Escola Tècnica Superior d'Enginyeria ETSE-UV

MÁSTER PROPIO EN INTELIGENCIA ARTIFICIAL AVANZADA Y APLICADA

TRABAJO DE FIN DE MÁSTER

ESTUDIO PRELIMINAR DE ALGORITMOS DE IA PARA LA RECONSTRUCCIÓN DE ENERGÍAS EN EL EXPERIMENTO ATLAS DEL LHC

AUTOR:
ANDREU TARRASA LÓPEZ

TUTORES:
LUCA FIORINI
VALERO LAPARRA

SEPTIEMBRE, 2023

Resumen

La mejora de alta luminosidad del acelerador de partículas LHC del CERN conlleva un aumento notable del número de colisiones de partículas en cada cruce de haces. Esto implica por una parte la necesidad de filtrar y digitalizar a mayor ritmo las sucesivas lecturas de energía en el calorímetro del experimento ATLAS, y por otra el reto de reconstruir energías con apilado de señal. Es necesario por tanto la renovación de hardware y software para solventar estos desafíos. Este trabajo consiste en un estudio preliminar del desempeño de distintos modelos de inteligencia artificial en la reconstrucción de energías apiladas para sustituir los algoritmos utilizados en la anterior fase del experimento.

Abstract

The upgrade of high luminosity for the particle accelerator LHC from CERN entails an important increase of particle collisions in each bunch crossing. This implies in one hand the need of a higher rate capacity in the filtering and digitizing of successive energy readings in the ATLAS experiment calorimeter, and in the other the challenge of the reconstruction of high pileup energies. It is then required an upgrade in hardware and software to solve this problems. This work is a preliminary study of the performance of multiple artificial inteligence models in high pileup energy reconstruction intended to replace the algorithms used in the previous phase of the experiment.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación

Este Trabajo de Fin de Máster se integra en un proyecto de colaboración del acelerador de partículas LHC del CERN en el que participa el Instituto de Física Corpuscular (IFIC), institución fundada por la Universidad de Valencia (UV) y el Consejo Superior de Investigaciones Científicas (CSIC). Los experimentos ubicados en el anillo del LHC procesan cientos de TB/s y deben decidir qué datos almacenar de forma permanente con una latencia de pocos microsegundos. La próxima actualización del LHC [1] aumentará todavía más la producción de datos, lo que implica una renovación del hardware utilizado en el experimento ATLAS [2] para procesar esta información. Además entra en juego un nuevo fenómeno llamado apilado, del término inglés *pile up*, que consiste en la lectura simultánea de la energía originada por las colisiones que ocurren en el mismo BC y en BC cercanos. Esto ocurre al aumentar los eventos de colisión durante el tiempo de lectura de las señales en el detector del calorímetro *TileCal*. El proyecto consiste en diseñar y probar algoritmos de inteligencia artificial para ser ejecutados en aceleradores de hardware de tipo FPGAs para reconstruir las lecturas de energía con precisión en una ventana de tiempo restringida.

1.2. Objetivos

Diseñar, entrenar y comparar diferentes modelos de IA para la reconstrucción de energías a partir de datos generados en simulaciones siguiendo la configuración de la próxima actualiza-

ción del LHC de Alta Luminosidad (HL-LHC), comparando su rendimiento frente al algoritmo de reconstrucción de energías utilizado previamente *Optimal Filtering* (sección 4.2.1). Se estudiará la distribución de energías y su filtrado según el procedimiento de lectura de las señales dependiendo de la ganancia, *high-gain* (HG) y *low-gain* (LG), y se comparará el desempeño de las redes neuronales buscando su optimización para distintos tamaños de modelos según el número de parámetros entrenables, con el objetivo de optimizar la inferencia en el hardware. El objetivo y procedimiento de este trabajo se resume en el esquema de la figura 1.1.

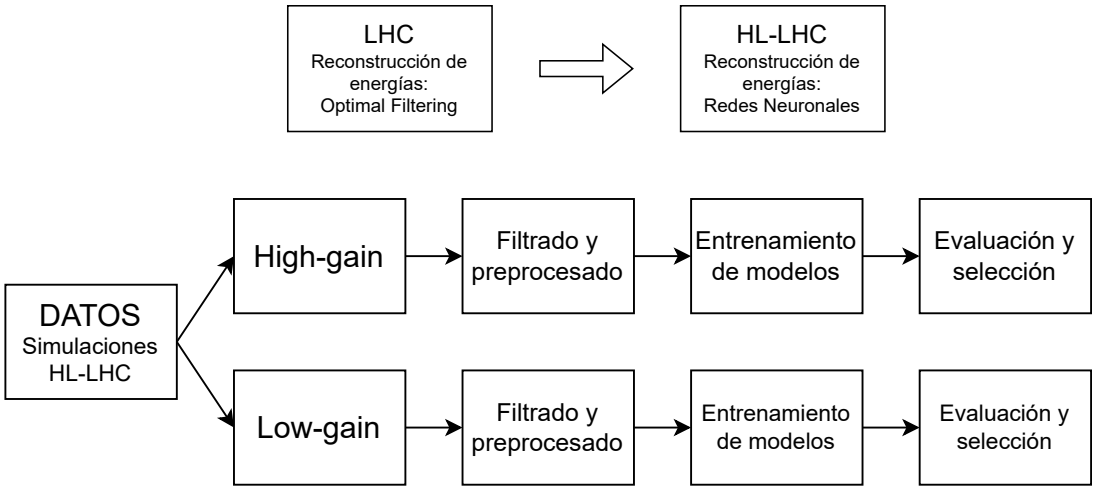


Figura 1.1: Esquema del desarrollo del trabajo.

1.3. Estudios relacionados

Ortiz Arciniega et al. [3] en 2019 analizaron el uso de modelos de *Deep Learning* para la reconstrucción de energías en mismas condiciones de alto ratio de colisiones y apilado de señal. Para ello entrenaron una red neuronal de capas densas, un perceptrón multicapa, con una función de activación del tipo sigmoide adaptada para permitir resultados negativos. El desempeño del modelo fue comparado con el algoritmo Optimal Filtering utilizado previamente en TileCal mostrando mejores resultados.

Duarte et al. [4] en 2019 propusieron el uso de deconvolución de señal mediante varios filtros FIR (*Finite Impulse Response*) de bajo consumo computacional y métodos iterativos basados en algoritmos de descenso por gradientes (GD), obteniendo este último mejor rendimiento en condiciones de alto apilado de señal y pudiendo ser adaptado en FPGAs.

Aad et al. [5] en 2021 estudiaron el uso de de modelos convolucionales (CNN) con diferentes configuraciones de capas, redes neuronales recurrentes (RNN) simples y del tipo *Long Short-Term Memory* (LSTM) para la reconstrucción de señal. En comparación con el algoritmo *Optimal Filtering* todas las versiones de redes neuronales obtuvieron mejores resultados, especialmente para bajas energías, siendo el modelo LSTM quien tiene un mejor desempeño a costa de utilizar más parámetros entrenables.

Capítulo 2

Experimento

2.1. LHC

Localizado en el CERN (Organización Europea para la Investigación Nuclear), en la frontera entre Suiza y Francia, el LHC (Gran Colisionador de Hadrones) es el acelerador de partículas más grande y potente del mundo. El LHC consiste en una estructura en forma de anillo con una longitud de 27km, construido en un túnel a unos 100m bajo suelo de media (figura 2.1), y hace uso de imanes superconductores de distintos tipos y tamaños para guiar las partículas mediante fuertes campos magnéticos a lo largo de su recorrido.

Dos haces son acelerados en dirección opuesta a través de dos tubos en vacío casi absoluto y hechos colisionar en distintos puntos clave del recorrido donde se alojan los cuatro detectores principales: los experimentos ATLAS (Aparato Toroidal del LHC) y CMS (Solenoide Compacto de Muones) [7], dos detectores de partículas de propósito general que investigan distintas ramas de la física como el bosón de Higgs, dimensiones extra y partículas que formarían la materia oscura; el experimento ALICE (Gran Experimento Colisionador de Iones), especializado en física de iones pesados y de materia con fuerte interacción a densidades de energía extremas; y el experimento LHCb (Gran Colisionador de Hadrones *bottom*) que se centra en investigar las diferencias entre materia y antimateria a partir de las desintegraciones del quark *bottom* o *beauty*.

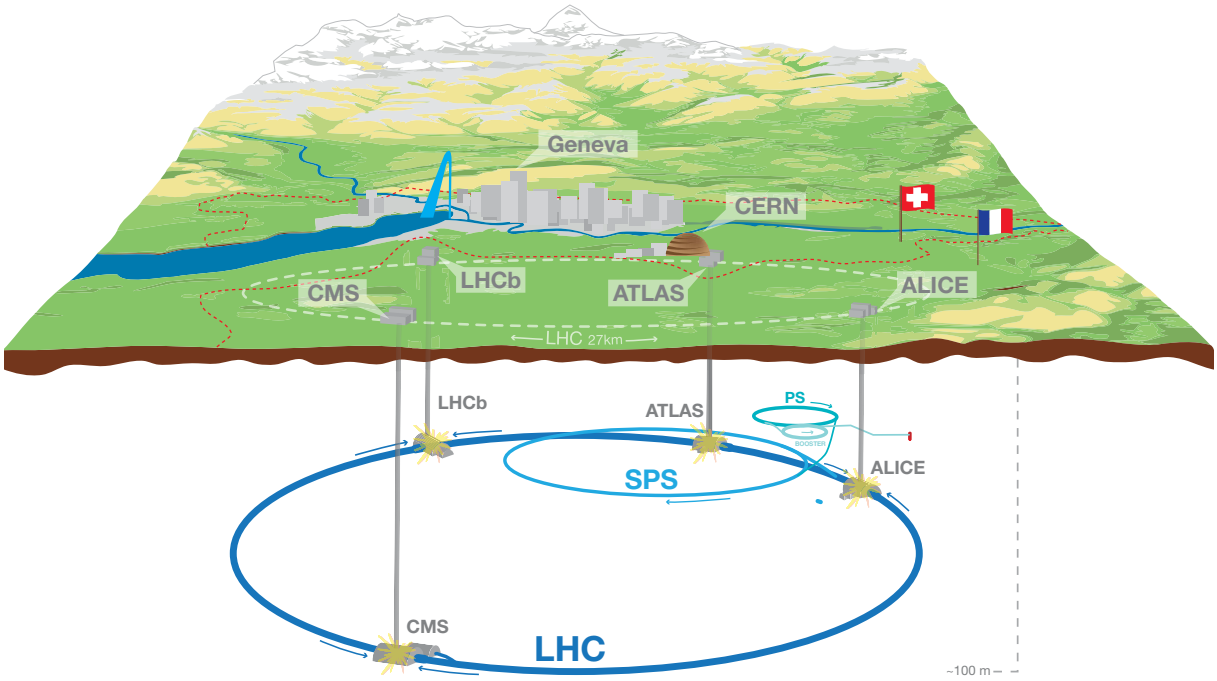


Figura 2.1: Vista general del LHC, incluyendo los experimentos ALICE, ATLAS, CMS,y LHCb [6].

2.2. ATLAS

A pesar de compartir los experimentos ATLAS y CMS los mismos objetivos científicos, cada uno utiliza diferentes soluciones técnicas y sistemas magnéticos. En el detector ATLAS los haces son dirigidos para colisionar en su centro y sus seis subsistemas organizados por capas alrededor del punto de impacto registran la trayectoria, momento y energía de las partículas resultantes del choque (figura 2.2). Las interacciones en el detector generan una gran cantidad de datos que deben ser filtrados por un sistema de selección antes de ser almacenados para su posterior procesado. La actualización al HL-LHC implica una mayor cantidad de interacciones *pp* (protón-protón) al producirse del orden de 200 choques en cada cruce de protones, llamados *bunch-crossings* (BC), lo que hace necesario la renovación de la técnica de filtrado inicial de las señales y de la electrónica.

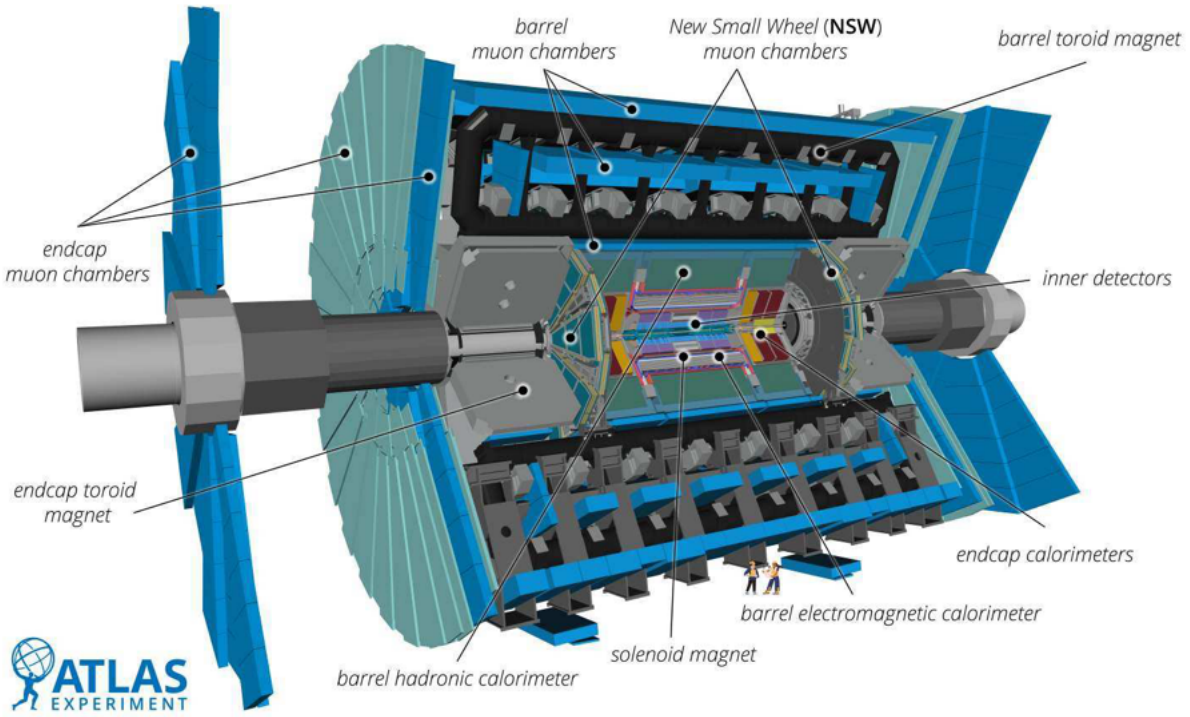


Figura 2.2: Esquema ilustrativo del experimento ATLAS incluyendo los nuevos sistemas instalados durante la parada de mantenimiento *Long Shutdown* 2 (LS2) en 2018 [8].

2.3. Calorímetro

El *Tile Hadronic Calorimeter* (Calorímetro Hadrónico de Placas, *TileCal*) [9] muestrea la energía de los hadrones (partículas con quarks, como los protones y los neutrones) surgidos de las colisiones en el centro del detector. Su estructura está ilustrada en la figura 2.3. El calorímetro tiene forma de corteza cilíndrica con un espesor radial de 1,95 m y está dividido longitudinalmente en cuatro grandes secciones: dos centrales (*central long barrels* LBA y LBC) con una longitud de 2,82 m cada una, y dos laterales externas (*endextended barrels* EBA y EBC) con una longitud de 2,91 m cada una.

El calorímetro está constituido por módulos compuestos de capas de acero y placas centelleadoras de plástico, ilustrados en la figura 2.4. La señal de cada centelleador es transportada desde sus dos extremos por fibras de cambio de longitud de onda (*wavelenght shifting fibers*, WLS). Estas fibras WLS se agrupan para formar celdas y conectan con los tubos fotomultiplicadores (PMT). El calorímetro está dividido en aproximadamente 4670 celdas, cada una leída por

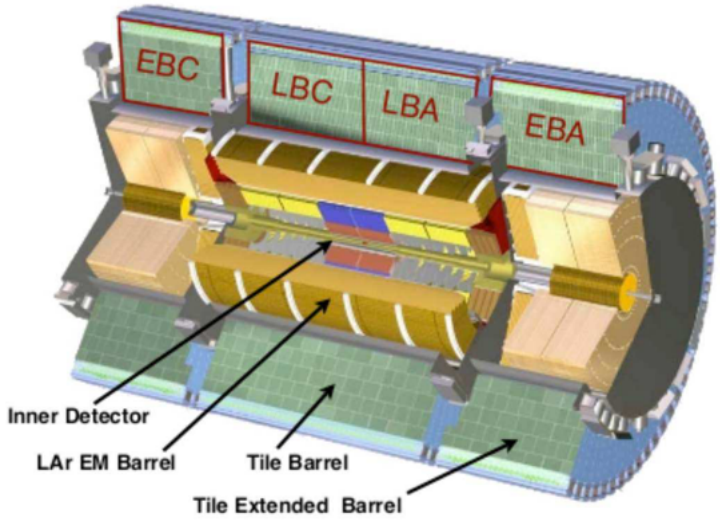


Figura 2.3: Distribución de las cuatro grandes secciones cilíndricas del calorímetro, las dos *central long barrels* LBA y LBC y las dos laterales externas *extended barrels* EBA y EBC.

dos PMT, y dentro de cada *barrel* repartidas en 64 módulos distribuidos a lo largo del ángulo acimutal ϕ alrededor del eje del haz de partículas. La figura 2.5 muestra el mapa de celdas para las cuatro secciones alrededor del centro del detector.

La señal de cada PMT es amplificada según dos ganancias, *high-gain* (HG) y *low-gain* (LG). En el cambio a la fase HL-LHC el ratio entre estas ganancias pasa de 64 a 40 [11] y los DAC (*Digital to Analog Converter*) previamente de 10-bits han sido sustituidos por versiones de 12-bits, lo que da a cada ganancia un rango de energías de 0-4095 ADC *counts*. Estos rangos equivalen a una lectura mínima fiable de entorno a 10 MeVs para los valores más bajos en HG (puede variar ligeramente entre celdas) y un máximo alrededor de 2 TeVs para los valores más altos en LG. Debido al ratio entre ganancias una misma señal leída en HG con un valor de 4095 ADC *counts* será leída en LG con un valor de 102 ADC *counts*. Las señales que no saturen la lectura en HG, es decir aquellas cuyo valor de lectura no supere el límite superior de 4095 ADC *counts*, serán leídas en esta misma ganancia (por tener mayor resolución), y aquellas que la saturen serán leídas en LG.

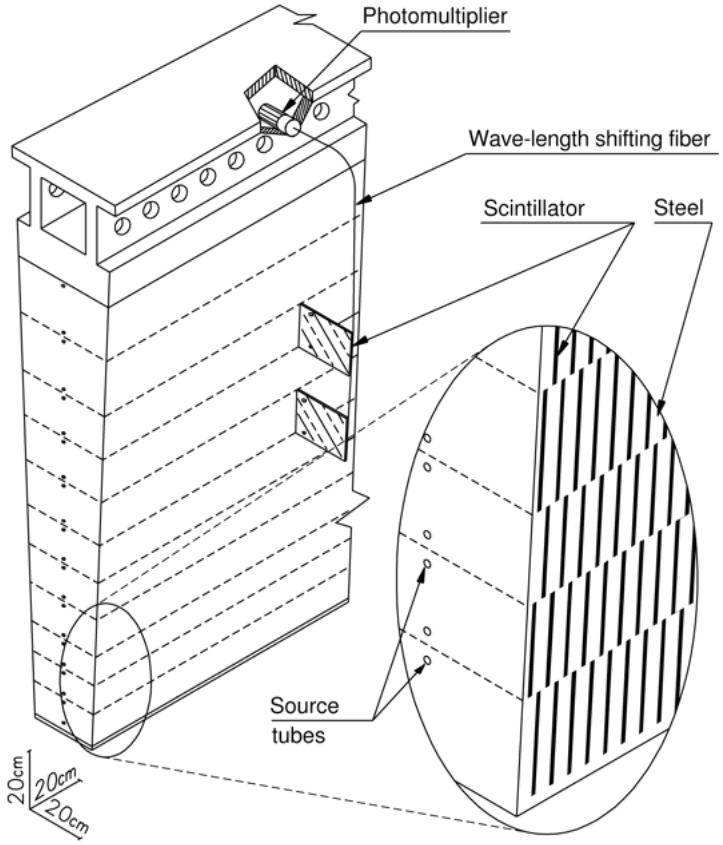


Figura 2.4: Uno de los 64 módulos distribuidos a lo largo del ángulo acimutal ϕ alrededor del eje del haz de partículas en el centro del calorímetro [9]. En esta ilustración el eje vertical está orientado según el eje radial del detector (la sección inferior apunta al centro del detector, y la sección superior con los fotomultiplicadores apunta al exterior).

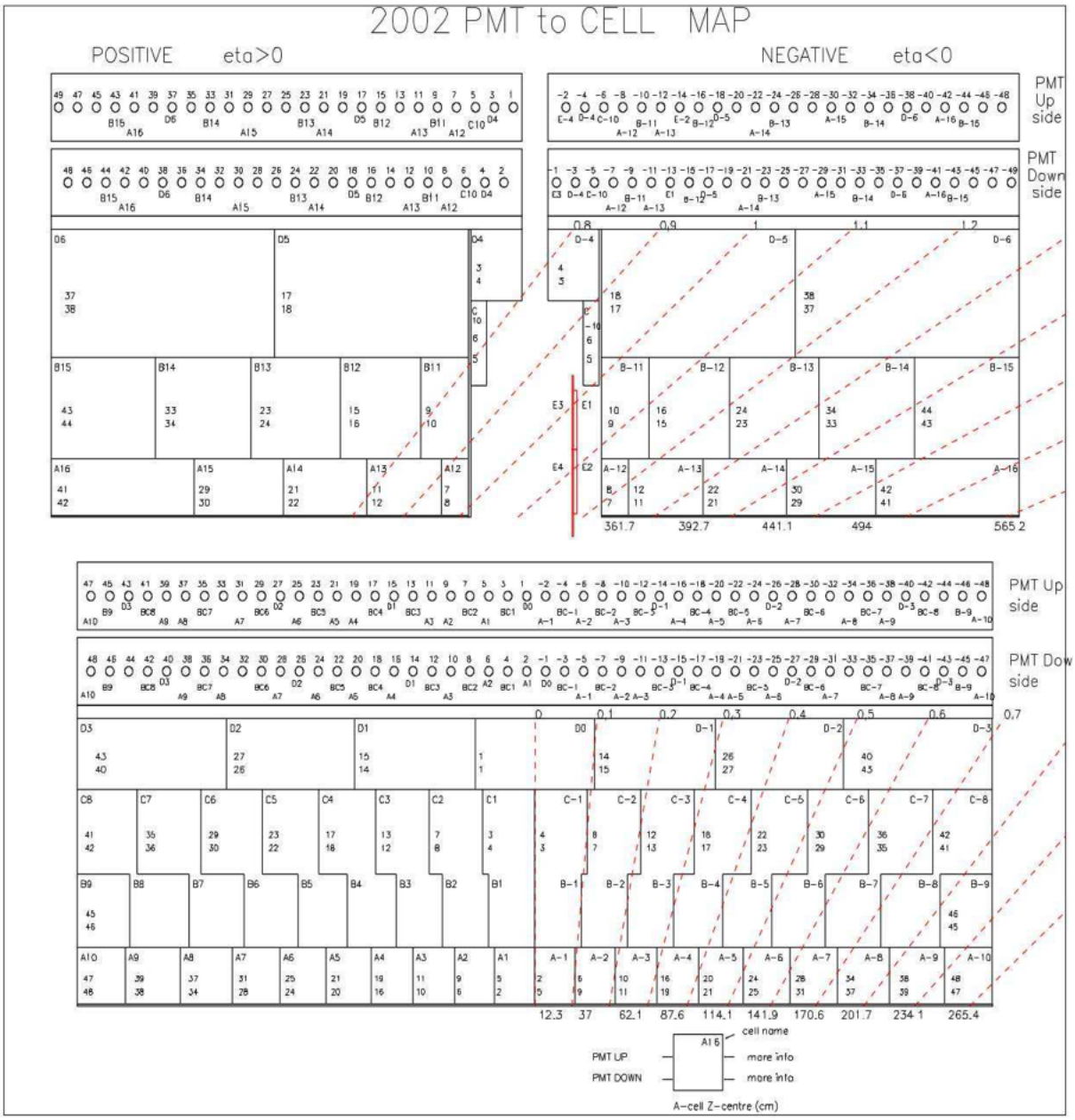


Figura 2.5: Mapa de celdas del calorímetro para los dos *long barrels* (abajo, LBC mitad izquierda y LBA mitad derecha) y los dos *extended barrels* laterales (arriba, EBC izquierda y EBA derecha). Las cuatro secciones se disponen a lo largo del mismo eje longitudinal (aquí eje horizontal) tal y como se muestra en la figura 2.3. Las líneas discontinuas rojas apuntan al centro del detector donde colisionan las partículas y corresponden a distintos valores del ángulo η llamado pseudorapidez, que se define como $\eta = -ln(tan\frac{\theta}{2})$ siendo θ el ángulo polar medido desde el eje del haz de partículas. Figura extraída de [10].

Capítulo 3

Datos

Los datos utilizados para entrenar los modelos consisten en simulaciones realizadas con el Simulador de Pulsos (*Pulse Simulator*) de la biblioteca de software *Athena* de la colaboración ATLAS del CERN [12]. Esta herramienta permite simular la señal de salida de la interfaz electrónica del calorímetro *TileCal* y obtener una digitalización de lecturas de las celdas con la configuración actualizada del HL-LHC. En concreto nos interesa el efecto de la electrónica en el apilado de la señal de alta luminosidad. En siguientes etapas del proyecto se pretende conseguir que los modelos entrenados sean capaces de reconstruir otros parámetros adicionales como el desfase entre el pico del pulso reconstruido y el pico real de la deposición de energía en el detector (figura 3.1).

