Facultad
de
Ciencias

SIMULACIONES REALISTAS DE COLISIONES PROTÓN-PROTÓN EN EL LHC USANDO UNA RED NEURONAL CONVOLUCIONAL EXTRACTORA DE CORRELACIONES LOCALES

(Realistic simulations of proton-proton collisions at the LHC using a convolutional neural network as an extractor of local correlations)

Trabajo de Fin de Grado
para acceder al

GRADO EN FÍSICA

Autor: Mario Señas Gómez

Director: Pablo Martínez Ruiz del Árbol

Co-Director: Lara Lloret Iglesias

Septiembre - 2018

Índice

1 Introducción **3**

1.1 Física de Partículas y análisis de datos 3

1.1.1 LHC, CMS y CERN 3

1.1.2 El problema de la Materia Oscura 6

1.1.3 Materia oscura en el LHC 7

1.2 Machine Learning 9

1.2.1 Redes neuronales artificiales 11

1.2.2 Redes convolucionales 14

2 Producción de Montecarlo realista **17**

2.1 La importancia de los fondos 17

2.2 Propuesta de Solución: Estilo 18

3 Creación de RRNN sencillas usando Keras y Tensorflow **21**

3.1 Introducción: TF y Keras 21

3.2 Red sencilla para discriminar poblaciones 22

3.3 Red para discriminar sucesos de Altas Energías 25

4 Representación de sucesos de Altas Energías como imágenes **30**

4.1 Expresar un suceso en una foto 30

4.2 Aplicación de sesgos de forma controlada 31

5 Aplicación de una CNN a las imágenes **33**

6 Conclusiones **40**

Agradecimientos

Gracias a todos los que de una manera u otra me han ayudado o apoyado para llegar hasta aquí. Sin vosotros habría sido de todo punto irrealizable.

A Pablo Martínez, por llamar mi atención con una idea que jamás hubiese esperado, por su personal fascinación con el tema, por sus enseñanzas y su paciencia y entrega. Y a Lara Lloret, por su silenciosa colaboración casi chamánica en los momentos en que hizo falta.

A toda la gente del IFCA, por el ambiente que allí se vive.

A la Universidad de Cantabria por todo lo aprendido y las oportunidades concedidas. A mis profesores, con los que he aprendido no sólo de ciencia. A mis compañeros, a los que siguen por aquí y a los que partieron, y a los que partirán.

A mis amigos, a los que nunca dedico el tiempo suficiente.

A mi familia, lo más grande que tengo. Por lo bien que lo pasamos, pese a lo mal que lo he pasado. A mis padres por su confianza ciega, y por ser cada día fuente de inspiración. A mi hermana, para quien siempre quise ser el mejor referente posible y quien puede terminar siéndolo para mí. A los que ya no están.

GRACIAS

Resumen

Este Trabajo de Fin de Grado se ha planteado varios objetivos en relación con la aplicación de técnicas de aprendizaje automático al análisis de datos en física de partículas, repartidos en dos fases.

El primer objetivo consiste en comprobar las capacidades de las redes neuronales para cubrir tareas de clasificación de casos sencillos y aplicarlas a la clasificación de datos correspondientes a simulaciones de Montecarlo de eventos de producción de pares de quarks top-antitop y de producción de materia oscura en asociación con estos pares $t\bar{t}$.

El segundo objetivo es estudiar una técnica existente de redes neuronales convolucionales desarrollada para extraer el estilo artístico de obras pictóricas (basado en las correlaciones locales de la imagen), y realizar un estudio de viabilidad para su aplicación a un caso de generación de simulaciones más realistas de eventos de colisiones de partículas en el LHC.

Se han desarrollado redes neuronales sencillas de tipo perceptrón multicapa. La aplicación de este método consigue clasificar puntos tomados de dos gaussianas bidimensionales contiguas con hasta un 97,7 % de precisión.

Otro sistema de arquitectura similar se ha utilizado para clasificar eventos simulados de producción de $t\bar{t}$ y de producción de materia oscura junto con $t\bar{t}$. No se han obtenido mejoras significativas con respecto a una separación en función del valor del momento transverso ausente, $p_{T,Miss}$.

Para aplicar una red neuronal convolucional de tratamiento de imágenes a datos de colisiones en el LHC, se ha diseñado un formato para expresar estos datos en forma de imágenes. Se han alterado voluntariamente algunos de los datos introducidos, reflejando en las imágenes un estilo concreto. El procesamiento de las imágenes no consigue aplicar a las imágenes originales el estilo introducido en las imágenes modificadas. El método utilizado parece que no es viable, aunque se contemplan estudios adicionales con estructuras diferentes tanto para el formato de las imágenes como de la red convolucional.

Palabras clave: Aprendizaje automático, Redes neuronales convolucionales, LHC, Materia oscura, Simulación de Montecarlo.

Abstract

This final degree project has considered several objectives regarding the application of machine learning techniques to data analysis in particle physics, divided in two stages.

The first objective consists in checking neural networks' capacities to fulfill classification tasks with simple cases and then applying them to data correspnding to Montecarlo simulations of top-antitop quark pairs production events and dark matter production events in association with $t\bar{t}$.

The second objective is to study an existing convolutional neural networks technique developed for extracting the artistic style from paintings (based on local correlations of the image), and perform a feasibility study for its application to one case of producing more realistic simulations of particle collisions events at the LHC.

Some multilayer perceptron-type vanilla neural networks have been developed. The implementation of this method achieves to classify, with a 97.7% accuracy, points taken from two contiguous bidimensional gaussians.

Another system with similar architecture has been utilized to classify simulated events of $t\bar{t}$ production and dark matter production in along with $t\bar{t}$. No significant improvements have been obtained with respect to simple separation in terms of the value of the missing transverse momentum, $p_{T,Miss}$.

In order to apply a convolutional neural network for image processing to data from collisions at the LHC, a format for expressing those data as images has been designed. Some of the introduced data have been intentionally altered, capturing a particular style in the images. The handling of the images by the network does not manage to apply on the original images the style introduced in the modified ones. The method used appears to be not viable, although further studies with different structures for the formatting of the images as well as for the convolutional network are envisaged.

Keywords: Machine learning, Convolutional neural networks, LHC, Dark matter, Montecarlo simulation.

1 Introducción

Este trabajo se enmarca en el campo de la física de partículas y las búsquedas de materia oscura; en concreto materia oscura asociada con pares de quarks $t\bar{t}$. Se busca poner a prueba nuevos métodos para hacer más realistas y mejorar la calidad de los modelos de predicción para sucesos de $t\bar{t}$, que son el principal fondo para la señal que se espera. Se utilizan métodos de inteligencia artificial como son las redes neuronales convolucionales para tratar de extraer las correlaciones en los datos inherentes al proceso de detección.

1.1 Física de Partículas y análisis de datos

La física de partículas es una rama de la física dedicada a estudiar la estructura de la materia en las escalas de tamaño más pequeñas; sus constituyentes fundamentales, las partículas elementales, y las interacciones entre ellas. Sus áreas de investigación son la determinación de parámetros fundamentales (como los acoplamientos de las distintas interacciones), y la búsqueda de nuevas teorías más completas. Dado el elevado coste de sus experimentos más importantes, los grupos de investigación se aúnan en colaboraciones internacionales, las cuales a su vez participan en grandes proyectos como el CERN

1.1.1 LHC, CMS y CERN

En Ginebra se encuentra la sede de la Organización Europea para la Investigación Nuclear (CERN), y junto a ella unas instalaciones que albergan el Gran Colisionador de Hadrones (LHC)(cf. Figura 1). El LHC es un acelerador colisionador de partículas de tipo sincrotrón instalado en un túnel de 27 km de circunferencia que previamente alojaba otro acelerador, el LEP. Por este acelerador se hacen circular en sentido contrario dos haces (*beams*, en inglés) de protones divididos en varios grupos (*bunches*) de unos 10^{11} protones, los cuales se cruzan y se hacen colisonar en 4 puntos de interacción (IP) distribuidos a lo largo del anillo. En cada uno de ellos se ubica un detector que recoge los productos de la interacción. Uno de estos detectores es el Solenoide Compacto de Muones (CMS).

Figura 1: Fotografía de la ubicación en superficie, entre Francia y Suiza, del CERN y el LHC con sus experimentos. El acelerador se encuentra en un túnel subterráneo cuyo recorrido representa la circunferencia en la imagen.

CMS es un detector de partículas de propósito general, de estructura cilíndrica. Este detector, junto con el otro detector ATLAS del LHC, ha logrado completar el zoo de partículas comprendidas en el Modelo Estándar (SM) de física de partículas hallando evidencias del bosón de Higgs[1] [2] , y ahora se dedica a buscar señales que indiquen comportamientos no predichos por el SM. El sistema de referencia se fija con el eje z en la dirección del haz de protones en sentido antihorario (desde arriba), el eje x hacia el centro del anillo y el eje y hacia arriba. El plano xy es por lo tanto transversal al movimiento del haz.

La estructura del detector consta a grandes rasgos de tres subestructuras:

- *Barrel* o barril. El cuerpo central del detector y el que le da su aspecto cilíndrico. Sus sistemas detectores se ubican alrededor del tubo del haz.
- *EndCaps* o tapas. Dos, cada una a un extremo del *Barrel*. Los detectores están orientados perpendicularmente al haz.

Dadas las características de simetría y de energías relativistas del experimento, las coordenadas manejadas son el ángulo acimutal ϕ y la variable η en lugar del ángulo polar θ, tal que

$$\eta = -\tan\left(\frac{\theta}{2}\right),$$

de modo que las diferencias en η son invariantes Lorentz.

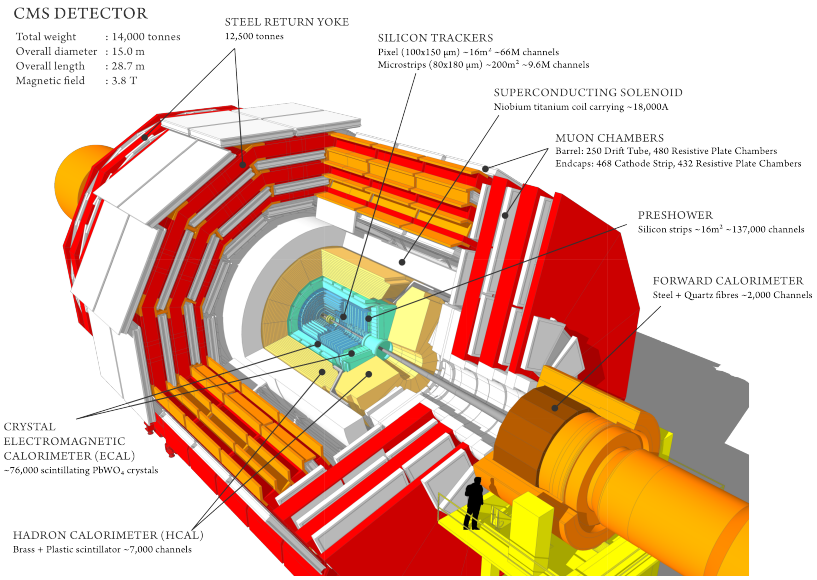


Figura 2: Esquema de CMS. Se muestra la vista la estructura interna del detector en cortes transversal y longitudinal.(CERN)

Entonces, cuando se producen colisiones en el punto de interacción, las partículas generadas salen en todas direcciones, interaccionando con las distintas partes del detector y depositando energía en ellas. Para cada partícula identificada, se calculan η y ϕ, además de la componente de su momento en el plano transverso, p_T, ya que el haz inicial es perpendicular a este plano; y la masa.

CMS, desde la parte más interna (el haz) hasta el exterior, está compuesto por distintos detectores dispuestos en capas, cada uno con una especialidad, y que deben soportar durante tiempos suficientemente prolongados condiciones de alta radiación ionizante y alto campo magnético. Son, desde el interior: el *Tracker*, el calorímetro electromagnético, el calorímetro hadrónico y la cámara de muones. Se muestra un esquema en la Figura 2. Entre el calorímetro hadrónico y las cámaras de muones se ubica un solenoide, el cual genera un campo magnético paralelo al haz de forma que las partículas cargadas describan trayectorias curvadas en el plano transverso. Esto sirve como herramienta de identificación. Las características de los detectores son:

- *Tracker* o trazador. Son detectores de silicio de dos tipos.
  - De píxeles. Es la capa más cercana al punto de interacción. Tienen mucha precisión y se utilizan para detectar los vértices de

producción de las partículas.

  - De tiras. En la capa inmediatamente exterior se ubican estos detectores, con precisión inferior pero mucha más cantidad.

- Calorímetro electromagnético. Su cometido es absorber los electrones y fotones. Está compuesto por cristales centelleadores de $PbWO_4$ de respuesta rápida (80 % de la energía es liberada en el tiempo entre dos *bunches*). Para recoger el centelleo se utilizan detectores de silicio.

- Calorímetro Hadrónico. Absorbe los hadrones, partículas que interaccionan por la fuerza fuerte

- Cámaras de muones. Detectan muones, que son partículas de alta penetración y las únicas detectables que son capaces de atravesar toda la estructura interior. Dada la valiosa información de los muones, se destina un tipo de dispositivo específico a su detección.

1.1.2 El problema de la Materia Oscura

Uno de los mayores desafíos de la física en la actualidad es la naturaleza de la materia oscura (DM). La DM es, supuestamente, un tipo de materia no descrito por el SM que no interacciona electromagnéticamente. Por tanto, no emite ni dispersa la luz y no podemos verla. Uno de los mayores experimentos dedicados a investigar la DM es el LHC.

La DM fue por primera vez propuesta en la primera mitad del siglo XX como origen a la dispersión de velocidades de galaxias dentro del Cúmulo de Coma, cuando el astrónomo Fritz Zwicky realizó una serie de estimaciones a partir del Teorema del Virial[3].

