Memoria de prácticas externas

Carlos Lamas Rodríguez

Julio 2021

Resumen

El objetivo de las prácticas es la formación en técnicas de análisis e identificación de rayos cósmicos primarios. Durante el desarrollo de las mismas se utilizan técnicas de redes neuronales y big data, en concreto, se utiliza la librería Pandas de python para el análisis de datos y Keras, que se ejecuta sobre Tensorflow, para la implementación de la red neuronal. El programa desarrollado aparece recogido en el anexo del final de la memoria.

1. Introducción

El proyecto a realizar en estas prácticas consiste en la elaboración de un programa que permita identificar la energía y el tipo de núcleo de un rayo cósmico primario a partir de los datos medidos a la altura del suelo. El objetivo final es poder aplicar este programa a las partículas captadas por un detector tipo Trasgo, y en particular al TRAGALBAGAS, detector elaborado en el Laboratorio Carmen Fernández de la Universidad de Santiago de Compostela.

Los detectores tipo Trasgo son detectores de rayos cósmicos basados en la tecnología tRCPs (timing Resistive Chambers Plate) que permiten realizar un tracking de las partículas cargadas que llegan a este con una resolución temporal del orden de ns. Configurados correctamente, este tipo de detectores permite diferenciar entre electrones y muones y estimar parámetros como la velocidad de la partícula que llega al detector y su energía, lo que resulta muy útil para tratar de identificar el rayo cósmico primario.

Dadas las grandes prestaciones de este tipo de detectores, pueden resultar una alternativa a los detectores habituales de muones. Además, pueden utilizarse de forma complementaria a las grandes redes de detectores más sencillos, como aquellos basados en el efecto Cherenkov, centelladores o RCPs sin capacidad de tracking.

Los detectores que tratan de identificar rayos cósmicos son muy diversos. Por ello, aunque nuestro código esté orientado al TRAGALDABAS de la USC, es interesante crear un código abierto y flexible para que pueda ser aplicado a otros detectores por otros grupos de investigación.

2. Preanálisis de los datos

Antes de comenzar con el programa es conveniente realizar un preanálisis de los datos para determinar cuales de las variables son las más significativas. Para estudiar la correlación entre las distintas variables con las que estamos trabajando , utilizamos la función *corr()* de Pandas. Esta función nos devuelve la matriz de correlación que buscamos. Para visualizar este resultado, en primer lugar representamos la matriz en forma de mapa de calor.

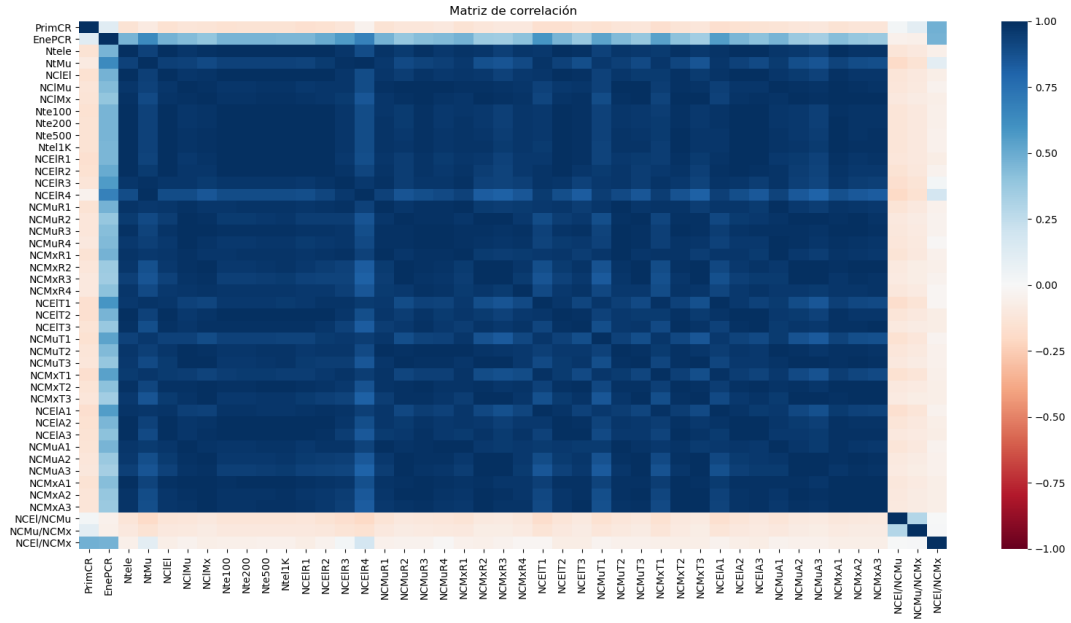

Figura 1: Matriz de correlación del primer conjunto de datos

Para visualizar los resultados con mayor claridad, y dado que lo que nos interesa es saber en que grado contribuyen nuestras variables a determinar la masa y la energía del rayo cósmico primario, hemos representado también por separado la correlación de estadas dos variables con el resto de datos (Figuras 2 y 3).

