

# Predicción de futuros sitios de inserción de las secuencias Alu en el ADN

TITULACIÓN:  
Máster en  
Inteligencia Artificial

Curso académico:  
2020-21

Lugar de residencia:  
Palma de Mallorca,  
Noviembre/2021

Alumno/a:  
**Bardají Cusó, Jordi**

D.N.I: 43000601P

Director:  
**Lebrón Aguilar, Ricardo**

Convocatoria:  
**Primera**

Orientación:  
Aplicación a un  
problema  
Créditos: 12



**Universidad**  
Internacional  
de Valencia

# Resumen

El presente Trabajo Final de Máster en Inteligencia Artificial lleva por título: “Predicción de sitios de inserción de Alu”.

El trabajo trata de la aplicación de la inteligencia artificial en la predicción de los lugares dónde se insertan unas determinadas secuencias de ADN (Ácido Desoxirribonucleico) que sólo se encuentran en el genoma humano y de primates.

Las secuencias que son la base de este estudio se denominan Alu y pertenecen a una familia más general de secuencias que se conocen como transposones. El término transposón indica que son secuencias que no permanecen estáticas en los cromosomas sino que se generan copias de ellas, que luego se reproducen en distintos lugares del ADN, cada cierto tiempo. El presente trabajo pretende averiguar si se puede predecir dónde puede insertarse una Alu en el ADN cuando la Alu inserte una copia. O, lo que es prácticamente lo mismo, si es posible determinar si en una secuencia determinada es posible que se inserte una copia de una Alu o no.

Aunque se han usado redes neuronales convolucionales (CNN) anteriormente en investigación del ADN, lo novedoso de este trabajo es que se basa en la conversión de las secuencias de texto que forman el ADN en imágenes en escala de grises que luego son analizadas por la CNN. De esta manera se transforma el problema inicial de secuencias de texto en un problema de clasificación binaria de si una imagen corresponde a una secuencia dónde se insertó (o se puede insertar) una Alu o no.

Las fuentes de datos serán ficheros “**.fasta** o **.fa**” i ficheros “**.rmsk**”. Estos son formatos estándar de dominio público, usados en Genómica, que contienen información de las secuencias de bases (A,C,T,G) de cada cromosoma. Hay organismos que tienen copias de estos ficheros alojados en páginas web. Ocupan unos cuantos GB cada uno. Se descargan comprimidos.

En concreto, el fichero **fasta** usado es el que corresponde a la versión de 2013 del genoma humano (**hg38.fa**) (la última versión en el momento de empezar el TFM).

El fichero **.rmsk** es un fichero que contiene (entre otras) información de qué tipo de secuencia hay desde una posición a otra y a qué cromosoma pertenece.

En resumen, la estrategia que se usará será elegir fragmentos de ADN combinando la información contenida en los ficheros .fasta y .rsmk para entrenar la red neuronal convolucional habiendo convertido las secuencias en imágenes de escala de grises. Estas imágenes serán las que alimentarán a la CNN.

Como lenguaje de programación se ha usado python y la librería tensorflow tensorflow-gpu para implementar la CNN y realizar el tratamiento de los datos. Otras librerías como PIL se han usado para tratar las imágenes. El entorno de programación usado ha sido jupyter-notebook.

El presente trabajo **no** se basa en trabajo previo realizado por el tutor, un proyecto anterior o similar. Y es un trabajo de aplicación como se puede deducir del título. Aunque debido a su campo de aplicación (la genómica) ha requerido un mínimo estudio.

## Índice

Resumen.....	2
Objetivos.....	7
Introducción a los aspectos biológicos del problema.....	7
La Genómica.....	7
El ADN.....	7
Los cromosomas.....	9
El estudio de la naturaleza del problema.....	10
Estado del arte de la tecnología empleada.....	12
Frameworks específicos para genómica.....	12
Qué trabajos hay relacionados con la temática del estudio.....	12
Tipología de los ficheros de datos.....	13
El fichero fasta.....	14
El fichero rnsk.....	15
Desarrollo del proyecto.....	16
Establecer la longitud de las secuencias que se van a usar en el estudio.....	16
Obtención y depuración de los datos.....	18
Transformación de los datos a imágenes.....	21
Conjuntos de datos que van a ser usados.....	22
Dataset de Secuencias completas. Partición de los datos en datasets de Train, Validation y Test.....	23
Test Dataset.....	23
Train y Validation datasets.....	24
Dataset de secuencias de flanco izquierdo y del derecho. Partición de los datos en datasets de Train, Validation y Test.....	24

Creación de la estructura del modelo de la CNN.....	25
El detalle del Base Model.....	26
Detalle del Top Model:.....	27
Summary del modelo. Número de parámetros entrenables.....	28
Problemas derivados de la magnitud del volumen de los datos y la complejidad del modelo.....	29
Resultados.....	32
Dataset de secuencias con dos flancos unidos. Secuencias completas.....	32
Resultados para Train y Validation.....	32
Resultados con el dataset de Test.....	33
Dataset de secuencias con únicamente flanco izquierdo.....	34
Resultados con el dataset de Test.....	36
Dataset de secuencias con únicamente flanco derecho.....	37
Resultados con el dataset de Test.....	38
Valoración y discusión de los resultados.....	39
Características aplicables a los tres conjuntos de datos.....	39
Comparación del modelo entre los distintos conjuntos de datos.....	40
Conclusiones y desarrollos futuros.....	41
Conclusiones.....	41
Futuros desarrollos.....	42
Eliminar las clases a priori negativas que no lo son.....	42
Determinación de la longitud de influencia de los flancos que determina la inserción de la Alu en un cierto punto.....	43
Balancear las clases positivas con los supuestas negativas.....	43
Modo de construcción de las imágenes a partir de las secuencias de bases.....	44

Aumentar la riqueza de los datos de entrada convirtiendo las secuencias de bases en lo que los datos originalmente son: datos en las tres dimensiones espaciales....44

Anexos.....46

Anexo 1. Solicitud de área temática para el Trabajo Fin de Máster.....46

Anexo 2. Solicitud para la presentación y defensa del Trabajo Fin de Máster....47

Anexo 3. Informe de autorización del Trabajo Fin de Máster.....48

Anexo 5. Referencias Bibliográficas.....49

## Objetivos

El objetivo de este TFM es tratar de averiguar si es posible predecir dónde se podría insertar una secuencia de tipo Alu en una eventual migración.

El primer objetivo fue, como siempre en todos los casos tratados en el máster: estudiar el problema en su conjunto y comprender su naturaleza, para luego seguir con estudiar los datos, estructura, qué datos hay que recolectar, depuración a realizar de estos, ... y, por último, decidir la estrategia a seguir y aplicarla, para luego obtener conclusiones.

## Introducción a los aspectos biológicos del problema

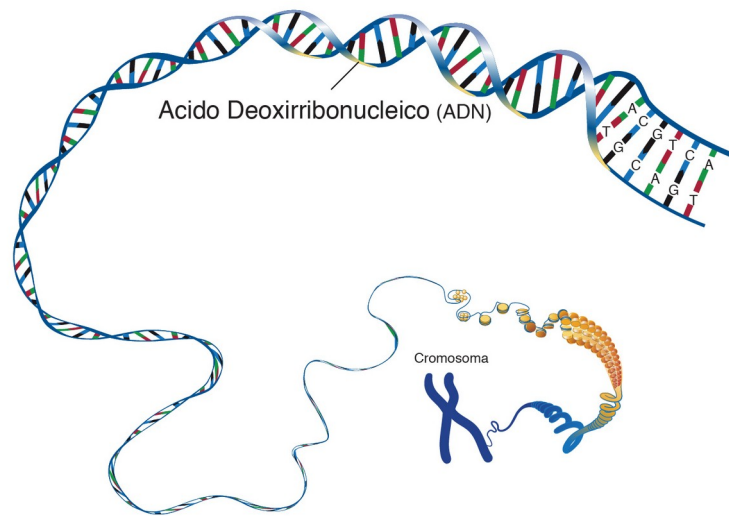
En este apartado se explican los conceptos generales necesarios ligados a la temática del trabajo en el aspecto biológico. En problemas de esta naturaleza un conocimiento del tema a estudiar es imprescindible. Como mínimo en el caso que nos ocupa, ya que nos puede ayudar a determinar cual es el mejor enfoque.

### La Genómica

El campo de aplicación será la IA aplicada a la Genómica (Shaw et al., s. f.). La Genómica es la parte de la biología que estudia el genoma. La Genómica es el estudio del ADN en su conjunto. La Genética es el estudio de la herencia de los genes de una generación a otra y como afecta a los individuos.

### El ADN

La base del genoma es el ADN (*ADN (Ácido Desoxirribonucleico)* | *NHGRI*, s. f.). El ADN se compone de secuencias de cuatro bases (moléculas) representadas por las letras A, C, T y G, las cuáles son las abreviaturas de **A**denina, **C**itosina, **T**imina y **G**uanina. Las distintas combinaciones en cadenas de estas cuatro bases componen lo que se conoce como el ADN. El ADN está presente en la mayoría de los seres vivos e incluso en algunos virus.



*Figura 1: ADN. "Courtesy: National Human Genome Research Institute": <https://www.genome.gov>*

En el trabajo que nos ocupa nos centraremos en el genoma humano. La palabra genoma deriva de gen(*Gen* | *NHGRI*, s. f.). Un gen es una parte del ADN (un conjunto de bases) que condiciona la biología de los seres vivos. Hace un tiempo se creía que todo el genoma tenía información de relevancia en la biología y que era como un libro de instrucciones. En realidad hoy se sabe que sólo un 2%, aproximadamente, del genoma humano, lo forman Genes. De buena parte del resto se desconoce su función, hay incluso partes del ADN que se suponía que eran “basura”. Una parte de esta supuesta “basura” la forman las Alu(Deiningering, 2011). Recientemente se empieza a sospechar que esta “basura” ha sido crucial en la evolución. En el genoma humano hay unos 20.000 genes.

El concepto de gen se está volviendo difuso: La longitud de los genes varia de unas pocas cientos a más de 80.000 bases. Los genes se asociaban a proteínas, pero se sabe que el cometido de lo llamado gen no es, en muchos casos, tan directo.

En los genes se pueden producir cambios. Que alguna de las bases que lo forman sea substituida por otra. Es lo que se denomina una mutación.



Una mutación en la estructura de un gen modifica sus efectos sobre el ser en el que se produce. Hay mutaciones que pueden ser beneficiosas y otras que no lo son y dan lugar a enfermedades o ser propenso a ellas. Las Alu y su movimiento a lo largo del genoma pueden ser responsables de buena parte de estas mutaciones (Payer et al., s. f.). De ahí la importancia de su estudio.

## Los cromosomas

El ADN está compuesto de cromosomas (*Chromosome*, s. f.). Cada cromosoma se compone de dos cadenas (secuencias) de bases, unidas entre sí, que son complementarias. Formando lo que se conoce como la doble hélice del ADN en cada cromosoma.

Si nos referimos al ADN humano, éste está repartido en 23 pares de cromosomas. Los primeros 22 cromosomas (se llaman cromosomas autosómicos) están duplicados (clonados, ya que son copia exacta uno del otro), formando los 22 primeros pares. El último par (el 23) puede estar formado, en principio, o por dos cromosomas del tipo X o por un cromosoma del tipo X y otro del Y. Esta última pareja condiciona el sexo. Si un humano tiene el par XX, es un humano de sexo femenino y si tiene el par XY es de sexo masculino. El ADN humano está contenido dentro del núcleo de las células del organismo.

