

Benemerita Universidad Autonoma de Agascalientes
Centro de Ciencias Básicas
Departamento de Ciencias de la computación

Directores

Eunice Esther Ponce de León Sentí
Mauricio Eduardo Martín Álvarez Tostado

Comite Tutorial

Francisco Javier Alvarez
Alejandro Padilla Díaz
Julio Cesar Ponce Gallegos

**Diseño e implementación de un método de
aprendizaje profundo para la búsqueda de
regiones conservadas en proteínas.**

Raúl Sánchez Vázquez

Junio 2021



BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE
AGUASCALIENTES

CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Tesina

“Diseño e implementación de un método de aprendizaje profundo para la búsqueda de regiones conservadas en proteínas”

Presenta

“Raúl Sánchez Vázquez”

Para Obtener el Grado de Ingeniería en Computación Inteligente

Director:

Dra. Eunice Esther Ponce de León Sentí

C.Dr. Mauricio Eduardo Martín Álvarez Tostado

Comité Tutorial *:

Dr. Francisco Javier Álvarez Rodríguez

Dr. Alejandro Padilla Díaz

Dr. Julio Cesar Ponce Gallegos

(*) El comité tutorial del alumno está constituido por los tres profesores de la materia de Seminario de Investigación II que han acompañado al alumno en las revisiones semanales y presentaciones parciales. Si el director de tesis considera que otro profesor ha participado activamente para considerarlo en el comité tutorial, se puede añadir previa argumentación por correo electrónico enviado a los tres profesores designados para la materia.

Aguascalientes, Ags., 7 de Junio de 2021

A quien corresponda:

Por medio de la presente me permito informar que el alumno **Raúl Sánchez Vázquez** de la carrera Ingeniería en Computación Inteligente con ID: **148837**, ha terminado satisfactoriamente su tesina titulada: **“Diseño e implementación de un método de aprendizaje profundo para la búsqueda de regiones conservadas en proteínas”** correspondiente a la materia de Seminario de Investigación II.

Para los fines que al interesado convengan.

ATENTAMENTE



Dra. Eunice Esther Ponce de León Sentí
Directora

Aguascalientes, Ags., 7 de Junio de 2021

A quien corresponda:

Por medio de la presente me permito informar que el alumno **Raúl Sánchez Vázquez** de la carrera Ingeniería en Computación Inteligente con **ID: 148837**, ha terminado satisfactoriamente su tesina titulada: **“Diseño e implementación de un método de aprendizaje profundo para la búsqueda de regiones conservadas en proteínas”** correspondiente a la materia de Seminario de Investigación II.

Para los fines que al interesado convengan.

ATENTAMENTE



C.Dr. Mauricio Eduardo Matín Alvares Tostado
Director


Aguascalientes, Ags., 14 de junio de 2021

A quien corresponda:

Por medio de la presente me permito informar que el alumno(a) **Raúl Sánchez Vázquez** de la carrera de Ingeniería en Computación Inteligente con **ID: 148837**, ha terminado satisfactoriamente su tesina titulada: **“Diseño e implementación de un método de aprendizaje profundo para la búsqueda de regiones conservadas en proteínas”**, correspondiente a la materia de Seminario de Investigación II.

Para los fines que al interesado convengan.

ATENTAMENTE



Dr. Francisco Javier Álvarez Rodríguez
Miembro del Comité Tutorial.

Aguascalientes, Ags., 16 de Junio de 2021

A quien corresponda:

Por medio de la presente me permito informar que el alumno **Raúl Sánchez Vázquez** de la carrera Ingeniería en Computación Inteligente con **ID: 148837**, ha terminado satisfactoriamente su tesina titulada: **“Diseño e implementación de un método de aprendizaje profundo para la búsqueda de regiones conservadas en proteínas”** correspondiente a la materia de Seminario de Investigación II.

Para los fines que al interesado convengan.

ATENTAMENTE



Dr. Alejandro Padilla Diaz
Miembro del comité tutorial

Aguascalientes, Ags., 15 de Junio de 2021

A quien corresponda:

Por medio de la presente me permito informar que el alumno **Raúl Sánchez Vázquez** de la carrera Ingeniería en Computación Inteligente con **ID: 148837**, ha terminado satisfactoriamente su tesina titulada: **“Diseño e implementación de un método de aprendizaje profundo para la búsqueda de regiones conservadas en proteínas”** correspondiente a la materia de Seminario de Investigación II.

Para los fines que al interesado convengan.

ATENTAMENTE



Dr. Julio Cesar Ponce Gallegos
Miembro del comité tutorial

Resumen

Los motivos conservados en proteínas son patrones que suelen ser asociados con funciones de una proteína lo cual dilucida la potencialidad de una proteína o inclusive ayuda a definir al conjunto de proteínas conocidas como familia.

Sin embargo, la búsqueda de motivos conservados es un problema que si bien puede ser abordado desde diferentes ángulos, es en su forma más primitiva, el análisis secuencial, donde se encuentra el inconveniente de la computabilidad ya que es poco factible en término de recursos.

En el presente trabajo se plantea un enfoque bajo algoritmos de aprendizaje profundo, en concreto el uso de redes neuronales convolucionales. Tal enfoque es posible gracias a la reinterpretación de la forma en que puede visualizarse una proteína a partir de su naturaleza como una secuencia de aminoácidos.

Dicha representación aunada a una base donde se cuenta con un conjunto de datos agrupados en familias permite estudiar la correlación estadística que existe entre los pares de aminoácidos de una proteína con lo cual es posible generar una abstracción donde la correlación es relativa a la distancia física entre aminoácidos en una disposición tridimensional, obteniendo en consecuencia un acercamiento a un modelo bidimensional simplificado de una proteína, que a su vez simplifica las propiedades percibidas en un modelo tridimensional.

Con tal información se genera un clasificador basado en redes neuronales convolucionales, el cual recibiendo como entrada los modelos bidimensionales de proteínas permite la clasificación de proteínas de acuerdo a la familia a la que pertenece.

La construcción de dicho modelo es crucial ya que esto permite el estudio del comportamiento interno de la red neuronal convolucional, de donde se extraen patrones conocidos como mapas de activación de características que su vez se pueden definir como motivos conservados.

Tal desarrollo se ve posibilitado gracias a la obtención de tres resultados generales, iniciando por la generación de imágenes de correlación de proteínas, posteriormente la salida de un clasificador de aprendizaje profundo y por ultimo la visualización de mapas de características que resaltan los patrones de cada familia.

Índice general

1. Introducción	11
2. Planteamiento del problema	12
3. Justificación	13
4. Objetivos	14
4.1. Objetivo general	14
4.2. Objetivos particulares	14
5. Preguntas de investigación	15
5.1. General	15
5.2. Específica	15
6. Marco Teórico	16
6.1. Proteínas	16
6.1.1. Aminoácidos	16
6.1.2. Plegamiento de las proteínas	17
6.2. Virus	18
6.2.1. Origen de los virus	18
6.2.2. Microbiología	18
6.2.3. Clasificación	19
6.3. Machine Learning	19
6.3.1. Componentes	20
6.3.2. Aprendizaje supervisado	21
6.3.3. Aprendizaje no supervisado	25

6.3.4.	Aprendizaje reforzado	29
6.3.5.	Aprendizaje profundo	30
6.4.	Trabajos Relacionados	36
6.4.1.	Predicción mejorada de la estructura de proteínas usando potenciales del aprendizaje profundo	36
6.4.2.	Alta precisión en la predicción del contacto de proteínas utilizando redes neuronales totalmente convolucionales y características de secuencia mínima	37
7.	Metodología	38
7.1.	Investigación	39
7.2.	Diseño e implementación	39
7.2.1.	Selección de la información	39
7.2.2.	Procesamiento de los datos	40
7.2.3.	Aprendizaje profundo	40
7.2.4.	Búsqueda de patrones	40
7.3.	Análisis de resultados	41
8.	Desarrollo	42
8.1.	Alineamiento múltiple de secuencias	42
8.1.1.	Puntuación	42
8.1.2.	Tipos de alineamiento múltiple de secuencias	42
8.1.3.	Uso de Clustal Omega	43
8.2.	Estudio de correlación	43
8.2.1.	Covarianza generalizada	44
8.2.2.	Métodos	45
8.3.	Generación de imagen	46
8.3.1.	Modelos de color	46
8.3.2.	Índices de correlación a imágenes RGB	47
8.4.	Red neuronal convolucional	49
8.4.1.	Tuberías de datos	49
8.4.2.	Librerías	50
8.4.3.	Exploración del dataset	50

8.4.4. Conjunto de datos de entrenamiento y validación	50
8.4.5. Segmentación de los datos en lotes	51
8.4.6. Modelo base para la clasificación de imágenes	52
8.4.7. Modelo de clasificación	52
8.4.8. Configuración del dispositivo de procesamiento	52
8.4.9. Ajuste del modelo	53
8.4.10. Evaluación mediante datos de prueba	53
8.5. Visualización de características	53
9. Experimentación y pruebas	55
9.1. Secuencias de aminoácidos	55
9.1.1. Matriz de índices de correlación	56
9.2. Imágenes	56
9.3. Mapas de activación de clase	59
9.3.1. Red neuronal convolucional	59
9.3.2. Visualización de características	59
10. Análisis e interpretación de resultados	61
10.1. Imágenes de correlación	61
10.2. Mapas de activación de clases	64
11. Conclusiones	67
11.1. Trabajos futuros	69
I Anexos	76
12. Lista de organismos estudiados	77
13. Código parcial del método propuesto	79
13.1. Bloques ipython	79
13.1.1. Inferencia de sitios acoplados usando covarianza para la generación de imágenes de correlación	79
13.1.2. Red Neuronal Convolucional	84

13.1.3. Visualización de características	95
--	----

Índice de cuadros

12.1. Conjunto de organismo en clique de tamaño 64	78
--	----

Índice de figuras

6.1. Problemas de clasificación	22
6.2. Cómo funciona el descenso del gradiente para un parámetro w	33
7.1. Descripción gráfica de la metodología para la búsqueda de patrones en secuencias de proteínas.	38
9.1. Representación gráfica de los índices de correlación: img064-0-3	57
9.2. Diferencia de imágenes de índices de correlación	58
10.1. Fragmento de matriz de índices de correlación: corr064-17-0	62
10.2. Imagen de correlación generada con el intervalo $[-1,1]$ (img064-0-0_11)	63
10.3. Distribución de los datos de la proteína 064-0-0	63
10.4. Imagen de correlación generada con el intervalo $[0, 1 * 10^{-24}]$ (img064-0-0)	64
10.5. Performance del entrenamiento de la CNN	65
10.6. Descubrimiento de patrón mediante mapa de activación de clases	66

Código fuente

13.1. Librerías	79
13.2. Lista de proteínas del clique	79
13.3. Descarga de secuencias de aminoácidos por proteína	80
13.4. MSA mediante Clustal Omega	80
13.5. Lista de aminoácidos	81
13.6. Estructura de directorios	81
13.7. Transformación de secuencias alineadas a representación gráfica	81
13.8. Tubería de datos	84
13.9. Tubería de datos	85
13.10 Exploración del dataset	85
13.11 Conjuntos de datos de entrenamiento y validación	87
13.12 Segmentación de los datos en lotes	87
13.13 Modelo base para la clasificación de imágenes	87
13.14 Modelo de clasificación	88
13.15 Configuración del dispositivo de procesamiento	90
13.16 Ajuste del modelo y aprendizaje	91
13.17 Gráficos de precisión y pérdidas del modelo	93
13.18 Evaluación mediante datos de prueba	93
13.19 Evaluación sobre imágenes	94
13.20 Filtros convolucionales	95
13.21 Estructura de directorios	97
13.22 Librerías	97
13.23 Búsqueda de mapas de activación de clases	98
13.24 Visualización de mapas de activación de clases	100

13.25CAM Bloque 1 101

13.26CAM Bloque 2 102

13.27CAM Bloque 3 103

Capítulo 1

Introducción

La interacción de las ciencias de la computación y las ciencias biológicas han demostrado ser una de las mancuernas más inesperadas y que mayor sinergia han logrado demostrar.

No ha sido fácil, sin embargo, la evolución de las disciplinas computacionales ha expandido los métodos de abstracción con lo cual se sustenta la representación del mundo real, en particular, la biología, la cual en sus niveles mas bioquímicos ha logrado desarrollar un lenguaje capaz de facilitar la comunicación con dispositivos de computo.

Es por ello, que la solución de problemas referentes a organismos vivos se ha vuelto un tema de unos y ceros, pues gracias al poder de procesamiento actual la resolución de problemas ha visto reducida la complejidad e inclusive ha aumentado la factibilidad y eficiencias en la modelación de organismos.

Dicho esto se presenta el estudio de proteínas considerando su lenguaje los aminoácidos echando mano de técnicas de inteligencia artificial tal que la búsqueda de motivos conservados sea llevada a cabo reduciendo los tiempos e inclusive planteando la posibilidad de nuevas representaciones que favorezcan el estudio de los procesos bioquímicos de los organismos vivos.

Capítulo 2

Planteamiento del problema

Analizar secuencias de proteínas es una tarea que de forma tradicional requiere una serie de técnicas rigurosas y equipo de alto costo, esto con el fin de comprender mejor el comportamiento que da sentido a las funciones que desempeñan cada tipo de proteína [1].

Cuando hablamos de proteínas se está teniendo en cuenta que éstas se conforman de secuencias de aminoácidos las cuales gracias a sus propiedades químicas y biológicas aportan una función particular, pero previo a ello, dichas propiedades determinan el comportamiento molecular que éstas poseen [1][2].

El comportamiento molecular está dado en mayor medida por la disposición de los aminoácidos en las secuencias, ya que esto favorece asociaciones y a su vez enlaces que permiten la configuración estructural de una proteína, lo cual influye en las funciones celulares de los organismos, tal es el caso de los virus [3].

Teniendo en cuenta que las secuencias de aminoácidos se pueden representar mediante abstracciones computacionales se plantea el problema de buscar los patrones o constantes que prevalecen en un conjunto de proteínas.

Capítulo 3

Justificación

Al día de hoy el estudio de los fenómenos biológicos resulta relevante, más si al campo de validar la eficacia de los modelos de virus nos referimos, pues previo a cualquier solución tal como medicamentos o vacunas es necesario conocer el comportamiento de los entes biológicos [4].

Para ello resulta importante entender las estructuras moleculares, donde las proteínas resultan un componente vital ya que sus variantes proveen de un componente vital para el funcionamiento de los organismos sean benignos o no [5].

Es importante mencionar el hecho de que las proteínas son un elemento de la naturaleza cuya abstracción es altamente compatible con modelos matemáticos y estadísticos, y por ende computacionales, ya que, su constitución a base de secuencias se puede conceptualizar como cadenas finitas de caracteres. Una vez que han sido abstraídas el compendio de cadenas puede ser procesado de múltiples maneras en pro de su análisis [5].

Capítulo 4

Objetivos

4.1. Objetivo general

Diseñar e implementar un método que utilice el aprendizaje profundo mediante redes neuronales convolucionales de tal forma que sea posible encontrar los patrones que definen a las regiones conservadas en las secuencias de aminoácidos de una familia de proteínas.

4.2. Objetivos particulares

1. Delimitar un conjunto de datos proteicos relacionados entre si y posteriormente aplicar el alineamiento múltiple de secuencias mediante el algoritmo *clustal*.
2. Conocer y transformar la correlación estadística entre los aminoácidos de una familia de proteínas en un *input* apto para el paradigma convolucional de aprendizaje profundo.
3. Diseñar e implementar una red neuronal convolucional capaz de clasificar a una proteína dado un conjunto de familias.
4. Descubrir patrones en familias de proteínas a partir del output de una red neuronal convolucional.

Capítulo 5

Preguntas de investigación

5.1. General

¿Es posible abstraer secuencias de aminoácidos mediante el paradigma de convolución de tal forma que se descubran los patrones que definen a las familias de proteínas?

5.2. Específica

- ¿Cómo funciona el alineamiento múltiple de secuencias?
- ¿De qué forma se puede calcular la correlación estadística entre los aminoácidos de una familia de proteínas?
- ¿Qué se requiere para que una matriz de correlación se adapte al paradigma convolucional de aprendizaje profundo?
- ¿Cuáles son los parámetros y funciones necesarias para implementar una red neuronal convolucional?
- ¿De qué forma se pueden extraer patrones de un conjunto de imágenes de correlación proteicas?

Capítulo 6

Marco Teórico

El presente trabajo requiere intersectar el conocimiento de dos campos la biología y las ciencias computacionales, esto ya que el problema corresponde al primero y se busca una solución computacional. Por tanto se realizará un recorrido conceptual iniciando por la idea de la proteína así como su relación con los virus. Posteriormente se hará un recorrido por el aprendizaje de máquina o machine learning considerando sus divisiones así como los principales algoritmos. Finalmente se verá la relación que existe entre los conceptos presentados mediante el estudio de trabajos relacionados.

6.1. Proteínas

Las proteínas son una clase importante de moléculas que se encuentran en todas las células vivas. Una proteína se compone de una o más cadenas largas de aminoácidos, cuya secuencia corresponde a la secuencia de ADN del gen que la codifica. Las proteínas desempeñan gran variedad de funciones en la célula, incluidas estructurales (citoesqueleto), mecánicas (músculo), bioquímicas (enzimas), y de señalización celular (hormonas). Las proteínas son también parte esencial de la dieta [2].

6.1.1. Aminoácidos

Los aminoácidos son unas moléculas que se unen en cadena para formar proteínas. De hecho, hay 20 tipos distintos de aminoácidos. Es como si tuviéramos un collar de perlas de distintos colores, donde cada perla es un aminoácido. Estas cadenas son lo que llamamos polipéptidos, un conjunto de aminoácidos en un orden o secuencia determinada. Pero, lo realmente interesante de

los aminoácidos que conforman esta secuencia es que, cuando se unen entre ellos y forman cadenas, éstas se repliegan y dan una forma concreta a la proteína resultante. Dependiendo de la forma, la proteína podrá desempeñar una función u otra dentro de la célula [6].

6.1.2. Plegamiento de las proteínas

Una proteína comienza en la célula como una cadena larga de, en promedio, 300 bloques de construcción llamados aminoácidos. Hay 22 tipos diferentes de aminoácidos y su orden determina cómo se plegará la cadena de proteínas sobre sí misma. Al plegar, generalmente se forman primero dos tipos de estructuras. Algunas regiones de la cadena de proteínas se enrollan en formaciones parecidas a escurridizas llamadas "hélices alfa", mientras que otras regiones se pliegan en patrones en zigzag llamados "hojas beta", que se asemejan a los pliegues de un abanico de papel. Estas dos estructuras pueden interactuar para formar estructuras más complejas. Por ejemplo, en una estructura de proteína, varias hojas beta se envuelven alrededor de sí mismas para formar un tubo hueco con algunas hélices alfa que sobresalen de un extremo. El tubo es corto y rechoncho, de modo que la estructura general se asemeja a serpientes (hélices alfa) que emergen de una lata (tubo de lámina beta) [7].

