiento de Datos.
  - ETL (Extract, Transform, Load). Configuración de los procesos ETL para transformar y cargar los datos en su forma procesada. La gestión de este proceso se puede realizar con la herramienta DLH, de manera que se separa la gestión y el procesamiento de datos del módulo de entrenamiento de los datos, aislando así las distintas etapas del uso de los datos para la creación de modelos de ML, facilitando la solución de errores y el mantenimiento de los módulos.
  - Herramientas de Procesamiento. En este caso, la propuesta es usa Apache Spark, debido a su compatibilidad con Python usando la biblioteca PySpark. De esta forma, se facilita el uso de DLH para la gestión de grandes volúmenes de datos.
- Metadatos y Gestión de Datos.
  - Catálogo de Datos. Se implementa un catálogo de datos como AWS Glue Data Catalog, Apache Atlas, o Azure Purview para la gestión de metadatos.

  - Gestión de Esquemas. Usar *Table Formats* como Apache Iceberg para manejar la evolución del esquema y las transacciones ACID (*Atomicity, Consistency, Isolation, Durability*).

- Seguridad y Gobernanza. De cara a mantener la privacidad de los datos más sensibles y poder permitir acceso a datos liberados para el uso de público en general o investigador, se deberá de gestionar un control de acceso usando un sistema IAM (Identity and Access Management). Por otro lado, se deberán implementar auditorías, encargadas del *Logging* y *Access History*, de manera periódica, para asegurar que se cumplen las normativas de seguridad y gobernanza. De forma complementaría, se implementaría un sistema de monitoreo de seguridad y de rendimiento, con el objetivo de supervisar a tiempo real actividades sospechosas o no autorizadas, y evitar problemas de rendimiento tales como cuellos de botella, altas latencias u otros problemas que puedan afectar al rendimiento del *Data Lake*.

- Análisis y ML.

  - La biblioteca fundamental propuesta para este proceso es CTLearn, con el conjunto de dependencias explicitadas anteriormente. Sería potencialmente útil una separación en módulos de esta biblioteca al integrarse en el *Data Lake*, de manera que distintos procesos (lectura, entrenamiento, predicción, *logs*, escritura) se encuentren aislados y con comunicación continua con la base de datos del *Data Lake*, asegurando el registro y guardado de los estados intermedios de los datos, así como facilitar el almacenamiento de registros de entrenamiento y medidas de calidad de los algoritmos usados.

- Interfaz de Usuario. Usando herramientas como Tableau, se pueden crear *dashboards* y herramientas de visualización de datos, facilitando así el acceso, monitoreo y uso de las herramientas expuestas hasta ahora, así como facilitar considerablemente la visualización de datos de entrenamiento y predicciones. Por último, se habrá de proporcionar APIs RESTful, interfaces que permiten comunicación entre sistemas utilizando los principios REST (Representational State Transfer), proporcionando interoperabilidad, escalabilidad (debido a su naturaleza *stateless*) y flexibilidad.

Capítulo 5

Experimentación y Resultados

5.1. Configuración de los modelos

En los siguientes subapartados se desarrollan las estructuras concretas que tendrán los dos modelos de redes neuronales usados para la experimentación. Tanto TRN como *ParticleNet* comprenderán el *backbone* de cada modelo completo, que se obtendrá añadiendo dos capas densas de 512 y 256 neuronas, además de la salida dependiente del tipo de tarea, explicada en la sección 3.6.

5.1.1. TRN

La estructura del modelo TRN usado (sección 3.4) es la presente en los archivos de configuración por defecto. La elección de esta estructura se debe a que ha demostrado un mejor funcionamiento entre aquellas probadas por los desarrolladores de la biblioteca (Nieto, Brill, Feng, Humensky et al., 2019; Miener et al., 2022).

Con los parámetros por defecto, la estructura de la red TRN se compone de 4 bloques residuales o *ResNet*. Sus estructuras se muestran en la tabla 5.1.

# ResNet	Nº Bloques	Filtros	Ratio Self Attention
1	2	48	16
2	3	96	16
3	3	128	16
4	3	256	16

Tabla 5.1: Estructura de los distintos bloques *ResNet*. La primera columna indica el bloque descrito (orden en que se añade a la red TRN) y la segunda columna el número de sub-bloques que componen cada bloque *ResNet*. La tercera y la cuarta columna son los parámetros `filters` y `ratio` descritos en la sección 3.4.

Como es común en los modelos de redes neuronales profundas, el número de filtros (profundidad de los datos) va aumentando a medida que se avanza en la red, permitiendo a esta aprender patrones y características cada vez más complejas.

5.1.2. ParticleNet

Para el caso del modelo *ParticleNet*, puesto que es la primera vez que se aplica en este contexto, se han diseñado estructuras partiendo de elementos del diseño original (Qu & Gouskos, 2020), como el número de bloques o la dimensión de estos. La cabeza del modelo ha sido sustituida al completo por la presente en CTLearn, y se han modificado otras características buscando adaptar la estructura al problema de reconstrucción de eventos de CR. Las distintas configuraciones para la experimentación se muestran en el cuadro 5.2.

Configuración	K	Canales	Número de puntos
1	(16, 16, 16)	(64, 128, 256)	400
2	(32, 16, 8)	(64, 128, 256)	400
3	(16, 16, 16,16)	(64, 128, 256, 512)	600
4	(16, 16, 16)	(64,128,256)	100

Tabla 5.2: Parámetros e hiperparámetros de configuración de *ParticleNet*. La primera columna indica el índice de la configuración. La segunda columna muestra el número de vecinos usado para cada bloque *EdgeConv* y la tercera columna el número de filtros/canales usados en las capas convolucionales de dicho bloque. La última columna indica el número de puntos usados para construir las PC.

En términos de estructura de la red, las configuraciones 1 y 4 son exactamente iguales a la usada en la original, mientras que la Configuración 2

aplica una estrategia de reducción en el número de vecinos para calcular las coordenadas relativas a medida que se avanza en la red. Por otro lado, la Configuración 3 aumenta la profundidad para beneficiarse de la cantidad de datos disponible para entrenar.

En cada caso se adapta el número de puntos a la estructura concreta. Teniendo en cuenta que el número de píxeles de las imágenes de entrenamiento para el modelo TRN es del orden de 10^4, la reducción de uno o dos órdenes elimina una cantidad considerable de información de la imagen. La elección del número de píxeles se basa en la gráfica superior de la figura 4.4; puesto que la mayor parte de las celdas detectan entre 0 y 10 fotoelectrones, y la selección de píxeles se hace en orden descendente de intensidad, se almacena un número de valores del orden de las centenas para transformar a PC. Por ello, se establece $N = 400$ para las configuraciones más parecidas a la original (1 y 2), aumentando hasta $N = 600$ para la configuración con más capas (Configuración 3) y reduciendo hasta $N = 100$ para estudiar un límite inferior de información útil.

5.2. Preprocesamiento

Con el objetivo de mostrar el resultado de realizar el preprocesamiento a PC, se han transformado dos de las imágenes de entrada al modelo *ParticleNet*, usando posteriormente la información presente en dicho formato para reconstruirlas, para lo que se aplica el método `reconstruct_image` de la clase presente en el bloque de código 4.4.

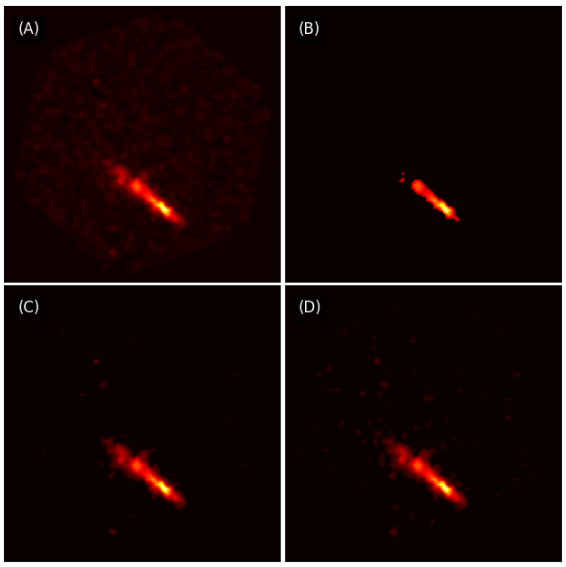


Figura 5.1: **(A)** Imagen original (preprocesamiento para modelo TRN). **(B)** Imagen reconstruida a partir de la nube de puntos con $N = 100$. **(C)** Imagen reconstruida a partir de la nube de puntos con $N = 400$. **(D)** Imagen reconstruida a partir de la nube de puntos con $N = 600$.

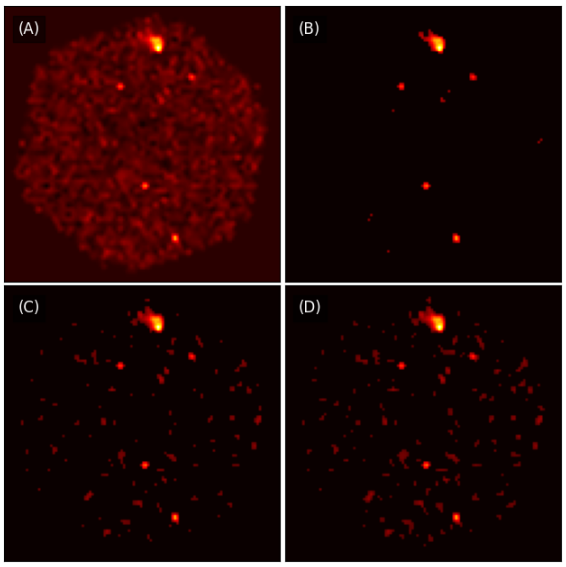


Figura 5.2: **(A)** Imagen original (preprocesamiento para modelo TRN). **(B)** Imagen reconstruida a partir de la nube de puntos con $N = 100$. **(C)** Imagen reconstruida a partir de la nube de puntos con $N = 400$. **(D)** Imagen reconstruida a partir de la nube de puntos con $N = 600$.

Como se puede observar en las figuras 5.1 y 5.2, el proceso de reconstrucción de las imágenes permite aislar las zonas del evento con mayor intensidad, con la consecuencia de perder información sobre el resto de píxeles. Esto implica que la cantidad de información que el modelo recibe como entrada es menor, y aumenta a medida que lo hace N. En la figura 5.1 se aprecia cómo este proceso permite aislar la isla de la imagen original con solo unos pocos píxeles activados en la imagen reconstruida, mientras que en la figura 5.2 la dispersión de la carga en la imagen original es mayor, lo que se traduce en varias zonas intensas en las reconstrucciones, con una predominante en la parte superior.

5.3. Entrenamiento

A continuación se exponen los resultados obtenidos en el entrenamiento de los modelos *ParticleNet* y *TRN* para cada una de las tareas de reconstrucción de eventos, contemplando tanto las métricas obtenidas como los tiempos requeridos para el entrenamiento.

Los resultados se muestran de forma estructurada en 3 subapartados, cada uno de ellos dedicado a una de las tareas para la reconstrucción de los eventos de CR. En cada uno de estos apartados se muestran las curvas de entrenamiento para las 4 configuraciones del modelo *ParticleNet* y la configuración del modelo TRN, así como una tabla con los resultados obtenidos en el entrenamiento de cada uno de los modelos, donde además se muestran los tiempos de entrenamiento.