3.1. Simulaciones

Con respecto a este trabajo las etapas más importantes de la construcción de la simulación del *Pulse Simulator* son las siguientes (representadas en la figura 3.2):

1. se simulan eventos de choques de partículas en el centro del detector (BC individuales de uno en uno), las trayectorias derivadas de los choques, y el impacto de esas partículas en las celdas. Se simula colisiones de 200 partículas simultáneas (parámetro μ) en cada BC;

2. se simula la energía depositada en cada celda;

3. se simula el comportamiento de la electrónica conectada a las celdas. El uso de condensadores, en concreto de su carga y descarga, degenera la señal puntual proveniente de las

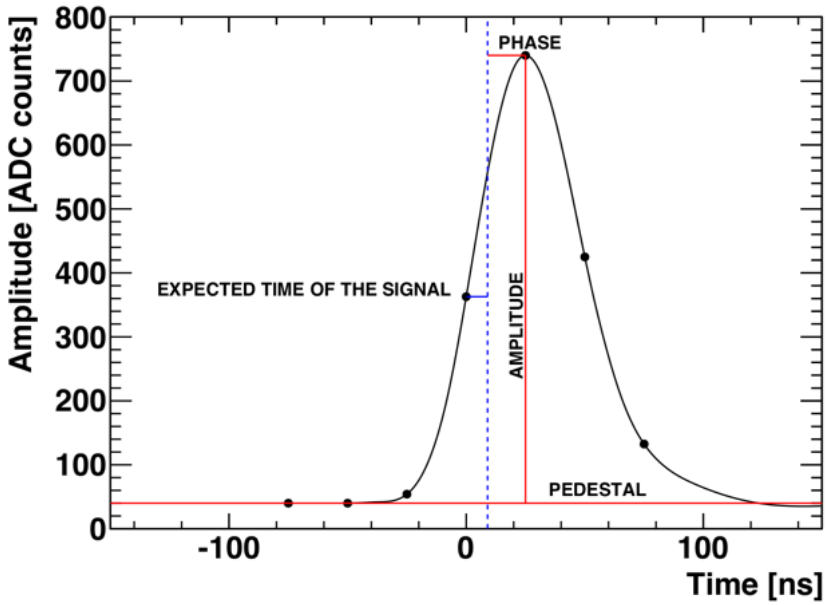


Figura 3.1: Forma del pulso físico (*pulse shape*) con definición de amplitud, fase reconstruida y pedestal. Los puntos representan siete lecturas o *samples* transmitidos por la electrónica. Figura extraida de [13].

celdas en una curva que se extiende a lo largo de varias decenas de nanosegundos. Esta curva de energía o pulso (*pulse shape*) es característica de esta electrónica. Se simula la señal a través de la electrónica escalando la curva del pulso con el valor de la energía detectado por la celda en cada BC;

4. se simula el solapamiento de las curvas de los pulsos de energía de sucesivos BC. Al extenderse las curvas de energía asociadas a cada BC más allá de los 25ns que separan sus lecturas, se toma en consideración el solapamiento de las colas de los sucesivos pulsos. Este solapamiento se simula sumando las propias curvas a lo largo de decenas de BC;

5. se convierte la señal de analógica a digital tomando una lectura puntual de esa curva apilada resultante (con un máximo saturado de 4095 ADC *counts*) cada 25ns. Estas lecturas son los *samples* de cada BC en nuestros datos de entrenamiento;

6. se suma un valor arbitrario fijo, llamado pedestal, a todos los *samples*. En nuestro caso este pedestal es de 100 ADC *counts*

Los valores leídos cada 25 ns (*samples*), son los registros digitales de las lecturas de los

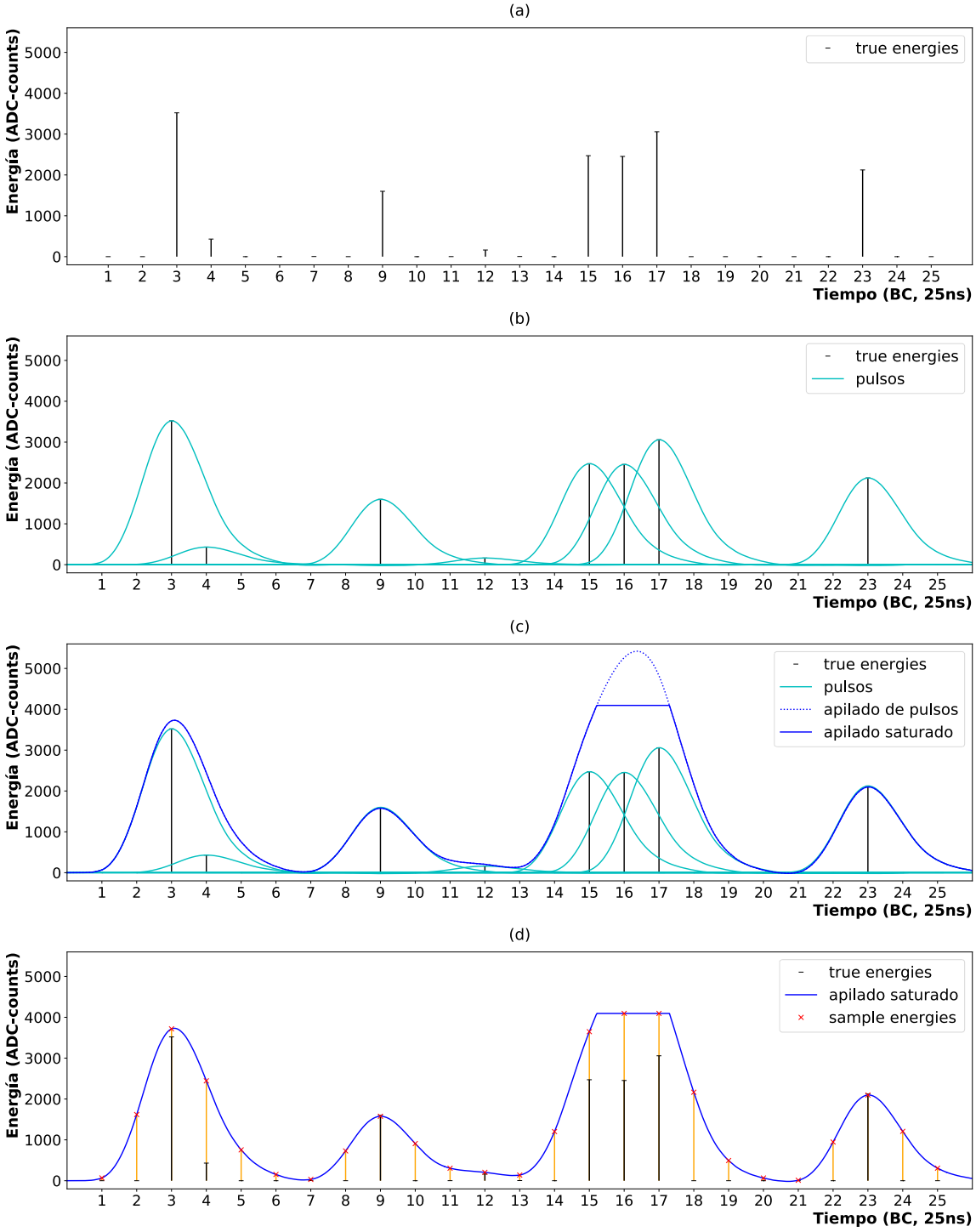


Figura 3.2: (a) Valores de energías depositadas por cada BC. (b) *Pulse shapes* debidas a la electrónica de cada BC. (c) *Pileup* por solapamiento de los pulsos y saturación de su digitalización en 4095 ADC *counts*. (d) Digitalización de los *samples* de energía en cada BC a partir de la curva de *pileup* saturada.

PMT. Hay que resaltar que en estas simulaciones no hay relación de índole física entre partículas para sucesivos BC ya que son simulados de forma independiente, y por tanto tampoco hay relación entre las energías depositadas sucesivas. La relación entre sucesivos datos aparece al sumar las curvas de energía, y por tanto los modelos solo aprenderán el efecto de la electrónica en la lectura y el registro de las señales de las celdas.

De las partículas generadas en cada colisión con $\mu = 200$ protones solo una pequeña fracción es considerada como señal, entendiéndose por ello las partículas o procesos de interés para el experimento en cuestión. La energía depositada en las celdas del detector por el resto de partículas generadas será considerada ruido de apilado o *pileup*. Diferenciamos entre *in-time pileup* y *out-of-time pileup*. *In-time pileup* hace referencia a las interacciones *pp* cuya deposición de energía coincide temporalmente con la del evento de señal, y cuyos pulsos coinciden por tanto en la digitalización de la lectura del BC. Estos eventos no pueden ser diferenciados sin la ayuda de sensores adicionales y su *true energy* está incluida en el valor de la energía del BC (*target*) junto con la del evento de señal (los modelos no aprenderán a diferenciarlos). *Out-of-time pileup* hace referencia a las interacciones que ocurren en BC anteriores o siguientes al evento de interés, de modo que las colas de sus pulsos solapan con el pulso del evento de señal medido en dicho BC, añadiendo energía por solapamiento. La *true energy* de estos eventos de ruido no se incluye en la del BC y se espera que los modelos aprendan a diferenciar estos eventos y a eliminar su energía en las reconstrucciones de los eventos de señal.

Las simulaciones se han realizado con 3 configuraciones distintas, generando para cada una de ellas un millón de BC. La simulación *Pileup* solo contiene eventos aleatorios de ruido de *pileup* y su característica es que son mayormente eventos de baja energía. Las otras dos simulaciones (HG y LG) han sido configuradas con un 80 % de estos mismos eventos de ruido aleatorios y con un 20 % de eventos de señal (con una distribución plana de energías), favoreciendo en cada simulación energías que serán leídas en la ganancia correspondiente. Estas dos simulaciones con señal son las siguientes:

- HG; las energías de los eventos de señal simulados no saturan la lectura de HG, salvo cuando tengan *pileup* debido a otros eventos que puedan producir lecturas por encima de 4095 ADC *counts*. Estos eventos, al saturar la lectura en HG, serán leídos en la ganancia

LG con valores por encima de 102 ADC *counts*.

- LG; los eventos de señal simulados saturan en muchos casos la lectura de ganancia HG y abarcan todo el rango de energías en la lectura de LG. Los valores de *true energy* en ganancia LG pueden superar ligeramente 4095 ADC *counts* debido al *pileup*.

Se muestran las distribuciones de energía finales de las simulaciones en la figura 3.3.

Para el entrenamiento de los modelos se ha utilizado la librería Tensorflow [14]. Debido a la gran cantidad de datos disponible se ha considerado varias opciones para el pipeline de preprocesado de datos y entrenamiento de modelos. Entre ellos se ha probado uso de la clase *tf.data.Dataset* junto con métodos de creación de datasets como generadores o slicing de tensores. Esto permite preprocesar durante el propio entrenamiento las siguientes secuencias de timesteps en CPU paralelamente al procesado de los datos en GPU. La organización interna de los datos en los archivos resultado de las simulaciones (formato *ROOT* [15]) implica el uso de loops anidados para acceder de manera secuencial a los datos deseados, y tras distintas pruebas se ha optado por la siguiente metodología:

- realizar inicialmente una copia completa en bruto de los datos de las simulaciones desde el archivo formato *ROOT* a un fichero en formato *h5*. El acceso optimizado a matrices de datos de las librerías h5py y Numpy permite extraer posteriormente de forma eficiente los canales a utilizar a archivos en formato *npy*, reduciendo el tamaño de los ficheros a almacenar durante la fase de desarrollo. Realizar la secuencia posterior de preprocesado directamente sobre estos arrays de Numpy permite limitar el uso de loops y disminuir drásticamente el tiempo de preprocesado y entrenamiento de días a horas. Finalmente se utilizará este tipo de arrays durante el entrenamiento de los modelos.

- reducir la cantidad de datos para los entrenamientos y evaluaciones durante el desarrollo inicial del proyecto. Se ha trabajado solo con las señales de una celda del detector, en concreto con los dos PMT de las celdas A1 de LBA y LBC, para un total de 4 PMT. Por conveniencia en la partición de archivos se ha integrado los datos de los 64 módulos acimutales de las celdas A1 en un mismo entrenamiento, ya que se espera una detección similar de partículas para las mismas celdas a lo largo de los módulos debido a la simetría axial del detector. Además, del conjunto de 1 millón de BC de cada simulación

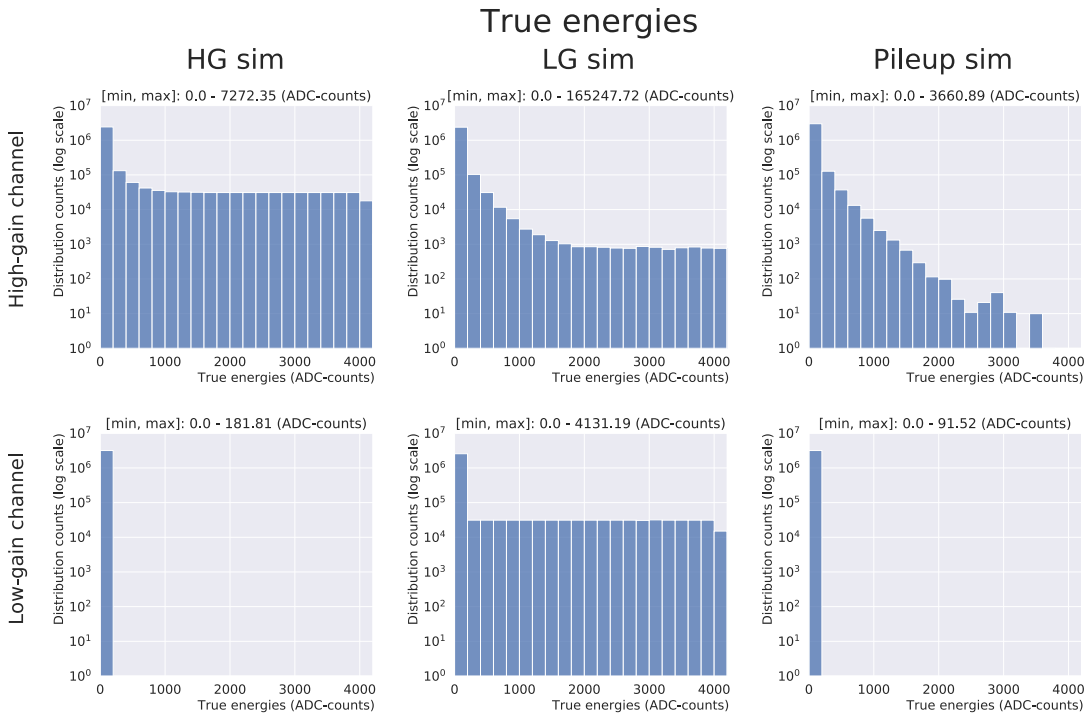
se ha extraído únicamente 50.000 BC para estos entrenamientos. Según la configuración del preprocesado, que se discutirá a continuación, resultará en un total de entre 2 y 10 millones de entradas de datos para cada ganancia (HG y LG).

3.2. Preprocesado

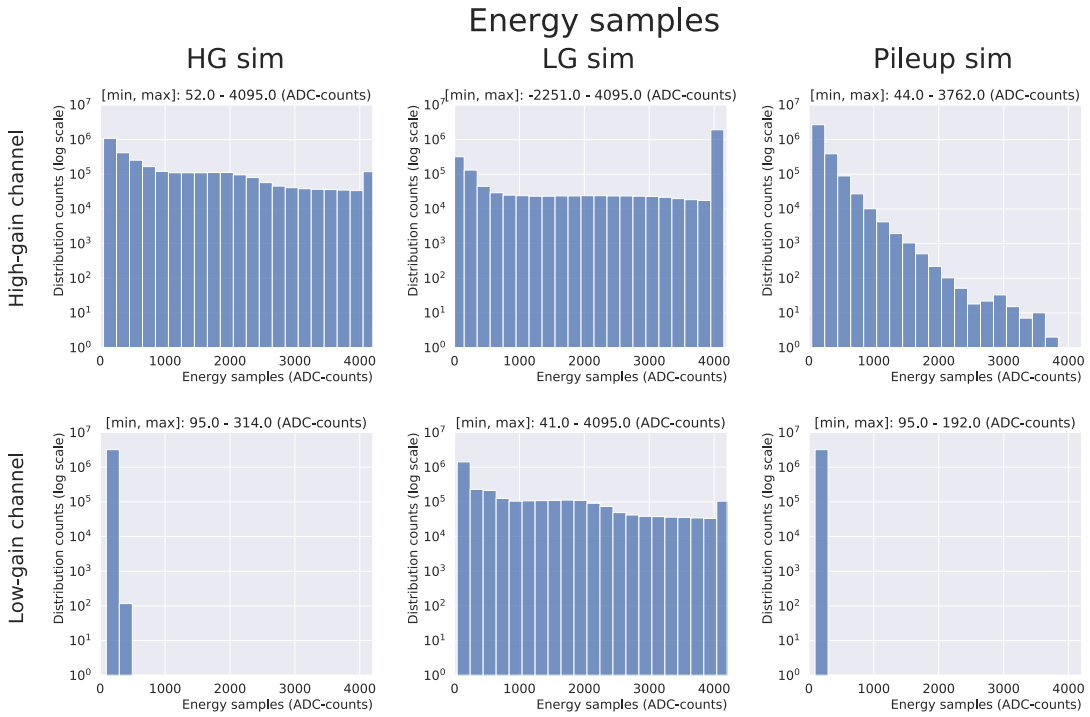
El preprocesado final de los datos ha consistido principalmente en dos partes: el filtrado de datos y la estructuración de enventanado. Debido a que el primero depende del segundo realizamos primero el enventanado, como se muestra a continuación.

El objetivo de los modelos será reconstruir la energía real (conocida en las simulaciones) a partir de la digitalización de las mediciones de las celdas. Es decir se trata de un problema de predicción o regresión. Tal y como se muestra en la figura 3.4 agrupamos 15 *samples* (BC) consecutivos, que serán nuestros datos de entrada, y asignamos como target la energía real del BC central de la ventana. Se pretende que los modelos aprendan a reconstruir esa energía a partir del *sample* del propio BC y de sus 14 vecinos más próximos (7 previos y 7 posteriores). Se considera esta cantidad de BC por ser suficientemente amplia como para abarcar la mayor influencia de los pulsos vecinos en el *pileup* del BC central, y a la vez suficientemente pequeña como para no dificultar demasiado los cálculos durante la inferencia de los modelos.

Con esta estructura fijada realizamos un filtrado según varios criterios explicados a continuación. Se propone inicialmente utilizar datos de las 3 simulaciones para el entrenamiento de cada ganancia, aunque finalmente se entrena los modelos de LG únicamente con datos de la simulación LG. Debido a que la ganancia HG amplifica más la señal del ruido, se espera obtener mejores resultados para los entrenamientos en LG, donde el ruido se confundirá menos con la señal. En cuanto a la distribución de energías, debido a la desigualdad entre señal y ruido tenemos muchos más eventos de baja energía (ruido poco energético) que de alta energía (señal), un bias que podemos visualizar en la figura 3.3 (nótese la escala logarítmica para el conteo de la distribución).



(a)



(b)

Figura 3.3: Distribución de energías (*energy samples* y *true energies*) de las 3 simulaciones (antes del preprocesado) para HG y LG (conteo de las distribuciones en escala logarítmica), con energías mínima y máxima en cada caso, y rangos de ejes acotados entre 0 y 4095 ADC *counts*.

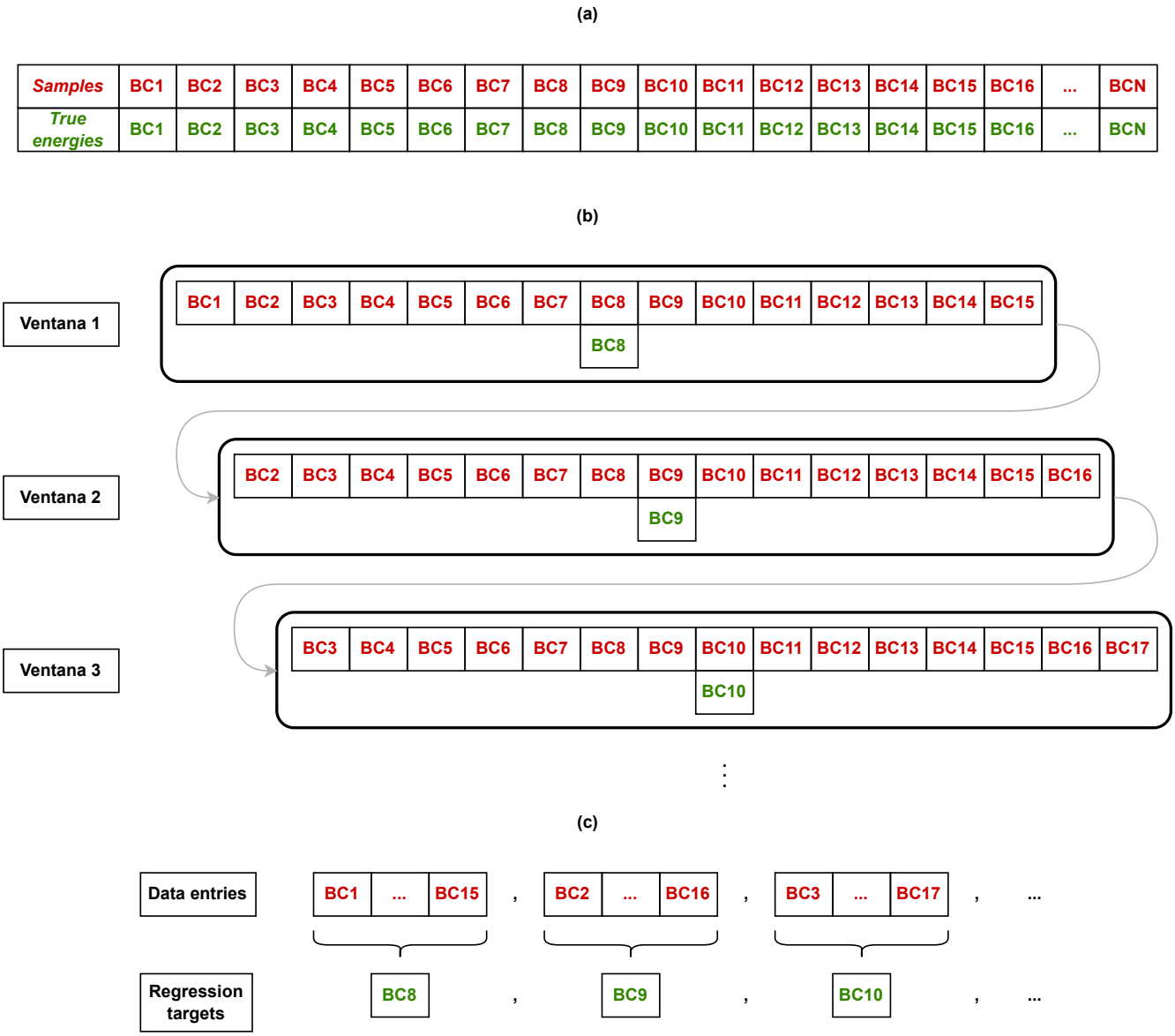


Figura 3.4: (a) *Samples* (datos) y respectivas *true energies* (targets). (b) Enventanado de 15 BC para datos, deslizando la ventana de corte en pasos de 1 BC, y selección del target central. (c) Datasets finales de tamaño (N-14, 15) para datos y (N-14) para targets.

Realizamos los siguientes filtrados en los datos de HG:

- eliminamos todas las ventanas de 15 BC en las que al menos uno de los *samples* sature la lectura de HG, es decir que tenga un valor de 4095 ADC *counts*. En los experimentos con el detector se utiliza las lecturas de LG cuando HG está saturado, por lo que podemos quitar esos valores (que no se usarían) y que además falsean la forma de la curva de energía (ya que se estancan en el techo de 4095 ADC *counts*) dificultando el aprendizaje del comportamiento real al modelo. Esto elimina en la práctica la mayoría de datos de la simulación LG (en el entrenamiento de HG), y los eventos muy energéticos del pileup que puedan saturar la lectura de la ganancia.

- eliminamos todas las ventanas de 15 BC con al menos un *sample* con valor por debajo de 10 ADC *counts*. La *pulse shape* utilizada en las simulaciones tiene un tramo final negativo en la cola de la curva llamada *undershoot*, que en el caso de apilado de varios pulsos de alta energía (incluido eventos energéticos del ruido en la simulación HG) puede producir samples con valores por debajo del pedestal. Se decide eliminar esos datos de los entrenamientos.

- eliminamos igualmente ventanas cuya energía real *target* sea inferior a 10 ADC *counts*. Energías tan bajas son compatibles con las contribuciones de *pileup* y del ruido electrónico, por lo que estos valores no son utilizados en los experimentos.

Por otra parte para los datos de LG realizamos el siguiente filtrado:

- utilizamos únicamente datos de la simulación LG. La mayoría de datos de señal de la simulación HG no se leen en ganancia LG.

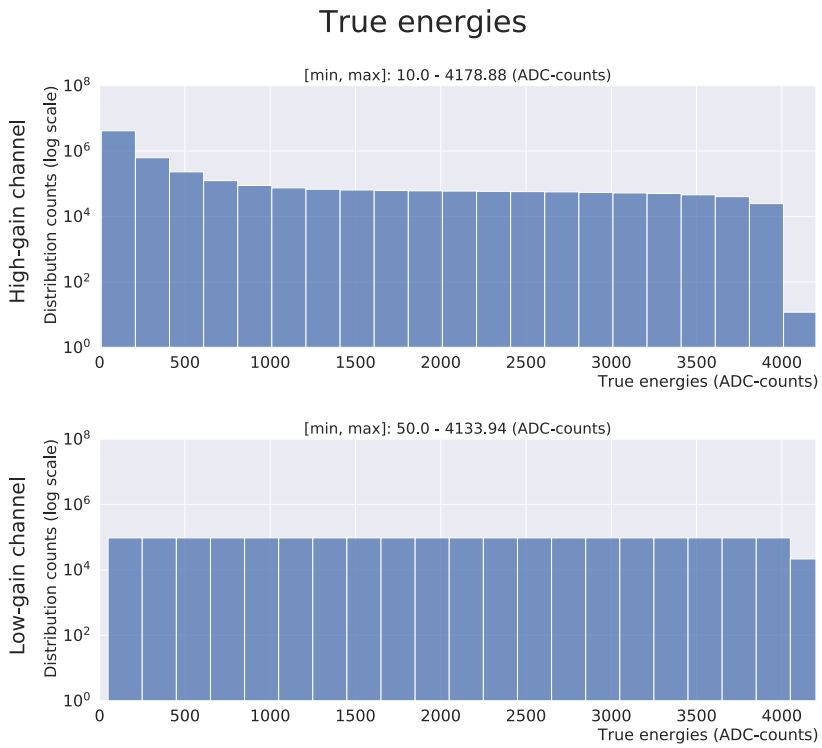
- eliminamos todas las ventanas con energía real por debajo de 50 ADC *counts*. Este valor es conservador y podría aumentarse a 100 ADC *counts* ya que valores inferiores no saturan la lectura de HG y por tanto no sería usado LG para leerlos.

A continuación se realiza una separación de los datos, para cada ganancia, en 3 datasets de *Train* (75 % de los datos), *Validation* (12,5 %) y *Test* (12,5 %). Esta distribución se realiza seleccionando entradas de forma aleatoria, sin alterar el orden de los BC dentro de cada ventana.

