UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

RECONSTRUCCIÓN DE TRAYECTORIAS EN TELESCOPIOS SUBMARINOS DE NEUTRINOS MEDIANTE TÉCNICAS DE MACHINE LEARNING

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Jose Luis Rocabado Rocha

TUTORIZADO POR

Alicia Herrero Debón

Salva Ardid

CURSO ACADÉMICO: 2019/2020

Dedicatoria

A mi familia, y en especial a mis padres, por estar siempre a mi lado apoyándome. Por creer en mí, ayudarme siempre en todo lo posible y por regalarme los valores de humildad y trabajo duro que me acompañaran para siempre. Ellos han sido la base que me ha ayudado a llegar hasta aquí.

También a todos mis compañeros del grado en ingeniería electrónica y automática por regalarme una experiencia universitaria inolvidable.

Por último, a las personas extraordinarias que conocí durante mi intercambio como estudiante Erasmus que además de diversión, me regalaron diferentes puntos de vista que me ayudaron a crecer como persona.

Agradecimientos

A todos los profesores, particularmente a los que han conseguido transmitirme los conocimientos técnicos necesarios en nuestro grado y enseñarme que es necesario mantenerse actualizado, en el estado del arte.

Especial agradecimiento a Alicia Herrero Debón, Miguel Ardid Ramírez y Salva Ardid Ramírez, que a pesar de la situación que hemos vivido con la COVID-19 han sabido orientarme y ayudarme en la realización de este proyecto.

Resumen

En este proyecto se describe la utilización de algoritmos de Machine Learning para reconstruir la trayectoria de neutrinos procedentes de fuentes de energía astrofísicas lejanas utilizando la información de la radiación de Cherenkov captada en la matriz de fotomultiplicadores de los telescopios submarinos de neutrinos.

El objetivo de este proyecto es generar un modelo de Deep Learning, desde un punto de vista de detección de objetos mediante una red convolucional, capaz de predecir las componentes de la orientación del neutrino y su energía de eventos de una sola línea, es decir, eventos cuya luz de Cherenkov solo active los fotomultiplicadores de una fila vertical de la matriz. Con este objetivo, se utilizarán datos procedentes de una simulación.

Resum

A aquest projecte es descriu la utilització de algoritmes de Machine Learning per a reconstruir la trajectòria de neutrins procedents de fonts de energia astrofísiques llunyanes utilitzant la informació de la radiació de Cherenkov captada a la matriu de fotomultiplicadors dels telescopis submarins de neutrins.

L'objectiu de aquet projecte es generar un model de Deep Learning, des d'un punt de vista de detecció de objectes mitjançant una xarxa convolucional, capaç de predir les component de la orientació del neutrí y la seua energia a esdeveniments d'una sola línia, es a dir, esdeveniments on la llum de Cherenkov no més active els fotomultiplicadors d'una fila vertical de la matriu. Amb aquest objectiu, s'utilitzaran dades procedents d'una simulació.

Abstract

The using of Machine Learning algorithms for track reconstruction of neutrinos from remote astrophysical energy sources by analyzing the Cherenkov radiation information measured in the photomultipliers matrix of underwater neutrino telescopes is described in this project.

This project aim is to build a Deep Learning model, from object detection point of view by using a convolutional network. The model must be capable of predicting both orientation components and energy on a single line event (events whose Cherenkov light produces a hit on the photomultipliers placed in a single matrix vertical line). Simulated data will be used to achieve it.

Índice

Índice de figuras

Índice de tablas

DOCUMENTO I. MEMORIA

1. Objeto del proyecto (motivación, objetivos, metodología (?))

En la actualidad, nos encontramos en un contexto en el que la cantidad de datos que se generan y que podemos adquirir a través de diferentes técnicas o procesos tiende a infinito. Por otro lado, se plantean una serie de dificultades a los que hemos de hacer frente: el procesamiento y la extracción conclusiones relevantes de forma eficiente.

Sin embargo, los avances tecnológicos en poder computacional nos facilitan de manera importante dicha tarea gracias a la multitud de algoritmos de *Machine Learning* (ML) que buscan ofrecer una solución a los principales problemas de esta área (clasificación, regresión y *clustering*). Además, existen diferentes librerías de código libre que proporcionan una implementación fácil y orientada tanto a usuarios que se inician en el campo del ML como a usuarios que utilizan el análisis de datos en sus investigaciones. Estas librerías cuentan con una gran comunidad y una documentación muy detallada con diferentes ejemplos.

Es posible encontrar infinidad de aplicaciones y en diferentes áreas. Desde el ámbito empresarial, para poder hacer predicciones de stock o segmentar clientes, hasta el ámbito de la salud, donde se busca realizar diagnósticos con mayor rapidez y fiabilidad.

La aplicación desarrollada en este proyecto se encuentra en el área de la astrofísica y más específicamente, en la astronomía de neutrinos, pero ¿Por qué astronomía de neutrinos?

Gran parte del conocimiento que tenemos sobre el universo ha sido gracias a la observación de fotones. Sin embargo, las fuentes de energía astrofísicas son opacas a los fotones por lo que es imposible investigar las propiedades de estas regiones mediante una observación directa [1].

Por ello, para poder observar los mecanismos internos de los objetos astrofísicos y obtener una descripción del universo en un gran rango de energías, es necesario utilizar métodos indirectos y los neutrinos son los únicos candidatos conocidos. Los neutrinos son eléctricamente neutros, lo que permite que su trayectoria no se vea afectada por campos magnéticos, estables, lo que hace que puedan ser recibidas desde focos lejanos, y de interacción débil por lo que pueden penetrar regiones opacas a los fotones [1].

El objetivo del proyecto consistirá, por lo tanto, en reconstruir la trayectoria de los neutrinos mediante técnicas de ML para poder estudiar su focos de emisión.

2. Estado del arte

A lo largo de este apartado, se discurrirá desde los principios de detección, pasando por la descripción de la estructura del sensor hasta el análisis del actual algoritmo de reconstrucción de trayectorias actual y se finalizará con un análisis de la situación actual

2.1. Principio de detección

El telescopio ANTARES está optimizado para detectar neutrinos ascendentes de gran energía mediante la observación de la radiación de Cherenkov producida por leptones secundarios (muones), originados en la interacción de los neutrinos con la materia, en el agua del mar [2].

La radiación de Cherenkov se produce cuando una partícula alcanza una velocidad superior a la velocidad máxima de la luz (en el medio) originando un cono de fotones. Para detectar dicha radiación, el detector está formado por una matriz de detectores de luz o fotomultiplicadores contenidos en esferas de cristal denominados Módulos Ópticos (MO) dispuestos en líneas ancladas al fondo del mar [2]. Los MO, cuya posición es conocida, miden los tiempos de recepción de los fotones.

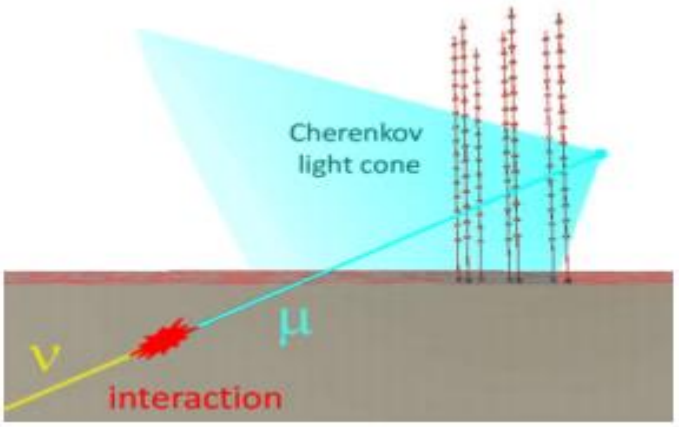

Fig. 1. Principio de detección de muon-neutrinos de gran energía en un telescopio submarino. El neutrino (ν) interacciona con la materia alrededor del detector para crear un muon (μ). El muon genera luz de Cherenkov en el agua del mar, la cual es detectada por una red tridimensional de sensores de luz. El espectro original de luz emitida por el muon es atenuada en el agua de forma que el rango de longitudes de onda dominantes detectados es entre 350 y 500 nm.[2]

2.2. Estructura del sensor

El detector está formado por una matriz de fotomultiplicadores distribuidos en 12 líneas verticales. En figura 2, podemos observar un esquemático con los principales componentes del detector en la superficie del lecho marino.

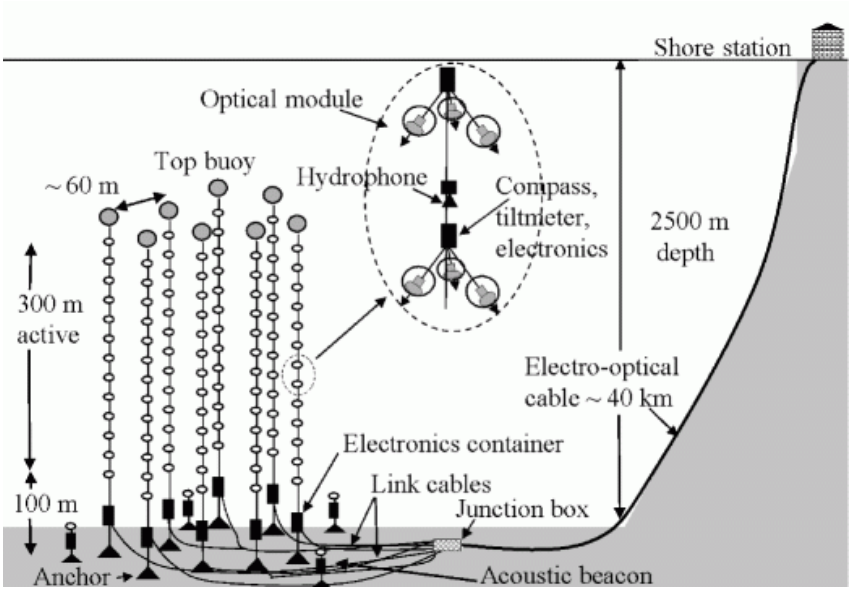

Fig. 2. Esquemático del detector del telescopio ANTARES. [3]

Los MO contienen dentro de una esfera de vidrio a los fotomultiplicadores además de otros sensores y la electrónica correspondiente que se encargará de adquirir información adicional. Si desea encontrar información técnica detallada consulte la referencia [2]. De la misma forma, cada piso está conformado por tres MO separados entre sí en 120º. Además, los MO están inclinados 45º por debajo de la horizontal favoreciendo la detección de los neutrinos ascendentes [3].

En el esquemático del detector, observamos que el detector está formado por 12 líneas con una altura aproximada de 350 m. La separación entre líneas es de aproximadamente 70 m y cada línea contiene un total de 25 pisos separados verticalmente 14.5 m entre sí [3]. En la siguiente figura podemos observar la disposición de las líneas en la superficie del suelo marino.

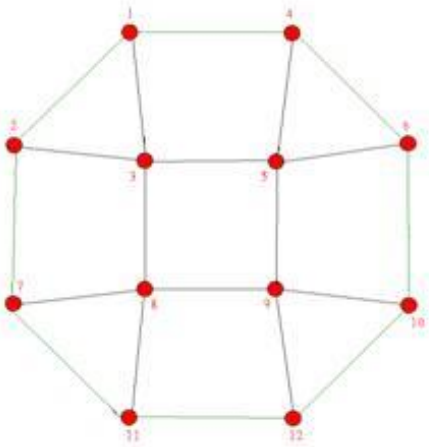

Fig. 3. Disposición de las 12 líneas en el suelo marino. [3]

2.3. Algoritmo de reconstrucción actual

Podemos distinguir dos tipos diferentes de eventos según que fotomultiplicadores han detectado fotones (*hits*). Si se han producido hits en diferentes líneas entonces podemos considerar que se trata de un evento multilínea. Por otra parte, si los *hits* solo se producen en una línea, entonces se considera que se trata de un evento unilínea (de una sola línea). Los eventos de una sola línea están asociados a muones de baja energía y solo es posible conocer su dirección en Zenit. En cambio, en los eventos multilínea es posible conocer toda la información de la dirección Azimut y Zenit [4].

Actualmente, se usan dos algoritmos de reconstrucción de trayectorias, AAFit y BBFit. El algoritmo AAFit es un algoritmo de basado en probabilidades que utiliza los parámetros lambda, parámetro de calidad, y beta, estimación del error angular. Este algoritmo, al contrario que el BBFit, es más eficiente en la detección de muones de alta energía [4].

El algoritmo BBFit, se basa en la minimización de un parámetro de calidad Q con ajuste chi cuadrado que tiene en cuenta la diferencia entre el tiempo esperado y el tiempo medido de los fotones en los MO. Además, la calidad Q tiene en cuenta la carga del *hit* medida y las distancias de viaje calculadas del fotón [5].

2.4. Análisis de la situación actual

En el apartado anterior, se he comentado de forma superficial las características de los algoritmos que se usan actualmente para determinar la trayectoria de los neutrinos. Estos algoritmos seleccionan los eventos de neutrinos ascendentes de los neutrinos descendentes producidos de la interacción de los rayos cósmicos con la atmosfera. También podemos encontrar como fondo neutrinos atmosféricos producidos por la propia radiación de la tierra [4].

Esta selección adquiere una mayor importancia en eventos de baja energía, puesto que las fuentes de fondo son captadas en el detector como ruido. En [5] podemos observar que el algoritmo BBFit tiene un gran desempeño a la hora de obtener información de la dirección en Azimut y Zenit. Sin embargo, deja un margen de mejora en cuanto a los eventos de una sola línea puesto que solo es capaz de ofrecer la dirección en Zenit.

Puesto que dentro de los eventos de neutrinos ascendentes, encontramos un porcentage de eventos de una sola línea lo suficientemente grande como para obviar, es necesario complementar el algoritmo BBFit para obtener una mejora significativa en el analisis de detección de neutrinos a baja energia en ANTARES.

3. Estudio de necesidades

Tal y como se ha constatado en los anteriores apartados, el objetivo principal de este proyecto es crear un algoritmo de reconstrucción de trayectorias de neutrinos mediante técnicas de ML usando los datos del detector de ANTARES que mejore los resultados para eventos de una sola línea del algoritmo BBFit. Sin embargo, es necesario acotar nuestra solución en base a diferentes consideraciones y condiciones que debe cumplir nuestro sistema.

3.1. Consideraciones

A pesar de que las líneas del detector se encuentras ancladas por un extremo al lecho del mar y por el otro a una boya para mantenerlas completamente rectas, la acción de las corrientes marinas puede inclinar las líneas respecto a la vertical. Se considera por lo tanto que las líneas se encuentran completamente verticales, sin tener en cuenta la acción de la marea.

Por el mismo motivo, se ignora las variaciones de posición de cada piso considerando, por lo tanto, los ángulos de los MO no varían y por extensión, la orientación de los fotomultiplicadores tampoco.

Por otro lado, se utilizarán datos de simulaciones generadas a partir de la estadística de los eventos reales. Además, estos datos incluyen información de las predicciones de los algoritmos BBFit y AAFit. Esta información adicional nos resultará útil puesto que nos permitirá trabajar solo con eventos de una sola línea utilizando la información de BBFit. De esta forma, es posible omitir la selección de eventos, es decir, que nuestros datos no tienen en consideración los eventos de fondo.

3.2. Condiciones

En este apartado, se definirán las condiciones que debe cumplir nuestro sistema y nos servirán como una guía para el diseño y nos ayudarán a valorar los resultados obtenidos.

De esta forma, el modelo debe de ser un modelo de aprendizaje supervisado, puesto que utilizamos registros de datos con la información correcta de las variables de interés. Además, puesto que las variables de interés de la dirección son valores continuos, se utilizará un modelo de regresión.

Además, se pretende diseñar un modelo donde el interés está en predecir una o varias salidas con un error mínimo según las entradas, sin tener en cuenta el proceso intermedio como si de una caja negra se tratara. El diseño del modelo se realiza mediante el ajuste de una serie de hiperparámetros, cuyo número varía según el algoritmo a usar. Sin embargo, no hay una regla que permita ajustar estos hiperparámetros, por lo que se definirán las condiciones de salida del sistema, obviando que las entradas deben de tener una correlación con la salida.

Para el caso de estudio del proyecto las variables, cuyo error se debe minimizar al máximo posible, de salida del modelo serán:

- Ux: La componente x de la dirección del neutrino
- Uy: La componente y de la dirección del neutrino. Esta variable junto con la Ux, son las de mayor interés puesto que son las variables que el algoritmo BBFit no puede definir en eventos de una sola línea.
- Ux · Uy: La multiplicación de ambas componentes Ux y Uy. Esta variable será de ayuda para la comprobación de los resultados de estas dos variables
- Uz: La componente z de la dirección del neutrino

4. Planteamiento de alternativas y justificación de la solución adoptada

En este apartado se discutirán las diferentes alternativas que podemos optar para abordar el problema planteado. Se hablará tanto de los diferentes algoritmos que permiten realizar una regresión como de las librerías y *frameworks* de código abierto que nos permitirán implementar dichos algoritmos. La mayoría de estas librerías están implementadas en el lenguaje de programación Python, aunque es posible encontrar cierta compatibilidad con C++.

4.1. Algoritmos

Podemos encontrar diferentes algoritmos de regresión con salida múltiple. A continuación, realizaremos una comparación entre las diferentes alternativas según sus capacidades predictivas:

- Métodos de transformación:
  Estos métodos consisten en transformar el problema de regresión con salida múltiple en varios métodos de regresión con una única salida. Posteriormente, se construye un modelo para cada objetivo para finalmente concatenar todas las predicciones realizadas por cada modelo. Dentro de este grupo de regresiones podemos encontrar a su vez otros algoritmos que se diferencian en la forma en la que concatenan las predicciones. El principal defecto de estos métodos es que las relaciones entre las salidas son ignoradas, puesto que existe un modelo diferente para cada salida, por lo que las predicciones son independientes. Esto puede afectar a la calidad global de las predicciones.