Las primeras observaciones que apoyaban esta hipótesis llegaron más de 30 años después cuando la astrónoma Vera Rubin empezó a medir velocidades de rotación de las galaxias espirales y a comprobar que no se correspondía con lo predicho por la dinámica a partir de las relaciones de masa-luminosidad[4]. Según aumenta el radio respecto del centro de la galaxia, los astros deben incrementar su velocidad debido al incremento de la masa contenida en ese radio. Pero al incrementar más el radio, abandonando el bulbo galáctico y progresando hacia el exterior del disco, la materia contenida crece más lentamente, por lo que la velocidad debería disminuir[5]. Sin embargo, lo que

se observa es que las velocidades se mantienen aproximadamente iguales para radios crecientes, y esto conduce a la presencia de una masa no brillante.

El tercer pilar sobre el cual se sostiene la DM es el efecto lente gravitatoria. Este es uno de los fenómenos predichos por la teoría de la Relatividad General de Einstein. La luz también sufre los efectos de la gravedad. Así, una distribución de masa lo suficientemente densa que se interponga en la línea de visión de otro objeto más lejano será capaz de curvar los rayos procedentes de dicho objeto, alterando su apariencia de modo similar a como puede ocurrir con una lente óptica. Analizando, por ejemplo, las imágenes múltiples producidas, es posible inducir la distribución de masa que genera la lente. Este método fue también aplicado por Zwicky en 1937[6][7]. De nuevo, la masa obtenida por este método es superior a la calculada a partir del brillo.

Otras señales de las décadas más recientes que refuerzan esta teoría son:

- La estructura a gran escala del universo. Según el modelo cosmológico actual, para cuando el universo alcanzó la época de la recombinación - origen del fondo cósmico de microondas (CMB)-, ya existían pequeñas inhomogeneidades en la distribución de materia (propiciadas por la materia oscura fría) que se fueron amplificando. La materia bariónica aún muy caliente no podría formar estructuras a esas escalas.
- El Cúmulo Bala. Un cúmulo de galaxias formado por dos cúmulos en colisión. En la imagen de rayos X del gas caliente se observa que en la colisión el gas ha interaccionado frenándose, mientras que a partir del perfil gravitatorio extraído por efecto lente, se observa que la mayor parte de la masa de los dos cúmulos se ha atravesado sin interaccionar, no teniendo las galaxias suficiente masa para explicarlo.[8][9]

Así pues el tipo de materia que se busca como candidato de la materia oscura es una partícula sensible a la interacción gravitatoria, pero no a la electromagnética, de tipo no bariónico, suficientemente masiva como para haberse enfriado en la época del desacoplo.

1.1.3 Materia oscura en el LHC

La búsqueda de materia oscura en aceleradores de partículas, y en el LHC en particular, se lleva a cabo con modelos teóricos simplificados que intentan

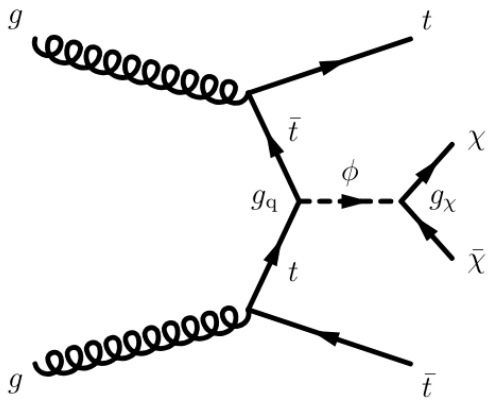


Figura 3: Diagrama de Feynman de un proceso de producción de DM en asociación con $t\bar{t}$.

ser lo más agnósticos posible.

Uno de los modelos propuestos, plantea la existencia de una partícula mediadora de espín 0 (similar al bosón de Higgs) en las interacciones entre la DM y la materia ordinaria.

Producción de DM en asociación con pares de quarks top. En las colisiones protón-protón en el LHC, una de las reacciones que se pueden producir es una interacción entre dos gluones que decaen a un par de quarks top-antitop ($t\bar{t}$) y a un mediador que se desintegra en un par de fermiones de DM [10]. El diagrama de Feynman asociado a este proceso puede verse en la Figura 3, en el que los quarks top producidos decaen cada uno a un bosón W y un quark b, y el W decae a su vez a un leptón y un neutrino.

Existe un proceso que no involucra DM, pero tiene un aspecto similar: dos gluones que interaccionan dando lugar a un par $t\bar{t}$ decayendo cada uno de los t por interacción débil a un quark b que *hadroniza* (debido a la intensidad de la interacción fuerte se desintegra en una cascada de hadrones que se desintegran en otros cada vez menos energéticos) formando *jets*, y a un bosón W que da lugar a un leptón y su correspondiente antineutrino. Su diagrama de Feynman se puede ver en la Figura4. Esta es una geometría característica que se identifica en el detector a partir de los leptones y los jets. Los neutrinos no interaccionan en el detector, por lo que la signatura

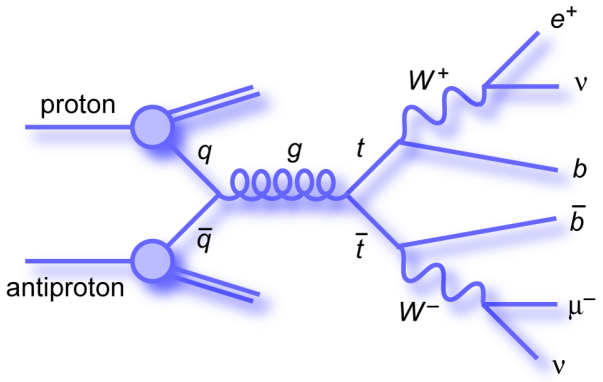


Figura 4: Diagrama de Feynman de un proceso de producción de pares $t\bar{t}$.

que el detector lee es que todas las partículas detectadas se dirigen hacia un lado, y ninguna señal en el lado opuesto del detector. Dada la simetría cilíndrica del experimento respecto del eje que establece el haz, esto supone un desequilibrio en la energía-momento transversal, ya que previamente a la colisión su valor es cero. De este modo se determina una variable denominada *Missing ET* o MET (energía transversa ausente), que suele indicar una correspondencia con una partícula invisible. Esto permite reconstruir sus los momentos de los neutrinos[11][12].

Sin embargo en la producción de pares $t\bar{t}$ en asociación con DM el momento ausente se atribuye tanto a los neutrinos como al par de partículas de DM generadas, por lo que no se pueden calcular analíticamente los momentos de los productos. El proceso de producción de pares top-antitop supone, para el proceso de producción de materia oscura y $t\bar{t}$, un fondo irreducible.

1.2 Machine Learning

El aprendizaje automático o *machine learning*, es una rama de la inteligencia artificial que estudia el desarrollo de programas informáticos destinados a la ejecución de una tarea concreta que sean capaces de mejorar su rendimiento (*aprendizaje*) en tal tarea por sí mismos (*automático*) a partir de su propia experiencia [13].

Se trata de una disciplina que se vale de herramientas estadísticas y computacionales para, a partir de los datos que se le proporcionan, opti-

mizar sus algoritmos. Así, llega a resolver problemas de generalización de comportamientos como: clasificación, regresión, reducción dimensional, identificación de anomalías, etc.

Una vez desarrollada, una herramienta de machine learning idealmente es capaz de predecir comportamientos futuros a partir de datos que no ha aprendido previamente. Se puede decir que es una expresión computacional del método científico.

Su utilidad emerge en contextos en los que es difícil, o imposible, desde la perspectiva humana reflejar en un algoritmo las instrucciones necesarias para llevar a cabo una tarea en cuestión. Esto puede ocurrir por manejar una cantidad de datos inabarcable, depender de muchos parámetros, imposibilidad de conocer las condiciones de aplicación futuras, o incluso desconocer la naturaleza del problema o sus métodos de resolución.

El aprendizaje automático se divide usualmente en dos vertientes atendiendo a la filosofía del "aprendizaje" en cuestión:

1. Aprendizaje Supervisado,
2. Aprendizaje No Supervisado.

En el aprendizaje supervisado al programa se le proporcionan los datos de entrada (o *inputs*) de los que tiene que aprender asociados con las respuestas (o *outputs*) que para ellos se esperan del programa. A este conjunto de datos se le llama conjunto de entrenamiento (*training*). El programa se encarga de alterar sus parámetros para que la respuesta sea óptima. Dependiendo de si la respuesta es bien un número, bien una etiqueta (*label*) o clase, será un problema de regresión o de clasificación.

En el caso del aprendizaje no supervisado al programa no se le proporcionan las respuestas deseadas, sino solamente los datos de entrada, y él debe encontrar algún tipo de estructura en ellos, detectando por ejemplo, características desconocidas.

La mayor parte de los métodos más exitosos, y los más usados, pertenecen al campo del aprendizaje supervisado, ya que resulta más fácil evaluar su precisión e implementarlos en la industria. Los métodos no supervisados crecen más despacio, acostumbrando a estar más limitados al ámbito acadé-

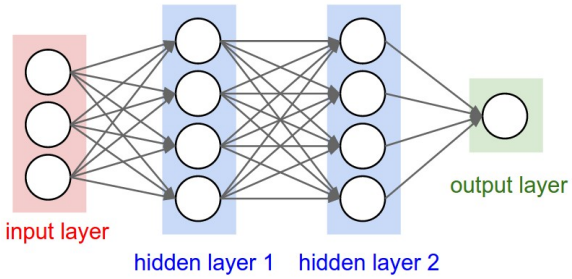


Figura 5: Esquema de Red Neuronal de tipo perceptrón multicapa. Es una red neuronal sin realimentación (*feedforward*), de 3 capas (sin contar la entrada), o de 2 capas ocultas. (Crédito Karpathy et al.[14])

mico. Son más presentes en tareas de extracción de conocimiento, debido a la imposibilidad -por definición- de llevar a cabo métodos supervisados. Es habitual su utilización para obtener una primera aproximación a la solución de problemas de aprendizaje supervisado de elevada complejidad.

1.2.1 Redes neuronales artificiales

Existen múltiples algoritmos de aprendizaje supervisado, como la regresión lineal o los k vecinos más cercanos (kNN, *k-nearest neighbors*). Uno de los que más fama han ganado en los últimos años son las *redes neuronales* artificiales.

Se conoce por redes neuronales a un tipo de sistemas formado por nodos interconectados unos con otros. Un ejemplo de red neuronal sencillo se muestra en la Figura 5. El epíteto *neuronales*, se debe a que a estos nodos también se los denomina neuronas. La razón no es tanto que estos nodos hagan (o traten de hacer) lo que una neurona, sino más bien la apariencia visual que tienen al representarlas en forma de grafos. Cada nodo de cálculo, en general, recibe múltiples entradas x_i, las cuales pondera con unos pesos w_i y suma.También puede sumar un término de compensación o sesgo b. Al valor de la suma se le aplica una función no lineal o *función de activación* y el resultado final es el valor de salida. La analogía con la fisiología neuronal es (I) dendritas-entradas, (II) excitación celular-función de activación, y (III) axón-salida (6).

Las redes se estructuran por capas, cada una compuesta por una o más neuronas. De este modo, una neurona artificial recibe varias señales de otras neuronas de la capa inmediatamente anterior y transmite una sola señal a

una o varias neuronas de la capa inmediatamente posterior, formando la red.

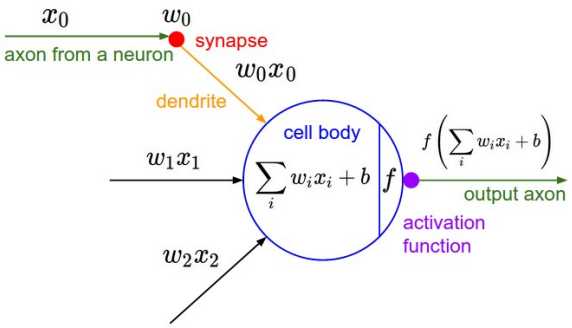


Figura 6: Esquema de una neurona computacional con sus analogías respecto de una neurona biológica. (Crédito Karpathy et al.[14])

Los datos disponibles para configurar la red (ejemplos etiquetados con la respuesta correcta) se suelen dividir en tres grupos:

- Muestra de entrenamiento (*training set*). Se utiliza en la fase de optimización de la red. La red procesa los objetos de la muestra como si fuese un caso real y comprueba si sus predicciones se corresponden con las etiquetas. En función del resultado se reconfiguran los parámetros de la red.
- Muestra de test (*test set*). Una vez ha finalizado la configuración del algoritmo se procesan una sola vez los ejemplos de la muestra de test para comprobar qué rendimiento tiene la red con datos nuevos a los que no ha visto aún.
- Muestra de validación (*validation*). Tras efectuar un intento de entrenamiento sobre la muestra de *training*, se puede evaluar la precisión de la red con la muestra de validación y comprobar si está progresando adecuadamente.

La red es alimentada con ejemplos (vectores) de la muestra de entrenamiento, y computa todos los cálculos neurona por neurona y capa por capa hasta llegar a la salida. Entonces compara el resultado del cálculo por la red y la respuesta incluida en su etiqueta. Se define una *función de pérdida* o *de coste* (*Loss/cost* function) $\mathcal{L}$ que da cuenta de lo lejos que se encuentra la

predicción de la realidad. Esta función no tiene por qué ser siempre igual. Un ejemplo puede ser el error cuadrático medio, que toma el valor real y el dado por una función y calcula el cuadrado de la diferencia.

Entonces, de modo similar a como se haría un ajuste por mínimos cuadrados, se trata de encontrar los parámetros (o pesos) que hagan que la función de loss se encuentre en el mínimo.