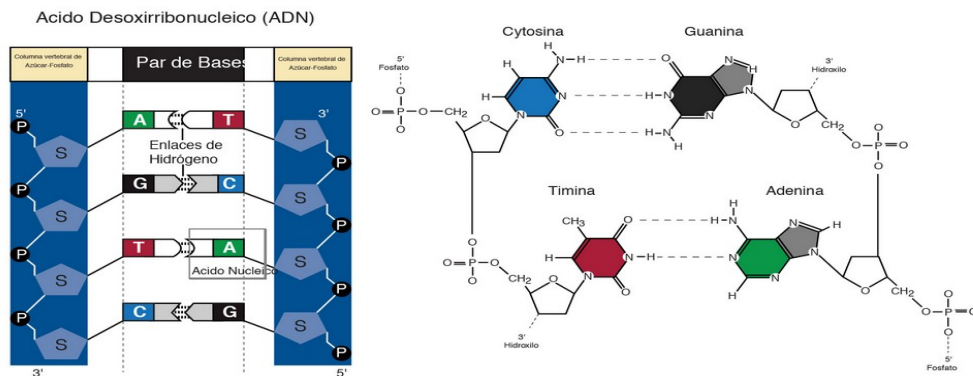


Figura 2: Detalle de la composición del ADN. "Courtesy: National Human Genome Research Institute": <https://www.genome.gov/>

Los cromosomas se nombran así: cromosoma 1, cromosoma 2, cromosoma 3, ..., cromosoma 22 y cromosoma X o/y cromosoma Y en el caso de los cromosomas responsables del sexo.

Si nos adentramos en lo que es un cromosoma, tenemos dos hebras complementarias que están formadas por grupos azúcar y un fosfato formando las cadenas laterales. Cada grupo fosfato está unido a una base. Al decir que las dos hebras son complementarias nos referimos a que si en una hebra en la posición que ocupa el lugar n hay una A, en la otra hebra en la misma posición n hay una T y viceversa, análogamente sucede con las bases C y G. Cada unión de una base con su complementaria en la otra hebra se hace por un tipo de interacción química que se denomina “puentes de hidrógeno”.

En cada hebra, en principio, puede haber cualquier secuencia en cualquier orden e incluso repeticiones de una misma base. Con esta disposición se consiguen larguísimas cadenas de bases. Estas cadenas forman la secuencia de bases que se unen a la cadena complementaria, una cadena y su complementaria se unen para formar un cromosoma.

## **El estudio de la naturaleza del problema**

El ADN es una secuencia de bases unidas, hay secuencias con significado biológico conocido (por ejemplo, lo que llamamos genes) y otras que no (o como mínimo su papel en la biología no es conocido).

En nuestro caso, el tipo de secuencias que son el objeto de este trabajo (las Alu), van insertando copias de sí mismos en el genoma en sucesivas generaciones (aproximadamente cada 200) dividiendo una secuencia en dos partes e insertándose en medio (Chen et al., 2007).

A partir de lo expresado en la figura siguiente y teniendo en cuenta los condicionantes biológicos se puede suponer que:

- Si una secuencia (Alu, en nuestro caso) inserta una copia en un punto, esta inserción estará condicionada por los alrededores de este punto (naturaleza de los flancos izquierdo y derecho).

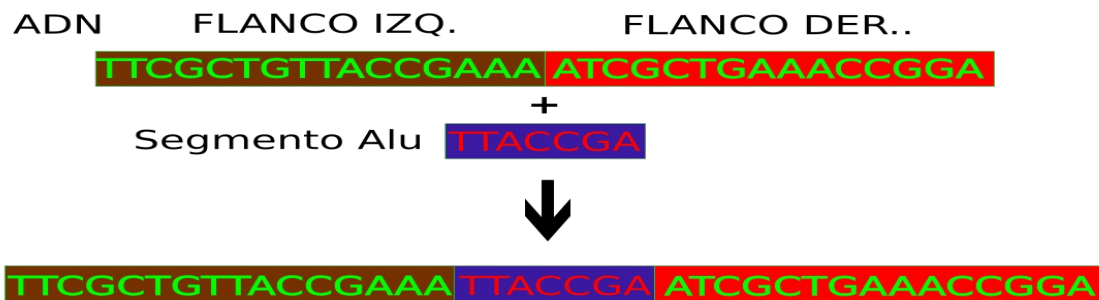


Figura 3: Inserción de Alu. Imagen propia.

- Al estudiar los alrededores de los puntos de inserción conocidos, ya que estos condicionan, según nuestra hipótesis, la inserción o no en este punto de una Alu, conseguiremos extraer características de los alrededores que nos definan que ahí podrá situarse una Alu o no.
- Los casos negativos son secuencias en las que no se ha insertado una Alu. Debe señalarse que este hecho, no nos garantiza que esta secuencia que tomamos como negativa no sea una candidata a una futura inserción.

Con estas premisas, estamos derivando el problema a un problema de clasificación binaria. Le proporcionamos secuencias en las que sabemos que se ha insertado una Alu (**ConAlu**) y secuencias en las que, hasta el momento, no se ha insertado una Alu (**SinAlu**).

Este planteamiento tiene, no obstante, un pero y hay que tenerlo en cuenta; ya que no sabemos a ciencia cierta que las copias de Alu se han insertado en todos los sitios posibles en los que se pueden insertar. Como no todas las inserciones tienen por qué haberse producido, al elegir lugares en los que sabemos que no se ha introducido una Alu hasta el momento, estamos suponiendo que en los puntos contenidos en las secuencias negativas no se va a insertar. Pero no podemos asegurar que la secuencia tomada como negativa sea efectivamente una secuencia negativa (y no pueda ser una secuencia candidata): **Resultados pasados no implican resultados futuros en los casos actuales de no inserción.**

En el apartado **Futuras mejoras** se analizarán posibles soluciones a esta circunstancia.

# Estado del arte de la tecnología empleada

## Frameworks específicos para genómica

En la primera reunión el tutor nos nombró los frameworks más usados en genómica. Estos son: Keras\_dna(*Home - Keras\_dna Documentation*, s. f.), Janggu(*Tutorial — Janggu 0.10.2 documentation*, s. f.), Kipoi, Selene y TensorSignatures, entre otros.

Se hizo un estudio preliminar de Keras\_dna y Janguu.

Se desechó su uso. La razón de desechar el uso de los frameworks fue que con los formatos de los datos, contenidos en los ficheros .rmsk y .fasta, que se usaron en el trabajo, la extracción de los datos no resulta demasiado complicada. Resultó más rentable hacer código adhoc que aprender el funcionamiento de los frameworks. Como se verá a lo largo del trabajo, la principal complicación no fue la extracción de los datos sino el volumen de estos.

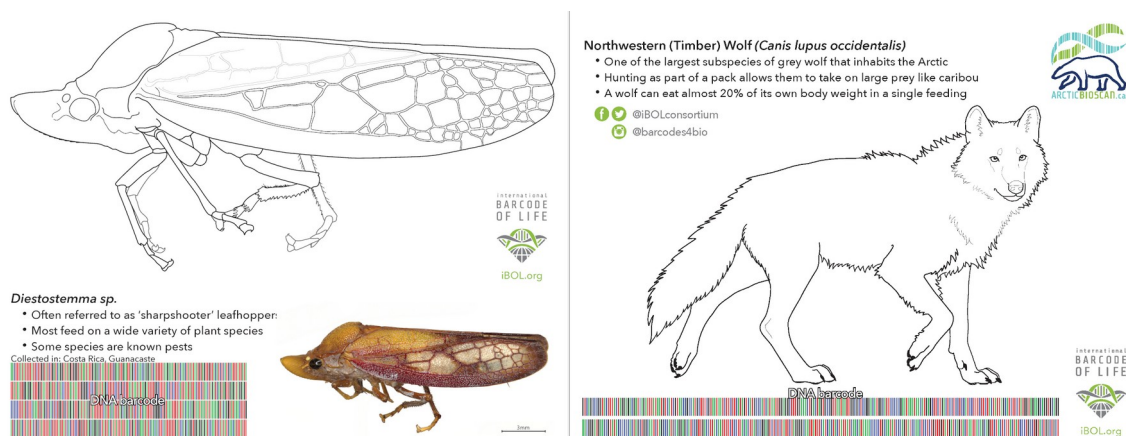
## Qué trabajos hay relacionados con la temática del estudio

En las búsquedas con las palabras: DNA “alu” Convolutional Neural Network, en las referencias que aparecen en ninguna de ellas se trabaja con imágenes como tales para alimentar la CNN para hacer predicciones de inserción, al menos yo no las he encontrado.

Es verdad que se hace uso de la conversión de secuencias de bases del ADN a códigos de barra (DNA Barcoding(*DNA Barcoding - an overview | ScienceDirect Topics*, s. f.; Kress & Erickson, 2008)) pero no con el propósito de predecir lugares de inserción sino de proveer de un método de fácilmente automatizar el reconocimiento automático de secuencias para detectar patrones que sugieran cáncer por ejemplo o alguna patología (Gunasekaran et al., 2021).

También existen algunas aproximaciones que convierten secuencias de ADN a imágenes para encontrar similitud entre secuencias para la detección de anomalías genómicas (Konkel et al., 2015).

En lo que respecta a la predicción de sitios predilectos para la inserción de Alu u otras secuencias, en el ADN. No he encontrado ninguno. Es una aplicación novedosa.



*Figura 4: Imágenes RGB de ADN barcodes representativos de especies. (@iBOLConsortium - ibol.org)*

Por otro lado, existe la arquitectura denominada DanQ (Quang & Xie, 2016), que usa bloques convolucionales con bloques previos de LSTM. DanQ usa LSTM previo para que las inputs del modelo sea con la secuencia de bases directamente. Creo que así se pierde el sentido espacial para privilegiar el propósito de las LSTM que es poder prever secuencias futuras en la cadena a partir de las pasadas. Además, DanQ se usa para la cuantificación de la función de las secuencias, no es este nuestro objetivo.

## Tipología de los ficheros de datos

Se procederá a explicar la estructura de los ficheros **fasta** («FASTA Format», 2021; *FASTA format description*, s. f.) y **rsmk** (*Schema for RepeatMasker - Repeating Elements by RepeatMasker*, s. f.-a) (*Schema for RepeatMasker - Repeating Elements by RepeatMasker*, s. f.-b) y el uso que se les ha dado. Hay otros tipos de formatos pero con los citados se puede obtener toda la información necesaria para el propósito del estudio.

## El fichero fasta

El formato fasta es un fichero de texto plano en el que está cromosoma por cromosoma la secuencia que contiene el ADN o ácidos nucleicos en general (aunque también se usa para almacenar la secuencia de aminoácidos en proteínas). Es un texto construido a base de cadenas inmensas de texto formadas únicamente, en principio, con los caracteres A,C,T,G (en mayúsculas o minúsculas) en el caso del ADN.

Si el fichero fasta fuera un libro, podríamos asimilar cada capítulo del fichero fasta a la cadena de bases de cada cromosoma. En cada capítulo hay dos párrafos. Cada párrafo estaría formado por una única línea.

Hay capítulos de los que no se conoce exactamente a qué cromosoma pertenecen o, que si se sabe, no se conoce su ubicación exacta en él. Son párrafos aparte. Estos “capítulos” se han descartado en el trabajo.

La información de cada cromosoma dentro del fichero **fasta** está recogida en dos líneas:

- La primera línea: que tiene la siguiente estructura: “>”+nombre del cromosoma o nombre de la secuencia.
- La segunda línea: la cadena de texto (la secuencia de bases). En la cadena de texto, alguna vez, aparece el carácter “N”. Significa que no se sabe o no se ha podido generalizar que hay en esta posición. Como cada cromosoma tiene dos hebras complementarias, sólo aparece una de las hebras del cromosoma.

