UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Tesis de Licenciatura

Inmersiones de Poincaré y redes neuronales siamesas para reconstrucción de árboles filogenéticos a partir de cantos de aves

Ciro Carvallo

Director: Pablo Groisman

Codirector: Gabriel Mindlin

Fecha de Presentación: 17 de abril de 2024

Agradecimientos

A mi mamá y mi papá, por el apoyo durante todo este tiempo. Por inculcarme la importancia del estudio desde chico y todo el esfuerzo que hicieron por mí. Este logro también es suyo.

A mi familia, por todo su apoyo.

A Pablo Groisman, por aceptar ser director de esta tesis. Por el tiempo y disposición cada vez que necesité ayuda, y por transmitirme su pasión y entusiasmo por la matemática.

A mi codirector, Gabriel Mindlin, por presentarme el tema de tesis y guiarme en el desarrollo de la misma.

A Nicolás Saintier, por aceptar ser jurado de esta tesis.

A quienes hicieron y hacen posible que Argentina tenga educación pública.

Índice general

Capítulo 1

Introducción

La filogenética es una disciplina que se ocupa del estudio de las relaciones históricas entre diferentes grupos de organismos basándose en diversos caracteres tanto morfológicos como moleculares. Estos caracteres pueden reflejarse en propiedades comportamentales, emergentes de las características antes mencionadas. Caracterizar un comportamiento en términos de pocos parámetros no es una tarea sencilla. Por otro lado, una caracterización de este tipo abre perspectivas muy interesantes para nuestra comprensión de la vida, ya que el aprendizaje juega un papel en algunos comportamientos, permitidos pero no determinados por la genética.

Recientemente, diversas técnicas matemáticas han permitido atacar este tipo de preguntas complejas, cuya respuesta no se basa en la predeterminación de las variables de un problema y la elucidación de los mecanismos que las vinculan, sino en las inferencias dictadas por los datos. En el campo de las redes neuronales, uno entrena a una red (es decir, ajusta a los parámetros) a partir de una serie de ejemplos preseleccionados. La red así "aprende" a generalizar, traduciendo los ejemplos a una representación muchas veces no interpretable, pero capaz de resolver nuevas situaciones a las que no fue nunca expuesta.

El objetivo de esta tesis es desarrollar una herramienta computacional capaz de ayudarnos a reconstruir, de forma automática, un árbol a partir de un comportamiento, por ejemplo, el canto de un ave, o más general, una manifestación fenotípica expresable en términos de una imagen, en el caso del canto de un ave, el sonograma asociado a dicho canto.

No siempre las características fenotípicas permiten reconstruir la filogenia de especies, por lo tanto, a partir del desarrollo de un algoritmo, se busca estudiar la hipótesis de que para un grupo de aves seleccionadas, las cuales aprenden su canto, es posible explicar las relaciones evolutivas a partir sus vocalizaciones. Las similitudes o diferencias que se pueden encontrar entre especies están influenciadas por su cercanía dentro del árbol histórico de ancestros. De esta forma es esperable que aquellas que tengan un ancestro en común cercano compartan más características morfológicas y vocales que

aquellas que no lo tengan.

En el entendimiento de la evolución de especies de pájaros, su canto ha sido un punto importante de estudio. El canto de los pájaros varía enormemente entre diferentes especies. Un tema de estudio ha sido el motivo por el que existen estas diferencias. Una posible explicación es que sean útiles para los miembros de la propia especie en la identificación de sus pares. La importancia de los cantos a la hora de atraer a una pareja o disuadir a un rival también es distinta según la especie y puede verse reflejada en la variación de las notas entre estas. Otro factor determinante que contribuye a acrecentar las diferencias es el ambiente en el que se desarrollan.

Se estima que existen en la naturaleza aproximadamente diez mil especies de aves. La mitad de estas especies tienen su canto determinado genéticamente, es decir, que el aprendizaje no cumple ningún rol. La otra mitad requiere de la exposición a un tutor para desarrollar su canto. Para aquellas especies que lo aprenden, no resulta obvio que diferencias en esta característica tengan un paralelo con diferencias en la genética.

Estudios de mitad del siglo veinte intentan reconstruir la filogenia a partir de características vocales o de comportamiento. Por ejemplo, en 1941 Konrad Lorenz mostró que la clasificación de patos y gansos a parir de características de comportamiento era similar a la obtenida a partir de características morfológicas. Utilizando técnicas estadísticas, en 1996, Gittlerman, Wimberger y Queiroz mostraron la validez de las características de comportamiento para reconstruir filogenia. En 1993 Chappuis y Erard examinaron la relación entre especies y subespecies de aves Bulbuls del género Bleda a partir de las variaciones en características vocales [17].

El grupo de pájaros estudiados está compuesto por las especies *Stizoptera bichenovii, Poephila acuticauda, Lagonosticta senegala, Lonchura striata, Uraeginthus bengalus, Amandava subflava*. Moises Rivera et al. [15] estudiaron la reconstrucción de la filogenia de estas especies a partir de propiedades seleccionadas de sus cantos mediante el análisis con métodos estadísticos. En el presente trabajo, la reconstrucción de la filogenia a partir de cantos se realizó con métodos que no exigen seleccionar manualmente las propiedades relevantes que potencialmente permitan la diferenciación de las especies, sino con métodos que detectan automáticamente las características importantes del canto.

Si bien la motivación del trabajo surge del ejemplo de los cantos de pájaros, el problema de reconstruir la filogenia puede pensarse de forma más general como el problema de reconstruir árboles partiendo de información sobre sus hojas. Se espera que las imágenes de los sonogramas, representaciones del sonido en términos de la frecuencia a lo largo del tiempo, formen las hojas de un árbol evolutivo. De la misma forma, se puede intentar reconstruir el árbol de matrices que comparten determinadas relaciones jerárquicas.

El trabajo se enfoca en analizar imágenes con relaciones jerárquicas y obtener una representación de las mismas en un espacio de baja dimensión. Posteriormente, se busca reconstruir el árbol del cual provienen a partir de las distancias entre las distintas clases

en el espacio de inmersión, también llamado espacio de embeddings. Para el ejemplo de los pájaros, es posible comparar el árbol generado a partir de los cantos con el generado a partir de datos genéticos. En la figura 1.1 pueden observarse algunos sonogramas utilizados.

Los embeddings deben ser representaciones más simples de los datos y a su vez trasladar ciertas relaciones que se observan en el espacio original a este nuevo espacio, típicamente de dimensión más pequeña. En este sentido, se introduce el modelo de la bola de Poincaré. Este modelo de geometría hiperbólica fue utilizado por Maximilian Nickel y Douwe Kiela en 2017 con la idea de explotar algunas propiedades que lo convertían en un espacio adecuado para generar embeddings de datos con estructuras jerárquicas, como los árboles [12]. En particular, su trabajo consistió en generar embeddings de la clausura transitiva de un conjunto de relaciones de hiperonimia entre sustantivos. A partir de estas relaciones entre pares de sustantivos generaron árboles dentro de la bola de Poincaré que representan su estructura jerárquica. Partiendo de estas ideas, se decidió utilizar este espacio en el contexto de representar datos de imágenes que potencialmente posean una estructura jerárquica.

El método para generar los embeddings partiendo de imágenes fue optimizar un modelo de red neuronal siamesa. Una de las principales ventajas con respecto a otros métodos estadísticos o de machine learning es que no necesita especificar las propiedades que permitan diferenciar las clases, sino que a través de redes convolucionales las imágenes son transformados en puntos dentro de la bola de Poincaré. En 2022, Roberto Bistel, Alejandro Martinez y Gabriel B. Mindlin utilizaron redes siamesas para generar embeddings de cantos de *Zonotrichia capensis* [2].

Una vez obtenida la representación de las imágenes en el espacio hiperbólico, la distancia entre las distintas clases puede medirse en función de la métrica definida en este nuevo espacio. Utilizando algoritmos que permiten reconstruir árboles a partir de la distancia entre sus hojas, el algoritmo *neighbor-joining* y agrupamiento jerárquico generan árboles donde cada una de las hojas representa a cada una de las clases.

En el capítulo 2 se describen los espacios hiperbólicos y el modelo de Poincaré. El capítulo 3 presenta las principales características de las redes neuronales convolucionales y las redes siamesas. En el capítulo 4 se estudian algoritmos para la reconstrucción de árboles y en el capítulo 5 se presentan los resultados de la herramienta desarrollada, aplicados a un conjunto de datos generados artificialmente y al conjunto de sonogramas de las especies de pájaros descriptas.

(a) Sonograma de un canto de la especie Lonchura striata

(b) Sonograma de un canto de Stizoptera bichenovii

Figura 1.1: Sonogramas de cantos.

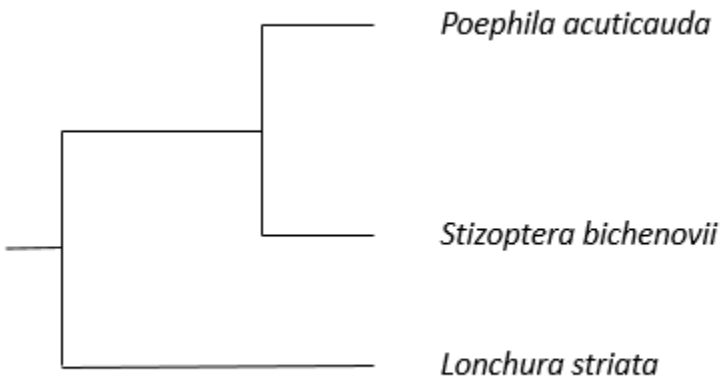


Figura 1.2: Ejemplo de árbol filogenético de las especies de aves Poephila acuticauda, Stizoptera bichenovii y Lonchura striata.

Capítulo 2

Embeddings de Poincaré

2.1. Geometría hiperbólica

La geometría hiperbólica es un tipo de geometría no euclidiana, es decir, que surge a partir de eliminar uno de los axiomas de Euclides. Está relacionada, por ejemplo, con el espacio-tiempo de Minkowski en la teoría de la relatividad especial. Los espacios hiperbólicos han comenzado a recibir atención en la comunidad de machine learning, ya que son adecuados para modelar datos.

Los axiomas de la geometría euclidiana son:

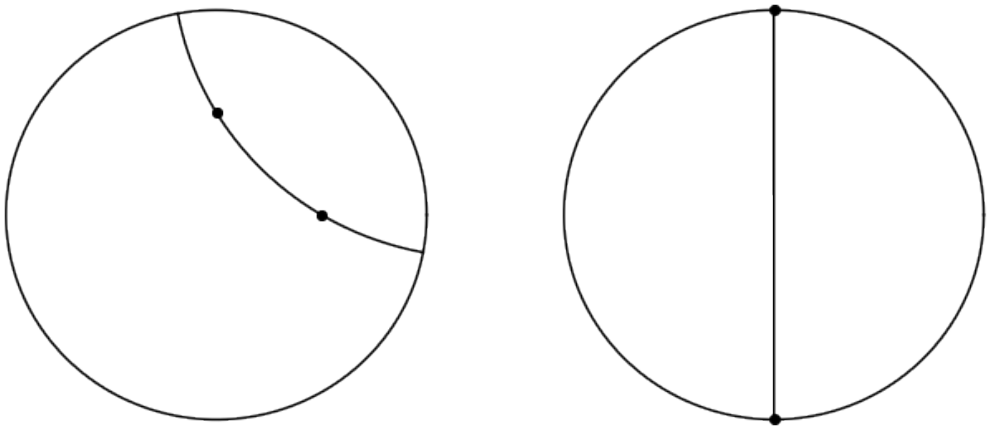
1. Cada par de puntos puede unirse por un y solo un segmento de línea recta.
2. Cada segmento de línea recta puede extenderse infinitamente en cada dirección.
3. Existe un solo círculo de un determinado radio para un determinado centro.
4. Todos los ángulos rectos son congruentes entre sí.
5. Si una línea recta corta a otras dos, de tal manera que la suma de los dos ángulos interiores del mismo lado sea menor que dos rectos, las dos rectas se cortan, al prolongarlas, por el lado en el que están los ángulos menores que dos rectos.

Dados los primeros cuatro postulados, el quinto es equivalente a:

5'. Dada una línea y un punto que no está en ella, existe solo una línea que pasa por el punto y es paralela a la primera.

A partir de la negación de este quinto axioma surge una teoría consistente. De esta forma, el quinto axioma para la geometría no euclidiana es:

> Dada una línea y un punto que no esté en ella, existe más de una línea que pasa por el punto y es paralela a la primera.



(a) Línea en el disco de Poincaré formada por un arco de un círculo euclídeo.

(b) Línea en el disco de Poincaré formada por un diámetro euclídeo.

Figura 2.1: Disco de Poincaré

Tomando este nuevo axioma aparecen ciertas ramificaciones:

- La suma de los ángulos en un triángulo es menor que la de dos ángulos rectos.
- No hay dos líneas que son equidistantes en todo punto entre ellas.
- Si una línea interseca una de dos líneas paralelas, puede no intersecar a la otra.
- Líneas paralelas a la misma línea no tienen que ser paralelas entre ellas.
- Dos líneas que se intersecan pueden ser paralelas a la misma.

El modelo usual para la geometría euclidiana es $\mathbb{R}^2$, el plano cartesiano. Para visualizar la geometría hiperbólica existen diferentes modelos que la interpretan de forma consistente con los axiomas. Algunos de ellos son el modelo de Klein, el modelo del hiperboloide, el modelo del semiplano de Poincaré y el modelo del disco de Poincaré. Todos estos modelos son isomorfos entre sí, para estudiar sus propiedades puede verse [10].

2.2. Modelo del disco de Poincaré

El modelo del disco de Poincaré utiliza el interior de un disco unitario euclídeo como espacio de representación.

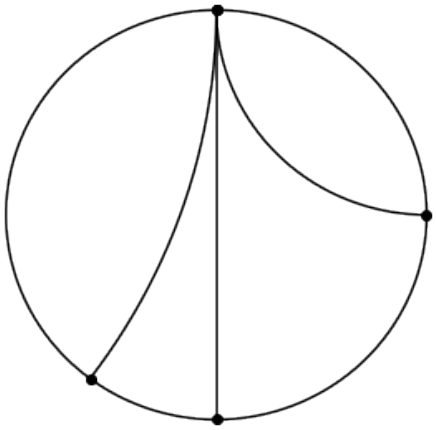


Figura 2.2: Líneas asintóticamente paralelas en el disco de Poincaré.

$$D = \{(x, y) \, | x^2 + y^2 < 1\} \tag{2.1}$$

Notar que el borde Γ de D no pertenece al disco. Para definir el espacio hay que determinar cómo son las líneas y las distancias.

Existen dos tipos de líneas en el disco de Poincaré. Por un lado, aquellas que son diámetros de Γ, es decir, el primer tipo de líneas son las que están formadas por los puntos dentro de D tal que su largo euclídeo es el diámetro de Γ. El segundo tipo de líneas son los arcos de círculos euclídeos que se intersecan de forma ortogonal con Γ. Ambos tipos de líneas pueden observarse en la figura 2.1. El ángulo entre dos líneas de Poincaré es definido midiendo el ángulo euclídeo entre sus líneas tangentes.

Definición 2.2.1. Dos líneas son paralelas si no se intersecan en D. Si se intersecan en el borde de D entonces se llaman asintóticamente paralelas.

En la figura 2.2 puede verse un ejemplo de líneas paralelas asintóticas. Es fácil ver que dada una línea y un punto existen infinitas líneas paralelas que pasan por ese punto.

Proposición 2.2.1 (Clasificación de paralelas)**.** *Si dos líneas son paralelas pero no asintóticamente paralelas, es decir, que no se acercan a un punto en común en Γ, entonces deben admitir una línea perpendicular a ambas.*

Demostración. La demostración puede encontrarse en [8]. □

La distancia entre los puntos se define de la siguiente manera

$$d\,(u, v) = \operatorname{arcosh}\left(\frac{1 + 2\|u - v\|^2}{(1 - \|u\|^2)(1 - \|v\|^2)}\right) \tag{2.2}$$

Especializando en el caso en que uno de los puntos es el centro del círculo y tomando r como la distancia euclídea, entonces la distancia hiperbólica es:

$$\ln\left(\frac{1+r}{1-r}\right) = 2\text{artanh}\,(r) \tag{2.3}$$

Notemos que esta distancia crece rápidamente entre puntos que están cerca del borde

Lema 2.2.1. *La distancia hiperbólica de un punto en el interior de Γ al borde del disco es infinita.*

Demostración. Trivial a partir de la ecuación para la distancia hiperbólica 2.3. □

Con respecto a los círculos en este espacio, es posible mostrar que si se tiene un punto que no está en el centro del disco, entonces la distancia hiperbólica en una dirección se ve sesgada con respecto a la distancia euclídea. Por lo tanto, podría ser esperable que los círculos en el disco de Poincaré tomen una forma elíptica o similar; sin embargo, no es el caso. Los círculos hiperbólicos se ven como los euclídeos, con la salvedad de que el centro se encuentra corrido.

Proposición 2.2.2. *Para cada círculo de Poincaré existe un punto A y un r tal que es igual al círculo euclídeo de centro A y radio r*

Demostración. La demostración puede encontrarse en la Proposición 7.12 de [8]. □

2.3. Embeddings de Poincaré

El motivo para estudiar las propiedades de la geometría hiperbólica es comprender mejor el espacio en el cual se representaron los datos en el trabajo.

Si consideramos datos con una jerarquía como un árbol, entonces el espacio euclídeo no es ideal. Por ejemplo, un árbol con un factor de ramificación b tiene $(b+1)b^{l-1}$ nodos en el nivel l y $\frac{(b+1)b^l-2}{b-1}$ nodos en niveles menores o iguales a l. De esta forma, a medida que crecen los niveles del árbol, el número de nodos crece exponencialmente.

Utilizando como algoritmo la idea de colocar la raíz del árbol en el centro, los hijos de forma equidistante y repitiendo este procedimiento de forma recursiva en cada rama, podemos representar nuestro árbol dentro del espacio hiperbólico. En la figura 2.3 podemos visualizar este embedding. En ella, todos los puntos son equidistantes de sus padres. Este tipo de construcción es posible debido a que en el espacio hiperbólico el área del círculo crece de manera exponencial, lo cual posibilita que esta estructura de árbol se pueda modelar fácilmente con dos dimensiones.