Para hacer esto, el método de optimización utilizado es el descenso de gradiente. El gradiente ∇ de una función ($\mathcal{L}$) tiene la dirección de la máxima tasa de variación de la función en el punto. Así, por este método se modifican los parámetros en el sentido opuesto del gradiente, haciendo que el valor de $\mathcal{L}$ disminuya.

Como las redes pueden llegar a ser muy complejas, y las funciones de loss no poderse expresar fácilmente de manera analítica, el cálculo se hace gradiente con un método numérico y de forma semejante a la regla de la cadena.

Una vez que se ha procesado un ejemplo y se halla el valor de $\mathcal{L}$, se calcula el gradiente respecto a los parámetros de la última capa, y el gradiente de estos respecto de la capa anterior, y así sucesivamente hasta llegar a la primera capa. Es la *backpropagation*. Una vez que se ha calculado el gradiente, los pesos se actualizan en función de él para reducir la función de pérdida.

El proceso de cómputo, cálculo de gradiente y actualización puede repetirse de nuevo para seguir mejorando la precisión de la red. Cada uno de las iteraciones en que se procesa la totalidad de la muestra de entrenamiento y se actualizan los pesos se denomina *época*.

El descenso de gradiente puede aplicarse, en general, de 3 formas:

- *Stochastic Gradient Descent* (SGD). Se procesan los objetos de la muestra de uno en uno, actualizando sucesivamente los parámetros. Mediante este método el aprendizaje es paulatino, pero el hecho de calcular el gradiente y actualizar a cada paso requiere más recursos computacionales y más tiempo. Al tomar cada vez un ejemplo diferente (estocástico) hace que la red se desplace mucho por el espacio de parámetros, lo que puede ralentizar la convergencia hacia el mínimo de $\mathcal{L}$. Sin embargo, esto también puede ayudar a evitar mínimos locales y asegurar la

optimización.

- *Batch* (de lote). Por este método se calcula la contribución a la función de coste para todos los ejemplos de la muestra de entrenamiento, y posteriormente se actualizan los parámetros solamente una vez. Por este método se evita la mayor parte del cálculo relativo a la actualización de pesos. En cambio, se necesita almacenar el gradiente para toda la muestra entre cada época. Al contrario que en SGD, al calcular el gradiente con toda la muestra, su convergencia es más rápida, ya que no atiende a los valores particulares de cada objeto. Otra de sus contrapartidas es que el aprendizaje se produce a saltos, y para muestras grandes puede resultar difícil evaluar el progreso.

- *Mini-batch*. Se trata de un punto medio de compromiso entre los dos anteriores. En este caso se procesan varios ejemplos de la muestra y se modifican consecuentemente los parámetros, repitiendo el proceso hasta recorrer toda la muestra de entrenamiento. Es más rápido que SGD y más fiable que con batch.

El proceso de entrenamiento puede llevar a un defecto de *overfitting* (sobreajuste), es decir, que la función de coste se optimiza teniendo en cuenta irregularidades en los datos debidas a valores atípicos o dispersos. Para evitar esto se toman medidas de *regularización*, cuya aplicación implica que en el proceso de aprendizaje parte de este se pierde para favorecer soluciones más sencillas. Algunas ejemplos son la regularización *L2*, que introduce un término en $\mathcal{L}$ que penaliza los pesos con valores elevados; o el *dropout* (abandono), que aleatoriamente desactiva algunas neuronas temporalmente en cada paso del entrenamiento.

1.2.2 Redes convolucionales

Dentro de las redes neuronales, las *redes neuronales convolucionales* o, simplemente, redes convolucionales son uno de los motivos por los que el campo del aprendizaje automático está creciendo tanto. En 2012, una red convolucional ("AlexNet")[15] diseñada por el equipo de Alex Krizhevshy ganó el reputado concurso de reconocimiento visual de ImageNet, ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) con un porcentaje de error 10,8 puntos mejor que el segundo y 9,8 puntos mejor que el ganador del año 2011. La mejora del año anterior había sido de solo 2,4.

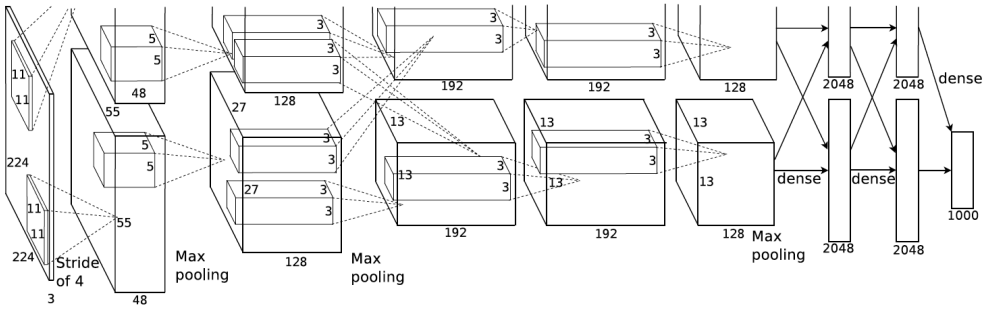


Figura 7: Diagrama de la red AlexNet. A la entrada de una imagen cuadrada se le aplican 2 grupos de 48 filtros convolucionales que se propagan por dos flujos diferentes con agrupamientos y convoluciones hasta las 3 capas *Fully Connected* finales.(Krizhevsky et al.)

En las redes convolucionales entre capa y capa de neuronas, en lugar de introducir una matriz de pesos que relacione todas las neuronas de una capa con todas las neuronas de la otra, se determina un *filtro*. Cada filtro tiene un tamaño determinado (de tamaño máximo el de la capa a la que aplica, pero normalmente mucho menor), con unos pesos que sirven para seleccionar las señales esperadas. Este filtro se aplica sobre toda la capa anterior en diferentes posiciones y cada posición da lugar (función de activación mediante) a un valor en la capa posterior. Según las distintas configuraciones, entre cada dos aplicaciones del filtro este se puede desplazar 1 o más neuronas cada vez, aplicando un "paso" (*stride*) mayor o menor.

Al utilizar un conjunto reducido de pesos que se aplican repetidamente para toda la capa, este método se ahorra manejar matrices más pesadas.

El motivo de que a este tipo de redes se le llame "convolucionales" se debe a que se ejerce un filtrado componiendo la señal de entrada con el filtro. Tienen un gran rango de aplicación en campos como el procesamiento de imagen o de señal, por esta razón. No obstante, cada vez se aplican a casos más dispares.

También es habitual, tras varias capas convolucionales, incluir una capa de *pooling* o agrupamiento, que realiza una operación de reducción de resolución (*downsampling*). Agrupa los valores que recibe (p.ej. en regiones de 2×2) y pasa un solo valor que usualmente es el máximo (*maxpool*).

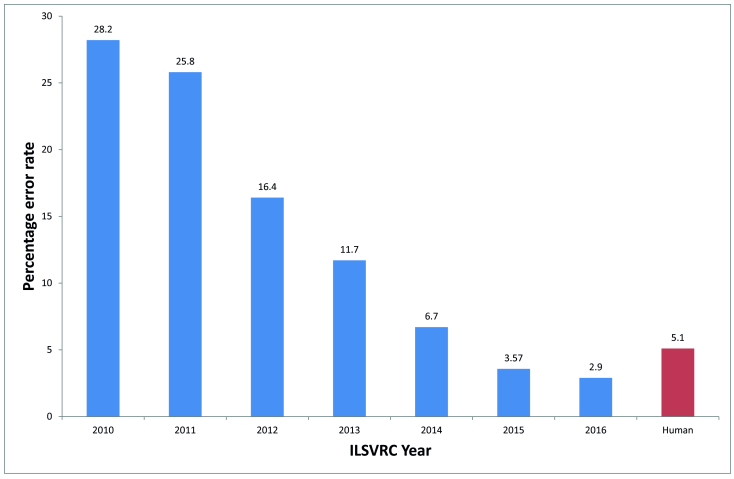


Figura 8: Evolución de la precisión de la propuesta ganadora de ILSVRC en función del tiempo. También se muestra una referencia con la efectividad de una persona entrenada para clasificar el mismo tipo de imágenes.(Crédito: Karpathy et al.[14])

La eclosión de esta clase de redes neuronales en los últimos años es gracias, entre otras cosas, a la industria del videojuego. El desarrollo de juegos con gráficos de cada vez mayor calidad y mejor tiempo de respuesta ha llevado a las empresas a desarrollar para los computadores de videojuegos (PCs y consolas) tarjetas gráficas (GPU) muy potentes. Estas, a diferencia de los microprocesadores centrales (CPU) de la máquina, no se enfocan a resolver muy rápido operaciones secuenciales, sino a realizar muchos cálculos en paralelo (píxeles de una sola imagen, por ejemplo) aunque sea más despacio. En lugar de tener unos pocos núcleos de procesamiento muy eficientes, integran varios centenares más sencillos.

La estructura de estos dispositivos se adapta muy bien al cálculo de operaciones vectorizadas como puede ser aplicar los pesos a todas las neuronas de una capa.

2 Producción de Montecarlo realista

2.1 La importancia de los fondos

En física de altas energías, parte importante del esfuerzo de investigación consiste en las búsquedas de nueva física[16], como la supersimetría o la materia oscura, y las nuevas partículas de estos modelos se buscan en los sucesos de los colisionadores.

Normalmente, en búsquedas de nueva física, las partículas que se buscan son indetectables. Por una u otra razón, estas partículas no interaccionan con los detectores; de no ser así, es probable que ya se hubiesen detectado. Para este tipo de partícula, la señal típica en el detector es que aparezca un desequilibrio en la energía-momento transversal o *Missing ET*. Sin embargo, también hay partículas invisibles en el Modelo Estándar, como son los neutrinos. Normalmente se hace un análisis de la MET, representando su distribución para todos los sucesos medidos. En caso de que hubiera nueva física, que en estos sucesos incluyera otra partícula nueva invisible, la distribución cambiaría ligeramente. Si las distribuciones fueran restringidas y suficientemente distintas bastaría con seleccionar los sucesos a estudiar en base a la MET, quedándose solamente con aquellos que quedasen fuera del modelo.

El problema es que el modelo estándar da una distribución con una cola muy extendida y de baja estadística, por lo que es más difícil hacer algún contraste. Además, los sucesos de nueva física suelen tener poca probabilidad, y no se puede distinguir claramente de la cola del modelo estándar.

Si se conociese a ciencia cierta lo que se espera medir del modelo, aunque las posibles señales de nueva física fuesen escasas, se podría al menos discutir si los datos obtenidos se desvían o no del modelo. Esto es lo que ahora se aborda mediante métodos de simulación de Montecarlo. Conocido el modelo, se simulan millones de sucesos en las mismas condiciones y se genera la distribución esperada, para comparar con los datos experimentales. Una desviación de los datos respecto del modelo con suficiente significancia estadística supondría un signo de nueva física.

De nuevo, en esta suposición hay otro problema implícito, y es que se asume conocer el modelo de manera perfecta para realizar la simulación.

Aunque las ecuaciones del modelo se conocen perfectamente, al realizar la simulación también se debe modelizar el detector (con un tamaño comparable a un edificio de 4 plantas), todo el cual es susceptible de interaccionar con las partículas, y cualquier mínimo detalle de su estructura que no se tenga en cuenta podría causar efectos imprevistos. La simulación del detector es una buena aproximación, pero no es perfecta. Como cualquier simulación, hay un cierto nivel de abstracción, ya que es imposible simular átomo por átomo.

Una vez hecha la simulación es cuando se comparan los datos. Para poder afirmar que se ha observado una señal, se debe tener una gran confianza en que los datos del Montecarlo reflejan realmente la respuesta esperada del modelo. Es preciso también asegurarse de que la razón por la que se observa un exceso en los datos es la existencia de nueva física y no algún efecto que no ha sido contemplado en simulación (por ejemplo que una zona del detector esté midiendo algo diferente de lo esperado).

2.2 Propuesta de Solución: Estilo

Tras la irrupción de los métodos de aprendizaje automático en el día a día de la investigación, han surgido todo tipo de aplicaciones en múltiples disciplinas. Existe un algoritmo [17] con redes neuronales convolucionales que da solución a un problema interesante. En ese trabajo se entrena el algoritmo con imágenes de obras de pintores reconocidos y el programa extrae el estilo. En este caso, el *estilo* se corresponde con las correlaciones de corto alcance. Los píxeles de una pequeña región A de la imagen están correlacionados de una manera muy diferente, en general, de los de otra región B en el otro extremo de la imagen.

En cambio, el *contenido* de la imagen tiene que ver con correlaciones a gran escala. Entre una fotografía y una imagen artística que representan una misma escena, detalles como los colores o los trazos pueden diferir, pero los objetos que se representan tienen formas, localizaciones y relaciones similares.

Con este método, se extrae el estilo de una pintura, y se consigue generar una nueva imagen con el contenido de una fotografía, pero el estilo pictórico. Un ejemplo de esta aplicación se ofrece en la Figura 9.

En concreto, en el artículo de referencia [17], utilizan una red convolucional usando como entrada un par de imágenes: la foto de estilo y la foto

(a) Palacio de la Magdalena, Santander. Aporta el contenido.

(b) *Amapolas* (C. Monet). Aporta el estilo.