Estas estructuras complejas permiten que las proteínas realicen sus diversas funciones en la célula. La proteína "serpientes en una lata", cuando está incrustada en una membrana celular, crea un túnel que permite el tráfico hacia adentro y hacia afuera de las células. Otras proteínas forman estructuras con cavidades llamadas "sitios activos" que tienen la forma perfecta para unirse a una molécula en particular, como un candado y una llave. Al plegarse en formas distintas, las proteínas pueden desempeñar funciones muy diferentes a pesar de estar compuestas por los mismos bloques de construcción básicos. Para hacer una analogía, todos los vehículos están hechos de acero, pero la forma elegante de un auto de carreras gana carreras, mientras que un autobús, camión volquete, grúa o zamboni tienen cada uno una forma para realizar sus propias tareas únicas [7].

6.2. Virus

6.2.1. Origen de los virus

Al estudiar el origen de los virus, hay que considerar previamente que los virus son agentes infecciosos acelulares que infectan células y producen viriones para difundir genes entre sus huéspedes; por lo que en su origen se debe considerar la interacción entre el virus y su huésped. Igualmente destaca que la mayoría de las proteínas virales no tienen homólogos en las células modernas, en contradicción con la visión tradicional de los virus como los «ladrones de genes celulares». Esto sugiere que los genes virales básicamente tienen su origen durante la multiplicación de los genomas virales, o provendrían de linajes celulares ahora extintos (ya que algunas proteínas virales específicas están presentes en virus que infectan a los miembros de los tres dominios de la vida, lo que sugiere que los virus son en realidad muy antiguos en la historia de la vida). En particular, los análisis estructurales de proteínas de la cápside han revelado que al menos dos tipos de viriones se habrían originado de manera independiente antes que LUCA (el último antepasado común universal de la vida celular). Aunque recientemente se han propuesto varias hipótesis para explicar el origen de los virus, sigue sin explicarse completamente la aparición de viriones como mecanismo específico para la difusión de genes [8].

6.2.2. Microbiología

Ácido Nucleico

El término ácido nucleico es utilizado para describir unas moléculas específicas y grandes en la célula. En realidad están hechas de cadenas de unidades de polímeros que se repiten; los dos ácidos nucleicos más relevantes son el ADN y el ARN. Los ácidos nucleicos trabajan en la célula almacenando información. La célula codifica información, como cuando se graba en una cinta, en los ácidos nucleicos. El nombre de ácido nucleico proviene del hecho de cómo fueron descritos por primera vez, ya que en realidad tienen propiedades ácidas. Y el término nucleico viene del hecho de dónde se aislaron por primera vez, ya que se encontraron en el núcleo[9].

Estructura

Una partícula vírica completa, conocida como virión, consiste en un ácido nucleico rodeado por una capa de protección proteica llamada cápside. Las cápsides están compuestas de subunidades proteicas idénticas llamadas capsómeros [10]. Algunos virus tienen un «envoltorio lipídico» derivado de la membrana celular del huésped (virus con envoltorio), mientras que otros carecen de ella (virus desnudos). La cápside está formada por proteínas codificadas por el genoma vírico, y su forma es la base de la distinción morfológica [11]. Las subunidades proteicas codificadas por los virus se autoensamblan para formar una cápside, generalmente necesitando la presencia del genoma viral. Sin embargo, los virus complejos codifican proteínas que contribuyen a la construcción de su cápside. Las proteínas asociadas con los ácidos nucleicos son conocidas como nucleoproteínas, y la asociación de proteínas de la cápside vírica con ácidos nucleicos víricos recibe el nombre de nucleocápside.

6.2.3. Clasificación

Clasificación Baltimore

En este sistema de clasificación los virus están agrupados en grupos dependiendo de su tipo de genoma (ADN, ARN, monocatenario o bicatenario etc.) y en su método de replicación. Clasificar los virus según su genoma implica que los que quedan encuadrados en la misma categoría se comportarán básicamente de la misma manera, lo cual facilita las investigaciones [12].

6.3. Machine Learning

Un algoritmo de aprendizaje automático es un algoritmo que puede aprender de los datos, tal que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas y mide el desempeño D , si su desempeño en tareas en T , medido por D , mejora con la experiencia E , por ende es necesario desglosar los parámetros E , D y T [13].

6.3.1. Componentes

Tareas (T)

El aprendizaje automático nos permite abordar tareas que son demasiado difíciles de resolver con programas fijos, escritos y diseñados por seres humanos. Desde un punto de vista científico y filosófico, el aprendizaje automático es interesante puesto que permite desarrollar nuestra comprensión del mismo favoreciendo la comprensión de los principios que subyacen a la inteligencia. Donde la palabra tarea, puede entenderse como el proceso de aprendizaje en sí mismo no es la tarea. El aprendizaje es nuestro medio para lograr la capacidad de realizar la tarea [13].

Las tareas de aprendizaje automático suelen ser descritas en términos de cómo el sistema de aprendizaje automático debe procesar un ejemplo tal que este es una colección de características que se han medido cuantitativamente a partir de algún objeto o evento que queremos que procese el sistema de aprendizaje automático. Normalmente es representado como un vector x donde cada entrada x del vector es otra característica [13].

Desempeño (D)

Para evaluar las capacidades de un algoritmo de aprendizaje automático, se debe diseñar una medida cuantitativa de su rendimiento. Por lo general, esta medida de desempeño es específica de la tarea T que está llevando a cabo el sistema. Para tareas como clasificación, clasificación con entradas faltantes y transcripción, a menudo se utiliza la precisión del modelo [13].

La precisión no es más que la proporción de ejemplos para los que el modelo produce el resultado correcto. También es posible obtener información equivalente midiendo la tasa de error o dicho de otra forma la proporción de ejemplos para los que el modelo produce una salida incorrecta.

A menudo la tasa de error se considera como la pérdida esperada 0-1. La pérdida 0-1 en un ejemplo particular donde es 0 si se clasifica correctamente y 1 si no lo está.

Para tareas como la estimación de densidad, no tiene sentido medir la precisión, la tasa de error o cualquier otro tipo de pérdida 0-1. En su lugar, se debe utilizar una métrica de desempeño diferente que dé al modelo una puntuación de valor continuo para cada ejemplo. El enfoque más común es informar la probabilidad logarítmica promedio que el modelo asigna a algunos ejemplos.

Experiencia (E)

Los algoritmos de aprendizaje automático se pueden clasificar en términos generales como no supervisados o supervisados por el tipo de experiencia que se les permite tener durante el proceso de aprendizaje [13].

Se puede entender que la mayoría de los algoritmos de aprendizaje permiten experimentar un conjunto de datos completo. Un conjunto de datos es una colección de muchos ejemplos. A veces llamados puntos de datos de ejemplos.

Uno de los conjuntos de datos más antiguos estudiados por estadísticos e investigadores de aprendizaje automático es el conjunto de datos Iris. Es una colección de medidas de diferentes partes de 150 plantas de iris. Cada planta individual corresponde a un ejemplo. Las características dentro de cada ejemplo son las medidas de cada parte de la planta: la longitud del sépalo, el ancho del sépalo, la longitud del pétalo y el ancho del pétalo. El conjunto de datos también registra a qué especie pertenecía cada planta. Tres especies diferentes están representadas en el conjunto de datos.

6.3.2. Aprendizaje supervisado

Los algoritmos de aprendizaje automático supervisados están diseñados para aprender con el ejemplo. El nombre de aprendizaje “supervisado” se origina en la idea de que entrenar este tipo de algoritmos es como tener un profesor supervisando todo el proceso [14].

Al entrenar un algoritmo de aprendizaje supervisado, los datos de entrenamiento consistirán en entradas emparejadas con las salidas correctas. Durante el entrenamiento, el algoritmo buscará patrones en los datos que se correlacionen con los resultados deseados. Después del entrenamiento, un algoritmo de aprendizaje supervisado tomará nuevas entradas invisibles y determinará en qué etiqueta se clasificarán las nuevas entradas según los datos de entrenamiento anteriores. El objetivo de un modelo de aprendizaje supervisado es predecir la etiqueta correcta para los datos de entrada recién presentados. En su forma más básica, un algoritmo de aprendizaje supervisado se puede escribir simplemente como:

$$Y = f(x) \tag{6.1}$$

Donde Y es la salida prevista que está determinada por una función de mapeo que asigna una clase a un valor de entrada x . El modelo de aprendizaje automático crea la función utilizada para conectar las características de entrada a una salida prevista durante el entrenamiento [14].

Clasificación

La Clasificación es una subcategoría del aprendizaje supervisado donde el objetivo es predecir las etiquetas de clase categóricas (valores discretos, no ordenados, pertenencia a grupos) de nuevas instancias basadas en observaciones pasadas.

Hay dos tipos principales de problemas de clasificación:

- Clasificación binaria: el ejemplo típico es la detección de correo no deseado, en el que cada correo electrónico es correo no deseado $\rightarrow 1$ correo no deseado; o no es $\rightarrow 0$.
- Clasificación de clases múltiples: como el reconocimiento de caracteres escritos a mano (donde las clases van de 0 a 9).

Problema de las clases

Es importante señalar que no todos los modelos de clasificación serán útiles para separar correctamente clases diferentes de un conjunto de datos. Algunos algoritmos, como el perceptrón (que se basa en redes neuronales artificiales básicas), si las clases no pueden separarse mediante un límite de decisión lineal, no convergerán al aprender los pesos del modelo [14].

Algunos de los casos más típicos se representan en la siguiente imagen:

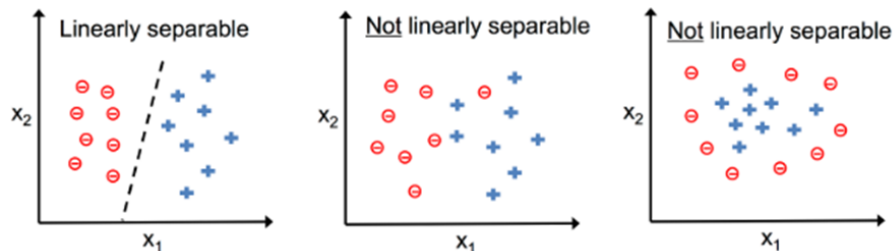


Figura 6.1: Problemas de clasificación

Algoritmos de clasificación

Regresión logística

La Regresión Logística es un algoritmo de Machine Learning que se utiliza para los problemas de clasificación, es un algoritmo de análisis predictivo y se basa en el concepto de probabilidad [15].

Se puede llamar a una regresión logística un modelo de regresión lineal, pero la regresión logística utiliza una función de costo más compleja, esta función de costo se puede definir como la función sigmoidea o también conocida como la función logística en lugar de una función lineal.

La hipótesis de regresión logística tiende a limitar la función de costo entre 0 y 1. Por lo tanto, las funciones lineales no lo representan, ya que puede tener un valor mayor que 1 o menor que 0, lo que no es posible según la hipótesis de regresión logística [15].

Máquinas de soporte vectorial (SVM)

La máquina de soporte vectorial es una generalización de un clasificador llamado clasificador de margen máximo. El clasificador de margen máximo es simple, pero no se puede aplicar a la mayoría de los conjuntos de datos, ya que las clases deben estar separadas por un límite lineal [16].

Por eso, la máquina de soporte es un clasificador que se introdujo como una extensión del clasificador de margen máximo, que se puede aplicar en una gama más amplia de los casos.

Finalmente, la máquina soporte vectorial es simplemente una extensión adicional del clasificador de vectores de soporte para adaptarse a los límites de clase no lineales.

Algoritmo de árboles de decisión

Los algoritmos de árboles de decisión desglosan el conjunto de datos haciendo preguntas hasta que han reducido los datos lo suficiente como para hacer una predicción [17].

Según las características del conjunto de entrenamiento, el árbol de decisiones aprende una serie de preguntas para inferir las etiquetas de clase de las muestras.

El nodo inicial se llama raíz del árbol y el algoritmo dividirá el conjunto de datos en la característica que contiene la máxima ganancia de información de forma iterativa, hasta que las hojas (los nodos finales) sean puros.

Hiperparámetros de árboles de decisión

- Profundidad máxima: La profundidad máxima es la longitud más grande desde la raíz hasta la hoja. Una profundidad grande puede causar un ajuste excesivo y una profundidad pequeña puede causar un ajuste insuficiente. Para evitar el ajuste excesivo, podaremos el árbol de decisión estableciendo un hiperparámetro con la profundidad máxima [17].
- Número máximo de muestras: Al dividir un nodo, uno podría encontrarse con el problema de tener 99 muestras en una de las divisiones y 1 en la otra. Esto será un desperdicio de recursos, para evitarlo, podemos establecer un máximo para el número de muestras que permitimos para cada hoja. Puede especificarse como un número entero o como un flotante [17].
- Número mínimo de muestras: Análogo al anterior, pero con valores mínimos [17].
- Número máximo de funciones: Muy a menudo, tendremos muchas características para construir un árbol. En cada división, tenemos que verificar el conjunto de datos completo en cada una de las características, lo que puede ser muy costoso. Una solución a este problema es limitar la cantidad de características que se buscan en cada división. Si este número es lo suficientemente alto, es probable que encontremos una buena característica entre las que buscamos (aunque puede que no sea la perfecta). Sin embargo, si no es tan grande como el número de funciones, acelerará los cálculos de manera significativa [17].

K-Vecinos más cercanos (KNN)

Los algoritmos del vecino K-más cercano o KNN, pertenecen a un tipo especial de modelos de aprendizaje automático que con frecuencia se denominan algoritmos lazy [18].

Reciben este nombre porque no aprenden a discriminar el conjunto de datos con una función optimizada, sino que lo memorizan.

El nombre algoritmo perezoso también se refiere al tipo de algoritmos llamados no paramétricos. Estos son algoritmos basados en instancias, se caracterizan por memorizar el conjunto de datos de entrenamiento, y el aprendizaje perezoso es un caso específico de estos algoritmos, asociado a un costo computacional cero durante el aprendizaje [18].

Algoritmo KNN

El proceso general que sigue el algoritmo es:

- Elección del número de k y la métrica de distancia.
- Encontrar el vecino k más cercano de la muestra para clasificar.
- Asignar la etiqueta de clase por mayoría de votos.

6.3.3. Aprendizaje no supervisado

En algunos problemas de reconocimiento de patrones, los datos de entrenamiento consisten en un conjunto de vectores de entrada x sin ningún valor objetivo correspondiente. El objetivo en estos problemas de aprendizaje sin supervisión puede ser descubrir grupos de ejemplos similares dentro de los datos, donde se denomina agrupamiento, o determinar cómo se distribuyen los datos en el espacio, lo que se conoce como estimación de densidad. Para decirlo en términos más simples, para un espacio de n muestras x_1 a x_n , no se proporcionan etiquetas de clase verdaderas para cada muestra, por lo que se conoce como aprendizaje no supervisado [19].

Clasificación del aprendizaje no supervisado

Aprendizaje paramétrico no supervisado

En este caso, asumimos una distribución paramétrica de datos. Supone que los datos de muestra provienen de una población que sigue una distribución de probabilidad basada en un conjunto fijo de parámetros. Teóricamente, en una familia normal de distribuciones, todos los miembros tienen la misma forma y están parametrizados por media y desviación estándar. Eso significa que si conoce la media y la desviación estándar, y que la distribución es normal, conoce la probabilidad de cualquier observación futura. El aprendizaje paramétrico no supervisado implica la construcción de modelos

de mezcla gaussianos y el uso del algoritmo de maximización de expectativas para predecir la clase de la muestra en cuestión. Este caso es mucho más difícil que el aprendizaje supervisado estándar porque no hay etiquetas de respuesta disponibles y, por lo tanto, no hay una medida correcta de precisión disponible para verificar el resultado [19].

Aprendizaje no paramétrico no supervisado

En la versión no paramétrica del aprendizaje no supervisado, los datos se agrupan en grupos, donde cada grupo (con suerte) dice algo sobre las categorías y clases presentes en los datos. Este método se usa comúnmente para modelar y analizar datos con tamaños de muestra pequeños. A diferencia de los modelos paramétricos, los modelos no paramétricos no requieren que el modelador haga suposiciones sobre la distribución de la población y, por lo tanto, a veces se denominan métodos sin distribución [19].

Problema de agrupación

La agrupación en clústers puede considerarse el problema de aprendizaje no supervisado más importante, ya que, como cualquier otro problema de este tipo, trata de encontrar una estructura en una colección de datos sin etiquetar. Una definición imprecisa de agrupamiento podría ser “el proceso de organizar objetos en grupos cuyos miembros son similares de alguna manera”. Un clúster, por tanto, es una colección de objetos que son “similares” entre ellos y son “diferente” a los objetos que pertenecen a otros grupos [19].

Algoritmos de agrupación

Los algoritmos de agrupación en clústeres se pueden clasificar como:

- **Agrupación exclusiva:** Los datos se agrupan de forma exclusiva, de modo que si un determinado punto de datos pertenece a un clúster definido, no podría incluirse en otro clúster.
- **Agrupación superpuesta:** La agrupación superpuesta, utiliza conjuntos difusos para agrupar datos, de modo que cada punto puede pertenecer a dos o más agrupaciones con diferentes grados de pertenencia. En este caso, los datos se asociarán a un valor de membresía apropiado.

- Agrupación jerárquica: Un algoritmo de agrupamiento jerárquico se basa en la unión entre los dos grupos más cercanos. La condición inicial se realiza configurando cada punto de datos como un grupo. Después de algunas iteraciones, llega a los grupos finales deseados.
- Agrupación probabilística: Su enfoque es completamente probabilístico por lo cual será tratado con detalle a continuación.

Agrupación K-medias

K-means es uno de los algoritmos de aprendizaje no supervisados más simples que resuelve el conocido problema de agrupamiento. El procedimiento sigue una manera simple y fácil de clasificar un conjunto de datos dado a través de un cierto número de grupos (supongamos k grupos) fijados a priori. La idea principal es definir k centros, uno para cada grupo. Estos centroides deben colocarse de manera inteligente debido a que la ubicación diferente causa resultados diferentes. Por lo tanto, la mejor opción es colocarlos lo más lejos posible entre sí. El siguiente paso es tomar cada punto que pertenece a un conjunto de datos dado y asociarlo al centroide más cercano. Cuando no hay ningún punto pendiente, se completa el primer paso y se realiza un grupaje temprano. En este punto, necesitamos volver a calcular k nuevos centroides como baricentros de los conglomerados resultantes del paso anterior. Después de tener estos k nuevos centroides, se debe realizar una nueva vinculación entre los mismos puntos de conjunto de datos y el nuevo centroide más cercano. Se ha generado un bucle. Como resultado de este ciclo, podemos notar que los k centroides cambian su ubicación paso a paso hasta que no se realizan más cambios. En otras palabras, los centroides ya no se mueven [20].