Por ejemplo, para el caso de dos dimensiones en el espacio hiperbólico con una curvatura de $K = -1$, la longitud L de un círculo y el área de un disco son

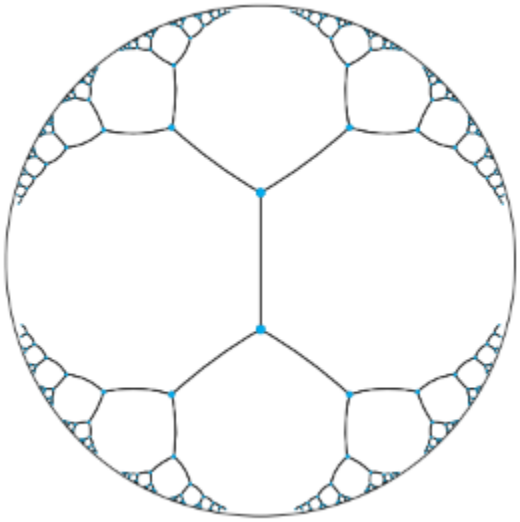


Figura 2.3: Embedding de un árbol con factor de ramificación 2 dentro del disco de Poincaré. Imagen tomada de [12].

$$L = 2\pi \sinh(r)$$
$$A = 2\pi(\cosh(r) - 1)$$

De esta forma, si se coloca la raíz de un árbol en el origen la distancia relativa a otros nodos es pequeña. Sin embargo, las hojas pueden ubicarse cerca del borde, ya que la distancia crece muy rápido para puntos con norma cercana a 1.

Capítulo 3

Redes Neuronales

El objetivo de calcular embeddings de imágenes, tanto de las matrices artificiales como de los sonogramas, es conseguir una representación con una buena noción de distancia. Simplifican la representación de imágenes disminuyendo la dimensión y a la vez conservan las relaciones entre ellas.

La bola de Poincaré se utiliza como espacio de embeddings para los datos de matrices artificiales y sonogramas debido a que ambos poseen una estructura jerárquica de la que forman parte. En el caso de las matrices, representan las hojas de un árbol construido ramificando áreas de la matriz en ceros o unos. Los sonogramas son los cantos de distintas especies que pueden relacionarse mediante un árbol filogenético, la hipótesis es que los sonogramas son las hojas del árbol evolutivo.

Los embeddings se obtienen a partir de aplicar redes neuronales a los datos. Una de las ventajas que ofrecen las redes sobre otros métodos de aprendizaje automático es que no es necesario especificar cuáles son las características relevantes de las imágenes para identificar patrones. Este capítulo introduce las principales características de las redes neuronales convolucionales y de las redes siamesas.

3.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son utilizadas principalmente para procesar datos que posean una topología de grilla [7]. Por ejemplo, series de tiempo pueden pensarse como grillas en una dimensión o imágenes, en dos dimensiones. Estas últimas, debido a la alta dimensión en general, requieren una arquitectura especializada.

Típicamente píxeles que se encuentran cerca están relacionados entre sí; sin embargo, las redes neuronales totalmente conectadas no están diseñadas para capturar esta noción de cercanía.

Las redes para el tratamiento de imágenes deben ser invariantes ante transformaciones geométricas. Es decir, deben ser capaces de identificar imágenes a las que se le apliquen translaciones, rotaciones, deformaciones.

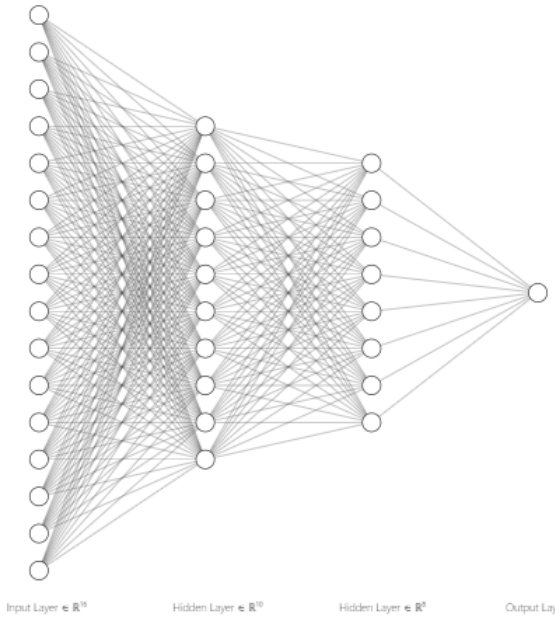


Figura 3.1: Red neuronal totalmente conectada con dos capas ocultas.

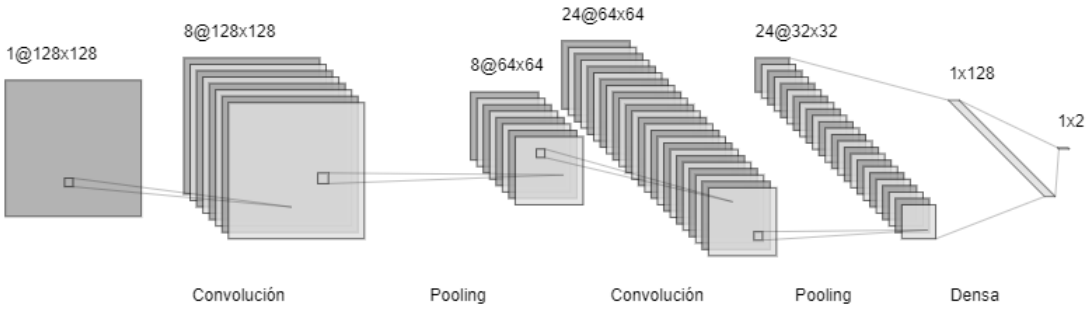


Figura 3.2: Red neuronal convolucional con dos capas convolucionales, dos capas de pooling y capas densas terminando en un output de dimensión 2.

Estas redes están compuestas por capas convolucionales que procesan cada región local de las imágenes de forma independiente. Estas explotan relaciones espaciales entre píxeles cercanos. De esta forma, una red compuesta mayoritariamente por capas convolucionales se conoce como red neuronal convolucional.

3.1.1. Capa Convolucional

La operación de convolución con un input $x(t)$ y un núcleo (*kernel*) $w(t)$ se define como

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{3.1}$$

La convolución discreta se define de la siguiente manera

$$s(t) = (x * w)(t) = \sum_{-\infty}^{\infty} x(a)w(t-a)da \tag{3.2}$$

Una capa convolucional para un input de una dimensión con $a[*]$ función de activación, input x, output h y núcleo w de tamaño 3 se define como

$$h_i = a\left[\beta_i + \sum_{j=1}^{3} w_j x_{i+j-2}\right] \tag{3.3}$$

Este es un caso especial de una capa totalmente conectada o densa, la misma calcula cada unidad oculta h_i como

$$h_i = a\left[\beta_i + \sum_{j=1}^{d} w_{ij} x_j\right] \tag{3.4}$$

El input x tiene dimensión d y el output h también tiene esa misma dimensión. Por lo tanto, la matriz w tiene dimensión $d \times d$.

Adaptando estas ideas a dos dimensiones, se define un núcleo como una matriz por la cual se pasa la matriz input para obtener el mapa de features. Un núcleo de 3×3 aplicado a un input de dos dimensiones compuesto por elementos x_{ij} calcula un output h_{ij} convolucionando el input, agregando un sesgo β y aplicándole una función de activación $a[*]$:

$$h_{ij} = a\left[\beta + \sum_{m=1}^{3}\sum_{n=1}^{3} w_{mn} x_{i+m-2,j+n-2}\right] \tag{3.5}$$

donde w_{mn} son las componentes de la convolución. El núcleo genera un output aplicándolo horizontalmente y verticalmente sobre la imagen.

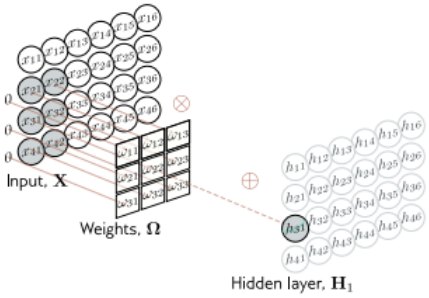


Figura 3.3: Convolución con un núcleo de 3×3 con *zero padding*. Imagen tomada de [14].

Al aplicar el núcleo, para calcular $h_{i,j}$ se suman los inputs de los $x_{i,j}$ cercanos pesándolos por w. Los diferentes outputs se calculan trasladando el núcleo a lo largo de la imagen. Para lidiar con los bordes, donde no hay inputs previos, se suele utilizar *zero padding* que consiste en rellenar con ceros las posiciones por fuera de los bordes de la imagen. De esta forma se evita que el output reduzca su dimensión luego de cada convolución. Puede observarse el procedimiento en la figura 3.3.

Las capas convolusionales están formadas por una serie de núcleos. Tienen ventajas computacionales con respecto a una capa total conectada debido a que limita fuertemente el número de pesos que deben ser entrenados. En cada una de las capas, cada unidad oculta se calcula utilizando un promedio ponderado de inputs cercanos con los el núcleo, agregando sesgo y aplicándole la función de activación. Estos pesos y sesgo son los mismos para todas las posiciones espaciales, por lo tanto, se reduce considerablemente la cantidad de parámetros con respecto a una capa totalmente conectada. Además, el número de parámetros no se incrementa con el tamaño del input.

Generalmente, después de una capa de convolución sigue una de *pooling*. Pooling es una forma de submuestreo que introduce invariancia por translación a la red. Una posibilidad es una capa de *max pooling*, la cual consiste en seleccionar submatrices de $n \times n$ y tomar el máximo para cada una. De esta forma se reduce la dimensión del mapa de features. Por ejemplo, max pooling con n igual a 2 toma el máximo de los valores de un input de 2×2, reduciendo cada dimensión a la mitad. Esto induce invariancia a las translaciones, ya que si el input se traslada por un pixel, muchos de los máximos continúan siendo los mismos. *Mean pooling* realiza un promedio de los inputs.

Las redes convolucionales usualmente consisten en capas convolucionales intercaladas con capas de pooling. De esta forma a medida que se avanza en la red, la dimensión va disminuyendo a la vez que la cantidad de canales crece. Al final de la red, típicamente hay una capa totalmente conectada para llegar al output.

3.1.2. Funciones de activación

La principal tarea de las funciones de activación es producir no linealidades en la red neuronal. El mapa de features pasa a una función de activación para que la red se adapte a características no lineales de los inputs.

Definición 3.1.1. La función de activación perceptrón se define como

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{sino} \end{cases} \tag{3.6}$$

Esta función es muy simple, sin embargo, usualmente no es utilizada para redes profundas. Otra posible función de activación es la sigmoidea.

Definición 3.1.2. La función de activación sigmoidea se define como

$$f(x) = \frac{1}{e^{-x} + 1} \tag{3.7}$$

Resulta ser una versión suavizada y continua de la función perceptrón. La sigmoidea se parece cada vez más a uno a medida que $x \to$ ínf y tiende a cero a medida que $x \to -$ ínf.

La función de activación más utilizada para redes convolucionales es la capa de unidad lineal rectificada (ReLU). Una ventaja es que requiere menos trabajo computacional en comparación a otras.

Definición 3.1.3. La función de activación ReLu se define como

$$f(x) = \text{máx}(0, x) \tag{3.8}$$

ReLu asigna cero a valores menores a cero y el mismo valor para los que son mayores a cero. Al utilizar ReLu, en algunos casos un gran número de neuronas son empujadas a estados en los cuales quedan inactivas para casi cualquier input. En este caso, ningún gradiente fluye a través de la neurona y queda permanentemente en un estado inactivo. En general ocurre cuando la tasa de aprendizaje es alta.

Definición 3.1.4. La función de activación Leaky ReLu se define como

$$f(x) = \begin{cases} x & \text{si } x > 0 \\ 0{,}01x & \text{sino} \end{cases} \tag{3.9}$$

Leaky ReLu permite un gradiente pequeño y positivo cuando la unidad no está activa, lo que ayuda a mitigar el problema del gradiente que desaparece.

3.1.3. Optimización

Algunos de los principales algoritmos de optimización para redes neuronales son descenso de gradiente estocástico, descenso de gradiente estocástico con momentum, RMSProp y Adam. En esta sección se describen el descenso de gradiente estocástico con momentum y Adam [7].

El método de momentum está diseñado para acelerar el aprendizaje del descenso de gradiente estocástico. Este método acumula una media móvil que decae exponencialmente de los gradientes pasados. Este algoritmo introduce una variable que cumple el rol de velocidad. La velocidad se define como un promedio de los gradientes negativos que decae exponencialmente. El nombre deriva de una analogía con la física en donde el gradiente es una fuerza moviendo una partícula. El hiperparámetro α toma valores entre cero y uno, y determina que tan rápido decae exponencialmente la contribución de gradientes pasados.

$$v \leftarrow \alpha v - \epsilon g \tag{3.10}$$

Cuanto más grande sea α relativo a ϵ, más importancia tienen los gradientes anteriores en determinar la dirección actual.

Algoritmo 1: Descenso con gradiente estocástico con momentum

Parámetro: learning rate ϵ
Parámetro: momentum α
mientras θ *no converja* **hacer**
  Elegir un minibatch de m ejemplos x_1,..,x_m con targets y_1,...,y_m
  $t \leftarrow t + 1$
  Calcular el gradiente $g \leftarrow \frac{1}{m} \nabla_\theta \sum L(f(x_i; \theta), y_i)$
  Calcular la velocidad v: $v \leftarrow \alpha v - \epsilon g$
  Actualizar θ: $\theta \leftarrow \theta + v$
fin

Adam es un algoritmo adaptativo para optimización de funciones objetivo estocásticas. El nombre surge a partir de que utiliza estimaciones del primer y segundo momento del gradiente g para adaptar la tasa de aprendizaje. Este incorpora una estimación del gradiente con un peso exponencial y también incluye correcciones de sesgo a las estimaciones de los momentos.

El algoritmo adapta la tasa de aprendizaje haciendo uso del promedio de los segundos momentos de los gradientes. Calcula la media móvil exponencial de gradientes y gradientes cuadrados. Los parámetros ρ_1 y ρ_2 se utilizan para controlar la tasa de decaimiento de estos promedios. Las medias móviles s y r son estimaciones de la media y varianza no centrada del gradiente. Estas se inicializan como vectores de ceros, lo

cual genera que los estimadores tengan sesgo y por este motivo luego se realiza una corrección definiendo $\hat{s}$ y $\hat{r}$.

En cuanto a la regla de actualización de Adam, el paso tomado en el espacio de parámetros es $\Delta\theta = -\epsilon\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$. En general, $|\frac{\hat{s}}{\sqrt{\hat{r}}}| \approx 1$ debido a que $|\mathbb{E}[g]/\mathbb{E}[g^2]| \leq 1$ [11]. Por lo tanto, la magnitud del paso $\Delta\theta$ tomado en cada iteración está aproximadamente acotado por $|\Delta\theta| \leq \epsilon$.

Para estimar el segundo momento del gradiente, utiliza la media móvil exponencial del cuadrado de los gradientes con una tasa de decaimiento ρ_2. Notar que $r_t = \rho_2 r_{t-1} + (1-\rho_2)g_t^2$ puede escribirse como:

$$r_t = (1-\rho_2)\sum_{i=1}^{t}\rho_2^{t-1}g_i^2 \tag{3.11}$$

Calculando $\mathbb{E}[r_t]$ se obtiene una expresión que lo relaciona con $\mathbb{E}[g_t]$:

$$\begin{aligned}
\mathbb{E}[r_t] &= \mathbb{E}\left[(1-\rho_2)\sum_{i=1}^{t}\rho_2^{t-1}g_i^2\right] \\
\mathbb{E}[r_t] &= \mathbb{E}[g_t^2](1-\rho_2)\sum_{i=1}^{t}\rho_2^{t-1} + \xi \\
\mathbb{E}[r_t] &= \mathbb{E}[g_t^2](1-\rho_2) + \xi
\end{aligned} \tag{3.12}$$

El algoritmo divide r_t por $1-\rho_2$ para corregir el sesgo.

3.2. Redes Neuronales Siamesas

Las redes neuronales siamesas fueron introducidas por Bromley y LeCun con el objetivo de resolver el problema de verificación de firmas [3]. La red toma dos inputs distintos y el output es una medida de similitud entre los inputs. Dos sub-redes convolucionales idénticas actúan sobre cada uno de los inputs y los transforma en puntos de una dimensión menor. Se calcula la distancia entre los dos outputs de cada una de las redes convolucionales y de esta forma se mide la similitud entre los inputs. El esquema de una red neuronal siamesa puede observarse en 3.4.

El entrenamiento de la red siamesa se realiza optimizando una función de pérdida que toma la distancia obtenida como output. Esta función debe ser elegida de manera que minimizarla implique minimizar la distancia entre inputs de la misma clase y maximizar la distancia entre inputs de clases diferentes. Los pesos de las redes convolucionales que se aprenden en el entrenamiento tienen la restricción de que deben ser idénticos para ambas sub-redes. Como consecuencia la red tiene la propiedad de ser simétrica, es decir, que intercambiando el orden de las imágenes se obtiene la misma métrica como resultado final.

Algoritmo 2: Algoritmo Adam

Adam;
Parámetro: learning rate ϵ
Parámetro: decaimiento exponencial de los estimadores, ρ_1 y ρ_2
Parámetro: constante pequeña δ
Inicializar el primer y segundo momento s y $r = 0$
$t \leftarrow 0$
mientras θ_t *no converja* **hacer**
  Elegir un minibatch de m ejemplos x_1,..,x_m con targets y_1,...,y_m
  $t \leftarrow t + 1$
  Calcular el gradiente $g_t \leftarrow \frac{1}{m} \nabla_\theta \sum L(f(x_i; \theta_{t-1}), y_i)$
  Actualizar el estimador del primer momento: $s_t \leftarrow \rho_1 s_{t-1} + (1 - \rho_1) g_t$
  Actualizar el estimador del segundo momento: $r_t \leftarrow \rho_2 r_{t-1} + (1 - \rho_2) g_t \odot g_t$
  Corregir el sesgo en el primer momento: $\hat{s} \leftarrow \frac{s}{1-p_1^t}$
  Corregir el sesgo en el segundo momento: $\hat{r} \leftarrow \frac{r}{1-p_2^t}$
  $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$
  $\theta \leftarrow \theta + \Delta\theta$
fin

El objetivo es que una vez finalizado el entrenamiento, la sub-red convolucional traslade los inputs al nuevo espacio de embeddings y sus distancias representen las distancias entre los inputs. De esta forma, las redes siamesas aprenden una métrica de similitud a entre los datos. Además, es apropiada para problemas en los que no se cuenta con una gran cantidad de ejemplos por categoría. En particular, al utilizar redes convolucionales para generar las representaciones de los datos, las cuales son robustas para distorsiones geométricas, la métrica de similitud resultante se espera que sea robusta a diferencias pequeñas entre pares de imágenes.