(c) Recreación

Figura 9: Ejemplo de aplicación del estilo pictórico a una fotografía mediante una red convolucional. Las imágenes originales que aportan el contenido y el estilo se muestran en (a) y (b) respectivamente. El resultado puede verse en la subfigura (c).

de contenido que se quiere modificar. Para generar la imagen se comienza a partir de una foto con ruido blanco gaussiano. La función de pérdida se define como la suma ponderada de dos funciones de pérdida a las que llamaremos función de pérdida de contenido, y función de pérdida de estilo[17]:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}. \tag{1}$$

La función de pérdida de contenido simplemente compara las activaciones en cada neurona y capa con las activaciones que produciría la foto de contenido:

$$\mathcal{L}_{content} = \frac{1}{2}\sum_{i,j}(F^l_{i,j} - P^l_{i,j})^2 \tag{2}$$

donde $F^l_{i,j}$ y $P^l_{i,j}$ son las activaciones del i-ésimo filtro en la posición j-ésima de la capa l para la imagen generada y la original con el contenido respectivamente. La minimización de dicha función de pérdida por descenso de gradiente respecto de la imagen de entrada modificando sus valores en la dirección de convertir la entrada inicial de ruido blanco en la imagen de contenido. Por otra lado, la función de pérdida de estilo, no compara las activaciones, sino que compara las correlaciones existentes entre las activaciones de los filtros de cada una de las capas consigo mismas:

$$\mathcal{L}_{style} = \sum_{l=0}^{L} w_l E_l, \tag{3}$$

donde la contribución de cada capa es

$$E_l \propto \sum_{i,j}(G^l_{i,j} - A^l_{i,j})^2, \tag{4}$$

con $G^l_{i,j}$ y $A^l_{i,j}$ siendo los productos internos entre los vectores de activaciones (del i-ésimo y del j-ésimo filtros) para la imagen generada y la original con el estilo respectivamente. Al hacer una función de pérdida que considera estas dos funciones, el efecto del entrenamiento sobre una imagen neutra da lugar a una foto parecida en contenido, sin embargo, el efecto de la otra función hace que las correlaciones entre filtros, y por lo tanto el estilo sea también parecido.

En conexión con la física de altas energías, el realismo al nivel de detalle que presenta el detector frente al modelo con el que se simula, puede ser interpretado como el *estilo* particular del detector. Así, se tiene el modelo estándar del que se espera una distribución Q (que sería el verdadero contenido), pero el detector aplica una serie de correlaciones a pequeña escala

(como pudiera ser una eficiencia disminuida para detectar los electrones, que tuvieran un pequeño error sistemático sin corregir).

Partiendo de estas consideraciones, la idea que se presenta consiste en entrenar una red neuronal del mismo tipo, con datos reales de colisiones del LHC. La red aprendería el estilo del detector, lo que abriría la posibilidad de generar (de forma similar a las fotografías con estilo pictórico) simulaciones Montecarlo que mostrasen lo que el detector realmente mida. Así, a la hora de efectuar el análisis de los datos frente a la simulación, el grado de confianza en esta sería mayor, y las conclusiones más firmes.

Por ello, el objetivo fundamental de este trabajo consiste en comprobar la viabilidad de esta idea con datos de simulación.

3 Creación de RRNN sencillas usando Keras y Tensorflow

3.1 Introducción: TF y Keras

El auge de las redes neuronales en la última década, ha propiciado el desarrollo de multitud de *bibliotecas* (colecciones de funciones informáticas) destinadas a ser utilizadas en aplicaciones de machine learning.

Google (cuyo modelo de negocio radica en estudiar, analizar y explotar el comportamiento en la Red de sus usuarios), como no podía ser de otra manera, ha dedicado una parte importante de sus esfuerzos recientes a profundizar en esta tecnología. El resultado fue *TensorFlow*, una biblioteca concebida para optimizar el desarrollo de sistemas de machine learning. Su núcleo de ejecución está desarrollado en C++, pero la cuenta con una interfaz de programación en Python, lo que facilita su aplicación. El nombre refleja sus características, que son el manejo de datos en arreglos multidimensionales (*tensores*) y una cálculo estructurado según diagramas de flujo.

Al contrario que la mayoría de bibliotecas existentes hasta su aparición, desarrolladas para su empleo en labores de investigación, esta ha sido diseñada desde el principio (basada en la experiencia previa de Google en machine learning) para un objetivo global. Presenta la posibilidad de, con mínimos

cambios en el código, habilitar el diseño de las redes neuronales para su ejecución en paradigmas de computación distribuida. Es capaz de repartir el cálculo entre unidades CPU y GPU, entre varias GPU, o distintas máquinas, en caso de disponer de ellas. Esto abre las puertas a métodos de *paralelismo de modelo*, en que diferentes unidades entrenan diferentes partes del algoritmo; y de *paralelismo de datos*, en que diferentes unidades entrenan el modelo con distintos conjuntos de datos.

Keras es otra biblioteca de código abierto dedicada a las redes neuronales, y que funge como interfaz entre el usuario y otras bibliotecas como TensorFlow, Theano, o CNTK(Microsoft). Supone una abstracción de más alto nivel de las posibilidades del motor sobre el que funciona (TensorFlow) y que facilita y agiliza el desarrollo y depuración de redes neuronales. Esto permite una retroalimentación más eficiente con los resultados en el proceso de investigación.

3.2 Red sencilla para discriminar poblaciones

Para comenzar a comprobar las posibilidades las herramientas planteadas, se empieza por un ejemplo sencillo y más fácil de controlar. Se trabaja con programas en lenguaje Python haciendo uso de las bibliotecas *NumPy* (para cálculos con matrices) y *matplotlib* (para representación gráfica).

En este primer caso se trabaja con conjuntos (o categorías) de datos generados aleatoriamente a partir de dos gaussianas bidimensionales. Cada muestra contiene 5000 pares de valores. Se crea una muestra de entrenamiento y otra de prueba para las dos categorías. Los datos se extraen de gaussianas con media μ y matriz de covarianza Σ. Para la primera categoría los valores son

$$\mu_1 = \begin{pmatrix} 20 \\ 20 \end{pmatrix} \text{ y } \Sigma_1 = \begin{pmatrix} 10 & 9 \\ 9 & 15 \end{pmatrix},$$

y para la segunda categoría son

$$\mu_2 = \begin{pmatrix} 30 \\ 10 \end{pmatrix} \text{ y } \Sigma_2 = \begin{pmatrix} 10 & -9 \\ -9 & 25 \end{pmatrix}.$$

Haciendo uso de Keras, se crea una red neuronal a la que se le pasan los datos generados, éstos se almacenan en *numpy arrays* (el objeto fundamental

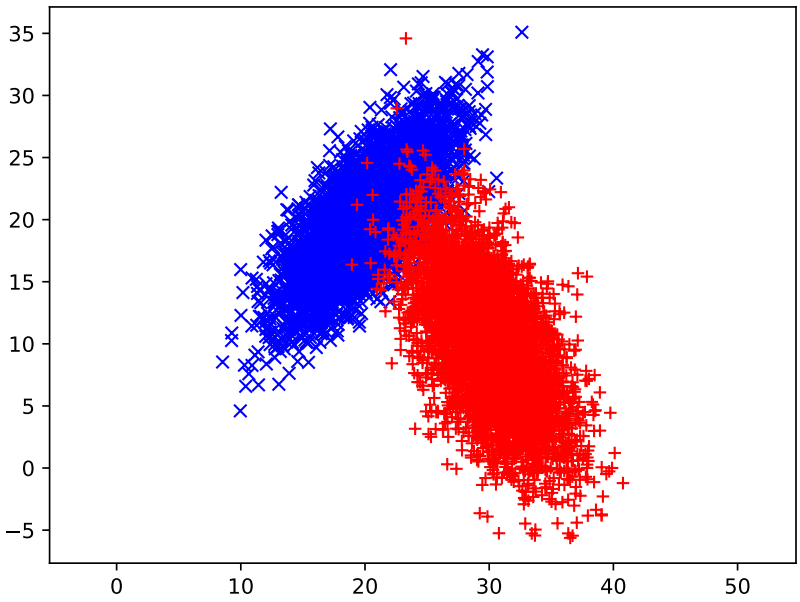


Figura 10: Representación gráfica de los tipos de datos extraídos de gaussianas que se pasan a la red sencilla. Los datos marcados con "X" corresponden a la primera categoría y los marcados con "+" a la segunda.

de esta biblioteca). Los datos se preparan añadiéndoles etiquetas en función de su categoría.

La red es de tipo perceptrón multicapa, esto es, tiene una o varias capas ocultas y todas las neuronas de una capa reciben señal de todas las neuronas de la capa anterior (este tipo de capa se llama *Fully Connected* o FC). En este caso consta de la capa de entrada, con 2 nodos (se pasan pares de valores$\{x, y\}$), 2 capas ocultas de 8 y 4 neuronas respectivamente y una capa de salida de 2 neuronas (puesto que se quieren distinguir 2 categorías). La función de activación para las neuronas ocultas es un rectificador de tipo *ReLU* (por *Rectified Linear Unit*) cuya expresión es:

$$f(x) = max(0, x),$$

es decir, que si la señal es positiva la conserva y si es negativa la anula. También se aplica, como medida de regularización, una tasa de abandono (*Dropout*) de 0,4 .

En la capa de salida la función de activación es de tipo sigmoide:

$$\sigma(x) = \frac{e^x}{e^x + 1}.$$

nº c. ocultas	época	loss	acc	val-loss	val-acc
1	1	0,4500	0,7822	0,2032	0,9723
	2	0,4424	0,7856	0,1929	0,9741
2	1	0,5054	0,7547	0,1776	0,9773
	2	0,3226	0,8737	0,1226	0,9770
3	1	0,5683	0,6567	0,0907	0,9733
	2	0,4511	0,7543	0,1562	0,9723
4	1	0,5106	0,7412	0,1779	0,9762
	2	0,4060	0,8354	0,1572	0,9762

Tabla 1: Resultados del aprendizaje de la red separadora de gaussianas en función del número de capas ocultas y para cada época del entrenamiento. Se muestran los valores de la función de *loss* y de la precisión (*acc*) para el entrenamiento y la validación (val-).

Se prueba variando el número de capas ocultas desde 1 hasta 4. Cada nueva capa se introduce a continuación de la capa de entrada y tienen, respectivamente, 4, 8, 12 y 16 neuronas; ampliando el número de parámetros intermedios. Se observa que el mejor rendimiento es con 2 capas ocultas.

Tras entrenar a la red con los datos preparados y recorrer dos veces la muestra de entrenamiento, en la validación la función de pérdida sube de $\mathcal{L} = 0{,}1391$ a $\mathcal{L} = 0{,}1654$, mientras que la tasa de acierto mejora de $r = 0{,}9854$ a $r = 0{,}9883$.

Los resultados para cada número de capas se muestran en la Tabla 1.

Como se puede observar, ya con 2 capas ocultas la red alcanza un máximo en la ratio de aciertos, y en las redes con mayor número de capas no se mejora. Esto quiere decir que para esta arquitectura, a partir de 2 capas ocultas es difícil obtener mejores resultados. Las capas adicionales dotan a la red de mayor capacidad de separación, pero los dos grupos de datos no tienen una frontera demasiado particular. Las redes más profundas acaban aprendiendo de características peculiares de la muestra de entrenamiento, lo que les hace perder rendimiento en la validación. Esto es un ejemplo de *overfitting*.

3.3 Red para discriminar sucesos de Altas Energías

Tras realizar el primer estudio para un caso sencillo, se pone a prueba este tipo de arquitectura de red para clasificar datos provenientes de CMS. Una categoría son datos de simulaciones del proceso $t\bar{t}$ realizadas con el simulador aMC@NLO. La otra categoría son datos de sucesos simulados para el modelo de producción $t\bar{t}$ + DM usando el mismo simulador. En particular, se simula el caso en que el mediador tiene una masa de 100 GeV y la interacción es de tipo pseudoescalar.

El objetivo es poner a prueba este tipo de red neuronal y comprobar cuál es su capacidad a la hora de distinguir la señal de producción de materia oscura respecto de sucesos de un único tipo de fondo pero con una signatura muy similar como es $t\bar{t}$.

Los datos se obtienen de la simulación en formato ROOT. ROOT es un entorno de análisis de datos (y su formato asociado) desarrollado por el CERN para su trabajo en física de partículas. Los archivos obtenidos (uno para cada tipo de suceso) contienen tantas *n-tuplas* de los valores medidos (o simulados) para n variables como sucesos se incluyan. Las variables incluidas son la magnitud del momento transverso p_T, el ángulo ϕ, la seudorapidez η y la masa m de las partículas identificadas -en este caso leptones y jets asociados a quarks b (*b-jets*-. También se incluyen estas variables para la MET.

Se escribe una rutina de Python (dumpTry.py) que extrae de los archivos de ROOT los datos de p_T y ϕ de los leptones y los b-jets y el momento de la MET. Calcula la diferencia $\Delta\phi$ entre los ángulos de los dos leptones y la de los dos jets. Finalmente, se almacenan en un array los datos de p_T de la MET, los leptones y jets y $\Delta\phi$ para leptones y jets. Este array es el retorno de la rutina.