Esta clase de representaciones son de gran utilidad, ya que nos permiten determinar que variables son las más relevantes a la hora de identificar la energía y la naturaleza del rayo cósmico primario. De este modo, conocemos que características nos interesa que los detectores determinen con mayor precisión, lo cual puede resultar interesante de cara a la construcción de futuros detectores.

A primera vista, se observa la elevada correlación de la energía del rayo cósmico con los datos medidos en Tierra. Sin embargo, la correlación del tipo del rayo cósmico con estos datos es mucho menor. Destaca que la energía parece más relacionada con el número de cascadas formadas, su dispersión, ángulo entre las partículas y demás variables; mientras que el tipo de primario parece estar fuertemente relacionado con la proporción entre clústers de electrones y mixtos formados en las mencionadas cascadas. Entendemos por clústers los conjuntos de partículas que llegan a un área aproximadamente del tamaño de nuestro detector, procedentes del mismo rayo cósmico primario.

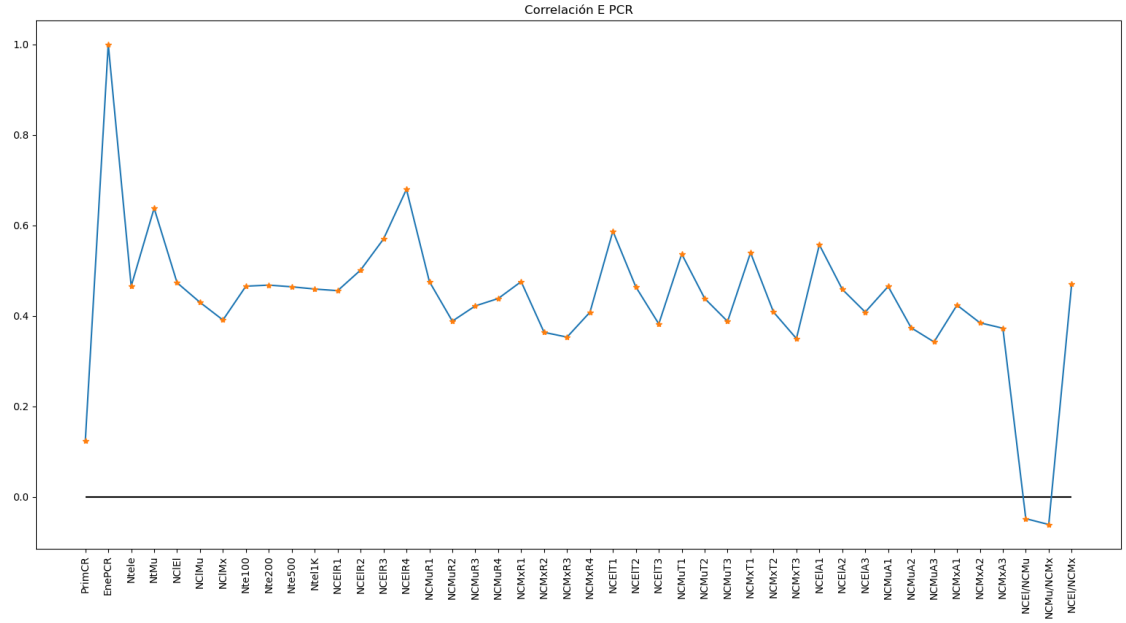

Figura 2: Correlación de la energía con el resto de variables

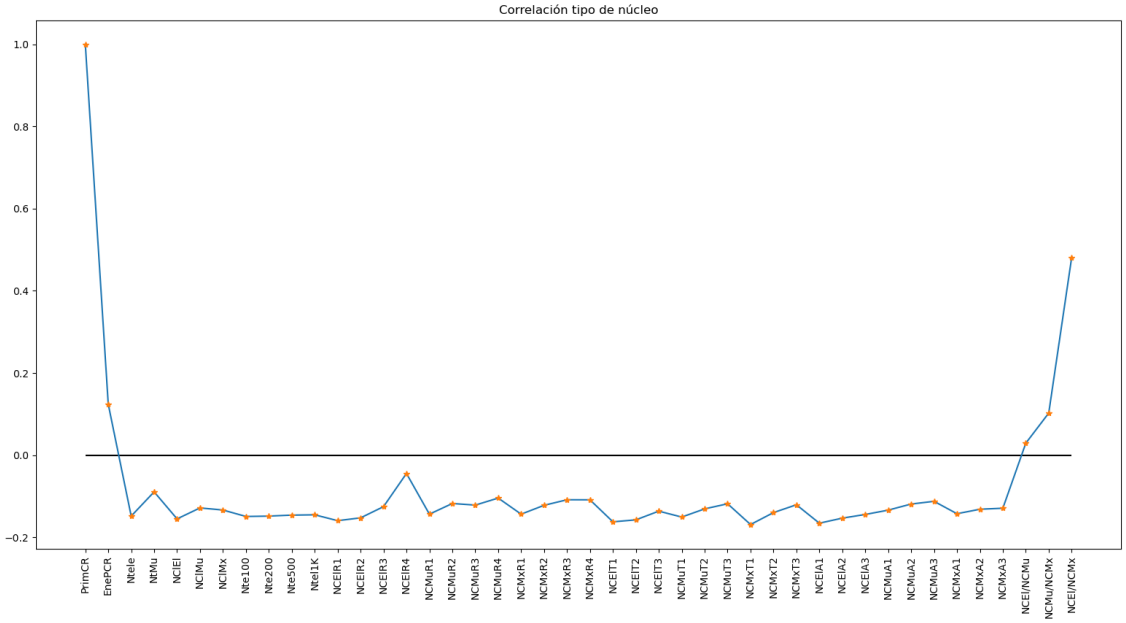

Figura 3: Correlación del tipo de núcleo con el resto de variables

Debido a unas anomalías detectadas en las simulaciones, se decidió simular un segundo conjunto de datos con unas ciertas modificaciones. Aprovechando la ocasión y tras haber realizado el análisis anterior al primer conjunto de datos, se decidieron modificar algunos de los features con los que entrenamos a nuestro programa.

En primer lugar, se eliminaron los datos relacionados con la dispersión de las partículas procedentes

de un mismo rayo cósmico, ya que esta no se puede medir con los medios de los que disponemos ahora mismo, y el objetivo de este trabjo es que sea aplicable a datos experimentales reales. Además, se dividió la energía de los electrones que llegan al detector en 8 intervalos en lugar de en 4, el tiempo entre que se detecta la primera partícula y la última en 6 intervalos en lugar de 3 y lo mismo para el ángulo entre partículas. Por último, se redujo la distancia máxima que puede haber entre dos partículas para que sean consideradas parte de un mismo clúster, para adaptar así el programa a las dimensiones de nuestro detector.

Una vez simulado el nuevo conjunto de datos, se realizó el mismo preanálisis que al primer conjunto, resultando las figuras 4, 5 y 6.