3.2.1. Contrastive Loss

La red convolucional debe transformar inputs de un espacio de dimensión alta a un espacio de baja dimensión con la propiedad de que inputs similares tienen que transformarse en vectores cercanos e inputs disímiles en vectores distantes. Una función de pérdida deseable cumple que su minimización produce una red con estas propiedades.

Tomando X_1 y X_2 como inputs, F como la sub-red neuronal y w como los parámetros a optimizar, entonces la distancia entre inputs y la función de pérdida se definen como

Figura 3.4: Red neuronal siamesa que toma dos imágenes de sonogramas como input y calcula la distancia entre los embeddings.

$$
\begin{aligned}
&D_w(X_1, X_2) = dist(F_w(X_1), F_w(X_2)) \\
&\mathcal{L}(w) = \frac{1}{p}\sum_{i=1}^{p} L(w, Y^i, X_1^i, X_2^i) \\
&L(w, Y, X_1, X_2) = YLS(w, X_1, X_2) + (1 - Y)LD(w, X_1, X_2)
\end{aligned}
\tag{3.13}
$$

LS es la función de pérdida si los inputs son de la misma clase y LD para el caso en que son de distintas clases. La variable $Y \in \{0, 1\}$. Cuando son de la misma clase, Y vale uno y en el caso contrario vale cero, p es la cantidad de pares en la muestra. LS y LD tienen que cumplir que al minimizarlas en función de w se deben obtener valores pequeños de D_w cuando los inputs son similares y valores grandes cuando no. Una posible elección para estas funciones es la siguiente:

$$
\begin{aligned}
&LS(w, X_1, X_2) = D_w^2 \\
&LD(w, X_1, X_2) = max(0, m - D_w)^2
\end{aligned}
\tag{3.14}
$$

Por lo tanto la pérdida queda definida como:

$$
L(w, Y, X_1, X_2) = YD_w^2 + (1 - Y)max(0, m - D_w)^2 \tag{3.15}
$$

La contrastive loss toma los outputs de la sub-red convolucional y calcula su distancia si provienen de dos inputs de la misma clase. En el caso de que sean de distintas

clases calcula la diferencia entre un margen m y la distancia entre los inputs, si esta diferencia es negativa entonces la función de pérdida vale cero.

De esta forma, cuando los inputs son de la misma clase se minimiza la distancia de los vectores en el espacio de embeddings y cuando no están en la misma clase se obtienen valores pequeños al maximizar la distancia en el espacio de embeddings.

Si definimos los outputs de la red convolucional como $x_1 = F_w(X_1)$ y $x_2 = F_w(X_2)$ la función de perdida puede expresarse de forma más simple como

$$L(x_1, x_2, Y) = Y dist(x_1, x_2)^2 + (1 - Y) max(0, m - dist(x_1, x_2))^2 \tag{3.16}$$

En general, las métricas de distancias utilizadas son $\|F_w(X_1) - F_w(X_2))\|_2$ o $\|F_w(X_1) - F_w(X_2))\|_1$. En este trabajo se eligió la distancia de Poincaré definida en la ecuación 2.2.

3.3. Tarea de verificación

Con el objetivo de medir el desempeño del modelo, se eligen al azar k clases de inputs. Para cada una de estas clases se selecciona un input al azar y de esta forma se compone un conjunto de datos de soporte. A su vez, se selecciona al azar un input de prueba de una de estas clases. Luego es posible comparar el input de prueba con cada uno de los del conjunto de soporte. Se le aplica el modelo F_w a cada uno y luego se mide la distancia entre los embeddings. Si la mínima distancia se produce entre elementos de la misma clase, el resultado es exitoso y en el caso contrario no lo es.

Repitiendo este procedimiento n veces obtenemos una métrica de *accuracy* para el modelo. Definiendo s_j con j entre 0 y $k-1$ a los inputs de soporte, t al input de prueba y asumiendo que están ordenados de manera que en $j = 0$ coincide la clase del soporte con la de prueba, obtenemos

$$v = \begin{cases} 1 & \text{si } \arg\min_j d(s_j, t) = 0 \\ 0 & \text{sino} \end{cases} \tag{3.17}$$

$$\text{accuracy} = \frac{1}{n} \sum_{i}^{n} v_i \tag{3.18}$$

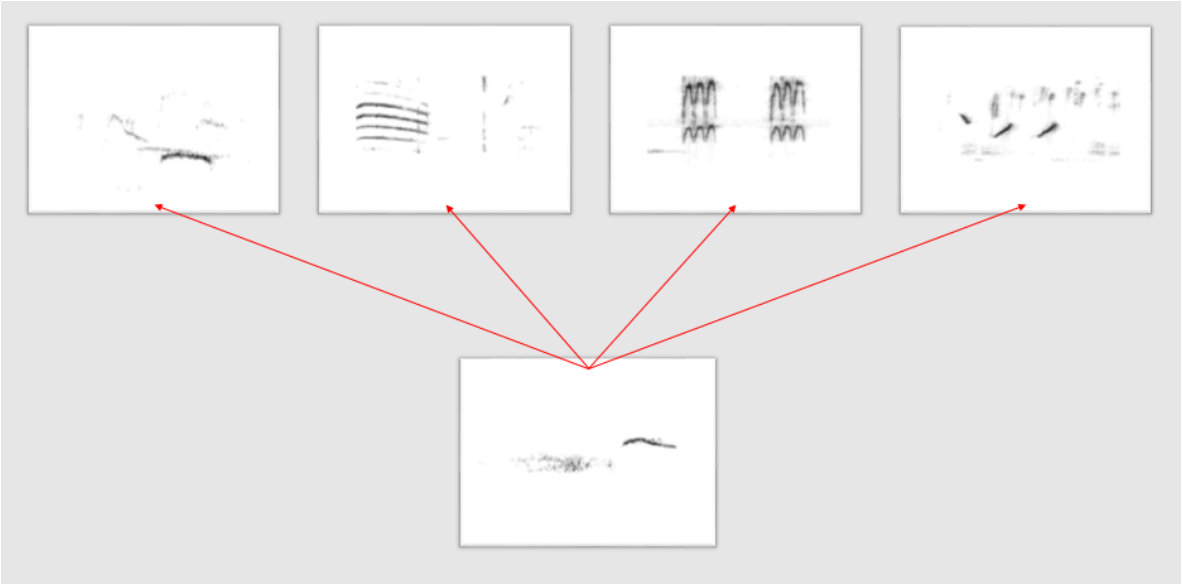


Figura 3.5: Tarea de verificación con cuatro imágenes de distintas clases de sonogramas que forman el conjunto de soporte y una imagen de prueba.

Capítulo 4

Reconstrucción de árboles evolutivos

4.1. Árbol filogenético

Los árboles filogenéticos permiten visualizar el desarrollo evolutivo de diferentes especies, subespecies o poblaciones llamadas *taxones* y las relaciones existentes entre ellas. Cada una de las hojas de estos árboles representan a taxones diferentes, los vértices interiores representan ancestros comunes de los cuales derivan y en el caso de que exista un ancestro común a todos los taxones se encuentra en la raíz del árbol.

Un árbol puede tener raíz o no tenerla. Conocer la ubicación de la raíz proporciona información acerca del orden de las bifurcaciones en el tiempo. Partiendo desde este ancestro en común a todos se puede ver como se derivan distintos taxones y sus ramificaciones.

En el caso de que las aristas del árbol no especifiquen distancias, solamente permite analizar las ramificaciones entre los taxones. Es decir, se considera únicamente la topología del árbol. Dos árboles sin raíz son topológicamente equivalentes si existe un isomorfismo entre los grafos que representan, por ejemplo, los árboles de la figura 4.1 son equivalentes.

Además de una topología, los árboles pueden tener una estructura métrica si a cada una de las aristas se le asigna una longitud. Típicamente, esta longitud representa la distancia entre taxones, puede graficarse especificando la longitud al costado de la arista o simplemente quedar sugerida por el tamaño con el cual se dibuja la arista.

A medida que aumenta el número de taxones también aumenta de manera exponencial número de árboles evolutivos que los relacionan. Debido al número elevado de árboles a analizar, construir un buen árbol puede ser muy lento. Un algoritmo que encuentre el mejor posible analizando individualmente cada uno de ellos es demasiado costoso en tiempo cuando se estudian más de unos pocos taxones.

Un árbol puede definirse como un grafo conexo sin ciclos. Los taxones más actuales

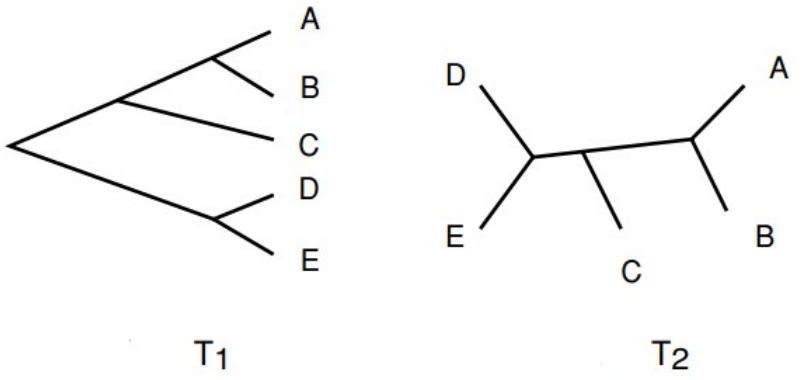


Figura 4.1: Dos árboles T_1 y T_2, como árboles sin raíz, son topológicamente equivalentes. Imagen tomada de [5].

se representan como hojas de los árboles. Teniendo en cuenta que el grado de un nodo se define como la cantidad de vecinos, las hojas del árbol se caracterizan por tener grado uno. Los nodos que poseen grado mayor a uno son los internos y representan un ancestro desconocido a partir del cual derivan otros taxones. Dada una hoja j, al único nodo con el que se conecta se lo llama padre de j. Además, a la arista que conecta una hoja con su padre se la llama rama.

En la construcción de árboles filogenéticos, típicamente se desea relacionar taxones que están presentes en la actualidad. Se posee información de estos, pero no de los representados por nodos internos. A partir de las distancias entre los taxones terminales se desea reconstruir las relaciones evolutivas. La información de las distancias puede obtenerse de diferentes formas, por ejemplo, a partir del análisis de la genética de los taxones. En el caso de las especies de aves analizadas en este trabajo, la información de las distancias surge a partir de diferencias y similitudes entre sus cantos.

El objetivo de la primera parte de este capítulo es estudiar propiedades de matrices de distancias entre taxones que luego sirvan para aplicar algoritmos de reconstrucción de árboles evolutivos.

4.1.1. Matrices de distancia

Una matriz de distancias D de $n \times n$ es aquella que se utiliza para representar las distancias que existen entre diferentes taxones. Por ejemplo, $D_{i,j}$ puede representar las diferencias entre genes del taxon i y del taxon j. Este tipo de matrices deben satisfacer una serie de propiedades.

Definición 4.1.1. Dada una matriz D de $n \times n$, se llama matriz de distancias si satisface que es simétrica, no negativa y satisface la desigualdad triangular, es decir, $D_{i,k} \leq D_{i,j} + D_{j,k}$.

Para representar la distancia evolutiva entre taxones se le asigna a cada arista una longitud entre los nodos que conecta. La longitud de un camino dentro del árbol está definido como la suma de las distancias de cada arista. De esta forma, para un árbol T la distancia evolutiva entre dos taxones presentes i y j, llamada $d_{i,j}(T)$, corresponde a la longitud del camino que las une.

El problema a estudiar es la construcción de un árbol filogenético a partir de la matriz de distancias de las hojas del árbol, es decir, de los taxones terminales. Se dice que un árbol T se ajusta a una matriz de distancias D, si para i,j hojas del árbol vale que $d_{i,j}(T) = D_{i,j}$. La primera pregunta que surge es si dada una matriz D, existe siempre un árbol que se ajuste a D. La respuesta es que no, no siempre existe.

Definición 4.1.2. Una matriz de distancia D se llama aditiva si existe un árbol T que se ajuste a D.

La unicidad tampoco está garantizada, es decir, dada una matriz de distancias D es posible que exista más de árbol que la represente. En la figura 4.3 se puede encontrar un ejemplo en el cual esto no se cumple, ya que existen dos árboles que son representados por la misma matriz de distancias. Simplemente, se obtienen agregando nodos intermedios.

Definición 4.1.3. Se dice que un camino de un árbol es no ramificado si cada nodo, con excepción de los extremos, tiene grado igual a dos.

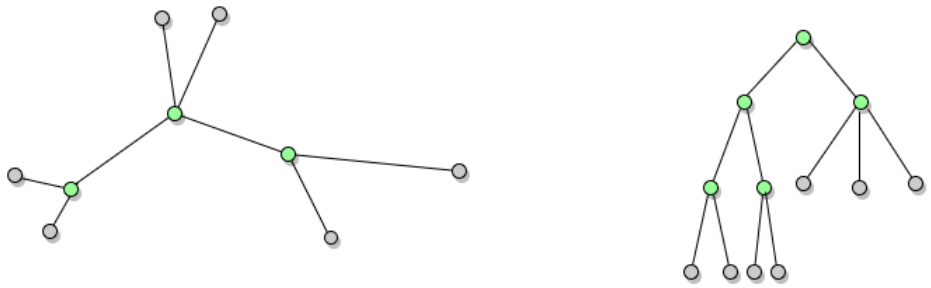
Un camino no ramificado es maximal si no es un subcamino de otro más largo. Notar que en la figura 4.3 si en el árbol superior se substituyen los caminos no ramificados por una sola arista que tenga la misma longitud, entonces se obtiene el inferior. El árbol resultante no tiene nodos de grado dos, llamamos *árbol simple* a aquellos que tengan esta característica.

Si un árbol simple se ajusta a una matriz de distancia D y tiene aristas internas con peso cero, entonces es posible removerlas fácilmente juntando los nodos que conecta y de esta forma se obtiene otro árbol que se ajusta a D. Luego, en lo siguiente se asumen que no solo el árbol es simple, sino que tampoco contiene aristas internas con nodos con peso cero.

Proposición 4.1.1. *Dada una matriz D aditiva, existe un único árbol simple que se ajusta a D.*

Demostración. La demostración puede encontrarse en [13]. □

A partir del desarrollo previo, el problema puede replantearse como la búsqueda de un método para reconstruir el árbol simple, llamado $ARBOL(D)$, a partir de la matriz aditiva D.



(a) Árbol sin raíz.

(b) Árbol con raíz, la raíz representa el ancestro de todos los nodos en el árbol.

Figura 4.2: Árboles

$$\begin{array}{c} \\ i \\ j \\ k \\ l \end{array} \begin{array}{c} \begin{array}{cccc} i & j & k & l \end{array} \\ \begin{bmatrix} 0 & 3 & 6 & 4 \\ 3 & 0 & 7 & 5 \\ 6 & 7 & 0 & 2 \\ 4 & 5 & 2 & 0 \end{bmatrix} \end{array}$$

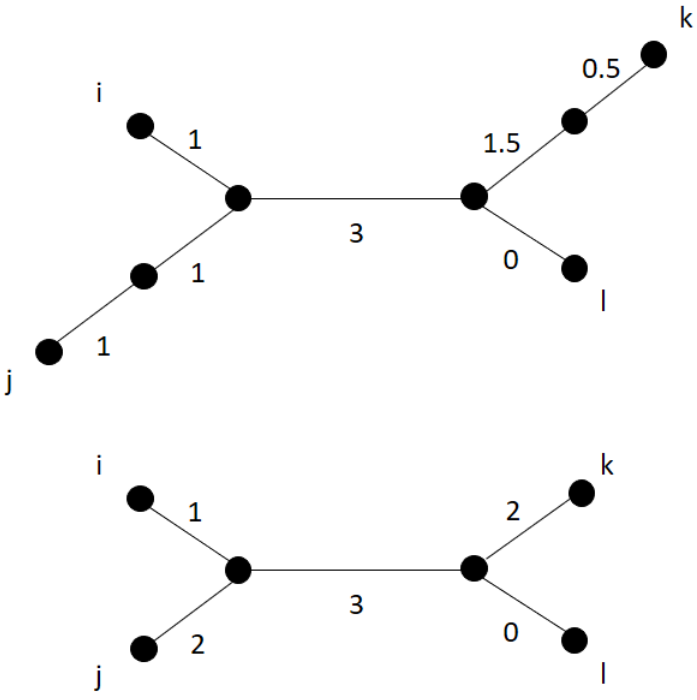


Figura 4.3: (izquierda) matriz de distancias y (derecha) par de árboles sin raíz que representan la misma matriz. Imagen modificada de [13].

4.2. Algoritmo Additive-Philogeny

La idea básica del algoritmo es realizar de forma recursiva los siguientes pasos:

- Elegir una hoja y calcular la distancia entre la hoja y su padre
- Construir una nueva matriz D^* que represente al árbol sin esta rama
- Aplicar el algoritmo a D^* para obtener $ARBOL(D^*)$
- Encontrar el punto en $ARBOL(D^*)$ donde unir la rama con la distancia calculada para formar $ARBOL(D)$

En primer lugar, va a ser necesario calcular el largo la arista que une una hoja con su padre. Se define $largoDeRama(D, n)$ como el largo de la rama que conecta la hoja n a su padre. Para calcularlo se tiene en cuenta el siguiente teorema.

Teorema 1. *Dada una matriz aditiva D y una hoja j, $largoDeRama(D, j)$ es igual al mínimo valor de $(D_{i,j} + D_{j,k} - D_{i,k})/2$ entre todos los pares de hojas j y k.*

Demostración. Como el árbol es simple, el padre de j tiene por lo menos grado tres. De esta forma puede pensarse al padre de j dividendo el árbol en tres subárboles por lo menos. Se llama T_i y T_j a los subárboles que contienen a la hoja i y j respectivamente.