Con Keras se construyen dos redes similares a la anterior, pero ahora con 7 datos de entrada y de 2 y 3 capas ocultas respectivamente (con 4 y 4 neuronas y 7, 4 y 7 neuronas respectivamente). Se entrena la red en 3 épocas. Dadas las características del problema, se pone 1 sola neurona en la capa de salida, ya que al haber 2 categorías se trata de un caso binario, y la menor complejidad de esta opción favorece la convergencia. El resto de hiperparámetros se dejan igual. Las variables de input son las descritas anteriormente: el momento transverso de ambos leptones, de ambos jets, y el

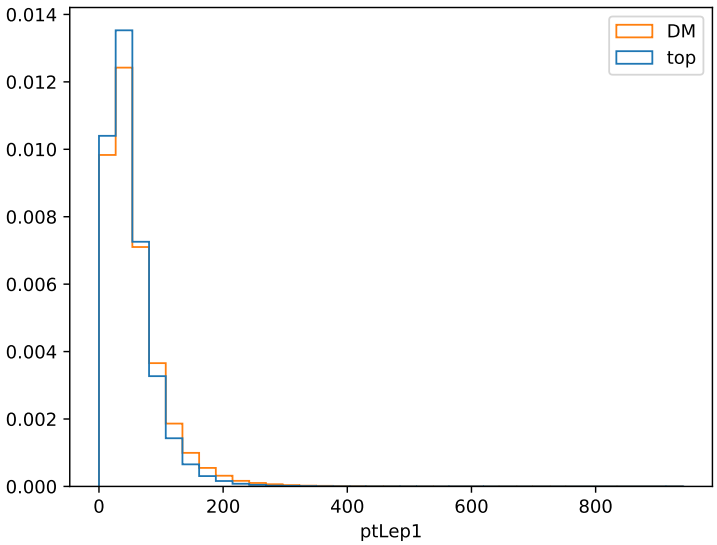


Figura 11: Histograma de las magnitudes de los momentos transversos para uno de los leptones de cada pareja.

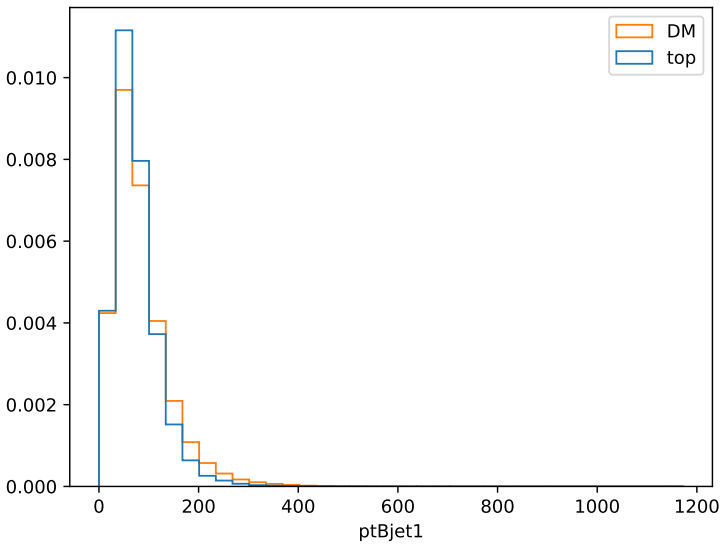


Figura 12: Histograma las magnitudes de los momentos transversos para uno de los jets de cada pareja.

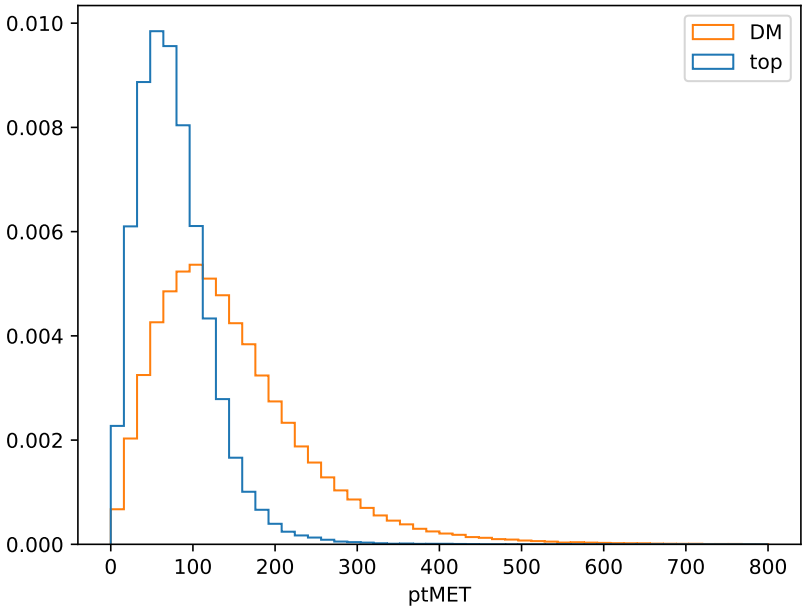


Figura 13: Histograma de la distribución del momento transverso ausente para los dos tipos de sucesos.

ausente ($p_{T,l1}$, $p_{T,l2}$, $p_{T,b1}$, $p_{T,b2}$, $p_{T,Miss}$); y el ángulo entre los leptones y entre los jets ($\Delta\phi_l$, $\Delta\phi_b$). Los histogramas de estas variables se representan en las Figuras 13, 14, 15, 11 y 12. Las distribuciones para $p_{T,l2}$ y $p_{T,b2}$ se omiten por ser redundantes con las Figuras 11 y 12.

Un resumen de los resultados se muestra en la Tabla 2.

El mejor resultado que se obtiene es de una tasa de acierto de $r = 0{,}714$. Este resultado es tan solo ligeramente mejor que el obtenido al efectuar un corte en las distribuciones de $p_{T,Miss}$, el momento transverso de la MET. A partir del análisis de la distribución de $p_{T,Miss}$, el corte se puede efectuar clasificando como $t\bar{t}$ todos los sucesos que se hallan en el rango en que la distribución de $t\bar{t}$ supera a la de producción de DM, y viceversa. En este caso el umbral de separación se sitúa en $p_{T,Miss} = 110\,\mathrm{GeV}\,c^{-1}$. Con este método se obtiene un acierto de $r = 0{,}701$.

Como se ha planteado un caso binario, se hace uso de un análisis de curva ROC (*receiver operating caracteristic*), mediante el que se representa la

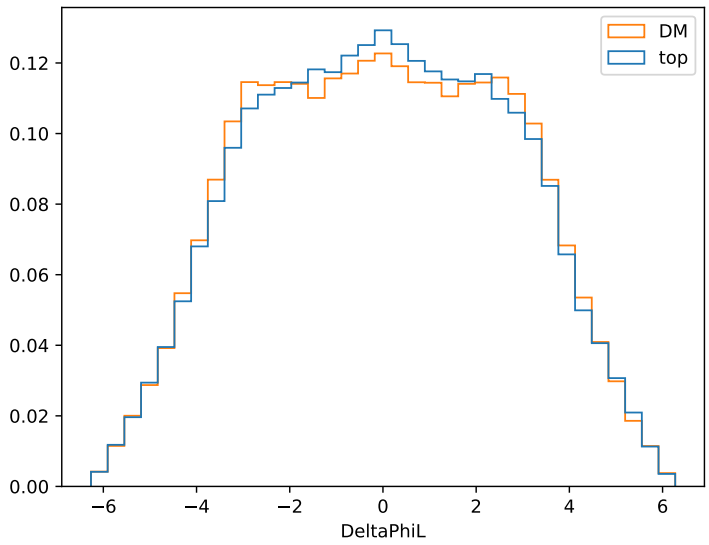


Figura 14: Histograma del ángulo de separación entre los leptones para los dos tipos de sucesos.

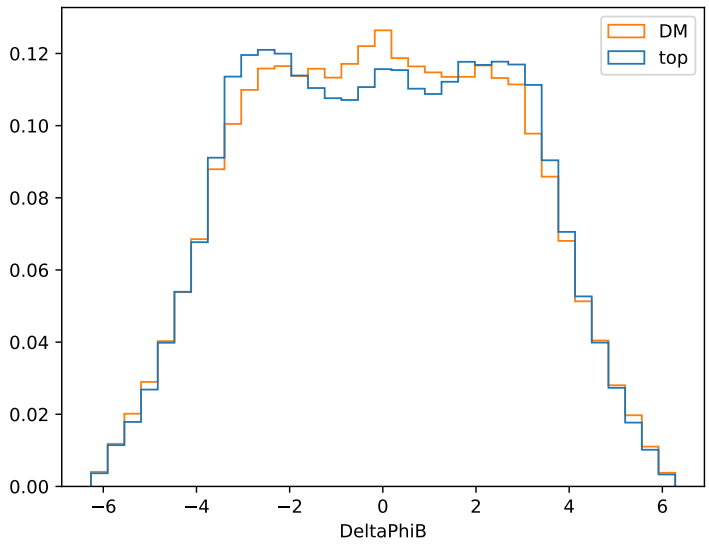


Figura 15: Histograma del ángulo de separación entre los b-jets para los dos tipos de sucesos.

nº c. ocultas	época	loss	acc
2	1	0,5978	0,7068
	2	0,5190	0,7130
	3	0,5168	0,7146
	Test	0,5118	0,7143
3	1	0,6322	0,7010
	2	0,5179	0,7140
	3	0,5167	0,7130
	Test	0,5153	0,7133

Tabla 2: Resultados del aprendizaje de la red separadora de sucesos de altas energías en función del número de capas ocultas y para las épocas del entrenamiento y para el test. Se muestran los valores de la función de *loss* y de la precisión (*acc*).

sensibilidad o tasa de verdaderos positivos frente a la tasa de falsos positivos. Esto permite analizar todos los puntos de trabajo del clasificador. La curva ROC se muestra en la Figura 16.

En la Figura 16 se observa que el comportamiento de las dos redes es equivalente. Tienen un buen rendimiento, pero no óptimo.

La red tiene dificultades para encontrar las características óptimas para clasificar. Aparentemente, la única característica de la que extrae fácilmente información es ptMET.

Añadir profundidad a la red no parece implicar mejoras. Al contrario, el rendimiento de la red se estanca con el entrenamiento. Observando las distribuciones de las variables que se muestran en las Figuras 13, 14, 15, 11, y 12 se puede decir que -aparte de la $p_{T,Miss}$- su apariencia es muy similar (aún mas para las distribuciones de momentos). Esto puede explicar la dificultad del aprendizaje.

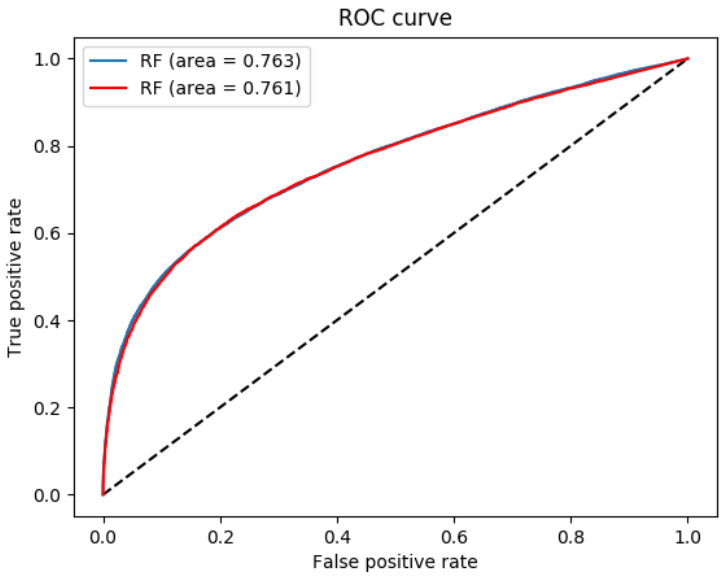


Figura 16: Curva característica operativa (ROC) de la red clasificadora para sucesos de $t\bar{t}$ y producción de materia oscura.

4 Representación de sucesos de Altas Energías como imágenes

4.1 Expresar un suceso en una foto

Dado que la red extractora de estilo procesa imágenes, se busca la forma de reflejar en forma de píxeles en un archivo PNG O JPEG los datos de las variables de que se dispone.

Haciendo uso de *TLorentzVector*, un módulo de ROOT en Python para hacer cálculos con cuadrivectores, se pueden obtener algunas variables secundarias. Las variables primarias (las que corresponden a detecciones) p_T, η, ϕ y m se extraen de modo similar al primer método, y con ellas se construyen los cuadrimomentos correspondientes a los dos leptones, los dos b-jets y a la MET.

Con los cuadrimomentos se obtienen fácilmente variables asociadas a otros sistemas de referencia, como son la masa del sistema de 2 leptones

m_{ll}; las masas en la componente transversa de los sistemas MET + leptón para cada uno de los leptones m_{T,l_1MET} y m_{T,l_2MET}; la masa de los dos jets m_{jj}; el momento transverso de los dos leptones $p_{T,ll}$ y el de los dos jets $p_{T,jj}$; así como la suma de las magnitudes del momento transverso de los jets H_T y la suma de magnitudes del momento transverso de jets y leptones S_T.

Tras este proceso se obtiene un total de 21 variables, 12 primarias (las asociadas directamente a cada uno de los leptones y los jets: p_T, η y ϕ, se excluye la masa) y 9 secundarias.

Se diseña un formato en que se construya para cada evento un bloque de 12×9 píxeles. En cada una de las 12 filas se introduce una de las variables primarias, ocupando el canal rojo en el modelo RGB para los 9 píxeles de esa fila. De forma análoga para las variables secundarias repartidas en las 9 filas en el canal azul, ocupando cada una los 12 píxeles de una columna. De esta forma cada píxel tendría la información de una variable primaria en el rojo y una variable secundaria en el azul.

Sin embargo, puesto que queda disponible el canal verde, se replantea darle uso y "repartir" sus 8 bits entre los otros dos canales, de forma que se tenga un rango más denso de representación. Se pasa de 8 bits (enteros de 0 a 255) a 12 bits (enteros de 0 a 4095) para cada variable. Como debemos ajustar valores reales a una escala cuantizada, se analizan las variables y se normalizan sus valores según sus cotas superior e inferior, adecuándolos a la escala [0,4095].

Una vez se tienen los valores en la nueva escala, se traducen a sistema binario, se redistribuyen en tres bytes, y se vuelven a traducir en valores decimales, ubicándolos en los colores RGB.

En las Figuras 17 y 18 se muestran ejemplos de las imágenes creadas por este método para los dos tipos de eventos.