Los ficheros **fasta** pueden presentarse con varias extensiones. La extensión **.fa** es una de las más comunes. A causa de su tamaño, por razones obvias, está en la red comprimido. El fichero fasta usado se puede encontrar en esta dirección entre otras: <http://hgdownload.cse.ucsc.edu/goldenpath/hg38/bigZips/hg38.fa.gz>.

El tamaño del fichero comprimido es de 938 MB. El fichero se denomina “hg38.fa.gz”. Lo que significa que corresponde a **human genome** versión **38** correspondiente a la compilación del año 2013 y que al descomprimirlo obtendremos un fichero hg38.fa

(formato fasta). El tamaño del fichero descomprimido (hg38.fa) ocupa en mi ordenador unos 3,3 GB.

## El fichero rmsk

El fichero **rmsk** (RepeatMasker) con el que se ha trabajado en el estudio, contiene la situación de secuencias.

Es básicamente un fichero de coordenadas de secuencias del **human genome** versión **38** que corresponde al fichero **hg38.fa** que ya fue descrito anteriormente. Conviene también descargarse el fichero rmsk.sql. Se descargan de:

- <https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/rmsk.txt.gz>
- <https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/rmsk.sql>

El rmsk.txt.gz ocupa 147 MB comprimido y al descomprimirlo se obtiene un fichero de texto separado por tabuladores (rmsk.txt). El aspecto de las dos primeras líneas es el siguiente:

```
(base) Jordi@jordi-portatil: ~/Documents/MasterIAVIU/TFM/datasets$ head --lines=2 rmsk.txt
585  463  13  6  17  chr1  10000  10468  -248945954  +  (TAACCC)n  Simple_repeat  Simple_repeat  1  471  0  1
585  3612  114  215  13  chr1  10468  11447  -248944975  -  TAR1  Satellite  telo  -399  1712  483  2
```

El fichero rmsk.sql contiene la sentencia sql de creación de la tabla rmsk. Una vez creada la estructura de la tabla rmsk. Se procedió a importar los datos del fichero rmsk.txt en la tabla mediante una sentencia sql de importación.

Los campos de la tabla rmsk de interés para este estudio son los que hacen referencia al cromosoma en el que se ha localizado la secuencia, el tipo de secuencia (para este estudio elegiremos las de tipo Alu), las coordenadas de inicio y fin de la secuencia dentro del cromosoma (empezando por 0 la primera posición) y en la hebra en la que se ha localizado la secuencia.

SELECT \* FROM `rnsk`

☐ Perfiles [Edita en línea] [Edita] [Desarrolla l'SQL] [Crea el codi PHP]

1 > >> Nombre de files: 25 Filtra les files: Cerca aquesta taula Ordena per la clau: Res

+ Opcions

bin	swScore	milliDiv	milliDel	milliIns	genoName	genoStart	genoEnd	genoLeft	strand	repName	repClass	repFamily	repStart	repEnd	repLeft	id
585	484	251	132	0	chr1	11504	11675	-248944747	-	L1MC5a	LINE	L1	-2382	395	199	3
585	3612	114	215	13	chr1	10468	11447	-248944975	-	TAR1	Satellite	telo	-399	1712	483	2
585	463	13	6	17	chr1	10000	10468	-248945954	+	(TAACCC)n	Simple_repeat	Simple_repeat	1	471	0	1
585	239	294	19	10	chr1	11677	11780	-248944642	-	MER5B	DNA	hAT-Charlie	-74	104	1	4
585	318	230	37	0	chr1	15264	15355	-248941067	-	MIR3	SINE	MIR	-119	143	49	5
585	18	232	0	19	chr1	15797	15849	-248940573	+	(TGCTCC)n	Simple_repeat	Simple_repeat	1	52	0	6
585	18	137	0	0	chr1	16712	16744	-248939678	+	(TGG)n	Simple_repeat	Simple_repeat	1	32	0	7
585	239	338	129	0	chr1	18906	19048	-248937374	+	L2a	LINE	L2	2942	3104	-322	8

*Figura 5: Primeros registros de la tabla rnsk después de la importación.*

Los campos de la tabla que usaremos son:

- **genoName:** Nombre del cromosoma.
- **genoStart:** coordenada de inicio de la secuencia.
- **genoEnd:** coordenada de fin de la secuencia.
- **strand:** En qué hebra se encuentra la secuencia. (“+” o “-”).
- **repFamily:** de qué familia es la secuencia. En nuestro caso elegiremos las secuencias con valor **Alu**.

## Desarrollo del proyecto

### Establecer la longitud de las secuencias que se van a usar en el estudio

Teniendo en cuenta todo lo dicho hasta el momento para las secuencias **ConAlu**, si nuestra hipótesis es correcta, deberemos establecer qué longitud, a ambos lados del punto de inserción, usaremos como inputs para conseguir un modelo de clasificación.

Para ello, se hizo necesario un estudio de los datos disponibles. El fichero que contiene información de dónde ha habido la inserción de una Alu es el fichero rnsk. De este fichero se puede extraer la longitud de cada inserción a partir de las posiciones inicio y fin de la Alu y en qué cromosoma se ha producido, además de otros datos de interés, como ya se ha visto anteriormente.

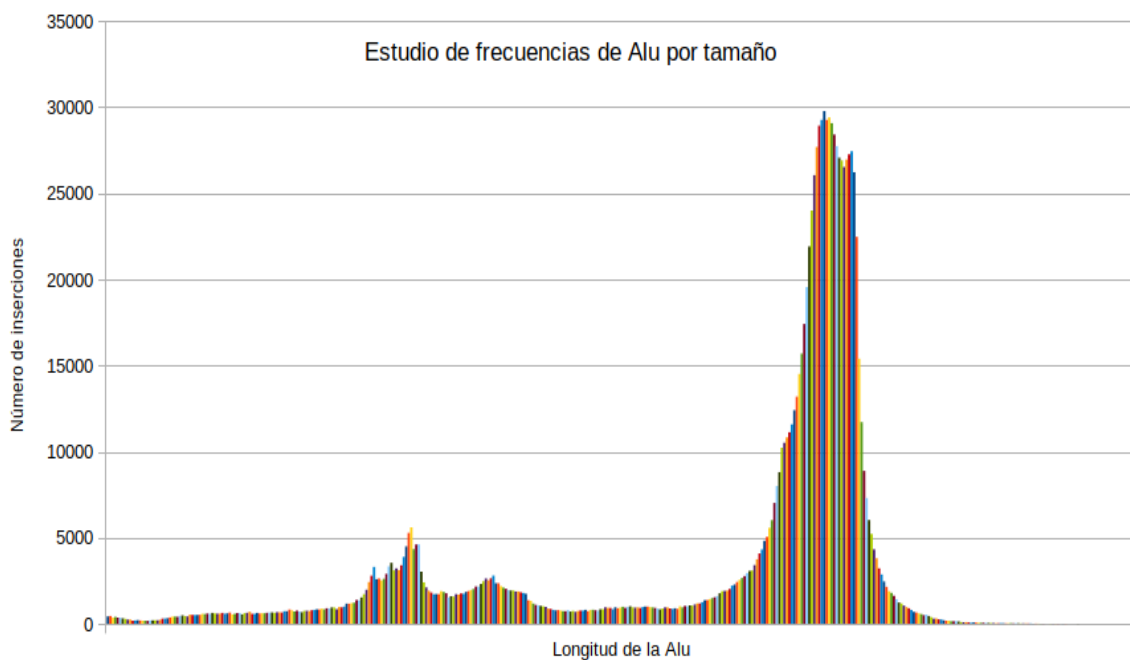
El análisis de las secuencias se ha realizado con consultas Sql de la tabla que se obtiene del fichero sql dump de rnsk.



Habiendo hecho el análisis de las frecuencias de longitudes se puede ver que la máxima longitud de las secuencias clasificadas como Alu es de 814 bases y la más pequeña de 11.

Hay una única secuencia de longitud máxima (814 bases), de secuencias de longitud 11 hay 433 ocurrencias. Si nos atenemos a la variabilidad en cuanto a longitudes podemos ver que hay 414 longitudes distintas de Alu. El espectro de longitudes va desde 11, aumentando de una en una, hasta 405. Luego hay ocurrencias de estas longitudes: 406, 407, 409, 410, 411, 413, 414, 415, 419, 420, 421, 426, 427, 429, 435, 439, 443, 700 y 800.

El gráfico de frecuencias asociado es:



*Figura 6: Número de inserciones de Alu de cada longitud. Elaboración propia.*

Las Alu se clasifican en biología por tipos, pero las Alu de igual tipo (según su clasificación biológica) no son de la misma longitud ni son iguales en cuanto a orden de bases. Tampoco son iguales las Alu de igual tipo y misma longitud. Lo cual elimina la posibilidad de hacer un estudio según longitudes o tipos de Alu por separado.

Siendo así, y imaginando el mecanismo de inserción de una Alu, parece bastante sensato pensar que, como ya se ha apuntado anteriormente, el mecanismo biológico de inserción en una posición determinada habrá estado condicionado por las bases que están alrededor de este punto tanto a izquierda como a derecha de él. Por esta razón inicialmente se decide que la longitud de los inputs a obtener será el doble de la longitud máxima de las Alu. Con esto presuponemos que el radio de influencia de las bases será, a lo sumo, a un radio igual a la longitud de la Alu de máxima extensión en lo que se refiere a la inserción de la Alu.

En nuestro caso, la longitud sería de  $814 \times 2 = 1628$ . Más adelante veremos que en realidad se despreciaron 28 bases y qué motivó esta decisión (14 bases iniciales a la izquierda y 14 a la derecha al final). Nos quedamos por tanto, con 1600 bases alrededor del punto de inserción de las Alu.

## Obtención y depuración de los datos

Una vez obtenidos los datos en bruto de todo el fichero hg38. Se procedió a la segmentación del fichero fasta hg38.fa en ficheros fasta individuales, uno por cada cromosoma. En cada fichero aparecen únicamente los caracteres “N” y “A”, “C”, “T” y “G”, pudiendo estar estos cuatro últimos caracteres en mayúsculas o minúsculas.

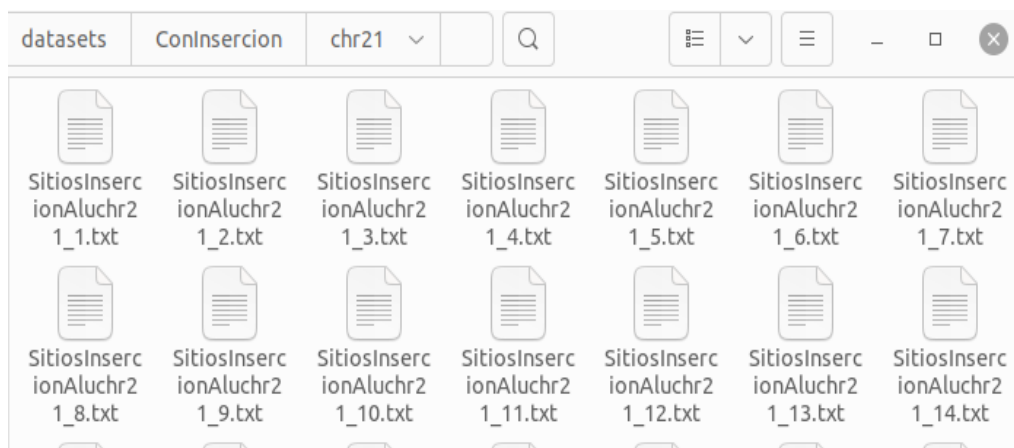
En cada uno de ellos se convirtieron a mayúsculas todos caracteres que representan las bases. Los ficheros obtenidos están en texto plano y tienen únicamente la secuencia de bases, se les ha eliminado la cabecera que indica el cromosoma. El nombre del cromosoma forma parte del nombre del fichero e indica a qué cromosoma pertenece la secuencia. Los ficheros obtenidos tienen nombres con esta tipología: Cleared\_fasta\_chrNN.fa. NN toma valores desde 1 a 22 y X o Y.