Dados un par de hojas i y k, estas pueden pertenecer al mismo subárbol o a diferentes. En primer lugar, si se asume que pertenecen a diferentes subárboles T_i y T_k, como j conecta el camino entre i y k entonces vale:

$$\begin{aligned} d_{i,j} &= d_{i,padre(j)} + largoDeRama(j) \\ d_{j,k} &= d_{k,padre(j)} + largoDeRama(j) \end{aligned} \tag{4.1}$$

Por lo tanto, se obtiene

$$\begin{aligned} d_{i,j} + d_{j,k} &= d_{i,padre(j)} + d_{k,padre(j)} + 2largoDeRama(j) \\ d_{i,j} + d_{j,k} &= d_{i,k} + 2largoDeRama(j) \end{aligned} \tag{4.2}$$

Despejando estas ecuaciones $largoDeRama = \frac{d_{i,j}+d_{j,k}-d_{i,k}}{2}$

Por otro lado, si se considera que tanto i como k pertenecen al mismo subárbol, el camino entre estas hojas no pasa por el padre de j. Luego se obtiene

$$d_{i,k} \leq d_{i,padre(j)} + d_{k,padre(j)} \tag{4.3}$$

A partir de esta observación, puede verse que

$$\begin{aligned} d_{i,j} + d_{j,k} &= d_{i,padre(j)} + d_{k,padre(j)} + 2largoDeRama(j) \\ largoDeRama(j) &= \frac{d_{i,j} + d_{j,k} - (d_{i,padre(j)} + d_{k,padre(j)})}{2} \end{aligned} \tag{4.4}$$

Por último, con la desigualdad 4.3 y la última ecuación se puede acotar $largoDeRama$ por

$$largoDeRama(j) \leq \frac{d_{i,j} + d_{j,k} - d_{i,k}}{2} \tag{4.5}$$

Es decir que la cota vale para todo par de i y k. Como siempre pueden elegirse i y k tal que estén en diferentes subárboles entones $largoDeRama$ toma el mínimo valor entre todos los posibles i y k. □

Una vez que se tiene un algoritmo para calcular el largo de cada rama, puede utilizarse para construir un método recursivo que permita obtener un árbol simple a partir de D. El primer paso consiste en tomar una hoja j y recortar la rama que la une a su padre. Como se desconoce cuál es el árbol simple, es necesario expresarlo en términos de la matriz D.

Antes de obtener la matriz D^*, se calcula una matriz intermedia D_{inter} :

$$\begin{aligned} D^{inter}_{j,m} &= D_{j,m} - largoDeRama(j) \qquad m \neq j \\ D^{inter}_{n,j} &= D_{n,j} - largoDeRama(j) \qquad n \neq j \\ D^{inter}_{n,m} &= D_{n,j} \qquad n \neq j \wedge m \neq j \end{aligned} \tag{4.6}$$

Luego de restarle $largoDeRama(j)$ a cada elemento de la columna j y de la fila j con excepción de la diagonal, el largo de la rama j del árbol asociado a D^{inter} es cero. Notar que el largo de la rama para cualquier otra hoja continúa siendo el mismo en D^{iter} que en D. Eliminando la columna j y la fila j, lo cual representa recortar la rama del árbol, se obtiene la nueva matriz D^* de tamaño $(n-1) \times (n-1)$. Sobre esta matriz puede aplicarse de forma recursiva el algoritmo para encontrar el árbol con $n-1$ hojas que se ajusta a D^*.

Posteriormente se busca el punto en el cual debe agregarse la rama al árbol simple. Observar que el árbol obtenido por D^{inter} es el mismo que el de D, con la única diferencia que el primero tiene $largoDeRama(D^{inter}, j)$ igual a cero. Por lo tanto, existen i y k tal que

$$\begin{aligned} \frac{D^{inter}_{i,j} + D^{inter}_{j,k} - D^{inter}_{i,k}}{2} &= 0 \\ D^{inter}_{i,k} &= D^{inter}_{i,j} + D^{inter}_{j,k} \end{aligned} \tag{4.7}$$

Esto implica que el punto para agregar la rama debe estar a distancia $D^{inter}_{i,j}$ de la hoja i en el camino que une i con k en el árbol recortado. El punto de unión puede ser en un nodo ya existente o bien en un nuevo nodo que corta una arista del árbol.

A partir estas observaciones surge el algoritmo $AdditvePhylogeny$. El pseudocódigo puede verse en el algoritmo 3.

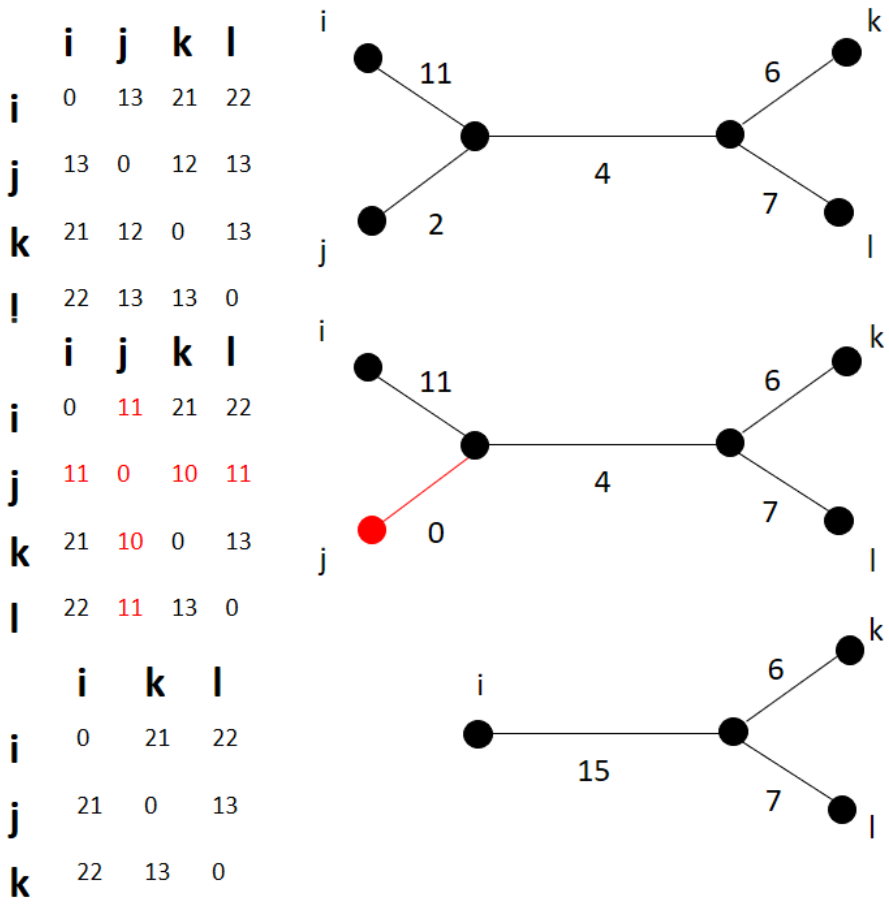


Figura 4.4: (arriba) matriz D de distancias y su árbol correspondiente. (medio) matriz intermedia D^{inter} y el árbol con el largo de la rama j igual a 0. (abajo) matriz D^* sin la rama j. Imagen modificada de [13].

Algoritmo 3: Algoritmo para reconstruir árboles

```
1  AdditivePhylogeny (D, n);
   Input  : matriz de distancias D y numero total de hojas n
   Output: Arbol(D)
2  si n = 2 entonces
3  |  T ← árbol que consiste en una sola arista de distancia D_{1,2};
4  |  return T
5  en otro caso
6  |  largoDeRama ← largoDeRama(D, n);
7  |  para i ← 1 a n − 1 hacer
8  |  |  D_{j,n} ← D_{j,n} − largoDeRama
9  |  |  D_{n,j} ← D_{j,n}
10 |  fin
11 |  (i, n, k) ← hojas tal que D_{i,k} = D_{i,n} + D_{n,k}
12 |  x ← D_{i,n}
13 |  eliminar la fila n y columna n de D
14 |  T ← AdditivePhylogeny(D, n − 1)
15 |  v ← el nodo en T a distancia x de i en el camino entre i y k
16 |  Agregar una hoja n nuevamente a T creando la rama (v,n) de largo
   |   largoDeRama
17 |  return T
18 fin
```

El problema que surge es que en general las matrices generadas a partir de datos reales no son aditivas, dado que las distancias no son medidas de forma exacta. Por lo tanto es necesario pensar un algoritmo que pueda ser utilizado para estas. La pregunta que se responde en la siguiente sección es si es posible encontrar un algoritmo que siempre identifique las hojas vecinas para matrices aditivas, pero también tenga buenos resultados para matrices de distancia no aditivas.

4.3. Algoritmo Neighbor-Joining

Uno de los métodos más utilizados para reconstruir árboles evolutivos es el algoritmo $Neighbor - Joining$. Este fue desarrollado por Naruya Saitou and Masatoshi Nei [16]. Para dimensionar la importancia que tuvo el desarrollo del algoritmo en la reconstrucción de árboles filogenéticos, es interesante notar que este paper de 1987 es uno de los 20 más citados en la historia de la ciencia, con más de 70000 citas. Por ejemplo, $Neighbor - joining$ permitió el analisis filogenético de los genomas de SARS-CoV-2 aislados de diversas regiones del mundo.

La idea es que dada una matriz de distancias, el algoritmo elige pares de hojas y las substituye por una sola hoja, de esta forma reconstruye un árbol repitiendo el procedimiento de forma recursiva. A pesar de que encontrar un mínimo en la matriz de distancia no garantiza encontrar un par de vecinos en el árbol, es posible transformar esta matriz en otra cuyo mínimo elemento sí sea un par de vecinos.

Se define la $distanciaTotal(D, i)$ como la suma de las distancias $D_{i,j}$ a todas las otras hojas j.

$$distanciaTotal(D, i) = \sum_j D_{i,j} \tag{4.8}$$

La matriz intermedia D^* cumple que los elementos de la diagonal son cero y lo siguiente:

$$D^*_{i,j} = (n-2)D_{i,j} - distanciaTotal(D, i) - distanciaTotal(D, j) \tag{4.9}$$

Cada $D^*_{i,j}$ puede reescribirse en función de la longitud de las aristas d del árbol. Se define $multiplicidad_{i,j}(e)$ como el coeficiente que acompaña a la arista e en $D^*_{i,j}$.

Lema 4.3.1. *Dada una matriz aditiva D y un par de hojas i y j en el árbol de D, $multiplicidad_{i,j}(e) = -2$ para cualquier rama e en el árbol. [13]*

Definición 4.3.1. Dada una arista e y dos hojas i y j, e separa el árbol en dos subárboles T_1 y T_2. En el caso de que i y j se encuentren en el mismo subárbol, se define $hojas_{i,j}(e)$ como la cantidad de hojas del subárbol que no contiene a i y j.

Teorema 2 ($Multiplicidad$)**.** *Dada una matriz aditiva D, la $multiplicidad_{i,j}$ de una arista interna e es -2 si está en el camino entre i y j, en caso contrario es $-2hojas_{i,j}(e)$.*

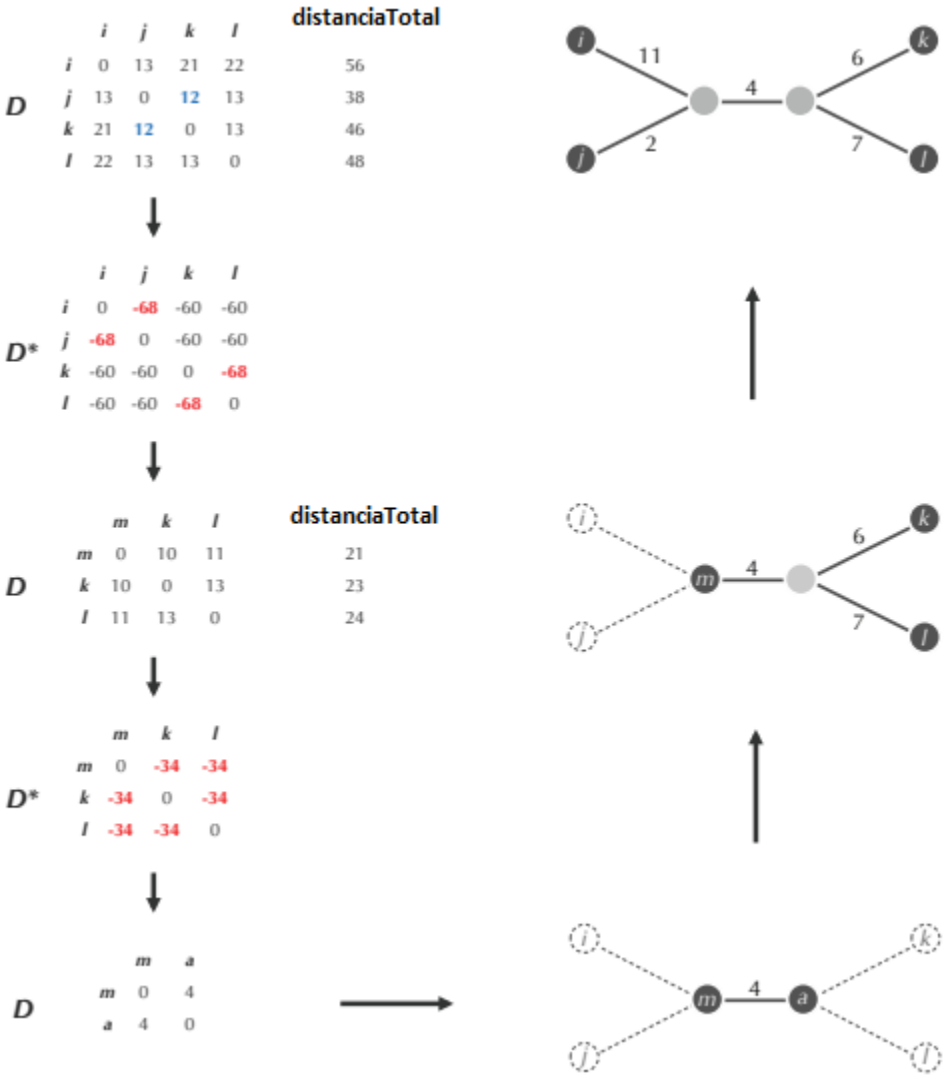


Figura 4.5: método $neighbor - joining$ para una matriz aditiva D. Imagen tomada de [13].

Demostración. En el caso de que la arista interna e se encuentre en el camino entre i y j, e tiene coeficiente $n-2$ en el término $(n-2)D_{i,j}$ de $D^*_{i,j}$. Consideremos los árboles T_i y T_j producto de eliminar la arista e. Para cada una de las hojas k en T_i el camino entre k y j pasa por e, por lo tanto, suma 1 en el coeficiente de e en $distanciaTotal(j)$. En cambio, el camino entre i y k no pasa por e por lo tanto no contribuye al coeficiente en $distanciaTotal(j)$. Se puede hacer un razonamiento equivalente para cada una de las hojas k en T_j.

Cada una de las hojas del árbol contribuye en uno al coeficiente de e o bien en $distanciaTotal(j)$ o bien en $distanciaTotal(i)$. Como la suma coeficiente de e en $distanciaTotal(j)$ + $distanciaTotal(i)$ es n. Se obtiene que

$$multiplicidad_{i,j}(e) = (n-2) - n = -2 \tag{4.10}$$

Por otro lado, si e no se encuentra en el camino entre i y j, luego e tiene coeficiente 0 en $(n-2)D_{i,j}$. Además, tomando k como una hoja en el subárbol donde no se encuentran i y j, para llegar a k se debe pasar por e. Por lo tanto, el coeficiente de e en $distanciaTotal(j)$ y $distanciaTotal(i)$ es $hojas_{i,j}(e)$, lo cual implica que $multiplicidad_{i,j}(e) = 0 - hojas_{i,j}(e)$, □

Como consecuencia del teorema se obtiene el siguiente lema

Lema 4.3.2. *Si i y j son vecinos en el $ARBOL(D)$ y k no es vecino de i, entonces $D^*_{i,j} < D^*_{i,k}$. [13]*

Teorema 3 ($Neighbor - Joining$)**.** *Dada una matriz aditiva D, el elemento más pequeño D^*_{ij} de la matriz de $neighbor - joining$ corresponde a un par de hojas vecinas i y j en el árbol.*

Demostración. Se toma como hipótesis que $D^*_{i,j}$ es el mínimo elemento de D^* pero i y j no son vecinos en el $ARBOL(D)$. El objetivo es encontrar un par de vecinos k y l tal que $D^*_{k,l} < D^*_{i,j}$. Por el enunciado anterior, ni i ni j pueden tener un vecino si $D^*_{i,j}$ es un elemento mínimo de D^*. Por lo tanto i y j son las únicas hojas conectadas al $padre(i)$ y $padre(j)$ respectivamente.

Dado que $ARBOL(D)$ es simple, $padre(i)$ y $padre(j)$ tienen grado igual o mayor a tres, lo cual significa que cada uno de los nodos está conectado a por lo menos otros dos del $ARBOL(D)$, uno de los cuales se encuentra en el camino entre i y j. El otro nodo es parte de su propio subárbol. Estos subárboles se nombran T_1 y T_2.

Asumiendo que el número de hojas en T_1 no supera el número de hojas en T_2 y definiendo a n como el número total de hojas en $ARBOL(D)$, debido a que i y j no están en T_1 o en T_2, T_1 debe contener menos de $n/2$ hojas y el resto de $ARBOL(D)$ debe tener más de $n/2$ hojas.

Como i no posee vecinos, T_1 debe tener al menos dos hojas. Esto implica que T_1 posee un par de vecinos k y l. Para ver que $D^*_{k,l} \leq D^*_{i,j}$, se elige un vértice e del árbol.

Si e se encuentra en el camino entre i y j, entonces por el teorema de multiplicidad de aristas, $multiplicidad_{i,j}(e) = -2$.

Si e se encuentra en el camino entre i y k, entonces remover esta arista desconecta el árbol en dos. Uno de ellos contiene a k y a l, y el otro contiene a i y j. El primero cumple que $hojas_{i,j}(e) < n/2$ y el segundo, $hojas_{k,l}(e) > n/2$. Luego por el teorema de multiplicidad de aristas

$$\begin{aligned} multiplicidad_{k,l}(e) &= -2hojas_{k,l}(e) \\ multiplicidad_{i,j}(e) &= -2hojas_{i,j}(e) \\ multiplicidad_{k,l}(e) &< multiplicidad_{i,j}(e) \end{aligned} \tag{4.11}$$

El tercer caso posible es que e no esté en el camino entre i y j ni tampoco en el camino entre i y k. En este caso, el subárbol que no contiene a i y j cuando se remueve a e es el mismo que no contiene a k y l. Por lo tanto, $hojas_{i,j}(e) = hojas_{k,l}(e)$ y de esta forma, $multiplicidad_{i,j}(e) = multiplicidad_{k,l}(e)$.

Por último, falta ver que la desigualdad es estricta. El camino entre i y k debe contener una arista e ya que i y k no son vecinos. Luego, $multiplicidad_{k,l}(e) < multiplicidad_{i,j}(e)$. Como todas las aristas tienen largo mayor a cero por hipótesis, se obtiene el resultado.