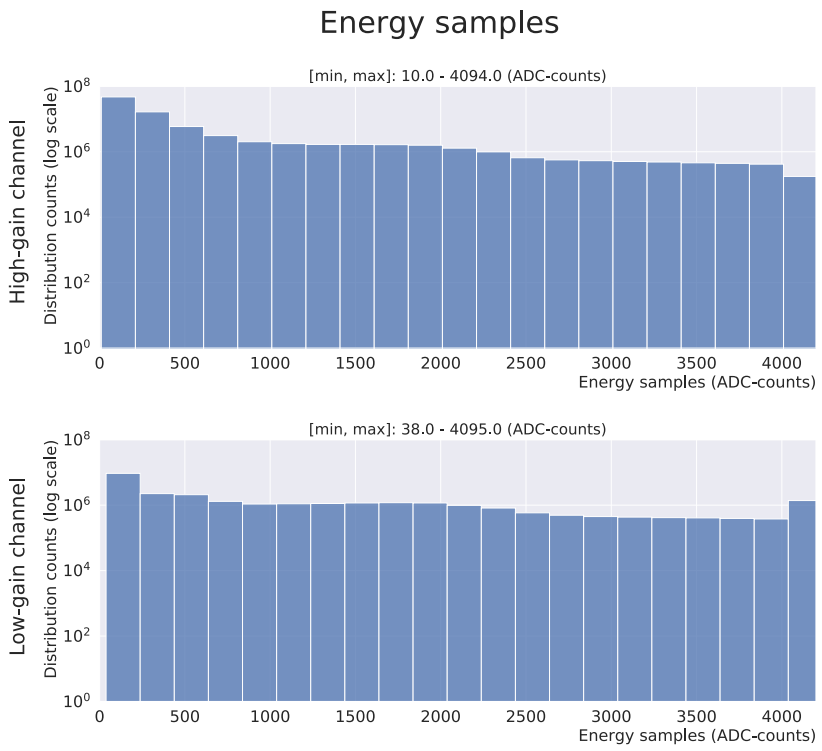
Finalmente se normaliza los valores de los datos entre 0 y 1. Sabiendo que las energías filtradas deben de estar contenidas en el rango 0-4095 ADC *counts* utilizamos estos márgenes tanto para *samples* como para *true energies*.

Posteriormente a estos filtrados, los datasets de *Train* cuentan con aproximádamente 6 millones de entradas de datos (ventanas de 15 BC) en el caso de HG y 2 millones para el caso de LG (el filtrado ha eliminado el 80 % de los datos). En una segunda fase se decide aumentar la muestra de datos de LG utilizados para igualar la muestra de 6 millones de HG.

Se puede apreciar en la figura 3.5 que la distribución de energías en LG tras los filtrados es mucho más homogénea. En el caso de HG seguimos teniendo un sesgo pronunciado en la muestra de eventos con energías bajas y probaremos a realizar un pesado de los datos en el entrenamiento para intentar contrarrestarlo.



(a)



(b)

Figura 3.5: Distributión de energías (*energy samples* y *true energies*) del dataset de *Train* (después del preprocesado) para ganancias HG y LG (conteo de las distribuciones en escala logarítmica), con energías mínima y máxima en cada caso, y rangos de ejes acotados entre 0 y 4095 ADC *counts*.

Capítulo 4

Modelos y entrenamiento

Uno de los objetivos de este proyecto es el de obtener modelos eficientes durante la inferencia, tanto en el tiempo de cómputo como en los requisitos de memoria, siempre con vistas a ser ejecutados en FPGAs. En la futura fase del experimento HL-LHC se dispondrá de tarjetas FPGA Xilinx KU115, debiendo procesar cada una la lectura de 87 canales simultáneamente, a una frecuencia de 40MHz (debido a la lectura de BC cada 25ns). Por este motivo se ha optado por el uso de modelos sencillos en su estructura, sin grandes bloques de capas, y con un número relativamente pequeño de parámetros entrenables. Se han entrenado en dos versiones, una primera con menos restricciones en el número de parámetros entrenables (alrededor de $4 \cdot 10^3$), con el fin de no limitar demasiado el posible aprendizaje de cada tipo de modelo y poder ver más fácilmente si hay diferencias entre ellos, y otra con un número inferior (alrededor de 10^3), para poder comparar cuánto empeora cada tipo de modelo al reducir esta variable, y para tener una idea más cercana al caso final de uso real de los modelos en FPGAs. Además de esta reducción de parámetros entrenables previa al entrenamiento se ha propuesto para futuros pasos en el proyecto un segundo modo de optimización de modelos consistente en utilizar *weight-regularizer* de tipo L1 [16], un método de regularización durante el entrenamiento que permite reducir a cero el peso de las neuronas menos relevantes durante el cálculo de la función de pérdida, reduciendo así de forma selectiva el número final de parámetros entrenables.

4.1. Función de pérdida y pesado de los datos

A la salida de la neurona, o de la última capa de neuronas en el caso de redes, terminado el *forward pass* a través de todas las capas (conjunto de cálculos del modelo sobre los datos hasta obtener una predicción), el resultado final es evaluado. Dependiendo de cada tipo de problema este paso será distinto, pudiendo tratarse por ejemplo de la comparación de la categoría predicha por el modelo para un caso de clasificación con la categoría real del dato (conocida). En nuestro caso, tratándose de una regresión (una predicción de valores) comparamos el valor predicho con el valor real asociado a la entrada de datos en cuestión. Esta evaluación del resultado se realiza con la llamada función de pérdida (*loss function*), obteniendo el valor de pérdida (*loss*).

Para todos los modelos hemos usado como función de pérdida la media del error absoluto (*Mean Absolute Error*, MAE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - x_i| \tag{4.1}$$

siendo y_i las energías reales, x_i las energías predichas, y N el número total de datos sobre los que se realiza la evaluación.

Visto el gran sesgo (*bias*) de los datos (mucho mayor número de bajas energías que de altas), no corregido para los datos de la ganancia HG, se propone el uso de la técnica del pesado de los datos (*sample weighting*) que consiste en dar una importancia distinta a cada dato durante el cálculo de la función de pérdida, en nuestro caso en función de la energía. Se ha estudiado dos funciones distintas para la distribución de los pesos, representadas en la figura 4.1:

- tangente hiperbólica: asigna un peso de 0 a energías nulas y crece con una pendiente ligeramente decelerada hasta un peso máximo de 0.76 para la energía máxima (normalizadas entre 0 y 1);
- exponencial negativa inversa: ajustada de forma aproximada a la distribución de energías según una función exponencial negativa y aplicada inversamente sobre los datos, aplica un peso casi nulo la sección inicial (donde hay mayor distribución de energías) y tras crecer exponencialmente al disminuir los eventos se mantiene constante el resto del rango de energías.

Se ha de tener en cuenta que los pesos solo se aplican a los datos del dataset de *Train*, no a *Validation*, por lo que la pérdida calculada será distinta en cada caso (visible en la curva de

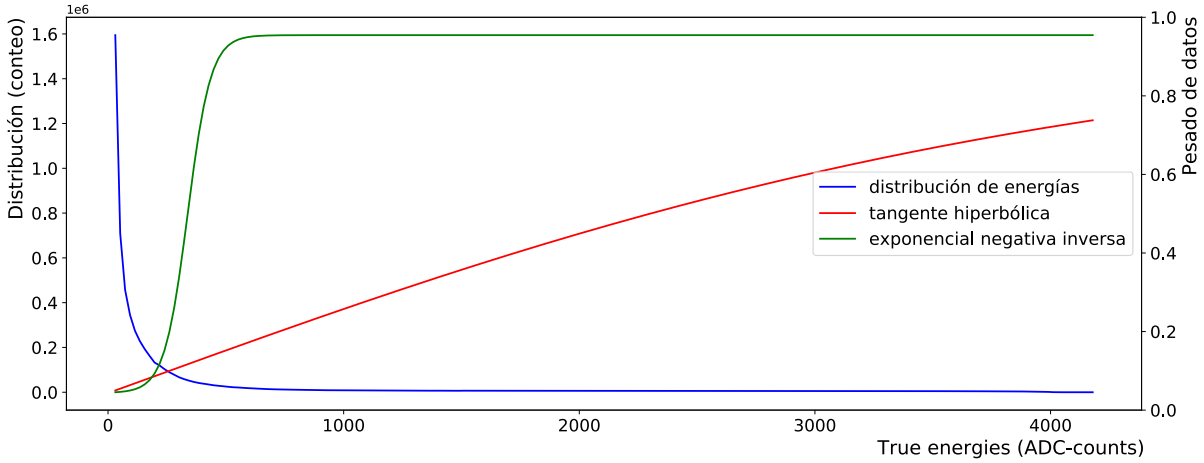


Figura 4.1: Distribución de energías y funciones tangente hiperbólica y exponencial negativa invertida utilizadas para pesar los datos.

aprendizaje de muestra A.1). Al no afectar a *Validation* ni *Test* puede seguir interpretándose sus pérdidas como la media del error absoluto del modelo sobre esos datasets.

4.2. Método de referencia y modelos neuronales entrenados

Se muestra un resumen de los modelos entrenados con sus estructuras de capas en la sección 4.4.

4.2.1. *Optimal Filtering* (OF)

Optimal Filtering (OF) es el método lineal de reconstrucción de energías utilizado en la anterior fase del experimento [17], que se pretende sustituir con modelos de redes neuronales. Este método toma en cuenta el BC central y los 6 vecinos más próximos para reconstruir la amplitud A de la energía (fórmula 4.2):

$$A = \sum_{i=1}^{7} w_i S_i \tag{4.2}$$

siendo S_i las amplitudes de los *samples* utilizados y w_i sus pesos asignados. Hemos igualado a cero los pesos de los BC restantes hasta completar la ventana de 15. La versión de OF utilizada hasta el momento en este trabajo es una adaptación del método original a un *pileup* de $\mu = 50$. Está previsto utilizar en futuras comparaciones una versión con $\mu = 200$ (acorde a las

simulaciones y los futuros datos en el detector).

4.2.2. Conceptos básicos: neurona, Perceptrón Simple y modelo Lineal

Una neurona realiza el cálculo básico dentro de una red. Este cálculo consiste en una combinación lineal de los datos de entrada ponderados con unos pesos entrenables. Esta combinación lineal tiene además un término independiente, y a su resultado se le aplica comúnmente una función de no linealidad llamada función de activación. El resultado final tras aplicar esta función se considera como la salida de la capa que aloja a la neurona, y será usado como dato de entrada de la neurona (o neuronas) de la siguiente capa, si la hubiera. Esta estructura básica es llamada perceptrón simple (figura 4.2).

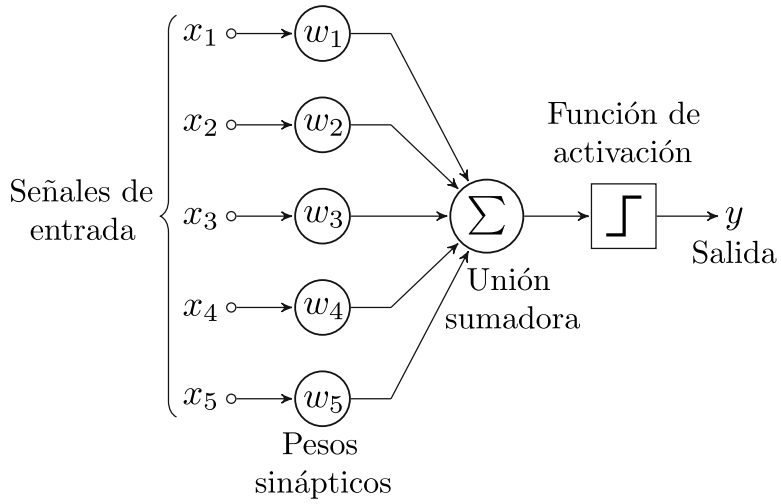


Figura 4.2: Esquema de una neurona y sus conexiones formando un perceptrón simple (término independiente de la combinación lineal no representado), junto con la función de activación y resultado de salida.

El modelo lineal consiste simplemente en una única neurona que aplica la combinación lineal sobre los *timesteps* de la entrada de datos. A su salida no se utiliza ninguna función de activación, para mantener la linealidad del cálculo.

4.2.3. Perceptrón Multicapa (MLP)

Un perceptrón multicapa (*Multi-Layer Perceptron*, MLP) [18], es una red neuronal de múltiples capas densas, un tipo de capa donde todas las neuronas (o datos de entrada para la primera capa) conectan con todas las neuronas de la siguiente capa. Es una generalización del perceptrón simple en una red profunda mediante el añadido de capas adicionales intermedias llamadas capas ocultas (figura 4.3). Se ha convertido en uno de los modelos más conocidos y frecuentemente usados.

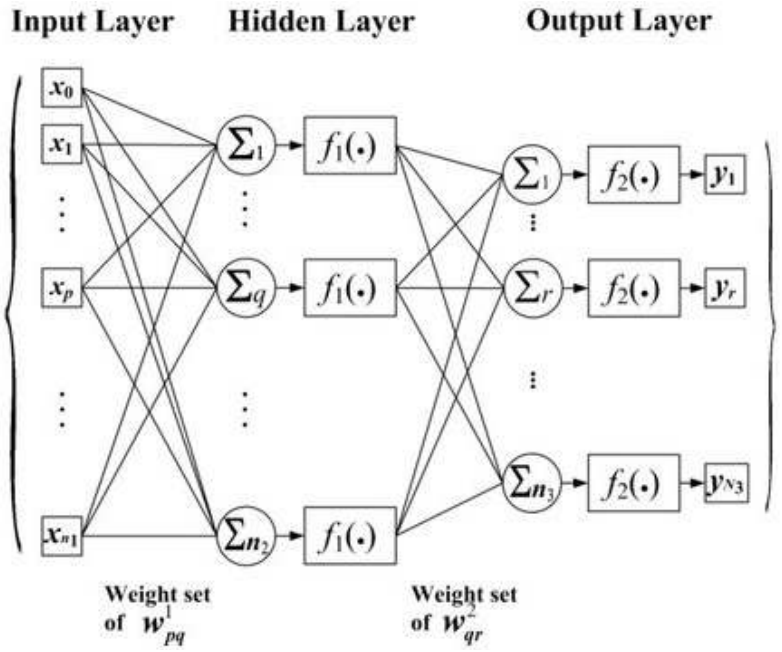


Figura 4.3: Esquema de un perceptrón multicapa, con las conexiones entre todas las neuronas de cada capa, las funciones de activación f_i a la salida de cada neurona y los resultados de salida en la última capa.

4.2.4. Red Neuronal Convolucional (CNN)

Las redes neuronales convolucionales (*Convolutional Neural Network*, CNN) [19] se diferencian de los perceptrones multicapa principalmente por la extracción de características de los datos mediante el uso de capas convolucionales. Estas, a diferencia de las capas densas, realizan una convolución de un filtro o *kernel* (análogo a una neurona) a lo largo de los datos de entrada. En nuestro caso los datos de entrada y los filtros son vectores de una dimensión, por

lo que el resultado final de cada convolución es también un nuevo vector (figura 4.4). El uso de distintos filtros sobre los datos permite aprender distintos patrones, dando como resultado el llamado mapa de características (*feature map*). En un ejemplo de procesado de imágenes con detección de formas geométricas una red de capas densas necesitaría aprender a reconocer los mismos patrones en nuevos datos de entrada si estos se encuentran en una zona distinta de cada imagen. En cambio, en una red convolucional, el hecho de realizar el mismo cálculo (usando los mismos pesos para cada filtro) en distintas ”subsecciones”de los datos de entrada implica una cierta invariancia traslacional de los patrones aprendidos. En el mismo ejemplo, una red CNN puede extraer patrones de formas geométricas específicas sin importar su localización dentro de la imagen, por lo que este tipo de redes es más eficiente necesitando menos datos para su aprendizaje. Además, el uso de distintas capas convolucionales consecutivas (con nuevos filtros) permite detectar nuevos patrones más complejos y específicos a partir de los aprendidos previamente.

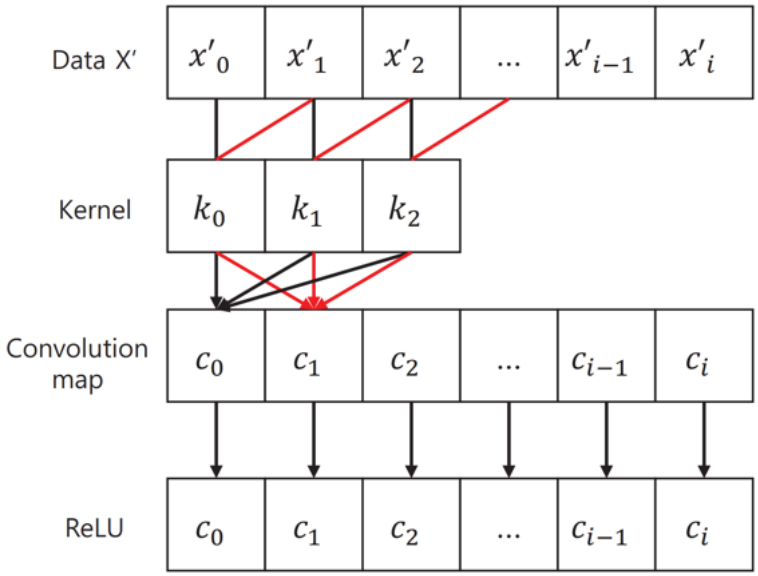


Figura 4.4: Esquema de una capa convolucional utilizando como con datos de entrada y filtros vectores 1D y activación ReLU a su salida [20]. Las flechas negras y rojas entre la capa *Data X'* y la capa de *Kernel* representan el primer y segundo paso respectivamente de la convolución del kernel sobre los datos de entrada.

En problemas donde se pretende extraer muchas características, de alta complejidad, utili-

zando además grandes bases de datos, suele ser habitual utilizar capas llamadas *pooling*. Estas capas agregan la información de píxeles o valores vecinos mediante distintos métodos (suma, promedio, selección de máximos, etc), lo que permite reducir la dimensionalidad del resultado de cada convolución conservando la información más relevante. El uso de estas capas reduce el número de parámetros entrenables de la red y ayudan a evitar el sobreajuste. Además, contribuyen a hacer el modelo más resistente a distorsiones de los patrones, como su escala u orientación en el ejemplo del procesado visual. Dependiendo del problema en cuestión (clasificación, regresión, etc) se añaden nuevas capas tras los bloques convolucionales, como *flatten* antes de una capa densa de salida, *softmax*, etc.

4.2.5. Red Neuronal Recurrente *Long Short-Term Memory* (LSTM)

Tratándose de un problema de regresión con series temporales se opta por incluir un modelo de red neuronal recurrente (*Recurrent Neural Network*, RNN) [21]. Una de las ventajas de estas redes es que tienen en cuenta la dependencia temporal y la información contextual de los datos, al contrario que las redes de capas densas como MLP que toman los *timesteps* de las secuencias de datos de entrada como valores sin orden secuencial. Esto se hace posible al utilizar como datos adicionales de entrada durante el cálculo sobre un *timestep* el resultado obtenido del cálculo sobre el *timestep* anterior.

Uno de los modelos más punteros en el campo de las redes recurrentes es el *Long Short-Term Memory* (LSTM) [22]. Su aspecto más característico es el uso de puertas lógicas para conservar información, llamada estados, de manera inalterada a través de entrenamiento sobre sucesivos *timesteps*. De esta manera los cálculos no dependen únicamente del paso anterior, sino que pueden conservar información a medio y largo plazo en secuencias de datos de mayor longitud. Las redes LSTM en concreto utilizan tres puertas lógicas (*forget*, *input* y *output gates*) y dos estados internos (*cell* y *hidden states*), y su configuración interna depende de la variante del modelo (se muestra en la figura 4.5 un ejemplo de proceso interno en una capa LSTM).

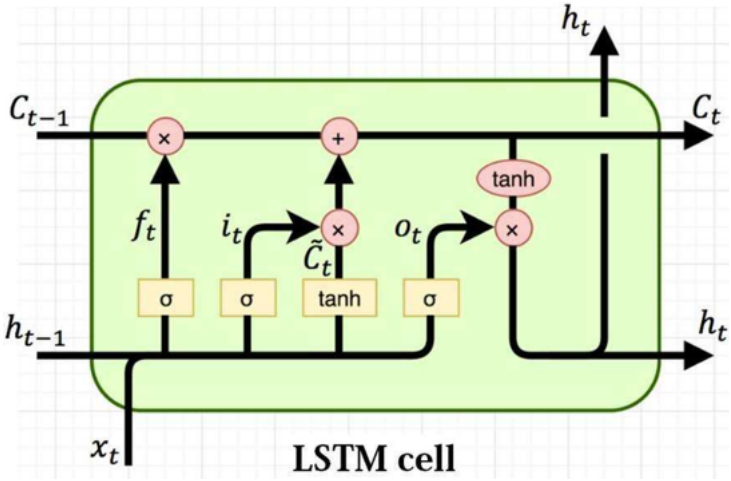


Figura 4.5: Ejemplo de cálculo de una célula en una red neuronal LSTM. Las puertas lógicas están representadas por los recuadros amarillos y constan de una combinación lineal de datos de entrada con pesos entrenables más una función de activación (σ para la función sigmoide y *tanh* para la tangente hiperbólica). C_t es el estado interno de la célula, que depende por una parte de la eliminación o no del estado anterior según el resultado f_t de la *forget gate*, y por otra de los candidatos a nuevo estado interno de la célula $\widetilde{C}_t$ resultado de la *input gate*. o_t es el resultado de la *output gate* que se combina con el nuevo estado interno C_t para dar el nuevo estado oculto h_t. Figura extraída de [23]

4.3. Aprendizaje y optimización

4.3.1. *Backpropagation* y optimizador

El proceso de *backpropagation* o retropropagación (ejemplo detallado en la figura 4.6) se encarga de optimizar los pesos del modelo para mejorar su rendimiento a partir del resultado de la función de pérdida, y su algoritmo concreto viene determinado por el optimizador utilizado. Se ha elegido Adam (*Adaptive Moment Estimation*) como optimizador ya que ajusta de forma dinámica el *learning rate* de cada parámetro y mejora el rendimiento de otros algoritmos como RMSProp (*Root Mean Square Propagation*) al añadir momentos de primer y segundo orden sobre el gradiente [24].

Normalmente no se realiza el cálculo de la pérdida para cada entrada de datos sino tras aplicar el *forward pass* a un número concreto de entradas, llamado lote o *batch*. Este cálculo se realiza sobre el conjunto de resultados de todas las entradas del *batch* (en nuestro caso con la

media de los valores absolutos de sus errores, sección 4.1), y una vez actualizados los pesos de la red se repite todo el proceso de aprendizaje sobre las entradas de datos del siguiente *batch*.

Una vez recorrido todo el dataset *Train* se considera terminado el aprendizaje para la época actual, y se utilizan los pesos finales de la red para evaluar la función de pérdida sobre los datos del dataset *Validation*. Esta pérdida en validación sirve para evaluar el aprendizaje del modelo sobre datos distintos, comparándose con la pérdida de *Train*, y verificando que el modelo generaliza correctamente y no cae en un ajuste excesivo a los datos concretos del entrenamiento, llamado *overfitting*. Así mismo el análisis de la pérdida en validación es utilizado para ajustar hiperparámetros del entrenamiento como el *learning rate*.

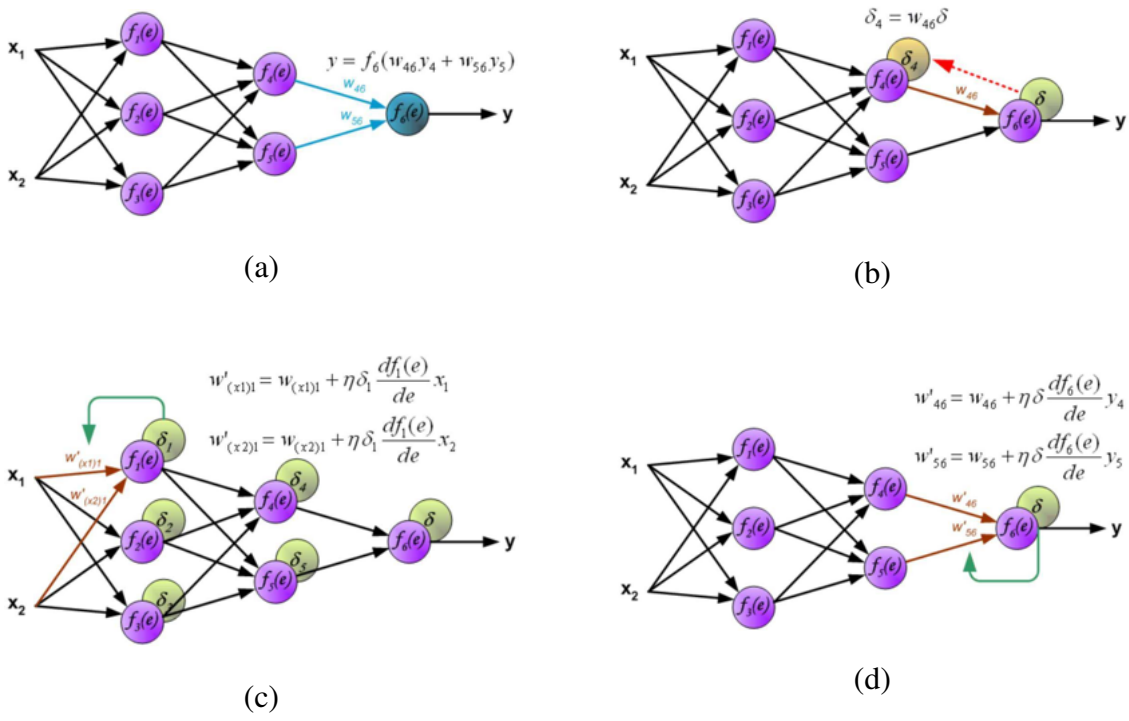


Figura 4.6: Ejemplo de desarrollo de la *backpropagation* para un modelo de capas densas: (a) se realiza el *forward pass* obteniendo un resultado final y a la salida de la última capa; (b) se compara mediante la función de pérdida la predición y con el target, obteniendo el error δ, y se calcula los δ_i de las neuronas previas a partir del error de las neuronas de la capa actual y de los pesos obtenidos durante el *forward pass*; (c) y (d) una vez calculados todos los δ_i se modifica de principio a fin los pesos entrenables a partir del error de cada neurona, de la derivada de su función de activación, y del *learning rate* η. Figuras extraídas de [25].

4.3.2. Hiperparámetros

Se llama hiperparámetros a aquellos parámetros que afectan al rendimiento del entrenamiento, siendo los más importantes el *learning rate* y el *batch size*. Un *learning rate* mayor implica que las modificaciones de los pesos durante la *backpropagation* son más pronunciadas, por lo que el modelo avanza a pasos más grandes en su aprendizaje pero a costa de tener un comportamiento más errático durante la minimización de la función de pérdida. Podemos ver este comportamiento en curvas de aprendizaje con picos pronunciados en vez de una progresión suave. Un *learning rate* bajo implica un avance más certero pero a la vez lento, lo que retrasa la convergencia del aprendizaje del modelo. Por su parte un *batch size* grande implica menos cálculos de *backpropagation* durante cada época, por lo que el tiempo de cómputo es menor, pero penalizando el aprendizaje al actualizar los pesos menos veces cada época. Inversamente, un *batch size* menor implica que el modelo aprende más durante cada época aumentando el tiempo de cómputo. Idealmente se busca la combinación de *learning rate* y *batch size* que reduzca el tiempo de entrenamiento y minimice la función de pérdida en mejor medida. Se puede hacer uso además de métodos como la modificación del *learning rate* durante el propio entrenamiento, favoreciendo saltos mayores en sus primeras épocas y reduciéndolos más adelante para afinar el aprendizaje. En nuestro caso hemos dejado en manos del optimizador el ajuste dinámico de este parámetro.