Este algoritmo tiene como objetivo minimizar una función objetivo, en este caso una función de error al cuadrado. La función objetivo:

$$J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - C_j\|^2 \quad (6.2)$$

donde

$$\|x_i^{(j)} - C_j\|^2 \quad (6.3)$$

es una medida de distancia elegida entre un punto de datos x_i y el centro del grupo c_j , es un indicador de la distancia de los n puntos de datos desde sus respectivos centros del grupo [20].

Agrupación k-medias difuso

En el agrupamiento difuso, cada punto tiene una probabilidad de pertenecer a cada grupo, en lugar de pertenecer completamente a un único grupo como es el caso de las k-medias tradicionales. K-medias difuso trata específicamente con el problema en el que los puntos están algo entre centros o son ambiguos reemplazando la distancia con la probabilidad, que por supuesto podría ser alguna función de la distancia, como tener probabilidad relativa a la inversa de la distancia. Las k-medias difusas utilizan un centroide ponderado basado en esas probabilidades. Los procesos de inicialización, iteración y terminación son los mismos que se utilizan en k-medias. Los grupos resultantes se analizan mejor como distribuciones probabilísticas en lugar de una asignación estricta de etiquetas [19].

El algoritmo de k-medias difuso es el siguiente:

- Se supone un número fijo de conglomerados K .
- Se inicializa aleatoriamente las k-medias μK asociadas con los grupos y se calcula la probabilidad de cada punto de datos X_i sea miembro de un grupo determinado K , $P(Point X_i Has Label K | X_i, K)$.
- Se vuelve a calcular el centroide del cluster como el centroide ponderado dadas las probabilidades de pertenencia de todos los puntos de datos X_i :

$$\mu_k(n+1) = \frac{\sum_{x_i \in k} x_i \times P(\mu_k | x_i)^b}{\sum_{x_i \in k} P(\mu_k | x_i)^b} \quad (6.4)$$

- Se itera hasta la convergencia o hasta que se haya alcanzado un número de iteraciones especificado por el usuario (la iteración puede quedar atrapada en algunos máximos o mínimos locales).

Agrupamiento jerárquico

Dado un conjunto de N elementos para agrupar y una matriz de distancia (o similitud) NN , el proceso básico de agrupamiento jerárquico es el siguiente:

- Se asigna cada elemento a un grupo, de modo que si tiene N elementos, ahora tiene N grupos, cada uno con un único elemento. Deje que las distancias (similitudes) entre los grupos sean las mismas que las distancias (similitudes) entre los elementos que contienen.
- Se encuentra el par de clústeres más cercano (el más similar) y combínelos en un único clúster, de modo que ahora tenga un clúster menos.
- Calcular distancias (similitudes) entre el nuevo grupo y cada uno de los grupos antiguos.
- Repetir los pasos 2 y 3 hasta que todos los elementos estén agrupados en un único grupo de tamaño N [21].

6.3.4. Aprendizaje reforzado

El aprendizaje por refuerzo (RL) es un tipo de técnica de aprendizaje automático que permite a un agente aprender en un entorno interactivo mediante prueba y error utilizando la retroalimentación de sus propias acciones y experiencias [22].

Tanto el aprendizaje supervisado como el reforzado utilizan un mapeo entre entrada y salida, a diferencia del aprendizaje supervisado donde la retroalimentación proporcionada al agente es un conjunto correcto de acciones para realizar una tarea, el aprendizaje reforzado usa recompensas y castigos como señales de comportamiento positivo y negativo.

En comparación con el aprendizaje no supervisado, el aprendizaje por refuerzo es diferente en términos de objetivos. Si bien el objetivo del aprendizaje no supervisado es encontrar similitudes y diferencias entre los puntos de datos, en el caso del aprendizaje reforzado, el objetivo es encontrar un modelo de acción adecuado que maximice la recompensa acumulativa total del agente. La figura siguiente ilustra el ciclo de retroalimentación acción-recompensa de un modelo RL genérico.

6.3.5. Aprendizaje profundo

Redes neuronales artificiales

Las redes neuronales son un método de inspiración biológica para crear programas de computadora que pueden aprender y encontrar conexiones de forma independiente en los datos. Las redes son una colección de "neuronas" de software dispuestas en capas, conectadas entre sí de una manera que permite la comunicación [23].

Neurona simple

Cada neurona recibe un conjunto de valores de x (numerados de 1 a n) como entrada y calcula el valor de y predicho. El vector x realmente contiene los valores de las características en uno de los m ejemplos del conjunto de entrenamiento. Además, cada una de las unidades tiene su propio conjunto de parámetros, generalmente denominados w (vector de columna de ponderaciones) y b (sesgo), que cambia durante el proceso de aprendizaje. En cada iteración, la neurona calcula un promedio ponderado de los valores del vector x , basado en su vector de peso actual w y agrega sesgo. Finalmente, el resultado de este cálculo se pasa a través de una función de activación no lineal g [24][23].

$$Z = W_1X_1 + W_2X_2 + W_3X_3 + \dots + W_nX_n = W^T X \quad (6.5)$$

Funcionamiento de una capa

En principio, consideremos cómo se realizan los cálculos para una capa completa de la red neuronal. Para ello se usará el conocimiento de lo que está sucediendo dentro de una sola unidad y lo vectorizaremos en toda la capa para combinar esos cálculos en ecuaciones matriciales. Para unificar la notación, se escribirán las ecuaciones para la capa seleccionada $[l]$. Por cierto, el subíndice i marca el índice de una neurona en esa capa [24].

Cuando escribimos las ecuaciones para una sola unidad, usamos x y y , que eran respectivamente el vector columna de características y el valor predicho. Al cambiar a la notación general para la capa, usamos el vector a , es decir, la activación de la capa correspondiente. Por tanto, el vector x es la activación de la capa 0 - capa de entrada. Cada neurona de la capa realiza un cálculo similar

de acuerdo con las siguientes ecuaciones:

$$Z_i^{[l]} = W_T^i * a^{[l-1]} + b_i \quad a_i^{[l]} = g^I(Z_i^{[l]}) \quad (6.6)$$

Y para la capa 2, serían las siguientes:

$$Z_1^{[2]} = W_T^1 * a^{[1]} + b_1 \quad a_1^{[2]} = g^{[2]}(Z_1^{[2]}) \quad (6.7)$$

$$Z_2^{[2]} = W_T^2 * a^{[1]} + b_2 \quad a_2^{[2]} = g^{[2]}(Z_2^{[2]}) \quad (6.8)$$

$$Z_3^{[2]} = W_T^3 * a^{[1]} + b_3 \quad a_3^{[2]} = g^{[2]}(Z_3^{[2]}) \quad (6.9)$$

$$Z_4^{[2]} = W_T^4 * a^{[1]} + b_4 \quad a_4^{[2]} = g^{[2]}(Z_4^{[2]}) \quad (6.10)$$

$$Z_5^{[2]} = W_T^5 * a^{[1]} + b_5 \quad a_5^{[2]} = g^{[2]}(Z_5^{[2]}) \quad (6.11)$$

$$Z_6^{[2]} = W_T^6 * a^{[1]} + b_6 \quad a_6^{[2]} = g^{[2]}(Z_6^{[2]}) \quad (6.12)$$

Como puede apreciar, para cada una de las capas se realizan una serie de operaciones muy similares. Usar for-loop para este propósito no es muy eficiente, por lo que para acelerar el cálculo se usa la vectorización. En primer lugar, por apilamiento se unen los vectores horizontales de los pesos w (transpuesto) y se construye la matriz W . De manera similar, se apila el sesgo de cada neurona en la capa creando el vector vertical b . Por tanto es posible construir una única matriz de ecuaciones que permita realizar cálculos para todas las neuronas de la capa a la vez.

La ecuación elaborada hasta el momento implica solo un ejemplo. Durante el proceso de aprendizaje de una red neuronal, normalmente trabaja con grandes conjuntos de datos, hasta millones de entradas. Por tanto, el siguiente paso será la vectorización en varios ejemplos. Supóngase que el conjunto de datos tiene m entradas con nx características cada una. En primer lugar, se ponen juntos los vectores verticales x , a , y z de cada capa de la creación de la X , A y Z matrices, respectivamente, para posteriormente reescribir las ecuaciones previas [24].

Función de activación

Las funciones de activación son uno de los elementos clave de la red neuronal. Sin ellos, la red neuronal se convertiría en una combinación de funciones lineales, por lo que sería solo una

función lineal en sí misma. El modelo tendría una expansividad limitada, no mayor que la regresión logística. El elemento de no linealidad permite una mayor flexibilidad y la creación de funciones complejas durante el proceso de aprendizaje. La función de activación también tienen un impacto significativo en la velocidad de aprendizaje, que es uno de los principales criterios para su selección [24].

Función de pérdida

La fuente básica de información sobre el progreso del proceso de aprendizaje es el valor de la función de pérdida. En términos generales, la función de pérdida está diseñada para mostrar qué tan lejos estamos de la solución ideal.

Aprendizaje de la redes neuronales

El proceso de aprendizaje busca cambiar los valores de los parámetros W y b de modo que la función de pérdida se reduce al mínimo. Para lograr este objetivo, se busca mejorar el cálculo y usando el método de descenso de gradiente para encontrar una función mínima. En cada iteración se calculan los valores de las derivadas parciales de la función de pérdida con respecto a cada uno de los parámetros de nuestra red neuronal. Donde la derivada tiene una habilidad fantástica para describir la pendiente de la función. Gracias a eso es posible manipular variables para movernos cuesta abajo en el gráfico [23].

Descenso del gradiente

El descenso del gradiente es un proceso que ocurre en la fase de retropropagación donde el objetivo es volver a muestrear continuamente el gradiente del parámetro del modelo en la dirección opuesta en función del peso w , actualizando consistentemente hasta alcanzar el mínimo global de la función $J(w)$ [25].

En pocas palabras, se utiliza el descenso de gradiente para minimizar la función de costo $J(w)$.

Retropropagación

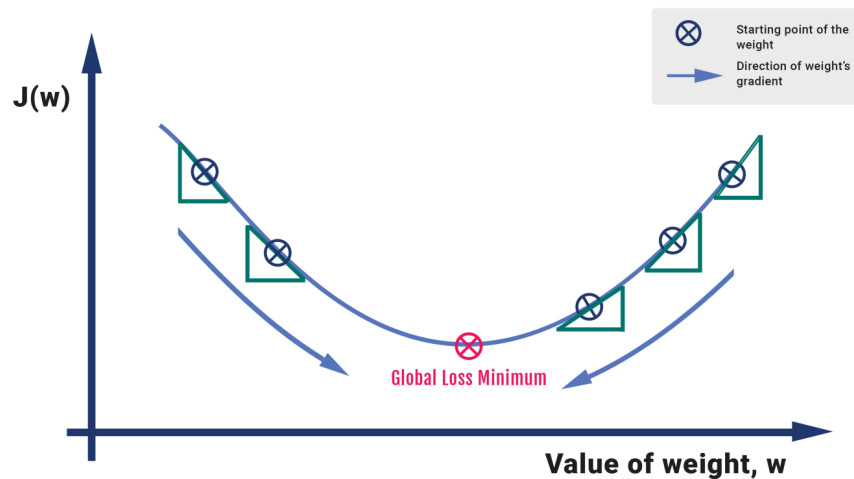


Figura 6.2: Cómo funciona el descenso del gradiente para un parámetro w

Backpropagation es un algoritmo que nos permite calcular un gradiente muy complicado. Los parámetros de la red neuronal se ajustan de acuerdo con las siguientes fórmulas:

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]} \quad (6.13)$$

$$b^{[l]} = b^{[l]} - \alpha db^{[l]} \quad (6.14)$$

En las ecuaciones anteriores, α representa la tasa de aprendizaje, un hiperparámetro que le permite controlar el valor del ajuste realizado. La elección de una tasa de aprendizaje es crucial: la establecemos demasiado baja, nuestra red neuronal aprenderá muy lentamente, la establecemos demasiado alta y no podremos alcanzar el mínimo. dW y db se calculan usando la regla de la cadena, función de derivadas parciales de pérdida con respecto a W y b [26].

Redes neuronales convolucionales

Una red neuronal convolucional es un algoritmo de aprendizaje profundo que puede tomar una imagen de entrada, asignar importancia (pesos y sesgos aprendibles) a varios aspectos/objetos en la imagen y poder diferenciar uno del otro. El procesamiento previo requerido en una red convolucional es mucho menor en comparación con otros algoritmos de clasificación. Mientras que en los métodos primitivos los filtros se diseñan a mano, con suficiente entrenamiento, las redes convolucionales tienen la capacidad de aprender estos filtros/características.

La arquitectura de una red convolucional es análoga a la del patrón de conectividad de las neuronas en el cerebro humano y se inspiró en la organización de la corteza visual. Las neuronas individuales responden a los estímulos solo en una región restringida del campo visual conocida como campo receptivo. Una colección de estos campos se superponen para cubrir toda el área visual [27].

Una red convolucional es capaz de capturar con éxito las dependencias espaciales y temporales en una imagen mediante la aplicación de filtros relevantes. La arquitectura se adapta mejor al conjunto de datos de imágenes debido a la reducción en el número de parámetros involucrados y la reutilización de pesos. En otras palabras, se puede entrenar a la red para que comprenda mejor la sofisticación de la imagen [27].

Proceso de una red convolucional

Imagen de entrada

El papel de las redes convolucionales es reducir las imágenes a una forma que sea más fácil de procesar, sin perder características que son críticas para obtener una buena predicción.. Esto es importante cuando vamos a diseñar una arquitectura que no solo sea buena para aprender características, sino que también sea escalable a conjuntos de datos masivos [27].

Capa de convolucion: el núcleo

El objetivo de la operación de convolución es extraer las características de alto nivel, como los bordes, de la imagen de entrada. No es necesario que las redes convolucionales se limiten a una sola capa convolucional. Convencionalmente, la primer capa convolucional es responsable de capturar las características de bajo nivel, como los bordes, el color, la orientación del degradado, etc. Con capas agregadas, la arquitectura se adapta también a las características de alto nivel, brindándonos una red que tiene una comprensión completa de imágenes en el conjunto de datos, de forma similar a como lo haríamos [27].

Hay dos tipos de resultados para la operación: uno en el que la característica convolucionada se reduce en dimensionalidad en comparación con la entrada, y el otro en el que la dimensionalidad aumenta o permanece igual. Esto se hace aplicando Relleno válido en el caso del primero, o Mismo relleno en el caso del segundo.

Capa de agrupación

Similar a la capa convolucional, la capa de agrupación es responsable de reducir el tamaño espacial de la entidad convolucionada. Esto es para disminuir la potencia computacional requerida para procesar los datos a través de la reducción de dimensionalidad. Además, es útil para extraer características dominantes que son invariantes rotacionales y posicionales, manteniendo así el proceso de entrenamiento efectivo del modelo [27].

Hay dos tipos de agrupación: agrupación máxima y agrupación media. La agrupación máxima devuelve el valor máximo de la parte de la imagen cubierta por el Kernel. Por otro lado, la agrupación promedio devuelve el promedio de todos los valores de la parte de la imagen cubierta por el núcleo.

Redes neuronales recurrentes

Red neuronal recurrente es una generalización de la red neuronal feedforward que tiene una memoria interna. La red recurrente es de naturaleza tal ya que realiza la misma función para cada entrada de datos, mientras que la salida de la entrada actual depende del último cálculo. Después de producir la salida, se copia y se envía de nuevo a la red recurrente. Para tomar una decisión, considera la entrada actual y la salida que ha aprendido de la entrada anterior [28].

A diferencia de las redes neuronales feedforward, Los redes recurrentes pueden usar su estado interno (memoria) para procesar secuencias de entradas. Esto los hace aplicables a tareas como el reconocimiento de escritura a mano conectado y no segmentado o el reconocimiento de voz. En otras redes neuronales, todas las entradas son independientes entre sí. Pero en una red recurrente, todas las entradas están relacionadas entre sí.

6.4. Trabajos Relacionados

6.4.1. Predicción mejorada de la estructura de proteínas usando potenciales del aprendizaje profundo

La predicción de la estructura de la proteína se puede utilizar para determinar la forma tridimensional de una proteína de su secuencia de aminoácidos. Este problema es de fundamental importancia ya que la estructura de una proteína determina en gran medida su función, sin embargo, la estructura de las proteínas puede ser difíciles de determinar experimentalmente. Ha habido un progreso considerable se ha realizado recientemente aprovechando la información genética. Es posible inferir que los residuos de aminoácidos están en contacto analizando la covariación en secuencias homólogas, que ayuda en la predicción de estructuras de proteínas. Aquí se muestra que es posible entrenar una red neuronal para hacer predicciones precisas de las distancias entre pares de residuos que transmiten más información sobre la estructura que el contacto de predicciones. Usando esta información, se contruye un potencial de fuerza media, lo que puede describir con precisión la forma de una proteína. Se encuentra que el potencial resultante puede ser optimizado por un algoritmo de descenso de gradiente simple para generar estructuras sin procedimientos de muestreo complejos. El sistema resultante, llamado AlphaFold, alcanza una alta precisión, incluso para secuencias con menos secuencias homólogas. [29][30].

En la reciente Crítica Evaluación de la predicción de la estructura de proteínas (CASP13): una evaluación ciega del estado del campo: AlphaFold creó estructuras de alta precisión (con modelado de plantillas (TM) puntuaciones de 0,7 o superior) para 24 de los 43 dominios de modelado gratuitos, mientras que el siguiente mejor método, que utilizó muestreo e información de contacto, logró tal precisión solo para 14 de 43 dominios. AlphaFold representa un avance considerable en la predicción de la estructura de proteínas [30].

Se espera que esta mayor precisión permita obtener información en la función y mal funcionamiento de las proteínas, especialmente en los casos en los que las estructuras para proteínas homólogas se han determinado experimentalmente.

6.4.2. Alta precisión en la predicción del contacto de proteínas utilizando redes neuronales totalmente convolucionales y características de secuencia mínima

Además de los datos de frecuencia de sustitución de las alineaciones de secuencias de proteínas, muchos métodos de vanguardia para la predicción de contactos se basan en fuentes adicionales de información, o características, de secuencias de proteínas para predecir los contactos residuo-residuo, como la accesibilidad de los disolventes, estructura secundaria predicha y puntuaciones de otros métodos de predicción de contactos. No está claro cuánta de esta información se necesita para lograr resultados de vanguardia. En dicho trabajo se muestra que utilizando modelos de redes neuronales profundas, las estadísticas de alineación simples contienen información suficiente para lograr una precisión de vanguardia. El método de predicción planteado, DeepCov, utiliza redes neuronales totalmente convolucionales que operan en datos de covarianza o frecuencia de pares de aminoácidos derivados directamente de alineaciones de secuencias [27].

Capítulo 7

Metodología

El presente proyecto tiene por objetivo el diseño e implementación de un método que posibilite el descubrimiento de patrones en proteínas tal que las características apreciables en modelos tridimensionales sean apreciables sin necesidad de algoritmos pesados y tiempos de computo prolongados.