□

La idea principal del algoritmo es seleccionar el elemento mínimo en la matriz D^* de $neighbor - joining$. Por el teorema anterior, el par i y j corresponde a un par de hojas vecinas. Luego se reemplazan estas hojas por una nueva llamada m y se calcula la distancia de m a las otras hojas de la siguiente manera:

$$D_{k,m} = \frac{D_{k,i} + D_{k,j} - D_{i,j}}{2} \tag{4.12}$$

De esta forma, puede reducirse la matriz reemplazando las hojas i y j por m. Por lo tanto, se pasa de una matriz de $n \times n$ a otra de $(n-1) \times (n-1)$. Luego se aplica de forma recursiva $neighbor - joining$ a esta nueva matriz más pequeña y se arma el árbol agregándole dos ramas que unen m con i y j.

El largo de estas ramas se define como:

$$\begin{aligned} \Delta_{i,j} &= \frac{distanciaTotal(D,i) - distanciaTotal(D,j)}{n-2} \\ largoDeRama(i) &= \frac{D_{i,j} + \Delta_{i,j}}{2} \\ largoDeRama(j) &= \frac{D_{i,j} - \Delta_{i,j}}{2} \end{aligned} \tag{4.13}$$

Utilizando estas definiciones, el pseudocódigo se presenta en el algoritmo 4.

Algoritmo 4: Algoritmo Neighbor-Joining para reconstruir arboles

```
1  Neighbor-Joining (D, n);
   Input  : matriz de distancias D y numero total de hojas n
   Output: Arbol(D)
2  si n = 2 entonces
3  |  T ← árbol que consiste en una sola arista de distancia D_{1,2};
4  |  return T
5  en otro caso
6  |  D* ← la matriz neighbor-joining construida a partir de la matriz D;
   |    Encontrar elementos i y j tal que D*_{i,j} es un minimo no-diagonal de D*
   |    Δ ← (distanciaTotal(D, i) - distanciaTotal(D, j)) / (n - 2)
   |    largoDeRama_i ← (D_{i,j} + Δ)/2;
7  |  largoDeRama_j ← (D_{i,j} − Δ)/2;
8  |  Agregar una nueva columna/fila m a D tal que para todo k
   |    D_{k,m} = D_{m,k} = (D_{k,i} + D_{k.j} − D_{i,j})/2;
9  |  Remover filas i y j de D;
10 |  Remover columnas i y j de D;
11 |  T ← Neighbor − joining(D, n − 1);
12 |  Agregar dos nuevas ramas conectando el nodo m con i y j al arbol T;
13 |  Asignar largoDeRama_i a Rama(i);
14 |  Asignar largoDeRama_j a Rama(j);
15 |  return T
16 fin
```

4.3.1. Enraizamiento de árboles

El algoritmo $Neighbor-joining$ produce un árbol filogenético sin raíz. Sin embargo, conocer la raíz es de gran importancia para la interpretación de las relaciones evolutivas, ya que representa al ancestro común a todos los taxones del árbol y provee al árbol con un camino de evolución de las especies.

Existen diversos métodos para colocar la raíz a un árbol, el más utilizado se llama *outgroup rooting*. Una o más especies se identifican como un grupo externo, al resto de los taxones se los llama grupo interno. Cuando el grupo externo se separa del resto por una arista, puede agregarse la raíz de manera que separe al grupo externo del interno. El método requiere conocimiento biológico previo de cuál es el grupo externo apropiado, esto constituye una desventaja cuando no existe consenso sobre cuál seleccionar.

Otro método utilizado es *midpoint rooting*. Está basado en las distancias calculadas por el algoritmo de reconstrucción de árboles y estima la raíz basándose en hipótesis acerca del modelo de evolución. Consiste en ubicar el punto medio del camino más largo entre las hojas del árbol y colocar ahí la raíz. Cuando la hipótesis del reloj molecular

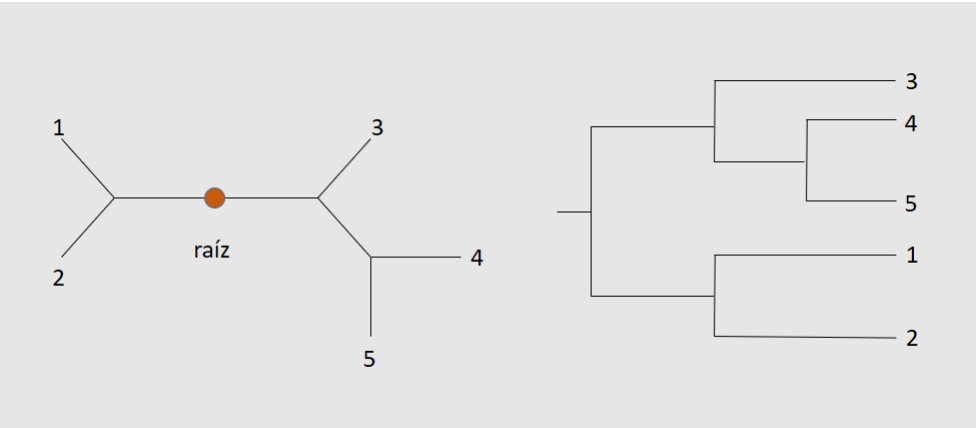


Figura 4.6: (derecha) árbol sin raíz y (izquierda) mismo árbol agregando una raíz.

aplica, es decir, proteínas evolucionan a una tasa relativamente constante para los individuos, el método de *midpoint rooting* tiende a ser preciso en la elección de la raíz. Sin embargo, cuando la hipótesis no se cumple, los resultados pueden ser pobres.

4.4. Clustering jerárquico

En esta sección, se describe el algoritmo de clustering jerárquico para la construcción de árboles. Dadas n hojas, la idea consiste en generar n particiones de forma progresiva en grupos. La primera partición tiene n grupos diferentes representados por cada una de las hojas. Luego, la segunda partición junta los dos grupos más cercanos en uno solo de dos elementos. De esta forma, en la k-ésima iteración, el algoritmo junta los dos grupos más cercanos de la iteración previa y obtiene $n - k + 1$ grupos.

En este caso, la forma elegida para juntar clusters y tomar un representante de los mismos es utilizar el promedio de Frechet. Dada una serie de puntos $x_1, .., x_n$ en un espacio M:

$$\begin{aligned} \psi(p) &= \sum_{i=1}^{n} w_i d^2(p, x_i) \\ m &= \arg min_{p \in M} \psi(p) \end{aligned} \tag{4.14}$$

Al inicio, cuando hay n clusters o grupos, luego de encontrar a C_i y C_j cuya distancia es mínima, el nuevo grupo C^* se define como el promedio de Frechet tomando como w a $1/2$. En general, dados C_i y C_j el nuevo representante es el promedio de Frechet donde w_i es el peso dado por la proporción de nodos que contiene el grupo i y w_j es la proporción de nodos que contiene el grupo j.

Algoritmo 5: Algoritmo Hierarchical Clustering para construir árboles

```
Hierarchical Clustering;
Output: Arbol
Clusters ← n hojas
Construir un grafo T con n nodos aislados
mientras Hay más de un cluster hacer
    Encontrar los dos clusters C_i y C_j más cercanos
    Juntar C_i y C_j en uno nuevo con los elementos de ambos
    Agregar a T un nuevo nodo C* que se conecta con C_i y C_j
    Remover los clusters C_i y C_j de Clusters
    Agregar C* a Clusters
fin
raíz ← el nodo en T que corresponde al cluster restante
return T
```

Capítulo 5

Resultados

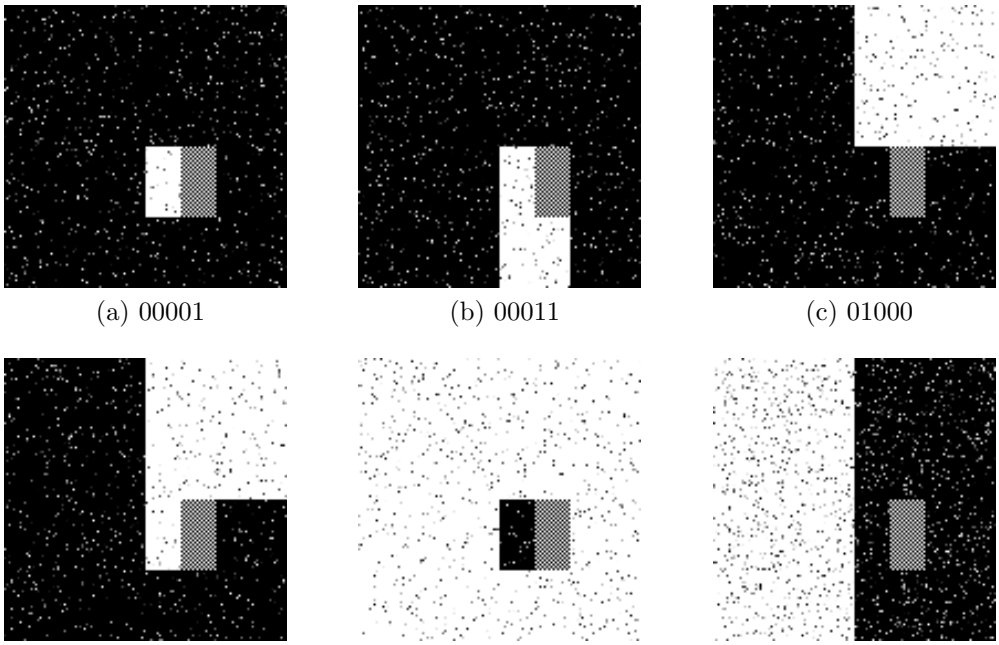
En este capítulo se describen los datos y modelos de redes neuronales utilizados para obtener embeddings. El trabajo contó con datos de sonogramas de seis especies de pájaros a partir de los cuales se generó un modelo que permitió representarlos en un espacio de bajas dimensiones. Antes de analizar los sonogramas, se estudió el comportamiento del modelo y del algoritmo de *neighbour-joining* con datos artificiales.

5.1. Datos artificiales

Se generaron imágenes artificiales con una estructura jerárquica de árbol. El algoritmo para construir las imágenes es el siguiente: la mitad izquierda de una imagen toma el color negro o blanco representando una primera rama que se separa en un cero o un uno. Luego, de la mitad derecha restante, la mitad superior se colorea de blanco o negro. Esto define una nueva ramificación del árbol en cero o uno. Queda un cuarto de la imagen sin decidir que color le corresponde. En el siguiente paso, la mitad izquierda del cuarto inferior derecho toma el color blanco o negro. Repitiendo el procedimiento durante cinco pasos, eligiendo en forma de espiral el sector de la imagen a colorear en cada paso, se obtienen diferentes clases que representan las hojas de un árbol. Luego de cinco ramificaciones, el área restante de la imagen se colorea de negro y blanco en proporciones iguales.

Con este procedimiento, hay 32 hojas posibles. Se eligen al azar 6 de estas hojas y se generan 80 imágenes para entrenar el modelo y 20 para validarlo. Para esto, a cada una de las imágenes se le agregó ruido, cambiando el 10 % de los píxeles por otros de diferentes tonalidades de gris. Las imágenes que corresponden a las hojas pueden verse en la figura 5.1. La estructura del árbol que las representa en la figura 5.2 y el árbol (b) es el resultado deseado que se busca reconstruir.

El modelo, que tiene como objetivo encontrar embeddings que permitan representar los datos, está compuesto por una red neuronal siamesa. Utilizando la notación $N@k \times k$



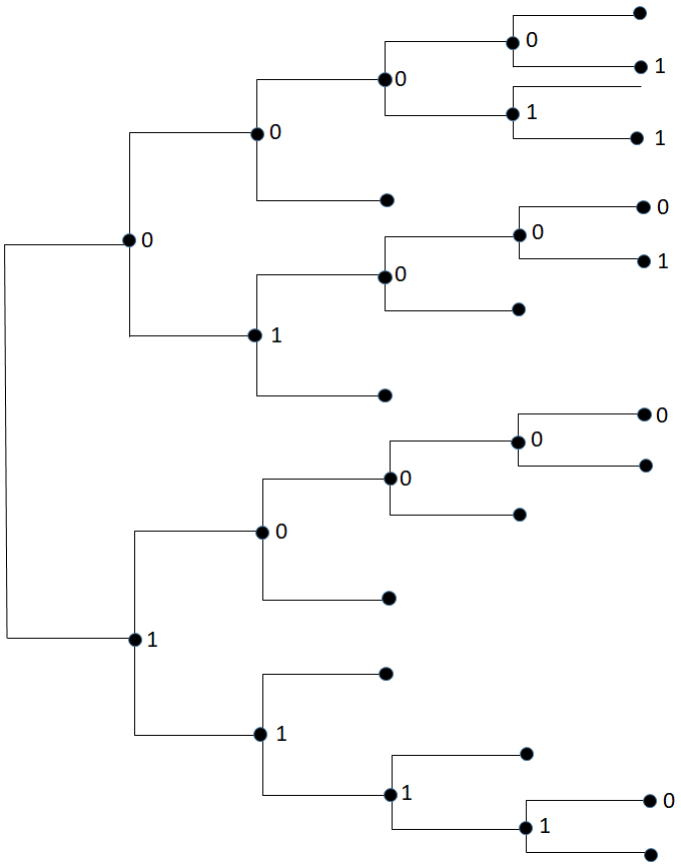
(a) 00001 (b) 00011 (c) 01000

(d) 01001 (e) 11110 (f) 10000

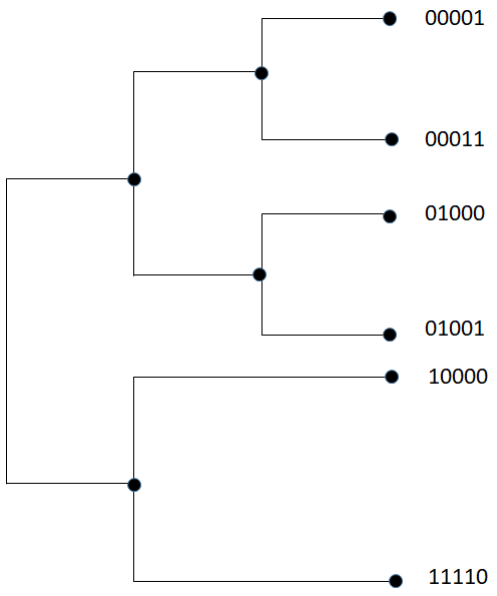
Figura 5.1: Seis imágenes con jerarquía implícita.

para una capa convolucional con N filtros y un núcleo de $k \times k$, la estructura de la red convolucional es la siguiente y puede visualizarse en la figura 5.3.

- Convolución + Relu 64@10×10
- Max Pool 2×2
- Convolución + Relu 128@7×7
- Max Pool 2×2
- Convolución + Relu 256@4×4
- Max Pool 2×2
- Convolución + Relu 512@4×4
- Max Pool 2×2
- Capa Densa 512 + Relu
- Capa Densa 2 + Lineal



(a) Arbol binario



(b) Arbol binario simplicado

Figura 5.2: Estructura jerárquica implícita de las matrices.

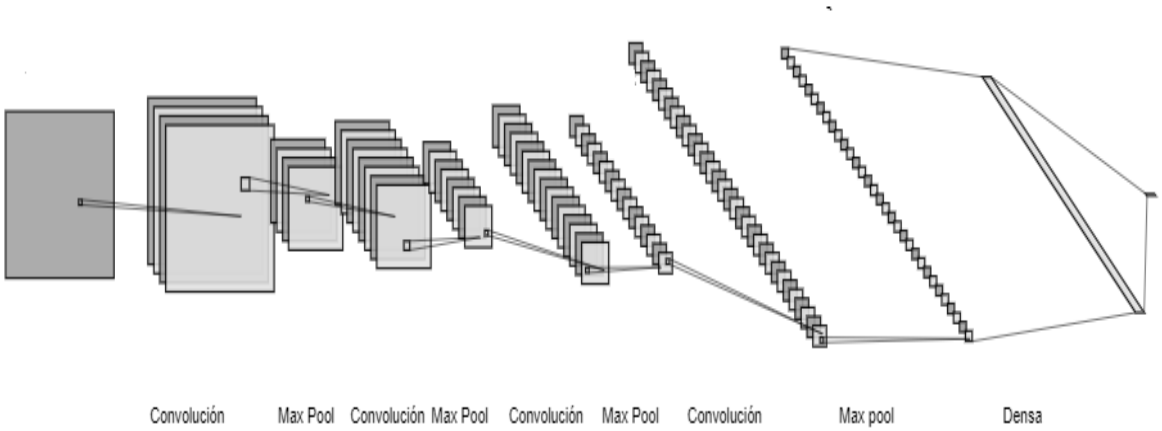


Figura 5.3: Red neuronal convolucional seleccionada para la red siamesa. Cuenta con cuatro capas convolucionales de 64, 128, 256 y 512 filtros respectivamente; con dos capas de pooling y una capa densa, terminando en un embedding de dimensión 2 dentro del disco de Poincaré.

- $x^* \leftarrow 0{,}99 \tanh(\|x\|)\frac{x}{\|x\|}$

De esta forma, la red neuronal siamesa queda definida utilizando dos redes convolucionales con función de activación *ReLu*; cada una de las cuales toma una imagen y produce un vector dentro del disco de Poincaré en dos dimensiones. La red sigue la estructura típica de las redes siamesas, luego de cada capa convolucional, que duplica la cantidad de filtros que la anterior, se coloca una capa de pooling que reduce la dimensión.

La idea es medir la distancia entre estos vectores utilizando la distancia de Poincaré definida en 2.2.

$$d\,(u,v) = \operatorname{arcosh}\left(\frac{1+2\|u-v\|^2}{(1-\|u\|^2)(1-\|v\|^2)}\right)$$

Para poder medirla es necesario que los vectores pertenezcan a la bola de radio euclídeo uno; entonces al vector x, obtenido como el resultado de dimensión dos de la última capa densa, se le aplica:

$$\begin{aligned} dx &= 0{,}01 \\ x^* &= (1-dx)\tanh(\|x\|)\frac{x}{\|x\|} \end{aligned} \tag{5.1}$$

Con esta transformación, x^* tiene radio menor a uno. A su vez, al utilizar la función tanh, aquellos puntos que se encuentran cerca del centro se mantienen cerca y los que tienen norma grande se ubican más cerca del límite de la bola. El nuevo punto x^* mantiene la dirección de x. Una vez obtenidos ambos vectores dentro de la bola de radio uno, es posible calcular la distancia de Poincaré entre ellos.