5.3.1. ParticleType

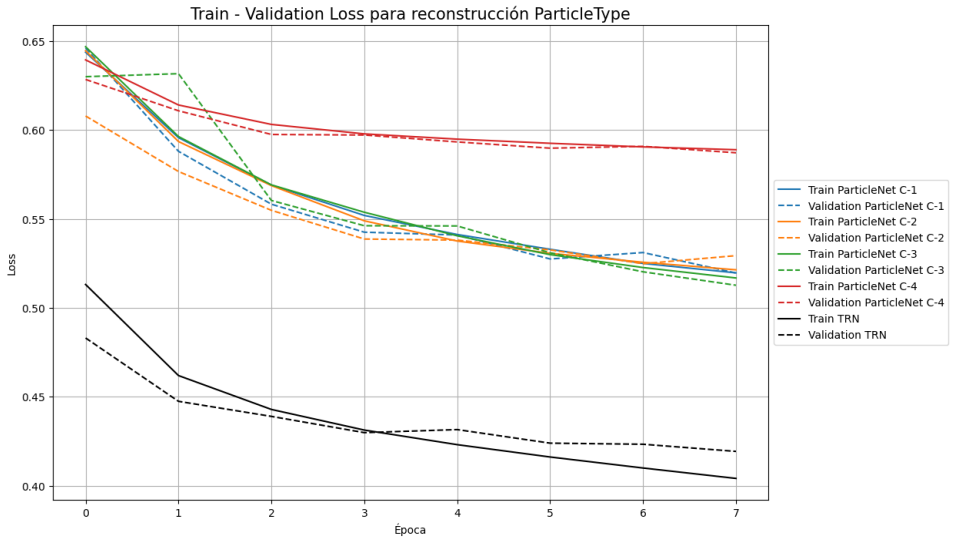


Figura 5.3: Evolución de la función de pérdida durante el entrenamiento de los distintos modelos para la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

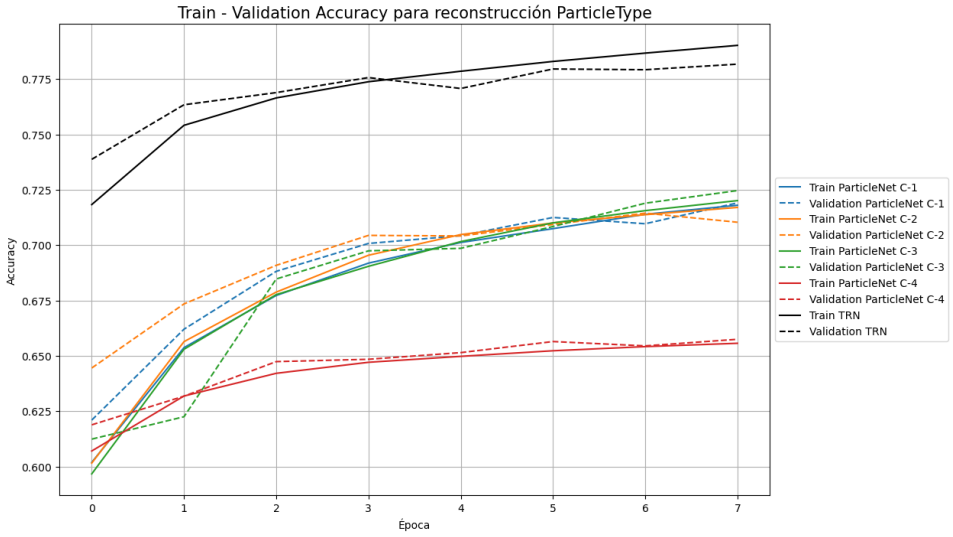


Figura 5.4: Evolución de la precisión durante el entrenamiento de los distintos modelos para la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

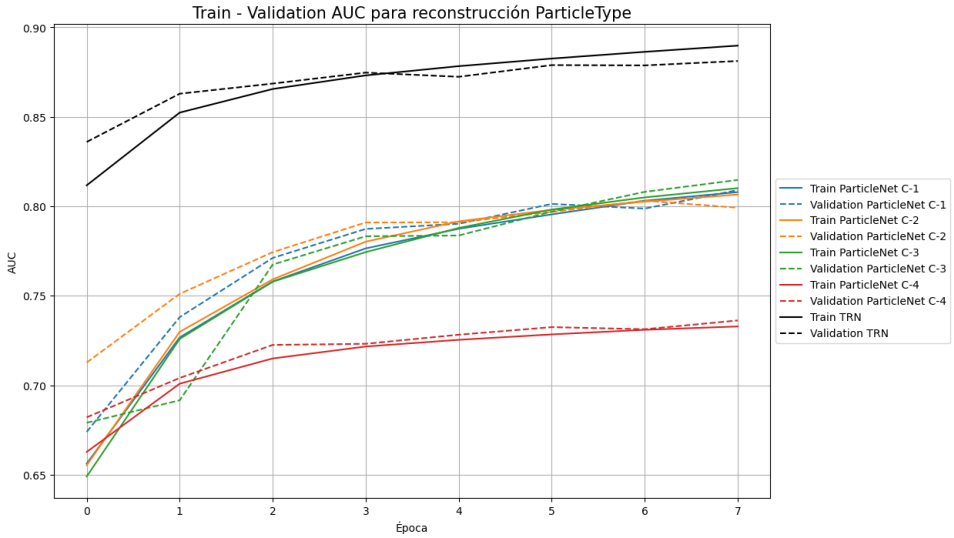


Figura 5.5: Area bajo la curva ROC (AUC) obtenida en el entrenamiento de los distintos modelos para la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.6019	0.6015	0.5967	0.6070	0.7183
2	0.6537	0.6564	0.6529	0.6318	0.7541
3	0.6772	0.6788	0.6776	0.6421	0.7665
4	0.6919	0.6954	0.6904	0.6471	0.7738
5	0.7012	0.7048	0.7016	0.6498	0.7785
6	0.7074	0.7100	0.7101	0.6523	0.7830
7	0.7139	0.7137	0.7156	0.6542	0.7867
8	0.7180	0.7170	0.7201	0.6557	0.7902

Tabla 5.3: Valores de la métrica Accuracy para los datos de entrenamiento del conjunto de los modelos en la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.6210	0.6445	0.6124	0.6188	0.7387
2	0.6620	0.6734	0.6225	0.6318	0.7634
3	0.6882	0.6909	0.6847	0.6474	0.7689
4	0.7007	0.7044	0.6974	0.6484	0.7757
5	0.7043	0.7041	0.6986	0.6514	0.7707
6	0.7125	0.7093	0.7084	0.6564	0.7795
7	0.7097	0.7144	0.7189	0.6545	0.7792
8	0.7191	0.7103	0.7246	0.6575	0.7817

Tabla 5.4: Valores de la métrica Accuracy para los datos de validación del conjunto de los modelos en la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.6563	0.6553	0.6490	0.6628	0.8117
2	0.7268	0.7299	0.7259	0.7009	0.8524
3	0.7580	0.7592	0.7579	0.7150	0.8656
4	0.7765	0.7803	0.7744	0.7217	0.8732
5	0.7875	0.7916	0.7879	0.7254	0.8783
6	0.7955	0.7981	0.7982	0.7284	0.8826
7	0.8031	0.8026	0.8050	0.7310	0.8863
8	0.8079	0.8066	0.8102	0.7329	0.8898

Tabla 5.5: Valores de la métrica AUC para los datos de entrenamiento del conjunto de los modelos en la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.6740	0.7127	0.6791	0.6821	0.8360
2	0.7381	0.7511	0.6916	0.7041	0.8629
3	0.7712	0.7744	0.7675	0.7226	0.8686
4	0.7874	0.7910	0.7833	0.7232	0.8747
5	0.7903	0.7911	0.7837	0.7283	0.8724
6	0.8014	0.7969	0.7970	0.7325	0.8789
7	0.7988	0.8032	0.8081	0.7313	0.8787
8	0.8091	0.7992	0.8147	0.7362	0.8812

Tabla 5.6: Valores de la métrica AUC para los datos de validación del conjunto de los modelos en la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.6438	0.6446	0.6469	0.6395	0.5131
2	0.5957	0.5935	0.5964	0.6141	0.4619
3	0.5690	0.5688	0.5692	0.6032	0.4428
4	0.5520	0.5489	0.5538	0.5978	0.4312
5	0.5412	0.5376	0.5405	0.5949	0.4230
6	0.5329	0.5307	0.5299	0.5926	0.4161
7	0.5250	0.5256	0.5227	0.5905	0.4099
8	0.5197	0.5214	0.5168	0.5889	0.4040

Tabla 5.7: Valores de la función de pérdida para los datos de entrenamiento del conjunto de los modelos en la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.6461	0.6079	0.6300	0.6285	0.4831
2	0.5880	0.5767	0.6317	0.6108	0.4474
3	0.5583	0.5548	0.5604	0.5975	0.4389
4	0.5425	0.5387	0.5462	0.5972	0.4298
5	0.5410	0.5381	0.5461	0.5933	0.4315
6	0.5275	0.5326	0.5310	0.5898	0.4239
7	0.5311	0.5248	0.5203	0.5908	0.4233
8	0.5196	0.5294	0.5127	0.5872	0.4192

Tabla 5.8: Valores de la función de pérdida para los datos de validación del conjunto de los modelos en la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
2	1.110	1.105	3.229	0.968	2.340
3	2.216	2.206	6.462	1.917	4.681
4	3.323	3.306	9.694	2.872	7.022
5	4.429	4.410	12.927	3.829	9.359
6	5.535	5.509	16.159	4.782	11.700
7	6.638	6.614	19.390	5.715	14.042
8	7.742	7.715	22.621	6.676	16.382

Tabla 5.9: Tiempo relativo (en horas) a la primera época obtenido para el entrenamiento de cada modelo en la tarea de clasificación del tipo de partículas que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

5.3.2. Energy

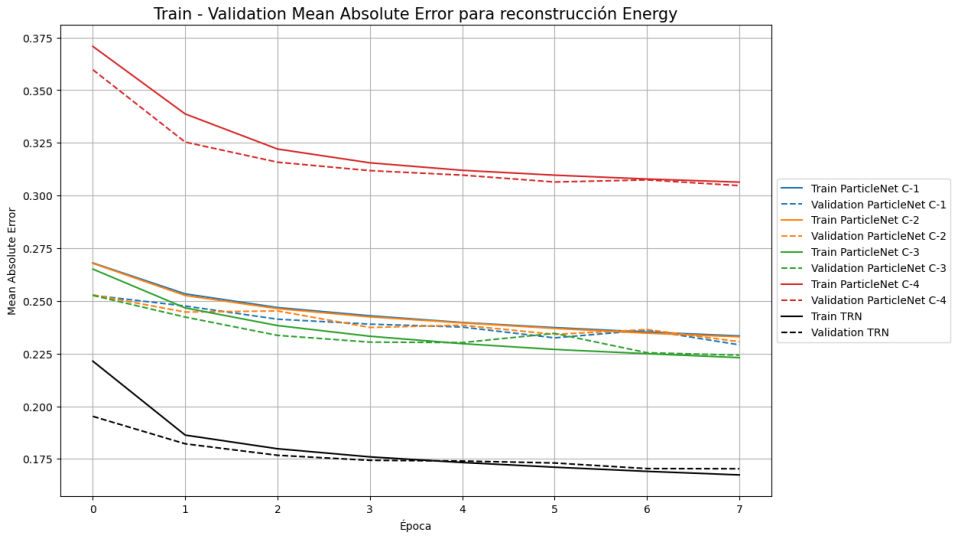


Figura 5.6: Evolución de la función de pérdida (*Mean Absolute Error*, MAE) durante el entrenamiento de los distintos modelos para la tarea de regresión de energía. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.2680	0.2678	0.2651	0.3708	0.2214
2	0.2533	0.2526	0.2467	0.3387	0.1863
3	0.2468	0.2463	0.2383	0.3221	0.1798
4	0.2429	0.2424	0.2332	0.3155	0.1760
5	0.2397	0.2396	0.2297	0.3120	0.1733
6	0.2373	0.2369	0.2270	0.3097	0.1711
7	0.2352	0.2347	0.2249	0.3079	0.1691
8	0.2334	0.2329	0.2231	0.3064	0.1674