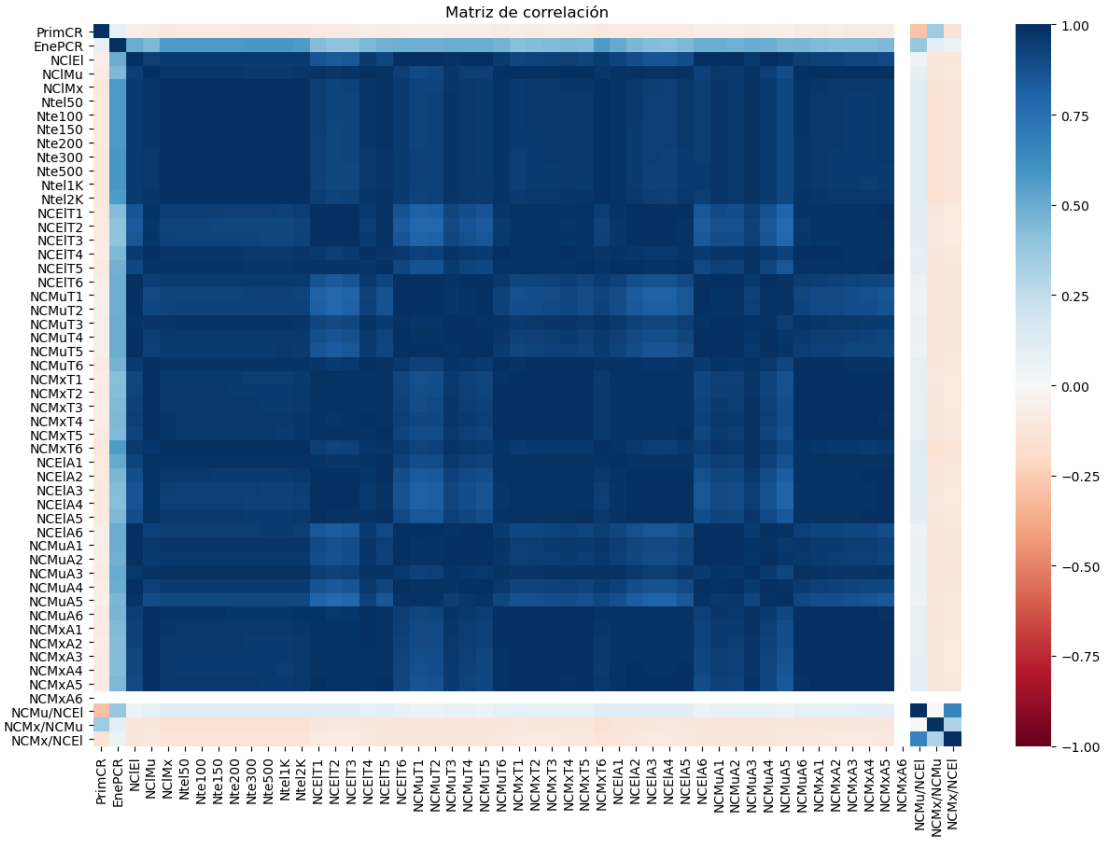

Figura 4: Matriz de correlación del segundo conjunto de datos

Se aprecia como las relaciones entre variables son similares al caso anterior, estando la energía muy relacionada con el número de partículas generado en las cascadas y las relaciones de energía, velocidad, ángulo entre estas; mientras que los distintos núcleos se diferencian en la relación entre los tipos de clústers producidos en las cascadas. Independientemente de esto, llama la atención que no existan clústers mixtos en el intervalo más grande de la diferencia de ángulo entre partículas (esto se puede consultar recurriendo a los datos simulados, y es el motivo por el que no aparece correlación con esta variable en nuestro análisis), lo que nos indica que la dispersión angular es menor en la formación de esta clase de clústers.