- Vector de soporte de regresión multisalida (M-O SVR):

  Este algoritmo está basado en el algoritmo de clasificación SVM (*Support Vector Machine*). El objetivo de este algoritmo considerar los puntos que se encuentra dentro de unos límites de decisión para encontrar un hiperplano que contenga el máximo número de puntos. De esta forma, el hiperplano mapeara las entradas para ofrecer las predicciones del modelo. El algoritmo de M-O SVR está optimizado para mejorar el rendimiento en la predicción.

- Arboles de regresión multisalida:

  Consiste en un algoritmo basado en árboles de regresión. Este algoritmo identifica de mejor manera, las dependencias entre las diferentes variables de salida. Su estructura en forma de árbol permite una mejor interpretación del modelo. Además, es posible incrementar su rendimiento en las predicciones si se usan técnicas de aprendizaje conjunto como el *Random Forest.* Este modelo tiene una buena calidad en las predicciones puesto que tiene una tendencia hacia la generalización, es decir, no produce *overfitting.*

- Redes neuronales artificiales:

  Algoritmo cuya unidad básica es una neurona simple. El cálculo realizado para calcular la predicción del modelo ($\hat{y}$) consiste en una suma ponderada de los *inputs* (X) y los *weights* (W) más un término denominado *bias* (b). Es posible utilizar una función de activación para introducir no-linealidad a la salida de la neurona.

  Las neuronas simples se apilan formando capas y a su vez, se acumulan capas formando la red neuronal. Normalmente hay una capa de entrada, varias capas intermedias o *hidden layers* y una capa de salida. En las *hidden layers* podemos encontrar diferentes tipos de capas como las capas de convolución. Es el algoritmo más conocido y es capaz de encontrar patrones ocultos en nuestro *dataset*, per esta razón necesita una mayor cantidad de datos.

Tras analizar todas las alternativas, se decide utilizar los algoritmos basados en redes neuronales artificiales, específicamente algoritmos de *Deep Learning* (DL). Se considera que es la mejor alternativa dado que tienen un buen nivel de adaptación a problemas complejos ya que combinan gran cantidad de ecuaciones no lineales con el aprendizaje. Además, este tipo de algoritmos es el más utilizado por los científicos de datos pues se considera como estado del arte dentro del ML.

La información de las demás alternativas se ha extraído de [6], consulte dicha referencia para obtener una mayor información tanto de las alternativas consideradas en este documento como otras posibles alternativas.

4.2. Librerías/*Frameworks*

Una vez seleccionado el tipo de algoritmo que se pretende utilizar, los *frameworks* que se pueden usar para implementar el modelo deseado son:

- Pytorch:

  Este *framework* basado en grafos computacionales dinámicos, esta optimizado para ofrecer velocidad y flexibilidad en el diseño de nuevos modelos. Optimiza la librería NumPy para acelerar notablemente las operaciones realizadas con los tensores utilizando GPU's de NVIDIA gracias al soporte de la librería de c/c++ CUDA.

  Además de los grafos computacionales dinámicos, Pytorch es considerado como *pythonic* por lo que la programación es clara, concisa y mantenible.

- Theano:

  Desarrollado por la universidad de Montreal, Theano es un compilador de expresiones matemáticas que integra la librería NumPy centrándose en el uso de GPU's con CUDA.

  Incluye un extenso soporte para realiza *unit-testing* y *self-verification* para poder detectar y diagnosticar diferentes tipos de errores.

- Tensorflow/Keras:

  TensorFlow es un *framework* desarrollado por Google que ofrece una API tanto de bajo nivel como de alto nivel dándole una gran flexibilidad al diseño del modelo además de una programación fácil. Originalmente, se basaba en el uso de grafos computacionales estáticos donde primero se creaba un grafo que marcaba el flujo de los datos de la computación y posteriormente se creaba una sesión para ejecutar los cálculos en el grafo. Finalmente, desde la versión 1.7, introdujeron el *eager mode* que permitía ejecutar los grafos creados de manera dinámica.

  Este *framework* integra en la versión 2.0 el API de alto nivel de Keras para construcción y entrenamiento de modelos de DL. Además, es compatible también con ejecución en GPU's gracias a CUDA e incluye la herramienta TensorBoard, un interfaz utilizado para visualiza el gráfico de nuestro modelo y otras herramientas para poder comprenderlo, depurarlo y optimizarlo.

Una vez realizada una comparativa entre los diferentes *frameworks*, se considera que la mejor opción es utilizar Tensorflow junto con el API de alto nivel de Keras ya que tiene una comunidad mayor que las demás alternativas, ofreciendo un mayor soporte a la hora de solucionar problemas que podamos encontrar a la hora de programar.

5. Descripción detallada de la Solución adoptada

A lo largo de este apartado se realizará una exposición detallada del procedimiento seguido para poder realizar la implementación de la solución adoptada. La opción seleccionada consiste en calcular las componentes de la dirección del neutrino mediante la construcción de una red neuronal artificial usando como datos de entrada los hits del evento de una sola línea.

Como en cualquier proyecto de DL, podemos encontrar al menos tres etapas en el diseño y desarrollo de un modelo. Estas etapas consisten en el pre-procesamiento de los datos a utilizar, entrenamiento del modelo diseñado y validación del modelo diseñado.

Para que sea posible obtener un modelo optimizado y con una buena predicción con datos externos a la muestra, se decide utilizar la técnica de t*rain/validation/test split.* Por lo tanto, se utilizará el set de *train* para entrenar nuestro modelo y ajustar los hiperparámetros validando el modelo con el set de *validation*. Finalmente, con el modelo ajustado, se evaluará su capacidad de predicción con el set de *test*.

5.1. Pre-procesamiento

La solución adoptada tendrá un enfoque alternativo, desde un punto de vista de análisis de visión por computador. De esta forma, se codificará la información relevante de cada evento para formar una imagen que alimentará nuestro modelo. Normalmente la primera etapa del modelo consiste en una etapa de convolución, donde se analiza la imagen para extraer sus características. En nuestro caso esta etapa se realizará de forma *offline* como se explica más adelantes. A continuación, la segunda etapa consiste en una etapa densa que se encargará de analizar las características extraídas en la etapa previa y realizará la regresión.

5.1.1. Adquisición de datos

Los datos obtenidos por la simulación se encuentran guardados en diferentes documentos de texto. Dentro de cada documento, podemos encontrar la información tanto de eventos multilínea como de una sola línea descrita en el apartado “3.1 Consideraciones”.

Para nuestro caso, solo seleccionaremos los eventos de una sola línea. Estos son los eventos que en la fila de bbfit_track los valores de la dirección X e Y son valores nan como observamos en figura 4.

MC_026030_anumu_CC_a_reco.i3.gz.txt

```
start_event  1
26030   80653   0   1171626940  2007-02-16 11:55:40.000,000,000,0 UTC
weights  1.844e+11 2073000.0 1000000.0
nu -0.483299074594 -0.199192030744 0.852493131576 32.449 79.169 17.009 238.993 -14
muon -0.476788950585 -0.231268976031 0.848048912107 32.449 79.169 17.009 89.6171 -13
aafit  0.411773266285 0.220312189554 0.884254101663 -18.6977165328 85.4393156683 1.11634941275 -6.74639801074 0.00886725723379
bbfit_track  nan nan 0.935263406199 1.81112697752 0.0 50.3841032889 1.56999996351
bbfit_bright  nan nan nan 18.1031898922 0.0 28.5485235608 1.85757122139
gridfit  0.0252236225964 -0.0884638511434 0.995759969021 9.00288925482 80.083035609 6.63149216643 7.40638889265
hit    1  3   2  2 -10.540966  37.058107 -156.955892 -0.694724 -0.131753 -0.707106  1325.764750   1.289410  246.181488
hit    2  3   2  2 -10.540966  37.058107 -156.955892 -0.694724 -0.131753 -0.707106   943.224130   1.366751  246.181488
hit    3  4  17  1  44.808764   8.549579   60.984108 -0.192247 -0.680472 -0.707106   690.171248   0.951508  209.274292
hit    4  4  17  1  44.808764   8.549579   60.984108 -0.192247 -0.680472 -0.707106 -1825.629739   0.950578  209.274292
hit    5  2   7  2  75.790857  57.183487  -84.226892 -0.680276  0.192937 -0.707106  -557.966919   0.778958  364.284515
                                                    ...
hit  212  1   7  2   4.457698  96.844844  -92.851892  0.123330  0.696269 -0.707106   -70.049168   1.088305  406.532288
BBFit selected pulses:
hit  1  1  15  0  3.789429  96.165758  23.344108 -0.694183 -0.134576 -0.707106  156.296461   1.663323    0.000000
hit  2  1  16  0  4.472172  95.709381  37.860108  0.141063 -0.692894 -0.707106  175.243045  13.308880  400.409698
hit  3  1  17  0  4.665830  96.764255  52.380108  0.377979  0.597606 -0.707106  221.973197   1.609686  154.914856
hit  4  1  18  0  4.721748  95.827495  66.890108  0.446387 -0.548397 -0.707106  269.716387   7.823990  370.330811
hit  5  1  19  0  3.839148  96.018753  81.433108 -0.633358 -0.314418 -0.707106  335.670843   3.570841  368.518829
end_event

start_event  2
```

Fig. 4. Recorte de la disposición de los datos en uno de los documentos de texto. El recuadro rojo nos indica que se trata de un evento de una sola línea.

La información será guardada por eventos en un *DataFrame* (DF) de la librería Pandas. En la siguiente figura observamos como se ha distribuido la información en el DF de eventos.

	run	numu	Event ID	Weights data	Nu data	Muon Data	bbfit track data	aafit data	Selected hits	Tr
270	025880	0	254	Weigths 0 2.56e+16 1 278600.0 2 1000...	Neutrino X 0.140462 Y 0.15...	Muon X 0.140877 Y 0.15...	bbfit track X ...	aafit X 0.152942 Y 0.14...	floor x y z ...	810.276849
271	025880	0	256	Weigths 0 2.08e+16 1 245400.0 2 1000...	Neutrino X 0.387216 Y -0.68...	Muon X 0.391666 Y -0....	bbfit track X ...	aafit X 0.373373 Y -0.71...	floor x y z ...	1077.629295
272	025880	0	260	Weigths 0 4.106e+15 1 103200.0 2 1000...	Neutrino X 0.021959 Y -0.18...	Muon X 0.021575 Y -0.18...	bbfit track X ...	aafit X 0.148830 Y -0.19...	floor x y z ...	1473.741722
273	025880	0	269	Weigths 0 2.931e+16 1 338800.0 2 1000...	Neutrino X 0.136278 Y -0.33...	Muon X 0.133983 Y -0.33...	bbfit track X ...	aafit X 0.341193 Y 0.04...	floor x y z ...	497.744290
274	025880	0	270	Weigths 0 1.609e+16 1 68350.0 2 1000...	Neutrino X -0.177463 Y -0.20...	Muon X -0.177386 Y -0.20...	bbfit track X ...	aafit X 0.024413 Y -0.04...	floor x y z ...	235.409485

Fig. 5. Recorte del DF que contiene la información de cada evento.

Las columnas "*run"* y "*numu"* nos permiten identificar el documento de texto al que pertenece el evento y se extraen del nombre del documento. *Numu* es una variable categórica codificada en binario que representa el tipo de interacción del evento, donde un valor de 0 indica que el evento es *anumu* (antineutrino - muon), mientras un 1 indica que es *numu* (neutrino - muon).

La columna de "*Event ID*" indica qué evento es dentro del documento y se extrae de la línea de "*start event*". Posteriormente, podemos encontrar columnas que contienen un DF anidado. En las tablas siguientes, podremos observar su distribución además de las columnas del documento de texto de donde se han adquirido los datos.

index	weights	line column
0	w0	1
1	w1	2
2	w2	3

Tabla 1. Distribución de los datos correspondientes a los pesos usados en el algoritmo BBFit además de la columna de la línea de la que se escoge el dato.

Las columnas de "*Nu data*" y "*Muon data*" contienen las mismas variables de interés. Debido a la interacción neutrino-muon las componentes de la dirección deben de ser similares y nos servirán para comprobar si la lectura de datos es correcta. Sin embargo, solo utilizaremos para el entrenamiento los datos de la columna "Nu data".

index	Muon/Neutrino	line column
X	dirección X	1
Y	dirección Y	2
Z	dirección Z	3
Energy	energia	7
Type	tipo	8

Tabla 2. Distribución de los datos del DF de las columnas "Nu data" y "Muon data" además de la columna de la línea de la que se escoge el dato.

index	Bbfit track	line column
X	nan	1
Y	nan	2
Z	dirección Z	3
zc	componente z del punto más cercano a la línea	4
dc	distancia a la línea	6
Resconstruction quality	Ajuste chi2	7

Tabla 3. Distribución del DF de la columna "bbfit track data". Contiene los resultados del algoritmo BBFit además de la columna de la línea de la que se escoge el dato.

index	aafit	line column
X	dirección X	1
Y	dirección Y	2
Z	dirección Z	3
lambda	Calidad de reconstrucción	7
beta	Precisión de la reconstrucción	8

Tabla 4. Distribución del DF de la columna "aafit data". Contiene los resultados del algoritmo AAFit además de la columna de la línea de la que se escoge el dato.

índex	floor	x	y	z	ux	uy	uz	t	A
n	piso del detector	posición x	posición y	posición z	orientación x MO	orientación y MO	orientación z MO	tiempo	amplitud
line column	3	5	6	7	8	9	10	11	12

Tabla 5. Distribución del DF de la columna "Selected hits". El índice n dependerá del número de hits seleccionados en cada evento.

El procedimiento para seleccionar los *hits* del documento de texto es el siguiente:

- Del apartado de "*BBFit selected pulses*" escogeremos el tiempo de referencia Tr como el menor tiempo de los *hits* seleccionados del evento.
- Una vez obtenido Tr, se escogerán los hits cuyos tiempos pertenezcan a un intervalo temporal de tal forma que, $\text{Selected hits}_n = \text{hits}(t_n \in [Tr + low_{tol}, Tr + up_{tol}])$ siendo low_{tol} la tolerancia inferior del intervalo temporal y up_{tol} la tolerancia superior del intervalo temporal. Para reducir el coste computacional, low_{tol} = -200 ns y up_{tol} = 600 ns, obteniendo una ventana temporal de 800 ns

Una vez guardado los datos de los documentos de texto, se obtiene un DF con 1245656 eventos de una sola línea. Puesto que tenemos un set de datos relativamente grande, se selecciona un *train/validation/test split* con un ratio de 98/1/1.

Antes de realizar la partición, se mezclan los eventos del DF mediante la función "DF.sample" de la líbreria pandas. Posteriormente, se utiliza una máscara para seleccionar aleatoriamente un 1% del set de datos para crear el set de *test* obteniendo un set de 12319 eventos.

A continuación, se guarda el resto del set de datos como el set de *train* y se selecciona nuevamente un 1% obteniendo un set de *validation* de 12333 eventos. Por último, se actualiza el set de *train*, reducido a 1221004 eventos. Nótese que la partición no es exactamente de 98/1/1, por lo que el tamaño de los sets de *validation* y *test* no son iguales.

Finalmente, se guardan los sets en documentos binarios *pickle*. Para evitar problemas de memoria en el procesamiento de los datos se divide el set de *train* en archivos *pickle* con 10000 eventos por archivo obteniendo 122 archivos con 10000 eventos y 1, con 1004.

5.1.2. Procesamiento de datos

Siguiendo la analogía a una imagen RGB, se calcula el ángulo zenit de cada *hit* usando la conversión zenit = atan2(uy/ux) y se calcula la distancia angular a los ejes RGB separados entre sí 120º como se observa en la figura 6.

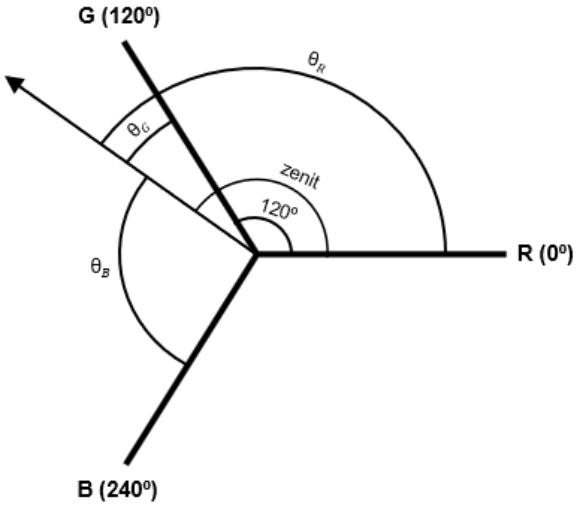

Fig. 6. Esquemático de la codificación en formato RGB de la orientación Zenit del MO

Posteriormente se calculan los pesos normalizados de cada canal usando su distancia angular θ con la siguiente fórmula:

$$w_{canal} = \begin{cases} \dfrac{120 - |\theta_{canal}|}{120} \; si \; \theta_{canal} < 120° \\ 0 \; si \;\; \theta_{canal} \geq 120° \end{cases} \qquad (1)$$

Una vez calculados los pesos, el valor RGB de cada píxel será [$w_R \cdot A$, $w_G \cdot A$, $w_B \cdot A$], donde A es la amplitud del pulso captado en el *hit*. Nótese que el valor del canal de color será equivalente a cero sí su distancia angular es mayor o igual que 120º.

La imagen generada tendrá, por lo tanto, una dimensión de [25, Nt, 3] donde Nt es el ratio de la ventana temporal entre el periodo de muestreo Ts + 1 ya que se tienen en cuenta los extremos del intervalo temporal, 25 es el número de pisos que hay en una línea y 3 son los canales de color. Se decide seleccionar un Ts de 5 ns para reducir el coste computacional, obteniendo un Nt de 161, por lo que la dimensión de la imagen del evento será [25,161,3].