Tabla 5.10: Valores de la métrica MAE (*Mean Absolute Error*) para los datos de entrenamiento del conjunto de los modelos en la tarea de regresión de la energía de la partícula que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.2526	0.2529	0.2526	0.3597	0.1952
2	0.2476	0.2447	0.2423	0.3254	0.1822
3	0.2413	0.2452	0.2336	0.3158	0.1767
4	0.2390	0.2374	0.2304	0.3118	0.1743
5	0.2376	0.2384	0.2302	0.3097	0.1740
6	0.2325	0.2341	0.2346	0.3064	0.1731
7	0.2362	0.2364	0.2254	0.3074	0.1704
8	0.2292	0.2306	0.2243	0.3047	0.1703

Tabla 5.11: Valores de la métrica MAE (*Mean Absolute Error*) para los datos de validación del conjunto de los modelos en la tarea de regresión de la energía de la partícula que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
2	0.528	0.536	1.587	0.501	1.163
3	1.056	1.068	3.174	1.002	2.326
4	1.575	1.601	4.760	1.512	3.488
5	2.103	2.129	6.346	2.014	4.652
6	2.632	2.658	7.931	2.513	5.824
7	3.150	3.186	9.517	3.013	6.988
8	3.677	3.715	11.104	3.521	8.152

Tabla 5.12: Tiempo relativo (en horas) a la primera época obtenido para el entrenamiento de cada modelo en la tarea de regresión de la energía de la partícula que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

5.3.3. Direction

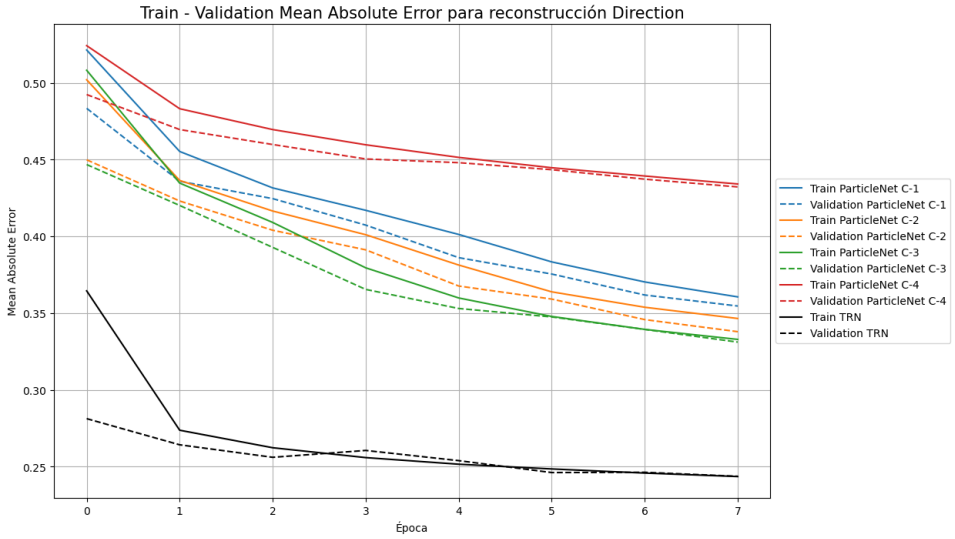


Figura 5.7: Evolución de la función de pérdida (*Mean Absolute Error*, MAE) durante el entrenamiento de los distintos modelos para la tarea de regresión múltiple de dirección. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.5214	0.5021	0.5083	0.5243	0.3646
2	0.4553	0.4365	0.4348	0.4832	0.2738
3	0.4316	0.4166	0.4091	0.4696	0.2624
4	0.4170	0.4011	0.3796	0.4597	0.2560
5	0.4013	0.3814	0.3599	0.4515	0.2516
6	0.3834	0.3639	0.3479	0.4448	0.2486
7	0.3704	0.3539	0.3395	0.4394	0.2459
8	0.3606	0.3466	0.3329	0.4342	0.2436

Tabla 5.13: Valores de la métrica MAE (*Mean Absolute Error*) para los datos de entrenamiento del conjunto de los modelos en la tarea de regresión múltiple de la dirección de la partícula que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
1	0.4834	0.4499	0.4468	0.4924	0.2813
2	0.4360	0.4231	0.4202	0.4696	0.2643
3	0.4246	0.4041	0.3929	0.4599	0.2561
4	0.4074	0.3913	0.3655	0.4505	0.2606
5	0.3862	0.3677	0.3531	0.4481	0.2540
6	0.3755	0.3592	0.3476	0.4435	0.2462
7	0.3619	0.3458	0.3394	0.4373	0.2464
8	0.3547	0.3380	0.3312	0.4323	0.2438

Tabla 5.14: Valores de la métrica MAE (*Mean Absolute Error*) para los datos de validación del conjunto de los modelos en la tarea de regresión múltiple de la dirección de la partícula que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

Época	P.Net C-1	P.Net C-2	P.Net C-3	P.Net C-4	TRN
2	1.549	1.572	1.792	1.526	1.657
3	3.090	3.136	3.584	3.034	3.302
4	4.631	4.699	5.376	4.558	4.949
5	6.184	6.266	7.173	6.071	6.602
6	7.734	7.835	8.962	7.576	8.246
7	9.288	9.405	10.751	9.085	9.887
8	10.840	10.978	12.539	10.595	11.547

Tabla 5.15: Tiempo relativo (en horas) a la primera época obtenido para el entrenamiento de cada modelo en la tarea de regresión múltiple de la dirección de la partícula que inicia la cascada. Para la configuración de cada modelo *ParticleNet* referirse a la tabla 5.2.

5.3.4. Análisis del entrenamiento

Modelo	Accuracy	Val Accuracy	AUC	Val AUC
P.Net C-1	0.7180	0.7191	0.8079	0.8091
P.Net C-2	0.7170	0.7103	0.8066	0.7992
P.Net C-3	0.7201	0.7246	0.8102	0.8147
P.Net C-4	0.6557	0.6575	0.7329	0.7362
TRN	**0.7902**	**0.7817**	**0.8898**	**0.8812**

Tabla 5.16: Resumen de los valores finales de las métricas y tiempo de entrenamiento para el conjunto de los modelos en la tarea de clasificación del tipo de partícula que inicia la cascada. Se destacan en negrita los mejores valores obtenidos.

Modelo	Energy		Direction	
	MAE	Val MAE	MAE	Val MAE
P.Net C-1	0.2334	0.2292	0.3606	0.3547
P.Net C-2	0.2329	0.2306	0.3466	0.3380
P.Net C-3	0.2231	0.2243	0.3329	0.3312
P.Net C-4	0.3064	0.3047	0.4342	0.4323
TRN	**0.1674**	**0.1703**	**0.2436**	**0.2438**

Tabla 5.17: Resumen de los valores finales de las métricas y tiempo de entrenamiento para el conjunto de los modelos en las tareas de regresión de la energía y dirección de la partícula que inicia la cascada. Se destacan en negrita los mejores valores obtenidos.

Modelos	Type	Energy	Direction
P.Net C-1 - TRN	52.68 %	54.78 %	6.30 %
P.Net C-2 - TRN	52.88 %	54.22 %	5.01 %
P.Net C-3 - TRN	-38.07 %	-36.34 %	-8.57 %
P.Net C-4 - TRN	59.08 %	56.82 %	8.06 %

Tabla 5.18: Diferencia media relativa (%) al modelo TRN en tiempos de entrenamiento por época. Un valor negativo indica que el modelo TRN ha sido más rápido.

Como se puede apreciar en las figuras 5.3 a 5.7, el conjunto de los entrenamientos consigue converger en todas las tareas, mostrando por tanto que todos los modelos son capaces de, con mayor o menor dificultad, aprender de los datos proporcionados para resolver las tareas designadas.

Atendiendo a los resultados de las métricas de entrenamiento (figuras 5.3 a 5.7 y tablas 5.16 y 5.17), se aprecia como el modelo por defecto de la biblioteca CTLearn TRN obtiene las mejores métricas tanto en entrenamiento como en validación. Para la tarea de clasificación, obtiene una precisión del 78.17 %, un AUC de 88.12 % y un valor de la función de pérdida de 0.4192 para el conjunto de validación, suponiendo una mejora del 7.3 %, del 7.5 % y del 22.3 % respectivamente comparando con el modelo *ParticleNet* que obtiene mejores resultados (Configuración 3).

Por otro lado, en la regresión de energía, el modelo TRN obtiene un MAE de 0.1703 en validación, lo que supone una mejora del 31.7 % si se compara con el modelo *ParticleNet* que obtiene mejores resultados (Configuración 3).

Finalmente, para la regresión de dirección, el modelo TRN obtiene un MAE de 0.2438 en validación, implicando una mejora del 35.8 % comparando con el modelo *ParticleNet* que obtiene mejores resultados (Configuración 3), siendo este caso en el que más diferencia se aprecia entre el modelo TRN y

el resto de modelos usados.

En otro aspecto, se observa que, en general, los modelos *ParticleNet* tienen tiempos de entrenamientos considerablemente menores al modelo TRN (ver tabla 5.18), destacando especialmente en las tareas de clasificación y regresión de energía, donde los modelos *ParticleNet* son hasta un 59 % y un 56 % más rápidos respectivamente en el modelo más ligero, disminuyendo hasta un 52 % y un 54 % en el segundo mejor modelo (Configuración 1). En la tarea de regresión de dirección, la diferencia es menor, siendo $< 10\,\%$ en el conjunto de los casos.

Es de interés notar también que el modelo *ParticleNet* que mejores resultados consigue (Configuración 3) obtiene unos tiempos de entrenamiento mayores al modelo TRN, sin conseguir mejorar los resultados de este en ninguna de las tareas, como se ha expuesto anteriormente.

5.4. Predicción de test

Para la evaluación de la predicción del conjunto de eventos de test se usará una doble estrategia: para las predicciones de energía y dirección se utilizará la misma métrica (MAE) que se ha usado para la evaluación de los modelos en entrenamiento y validación, mientras que para la predicción del tipo de partícula se visualizarán las distribuciones de la variable de predicción *gammaness* para cada tipo de partícula. Además, el desempeño de los que han resultado los mejores modelos será valorado de forma cualitativa mediante el uso de las IRFs, que permitirán visualizar la eficiencia de detección de los modelos en función de la energía de la partícula que inicia la cascada (sección 5.5). Por otro lado, debido a que la Configuración 4 del modelo *ParticleNet* obtiene los peores resultados en todas de las tareas, no se incluirá en la evaluación de los resultados de test.