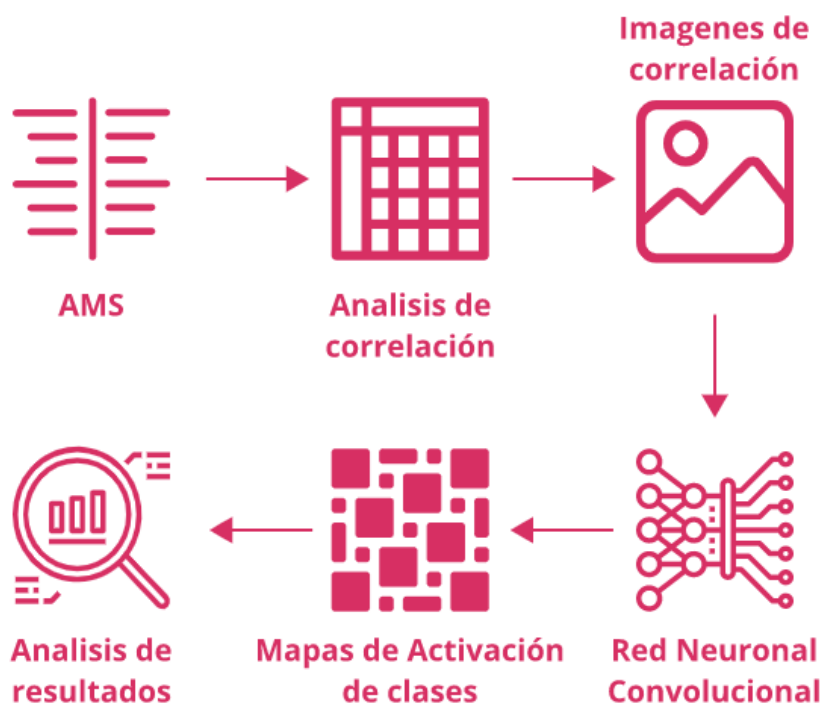


Figura 7.1: Descripción gráfica de la metodología para la búsqueda de patrones en secuencias de proteínas.

Previo a los pasos de investigación es necesario tener en cuenta los generales del proyecto ya que este se trata de un híbrido entre una investigación teórica y aplicada, pues por un lado se busca generalizar parámetros por el simple hecho de describir un fenómeno, sin embargo, en posteriores abordajes será posible utilizarla en el desarrollo de proyectos aplicados ya que el producto generado será una serie de implementaciones cuya adaptación permitirá el desarrollo de productos enfocados a un propósito.

El desarrollo y buen cause de la investigación considera tres etapas¹ generales las cuales serán desglosadas a continuación.

7.1. Investigación

Al hablar de investigación se considera en primer lugar la selección del tema y delimitación del mismo para lo cual se tiene como base el proyecto de investigación *Alphafold*, del cual se desprende la estructura primigenia para la delimitación del proyecto.

Posteriormente y tomando como base las referencias de *Alphafold* se genera un árbol temático el cual plantea las temáticas necesarias para el entendimiento y solución de la problemática, tal que sea posible sintetizar una serie de conocimientos al grado de replicarlos, adaptarlos y/o mejorarlos.

7.2. Diseño e implementación

Una vez que se cuenta con el bagaje teórico suficiente es posible comenzar planteando una serie de algoritmos de forma lógica-simbólica tal que se delimite el procedimiento computacional a seguir.

7.2.1. Selección de la información

A priori es necesario seleccionar el conjunto de datos que alimentarán a los algoritmos y por ende generarán conocimiento.

¹Las etapas posteriores al proceso de investigación se visualizan de forma generalizada en la Figura 7.1

Esto implica la delimitación que para el caso son los virus, considerando a la familia de los coronaviridae, tal que sea posible recabar familias de secuencias proteicas con las características mencionadas.

7.2.2. Procesamiento de los datos

Como punto de partida algorítmico es necesario considerará que el proyecto tiene como base a la ciencia de datos y como tal considera sus etapas. Por tanto el procesamiento de los datos crudos es vital ya que nos permite la depuración y extracción de elementos esenciales.

Una vez que a los datos le han sido eliminadas las impurezas se procede con el alineamiento múltiple de secuencias tal que sea posible homogeneizar y al conjunto de datos.

Posteriormente se genera una matriz de correlación a partir del alineamiento tal que sea posible conocer la dependencia entre aminoácidos.

Por ultimo, considerando el paradigma convolucional en el aprendizaje profundo se transforma a la matriz de correlación en una imagen a escala de grises.

7.2.3. Aprendizaje profundo

En este punto se considera el desarrollo del algoritmo de aprendizaje profundo, que para el caso es una red neuronal convolucional para la cual además de su implementación se deberán definir y configurar parámetros de inicio tal que sea posible obtener distogramas de distancia entre aminoácidos.

7.2.4. Búsqueda de patrones

Puesto que la salida de la red neuronal se presenta en forma de gráficas (imágenes) es preciso estudiar el conjunto de resultados mediante los mapas de activación de características con el fin de encontrar los patrones que pueden resultar en regiones conservadas.

7.3. Análisis de resultados

Para finalizar, una vez obtenido el producto final se procede con la definición de resultados y conclusiones, tal que se considere la relación con los planteamientos generados, así como la definición de los nuevos avances y descubrimientos.

Capítulo 8

Desarrollo

Una vez concluida la fase de investigación y recopilación de referencia se cuenta con el bagaje suficiente para comenzar el desarrollo del método propuesto, por ende se procederá a presentar de forma documental el proceso de diseño e implementación de los algoritmos.

8.1. Alineamiento múltiple de secuencias

En la alineación múltiple de secuencias (AMS) intentamos alinear tres o más secuencias relacionadas para lograr la máxima coincidencia entre ellas. El objetivo de AMS es organizar un conjunto de secuencias de tal manera que coincidan tantos caracteres de cada secuencia de acuerdo con alguna función de puntuación.

8.1.1. Puntuación

El proceso de puntuación de AMS se basa en la suma de las puntuaciones de todos los posibles pares de secuencias en el alineamiento múltiple de acuerdo con alguna matriz de puntuación.

8.1.2. Tipos de alineamiento múltiple de secuencias

Alinear tres o más secuencias puede ser difícil y casi siempre lleva mucho tiempo alinearlos manualmente. Por lo tanto, se utilizan algoritmos computacionales para producir y analizar estas alineaciones. La mayoría de los algoritmos de AMS utilizan programación dinámica y métodos heurísticos.

Construcción de alineación progresiva Este método, también conocido como método jerárquico o de árbol, fue desarrollado por Paulien Hogeweg y Ben Hesper en 1984. Construye un AMS final combinando alineaciones por pares comenzando con el par más similar y progresando hasta el par más distante.

Dos de los métodos populares de alineación progresiva que se utilizan en la actualidad son:

- Clustal Omega: Clustal Omega es un nuevo programa de alineación de secuencias múltiples que utiliza árboles guía sembrados y técnicas de perfil HMM para generar alineaciones entre tres o más secuencias.
- T-Coffee

8.1.3. Uso de Clustal Omega

1. Descargar el software Clustal Omega.
2. Crear un script con la siguiente estructura

```
#!/bin/bash
```

```
clustalo -i secuencia_original.fasta -o secuencia_alineada.fasta --auto -v
```

3. Ejecutar el script considerando la ruta del archivo original y la ruta de destino.

8.2. Estudio de correlación

Se sabe que los contactos residuo-residuo desempeñan un papel fundamental en el mantenimiento del pliegue nativo de las proteínas y en la guía del plegamiento de las proteínas [31]. Dichos contactos de residuos suelen estar bien separados con respecto a la secuencia primaria, pero muestran una gran proximidad dentro de la estructura 3D. Aunque en la literatura se han dado diferentes criterios geométricos para definir los contactos, típicamente, los contactos se consideran aquellos pares de residuos donde los átomos C- β se acercan dentro de 8 Å entre sí.

Para los enfoques puramente basados en secuencias para la predicción de contactos, hay dos fuentes principales de ruido en el análisis de mutaciones correlacionadas: el sesgo filogenético y los efectos de acoplamiento indirectos.

El último problema de determinar los efectos de acoplamiento directos e indirectos en el análisis de correlación mutación parece haber recibido la menor cantidad de atención en la literatura. Se ha relacionado el problema de desacoplar correlaciones de mutación en alineamientos de secuencia con el problema de Ising inverso en física estadística, donde se propuso una solución basada en la maximización de la entropía.

Por tanto se propone el uso de técnicas de estimación de covarianza inversa dispersa [32] para tratar los efectos de acoplamiento.

8.2.1. Covarianza generalizada

La covarianza mide la relación lineal entre dos variables. Aunque la covarianza es similar a la correlación entre dos variables, difieren de las siguientes maneras:

- Los coeficientes de correlación están estandarizados. Por lo tanto, una relación lineal perfecta da como resultado un coeficiente de 1. La correlación mide tanto la fuerza como la dirección de la relación lineal entre dos variables.
- Los valores de covarianza no están estandarizados. Por consiguiente, la covarianza puede ir desde infinito negativo hasta infinito positivo. Por lo tanto, el valor de una relación lineal perfecta depende de los datos. Puesto que los datos no están estandarizados, es difícil determinar la fuerza de la relación entre las variables.

Los valores de covarianza positivos indican que los valores por encima del promedio de una variable están asociados con los valores por encima del promedio de la otra variable y los valores por debajo del promedio están asociados de manera similar. Los valores de covarianza negativos indican que los valores por encima del promedio de una variable están asociados con los valores por debajo del promedio de la otra variable.

El coeficiente de correlación depende de la covarianza. El coeficiente de correlación es igual a la covarianza dividida entre el producto de las desviaciones estándar de las variables. Por lo tanto, una

covarianza positiva siempre producirá una correlación positiva y una covarianza negativa siempre generará una correlación negativa.

8.2.2. Métodos

Información mutua

El método más común para identificar mutaciones correlacionadas en AMS es calcular el IM entre dos sitios:

$$MI = \sum_{ab} f(A_i B_j) \log \frac{f(A_i B_j)}{f(A_i) f(B_j)} \quad (8.1)$$

donde:

- $f(A_i B_j)$ es la frecuencia relativa observada de par aminoácido ab en las columnas ij
- $f(A_i)$ es la frecuencia relativa observada de aminoácido a en la columna i
- $f(B_j)$ es la frecuencia relativa observada de aminoácido a en la columna j

Inferir sitios acoplados directamente usando covarianza

El punto de partida de nuestro método es considerar una alineación con m columnas y n filas, donde cada fila representa una secuencia homóloga diferente y cada columna un conjunto de aminoácidos equivalentes a lo largo del árbol evolutivo, con espacios considerados como un tipo de aminoácido adicional.

$$S_{ij}^{ab} = \frac{1}{n} \sum_k n_k = 1 (x_i^{ak} - x_i^{-a})(x_j^{bk} - x_j^{-b}) \quad (8.2)$$

donde

- X_i^{ak} es una variable binaria $x \in 0, 1$ que indica la presencia o ausencia del aminoácido a en la columna i en la fila k .
- X_j^{bk} la variable equivalente para observar el tipo de residuo b en la columna j en la fila k .

Este calculo de covarianza basado en variable de aminoácidos binarios es similar a [33], el cual determina unidades independientes. Con ello y tomando en cuenta la identidad estándar para la covarianza de $Cov(X, Y) = E(XY) - E(X)E(Y)$, y la expectativa de que una variable binaria $E(X)$ sea equivalente a la probabilidad de una observación positiva $p(x = 1)$, esto se simplifica a la

siguiente expresión basada en las frecuencias marginales observadas de aminoácidos $f(A_i)$ y $f(B_j)$ y pares de aminoácidos $f(A_i B_j)$ en los sitios dados en el conjunto de secuencias alineadas:

$$S_{ij}^{ab} = E(X_i^a X_j^b) - E(X_i^a)E(X_j^b) = f(A_i B_j) - f(A_i)f(B_j) \quad (8.3)$$

Cualquier elemento individual de esta matriz da la covarianza del aminoácido tipo a en la posición i con el aminoácido tipo b en la posición j . Al calcular la matriz inversa de la matriz de covarianza, se obtiene la matriz de precisión o concentración (θ), a partir de la cual se puede calcular una matriz de coeficientes de correlación parcial para todos los pares de variables de la siguiente manera:

$$\rho_{ij} = -\frac{\theta_{ij}}{\sqrt{\theta_{ii}\theta_{jj}}} \quad (8.4)$$

8.3. Generación de imagen

8.3.1. Modelos de color

Los modelos de color son una herramienta importante en el procesamiento digital de imágenes, ya que permiten analizar cada píxel desde un punto de vista particular, y así aprovechar toda la información presente dentro de la imagen.

Existen numerosos modelos de color, atendiendo cada uno a necesidades tan dispares que van desde la fisiología del ojo humano (Espacio de Hering, o espacio de colores oponentes), hasta el modelo de color sustractivo usado en la impresión sobre papel (CMYK). Destacando que algunos de estos modelos de color no tienen como objetivo hacerla visualización de colores más fiel a la realidad, sino que son abstracciones matemáticas, generalmente no lineales, que hacen posible el tratamiento de ciertas propiedades de la imagen.

Modelo RGB

La representación gráfica del modelo RGB se realiza mediante un cubo unitario con los ejes R, G y B. El origen (0,0,0) representa el negro y las coordenadas (1,1,1) el blanco. Los vértices del cubo en cada eje R, G y B, de coordenadas (1,0,0), (0,1,0) y (0,0,1) representan los colores primarios rojo, verde y azul. Los restantes tres vértices (1,0,1), (0,1,1) y (1,1,0) al magenta, cian y

amarillo respectivamente, colores secundarios y respectivamente complementarios del verde, rojo y azul. La diagonal del cubo representa la gama de grises desde el negro al blanco. En esta diagonal cada punto o color se caracteriza por tener la misma cantidad de cada color primario.

Modelo HSV

Las siglas H, S y V corresponden a Tono (hue), Saturación (saturation) y valor (value) respectivamente. También se denomina HSB, siendo B el brillo (brightness). El sistema coordenado es cilíndrico, y el subconjunto de este espacio donde se define el color es una pirámide de base hexagonal.[34]

El espacio HSV tiene la ventaja de ser invariante a las condiciones de luz; sin embargo, su alta complejidad computacional lo convierte en un recurso de difícil implementación.

8.3.2. Índices de correlación a imágenes RGB

En este punto del desarrollo la elección de un sistema de color es de gran importancia ya que se busca representar de forma gráfica valores que oscilan en un intervalo de $(-1, 1)$ tal que no existe una definición concreta de subintervalos sino que el espacio es tan grande como la complejidad de la matriz de índices de correlación.

Por otro lado es preciso optar por un sistema cuya representación sea simple y posibilite la abstracción de los patrones que definen la estructura de las secuencias de aminoácidos.

Teniendo en cuenta los argumentos presentados se puede inferir la viabilidad de optar por el modelo de color RGB, ya que este provee una abstracción simple así como la posibilidad de responder a un escenario con necesidad de un gran número de intervalos.

Definición de intervalos

Dado que el intervalo de acción es $(-1, 1)$ el cual refiere a la normalización de la correlación estadística, es necesario segmentar dicho intervalo en un número definido de fragmentos, tal que a los valores acotados por el fragmento se les asigne un color bajo el modelo RGB.

Para tal efecto se considera el total de fragmentos como la cantidad de posibles colores en el modelo RGB.

$$255 * 255 * 255 = 16,581,375 \quad (8.5)$$

Generación de intervalos

Una vez que se cuenta con el total de fragmentos realiza la operación teniendo en cuenta como entrada la matriz de índices de correlación generada en la fase anterior. Dicha matriz es recorrida celda a celda aplicando la función (7.6) y almacenando el valor en un arreglo bidimensional.

$$cRGB_{ij} = \frac{corr_{ij}}{\frac{1-(-1)}{255^3}} \quad (8.6)$$

Ajuste de intervalo

Aunque idealmente se considera que los datos se encuentran el intervalo de $(-1, 1)$, existe una gran disparidad en la practica ya que puede existir una concentración de datos que no corresponda al intervalo como tal, teniendo así una cota inferior y superior bastante alejada de los valores propuestos.

Por tanto, es preciso definir una cota superior e inferior acorde con el conjunto de datos. Esto se logra buscando de entre el total de las matrices de índice de correlación el valor máximo y mínimo los cuales paran a ser las nuevas cotas superior e inferior.

$$(maxCorr, minCorr) \quad (8.7)$$

Exportación a imagen

Una vez que se ajusto y genero el intervalo se tiene un arreglo bidimensional con valores de $[0, 16581375]$, los cuales requieren ser representados en su forma RGB para lo cual se utiliza un arreglo tridimensional del tipo $n * m * 3$ donde el espacio bidimensional $m * n$ representa las dimensiones de la matriz de índices de correlación y el valor 3 las capas RGB.

Posteriormente se transforma la matriz RGB en una imagen PNG mediante el método *fromarray()* de la librería Pillow.(Véase Anexos)

8.4. Red neuronal convolucional

Hasta este punto el desarrollo ha si dedicado a la obtención de imágenes que de forma gráfica ilustren la correlación entre los aminoácidos de una secuencias dada, esto con el fin de abstraer la topología y distancia entre aminoácidos propuesta por la inferencia de sitios acoplados.

Ahora con el set de datos transformado en un abstracción simplificada como lo son las imágenes es posible utilizar un algoritmo de aprendizaje profundo, en particular las redes neuronales convolucionales (CNN), las cuales permiten un procesamiento eficiente de sets de imágenes de tal forma que sea posible generar inferencias.

Para el caso planteado la utilización de CNN es imprescindible ya que se busca la clasificación de las proteínas, esto bajo el supuesto de que el conjunto de proteínas analizadas se encuentran subdivididas mediante clases, donde cada clase refiere a la pertenencia a una familia de proteínas particular.

8.4.1. Tuberías de datos

Cuando se desarrolla un proyecto de Machine Learning es necesario tener en cuenta las diferentes necesidades que existen ya que en un escenario de desarrollo la generación de inferencias es la ultima de varias fases, y como se ha visto hasta el momento se ha hecho una labor donde la ingeniería de datos ha sido el campo ha desarrollar.

Por tanto, para hacer la correlación entre un marco de ingeniería de datos y un marco de ciencia de datos (implementación de aprendizaje profundo) es necesario considerar las necesidades de los datos, pues su transformación aunque de gran complejidad no es la fase donde la dificultad se ve representada, sino en el almacenamiento y puesta a punto como input de un algoritmo de aprendizaje profundo. Es aquí donde las tuberías de datos juegan un papel fundamental.

En el caso particular del proyecto se precisa la necesidad de automatización y gestión de los flujos de datos, pues el procesamiento de un dataset como el que las secuencias de aminoácidos precisa que puede llegar a unos cientos de megabytes al cabo de las transformaciones matemáticas puede alcanzar el orden de giga o hasta terabytes.

En el algoritmo estas tuberías se representan como una serie de bash scripts los cuales regulan el flujo de datos ajustando los resultados al espacio de trabajo que esta limitado a 10 Gigabytes, esto implica una serie de eliminaciones múltiples donde lo único que prevalece son las secuencias alineadas y las imágenes obtenidas.