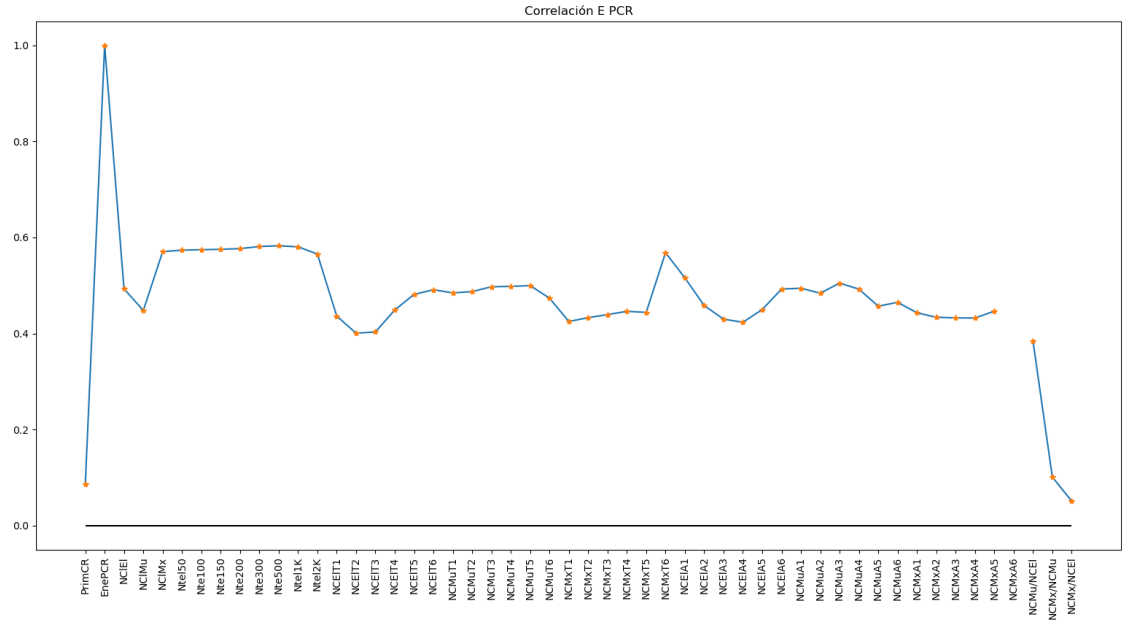

Figura 5: Correlación de la energía con el segundo conjunto de datos

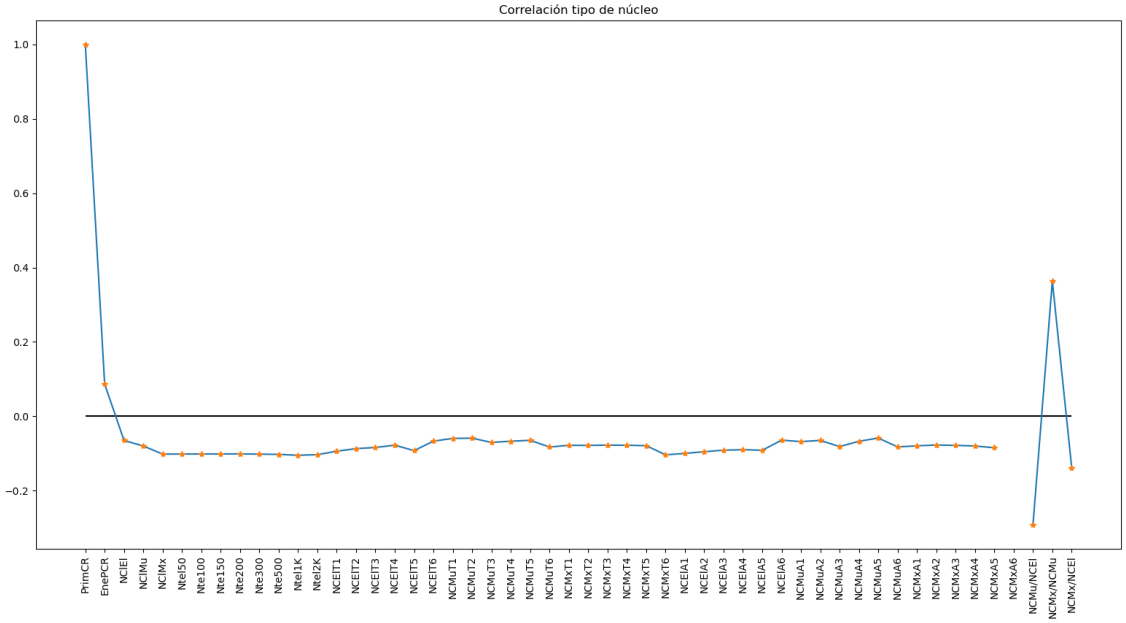

Figura 6: Correlación del tipo de núcleo con el segundo conjunto de datos

3. Red neuronal

Tras realizar este análisis de los datos utilizando Pandas, se procede a elaborar la red neuronal que queremos utilizar para identificar los rayos cósmicos. El programa a desarrollar es, en última instancia, una red neuronal capaz de resolver un problema de regresión de aprendizaje supervisado.

La red neuronal se elaboró utilizando la librería Keras de python, que se ejecuta sobre Tensorflow. Debido a la gran capacidad computacional que requiere este proceso, se recomienda ejecutar el programa utilizando una gpu potente. En nuestro caso, se accedió a los servidores de computación con gráficas del citius, en concreto a la gpu3.

En cuanto a la red, se utilizó una función *ReLu* como función de activación de cada capa de neuronas, y el optimizador *adam* de Keras para minimizar la función de error. Además, usamos como función de error *mean squared error* y como métrica *mean absolute percetage error*.

En primer lugar separamos 3 de los datos para utilizarlos como datos de test. De los restantes, el 20 % de los datos se utiliza como set de validación y el restante 80 % como datos de entrenamiento. Para que los datos seleccionados como test no sean siempre los mismos, lo que se hace es barajar los datos utilizando Pandas y después seleccionar las tres primeras filas.