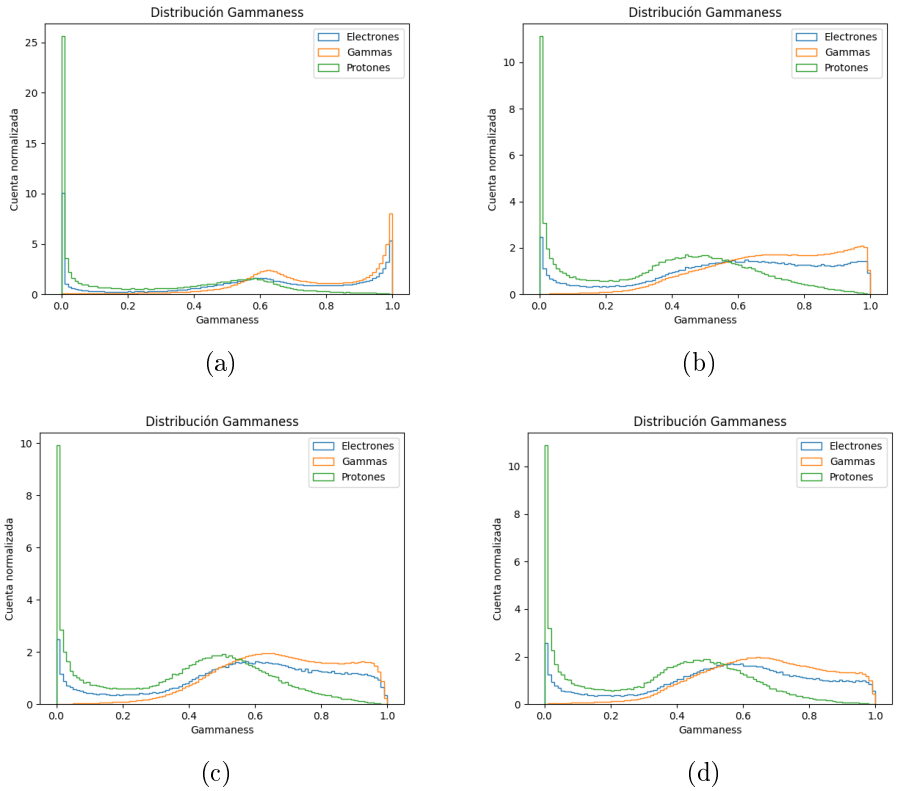


Figura 5.8: Distribución normalizada (normalización de probabilidad) para las predicciones de *gammaness* en los datos test de diferentes modelos. Figura (a) representa el modelo TRN, Figura (b) el modelo *ParticleNet* (Configuración 1), Figura (c) el modelo *ParticleNet* (Configuración 2) y Figura (d) el modelo *ParticleNet* (Configuración 3).

Partícula	Energy				Direction			
	TRN	PN C1	PN C2	PN C3	TRN	PN C1	PN C2	PN C3
Electrones	**0.715**	117.260	10.854	1.201	**1.643**	2.546	2.521	2.489
Gammas	**0.162**	0.309	0.315	0.300	**0.734**	0.835	0.868	0.805
Protones	**3.549**	3.699	3.676	3.784	**3.289**	3.876	3.827	3.850

Tabla 5.19: MAE para las predicciones de energía y dirección para cada tipo de partícula en los diferentes modelos. PN hace referencia a *ParticleNet* y C1, C2 y C3 a las 3 configuraciones. La negrita indica el menor valor de MAE para cada tipo de partícula.

En las figuras 5.8 se observa la distribución de las predicciones de *gammaness* para cada tipo de partícula en los datos de test. En este caso, se puede observar una superioridad en términos de discriminación del modelo

TRN, al acentuarse las diferencias entre las distribuciones de las predicciones de *gammaness* para los protones y los gammas, mientras que en el caso de los modelos *ParticleNet*, incluso en aquel que consigue las mejores métricas, la distribución de los valores de *gammaness* para los gammas es más alargada hacia valores bajos de este y no presenta un pico claro en torno a 1, lo que indica una peor capacidad de discriminación entre los gammas y los protones. En cuanto a la detección de electrones, todos los modelos presentan una distribución que se asemeja a los gammas en los valores altos de *gammaness*, y a los protones en los valores bajos, lo que indica una peor capacidad de discriminación entre los electrones y los otros tipos de partículas.

Los resultados para la reconstrucción de energía y dirección (tabla 5.19), muestran de nuevo la superioridad del modelo TRN, obteniendo un error un 50 % menor al resto de modelos para el caso de la energía. La diferencia, no obstante, se vuelve más pequeña en el caso de la dirección, donde el modelo TRN obtiene un error que es solo un 10 % menor al mejor modelo de *ParticleNet* (0.734 frente a 0.805).

Observando, finalmente, los errores para la energía y dirección de electrones y protones, se aprecia que son considerablemente mayores que los obtenidos para los gammas, lo que no obstante es esperado, ya que los modelos solo se entrenan para esta tarea con los datos de los gammas, y por tanto no se pretende que tengan la misma capacidad de reconstrucción para los otros tipos de partículas.

5.5. Instrument Response Functions

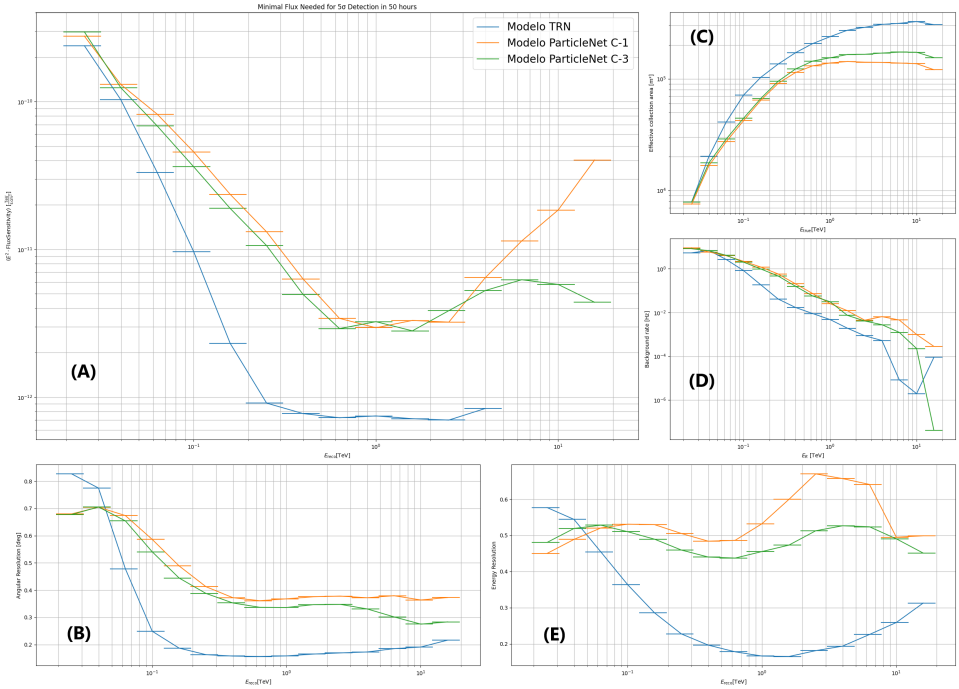


Figura 5.9: Conjunto de las IRFs para los modelos TRN y *ParticleNet* con Configuración 1 (C-1) y Configuración 3 (C-3). **(A)** Flujo mínimo para obtener una detección significativa en 50h en función de la energía reconstruida. **(B)** Resolución angular (grados decimales) en función de la energía reconstruida. **(C)** Área efectiva del sistema en función de la energía real. **(D)** Tasa de eventos *background* por unidad de ángulo sólido en función la energía reconstruida para los mejores modelos obtenidos. **(E)** Resolución energética en función de la energía reconstruida.

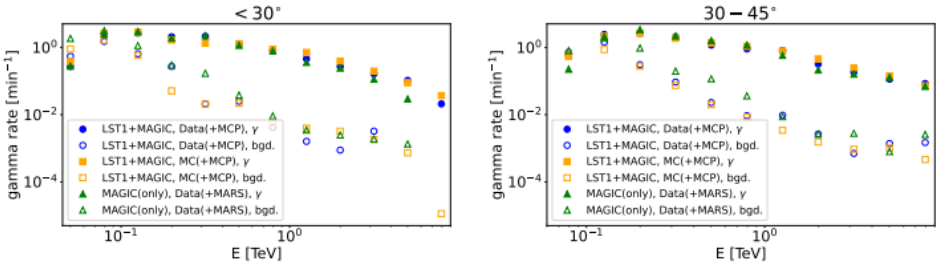


Figura 5.10: Tasas de eventos gammas (marcas rellenas) y fondo (marcas vacías) para distintos valores de sensibilidad y telescopios (Abe, Abe, Abe, Acciari et al., 2023).

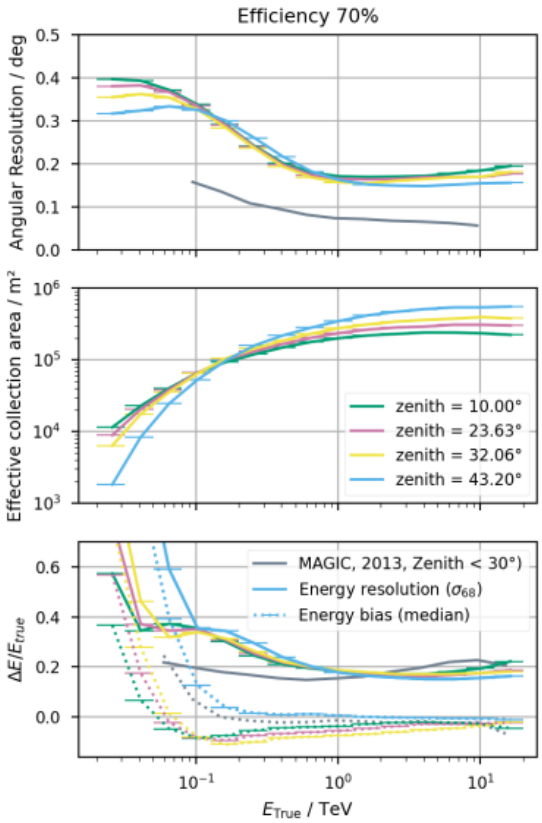


Figura 5.11: Resolución energética, resolución angular y area efectiva para distintos ángulos de recolección del LST de CTAO y una eficiencia en discriminación de *background* del 70 % (Abe, Abe, Abe, Aguasca-Cabot et al., 2023).

En esta sección se comparan las curvas de sensibilidad y respuesta del sistema conformado por las simulaciones y los modelos de DL usados para la reconstrucción de los eventos. Para ello, se han generado las IRFs para los modelos TRN y *ParticleNet* (Configuración 1 y 3) de acuerdo con lo expuesto en la sección 5.5. Se han obtenido los valores de área efectiva, tasa de eventos *background*, resolución angular, resolución energética y flujo mínimo para obtener una detección significativa en función de la energía de la partícula que inicia la cascada. Los resultados se muestran en la figura 5.9 de forma compacta y en el Anexo 2 desglosadas.

Comenzando con el area efectiva (figura 5.9**(C)**), es apreciable una clara superioridad por parte del modelo TRN, que muestra un valor superior en todo el espectro, aumentando la diferencia a medida que se incrementa la energía. No obstante, los modelos *ParticleNet* de Configuración 1 y 3 consiguen mantenerse en el mismo orden en todo el rango. Estos resultados se

encuentran en el mismo orden de resultados que el área efectiva del LST de CTAO, mostrado en la figura 5.11.

En la gráfica de resolución angular (figura 5.9**(B)**), se observa que el modelo TRN presenta una superioridad notable en prácticamente todo el espectro de energía, siendo únicamente superado en energías por debajo de $\sim 0.03 TeV$, donde los modelos *ParticleNet* de Configuración 1 y 3 presentan una resolución angular similar y ligeramente menor a la del modelo TRN. En el resto del espectro, el modelo TRN se muestra como el mejor en términos de resolución angular, aunque la diferencia no es tan marcada como en el caso de la energía. Podemos comparar esta resolución con la presente en la figura 5.11 para el LST de CTAO, donde se observa que en estos experimentos se obtiene una resolución angular similar a la de este telescopio entre los $0.1 TeV$ y los $10 TeV$ con el modelo TRN, mientras que los modelos *ParticleNet* presentan una resolución angular ligeramente peor. Es destacable como el conjunto de los modelos obtienen resoluciones angulares considerablemente peores en rangos de energía por debajo de los $0.1 TeV$.