8.4.2. Librerías

Previo al desarrollo de los algoritmos necesarios para un aprendizaje profundo es preciso considerar las capas de abstracción utilizadas, en esta caso el framework que simplifica la construcción de redes neuronales profundas.

El presente proyecto desarrolla los algoritmos de aprendizaje profundo mediante la librería pytorch.

8.4.3. Exploración del dataset

Una vez que el espacio de trabajo se tiene a punto es posible desarrollar los algoritmos como tal, es por tal motivo que la fase inicial refiere a la exploración del set de datos, pues es en este punto donde como primer paso es necesario canalizar un tubería de datos que desemboque en la antesala de la CNN.

Dicho punto es un paso previo en la entrada de datos de la CNN, ya que se precisan ajustes donde la base es la homogeneización de los datos. Esto es vital ya que al estar trabajando con proteínas que van de los 500 a los 1500 aminoácidos no se cuentan con inputs uniformes.

Esto sugiere que la homogeneización de datos parte del redimensionamiento de las imágenes a un estándar de 150px * 150px, ya que así en la fase de procesamiento convolucional la red visualizará elementos semejantes.

8.4.4. Conjunto de datos de entrenamiento y validación

La creación de un modelo de aprendizaje automático supervisado se trata de crear un programa que sea capaz de generalizar para introducir muestras que nunca antes había visto. Esta tarea

requiere exponer el modelo, durante el entrenamiento, a un cierto número de variaciones de ejemplos de entrada, lo que probablemente conducirá a una precisión suficiente.

Conjunto de entrenamiento

Este conjunto de datos incluye el conjunto de ejemplos de entrada en los que se ajustará el modelo, o en el que se entrenará, ajustando los parámetros (es decir, ponderaciones en el contexto de las redes neuronales).

Conjunto de validación

Para que el modelo sea entrenado, necesita ser evaluado periódicamente, y para eso es exactamente el conjunto de validación. Mediante el cálculo de la pérdida (es decir, la tasa de error) que el modelo arroja sobre el conjunto de validación en cualquier punto dado, podemos saber qué tan preciso es. Ésta es la esencia del entrenamiento. Posteriormente, el modelo ajustará sus parámetros en función de los resultados de la evaluación frecuente en el conjunto de validación.

Conjunto de prueba

Esto corresponde a la evaluación final por la que pasa el modelo una vez finalizada la fase de entrenamiento (utilizando conjuntos de entrenamiento y validación). Este paso es fundamental para probar la posibilidad de generalizar el modelo. Al usar este conjunto, es posible obtener la precisión de trabajo de nuestro modelo.

8.4.5. Segmentación de los datos en lotes

No es recomendable pasar todo el conjunto de datos al modelo para entrenarlo, porque el tamaño de memoria es fijo y existe una alta probabilidad de que los datos de entrenamiento excedan la capacidad de memoria de la CPU o GPU, por lo que se divide el conjunto de datos en lotes y en lugar de entrenar el modelo en su totalidad en una sola fase. El tamaño del lote se puede decidir de acuerdo con la capacidad de la memoria, generalmente, toma una potencia de 2. Por ejemplo, el tamaño del lote puede ser 16, 32, 64, 128, 256, etc.

Por tanto se toman lotes de imágenes de tamaño 128 y 200 de los datos para validación y el resto de datos para entrenamiento. Para dividir aleatoriamente las imágenes en entrenamiento y

prueba, PyTorch proporciona `randomSplit()`.

Los datos se dividen en lotes utilizando la clase PyTorch `DataLoader`. Se crean dos objetos `trainDl` y `valDl` para los datos de entrenamiento y validación, respectivamente, proporcionando parámetros, datos de entrenamiento y tamaño de lote en la clase `DataLoader`.

8.4.6. Modelo base para la clasificación de imágenes

Primero, preparamos una clase base que amplía la funcionalidad de `torch.nn.Module` (clase base utilizada para desarrollar todas las redes neuronales). Agregamos varias funcionalidades a la base para entrenar el modelo, validar el modelo y obtener el resultado de cada época. Esto es reutilizable y se puede utilizar para cualquier modelo de clasificación de imágenes, no es necesario reescribirlo cada vez.

8.4.7. Modelo de clasificación

En este modelo, hay 4 bloques CNN, y cada bloque consta de 2 capas de convolución y 1 capa de agrupación máxima. La función de activación Relu se utiliza para eliminar valores negativos del mapa de características porque no puede haber valores negativos para ningún valor de píxel. Stride (1,1) usado y el Padding también es 1.

Después de aplicar la convolución y extraer características de la imagen, se usa una capa de aplanar para aplanar el tensor que tiene 3 dimensiones. La capa plana convierte el tensor en unidimensional. Luego se agregaron 3 lineales para reducir el tamaño del tensor y aprender las características.

8.4.8. Configuración del dispositivo de procesamiento

En los casos en los que se desarrollan redes neuronales realmente profundas, el entrenamiento en la CPU durará mucho tiempo, lo cual resulta impráctico o puede entorpecer el flujo de desarrollo debido a los altos tiempos de espera.

Las operaciones de álgebra lineal se realizan en paralelo en la GPU y, por lo tanto, se puede lograr una disminución de alrededor de 100 veces en el tiempo de entrenamiento. Las implicaciones de este

paradigma son bastas ya que también es una opción para realizar entrenamiento en múltiples GPU, lo que una vez más disminuiría el tiempo de entrenamiento.

En este sentido es posible implementar procesamiento por GPU para cualquier operación destacando su efectividad en procesos que involucren el calculo con matrices.

Sin embargo, para el presente desarrollo se utilizará pytorch-cuda únicamente para el proceso de entrenamiento del modelo de aprendizaje, dejando para futuras revisiones el procesamiento GPU de la mayoría de las operaciones matriciales.

8.4.9. Ajuste del modelo

Ahora se procede a entrenar el modelo de clasificación propuesto en el conjunto de datos de entrenamiento. Entonces, eso primero define los métodos de ajuste, evaluación y precisión.

8.4.10. Evaluación mediante datos de prueba

Una vez que el modelo se ha entrenado y validado es realizar ensayos en los cuales se introducen imágenes a la red de tal forma que esta infiera la clase a la cual pertenece.

Es en este punto donde la red se muestra como un producto funcional ya que cumple de forma general su propósito como clasificador, sin embargo, para el fin que ocupa al presente proyecto está es la antesala de lo que será el descubrimiento de patrones los cuales definen a una familia de proteínas en particular.

8.5. Visualización de características

En la literatura y en los medios se ha presentado a los modelos de aprendizaje profundo como algoritmos de caja negra en donde existen inferencias y resultados generados con cierta precisión, no obstante, no se precisa de forma clara el comportamiento de los mecanismos internos que conllevan a la generación de resultados.

Sin embargo, hay una manera de interpretar lo que cada filtro está haciendo en una red neuronal convolucional, y el tipo de imágenes que está aprendiendo a detectar.

Para ello se entrenaría a una CNN alimentándola con imágenes y etiquetas, y usando descenso del gradiente o un método de optimización similar para ajustar los pesos de la red neuronal para que prediga la etiqueta correcta.

A lo largo de este proceso, se esperaría que la imagen permaneciera intacta, y lo mismo se aplica a la etiqueta.

Sin embargo, si se toma cualquier imagen, eligiendo un filtro convolucional en la red (ya entrenada) y se aplica un optimizador en la imagen de entrada para maximizar la salida de ese filtro mientras se dejan constantes los pesos de la red cambia la perspectiva; ya se está entrenando un modelo para predecir la etiqueta de una imagen sino que ahora se está ajustando la imagen al modelo, para que genere el resultado deseado.

Es en este sentido que se propone una solución basada en la visualización de características donde la búsqueda de patrones puede ser guiada por el análisis de los mapas de activación de clases.

De forma general se requiere un conjunto de proteínas (mediante su representación gráfica) las cuales hayan sido verificados como miembros de una familia (la cual refiera a una clase considerada en el entrenamiento de la CNN) y generar el histograma que refleje la influencia de cada filtro convolucional donde los picos sean candidatos a ser el elemento activador que excita a una neurona para generar una inferencia particular. Una vez obtenido el histograma por proteína se procede a la comparación de los picos tal que sea posible encontrar un mapa o un conjunto de mapas de activación los cuales serán conocidos como patrones de definición de clase.

Para tal análisis se precisan dos posibilidades, un estudio exhaustivo o un estudio estocástico. Para el primer caso se considera al total de proteínas pertenecientes a una familia dada y en el segundo caso se emplea un subconjunto de una familia generado seleccionado mediante un muestreo aleatorio simple.

Una vez concluido el experimento se considerará el descubrimiento de un patrón si para una familia existe a lo menos un mapa de activación común que dicho de otra forma puede representar un motivo conservado para esa familia.

Capítulo 9

Experimentación y pruebas

El desarrollo del presente proyecto considera una serie de etapas entre las cuales existe un flujo de información, en esencia la información es la misma, donde su diferencia se denota bajo su estructura y representación.

El flujo comienza con la introducción de secuencias de aminoácidos los cuales transforman su representación a través de diversas estructuras computacionales.

De dichas estructuras es posible resaltar aquellas que refieren a 3 etapas:

- Datos en bruto: Secuencias de aminoácidos
- Entrada del modelo de aprendizaje profundo: Imágenes
- Salida del modelo de aprendizaje profundo: Mapas de activación de clase

9.1. Secuencias de aminoácidos

Las secuencias de aminoácidos son un conjunto de datos de dimensión n los cuales se obtuvieron tomando en cuenta los resultados presentados en [35]. Para lo cual se consideró el siguiente proceso:

- Seleccionar el conjunto con el mayor numero de cliques.
- Seleccionar una de las filas de relación.
- Descargar el conjunto de secuencias referidas en los identificadores de proteína.

- Concatenación de secuencias de aminoácidos en un sólo archivo .fasta.

Una vez que se concentró el total de los datos el resultado elemental es un archivo punto .fasta cuya concentración representa una posible familia. Dicho archivo tiene por objetivo servir como base para los posteriores procesamiento.

9.1.1. Matriz de índices de correlación

Previo al input directo del modelo de aprendizaje profundo fue necesario hacer una serie de transformaciones que resultaron en matrices de covarianza.

En una primera versión el desarrollo de este experimento se realizó mediante el análisis de una familia, la cual se encontraba contenida en un conjunto de 64 familias de 68 organismos distintos.

Sin embargo, al analizar el propósito de las matrices de covarianza se concluyó que era insuficiente el trabajar con una sola familia ya que se precisaba necesario el poseer un dataset de tamaño considerable (en el orden de los miles de datos) ya que la CNN precisa una alta cantidad de datos así como la necesidad de una variedad de clases que permitiera la generación un clasificador.

Por tanto, se modificó el algoritmo de tal forma que fuese posible:

1. Recuperar las secuencias
2. Agrupar las secuencias por familia en archivos .fasta
3. Generar la alineación múltiple de secuencias por familia
4. Generar las matrices de correlación del total de las secuencias en una familia
5. Generar las representaciones gráficas (imágenes) de las matrices de correlación

9.2. Imágenes

Versión 1

Teniendo en cuenta el conjunto de datos inicial (Véase Anexos) se realizó el procesamiento considerando el procedimiento planteado (Véase Capítulo 7: Desarrollo) tal que fue posible obtener

64 imágenes las cuales refieren al número al tamaño del clique para la fila analizada del conjunto de datos.

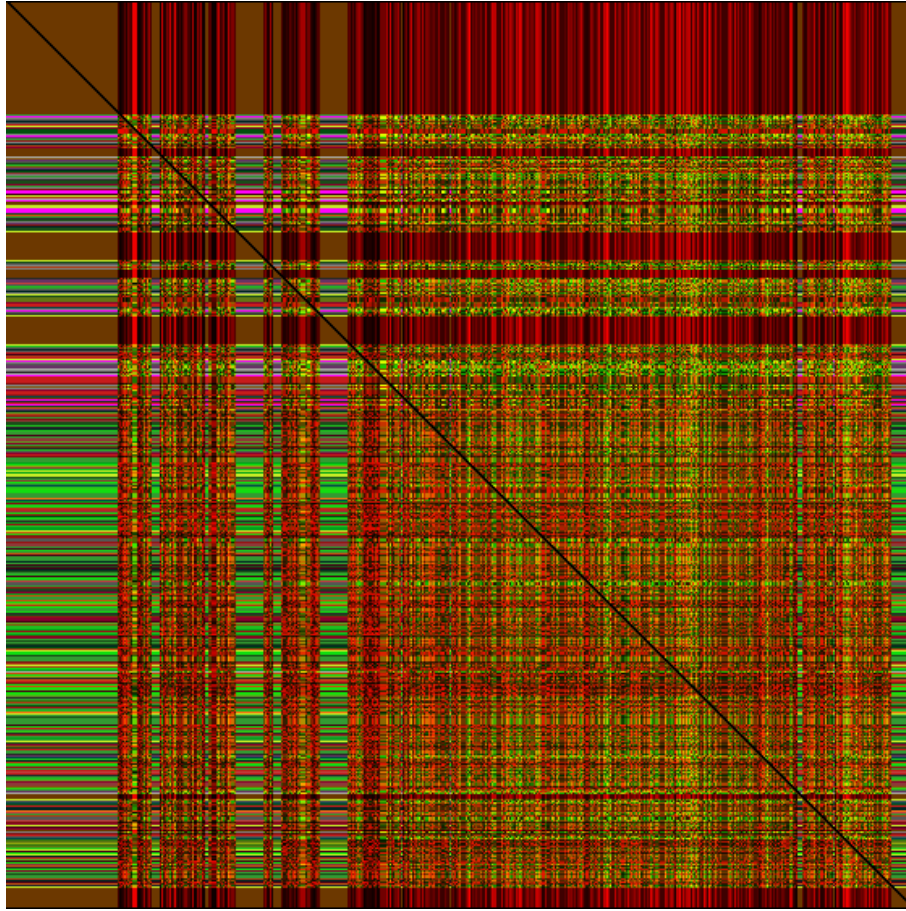


Figura 9.1: Representación gráfica de los índices de correlación: img064-0-3

Una vez que se obtuvieron las imágenes se realizó un paso de verificación con el objetivo de validar la diversidad y en la representación de cada imagen, tal que no existiera algún error o hubiera algún dato repetido de forma deliberada.

Para ello se tomó una muestra de imágenes las cuales se compararon de forma "manual" mediante un software de edición de imágenes. Esto considerando pares de imágenes donde el resultado fue una tercera imagen donde se mostraban los píxeles de diferencia.

Versión 2

Una vez realizada la implementación del algoritmo para el procesamiento de una familia (Véase 8.2 Imágenes, Versión 1) se procedió a la adaptación del algoritmo tal que fuere posible analizar y

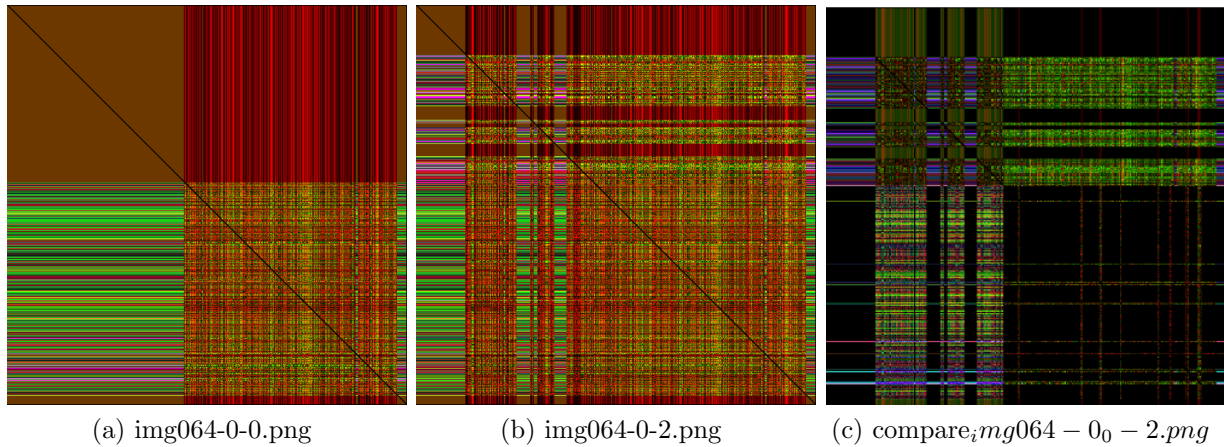


Figura 9.2: Diferencia de imágenes de índices de correlación

transformar no solo a una familia en particular sino a un conjunto de familias pertenecientes a un clique.

Para esto se tuvo en cuenta un total de 66 familias correspondientes a 68 organismos lo cual se traduce en aproximadamente 4488¹ proteínas que pasarían a una representación gráfica (imagen).

La puesta en marcha de algoritmo representó una serie de dificultades y retos técnicos, ya que al estar trabajando con operaciones matriciales se considera un tiempo de cómputo considerable. A continuación el resumen correspondiente:

- Se procesaron el total de las 66 familias obteniendo 3800 imágenes.
- El tiempo de procesamiento por familia osciló entre 30 y 90 minutos.
- Hubo una pérdida del 10% de los datos debido a la precisión del 0 del procesador².
- Se consideraron 1800 imágenes referentes a 32 familias
- El total en Gigabytes del conjunto de procesamiento³ ascendió a 171.

¹Se considera aproximado, ya que no todas las familias poseen 68 organismos.

²Esto habla del truncamiento de los datos por parte de la computadora, lo cual en estos casos se interpretó como un valor infinito, lo cual anuló la operación en curso.

³Refiere a los datos derivados del cálculo de correlación.

9.3. Mapas de activación de clase

El esfuerzo de transformar las proteínas en su forma de secuencias a una abstracción gráfica representa la construcción del dataset necesario para alimentar al algoritmo de aprendizaje profundo.

9.3.1. Red neuronal convolucional

Los algoritmos de aprendizaje profundo, en particular las redes neuronales convolucionales son un constructo algorítmico dual en el sentido de poseer una complejidad intrínseca, sin embargo, dicha complejidad se ve reducida a una simpleza sustancial al ver que la composición interna de las neuronas se reduce a sumas ponderadas y transformaciones lineales.

Previamente se mostró la composición y arquitectura de la CNN, lo cual deja de manifiesto el extenso esfuerzo plasmado, sin embargo, no es hasta que es ejecutada bajo el dataset de entrenamiento.

Es importante hacer notar que el dataset aunque clasificado fue necesario segmentarlo. En primera instancia se redujo el número de clases⁴ a la mitad con el propósito de reducir principalmente la carga computacional, por tanto las clases a estudiar fueron 32.

Una vez delimitado el número de clases se definieron los conjuntos de entrenamiento, validación y prueba bajo la proporción 80-20-20; donde el conjunto de entrenamiento fue el 80 % del total, el conjunto de validación fue el 20 % del conjunto de entrenamiento, y el conjunto de prueba el 20 % del total.