Al ejecutar el programa, TensorFlow se encarga de entrenar la red a partir de los datos de entrenamiento y mide el error cometido sobre los datos de entrenamiento y los de validación. Estos errores son los que aparecen representados en las gráficas de error (Figuras 7, 8 y 9), donde las líneas azules son referidas a los datos de entrenamiento y las naranjas a los de validación. Esta clase de representaciones son muy útiles, ya que, si el error sobre los datos de entrenamiento es muy pequeño y sobre los datos de validación es muy grande, nos indica que la reda podría estar sufriendo overfitting, mientras que si ambos errores son muy grandes, estaría sufriendo underfitting. Así, este tipo de representación nos ayuda a saber como modificar nuestro programa para que funcione correctamente.

Por último, creamos una copia de los datos de test, y eliminamos en esta las variables que queremos estimar. Entonces, le pedimos al programa que nos estime la energía y el tipo de núcleo correspondiente a los mencionados datos, y comparamos los resultados proporcionados por el programa con los resultados reales de la simulación. De este modo, podemos comprobar manualmente en que proporción la red neuronal identifica correctamente el rayo cósmico.

En cuanto a la arquitectura de la red, se fue probando con distintas configuraciones hasta encontrar la óptima. Para decidir entre dos modelos de red se utilizaron dos factores. En primer lugar las gráficas de error, y en segundo lugar el número de aciertos y errores de cada uno de los modelos a la hora de estimar la energía y el tipo de núcleo del rayo cósmico primario. A la hora de estimar el número de aciertos y errores, se contó el número de veces que ha identificado correctamente la energía, el número de veces que ha identificado el tipo de núcleo, y el número de veces que ha identificado ambos simultáneamente, ya que el hecho de que determine ambos simultáneamente es un indicativo de que el programa está funcionando correctamente y no acierta por pura casualidad.