Para conseguir un suavizado en la foto se decide usar un filtro de regresión gaussiana. La regresión se realiza para cada piso del evento y para cada canal de color y una vez obtenida la imagen se normalizan los valores. En la figura 7 podemos observar un ejemplo de la regresión gaussiana y en la figura 8 un ejemplo de la foto del evento.

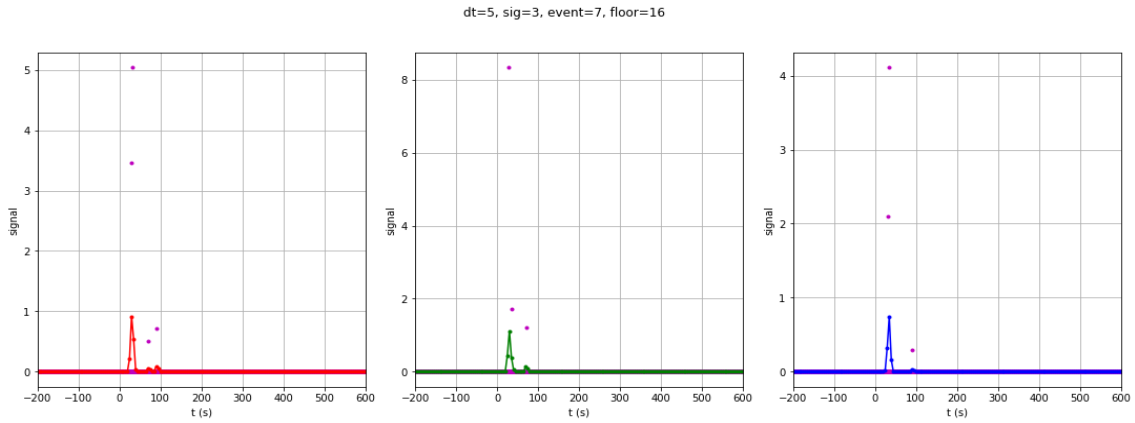

Fig. 7. Gráficas comparativas de la señal original (puntos morados) del piso 16 y el resultado de la regresión con un Ts = 5 ns y una sigma de 3 (de derecha a izquierda: rojo, verde, azul). El valor resultante de la regresión esta amplificado por una ganancia de 10 para poder apreciar el efecto de la regresión.

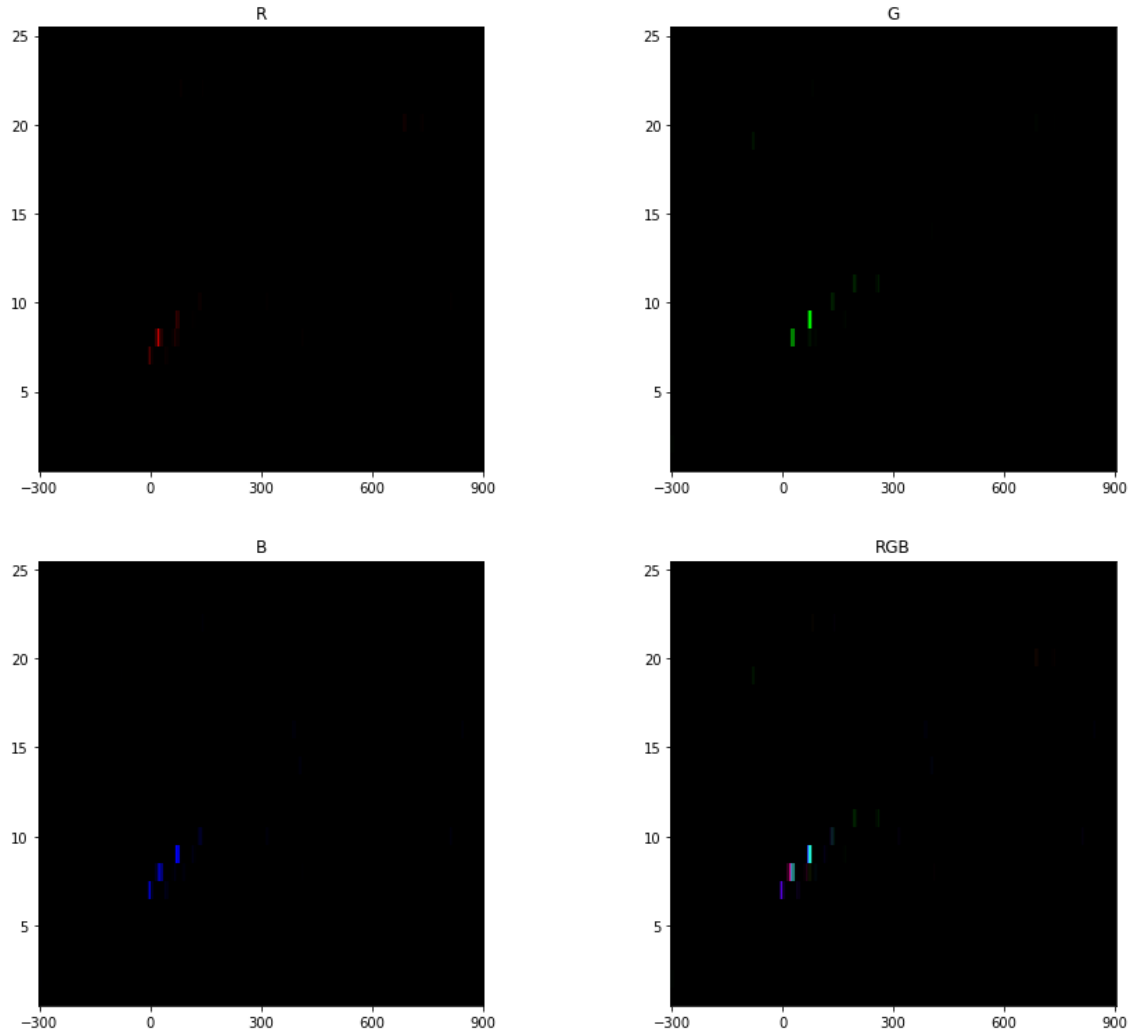

Fig. 8. De derecha a izquierda y de arriba abajo: Imagen del canal R, imagen del canal G, imagen del canal B e imagen RGB del evento.

Tras visualizar varias series temporales de la regresión de suavizado, se observa que en ciertos pixeles obtenemos un valor de señal con signo negativo debido a que, en algunos hits, la amplitud de la señal captada es negativa. Estos valores no tienen una explicación desde el punto de vista físico por lo que se considera como ruido. Para reducir la complejidad del código y dado que el set de datos completo es amplio se decide prescindir de los eventos en los que se encuentre algún canal de color negativo dejándonos con un *split* de 1206452 eventos de *train*, 12208 eventos de *validation* y 12169 de *test*.

Finalmente, para reducir el coste computacional se decide realizar la etapa convolucional *off-line* usando la función “tf.nn.conv2d” del *framework* de TensorFlow. Se utiliza un *stride* de 1 además de *zero-padding* para obtener el mismo tamaño de imagen. Puesto que se quiere resaltar el evento de la imagen los pesos del filtro equivalen a 1. Con el mismo motivo, la convolución 2D se aplicará a la imagen del evento transformado a escala de grises. Está transformación se realizará mediante el promediado de los tres canales de color.
En la siguiente figura podemos encontrar una comparativa de los resultados de la convolución con diferentes tamaños de *kernel*.

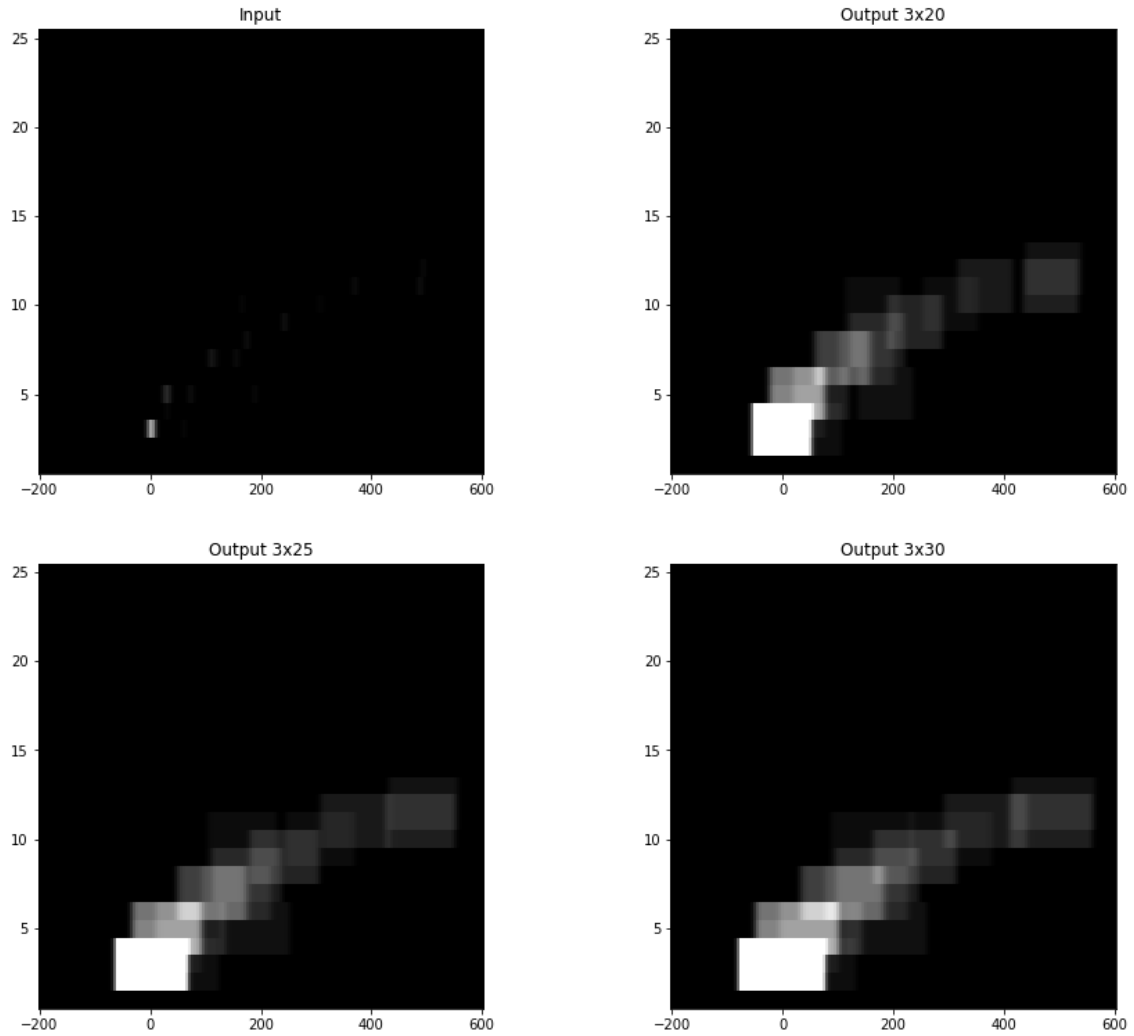

Fig. 9. Imagen comparativa de la imagen del evento en escala de grises y las salidas de la convolución 2D con un kernel de dimensión 3x20, 3x25 y 3x30.

Se considera que el *kernel* de 3x20 es el que mejor resalta el evento sin generar demasiada distorsión. La información de la salida obtenida tras realizar la convolución 2D se añadirá al tensor final “X1” que utilizaremos para entrenar la red neuronal artificial obteniendo un tensor final con una dimensión de [nº eventos, 25, 161, 4] donde 4 se corresponde a los 3 valores RGB más el valor de la salida de la convolución.

Se utilizará también el tensor “X2” para realizar el entrenamiento. El tensor “X2” incluye la información de “Z”, “dc”, “zc” y “*Reconstruction quality*” del DF “bbfit *track data*” (tabla 3) por lo que la dimensión de “X2” es [nº eventos, 4].

Una vez obtenidos todos los tensores, es necesario normalizar los datos para la aportación de las diferentes variables al modelo sean iguales, es decir, que se encuentren en el mismo rango. A las imágenes del tensor “X1” se realizará un escalado de los valores para conseguir un rango de valores entre 0 y 1. A continuación se aplicará la normalización *z-score* a las imágenes y a los valores de la convolución 2D a nivel de evento. Por último, se aplica la normalización *z-score* al tensor “X2” y al tensor de salida a nivel del set de datos completo.

Puesto que todo el procesamiento se realiza usando los datos de los DF de cada set, se crearán diferentes directorios, uno para cada set, que contendrán documentos comprimidos con extensión *npz* con los tensores de entrada del modelo y la salida real deseada.

5.2. Entrenamiento y ajuste

Una vez realizado el preprocesamiento de los datos, se procede a realizar el entrenamiento y ajuste del modelo. El procedimiento que se sigue es el siguiente:

- Seleccionar los hiperparámetros del modelo
- Entrenar el modelo con el set de *train* y realizar la validación con el set de *validation*.
- Visualizar las variables del entrenamiento y de la validación en la herramienta TensorBoard
- Ajustar hiperparámetros y repetir los dos pasos anteriores.

En la siguiente tabla podemos encontrar la nomenclatura de los hiperparámetros además de una breve descripción.

Hiperparámetros	Descripción
Nº *Hidden Layers*	Número de capas ocultas que tendrá nuestro modelo.
Nº *Neurons/Layer*	Número de neuronas artificiales que utilizaremos en cada capa. Junto con el número de capas ocultas, se conforma la estructura del modelo
Activation function	Función de activación que usaran las neuronas de cada capa.
Loss function	Función de coste o pérdida. Se trata de la función a optimizar. En regresión las más usadas son el error medio absoluto (MAE), el error cuadrático medio (MSE) y el error de Huber
Optimizer	Método para optimizar la función de coste actualizando los pesos W y el *bias* b de cada neurona
Metrics	Métricas para evaluar por el modelo durante el entrenamiento
Learning rate	Velocidad de convergencia. Un valor muy elevado, puede hacer que el modelo no converja. Por el contrario, un valor bajo hace que el modelo encuentre el mínimo lentamente
Batch size	Número de muestras del set que utilizados antes de que nuestro modelo actualice los parámetros W y b
Nº *Epochs*	Número de veces que el set de entrenamiento alimenta nuestro modelo

Tabla 6. Enumeración y descripción de los principales hiperparámetros de una red neuronal artificial que se deben ajustar según el objetivo de nuestro modelo.

Para empezar, se decide utilizar una estructura simple como se muestra en la tabla VII. Como función de coste se selecciona el MAE puesto que otorga mayor robustez al modelo respecto a valores atípicos. En cuanto al método de optimización, se escoge el algoritmo adaptativo ADAM, que facilita la convergencia adaptando el *learning rate*, sin embargo, es necesario utilizar un valor inicial de este, que en nuestro caso se utilizará el valor recomendado de 0.001 en [7]. Además, se seleccionan las métricas de MAE y un *batch size* de 64. Respecto al número de *epochs* se fija a 50. Sin embargo, este parámetro no es tan relevante puesto que se implementará el método *early stopping* para prevenir *overfitting*. Se debe tener en cuenta que, puesto que nuestra red tiene 5 salidas, el valor de la función de coste consistirá en la suma ponderada de los costes de cada salida. Dicha ponderación es, por defecto, igual a 1.

Tipo de capa	Dimensión de salida
X1 input + Flatten	16100
X2 input	4
Concatenated input	16104
Dense + ReLu	128
Dense + ReLu	32
Dense	5

Tabla 7. Estructura inicial del modelo de red neuronal artificial de regresión con múltiples salidas.

Mediante la herramienta TensorBoard ofrecida por TensorFlow es posible monitorizar el desarrollo del entrenamiento del proyecto. TensorBoard registra y muestra los valores de las métricas seleccionadas y las funciones de coste de cada salida, así como la función de coste total en cada *epoch*. También te permite visualizar los grafos de computación y los flujos de datos. En la figura 10 encontramos la visualización del grafo y en la figura 11, las gráficas de los valores registrados.

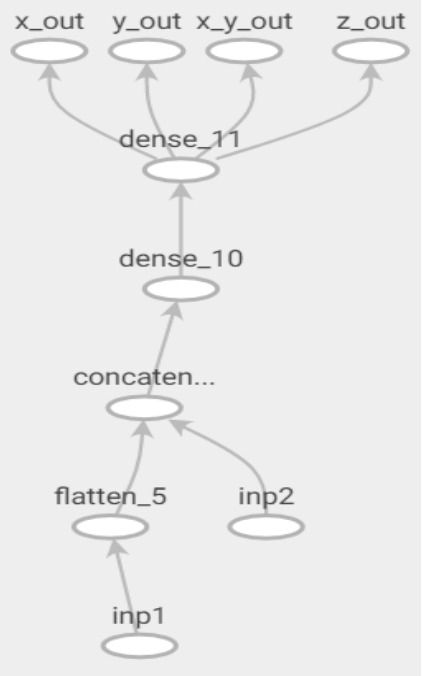

Fig. 10. Recorte de la visualización en la herramienta TensorBoard de la estructura del grafo (modelo).