4.2 Aplicación de sesgos de forma controlada

Para poder comprobar la capacidad de entrenamiento de la red, es preciso realizar una prueba controlable, restringiendo los grados de libertad.

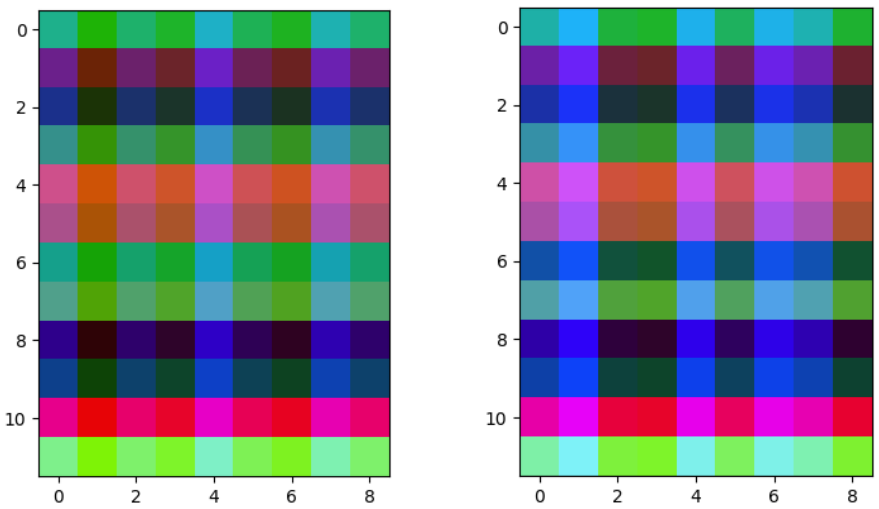
Se propone un ejemplo de aplicación en que el detector responda recons-

truyendo los momentos de los b-jets en valores más bajos de los reales. Se añade para ello una variable de sesgo (p.ej. bias = 0,5).

Tras obtener los datos y antes de calcular las variables secundarias, se le aplica este factor de sesgo a los momentos p_{T,b_1} y p_{T,b_2}. Con esta modificación, además de estas dos variables, se deben ver afectados los valores de $p_{T,Miss}$, H_T, m_{jj} y $p_{T,jj}$.

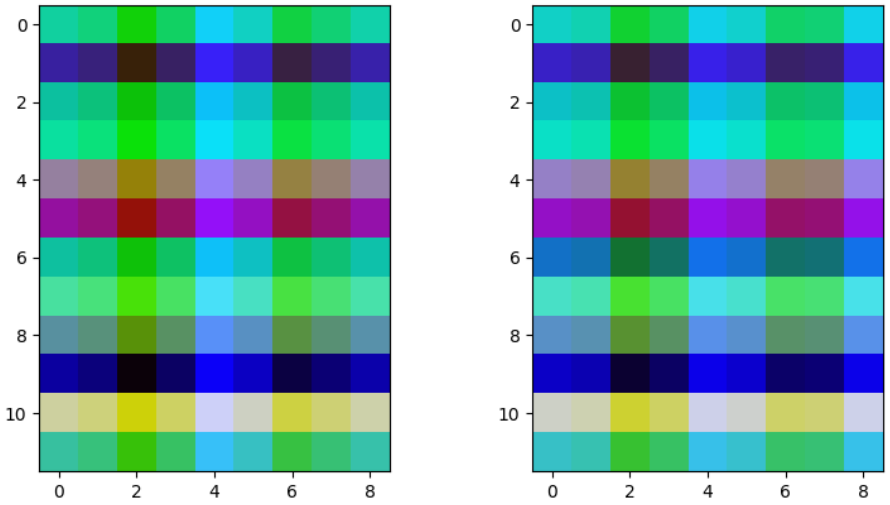
Así, se tiene un caso controlado de aplicación de "estilo" a los datos. Con el mismo proceso que en el apartado anterior, se extraen las variables de los archivos y, antes de obtener las variables secundarias, se aplica el factor a los $p_{T,b}$ y se corrigen manualmente las componentes de $p_{T,Miss}$, ya que su valor es obtenido también del archivo. El resto de variables se calculan como anteriormente, y tanto aquellas que se ven afectadas como las que no toman sus valores correspondientes.

Los ejemplos anteriormente presentados pueden compararse una vez aplicado el factor en las Figuras 17 y 18.



(a) Sin modificar (b) Modificado

Figura 17: Ejemplo de un bloque de píxeles creado para un suceso (Nº 27) del conjunto de $t\bar{t}$ antes (a) y después (b) de haberle aplicado un sesgo a la variable $p_{T,b}$.



(a) Sin modificar (b) Modificado

Figura 18: Ejemplo de un bloque de píxeles creado para un suceso (Nº 35) del conjunto de $t\bar{t}$ + DM antes (a) y después (b) de haberle aplicado un sesgo a la variable $p_{T,b}$.

Una vez se tienen los eventos en forma de imágenes se procede a acondicionar la red neuronal para procesarlos.

5 Aplicación de una CNN a las imágenes

En un contexto en el que el machine learning está cada vez más implantado en la industria y la investigación, y las redes convolucionales, su piedra angular, una herramienta pluripotencial aún con terreno por conquistar, se encuentra una disciplina ávida de encontrar soluciones para sus problemas con un método que es capaz de enfrentarse a los desafíos más diversos.

La idea de tratar de extraer el estilo particular del detector CMS para mejorar el realismo de las simulaciones por Montecarlo es una idea original, que, en lo que sabemos, aún nadie ha probado y merece la pena comprobar su viabilidad.

La red implementada, a imagen y semejanza de la propuesta para el estilo artístico es una modificación al esquema de VGGNET-19 [18], otra red presentada al concurso de ILSVRC consiguiendo la segunda posición, que se ha convertido en uno de los arquetipos por su simplicidad. Tan sólo utiliza capas convolucionales de un tipo, con filtros de 3×3 pixeles. Para esta aplicación, las últimas capas de tipo *Fully Connected*, que se dedican a la clasificación, se eliminan, utilizando la estructura restante como extractor de características.

La estructura de una red como la utilizada puede verse en la Figura 19, muestra la red VGGNET en su modelo de 16 capas con pesos. Con respecto a la Figura 19, se retiran las capas posteriores a la última capa *maxpool* y se añade una tercera capa a las dos primeras series de capas convolucionales.

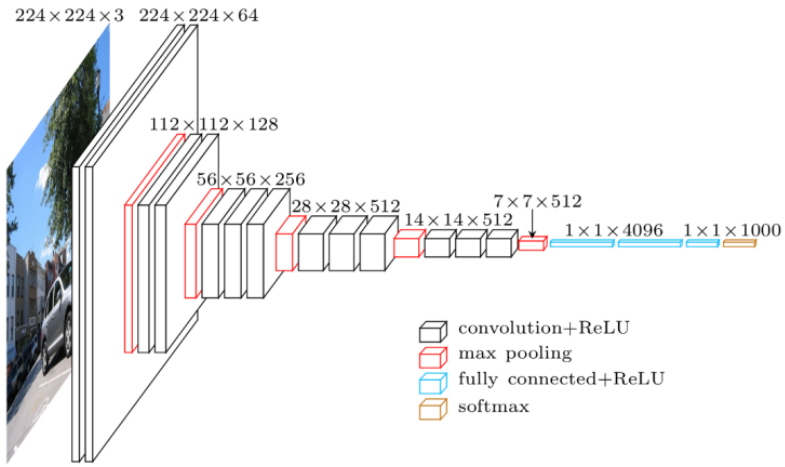


Figura 19: Esquema de la arquitectura de la red convolucional VGGNET-16.[19]

Para poder aplicar el procedimiento de la red extractora de estilo, es necesario generar los datos en un formato compatible con lo que la red espera. Se cuenta con una implementación de la red ya entrenada y optimizada que está escrita con TensorFlow. Los datos de entrada de esta red son imágenes de 300 píxeles de altura y 400 píxeles de anchura. A partir de los bloques generados para cada evento, si se consideran bloques de 12×10 píxeles, se puede formar una composición de 25×40 bloques, y con los datos de 1000 eventos se puede configurar un mosaico y se obtiene una imagen. Así pues, a las variables secundarias consideradas se añade el ángulo de la MET, ϕ_{Miss}.

La especial distribución de algunas variables (*cf.* Figuras 11, 12, 13),

puede dificultar el trabajo, ya que la mayoría de la información se concentra en un rango inferior al disponible. Para tratar de sortear esto, una vez se normaliza cada variable a sus valores extremos, a aquellas con distribuciones más concentradas se les aplica una transformación del tipo:

$$f(t) = \exp(-\alpha \cdot t)$$

donde α es un parámetro entre 5 y 22 en función de la amplitud de cada variable, y t es el valor normalizado.

En esta implementación no se ocupan los 12 bits disponibles por píxel, sino que se utilizan los 8 bits de los canales R y G. La implementación de 3 canales se deja como una posibilidad.

Se construyen las imágenes de 300×400 píxeles y se obtienen sus correspondientes parejas simulando estilo. Una vez hecho esto, las imágenes se le pasa a la red una imagen con los datos puros (que funciona como *contenido*) y otra imagen con datos sesgados correspondiente a otros 1000 sucesos diferentes. El resultado generado por la red debería parecerse a la imagen pareja de la primera con datos manualmente sesgados.

La generación de imágenes se puede llevar a cabo con diferentes proporciones de estilo y contenido. Se consideran casos desde un 20-80 % hasta un 50-50 %. La imagen con el contenido se muestra en la Figura 20.

La imagen de la que se extrae el estilo se muestra en la Figura 21. En la Figura 22 se muestran los resultados de la generación de imágenes con estilo junto con la imagen con el "estilo" aplicado a mano y a la cual deberían parecerse.

A modo de diagnóstico, se representan las distribuciones del momento transverso de uno de los jets para el caso sin modificar, el caso sesgado y el generado por la red (Figuras 23, 24, 25, 26). Como se puede observar, las distribuciones de las imágenes generadas están más próximas a las distribuciones originales que a las manipuladas. El resultado tiene poca variabilidad con la razón estilo-contenido.

Por lo tanto, la técnica parece no ser viable. La razón puede tener que ver con la representación gráfica de los sucesos, considerando las diferencias cinemáticas como de corto alcance. Quizás un ordenamiento diferente de las variables en las imágenes hubiera sido más favorecido.

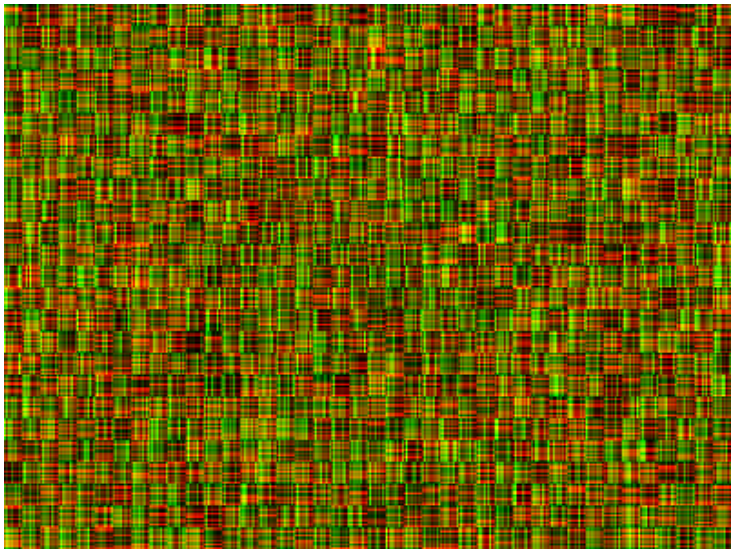


Figura 20: Imagen compuesta a partir de los bloques de los primeros 1000 eventos disponibles del proceso $t\bar{t}$ sin modificar. Proporciona a la red el contenido.

6 Conclusiones

La física de partículas, como uno de los adalides de la investigación, desarrollo e innovación tecnológica y científica no se acaba hallando la última partícula que da sentido a un modelo al que no se deja de poner a prueba. Porque a pesar de ser muy consistente y preciso, sigue dejando fenómenos sin explicar, como el origen de una materia que conforma el 27 % de la energía del universo y más del 80 % de su masa. Por eso en las próximas décadas se van a seguir empujando las fronteras de lo oculto, desarrollando mejores detectores, más precisos y eficientes y generando cada vez más datos y a mayores energías.

Junto con nuevas teorías que nos ayuden a comprender, describir y predecir lo que aún no entendemos ni imaginamos, es imprescindible el desarrollo de nuevas técnicas, nuevas herramientas que proporcionen otro punto de vista desde el que contemplar el mundo. Técnicas de medida y técnicas de análisis.

Algunas de estas técnicas pueden, a todas luces, nacer del machine learning, vistos sus prometedores resultados en el pasado más reciente. Sus aplicaciones están presentes en cada vez más aspectos de nuestra vida cotidiana; tanto más en la investigación científica.