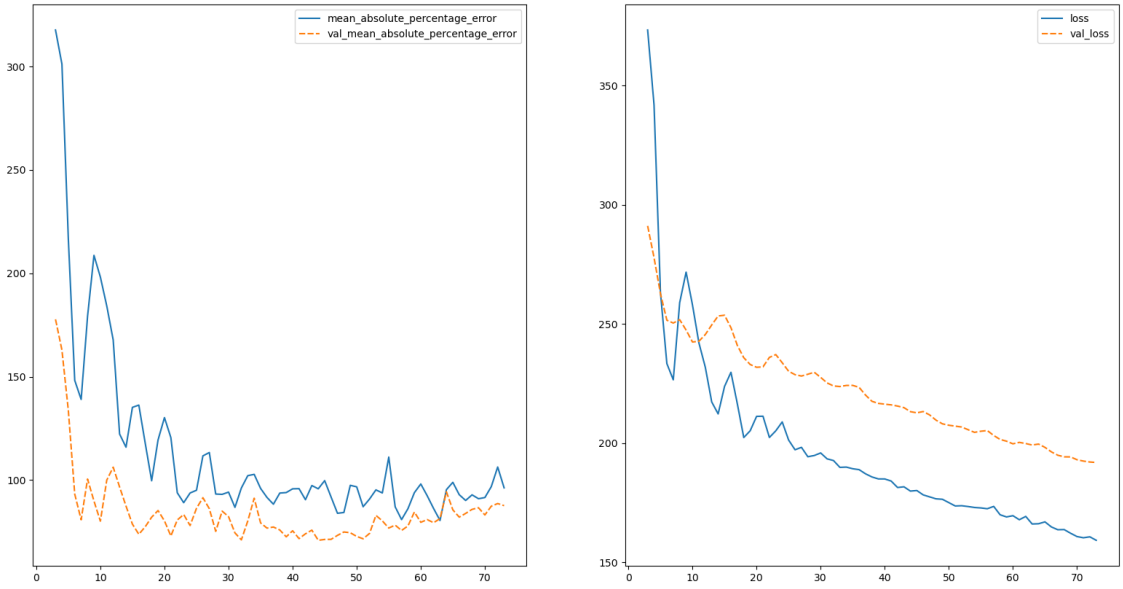

Figura 7: Gráfica de errores programa primer conjunto de datos

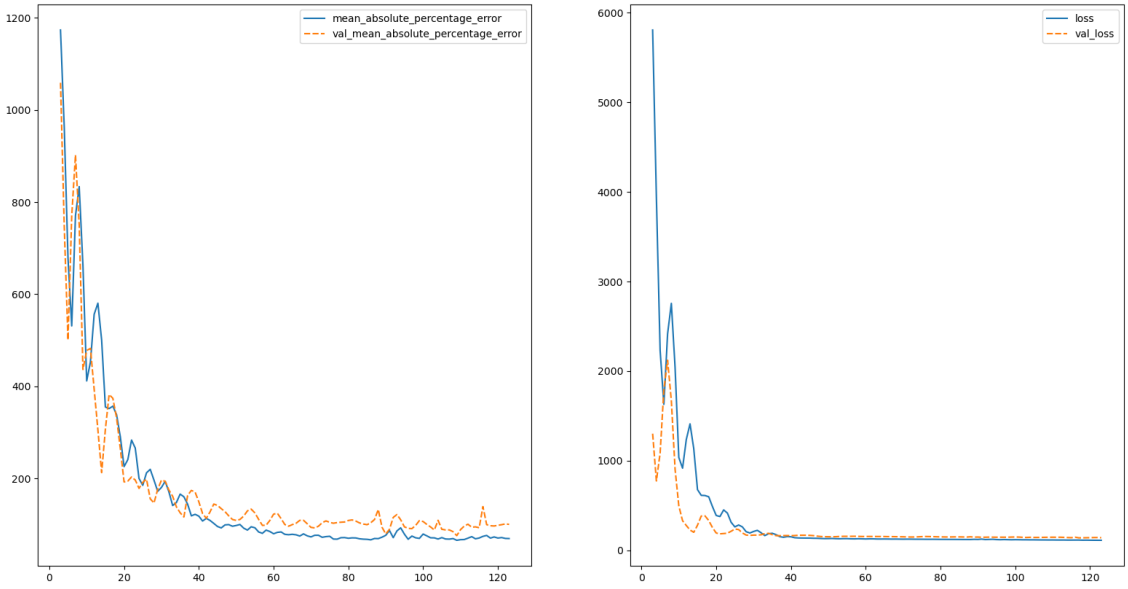

Figura 8: Gráfica de errores programa segundo conjunto de datos

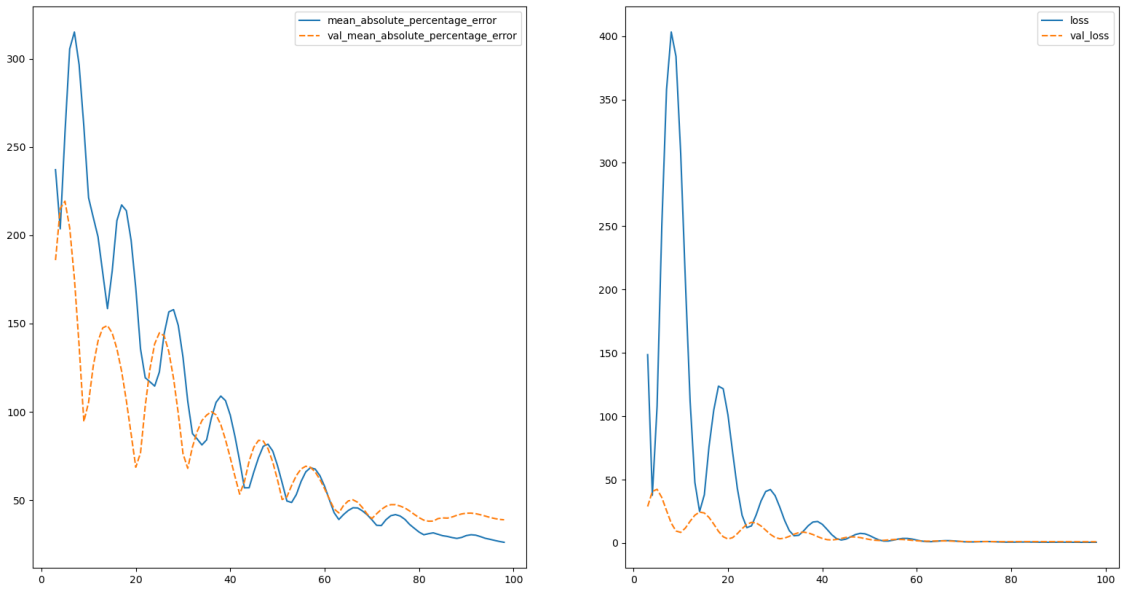

Figura 9: Gráfica de errores estimación energía protones

4. Desarrollo del programa

4.1. Primer conjunto de datos

En primer lugar, realizamos un programa más rudimentario para determinar en que medida somos capaces de estimar la energía de un conjunto de rayos cósmicos en el que todos los núcleos son protones. Una vez conseguimos que este programa funcione correctamente, introducimos el resto de núcleos y tratamos que el programa identifique correctamente la energía y el tipo de núcleo. En ambos casos, y tanto para el primer conjunto de datos como para el segundo, parece que lo óptimo es una red de dos capas más la salida, pero el número de neuronas en cada capa depende del caso.

Para el primer conjunto de datos, a partir de las gráficas de error se podía determinar que el programa estaba sufriendo de overfitting. Por lo tanto, nos vimos en la necesidad de deshacernos de algunas de las características de los datos para que el programa funcionase correctamente.

Tras diversas pruebas, una primera opción era deshacerse del número total de los clústers de cada tipo, así como de toda la información relacionada con los clústers mixtos. Aunque esto daba buenos resultados a nivel informático, no tenía demasiado sentido físico, ya que los clústers mixtos son una de las principales características que puede medir nuestro detector, mientras que estábamos dando un mayor peso a otros features quizás más complicados de medir con un detector real. Con esta configuración se conseguía determinar simultáneamente la energía y tipo de núcleo del rayo cósmico un 40 % de las veces, siendo la energía determinada correctamente un 43,2 % de las veces y el tipo de núcleo un 50 %, utilizando una red de 2 capas de 100 neuronas, realizando 75 iteraciones.