9.3.2. Visualización de características

Como ya se mencionó el objetivo central del método se enfocó en el descubrimiento de patrones los cuales pudieran ser considerados como motivos conservados, sin embargo, lograr tal objetivo bajo la arquitectura estándar de una CNN presentaba inconvenientes ya que la definición de las salidas implicaba un diseño minucioso que consideró una investigación y prueba precisa de los

⁴Al hablar de una clase o categoría se hace referencia al número de familias de proteínas pertenecientes al conjunto de datos total.

componentes, lo cual requería tiempo y recursos con los que no se contaban. Por tal motivo se exploró una idea más abstracta y por tanto interesante, la visualización de características.

Esta técnica precisa un procedimiento de dos pasos, por un lado la recuperación y representación de los filtros convolucionales obteniendo un set de filtros (imágenes en escala de grises) relativos a cada capa convolucional.

Una vez obtenidos los filtros convolucionales fue posible contrastar imágenes específicas con el total de filtros⁵, tal que al final de la iteración se obtuvo un arreglo con el conjunto de probabilidades de activación de cada filtro sobre la imagen.

Al hablar de probabilidad de activación se está hablando de la influencia que tuvo un filtro convolucional sobre la imagen tal que las neuronas clasificaran a la imagen en una clase en particular, lo cual es medido mediante una probabilidad en el intervalo de 0 a 1, tal que si el valor tiende a ser tan significativo que la influencia del filtro sobre la imagen es alta. Al filtro que resulta con la mayor probabilidad se le conoce como mapa de activación de clase.

Para este proceso se tomó una muestra aleatoria de cinco elementos de cada una de las clases⁶ tal que fuera posible obtener el mapa de activación para cada imagen de la muestra por clase.

Una vez finalizada la corrida de pruebas para el conjunto de datos se procedió con el análisis de los mapas de activación por familia, donde el análisis parte de la comparación de los mapas con el fin de encontrar coincidencias.

⁵A este proceso se le conoce como generación de mapas de características

⁶Esto considerando el conjunto de datos que únicamente se segmentó al 50 %

Capítulo 10

Análisis e interpretación de resultados

Una vez realizadas el total de pruebas y experimentos se recabó un conjunto de elementos los cuales hacen las veces de evidencia del funcionamiento y análisis de los elementos estudiados.

Aunque en el proceso se obtuvieron una gran variedad de estructuras informáticas las cuales hacen las veces de resultados, no todas cuentan con la relevancia suficiente como para ser analizadas en este documento. Por principio de cuentas se debe recalcar la presencia de dos fases en el desarrollo del método lo cual remite a la existencia de dos tipos de resultados, por un lado se tienen las imágenes que representan la correlación entre aminoácidos y por otro lado se tienen los mapas de características de clase obtenidos de la CNN.

La generación de las imágenes de correlación son el resultado de un proceso de análisis estadístico y algebraico donde se utilizaron tres tipos de herramientas: covarianza, correlación y álgebra lineal.

10.1. Imágenes de correlación

Una vez que se obtuvieron el total de las secuencias (desde el repositorio de NCBI) la tarea más importante consideraba la búsqueda de la correlación entre los pares de aminoácidos de una proteína dada, por tanto, como resultado se pretendía generar un conjunto de matrices correlación (semejante al número de proteínas) cuyo valor en cada celda se encontrará en el rango de -1 a 1¹, sin embargo, una vez que se sometió al conjunto de datos a dicha prueba se encontró que existía

¹Esto considerando el rango normalizado de la correlación estadística, donde la tendencia hacia -1 o 1 indican una correlación perfecta.

una tendencia en la cota de los intervalos la cual no sobrepasaba el $1 * 10^{-24}$ en su cota superior y el 0 en su cota inferior.

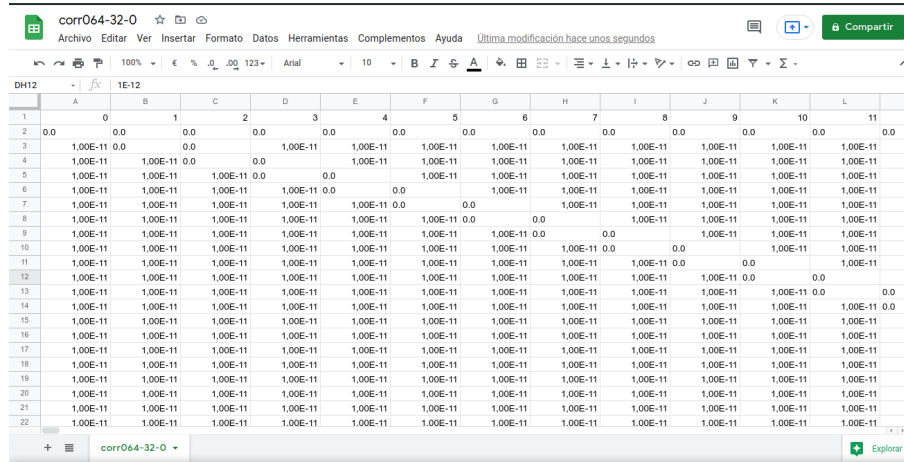


Figura 10.1: Fragmento de matriz de índices de correlación: corr064-17-0

Tal hecho no impedía un correcto desarrollo de los pasos subsecuentes, sin embargo, una vez que se pretendía generar la representación gráfica (imagen de correlación) los resultados eran ineficientes ya que las imágenes se presentaban como un espacio con píxeles RGB(0,0,0) en prácticamente la totalidad de la imagen y sólo una pequeña fracción de píxeles RGB(255,255,255)². A priori esto no se mostraba como un impedimento para el estudio de los resultados, no obstante, la intención de la etapa mencionada pretendía la generación de imágenes de correlación que presentaran la diversidad de los datos, la cual existía, ya que se podía observar tal diversidad en las matrices de correlación, con la consideración de que tal diversidad era observable en la cota $[0, 1 * 10^{-24}]$ y no en $[-1, 1]$ como se tenía definido en los algoritmos.

Tal hecho planteó la posibilidad de modificar el enfoque, pero fue gracias al estudio de la distribución (Figura 10.3) de los datos³ en las matrices de correlación que se logró identificar la existencia de datos aberrantes los cuales en proporción eran pocos y podían ser atribuibles a causas aleatorias (o un suceso poco común), por lo cual se tomó la decisión de despreciarlos y con ello ajustar la cota de correlación al intervalo $[0, 1 * 10^{-24}]$.

Una vez realizada la modificación se observó un importante cambio en la generación de las imágenes de correlación, ya que ahora presentaban una diversidad semejante a la de sus pares

²Esto representa una imagen negra con algunos puntos blancos como se observa en la Figura 10.2

³Mediante un diagrama de caja y brazos

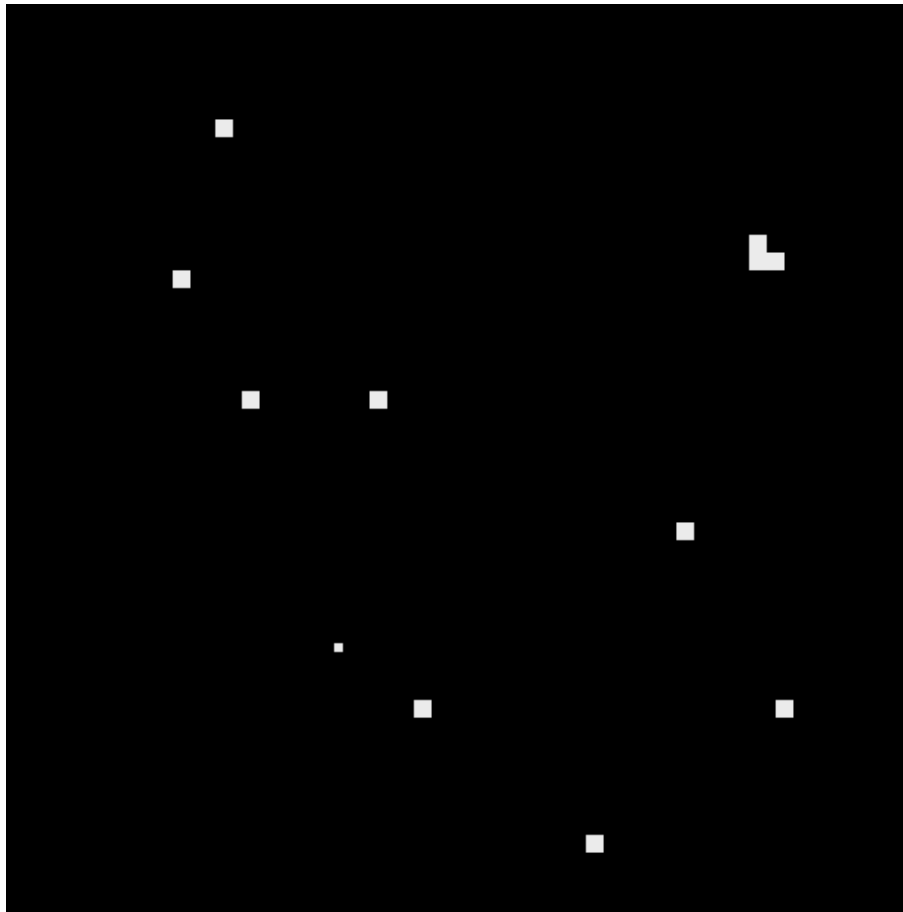


Figura 10.2: Imagen de correlación generada con el intervalo $[-1,1]$ (img064-0-0_11)

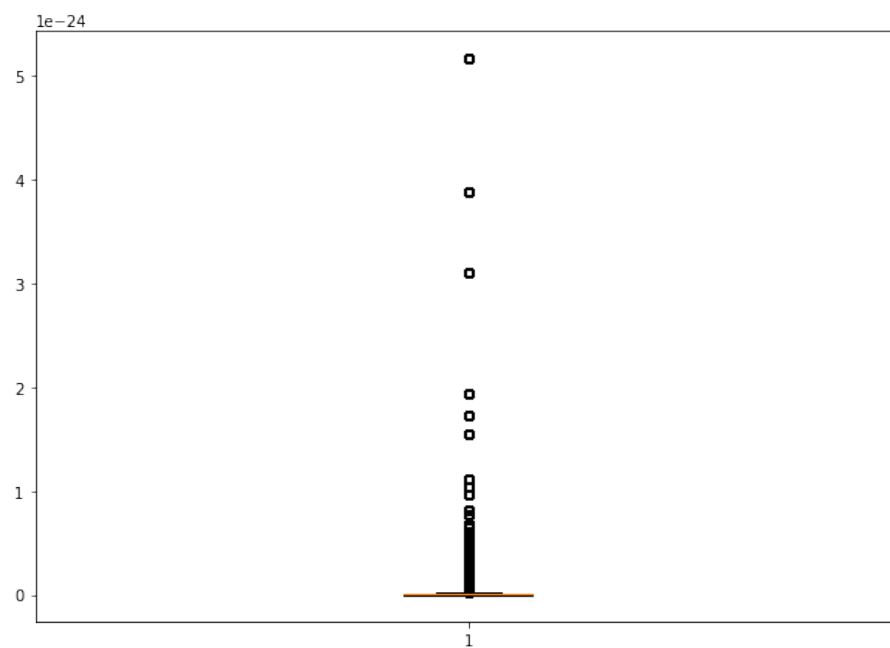


Figura 10.3: Distribución de los datos de la proteína 064-0-0

matriciales (Figura 10.4).

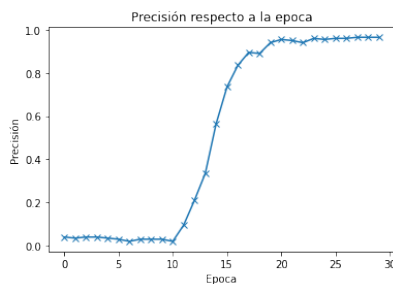


Figura 10.4: Imagen de correlación generada con el intervalo $[0, 1 * 10^{-24}]$ (img064-0-0)

10.2. Mapas de activación de clases

Habiendo obtenido el total de las imágenes de covarianza, el siguiente paso lógico era la puesta en marcha de la CNN la cual requería un conjunto etiquetado de imágenes de covarianza (Como se explicó en los capítulos 8 y 9).

Como primer punto se presenta el proceso de entrenamiento del modelo el cual se realizó durante 30 épocas, esto posibilitando su convergencia a una precisión que tienda a 1. En un principio dicho entrenamiento comenzó bajo (por debajo del 0.1), y la tendencia prosiguió por alrededor de 10 épocas siendo en la 11 y 12 cuando la tendencia mostraba un aumento y fue a partir de la época 14-15 que la precisión logró sobrepasar el 0.8 llegando hasta 0.96 y así en las épocas finales.



(a) Gráfico de precisión



(b) Gráfico de pérdida

Figura 10.5: Performance del entrenamiento de la CNN

Una vez entrenado el modelo fue posible someterlo a prueba utilizando datos (imágenes de covarianza) destinados para ello, logrando alcanzar una precisión de 94 %, logrando acertar en la clasificación en todas la ocasiones.

Posteriormente se alcanzó la etapa final en la cual se lograrón obtener los filtros convolucionales y realizando una iteración sobre un conjunto de imágenes de covarianza se encontró el mapa de activación de clases de los ejemplo sometidos a prueba (Figura 10.6). Con esto se pudo demostrar la existencia de patrones dentro de las familias, ya que se pudo apreciar que para el conjunto de ejemplos (seleccionados de forma aleatoria) se encontró que el mismo filtro convolucional lograba excitar las neuronas con una alta probabilidad lo cual refiere a que la clasificación ocurrió gracias a la existencia de elementos comunes los cuales pudieron ser detectados.

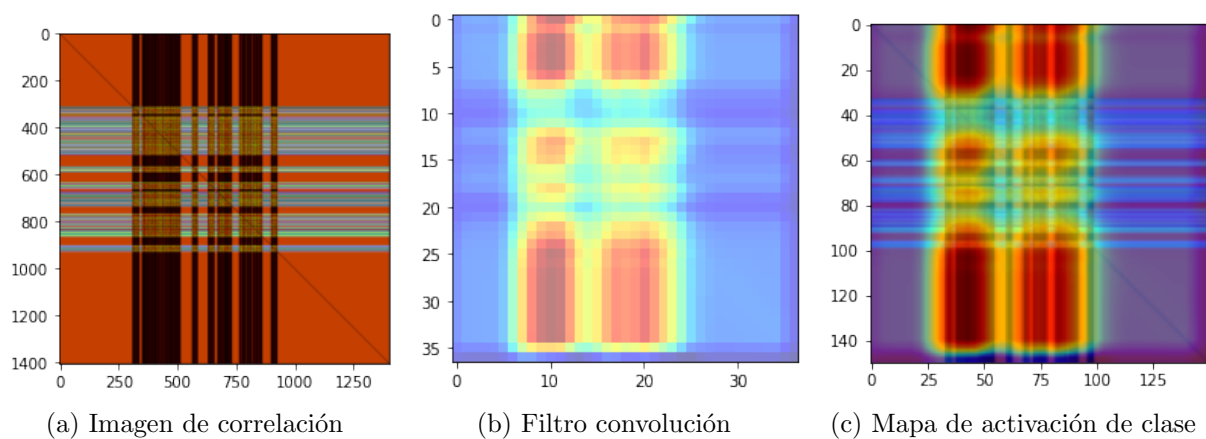


Figura 10.6: Descubrimiento de patrón mediante mapa de activación de clases

Capítulo 11

Conclusiones

El presente trabajo es la suma de esfuerzos basados en el seguimiento de una metodología lógica-formal, atendiendo a la integración de las ciencias de la computación y las ciencias biológicas, lo cual refiere a un vasto cumulo de conocimientos necesarios para atender a la problemática planteada, teniendo en cuenta la búsqueda de la verdad en cuanto a la verdad.

Fue tal la investigación y desarrollo que se logró a cabalidad responder a los objetivos y cuestionamientos referentes al problema.

Todo gran proyecto implica retos que pueden mermar la calidad de su desarrollo, en este caso el desarrollo requería tener un conjunto de datos adecuado, sin embargo, no fue esto lo que marcó el inicio ya que en la búsqueda de un trabajo eficiente y con posibilidad de generalización se aplazó la búsqueda del set adecuado para las etapas intermedias de desarrollo.

Teniendo en cuenta el trabajo [35] fue posible delimitar el set adecuado para la realización de las pruebas de tal forma que se contara con una base supervisada con la cual contrastar los resultados encontrados.

A priori el algoritmo de AMS marcaba la pauta de inicio del estudio de secuencias proteicas ya que de su eficiencia dependía la calidad del primer procesamiento, fue por ello que se hecho mano de un algoritmo conocido y probado tal que la complejidad e innovación se cargara en las fases intermedias.

Una vez alineadas las secuencias se presentó el primer gran reto, la generación de matrices de correlación estadística. Dicha tarea se volvió compleja dada la necesidad de adaptar un método de variables cuantitativas a variables cualitativas. No obstante la investigación basada en referencias (ligadas al estado del arte) posibilitó la adaptación del método a las necesidades.

Posteriormente la búsqueda de la abstracción gráfica adecuada representó el segundo gran reto, pues la complejidad de estudiar la correlación de las secuencias, adaptar dichos valores a un umbral y conseguir una representación topológicamente eficaz se tornó en un problema cuya complejidad algorítmica y computacional diezma el avance pero sobre todo retaba los objetivos al pretender buscar una solución basada en tecnologías open source con un coste mínimo.

En consecuencia el desarrollo de una representación gráfica eficiente permitió el uso de un modelo convolucional básico ya que la calidad del dataset era óptima tal que prácticamente cualquier clasificador realizaría un adecuado trabajo, sin embargo, se precisó la necesidad de un modelo convolucional pues los mecanismos internos de procesamiento eran necesarios para la búsqueda de patrones.

Y fue la búsqueda de patrones bajo aprendizaje profundo lo que motivó la investigación e implementación final. Esto refiere al entendimiento de las neuronas de una red donde los mapas de activación de clases fueron el mayor logro en cuanto a implementación del proyecto se refiere, ya que sin dicho constructo hubiera resultado inútil el esfuerzo previo, pues es en esta fase donde se logra presentar y validar la existencia de patrones en las proteínas.

Sin duda, el campo de interacción entre la biología y la computación son una de las mancuernas más necesarias y eficientes en el desarrollo de la ciencia y es este proyecto una muestra palpable de la complejidad que un sistema bioinformático puede implicar, no obstante, también es muestra del potencial que existe.

En suma se ha presentado una memoria de los resultados y avances obtenidos donde se llevó a término el total de los objetivos perseguidos logrando dilucidar una nueva perspectiva en cuanto a la representación y entendimiento de la composición de una proteína.

11.1. Trabajos futuros

En este punto se marca el fin de una etapa en la búsqueda de motivos conservados, pues aun quedan aspectos a entender pero sobre todo elementos a mejorar. Esto ya que a pesar de los resultados obtenidos queda como una iteración posterior la representación exacta de los elementos (aminoácidos), pues en el resultado final se obtuvo una representación que modela de forma gráfica y no textual la secuencia conservada.