En la resolución energética (figura 5.9**(E)**) es donde el modelo TRN muestra una mayor superioridad, presentando una resolución mucho menor que los modelos *ParticleNet* en todo el espectro de energía, especialmente en el rango de los 0.1 a los 10TeV, donde la diferencia es más notable. Por otro lado, los dos modelos *ParticleNet* muestran una resolución energética similar, aunque el modelo *ParticleNet* con Configuración 3 alcanza una resolución ligeramente mejor en todo el espectro, a excepción del rango por debajo de los $\sim$ 0.03TeV, donde el modelo *ParticleNet* (Configuración 1) lo supera levemente, y ambos consiguen, de nuevo, superar al modelo TRN. Al compararlo con la resolución energética del LST de CTAO (figura 5.11), el comportamiento expuesto se mantiene, siendo TRN el único modelo que consigue obtener valores a los del LST en todo el rango, mientras que los modelos *ParticleNet* empeoran su resolución a medida que aumenta la energía.

Atendiendo al flujo mínimo para obtener una detección significativa en 50h (figura 5.9**(A)**), se observa que el modelo TRN presenta un flujo mínimo mucho menor que los modelos *ParticleNet* en todo el espectro de energía, siendo especialmente notable en el rango de los 0.1 a los $\sim$ 5TeV, donde la diferencia es de prácticamente un orden. Es además, en ese mismo rango, donde el modelo *ParticleNet* con Configuración 3 muestra de nuevo superioridad sobre el modelo *ParticleNet* con Configuración 1, siendo más marcada que en el caso de la resolución energética. Comentar que los tres últimos puntos del modelo TRN no se calcularon debido a la falta de suficientes eventos simulados que sobreviviesen los cortes de análisis aplicados aquí, motivo por el cual no se muestra dicha información en la gráfica.

Finalmente, en la tasa de eventos *background* por unidad de ángulo sólido (figura 5.9**(D)**) es una gráfica más compleja de estudiar, ya que necesita

de comparaciones con fuentes reales para poder ser interpretada de forma adecuada. No obstante, podemos utilizar los datos publicados en octubre de 2023 por Abe. H y colaboradores (Abe, Abe, Abe, Acciari et al., 2023) (figura 5.10) para observar las tendencias, que son decrecientes para la energía. Aunque todos los modelos cumplen con esta tendencia, el modelo TRN presenta un aumento pasado los 10TeV, lo que puede denotar cierta inferioridad en este caso. Por otro lado, se observa que en general las tasas de detección son un orden de magnitud menor en los modelos usados frente a los datos reales, indicando que detectan más eventos de ruido de los presentados como reales.

5.6. Motivaciones Data Lake

Las motivaciones que llevan a proponer la implementación de un *Data Lake* como solución a largo plazo para la gestión de datos derivados de CTA u otros telescopios de Cherenkov se deriva de la experiencia de usuario actual de la biblioteca. Si bien CTLearn proporciona métodos sencillos para ejecutar el conjunto de las operaciones necesarias para entrenar y evaluar un modelo para la reconstrucción de eventos, presenta también algunas dificultades como las que se explicitan a continuación:

- Gestión de errores. La integración de la biblioteca DLH en CTLearn para la gestión y transformación de datos permite realizar el *pipeline* completo desde los datos en su formato DL0 a DL1, su preprocesamiento, y la posterior escritura de las predicciones (formato DL2). Sin embargo, también dificulta el aislamiento y rastreo de errores, así como la solución de estos y el mantenimiento aislado de las bibliotecas, ya que existen dependencias internas de CTLearn que se encuentran completamente englobadas en las funciones de entrenamiento y predicción. La modularización de estos procesos en el *Data Lake* facilitaría el mantenimiento de estas bibliotecas, de forma que se redujese al máximo la dependencia entre el proceso de preprocesamiento y gestión de datos del entrenamiento y la evaluación de los modelos.

- Gestión de datos. Actualmente, la gestión de datos y almacenamiento se lleva a cabo en el clúster de CTA La Palma, en un entorno *linux*. El volumen de datos que genera este tipo de experimentos, la necesidad de un acceso sencillo a estos para las aplicaciones de técnicas de ML, la visualización de resultados y el estudio de los resultados son algunos elementos que apoyan la necesidad de una estructura de datos más desarrollada y adaptada a un público con distintos niveles de conocimiento informático. Una propuesta para el formato de datos consistiría en integrar herramientas de visualización y edición, como

ViTables (Mas, 2023) (asegurando siempre la integridad de los datos), más accesibles, o la aplicación de un algoritmo recursivo para su exploración. Estas soluciones permitirían simplificar la visualización de los datos y acercarlos a un público investigador general sin la necesidad de poseer conocimientos específicos de informática y/o programación.

- Concretamente en el ámbito de la investigación, especialmente al utilizar algoritmos de ML, un *Data Lake* genera una considerable ventaja al combinarse estas herramientas con plataformas de bajo o nulo código. Esto permitirá a los investigadores crear y ajustar intuitivamente algoritmos de ML sin la necesidad de poseer conocimientos de programación. Mediante una interfaz fácil de usar, los científicos pueden organizar información, crear procesos de análisis y utilizar algoritmos en el *Data Lake* sin tener que programar. Dichos aspectos impulsan la accesibilidad a herramientas avanzadas de análisis de datos y ML, agilizando el descubrimiento y mejora de modelos, y permitiendo a los científicos concentrar los esfuerzos en la interpretación de resultados y progreso del conocimiento en vez de en la programación técnica.

En conclusión, la generación de un *Data Lake* para el almacenamiento y gestión de datos de LST-1 de CTAO (Large-Sized-Telescope of the Cherenkov Telescope Array Observatory) y la aplicación de herramientas de análisis de datos y ML, supondría un avance considerable en la aplicación de nuevas técnicas informáticas al campo científico, específicamente a la astrofísica, al crear un sistema que simplifica la interpretación y realización del *pipeline* que va desde el registro y almacenamiento de los datos brutos, hasta la obtención de los datos necesarios para realizar las investigaciones y extraer las conclusiones científicas pertinentes.

5.7. Discusión

Como se ha expuesto en la secciones 5.3 a 5.5, la biblioteca CTLearn ha demostrado ser una herramienta útil para la aplicación de técnicas de ML y DL en la reconstrucción de eventos de CR basado en las medidas obtenidas por IACT, y en particular en las simulaciones que permiten la calibración de estos telescopios, los modelos y las herramientas de análisis que los subyacen.

Si bien el modelo TRN se corresponde actualmente con el estado del arte en la reconstrucción de eventos de CR a partir de imágenes monoscópicas, los modelos *ParticleNet* presentan una alternativa potencialmente útil para esta misma tarea.

El motivo principal de esta afirmación reside en los tiempos de entrenamiento y en la reducción de la complejidad de los modelos y los datos

utilizados. Como se ha podido comprobar en la sección 5.3, especialmente en las tareas de reconstrucción de tipo y energía, los modelos *ParticleNet* (a excepción de la Configuración 3) presentan tiempos de entrenamiento considerablemente menores al modelo TRN, lo que supone una ventaja en términos de optimización de los modelos y en la capacidad de realizar ajustes y pruebas en un tiempo menor. Los tiempos actuales de entrenamiento del modelo TRN requerirían de, aproximadamente, $16 + 8 + 11 = 45h$ para el entrenamiento completo de los 3 modelos (reconstrucción de tipo de partícula, dirección y energía) frente a las $4 + 10 + 7 = 21h$ en el caso de usar *ParticleNet* (Configuración 1). Esto supone, como se ha explicitado anteriormente, un 50 % de reducción, lo que facilita aplicar técnicas de búsqueda de hiperparámetros al tratarse de tiempos más manejables.

Sin embargo, ambos modelos requieren de periodos relativamente largos de entrenamiento, lo que ha implicado no poder realizar tareas de optimización de hiperparámetros en el periodo de experimentación, debido a las limitaciones físicas por la disponibilidad de entornos de entrenamiento, pero también a los tiempos que esto requería, teniendo en cuenta que el entrenamiento de un modelo con una configuración concreta de hiperparámetros puede necesitar de varios días, y estas estructuras presentan una alta capacidad de personalización y por tanto una alta variabilidad en sus ajustes, sin contar otros elementos como la necesidad de realizar *cross validation*, que multiplica los tiempos requeridos.

Continuando con la comparación, el formato de los datos usado para los modelos *ParticleNet* permitiría manejar directamente la geometría hexagonal de los telescopios, eliminando así la necesidad de transformar los datos e introducir píxeles artificiales que pueden afectar negativamente a las tareas al aportar ruido. Un claro ejemplo de este riesgo se ve en las figuras 4.1 a 4.3, en el canal de tiempo de pico, donde se usa el mismo valor (0) para el momento de mayor intensidad y para los píxeles de relleno. De igual forma, se hace posible el uso de coordenadas "globales" a un *array* de telescopios, usando distancias entre ellos, adaptando estos modelos a la tarea de reconstrucción de eventos estereoscópicos y aportando información física (distancia real entre los dispositivos) al modelo.

Todos estos elementos se complementan en el hecho de que el conjunto de las configuraciones usadas para el modelo *ParticleNet* se mantienen en el orden de los resultados obtenidos por el modelo TRN, tanto en las métricas de entrenamiento y test, como en las IRFs, lo que indica que son capaces de aprender de los datos y de reconstruir los eventos de forma adecuada. No obstante, es importante tener en cuenta que los modelos *ParticleNet* presentan una mayor variabilidad en los resultados, especialmente si atendemos al mejor modelo (Configuración 3) y al peor (Configuración 4), lo que indica que la elección de la arquitectura y de los hiperparámetros es crucial para

obtener un buen rendimiento.

En este mismo sentido, se aprecia claramente cómo aumentar la profundidad de la red y el número de puntos tiene un aspecto negativo en el tiempo de entrenamiento, siendo superior al modelo TRN en todas las tareas. Es por este motivo que antes de continuar avanzando en desarrollar modelos con esta técnica, es necesario realizar un estudio más profundo de la arquitectura y los hiperparámetros, valorando aspectos como: número de puntos a seleccionar, técnica para selección de los puntos, características a reflejar en la estructura, sistema de coordenadas, valores de K, número de capas, uso de técnicas más avanzadas de conexiones residuales (como las mencionadas en la sección 3.4), etc. Además, existe la posibilidad de explorar estructuras más complejas y novedosas, como es el caso de las capas de atención de los modelos Transformers (Guo et al., 2021), que han demostrado ser útiles en la comprensión de sistemas físicos complejos y en la reconstrucción de eventos de física de partículas (Mikuni & Canelli, 2021; Xuan et al., 2022; Jiang et al., 2024).

Si bien los modelos *ParticleNet* son potencialmente útiles por los argumentos expresados anteriormente, los resultados expuestos demuestran que el modelo TRN presente en la biblioteca CTLearn es superior en todas las tareas de reconstrucción de eventos, proporcionando mejoras reflejadas en las IRFs, pero también en el entrenamiento y en la predicción de datos test. Se destaca especialmente la distribución de la variable *gammaness* en la figura 5.8a, que demuestra una mayor capacidad de discriminación entre los gammas y los protones. Estos elementos hacen que el sacrificio en términos de tiempos de entrenamiento siga siendo actualmente beneficioso y posicione al modelo TRN a la cabeza de las técnicas de reconstrucción de eventos de CR basadas en imágenes monoscópicas.

Finalmente, el desarrollo de este trabajo ha permitido identificar los puntos fuertes y débiles de la biblioteca CTLearn. Entre sus principales ventajas encontramos la fácil parametrización y registro de los modelos y los ajustes utilizados, así como la facilidad general de uso al no requerir más código que una serie de comandos en terminal.