Habiendo establecido la longitud máxima de los datos que serán los inputs del estudio, se trata de obtener las secuencias de estos input. Se hizo de la siguiente manera:

- 1) Mediante consulta SQL aplicada sobre los datos proporcionados por el fichero sql dump rmsk (rmskchr1-22XY.txt) obtenemos el cromosoma, el punto de inicio de cada Alu y el punto final. Como en el fichero rmsk hay indicación de qué secuencia está en una hebra o en la otra, elegimos sólo las secuencias marcadas como hebra ‘+’.

- 2) Obtenemos datos con este formato: "chr7","6328874","6328885" para cada una de las inserciones de Alu. Ponemos estos tres valores en una lista. Es evidente, que la primera cadena indica el cromosoma, la segunda la posición de inicio de la Alu y la tercera la posición final de la Alu.
- 3) Ahora por una parte tenemos:
  - 1) La secuencia de cada cromosoma en un fichero aparte cuyo nombre identifica a qué cromosoma pertenece la secuencia.
  - 2) Información del tipo "chr7","6328874","6328885" (nombre cromosoma, posición de inicio de la Alu, posición final de la Alu) extraída del rnsk en una lista (listaposiciones).
  - 3) Con estas premisas podemos obtener las secuencias de interés.
  - 4) Creamos dos carpetas: ConInsercion y SinInsercion. En cada una de ellas pondremos los ficheros de secuencias dónde se ha insertado la Alu y secuencias dónde no se ha insertado una Alu, diferenciando una carpeta por cada cromosoma.
- 4) **Obtención de secuencias ConAlu.** Con lo que tenemos hasta ahora podemos conseguir el contenido de las secuencias antes de que la Alu se insertara.
  - 1) Obtenemos los ficheros de secuencia que obtendrán 814 caracteres antes de la posición inicial y 814 caracteres después de la posición final. Hemos eliminado las bases que forman la Alu. Tenemos la secuencia de bases previa a la inserción de la Alu. Estas serán la secuencias **ConAlu**.
- 5) **Obtención de secuencias SinALu.** Ahora se trata de elegir las secuencias en las que no ha habido inserción de Alu. Para elegir las secuencias sin inserción de Alu, debemos tener en cuenta los siguientes condicionantes:
  - 1) Debemos asegurarnos de que no solapan con ninguna Alu.
  - 2) Se toma como método de elección de secuencias sin inserción previa de Alu secuencias que estén entre dos secuencias Alu.

- 3) La longitud de las secuencias **SinAlu** deben tener una longitud de 1628 bases (en principio).
- 4) El procedimiento seguido para elegir éstas es:
  - 1) Asegurarse que entre la posición final de una Alu y la posición inicial de la siguiente hay espacio suficiente para extraer una secuencia de bases de la longitud deseada.
  - 2) Si se pueden extraer, entonces obtenemos la posición intermedia entre la posición final de una Alu y la posición inicial de la siguiente, sino se descarta la secuencia.
  - 3) Extraemos la mitad de las bases antes de la posición intermedia entre dos Alu y la mitad después.



*Fi*

*gura 7: Detalle de la estructura de directorios y ficheros para el cromosoma 21 de secuencias en las que ha habido inserción de Alu.*

- 6) Ahora tenemos dos directorios de secuencias de texto que corresponden a las secuencias **ConAlu** (en el directorio *ConInsercion*) y **SinAlu** (en el directorio *SinInsercion*).

De estos directorios obtendremos las particiones de Train, Validation y Test.

Figura 8: Fichero de ejemplo con 1628 bases del cromosoma 17 del directorio ConInsercion.

## Transformación de los datos a imágenes

Tenemos las secuencias de caracteres que nos interesan en ficheros identificados con el nombre del cromosoma y si corresponden a una inserción o no.

En este punto es dónde decido convertir cada uno de los ficheros de secuencias a ficheros de imágenes (Brownlee, 2019). Me apercibo de que en realidad se trata de extraer las características que tienen las secuencias **ConAlu** y **SinAlu**. ¿Qué extrae mejor características de datos de este tipo? una CNN (Razzaq, 2021) (red neuronal convolucional). En realidad, el planteamiento elegido nos conduce a un problema como los de clasificación binaria de imágenes de perros (**ConAlu**) y gatos (**SinAlu**).

Cómo una CNN necesita como datos de entrada valores numéricos, convierto los valores de cada una de los caracteres que definen las bases A, C, T, G a numéricos de todas las secuencias obtenidas anteriormente. Substituyo cada ocurrencia del carácter “A” por 0., cada ocurrencia de “C” por 60., “T” por 125 y “G” por 255.

Se descartan las secuencias que contienen un carácter “N”. Además, como el objetivo es obtener imágenes, se eliminan los 14 primeros caracteres de la secuencia y los 14 últimos. Así se obtienen ficheros de 1600 caracteres. Que serán convertidos a imágenes de 40x40 píxeles.

Seguro que al lector no se le ha pasado por alto que la longitud máxima de las Alu es de 814, es verdad, pero sólo hay una inserción de una Alu de esa longitud y de 700 también hay sólo 1 ocurrencia. Por tanto, se decide prescindir de los 28 caracteres

que, eventualmente, podrían condicionar que el modelo reconociera la Alu de 814. Se podría ampliar la longitud de las secuencias a una longitud que fuera mayor de 1628, pero parece que no valdría la pena, además a mayor tamaño, más tiempo de computación. Estableciendo el tamaño de las secuencias a 1600, se estaría, a lo sumo, en el peor de los casos, despreciando una única secuencia Alu, la de longitud 814. El riesgo es mínimo.

Convierto estas cadenas de valores numéricos a imágenes y las guardo en ficheros.  
Obtengo 2.669.833 imágenes.

AATGCTTTTTTTAGAAATCACTATACCTCTTGGTAGGTTACTTAATTTTTAGCATAAACCACCAAAATCCCATGTCAATTGTTAAAAATGGATCTATTCAAGCGTCACAGTATAA  
ATTTATGGTTTCCCGAGAAATCAGAGAAAGTGGTTGTAAGAGGTAATTTAAAAATATTGCGCAATAAATCCCATGTTTAAAAATATCTTAATTAAGTGAATGTGAAAGTAGTATTA  
ATAACAGCATGTGTTCAAAGAAAATCTTAAGGCGAACATAAAAAATTTACTTGAAGGCGAATATTACAAGATTCAAGAGTTGCTGATCTCTTTGAACAGGTTTCGATGGGAGAT  
GATGATCTCTGTGATCTGTAGTAGAGAGCTCGAATTAACAAGTATCAGAAACCAAGCAGAGATAAGGCTCAATGATGAAATAGCCTATTTCCGTGCTGTGCTGGACCTAGGCTTTTG  
GGGATAGGCAGGCCTGATACAGAAAGTCGATGTAGCAAGAAGGAATGAAGGAAAAAGATGAAGAGACAAAAACATGCTGACACTGTCAATTTGTTCTCAAAATGTCAAAAATAAAAT  
TCAAAATGAAATCGATTTTTTAATTAATAAAATCTTCTGAGAAATGTTTATTTTCCCTCCAGACATAAGTAACATAAGGCAAGGAATCATATTTACCTCTGATATGCTTCAC  
ATAGAATTTGCCACTTTTTATCAGCAATATTAATAATGATTATGTGATGAATAGCTTTTCAGTCTATACGCCCTGAGAGCAGGCAAGAAAAATTTATCAAAAAACAAAAATGCTTCT  
CACTAATTAAGCCAAAAACAGCCTATGAAAAAGCAGCATCAATGATCTGTGTCACCAAGGCTAAGGCAACCCATTATGATTATTGAGGGGCTTATTTGGGTATAGAAGCATCAAGA  
ATTTATTCAAGCTCAAGTGGTTGTGGGCAAAAGCCAAAGAGGAAAAAGGATTTTTCACTTCAAGTCCAGCTATCATTTGTTTCCAACTCTGGAAGAACGAGCAAGAAATACAGAGGAAGAG  
AAAAAGAAAGAAATCTGGGAGAGGAACAAATTAAGCAGCAGATAGTTGTTTACTGAGAGAGACAAAAATCAGTGAAGAGATAGAGACATGAAGACCGCGCAGGTCATGTCATGAT  
ATGCACTGTGTGTGCTGATGTATGTCATGATGTGTGTGTGTGTGTCATGAGTGTGTGATGTGTGAGTGTGAGTGTGAGTGTGCAATCAAGAGACACAGCAGAGGCTCT  
CGGTGTGCAATGCGCGCTGACCTGCTGCTCGGAAGTACAAAAACCAAGCATATTTATTTATTCGAGCTCTGGAAGTACAAAAACAGCATCATTTATCATTTGTATTTGTTGAG  
ACGAGATCTCGCTTGTGCTGCAATCTCAGCTCAGCTGAGGCGGCAATCTCAGCTCAGTGAATCTCGCCTCCGCGCTCAAGTGAATTTCTCTGCTCAGCTCCCAAGTAGCTGG  
GATTACAGCGCACACACCATGCCAGCTAATTTTTGATTTTTTAGTAGAGACAGGGTTTACACATGTTGGACAGAGTGCTGTGTTCTCACTCTGCTGAGCGCCTGCTCTGGCC  
TCCC  
[[125. 0. ... 125. 60. 60.]]

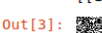
Out[3]: 

Figura 9: Secuencia de bases y la imagen obtenida en escala de grises.

## Conjuntos de datos que van a ser usados

De acuerdo con lo dicho anteriormente, la hipótesis de partida es que el lugar de inserción de una copia de una Alu dependerá de las bases en las proximidades del punto de inserción.

Se considera la posibilidad de que si el mecanismo biológico de inserción fuera direccional podría ser que sólo dependiera de las bases a la izquierda o a la derecha del punto de inserción. Razón por la cual, se usaron tres juegos de datos: los formados por las bases de sólo un flanco del punto de inserción y el otro formado por las bases a la izquierda y a la derecha del punto de inserción.

Se crean tres juegos de datasets que van a ser objeto de entrenamiento por separado:

- El dataset (secuencia completa) formado por secuencias que contienen los flancos izquierdo y derecho unidos.

- El dataset formado por secuencias que contienen los flancos izquierdo.
- El dataset formado por secuencias que contienen los flancos derecho.

## **Dataset de Secuencias completas. Partición de los datos en datasets de Train, Validation y Test**

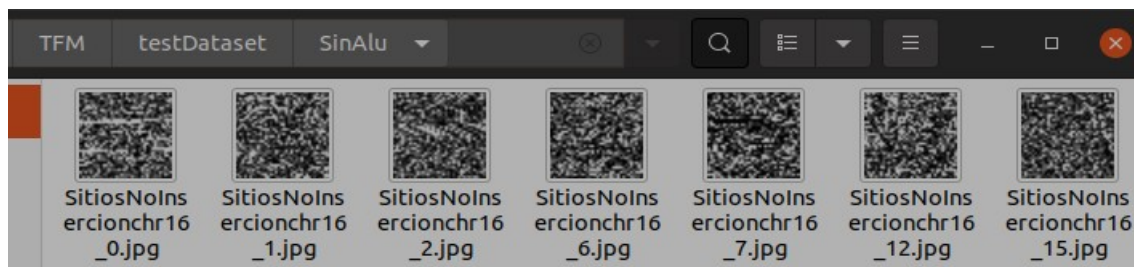
En este punto, se obtuvieron 2.669.833 válidas, en forma de secuencias de 1600 caracteres. Convertidas a imágenes de 40x40 píxeles. Se procedió a la construcción de los datasets de entrenamiento (Team, s. f.-a) (Brownlee, 2019).