Al ser nuestro interés comparar distintas arquitecturas de modelos y no tanto hacer un estudio avanzando de hiperparámetros ideales, hemos optado por una búsqueda sencilla inicial y hemos dejado fijos estos hiperparámetros para todos los modelos, facilitando también la comparación, ajustando *learning rate* y *batch size* de manera que las curvas de aprendizaje fueran razonablemente suaves y a la vez convergentes en un número de épocas abarcable. Para todos los modelos hemos utilizado un *batch size* de 128 entradas y un *learning rate* de 10^{-5}, con la excepción de los modelos LSTM que mostraban un comportamiento plano inicial no convergente solventado aumentando ligeramente el *learning rate*.

4.3.3. Callbacks: *Early-stopping*, *Checkpoint*

En una situación ideal, para comparar el desempeño de distintos modelos entre sí, se les dejaría aprender todo el tiempo posible con la idea de obtener la mejor versión de cada modelo.

En una situación real hay que detener el entrenamiento en algún momento. Se ha hecho uso de la técnica *early-stopping* para ahorrar tiempo de cómputo innecesario durante la fase de desarrollo y testeo y para unificar criterio para decidir cuándo se termina el aprendizaje de cada modelo . Esta detención del aprendizaje se basa en los siguientes hiperparámetros:

- monitorización: métrica sobre la que establecemos las condiciones, en este caso la pérdida en validación (MAE);
- delta mínima: la cantidad (en este caso mínima) de la métrica monitorizada que debe minimizarse (en este caso) para no desencadenar la parada;
- paciencia: número de épocas seguidas durante las cuales debe cumplirse la condición anterior.

Además esta técnica permite recuperar los pesos de la mejor época (según la métrica elegida) para guardar la mejor versión del modelo. Por otra parte, se active o no el early-stopping, se ha hecho uso también de la técnica *checkpoint* de Keras, que guarda regularmente en un archivo el modelo en caso de mejorar la métrica elegida tras cada nueva época.

4.4. Estructura y resumen de modelos

Modelo Lineal

En la práctica se construye con una única capa densa de una sola neurona y una activación lineal. Se ha utilizado con el fin de comprobar la correcta estructura de los datos y el funcionamiento del *pipeline* sin introducir complejidad en el propio modelo, además de servir como versión neuronal entrenable del *Optimal Filtering*. Estos dos modelos servirán de referencia para comparar el desempeño del resto de redes. En nuestro caso el modelo lineal consta de 16 parámetros entrenables, un por cada peso de los 15 BC del enventanado más el término independiente de la combinación lineal.

Modelo Lineal

16 parámetros entrenables:

- capa densa de 1 neurona (linear)

Modelos MLP

Con afán de ahorrar recursos reducimos la cantidad de neuronas tras una primera capa abundante que permita suficientes combinaciones de pesos, y utilizamos *Rectified Linear Unit* (ReLU) [26] como función de activación para todas las capas salvo en la de salida, que no lleva activación para permitir cualquier valor como resultado final.

Modelos MLP

4031 parámetros entrenables:	1065 parámetros entrenables:
▪ capa densa de 80 neuronas (ReLU)	▪ capa densa de 30 neuronas (ReLU)
▪ capa densa de 30 neuronas (ReLU)	▪ capa densa de 15 neuronas (ReLU)
▪ capa densa de 10 neuronas (ReLU)	▪ capa densa de 7 neuronas (ReLU)
▪ capa de salida de 1 neurona	▪ capa de salida de 1 neurona

Modelos CNN

Se ha entrenado dos redes CNN distintas (ambas para cada versión de parámetros entrenables). Una de las redes mantiene el tamaño de los datos utilizando *padding*, una técnica que añade ceros a los extremos de la ventana de datos de tal forma que el filtro puede desplazarse un número de pasos igual a la dimensión de los datos (por tanto dando un resultado de igual tamaño). La segunda red CNN en cambio no hace uso de padding (y utiliza un tamaño de kernel mayor que 1), lo que deriva tras el paso de la convolución en una reducción del tamaño del dato a la salida de la capa. Otra consecuencia de no usar padding es que el kernel se desplaza menos veces sobre los datos más extremos de la ventana, que en nuestro caso son aquellos que suponemos tienen menor influencia en el *pileup* de señal sobre el BC central (por ser los más lejanos). Se pretende utilizar esta distinción en los modelos para testear esta hipótesis.

De nuevo hemos utilizado *Rectified Linear Unit* (ReLU) como función de activación para las capas convolucionales, y ninguna en la capa de salida.

Modelos CNN sin reducción de dimensionalidad

4326 parámetros entrenables:

- capa convolucional de 25 neuronas, kernel tamaño 6, stride de paso 1, cero-padding (ReLU)
- capa conv. 1D de 25 neuronas, kernel tamaño 6, stride de paso 1, cero-padding (ReLU)
- capa flatten
- capa de salida de 1 neurona

1053 parámetros entrenables:

- capa convolucional de 12 neuronas kernel tamaño 6, stride de paso 1, cero-padding (ReLU)
- capa convolucional de 11 neuronas, kernel tamaño 6, stride de paso 1, cero-padding (ReLU)
- capa flatten
- capa de salida de 1 neurona

Modelos CNN con reducción de dimensionalidad

4051 parámetros entrenables:

- capa convolucional de 30 neuronas, kernel tamaño 7, stride de paso 1 (ReLU)
- capa convolucional de 30 neuronas, kernel tamaño 4, stride de paso 1 (ReLU)
- capa flatten
- capa de salida de 1 neurona

1059 parámetros entrenables:

- capa convolucional de 15 neuronas kernel tamaño 7, stride de paso 1, (ReLU)
- capa convolucional de 14 neuronas, kernel tamaño 4, stride de paso 1, (ReLU)
- capa flatten
- capa de salida de 1 neurona

Modelos LSTM

Se ha entrenado dos modelos LSTM diferentes, uno unidireccional y otro bidireccional, utilizando las clases correspondientes de *Keras* en *Tensorflow*. El segundo modelo se distingue del primero en que los datos (secuencias) se leen en ambos sentidos, hacia delante y hacia detrás, lo cual puede ayudar en casos en los que la regresión a partir de un dato dependa de los siguientes datos de la ventana, como ocurre en nuestro caso. Se espera que el modelo bidireccional pueda funcionar mejor ya que tener un BC muy energético justo después del central afecta a la digitalización de su señal. Se ha utilizado la tangente hiperbólica como función de activación (en lugar de ReLU) para la salida final de la célula LSTM, ya que es la única optimizada para ser ejecutada en GPU (en nuestro caso el tiempo de entrenamiento se reduce en un factor 5 frente a usar ReLU en CPU). La capa de salida no lleva función de activación.

Unicamente los modelos recurrentes LSTM han sido entrenados en GPU, el resto de redes han sido entrenadas en CPU debido principalmente a su mayor velocidad y también a la memoria disponible.

Modelos LSTM unidireccionales

4291 parámetros entrenables:	1107 parámetros entrenables:
▪ capa LSTM de 30 neuronas, retorno de los 15 estados ocultos (tanh)	▪ capa LSTM de 14 neuronas, retorno de los 15 estados ocultos (tanh)
▪ capa flatten	▪ capa flatten
▪ capa de salida de 1 neurona	▪ capa de salida de 1 neurona

Modelos LSTM bidireccionales	
4121 parámetros entrenables: ■ capa Bi-LSTM de 20 neuronas, retorno de los 15 estados ocultos (tanh) ■ capa flatten ■ capa de salida de 1 neurona	1063 parámetros entrenables: ■ capa Bi-LSTM de 9 neuronas, retorno de los 15 estados ocultos (tanh) ■ capa flatten ■ capa de salida de 1 neurona

4.5. Entrenamientos

Las curvas de aprendizaje mostradas en esta sección corresponden a HG para la configuración con los datos filtrados sin pesar, para la versión con $\sim 4 \cdot 10^3$ parámetros entrenables, y para el dataset de *Train* con tamaño 6M de entradas. Se incluye en el apéndice una muestra del aprendizaje del modelo MLP con datos pesados según la tangente hiperbólica donde se aprecia la diferencia en la pérdida de *Train* por la aplicación del pesado. La figura contienen una segunda gráfica con escala logarítmica utilizada para apreciar mejor visualmente la tendencia de aprendizaje del modelo en el momento de detener el entrenamiento. Las curvas de esta sección se muestran únicamente con el eje de la pérdida en escala lineal. Las gráficas de aprendizaje contienen una indicación de la época con los mejores pesos guardados antes del *early-stopping*.

Modelo lineal

Para el modelo lineal se han utilizado los mismos hiperparámetros para HG y LG.

Hiperparámetros:

- batch size = 128
- learning rate = 10^{-5}
- Early-stopping:
  - min delta = 10^{-6}
  - patience = 5

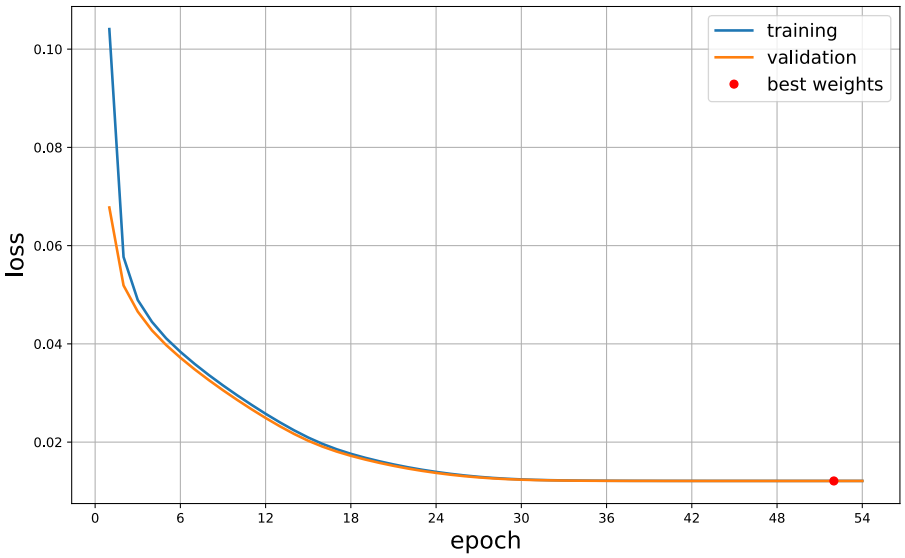


Figura 4.7: Curva de aprendizaje para el modelo lineal.

Perceptrón multicapa (*Multi-Layer Perceptron*, MLP)

Para el modelo MLP se han utilizado los mismos parámetros para HG y LG.

Hiperparámetros:

- batch size = 128
- learning rate = 10^{-5}
- Early-stopping:
  - min delta = 10^{-6}
  - patience = 5

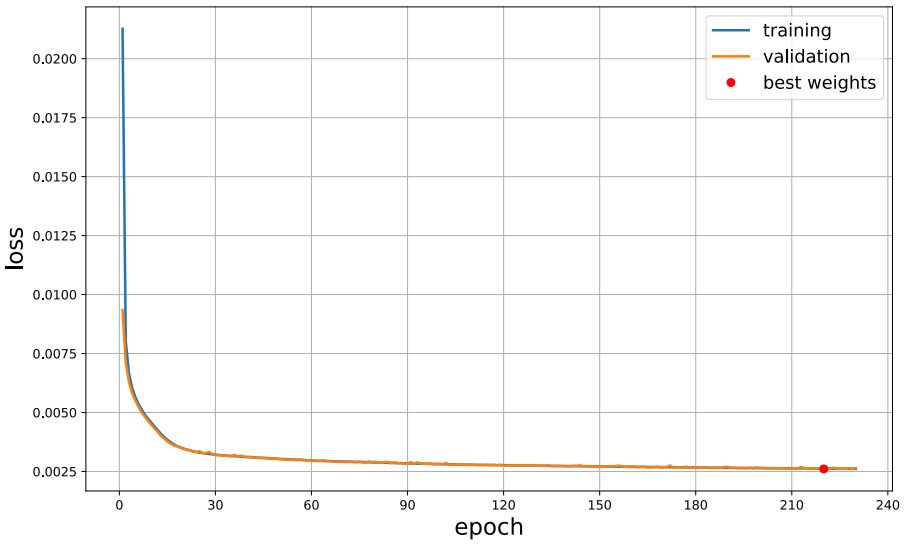


Figura 4.8: Curva de aprendizaje para el modelo MLP.

Red neuronal convolucional (*Convolutional Neural Network*, CNN)

Para los modelos CNN se han utilizado los mismos parámetros para HG y LG. Se muestra la gráfica para el modelo sin reducción de dimensionalidad.

Hiperparámetros:

- batch size = 128
- learning rate = 10^{-5}
- Early-stopping:
  - min delta = 10^{-6}
  - patience = 5

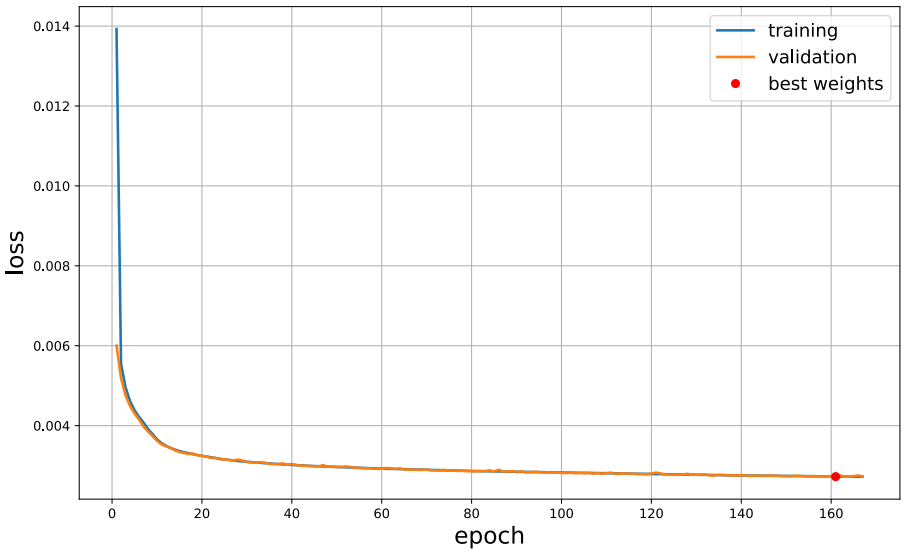


Figura 4.9: Curva de aprendizaje para el modelo CNN sin reducción de dimensionalidad.

Modelo *Long Short-Term Memory* (LSTM)

Para los modelos LSTM se han utilizado distintos parámetros para HG y LG. Se muestra la gráfica para el modelo unidireccional.

Hiperparámetros:

- batch size = 128
- learning rate HG = $2 \cdot 10^{-5}$
- learning rate LG = $4 \cdot 10^{-5}$
- Early-stopping:
  - min delta = 10^{-6}
  - patience HG = 8
  - patience LG = 10

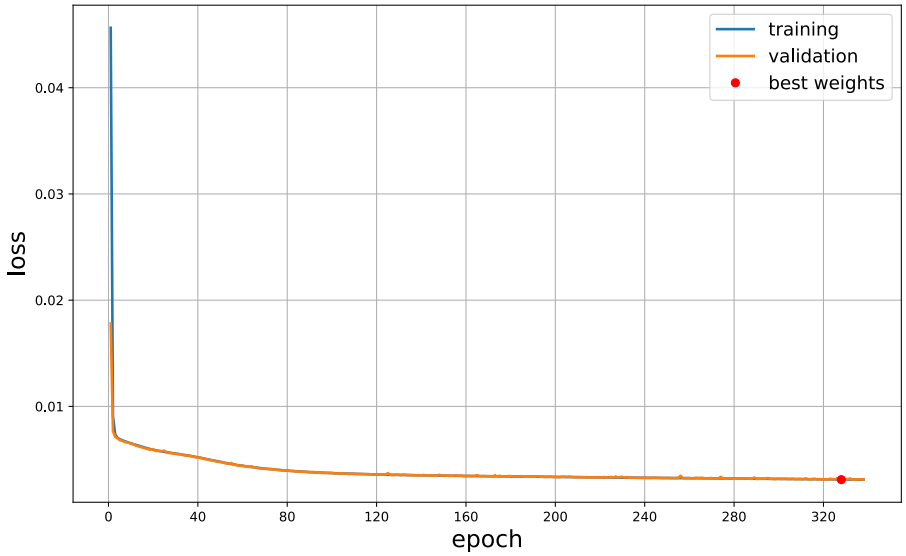


Figura 4.10: Curva de aprendizaje para el modelo LSTM unidireccional.

Capítulo 5

Resultados

En las figuras 5.1 y 5.2 se puede ver una muestra de las reconstrucciones de energías para distintos modelos junto con las energías reales, para HG con los datos sin pesar, y LG con datos filtrados, respectivamente. Se puede apreciar como los modelos de redes neuronales generan reconstrucciones de energía por lo general más cercanas al valor real que el modelo Lineal y significativamente mejores que el algoritmo *Optimal Filtering*. También se constata que únicamente las redes neuronales (excluido el modelo Lineal) son capaces de aprender que no puede haber energías negativas.

5.1. Error absoluto medio (MAE) y desviación estándar σ

Se ha calculado el error absoluto medio (MAE) (ecuación 4.1) y la desviación estándar σ de cada predicción (ecuación 5.1) para cada modelo y configuración sobre el conjunto completo de datos de *Test*.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}} \tag{5.1}$$

En la tabla 5.1 se muestra las MAEs y desviaciones estándar totales para los modelos en-

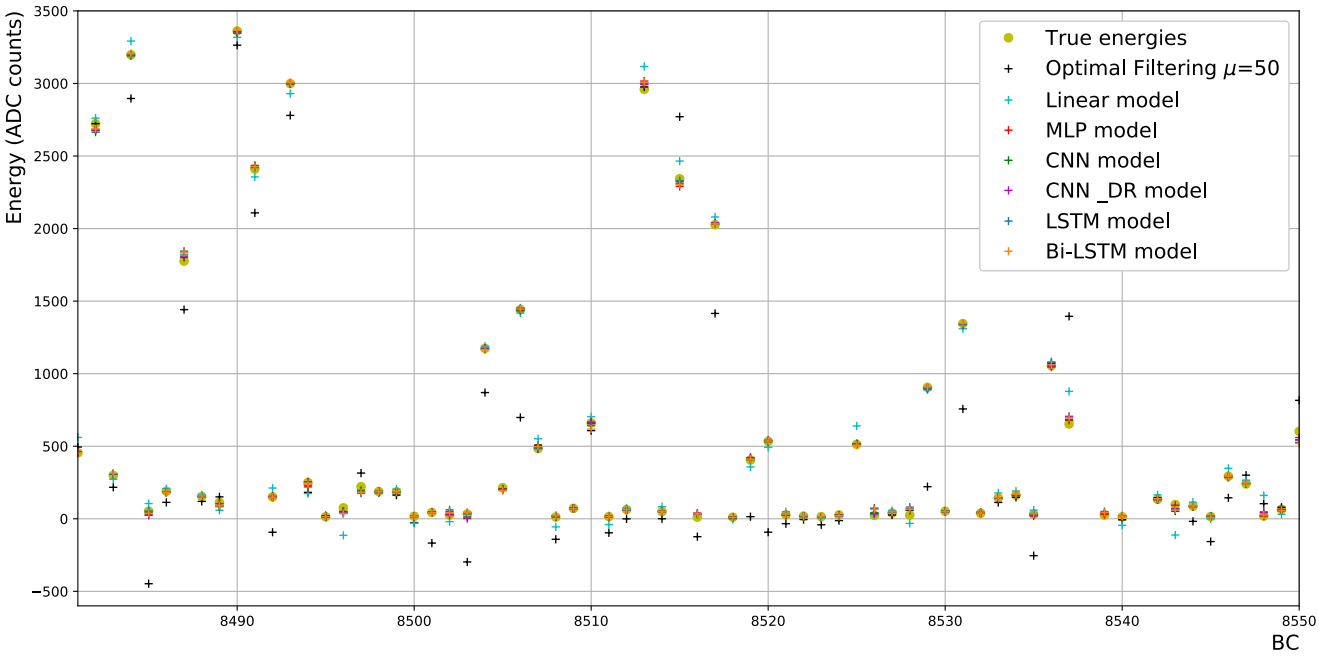


Figura 5.1: Pequeña muestra de energías reconstruidas y energías reales para modelos entrenados con datos de HG sin pesar.

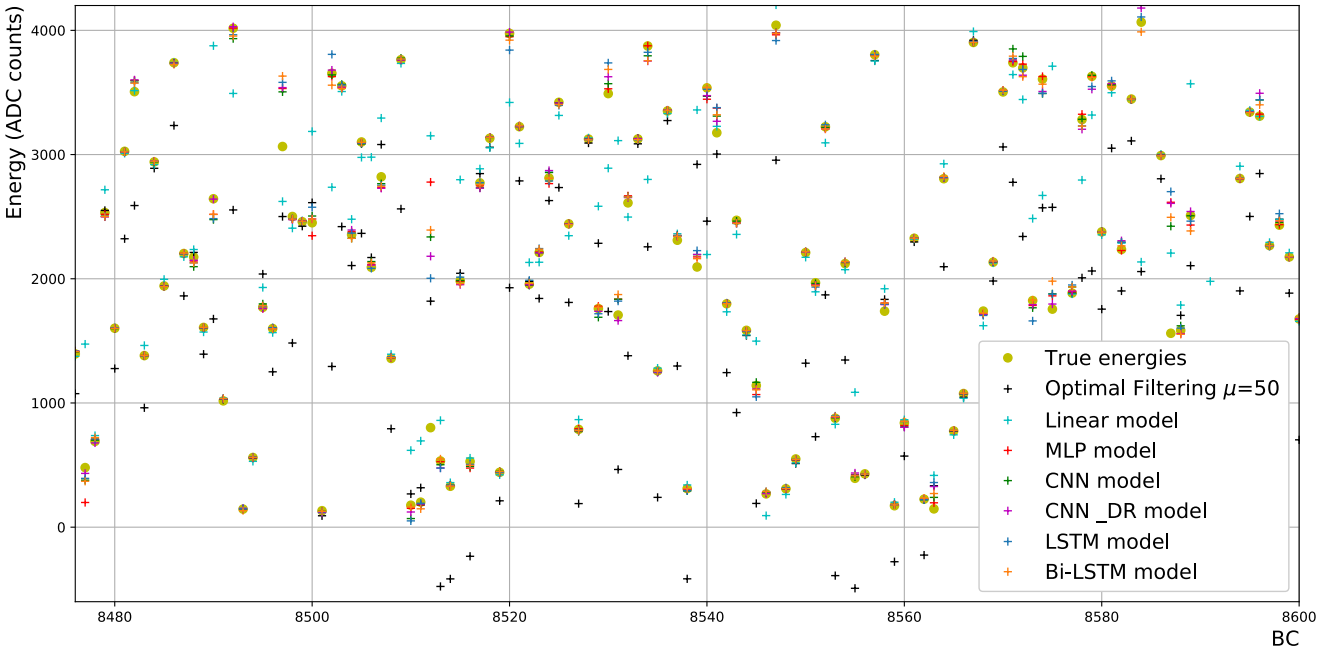


Figura 5.2: Pequeña muestra de energías reconstruidas y energías reales para modelos entrenados con datos de LG filtrados.

trenados sobre datos de HG con los tres tipos de pesado y para LG con datos filtrados y sin filtrar.

	HG						LG			
	sin pesar		pesado tanh		pesado exp		sin filtrar		filtrado	
Modelo	MAE	σ	MAE	σ	MAE	σ	MAE	σ	MAE	σ
OF μ=50	240	397	240	397	240	397	532	631	603	673
Lineal	49.5	75.9	53.4	81.9	63.8	83.9	134	314	242	452
MLP	**10.7**	21.3	12.4	24.4	14.4	27.5	13.0	89.0	43.5	140
CNN	11.1	21.2	13.3	24.6	15.0	26.9	16.3	89.5	48.3	148
CNN DR	11.0	**21.1**	13.2	24.8	14.7	26.2	16.3	94.9	48.0	146
LSTM	12.7	23.3	18.8	32.3	22.6	37.8	16.4	89.5	50.1	144
Bi-LSTM	13.0	23.7	19.8	34.4	26.4	43.7	16.8	86.0	43.3	129

Tabla 5.1: Error absoluto medio (MAE) y desviación estándar σ para los modelos con $\sim 4 \cdot 10^3$ parámetros entrenables, evaluados sobre el dataset de Test.

El modelo Lineal muestra resultados sustancialmente mejores que el algoritmo *Optimal Filtering*, disminuyendo la MAE total en un rango de 73-79 % para HG y la desviación estándar en un rango de 79-81 %, según el tipo de pesado. El resto de redes neuronales mejoran estos resultados sistemáticamente, en rangos de 89-96 % de reducción de MAE y 89-95 % de reducción de desviación estándar para datos de HG, según el tipo de pesado. De entre los modelos de redes neuronales MLP y las dos versiones de CNN obtienen sistemáticamente los mejores resultados en HG. Respecto de las métricas totales el entrenamiento con datos de HG sin pesar obtienen mejores resultados, pero no son los más indicados ya que la descompensación en la distribución de energías puede falsear la interpretación. Por ello en la sección 5.1.1 analizamos estas métricas en distintos tramos de energía.

Los modelos entrenados con datos de LG presentan resultados dispares. Por una parte muestran el mismo comportamiento entre modelos, siendo el Lineal mejor que *Optimal Filtering*, y el resto de redes neuronales sustancialmente mejores que los dos anteriores. En cambio globalmente todos los métodos presentan peores resultados entrenados con datos de LG filtrados frente a datos de HG, con MAEs del orden de 4 veces mayores, y desviaciones estándar del orden de 6-7 veces mayores. Un nuevo entrenamiento con datos de LG sin filtrar muestra MAEs mucho menores (mucho más cercanas a las obtenidas con HG) pero desviaciones estándar todavía

grandes (del orden de 4-5 veces). Se reproduce en la figura 5.3 una muestra de las predicciones de este nuevo entrenamiento.