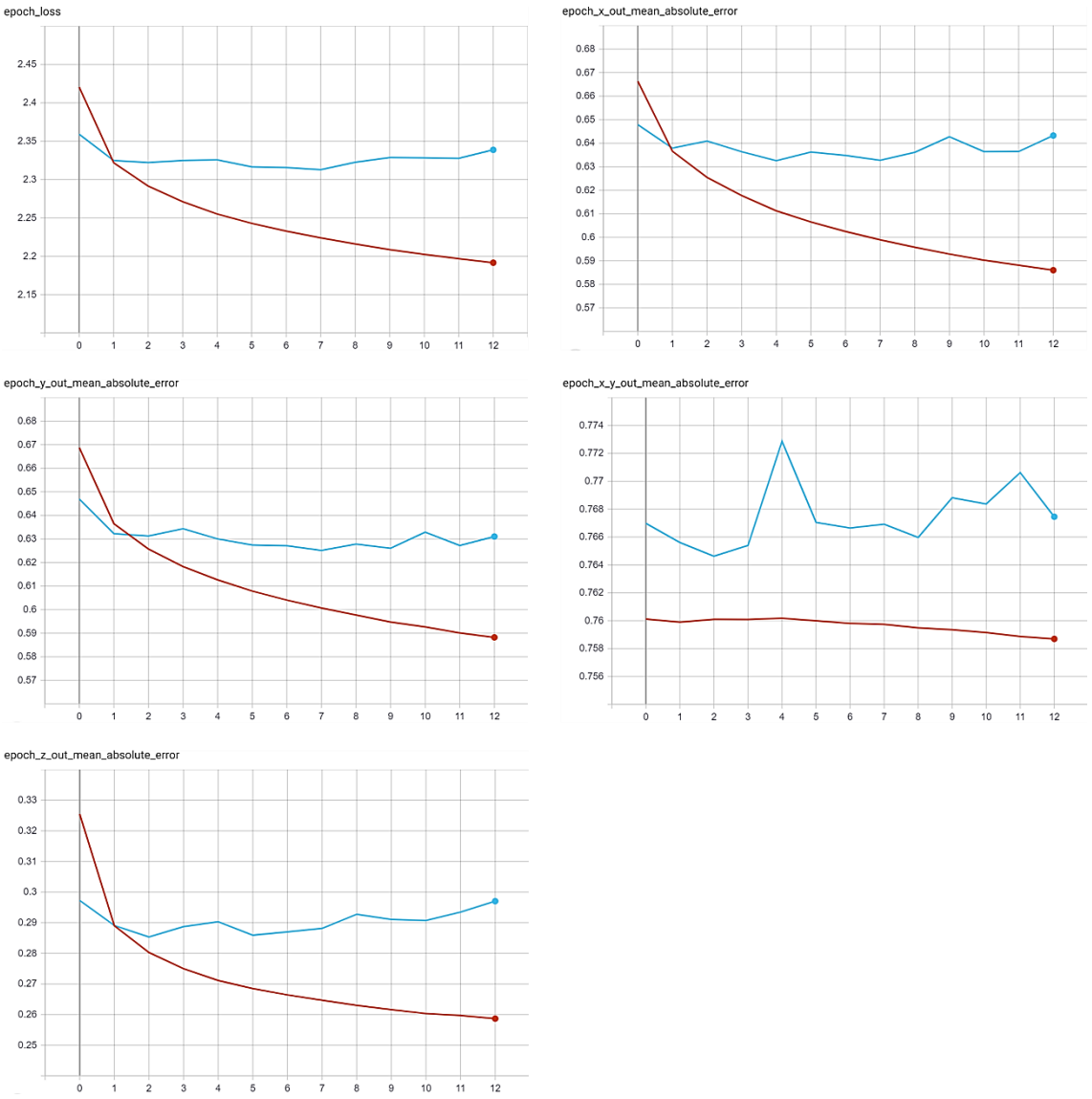

Fig. 11. Recortes de las gráficas del MAE por epoch del modelo inicial obtenidas mediante la herramienta TensorBoard (Rojo: entrenamiento, Azul: validación). De derecha a izquierda y de arriba abajo: función de coste total, MAE de la salida Ux, MAE de la salida Uy, MAE de la salida Ux · Uy y MAE de la salida Uz. Nótese que la implementación del early stopping se ha realizado para que el entrenamiento finalice si el coste total de validación no mejora pasados 5 epochs consecutivos.

Observamos que globalmente el modelo tiene una convergencia muy pronta entre el entrenamiento y la validación (alrededor del segundo *epoch*) indicando que nuestro modelo ha sufrido *overfitting* (la validación no sigue una tendencia descendente). Esto puede ser debido a que nuestro modelo no es lo suficientemente complejo como para poder encontrar una relación entre los datos de entrada y los de salida. También observamos que el modelo tarda alrededor de 6 *epochs* en comenzar a aprender la variable Ux·Uy. Dicho suceso podría deberse también a la poca complejidad del modelo. En la siguiente figura observamos el resultado de nuestro modelo tras incrementar la complejidad de la arquitectura del modelo y utilizando el método de *dropout* para prevenir el *overfitting*.

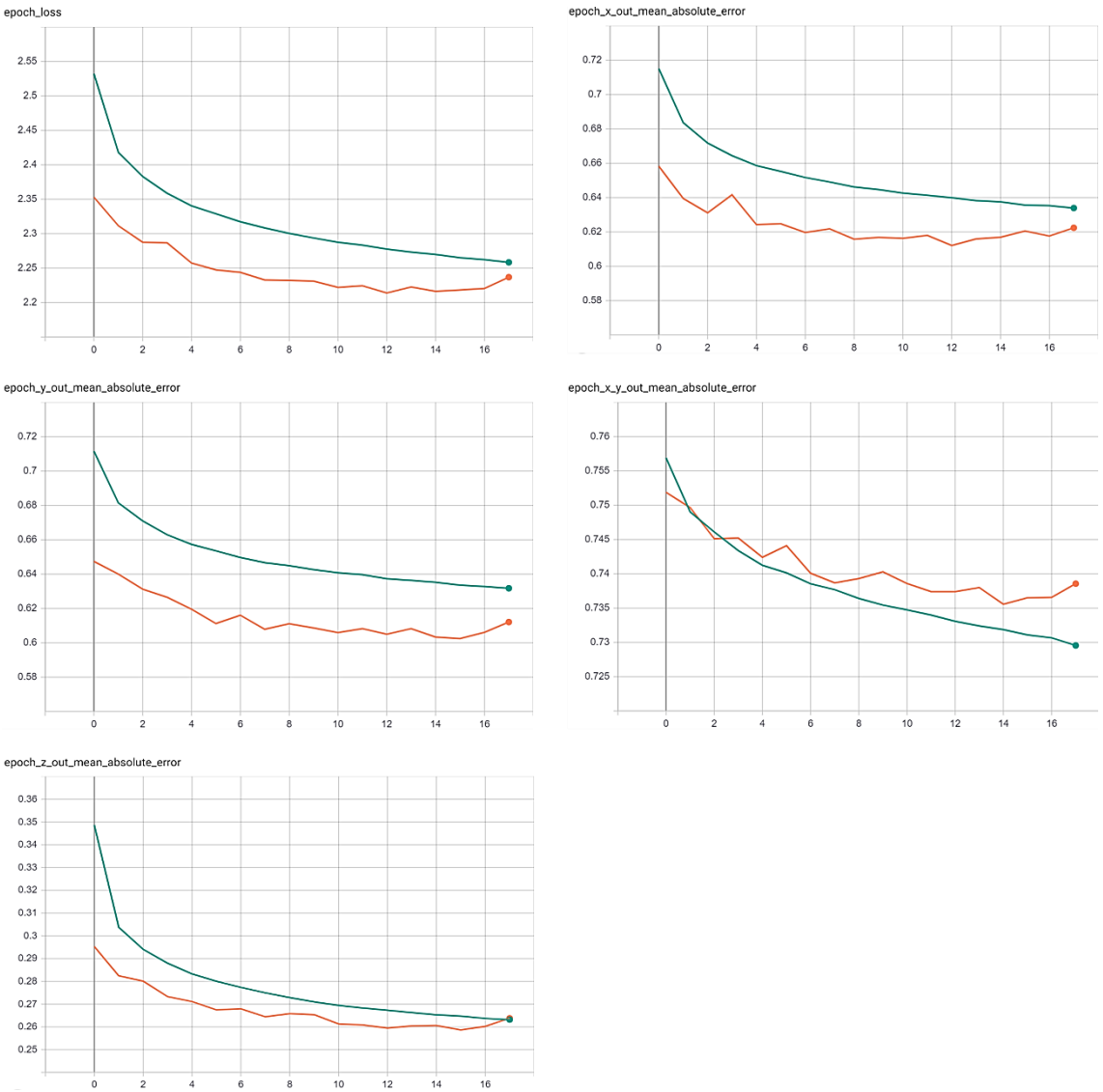

Fig. 12. Recortes de las gráficas del MAE por epoch del segundo modelo obtenidas mediante la herramienta TensorBoard (Verde: entrenamiento, Naranja: validación). De derecha a izquierda y de arriba abajo: función de coste total, MAE de la salida Ux, MAE de la salida Uy, MAE de la salida Ux · Uy y MAE de la salida Uz.

Tipo de capa	Dimensión de salida
X1 input + Flatten	16100
X2 input	4
Concatenated input	16104
Dropout (δ = 0.5)	16104
Dense + ReLu	128
Dense + ReLu	128
Dense + ReLu	32
Dense + ReLu	32
Dense	5

Tabla 8. Estructura del segundo modelo de red neuronal artificial de regresión con múltiples salidas. El símbolo δ representa la fracción de dropout usado.

Se comprueba que tanto el uso del *dropout* y el incremento de capas ocultas, como se observa en la tabla 8, mejora los resultados de nuestro modelo. De esta forma se decide ajustar los hiperparámetros relacionados con la arquitectura del modelo.

Finalmente, tras finalizar el ajuste de los hiperparámetros, el mejor modelo obtenido posee la arquitectura descrita en la siguiente tabla. Se comprueba que incrementar la complejidad no mejora significativamente, por el contrario, facilita el *overfitting* del modelo. Una práctica común cuando se usan funciones de activación como la función ReLu es utilizar capas de *batch normalization* para dar estabilidad al modelo y acelerar el entrenamiento. Además, añade cierta regularización que previene el *overfitting*. En la siguiente tabla encontramos la arquitectura seleccionada para nuestro modelo y en la figura 13 podemos observar los resultados obtenidos en TensorBoard.

Tipo de capa	Dimensión de salida
X1 input + Flatten	16100
X2 input	4
Concatenated input	16104
Dropout (δ = 0.5)	16104
Dense + ReLu	128
Batch Normalization	128
Dense + ReLu	128
Batch Normalization	128
Dense + Relu	32
Batch Normalization	32
Dense + Relu	32
Dense	5

Tabla 9. Estructura final del modelo de red neuronal artificial de regresión con múltiples salidas con entrada. El símbolo δ representa la fracción de dropout usado.

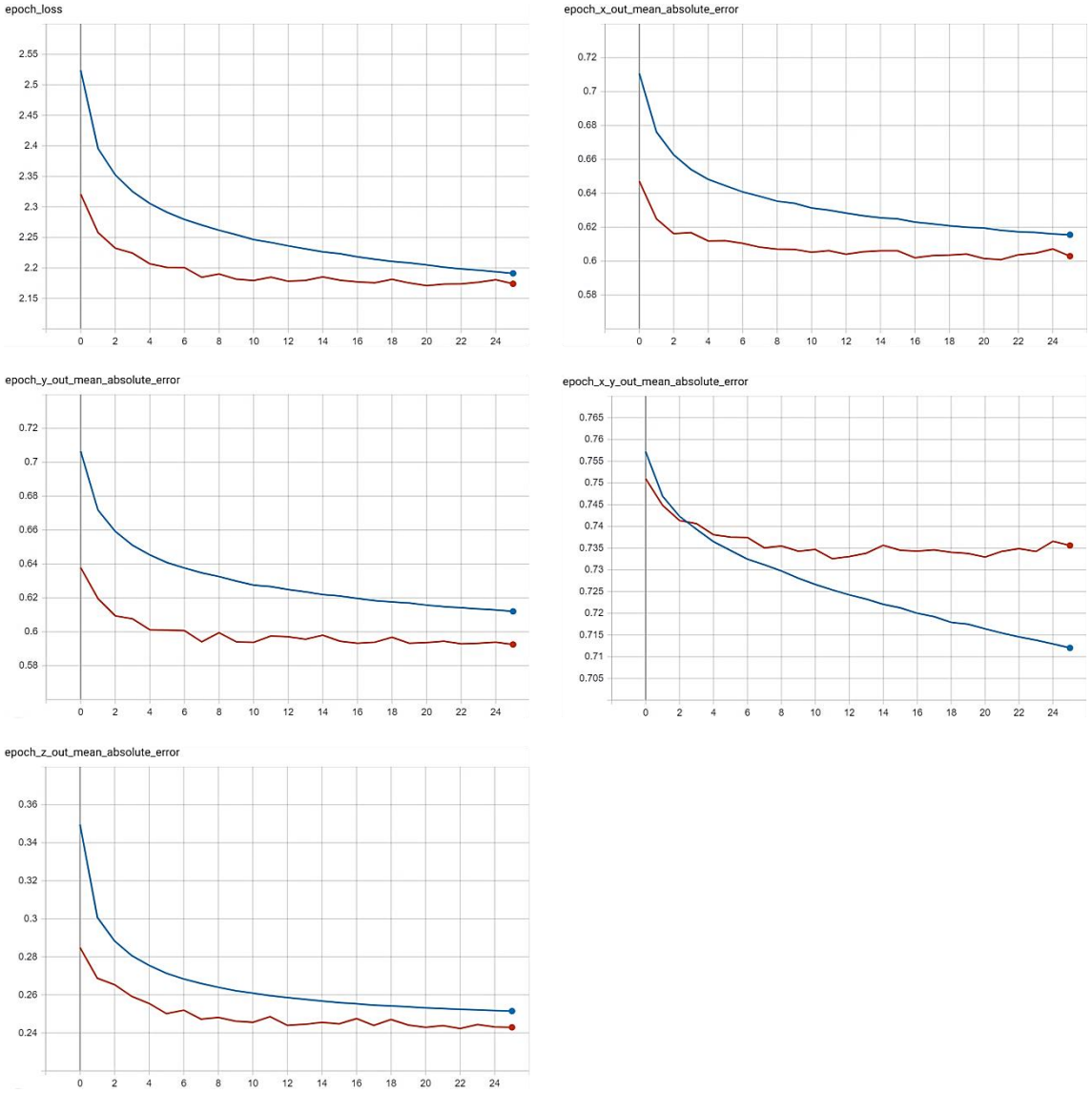

Fig. 13. Recortes de las gráficas del MAE por epoch del modelo final obtenidas mediante la herramienta TensorBoard (Azul: entrenamiento, Naranja: validación). De derecha a izquierda y de arriba abajo: función de coste total, MAE de la salida Ux, MAE de la salida Uy, MAE de la salida Ux · Uy y MAE de la salida Uz.

5.3. Evaluación

Para la realización de la evaluación del modelo se entrenará el modelo obtenido en el apartado anterior utilizando tanto los datos del set de *train* como los del set de *validation*. A continuación, utilizaremos el set de *test* para comprobar las respuestas de nuestro modelo a datos nunca vistos anteriormente y se compararán las predicciones obtenidas con las salidas verdaderas. También analizaremos las predicciones del set de *train* y *validation* y se realizará una comparativa con el modelo BBFit.

Los resultados obtenidos se discutirán en el apartado siguiente.

6. Justificación de la solución adoptada

En este apartado se exponen los resultados obtenidos con las predicciones utilizando el modelo final además de una comparativa en el rendimiento con el algoritmo BBFit. Para una evaluación más intuitiva y simple se utilizarán los ángulos de dirección azimut y zenit convirtiendo las componentes de la dirección. Dado que para la conversión a zenit Uz debe de estar entre -1 y 1, de no ser así devolverá un valor “nan”, se comprueba si hay predicciones de Uz fuera de este rango. Se observa que el porcentaje de eventos que se encuentran fuera de este rango para el set de *test, validation* y *train* es de 0.84%, 0.67% y 0.81% con un MAE de 0.0283, 0.0304 y 0.0261 respectivamente. De esta forma, puesto que el porcentaje de ocurrencia y el MAE es bajo, se decide corregir estas predicciones cambiándoles su valor a -1 o 1 según corresponda. También se realizará una comparación de las métricas MAE y RMSE.

Antes que nada, discurriremos las diferencias de rendimiento entre nuestro modelo y BBFit utilizando métricas de evaluación. En la siguiente tabla, podemos encontrar una comparación del modelo desarrollado y el algoritmo BBFit por medio de las métricas MAE y RMSE utilizando los datos del set de *test*.

Salida	MAE modelo	MAE BBFit	RMSE modelo	RMSE BBFit
Ux	0.3020	None	0.3998	None
Uy	0.2989	None	0.3929	None
Ux · Uy	0.1538	None	0.2099	None
Uz	0.1353	0.2029	0.2098	0.3431

Tabla 10. Comparación de los valores de las métricas MAE y RMSE obtenidas con nuestro modelo respecto a las obtenidas por el algoritmo BBFit usando el set de test.

Dado que el algoritmo BBFit solo proporciona la variable Uz no es posible calcular sus métricas respecto a las demás variables de salida por lo que únicamente es posible comparar la componente Uz. Se observa en nuestro modelo tiene una mejora en el MAE de alrededor de un 33 % y aproximadamente un 41% para el RMSE.

En la figura 14 podemos encontrar una primera visualización del resultado de las predicciones, tanto de nuestro modelo como de BBFit. Además de un *violin plot* que contiene información de la distribución del error angular absoluto, podemos encontrar un histograma 2D de las predicciones respecto a los valores reales. Se observa como el modelo obtiene predicciones mejores de zenit que BBFit. Cabe remarcar que en el histograma del ángulo de azimut encontramos dos zonas de densidades considerables alejadas de la diagonal ideal de predicciones (cuando el valor predicho es igual al real) debido a que azimut toma valores entre -180º y +180º. Por lo tanto, en los extremos, una predicción de -180º es igual a un valor real de 180º y al revés.

Una práctica común dentro del ML es realizar una comparativa de los resultados de nuestro modelo respecto a los diferentes sets de datos (*train*/*validation*/*test*) que hemos utilizado. En la tabla 11 podemos los resultados de las métricas se cada set de datos.