Para adaptar mejor el programa a las características de nuestro detector, se decidió prescindir del número total de partículas de cada tipo que llegan al detector y de la distancia a la que se encuentran los clústers del centro de la cascada, pero si incluir el número de clústers de cada tipo. Con esta configuración se consiguieron estimar simultáneamente ambas características un 30 % de las veces, siendo bien estimada la energía un 76,6 % de los intentos y el tipo de núcleo un 33,3 %. En esta ocasión, el programa parecía funcionar mejor con dos capas de 50 neuronas y aumentando el número de iteraciones a 125. Sin embargo, este programa parecía errar bastante a la hora de estimar los núcleos más pesados, dando casi siempre una predicción de número atómico menor o igual a 12.

Para solucionar este problema con la estimación de los núcleos pesados, se introdujo como dato la relación entre el número de clústers de cada tipo, característica de la que ya se ha hablado anteriormente. Así, conseguimos estimar ambas características simultaneamente un 33,3 % de las veces, la energía un 53,3 % y el tipo de núcleo un 43,3 %. Además, se regresa a la primera configuración de 2 capas de 100 neuronas y 75 iteraciones.

Si además nos tomamos la licencia de considerar que tenemos un continuo de detectores, de modo que podemos determinar la distancia del centro de la cascada a la que se encuentra el clúster, los resultados mejoran ligeramente. Con esta configuración, estimamos simultáneamente energía y tipo de núcleo un 36,6 % de las veces, siendo la primera estimada correctamente un 50 % de las ecuaciones y la segunda un 60 %. En esta ocasión se utiliza la configuración de 2 capas de 50 neuronas y 125 iteraciones.

4.2. Segundo conjunto de datos

El segundo conjunto de datos ya está más centrado en los datos que puede medir el detector, eliminado aquellos que implicarían tener un número infinito de detectores. Además, la distancia máxima entre partículas para que puedan ser consideradas parte de un mismo clúster se ha reducido, para adaptarse así a las dimensiones de nuestro detector. Esto, unido al menor tiempo del que se ha dispuesto para trabajar con este conjunto de datos provoca que los resultados sean ligeramente peores.