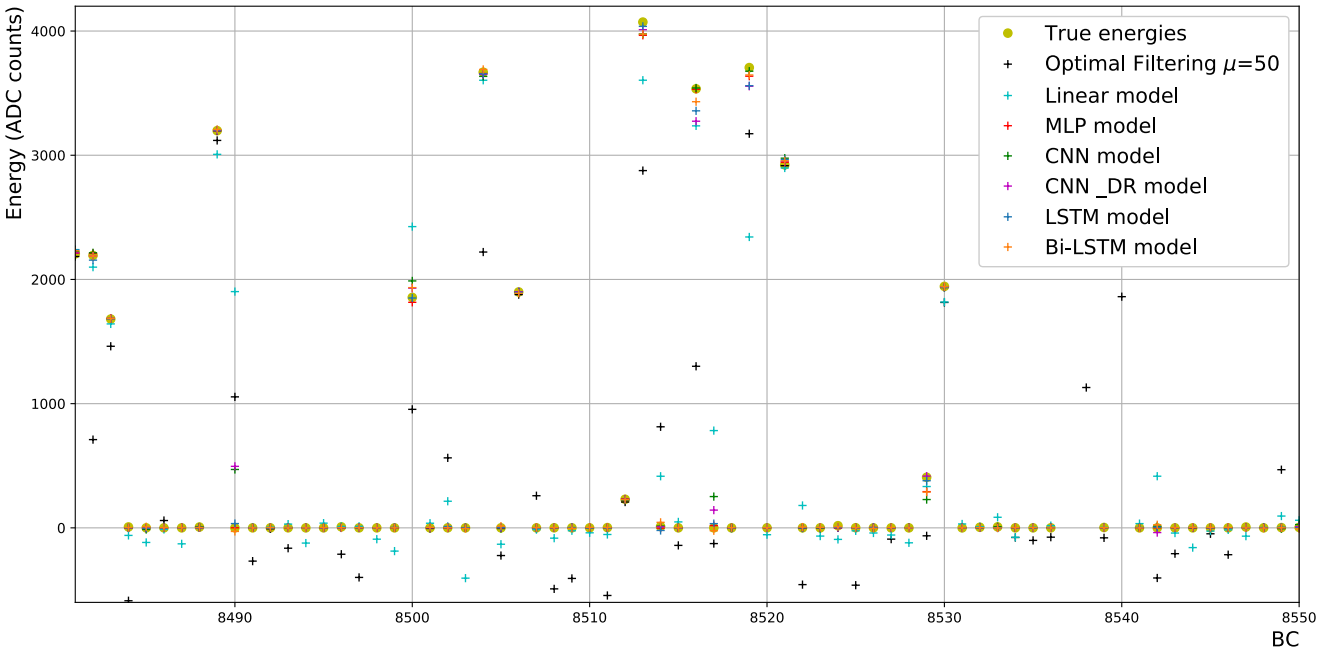


Figura 5.3: Pequeña muestra de energías reconstruidas y energías reales para modelos entrenados con datos de LG sin filtrar.

Cabe mencionar que se esperaba obtener mejores resultados con modelos entrenados con datos de LG que de HG ya que el ruido electrónico y el *pileup* son menores (debido a su menor amplificación por la ganancia). Se sospecha que la razón puede estar en la distribución de los datos resultado de su selección según la ganancia y se propone para futuros pasos en el proyecto entrenar un solo modelo con datos de HG y LG juntos en vez de por separado.

5.1.1. MAE y σ por tramos

Debido al pesado de datos en HG esperamos que los modelos con datos sin pesar (con un sesgo no corregido en favor de energías bajas) obtengan una MAE total menor al resultar más fácil predecir energías cercanas a 0 ADC *counts*, a costa de obtener peores resultados para el resto de rango de energías. Para intentar esclarecer la influencia del pesado sobre los datos analizamos el rendimiento de los modelos por distintos tramos de energía, mostrados en la figura 5.4.

Se visualiza de nuevo que los modelos de redes neuronales funcionan mucho mejor que

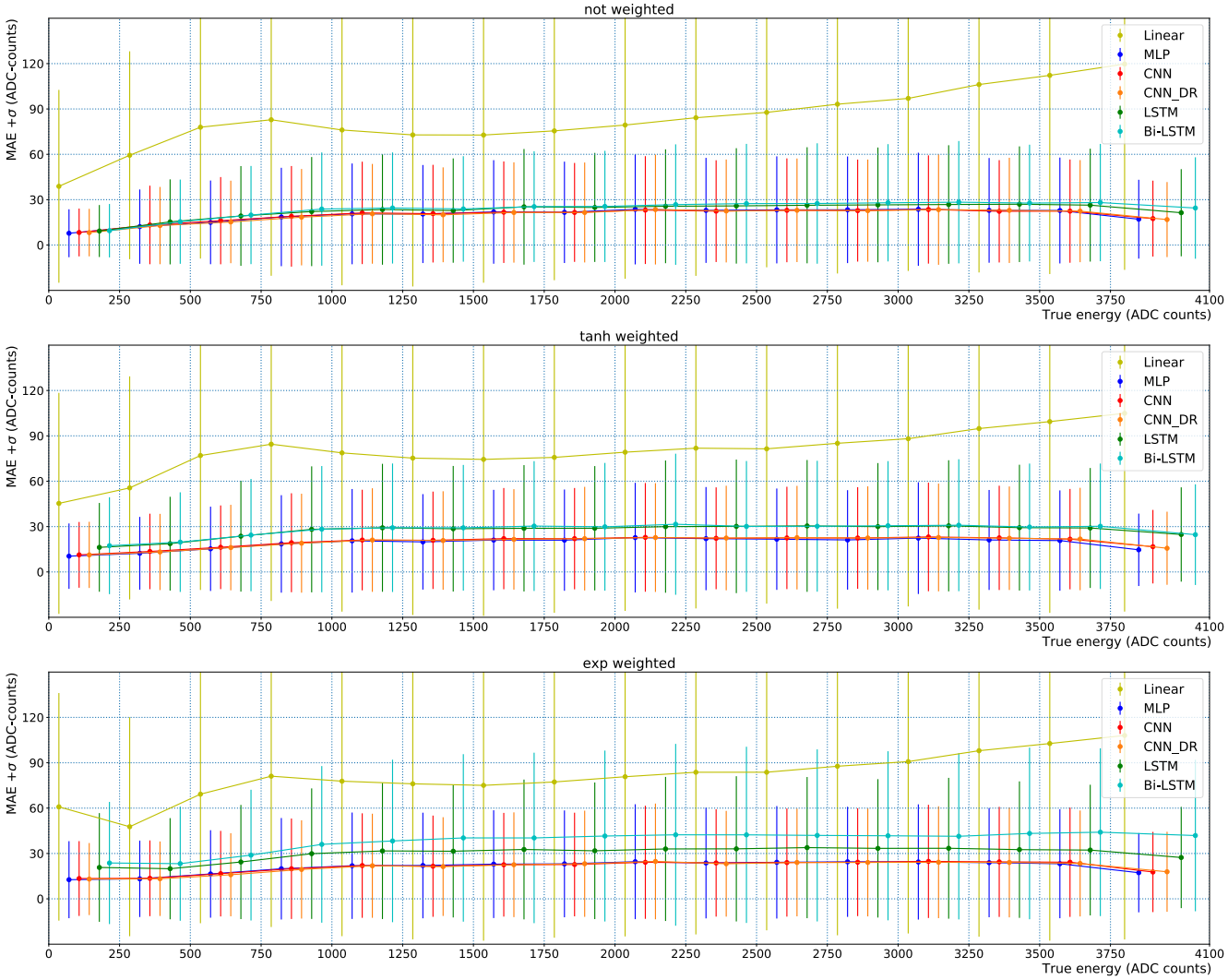


Figura 5.4: MAE y su desviación estándar por tramos para modelos entrenados con datos de HG según el tipo de pesado.

el modelo Lineal. Los modelos de redes recurrentes parecen empeorar a nivel global para los dos tipos de pesado de datos, además de obtener sistemáticamente peores resultados que los modelos MLP y CNN. Estas últimas redes neuronales, salvo para energías muy bajas, parecen mejorar ligeramente sus rendimientos. Comparamos en la tabla 5.2 los resultados numéricos de las métricas para el modelo MLP en dichos tramos de energía.

Se confirma mediante la tabla 5.2 que para energías superiores a 1250 ADC *counts* el modelo entrenado con datos pesados según la tangente hiperbólica obtiene de forma sistemática resultados ligeramente mejores, tanto en MAE como en desviación estándar. Este es el único

HG pesado tanh ($\sim 4 \cdot 10^3$ parámetros)								
Rango (ADC)	0-250		250-500		500-750		750-1000	
Pesado	MAE	σ	MAE	σ	MAE	σ	MAE	σ
sin pesar	**7.8**	**16.1**	**12.3**	24.8	**15.3**	28.3	**18.3**	**31.6**
tanh	10.5	21.7	12.4	**24.5**	15.5	**28.1**	**18.3**	32.0
exp	12.7	25.6	13.3	25.7	16.6	29.1	19.9	33.3
Rango (ADC)	1000-1250		1250-1500		1500-1750		1750-2000	
Pesado	MAE	σ	MAE	σ	MAE	σ	MAE	σ
sin pesar	20.5	**33.9**	20.6	32.9	21.5	33.9	21.8	33.9
tanh	**20.2**	34.0	**20.0**	**32.4**	**21.0**	**33.6**	**21.0**	**33.5**
exp	21.9	35.4	22.2	35.1	22.9	35.9	23.3	36.1
Rango (ADC)	2000-2250		2250-2500		2500-2750		2750-3000	
Pesado	MAE	σ	MAE	σ	MAE	σ	MAE	σ
sin pesar	22.8	34.7	23.2	35.4	23.2	35.8	23.5	35.9
tanh	**21.7**	**34.0**	**21.9**	**34.6**	**21.7**	**34.5**	**21.6**	**34.0**
exp	24.0	36.7	24.2	37.2	24.0	36.9	24.1	36.6
Rango (ADC)	3000-3250		3250-3500		3500-3750		3750-4095	
Pesado	MAE	σ	MAE	σ	MAE	σ	MAE	σ
sin pesar	23.6	36.3	22.8	34.7	22.2	33.9	17.6	27.2
tanh	**21.8**	**34.3**	**20.8**	**32.5**	**19.6**	**31.1**	**15.5**	**25.0**
exp	24.1	37.1	23.1	35.3	22.2	34.2	17.8	27.7

Tabla 5.2: MAE y desviación estándar en distintos tramos de energía del modelo MLP con $\sim 4 \cdot 10^3$ parámetros entrenado con datos de HG según su pesado.

de los dos tipos de filtrado que consigue mejorar las métricas. Utilizaremos este entrenamiento como base para los siguientes análisis, mostrando las gráficas para el resto de pesados en el apéndice.

5.2. Energía reconstruida frente a energía real

Se muestra en las figuras 5.5, 5.6 y 5.7 las energías reconstruidas frente a las reales, la media de las reconstrucciones y su desviación estándar por tramos de energía, y una regresión lineal a partir de las medias. La escala de color muestra la densidad puntos. Las escalas de los ejes y de color están igualadas para poder compararse los modelos entre sí y entre HG y LG, y para poder mostrarse los rangos más representativos de todos los modelos a la vez (los valores más extremos de dispersión y densidad de puntos pueden no verse reflejados en ciertas gráficas).

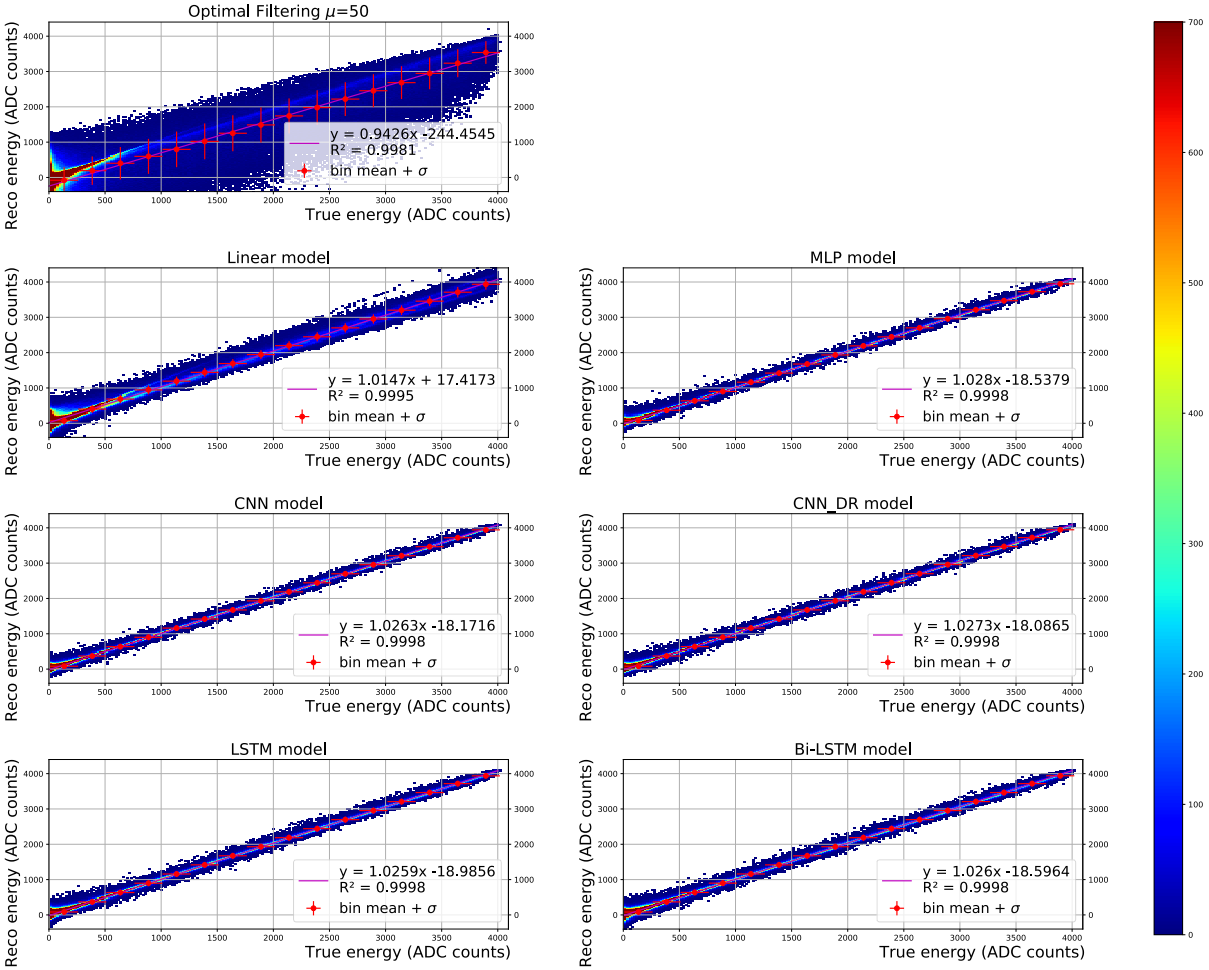


Figura 5.5: Energía reconstruida frente a energía real para modelos HG con $\sim 4 \cdot 10^3$ parámetros entrenables y datos pesados con la tangente hiperbólica.

Se visualiza en la figura 5.5 la mejora obtenida por los modelos entrenados. La dispersión de energías reconstruidas por las redes neuronales (salvo para el modelo Lineal) es mucho me-

nor que para el algoritmo *Optimal Filtering*, favoreciendo aparentemente las redes CNN y MLP sobre las recurrentes, y tienen un buen ajuste a una recta. En concreto se mejora el anterior resultado obtenido en el estudio de Arciniega et al. [3] para el coeficiente de determinación cuadrático, pasando de R^2=0.984 a R^2=0.9998, aunque empeorando principalmente la determinación del origen pasando de 3.5 ADC a 18.1 ADC en el mejor de los casos para energías nulas.

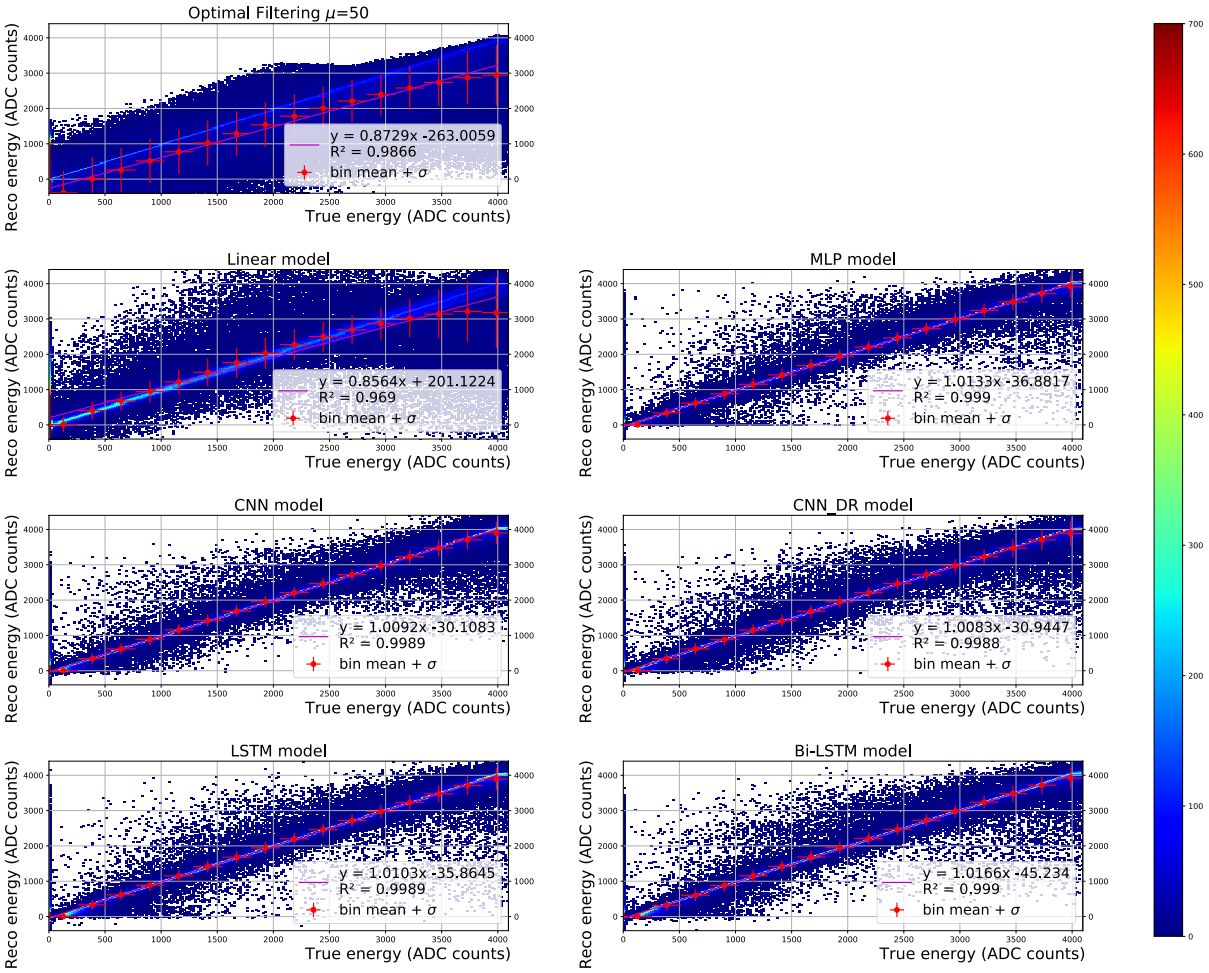


Figura 5.6: Energía reconstruida frente a energía real para modelos LG $\sim 4 \cdot 10^3$ parámetros entrenables y datos sin filtrar.

En el caso de modelos entrenados con datos de LG se observa distintos comportamientos (figuras 5.6 y 5.7). La mayoría de puntos (densidad en las gráficas) en la figura 5.6 se concentra en las energías más bajas. Esto es debido a la mayor población de energías en ese rango (figura 3.3 (a) True energies, LG sim); esto no ocurre en las gráficas de la figura (figura 5.7) ya que esos valores han sido filtrados. Por esta misma razón se observa en la figura 5.6 una tendencia

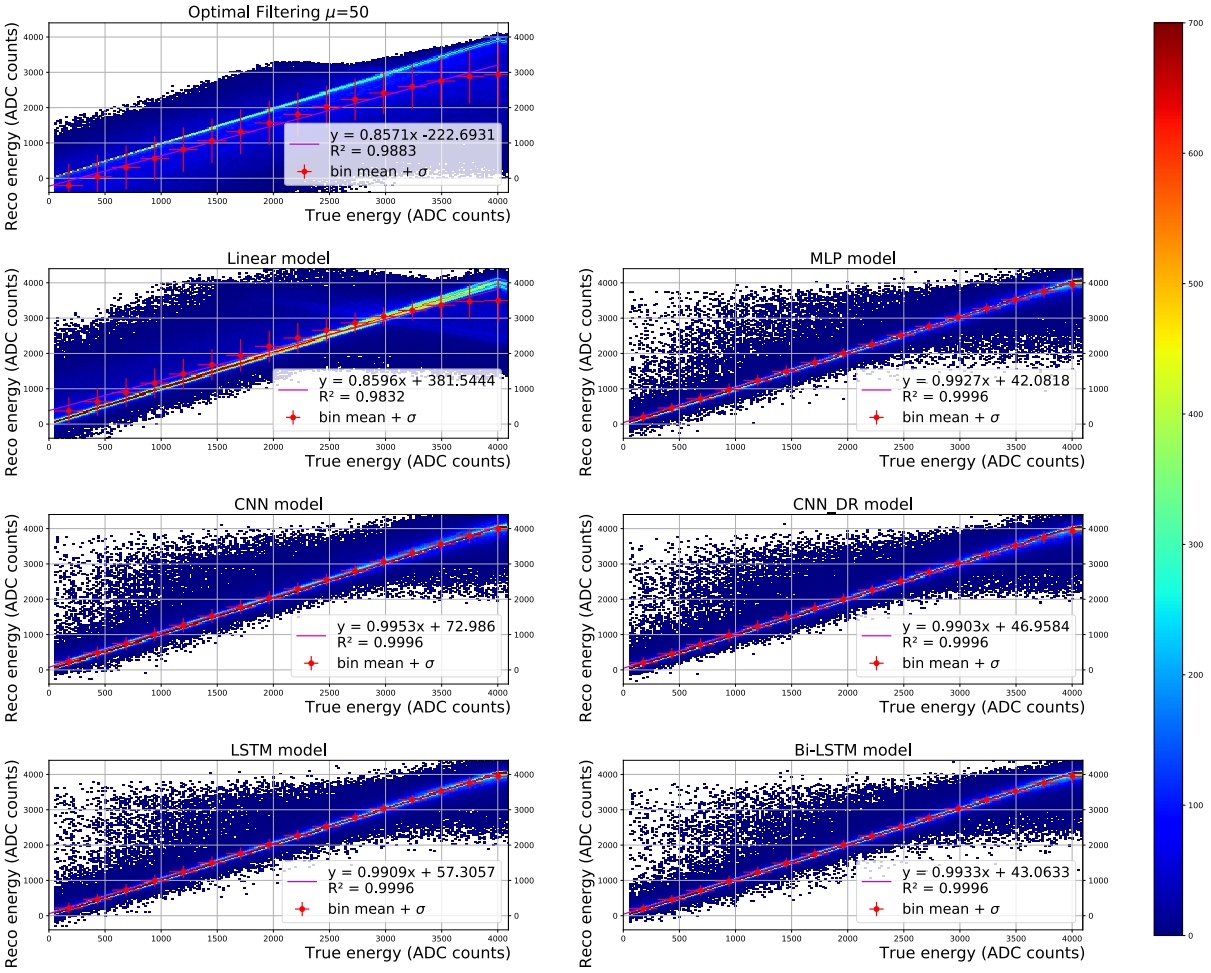


Figura 5.7: Energía reconstruida frente a energía real para modelos LG $\sim 4 \cdot 10^3$ parámetros entrenables y datos filtrados.

a predecir energías nulas en el rango 0-1000 ADC de *true energies*, que desaparece en la figura 5.7 cuando filtramos esos datos. Para energías mayores se observa que los ajustes a una recta son buenos a excepción de la ordenada en el origen, aunque en comparación con modelos entrenados en HG se aprecia una mayor dispersión en las energías reconstruidas. En el caso de la figura 5.7 se aprecia una tendencia descendente de las predicciones a altas energías debido a la saturación de los *samples* de energía en el límite superior de 4095 ADC *counts*, lo que impide a los modelos aprender a reconstruir estas energías correctamente.

5.3. Error relativo

Se ha calculado el error relativo a partir de las energías reconstruidas y reales como

$$\epsilon_{rel} = \frac{E_{reco} - E_{true}}{E_{true}} \tag{5.2}$$

En las gráficas 5.8, 5.9 y 5.10 se muestran los errores relativos de las predicciones, además de su media y su desviación estándar por tramos. La escala de color muestra la densidad de puntos.

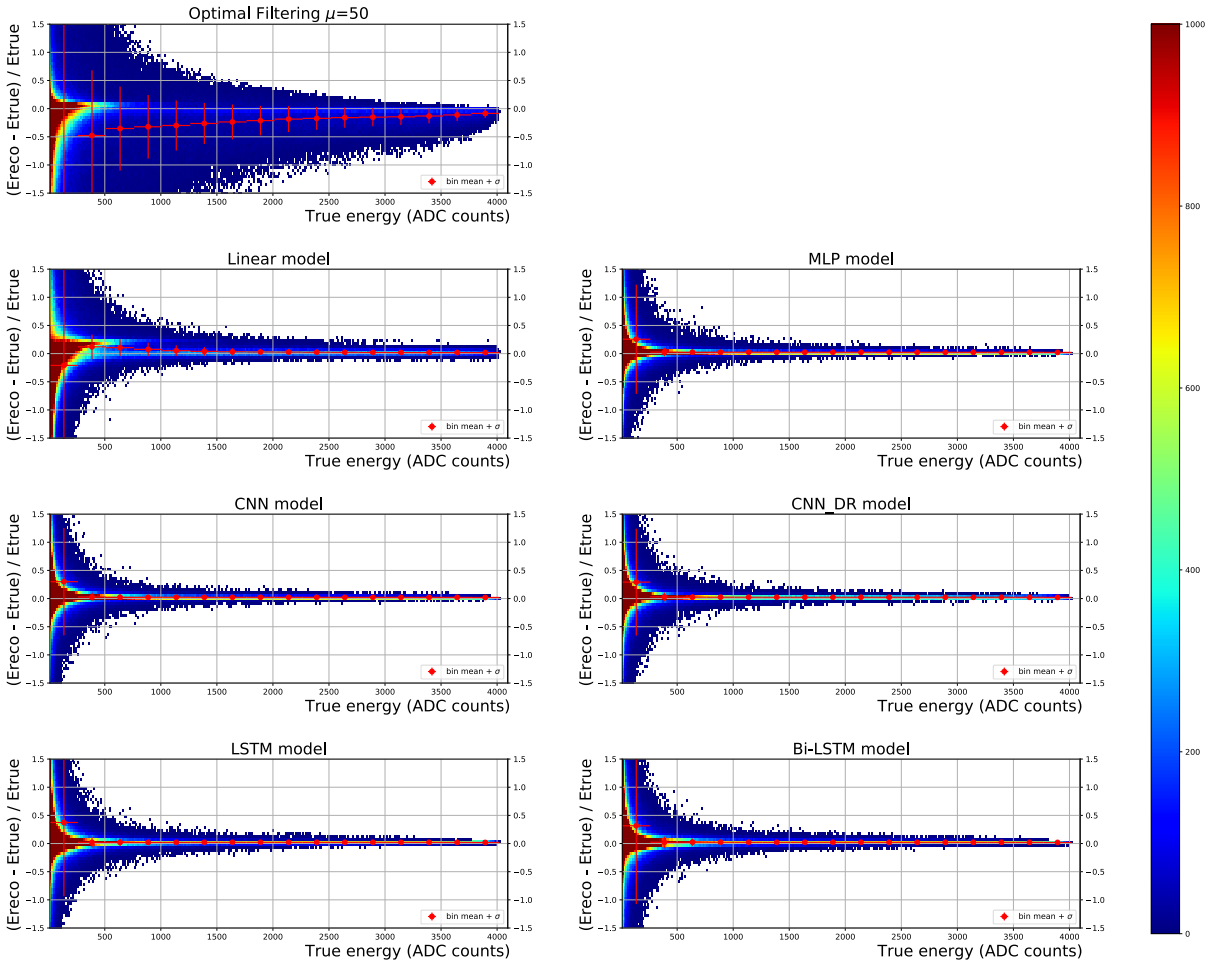


Figura 5.8: Error relativo para modelos HG $\sim 4 \cdot 10^3$ parámetros entrenables y datos pesados con la tangente hiperbólica.