En este trabajo se ha comparado uno solo de los modelos de materia

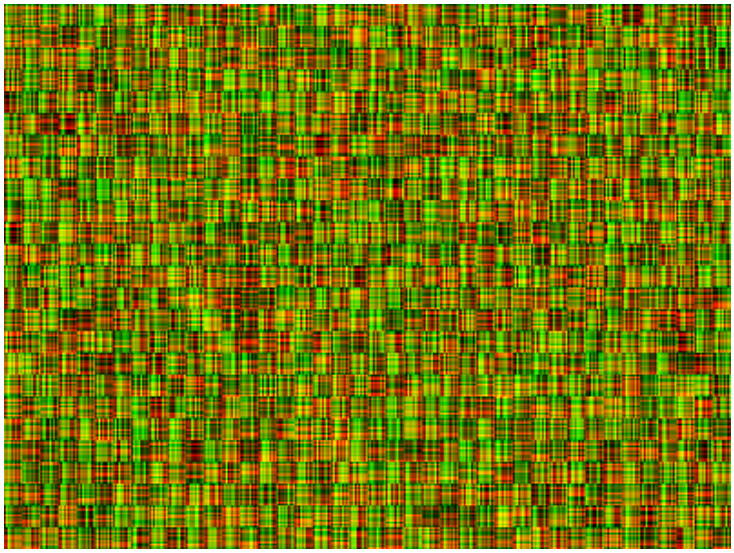


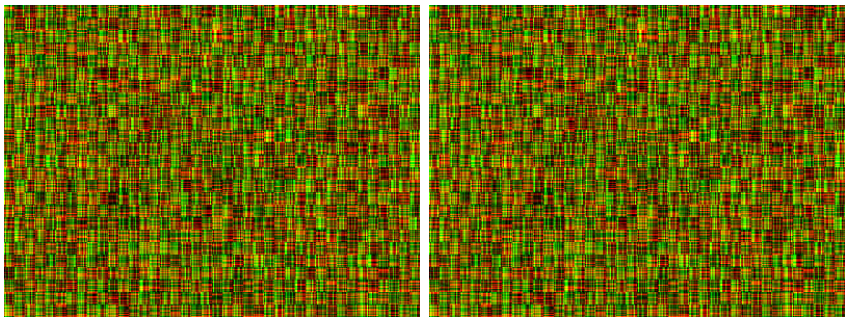
Figura 21: Imagen compuesta a partir de los bloques de los segundos 1000 eventos disponibles del proceso $t\bar{t}$ con las variables manualmente alteradas. Proporciona a la red el estilo.

oscura con más verosimilitud de la actualidad frente a la señal de fondo de la que, en principio, es inseparable. El empleo de redes neuronales sencillas (de tipo perceptrón multicapa) funciona muy bien en casos de clasificación sencillos y notablemente en casos moderadamente complicados. En este caso ha conseguido igualar, con implementaciones de 2 y 3 capas ocultas, los resultados de clasificación de eventos -de $t\bar{t}$ y del modelo de materia oscura en asociación con $t\bar{t}$ - en base al valor del momento ausente en el plano transverso $p_{T,Miss}$. Sin embargo, no parece que con el método considerado se puedan conseguir mejores resultados ni en sistemas con redes más profundas.

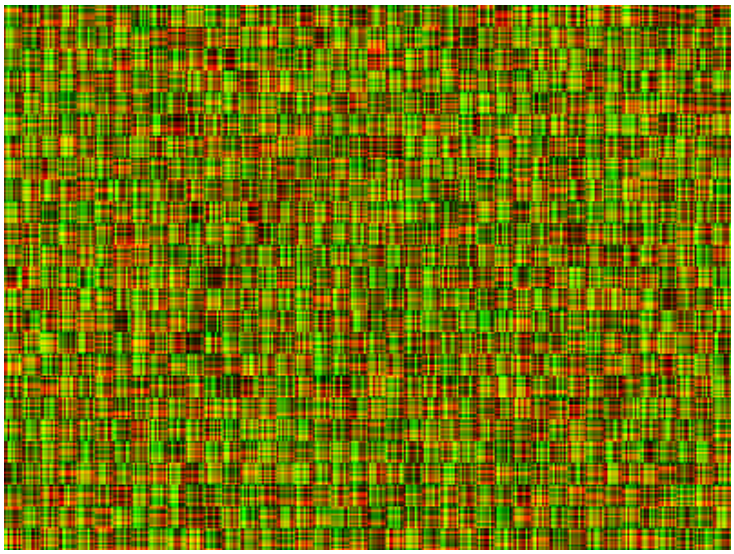
Sin embargo, quedan aún por poner a prueba otros tipos de arquitecturas de redes neuronales y también considerar otras variables.

En cuanto a la esperada mejora en la calidad de las simulaciones, el resultado no ha sido satisfactorio. El método utilizado no es viable. Ahora bien, en este proyecto solo se ha considerado la aplicación de una red convolucional tal como fue propuesta para abordar otro problema diferente, relativo a imágenes con contenidos de tipo completamente distinto al aquí considerado.

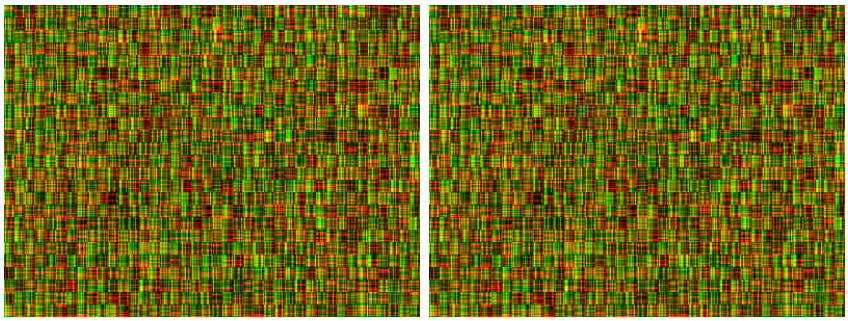
Se valora, por tanto, la posibilidad de continuar buscando la forma de extraer el estilo de CMS (así como para otros experimentos). Las líneas de



(a) Estilo 20-80 (b) Estilo 30-70



(c) Objetivo



(d) Estilo 40-60 (e) Estilo 50-50

Figura 22: En las subfiguras (a), (b), (d) y (e) se muestran las imágenes generadas a partir de la red para las diferentes importancias del estilo. En la subfigura (c) se muestra la imagen modificada manualmente, y que representa el objetivo que deben plasmar el resto.

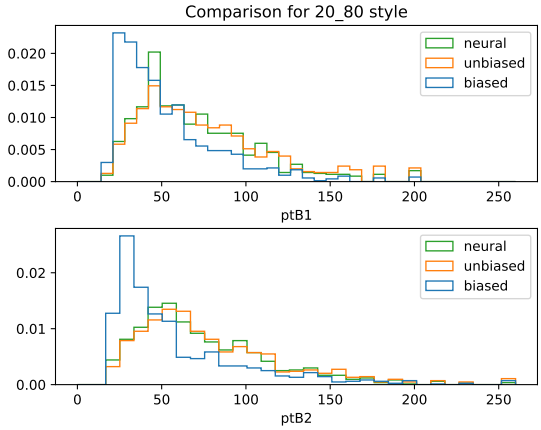


Figura 23: Histograma del momento transverso para el jet 1 en los 3 casos considerados: los datos sin sesgar (*unbiased*), los datos que simulan sesgo (*biased*) y los datos generados por la red neuronal (*generated*) para una relación de estilo-contenido de 20-80.

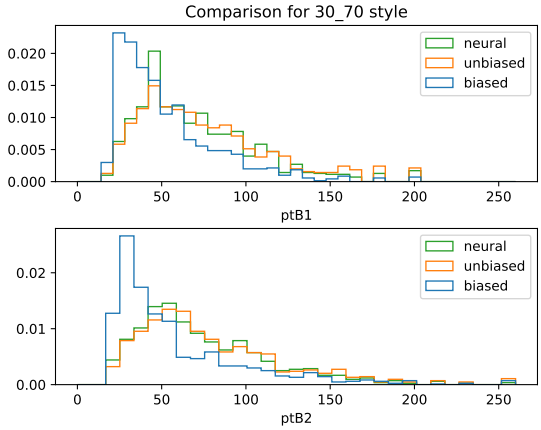


Figura 24: Histograma del momento transverso para el jet 1 en los 3 casos considerados: los datos sin sesgar (*unbiased*), los datos que simulan sesgo (*biased*) y los datos generados por la red neuronal (*generated*) para una relación de estilo-contenido de 30-70.

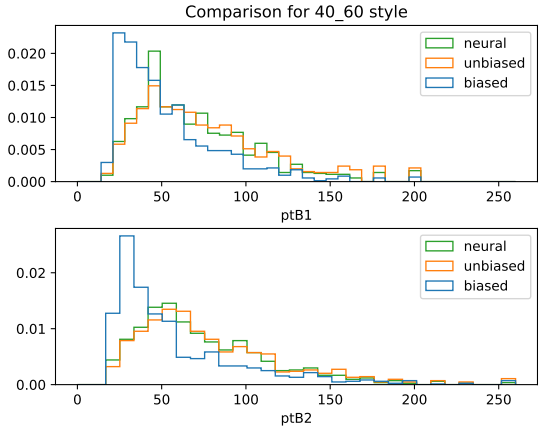


Figura 25: Histograma del momento transverso para el jet 1 en los 3 casos considerados: los datos sin sesgar (*unbiased*), los datos que simulan sesgo (*biased*) y los datos generados por la red neuronal (*generated*) para una relación de estilo-contenido de 40-60.

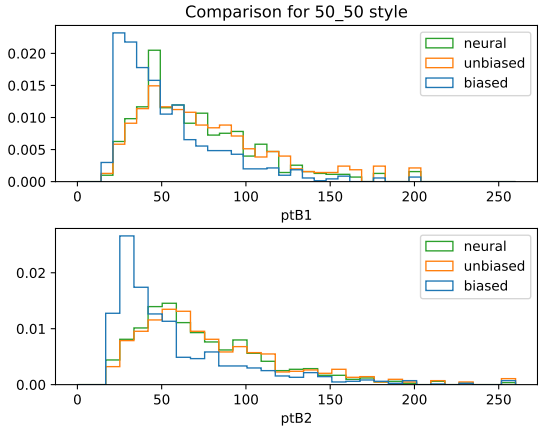


Figura 26: Histograma del momento transverso para el jet 1 en los 3 casos considerados: los datos sin sesgar (*unbiased*), los datos que simulan sesgo (*biased*) y los datos generados por la red neuronal (*generated*) para una relación de estilo-contenido de 50-50.

estudio que se abren a partir de ahora van desde la búsqueda de otros ordenamientos de la información en las imágenes u otro tipo de representación gráfica, a la puesta a punto de la red utilizada con un entrenamiento de sus niveles de más alta abstracción (las capas finales), a base de imágenes específicas del tipo estudiado. Todo esto puede conducir en el futuro a resultados más prometedores.

Esta idea, y otras que pudiera haber en el mismo sentido, pese a no haber hallado el resultado óptimo, todavía merecen el esfuerzo, porque el hallazgo de un método que funcionase a gran escala conllevaría una verdadera revolución del campo.

Referencias

[1] G. Aad and others. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1–29, September 2012.

[2] S. Chatrchyan and others. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Physics Letters B*, 716(1):30–61, September 2012.

[3] F. Zwicky. Republication of: The redshift of extragalactic nebulae. *General Relativity and Gravitation*, 41(1):207–224, January 2009.

[4] Vera C. Rubin and W. Kent Ford, Jr. Rotation of the Andromeda Nebula from a Spectroscopic Survey of Emission Regions. *The Astrophysical Journal*, 159:379, February 1970.

[5] Ms Roberts and Ah Rots. Comparison of Rotation Curves of Different Galaxy Types. *Astronomy & Astrophysics*, 26(3):483–485, 1973. WOS:A1973Q481600023.

[6] F. Zwicky. On the Masses of Nebulae and of Clusters of Nebulae. *The Astrophysical Journal*, 86:217, October 1937.

[7] F. Zwicky. Nebulae as Gravitational Lenses. *Physical Review*, 51(4):290–290, February 1937.

[8] Douglas Clowe, Anthony Gonzalez, and Maxim Markevitch. Weak-Lensing Mass Reconstruction of the Interacting Cluster 1e 0657–558: Direct Evidence for the Existence of Dark Matter. *The Astrophysical Journal*, 604(2):596, 2004.

[9] Douglas Clowe, Maruša Bradač, Anthony H. Gonzalez, Maxim Markevitch, Scott W. Randall, Christine Jones, and Dennis Zaritsky. A Direct Empirical Proof of the Existence of Dark Matter. *The Astrophysical Journal Letters*, 648(2):L109, 2006.

[10] Eugenio Del Nobile and Francesco Sannino. Effective Operators for Dark Matter Detection. *Centre for Cosmology and Particle Physics Phenomenology CP3-Origins and the Danish Institute for Advanced Study DIAS*, pages 140(26–35), November 2012.

[11] Lars Sonnenschein. Analytical solution of ttbar dilepton equations. *Physical Review D*, 78(7), October 2008. arXiv: hep-ph/0603011.

[12] Burton A. Betchart, Regina Demina, and Amnon Harel. Analytic solutions for neutrino momenta in decay of top quarks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 736:169–178, February 2014.

[13] Tom Michael Mitchell. *Machine learning*. McGraw-Hill series in computer science. McGraw-Hill, Singapore, international edition, 1997.

[14] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] K. A. Olive and Particle Data Particle Data Group. Review of Particle Physics. *Chinese Physics C*, 38(9):090001, 2014.

[17] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *arXiv:1508.06576 [cs, q-bio]*, 2015. arXiv: 1508.06576.

[18] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, September 2014. arXiv: 1409.1556.

[19] leonardblier. A brief report of the Heuritech Deep Learning Meetup #5, February 2016.