Salida	MAE *train*	MAE *validation*	MAE *test*	RMSE *train*	RMSE *validation*	RMSE *test*
Ux	0.2796	0.3028	0.3020	0.3710	0.4001	0.3998
Uy	0.2794	0.2985	0.2989	0.3697	0.3921	0.3929
Ux · Uy	0.1441	0.1552	0.1538	0.1966	0.2106	0.2099
Uz	0.1239	0.1324	0.1353	0.1909	0.2048	0.2098

Tabla 11. Comparación de los valores de las métricas MAE y RMSE obtenidas con el modelo final para cada set de datos.

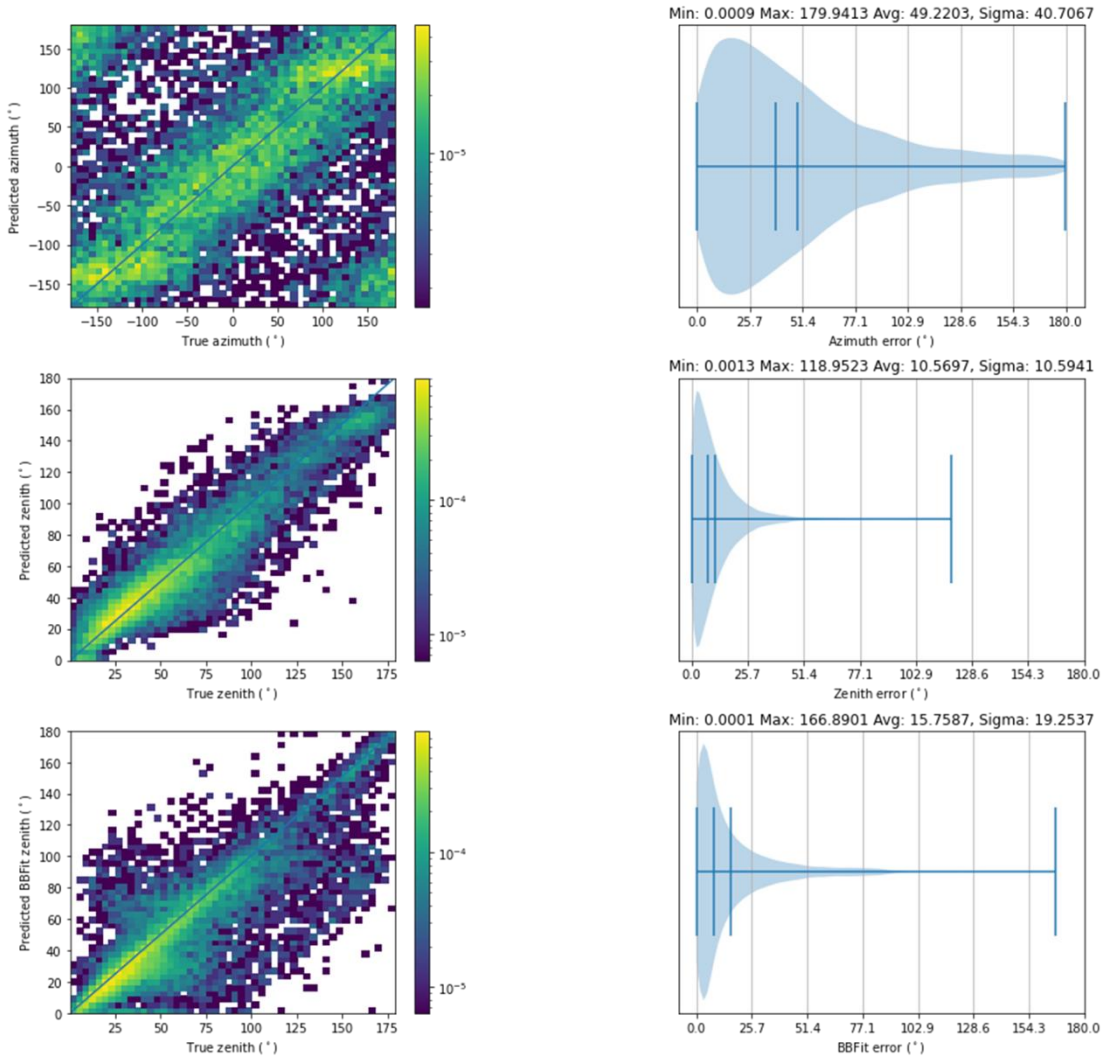

Fig. 14. De derecha a izquierda: histograma de densidad en dos dimensiones con escala de logarítmica de colores del ángulo predicho en función del valor real del ángulo, violin plot del error angular con la representación de la media y la mediana. De arriba a abajo: gráficas relativas al ángulo azimut del modelo, ángulo zenit del modelo y ángulo zenit del algoritmo BBFit. Estos resultados se han obtenido con el set de test.

Se observa que los valores de la métricas son similares entre sí. De todas formas, es posible observar que el MAE y el RMSE del set de *train* son menores que del resto de sets. Esto se puede explicar dado que para entrenar nuestro modelo hemos utilizado este set y en cierta medida, el modelo se ha ajustado para predecir valores de este set de datos.

Sin embargo, utilizar únicamente estas métricas para evaluar nuestro modelo y el algoritmo BBFit no nos aporta suficiente información respecto al rendimiento de las predicciones. Por esta razón evaluaremos el error angular por medio de los siguientes parámetros de fiabilidad:

- Chi2: el ajuste chi cuadrado utilizado por el algoritmo BBFit. Se corresponde a la variable "*Resconstruction quality"* del DF "bbfit *track data*".
- Ux_Uy – Ux *Uy: la diferencia entre la variable Ux·Uy predicha por el modelo y la multiplicación de Ux y Uy predichas.
- *Direction module*: el módulo del vector dirección creado a partir de las componentes predichas por el modelo. Nótese que se trata de un vector unitario.

En los siguientes subapartados se encuentran los resultados del error angular, considerado como la diferencia angular entre los valores predichos y los valores reales de azimut y zenit, respecto a los parámetros de fiabilidad además de una comparación de los errores absolutos de zenit y BBFit representados mediante un histograma en dos dimensiones.

6.1. Resultados respecto al parámetro de fiabilidad chi2

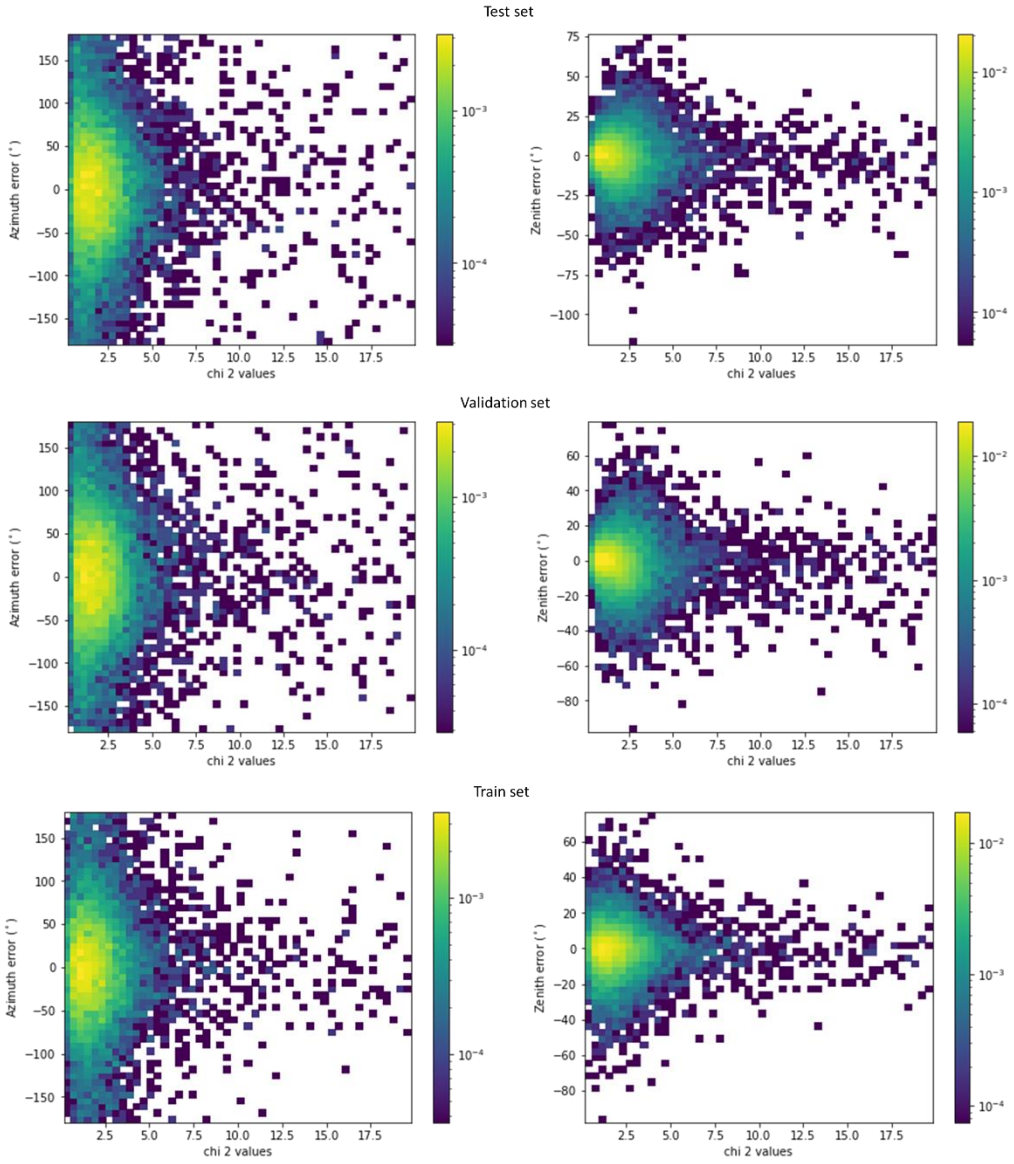

Fig. 15. Histogramas de densidad en dos dimensiones con escalado logarítmico. De derecha a izquierda: histograma del error angular de azimut respecto a los valores de chi2, histograma del error angular de zenit respecto a los valores de chi2. De arriba abajo: histogramas con el set de test, validation y train. Valores de chi2 superiores a 20 son considerados estadísticamente como outliers.

A nivel general, no se puede apreciar una gran diferencia entre los resultados de cada set de datos. Los resultados muestran una mayor densidad de errores cercanos a 0 para valores de chi2 entre 0 y 3 aproximadamente. Como era de esperar por los resultados de las métricas, en la figura 15 podemos observar que las predicciones del ángulo zenit son menores que las del ángulo azimut. Además, se observa una menor dispersión en el error a medida que aumenta chi2.

6.2. Resultados respecto al parámetro de fiabilidad Ux_Uy – Ux *Uy

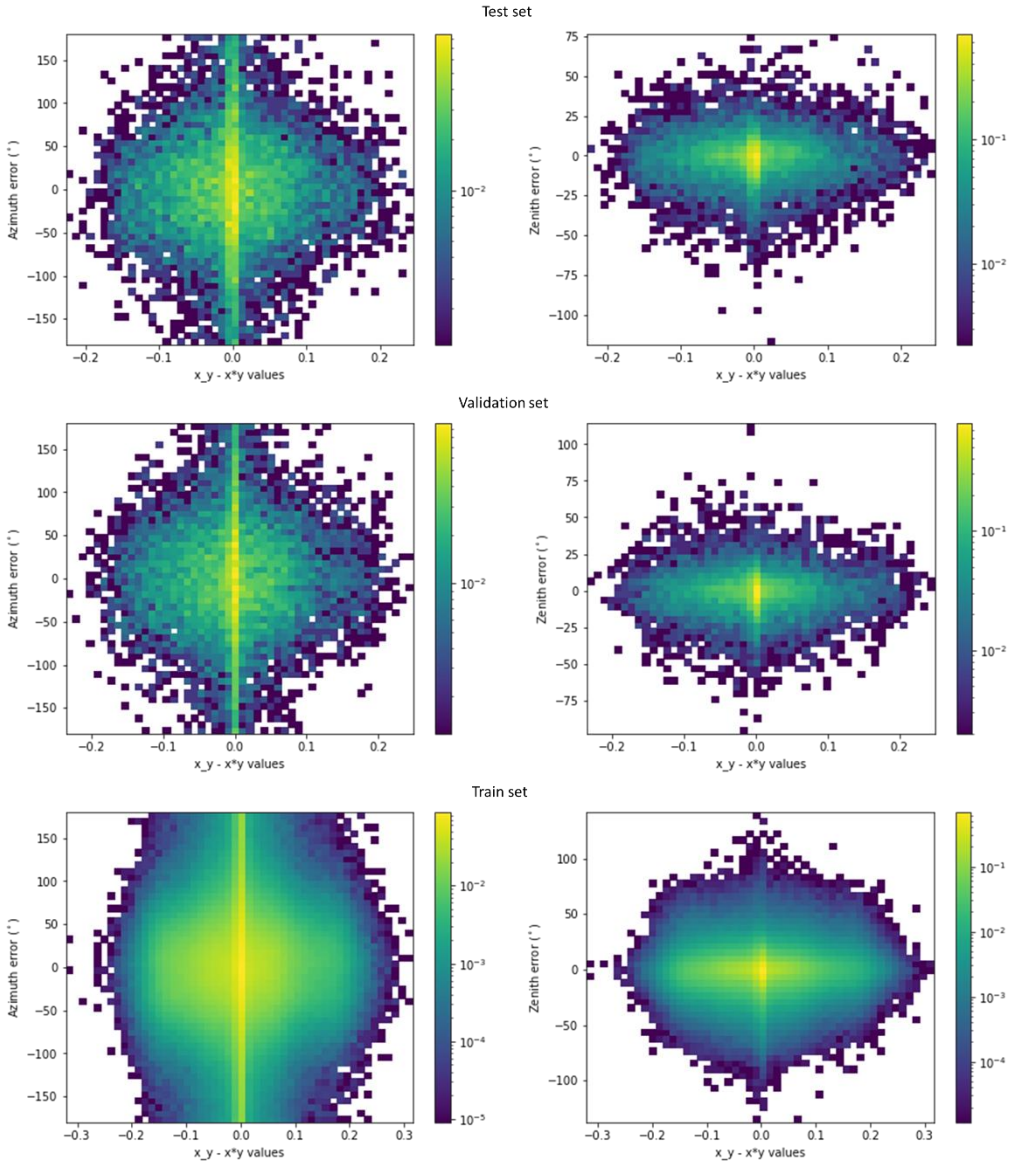

*Fig. 16. Histogramas de densidad en dos dimensiones con escalado logarítmico. De derecha a izquierda: histograma del error angular de azimut respecto a los valores de Ux_Uy – Ux*Uy, histograma del error angular de zenit respecto a los valores de Ux_Uy – Ux*Uy. De arriba abajo: histogramas con el set de test, validation y train.*

En el caso de la figura 16, la primera diferencia que podemos encontrar es que el set de *train* posee mayor resolución que los otros dos debido a que este set posee una mayor cantidad de eventos. Se comprueba que la zona con mayor densidad se encuentra donde se esperaba (alrededor del origen) pues, si nuestras predicciones son correctas, la diferencia entre la variable Ux·Uy y Ux*Uy deben de ser lo más cercano a 0. Nuevamente se comprueba que la predicción de Zenit tiene un mejor rendimiento que Azimut.

Por último, cabe comentar la clara línea vertical que encontramos para las predicciones de azimut. Dicha línea se obtiene dado que hay alrededor de un 26 % de predicciones cuyos valores de Ux·Uy, Ux y Uy son del orden de las centésimas e incluso milésimas para todos los sets. Por este motivo, hay una cantidad suficiente de eventos cuyo parámetro de seguridad Ux_Uy – Ux*Uy es muy cercano a 0.

6.3. Resultados respecto al parámetro de fiabilidad *Direction module*

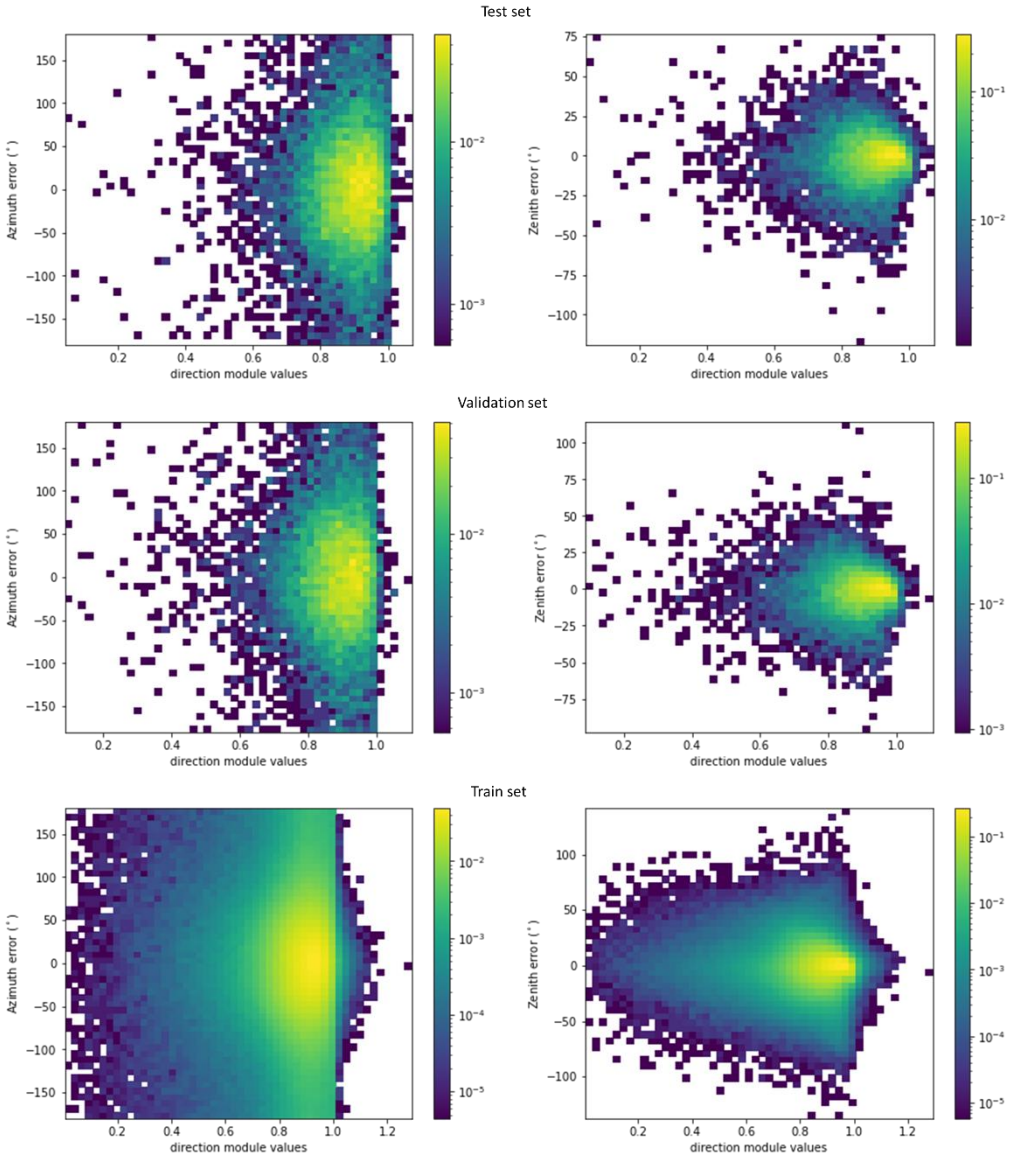

Fig. 17. Histogramas de densidad en dos dimensiones con escalado logarítmico. De derecha a izquierda: histograma del error angular de azimut respecto a los valores del módulo del vector dirección, histograma del error angular de zenit respecto a los valores del módulo del vector dirección. De arriba abajo: histogramas con el set de test, validation y train.