Se verifica de nuevo en la figura 5.8 el mejor rendimiento de los modelos de redes neuronales frente a *Optimal Filtering* y frente al modelo Lineal, y una aparente menor dispersión en los

modelos CNN respecto del resto.

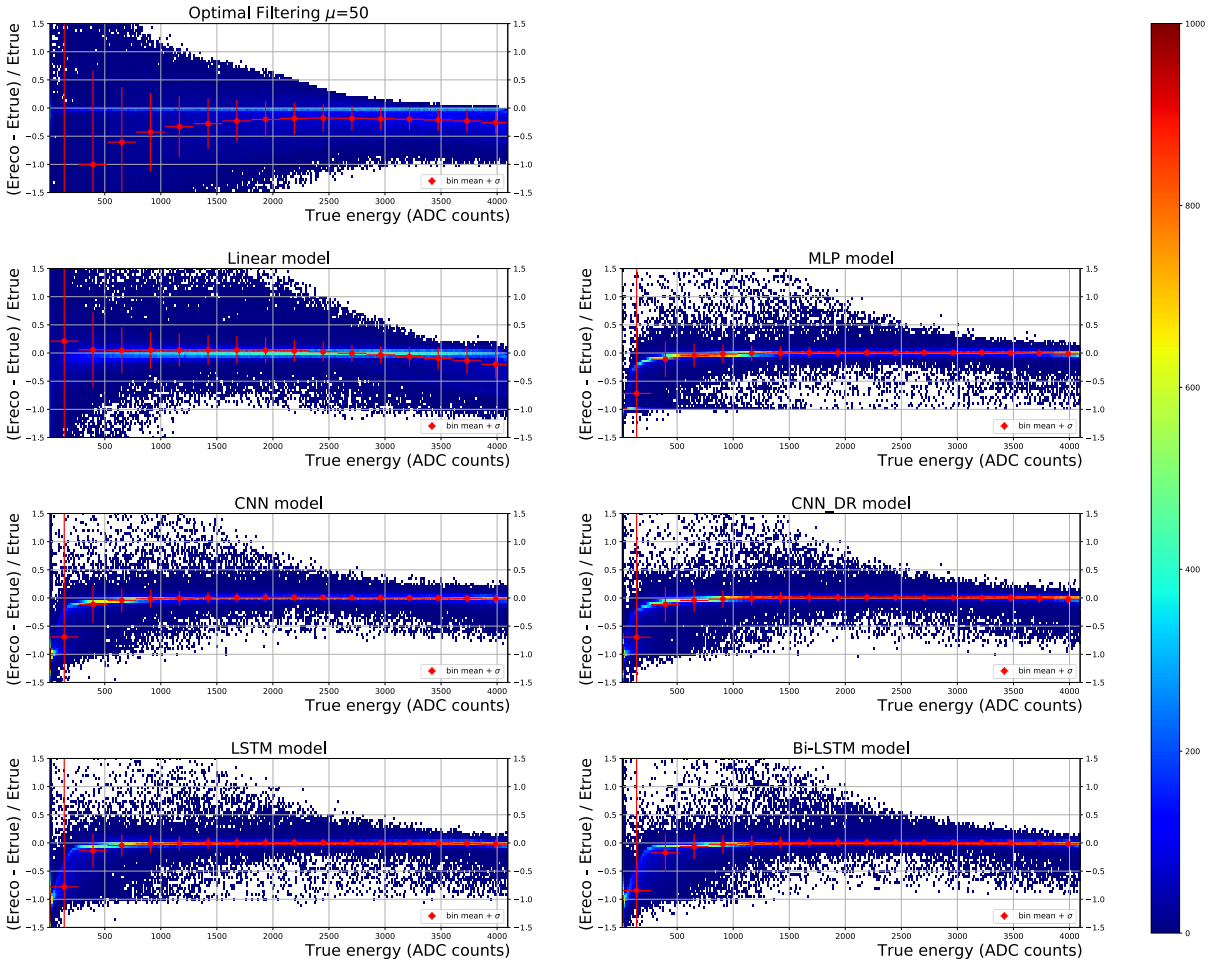


Figura 5.9: Error relativo para modelos LG $\sim 4 \cdot 10^3$ parámetros entrenables y datos sin filtrar.

Se aprecia en la figura 5.9 una tendencia para bajas energías a mostrar un error relativo cercano a $\epsilon = -1$ que corresponde con una predicción de energía de 0 ADC *counts*. Al igual que en la figura 5.6 este comportamiento se mantiene hasta los ~1000 ADC *counts*, en menor medida para el modelo CNN. Este comportamiento desaparece en la figura 5.10 al filtrar las energías bajas. Se aprecia también la tendencia a altas energías a predecir valores inferiores al real (aquí visualizado en un mayor número de errores relativos negativos). Aparece un nuevo fenómeno en los modelos entrenados con datos de LG (de forma más visible con datos filtrados en la figura 5.10) a energías medias y bajas, consistente en una curva hacia errores relativos altos, diferenciada de las colas típicas de bajas energías (que también aparecían en modelos de HG en la figura 5.5). Esta gran dispersión de errores relativos podría ayudar a esclarecer las peores

métricas en modelos de LG y se propone estudiarla más en profundidad en los siguientes pasos del proyeto.

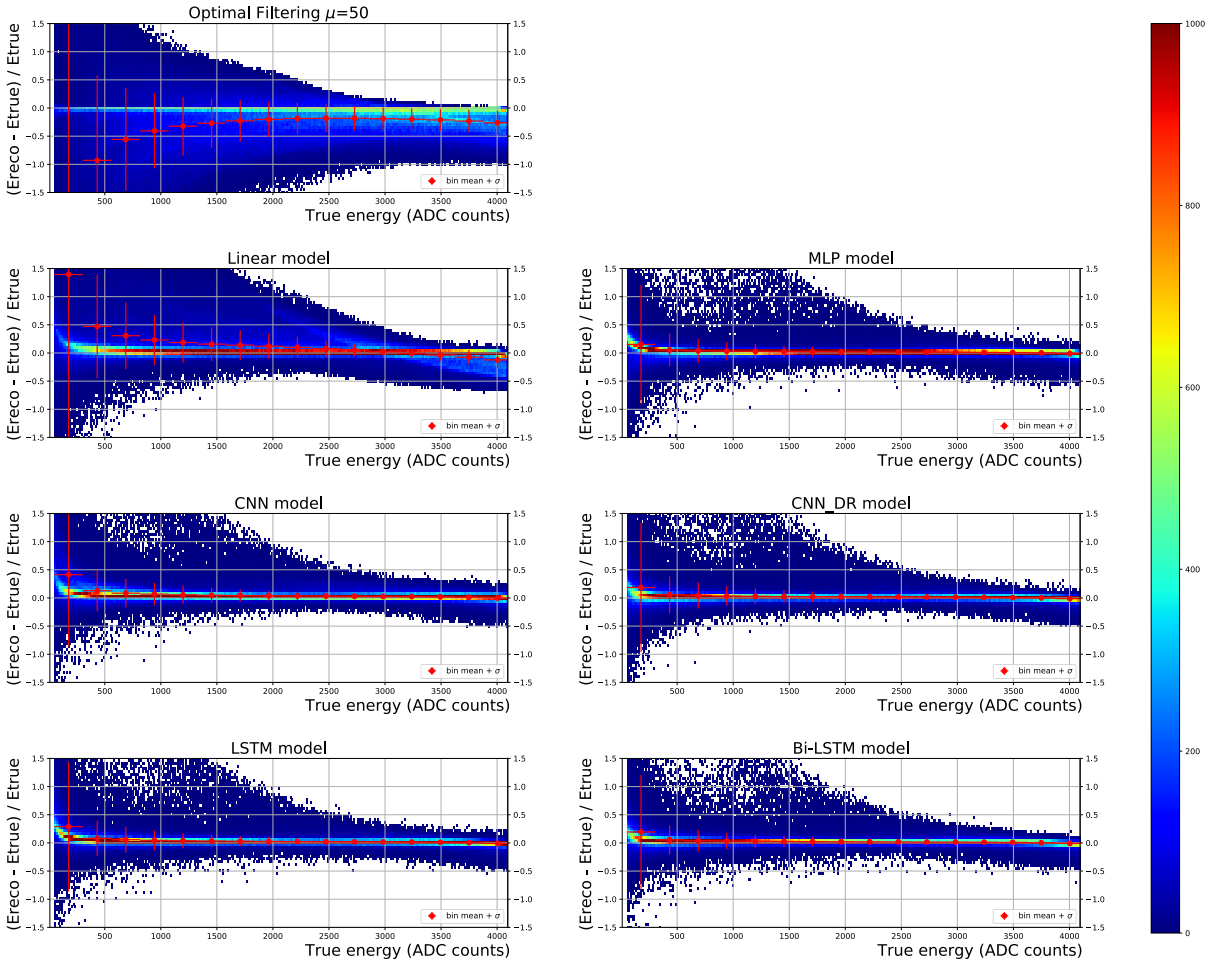


Figura 5.10: Error relativo para modelos LG $\sim 4 \cdot 10^3$ parámetros entrenables y datos filtrados.

5.4. Reducción de parámetros

Uno de los objetivos del proyecto es implementar de forma eficiente los modelos entrenados en tarjetas FPGA. En la tabla 5.3 se muestran los errores medios absolutos (MAE) y la desviación típica de las reconstrucciones comparando los modelos con en torno a $4 \cdot 10^3$ parámetros entrenables y con 10^3. No se muestra el modelo Lineal ya que únicamente se ha entrenado con 16 parámetros entrenables.

	HG						**LG**			
	sin pesar		pesado tanh		pesado exp		sin filtrar		filtrado	
Modelo	MAE	σ	MAE	σ	MAE	σ	MAE	σ	MAE	σ
MLP ($4 \cdot 10^3$)	**10.7**	**21.3**	**12.4**	**24.4**	**14.4**	**27.5**	**13.0**	**89.0**	**43.5**	**140**
MLP (10^3)	12.7	24.6	15.8	32.0	17.4	33.4	15.9	102	54.4	168
CNN ($4 \cdot 10^3$)	**11.1**	**21.2**	**13.3**	**24.6**	**15.0**	**26.9**	**16.3**	**89.5**	**48.3**	**148**
CNN (10^3)	12.6	24.0	17.9	34.3	16.4	29.1	25.7	131	65.7	175
CNN DR ($4 \cdot 10^3$)	**11.0**	**21.1**	**13.2**	**24.8**	**14.7**	**26.2**	**16.3**	**94.9**	**48.0**	**146**
CNN DR (10^3)	12.1	22.9	14.9	26.9	19.2	32.5	19.1	108	57.2	167
LSTM ($4 \cdot 10^3$)	**12.7**	**23.3**	**18.8**	**32.3**	**22.6**	**37.8**	**16.4**	**89.5**	**50.1**	**144**
LSTM (10^3)	13.6	24.7	33.9	58.8	29.0	48.0	17.3	91.8	51.2	**144**
Bi-LSTM ($4 \cdot 10^3$)	**13.0**	**23.7**	**19.8**	**34.4**	**26.4**	**43.7**	**16.8**	**86.0**	**43.3**	**129**
Bi-LSTM (10^3)	15.3	27.8	22.3	39.3	28.8	49.2	21.9	108	51.8	147

Tabla 5.3: Comparación de modelos entrenados con $\sim 4 \cdot 10^3$ y $\sim 10^3$ parámetros entrenables, evaluando el error absoluto medio (MAE) y desviación estándar para cada modelo.

Se observa que de forma sistemática los resultados de los modelos empeoran aunque razonablemente poco en la mayoría de los casos para la MAE y de forma más pronunciada para la desviación estándar. Para una reducción del $\sim 75\,\%$ de los parámetros entrenables los modelos entrenados sobre HG aumentan su MAE en torno a 1-5 ADC *counts* (un 7-35 %), y aumentan su desviación estándar en torno a 1-10 ADC *counts* (un 6-40 %), con la excepción del modelo LSTM que descartamos al tener resultados muy irregulares. En futuros pasos del proyecto se hace necesario un análisis pormenorizado de la reducción máxima aceptable de la precisión

de los modelos y su validez para reconstruir energías de distintas celdas. Se planten dos posibles escenarios, un menor número de modelos más grandes y robustos que puedan servir para procesar muchas celdas distintas, o un número mayor de modelos más pequeños y ligeros especializados en menos celdas, equilibrando en cualquier caso la memoria a ocupar en las tarjetas FPGA y la capacidad de cómputo para procesar todas las celdas.

Capítulo 6

Conclusiones y siguientes pasos

6.1. Conclusiones

Se ha estudiado la distribución de energías y filtrado en función de las características del experimento teniendo en cuenta las distintas ganancias. Se ha entrenado distintos modelos de redes neuronales y comparado con el anterior algoritmo de reconstrucción de energías *Optimal Filtering* adaptado a μ=50, confirmando que un modelo neuronal simple Lineal y en especial modelos más avanzandos de redes neuronales pueden mejorar significativamente la reconstrucción de energías con *pileup*. Concretamente los modelos MLP y CNN parecen obtener los mejores resultados. Se ha comprobado en particular que los modelos entrenados son capaces de obtener dichos resultados con un número reducido de parámetros entrenables, lo que los hace viables para ser utilizados como algoritmos de reconstrucción de energías en tarjetas FPGAs en el paso previo al filtrado y almacenado de datos durante los experimentos.

Se ha obtenido un rendimiento más pobre de lo esperado al entrenar los modelos con datos de LG a pesar de tener menos ruido. Se sospecha que pueda deberse a la distribución de energías propia de esta ganancia en los experimentos. Otra posible causa podría ser el agresivo filtrado durante el preprocesado eliminando el 80 % de los datos (quizá las bajas energías contengan información valiosa para el aprendizaje), o la distribución de señal y ruido en las simulaciones. En el caso de modelos entrenados con datos de LG sin filtrar obtenemos MAEs bajas pero desviaciones estándar considerablemente altas. Preferiríamos en principio el caso opuesto, menos dispersión aceptando errores mayores pero sistemáticos que pudieran corregirse a posteriori.

No se aprecia diferencia significativa entre los dos modelos CNN (con y sin reducción de

dimensionalidad). Esto confirmaría que los datos más alejados del BC central de la ventana de *timesteps* no aportan información crítica o relevante, ya que el modelo que los tiene menos en cuenta no obtiene peores resultados.

Puede parecer (al menos en HG) que a mismo número aproximado de parámetros entrenables los modelos LSTM obtienen peores resultados que MLP y CNN, lo que podría deberse a que la mayor complejidad interna de los LSTM repercute en más parámetros para un mismo número de capas y neuronas. Esta mayor complejidad podría jugar en contra de la optimización de los modelos en vistas a su rendimiento en FPGAs al requerir de más parámetros para un mismo nivel de aprendizaje. Además podría considerarse que la estructura de los datos no tiene complejidad suficiente como para aprovechar el potencial de este tipo de modelos frente a otros menos especializados, como la capacidad de retener información a largo plazo aplicada a ventanas pequeñas de solo 15 *timesteps*, o la relativa baja complejidad de la reconstrucción del apilado de pulsos frente a otros campos de aplicación de este tipo de redes.

6.2. Siguientes pasos

- Adaptar el algoritmo *Optimal Filtering* a μ=200 y verificar que los modelos de redes neuronales siguen obteniendo mejores resultados.
- Considerar el uso de un pesado de datos directamente inverso a la distribución de energías que se ajuste mejor a los datos específicos del problema.
- Implementar una optimización más exhaustiva de los hiperparámetros y explorar variaciones de la configuración interna de capas de los modelos.
- Realizar pruebas adicionales para explicar los diferentes resultados en LG filtrado, como entrenar un unico modelo con datos de las dos ganancias.
- Entrenar los modelos CNN con distintos tamaños de kernel para estudiar el peso de los datos vecinos al BC central con la intención de optimizar más el modelo.
- Entrenar los modelos utilizando otras funciones de pérdida como la raíz del error cuadrático medio RSME o el error relativo y priorizar una baja dispersión de las predicciones

antes que un bajo error de la media, ya que un error sistemático conocido es corregible a posteriori.

- Hacer uso de métricas y evaluaciones adicionales.
- Entrenar sobre el conjunto completo de datos simulados (1M BC) y ampliar el entrenamiento al resto de celdas, evaluando cuales pueden ser tratadas por un mismo modelo.
- Comprobar la idoneidad del tamaño de los modelos en su ejecución en FPGAs, tanto por el tamaño ocupado en la memoria como por el tiempo necesario de cómputo durante la inferencia. Hacer un estudio sobre la validez de un mismo modelo para distintas celdas del detector según su posición, y evaluar la preferencia entre usar modelos más grandes aprovechables en más celdas (por tanto necesitando menos modelos entrenados y almacenados) o utilizar modelos más pequeños pero especializados en menos celdas (necesitando más cantidad de modelos).
- Estudiar la posibilidad de optimización de modelos mediante el uso *weight-regularizer* de tipo L1 para igualar a cero los parámetros menos influyentes y discretizaciones con herramientas como Tensorflow Lite [27].
- Añadir un paso adicional (basado en redes neuronales) como filtro previo a la reconstrucción de la energía consistente en clasificar el BC entre señal y ruido.
- Añadir más variables de output a tener en cuenta durante la reconstrucción, como el desfase temporal entre la lectura del valor en la celda y el pico máximo que corresponde con el momento exacto de la deposición de energía (factor de calidad).

Apéndice A

A.1. Curvas de aprendizaje

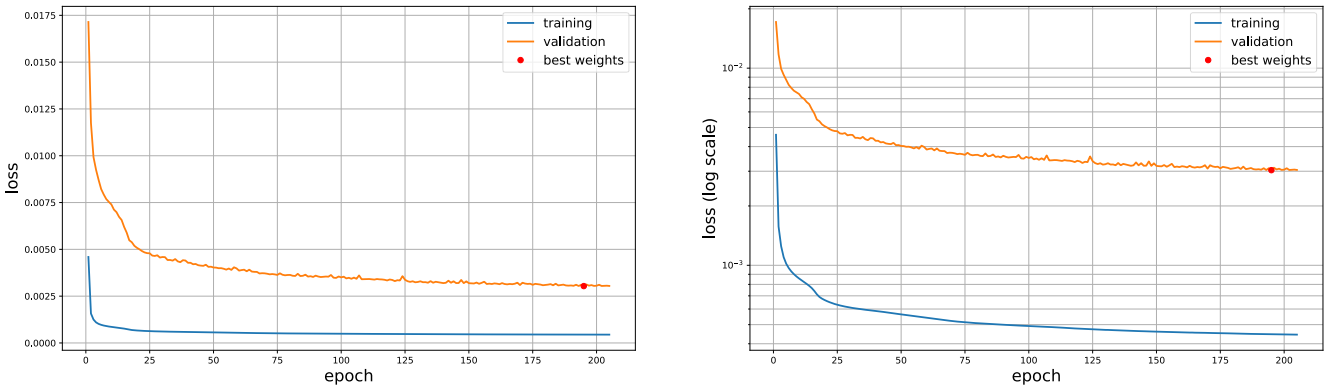


Figura A.1: Curva de aprendizaje para el modelo MLP con datos de HG pesados con la tangente hiperbólica y $\sim 4 \cdot 10^3$ parámetros entrenables.

A.2. Energía reconstruida frente a energía real

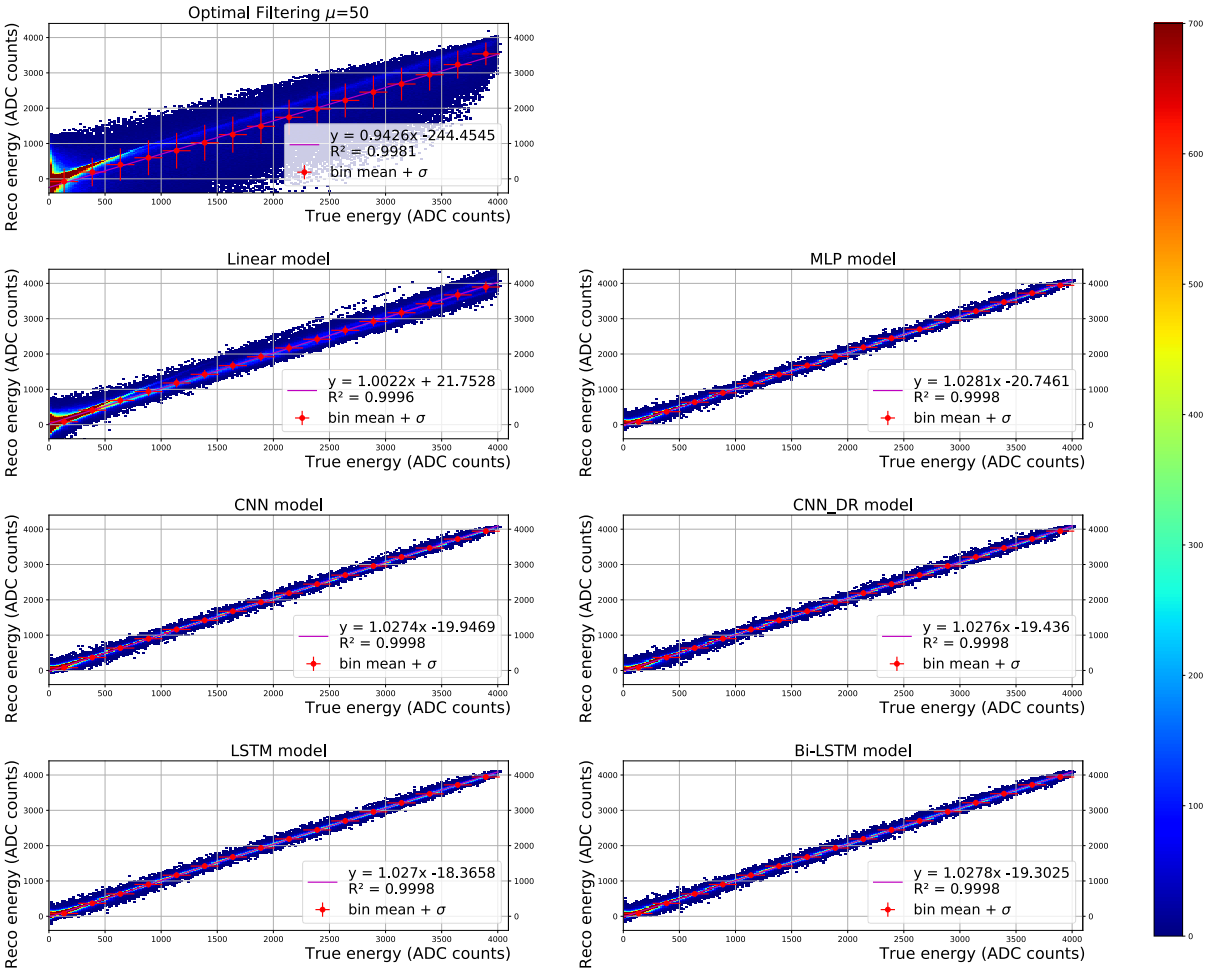


Figura A.2: Energía reconstruida frente a energía real para modelos HG con datos sin pesar y $\sim 4 \cdot 10^3$ parámetros entrenables.

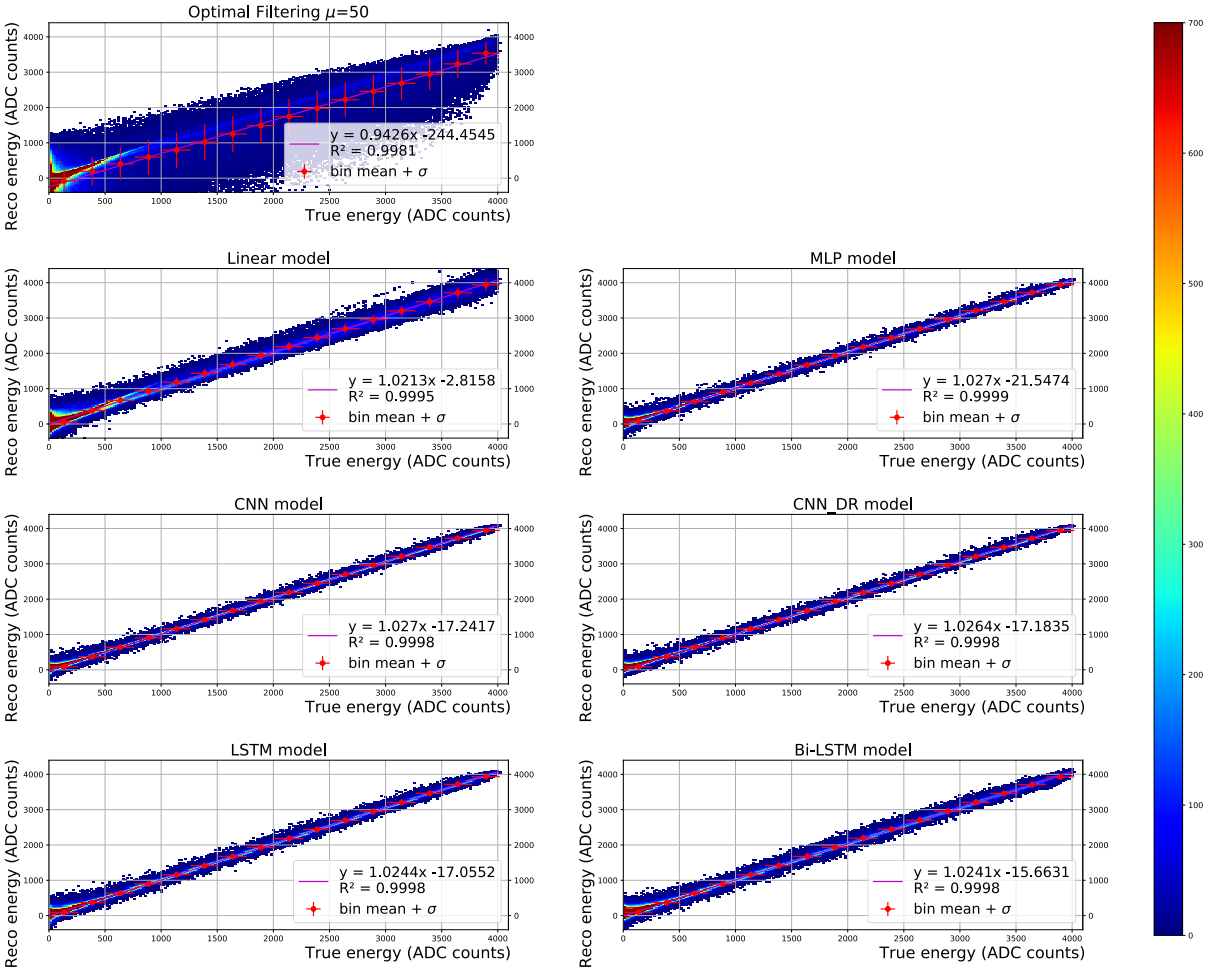


Figura A.3: Energía reconstruida frente a energía real para modelos HG con datos pesados con la exponencial y $\sim 4 \cdot 10^3$ parámetros entrenables.