Anexo

Código de Python

my_keras.py

```
import time

import numpy as np

from matplotlib import pyplot as plt

from keras.utils import np_utils

import keras.callbacks as cb

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation

from keras.optimizers import RMSprop

from keras.datasets import mnist

import sys

import h5py

#A este programa se le llama desde terminal junto a los archivos con los datos de train
y test

#Crea, entrena y prueba la red sencilla


class LossHistory(cb.Callback):

    def on_train_begin(self, logs={}):

        self.losses = []

    def on_batch_end(self, batch, logs={}):

        batch_loss = logs.get('loss')

        self.losses.append(batch_loss)


if not len(sys.argv) == 5:

    sys.exit("Usage: modif_keras training1 training2 test1 test2")


training1 = sys.argv[1]

training2 = sys.argv[2]

test1 = sys.argv[3]

test2 = sys.argv[4]


data_training1 = np.genfromtxt(training1, delimiter=',')

data_training2 = np.genfromtxt(training2, delimiter=',')

data_test1 = np.genfromtxt(test1, delimiter=',')

data_test2 = np.genfromtxt(test2, delimiter=',')
```

```
data_training1_y = np.zeros((data_training1.shape[0], 1))

data_training2_y = np.ones((data_training2.shape[0], 1))

data_test1_y = np.zeros((data_test1.shape[0], 1))

data_test2_y = np.ones((data_test2.shape[0], 1))


training_x = np.concatenate((data_training1, data_training2), axis=0)

training_y = np_utils.to_categorical(np.concatenate((data_training1_y,
data_training2_y), axis=0), 2)

test_x = np.concatenate((data_test1, data_test2), axis=0)

test_y = np_utils.to_categorical(np.concatenate((data_test1_y, data_test2_y), axis=0),
2)


start_time = time.time()

print ('Compiling Model ... ')

model = Sequential()

model.add(Dense(4, input_dim=2))

model.add(Activation('relu'))

model.add(Dropout(0.4))


model.add(Dense(12))

model.add(Activation('relu'))

model.add(Dropout(0.4))


model.add(Dense(8))

model.add(Activation('relu'))

model.add(Dropout(0.4))


model.add(Dense(4))

model.add(Activation('relu'))

model.add(Dropout(0.4))


model.add(Dense(2))

model.add(Activation('sigmoid'))

rms = RMSprop()

model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

print ('Model compield in {0} seconds'.format(time.time() - start_time))


history = LossHistory()

print ('Training model...')
```

```
model.fit(training_x, training_y, nb_epoch=2, batch_size=1,
          callbacks=[history],
          validation_data=(test_x, test_y), verbose=2)

print ("Training duration : {0}".format(time.time() - start_time))
#score = model.evaluate(training_x, training_y, batch_size=1)
score = model.evaluate(test_x, test_y, batch_size=1)
print ("Network's test score [loss, accuracy]: {0}".format(score))

#uncomment lines below if want to save model weights
#model.save('my_net_numbers.h5')
#print "Weights saved in my_net_numbers.h5"
```

dumpTry.py

```
import ROOT as r
import numpy as np



#
#this function returns a np.array with
#(ptMET, DeltaPhi of leptons, pt of both leptons, Deltaphi of b-jets, pt of both b-jets)
#

def dumpTry(process):

    if process=="ttbar":
        f = r.TFile.Open("ttbar-Madgraph-MLM.root")
    elif process=="ttdm":
        f = r.TFile.Open("ttdm-pseudoscalar_100_1.root")
    else:
        print "wrong process specified"

    x=[]
    for event in f.t:



        phi1 = event.philep1
       phi2 = event.philep2

       DeltaPhiL = phi1-phi2
```

```
        ptL1 = event.ptlep1
        ptL2 = event.ptlep2

       phi1 = event.phib1
        phi2 = event.phib2

        DeltaPhiB = phi1-phi2

        ptB1 = event.ptb1
        ptB2 = event.ptb2

       x.append([event.ptMET, DeltaPhiL, ptL1, ptL2, DeltaPhiB, ptB1, ptB2 ])

    a=np.array(x)
    return a
```

keras_ttdm.py

```
import time
import numpy as np
import matplotlib
matplotlib.use("TkAgg")
from matplotlib import pyplot as plt
from keras.utils import np_utils
import keras.callbacks as cb
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from keras.datasets import mnist
import sys
import h5py
from dumpTry import dumpTry
from keras import backend as K
from sklearn.metrics import roc_curve
from sklearn.metrics import auc


#En este programa se crean dos redes similares y se entrenan para clasificar ttbar de
DM; y se evaluan.
```

```
data_training1 = dumpTry("ttbar")
data_training2 = dumpTry("ttdm")


data_test1 = data_training1[0:data_training1.shape[0]//6,0:]
data_test2 = data_training2[0:data_training2.shape[0]//6,0:]
data_training1 = data_training1[data_training1.shape[0]//6:,0:]
data_training2 = data_training2[data_training2.shape[0]//6:,0:]

data_training1_y = np.zeros((data_training1.shape[0], 1))
data_training2_y = np.ones((data_training2.shape[0], 1))
data_test1_y = np.zeros((data_test1.shape[0], 1))
data_test2_y = np.ones((data_test2.shape[0], 1))

training_x = np.concatenate((data_training1, data_training2), axis=0)
training_y = np.concatenate((data_training1_y, data_training2_y), axis=0)
test_x = np.concatenate((data_test1, data_test2), axis=0)
test_y = np.concatenate((data_test1_y, data_test2_y), axis=0)

print ('Compiling Model ... ')
model = Sequential()
model.add(Dense(7, input_dim=7, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model2 = Sequential()
model2.add(Dense(4, input_dim=7, activation='relu'))
model2.add(Dense(4, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


print ('Training model...')
model.fit(training_x, training_y, epochs=3, batch_size=32)
model2.fit(training_x, training_y, epochs=3, batch_size=32)

score = model.evaluate(test_x, test_y, batch_size=1)
```

```
score2 = model2.evaluate(test_x, test_y, batch_size=1)

print ("Network's test score [loss, accuracy]: {0}".format(score))
print ("Network's test score [loss, accuracy]: {0}".format(score2))

pred_y = model.predict_proba(test_x)[:, 0]
pred2_y = model2.predict_proba(test_x)[:, 0]

fpr_rf, tpr_rf, thresholds_rf = roc_curve(test_y, pred_y)
fpr2_rf, tpr2_rf, thresholds2_rf = roc_curve(test_y, pred2_y)

auc_rf = auc(fpr_rf, tpr_rf)
auc2_rf = auc(fpr2_rf, tpr2_rf)

plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='RF (area = {:.3f})'.format(auc_rf))
plt.plot(fpr2_rf, tpr2_rf, color='red', label='RF (area = {:.3f})'.format(auc2_rf))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

PrepareVariables.py

```
import matplotlib
matplotlib.use("TkAgg")
from matplotlib import pyplot as plt
import ROOT as r
import numpy as np
from ROOT import TLorentzVector

#construye las imagenes

def getVars(filename, factor):

    f = r.TFile.Open(filename)

    x = []
    for event in f.t:
```

```
 #The leptons
 mL1 = event.mlep1
 mL2 = event.mlep2
 ptL1 = event.ptlep1
 ptL2 = event.ptlep2
 phiL1 = event.philep1
phiL2 = event.philep2
 etaL1 = event.etalep1
 etaL2 = event.etalep2
 if abs(etaL1) > 2.4 or abs(etaL2) > 2.4:
     continue
 if ptL1 < 20 or ptL2 < 20:
     continue
 l1 = TLorentzVector()
 l1.SetPtEtaPhiM(ptL1, etaL1, phiL1, mL1)
 l2 = TLorentzVector()
 l2.SetPtEtaPhiM(ptL2, etaL2, phiL2, mL2)

 #The bs
 mB1 = event.mb1
 mB2 = event.mb2
 ptB1 = event.ptb1
 ptB2 = event.ptb2
 ptB1f = event.ptb1 * factor
 ptB2f = event.ptb2 * factor
 phiB1 = event.phib1
 phiB2 = event.phib2
 etaB1 = event.etab1
 etaB2 = event.etab2
 if abs(etaB1) > 3.0 or abs(etaB2) > 3.0:
     continue
 if ptB1 < 20 or ptB2 < 20:
     continue
 b1 = TLorentzVector()
 b1.SetPtEtaPhiM(ptB1, etaB1, phiB1, mB1)
 b2 = TLorentzVector()
 b2.SetPtEtaPhiM(ptB2, etaB2, phiB2, mB2)

 b1f = TLorentzVector()
```

```
        b1f.SetPtEtaPhiM(ptB1f, etaB1, phiB1, mB1)
        b2f = TLorentzVector()
        b2f.SetPtEtaPhiM(ptB2f, etaB2, phiB2, mB2)

        #The MET
        ptMET = event.ptMET
        phiMET = event.phiMET
        etaMET = event.etaMET
        mMET = event.mMET
        met = TLorentzVector()
        met.SetPtEtaPhiM(ptMET, etaMET, phiMET, mMET)
        metf = met + b1f + b2f - b1 - b2

        ptMETf = metf.Pt()
        phiMETf = metf.Phi()
        #Composite variables
        HT = ptB1f + ptB2f
        ST = ptB1f + ptB2f + ptL1 + ptL2
        mLL = (l1 + l2).M()
        mJJ = (b1f + b2f).M()
        ptLL = (l1 + l2).Pt()
        ptJJ = (b1f + b2f).Pt()
        mt_L1_MET = (l1 + metf).Mt()
        mt_L2_MET = (l2 + metf).Mt()



       x.append([ptL1, etaL1, phiL1, ptL2, etaL2, phiL2, ptB1f, etaB1, phiB1, ptB2f,
etaB2, phiB2, ptMETf, phiMETf, HT, ST, mLL, mt_L1_MET, mt_L2_MET, mJJ, ptLL, ptJJ])


    f.Close()
    a = np.array(x)
    amax = np.amax(a, axis=0)
    amin = np.amin(a, axis=0)
    return [a, amax, amin]



def foldVars(a, amax, amin):


    norma  = (a - amin)/(amax - amin)
```

```
    norma[:, 0] = np.exp(-22.0*norma[:,0])
    norma[:, 1] = norma[:,1]
    norma[:, 2] = norma[:,2]
    norma[:, 3] = np.exp(-22.0*norma[:,3])
    norma[:, 4] = norma[:,4]
    norma[:, 5] = norma[:,5]
    norma[:, 6] = np.exp(-11.0*norma[:,6])
    norma[:, 7] = norma[:,7]
    norma[:, 8] = norma[:,8]
    norma[:, 9] = np.exp(-11.0*norma[:,9])
    norma[:, 10] = norma[:,10]
    norma[:, 11] = norma[:,11]
    norma[:, 12] = np.exp(-5*norma[:,12])
    norma[:, 13] = norma[:,13]
    norma[:, 14] = np.exp(-5*norma[:,14])
    norma[:, 15] = np.exp(-5*norma[:,15])
    norma[:, 16] = np.exp(-5*norma[:,16])
    norma[:, 17] = np.exp(-12.0*norma[:,17])
    norma[:, 18] = np.exp(-8*norma[:,18])
    norma[:, 19] = np.exp(-10.0*norma[:,19])
    norma[:, 20] = np.exp(-8.0*norma[:,20])
    norma[:, 21] = np.exp(-8.0*norma[:,21])

    norma = norma * 256.0
    return norma

def unfoldVars(a, amax, amin):

    norma = a/256.0

    norma[:, 0] = -np.log(norma[:,0])/22.0
    norma[:, 1] = norma[:,1]
    norma[:, 2] = norma[:,2]
    norma[:, 3] = -np.log(norma[:,3])/22.0
    norma[:, 4] = norma[:,4]
    norma[:, 5] = norma[:,5]
    norma[:, 6] = -np.log(norma[:,6])/11.0
    norma[:, 7] = norma[:,7]
    norma[:, 8] = norma[:,8]
```

```
    norma[:, 9] = -np.log(norma[:,9])/11.0
    norma[:, 10] = norma[:,10]
    norma[:, 11] = norma[:,11]
    norma[:, 12] = -np.log(norma[:,12])/5.0
    norma[:, 13] = norma[:,13]
    norma[:, 14] = -np.log(norma[:,14])/5.0
    norma[:, 15] = -np.log(norma[:,15])/5.0
    norma[:, 16] = -np.log(norma[:,16])/5.0
    norma[:, 17] = -np.log(norma[:,17])/12.0
    norma[:, 18] = -np.log(norma[:,18])/8.0
    norma[:, 19] = -np.log(norma[:,19])/10.0
    norma[:, 20] = -np.log(norma[:,20])/8.0
    norma[:, 21] = -np.log(norma[:,21])/8.0

    norma = (amax - amin) * norma + amin

    return norma



def pickBlockNumber(x, n):

    nrows = x.shape[0]
    nsubpictures = 25 * 40
    if (n + 1) * nsubpictures > nrows:
        return []
    else:
        return x[n * nsubpictures:(n+1) * nsubpictures, :]


def makeMatrixFromBlock(X):

    vsize = 12
    hsize = 10
    vdim = 25
    hdim = 40
    x = np.zeros((12 * vdim, 10 * hdim, 3), dtype=np.uint8)

    for vi in range(0, vdim):
```

```
        for hi in range(0, hdim):
            for vj in range(0, vsize):
                for hj in range(0, hsize):
                    x[vi * vsize + vj, hi * hsize + hj, 0] = np.rint(X[vi * vsize + hi,
vj])
                    x[vi * vsize + vj, hi * hsize + hj, 1] = np.rint(X[vi * vsize + hi,
hj + vsize])

    return x

[a, amax, amin] = getVars("ttbar-Madgraph-MLM.root", 1.0)
[af, amaxf, aminf] = getVars("ttbar-Madgraph-MLM.root", 0.5)

X = foldVars(a, amax, amin)
Xf = foldVars(af, amax, amin)

X0 = pickBlockNumber(X, 1)
X0f = pickBlockNumber(Xf, 1)

img0 = makeMatrixFromBlock(X0)
img0f = makeMatrixFromBlock(X0f)

print X0

plt.imsave('image1.jpg', img0, format='jpg')
plt.imsave('imagef1.jpg', img0f, format='jpg')
```