De la misma forma, en la figura 17, se observa que el set de *train* posee mayor cantidad de eventos en comparación a los otros sets. El objetivo de utilizar el módulo de la dirección, pues las componentes predichas son las componentes del vector unitario de la dirección, es corroborar si la densidad de los errores cercanos a 0 es mayor para valores del módulo de 1. En líneas generales se comprueba que se cumple el objetivo. De la misma forma se comprueba que las zonas más densas están menos dispersas para las predicciones de zenit, lo que nos indica un mejor rendimiento en estas predicciones.

6.4. Comparación del modelo y BBFit según los parámetros de fiabilidad chi2 y *Direction module*

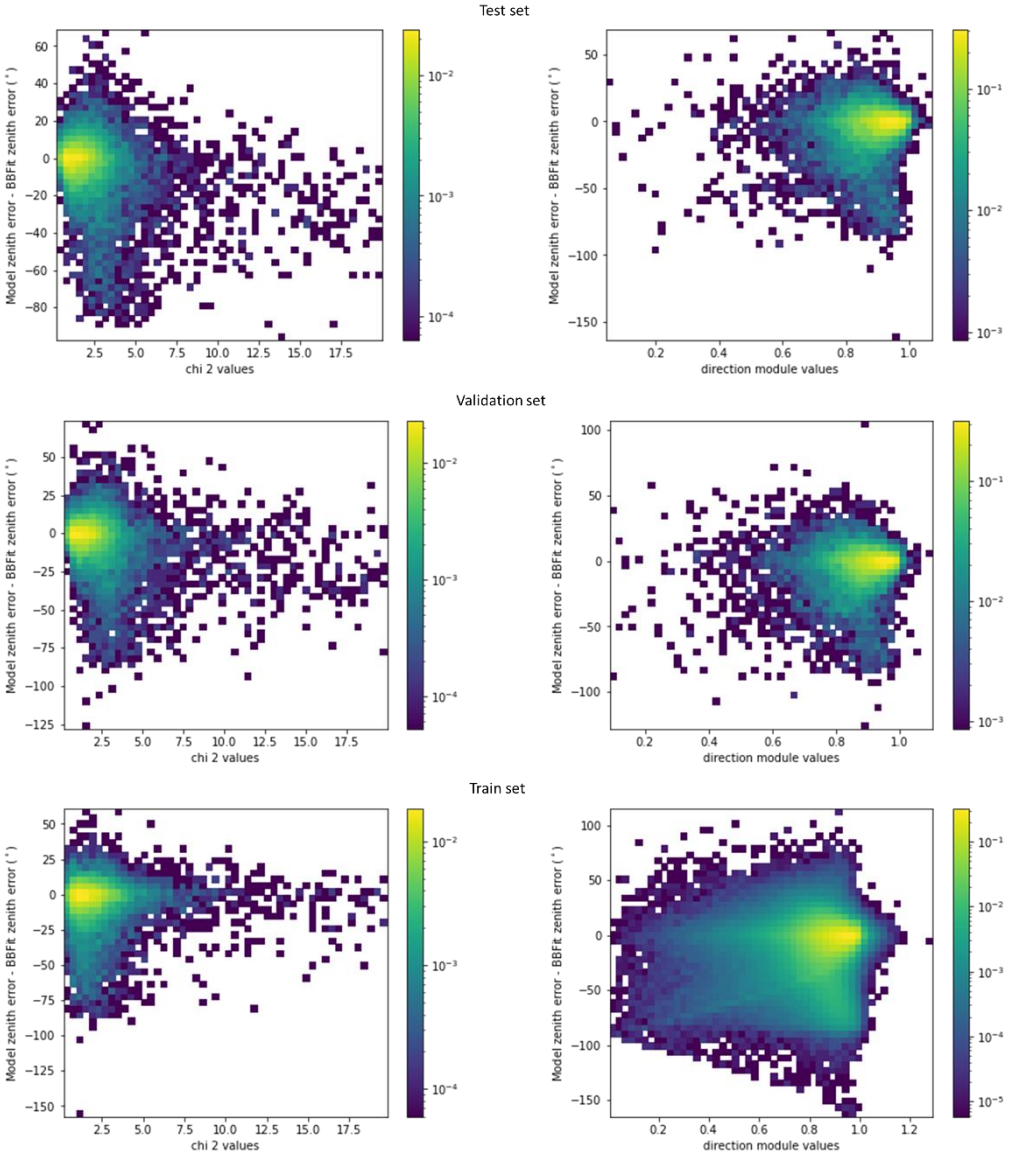

Fig. 18. Histogramas de densidad en dos dimensiones con escalado logarítmico. De derecha a izquierda: histograma de la diferencia absoluta entre el error angular de zenit y el error angular de BBFit respecto a los valores de chi2, histograma de la diferencia absoluta entre el error angular de zenit y el error angular de BBFit respecto a los valores del módulo del vector dirección. De arriba abajo: histogramas con el set de test, validation y train. Valores de chi2 superiores a 20 son considerados estadísticamente como outliers.

En la figura anterior, podemos observar una distribución de la diferencia similar a lo que se ha visto en los apartados anteriores. Lo que nos indica que ambos, nuestro modelo y el algoritmo BBFit, tienen un buen rendimiento en las predicciones de zenit, o lo que es lo mismo, de la componente Uz de la dirección. Sin embargo, se observa claramente una mayor distribución de densidad en los valores negativos de la diferencia, lo que nos indica que el error absoluto del zenit de BBFit es mayor que el error absoluto del zenit de nuestro modelo.

Más en detalle, se comprueba que las zonas más densas tienen una distribución que tiende a valores negativos como se observa en la figura 18.

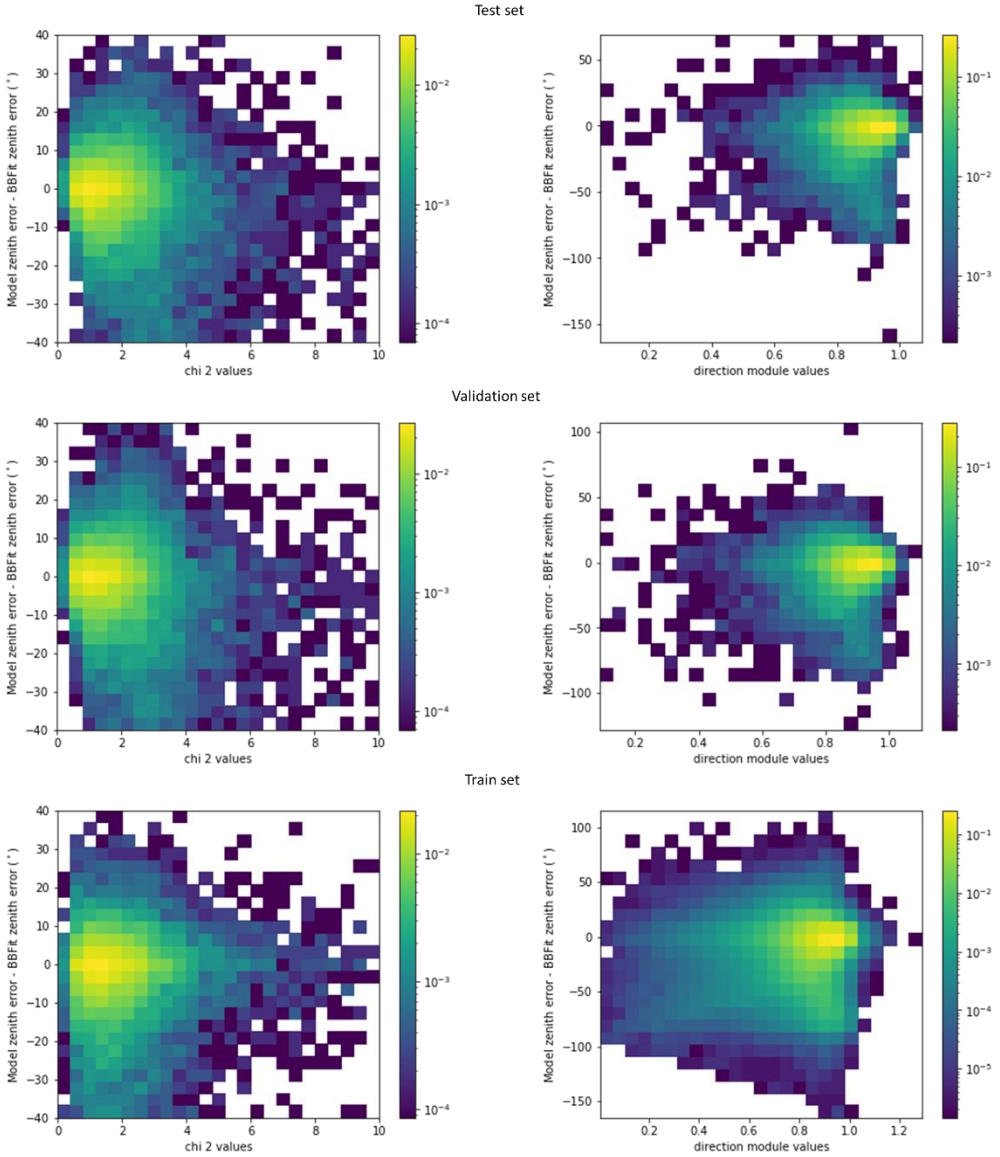

Fig. 19. Zoom de los histogramas de densidad en dos dimensiones con escalado logarítmico. De derecha a izquierda: histograma de la diferencia absoluta entre el error angular de zenit y el error angular de BBFit respecto a los valores de chi2, histograma de la diferencia absoluta entre el error angular de zenit y el error angular de BBFit respecto a los valores del módulo del vector dirección. De arriba abajo: histogramas con el set de test, validation y train.

7. Conclusiones y líneas futuras

Como hemos observado en el apartado anterior, el modelo de redes neuronales artificiales que hemos diseñado es capaz de predecir las componentes x e y de la dirección del neutrino con un MAE de 0.3 aproximadamente. Del mismo modo, también es capaz de predecir la componente Uz de la dirección, así como la multiplicación de las otras dos componentes con un MAE aproximado de 0.13 y 0.15 respectivamente. Estos datos en conjunto con el análisis realizado al error angular de las predicciones en función de los parámetros de fiabilidad chi2, Ux_Uy – Ux*Uy y *direction module* nos han permitido evaluar nuestro modelo con los diferentes sets de *test*, *validation* y *train* mostrando un modelo que ofrece un buen rendimiento en las predicciones.

En cuanto a las comparativas respecto al algoritmo BBFit se ha observado en primer lugar como nuestro modelo parece tener un mejor rendimiento gracias a las métricas MAE y RMSE. Incluso, en la comparación de nuestro modelo y el algoritmo BBFit en función de los parámetros de fiabilidad chi2 y *direction* module se detecta que el modelo diseñado en este proyecto posee un menor error en zenit por lo que se puede concluir que el modelo diseñado ha superado al algoritmo BBFit en la predicción de zenit además de proporcionar predicciones de azimut que BBFit no es capaz de generar en eventos de una sola línea.

En definitiva, se ha comprobado como un modelo de redes neuronales artificiales para predecir la trayectoria de neutrinos en eventos de una sola línea ha mejorado considerablemente los resultados del algoritmo BBFit y además ofrece un rendimiento aceptable para todas las salidas, como se ha observado en los histogramas de dos dimensiones y *violin plots* a lo largo del apartado anterior, mejorando completamente el estado del arte.

Sin embargo, se ha de tener en consideración que estos resultados podrían mejorar con un equipo con mejores prestaciones y orientado al desarrollo de redes neuronales artificiales. Un equipo mejor nos permitiría optimizar el ajuste de los hiperparámetros con un enfoque diferente al utilizado en este proyecto donde solo se trabajaba con un modelo a la vez (*babysitting*).

Por lo tanto, en líneas futuras sería posible encontrar un modelo aún más optimo y con mejores predicciones utilizando un *split* del set de datos más equitativo combinado con la técnica de *k-fold cross validation*. Se podría incluso utilizar la información proporcionada por el algoritmo AAFit como una entrada adicional a nuestro modelo y de esta forma unificar ambos algoritmos en uno mismo pudiendo así hacer predicciones tanto de eventos de una sola línea como de eventos multilínea. También sería interesante utilizar la desviación angular de la dirección predicha respecto a la dirección real mediante el producto escalar como función de coste unificando el modelo en un único error predictivo. Finalmente, se procedería a la implementación del algoritmo diseñado en las instalaciones del telescopio submarino de neutrinos ANTARES.

Anexo

En este apartado se muestran otras representaciones que añaden información adicional a la justificación de la solución adoptada.

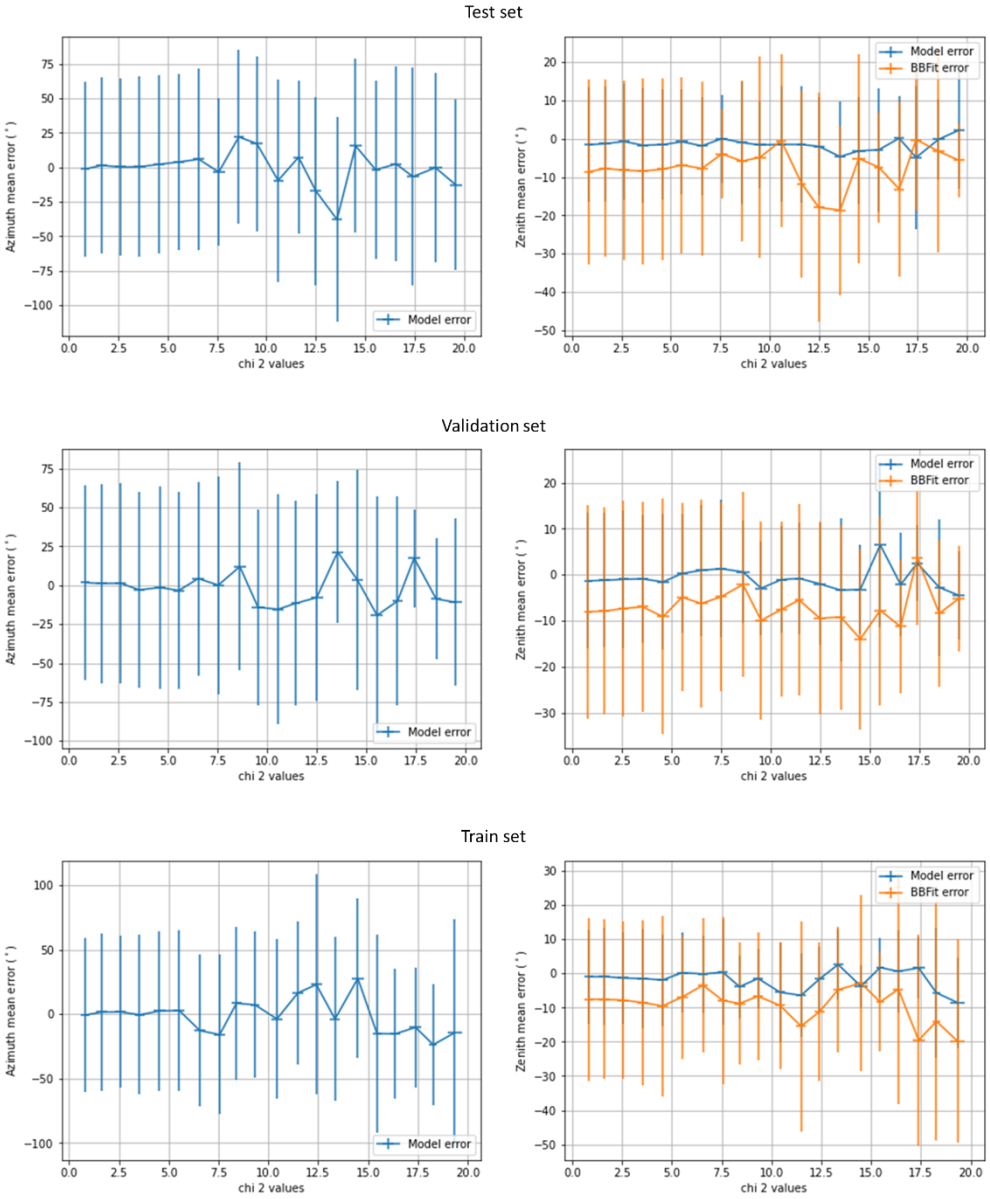

Fig. 20. De derecha a izquierda: gráfica del error medio angular de azimut respecto a los valores de chi2, gráfica del error medio angular de zenit respecto a los valores de chi2. De arriba abajo: gráficas realizadas con el set de test, validation y train. Valores de chi2 superiores a 20 son considerados estadísticamente como outliers. Las barras de error representan la desviación estándar de cada muestra.