Por otro lado en el desarrollo se vio la necesidad de adaptar los algoritmos a nuevos paradigmas que aceleren el cómputo y optimicen los recursos, pues aunque se emplearon paradigmas de cómputo paralelo de forma parcial no se utilizó en todos los escenarios posibles, además de que para una futura iteración se concibe el reemplazo de unidades de procesamiento gráfico (GPU) por unidades de procesamiento de tensores (TPU).

Una vez que hayan sido resultas las potencialidades mencionadas se concibe el generar un modelo basado en gramáticas libres de contexto donde la información obtenida (la cual esta en forma gráfica) sea simplificada como un conjunto de reglas que permitan el modelado de proteínas, donde una gramática particular sea capaz de generar todas las posibles proteínas pertenecientes a una familia dada.

Referencias

- [1] M. Chicurel, “Bioinformatics: Bringing it all together technology feature,” en, *Nature*, vol. 419, n.º 6908, págs. 752-755, oct. de 2002, ISSN: 1476-4687. DOI: 10.1038/419751a. dirección: <https://www.nature.com/articles/419751a> (visitado 14-06-2021).
- [2] *Proteína* — NHGRI, es. dirección: <https://www.genome.gov/es/genetics-glossary/Proteina> (visitado 01-03-2021).
- [3] J. Corum y C. Zimmer, “Bad News Wrapped in Protein: Inside the Coronavirus Genome,” en-US, *The New York Times*, abr. de 2020, ISSN: 0362-4331. dirección: <https://www.nytimes.com/interactive/2020/04/03/science/coronavirus-genome-bad-news-wrapped-in-protein.html> (visitado 01-03-2021).
- [4] J. Jiménez, *Moderna solo necesitó dos días para diseñar su vacuna contra el COVID: así es como la nueva tecnología de ARNm ya ha cambiado la fabricación de vacunas*, es, nov. de 2020. dirección: <https://www.xataka.com/medicina-y-salud/moderna-solo-necesito-dos-dias-para-disenar-su-vacuna-covid-asi-como-nueva-tecnologia-arnm-ha-cambiado-fabricacion-vacunas> (visitado 14-06-2021).
- [5] —, *DeepMind acaba de dar un salto de gigante para resolver uno de los grandes misterios de la biología molecular y abre la puerta a una nueva revolución biomédica*, es, dic. de 2020. dirección: <https://www.xataka.com/medicina-y-salud/deepmind-acaba-dar-salto-gigante-para-resolver-uno-grandes-misterios-biologia-molecular-50-anos-buscando-entender-plegamiento-proteinas> (visitado 14-06-2021).
- [6] *Aminoácido*, es. dirección: <https://www.genome.gov/es/genetics-glossary/Aminoacido> (visitado 01-03-2021).
- [7] *Protein Folding: The Good, the Bad, and the Ugly*, en-US, mar. de 2010. dirección: <https://sitn.hms.harvard.edu/flash/2010/issue65/> (visitado 01-03-2021).

- [8] P. Forterre, “The origin of viruses and their possible roles in major evolutionary transitions,” en, *Virus Research*, vol. 117, n.º 1, págs. 5-16, abr. de 2006, ISSN: 01681702. DOI: 10.1016/j.virusres.2006.01.010. dirección: <https://linkinghub.elsevier.com/retrieve/pii/S0168170206000293> (visitado 01-03-2021).
- [9] *Ácido nucleico* — NHGRI, es. dirección: <https://www.genome.gov/es/genetics-glossary/acido-nucleico> (visitado 01-03-2021).
- [10] D. L. Caspar y A. Klug, “Physical principles in the construction of regular viruses,” eng, *Cold Spring Harbor Symposia on Quantitative Biology*, vol. 27, págs. 1-24, 1962, ISSN: 0091-7451. DOI: 10.1101/sqb.1962.027.001.005.
- [11] F. H. C. Crick y J. D. Watson, “Structure of Small Viruses,” en, *Nature*, vol. 177, n.º 4506, págs. 473-475, mar. de 1956, ISSN: 1476-4687. DOI: 10.1038/177473a0. dirección: <https://www.nature.com/articles/177473a0> (visitado 05-06-2021).
- [12] D. Baltimore, “Expression of animal virus genomes.,” *Bacteriological Reviews*, vol. 35, n.º 3, págs. 235-241, sep. de 1971, ISSN: 0005-3678. dirección: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC378387/> (visitado 01-03-2021).
- [13] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning*, ép. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016, ISBN: 9780262035613.
- [14] A. Wilson, *A Brief Introduction to Supervised Learning*, en, oct. de 2019. dirección: <https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590> (visitado 01-03-2021).
- [15] A. Pant, *Introduction to Logistic Regression*, en, ene. de 2019. dirección: <https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148> (visitado 01-03-2021).
- [16] M. Peixeiro, *The Complete Guide to Support Vector Machine (SVM)*, en, mayo de 2020. dirección: <https://towardsdatascience.com/the-complete-guide-to-support-vector-machine-svm-f1a820d8af0b> (visitado 01-03-2021).
- [17] P. Gupta, *Decision Trees in Machine Learning*, en, nov. de 2017. dirección: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052> (visitado 01-03-2021).

- [18] D. Subramanian, *A Simple Introduction to K-Nearest Neighbors Algorithm*, en, ene. de 2020. dirección: <https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e> (visitado 01-03-2021).
- [19] S. Mishra, *Unsupervised Learning and Data Clustering*, en, mayo de 2017. dirección: <https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a> (visitado 01-03-2021).
- [20] I. Dabbura, *K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks*, en, ago. de 2020. dirección: <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a> (visitado 01-03-2021).
- [21] C. R. Patlolla, *Understanding the concept of Hierarchical clustering Technique*, en, mayo de 2020. dirección: <https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec> (visitado 01-03-2021).
- [22] S. Bhatt, *Reinforcement Learning 101*, en, abr. de 2019. dirección: <https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292> (visitado 01-03-2021).
- [23] P. Skalski, *Deep Dive into Math Behind Deep Networks*, en, feb. de 2020. dirección: <https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba> (visitado 01-03-2021).
- [24] M. S. Researcher PhD, *Simple Introduction to Neural Networks*, en, jul. de 2020. dirección: <https://towardsdatascience.com/simple-introduction-to-neural-networks-ac1d7c3d7a2c> (visitado 01-03-2021).
- [25] H. Mahmood, *Gradient Descent*, en, ene. de 2019. dirección: <https://towardsdatascience.com/gradient-descent-3a7db7520711> (visitado 01-03-2021).
- [26] S. Kostadinov, *Understanding Backpropagation Algorithm*, en, ago. de 2019. dirección: <https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd> (visitado 01-03-2021).
- [27] D. T. Jones y S. M. Kandathil, "High precision in protein contact prediction using fully convolutional neural networks and minimal sequence features," en, *Bioinformatics*, vol. 34, n.º 19, A. Valencia, ed., págs. 3308-3315, oct. de 2018, ISSN: 1367-4803, 1460-2059. DOI: 10.

- 1093/bioinformatics/bty341. dirección: <https://academic.oup.com/bioinformatics/article/34/19/3308/4987145> (visitado 01-03-2021).
- [28] A. Mittal, *Understanding RNN and LSTM*, en, oct. de 2019. dirección: <https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e> (visitado 01-03-2021).
- [29] *¡La Revolución de la INTELIGENCIA ARTIFICIAL en BIOMEDICINA! (AlphaFold2) — Feat. @La Hiperactina*, es-419. dirección: <https://www.youtube.com/watch?v=Uz7ucmqjZ08> (visitado 01-03-2021).
- [30] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Židek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu y D. Hassabis, “Improved protein structure prediction using potentials from deep learning,” en, *Nature*, vol. 577, n.º 7792, págs. 706-710, ene. de 2020, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1923-7. dirección: <https://www.nature.com/articles/s41586-019-1923-7> (visitado 01-03-2021).
- [31] M. M. Gromiha y S. Selvaraj, “Inter-residue interactions in protein folding and stability,” *Progress in biophysics and molecular biology*, vol. 86, n.º 2, págs. 235-277, 2004.
- [32] N. Meinshausen, P. Bühlmann y col., “High-dimensional graphs and variable selection with the lasso,” *Annals of statistics*, vol. 34, n.º 3, págs. 1436-1462, 2006.
- [33] N. Halabi, O. Rivoire, S. Leibler y R. Ranganathan, “Protein sectors: evolutionary units of three-dimensional structure,” *Cell*, vol. 138, n.º 4, págs. 774-786, 2009.
- [34] A. d. l. Escalera Hueso, “Visión por computador: Fundamentos y métodos,” 2001.
- [35] J. Reyes, “ALGORITMO EFICIENTE PARA LA AGRUPACIÓN DE PROTEÍNAS EN FAMILIAS BASADO EN MEJORES ACIERTOS BIDIRECCIONALES Y EL ÁRBOL FILOGENÉTICO,” es, Tesis doct., Benemerita Universidad Autonoma de Aguascalientes, Aguascalientes, jun. de 2019.
- [36] R. Eduardo, *Introducción a la bioinformática*, publisher: CINVESTAV. dirección: <https://www.tamps.cinvestav.mx/~ertello/bioinfo.php>.
- [37] V. Mallawaarachchi, *Multiple Sequence Alignment using Clustal Omega and T-Coffee*, en, ene. de 2018. dirección: <https://towardsdatascience.com/multiple-sequence-alignment-using-clustal-omega-and-t-coffee-3cc662b1ea82> (visitado 01-03-2021).

- [38] *CODIGO AMINOÁCIDoS*. dirección: http://bioinformatica.uab.es/genetica/pr7/codi%20aa_sp.htm (visitado 01-03-2021).
- [39] *TensorFlow y Redes Neuronales*. dirección: <https://relopezbriega.github.io/blog/2016/06/05/tensorflow-y-redes-neuronales/> (visitado 01-03-2021).
- [40] K. Pykes, *Tensors and Arrays*, en, ene. de 2021. dirección: <https://towardsdatascience.com/tensors-and-arrays-2611d48676d5> (visitado 01-03-2021).
- [41] A. Bonner, *The Complete Beginners Guide to Deep Learning*, en, mayo de 2019. dirección: <https://towardsdatascience.com/intro-to-deep-learning-c025efd92535> (visitado 01-03-2021).
- [42] J. Xiong, *Essential bioinformatics*. New York: Cambridge University Press, 2006, OCLC: ocm62078278, ISBN: 9780521840989 9780521600828.
- [43] N. C. Jones y P. Pevzner, *An introduction to bioinformatics algorithms*, ép. Computational molecular biology. Cambridge, MA: MIT Press, 2004, ISBN: 9780262101066.
- [44] Y. Zhang, *New Advances in Machine Learning*, en. BoD – Books on Demand, feb. de 2010, Google-Books-ID: XAqhDwAAQBAJ, ISBN: 9789533070346.
- [45] D. T. Jones, D. W. A. Buchan, D. Cozzetto y M. Pontil, “PSICOV: precise structural contact prediction using sparse inverse covariance estimation on large multiple sequence alignments,” *Bioinformatics*, vol. 28, n.º 2, págs. 184-190, ene. de 2012, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btr638. dirección: <https://doi.org/10.1093/bioinformatics/btr638> (visitado 01-03-2021).
- [46] J. Yin y J. Redovich, “Kinetic Modeling of Virus Growth in Cells,” en, *Microbiology and Molecular Biology Reviews*, vol. 82, n.º 2, e00066-17, /mmbr/82/2/e00066-17.atom, mar. de 2018, ISSN: 1092-2172, 1098-5557. DOI: 10.1128/MMBR.00066-17. dirección: <https://mmbr.asm.org/content/82/2/e00066-17> (visitado 01-03-2021).
- [47] *Severe Acute Respiratory Syndrome Coronavirus 2 Specific Antibody Responses in Coronavirus Disease Patients*. dirección: <https://stacks.cdc.gov/view/cdc/90207> (visitado 01-03-2021).
- [48] J. D. Watson, ed., *Molecular biology of the gene*, Seventh edition. Boston: Pearson, 2014, ISBN: 9780321762436 9780321905376 9780321902641.

- [49] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*, en, dic. de 2018. dirección: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> (visitado 01-03-2021).
- [50] J. Rocca, *Understanding Generative Adversarial Networks (GANs)*, en, jul. de 2020. dirección: <https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29> (visitado 01-03-2021).
- [51] V. Roman, *Supervised Learning: Basics of Classification and Main Algorithms*, en, mar. de 2019. dirección: <https://towardsdatascience.com/supervised-learning-basics-of-classification-and-main-algorithms-c16b06806cd3> (visitado 01-03-2021).
- [52] *Deep Reinforcement Learning: Pong from Pixels*. dirección: <http://karpathy.github.io/2016/05/31/r1/> (visitado 01-03-2021).
- [53] *Martin Weigt: "From generative models of protein sequences to evolution-guided protein design"*, es-419. dirección: <https://www.youtube.com/watch?v=M31bT01EOcI> (visitado 01-03-2021).

Parte I

Anexos

Capítulo 12

Lista de organismos estudiados

Índice	Organismo
1	<i>fusarium graminearum</i> PH-1
2	<i>nectria haematococca</i> mpVI77-13-4
3	<i>fusarium oxysporum</i> 4287
4	<i>fusarium verticillioides</i> 7600
5	<i>verticillium alfalfae</i> VaMs-102
6	<i>verticillium dahliae</i> VdLs-17
7	<i>trichoderma atroviride</i> IMI206040
8	<i>trichoderma virens</i> Gv29-8
9	<i>trichoderma reesei</i> QM6a
10	<i>chaetomium globosum</i> CBS148-51
11	<i>podospora anserina</i> Smat+
12	<i>neurospora crassa</i> OR74A
13	<i>magnaporthe oryzae</i> 70-15
14	<i>botrytis cinerea</i> B05-10
15	<i>sclerotinia sclerotiorum</i> 1980
16	<i>bipolaris maydis</i> C5
17	<i>pyrenophora tritici-repentis</i> Pt-1C-BFP
18	<i>parastagonospora nodorum</i> SN15
19	<i>pseudocercospora fijiensis</i> CIRAD86
20	<i>zymoseptoria tritici</i> IPO323
21	<i>coccidioides immitis</i> H538-4
22	<i>coccidioides immitis</i> RMSCC2394
23	<i>coccidioides immitis</i> RMSCC3703
24	<i>coccidioides immitis</i> RS
25	<i>coccidioides posadasii</i> RMSCC3488
26	<i>coccidioides posadasii</i> Silveira
27	<i>uncinocarpus reesii</i> 1704
28	<i>paracoccidioides brasiliensis</i> Pb03
29	<i>paracoccidioides brasiliensis</i> Pb18
30	<i>paracoccidioides lutzii</i> Pb01
31	<i>histoplasma capsulatum</i> NAm1
32	<i>aspergillus flavus</i> NRRL3357
33	<i>aspergillus oryzae</i> RIB40

Capítulo 13

Código parcial del método propuesto

13.1. Bloques ipython

13.1.1. Inferencia de sitios acoplados usando covarianza para la generación de imágenes de correlación

Listing 13.1: Librerías

```
from Bio import SeqIO
from Bio import Entrez
from PIL import Image as im
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import fastparser
import torch as tr
import math
import sys
import os
```

Listing 13.2: Lista de proteínas del clique

```
row_list = []
list_seq = pd.read_csv(dataPath + name_list + "/" + name_list + ".csv")
```

```

for rowIndex in range(0,int(name_list)):
    row_num = rowIndex
    row_seq = list_seq.iloc[row_num]
    row_list_aux = row_seq.to_list()
    row_list.append(row_list_aux)
print(row_list)

```

Listing 13.3: Descarga de secuencias de aminoácidos por proteína

```

count = 0
for row in row_list:
    seq = ""
    print(str(count))
    for protein_id in row:
        if(not(pd.isna(protein_id))):
            print(protein_id)
            Entrez.email = "raul@mail.com"
            handle = Entrez.efetch(db="protein", id= protein_id, rettype="gb", retmode="
                text")
            record = SeqIO.read(handle, "genbank")
            handle.close()
            seq += str(">" + record.id + "_" + record.description) + "\n" + record.seq + "
                \n"
    f = open(dataPath + name_list + "/hard_seq/" + name_list + "-" + str(count) + ".
        fasta", "w")
    f.write(str(seq))
    f.close()
    count += 1

```

Listing 13.4: MSA mediante Clustal Omega

```

% %bash
aux=$(ls /content/drive/MyDrive/ICI/tesina/src/data/064/hard_seq | wc -l)
chmod 777 /content/drive/MyDrive/ICI/tesina/src/clustal/clustalo-1.2.4-Ubuntu-

```

```

x86_64
for (( ind=0; ind<$aux; ind++ ))
do
  cd /content/drive/MyDrive/ICI/tesina/src/clustal
  ./clustalo-1.2.4-Ubuntu-x86_64 -i /content/drive/MyDrive/ICI/tesina/src/data
    /064/hard_seq/064-$ind.fasta -o /content/drive/MyDrive/ICI/tesina/src/data
    /064/alig_seq/064-$ind-al.fasta --auto -v
done

```

Listing 13.5: Lista de aminoacidos

```

aminos = pd.read_json("/content/drive/MyDrive/ICI/tesina/src/aminoacid.json")
aminos = aminos['Codigo1'].sort_values()

```

Listing 13.6: Estructura de directorios

```

% %bash
listAlig=$(ls /content/drive/MyDrive/ICI/tesina/src/data/064/alig_seq | wc -l)
echo $listAlig
for (( ind=0; ind<$listAlig; ind++ ))
do
  mkdir -p /content/drive/MyDrive/ICI/tesina/src/data/064/correlacion/064-$ind
  mkdir -p /content/drive/MyDrive/ICI/tesina/src/data/064/img_seq/img_rgb/064-$ind
done

```

Listing 13.7: Transformación de secuencias alineadas a representación gráfica

```

import os
import shutil

for fam_seq_index in range(31,int(name_list)):

  # Carga de secuencias alineadas
  with open(dataPath + name_list + "/alig_seq/" + name_list + "-" + str(

```

```

    fam.seq_index) + "-al.fasta") as fasta_file:
    auxtf = []
    parser = fastaparser.Reader(fasta_file)
    for seq in parser:
        aux = seq.sequence_as_string()
        aux = list(aux)
        auxtf.append(aux)
    seq_df = pd.DataFrame(auxtf)

# Frecuencias marginales observadas en aminoacidos  $f(A_i)$  y  $f(B_j)$ 
    aux = []
    for col in seq_df:
        freq = pd.Series(seq_df[col].value_counts())
        aux.append(freq)
    freq_ind = pd.DataFrame(aux).fillna(0).astype(int)
    #print(freq_ind)

# Frecuencia de los pares de aminoacidos  $f(A_iB_j)$ 
    from numpy import inf
    max_vec = []
    min_vec = []
    for i, row in seq_df.iterrows():
        matrix_cov = np.zeros((len(row), len(row)))
        for amino in range(0, len(row)-1):
            pair = [row[amino], row[amino+1]]
            pair = tuple(pair)
            freq_pair = 0
            for col in range(0, len(seq_df.columns)-1):
                aux_cols = zip(seq_df[col], seq_df[col+1])
                aux_cols = tuple(aux_cols)
                for j in range(0, len(aux_cols)):