Optimizar la red neuronal consiste en encontrar los pesos que minimicen la función de pérdida definida en 3.16 para x_1^* y x_2^*.

$$L(x_1^*, x_2^*, Y) = Y dist(x_1^*, x_2^*) + (1 - Y) max(0, m - dist(x_1^*, x_2^*))$$

El parámetro m seleccionado fue 25 y se utilizó el optimizador $Adam$, junto con un parámetro de tasa de aprendizaje de 10^{-6}.

El aprendizaje se realizó con una serie de minibatchs de 32 pares de imágenes. La elección de estos pares no fue completamente al azar, sino que la mitad de los mismos fueron positivos, es decir, que se eligieron de forma que pertenezcan a la misma clase, y los de la otra mitad fueron negativos, es decir que pertenecían a clases distintas. El modelo fue entrenado durante 150 epochs o iteraciones.

Se realizaron 10 ejecuciones del modelo, donde para cada una se varió la inicialización de los pesos de la red. Los núcleos de las capas convolucionales se inicializaron con una distribución normal de media 0 y desvió estándar 0.1. Mientras que el componente de sesgo se inicializó con una distribución normal con media de 0.5 y desvió de 0.1.

Para medir el desempeño de la red neuronal, luego de cada 10 iteraciones se realizan 100 tareas de verificación para calcular la métrica de *accuracy* definida en 3.18. En este caso se eligen al azar 10 clases permitiendo repetición. Para cada una de estas clases se selecciona una imagen al azar formando el conjunto de datos de soporte. Se elige una imagen de prueba y para compararla con las de soporte se le aplica el modelo convolucional a cada una y luego se mide la distancia de Poincaré entre los puntos obtenidos en el espacio de embeddings.

Los embeddings obtenidos para estos datos pueden verse en las figuras 5.4 y 5.5. El parámetro m elegido generó que los las distintas clases tiendan a ubicarse hacia los bordes de la bola. La red neuronal separó exitosamente las diferentes clases proporcionadas y obtuvo un 98,2 % de accuracy en el entrenamiento y 97 % en la validación. En la tabla 5.1 pueden observarse los resultados para cada ejecución.

A partir de la representación de los datos en este espacio, se calcularon los centros de Chebyshev de cada una de las clases utilizando la distancia de Poincaré. Para calcularlos se resuelve:

$$\min_{x^*, r} \Big\{ r : d(x^*, x)^2 < r, x \in Q \Big\} \tag{5.2}$$

Equivalentemente:

$$\arg\min_{x^*} \max_{x \in Q} d(x, x^*)^2 \tag{5.3}$$

Accuracy		
	Entrenamiento	Validación
1	0.99	0.98
2	0.99	0.97
3	0.98	0.97
4	0.99	0.99
5	0.97	0.95
6	0.96	0.97
7	0.99	0.96
8	0.98	0.95
9	0.98	0.99
10	0.99	0.97

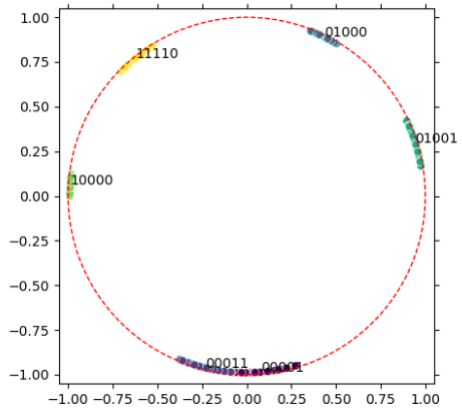
Tabla 5.1: *accuracy* para los datos de entrenamiento y validación de las 10 ejecuciones del modelo aplicado datos artificiales.

De esta forma, cada clase posee un representante en el espacio de embeddings. Utilizando estos representantes se generó la matriz de distancias de Poincaré entre las clases. Se aplicó el algoritmo $neigbor - joining$ para generar árboles y la raíz se eligió utilizando el criterio del punto medio (*midpoint rooting*). El resultado fue que en un 90 % de las ejecuciones se obtuvo el árbol (a) de la figura 5.6, por lo tanto, cada par de imágenes se identificó correctamente y el algoritmo fue robusto para este conjunto de datos. Luego de aplicar *midpoint rooting* para elegir la raíz, el 70 % de las ejecuciones obtuvo el árbol b. Por lo tanto, el 70 % de las veces se recuperó la totalidad de la estructura jerárquica de la figura 5.2.

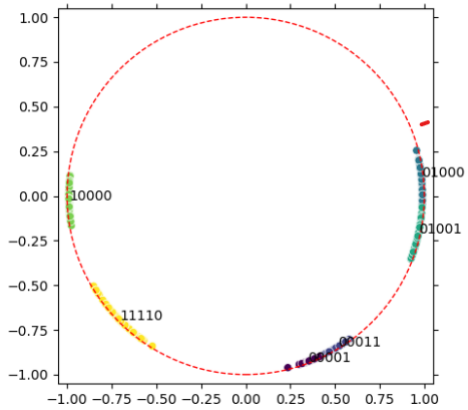
Además del algoritmo $neighbor - joining$, se aplicó el algoritmo de clustering jerárquico. Los árboles resultantes del 80 % de las ejecuciones pueden verse en la figura 5.7. Tanto el árbol (a) como el (b) no recuperaron la estructura deseada. El principal problema observado fue que debido al crecimiento exponencial de la distancia, cuando se juntan dos clusters cercanos al borde en un nuevo cluster, este se ubica más al centro que los anteriores. Como consecuencia, la distancia del nuevo cluster a todos los anteriores tiende a ser menor que la distancia entre los previos clusters. Este fenómeno genera multiples bifuraciones como puede observarse en las figuras.

5.2. Datos de sonogramas

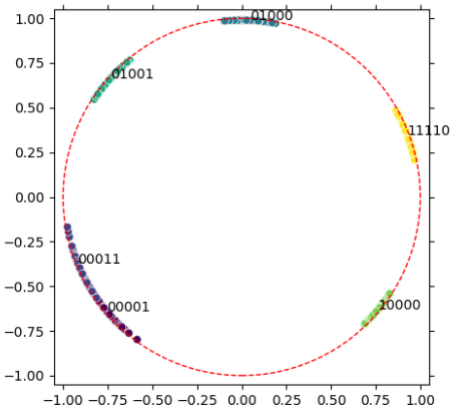
Las especies de pájaros analizadas fueron: *Stizoptera bichenovii, Poephila acuticauda, Lagonosticta senegala, Lonchura striata, Uraeginthus bengalus, Amandava subflava.* Los cantos fueron tomados de la base de datos *Xeno-canto* [6]. Estas especies fueron se-



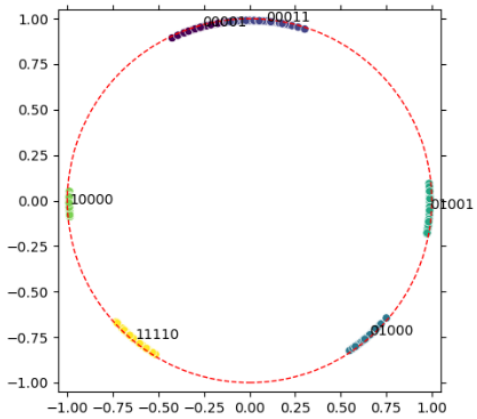
(a) Ejecución 1



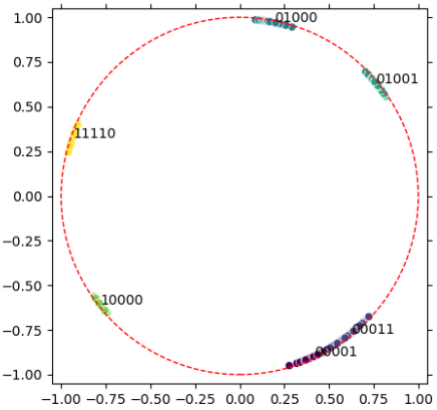
(b) Ejecución 2



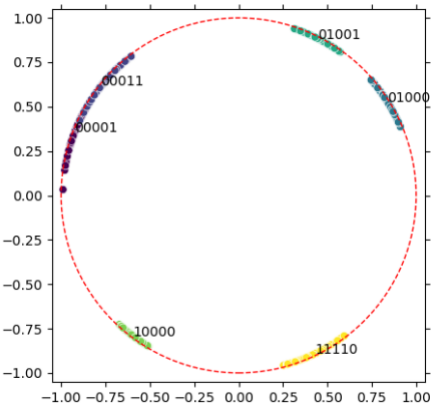
(c) Ejecución 3



(d) Ejecución 4

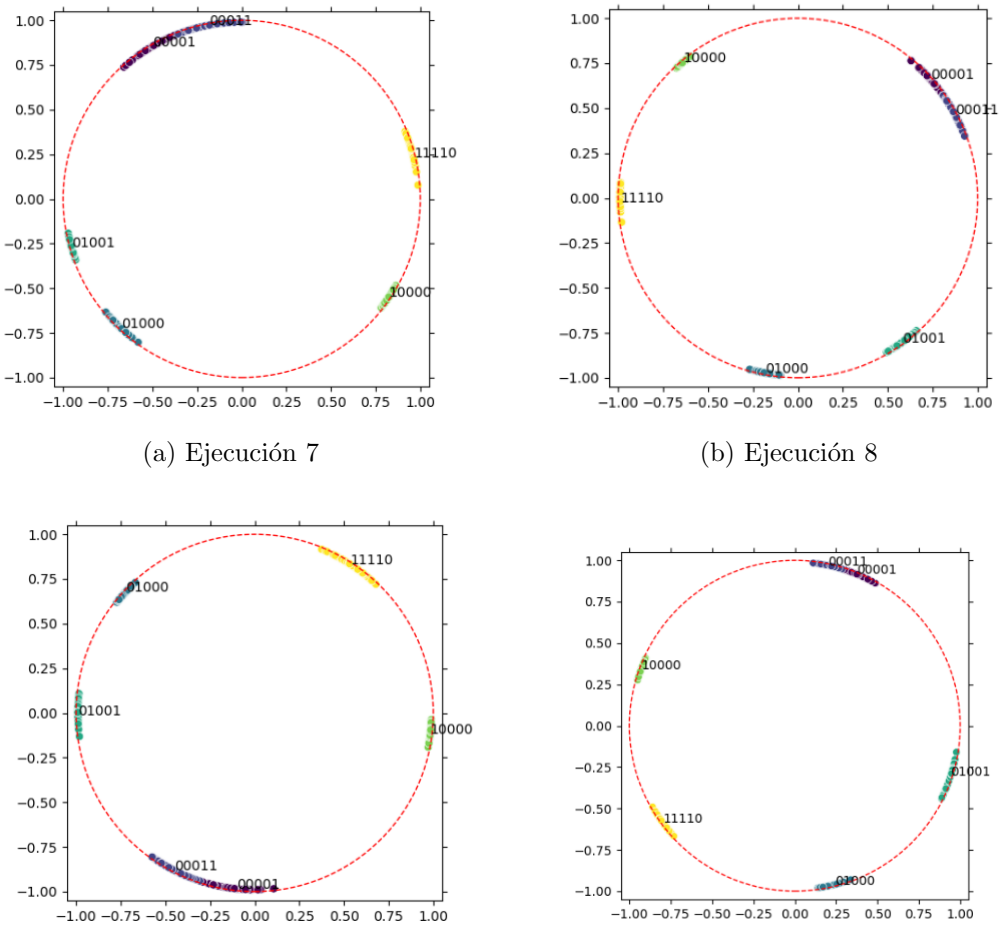


(e) Ejecución 5



(f) Ejecución 6

Figura 5.4: Embeddings en el disco de Poincaré.



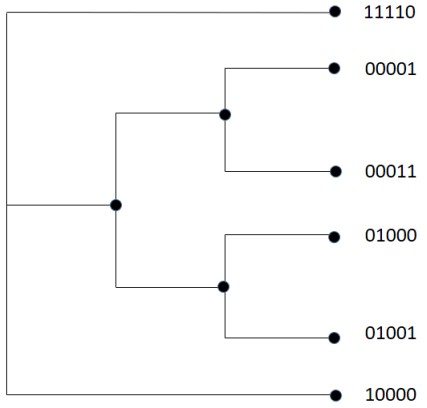
(a) Ejecución 7

(b) Ejecución 8

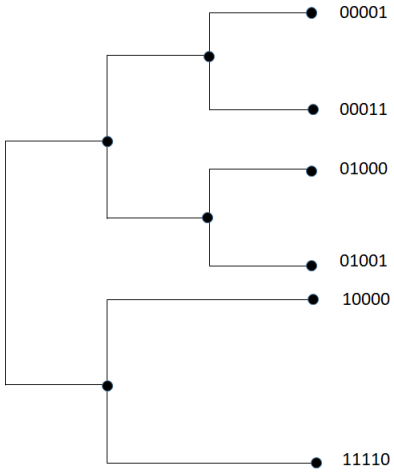
(c) Ejecución 9

(d) Ejecución 10

Figura 5.5: Embeddings en el disco de Poincaré.

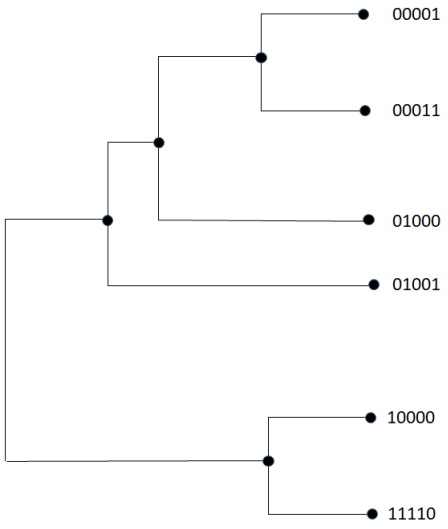


(a) árbol sin raíz obtenido un 90 % de las ejecuciones.

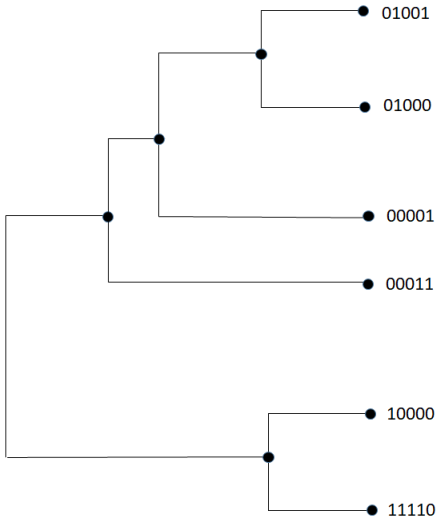


(b) árbol con raíz (*midpoint-rooting*) obtenido un 70 % de las ejecuciones.

Figura 5.6: árboles reconstruidos con $neighbor - joining$.

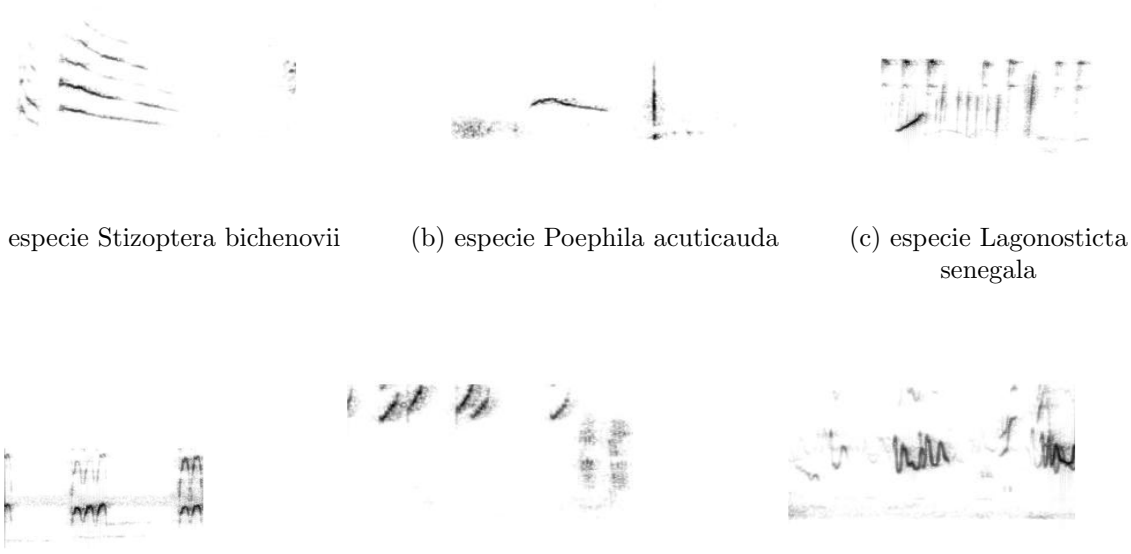


(a) árbol obtenido en un 50 % de las ejecuciones.



(b) árbol obtenido en un 30 % de las ejecuciones.

Figura 5.7: árboles reconstruidos con *clustering jerárquico*.



(a) especie Stizoptera bichenovii (b) especie Poephila acuticauda (c) especie Lagonosticta senegala

(d) especie Lonchura striata (e) especie Uraeginthus bengalus (f) especie Amandava subflava

Figura 5.8: Seis sonogramas de representantes de las distintas especies de aves.

leccionadas porque recientemente se comparó la organización filogenética de las mismas con la que emerge de comparar los cantos mediante la preselección de ciertas propiedades acústicas [15]. En esta tesis interesa por poner a prueba un método que no requiere de una pre especificación de propiedades para llevar a cabo la reconstrucción.

Para cada uno de estos se obtuvieron los sonogramas de sus cantos, pueden observarse en la figura 5.8. El dataset cuenta con 568 imágenes, de las cuales el 80 % fue utilizado para entrenar la red y el 20 % restante como validación. Además, los datos fueron aumentados aplicándoles transformaciones: rotaciones, translaciones, zoom.

En cuanto al modelo para encontrar embeddings que permitan representar los datos de sonogramas, la estructura de la red convolucional es similar a la utilizada con los datos artificiales. La única diferencia es que el resultado de la capa final tiene tres dimensiones en vez de dos.

- Convolución + Relu 64@10×10
- Max Pool 2×2
- Convolución + Relu 128@7×7
- Max Pool 2×2

- Convolución + Relu 256@4 × 4
- Max Pool 2 × 2
- Convolución + Relu 512@4 × 4
- Max Pool 2 × 2
- Capa Densa 512 + Relu
- Capa Densa 3 + Lineal
- $x^* \leftarrow 0{,}99 \tanh(\|x\|)\frac{x}{\|x\|}$

La red neuronal siamesa utiliza dos redes convolucionales, cada una de las cuales toma un sonograma y produce un punto x^* dentro de la bola de Poincaré en 3 dimensiones.