### **Test Dataset**

Para determinar el dataset de Test se calculó qué cromosomas suponen un 10% de la longitud del genoma en humanos. Aproximadamente la suma de la longitud de los cromosomas 16, 17, 18 y 19 representan el 10% de la longitud total del genoma humano.

Elegimos las secuencias de los cromosomas 16, 17, 18 y 19. Dedicamos, por tanto, a Test, las secuencias de bases correspondientes a los cromosomas anteriormente citados:

- Son 398.606 secuencias, de las cuales:
  - 286.602 son secuencias **ConAlu**
  - 112.004 son secuencias **SinAlu**
- Creamos un directorio testDataset dónde alojamos las secuencias de test. En dos subdirectorios: ConAlu y SinAlu.



*Figura 10: Ejemplos de secuencias convertidas a imágenes de testDataset/SinAlu*

## Train y Validation datasets

Nos quedan 1.475.940 secuencias **ConALu** y 793.893 secuencias **SinAlu** que dividiremos en los datasets de Train y Validation.

Se decidió particionar las secuencias restantes en dos conjuntos: Un 80% de los datos para Train y el 20% restante para el dataset de Validation.

Para obtener los conjuntos de Train y Validation, ya que tenemos un directorio de secuencias **ConAlu** y otro **SinAlu**, que ya no incluyen las secuencias usadas para Test, se usó `image_dataset_from_directory` de `tensorflow.keras.preprocessing`.

Se usó `image_dataset_from_directory` de `tensorflow.keras.preprocessing` con los parámetros que aseguran que la partición de los datos es aleatoria y con una semilla (seed = 42) todo ello para asegurar la reproducibilidad de los resultados. Otros parámetros de interés son un batch size de 128 y `color_mode="grayscale"`.

El uso de la función `image_dataset_from_directory` permite no cargar todos los datos en memoria, los va cargando a medida que los necesita.

## Dataset de secuencias de flanco izquierdo y del derecho. Partición de los datos en datasets de Train, Validation y Test

Para conseguir los datasets de secuencias de sólo flanco izquierdo y las de sólo flanco derecho se hace uso de la función `crop` de `PIL.image` que divide imágenes. Se divide



cada imagen de secuencia completa (40 x 40) horizontalmente en dos mitades. Cada mitad es de dimensiones 20 de alto x 40 píxeles de ancho. De esta forma obtenemos las imágenes de sólo cada uno de los flancos por separado. La mitad superior de cada imagen corresponde a las secuencias del lado izquierdo y la mitad inferior a las secuencias del lado derecho.



*Figura 11: Imágenes de ejemplo de secuencias de flanco izquierdo o derecho. A simple vista indistinguibles. Tamaño de cada imagen (20 x 40).*

Se crean los directorios `datasetFlancoDerecho` y `datasetFlancoIzquierdo` y los de test (`testDatasetFlancoDerecho` y `testDatasetFlancoIzquierdo`) en los que se depositan las imágenes que corresponden a las secuencias del flanco derecho y las correspondientes al flanco izquierdo respectivamente, divididos todos en las carpetas **ConAlu** y **SinAlu**.

## Creación de la estructura del modelo de la CNN

Se crea un modelo con la API Sequential (Team, s.f.-b) de Keras con bloques convolucionales en el Base Model y capas densas en el Top Model.

Las capas convolucionales usadas en la arquitectura son del tipo conv2D. Se podría haber usado capas conv1D, sí. Pero sólo si se creyera que la naturaleza del problema es secuencial, pero se descarta que sea esta sea su naturaleza. Si se piensa en

términos biológicos, el ADN es una secuencia espacial y podría ser que la inserción se produjera por una interacción por unas ciertas bases distribuidas espacialmente alrededor del punto de inserción. Por eso se asimiló las secuencias a imágenes, para intentar darle al problema una orientación espacial a los datos que se nos presentan, en principio, como una secuencia de 1 dimensión (una cadena de caracteres del alfabeto A,C,T,G).

La primera capa lo único que hace es un cambio de escala: divide todos los valores por 255. de manera que ahora todos los valores van desde 0 a 1. Esto se hace para que los cálculos sean con números menores y conseguir reducir la memoria consumida y más rapidez en los cálculos. Este paso de cambio de escala podría haberse hecho en el paso previo de conversión de letras a valores numéricos, pero no hubiera podido visualizar las imágenes. Hubiera tenido que hacer lo mismo en sentido inverso para visualizar las imágenes de las secuencias de bases. Y de esta forma, fue posible el uso de la función **image\_dataset\_from\_directory**. Complementariamente al uso de esta función se usó el método **prefetch()** de tensorflow que mejora el tiempo de ejecución preparando los datos antes de entrar en cálculo. Dicha función devuelve un eagerTensor.

## El detalle del Base Model

El Base model está formado por tres bloques convolucionales.

Cada bloque convolucional tiene 2 capas Convolucionales de tipo conv2D con el mismo número de filtros en todas las capas del bloque, como se ha mostrado en el máster. Con padding a same para procesar todos los datos de la entrada al pasar los inputs por el filtro.

Las capas conv2D del 1º bloque tienen **filtros = 64** en cada capa convolucional.

Las capas conv2D del 2º bloque tienen **filtros = 128** en cada capa convolucional.

Las capas conv2D del 3º bloque tienen **filtros = 256** en cada capa convolucional.

La estructura general de los bloques convolucionales es la siguiente:

- `model.add(Conv2D(filters=filtros,  
kernel_size=(3,3),activation='relu',padding='same',  
input_shape1=(40,40,1)))`
- `model.add(BatchNormalization())`
- `model.add(Conv2D(filters=filtros,  
kernel_size=(3,3), padding='same', activation = 'relu'))`
- `model.add(MaxPooling2D(pool_size=(2,2)))`
- `model.add(BatchNormalization())`
- `model.add(Dropout(0.25))`

Como se puede observar después de cada capa Conv2D:

- Hay una capa de MaxPooling para reducir la dimensionalidad (reducción de los parámetros a calcular). del mapa de activación que sale de la capa conv2D. Como he usado maxpooling nos quedamos con los valores máximos.
- Otra capa de BatchNormalization para hacer que los pesos sean menores a la salida de cada neurona después de cada minibatch.
- Una capa de dropout. En este caso lo que conseguimos es aleatoriamente tener en cuenta (“conectar con la siguiente capa”) únicamente un % de las neuronas de esa capa. EL parámetro de la función dropout indica el % de neuronas de capas desactivadas.

### Detalle del Top Model:

Después del base model, del cual salimos con datos del tipo matrices apiladas (tensores), entramos en el Top model con una capa Flatten. Ésta tiene por objeto convertir los datos a una dimensión. Con los datos que salen de esta capa ya “aplanados”, alimentaremos la siguientes capas densas. En ellas iremos bajando la dimensión de los datos de una capa a la siguiente hasta llegar a la capa que nos clasifica las entradas.

---

<sup>1</sup> input\_shape sólo aparece en la primera capa del primer bloque convolucional.

El top model queda así:

- Una primera capa Flatten que ya hemos descrito.
- Le siguen dos bloques con capas densas que siguen el siguiente patrón:
  - `model.add(Dense(NumeroNeuronas), activation='Relu')`
  - `model.add(BatchNormalization())`
  - `model.add(Dropout(0.5))`

**NumeroNeuronas = 256** en la primera y **NumeroNeuronas = 64** en la segunda.

Después de cada capa dense, le sigue un BatchNormalization y un Dropout con el mismo propósito que tenían en los bloques convolucionales. En el Top model no tiene sentido el Maxpooling ya que no hacemos convoluciones.

- Por último, está la última capa. Una capa Dense de 1 neurona con función de activación sigmoid que es la recomendada normalmente en clasificación binaria. Esta capa nos da la probabilidad entre 0 y 1. Si la probabilidad es menor o igual a 0.5 se obtiene se clasifica el dato como **clase 0 (ConAlu** en nuestro caso), si la salida es  $> 0.5$  entonces la entrada se clasifica como **clase 1 (SinAlu)**.

Las opciones de compilación son: el **optimizer=Adam**, **loss= binary\_crossentropy** y **metrics = binary\_accuracy**. El optimizer es la función elegida para calcular los pesos de forma eficiente. Se ha elegido Adam como optimizer, es uno de los más usados en Computer Vision.

El código está depositado en GitHub<sup>2</sup>.

### Summary del modelo. Número de parámetros entrenables.

El modelo usado para entrenar el dataset correspondiente a las secuencias completas (unión de los flancos izquierdo y derecho) tiene 2.804.289 parámetros en total. De los cuáles son entrenables 2.801.857 y no entrenables 2.432.

<sup>2</sup> <https://github.com/jordibardaji/TFMVIUIA>

El modelo usado para entrenar los datasets que contienen las imágenes obtenidas de los flancos por separado tiene 1.821.249 parámetros en total. De los cuáles son entrenables 1.818.817 y no entrenables 2.432.

La diferencia entre los parámetros entrenables de los dos modelos es debida a que la entrada de los datos en los modelos que entrenan con sólo uno de los dos flancos por separado, son imágenes de 20 x 40 en grayscale. En cambio, el modelo que entrena las secuencias de imágenes que incluyen la unión de los dos flancos tiene por entrada imágenes de 40 x 40 en grayscale por lo que tiene más parámetros entrenables. Hay el doble de píxeles por los que se hace la convolución.

## **Problemas derivados de la magnitud del volumen de los datos y la complejidad del modelo**

Una vez definida la arquitectura de la CNN se procedió al entrenamiento del modelo. Sólo el proceso inicial de obtención de secuencias llevó días, y cuando por fin se obtuvieron las secuencias, el entrenamiento al cabo de más de 72 horas aún no había acabado con sólo 20 épocas.

Había dos posibilidades: O usar Google Colab o la adquisición de un PC con GPU integrada. Como la solución con Colab no resultó muy satisfactoria durante las prácticas del máster ya que no siempre era posible usar una GPU y además con entrenamientos largos el sistema mostraba mensajes que indicaban que Colab estaba diseñado para ser usado de forma interactiva y se producía una desconexión no deseada, se optó por la adquisición de un PC portátil con GPU.

El modelo adquirido fue un ASUS TUF DASH F15. Las especificaciones de ese modelo son:

- Procesador Intel i7-11370H @ 3,3GHz x 8.
- RAM: 16GB.
- Disco duro: 512 GB PCIe g3 SSD.
- GPU GeForce RTX 3060 Laptop 6GB.

- Sin Sistema Operativo.

Se eligió este modelo sin sistema operativo ya que la idea era instalar la distribución Ubuntu de GNU/Linux.

La versión de ubuntu instalada fue la 20.04 ya que ésta tiene combinaciones de drivers de GPU, librerías de tensorflow, etc, ... que funcionan con la GPU 3060 de Nvidia.

Se consultó documentación en forma de manuales online (documentación de Nvidia de la página oficial, documentación de tensorflow) y video tutoriales para la instalación y puesta en funcionamiento de todo ello.

Una vez encontrada la combinación de software adecuada (supuso días de pruebas), se usó desde entonces un environment para evitar que nuevas versiones en alguna actualización hicieran el conjunto del sistema no operativo.

En la siguiente imagen se puede observar la configuración de la GPU con la versión de los drivers instalada. Con el comando `nvidia-smi` en la terminal de Linux se puede obtener la información que aparece en la imagen.