No obstante, durante este proceso también se han identificado una serie de problemas y limitaciones, debido principalmente al estado en desarrollo de la biblioteca, que han dificultado y aumentado los tiempos de trabajo (ver Anexo 1).

El primero de estos elementos es la falta de una documentación completa y actualizada sobre el conjunto de los parámetros, opciones y funciones de la biblioteca, ya que no se explicita, entre otros elementos, la necesidad de generar las predicciones en el mismo archivo para poder producir las IRFs, la necesidad de establecer el parámetro `output` para cargar un modelo ya entrenado, el papel de determinados archivos de generación automática

o cuales son los parámetros necesarios y cuáles opcionales para realizar un entrenamiento o una predicción. Este problema se ve agravado por la falta de ejemplos y tutoriales actualizados, que han implicado un contacto constante con los desarrolladores para resolver dudas y problemas.

Por otro lado, la falta de información en la misma documentación de la biblioteca sobre el formato concreto de los datos a utilizar, o la ausencia de una herramienta integrada de visualización de datos alarga la tarea de preprocesamiento y análisis, especialmente para usuarios alejados de la informática y la programación, o que no estén familiarizados con el formato de los datos de LST-1 de CTAO, y las bibliotecas que complementan a CTLearn.

Finalmente, los elementos que más dificultades han producido han sido la instalación del entorno de la biblioteca en el clúster de ejecución, que ha requerido de la intervención de los desarrolladores para solucionar problemas de dependencias, especialmente para el uso de los recursos de GPU (obligatorios para el entrenamiento), y la falta de un buen sistema de seguimiento de errores que, aún con conocimientos de la biblioteca, no se pudieron resolver en un tiempo extenso. Estos dos elementos juntos han supuesto un retraso de aproximadamente 2 meses para completar el entrenamiento de los modelos TRN, y un problema en prácticamente cada paso posterior (predicción y generación de IRFs).

En resumen, la biblioteca CTLearn es una herramienta útil y potente para la aplicación en la reconstrucción de eventos de CR usando las medidas obtenidas por IACT, pero requiere de una serie de mejoras y actualizaciones para facilitar su uso y ampliarla al público investigador en general, ya que actualmente los tiempos de aprendizaje de uso se encuentran en torno a las 2-3 semanas, al igual que los tiempos de resolución de problemas, considerando unos conocimientos informáticos y de programación medios.

5.8. Cumplimiento de Objetivos

A continuación se expone el grado de cumplimiento de los objetivos y su relación con los resultados obtenidos en este trabajo.

- **OB1.- Estudiar un posible diseño de Data Lake para el almacenamiento y la consulta de datos de astropartículas del LST-1 de CTAO (90/100).** El estudio se ha completado correctamente, valorando las principales ventajas y desventajas que supondría el uso de esta estructura. El uso intenso de la biblioteca CTLearn y su entorno ha permitido detectar los principales problemas y limitaciones de esta, y proponer una solución a largo plazo para la gestión de los datos de CTAO. Algunos elementos para el trabajo futuro son la elección de servicios concretos y la realización de un presupuesto.

- **OB2.- Analizar de forma comparativa modelos de clasificación, usando datos de simulación y datos reales de LST-1 de CTAO (70/100).** Se ha realizado un estudio comparativo del modelo estrella de CTLearn para la reconstrucción de eventos simulados a partir de imágenes monoscópicas con los modelos *ParticleNet* de diferentes configuraciones. Se han comparado los resultados obtenidos en las tareas de reconstrucción de tipo de partícula, energía y dirección, y se han analizado las IRFs obtenidas para los diferentes modelos. Por otro lado, no se han podido realizar comparaciones con datos reales, ya que estos no se encontraban disponibles. Este hecho ha sido compensado con el uso de la versión v0.9.0 de CTLearn, que introduce mejoras en los modelos al considerar elementos realistas como una orientación arbitraria de los telescopios.

- **Implementar metodología MLOPS para el ajuste de hiperparámetros y la evaluación (40/100).** Debido a los tiempos que han sido requeridos para el aprendizaje y uso completo de la biblioteca, los problemas de instalación y ejecución, y los tiempos de entrenamiento de los modelos, no se ha podido implementar una metodología MLOPS. La realización parcial de este objetivo ha consistido en la parametrización de un nuevo modelo que permite la reducción en tiempos de entrenamiento, y el cumplimiento del objetivo **OB1**, ya que supone la base estructural para el desarrollo de la metodología MLOPS. Además, la elección de las estructuras del modelo *ParticleNet* se ha realizado siguiendo un criterio racional para valorar distintas estrategias.

Capítulo 6

Conclusiones

La aplicación de técnicas de ML y DL a problemas relacionados con la física de partículas y la astrofísica son cada vez más comunes debido al gran volumen de datos a analizar y la necesidad de automatizar determinados procesos que permitan reducir al mínimo el trabajo humano en términos de discriminación de eventos, pero también para obtener medidas físicas que permitan contrastar las teorías actuales, aún incompletas o con aspectos en continua revisión.

Este trabajo analiza en detalle una de las bibliotecas más avanzadas para Python en la tarea de reconstrucción de eventos de cascada de partículas registrados a través de IACT, conteniendo el procesado de datos completo necesario para, a partir de datos simulados, obtener predicciones que permitan evaluar el conjunto del sistema: preprocesamiento, entrenamiento, evaluación, y análisis de resultados.

Por otro lado, se ha implementado un modelo (*ParticleNet*) basado en la estructura *EdgeConv*, que permite realizar operaciones convolucionales en PC siendo un método extendido en otros problemas de visión por computador y novedoso en la resolución de problemas de la física de partículas. Esta red presenta un potencial considerable por el formato de datos que habilita, permitiendo eliminar el ruido introducido por la adaptación de las imágenes a las capas convolucionales usuales, pero también por la reducción de la dimensionalidad de los datos de entrenamiento que. Además, es posible integrar capas avanzadas como las capas de atención de los modelos transformers.

Aunque los resultados presentados en este trabajo sitúan al modelo TRN de la biblioteca CTLearn como la opción más avanzada y con mejores métricas y resultados hasta el momento, se considera necesario continuar avanzando tanto en el desarrollo de este tipo de estructura, que aún no ha sido explotada en su totalidad, como en la investigación de los mejores hiper-

parámetros y estructuras de redes neuronales como *ParticleNet*, ya que sus características la hacen especialmente adecuada para la resolución de problemas de física de partículas. Igualmente, se deberán contrastar los resultados actuales con los obtenidos a partir de datos reales, para comprobar que el rendimiento se mantienen en un entorno real, y que este modelo es capaz de realizar la reconstrucción de manera apropiada.

A pesar de que CTLearn se encuentra en un estado avanzado de desarrollo, la experiencia de usuario y la documentación de la biblioteca son aspectos que deben ser pulidos y mejorados para facilitar su uso y su adopción por parte de la comunidad científica. En este sentido, la integración de la biblioteca en una estructura de tipo *Data Lake* supondría una ventaja considerable, ya que permitiría, entre otras cosas crear un sistema completo para el análisis, gestión y visualización de datos, y también para el desarrollo, evaluación y despliegue de modelos de DL.

En conclusión, se considera que el trabajo realizado por los desarrolladores de la biblioteca y el equipo de colaboradores en general supone un aporte considerable a la comunidad científica, y que la implementación de nuevos modelos de DL en la resolución de problemas de física de partículas y astrofísica es un campo en constante evolución y con un gran potencial para el desarrollo de nuevas técnicas y métodos que permitan avanzar en la comprensión de los fenómenos físicos que rigen el universo.

Bibliografía

De Angelis, A., & Pimenta, M. (2018). *Introduction to particle and astroparticle physics: multimessenger astronomy and its particle physics foundations*. Springer.

Spurio, M., Spurio & Bellantone. (2018). *Probes of Multimessenger Astrophysics*. Springer.

Abbott, B. P., Abbott, R., Abbott, T. D., Acernese, F., & Ackley. (2017). GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral. *Phys. Rev. Lett.*, *119*, 161101. https://doi.org/10.1103/PhysRevLett.119.161101

Jacquemont, M., Vuillaume, T., Benoit, A., Maurin, G., Lambert, P., & Lamanna, G. (2021). First full-event reconstruction from imaging atmospheric cherenkov telescope real data with deep learning. *2021 International Conference on Content-Based Multimedia Indexing (CBMI)*, 1-6.

Hillas, A. M. (1985). Cerenkov light images of EAS produced by primary gamma. *19th Intern. Cosmic Ray Conf-Vol. 3*, (OG-9.5-3).

Fiasson, A., Dubois, F., Lamanna, G., Masbou, J., & Rosier-Lees, S. (2010). Optimization of multivariate analysis for IACT stereoscopic systems. *Astroparticle Physics*, *34* (1), 25-32.

Albert, J., Aliu, E., Anderhub, H., Antoranz, P., Armada, A., Asensio, M., Baixeras, C., Barrio, J., Bartko, H., Bastieri, D., et al. (2008). Implementation of the random forest method for the imaging atmospheric Cherenkov telescope MAGIC. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, *588* (3), 424-432.

Krause, M., Pueschel, E., & Maier, G. (2017). Improved γ/hadron separation for the detection of faint γ-ray sources using boosted decision trees. *Astroparticle Physics*, *89*, 1-9.

Nieto, D., Brill, A., Kim, B., & Humensky, T. (2017). Exploring deep learning as an event classification method for the Cherenkov Telescope Array. *arXiv preprint arXiv:1709.05889*.

Mangano, S., Delgado, C., Bernardos, M. I., Lallena, M., Rodríguez Vázquez, J. J., & Consortium, C. (2018). Extracting gamma-ray information

from images with convolutional neural network methods on simulated cherenkov telescope array data. *Artificial Neural Networks in Pattern Recognition: 8th IAPR TC3 Workshop, ANNPR 2018, Siena, Italy, September 19–21, 2018, Proceedings 8*, 243-254.

Shilon, I., Kraus, M., Büchele, M., Egberts, K., Fischer, T., Holch, T. L., Lohse, T., Schwanke, U., Steppa, C., & Funk, S. (2019). Application of deep learning methods to analysis of imaging atmospheric Cherenkov telescopes data. *Astroparticle Physics*, *105*, 44-53.

Image Classification | Papers With Code [Accessed: 2024-09-02]. (2024).

Brill, A., Kim, B., Nieto, D., Miener, T., & Feng, Q. (2018). CTLearn: Deep learning for imaging atmospheric Cherenkov telescopes event reconstruction (0.6.0). https://doi.org/10.5281/zenodo.6842323

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Miener, T. (2024). Indirect Dark Matter Searches in the Gamma-ray Band and Development of New Analysis Techniques for Ground-based Gamma-ray Astronomy.