Los resultados de la figura 20 nos muestran principalmente como el error medio es próximo a 0 para valores de chi2 pequeños. También se observa como el error medio del BBFit es peor que el de nuestro modelo.

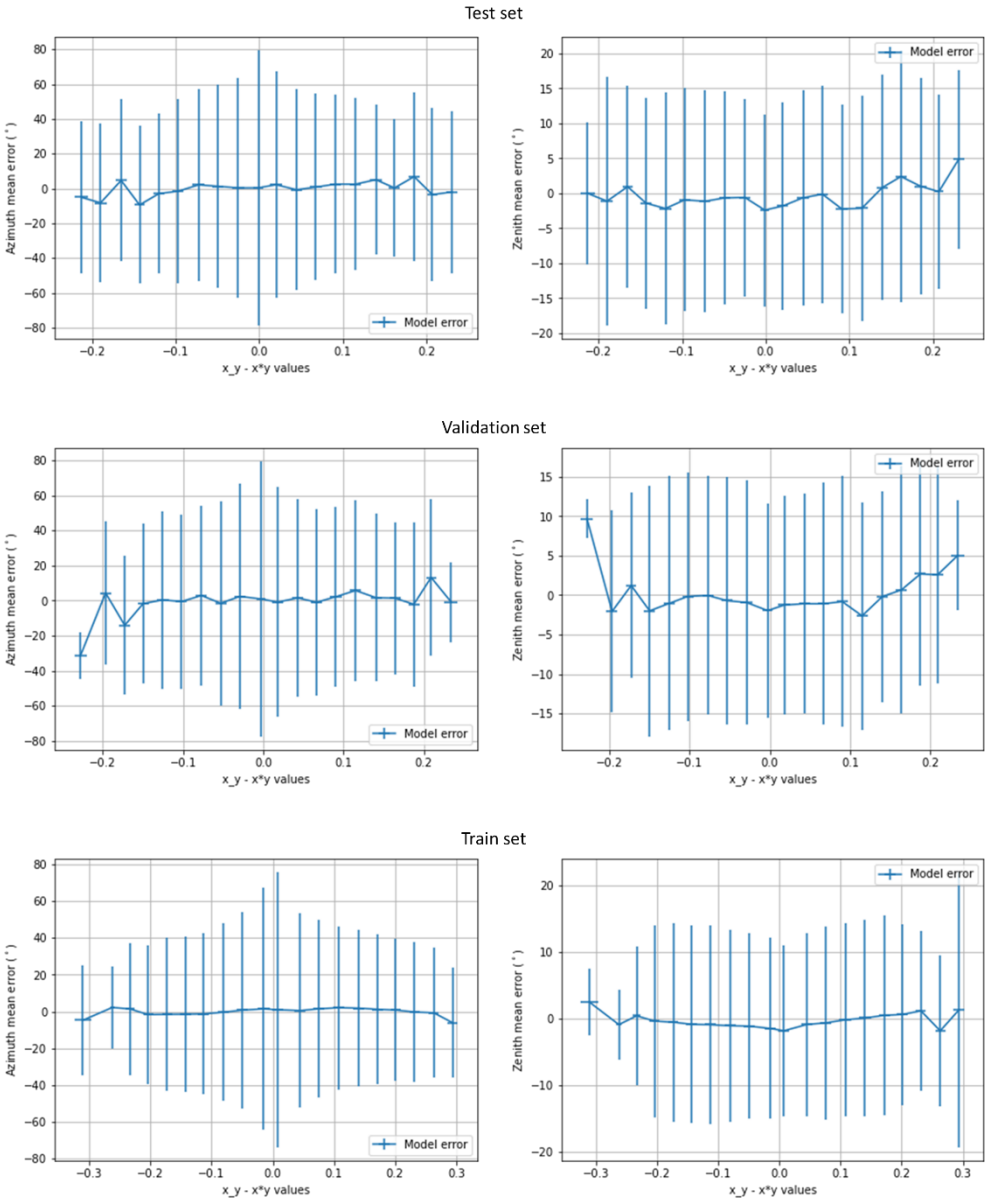

*Fig. 21. De derecha a izquierda: gráfica del error medio angular de azimut respecto a los valores de Ux_Uy – Ux*Uy, gráfica del error medio angular de zenit respecto a los valores de Ux_Uy – Ux*Uy. De arriba abajo: gráficas con el set de test, validation y train. Las barras de error representan la desviación estándar de cada muestra.*

Del mismo modo, en la figura 21 se observa, para todos los sets de datos, que el error medio toma valores alrededor de 0 cuando los valores del parámetro de fiabilidad Ux_Uy-Ux*Uy tienden a 0. Sin embargo, se observa una traza más estable para el set de *train*. Esto es debido a que es el set de datos que se ha utilizado en el entrenamiento del modelo y por lo tanto el modelo se ha ido ajustando a este set de datos siendo más fácil que “recuerde” los valores reales.

Nuevamente, en la siguiente figura, se observa los resultados esperados. El error es próximo a 0 para valores del módulo iguales a 1. Del mismo modo se puede notar que el set de *train* posee mejores resultados por el motivo anteriormente explicado. Además, también se comprueba que el error medio del modelo es mejor que el error medio del algoritmo BBFit.

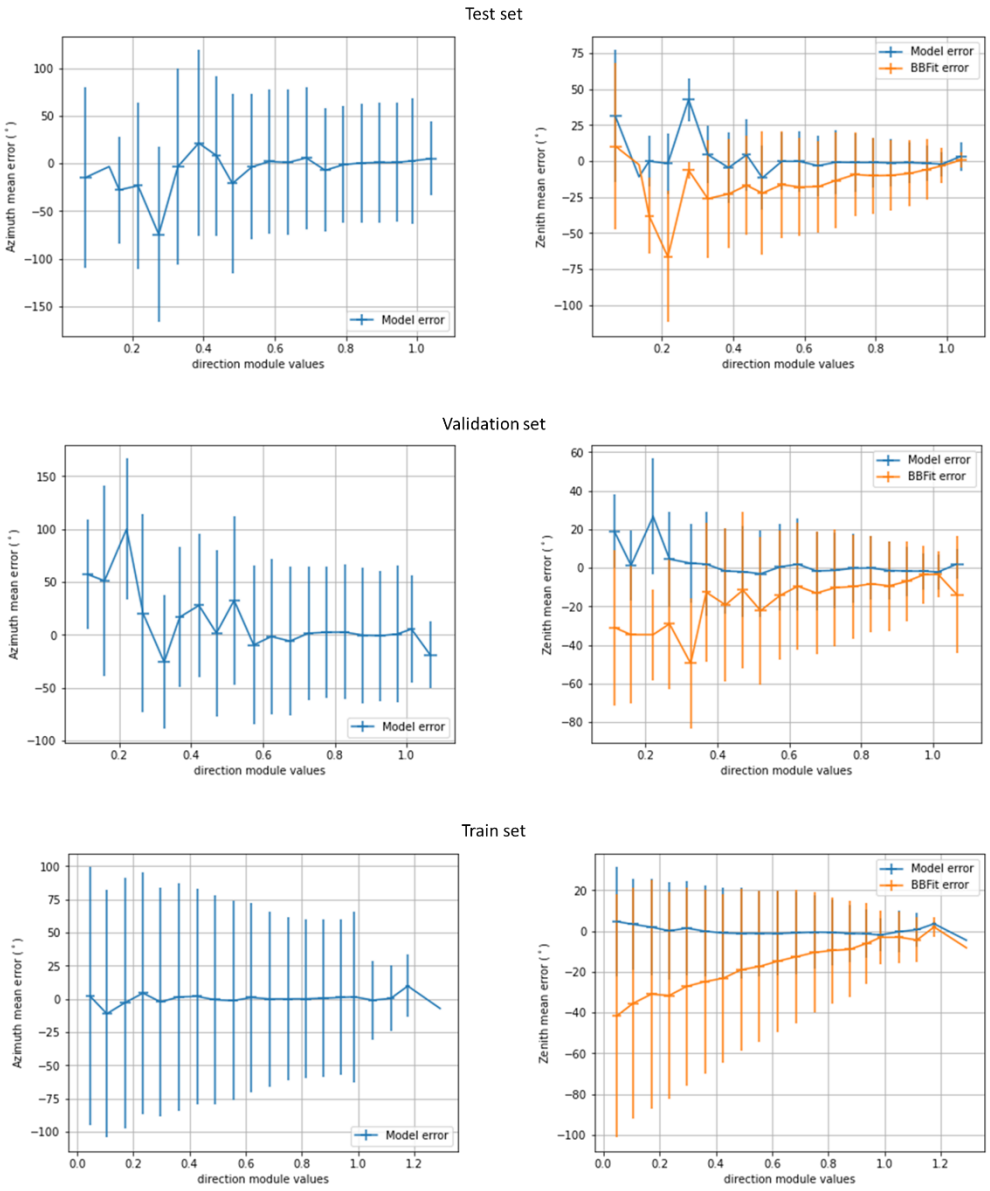

Fig. 22.. De derecha a izquierda: gráfica del error medio angular de azimut respecto a los valores del módulo del vector dirección, gráfica del error medio angular de zenit respecto a los valores del módulo del vector dirección. De arriba abajo: gráficas con el set de test, validation y train. Las barras de error representan la desviación estándar de cada muestra.

Por último, se representa, en la figura 23, la distribución de la desviación angular entre la dirección predicha y la dirección real para cada set de datos. Se remarca que la desviación promedio es de alrededor de 31º demostrando que las predicciones son de consideración.

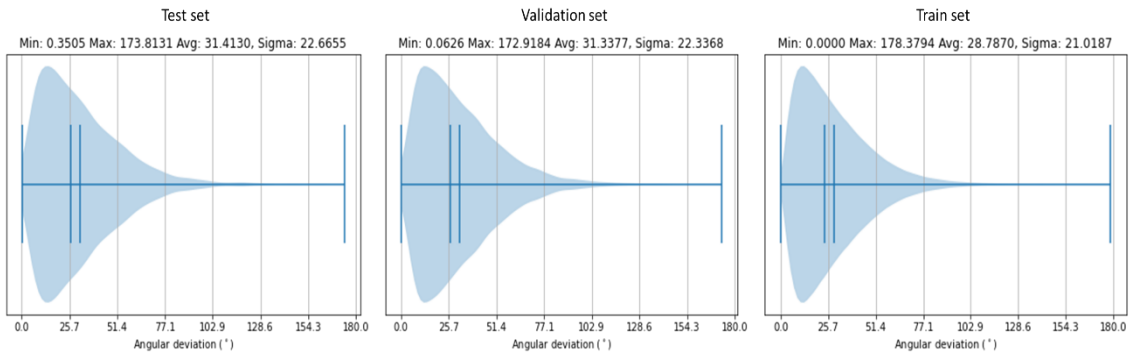

Fig. 23. Violin plots de la desviación angular entre la dirección predicha y la dirección real para el set de test, validation y train respectivamente.

DOCUMENTO II. Pliego de condiciones

1. Definición y alcance (objeto)

El objeto de este documento consiste en fijar las condiciones técnicas mínimas que se deben cumplir para llevar a cabo el proyecto de implementación del algoritmo de redes neuronales artificiales para la reconstrucción de trayectorias de neutrinos descrito en el "documento **I** Memoria", especificando los requisitos de fiabilidad y seguridad necesarios para el correcto desarrollo y explotación de dicho proyecto.

El ámbito de aplicación de este documento se centra únicamente en todos los sistemas informáticos utilizados para el desarrollo de la investigación, así como en el entorno de trabajo, sin tener en cuenta los sistemas eléctricos, mecánicos y electrónicos que forman parte de las instalaciones del centro de investigación de astronomía de neutrinos ANTARES. Se aceptarán soluciones diferentes a las exigidas en este documento siempre que su necesidad este justificada además de no implicar una reducción en la calidad mínima exigida.

2. Condiciones generales

En la directiva del consejo de la unión europea 90/270/ CEE, de 29 de mayo de 1990 se enumeran las disposiciones mínimas de seguridad y salud relativas al trabajo con equipos que incluyen pantallas de visualización.

Según la dicha directiva, se define como puesto de trabajo: "*el conjunto que consta de un equipo con pantalla de visualización provisto, en su caso, de un teclado o de un dispositivo de adquisición de datos y/o de un programa que garantice la interconexión hombre/máquina, de accesorios opcionales, de anejos, incluida la unidad de disquetes, de un teléfono, de un módem, de una impresora, de un soporte de documentos, de una silla y de una mesa o superficie de trabajo, así como el entorno laboral inmediato*. [8].", siendo una pantalla de visualización cualquier "*pantalla alfanumérica o gráfica, independientemente del método de representación visual utilizado.* [8].".

A continuación, se enumeran las disposiciones mínimas requeridas relativas al equipo, entorno e interconexión ordenador/hombre extraídos de [8].

2.1. Equipo

- Pantalla: La imagen en la pantalla deberá ser estable, sin reflejos ni reverberaciones que molesten al usuario; con los caracteres bien definidos, dimensión suficiente y espaciado adecuado entre caracteres y renglones. El usuario debe poder ajustar con facilidad la luminosidad y/o contraste de la pantalla, así como la orientación e inclinación.
- Teclado: Deberá ser inclinable e independiente de la pantalla para facilitar una postura cómoda que no provoque cansancio en brazos y manos. Los símbolos de las teclas deberán ser legibles desde la posición normal de trabajo. Tendrá que haber espacio suficiente delante del teclado para que el usuario pueda apoyar brazos y manos
- Mesa o superficie de trabajo: Tendrá una superficie poco reflectable, con dimensiones suficientes para permitir una colocación flexible de pantalla, teclado y material accesorio, así como para permitir una postura cómoda del trabajador.
- Asiento de trabajo: Asiento estable, que proporcione libertad de movimiento y postura confortable. La altura ha de ser ajustable con un respaldo inclinable. Se pondrá un reposapiés a quienes lo deseen.

2.2. Entorno

- Espacio: El puesto de trabajo deberá tener una dimensión suficiente y estar acondicionado para permitir cambiar de postura y de movimientos de trabajo.
- Iluminación: La iluminación general y la especial (lámparas de trabajo) deberán garantizar una luz suficiente y un contraste adecuado entre pantalla y entorno. Las fuentes de luz deben evitar deslumbramientos y reflejos molestos en el equipo.
- Reflejos y deslumbramientos: Fuentes de luz como ventanas y otras aberturas no deben provocar deslumbramiento directo y generen un mínimo de reflejos en pantalla. Las ventanas se equiparán con un dispositivo de cobertura adecuado y regulable para atenuar la luz que ilumine el entorno de trabajo.
- Ruido. El ruido producido tanto por el equipo y/o entorno de trabajo no debe perturbar la palabra ni la atención.
- Calor: Los equipos utilizados no deberán producir calor adicional que pueda ocasionar molestias a los trabajadores
- Emisiones: A excepción de las radiaciones del espectro electromagnético visible, deberán reducirse a niveles insignificantes para la protección de la seguridad y de la salud de los trabajadores.
- Humedad: La humedad se creará y mantendrá a niveles aceptables.

2.3. Interconexión ordenador/hombre

- Los sistemas deberán proporcionar a los trabajadores indicaciones sobre su desarrollo.
- Los sistemas deberán mostrar la información en un formato y ritmo adaptado a los operadores.
- Los principios de ergonomía deberán aplicarse en particular al tratamiento de la información por parte del hombre.

3. Condiciones especificas

En el presente documento solo se enumerarán las condiciones específicas de índole técnico, sin considerar las posibles condiciones facultativas, económicas y/o legales. Las siguientes especificaciones técnicas se refieren tanto a el dispositivo de computación (*hardware*) como a los programas que se utilizarán (*software*) durante el desarrollo y explotación del proyecto.

3.1. Hardware

Las características técnicas del equipo *hardware* con el que se trabajará durante el proyecto consistirá en una computadora que contenga las diferentes características:

- El procesamiento gráfico del computador debe estar equipado con la unidad gráfica de procesamiento (GPU) NVIDIA® TITAN RTX™. En caso de adoptarse una solución diferente, se deberá consultar con la figura facultativa correspondiente para asegurar que no haya una reducción en la calidad. La solución alternativa ha de ser de la gama NVIDIA® y no deberá tener una capacidad de computación inferior a 7.5. Además, la memoria no puede ser inferior de 12 GB.
- Las especificaciones de almacenamiento consisten en una memoria RAM de 16 GB o superior, con un disco duro o SSD principal con una capacidad de almacenamiento 120 GB o más junto con una unidad SSD extra de 1 TB.

- La unidad central de procesamiento (CPU) será un Intel Core i5-10600K 4.10 GHz. Además, será necesario incluir un ventilador propio para refrigerar el dispositivo CPU. Cualquier solución alternativa deberá tener la arquitectura INTEL® Core™ de 64 bits perteneciente a la familia i5 o superior con una frecuencia similar. La generación de la CPU será la mayor existente o la inmediatamente inferior.
- La alimentación deberá ofrecer una potencia mínima de 650 W.
- El sistema operativo de nuestra computadora será Windows 10 Home de 64 bits.

3.2. Software

Además de los *Notebooks* con la implementación del algoritmo desarrollado en el documento I, que se entregará mediante un repositorio en GitHub, será necesario el uso de software específico. En la siguiente tabla encontraremos los programas de software y librerías que se necesitarán para la realización del proyecto.