La versión del driver instalado de la GPU es la 470.74 y la versión de CUDA es 11.4. Se observa también la memoria de la GPU usada por cada proceso en ese momento.

```
jordi@jordi-ASUS-TUF-Dash-F15-FX516PM-FX516PM: ~/TFM/dataset/Si...
-rw-rw-r-- 1 jordi jordi 1298 oct 10 13:48 SitiosNoInsercionchrY_9998.jpg
-rw-rw-r-- 1 jordi jordi 1309 oct 10 13:45 SitiosNoInsercionchrY_99.jpg
jordi@jordi-ASUS-TUF-Dash-F15-FX516PM-FX516PM:~/TFM/dataset/SinAlu$ nvidia-smi
Sun Oct 31 13:42:21 2021

+-----+
| NVIDIA-SMI 470.74          Driver Version: 470.74          CUDA Version: 11.4          |
+-----+-----+
| GPU   Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           MIG M.           |
+-----+-----+
|  0  NVIDIA GeForce ...   Off      | 00000000:01:00.0  On  |          N/A         |
| N/A   42C    P8      12W /  N/A   |  4423MiB /  5946MiB |           1%      Default |
+-----+-----+

+-----+
| Processes:                                                       GPU Memory |
|  GPU   GI    CI          PID    Type   Process name                        Usage      |
+-----+-----+
|  0     N/A   N/A         885     G   /usr/lib/xorg/Xorg                   35MiB     |
|  0     N/A   N/A        1640     G   /usr/lib/xorg/Xorg                   141MiB    |
|  0     N/A   N/A        1809     G   /usr/bin/gnome-shell                 32MiB     |
|  0     N/A   N/A        2988     G   /usr/lib/firefox/firefox             140MiB    |
|  0     N/A   N/A        3234     G   /usr/lib/firefox/firefox              2MiB     |
|  0     N/A   N/A        3270     G   /usr/lib/firefox/firefox              2MiB     |
|  0     N/A   N/A        3323     G   /usr/lib/firefox/firefox              2MiB     |
|  0     N/A   N/A       101018     G   /usr/lib/firefox/firefox              2MiB     |
|  0     N/A   N/A       139114     C   .../jordi/TFM/TFM/bin/python         4047MiB   |
+-----+

jordi@jordi-ASUS-TUF-Dash-F15-FX516PM-FX516PM:~/TFM/dataset/SinAlu$
```

Figura 12: Configuración GPU RTX 3060 en Ubuntu 20.04.

El entorno de desarrollo fue en jupyter-notebook. Python 3.8 con tensorflow-gpu.

El entrenamiento del modelo con el uso de la GPU pasó de durar días y no finalizar, a horas.

En estas condiciones con GPU RTX 3060, con controladores apropiados instalados y el volumen de datos entrenamientos de 100 épocas tardaron 15 horas aproximadamente. Con la cantidad de datos a procesar, un poco más de 2.250.000 imágenes es razonable.

# Resultados

## Dataset de secuencias con dos flancos unidos. Secuencias completas.

El entrenamiento del modelo se hace en 100 épocas. Como ya se ha mencionado anteriormente, el parámetro `batch_size` es de 128 que se definió en la función `image_dataset_from_directory` que carga los datos a medida que los necesita dinámicamente de disco. Además, como también se ha mencionado antes, se usa la función `prefetch` de Tensorflow que precarga los datos en memoria antes de que sean usados para evitar tiempos de espera.

### Resultados para Train y Validation

La orden de entrenamiento es:

- `model.fit(model.fit(Ttrain, validation_data = Val, epochs= epochs))`

Los datos generados dinámicamente durante el entrenamiento para la epoch 100 son:

```
Epoch 100/100
14187/14187 [=====] - 551s 39ms/step - loss:
0.0588 - binary_accuracy: 0.9779 - val_loss: 0.0286 -
val_binary_accuracy: 0.9937
```

De estos se observa que procesar cada epoch consume 551 segundos aproximadamente. Y necesita unos 14187 lotes para cada época.

Los valores resultantes del dataset Train, por tanto:

- Loss = 0,0588
- binary\_accuracy = 0,9779

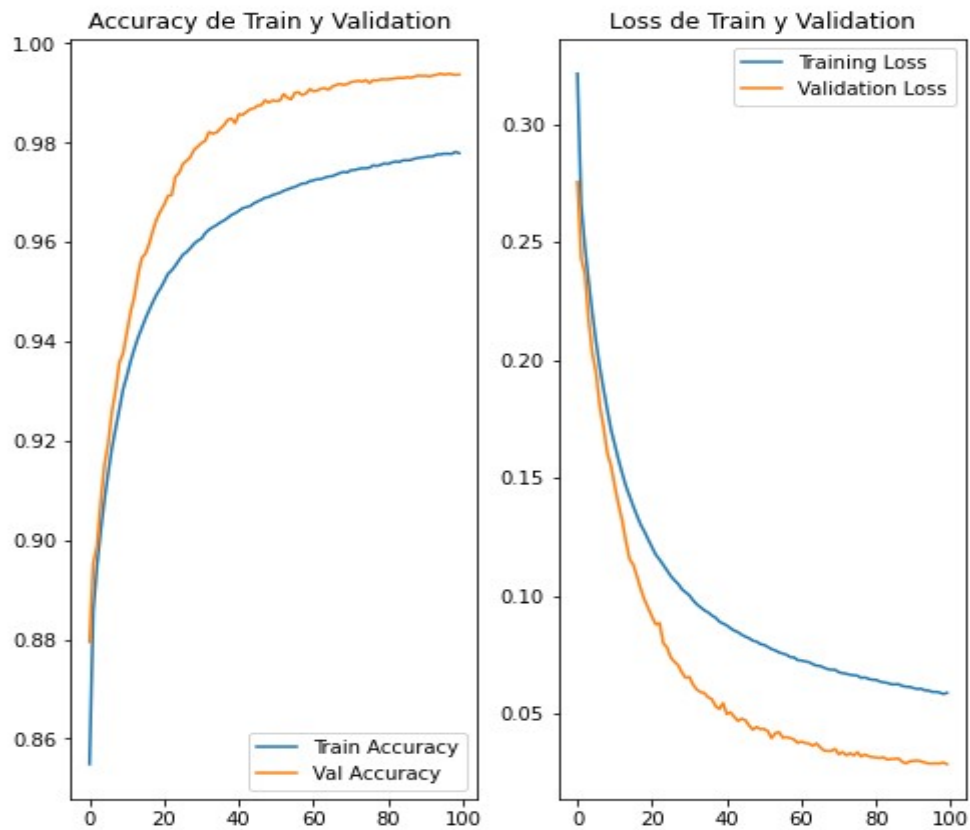
Los valores resultantes del dataset Validation son:

- Loss = 0,0286



- `binary_accuracy = 0,9937`

La gráficas de la evolución durante el entrenamiento para Train y Validation son:



*Figura 13: Gráficas de `binary_accuracy` y `loss` de Train y Validation. Dataset de secuencias completas (con los dos flancos unidos).*

## Resultados con el dataset de Test

```
score = model.evaluate(Test)
```

```
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

Obtenemos estos resultados:

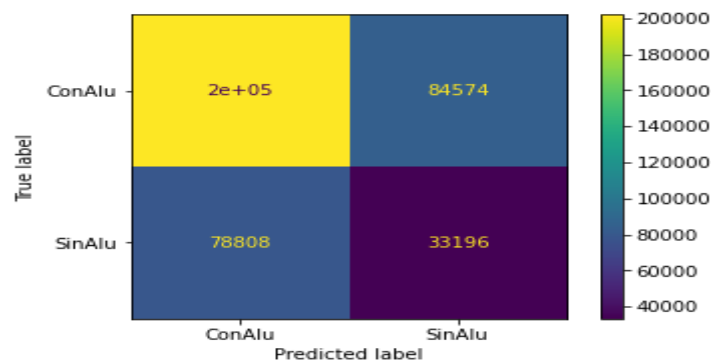
```
3115/3115 [=====] - 34s 11ms/step - loss:
0.3622 - binary_accuracy: 0.9088
Test loss: 0.36216849088668823 / Test accuracy: 0.9088423252105713
```

- Loss = 0,3622
- binary\_accuracy = 0,9088

Tabla de métricas de predicción de Test (Classification\_report):

score=model.predict(Test)

	precision	recall	f1-score	support
0	0.72	0.70	0.71	286602
1	0.28	0.30	0.29	112004
accuracy			0.59	398606
macro avg	0.50	0.50	0.50	398606
weighted avg	0.60	0.59	0.59	398606



*Figura 14: Matriz de confusión del dataset de Test de Secuencias completas*

## Dataset de secuencias con únicamente flanco izquierdo.

El entrenamiento del modelo se hace en 100 épocas como en el caso de secuencias completas. Se usa el mismo tamaño de batch\_size que en las secuencias completas, la función image\_dataset\_from\_directory y prefetch todo con los mismos parámetros.

La orden de entrenamiento es:

- `model.fit(model.fit(Train, validation_data = Val, epochs= epocas))`

Los datos generados dinámicamente durante el entrenamiento para la epoch 100 son:

Epoch 100/100

```
14187/14187 [=====] - 319s 22ms/step - loss:
0.1179 - binary_accuracy: 0.9540 - val_loss: 0.0610 -
val_binary_accuracy: 0.9834
```

De estos se observa que procesar cada epoch consume 391 segundos aproximadamente. Y necesita unos 14187 lotes para cada época. Los mismos que para las secuencias completas porque tienen el mismo número de muestras.

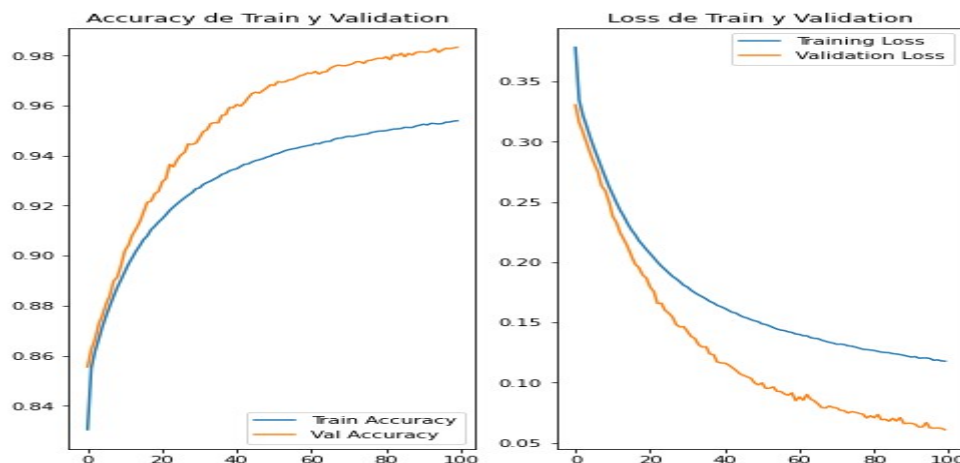
Los valores resultantes del dataset Train, por tanto, son:

- Loss = 0,1179
- binary\_accuracy = 0,9540

Los valores resultantes del dataset Validation son:

- Loss = 0,0610
- binary\_accuracy = 0.9834

Las gráficas de la evolución durante el entrenamiento para Train y Validation son:



*Figura 15: Gráficas de binary\_accuracy y loss de Train y Validation. Dataset de secuencias flanco izquierdo.*

## Resultados con el dataset de Test

```
score = model.evaluate(Test)
```

```
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

Obtenemos estos resultados:

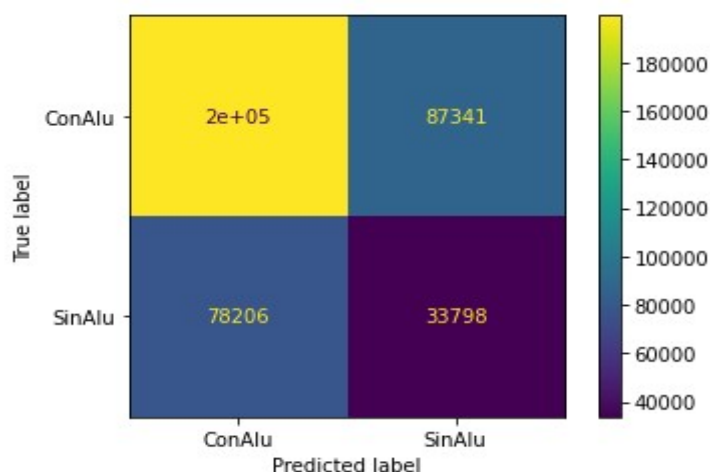
3115/3115 [======] - 27s 9ms/step - loss: 0.4448 -  
binary\_accuracy: 0.8704

Test loss: 0.44484439492225647 / Test accuracy: 0.8704359531402588

Tabla de métricas de de predicción de Test (Classification\_report):

```
score=model.predict(Test)
```

	precision	recall	f1-score	support
0	0.72	0.70	0.71	286602
1	0.28	0.30	0.29	112004
accuracy			0.58	398606
macro avg	0.50	0.50	0.50	398606
weighted avg	0.59	0.58	0.59	398606



*Figura 16: Matriz de confusión del dataset de Test de Secuencias Flanco Izquierdo.*

## **Dataset de secuencias con únicamente flanco derecho.**

El entrenamiento del modelo se hace en 100 épocas como en el caso de secuencias completas. Se usa el mismo tamaño de batch\_size que en las secuencias completas, la función image\_dataset\_from\_directory y prefetch, todo con los mismos parámetros.