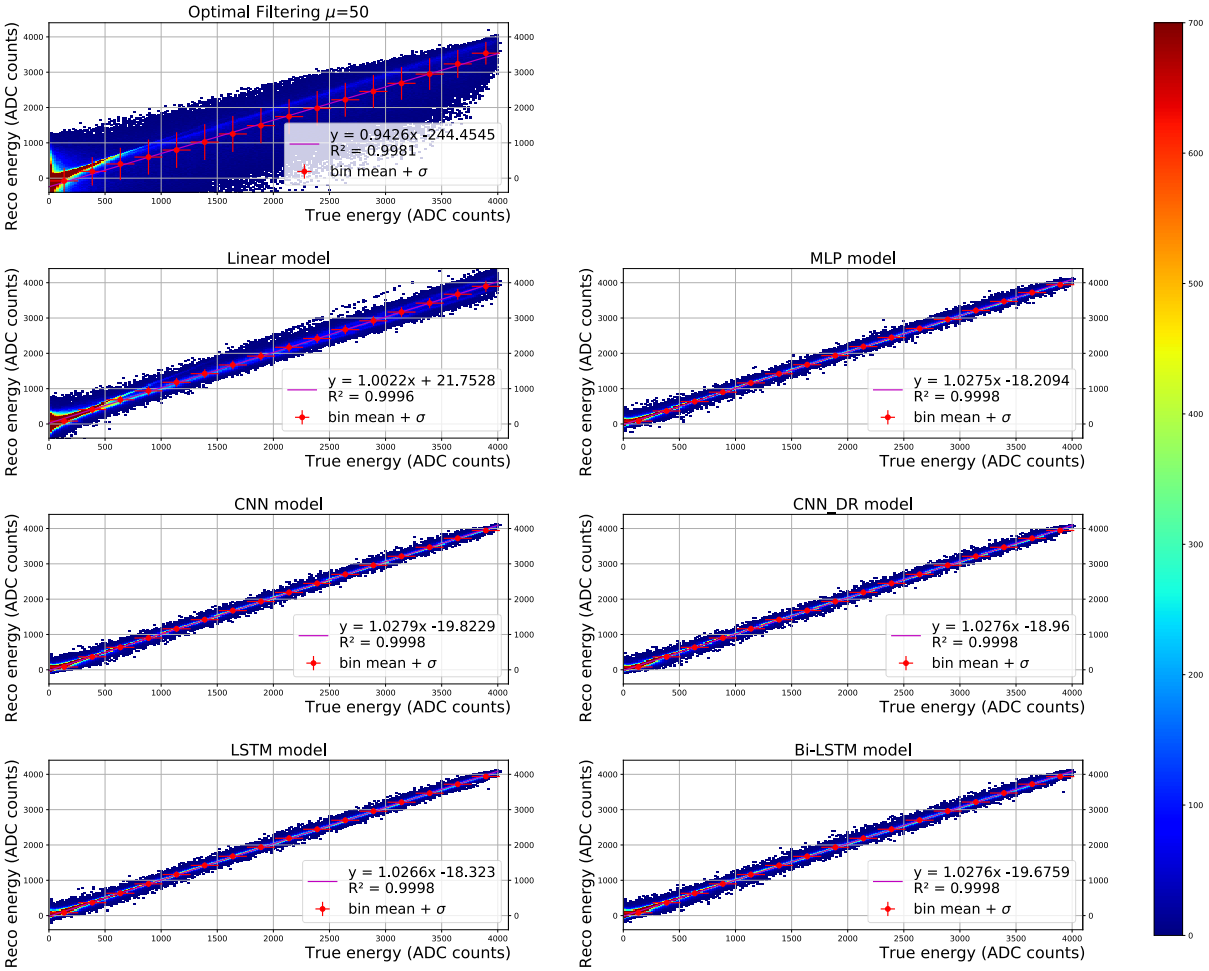


Figura A.4: Energía reconstruida frente a energía real para modelos HG con datos sin pesar y ~10^3 parámetros entrenables.

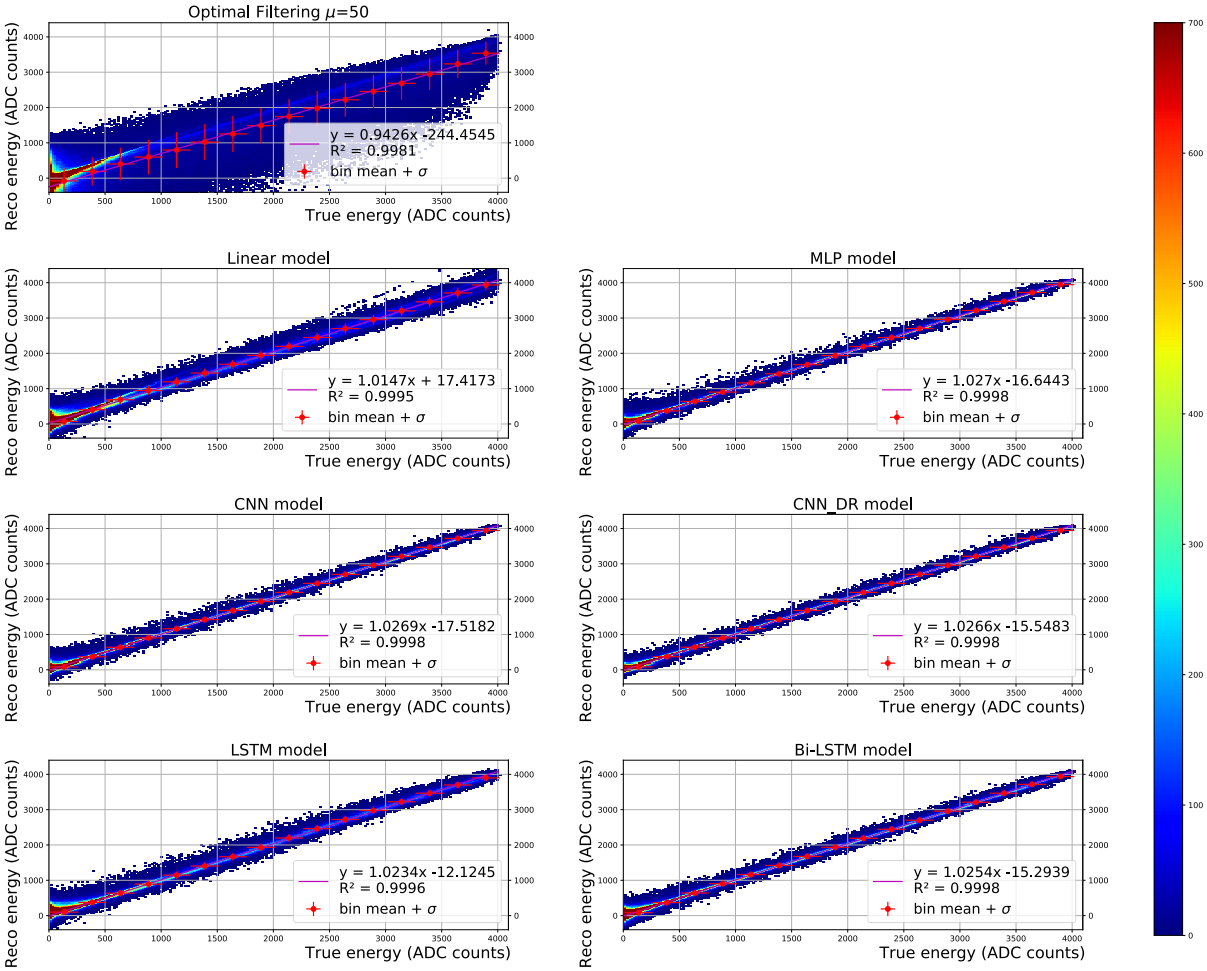


Figura A.5: Energía reconstruida frente a energía real para modelos HG con datos pesados con la tangente hiperbólica y $\sim 10^3$ parámetros entrenables.

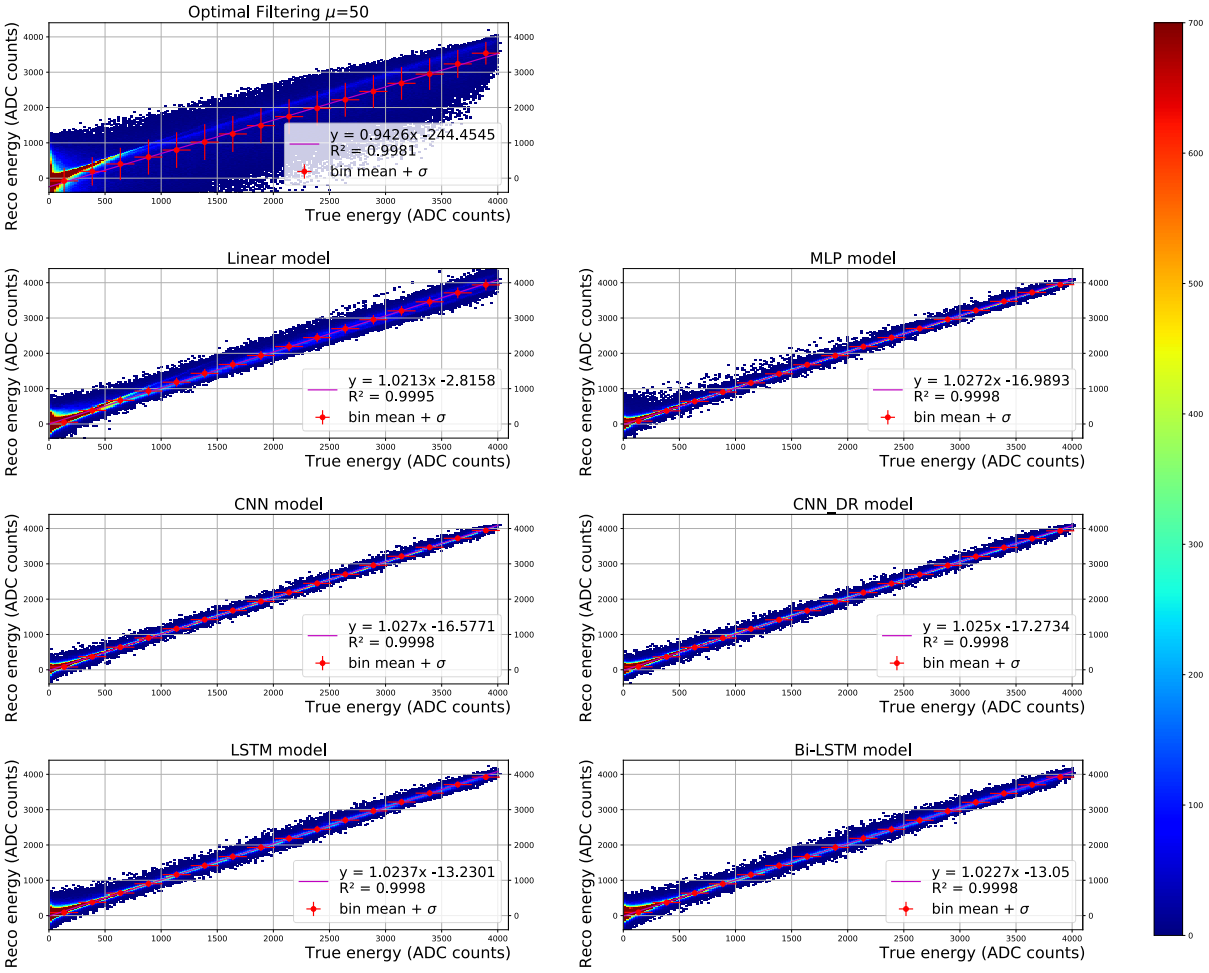


Figura A.6: Energía reconstruida frente a energía real para modelos HG con datos pesados con la exponencial y $\sim 10^3$ parámetros entrenables.

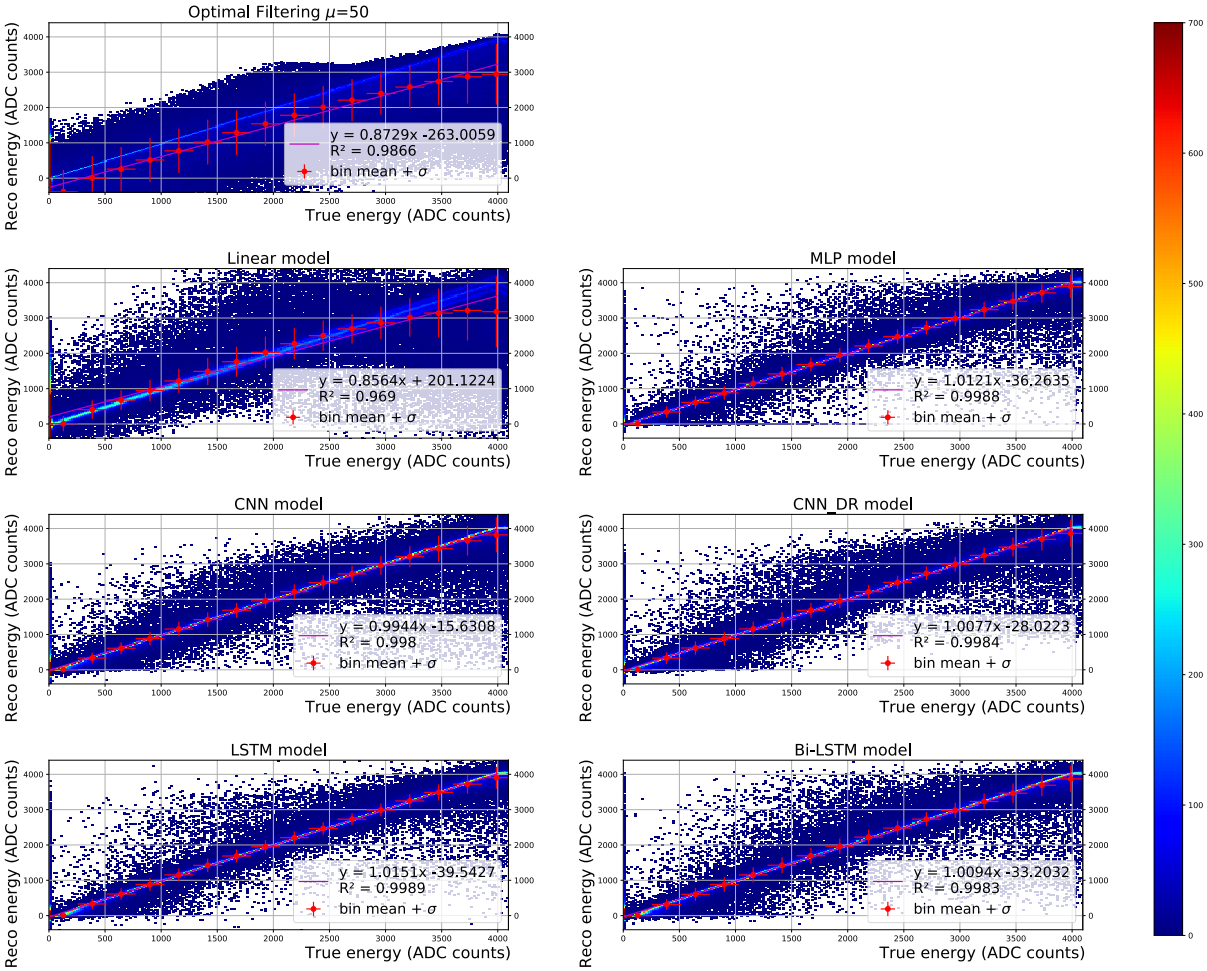


Figura A.7: Energía reconstruida frente a energía real para modelos LG con datos sin filtrar y $\sim 10^3$ parámetros entrenables.

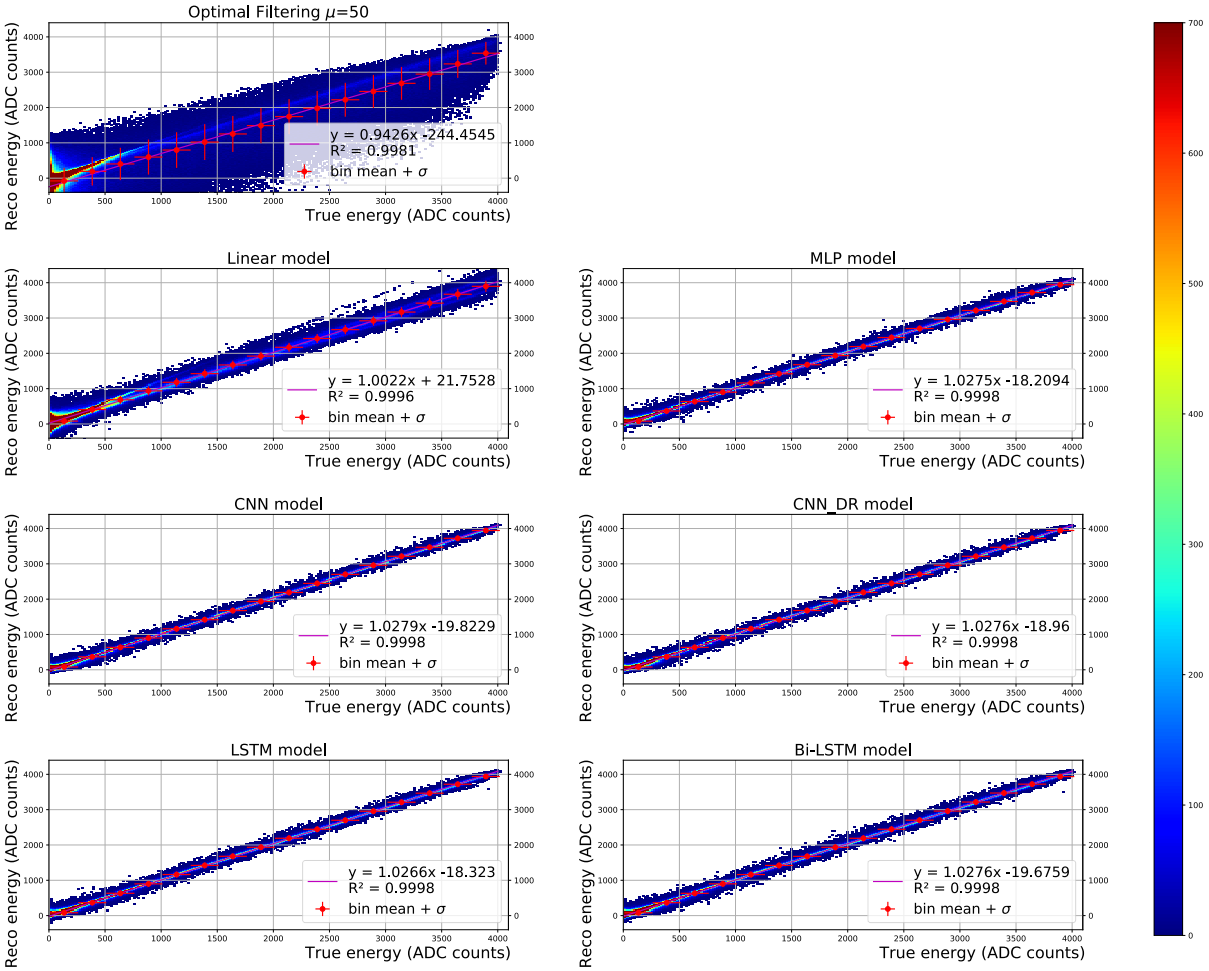


Figura A.8: Energía reconstruida frente a energía real para modelos LG con datos filtrados y ~10^3 parámetros entrenables.

A.3. Error relativo

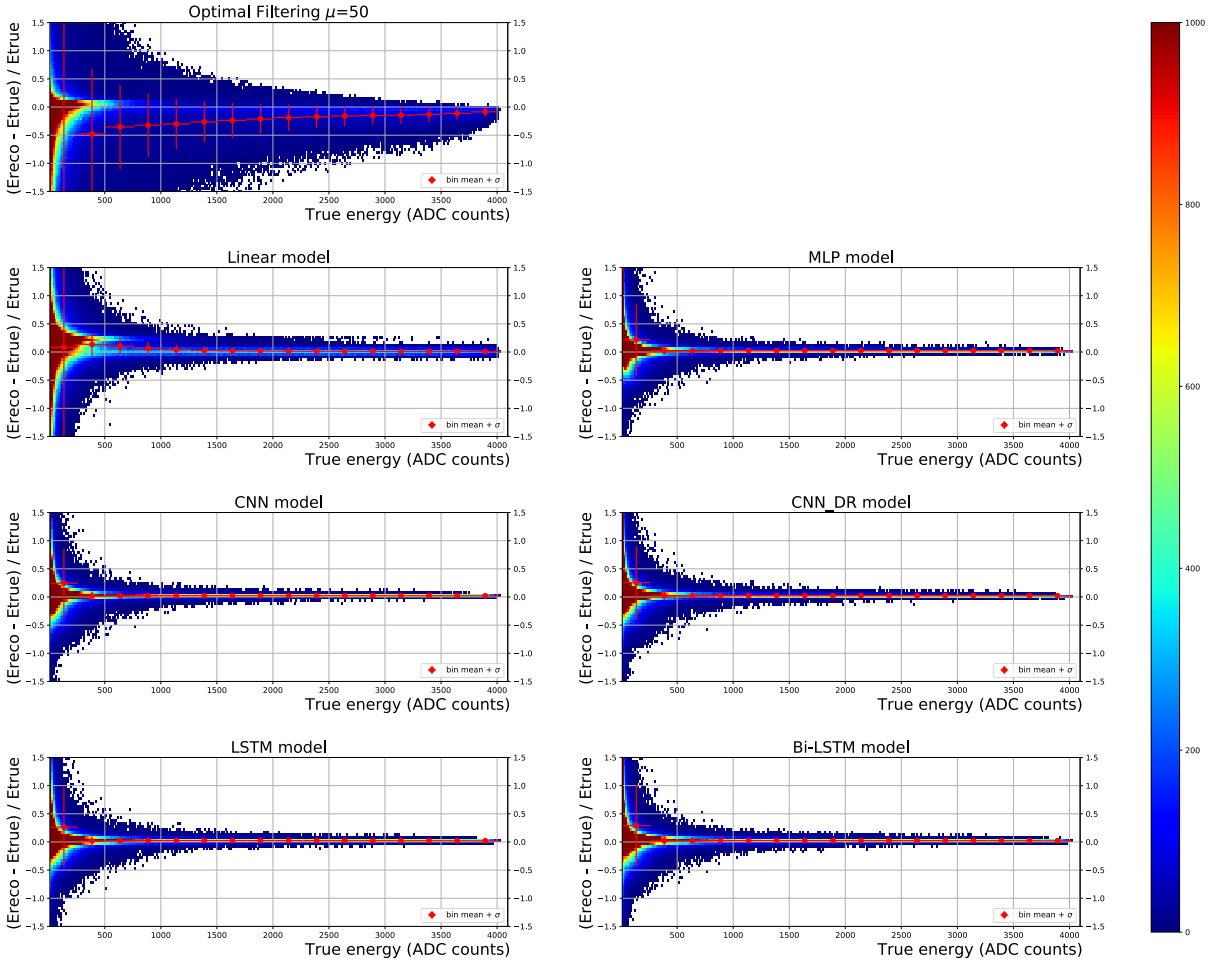


Figura A.9: Error relativo para modelos HG con datos sin pesar y $\sim 4 \cdot 10^3$ parámetros entrenables.

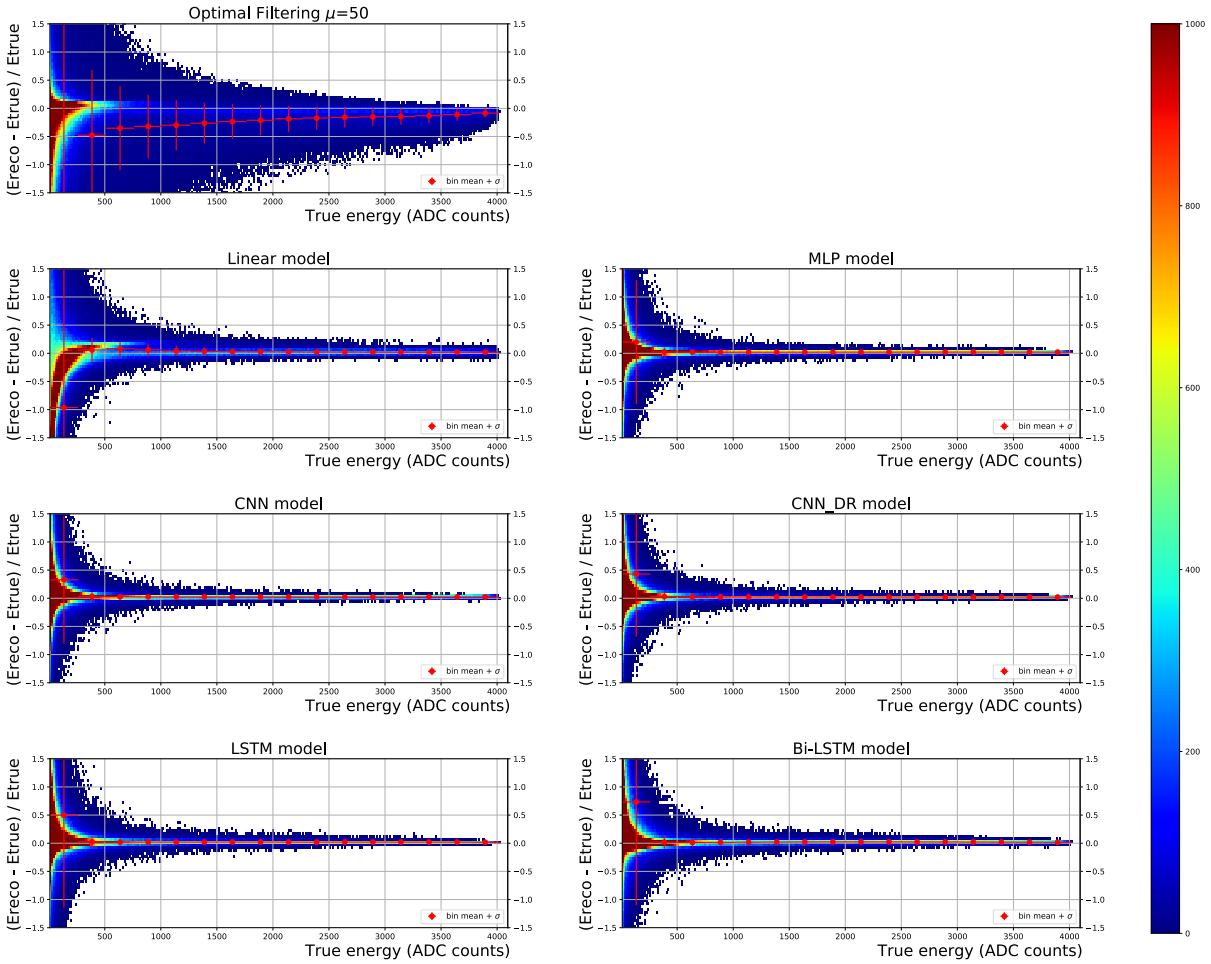


Figura A.10: Error relativo para modelos HG con datos pesados con la exponencial y $\sim 4 \cdot 10^3$ parámetros entrenables.