Comenzamos trabajando únicamente con protones, tratando de estimar su energía. La red que mejor parece funcionar para este caso es una red de 2 capas de 100 neuronas, realizando 100 iteraciones. Con esta configuración, logramos estimar correctamente la energía del protón un 56,6 % de las veces.

A continuación, introducimos el resto de núcleos y tratamos de estimar simultáneamente la energía y el tipo de núcleo del primario. Para prevenir el overffiting nos deshacemos del número total de partículas de cada tipo generado en las cascadas, manteniendo el número de clústers de cada tipo. Utilizando la misma red que para los protones, conseguimos estimar simultáneamente ambas incógnitas un 30 % de las veces, siendo el núcleo estimado correctamente un 36,6 % de las veces y la energía un 50 %.

5. Conclusiones

Este es un proyecto en sus comienzos y, aunque los resultados no son ideales, si que se pueden considerar prometedores. Queda de manifiesto que el método de utilizar una red neuronal para identificar la energía y el tipo de partícula que forma un rayo cósmico primario puede funcionar. En este sentido, de cara al futuro podría desarrollarse una red neuronal más potente y sofisticada capaz de identificar el rayo cósmico con mayor precisión.

Solo hemos utilizado una parte minúscula del potencial de TensorFlow. Así, una persona con amplios conocimientos en técnicas de redes neuronales podría mejorar el programa para evitar que el descenso del gradiente se quede en óptimos locales y evitar así que la red pueda emitir resultados erróneos. Además, se podría realizar un análisis de componentes principales para ser más rigurosos a la hora de sabes que variables descartar para prevenir el overffiting.

En definitiva, se trata de un proyecto que está empezando, pero cuyas posibilidades son infinitas, y que, con el tiempo, puede llegar a ser importante en muchos campos de la física.

6. Anexo: Programa informático

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 14 11:01:25 2021

@author: carlos lamas rodriguez carlos.lamas.rodriguez@rai.usc.es
"""

import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

print(tf.__version__)

plt.close()
datos=pd.read_excel('datos_pandas2.xlsx')
datos=datos.fillna(0)
#print(datos)

corr=datos.corr()
#print(corr)
plt.figure(figsize=(15, 15))
sns.heatmap(corr, cmap='RdBu', vmax=1, vmin=-1) #Representamos la matriz de correlación
plt.title('Matriz de correlación')
#np.savetxt('Matriz de correlación', corr)

corr_E=pd.DataFrame(corr,columns=['EnePCR'])
corr_PrimCR=pd.DataFrame(corr,columns=['PrimCR'])

plt.figure()
plt.plot(corr_PrimCR); plt.plot(corr_PrimCR, '*'); plt.xticks(rotation=90)
plt.hlines(0,'PrimCR', 'NCMx/NCEl', colors='black');
plt.title('Correlación tipo de núcleo')

plt.figure()
plt.plot(corr_E); plt.plot(corr_E, '*'); plt.xticks(rotation=90)
plt.hlines(0,'PrimCR', 'NCMx/NCEl', colors='black')
plt.title('Correlación E PCR')


datos_barajados=datos.sample(frac=1).reset_index(drop=True) #Reordenamos las filas
train=datos_barajados[3:]
test=datos_barajados[0:3]
```

```
resultados_test=pd.DataFrame(test,columns=['PrimCR', 'EnePCR'])
test=test.drop(['PrimCR', 'EnePCR'],axis=1)
#print(train)
#print(test)

X=train.drop(['PrimCR', 'EnePCR'],axis=1).values
Y=pd.DataFrame(train,columns=['PrimCR', 'EnePCR']).values
#print (Y)

assert X.shape[0]==Y.shape[0]
input_dim=X.shape[1]
output_dim=Y.shape[1]

model=keras.models.Sequential([keras.layers.Dense(
100,activation=tf.nn.relu,input_shape=(input_dim,)),
keras.layers.Dense(100,activation=tf.nn.relu), keras.layers.Dense(output_dim)])
model.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mean_absolute_percentage_error'])
model.summary()

validation_split=0.2
history=model.fit(X,Y,workers=4,epochs=100,verbose=2,validation_split=validation_split)

errores=pd.DataFrame(history.history)
f=plt.figure(figsize=(20,10))
rows=1; cols=2
ax=f.add_subplot(rows,cols,1)
sns.lineplot(data=errores[['mean_absolute_percentage_error',
'val_mean_absolute_percentage_error']].iloc[3:-1])
ax = f.add_subplot(rows, cols, 2)
sns.lineplot(data=errores[["loss", "val_loss"]].iloc[3:-1])
#f.savefig('Errores')

prediction=model.predict(test.values)
resultados=pd.DataFrame(prediction, columns=['PrimCR', 'EnePCR'])

print(resultados_test)
resultados_test.head()
print(resultados)
```