La orden de entrenamiento es:

- `model.fit(model.fit(Train, validation_data = Val, epochs= epocas))`

Los datos generados dinámicamente durante el entrenamiento para la epoch 100 son:

Epoch 100/100

```
14187/14187 [=====] - 320s 23ms/step - loss:
0.1920 - binary_accuracy: 0.9206 - val_loss: 0.1208 -
val_binary_accuracy: 0.9614
```

- Loss = 0,1920
- binary\_accuracy = 0,9206

Los valores resultantes del dataset Validation son:

- Loss = 0, 1208
- binary\_accuracy = 0,9614

La gráficas de la evolución durante el entrenamiento para Train y Validation son:

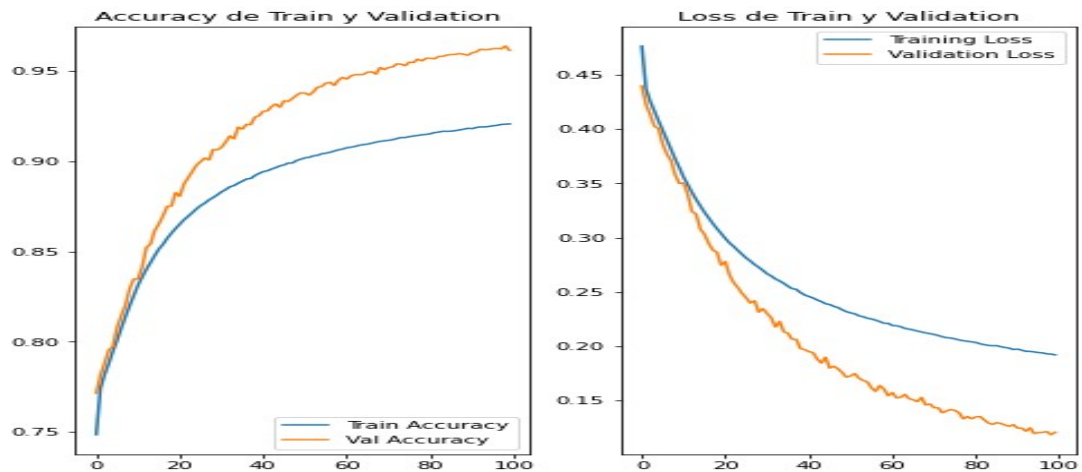


Figura 17: Gráficas de `binary_accuracy` y `loss` de Train y Validation. Dataset de secuencias flanco derecho.

## Resultados con el dataset de Test

```
score = model.evaluate(Test)
```

```
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

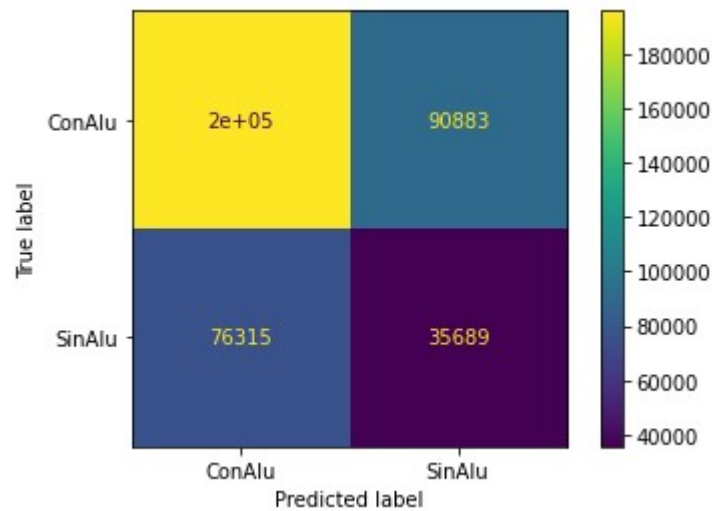
Obtenemos estos resultados:

```
3115/3115 [=====] - 22s 7ms/step - loss:
0.5709 - binary_accuracy: 0.7861
Test loss: 0.5709303021430969 / Test accuracy: 0.7861196398735046
```

Tabla de métricas de de predicción de Test (`Classification_report`):