Al igual que con los datos artificiales, el aprendizaje se realizó con una serie de minibatchs de 32 pares de imágenes donde la mitad correspondía a sonogramas de la misma clase y la otra mitad a sonogramas de distintas clases. El modelo requirió mayor cantidad de epochs para converger, fue entrenado durante 2500 epochs o iteraciones.

El optimizador de Adam fue seleccionado con un parámetro de tasa de aprendizaje de 10^{-6} y en la función de pérdida, definida en 3.16, m fue definida como 25.

Para cada una de las 10 ejecuciones del modelo se varió la inicialización de los pesos de la red. Los núcleos de las capas convolucionales tomaron valores iniciales según una distribución normal de media 0 y desvió estándar 0.1. El componente de sesgo se inicializó con una distribución normal con media de 0.5 y desvió de 0.1.

Para calcular la métrica de *accuracy* definida en 3.18 se realizaron 100 tareas de verificicación cada 100 iteraciones del modelo. Al igual que para los datos aritificiales, se eligieron al azar 10 clases de sonogramas, permitiendo repetición, como conjunto de soporte. Se eligió un sonograma de prueba y midiendo la distancia de Poincaré en el espacio de embedding se decidió si la verificación fue exitosa o no.

El resultado promedio obtenido fue de 88,3 % de *accuracy* para los datos de entrenamiento y 86,3 % para los de validación. En la tabla 5.2 pueden verse los valores de *accuracy* para cada una de las 10 ejecuciones del modelo.

Los embeddings obtenidos en el espacio de Poincaré para cada ejecución pueden observarse en las figuras 5.9 - 5.18.

A partir de la representación de los datos en este espacio, se calcularon los centros de Chebyshev de cada una de las clases usando la distancia de Poincaré. Utilizando esta matriz de distancias de Poincaré entre los centros, se aplicó el algoritmo $neighbor - joining$.

Los árboles sin raíz obtenidos por las 10 ejecuciones pueden observarse en la figura 5.19. Posteriormente, a cada uno de los árboles se le calculó la raíz utilizando midpoint

Accuracy		
	Entrenamiento	Validación
1	0.91	0.89
2	0.93	0.91
3	0.85	0.82
4	0.89	0.87
5	0.88	0.86
6	0.89	0.86
7	0.79	0.81
8	0.9	0.87
9	0.91	0.89
10	0.88	0.85

Tabla 5.2: *accuracy* para los datos de entrenamiento y validación de las 10 ejecuciones del modelo.

rooting. Los resultados pueden encontrarse en la figura 5.20. El 60 % de las ejecuciones del modelo obtuvo el mismo árbol (*b*), su estructura presenta similitudes con el árbol filogenético. En primer lugar, *Poephila acuticauda* y *Stizoptera bichenovii* son vecinos, al igual que en el árbol filogenético. *Amandava subflava* y *Lagonosticta senegala* derivan del mismo ancestro en común para ambos árboles. La principal diferencia entre estos dos árboles se encuentra en la ubicación de *Lonchura striata* y *Uraeginthus bengalus*. La primera comparte padre con *Lagonosticta senegala* en el árbol de cantos mientras que en el filogenético el ancestro común más inmediato es el padre de *Poephila acuticauda* y *Stizoptera bichenovii*. En cambio, para el árbol de cantos, *Uraeginthus bengalus* ocupa el lugar que tiene *Lonchura striata* en el filogenético.

Es destacable que el árbol filogenético se diferencia del árbol generado a partir del comportamiento en la posición de una especie: *Uraenginthus bengalus*. Esta especie tiene algunas características que ameritan una revisión de los datos empleados. En particular, la hembra canta, aunque un canto menos complejo que el del macho. Cabe la posibilidad de que nuestra base haya contado para esta especie con ejemplos de cantos de hembra, lo cual comprometería la clasificación. A futuro sería interesante revisar esta posibilidad.

Los dos restantes árboles de cantos tienen estructuras con más diferencias con respecto al filogenético. En todas las ejecuciones *Poephila acuticauda* y *Stizoptera bichenovii* son vecinos, y *Lagonosticta senegala* y *Lonchura striata* también lo son.

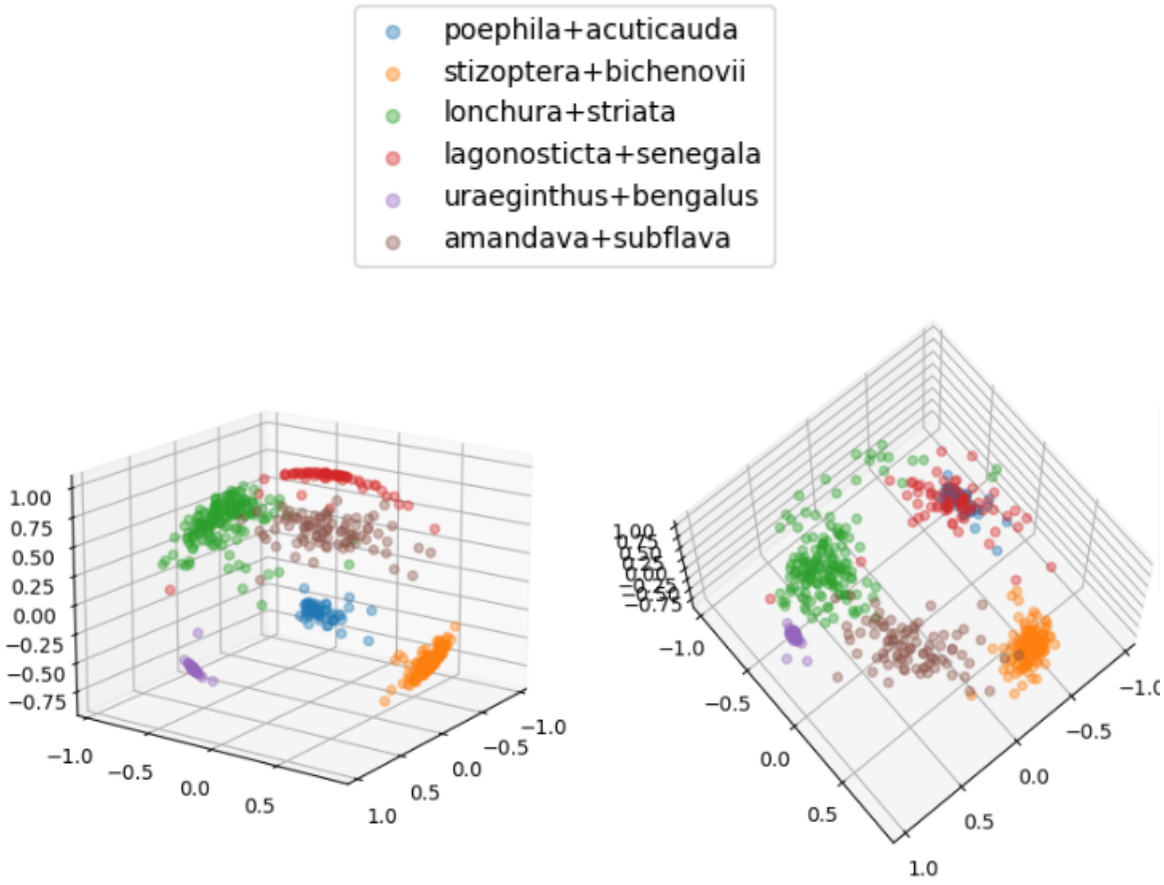


Figura 5.9: Ejecución 1: Embeddings de los sonogramas.

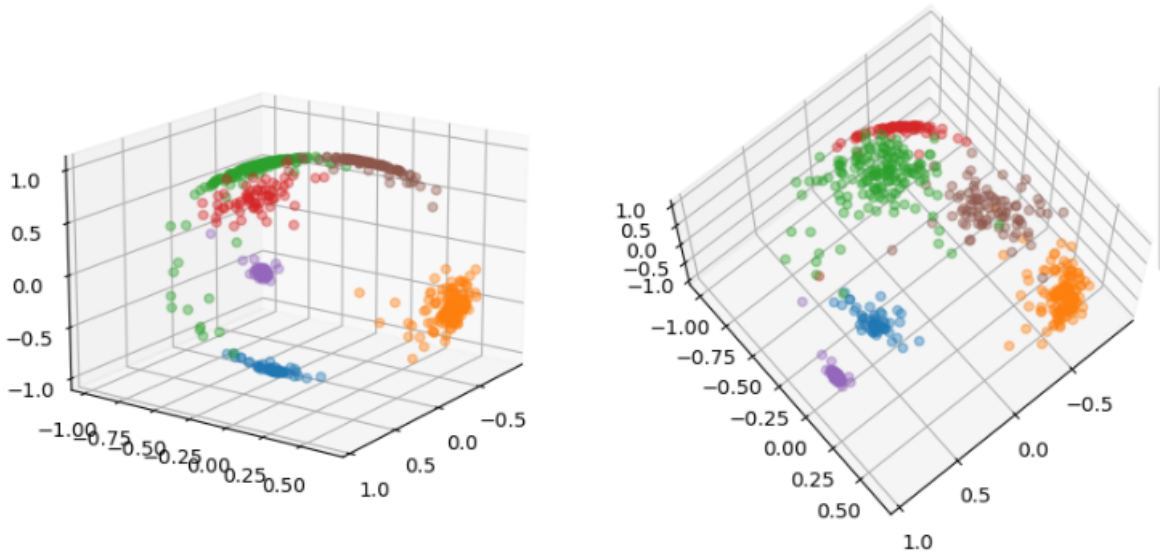


Figura 5.10: Ejecución 2: Embeddings de los sonogramas.

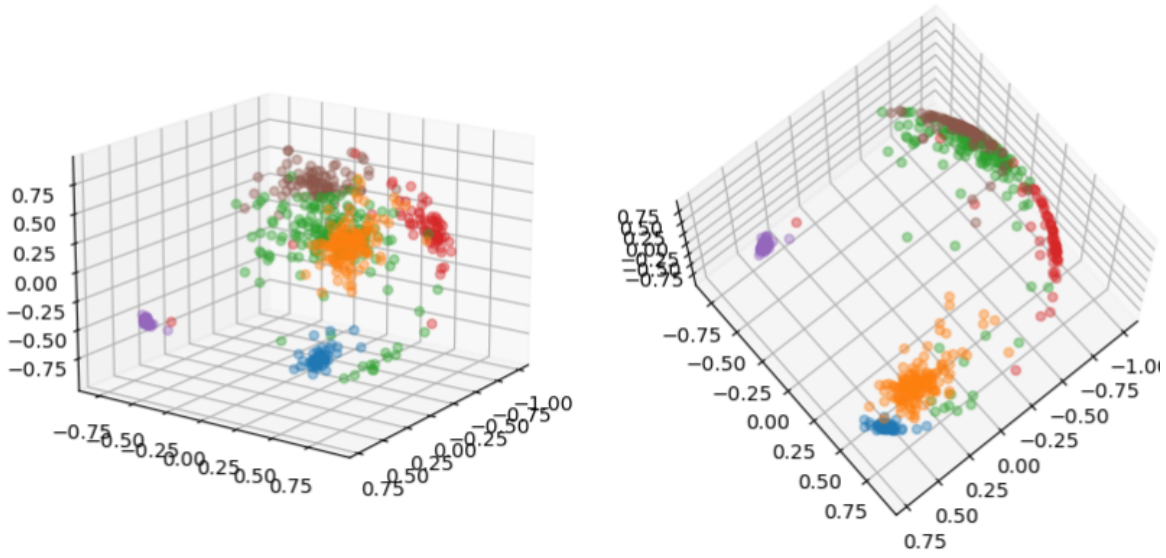


Figura 5.11: Ejecución 3: Embeddings de los sonogramas.

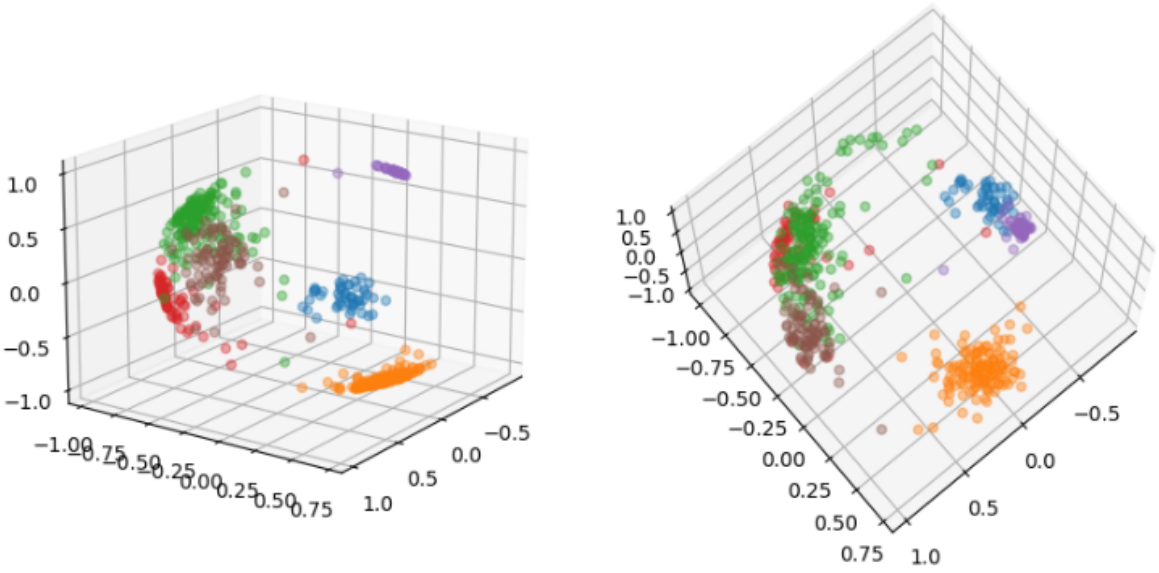


Figura 5.12: Ejecución 4: Embeddings de los sonogramas.

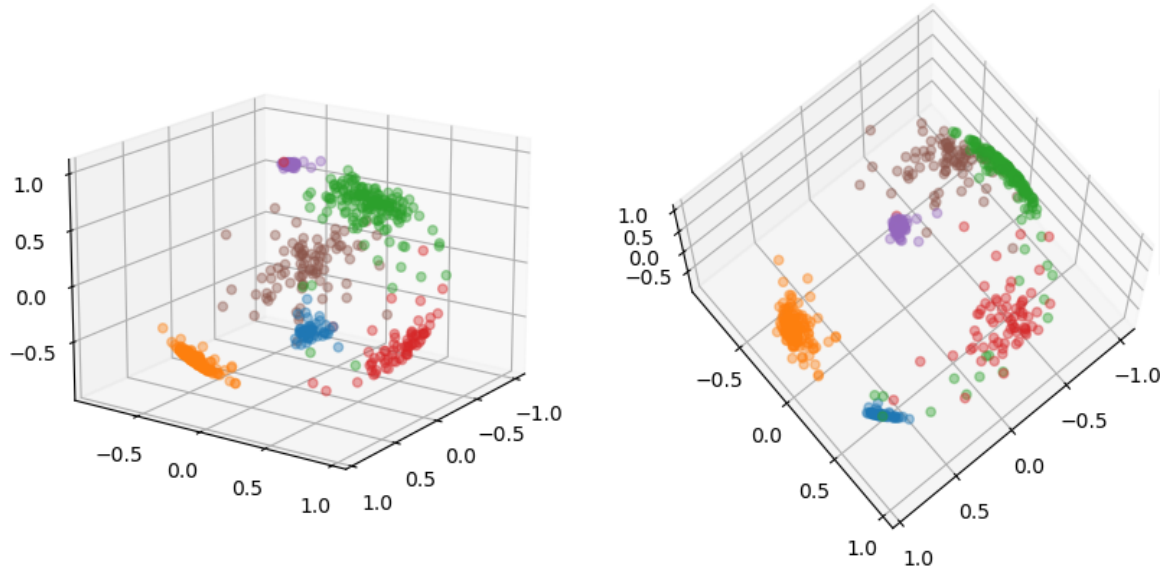


Figura 5.13: Ejecución 5: Embeddings de los sonogramas.

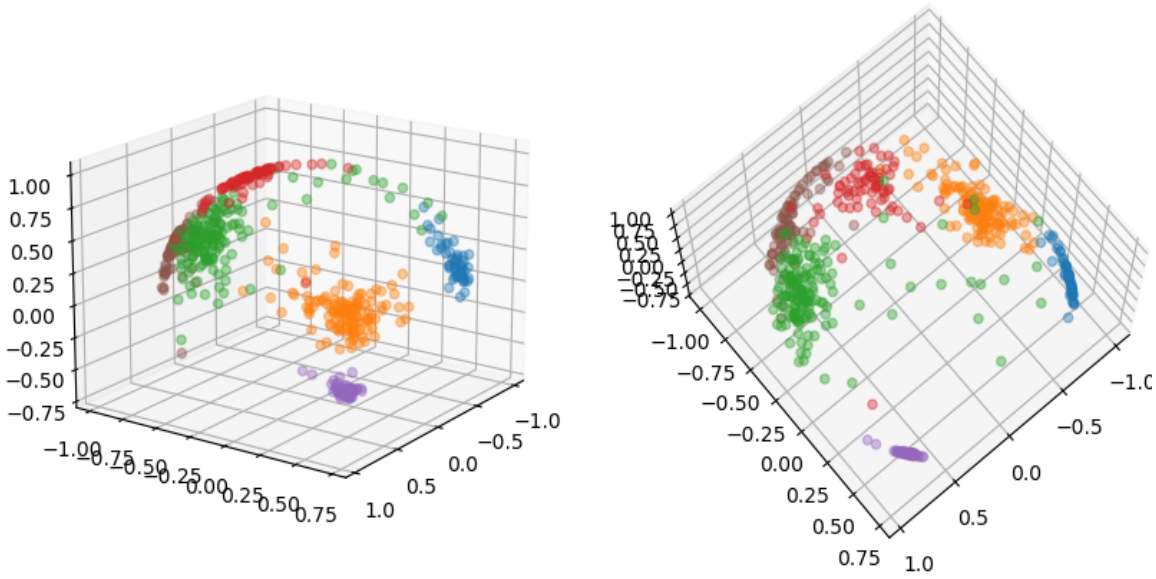


Figura 5.14: Ejecución 6: Embeddings de los sonogramas.