Romanato, L. (2020). Implementation of new Convolutional Neural Network methods for Imaging Atmospheric Cherenkov Telescope analysis and application to the Large-Sized Telescope of CTA. *Master Dissertation submitted for the degree of Laurea Magistrale in Informatics, Università degli Studi di Milano.*

Grespan, P. (2020). Convolutional Neural Network data analysis development for the Large Sized Telescope of CTA and broadband study of the blazar 1ES 1959+ 650. *Master Dissertation submitted for the degree of Laurea Magistrale in Fisica, University of Padova.*

Marinello, N. (2019). Convolutional Neural Network Single-Telescope Reconstruction for the Large Size Telescope of CTA. *Thesis submitted for the degree of Laurea Magistrale In Telecommunication Engineering, University of Padova.*

Mariotti, E. (2019). Deep Learning on MAGIC: a Performance Evaluation for Very High Energy Gamma-Ray Astrophysics. *Thesis submitted for the degree of Laurea Magistrale In Telecommunication Engineering, University of Padova, April.*

Kim, B., Brill, A., Miener, T., Nieto, D., & Feng, Q. (2024). DL1-DataHandler: DL1 HDF5 writer, reader, and processor for IACT data (v0.12.0). https://doi.org/10.5281/zenodo.12699489

Linhoff, M., Beiske, L., Biederbeck, N., Fröse, S., Kosack, K., & Nickel, L. (2023). ctapipe – Prototype Open Event Reconstruction Pipeline for the Cherenkov Telescope Array. *Proceedings, 38th International Cosmic Ray Conference*, *444*(703). https://doi.org/10.22323/1.444.0703

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., Earl, N., Starkman, N., Bradley, L., Shupe, D. L., Patil, A. A., Corrales, L., Brasseur,

C. E., Nöthe, M., Donath, A., Tollerud, E., Morris, B. M., Ginsburg, A., Vaher, E., Weaver, B. A., Tocknell, J., Jamieson, W., ... Astropy Project Contributors. (2022). The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. *The Astrophysical Journal*, *935*(2), Artículo 167, 167. https://doi.org/10.3847/1538-4357/ac7c74

Miener, T., Nieto, D., Brill, A., Spencer, S., & Contreras, J. L. (2021). Reconstruction of stereoscopic CTA events using deep learning with CTLearn. *arXiv preprint arXiv:2109.05809*.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, *38*(5), 1-12.

Qu, H., & Gouskos, L. (2020). Jet tagging via particle clouds. *Physical Review D*, *101*(5), 056019.

Katz, B., Blum, K., Morag, J., & Waxman, E. (2010). What can we really learn from positron flux 'anomalies'? *Monthly Notices of the Royal Astronomical Society*, *405*(3), 1458-1472. https://doi.org/10.1111/j.1365-2966.2010.16568.x

Barak, R., Belotsky, K., & Shlepkina, E. (2023). Proposition of FSR Photon Suppression Employing a Two-Positron Decay Dark Matter Model to Explain Positron Anomaly in Cosmic Rays. *Universe*, *9*(8), 370.

Gaisser, T. K. (1982). Cosmic rays and particle physics. *Comments on Nuclear and Particle Physics*, *11*(1), 25-39.

Wikimedia Commons. (2006, febrero). Protonshower [Accessed: [Insert the date you accessed the file]]. https://commons.wikimedia.org/wiki/File:Protonshower.jpg

Al Samarai, I., Deligny, O., Lebrun, D., Letessier-Selvon, A., & Salamida, F. (2015). An estimate of the spectral intensity expected from the molecular Bremsstrahlung radiation in extensive air showers. *Astroparticle Physics*, *67*, 26-32.

Max Planck Institute for Nuclear Physics. (2023). Detection of high-energy gamma rays [Accessed: 21 August 2024]. https://www.mpi-hd.mpg.de/hfm/HESS/pages/about/telescopes/images/detection2s.jpg

Satalecka, K. (2010). Multimessenger studies of point-sources using the IceCube neutrino telescope and the MAGIC gamma-ray telescope.

Bigongiari, C. (2005). The MAGIC telescope. *arXiv preprint astro-ph/0512184*.

Abdalla et al. (2018). HESS first public test data release. *arXiv preprint arXiv:1810.04516*.

Weekes, T., Badran, H., Biller, S., Bond, I., Bradbury, S., Buckley, J., Carter-Lewis, D., Catanese, M., Criswell, S., Cui, W., et al. (2002). VERITAS: the very energetic radiation imaging telescope array system. *Astroparticle Physics*, *17*(2), 221-243.

Hofmann, W., & Zanin, R. (2024). The Cherenkov Telescope Array. En *Handbook of X-ray and Gamma-ray Astrophysics* (pp. 2787-2833). Springer.

Völk, H. J., & Bernlöhr, K. (2009). Imaging very high energy gamma-ray telescopes. *Experimental Astronomy*, *25*, 173-191.

López-Coto, R., Mazin, D., Paoletti, R., Bigas, O. B., & Cortina, J. (2016). The Topo-trigger: a new concept of stereo trigger system for imaging atmospheric Cherenkov telescopes. *Journal of Instrumentation*, *11*(04), P04005-P04005. https://doi.org/10.1088/1748-0221/11/04/p04005

Heck, D., Knapp, J., Capdevielle, J. N., Schatz, G., & Thouw, T. (1998). CORSIKA: A Monte Carlo code to simulate extensive air showers.

Nieto, D., Brill, A., Feng, Q., Humensky, T., Kim, B., Miener, T., Mukherjee, R., & Sevilla, J. (2019). Ctlearn: Deep learning for gamma-ray astronomy. *arXiv preprint arXiv:1912.09877*.

Ruiz, J. E. (2020, septiembre). ESCAPE WP4 Provenance Workshop [08/09/2020].

Miener, T., Nieto, D., López-Coto, R., Contreras, J., Green, J., Green, D., & Collaboration, E. (2022). The performance of the MAGIC telescopes using deep convolutional neural networks with CTLearn. *arXiv preprint arXiv:2211.16009*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. arXiv e-prints. *arXiv preprint arXiv:1512.03385*, *10*.

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132-7141.

Pröve, P.-L. (2017). Squeeze-and-Excitation Networks — towardsdatascience.com [[Accessed 08-08-2024]].

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, *17*(3), 261-272.

Nieto, D., Brill, A., Feng, Q., Jacquemont, M., Kim, B., Miener, T., & Vuillaume, T. (2019). Studying deep convolutional neural networks with hexagonal lattices for imaging atmospheric Cherenkov telescope event reconstruction. https://arxiv.org/abs/1912.09898

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Brill, A., Kim, B., Miener, T., & Nieto, D. (2024). CTLearn 0.9.0 [Accessed on 15 July 2024]. https://pypi.org/project/CTLearn/

Dominik, R. M., Linhoff, M., & Sitarek, J. (2023). Interpolation of Instrument Response Functions for the Cherenkov Telescope Array in the Context of pyirf. *Proceedings, 38th International Cosmic Ray Conference*, *444*(618). https://doi.org/10.22323/1.444.0703

Apache Software Foundation. (2024, 17 de marzo). *Hadoop* (Ver. 3.4.0). https://hadoop.apache.org

Abe, H., Abe, K., Abe, S., Acciari, V., Aguasca-Cabot, A., Agudo, I., Crespo, N. A., Aniello, T., Ansoldi, S., Antonelli, L., et al. (2023). Performance of the joint LST-1 and MAGIC observations evaluated with Crab Nebula data. *Astronomy & astrophysics*, *680*, A66.

Abe, H., Abe, K., Abe, S., Aguasca-Cabot, A., Agudo, I., Crespo, N. A., Antonelli, L., Aramo, C., Arbet-Engels, A., et al. (2023). Observations of the Crab Nebula and Pulsar with the Large-Sized Telescope Prototype of the Cherenkov Telescope Array. *arXiv preprint arXiv:2306.12960*.

Mas, V. (2023). ViTables 3.0.3 [A viewer for HDF5 files].

Guo, M.-H., Cai, J.-X., Liu, Z.-N., Mu, T.-J., Martin, R. R., & Hu, S.-M. (2021). Pct: Point cloud transformer. *Computational Visual Media*, *7*, 187-199.

Mikuni, V., & Canelli, F. (2021). Point cloud transformers applied to collider physics. *Machine Learning: Science and Technology*, *2*(3), 035027.

Xuan, T., Borca-Tasciuc, G., Zhu, Y., Sun, Y., Dean, C., Shi, Z., & Yu, D. (2022). Trigger Detection for the sPHENIX Experiment via Bipartite Graph Networks with Set Transformer. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 51-67.

Jiang, Z., Yin, D., Khoda, E. E., Loncar, V., Govorkova, E., Moreno, E., Harris, P., Hauck, S., & Hsu, S.-C. (2024). Ultra Fast Transformers on FPGAs for Particle Physics Experiments. *arXiv preprint arXiv:2402.01047*.

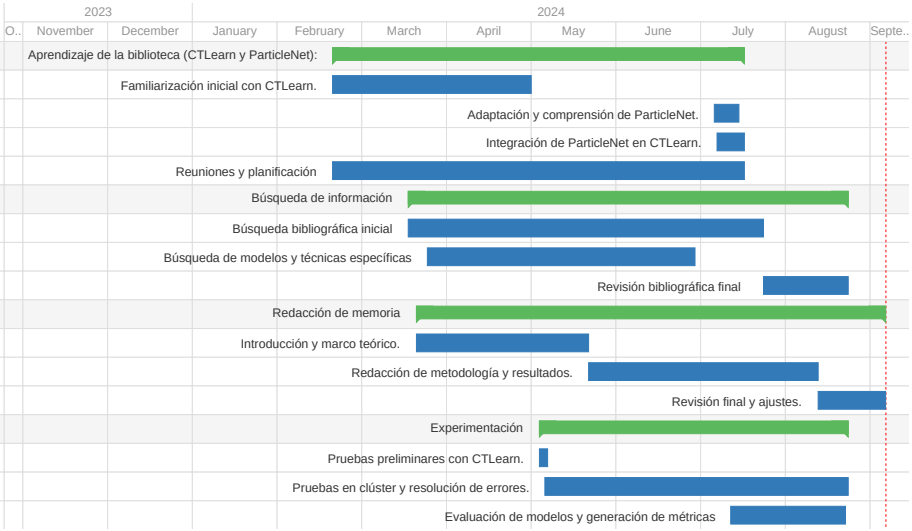
Anexos

Anexo 1: Planificación y Presupuesto

Los datos sobre los precios de coste se han obtenido de los siguientes enlaces:

- Coste uso GPU por hora: https://cloud.google.com/compute/gpus-pricing?hl=es-419
- Salario medio de un Data Scientist en España: https://es.talent.com/salary?job=data+scientist

Task	Start Date	End Date
Aprendizaje de la biblioteca (CTLearn y ParticleNet):	2/20/2024	7/16/2024
Familiarización inicial con CTLearn.	2/20/2024	4/30/2024
Adaptación y comprensión de ParticleNet.	7/6/2024	7/14/2024
Integración de ParticleNet en CTLearn.	7/7/2024	7/16/2024
Reuniones y planificación	2/20/2024	7/16/2024
Búsqueda de información	3/18/2024	8/23/2024
Búsqueda bibliográfica inicial	3/18/2024	7/23/2024
Búsqueda de modelos y técnicas específicas	3/25/2024	6/28/2024
Revisión bibliográfica final	7/24/2024	8/23/2024
Redacción de memoria	3/21/2024	9/6/2024
Introducción y marco teórico.	3/21/2024	5/21/2024
Redacción de metodología y resultados.	5/22/2024	8/12/2024
Revisión final y ajustes.	8/13/2024	9/6/2024
Experimentación	5/4/2024	8/23/2024
Pruebas preliminares con CTLearn.	5/4/2024	5/6/2024
Pruebas en clúster y resolución de errores.	5/6/2024	8/23/2024
Evaluación de modelos y generación de métricas	7/12/2024	8/22/2024