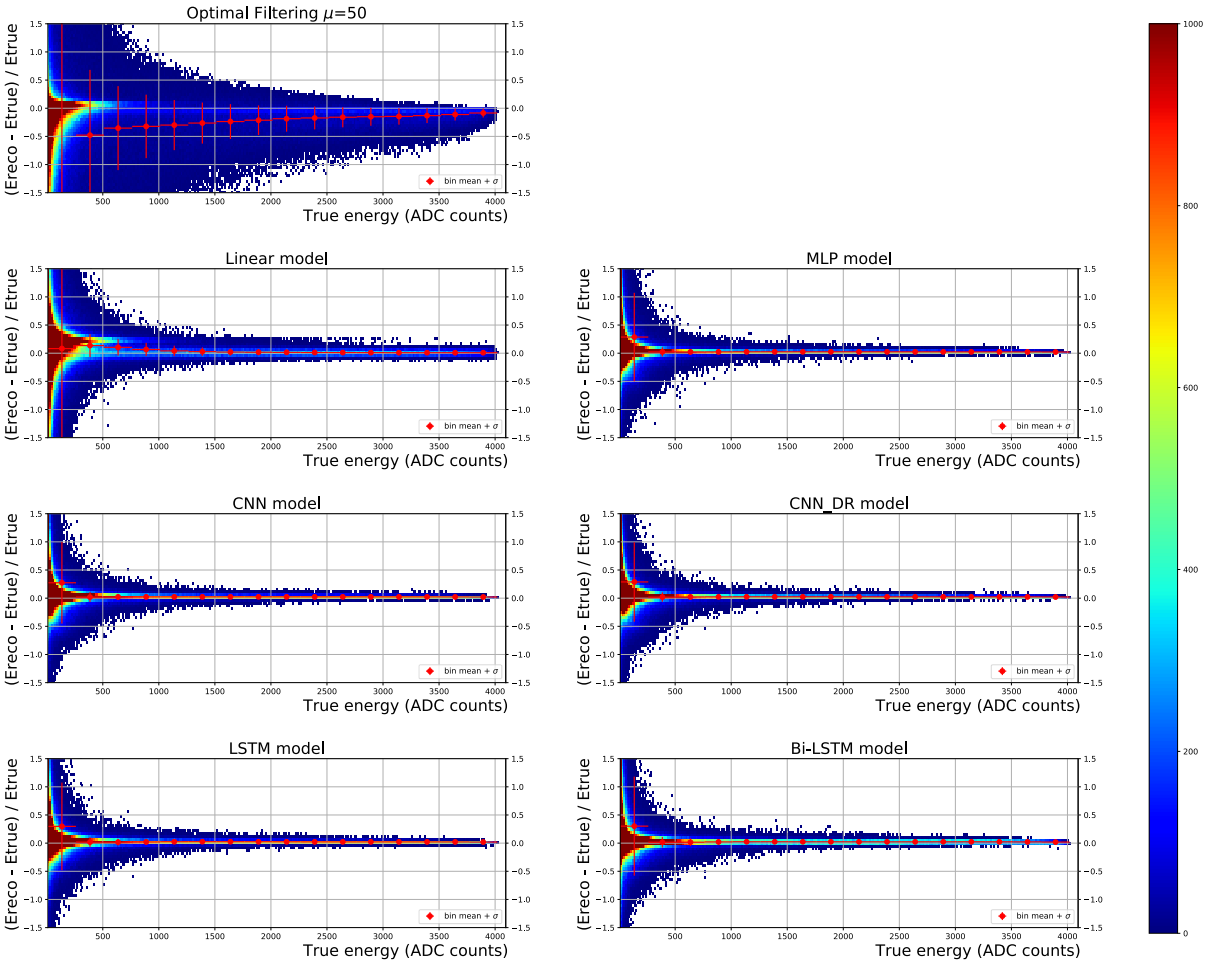


Figura A.11: Error relativo para modelos HG con datos sin pesar y $\sim 10^3$ parámetros entrenables.

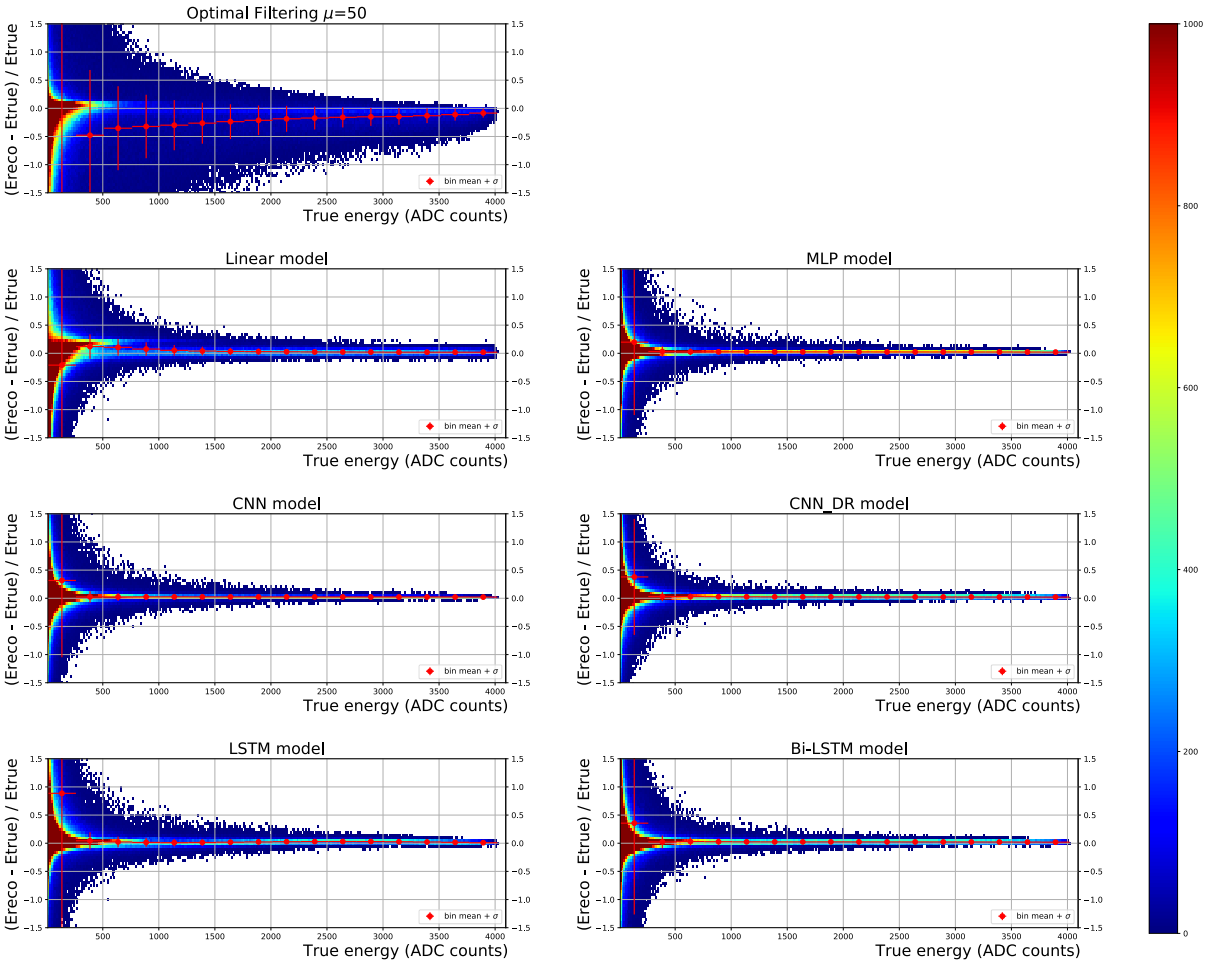


Figura A.12: Error relativo para modelos HG con datos pesados con la tangente hiperbólica y $\sim 10^3$ parámetros entrenables.

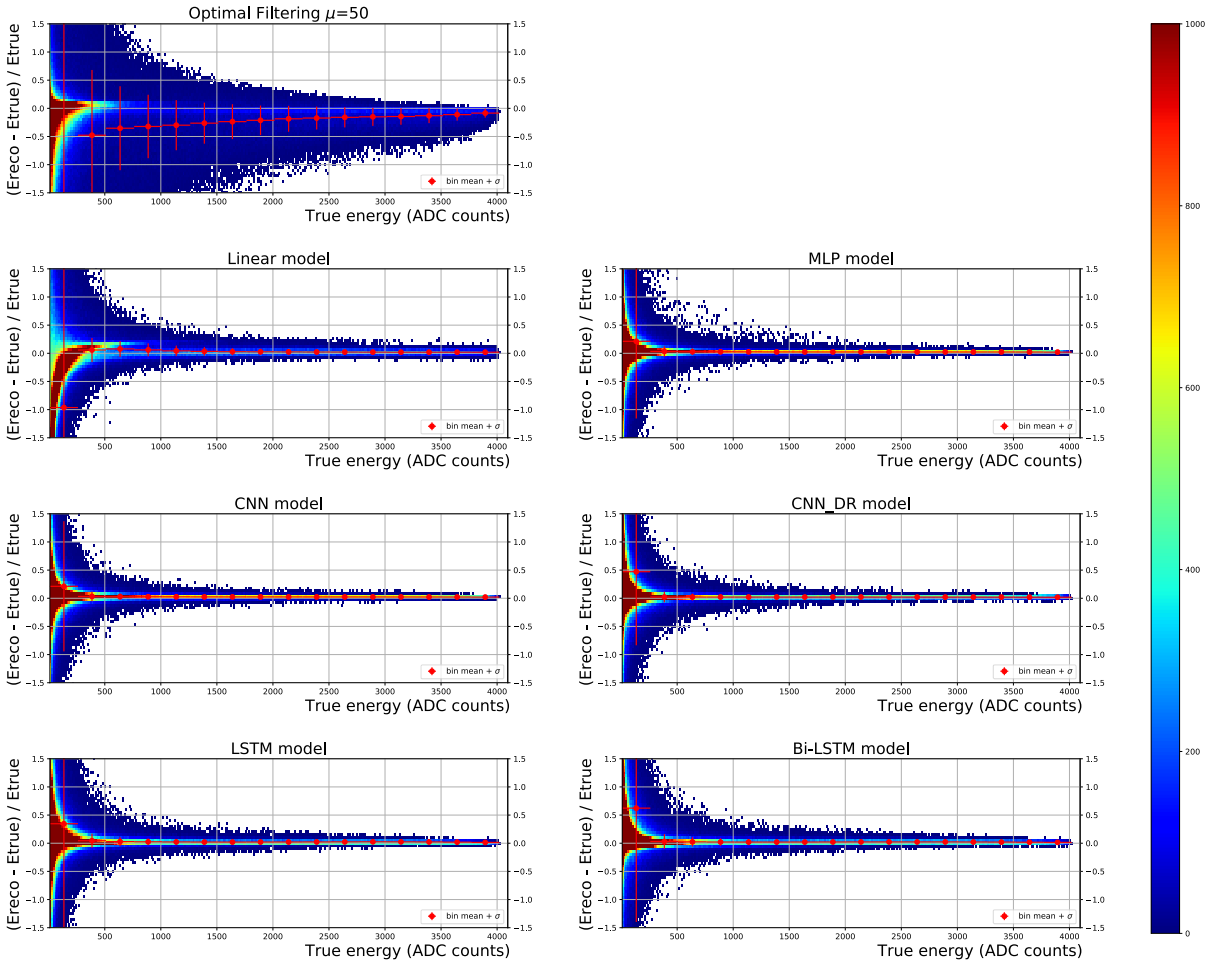


Figura A.13: Error relativo para modelos HG con datos pesados con la exponencial y $\sim 10^3$ parámetros entrenables.

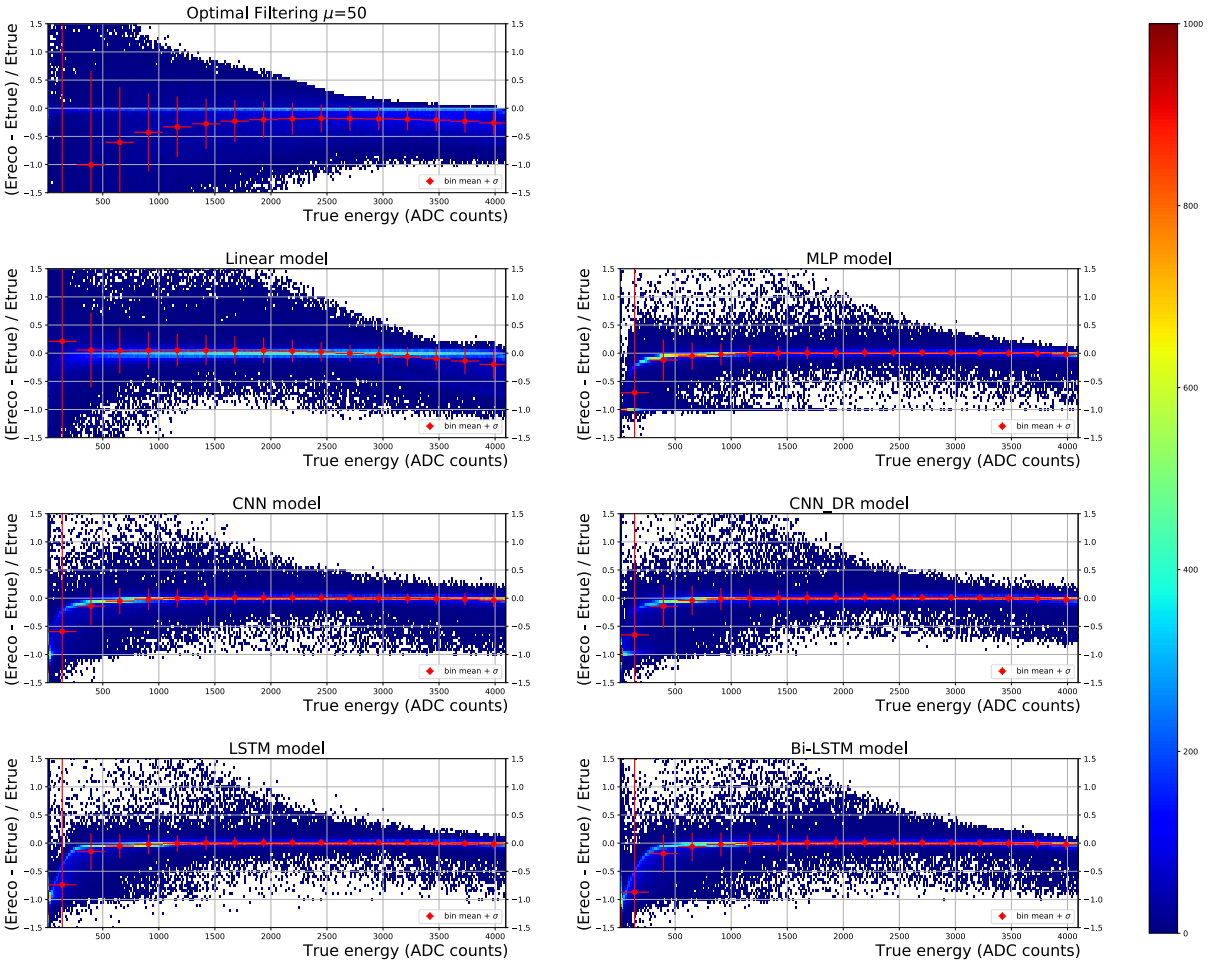


Figura A.14: Error relativo para modelos LG con datos sin filtrar y ~10^3 parámetros entrenables.

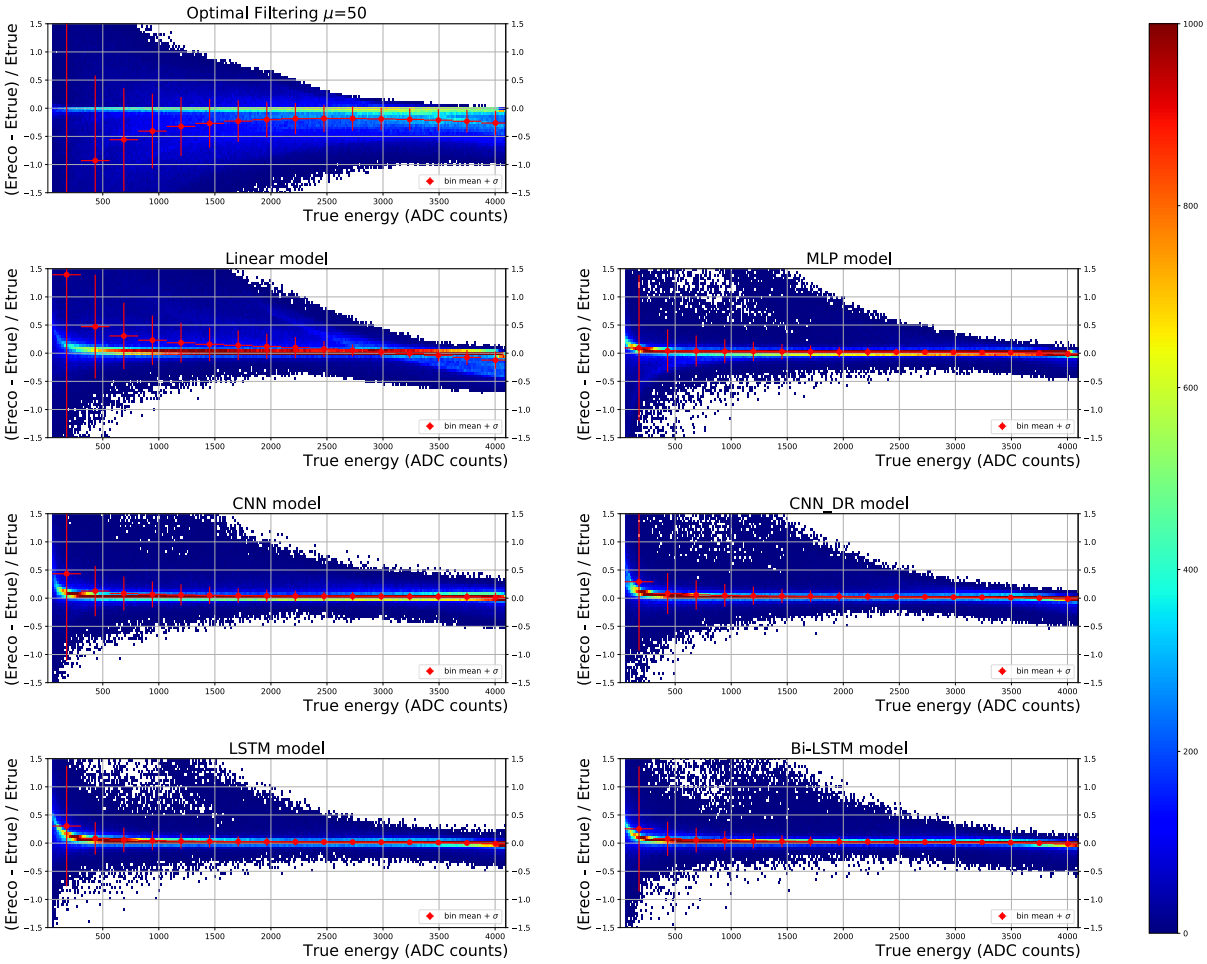


Figura A.15: Error relativo para modelos LG con datos filtrados y ~10^3 parámetros entrenables.

Bibliografía

[1] G Apollinari y col. *High Luminosity Large Hadron Collider HL-LHC*. en. 2015. DOI: 10.5170/CERN-2015-005.1. URL: https://cds.cern.ch/record/2120673 (vid. pág. 5).

[2] Peter Vankov. "ATLAS Upgrade for the HL-LHC: meeting the challenges of a five-fold increase in collision rate". En: *EPJ Web of Conferences* 28 (2012), pág. 12069. DOI: 10.1051/epjconf/20122812069. URL: https://doi.org/10.1051%2Fepjconf%2F20122812069 (vid. pág. 5).

[3] J.L. Ortiz Arciniega, F. Carrió y A. Valero. "FPGA implementation of a deep learning algorithm for real-time signal reconstruction in particle detectors under high pile-up conditions". En: *Journal of Instrumentation* 14.09 (2019), P09002. DOI: 10.1088/1748-0221/14/09/P09002. URL: https://dx.doi.org/10.1088/1748-0221/14/09/P09002 (vid. págs. 6, 49).

[4] J.P.B.S. Duarte y col. "Online energy reconstruction for calorimeters under high pile-up conditions using deconvolutional techniques". En: *Journal of Instrumentation* 14.12 (2019), P12017. DOI: 10.1088/1748-0221/14/12/P12017. URL: https://dx.doi.org/10.1088/1748-0221/14/12/P12017 (vid. pág. 6).

[5] Georges Aad y col. "Artificial Neural Networks on FPGAs for Real-Time Energy Reconstruction of the ATLAS LAr Calorimeters". En: *Computing and Software for Big Science* 5.19 (2021). DOI: 10.1007/s41781-021-00066-y. URL: https://doi.org/10.1007/s41781-021-00066-y (vid. pág. 7).

[6] Service graphique, CERN. "Overall view of the LHC. Vue d'ensemble du LHC". En: (2014). General Photo. URL: http://cds.cern.ch/record/1708849 (vid. pág. 9).

[7] Catrin Bernius, on behalf of the ATLAS y CMS Collaborations. “HL-LHC prospects from ATLAS and CMS”. En: *Journal of Physics: Conference Series* 1271.1 (2019), pág. 012004. DOI: 10.1088/1742-6596/1271/1/012004. URL: https://dx.doi.org/10.1088/1742-6596/1271/1/012004 (vid. pág. 8).

[8] Riccardo Maria Bianchi y ATLAS Collaboration. “ATLAS experiment schematic illustration”. General Photo. 2022. URL: http://cds.cern.ch/record/2837191 (vid. pág. 10).

[9] ATLAS Collaboration. *Technical Design Report for the Phase-II Upgrade of the ATLAS Tile Calorimeter*. Inf. téc. Geneva: CERN, 2017. URL: http://cds.cern.ch/record/2285583 (vid. págs. 10, 12).

[10] Yu A Kulchitskii y col. *Electromagnetic Cell Level Calibration for ATLAS Tile Calorimeter Modules*. Inf. téc. Geneva: CERN, 2006. URL: http://cds.cern.ch/record/1004187 (vid. pág. 13).

[11] Romeo Bonnefoy, David Calvet y Samuel Calvet. *Proposed modifications to the FE-NICS1 board and results of tests*. Inf. téc. Geneva: CERN, 2021. URL: https://cds.cern.ch/record/2756483 (vid. pág. 11).

[12] ATLAS Collaboration. *Athena (21.0.127). Zenodo.* https://doi.org/10.5281/zenodo.4772550. 2021 (vid. pág. 14).

[13] Alberto Valero. *The Back-End Electronics for the ATLAS Hadronic Tile Calorimeter at the Large Hadron Collider*. https://digital.csic.es/handle/10261/112262. 2014 (vid. pág. 15).

[14] Martín Abadi y col. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (vid. pág. 18).

[15] Fons Rademakers y col. *root-project/root: First release of the v6-18 series.* Ver. v6-18-00. Jul. de 2019. DOI: 10.5281/zenodo.3336325. URL: https://doi.org/10.5281/zenodo.3336325 (vid. pág. 18).

[16] Google for Developers Machine Learning Foundational courses. *Regularization for Sparsity: L1 Regularization*. https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization. 2022 (vid. pág. 25).

[17] A Valero. *The ATLAS TileCal Read-Out Drivers Signal Reconstruction*. Inf. téc. Geneva: CERN, 2009. URL: http://cds.cern.ch/record/1223960 (vid. pág. 27).

[18] Marius-Constantin Popescu y col. "Multilayer perceptron and neural networks". En: *WSEAS Transactions on Circuits and Systems* 8 (jul. de 2009) (vid. pág. 29).

[19] Yann LeCun, Y. Bengio y Geoffrey Hinton. "Deep Learning". En: *Nature* 521 (mayo de 2015), págs. 436-44. DOI: 10.1038/nature14539 (vid. pág. 29).

[20] Seungsoo Nam y col. "Forged Signature Distinction Using Convolutional Neural Network for Feature Extraction". En: *Applied Sciences* 8 (ene. de 2018), pág. 153. DOI: 10.3390/app8020153 (vid. pág. 30).

[21] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG] (vid. pág. 31).

[22] J. Brownlee. *Long Short-term Memory Networks with Python: Develop Sequence Prediction Models with Deep Learning*. Jason Brownlee, 2017. URL: https://books.google.es/books?id=ONpdswEACAAJ (vid. pág. 31).

[23] Savvas Varsamopoulos, Koen Bertels y Carmen Almudever. *Designing neural network based decoders for surface codes*. Nov. de 2018 (vid. pág. 32).

[24] Diederik P. Kingma y Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (vid. pág. 32).

[25] Adam Gołda. *Principles of training multi-layer neural network using backpropagation*. https://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html (vid. pág. 33).

[26] Vinod Nair y Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". En: *International Conference on Machine Learning*. 2010. URL: https://api.semanticscholar.org/CorpusID:15539264 (vid. pág. 36).

[27] *Deploy machine learning models on mobile and edge devices*. https://www.tensorflow.org/lite (vid. pág. 58).