Nombre	Versión	Función
Entorno Anaconda	Última versión	Esta plataforma nos permitirá usar la aplicación de Jupyter Notebook para editar el código fuente. También, nos permitirá instalar las librerías necesarias fácilmente usando el *Promt Shell*
NumPy	Última versión	Librería con orientación vectorial con herramientas de computación numérica.
Pandas	Última versión	Librería de administración de datos mediante la creación de *DataFrames*
TensorFlow	Última versión estable	*Framework* de desarrollo de nuestro modelo de red neuronal artificial
CUDA® *Toolkit*	10.1 (Tensorflow >= 2.1.0)	Kit de herramientas de la librería CUDA
GPU NVIDIA® *Drivers*	418.X o posterior (*Toolkit* = 10.1)	Drivers necesarios para la utilización de nuestra GPU
NVIDIA CUDA® *Deep Neural Network library* (cuDNN)	7.6 o posterior	Librería para desarrollar redes neuronales con el *framework* seleccionado

Tabla 12. Enumeración de las diferentes aplicaciones y librerías específicas para el desarrollo del proyecto.

La instalación del *software* específico del proyecto se realizará siguiendo la enumeración de la tabla anterior. De esta forma, para la instalación del entorno Anaconda, nos dirigiremos a la página web de Anaconda (https://www.anaconda.com/products/individual). Una vez ahí, nos dirigimos a la sección “Anaconda *Installers*” y seleccionamos el instalador “64-Bit Graphical Installer” cuya versión de Python sea el mayor. Una vez realizada la descarga se abrirá el archivo que ejecutará el *wizard installer* de Windows. Por último, se seguirá la instalación con la configuración por defecto.

Una vez instalado el entorno Anaconda, nos dirigimos al buscador de Windows y escribimos “Anaconda Prompt”. Hacemos clic en la primera opción para abrir un terminal de comandos donde escribiremos el comando “conda install -c numpy”. Posteriormente de ejecutar el comando, deberemos escribir “y” para permitir la instalación. Realizar el mismo procedimiento para la librería pandas con el comando “conda install -c pandas”.

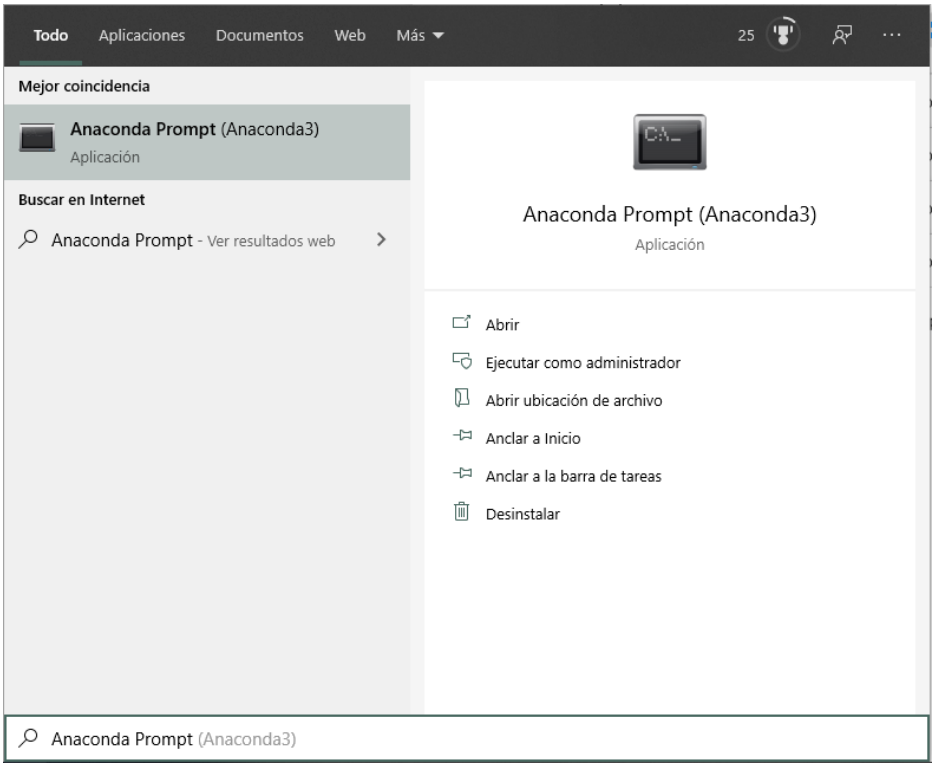

Fig. 24. Recorte del buscador de Windows para ejecutar la consola de comandos de Anaconda.

A continuación, para instalar la última versión estable del *framework* TensorFlow utilizaremos el comando "pip" de PyPI. Para ello, escribiremos en el terminal el comando "conda install pip". Posteriormente, para asegurarnos que usamos la última versión de "pip" ejecutaremos el comando "pip install --upgrade pip". Finalmente seguiremos el mismo procedimiento de instalación de las dos librerías anteriores usando esta vez el comando "pip install tensorflow".

```
>conda install -c numpy
>conda install -c pandas
>conda install pip
>pip install --upgrade pip
>pip install tensorflow
```

Fig. 25. Resumen de los comandos a utilizar para instalar las últimas versiones de las librerías de numpy, pandas y TensorFlow.

Para acabar el proceso de instalación del *software*, instalaremos las librerías y programas necesarios para la aceleración mediante el uso de la GPU. Tanto el procedimiento como las compatibilidades entre versiones se extraen de la página web de TensorFlow (https://www.tensorflow.org/install/gpu?hl=es-419) cuyo acceso es recomendable para revisar las actuales compatibilidades. En la tabla 8 podemos encontrar las versiones compatibles que se han usado durante la redacción de este documento.

Para la instalación de CUDA® *Toolkit* nos dirigimos al enlace https://developer.nvidia.com/cuda-toolkit-archive. A continuación, hacemos clic en "CUDA Toolkit 10.1 update2 (Aug 2019), Versioned Online Documentation" y seleccionamos el sistema operativo Windows con una arquitectura x86_64, versión 10 y tipo de instalador "exe (local)". Por último, se hará clic en "Download", se ejecutará, y se instalará siguiendo la configuración por defecto del *wizard installer*.

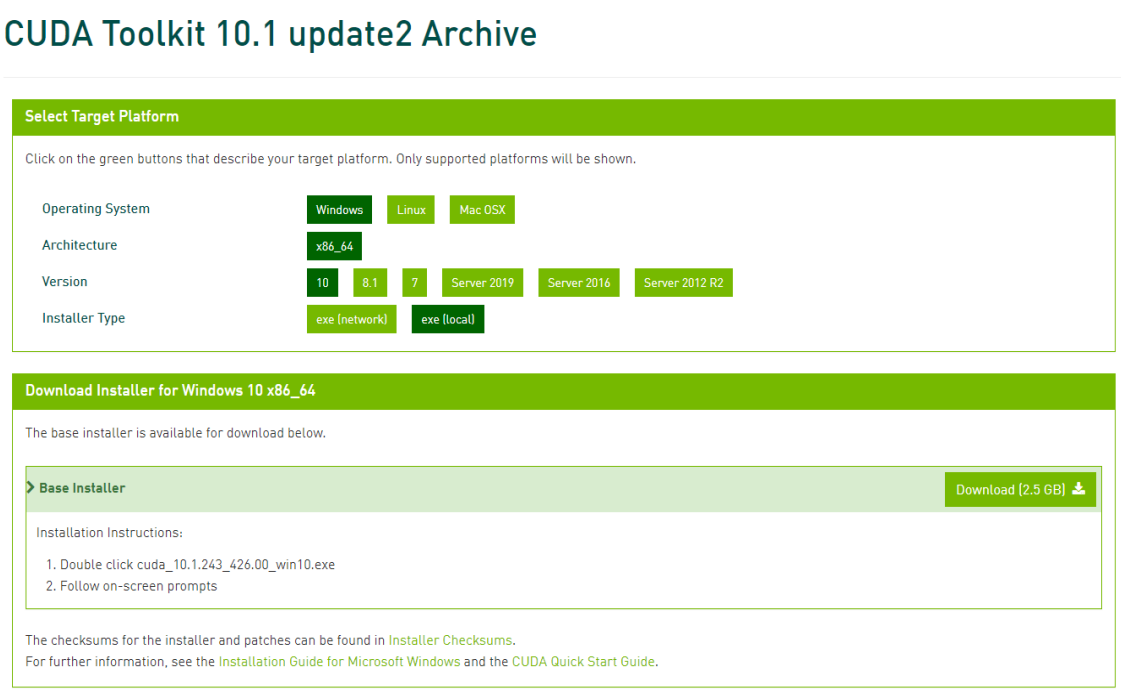

Fig. 26. Recorte de la configuración de descarga de la versión de CUDA® Toolkit compatible con TensorFlow.

Una vez instalado, se abrirá la ventana "GeForce *Experience*" en la que nos pedirá que nos registremos y/o iniciemos sesión. A continuación, se hará clic en la pestaña de "CONTROLADORES" y en la sección de "disponible" encontraremos la última versión disponible de los *drivers* de nuestra GPU. Hacemos clic en "DESCARGAR" y esperamos a que finalice la descarga e instalación de los drivers.

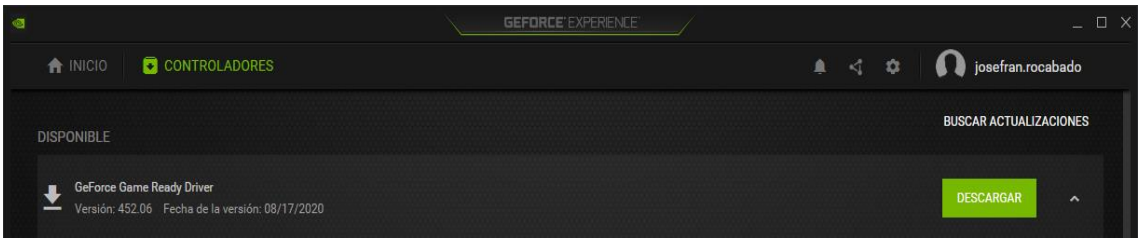

Fig. 27. Recorte de la localización de la sección para actualizar los drivers de NVIDIA en la ventana GeForce Experience

A continuación, instalaremos la librería cuDNN. Para ello nos dirigiremos a la web https://developer.nvidia.com/cudnn y haremos clic en "*Download* cuDNN". Posteriormente, para realizar la descarga tendremos que registrarnos y/o iniciar sesión en el NVIDIA *Developer Program*. Si es la primera vez que se registra, deberá rellenar un formulario. Posteriormente, se marcará la casilla de "*I Agree To the Terms of the cuDNN Software License Agreement*" y se seleccionará la última versión compatible con CUDA 10.1. Posteriormente, extraemos la carpeta "cuda" en el directorio "C:\tools\" y procederemos a reajustar las direcciones de acceso a las dependendicias de las librerías CUDA y cuDNN. Para ello necesitamos escribir los siguientes comandos en la consola de Anaconda:

- "cd \"
- "SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\bin;%PATH%"
- "SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\extras\CUPTI\lib64;%PATH%"
- "SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\include;%PATH%"

- SET PATH=C:\tools\cuda\bin;%PATH%

Notesé que, si hemos seguido los pasos anteriores, los directorios serán los mismos que los que aparecen en los comandos anteriores. De otra forma, habrá que modificar los comandos para hacer coincidir la dirección con la ubicación de las carpetas “bin”, “lib64”, “include” del directorio creado en la instalación de CUDA® *Toolkit* y “bin” de la líbreria cuDNN.

Por último, se deberá revisar que la compatibilidad de las versiones entre TensorFlow, CUDA® *Toolkit,* GPU NVIDIA® *Drivers* y cuDNN es correcta. En caso de no ser la adecuada, se deberá de repetir el procedimiento anterior para las versiones correspondientes.

DOCUMENTO III. Presupuesto

1. Introducción

En el presente documento se redactan los diferentes costes que se llevaran a cabo en el desarrollo del proyecto. Para los precios de los materiales se utilizan valores estimados y por lo tanto los costes podrían variar. Además, para dar un presupuesto más detallado se decide utilizar materiales que cumplen con las exigencias mínimas descritas en el "documento II Pliego de condiciones". Es por ello por lo que el ordenador de trabajo se customiza según estas exigencias.

2. Coste *hardware*

Unidad	Denominación	Cantidad	Precio	Total
Ensamblaje ordenador de torre				
Ensamblaje del dispositivo *hardware* customizado para desarrollar e implementar proyectos basados en redes neuronales artificiales.				
U		1	3,142.73 €	3,142.73 €
Materiales				
U	Placa base Asus ROG STRIX H470-I GAMING	1	144.55 €	144.55 €
U	CPU Intel Core i5-10600k 4.10 GHz	1	226.40 €	226.40 €
U	Memoria RAM de 16 GB	1	54.31 €	54.31 €
U	Memoria principal de 128 GB	1	39.49 €	39.49 €
U	Memoria secundaria de 1 TB	1	101.64 €	101.64 €
U	refrigeración CPU	1	49.57 €	49.57 €
U	GPU Nvidia Titan RTX 24GB GDDR6	1	2,271.90 €	2,271.90 €
U	Carcasa del ordenador con ventilación por convección forzada	1	41.31 €	41.31 €
U	Alimentación 650 W	1	51.24 €	51.24 €
U	Monitor	1	63.63 €	63.63 €
U	Kit ratón y teclado inalámbrico	1	23.13 €	23.13 €
Mano de obra				
h	Técnico informático	1.5	9.29 €	13.94 €
Costes directos complementarios				
%		2%	3,081.11 €	61.62 €

Tabla 13. Presupuesto de la partida del ensamblaje del ordenador de torre desglosado en materiales, mano de obra y costes directos complementarios (2%).

3. Coste *software*

Unidad	Denominación	Cantidad	Precio	Total
Instalación y configuración del *software*				
Instalación de los programas y librerías específicas en el ordenador de torre.				
U		1	116.40 €	116.40 €
Materiales				
U	Microsoft Windows 10 Home 64 bits	1	90.89 €	90.89 €
U	Entorno Anaconda	1	- €	- €
U	NumPy	1	- €	- €
U	Pandas	1	- €	- €
U	TensorFlow	1	- €	- €
U	CUDA® Toolkit	1	- €	- €
U	GPU NVIDIA® Drivers	1	- €	- €
U	cuDNN	1	- €	- €
Mano de obra				
h	Técnico informático	2.5	9.29 €	23.23 €
Costes directos complementarios				
%		2%	114.12 €	2.28 €

Tabla 14. Presupuesto de la partida de la instalación del software específico en el ordenador de torre desglosado en materiales, mano de obra y costes directos complementarios (2%).

4. Coste estudio de antecedentes

Unidad	Denominación	Cantidad	Precio	Total
Estudio de antecedentes				
Estudio de antecedentes al proyecto a desarrollar y estado del arte de la tecnología utilizada.				
U		1	2,371.20 €	2,371.20 €
Mano de obra				
h	Científico de datos	120	19.00 €	2,280.00 €
Costes directos complementarios				
%		4%	2,280.00 €	91.20 €

Tabla 15. Presupuesto de la partida del estudio de antecedentes desglosado en mano de obra y costes directos complementarios (4%). Se asume que el estudio tendrá una duración aproximada de 3 semanas.

5. Coste del desarrollo del proyecto

Unidad	Denominación	Cantidad	Precio	Total
Desarrollo del proyecto				
Presupuesto del desarrollo del proyecto de implementación, mejora del algoritmo de reconstrucción de trayectorias de neutrinos e implementación.				
U		1	3,744.00 €	3,744.00 €
Mano de obra				
h	Ingeniero de datos	240	15.00 €	3,600.00 €
Costes directos complementarios				
%		4%	3,600.00 €	144.00 €

Tabla 16. Presupuesto de la partida del desarrollo del proyecto desglosado en mano de obra y costes directos complementarios (4%). Se asume que el proyecto tendrá una duración aproximada de 6 semanas.

6. Resumen del presupuesto

Capítulo	Importe
Capítulo 1. Ensamblaje de ordenador de torre	3,142.73 €
Capítulo 2. Instalación y configuración del software	116.40 €
Capítulo 3. Estudio de antecedentes	2,371.20 €
Capítulo 4. Desarrollo del proyecto	3,744.00 €
TOTAL PRESUPUESTO DE EJECUCIÓN MATERIAL	9,374.33 €
medios auxiliares (4%)	374.97 €
TOTAL PRESUPUESTO DE EJECUCIÓN POR CONTRATA	9,749.30 €
IVA (21%)	2,047.35 €
TOTAL PRESUPUESTO BASE DE LICITACIÓN	11,796.66 €

Tabla 17. Resumen de las partidas del presupuesto y obtención del total de presupuesto base de licitación

Por lo tanto, el presupuesto base de licitación asciende a un total de once mil setecientos noventa y seis euros con sesenta y seis céntimos.

Valencia, 14 de septiembre de 2020

Ingeniero técnico industrial:

Rocabado Rocha, Jose Luis

DOCUMENTO IV. Bibliografía

[1] “Why neutrino astronomy?” https://antares.in2p3.fr/Overview/why.html (accessed Jul. 10, 2020).

[2] M. Ageron *et al.*, “ANTARES: The first undersea neutrino telescope,” *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 656, no. 1, pp. 11–38, 2011, doi: 10.1016/j.nima.2011.06.103.

[3] “Detector design.” https://antares.in2p3.fr/Overview/detector.html (accessed Jul. 24, 2020).

[4] M. Ardid, “Dark Matter Searches with the ANTARES Neutrino Telescope,” 2016, doi: 10.1016/j.nuclphysbps.2015.09.054.

[5]

[6] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, “A survey on multi-output regression.”

[7] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” 2015.

[8] Consejo de la unión Europea, *Directiva 90/270/ CEE (1990, Mayo 29)*. [en línea]. Disponible: https://eur-lex.europa.eu/legalcontent/ES/ALL/?uri=celex:31990L0270