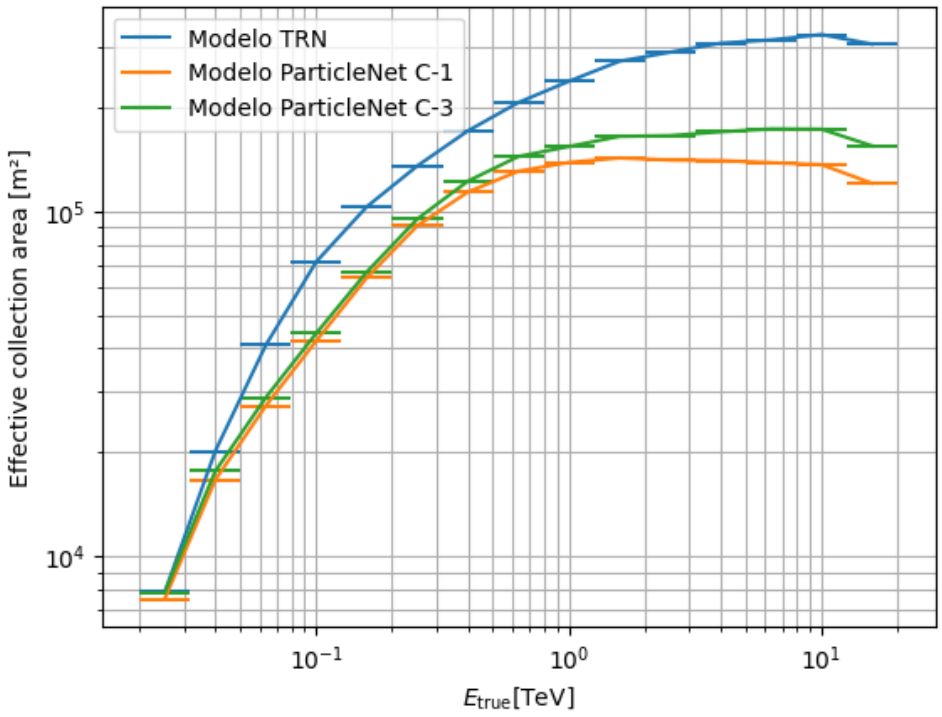
Motivo	Coste/hora
Precio medio GPU	0,45 €
Sueldo medio Data Scientist	20,94 €

Mes	Horas de trabajo	Coste
Febrero	6	125,64 €
Marzo	15,5	324,57 €
Abril	5,5	115,17 €
Mayo	9,5	198,93 €
Junio	25,5	533,97 €
Julio	67	1.402,98 €
Agosto	89	1.863,66 €
Septiembre	16	335,04 €
Total	234	4.899,96 €

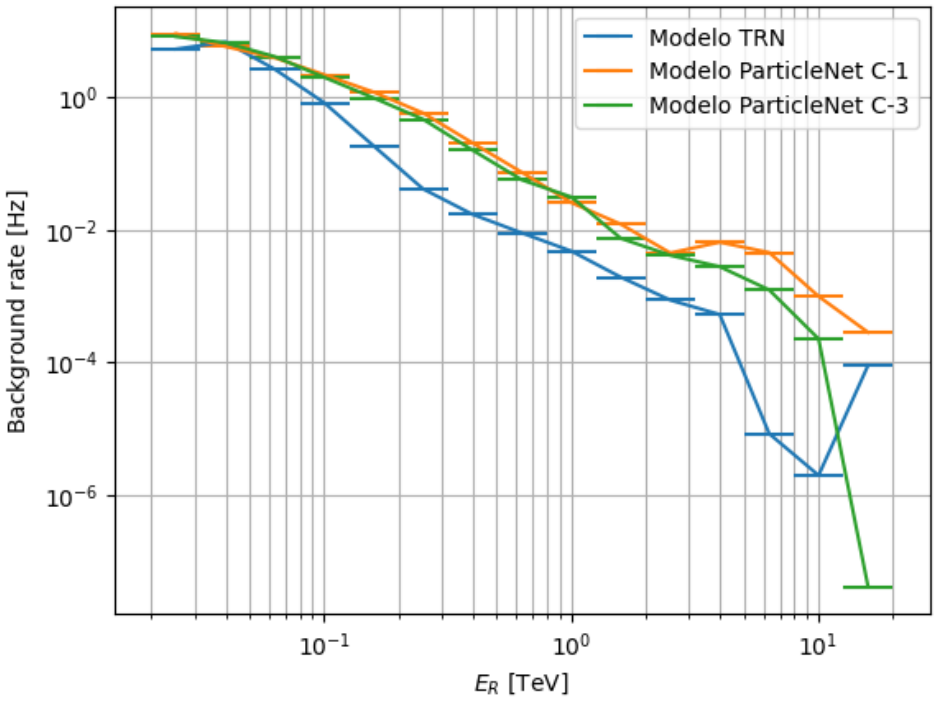
Evento	Horas de uso GPU	Coste
Experimentación	48	21,60 €
Entrenamiento Modelos		
ParticleNet	124	55,80 €
TRN	45	20,25 €
Predicción		
ParticleNet	36	16,20 €
TRN	9	4,05 €
Total	262	117,90 €

Coste total	
Horas Laborales	4.899,96 €
Infraestructura	117,90 €
Total	5.017,86 €

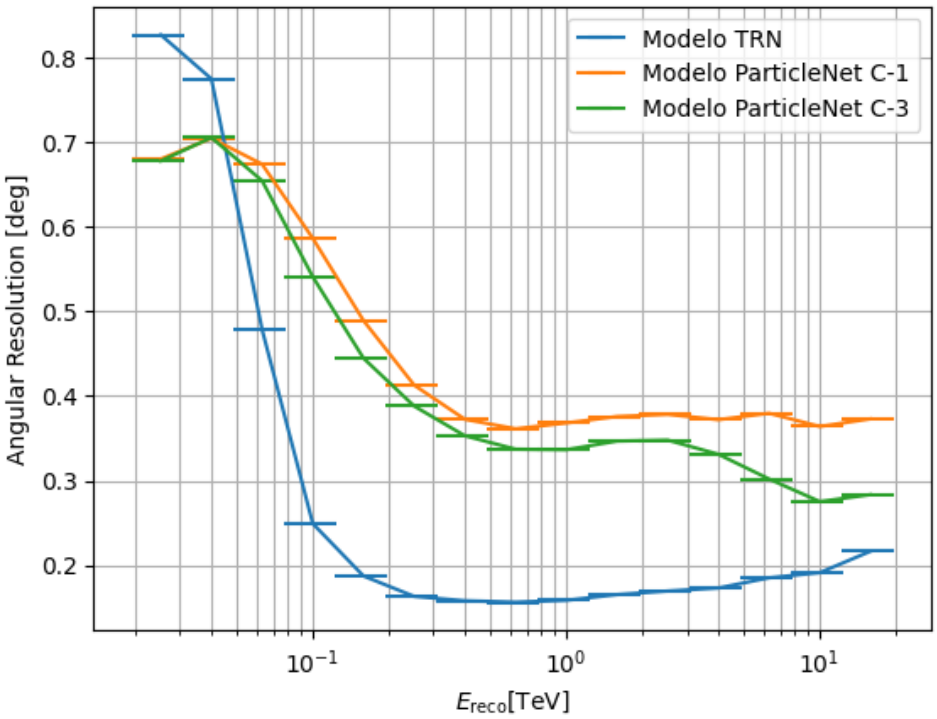
Anexo 2: IRFs



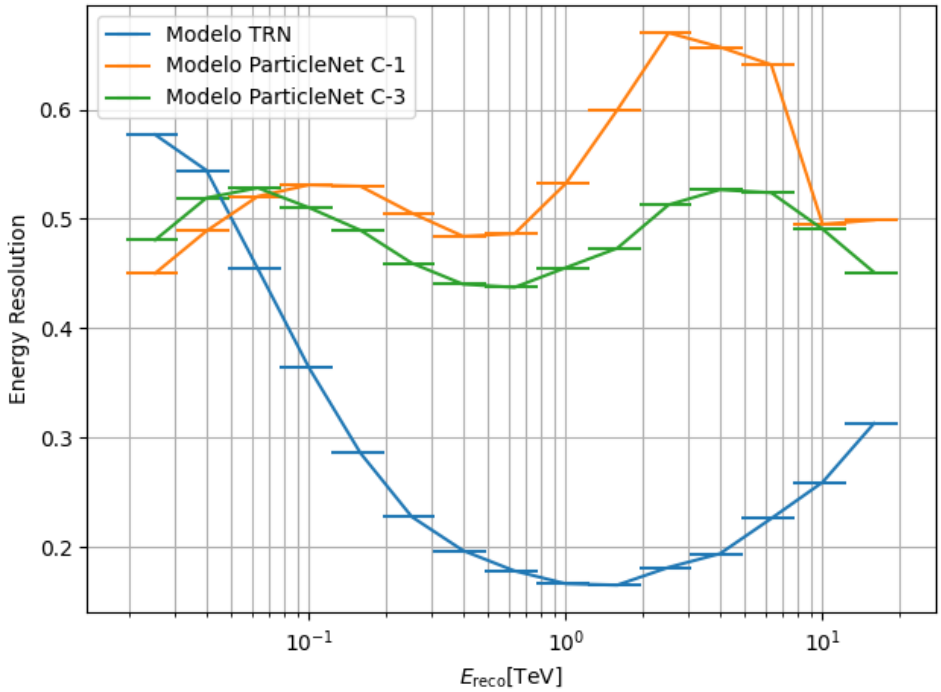
Área efectiva del sistema en función de la energía real para los modelos TRN y *ParticleNet* con Configuración 1 y Configuración 3.



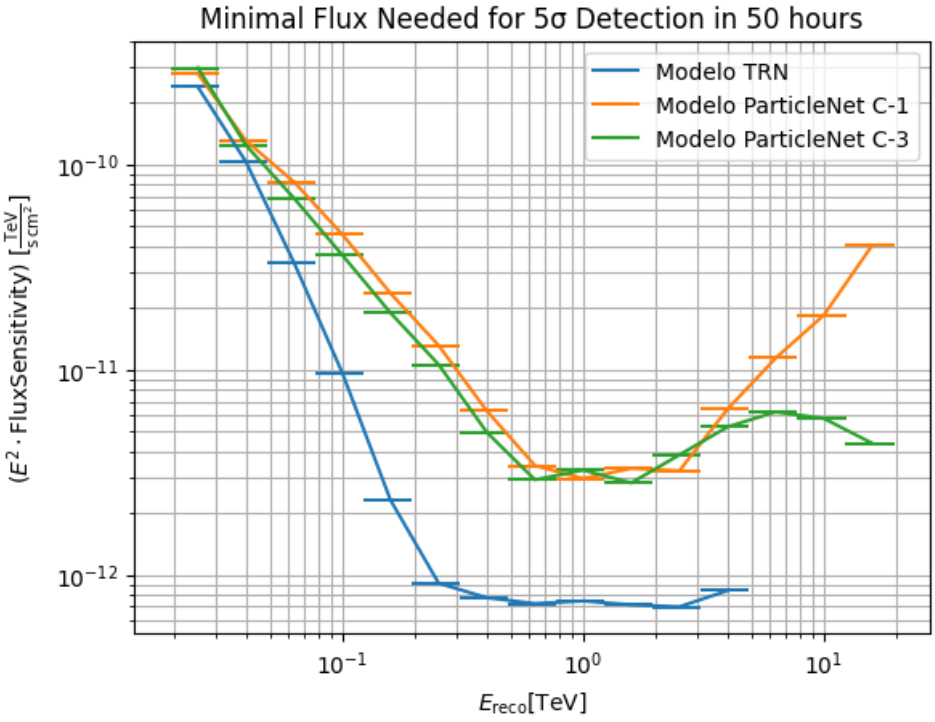
Tasa de eventos *background* por unidad de ángulo sólido reconstruida en función la energía reconstruida para los modelos TRN y *ParticleNet* con Configuración 1 y Configuración 3.



Resolución angular (grados decimales) en función de la energía reconstruida para los modelos TRN y *ParticleNet* con Configuración 1 y Configuración 3.



Resolución energética en función de la energía reconstruida para los modelos TRN y *ParticleNet* con Configuración 1 y Configuración 3.



Flujo mínimo para obtener una detección significativa en 50h en función de la energía reconstruida para los modelos TRN y *ParticleNet* con Configuración 1 y Configuración 3.