```
score=model.predict(Test)
```

	precision	recall	f1-score	support
0	0.72	0.68	0.70	286602
1	0.28	0.32	0.30	112004
accuracy			0.58	398606
macro avg	0.50	0.50	0.50	398606
weighted avg	0.60	0.58	0.59	398606



*Figura 18: Matriz de confusión secuencias Flanco Derecho*

## Valoración y discusión de los resultados.

### Características aplicables a los tres conjuntos de datos

Por regla general, el modelo se comporta bien en los tres conjuntos de datos (completo, flanco izquierdo y flanco derecho) con valores de loss muy bajos tanto en Train como en Validation.

**No se produce overfitting** con ninguno de los tres modelos, ya que la curva de `validation_binary_accuracy` está siempre por debajo de la de `binary_accuracy` de training.

Los valores de loss del conjunto de validation y el de training llegan a ser bajísimos. Del orden de 0,0x.

## Comparación del modelo entre los distintos conjuntos de datos

Los valores de binary\_accuracy para Train obtenidos con el conjunto de secuencias completas son siempre mejores que los obtenidos. Sucede lo mismo con los valores de binary\_accuracy para Validation.

Los valores de loss son menores en el conjunto de secuencias completas que en los conjuntos de flancos.

Los valores de las métricas para los datasets de Test también son mejores los del conjunto de secuencias completas.

De lo cual se deduce, evidentemente, que hay que tener en cuenta la secuencia a ambos lados del punto de inserción de la Alu para predecir nuevos puntos de inserción.

De las secuencias de sólo un flanco se comporta mejor las que contienen bases del flanco izquierdo.

Tabla comparativa de resultados para cada dataset de partida.

	Binary_accuracy			Loss		
Tipo de secuencia	Train	Validation	Test	Train	Validation	Test
<b>Completa</b>	0,9779	0,9937	0,9088	0,0588	0,0286	0,3622
<b>F. Izquierdo</b>	0,9540	0,9834	0,87	0,1179	0,0610	0,4448
<b>F. Derecho</b>	0,9206	0,9614	0,7861	0,1920	0,1208	0,5709

Como se puede observar en la tabla los datos del conjunto de secuencias formados por la unión de las bases del flanco izquierdo y las bases del flanco derecho (secuencias completas) son las que ofrecen mejores resultados.



Deberíamos por tanto, usar secuencias de 1600 bases convertidas a imágenes de 40 x 40 en escala de grises y valores normalizados entre 0 y 1 con el modelo usado en ellas para determinar si en esa secuencia se puede insertar una Alu y con qué probabilidad.

## Conclusiones y desarrollos futuros.

### Conclusiones

Una de las conclusiones que se pueden obtener es que en este trabajo casi la mayor carga de trabajo ha sido entender lo que hay debajo: la biología específica del problema. En estos problemas de campos ajenos al “científico de datos” es crucial entender qué está sucediendo. En este caso, si hubiésemos intentado plantear una estrategia sin conocer la primera opción parece que como se trata de “texto” se podrían usar planteamientos de NLP o de LTSM o RNN que en este caso según mi opinión no serían adecuados.

En segundo lugar, uno de los escollos mayores ha sido el volumen de datos con los que se trabaja en el campo de la genómica humana. Los procesos de lectura/escritura al disco se eternizan. Por no hablar de la cantidad de parámetros entrenables del modelo. Por eso, como ya he explicado, en mi caso tuve que elegir entre Google Colab o un PC con GPU integrada. Mi experiencia fue que tantos parámetros, y más tratándose de convoluciones que hacen un uso exhaustivo de cálculos matriciales, añadiendo un conjunto de datos del tamaño de más de 2 millones de imágenes, hace imprescindible el uso de GPU, abundante RAM y discos duros cuanto más rápidos, mejor, obviamente.

Relacionado con el uso de la GPU en un PC propio, es un trabajo, para nada desdeñable en tiempo y dedicación, encontrar la combinación de versión de SO, drivers y librerías de Tensorflow y python que sean compatibles para el uso de la GPU. Resultó muy valioso el uso de environments de desarrollo para congelar las versiones de todos los paquetes de software una vez se ha encontrado la combinación adecuada.

Otra de las conclusiones es que los Frameworks, al menos según mi experiencia, y hablando de este caso en concreto, no aportan lo suficiente por el coste de tiempo que supone aprender la API asociada.

## **Futuros desarrollos**

Los futuros desarrollos deberían ir en varias direcciones:

- Eliminar las clases a priori negativas que no lo son.
- Determinación de la longitud de los flancos (el número de bases a tener en cuenta) que determina la inserción de la Alu en un cierto punto.
- Balancear las clases positivas con los supuestas negativas.
- Modificar el modo de construcción de las imágenes a partir de las secuencias de bases. Veremos que este punto podría, posiblemente, facilitar la adquisición de características por parte de la CNN.
- Aumentar la riqueza de los datos de entrada convirtiendo las secuencias de bases en lo que los datos originalmente son: datos en las tres dimensiones espaciales.

## **Eliminar las clases a priori negativas que no lo son.**

Desde mi punto de vista, las mejoras deberían ser en el sentido antes apuntado de que las secuencias de entrada que han sido clasificadas como negativas pueden no serlo. Por tanto, sería interesante eliminar de las secuencias negativas las que son en realidad candidatas a serlo.

Para ello, la estrategia podría ser analizar con un algoritmo de clustering, que determine distancias entre las secuencias negativas que han sido calificadas como positivas por nuestro modelo con las secuencias realmente positivas. Eliminar las secuencias que son demasiado cercanas a las positivas para no considerarlas como entradas del tipo SinAlu (negativas).

## **Determinación de la longitud de influencia de los flancos que determina la inserción de la Alu en un cierto punto.**

Otro futuro campo de investigación derivado sería el de determinar la longitud real del campo de influencia en las inserción de las Alu en un punto concreto. En el estudio decidí tomar una longitud de flanco de 800 bases. Pero no tiene porqué ser esa la longitud de influencia. Este aspecto podría hacerse quizás con el uso de transformers. Jugando con la información recogida en el modelo con una GAN. Esta estrategia también sería una buena candidata para generar secuencias futuras de inserción que no están contempladas como tales actualmente. Sería semejante a lo que sucede con la utilidad web [thispersondoesnotexist.com](http://thispersondoesnotexist.com), en la que fruto del entrenamiento de un modelo de reconocimiento de caras, la GAN es capaz de crear nuevas imágenes que no existían antes de personas. Haríamos lo mismo pero en vez de caras, con secuencias.

## **Balancear las clases positivas con los supuestas negativas.**

Las clase **ConAlu** tiene un mayor número de elementos que la clase **SinAlu**. Esto es debido a que las sentencias **ConAlu** se conocen perfectamente. En cambio, las secuencias **SinAlu** se han construido a partir del punto medio encontrado entre el punto de final de una Alu con el punto de inicio de la siguiente, a partir de ese punto, se han obtenido 800 bases a la izquierda y 800 a la derecha y al unir los dos flancos obtenemos una secuencia **SinAlu**. Pero esta secuencia se descarta si la secuencia tiene algún carácter "N" o si la longitud entre las dos Alu contiguas es mayor que 1600. Por estos motivos, tenemos en los dataset menos clases SinAlu.

Se podría intentar aplicar métodos alternativos de búsqueda en el caso de que la secuencia **SinAlu** construida tiene caracteres "N" o no haya la longitud necesaria entre las dos Alu.

## **Modo de construcción de las imágenes a partir de las secuencias de bases.**

En este trabajo se ha dado por hecho que al convertir una secuencia lineal ordenada de caracteres a una imagen se conservan las propiedades de distancias originales.

En realidad este no es el caso. Si prestamos atención en una imagen creada a partir de las secuencias, tanto en las positivas como las negativas, vemos que teniendo en cuenta como los kernels van pasando por la imagen el punto final del flanco izquierdo y el punto inicial del flanco derecho no son contiguos para los kernels. Están situados a una distancia de 40 bases. La última base del flanco izquierdo está en la posición en la fila 20, columna 40 y la primera base del flanco derecho está situada en la posición columna 1 de la fila 21 ocupada por el píxel de la imagen.

Esto no es correcto, deberían estar contiguos. La primera base del flanco derecho debería situarse en la posición columna 40 de la fila 21. De esta manera, la segunda base del flanco derecho debería ocupar la columna 39 de la fila 21 y así sucesivamente.

Les debería afectar el paso del kernel conjuntamente en alguna ocasión a los píxeles que representan bases próximas (en el caso de kernels de 3x3 como los usados en el modelo).

De esta manera, no son afectados por el mismo kernel nunca. Por tanto, su proximidad física, a mi entender, tal y como me imagino que funciona la convolución se pierde un poco.

## **Aumentar la riqueza de los datos de entrada convirtiendo las secuencias de bases en lo que los datos originalmente son: datos en las tres dimensiones espaciales.**

La molécula del ADN la obtenemos de los ficheros como una cadena lineal de caracteres (bases). Pero realmente la disposición real de las bases no es tal. Es una cadena en tres dimensiones. La famosa hélice.

Teniendo en cuenta este dato, se podría intentar reproducir en matrices tridimensionales la estructura de escalera de caracol del ADN para las secuencias de entrada al modelo. La espiral estaría contenida en un cubo o paralelepípedo. Para reproducir las matrices tridimensionales se deberían tener en cuenta el diámetro de la hélice y la altura entre cada vuelta. Cuando algo parecido a AlphaFold se pueda aplicar al ADN, estos datos no serían hiperparámetros del problema. Se podría conocer como está plegado el ADN y las posiciones espaciales de cada base.

Esto significaría usar capas Conv3D y kernels también de 3 dimensiones.

## Anexos

### Anexo 1. Solicitud de área temática para el Trabajo Fin de Máster

DATOS DEL ALUMNO/A
<b>APELLIDOS:</b> BARDAJÍ CUSÓ
<b>NOMBRE:</b> JORDI
<b>DNI:</b> 43000601p
<b>Área temática sobre la que versará el trabajo</b> <b>El área temática es:</b> la IA aplicada a la genómica.
<b>Descripción breve del trabajo (5-10 líneas):</b>  El trabajo consiste en la aplicación de técnicas de IA (deep learning) para la predicción de los sitios probables de inserción futura de unas secuencias de bases que se replican y se insertan a lo largo del ADN. Estas secuencias únicamente aparecen en primates y en humanos (las secuencias Alu) .

En Palma de Mallorca, a 8 de Noviembre de 2021

Firmado: el/la alumno/a,

Jordi Bardají Cusó

## Anexo 2. Solicitud para la presentación y defensa del Trabajo Fin de Máster

DATOS DEL ALUMNO/A
APELLIDOS: BARDAJÍ CUSÓ
NOMBRE: JORDI
DNI: 43000601p
TÍTULO DEL TRABAJO: Predicción de futuros sitios de inserción de las secuencias Alu en el ADN
Convocatoria: 3r Periodo

Tras haber superado por completo las asignaturas obligatorias y optativas<sup>3</sup> del Máster **INTELIGENCIA ARTIFICIAL**

SOLICITA que se le autorice la defensa del Trabajo Fin de Máster, bajo la dirección del profesorado que se indica a continuación:

DATOS DEL PROFESOR/A
Director/a Ricardo Lebrón Aguilar

Como alumno/a del Máster **INTELIGENCIA ARTIFICIAL** autoriza<sup>4</sup> por este escrito la publicación total o parcial de su Trabajo Fin de título para uso didáctico y académico.

Firmado: el/la alumno/a,



Jordi Bardají Cusó

<sup>3</sup> Adjuntar a este Anexo, una copia informativa del expediente.

<sup>4</sup> En caso de no autorizar la publicación total o parcial de su Trabajo Fin de Título para uso didáctico y académico, indíquelo por escrito en este pie de página.

## Anexo 3. Informe de autorización del Trabajo Fin de Máster

**MÁSTER EN INTELIGENCIA ARTIFICIAL Curso 2020/2021**

Nombre y apellidos del estudiante
Jordi Bardají Cusó
Nombre y apellidos del/la director/a de TFM
Ricardo Lebrón Aguilar
Título del Trabajo
Predicción de futuros sitios de inserción de las secuencias Alu en el ADN
Informe del/la directora/a TFM:
---- Autorizo a la defensa de este trabajo ----- No autorizo a la defensa de este trabajo (marque con una cruz la opción que proceda)

En \_\_\_\_\_, a 8 de Noviembre de 2021

Firmado: El/la directora/a TFM

Ricardo Lebrón Aguilar



## **Anexo 5. Referencias Bibliográficas**

- ADN (Ácido Desoxirribonucleico)* | NHGRI. (s. f.). Genome.gov. Recuperado 7 de noviembre de 2021, de <https://www.genome.gov/es/genetics-glossary/ADN-acido-Desoxirribonucleico>
- Brownlee, J. (2019, abril 9). How to Load Large Datasets From Directories for Deep Learning in Keras. *Machine Learning Mastery*.  
<https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/>
- Chen, J.-M., Férec, C., & Cooper, D. N. (2007). Mechanism of Alu integration into the human genome. *Genomic Medicine*, 1(1), 9-17. <https://doi.org/10.1007/s11568-007-9002-9>
- Chromosome*. (s. f.). Genome.Gov. Recuperado 7 de noviembre de 2021, de <https://www.genome.gov/genetics-glossary/Chromosome>
- Deininger, P. (2011). Alu elements: Know the SINEs. *Genome Biology*, 12(12), 236. <https://doi.org/10.1186/gb-2011-12-12-236>
- DNA Barcoding—An overview* | ScienceDirect Topics. (s. f.). Recuperado 7 de noviembre de 2021, de <https://www.sciencedirect.com/topics/medicine-and-dentistry/dna-barcoding>
- FASTA format. (2021). En *Wikipedia*. [https://en.wikipedia.org/w/index.php?title=FASTA\\_format&oldid=1042093956](https://en.wikipedia.org/w/index.php?title=FASTA_format&oldid=1042093956)
- FASTA format description*. (s. f.). Recuperado 7 de noviembre de 2021, de <http://genetics.bwh.harvard.edu/pph/FASTA.html>
- Gen* | NHGRI. (s. f.). Genome.gov. Recuperado 7 de noviembre de 2021, de <https://www.genome.gov/es/genetics-glossary/Gen>
- Gunasekaran, H., Ramalakshmi, K., Rex Macedo Arokiaraj, A., Deepa Kanmani, S., Venkatesan, C., & Suresh Gnana Dhas, C. (2021). Analysis of DNA Sequence

- Classification Using CNN and Hybrid Models. *Computational and Mathematical Methods in Medicine*, 2021, e1835056. <https://doi.org/10.1155/2021/1835056>
- Home—Keras\_dna Documentation. (s. f.). Recuperado 7 de noviembre de 2021, de <https://keras-dna.readthedocs.io/en/latest/>
- Konkel, M. K., Walker, J. A., Hotard, A. B., Ranck, M. C., Fontenot, C. C., Storer, J., Stewart, C., Marth, G. T., & Batzer, M. A. (2015). Sequence Analysis and Characterization of Active Human Alu Subfamilies Based on the 1000 Genomes Pilot Project. *Genome biology and evolution*, 7(9), 2608-2622. <https://doi.org/10.1093/GBE/EVV167>
- Kress, W. J., & Erickson, D. L. (2008). DNA barcodes: Genes, genomics, and bioinformatics. *Proceedings of the National Academy of Sciences of the United States of America*, 105(18287050), 2761-2762. <https://doi.org/10.1073/pnas.0800476105>
- Payer, L. M., Steranka, J. P., Yang, W. R., Kryatova, M., Medabalimi, S., Ardeljan, D., Liu, C., Boeke, J. D., Avramopoulos, D., & Burns, K. H. (s. f.). *Structural variants caused by Alu insertions are associated with risks for many human diseases*. <https://doi.org/10.1073/pnas.1704117114>
- Quang, D., & Xie, X. (2016). DanQ: A hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Research*, 44(11), e107. <https://doi.org/10.1093/nar/gkw226>
- Razzaq, A. (2021, octubre 21). What is a Convolutional Neural Network? *MarkTechPost*. <https://www.marktechpost.com/2021/10/20/what-is-a-convolutional-neural-network/>
- Schema for RepeatMasker—Repeating Elements by RepeatMasker. (s. f.-a). Recuperado 7 de noviembre de 2021, de <https://genome-asia.ucsc.edu/cgi-bin/hgTables?>

db=mm10&hgta\_track=rmsk&hgta\_table=rmsk&hgta\_doSchema=describe+table+schema

*Schema for RepeatMasker—Repeating Elements by RepeatMasker.* (s. f.-b).

Recuperado 7 de noviembre de 2021, de

<http://genome.ucsc.edu/cgi-bin/hgTables>

Shaw, K. R. M., Horne, K. V., Zhang, H., & Boughman, J. (s. f.). *Genetics Education*

*Innovations in Teaching and Learning Genetics Essay Contest Reveals*

*Misconceptions of High School Students in Genetics Content.*

<https://doi.org/10.1534/genetics.107.084194>

Team, K. (s. f.-a). *Keras documentation: Image classification from scratch.* Recuperado

8 de noviembre de 2021, de

[https://keras.io/examples/vision/image\\_classification\\_from\\_scratch/](https://keras.io/examples/vision/image_classification_from_scratch/)

Team, K. (s. f.-b). *Keras documentation: The Sequential model.* Recuperado 7 de

noviembre de 2021, de [https://keras.io/guides/sequential\\_model/](https://keras.io/guides/sequential_model/)

*Tutorial—Janggu 0.10.2 documentation.* (s. f.). Recuperado 7 de noviembre de 2021,

de <https://janggu.readthedocs.io/en/latest/tutorial.html>