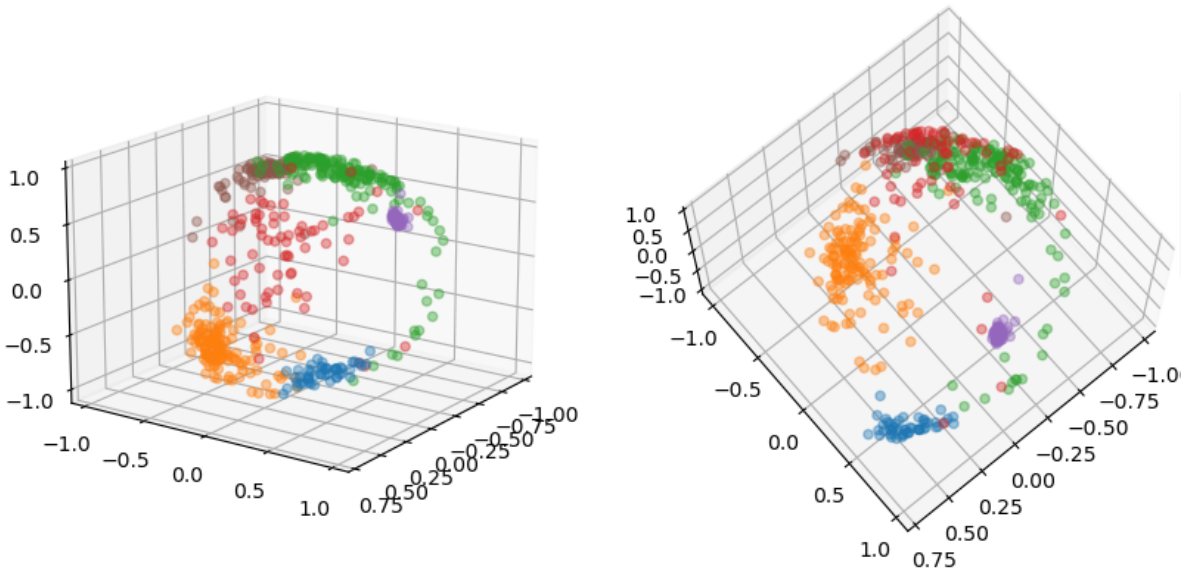


Figura 5.15: Ejecución 7: Embeddings de los sonogramas.

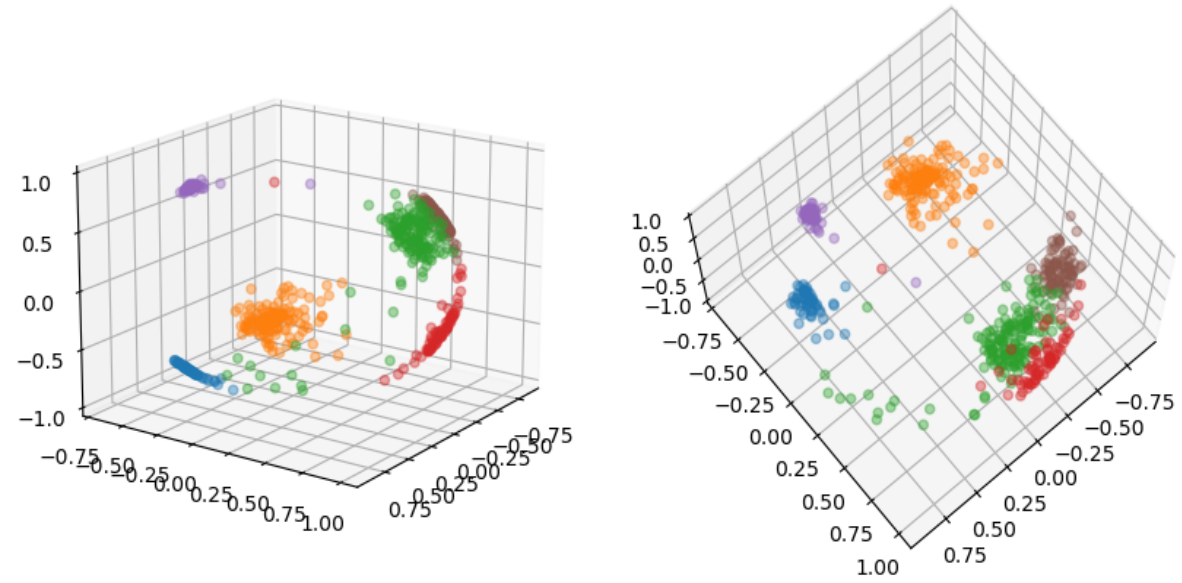


Figura 5.16: Ejecución 8: Embeddings de los sonogramas.

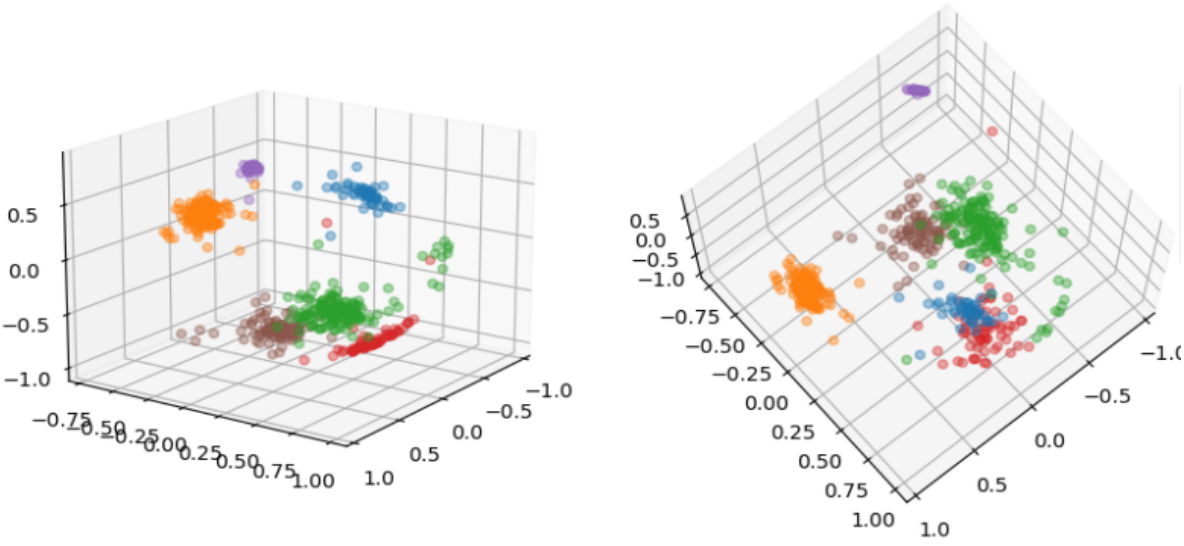


Figura 5.17: Ejecución 9: Embeddings de los sonogramas.

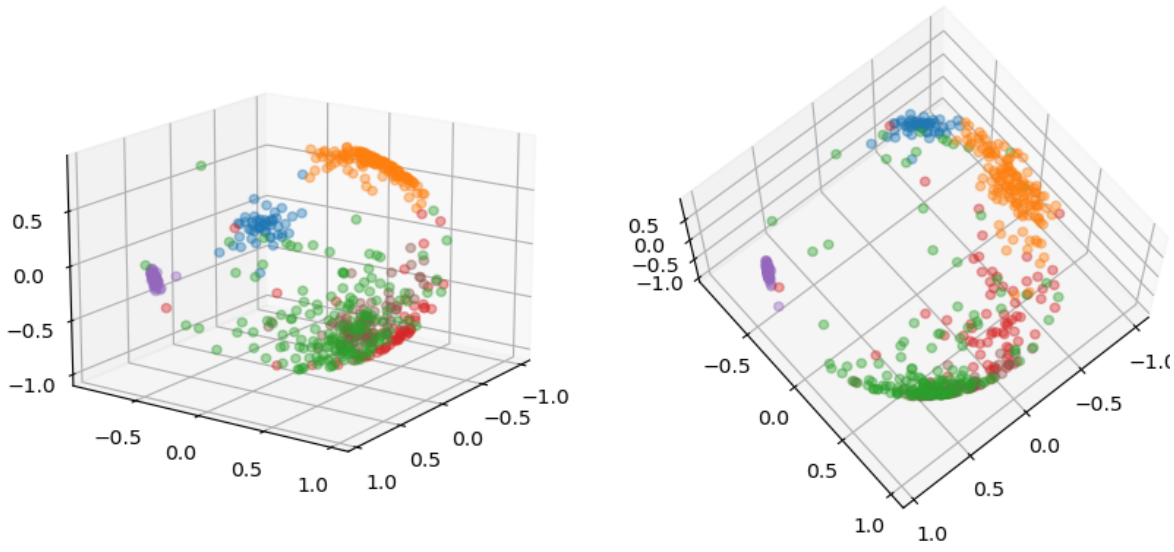
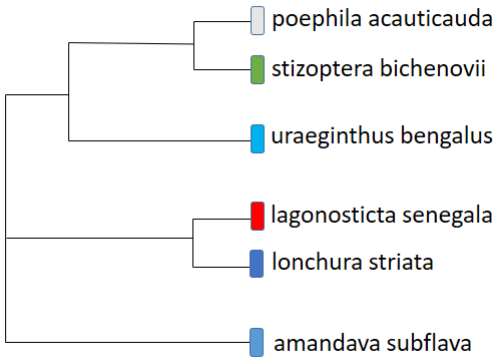
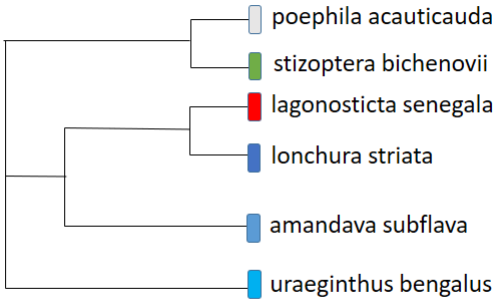


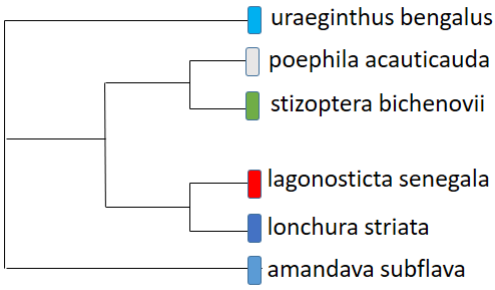
Figura 5.18: Ejecución 10: Embeddings de los sonogramas.



(a) árbol sin raíz generado a partir de sonogramas para el 70 % de las ejecuciones.

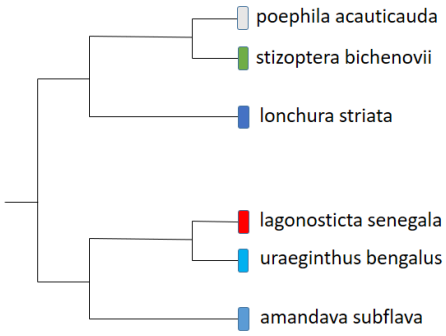


(b) árbol sin raíz generado a partir de sonogramas para el 20 % de las ejecuciones.

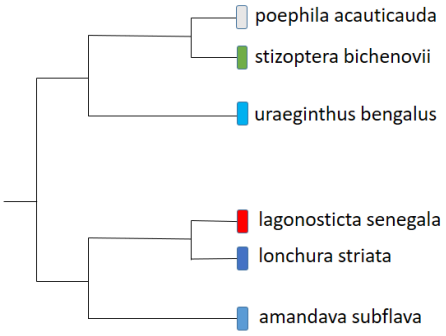


(c) árbol sin raíz generado a partir de sonogramas para el 10 % de las ejecuciones.

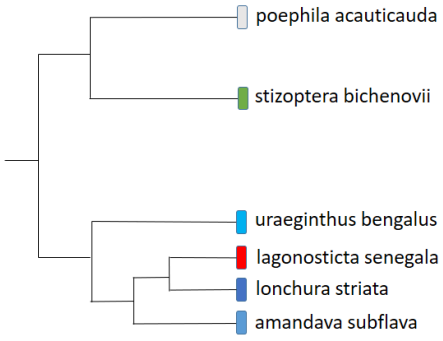
Figura 5.19: Árboles generados por $neighbor - joining$.



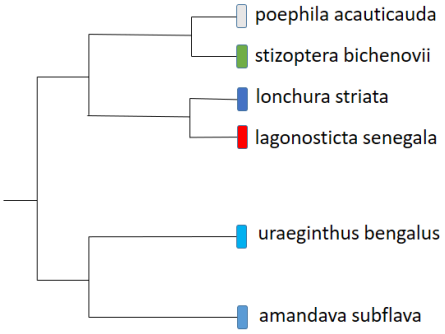
(a) árbol filogenético.



(b) árbol con raíz generado el 60 % de las ejecuciones.



(c) árbol con raíz generado el 30 % de las ejecuciones.



(d) árbol con raíz generado el 10 % de las ejecuciones.

Figura 5.20: (a) árbol filogenético de las especies de aves. $(b) - (d)$ árboles con raíz a partir de *midpoint rooting* y *neighbor-joining*.

Capítulo 6

Conclusiones

El objetivo principal del trabajo fue desarrollar una herramienta computacional que permitiera reconstruir de forma automática un árbol a partir de una manifestación fenotípica expresable en términos de una imagen que tuviera una estructura jerárquica implícita. Para desarrollar, se estudiaron diversos temas, tales como el modelo de Poincaré, redes neuronales convolucionales, redes siamesas y algoritmos de reconstrucción de árboles filogenéticos.

Los embeddings obtenidos en el disco de Poincaré permitieron diferenciar las distintas clases tanto para las matrices artificiales como para los sonogramas. Aplicando *clustering jerárquico* los resultados no fueron los esperados, en cambio aplicando $neighbor - joining$ con midpoint rooting sobre los datos artificiales se logró recuperar el árbol implícito en un 70 % de las ejecuciones. El embedding de sonogramas generado por la red siamesa permitió reducir la dimensión de los datos de cantos de manera exitosa, obteniendo un 86,3 % de accuracy. El árbol más común reconstruido con $neighbor - joining$ a partir de los embeddings presenta algunas similitudes con el generado a partir de datos genéticos. Las especies *Poephila acauticauda* y *Stizoptera bichenovii* comparten un ancestro en común inmediato para cada uno de los árboles reconstruidos a partir de la fenotipia como la genotipia. Sin embargo, *Lagonosticta senegala* y *Uraeginthus bengalus* no son vecinos en ninguno de los árboles de las diez ejecuciones, mientras que sí lo son en el árbol filogenético. La correspondencia entre los sonogramas de vocalizaciones y la genética puede ser que no sea la esperada para el grupo de pájaros analizados.

Es posible que habiendo aprendizaje de por medio, la filogenia no coincida con el árbol de la fenotipia, aunque las particularidades antes señaladas de *Uraeginthus bengalus* (específicamente, que pueda haber en nuestra base cantos de hembras), sugieren la necesidad de revisar la base empleada.

En el futuro, teniendo en cuenta el éxito del algoritmo para las matrices artificiales, sería positivo explorar su comportamiento en conjuntos de datos con relaciones jerárquicas más complejas que representen un mayor desafío para el modelo. Posibles ejemplos

a los cuales puede aplicarse la herramienta son los modelos computacionales de vida artificial descriptos por Richard Dawkins en [4], con los cuales generó imágenes que surgen a partir de un proceso de mutación y supervivencia de ciertas características.

Puede ser interesante aplicar el algoritmo estudiado en esta tesis a una mayor variedad de especies para reconstruir su filogenia a partir de características fenotípicas y compararla con árboles evolutivos que surgen a partir de datos genéticos. Es necesario profundizar el estudio para dar respuesta a la pregunta inicial sobre qué tan similar es un árbol de semejanzas, en un conjunto de especies que sí aprenden, reconstruido a partir de la fenotipia con respecto al árbol filogenético. La herramienta desarrollada en la tesis intenta ser un aporte a la hora de responder esta y otras preguntas similares.

Bibliografía

[1] Yoshua Bengio. *Learning Deep Architectures for AI*. 2009. DOI: 10.1561/2200000006.

[2] Roberto Bistel, Alejandro Martinez y Gabriel B. Mindlin. "An analysis of the persistence of Zonotrichia capensis themes using dynamical systems and machine learning tools". En: *Chaos, Solitons and Fractals* 165 (2022), pág. 112803. ISSN: 0960-0779. DOI: https://doi.org/10.1016/j.chaos.2022.112803.

[3] Jane Bromley et al. "Signature verification using a "Siamese"time delay neural network". En: Morgan Kaufmann Publishers Inc., 1993.

[4] Richard Dawkins. "The Evolution of Evolvability." En: *Artificial Life. The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems* (dic. de 2003). DOI: 10.1016/B978-012428765-5/50046-3.

[5] John A. Rhodes Elizabeth S. Allman. *Mathematical models in biology. An introduction*. 1.ª ed. Cambridge University Press, 2003.

[6] Xeno-canto Foundation y Naturalis Biodiversity Center. *Xeno-canto*. https://xeno-canto.org/.

[7] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. MIT Press, 2016.

[8] Marvin Jay Greenberg. *Euclidean and Non-Euclidean Geometries: Development and History*. 2007.

[9] Raia Hadsell, Sumit Chopra y Yann Lecun. "Dimensionality Reduction by Learning an Invariant Mapping". En: 2006, págs. 1735-1742. DOI: 10.1109/CVPR.2006.100.

[10] Richard Kenyon James W. Cannon William J. Floyd y Walter R. Parry. "Hyperbolic Geometry". En: *Flavors of Geometry* 31 (1997).

[11] Diederik P. Kingma y Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[12] Maximilian Nickel y Douwe Kiela. *Poincaré Embeddings for Learning Hierarchical Representations*. 2017. arXiv: 1705.08039 [cs.AI].

[13] Pavel Pevzner Phillip Compeau. *Bioinformatics Algorithms: An Active Learning Approach 1*. 2.ª ed. Active Learning Publ., 2015.

[14] Simon J. D. Prince. *Understanding Deep Learning*. The MIT Press, 2023.

[15] Moises Rivera et al. "Machine learning and statistical classification of birdsong link vocal acoustic features with phylogeny". En: *Scientific Reports* 13 (mayo de 2023). DOI: 10.1038/s41598-023-33825-5.

[16] Naruya Saitou y Masatoshi Nei. "The neighbor-joining method: a new method for reconstructing phylogenetic trees." En: *Molecular biology and evolution* 4 4 (1987), págs. 406-25.

[17] Carel Ten Cate. "Birdsong and evolution". En: dic. de 2004, págs. 296-317. DOI: 10.1016/B978-012473070-0/50013-X.