```


[illegible]

```

#####_#_Generacin_de_imagenes_a_partir_de_matriz_de_correlacin
#####_for_i_in_range(0,dirCorrLen-1):
#####_correlacion_=pd.read_csv(dataPath+_name_list+_'/correlacion/064-' +str(
    fam_seq_index)+'/corr064-' +str(fam_seq_index)+'-' +str(i)+''.csv')
#####_print(correlacion)
#####_correlacion_=correlacion.to_numpy()
#####_correlacion_=np.nan_to_num(correlacion)
#####_matrix_rgb_=np.zeros((len(correlacion),len(correlacion),3),dtype=np.uint8)
#####_intervalo_=abs((1e-24)-0.0)/(255*255*255))
#####_for_iAuxColor_in_range(0,len(correlacion)):
#####_for_jAuxColor_in_range(0,len(correlacion)):
#####_tmp_correlacion_=correlacion[iAuxColor][jAuxColor]
#####_correlacion[iAuxColor][jAuxColor]_=int(round(tmp_correlacion_/intervalo_)
    )
#####_for_xCorr_in_range(0,len(correlacion)):
#####_for_yCorr_in_range(0,len(correlacion)):
#####_rgb_=int(correlacion[xCorr][yCorr])
#####_matrix_rgb[xCorr,yCorr,0]_=rgb_&255_#blue
#####_matrix_rgb[xCorr,yCorr,1]_= (rgb_>>8)&255_#green
#####_matrix_rgb[xCorr,yCorr,2]_= (rgb_>>16)&255_#red
#####_img_=im.fromarray(matrix_rgb)
#####_img.save(dataPath+_name_list+_'/img_seq/img_rgb/064-' +str(fam_seq_index)_
    +'/img064-' +str(fam_seq_index)+'-' +str(i)+''.png')

#####_print("Familia "+str(fam_seq_index)+" terminada")

#####_shutil.rmtree(dataPath+_name_list+_'/correlacion/064-' +str(fam_seq_index))
#####_

```

13.1.2. Red Neuronal Convolutacional

Listing 13.8: Tubería de datos

```
% %bash

array=(uno dos tres cuatro cinco seis siete ocho nueve diez once doce trece catorce
      quince dieciseis diecsiete diecisocho diecinueve veinte veintiuno veintidos ventitres
      veinticuatro veinticinco veintiseis veintisiete veintiocho veintinueve treinta
      teintauno treintados)

for (( ind=0; ind<32; ind++ ))
do
    mv /content/drive/MyDrive/ICI/tesina/src/seq/064/datosV0.3.1/img_rgb_train
      /064-$ind /content/drive/MyDrive/ICI/tesina/src/seq/064/datosV0.3.1/
      img_rgb_train/${array[$ind]}
    echo ${array[$ind]}
done
```

Listing 13.9: Tubería de datos

```
import os
import torch
import torchvision
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split

%matplotlib inline
```

Listing 13.10: Exploración del dataset

```
# Nombre del proyecto
```

```

project_name = 'cnn-protein'

data_dir = "/content/drive/MyDrive/ICI/tesina/src/data/064/img_seq/img_rgb_train"
test_data_dir = "/content/drive/MyDrive/ICI/tesina/src/data/064/img_seq/
    img_rgb_test"

uno_files = os.listdir(data_dir + '/uno')
print(f'Numero_de_elemntos_en_categoria_uno:_{len(uno_files)}')
print(uno_files[:5])

# Homogenizacin de imagenes mediante redimensin a 150 * 150 pixeles
dataset = ImageFolder(data_dir,transform = transforms.Compose([
    transforms.Resize((150,150)),transforms.ToTensor()
]))
test_dataset = ImageFolder(test_data_dir,transforms.Compose([
    transforms.Resize((150,150)),transforms.ToTensor()
]))

img, label = dataset[6]
print(img.shape,label)

print(f'Imagenes_de_entrenamiento:_{len(dataset)}')
print(f'Imagenes_de_prueba:_{len(test_dataset)}')

print("La_clases_son:\n",dataset.classes)

def display_img(img,label):
    print(f'Etiqueta:_{dataset.classes[label]}')
    plt.imshow(img.permute(1,2,0))

# Comprobacin de la carga de imagenes mediante su visualizacin

```

```
display_img(*dataset[0])
```

Listing 13.11: Conjuntos de datos de entrenamiento y validación

```
random_seed = 2021
torch.manual_seed(random_seed)

val_size = 200
train_size = len(dataset) - val_size
train_data, val_data = random_split(dataset, [train_size, val_size])
print(f'Length of Train Data: {len(train_data)}')
print(f'Length of Validation Data: {len(val_data)}')
```

Listing 13.12: Segmentación de los datos en lotes

```
batch_size = 128

train_dl = DataLoader(train_data, batch_size, shuffle = True, num_workers = 4,
    pin_memory = True)
val_dl = DataLoader(val_data, batch_size*2, num_workers = 4, pin_memory = True)

from torchvision.utils import make_grid
def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize = (16, 12))
        ax.set_xticks([])
        ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break

# Reticula del lote
show_batch(train_dl)
```

Listing 13.13: Modelo base para la clasificación de imágenes

```

class ImageClassificationBase(nn.Module):

    def training_step(self, batch):
        images, labels = batch
        out = self(images) # Genera predicciones
        loss = F.cross_entropy(out, labels) # Calculo de perdida
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images) # Genera predicciones
        loss = F.cross_entropy(out, labels) # Calculo de perdida
        acc = accuracy(out, labels) # Calculo de precision
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean() # Combinacin de pedidas
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean() # Combinacin de precisiones
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch-[{}],_train_loss:_{:}.4f},_val_loss:_{:}.4f},_val_acc:_{:}.4f}" .format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))

def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

```

Listing 13.14: Modelo de clasificación

```
class ModelClassification(ImageClassificationBase):

def __init__(self):

    super().__init__()
    self.network = nn.Sequential(
        nn.Conv2d(3, 32, kernel_size = 3, padding = 1),
        nn.ReLU(),
        nn.Conv2d(32,64, kernel_size = 3, stride = 1, padding = 1),
        nn.ReLU(),
        nn.MaxPool2d(2,2),

        nn.Conv2d(64, 128, kernel_size = 3, stride = 1, padding = 1),
        nn.ReLU(),
        nn.Conv2d(128 ,128, kernel_size = 3, stride = 1, padding = 1),
        nn.ReLU(),
        nn.MaxPool2d(2,2),

        nn.Conv2d(128, 256, kernel_size = 3, stride = 1, padding = 1),
        nn.ReLU(),
        nn.Conv2d(256,256, kernel_size = 3, stride = 1, padding = 1),
        nn.ReLU(),
        nn.MaxPool2d(2,2),

        nn.Flatten(),
        nn.Linear(82944,1024),
        nn.ReLU(),
        nn.Linear(1024, 512),
        nn.ReLU(),
        nn.Linear(512,256),
        nn.ReLU(),
```

```

        nn.Linear(256,32)
    )

    def forward(self, xb):
        return self.network(xb)

model = ModelClassification()

model

for images, labels in train_dl:
    print('images.shape:', images.shape)
    out = model(images)
    print('out.shape:', out.shape)
    print('out[0]:', out[0])
    break

```

Listing 13.15: Configuración del dispositivo de procesamiento

```

def get_default_device():
    """ Configuración de GPU o CPU """
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    """Traslado de datos al dispositivo de procesamiento"""
    if isinstance(data, (list, tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking = True)

```



```

class DeviceDataLoader():
    """ Envoltura para el traslado de los datos al dispositivo de procesamiento """

    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """ Generacin de lote de datos despues del traslado """
        for b in self.dl:
            yield to_device(b,self.device)

    def __len__(self):
        """ Nmero de lotes """
        return len(self.dl)

device = get_default_device()
device

# Carga de datos en la GPU
train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)
to_device(model, device)

```

Listing 13.16: Ajuste del modelo y aprendizaje

```

@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

```

```

def fit(epochs, lr, model, train_loader, val_loader, opt_func = torch.optim.SGD):

    history = []
    optimizer = opt_func(model.parameters(),lr)
    for epoch in range(epochs):

        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)

    return history

#Carga del modelo en la GPU
model = to_device(ModelClassification(),device)

#Evaluacin inicial del modelo
evaluate(model,val_dl)

#Configuracin del No. de epocas, funcin de optimizacin y tasa de aprendizaje
num_epochs = 30
opt_func = torch.optim.Adam

```

```
lr = 0.001
```

```
#Ajuste del modelo a los datos de entrenamiento y registrar el resultado despues de cada  
poca
```

```
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

Listing 13.17: Graficos de precisión y pérdidas del modelo

```
def plot_accuracies(history):  
    """ Grafico de precisin por epoca """  
    accuracies = [x['val_acc'] for x in history]  
    plt.plot(accuracies, '-x')  
    plt.xlabel('Epoca')  
    plt.ylabel('Precisin')  
    plt.title('Precisin_respecto_a_la_epoca');  
  
plot_accuracies(history)  
  
def plot_losses(history):  
    """ Grafico de perdidas por epoca """  
    train_losses = [x.get('train_loss') for x in history]  
    val_losses = [x['val_loss'] for x in history]  
    plt.plot(train_losses, '-bx')  
    plt.plot(val_losses, '-rx')  
    plt.xlabel('Epoca')  
    plt.ylabel('Perdida')  
    plt.legend(['Entrenamiento', 'Validacin'])  
    plt.title('Perdida_respecto_a_la_epoca');  
  
plot_losses(history)
```

Listing 13.18: Evaluación mediante datos de prueba

```

# Aplicacin del modelo al conjunto de datos de prueba y obtencin los resultados
test_loader = DeviceDataLoader(DataLoader(test_dataset, batch_size*2), device)
result = evaluate(model, test_loader)
result

# Guardado del modelo
torch.save(model.state_dict(), '/content/drive/MyDrive/ICI/tesina/src/data/cnn-
    protein-dict.pth')
torch.save(model, '/content/drive/MyDrive/ICI/tesina/src/data/cnn-protein.pth')

```

Listing 13.19: Evaluación sobre imágenes

```

def predict_img_class(img,model):
    """ Prediccin de la clase de la imagen """
    img = to_device(img.unsqueeze(0), device)
    prediction = model(img)
    _, preds = torch.max(prediction, dim = 1)
    return dataset.classes[preds[0].item()]

from PIL import Image
# Carga de imagen
img = Image.open("/content/drive/MyDrive/ICI/tesina/src/data/064/img_seq/
    img_rgb_test/064-13/img064-13-40.png")
# Redimensionado de imagen
newsize = (150, 150)
img = img.resize(newsize)

# Conversin de imagen a tensor
img = transforms.ToTensor()(img)

# Impresin de imagen
plt.imshow(img.permute(1,2,0))

```

```
# Prediccin de categoria de la imagen
print(f'Predicted_Class:_{predict_img_class(img,model)}')
```

13.1.3. Visualización de características

Listing 13.20: Filtros convolucionales

```
import torch.optim as optim
from torchvision import models
import torchvision.datasets as dataset
import cv2 as cv

# Carga de imagen
img = Image.open("/content/drive/MyDrive/ICI/tesina/src/data/064/img_seq/
    img_rgb_test/064-17/img064-17-55.png")

transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomResizedCrop(150),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
img=np.array(img)
img=transform(img)
img=img.unsqueeze(0)
print(img.size())

no_of_layers=0
conv_layers=[]

model_children=list(model.children())
```

```

for child in model_children:
    if type(child)==nn.Conv2d:
        no_of_layers+=1
        conv_layers.append(child)
    elif type(child)==nn.Sequential:
        for layer in child.children():
            if type(layer)==nn.Conv2d:
                no_of_layers+=1
                conv_layers.append(layer)
print(no_of_layers)

img = img.cuda()

results = [conv_layers[0](img)]
for i in range(1, len(conv_layers)):
    results.append(conv_layers[i](results[-1]))
outputs = results

# visualize 8 features map from each layer
for num_layer in range(len(outputs)):
    plt.figure(figsize=(50, 10))
    layer_viz = outputs[num_layer][0, :, :, :]
    layer_viz = layer_viz.data
    print(len(layer_viz))
    print("Layer_", num_layer+1)
    for i, filter in enumerate(layer_viz):
        filter = filter.cpu()
        if i == 16:
            break
    plt.subplot(2, 8, i + 1)

```

```
plt.imshow(filter, cmap='gray')
plt.axis("off")
plt.show()
plt.close()
```

Mapas de características

Listing 13.21: Estructura de directorios

```
% %bash
array=(cero uno dos tres cuatro cinco seis siete ocho nueve diez once doce trece
catorce quince dieciseis diecisiete dieciocho diecinueve veinte veintiuno veintidos
ventitres veinticuatro veinticinco veintiseis veintisiete veintiocho veintinueve
treinta treintauno)
for (( ind=0; ind<32; ind++ ))
do
    mkdir /content/drive/MyDrive/ICI/tesina/src/data/064/cam_seq/fam_064-$ind
    echo /content/drive/MyDrive/ICI/tesina/src/data/064/cam_seq/fam_064-$ind
done
```

Listing 13.22: Librerías

```
%matplotlib inline

import numpy as np
import skimage.transform
import torch
import torch.nn as nn
import torch.nn. functional as F

from PIL import Image

from matplotlib.pyplot import imshow
```

```

from torchvision.utils import save_image
from torchvision import models, transforms

```

Listing 13.23: Búsqueda de mapas de activación de clases

```

class SaveValues():
    def __init__(self, m):
        # register a hook to save values of activations and gradients
        self.activations = None
        self.gradients = None
        self.forward_hook = m.register_forward_hook(self.hook_fn_act)
        self.backward_hook = m.register_backward_hook(self.hook_fn_grad)

    def hook_fn_act(self, module, input, output):
        self.activations = output

    def hook_fn_grad(self, module, grad_input, grad_output):
        self.gradients = grad_output[0]

    def remove(self):
        self.forward_hook.remove()
        self.backward_hook.remove()

class CAM(object):
    """ Class Activation Mapping """

    def __init__(self, model, target_layer):
        """
        Args:
            model: a base model to get CAM which have global pooling and fully
                  connected layer.

```



```

        target_layer: conv_layer before Global Average Pooling
        """

self.model = model
self.target_layer = target_layer

# save values of activation maps and gradients in target_layer
self.values = SaveValues(self.target_layer)

def forward(self, x):
    """
    Args:
        x: input image. shape => (1, 3, H, W)
    Return:
        heatmap: class activation mappings of the predicted class
    """

    # object classification
    score = self.model(x)
    prob = F.softmax(score, dim=1)
    max_prob, idx = torch.max(prob, dim=1)
    print(
        "predicted_object_ids_{ }\tprobability_{ }".format(idx.item(), max_prob.
            item()))

    # cam can be calculated from the weights of linear layer and activations
    weight_fc = list(
        self.model.network[22].parameters())[0].to('cpu').data
    cam = self.getCAM(self.values, weight_fc, idx.item())

    return cam

```

```

def __call__(self, x):
    return self.forward(x)

def getCAM(self, values, weight_fc, idx):
    '''
    values: the activations and gradients of target_layer
        activations: feature map before GAP. shape => (1, C, H, W)
    weight_fc: the weight of fully connected layer. shape => (num_classes, C)
    idx: predicted class id
    cam: class activation map. shape => (1, num_classes, H, W)
    '''

    print(weight_fc.shape)
    weight_fc = weight_fc.cuda()
    cam = F.conv2d(values.activations, weight=weight_fc[:, :, None, None])
    _, _, h, w = cam.shape

    # class activation mapping only for the predicted class
    # cam is normalized with min-max.
    cam = cam[:, idx, :, :]
    cam -= torch.min(cam)
    cam /= torch.max(cam)
    cam = cam.view(1, 1, h, w)

    return cam.data

```

Listing 13.24: Visualización de mapas de activación de clases

```

def reverse_normalize(x, mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]):
    x[:, 0, :, :] = x[:, 0, :, :] * std[0] + mean[0]
    x[:, 1, :, :] = x[:, 1, :, :] * std[1] + mean[1]
    x[:, 2, :, :] = x[:, 2, :, :] * std[2] + mean[2]

```

```

return x

def visualize(img, cam):
    """
    Synthesize an image with CAM to make a result image.
    Args:
        img: (Tensor) shape => (1, 3, H, W)
        cam: (Tensor) shape => (1, 1, H', W')
    Return:
        synthesized image (Tensor): shape => (1, 3, H, W)
    """

    _, _, H, W = img.shape
    cam = F.interpolate(cam, size=(H, W), mode='bilinear', align_corners=False)
    cam = 255 * cam.squeeze()
    heatmap = cv2.applyColorMap(np.uint8(cam), cv2.COLORMAP_JET)
    heatmap = torch.from_numpy(heatmap.transpose(2, 0, 1))
    heatmap = heatmap.float() / 255
    b, g, r = heatmap.split(1)
    heatmap = torch.cat([r, g, b])

    result = heatmap + img.cpu()
    result = result.div(result.max())

    return result

```

Listing 13.25: CAM Bloque 1

```

famImage = '064-2'
nameImage = '064-2-55.png'
image = Image.open('/content/drive/MyDrive/ICI/tesina/src/data/064/img_seq/

```

```

img_rgb+'/'+famImage+'/'+img'+ nameImage)
imshow(image)

# preprocessing. mean and std from ImageNet
normalize = transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
)
preprocess = transforms.Compose([
    transforms.Resize((150, 150)),
    transforms.ToTensor(),
    normalize
])

# convert image to tensor
tensor = preprocess(image)

# reshape 4D tensor (N, C, H, W)
tensor = tensor.unsqueeze(0)
tensor.shape

```

Listing 13.26: CAM Bloque 2

```

# Capa a visualizar (Debe ser conv2d)
target_layer = model.network[12]

# Envoltura
wrapped_model = CAM(model, target_layer)

# Transferencia de tensor de cpu a gpu
tensor = tensor.cuda()

```

```
# Funcin de busqueda de filtro de activacin
cam = wrapped_model(tensor)
cam = cam.cpu()

# Visualizacin de filtro de activacin
imshow(cam.squeeze().numpy(), alpha=0.5, cmap='jet')
```

Listing 13.27: CAM Bloque 3

```
# Normalizacin inversa para visualizacin
img = reverse_normalize(tensor)

# Generacin de mapa de calor
heatmap = visualize(img, cam)

# Guardado de imagen
save_image(heatmap, '/content/drive/MyDrive/ICI/tesina/src/data/064/cam_seq/fam_'
           +famImage+'/cam'+nameImage)

# Visualizacin en notebook
hm = (heatmap.squeeze().numpy().transpose(1, 2, 0) * 255).astype(np.int32)

imshow(hm)
```